

<b>Chapter 1,</b> p.001	<b>An Introduction to Visual Studio 2017 and Visual Basic</b>
<b>Chapter 2,</b> p.049	<b>Planning Applications and Designing Interfaces</b> plan a Windows Forms app; windows standards for lbl, btn, graphics, fonts, colors; Access keys - Exit; Tab order -TabIndex; create a Planning Chart; design GUI using Windows standards; add a (lbl) to the form; add a (txt) to the form
<b>Chapter 3,</b> p.073	<b>Coding with Variables, Named Constants &amp; Calculations</b> pseudocode & flowchart; Variables scope level: <b>For, Dim, Static, Private</b> ; Named Constants scope level: <b>Const, Private Const</b> ; Arithmetic Operators (#\$1-6): ^ - * / \ Mod + -; TryParse method; ToString method; Option statements; InputBox function; event procedures txt_TextChanged and txt_Enter
<b>Chapter 4,</b> p.119	<b>The Selection Structure &amp; operators &amp; (chk) &amp; (rad) &amp; KeyPress</b> selection structures: <b>If...Then...Else...ElseIf &amp; Select Case...To...Is</b> ; nested selection structures; operators: arithmetic & comparison & logical/boolean; string comparisons: methods ToUpper & ToLower & Trim; Check Box (chk); Radio Button (rad); GroupBox tool; event procedure txt_KeyPress & e.KeyChar & e.Handled = True & ControlChars.Back; arithmetic assignment operators += etc...; using TryParse and its return Boolean value to validate numbers; 4 common errors with selection structures; <b>If...Then...Else</b> s.s. example: swapping numeric values
<b>Chapter 5,</b> p.181	<b>Repetition Structures / Loops &amp; ListBox</b> Pretest loops: <b>Do While/Do Until...Loop</b> & counter-controlled loop <b>For...Next</b> ; Posttest loop: <b>Do...Loop While/Loop Until</b> ; Nested; Counters & Accumulators; String's concatenation operator & ListBox (1st); Financial. Class; default buttons: <b>AcceptButton &amp; CancelButton</b> ; Sys constants: <b>ControlChars.Back, ControlChars.NewLine, ControlChars.Tab</b> ;
<b>Chapter 6,</b> p.237	<b>Event &amp; Sub &amp; Function Procedures</b> Event-handling Sub Procedure (Event Procedure) - more in detail & multiple objects & multiple events; Independent Sub Procedure (Sub Procedure) + Calling Statement - no parameter & passing variable: by Value - <b>ByVal</b> & by Reference - <b>ByRef</b> ; Function Procedure (Function) + Calling Statement; rounding numbers - method <b>Math.Round</b> ; ComboBox tool (cbo); frmMain_FormClosing; MessageBox.Show method & return value <b>DialogBox</b>
<b>Chapter 7,</b> p.287	<b>String Manipulation</b> = <i>string</i> . & Generate Random numbers (CH7_A3) & Control's property Enabled (CH7_A4) and Control's method Focus (CH7_A5)
<b>Chapter 8,</b> p.341	<b>Arrays</b> = <i>Array of variables</i> 1D array & 2D array & For Each...Next loop & ReDim & LINQ with arrays & Structure & array of structure variables
<b>Chapter 9,</b> p.393	<b>Sequential Access Files / Text files &amp; creating Menus</b> Output files & Input files & Menu(mnu) & confirmation message
<b>Chapter 10,</b> p.427	<b>Classes and Objects</b> creating a <b>Class</b> , instantiating an <b>Object</b> from the <b>Class</b> , auto-implemented <b>Public Property</b> , overloaded method
<b>Chapter 11,</b> p.475	<b>SQL Server Databases &amp; Try...Catch statement</b> Try...Catch "safety net" statement = handling/catching app's exception/error during run time without terminating the app

**Chapter 12,** p.529

## **Database Queries with SQL**

= retrieve a specific information from a database

Parameter Query & Parameter marker @; use SQL to add a Calculated field; use SQL Aggregate functions

**Chapter 13,** p.565

## **Web Site Applications using "ASP.NET Web Forms Site" template**

Site.master; 2x Static Web pages: Default.aspx & About.aspx; Dynamic Web page: Calculator.aspx recycled from unused Contact.aspx  
Start Without Debugging Ctrl+F5; user input Validation Tools

**Appendix C,** p.632

## **Finding and Fixing Program Errors - Syntax, Logic, Run Time**

3 types of Errors: Syntax error, Logic error, and Run time error

**Appendix F**

## **SDI, MDI, TDI - multiple forms & dialog boxes**

SDI app - main form & splash screen, main form & dialog box, 3x main form

MDI app - 2x primary form = Parent & Child; quasi TDI app - main form & TabControl tool

**CH1\_FOCUS ON THE CONCEPTS LESSON**

- CH1\_F1** - Computer Programming Terminology
- CH1\_F2** - The programmer's Job
- CH1\_F3** - The Visual Basic Programming Language
- CH1\_F4** - The Visual Studio IDE: the Einstein's Equation application example
- CH1\_F5** - Assigning "(Name)" for coded or referred Objects

**CH1\_APPLY THE CONCEPTS LESSON**

- CH1\_A1** - Start and Configure Visual Studio Community 2017
- CH1\_A2** - To Create a Windows Forms Application: the Einstein's Equation application example
- CH1\_A3** - Manage the Windows in the IDE
- CH1\_A3.1** - to close, open, auto-hide, and display windows in the IDE : the Einstein's Equation application example
- CH1\_A4** - Change a Form File's Name
- CH1\_A5** - Change the Properties of a Form
- CH1\_A5.1** - to display the form's properties and the most commonly used
- CH1\_A6** - Save a Solution
- CH1\_A7** - Close and Open a Solution
- CH1\_A8** - Add a Control to a Form: the Einstein's Equation application example
- CH1\_A8.1** - To add a picture box to the Form: the Einstein's Equation application example
- CH1\_A8.2** - To add a button to the Form: the Einstein's Equation application example
- CH1\_A9** - Use the Format Menu
- CH1\_A9.1** - To adjust the Exit button's height and top border: the Einstein's Equation application example
- CH1\_A10** - Lock the Controls on the Form
- CH1\_A11** - Before you Start and End an Application - Verify the Startup Form
- CH1\_A11.1** - To verify the startup form: the Einstein's Equation application example
- CH1\_A11.2** - Start and End an Application
- CH1\_A11.3** - To change the name of the executable file, and then start and end the application: the Einstein's Equation application example
- CH1\_A12** - Enter Code and Comments in the Code Editor Window
- CH1\_A12.1** - To open the Code Editor Window
- CH1\_A12.2** - To collapse and expand a region of code
- CH1\_A12.3** - To select the btnExit control's Click event
- CH1\_A12.4** - the Me.Close() Statement
- CH1\_A12.5** - to code the btnExit\_Click procedure
- CH1\_A12.6** - Assignment Statements
- CH1\_A12.7** - Comments
- CH1\_A12.8** - To enter a comment and code: the Einstein's Equation application example
- CH1\_A13** - Print an Application's Code and Interface
- CH1\_A14** - Exit Visual Studio and Run an Executable File
- CH1\_A15** - **Splash Screen** - additional topic from **Appendix B** - template & code
- CH1\_A16** - Timer control - additional topic from **Appendix B** - code example
- CH1\_A17** - PrintForm Control - additional topic from **Appendix B** - code example; installing Visual Basic PowerPack

**CH1\_Summary****CH1\_Key Terms**

plan a Windows Forms app; windows standards for lbl, btn, graphics, fonts, colors; Access keys - Exit; Tab order - TabIndex; create a Planning Chart; design GUI using Windows standards; add a (lbl) to the form; add a (txt) to the form

## CH2\_FOCUS ON THE CONCEPTS LESSON

- CH2\_F1** - Planning a Windows Forms application: example with Restaurant Tip application
- CH2\_F2** - Windows standards for interfaces: lbl, btn, graphics, fonts, color
- CH2\_F3** - Access keys - Alt keys: E&xit = Exit
- CH2\_F4** - Tab order - TabIndex: example with Restaurant Tip application

## CH2\_APPLY THE CONCEPTS LESSON

- CH2\_A1** - Create a Planning Chart for a Windows Forms application: example with Jacobsons Furniture store application
- CH2\_A2** - Design an interface using the Windows standards: example with Jacobsons Furniture store application
- CH2\_A3** - Add a label control to the form (lbl)
- CH2\_A4** - Add a text box to the form (txt)
- CH2\_A5** - Set the Tab Order via Designer window

## CH2\_Summary

## CH2\_Key Terms

# Coding with Variables, Named Constants & Calculations

pseudocode & flowchart; Variables scope level: **For, Dim, Static, Private**; Named Constants scope level: **Const, Private Const**; Arithmetic Operators (#\$1-6): **^ - \* / \ Mod + -**; TryParse method; ToString method; Option statements; InputBox function; event procedures **txt\_TextChanged** and **txt\_Enter**

scope from smallest:		
	variable	constant
1. Block-level	<b>For</b>	
2. Procedure-level	<b>Dim, Static</b>	<b>Const</b>
3. Class-level	<b>Private</b>	<b>Private Const</b>
4. Form-level	<b>Friend</b>	

## CH3\_FOCUS ON THE CONCEPT LESSON

- CH3\_F1 - Pseudocode and Flowchart: beneficial planning tools for your code
- CH3\_F2 - Main Memory of a Computer: basic info & reserving/declaring a memory location: **variables & named constants**
- CH3\_F3 - Variable: declaring a variable memory location for your data: syntax & example
- CH3\_F3.1 - Data types for your **variables & named constants**: my tab with a detailed info
- CH3\_F4 - TryParse method: converts **string** into **numeric** data type - syntax & example
- CH3\_F5 - Arithmetic Operators (#\$1-6): **^ - \* / \ Mod + -**
- CH3\_F6 - Assigning a Value to an **existing Variable**:
- CH3\_F7 - ToString method and it's format specifiers **C** or **F** or **N** or **P**: converts **numbers** into **strings** - syntax & e.g.
- CH3\_F8 - Option statement - above all in a code - including explanation: **Option Explicit** & **Option Strict** & **Option Infer**
- CH3\_F9 - Named Constant declaring a constant memory location for your data: syntax & example

## CH3\_APPLY THE CONCEPT LESSON

- CH3\_A1 - determine a memory location's **Scope** and **Lifetime**: **Procedure-level** vs **Class-level**
- CH3\_A2 - use a **Procedure-Level Variables**: the **Dim** statement
- CH3\_A3 - use a **Procedure-Level Named Constants**: the **Const** statement
- CH3\_A4 - use a **Class-Level Variable**: the **Private** statement
- CH3\_A5 - use a **Procedure-Level Variable** with a **Class-Level scope**: the **Static** statement
- CH3\_A6 - use a **Class-Level Named Constant**: the **Private Const** statement
- CH3\_A7 - Professionalize Your Application's Interface: event procedures **txt\_TextChanged** and **txt\_Enter**
- CH3\_A8 - InputBox function - additional topic from **Appendix B** - interact with user while app is running - syntax & example

**CH3\_Summary**

**CH3\_Key Terms**

**CH3\_Exercises**

# The Selection Structure & operators & (chk) & (rad) & KeyPress

selection structures: **If...Then...Else...ElseIf** & **Select Case...To...Is**; nested selection structures;  
 operators: arithmetic & comparison & logical/boolean; string comparisons: methods **ToUpper** & **ToLower** & **Trim**; Check Box (chk);  
 Radio Button (rad); GroupBox tool; event procedure **txt\_KeyPress** & **e.KeyChar** & **e.Handled = True** & **ControlChars.Back**;  
 arithmetic assignment operators **+=** etc...; using **TryParse** and its return **Boolean** value to validate numbers;  
 4 common errors with selection structures; **If...Then...Else** s.s. example: swapping numeric values

## 3 basic control structures:

- **1. Sequence structure** = CH1, CH2, CH3
- **2. Selection structure** = If...Then...else, Select Case
- **3. Repetition structure** = For...Next, Loop

## Operators precedence:

- |                 |                      |   |
|-----------------|----------------------|---|
| <b>## 7:</b>    | comparison operators | - can override with ()<br><, <=, >, >=, =, <> |
| <b>## 8-11:</b> | logical operators    | Not, AndAlso, OrElse, Xor                     |

## CH4\_FOCUS ON THE CONCEPTS LESSON

- CH4\_F1 - 3 basic **control Structures** - **Sequence** structure, **Selection** structure, **Repetition** structure
- CH4\_F2 - **If...Then...Else** selection structure statement:
- CH4\_F3 - **Comparison Operators** (#\$ = 7): =, >, >=, <, <=, <>
- CH4\_F4 - **Logical - Boolean Operators** (#\$8-11): **Not**, **And**, **AndAlso**, **Or**, **OrElse**, **Xor**
- CH4\_F5 - **Summary of Operators**: **arithmetic** (#\$1-6), **comparison** (#\$7), **logical** (#\$8-11)
- CH4\_F6 - **String Comparisons**: methods **ToUpper** & **ToLower** & **Trim**
- CH4\_F7 - **Nested** selection structures aka s.s. **Nested Inside** other s.s.
- CH4\_F8 - **If...Then...Else** s.s.: **Multiple-Alternative** aka **Extended** selection structures using **ElseIf**
- CH4\_F9 - **Select Case** selection structure statement:
- CH4\_F9.1 - **Select Case** s.s. - specifying a range of values using keywords: **To**, **Is**

## CH4\_APPLY THE CONCEPTS LESSON

- CH4\_A1 - **Check Box (chk)** : add it to a Form
- CH4\_A2 - **Check Box (chk)** : code a **GUI** that contains it
- CH4\_A2.1 - **Check Box (chk)** : **chk\_CheckedChanged** event
- CH4\_A3 - **Radio Button (rad)** : add it to a Form
- CH4\_A4 - **Radio Button (rad)** : code a **GUI** that contains it
- CH4\_A4.1 - **Radio Button (rad)** : **rad\_CheckedChanged** event
- CH4\_A4.2 - **Radio Button (rad)** : **rad.Checked** compare using s.s. **Select Case** vs **If...Then...Else**
- CH4\_A5 - group Objects using a **GroupBox** tool/control located at: **Toolbox / Containers / GroupBox**
- CH4\_A6 - **Professionalize your app's GUI**: code the event procedure: **txt\_KeyPress** to restrict user from entering specified characters using s.s. & constant **ControlChars.Back**
- CH4\_A7 - **Professionalize your code** using **arithmetic assignment operators**(=): **+=**, **-=**, **\*=**, **/=**
- CH4\_A8 - using **TryParse** to validate data - additional topic from **Appendix B** - using a return **Boolean** value to validate numbers
- CH4\_A9 - 4 common errors made when writing **Selection Structures** - additional topic from **Appendix B**
- CH4\_A10 - **If...Then...Else** selection structure example: swapping numeric values - additional topic from **Appendix B**

**CH4\_Summary**

**CH4\_Key Terms**

**CH4\_Exercises**

Pretest loops: **Do While/Do Until...Loop** & counter-controlled loop **For...Next**; Posttest loop: **Do...Loop While/Loop Until**; Nested; Counters & Accumulators; String's concatenation operator & **ListBox** (1st); **Financial Class**; default buttons: **AcceptButton** & **CancelButton**; Sys constants: **ControlChars.Back**, **ControlChars.NewLine**, **ControlChars.Tab**;

**CH5\_MY CHEAT SHEET****CH5\_FOCUS ON THE CONCEPTS LESSON**

- CH5\_F1** - Repetition Structure / Loop - basic info
- CH5\_F2** - Pretest loop: **Do While/Do Until...Loop** statement
- CH5\_F3** - String's concatenation operator: & (zřetězení), and constant: **ControlChars.NewLine**
- CH5\_F4** - Pretest loop: **Do While/Do Until...Loop** statement's **Infinite loops** = mistake by the programmer
- CH5\_F5** - Posttest loop: **Do ... Loop While/Loop Until** statement
- CH5\_F5.1** - Posttest loop: **Do ... Loop Until** e.g.: **30.Fibonacci Solution\_EXERCISE 15**
- CH5\_F6** - Counters and Accumulators basic info & e.g.
- CH5\_F7** - Pretest loop: **For...As dataType/... = ...To...Step ... Next** statement - specific counter-controlled loop
- CH5\_F8** - Pretest loop: create counter-controlled loop - compare **Do...Loop** statement vs **For...Next** statement
- CH5\_F9** - Pretest loop: **For...Next** statement - specific counter-controlled loop - Flowcharting
- CH5\_F9.1** - Pretest loop: **For...Next** statement - create Counter and Accumulator - **07.You Do It 3 Solution**

**CH5\_APPLY THE CONCEPTS LESSON**

- CH5\_A1** - Pretest loop: **Do...Loop** statement - use a **Loop**, **Counter**, and **Accumulator** - e.g. **08.Projected Sales Solution**
- CH5\_A1.1** - Pretest loop: **For...Next** statement - use a **Loop**, **Counter**, and **Accumulator** - e.g. **09.Projected Sales Solution-ForNext**
- CH5\_A2** - **ListBox** - add a (1st) to a **Form**: basic info & basic properties
- CH5\_A2.1** - **ListBox** - add items using the **String Collection editor** in **Properties** - e.g. **10.ListBox Solution**
- CH5\_A2.2** - **ListBox** - the property **Sorted** - e.g. **10.ListBox Solution**
- CH5\_A2.3** - **ListBox** - the **SelectedItem** and **SelectedIndex** properties - when **SelectionMode = One** - e.g. **10.ListBox Solution**
- CH5\_A2.3.1** - **ListBox** - the **SelectedItems** and **SelectedIndices** properties - when **SelectionMode = MultiSimple / MultiExtended** - **31.Multi Solution\_EXERCISE 16**
- CH5\_A2.4** - **ListBox** - the **SelectedValueChanged** and **SelectedIndexChanged** events - e.g. **10.ListBox Solution**
- CH5\_A3.1** - **ListBox** - Items Collection: method **Add** to add items - e.g. **11.ListBox Items Solution**
- CH5\_A3.2** - **ListBox** - Items Collection: method **Insert** to add item at a desired position during run time
- CH5\_A3.3** - **ListBox** - Items Collection: method **Remove** to remove item during run time
- CH5\_A3.4** - **ListBox** - Items Collection: method **RemoveAt** to remove item during run time
- CH5\_A3.5** - **ListBox** - Items Collection: property **Count** to get the number of items - e.g. **11.ListBox Items Solution**
- CH5\_A3.6** - **ListBox** - Items Collection: method **Clear** to remove all of the items - e.g. **11.ListBox Items Solution**
- CH5\_A4** - **Financial Class** - introduction: index of **Financial**. methods
- CH5\_A4.1** - **Financial Class** - calculate a periodic payment using **Financial.Pmt** method
- CH5\_A4.2** - **Financial.Pmt** method & **ListBox & Loop** to calculate monthly mortgage payment - e.g. **13.Payment Solution**
- CH5\_A4.3** - **Financial.FV** method **research** - calculate the future value of an annuity e.g.: **\_Appendix E - Case Projects\_04.Savings Calculator-CH 01-05**
- CH5\_A5** - Nested repetition structures / loops = outer loop contains nested/inner loop - basic info
- CH5\_A5.1** - Nested repetition structures / loops = outer loop contains nested/inner loop - e.g. using 2x Pretest loop **For...Next**, e.g. **14.Savings Solution**
- CH5\_A6** - a caution about **Real numbers** = decimals: **As Double** less accurate than **As Decimal** -> can cause wrong result, e.g. **15.Savings Solution-Caution**
- CH5\_A7** - professionalize your application's interface: the **default buttons** set in **frmMain**'s Properties: **AcceptButton** = Enter key, **CancelButton** = Esc key

**CH5\_Key Terms****CH5\_Exercises**

# Event & Sub & Function Procedures

Event-handling Sub Procedure (Event Procedure) - more in detail & multiple objects & multiple events;  
 Independent Sub Procedure (Sub Procedure) + Calling Statement - no parameter & passing variable: by Value - **ByVal** & by Reference - **ByRef**;  
 Function Procedure (Function) + Calling Statement; rounding numbers - method **Math.Round**; ComboBox tool (**cbo**); **frmMain\_FormClosing**;  
**MessageBox.Show** method & return value **DialogBox**

## CH6\_FOCUS ON THE CONCEPTS LESSON

- CH6\_F1 - Event-Handling Sub Procedures aka Event Procedures: more in a detail & **multiple** objects and events associated
- CH6\_F1.1 - How to connect **multiple objects** and **events** to the same **procedure**: modify the **Monthly Payment application** example + how to break a line of code
- CH6\_F2 - **Independent Sub Procedures** (1st in code) + **Calling Statements** (2nd in code):
- CH6\_F2.1 - **No Parameter/Arguments** example: History Grade Application
- CH6\_F3 - **Passing Information to a Procedure** & passing a variable **by value** & passing a variable **by reference**
- CH6\_F3.1 - Passing Variables **by Value** example: **Gross Pay Application** -> **ByVal** keyword
- CH6\_F3.2 - **You Do It 1:** Passing Variables **by Value** -> **ByVal** keyword
- CH6\_F3.3 - Passing Variables **by Reference** example: **Concert Tickets Application** -> **ByRef** keyword
- CH6\_F4 - **Rounding Numbers** - method **Math.Round**
- CH6\_F4.1 - to round the Subtotal and Discount amount example: **Concert Tickets Application** -> **Math.Round** method
- CH6\_F4.2 - **You Do It 2:** Passing Variables **by Address** -> **ByRef** keyword
- CH6\_F5 - **Function Procedures + Calling Statements**, vs event procedures & Sub procedures
- CH6\_F5.1 - Function example: code the **Concert Tickets Application** -> **Function** procedure
- CH6\_F5.2 - **Sub** procedure vs **Function** procedure : **Concert Tickets Application** examples to compare
- CH6\_F5.3 - **You Do IT 3:** using a **Function**

## CH6\_APPLY THE CONCEPTS LESSON

- CH6\_A1 - Add a Combo box to the form: **ComboBox** tool
- CH6\_A1.1 - to modify the **Monthly Payment application** - replace **ListBox** for a **ComboBox**:
- CH6\_A2 - Add Items to a **ComboBox** and Select a Default Item: Cerruti Company Payroll application example 01
- CH6\_A3 - Code a **Combo Box's KeyPress Event Procedure**: Cerruti Company Payroll application example 02
- CH6\_A4 - Create an **Event-Handling Sub Procedure** to clear all labels when user input changes: Cerruti Company Payroll application example 03
- CH6\_A5 - Calculate Federal Withholding Tax - **FWT**: Cerruti Company Payroll application example 04
- CH6\_A6 - Invoke an **Independent Sub Procedure** and a **Function** (btnCalc\_Click procedure): Cerruti Company Payroll application example 05
- CH6\_A7 - Create an **Independent Sub Procedure**: **GetSingleFwt** Sub procedure for Single employee: Cerruti Company Payroll application example 06
- CH6\_A8 - Create a **Function**: **GetMarriedFwt** function for Married employee: Cerruti Company Payroll application example 07
- CH6\_A9 - Validate an Application's Code: Cerruti Company Payroll application example 08
- CH6\_A10 - Professionalize Your Application's Interface: **Message Box** - **MessageBox.Show** method syntax, and its return value: **DialogResult**.
- CH6\_A10.1 - Message Box - **MessageBox.Show** method - confirmation to kill the Cerruti Company Payroll application example 09
- CH6\_A11 - Cerruti Company Payroll application - completed code & GUI with items example 10

## CH6\_Summary

## CH6\_Key Terms

## CH6\_Exercises

**CH7\_FOCUS ON THE CONCEPTS LESSON**

- CH7\_F1** - Length property of the String: `string.Length`
- CH7\_F1.1** - Length property of the String: `string.Length` example: The Product ID Application:
- CH7\_F2** - Insert method of the String: `string.Insert(startIndex, value)`
- CH7\_F3** - aligning characters in string & columns - methods: `PadLeft` & `PadRight` & function: `String.Space` - additional topic from **Appendix B**
- CH7\_F3.1** - Insert method & `PadLeft` method example: **The Net Pay Application** (02.Net Pay Solution)
- CH7\_F3.2** - **You Do It 1:** Trim method, `PadRight` method, Insert method exercise: 03.You Do It 1 Solution
- CH7\_F4** - Search for a specific sequence of characters: `Contains` and `IndexOf` methods of the String:
  - CH7\_F4.1** - String search: `Contains` method
  - CH7\_F4.2** - String search: `IndexOf` method
  - CH7\_F4.3** - String search: `IndexOf` method example: **The City and State Application** (04.City State Solution)
  - CH7\_F4.4** - **You Do It 2:** `Contains` method and `IndexOf` method exercise: 05.You Do It 2 Solution
- CH7\_F5** - Substring method of the String
  - CH7\_F5.1** - Substring method of the String example: **The Rearrange Name Application** (06.Rearrange Solution)
  - CH7\_F5.2** - **You Do It 3:** Substring method exercise: 07.You Do It 3 Solution
- CH7\_F6** - Character Array: using `Character's index` to access a `Character` in a String
  - CH7\_F6.1** - using `Character's index` to access a `Character` in a String example: **The First Name Application** (08. Name Solution)
  - CH7\_F6.2** - **You Do It 4:** `Character's index` exercise: 09.You Do It 4 Solution
- CH7\_F7** - Remove method of the String:
- CH7\_F8** - Trim, `TrimStart` and `TrimEnd` methods of the String:
  - CH7\_F8.1** - `TrimEnd` method example: **The Tax Calculator Application** (10. Tax Solution)
- CH7\_F9** - Replace method of the String:
- CH7\_F10** - Like Operator to compare Strings:
  - CH7\_F10.1** - Like Operator to compare Strings example: **The Inventory Application** (11. Inventory Solution)
  - CH7\_F10.2** - **You Do It 5:** Like operator to compare strings exercise: 12.You Do It 5 Solution

**CH7\_APPLY THE CONCEPTS LESSON**

- CH7\_A1** - Code the **Check Digit Application** - thoroughly step by step : (13.Check Digit Solution)
- CH7\_A2** - Code the **Password Application** - thoroughly step by step : (14.Password Solution)
- CH7\_A3** - Generate Random numbers - object `Random` & method `Next` for integers, or method `NextDouble` for doubles\_additional topic from **Appendix B**
  - CH7\_A3.1** - Generate Random Integers example: code the **Guess a Letter Application** (15.Letter Guess Solution) - part 1/3
- CH7\_A4** - use the `Enabled` property of the Control
- CH7\_A5** - use the `Focus` method of the Control
  - CH7\_A6** - Generate Random Integers example: code the **Guess a Letter Application** (15.Letter Guess Solution) - part 2/3
- CH7\_A7** - Generate Random Integers example: code the **Guess a Letter Application** (15.Letter Guess Solution) - entire code - part 3/3
- CH7\_A8** - many of the concepts from CH7 example: code the **Guess the Word Game Application** (16.Word Guess Solution) + entire code

**CH7\_Summary:** Check digits, Random integers, Enable property & Focus method of the Control, Concepts used for String manipulations

**CH7\_Key Terms**

**CH7\_Exercises**

## CH8\_FOCUS ON THE CONCEPTS LESSON

- CH8\_F1 - [Array](#) of variables ([Array](#)) introduction: One-Dimensional and Two-Dimensional [Arrays](#)
- CH8\_F2.1 - One-Dimensional [Array](#): introduction
- CH8\_F2.2 - declaring One-Dimensional [Array](#) - version 1 & version 2 autopsy
- CH8\_F2.3 - storing data in a [1D Array](#) autopsy
- CH8\_F2.4 - determing the number of elements in a [1D Array](#): [Array](#)'s [Length](#) property
- CH8\_F2.5 - determing the highest [subscript](#) in a [1D Array](#): property [Length](#) -1 or method [GetUpperBound\(0\)](#)
- CH8\_F2.6 - You Do It 1: [1D Array](#)'s [Lengt](#) property vs [GetUpperBound](#) method exercise: 01.You Do It 1 Solution
- CH8\_F2.7 - Traversing a [1D Array](#) = look at each element using [loop](#)
- CH8\_F2.8 - Traversing [1D Array](#) example: coding the [Harry Potter Characters Application](#) (02.Potter Solution)
- CH8\_F3.1 - [For Each...Next](#) loop statement - great for collection/array with unknown therefore variable number of elements
- CH8\_F3.2 - using [For Each...Next](#) loop example: coding the [Harry Potter Characters Application](#) (03.Potter Solution-ForEachNext)
- CH8\_F3.3 - Traversing [1D Array](#): [Do...Loop](#) vs [For...Next](#) vs [For Each...Next](#) code example with [Harry Potter Application](#):
- CH8\_F3.4 - You Do It 2: Traversing [1D Array](#): [Do...Loop](#) vs [For...Next](#) vs [For Each...Next](#) exercise: 04.You Do It 2 Solution
- CH8\_F4.1 - Calculating the Average [1D Array](#) value: calculate the average price of a company's stock example: 05.Waterson Solution-Average
- CH8\_F4.2 - You Do It 3: calculating the Total [1D Array](#) value using [Do...Loop](#) & [For...Next](#) & [For Each...Next](#) exercise: 06.You Do It 3 Solution
- CH8\_F5.1 - finding the Highest [1D Array](#) value: display the highest stock price and number of days example: 07.Waterson Solution-Highest
- CH8\_F5.2 - You Do It 4: finding the lowest [1D Array](#) value using [For...Next](#) statement exercise: 08.You Do It 4 Solution
- CH8\_F6.1 - Sorting a [1D Array](#): [Array.Sort\(\)](#) & [Array.Reverse\(\)](#) methods
- CH8\_F6.2 - Sorting a [1D Array](#): [Array.Sort\(\)](#) & [Array.Reverse\(\)](#) methods example: 09.Continents Solution
- CH8\_F7.1 - Two-Dimensional [Array](#): introduction
- CH8\_F7.2 - declaring [2D Array](#) - version 1 & version 2 autopsy
- CH8\_F7.3 - storing data in a [2D Array](#) autopsy
- CH8\_F7.4 - determing the highest [subscripts](#) in a [2D Array](#): method [GetUpperBound\(0\)](#) for row and [GetUpperBound\(1\)](#) for column
- CH8\_F7.5 - Traversing a [2D Array](#) = look at each element using loops: [Do...Loop](#) vs [For...Next](#) vs [For Each...Next](#) code examples
- CH8\_F7.6 - Traversing a [2D Array](#) example using [Do...Loop](#) & [For...Next](#) & [For Each...Next](#): coding the [Months Application](#) (10.Months Solution)
- CH8\_F7.7 - Totaling the values stored in a [2D Array](#) example: coding the [Jenko Booksellers Application](#) (11.Jenko Solution)
- CH8\_F7.8 - You Do It 5: Totaling [2D Array](#) value: [Do...Loop](#) vs [For...Next](#) vs [For Each...Next](#) exercise: 12.You Do It 5 Solution

## CH8\_APPLY THE CONCEPTS LESSON

- CH8\_A1.1 - Associate a 1D **Array** with a (1st) i.e. associate **Array** elements with **Items Collection**, for they have a several **commonalities**
- CH8\_A1.2 - Associate a 1D **Array** with a (1st) example: coding the **Presidents and Vice Presidents Application** (13.Presidents Solution)
- CH8\_A2.1 - create **1D Array Accumulator** and **1D Array Counter** associated with (1st) example: **Chocolate Bars Application** (14.Warren Solution)
- CH8\_A2.2 - **You Do It 6: 1D Array Accumulator & Counter** associated with (1st) using **For Each...Next & For...Next & Do...Loop**
- CH8\_A3.1 - create **2x Parallel 1D Arrays** introduction & example
- CH8\_A3.2 - create **2x Parallel 1D Arrays** example: coding the **Paper Warehouse Application** (16.Paper Solution-Parallel)
- CH8\_A4.1 - search **2D Array** introduction & example vs. **2x Parallel 1D Arrays**
- CH8\_A4.2 - search **2D Array** instead of **2x Parallel 1D Arrays** example: coding the **Paper Warehouse Application** (17.Paper Solution-TwoDim)
- CH8\_A5 - my addendum from EXERCISES: **Array** statement **ReDim** - 37.ReDim Solution\_EXERCISE 20\_advanced
- CH8\_A6 - using **LINQ** - Language-Integrated Query [dotaz] - to retrieve **values** from **Array** + e.g. **40.Linq Array Solution** - additional topic from **Appendix B**
- CH8\_A6.1 - using **LINQ's Aggregate operators** to retrieve a **single value** from an **Array** + **41.Linq Aggregate Array Solution** - additional topic from **Appendix B**
- CH8\_A7 - **Structure** - composite of variables like **array**, but with a combination of a different data types - additional topic from **Appendix B**
- CH8\_A7.1 - compare passing variables to a procedure: **several separated** vs. **only 1 structure** - **42.Norbert Solution** - additional topic from **Appendix B**
- CH8\_A7.2 - **array of structure** variables = each **array element** contains **member** variables created within **Structure** - **43.Paper Solution-Structure - Appendix B** & comparison with 2x parallel **1D array** used in **16.Paper Solution-Parallel**, and with **1x 2D array** used in **17.Paper Solution-TwoDim**

**CH8\_Summary:** **1D Array**, **For Each...Next** loop statement, **Array.Sort** & **Array.Reverse** methods, **2D Array**, **Parallel 1D Arrays / 2D Array**

**CH8\_Key Terms**

**CH8\_Exercises**

- in addition to getting data from the keyboard and sending data to the computer screen, an application can also read data from and write data to a file on a disk
- in this chapter's Focus on the Concepts lesson, you will learn how to create and use a special type of file, called a sequential access file
- you will use sequential access files in the applications coded in this chapter's Apply the Concepts lesson
- the Apply lesson also covers the creation and coding of menus

#### CH9\_FOCUS ON THE CONCEPTS LESSON

- CH9\_F1** - File Handling - Sequential Access Files / Text files introduction: **Namespace System.IO**
- CH9\_F1.1** - **Namespace System.IO** and its **Classes** - basic info including messages(exceptions)
- CH9\_F2** - Sequential Access **Output** Files: **Class StreamWriter** - how to create and use them step by step
- CH9\_F2.1** - **Output Text** Files: **Class StreamWriter** example: code the **Game Show Application** (01.Game Show Solution-TextBox) 1/2
- CH9\_F3** - Sequential Access **Input** Files: **Class StreamReader** - how to create and use them step by step
- CH9\_F3.1** - **Input Text** Files: **Class StreamReader**: **ReadToEnd** function example: code the **Game Show Application** (01.Game Show Solution-TextBox) 2/2
- CH9\_F3.2** - Sequential Access **Input** Files: **Class StreamReader**: **ReadLine** function example: code the **Game Show Application** (02.Game Show Solution-ListBox)
- CH9\_F4** - **You Do It 1:** Sequential Access **Output & Input** file exercise: 03.You Do It 1 Solution  
-> AppendText, WriteLine, OpenText, ReadToEnd, Peek, ReadLine, Close, \_MouseClick

#### CH9\_APPLY THE CONCEPTS LESSON

- CH9\_A1** - Add a **Menu(mnu)** to a **Form: Toolbox / Menus & Toolbars / ToolStrip** control
- CH9\_A1.1** - **GUI** guidelines for **Menus(mnu)**:
- CH9\_A2** - **Menu** example: **Continents Application** - create Sequential Access File (**.txt**) to fill **Array** with values (04.Continents Solution) 1/4
- CH9\_A2.1** - **Menu** example: **Continents Application** - add **Menus** to the **GUI** (04.Continents Solution) 2/4
- CH9\_A2.2** - **Menu** example: **Continents Application** - code the **Items** on a **Menu** (04.Continents Solution) 3/4
- CH9\_A2.3** - **Menu** example: **Continents Application** - modify a **Menu** (04.Continents Solution) 4/4
- CH9\_A3** - **Accumulate the Values** Stored in a **File** example: **05.Harkins Solution**
- CH9\_A4** - easy way to **Sort** the **Data** contained in a **File** example: **06.States Solution**
- CH9\_A5** - **Professionalize Your Application's Interface:** **confirmation message** when the **file** is written (07.Game Show Solution-Professionalize)

**CH9\_Summary:** Sequential access files, **MenuStrip** tool, easy sorting of Sequential access file, confirmation message

**CH9\_Key Terms**

**CH9\_Exercises**

- you already are familiar with **Classes** and **Objects** because you have been using them since CH1 ->
  - > e.g. you used VB's **Class Button** to create a **button Object** and used its **Class String** to create a **String** variable
- in this chapter's Focus on the Concepts lesson, you will learn how to define your own **Classes** and then use them to instantiate **Objects** in an application
- the Apply the Concepts lesson expands on the topics covered in the Focus lesson

#### CH10\_FOCUS ON THE CONCEPTS LESSON

- CH10\_F1** - Object-Oriented Programming / OOP basic info and e.g.: **Class** and instance of the **Class** called **Object**
- CH10\_F2** - creating a **Class** - basic info
- CH10\_F2.1** - creating and adding a **Class file** to an open project: **Class's member** variables & **Property** section & **Behavior** section basic info
- CH10\_F3** - **Class's member** variables: info & syntax ; **Property** section of a **Class**: **Property** procedures: info & syntax
- CH10\_F3.1** - **Property** section of a **Class** example: creating the **Rectangle Class** for the **Franklin Decks app** (01.Franklin Solution) **1/3**
- CH10\_F4** - **Behavior** section of a **Class** info: usually **methods**: **Sub** procedures or **Functions**
- CH10\_F4.1** - **Behavior** section of a **Class**: **Sub** procedure method **Constructor** (default & parameterized) - info & syntax & e.g.
- CH10\_F4.2** - **Behavior** section of a **Class**: other **methods** than **Constructor** - **Function** procedure, **Sub** procedure - infos & syntaxes & examples
- CH10\_F4.3** - **Behavior** section of a **Class** example: complete the **Rectangle Class** for the: **Franklin Decks app** (01.Franklin Solution) **2/3**
- CH10\_F5** - instantiating an **Object** and invoking the **constructor** - info, syntax and example
- CH10\_F5.1** - using the **Rectangle Class** in the application example: pseudocode & GUI for the: **Franklin Decks app** (01.Franklin Solution) incl. entire code **3/3**
- CH10\_F6** - **You Do It 1:** create and use **Class** exercise: **02.You Do It 1 Solution** -> **Public Property**, default **constructor**, **function**
- CH10\_F7** - adding a **parameterized Constructor** example: a **Rectangle Class** of **03.Franklin Solution-Parameterized** **1/5**
- CH10\_F7.1** - comparing **default** vs **parameterized Constructor** example: **Rectangle Class** of the **03.Franklin Solution-Parameterized** **2/5**
- CH10\_F7.2** - using/invoking a **parameterized Constructor** example: a **btnCalc\_Click** procedure of **03.Franklin Solution-Parameterized** **3/5**
- CH10\_F7.3** - analyse the process of using/invoking a **parameterized Constructor**: a **btnCalc\_Click** procedure of **03.Franklin Solution-Parameterized** **4/5**
- CH10\_F7.4** - the entire code for a **parameterized Constructor** example and testing: **03.Franklin Solution-Parameterized** **5/5**
- CH10\_F8** - **You Do It 2:** create and use **Class** exercise: **04.You Do It 2 Solution** -> **Public Property**, parameterized **constructor**, **function**
- CH10\_F9** - reusing a **Class** example: previously created **Class Rectangle** used in: **05.Pizzeria Solution** - GUI & pseudocode & entire code & test

#### CH10\_APPLY THE CONCEPTS LESSON

- CH10\_A1** - use a **ReadOnly Property** procedure in a **Class** example: complete the **CourseGrade Class** used in: **06.Grade Solution** **1/2**
- CH10\_A1.1** - use a **ReadOnly Property** procedure in a **Class** example: complete the **frmMain Class** used in: **06.Grade Solution** **2/2**
- CH10\_A2** - create **auto-implemented Public Property**: it automatically creates hidden **Private** member variable: **\_variable** & hidden **Get** & **Set** block of codes
- CH10\_A2.1** - use **auto-implemented Public Property**: example with **CourseGrade Class** used in **07.Grade Solution-autoimplemented**
- CH10\_A3** - **You Do It 3:** create and use **auto-implemented Public Property** exercise: **08.You Do It 3 Solution**
- CH10\_A4** - use an **overload method** / a method with several signatures in a **Class** - info and example with **Employee Class**
- CH10\_A4.1** - use an **overload method** / a method with several signatures in a **Class** - example with **Employee Class** used in **09.Woods Solution**

**CH10\_Summary:** **Class** statement, instantiating an **Object** from a **Class**, **Property Procedure**, default and parameterized **Constructor**, invoking the **Constructor**, other methods than **Constructor** - **Function** procedure & **Sub** procedure, **auto-implemented Public Property**, **overload** the methods.

**CH10\_Key Terms**

**CH10\_Exercises**

Try...Catch "safety net" statement = handling/catching app's exception/error during run time without terminating the app

- most businesses need to keep track of vast amounts of information pertaining to their customers, employees, and inventory
- although a business could store the information in sequential access files, which you learned about in **Chapter 9**, it is much easier to manipulate the information when it is stored in computer databases
- in this chapter's **Focus on the Concepts lesson**, you will learn how to create computer databases and then use them in applications
- the **Apply the Concepts lesson** expands on the topics covered in the Focus lesson

### MySummary:

<b>CH11_FA</b>	- Create db with <b>1 table</b> & create <b>dataset</b> & use <b>DataGridView</b> control & include <b>Try...Catch</b> "safety net" statement into <b>Binding Navigator</b> 's events:
	1.create db & 2.create Table & 3.add Records & 4.create DataSet & 5.create bound control & 6.improve DataGridView control & 7.set db Copy... property & 8.include <b>Try...Catch</b> 'safety net' statement into Binding Navigator control
<b>CH11_FB</b>	- import db with <b>2 tables</b> & create <b>dataset</b> & use <b>DataGridView</b> control:
	1.add existing db containing 2 tables to dataset & 2.relate 2 tables by their common field & 3.create Query / custom dataset & 4.create bound control & improve DataGridView control
<b>CH11_AA</b>	- import db & create <b>dataset</b> & create <b>Data Form</b> (separate control for each table's field) instead of using <b>DataGridView</b> control &
	& add <b>Try...Catch</b> 'safety net' statement & KeyPress example: <b>03.Course Info Solution-Data Form</b>
<b>CH11_AB</b>	- already: imported db, created dataset & <b>Bind</b> table's field objects to <b>existing</b> Label controls by <b>dragging</b> them &
	& add <b>BindingNavigator</b> control example: <b>04.Course Info Solution-Labels</b>
<b>CH11_AC</b>	- already: imported db, created dataset, created DataGridView bound control & <b>Code</b> the <b>btnCalc_Click</b> procedure to:
	Loop all the rows/records and accumulate numbers if the field is not empty nor contains letter W example: <b>05.Course Info Solution-Total Hours</b>

### CH11\_FOCUS ON THE CONCEPTS LESSON

<b>CH11_F0</b>	- <b>SQL Server Database</b> : basic info & terminology & e.g. single-table db, two-table db
<b>CH11_F0.1</b>	- <b>SQL Server data types</b> : comparison with Visual Basic data types basic info: <b>bit</b> , <b>decimal(p, s)</b> , <b>float</b> , <b>int</b> , <b>char(n)</b> , <b>varchar(n)</b>
<b>CH11_F1</b>	- step <b>1/8</b> : create a <b>SQL Server Database</b> with <b>.mdf</b> extension: step-by-step example <b>01.Course Info Solution</b>
<b>CH11_F2</b>	- step <b>2/8</b> : add a <b>Table</b> to a <b>Database</b> and define it: step-by-step example <b>01.Course Info Solution</b>
<b>CH11_F3</b>	- step <b>3/8</b> : add <b>Records</b> to a <b>Table</b> : step-by-step example <b>01.Course Info Solution</b>
<b>CH11_F4</b>	- step <b>4/8</b> : create <b>Dataset</b> = connection between <b>Database</b> and <b>Application</b> - using <b>Data Source Configuration Wizard</b> &
	& view contents of your <b>dataset</b> : example <b>01.Course Info Solution</b>
<b>CH11_F5</b>	- info for <b>step 5/8</b> : about <b>Binding</b> /connecting an object in <b>DataSet</b> to the new or existing <b>control</b> in GUI: basic info & example with a <b>new control</b>
<b>CH11_F5.1</b>	- step <b>5/8 Bind</b> an object <b>Courses</b> table to default <b>DataGridView</b> control example <b>01.Course Info Solution</b>
<b>CH11_F6</b>	- info for <b>step 6/8</b> : about <b>DataGridView control</b> for displaying table data - basic info & example
<b>CH11_F6.1</b>	- step <b>6/8</b> : improve the appearance of the <b>DataGridView control</b> - set the <b>columns</b> & its <b>cells</b> example <b>01.Course Info Solution</b>
<b>CH11_F7</b>	- info for <b>step 7/8</b> : set the local database file's <b>*.mdf</b> property: <b>Copy to Output Directory</b> = correlation between source <b>*.mdf</b> and output <b>*.mdf</b>
<b>CH11_F7.1</b>	- step <b>7/8</b> : set the local database file's <b>*.mdf</b> property: <b>Copy to Output Directory</b> - example and testing <b>01.Course Info Solution</b>
<b>CH11_F8</b>	- info for <b>step 8/8</b> : automatically generated code when <b>binding</b> objects - autopsy & example with <b>01.Course Info Solution</b>
<b>CH11_F8.1</b>	- info for <b>step 8/8</b> : <b>Try...Catch</b> "safety net" statement = handling/catching app's <b>exception/error</b> during run time without terminating the app & example
<b>CH11_F8.2</b>	- step <b>8/8</b> : include a <b>Try...Catch</b> "safety net" statement to handle/catch possible exceptions/errors example <b>01.Course Info Solution</b>

CH11\_FB

CH11\_F9 - explore the **Two-table** database: basic info with example **02.Charleston Sales Solution**

CH11\_F9.1 - step 1/4: add an existing database file **Charleston.mdf** to an application & create a **dataset** &

& add it's **2 tables** to the **dataset** example **02.Charleston Sales Solution**

CH11\_F9.2 - step 2/4: relate 2 tables by their **common field**: **CountryCode** example **02.Charleston Sales Solution**

CH11\_F9.3 - step 3/4: create a **Database Query object** from **2 table objects** related by their common field, using **foreign & primary keys** =

= **single dataset** as a **combination** of fields from **2 tables** - info & example **02.Charleston Sales Solution**

CH11\_F9.4 - step 4/4: display the **Database Query/Query** information in the **DataGridView** control = create a bound control from your "custom" dataset &

& improve control's appearance - info & example **02.Charleston Sales Solution**

## CH11\_APPLY THE CONCEPTS LESSON

CH11\_A1 - Create a **Data Form** (instead of using **DataGridView** control) for displaying **dataset**, created from existing \*.mdf database file &

CH11\_AA

& code: add the **Try...Catch** 'safety net' statement & KeyPress procedures example: **03.Course Info Solution-Data Form**

CH11\_A2 - **Bind Field** objects to **existing** controls - basic info

CH11\_A2.1 - **Bind Field** objects to **existing** controls by dragging them to the form & add **BindingNavigator** control example: **04.Course Info Solution-Labels**

CH11\_AB

CH11\_A3 - perform **Calculations** on the **Fields** in a Dataset using loop: **For Each row As MyCoursesDataSet.CoursesRow In MyCoursesDataSet.Courses.Rows**

info & example: **05.Course Info Solution-Total Hours**

CH11\_AC

CH11\_Summary:

CH11\_Key Terms & terminology marked: ->

CH11\_Exercises

# Database Queries with SQL

= retrieve a specific information from a database

Parameter Query & Parameter marker @; use SQL to add a Calculated field; use SQL Aggregate functions

- in CH11, you learned how to:
  - create a SQL Server database
  - perform common database tasks: - adding, editing, deleting, saving records, performing calculations on fields
- in CH12, you will learn how to:
  - perform another common database task - create the database **queries** using Structured Query Language - **SQL**
  - use **SQL** to add a calculated field to a dataset
  - use the **SQL aggregate** functions.

- for an application to use a **query** during run time, you need to:
  - 1). **create** the query - by using the **TableAdapter Query Configuration Wizard**
  - 2). **save** the query - by associating the query with one or more methods (Fill a DataTable, Return a DataTable - methods **Fill**, **Get**)
  - 3). **invoke** it from code - by entering in procedure the **Fill** method associated with the query

default Fill method's calling/invoking statement:

`YourTableAdapter.Fill(YourDataSet.YourTable)`

parameterized Fill method's calling/invoking statement:

`YourTableAdapter.YourParameterizedMethodFill(YourDataSet.YourTable, YourParameter)`

## CH12\_FOCUS ON THE CONCEPTS LESSON

- CH12\_F1 - **SELECT** statement syntax & its **WHERE** clause's condition operators & e.g.
- CH12\_F2 - how to create a **Query** matching specific criteria, using **Query Builder** dialog box - several examples with **01.Oscars Solution-SELECT**
- CH12\_F2.1 - **You Do It 1:** practice creating **Query** using **Query Builder** dialog box - 3 examples from Mini-Quiz 12-1 with **02.You Do It 1 Solution**
- CH12\_F3 - **Parameter Query & Parameter Marker @** - basic info & e.g.
- CH12\_F3.1 - how to use a **Parameter Query & parameter marker @** - using 3x e.g. from previous lesson, with solution **03.Oscars Solution-Parameter Queries**
- CH12\_F3.2 - **You Do It 2:** practice creating **Parameter Queries** using **Query Builder** dialog box - 3 examples from Mini-Quiz 12-2 with **04.You Do It 2 Solution**
- CH12\_F4 - use **Parameter Query** during run time - step 1/2: **Create** and then **Save** Query by associating it with a **method** - e.g. **05.Oscars Solution-Save Query**
- CH12\_F5 - use **Parameter Query** during run time - step 2/2: **Invoke/Call a method** associated with a **Query** - syntax, e.g. **05.Oscars Solution-Save Query**
- CH12\_F6 - **My Test:** - use **Parameter Query** with a **parameter marker @** during run time with a **LIKE** operator & allow users to use its wildcards %, \_
  - plus Q & A on several FB programmer's groups about **LIKE** operator
  - e.g. with: **05.Oscars Solution-Save Query\_My Test - Parameter Query and LIKE operator**

## CH12\_APPLY THE CONCEPTS LESSON

- CH12\_A1 - add a **Calculated Field** to a Dataset by modifying the default **SELECT** statement / using **Query Builder** e.g. with **06.Ellington Solution**
- CH12\_A2 - use the **SQL Aggregate Functions** like **AVG**, **COUNT**, **MAX**, **MIN**, **SUM** = return a single value from a group of values - basic info & e.g. using **SUM**:  
 create additional **TableAdapter** in the **DataSet**, whose modified default **SELECT** statement  
 will **create a new field** containing a **SUM** of the cells - e.g. with **07.Ellington Solution-Aggregate**

**CH12\_Summary:**

**CH12\_Key Terms & terminology marked:** ->

**CH12\_Exercises**

# Web Site Applications using "ASP.NET Web Forms Site" template

Site.master; 2x Static Web pages: Default.aspx & About.aspx; Dynamic Web page: Calculator.aspx recycled from unused Contact.aspx  
 Start Without Debugging Ctrl+F5; user input Validation Tools

**Figure 13-7** 4 of the files included in the ASP.NET Web Forms Site template:

filename	purpose
About.aspx	contains information about the website, such as its history or purpose
Contact.aspx	provides contact information, such as a phone number or e-mail address
Default.aspx	serves as the home page
Site.master	creates a consistent layout for the pages in your application (for example, the same color, header, footer, and buttons)

**Figure 13-53** Toolbox / Validation controls:

CH13\_A4 - ...

CompareValidator	compare an entry with a constant value or with a control's property
CustomValidator	verify that an entry passes the specified validation logic
RangeValidator	verify that an entry is within the specified minimum and maximum values
RegularExpressionValidator	verify that an entry matches a specific pattern
RequiredFieldValidator	verify that a control contains data
ValidationSummary	display all of the validation error messages in a single location on a Web page

## CH13\_FOCUS ON THE CONCEPTS LESSON

CH13\_F1 - basic web terminology & ASP.NET basic info

CH13\_F2 - template **ASP.NET Web Forms Site** to create a Web Site Application (using files: **About.aspx**, **Contact.aspx**, **Default.aspx**, **Site.master**)  
 (VS 2017 v 15.9.36 & .NET Framework 4.5.2) e.g. with **01.Fishbowl 1/9**

CH13\_F3 - how to add **Start Without Debugging** option to the **Debug** menu in **VS 2017** step by step instructions &  
 & starting/testing a **Web Application** in VS 2017 v 15.9.36 **without debugging** option (**Ctrl + F5**) e.g. with **01.Fishbowl 2/9**

CH13\_F4 - modify the **0th** page - master page **Site.master** (the page creates a consistent layout for the pages in your app = affects all of the **.aspx** pages)  
 & brief info about tags: **<p>**, **<ul>**, **<li>**, **<% %>**, e.g. with **01.Fishbowl 3/9**

CH13\_F5 - create/personalize the **1st** page - **Static** Web page **Default.aspx** (Home page), e.g. with **01.Fishbowl 4/9**

CH13\_F6 - create/personalize the **2nd** page - **Static** Web page **About.aspx**, e.g. with **01.Fishbowl 5/9**

CH13\_F7 - starting/testing **Web Application** in VS 2017 v 15.9.36 with different browsers

CH13\_F8 - closing and opening a Web Site Application - VS 2017 book version vs updated VS 2017 v 15.9.36

## CH13\_APPLY THE CONCEPTS LESSON

CH13\_A1 - create **3rd** page - **Dynamic** Web page **Calculator.aspx**: 1/4 Repurpose an existing unused **Contact.aspx**, e.g. with **01.Fishbowl 6/9**

CH13\_A2 - create **3rd** page - **Dynamic** Web page **Calculator.aspx**: 2/4 add a **Table** and **Controls**, e.g. with **01.Fishbowl 7/9**

CH13\_A3 - create **3rd** page - **Dynamic** Web page **Calculator.aspx.vb**: 3/4 code a **Button Control**, e.g. with **01.Fishbowl 8/9**

CH13\_A4 - create **3rd** page - **Dynamic** Web page **Calculator.aspx**: 4/4 use a **Validation Control: RequiredFieldValidator** - basic info, e.g. with **01.Fishbowl 9/9**

CH13\_X1 - **01.Fishbowl** application code & GUI for the pages:

Site.master & Site.master.vb & Default.aspx & Default.aspx.vb & About.aspx & About.aspx.vb & Calculator.aspx & Calculator.aspx.vb

CH13\_X2 - **ASP.NET Web Forms Site** template - automatically generated code for the pages:

Site.master & Default.aspx & About.aspx & Contact.aspx & Contact.aspx.vb

CH13\_Summary

CH13\_Key Terms

CH13\_Exercises

## Syntax errors &amp; Error List window

**red squiggle** = syntax error - alerts you of syntax error  
**green squiggle** = warning - warns you of a potential problem



## Logic errors

- 1a). **F8** to run **Step Into Debugger**
- 1b). **F9** set a Breakpoint + **F5** run the application
- 2). compare variable's and control's expected values with an actual values

when debugging your code, always correct the **syntax errors** first  
because doing so will often remove any **warnings**

Appendix C\_01 - **Syntax Errors** & VS's Error List window - errors caught by Code Editor - **red** & **green** squiggle, e.g. with **01.Total Sales Solution**

Appendix C\_02 - **Logic Errors** & VS's **Debugger**: menu's **Debug / Step Into F8** option to step through code, e.g. with **02.Discount Solution**

Appendix C\_03 - **Logic Errors** & VS's **Debugger**: menu's **Debug / Toggle Breakpoint F9** option to set **Breakpoint**, e.g. with **03.Hours Worked Solution**

Appendix C\_04 - **Run Time Errors** - errors occurring while an application is running, e.g. with **04.Quotient Solution**

## Appendix C\_Summary

Appendix C\_Terminology marked:

## Appendix C\_Exercises

## Appendix F

# SDI, MDI, TDI - multiple forms & dialog boxes

SDI app - main form & splash screen, main form & dialog box, 3x main form

MDI app - 2x primary form = Parent & Child; quasi TDI app - main form & TabControl tool

Appendix F\_00 - SDI, MDI, and TDI applications - basic info & e.g.

Appendix F\_01 - SDI application - main form & splash screen form - e.g. with **01.Country Solution**

Appendix F\_02 - SDI application - main form & dialog box - info about **Tool dialog box & Custom dialog box** - e.g. with **Tool dialog boxes: 02.Font Color Solution**

Appendix F\_03 - SDI application - 3x main/primary form - how to: create them & step between them & share data between them & close entire solution  
e.g. with **03.Zappet Solution**

Appendix F\_04 - MDI application - 2x main/primary forms separated into: **Parent** Form/window/GUI & its **Child** Forms/windows/GUIs  
e.g. with **04.JotPad Solution** (VS2022 & .NET 6.0)

Appendix F\_05 - quasi TDI application - main form & TabControl tool step by step, e.g. with **05.Currency Converter Solution** (VS2022 & .NET 6.0)

### Appendix F\_Key Terms

how to create:	1). open <b>Add New Item</b> dialog box: menu <b>Project / Add New Item...</b> <b>Ctrl+Shift+A</b>
Primary form:	2). on the left pane: <b>Installed / Common Items / Windows Forms</b>
Custom Dialog Box:	3a). in the middle: <b>Windows Form</b> with a default name: <b>Form2.vb</b>
Splash screen:	3b). in the middle: <b>Dialog</b> with a default name: <b>Dialog1.vb</b>
	3c). in the middle: <b>Splash Screen</b> with a default name: <b>SplashScreen1.vb</b>

### share data between forms:

1). declare Friend variable in form:

```
Friend FriendVariableName As dataType = expression
```

2). refer to it from another form:

```
reference = OriginFormName.FriendVariableName
```

form's property Title bar text

```
formName.Text = TitleBarText
```

form's method Close

```
formName/Me.Close()
```

form's method Show

```
formName/Me.Show()
```

form's method Hide

```
formName/Me.Hide()
```

**CH1\_FOCUS ON THE CONCEPTS LESSON**

- CH1\_F1** - Computer Programming Terminology
- CH1\_F2** - The programmer's Job
- CH1\_F3** - The Visual Basic Programming Language
- CH1\_F4** - The Visual Studio IDE: the Einstein's Equation application example
- CH1\_F5** - Assigning "(Name)" for coded or referred Objects

**CH1\_APPLY THE CONCEPTS LESSON**

- CH1\_A1** - Start and Configure Visual Studio Community 2017
- CH1\_A2** - To Create a Windows Forms Application: the Einstein's Equation application example
- CH1\_A3** - Manage the Windows in the IDE
- CH1\_A3.1** - to close, open, auto-hide, and display windows in the IDE : the Einstein's Equation application example
- CH1\_A4** - Change a Form File's Name
- CH1\_A5** - Change the Properties of a Form
- CH1\_A5.1** - to display the form's properties and the most commonly used
- CH1\_A6** - Save a Solution
- CH1\_A7** - Close and Open a Solution
- CH1\_A8** - Add a Control to a Form: the Einstein's Equation application example
- CH1\_A8.1** - To add a picture box to the Form: the Einstein's Equation application example
- CH1\_A8.2** - To add a button to the Form: the Einstein's Equation application example
- CH1\_A9** - Use the Format Menu
- CH1\_A9.1** - To adjust the Exit button's height and top border: the Einstein's Equation application example
- CH1\_A10** - Lock the Controls on the Form
- CH1\_A11** - Before you Start and End an Application - Verify the Startup Form
- CH1\_A11.1** - To verify the startup form: the Einstein's Equation application example
- CH1\_A11.2** - Start and End an Application
- CH1\_A11.3** - To change the name of the executable file, and then start and end the application: the Einstein's Equation application example
- CH1\_A12** - Enter Code and Comments in the Code Editor Window
- CH1\_A12.1** - To open the Code Editor Window
- CH1\_A12.2** - To collapse and expand a region of code
- CH1\_A12.3** - To select the btnExit control's Click event
- CH1\_A12.4** - the Me.Close() Statement
- CH1\_A12.5** - to code the btnExit\_Click procedure
- CH1\_A12.6** - Assignment Statements
- CH1\_A12.7** - Comments
- CH1\_A12.8** - To enter a comment and code: the Einstein's Equation application example
- CH1\_A13** - Print an Application's Code and Interface
- CH1\_A14** - Exit Visual Studio and Run an Executable File
- CH1\_A15** - **Splash Screen** - additional topic from **Appendix B** - template & code
- CH1\_A16** - Timer control - additional topic from **Appendix B** - code example
- CH1\_A17** - PrintForm Control - additional topic from **Appendix B** - code example; installing Visual Basic PowerPack

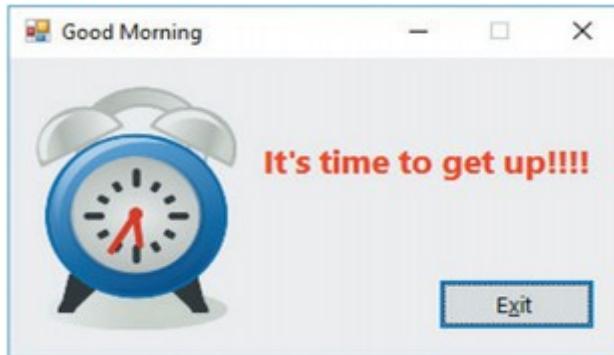
**CH1\_Summary****CH1\_Key Terms**

## CH1\_FOCUS ON THE CONCEPTS LESSON

### CH1\_F1 - Computer Programming Terminology:

- in essence, the word **programming** means: giving a mechanism the directions to accomplish a task ->
  - > when the mechanism is a computer, the directions are typically referred to as instructions
- a set of instructions that tells a computer how to accomplish a task is called a **computer program**, or more simply, a **program**
- programs are written by programmers using a variety of special languages called programming languages
  - some popular programming languages are Visual Basic, C#, C++, and Java
- you will write programs using the Visual Basic programming language, which is built into Microsoft's integrated development environment: Visual Studio 2017
- integrated development environment - **IDE** is an environment that contains all of the tools and features you need to create, run, and test your programs
- you also will use the **IDE** to create graphical user interfaces for your programs
- a graphical user interface - **GUI** is what the user sees and interacts with while your program is running
- the user interface and its program instructions are referred to as an **application**

e.g.: **Good Morning application**



GUI for Good Morning application

```
1  Public Class frmMain
2      Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
3          Me.Close()
4      End Sub
5
6      Private Sub tmrGetUp_Tick(sender As Object, e As EventArgs) Handles tmrGetUp.Tick
7          ' Blink the message
8
9          lblMessage.Visible = Not lblMessage.Visible
10     End Sub
11 End Class
```

code for Good Morning application, using **IDE** Visual Studio

### CH1\_F2 - The Programmer's Job:

- when a company has a problem that requires a computer solution, typically it is a programmer who comes to the rescue
  - the programmer might be an employee of the company; or he/she might be a freelance programmer, who works on temporary contracts rather than for a long-term employer
- first, the programmer meets with the person/people responsible for describing the problem
  - this person might be the one who will eventually use the solution
  - or he/she might be a software developer, who serves as an intermediary between the user and the programmer
  - the software developer will meet with the user and then outline the problem specification for the programmer
  - after the programmer understands the problem, he/she will begin planning an appropriate solution
  - after the planning is complete, the programmer will translate the solution into computer instructions - a process called **coding**
  - the programmer then will test the program rigorously (pečlivě) with sample data to make sure it works both correctly and to the user's satisfaction
  - depending on the complexity of the problem, multiple programmers might be involved in the planning and coding phases
  - programming teams often contain subject-matter (téma, podstata, předmět) experts, who might or might not be programmers
  - e.g.: an accountant might be part of a team working on a program that requires accounting expertise

### **CH1\_F3 - The Visual Basic Programming Language:**

- Visual Basic is an **object-oriented programming language**, which is a language that allows the programmer to use objects to accomplish a program's goal
- in **OOP**, an object is anything that can be:
  - seen
  - touched
  - used
- i.e. an object is nearly any thing
- programs written for the Windows environment typically use objects such as check boxes, list boxes, and buttons
- every object in **OOP** is created from a class, which is a pattern (předloha, šablona) that the computer uses to create the object
- the class contains the instructions that tell the computer how the object should look and behave
- an object created from a class is called an instance of the class and is said to be instantiated from the class
- e.g. - an analogy involving a cookie cutter, and cookies is often used to describe a class and its objects:
  - the cookie cutter is the class
  - and the cookies are the objects instantiated
- you will learn more about classes and objects throughout this book

info z netu o OOP, co to je CLASS, OBJECT :

- Class**
- třída; objektový typ
  - skupina objektů se stejnými kvalitativními vlastnostmi
  - např. všechny Objekty třídy PERSON mají vlastnost NAME, nicméně to jméno je pro každého jiný
  - třída je šablonou objektu, volání třídy se pak označuje jako instance
  - třída objektu definuje všechny vlastnosti objektu a jejich přípustný hodnoty nebo rozsahy, definuje taky funkce objektu - jeho chování a reakce na okolí
- Object**
- svět je objektově orientovaný
  - základní myšlenkou OOP je pohled na program jako na systém objektů
  - konkrétní jedinec příslušné třídy
  - mají vlastnosti jako PROMĚNNÉ a METODY, které je potřeba deklarovat
  - na rozdíl od procedurálního programování, v OOP jsou data a funkce navzájem svázány do struktury, zvané OBJEKT
  - pojmy INSTANCE a OBJEKT jsou v podstatě synonyma, často však mluvíme o konkrétním objektu jako o instanci,
    - termín objekt se používá, pokud mluvíme o objektech obecně
  - objekt je vlastně vyjádřením třídy v reálném čase
  - objekt se skládá z vlastností (atributy) a operací-funkcí (metody), které s ním lze provádět a v podstatě určují vlastní chování objektu

## CH1\_F4 - The Visual Studio IDE: the Einstein's Equation application example

- contains many different windows, each with its own special purpose
- the **four** windows you will use most often when **designing** your GUI: Toolbox, designer, Solution Explorer, Properties

Figure 1-5

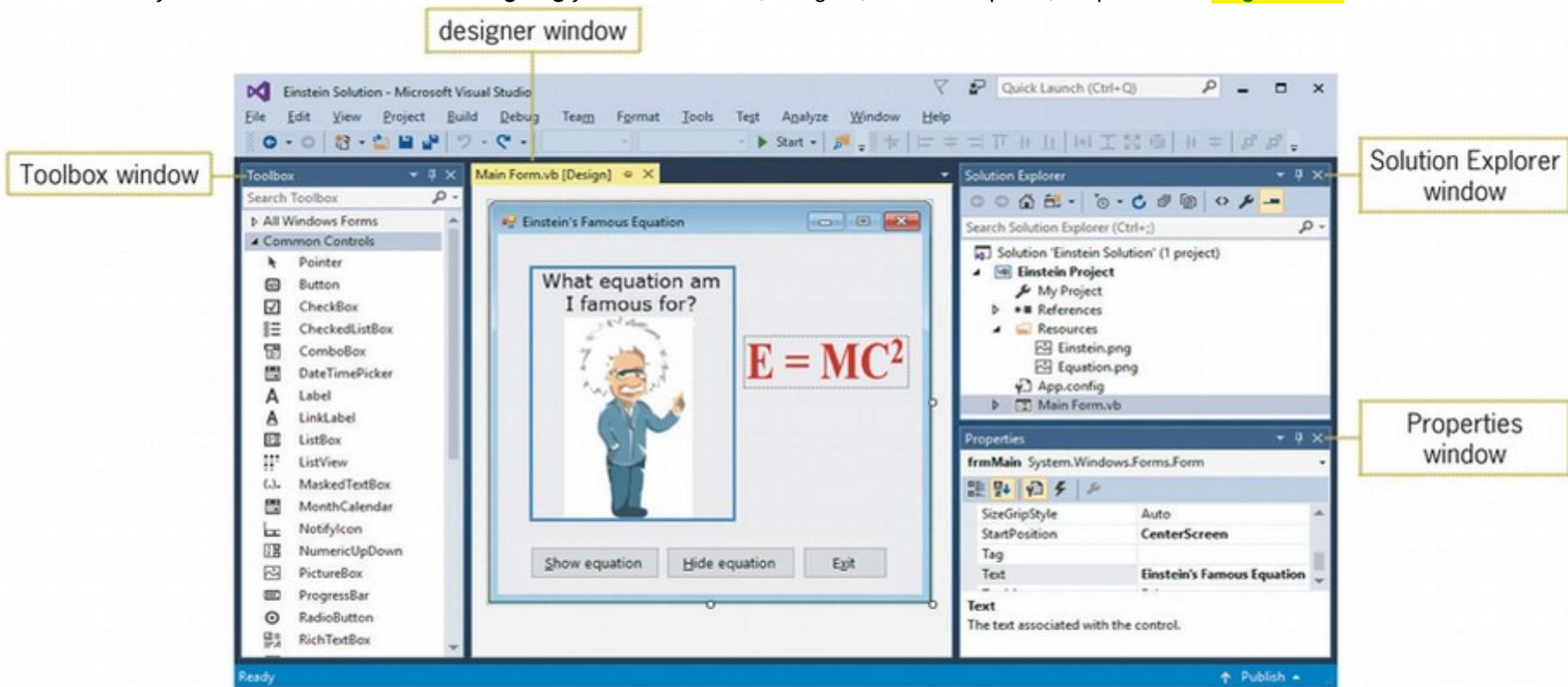


Figure 1-5 Visual Studio IDE

**designer window:** - is where you create (or design) your application's GUI

- a Windows Form object, or form, appears in the designer window
- a form is the foundation for the user interface in an application created for the Windows environment

- as you learned earlier, all objects in an OOP are instantiated (created) from a class

- a form, for example, is an instance of the Windows Form class

- the form (an object) is automatically instantiated for you when you create a Windows Forms application in Visual Basic

**Toolbox window:** - is used to add other objects, called **controls**, to the form

- each tool listed in the Toolbox window represents a class
- you add an object by clicking its corresponding tool (class) in the toolbox and then dragging it with your mouse pointer to the form

- when you drag the tool to the form, Visual Basic creates (instantiates) an instance of the class (an object) and places it on the form

- e.g.: the two picture box objects shown in **Figure 1-5** were instantiated (created) by dragging the PictureBox tool from the toolbox to the form

- similarly, the three button objects were instantiated using the Button tool

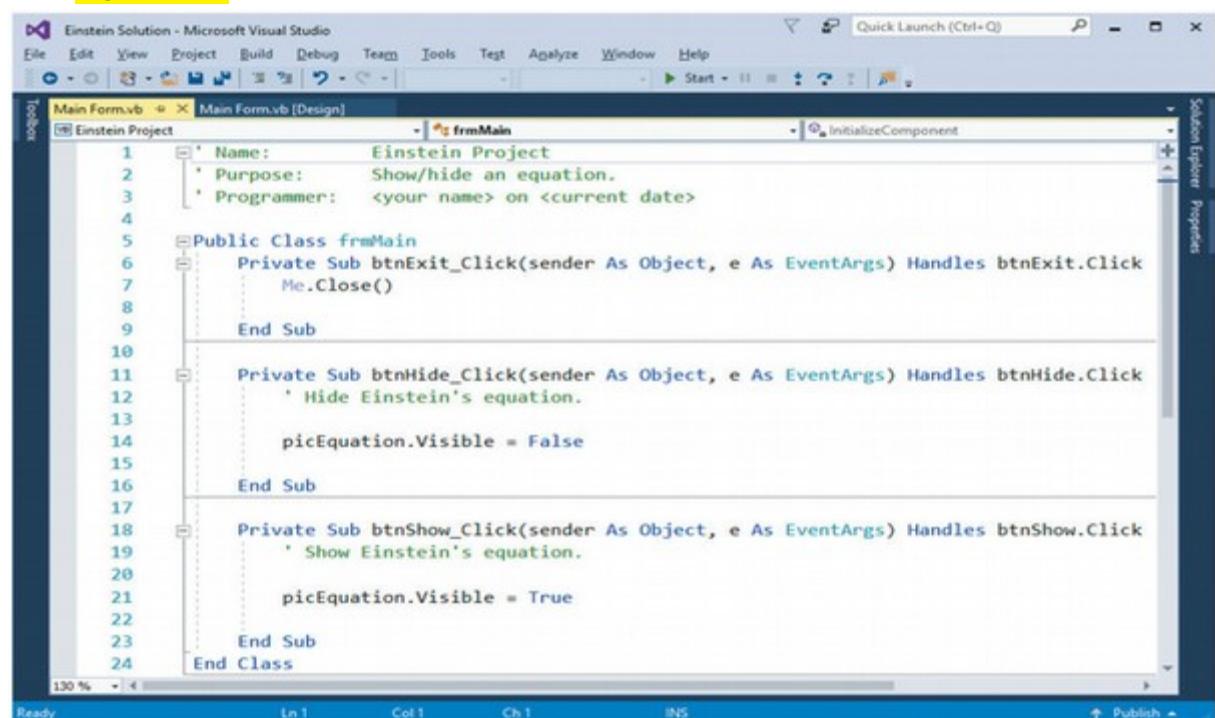
- Properties window:**
- each object has a set of attributes that determine its appearance and behavior
  - the attributes, called **properties** are listed in the **Properties window** when the object is selected in the designer window
  - you can use the Properties window to change the value of an object's property:
    - e.g.: - in **Figure 1-5**, the form is selected, and the names of its properties, along with their values, appear in the Properties window
    - for example, you can use it to change the form's Text property, which appears in the form's title bar
- ```

StartPosition      = CenterScreen
Text              = Einstein's Famous Equation
  
```

- Solution Explorer:**
- Windows applications in Visual Basic are composed of solutions, projects, and files
    - a **solution** is a container that stores the projects and files for an entire application
    - a **project** is also a container, but it stores only the files associated with that particular project
  - the Solution Explorer window displays a list of the projects contained in the **current solution** and the items contained in each project
  - e.g.: - the Solution Explorer window shown in **Figure 1-5** indicates that the Einstein Solution contains the Einstein Project, which contains several items:
    - the **Einstein.png** and **Equation.png** items are the names of files on your disk ->
      - > these files contain the images that appear in the picture boxes on the form
    - the **Main Form.vb** item is also the name of a file on your disk ->
      - > this file stores the program instruction - **code**, that tell the three buttons how to respond when the user clicks them

- you enter the **code** in the **Code Editor window**, which is shown in **Figure 1-6:**

- the green lines of the text are comments and are not executed by the computer when the application is run -> they serve simply to internally document the program
- the code on lines 6 through 9 tell the computer to close (end) the application when the Exit button is clicked
- the code on lines 11 through 16 indicates that the computer should hide the equation picture box when the Hide equation button is clicked
- the code on lines 18 through 23 tell the computer to show the equation picture box when the Show equation button is clicked



The screenshot shows the Microsoft Visual Studio Code Editor window for the Main Form.vb [Design] file. The code editor displays the following VB.NET code for the frmMain class:

```

1  ' Name: Einstein Project
2  ' Purpose: Show/hide an equation.
3  ' Programmer: <your name> on <current date>
4
5  Public Class frmMain
6      Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
7          Me.Close()
8      End Sub
9
10     Private Sub btnHide_Click(sender As Object, e As EventArgs) Handles btnHide.Click
11         ' Hide Einstein's equation.
12
13         picEquation.Visible = False
14
15     End Sub
16
17     Private Sub btnShow_Click(sender As Object, e As EventArgs) Handles btnShow.Click
18         ' Show Einstein's equation.
19
20         picEquation.Visible = True
21
22     End Sub
23
24 End Class
  
```

The code includes comments at the top and three event handlers for button click events: btnExit\_Click, btnHide\_Click, and btnShow\_Click. The picEquation variable is used to control the visibility of a picture box.

**Figure 1-6** Code Editor window

## CH1\_F5 - Assigning "(Name)" for coded or referred Objects:

- typically, you assign names to only objects that are either coded or referred to in code
- as mentioned earlier, each **object** has a set of properties attached to it - one of the most important of these properties is the Name property ->
  - > because you use the **Name** property to refer to the object in **code**
- e.g.: - the code in **Figure 1-6** refers to objects named **frmMain**, **btnExit**, **btnHide**, **btnShow**, and **picEquation**

### - some rules and conventions this book will follow when naming objects:

- rules required by Visual Basic:
  1. each object must have a unique name
  2. each unique name must begin with a letter
  3. each unique name contain only letters, numbers, and the underscore character
- naming conventions used in this book:
  1. each name will begin with an ID of three or more lower case characters that represents the object's type
  2. the remaining characters after the ID will indicate the object's purpose and will be entered using **CamelCase**
- **CamelCase** = refers to the fact that the UPPERCASE letters appear as "humps (hrby)" in the name because they are taller than the lowercase letters  
= the first letter in each subsequent word in the name is Capitalized

|         |                     |                                                               |
|---------|---------------------|---------------------------------------------------------------|
| - e.g.: | <b>frmMain</b>      | the main form in a project                                    |
|         | <b>btnExit</b>      | a button that ends the application when clicked               |
|         | <b>txtFirstName</b> | a text box for entering a customer's first name               |
|         | <b>picEquation</b>  | a picture box that displays an image of an equation           |
|         | <b>lblTotalDue</b>  | a label that displays the total amount a customer owes        |
|         | <b>chkDiscount</b>  | a check box for specifying whether a customer gets a discount |

### Mini-Quiz 1-2:

1. Which window in the IDE lists the tools you can use to add objects to a form?
2. While designing an interface, which window in the IDE allows you to change the default value of an object's property?
3. Using the naming rules and conventions, which of the following are valid names and which are not? Explain why the names are not valid.
  - a. **lblTotal**
  - b. **txtFirst.Name**
  - c. **lblCity&State**
  - d. **btnCalc Total**
  - e. **txtFirstQuarter**

3...invalid; b. - a period is not permitted, c. - an ampersand is not permitted, d. - a space is not permitted

2...always the Properties Window, what can vary depending the object

1...Tool box Window

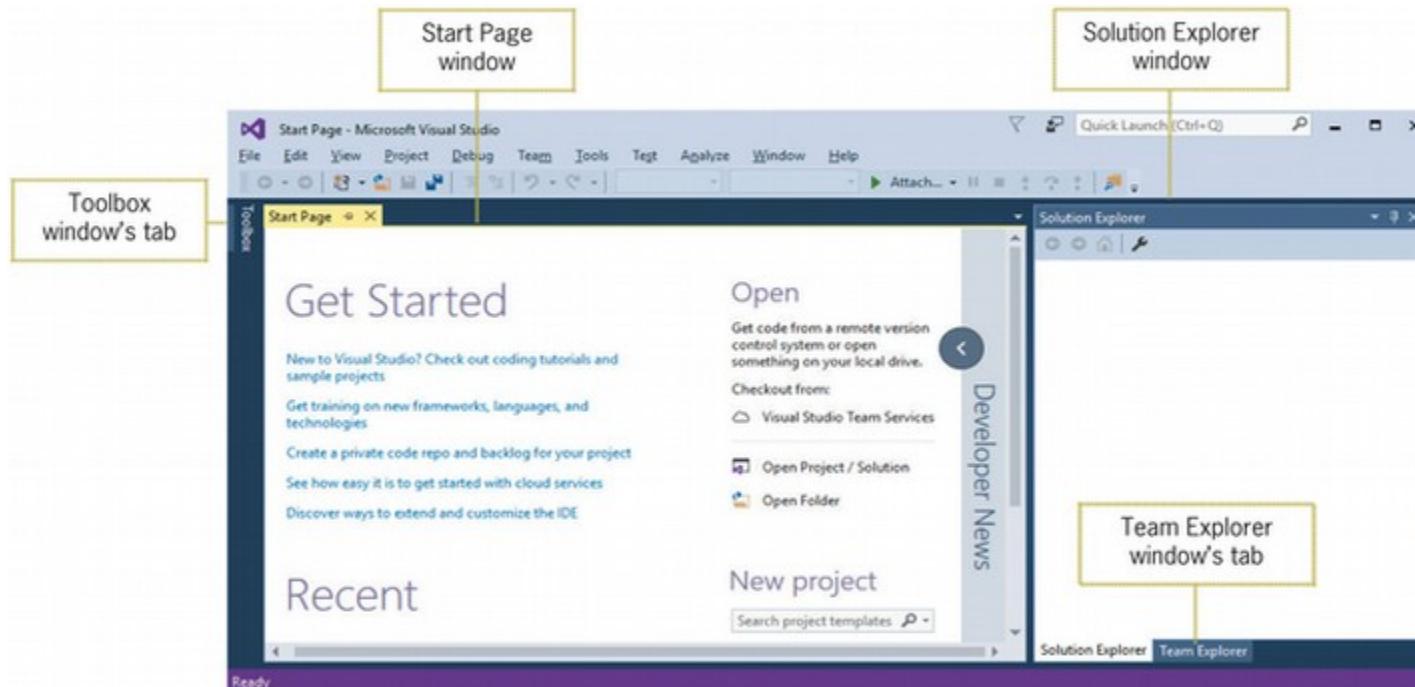
## CH1\_APPLY THE CONCEPTS LESSON

### CH1\_A1 - Start and Configure Visual Studio Community 2017:

- in this Apply lesson, you will create the Einstein's Famous Equation application that you viewed in the Focus lesson
- but first you need to start and configure Visual Studio Community 2017, which contains the Visual Basic language
- mentioned steps might differ slightly if you are using a different edition of Visual Studio 2017

1. open the Visual Studio 2017
2. on the menu bar open: **Tools / Import and Export Settings** and:
  - 2a). select the radio button: **Reset all settings** and click button **Next** and
  - 2b). click **Visual Basic**, and then click the button **Finish** and
  - 2c). click the button **Close** to close Import and Export Settings Wizard dialog box
3. on the menu bar open: **Reset Window Layout**, and then confirm the dialog box by clicking the button **Yes**

- when you start Visual Studio 2017, your screen will appear similar to **Figure 1-8**, however your menu bar might not contain the underlined letters, called access keys -> you can show/hide them by pressing the Alt key on your keyboard

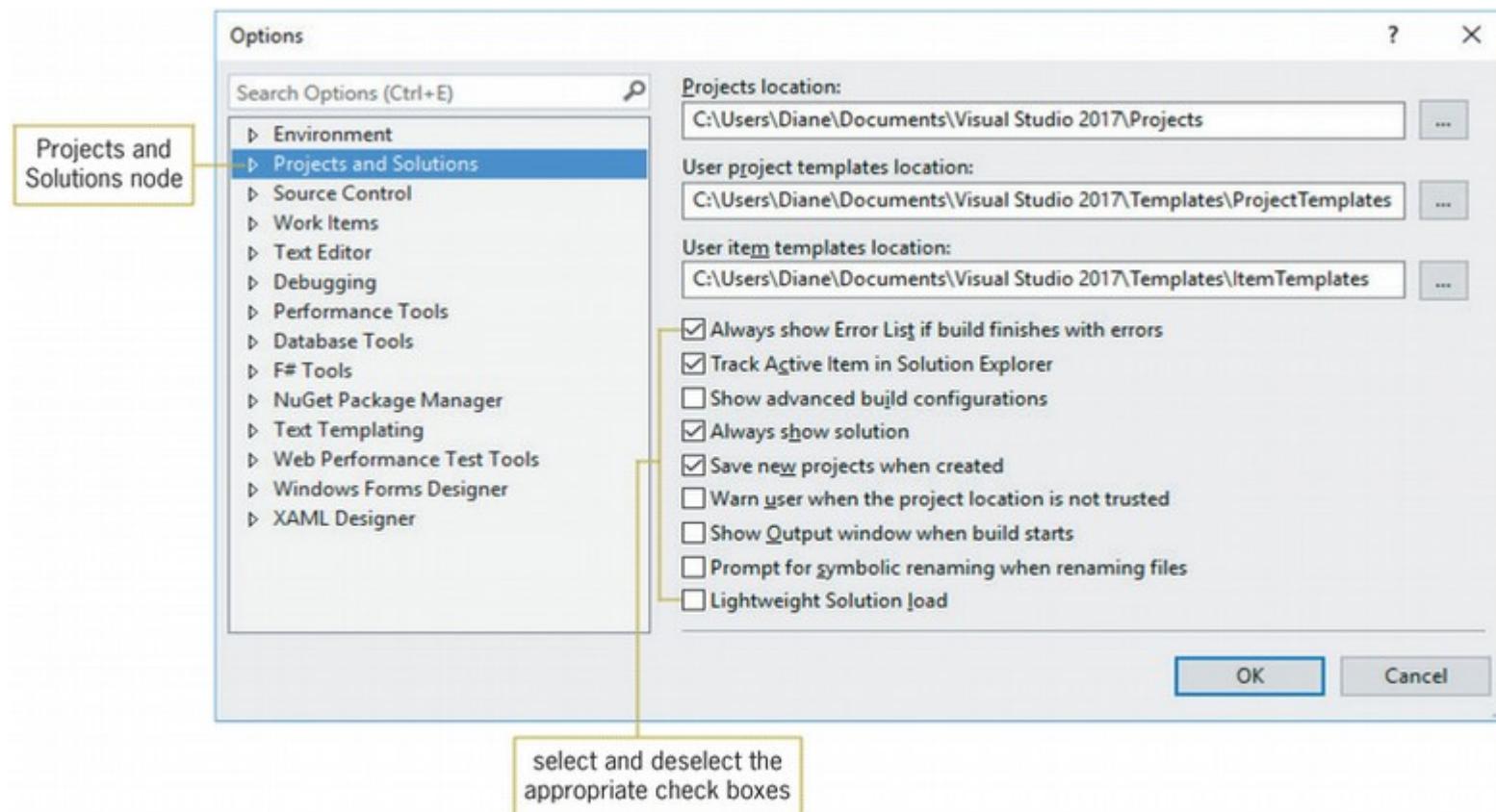


**Figure 1-8** Microsoft Visual Studio Community 2017 startup screen

- next, you will configure Visual Studio Community 2017 so that your screen and tutorial steps agree with the figures and tutorial steps in this book
- but - your screen might vary in some instances if you are using a different edition of Visual Studio or another version of Windows

4. on the menu bar open: **Tools / Options** to open the Options dialog box and:

4a). on the left side, click the node: **Projects and Solutions** and use the information shown in **Figure 1-9** to select and deselect the appropriate check boxes:



**Figure 1-9** Projects and Solutions options in the Options dialog box

4b). on the left side, click the node: **Text Editor / Basic / Code Style / Naming** and on the right side:

4b.1) in the row **Types** and column **Severity** - choose **None**

4b.2) in the row **Non-Field Members** and column **Severity** -> choose **none**

4c). on the left side, click the node: **Text Editor / Basic / Code Style** and on the right side:

4c.1) in the row **Qualify property access with 'Me'** and column **Preference** -> choose **Prefer 'Me'**

4c.2) in the row **Qualify method access with 'Me'** and column **Preference** -> choose **Prefer 'Me'**

4d). on the left side, scroll down until you locate the node **Debugging**, expand it and on the right side:

4d.1) locate **Step over properties and operators (Managed only)** -> deselect the check box if necessary

4d.2) locate **Enable Diagnostic Tools while debugging** -> deselect the check box if necessary

4d.3) locate **Show elapsed time PerfTip while debugging** -> deselect the check box if necessary

- click button **OK** to close the Options dialog box

- Note: if you change your default environment settings (steps 2 and 3) after performing the step 4, you will need to perform the step 4 again

## CH1\_A2 - To Create a Windows Forms Application: the Einstein's Equation application example

Figure 1-11

- the Einstein's Famous Equation application will be a Windows Forms application, which is an application that has a Windows user interface and runs on a computer

1. on a menu bar click **File** and then click **New Project** **Ctrl + N** to open the New Project dialog box
2. on the left side in the **Installed** Templates list, choose the node **Visual Basic**, and in the middle column choose **Windows Forms App (.NET Framework)**
3. in the grey box lower, locate **Name** and change the name entered **WindowsApp1** to **Einstein Project**
4. in the grey box lower, locate **Location** and click the **Browse...** button to open the Project Location dialog box and choose your desired location  
-> in this case: **VB2017\Chap01** folder, and click the **Select Folder** button to close the Project Location dialog box
5. in the grey box lower, locate **Solution name** and change the name entered **WindowsApp1** to **Einstein Solution**
  - select the check box: **Create directory for solution**
6. click the **OK** button to close the New Project dialog box

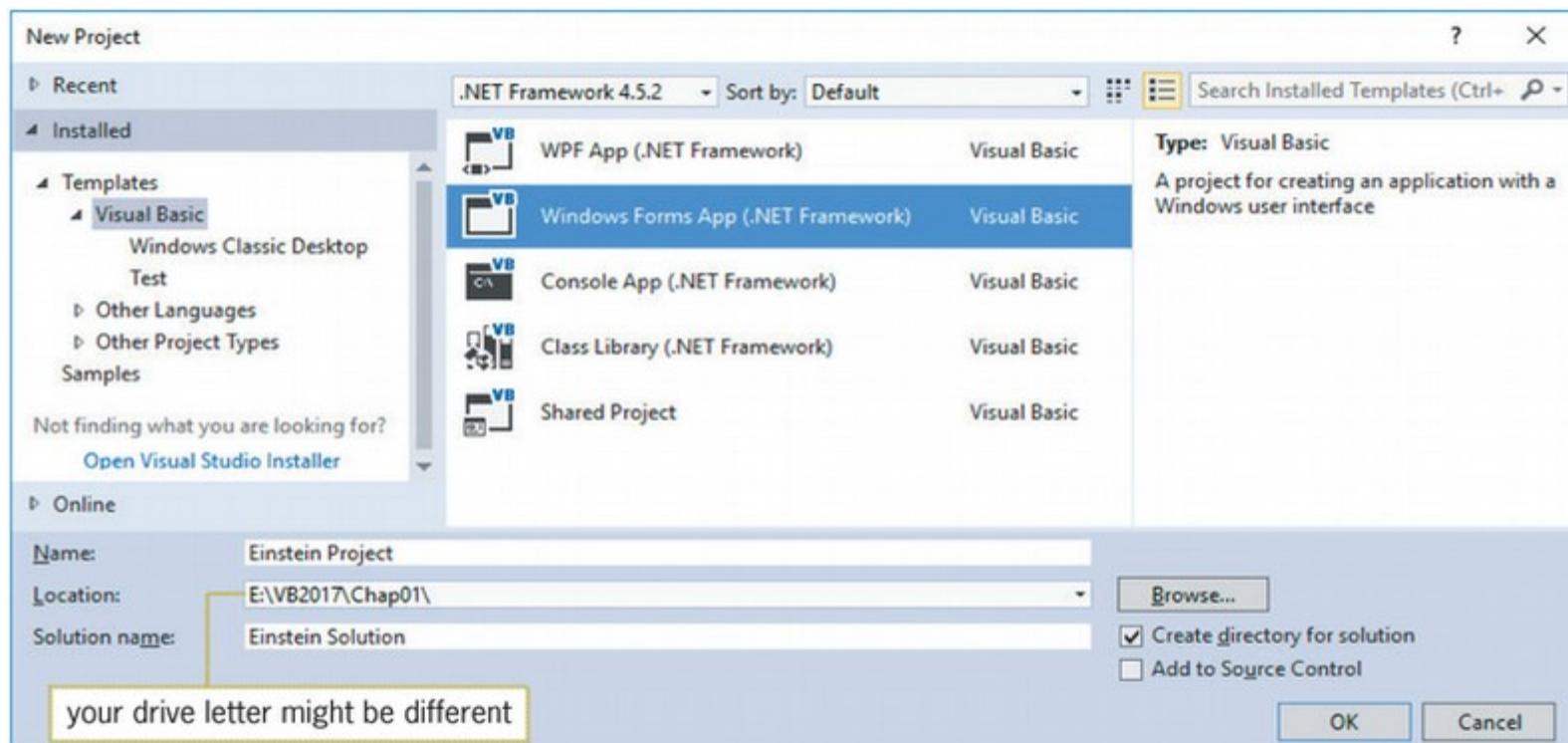
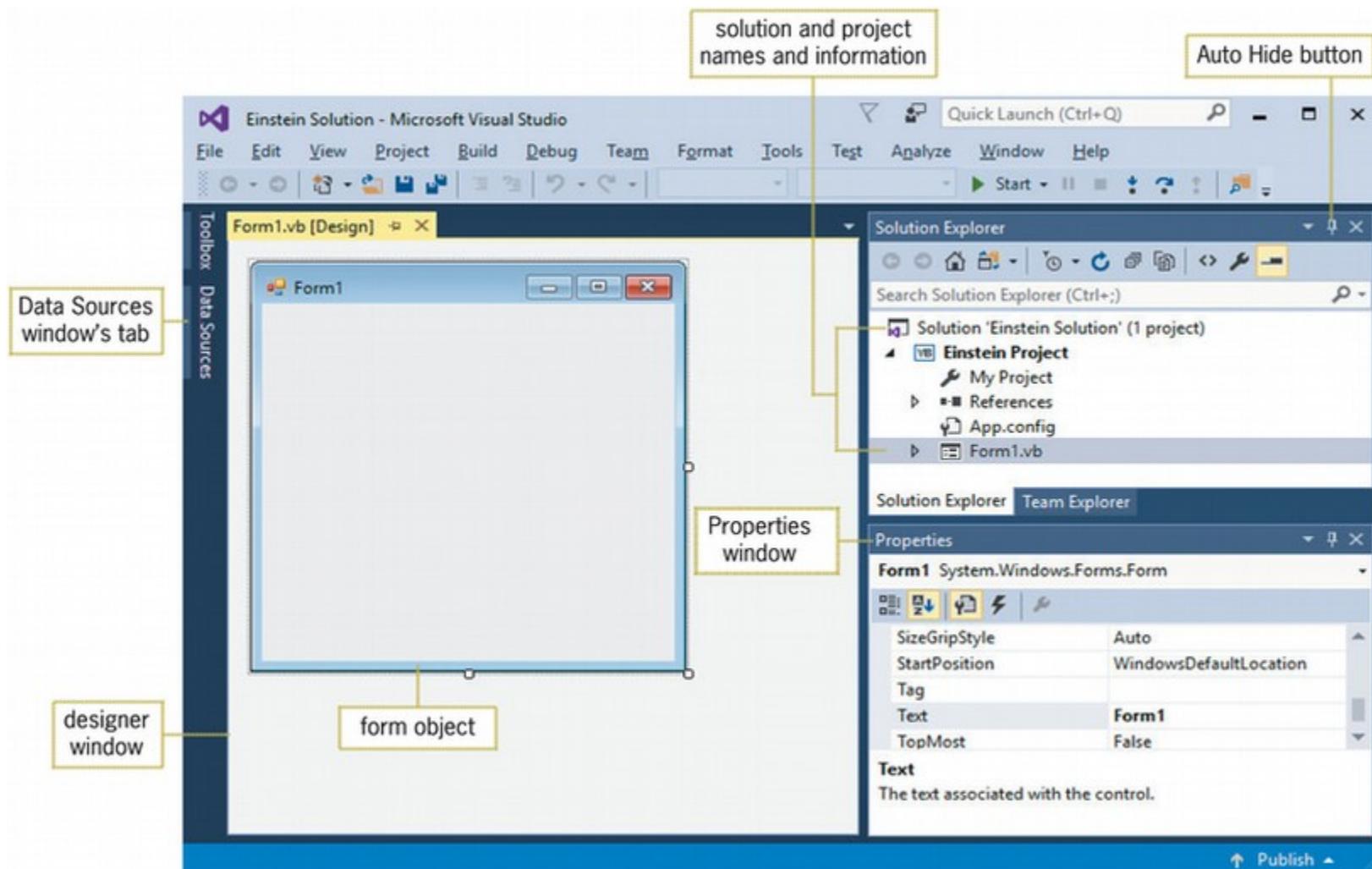


Figure 1-11 Completed New Project dialog box

- notes:  
- your dialog box might look slightly different if you are using another edition of Visual Studio  
- do not be concerned if your dialog box shows a different version of the .NET Framework

- the computer creates a solution and adds a Visual Basic project to the solution
- the names of the solution and project, along with other information pertaining (náleží) to the project, appear in the Solution Explorer window
- Visual Basic also automatically instantiates - creates a form object, which appears in the designer window: **Figure 1-12**



**Figure 1-12** Solution and Visual Basic project

**CH1\_A3 - Manage the Windows in the IDE:**

- in most cases, you will find it easier to work in the IDE if you either close or auto-hide the windows you are not currently using
- in the **View** menu you will find the options for opening a closed window
- title bar buttons:
  1. arrow down
  2. pushpin - auto-hide window -> it is a toggle button: clicking it once activates it and clicking it again deactivates it
  3. x - close an open window
- the **Toolbox** window and **Data Sources** window are auto-hidden

### CH1\_A3.1 - to close, open, auto-hide, and display windows in the IDE : the Einstein's Equation application example

Figure 1-12

- note: to reset the window layout in the IDE, click **menu bar / Window / Reset Window Layout** and confirm by **Yes**

1. to close: - locate the **Properties** window and on its title bar click the **x** to close the window  
- click the **menu bar / View / Properties Window F4** to open the window again -> it will spawn in its previous position
2. to close: - click the tab **Team Explorer** and when the window appears, on its title bar click the **x** button to close the tab
3. to auto-hide: - locate the **Solution Explorer** window and on its title bar click the **vertical pushpin** to Auto-hide  
- the window now appears as a tab on the edge of the IDE (the pushpin changes to horizontal)
4. to temporarily display auto-hidden: - click the tab **Solution Explorer** and the window appears  
- notice the **pushpin** is now a **horizontal**  
- to return the window to its auto-hidden state, click the tab again
5. to permanently display auto-hidden: - click the tab **Solution Explorer** and the window appears  
- on its title bar click the **horizontal pushpin** button and the window will always display
6. to show object properties: - in the **Solution Explorer** window click **Form1.vb**
  - the name of the selected object (Form1.vb) appears in the **Properties** window's Object box
  - the **Properties** window also contains 2 columns of information:
    - the **left column:** - called the **Properties list**  
- displays the names of the selected object's properties
    - the **right column:** - called the **Setting boxes**  
- each box displays the current value or settings of its associated property
  - if the names listed in the **Properties list** do not appear in alphabetical order, click the button **Alphabetical**, which is the second button on the **Properties** window's toolbar

Figure 1-13:

- shows the current status of the windows in the IDE:
- open: designer, Solution Explorer, and Properties
- auto-hidden: Toolbox window

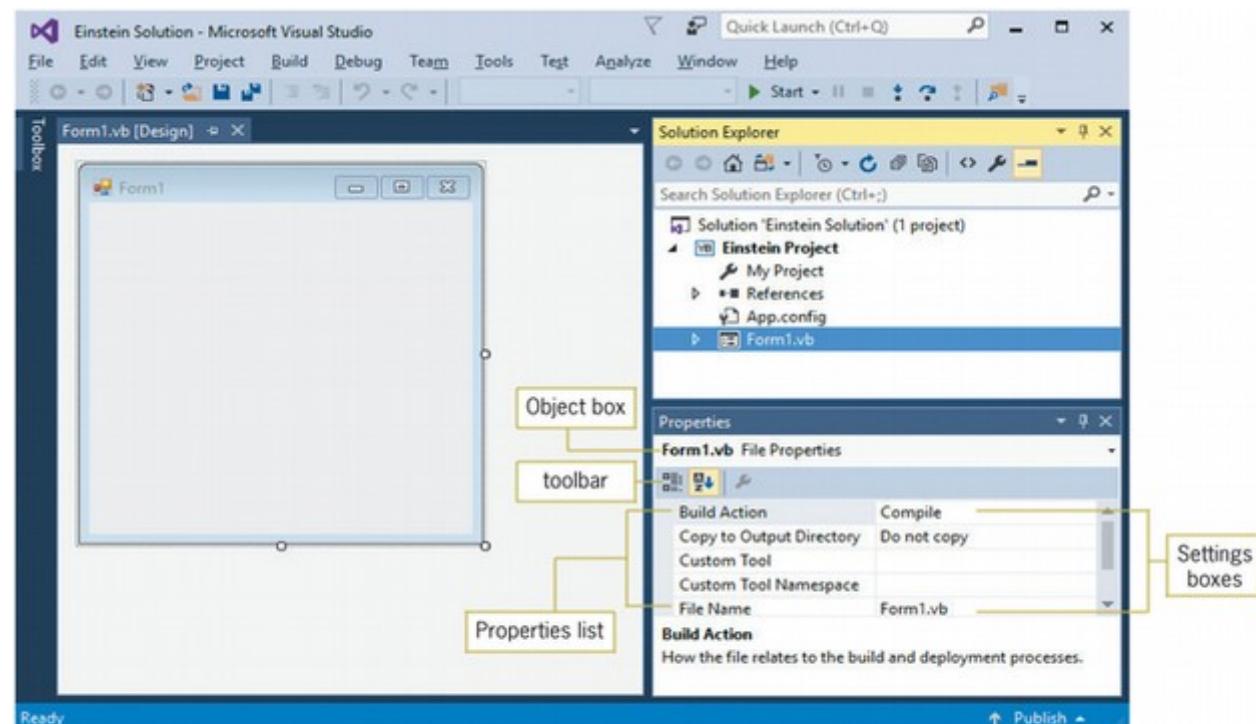


Figure 1-13 Current status of the windows in the IDE

## CH1\_A4 - Change a Form File's Name:

- the code associated with the 1st form included in a project is automatically stored in a file, referred to as a form file, named **Form1.vb** ->
  - > it's called a form file because it contains the code associated with the form
  - all files with a **.vb** filename extension are also referred to as source files because they contain Visual Basic code
  - the code associated with the 2nd form in the same project is stored in a file named **Form2.vb**, and so on
  - to help you keep track of the various form files in a project, **you should give each a unique and meaningful name**
- 2 ways to change a form file's name:
- 1). in the **Solution Explorer**, right-click on the **Form1.vb**, then on the context menu click **Rename**, and type **Main Form.vb**
  - 2). in the **Solution Explorer**, choose the **Form1.vb**, and on its **Properties list** locate row named **(Name)** and on its **Settings box** type **Main Form.vb** and Enter
    - be sure to include the **.vb** extension on the filename - otherwise the computer will not recognize the file as a source file
- after either way chosen, the **Main Form.vb** appears 3-times: in the **Solution Explorer** window, **Properties** window, and on the **designer** window's tab

### Mini-Quiz 1-3:

1. Which menu provides options for opening a closed window in the IDE?
2. What is the name of the vertical pushpin button on a window's title bar?
3. What filename extension indicates that the file is a Visual Basic source file?

3... vb  
2... auto-hide  
1... View

## CH1\_A5 - Change the Properties of a Form:

- in order to display the properties of a form in the **Properties** window, the form must be selected in the **designer** window --->
- > it is easy to confuse a **form file**, whose **File Name** property you changed in the previous section, with a form
  - > the **form file** is an actual file that resides on your disk and contains code shown earlier in **Figure 1-6**
  - > the form is the object that appears in the designer window

### CH1\_A5.1 - to display the **form's** properties and the most commonly used:

- in the **designer** window click the form and on the **Properties list** scroll to the top and click **(Name)** -> see

Figure 1-15

Figure 1-15

- the Object box shows the text: **Form1 System.Windows.Forms.Form** , where:

- Form1** - is the name of the form that appears in the **designer** window
  - the name is automatically assigned to the form when the form is instantiated - created

**System.Windows.Forms.Form** - is the name of the class used to instantiate the form

- System.Windows.Forms** - is the namespace that contains the Form **class definition**
  - a class definition = block of code that specifies or defines an object's appearance and behavior
    - all class definitions in Visual Basic are contained in namespaces
  - a namespace = picture it as blocks of memory cells inside the computer
    - each namespace contains the code that defines a group of related classes

- System.Windows.Forms** namespace - contains the definition of the Windows Form class
  - also contains the class definitions for objects you add to a form, such as buttons and text boxes

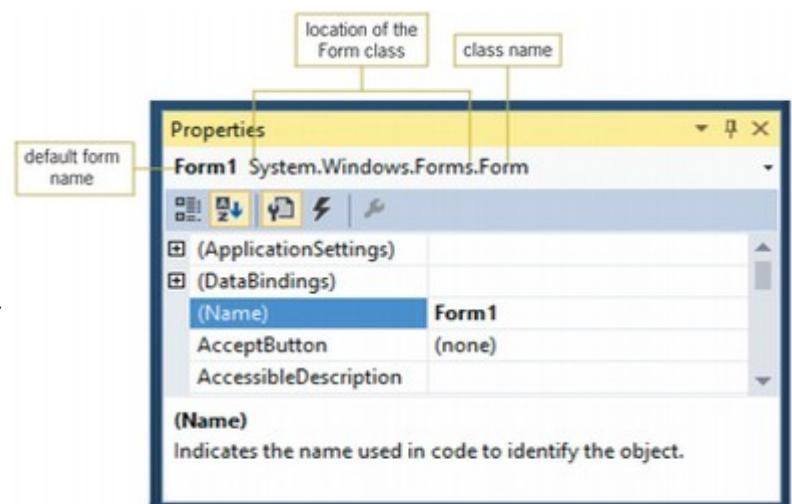


Figure 1-15 Partial list of the form's properties

the period . that separates each word = called the **dot member access operator**

- similar to the backslash \ in a folder path -> indicates a hierarchy, but of namespaces rather than folders

e.g.: - the **backslash** in the path: **E:\VB2017\Chap01\Einstein Solution\Einstein Project\Main Form.vb** indicates that:

- 1). the **Main Form.vb** file is contained in - or is a member of - the **Einstein Project** folder,
- 2). which is a member of the **Einstein Solution** folder,
- 3). which is a member of the **Chap01** folder,
- 4). which is a member of the **VB2017** folder,
- 5). which is a member of the **E:** drive

- likewise, the name **System.Windows.Forms.Form** indicates that:

- 1). the **Form** class is a member of the **Forms** namespace,
- 2). which is a member of the **Windows** namespace,
- 3). which is a member of the **System** namespace

- the **dot member access operator** allows the computer to locate the **Form** class in the computer's main memory,

similar to the way the **backslash \** allows the computer to locate the **Main Form.vb** file on your computer's disk

#### - the most commonly used properties of a form:

**(Name)** - following the naming rules and conventions shown earlier, you will name the current form **frmMain**, where:  
- **frm** = identifies the object as a form, and  
- **Main** = reminds you of the form's purpose, which is to be the main form in the application

**AcceptButton** - specify a default button that will be selected when the user presses the **Enter** key  
- i mostly use for buttons like Calculate, Display, Show etc.

**BackColor** - specify the background color of the form  
- default is: Control

**CancelButton** - specify a cancel button that will be selected when the user presses the **Esc** key  
- i mostly use for the buttons like Cancel, Exit

**ControlBox** - indicate whether the form contains the **Control box** and **Minimize**, **Maximaze**, and **Close** buttons

**Font** - specify the font to use for all texts  
- determines the type, style, and size of the font used to display the text on the form  
- a font is the general shape of the characters in the text  
- font sizes are typically measured in points, with one point **pt** equaling 1/72 of an inch (2.54 cm)  
- the recommended font for applications created for systems running Windows 10 is **Segoe UI** because it offers improved readability  
- Segoe is pronounced "SEE-go", and UI stands for user interface  
- for most of the elements in the interface, you will use a **9pt** font size, however, to make the figures in the book more readable, some of the interfaces created in this book will use a slightly larger font size  
**- you have to buy a licence to use a font from Microsoft, or other creators !!!**  
**- use only free fonts, but make sure they are really free, not any restriction included !!!**  
**- free typefaces : Arvo (s padkama), Source Sans Pro (bezpadkovy), Audrey (elegant), ....**

|                        |                                                                                                                                                                                                                                                                                                          |
|------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>FormBorderStyle</b> | - specify the appearance and behavior of the form's border - the default is Sizable<br>- can choose from: None, FixedSingle, Fixed3D, FixedDialog, Sizable, FixedToolWindow, SizableToolWindow                                                                                                           |
| <b>MaximizeBox</b>     | - specify the state of the Maximize button - can choose from: True, False<br>- you can use a form's MaximizeBox property to disable the Maximize button, which appears in the form's title bar -><br>-> doing this prevents the user from maximizing the user interface while the application is running |
| <b>MinimizeBox</b>     | - specify the state of the Minimize button - can choose from: True, False                                                                                                                                                                                                                                |
| <b>StartPosition</b>   | - indicate the starting position of the form - the default is: WindowsDefaultLocation<br>- can choose from: Manual, CenterScreen, WindowsDefaultLocation, WindowsDefaultBounds, CenterParent                                                                                                             |
| <b>Text</b>            | - specify the text that appears in the form's title bar and on the taskbar<br>- Form1 is the default value assigned to the Text property of the 1st form in a project<br>- you better choose some more descriptive value, e.g.: "Mbank - the Mortgage calculator", etc.                                  |

- e.g.: the Einstein's Equation application example: Change the Properties

1. in the **Properties** list, select the form's (**Name**) property, type **frmMain** and press **Enter**
2. in the **Properties** list, click **Font** and then in the **Settings** box click the ... button to open the Font dialog box ->  
-> in the Font box locate and then click **Segoe UI**, in the Size box click **10**, and then click button **OK**  
- note: do not be concerned if the size of the form changes, and if the Font property shows 9.75pt rather than 10pt
3. in the **Properties** list, click **MaximizeBox**, and in the **Settings** box click the list arrow and choose **False**
4. in the **Properties** list, click **StartPosition**, and in the **Settings** box click the list arrow and choose **CenterScreen**
5. in the **Properties** list, click **Text**, type **Einstein's Famous Equation** and then press **Enter**  
- notice: rather than **Form1**, the **Einstein's Famous Equation** appears in the form's title bar  
- note: the **asterisk \*** on the designer window's tab indicates that the form has been changed since the last time it was saved

**Mini-Quiz 1-4:**

1. What character is the dot member access operator?
2. What is the recommended font type for Windows 10 applications?
3. Which of a form's properties determines the location of the form when the application is started?
4. To display the words "ABC Company" in a form's title bar, you need to set which of the form's properties?

1...a period .

2...Segoe UI, but it's not freeli

3...StartPosition

4...Text

**CH1\_A6 - Save a Solution:**

- as mentioned earlier, an **asterisk \*** on the designer window's tab indicates that a change was made to the form since the last time it was saved
- it is a good idea to save the current solution every 10 or 15 minutes so that you will not lose a lot of your work if a problem occurs with your computer
- you can save the solution by clicking on the **menu** bar on **File** and then clicking: **Save All Ctrl+Shift+S**
- you can also click on the **Standard toolbar**, on the button **Save All** -> looks like 2 diskets
- when you save the solution, the computer saves any changes made to the files included in the solution, and it also removes the asterisk \*

## CH1\_A7 - Close and Open a Solution:

- before learning how to add a control to the form, you will close the current Einstein Solution and open a partially completed one
- you should always close a solution when you are finished working on it -> doing so ensures that all of the projects and files contained in the solution are closed

- e.g.: the Einstein's Equation application example: To close the current solution and open a partially completed one

1. on the **menu** bar, click **File** -> notice that the menu contains a **Close** option and a **Close Solution** option
  - the **Close** option closes only the designer window in the IDE, however, it does not close the solution itself
  - only the **Close Solution** option closes the solution
2. click **Close Solution** -> the Solution Explorer window indicates that no solution is currently open in the IDE
3. on the **menu** bar, click **File** and then click **Open Project Ctrl+O** to open the Open Project dialog box
  - locate and then open the folder: **VB2017\Chap01\Partial Einstein Solution**
4. the names of solution files end with **.sln**
5. in the list of filenames click **Einstein Solution.sln** and then click the **Open** button
6. depending on how Windows is set up on your computer, you might not see the **.sln** extension on the filename
7. the **Solution Explorer** window indicates that the solution is open ->
  - > if the **designer** window is not open, in the **Solution Explorer** window right-click **Main Form.vb** and then click **View Designer**
8. in the **Solution Explorer** window expand the **Resources** node -> the interface contains a picture box and 2 buttons: **Figure 1-17**

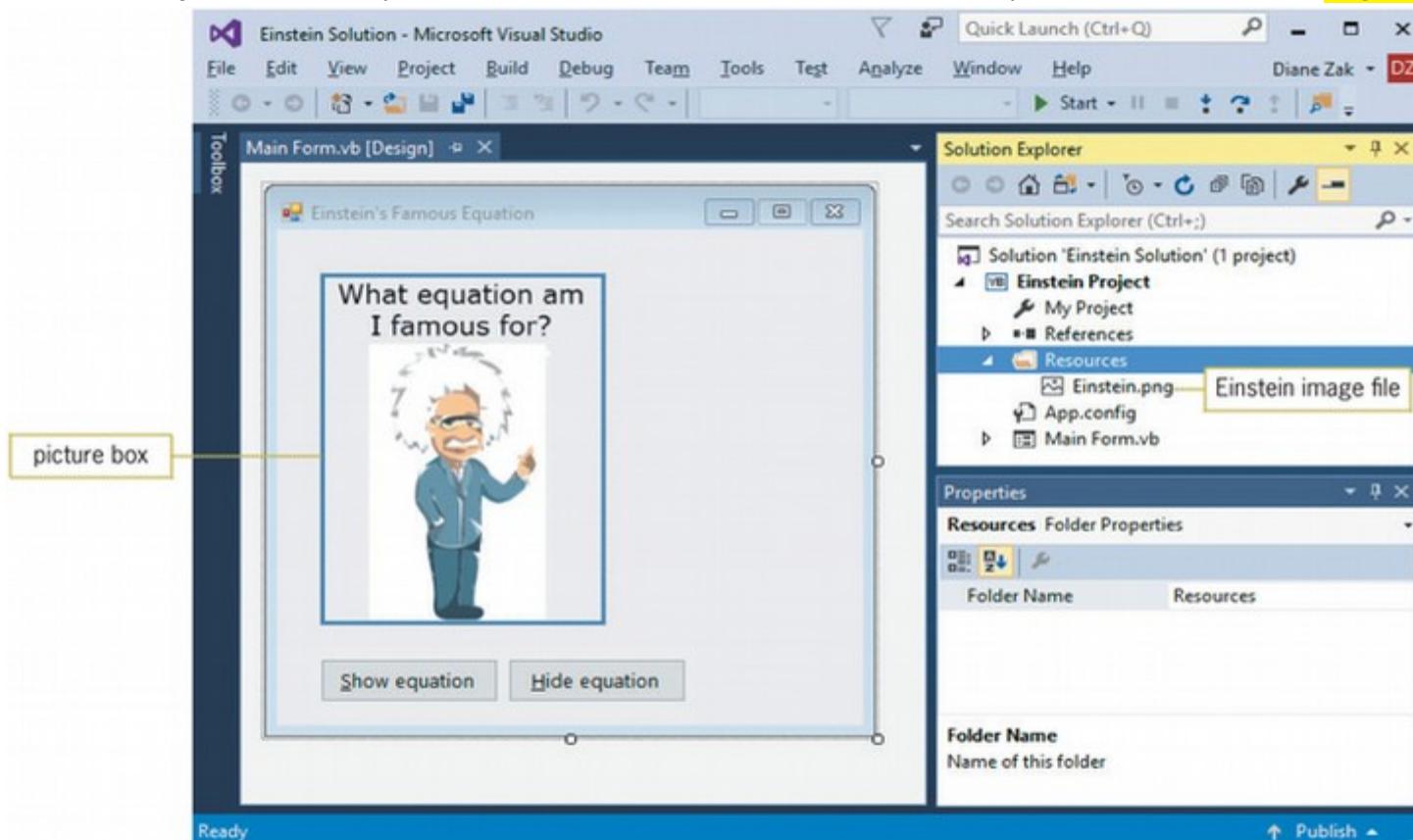


Figure 1-17 Partially completed solution

## CH1\_A8 - Add a Control to a Form: the Einstein's Equation application example

- 2 controls are missing from the interface shown in **Figure 1-17**:
  - 1). a **picture box** that displays Einstein's famous equation E=MC<sup>2</sup>
  - 2). and an **Exit** button

**CH1\_A8.1...**

### CH1\_A8.1 - To add a **picture box** to the Form: the Einstein's Equation application example

- the most commonly used properties of a **picture box**:

|          |                                                                                                                                                                  |
|----------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| (Name)   | - picMeaningfulName                                                                                                                                              |
| Image    | - specify the image to display                                                                                                                                   |
| SizeMode | - specify how the image should be displayed <ul style="list-style-type: none"><li>- can choose from: Normal, StretchImage, AutoSize, CentreImage, Zoom</li></ul> |
| Visible  | - hide / display the picture box <ul style="list-style-type: none"><li>- can choose from: True, False</li></ul>                                                  |

- 1). very left, on the **Toolbox** tab, expand the node **Common Controls**, and locate **PictureBox** tool ->

- > click on it, but hold down the mouse button as you drag the tool to the empty space on the form, and release the mouse button
- the PictureBox tool (class) instantiates a picture box control (object) and places it on the form

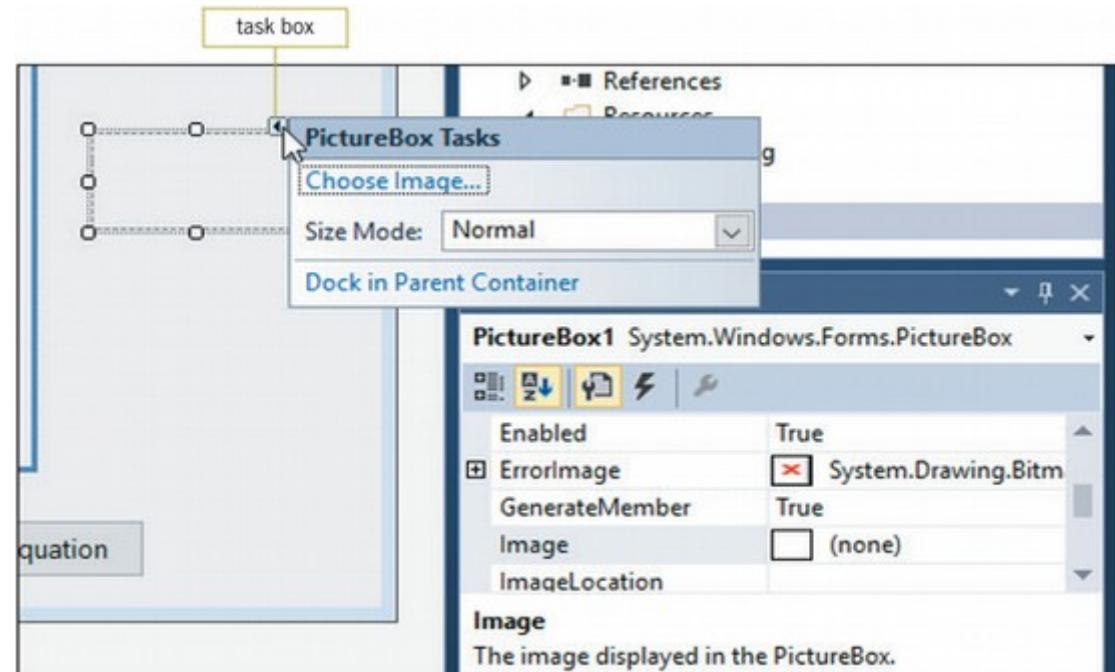
- the picture box's properties appear in the **Properties** list, and a box containing a triangle appears in the upper-right corner of the control:

- the box is referred to as the **task box** because when you click it, it displays a list of the tasks associated with the control
- each task in the list is associated with one or more properties ->
  - > you can set the properties using the:
    - a). **task list (triangle)**, or
    - b). **Properties** window

- 2). click the **task box** on the control **PictureBox1**: **Figure 1-20**

- 3). click **Choose Image** to open the **Select Resource** dialog box

note: the task **Choose Image** is associated with the **Image** property in the **Properties** window



**Figure 1-20** Open task list for a picture box

**Figure 1-21**

- to include the image file within the project itself, the radio button **Project resource file** must be selected in the dialog box:

- 4). verify that the radio button is selected, and then click the **Import** button to open the **Open dialog box**

- 5). open the folder **VB2017\Chap01** and click **Equation.png** in the list of filenames, click the **Open** button, and then the **OK** button

- see what happened:

- a small portion of the image appears in the picture box control on the form
- **Einstein\_Project.My.Resources.Resources.Equation** appears in the control's **Image** property in the **Properties** window -> it indicates, that the **Equation.png** file is stored in the project's **Resources** folder
- in addition, **Equation.png** appears in the **Solution Explorer** window, in the **Resources** folder

- 6). fix the small portion of the image appeared in the picture box:

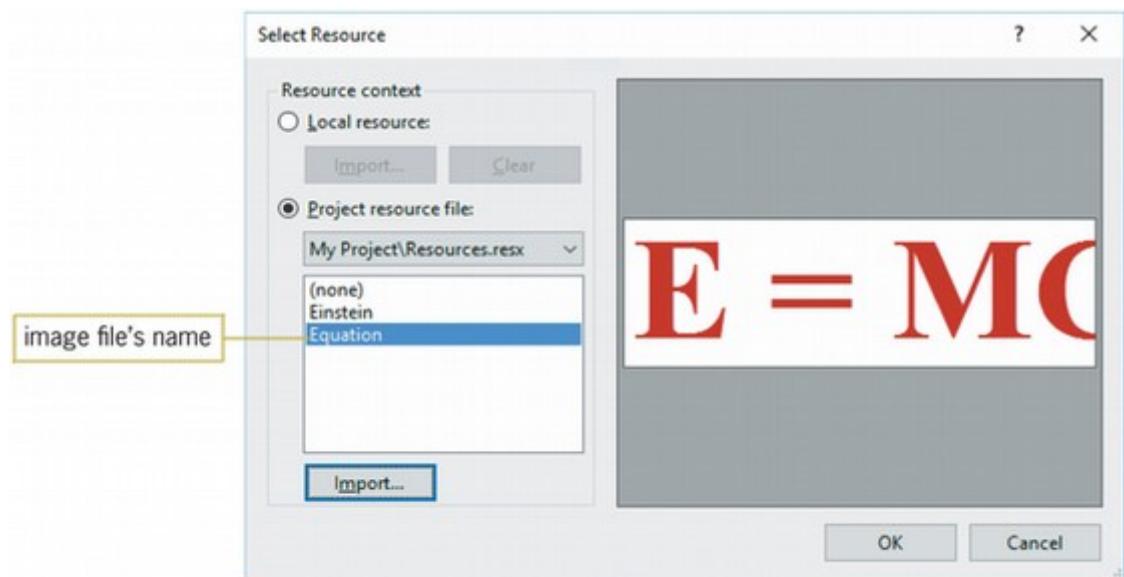
- a). on the control itself, click the **task box** to open the task list, and in the **Size Mode** box, click the list arrow and then choose the **StretchImage**
- b). or in the **Properties** window locate the **SizeMode** and choose **StretchImage** (the default is **Normal**)

- 7). the picture box will be referred to in code, so you will give it a more meaningful name - the three-character ID for a picture box is **pic**  
-> so in the **Properties**, change the **(Name)** property to **picEquation**

- 8). make the **picEquation** control slightly wider by placing your mouse pointer on the control's middle-right sizing handle and then dragging to the right ->  
-> stop dragging when a thin blue line appears between the control's border and the form's border: **Figure 1-22** and release the mouse button  
- the designer provides a blue margin line to assist you in spacing the controls properly

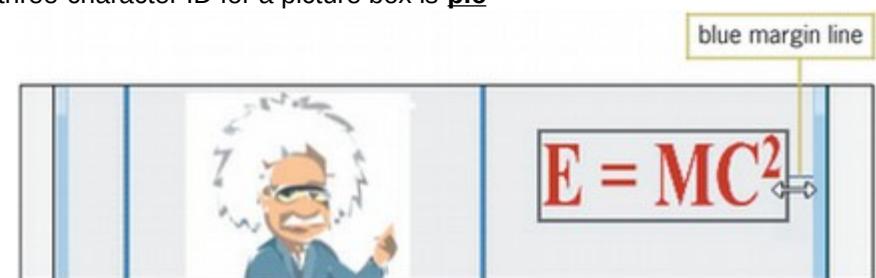
- 9). the **picEquation** control should not be visible when the interface appears on the screen after the application is started ->  
-> in the **Properties**, set the control's **Visible** property to **False**

- 10). that's all for now



**Figure 1-21** Completed Select Resource dialog box

**Figure 1-20**



**Figure 1-22** Result of dragging the sizing handle

## CH1\_A8.2 - To add a **button** to the Form: the Einstein's Equation application example

- now you will add the missing **Exit** button to the form
- you use a button to perform an immediate action when clicked - in this case, the button will exit (end) the application

### - the most commonly used properties of a **button**:

|                   |                                                                                                                                                  |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>(Name)</b>     | - <b>btnMeaningfulName</b>                                                                                                                       |
| <b>BackColor</b>  | - specify the background color<br>- default is Control -> the actual color depends on your Windows system settings                               |
| <b>Enabled</b>    | - indicate whether the button can respond to the user's action - you can choose from: True, False                                                |
| <b>Font</b>       | - specify the font to use for text                                                                                                               |
| <b>ForeColor</b>  | - specify the font color                                                                                                                         |
| <b>Image</b>      | - default is ControlText -> the actual color depends on your Windows system settings                                                             |
| <b>ImageAlign</b> | - specify the image to display on the button's face                                                                                              |
| <b>TabStop</b>    | - indicate the alignment of the image on the button's face                                                                                       |
| <b>Text</b>       | - allow the user to access the button via Tab - default is True                                                                                  |
| <b>TextAlign</b>  | - specify the text that appears on the button's face<br>- should include an access key (Alt key)<br>- sets a text alignment on the button's face |

- 1). very left, on the **Toolbox** tab, expand the node **Common Controls**, and locate **Button** tool ->  
-> so, drag the tool and position it to the right of the button "Hide equation", using the blue margin and snap lines, and release the mouse button **Figure 1-24**

- 2). the Button tool (class) instantiates a button control (object) and places it on the form  
- the button will be coded, so you will give it a more meaningful name ->  
-> so, change the button's **(Name)** to **btnExit**

- 3). a button's **Text** property determines the text that appears on the button's face  
- it should include an access key, but you will learn about it in: **CH2\_F3 - Access keys - Alt keys: E&xit = Exit**  
- for now, you just need to know that a button's access key allows the user to select the button by pressing the **Alt** key in combination with a character that appears in the **Text** property  
- you designate the character by preceding it with an **ampersand &**  
- e.g.: to designate the letter **x** as the access key for an **Exit** button, you enter **E&xit** in the button's **Text** property  
- the access key will appear underlined on the button's face: **Exit**  
-> so, set the button's **Text** property to **E&xit**

- 4). now drag the **Exit** button's bottom sizing handle down slightly until the underline below the letter **x** is visible, and then release the mouse button

- 5). Save All

### Mini-Quiz 1-4:

1. What is the three-character ID used when naming picture boxes?
2. What is the three-character ID used when naming buttons?
3. What is the purpose of an access key?
4. What character is used to designate an access key?

1...pic  
2...btn  
3...to access the control using Alt + designated key  
4...ampersand &

## CH1\_A9 - Use the Format Menu:

- Visual Basic's **Format** menu provides many options that you can use when designing your user interface
- before you can use the **Format** menu to change the alignment or size of two or more controls, you first must select the controls
- you should always select the reference control first
- the reference control:
  - is the one whose **size** and/or **location** you want to match
  - will have **white sizing handles**, whereas the other selected controls will have **black sizing handles**
  - choose by clicking on it (white squares around appears), and the "slave" controls choose by holding **Ctrl** + clicking on it (black squares)

### - all **Format** menu options used when designing GUI:

|                           |                                                                                                                         |
|---------------------------|-------------------------------------------------------------------------------------------------------------------------|
| <b>Align</b>              | - align two or more controls by their borders                                                                           |
| <b>Make Same Size</b>     | - can choose from: Lefts, Centres, Rights, Tops, Middles, Bottoms, to Grid<br>- make two or more controls the same size |
| <b>Horizontal Spacing</b> | - can choose from: Width, Height, Both, Size to Grid<br>- adjust the horizontal spacing between two or more controls    |
| <b>Vertical Spacing</b>   | - can choose from: Make Equal, Increase, Decrease, Remove<br>- adjust the vertical spacing between two or more controls |
| <b>Center in Form</b>     | - can choose from: Make Equal, Increase, Decrease, Remove<br>- center one or more controls on the form                  |
| <b>Order</b>              | - can choose from: Horizontally, Vertically<br>- specify the layering of one or more controls on the form               |
| <b>Lock Controls</b>      | - can choose from: Bring to Front, Send to Back<br>- lock the controls in place on the form                             |

## CH1\_A9.1 - To adjust the **Exit** button's height and top border: the Einstein's Equation application example

- 1). click the reference control button **Hide equation** and then hold **Ctrl** key while click the **Exit** button  
notice the white squares around the reference control, and black squares around the "slave" control
- 2). on the menu bar, click **Format**, choose **Make Same Size**, and then click **Height**
- 3). on the menu bar, click **Format**, choose **Align**, and then click **Tops**
- 4). click the form's title bar to deselect the selected controls

## CH1\_A10 - Lock the Controls on the Form:

- after placing all of the controls in their appropriate locations, you should lock them, which prevents them from being moved inadvertently as you work in the IDE
- you can lock the controls by:
  - a). clicking the form (or any control on the form) and then on the **Format** menu by clicking the option **Lock Controls**  
note: you can follow the same procedure to unlock the controls
  - b). you can also lock and unlock the controls by right-clicking the form (or any control on the form) and then on the context menu by clicking **Lock Controls**
  - c). on the control **Properties** by setting **Locked** either True or False
- when a control is locked, a small lock appears in the upper-left corner of the control and you are not able to drag the control to a different location

## CH1\_A11 - Before you Start and End an Application - Verify the Startup Form:

- before you start an application for the first time, you should open the **Project Designer** window and verify the name of the startup form, which is the form that the computer automatically displays each time the application is started
- you can open the **Project Designer** window:
  - a). in the **Solution Explorer** window, right-click on **My Project**, and on the context menu click **Open**
  - b). on the menu bar, click **Project**, and choose the option on very bottom: "project name" **Project Properties**

### CH1\_A11.1 - To verify the startup form: the Einstein's Equation application example

- 1). choose one of the methods above, either a). or b)., to open the **Project Designer** window
- 2). on very left, click the tab **Application**, to display the Application pane Figure 1-27  
-> if **frmMain** does not appear in the **Startup form** list box, click the list arrow and then choose **frmMain** in the list  
**note:** do not be concerned if your **Targeted framework** list box shows a different value
- 3). save the solution and close the **Project Designer** window by clicking it's Close button

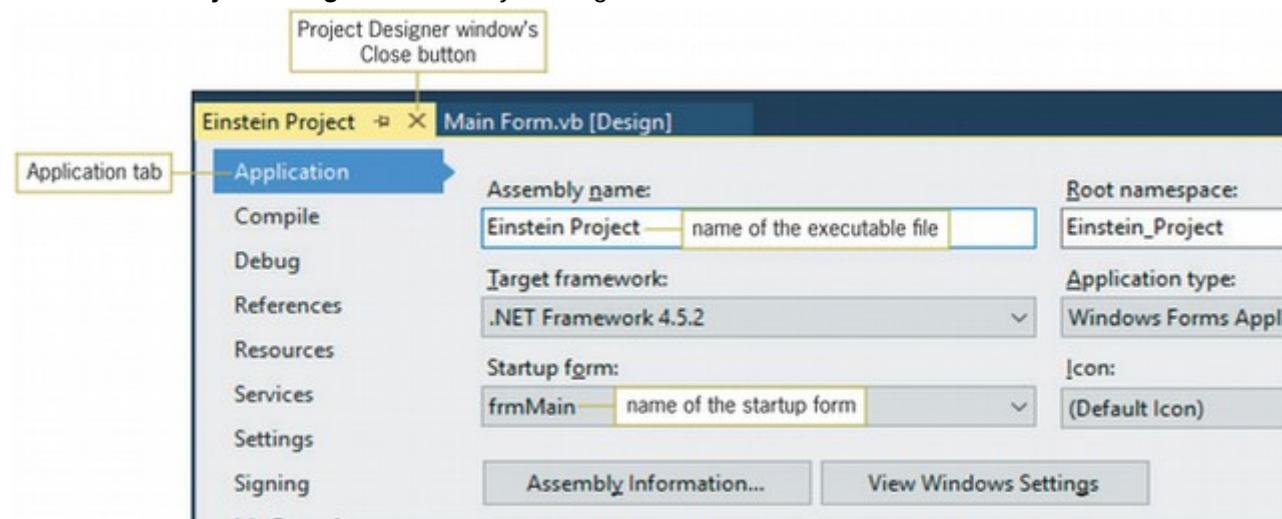


Figure 1-27 Application pane in the Project Designer window

## CH1\_A11.2 - Start and End an Application:

- you can start an application:
    - a). on the menu bar, by clicking **Debug**, and then clicking **Start Debugging F5**
    - b). by pressing the **F5** key on your keyboard
    - c). on the **Standard toolbar**, by clicking the **Start** button (right green arrow)
  - you can stop an application:
    - a). on the menu bar, by clicking **Debug**, and then clicking **Stop Debugging Ctrl + Alt + Break**  
**note:** compact and notebook keyboards often do not have a dedicated **Break** key
    - b). on the form's title bar, bar clicking the **Close** button
    - c). on the **Standard toolbar**, by clicking the **Stop Debugging** button (red square)
  - when you start a **Windows Forms** application from within the IDE, the computer automatically creates a file that can be run outside of the IDE
    - the file is referred to as an executable file and the name is same as the project's name, except it ends with **.exe**
    - however you can change the executable file's name in **Project Designer** window, if you desire
  - the computer stores the executable file in the project's folder: **bin\Debug**
- note:**
- when you are finished with an application, you typically give the user only the executable file because it does not allow the user to modify the application's code
  - to allow someone to modify the code, you need to provide the entire solution

### CH1\_A11.3 - To change the name of the executable file, and then start and end the application: the Einstein's Equation application example

1). open again the **Project Designer** window and **Application** pane, where you can change the name of the executable file in **Assembly name** -> **Figure 1-27**

-> change the filename "Einstein Project" to **My Einstein**

2). save the solution and close the **Project Designer** window by clicking it's Close button

3). start the application -> choose either a), b), or c). from **CH1\_A11.2 ...**

**note:** do not be concerned about any windows that appear at the bottom of the screen

4). recall that the purpose of the **Exit** button is to allow the user to end the application ->

-> click the **Exit** button - but nothing happens because you have not yet entered the instructions that tell the button how to respond when clicked

5). choose a method to stop the application

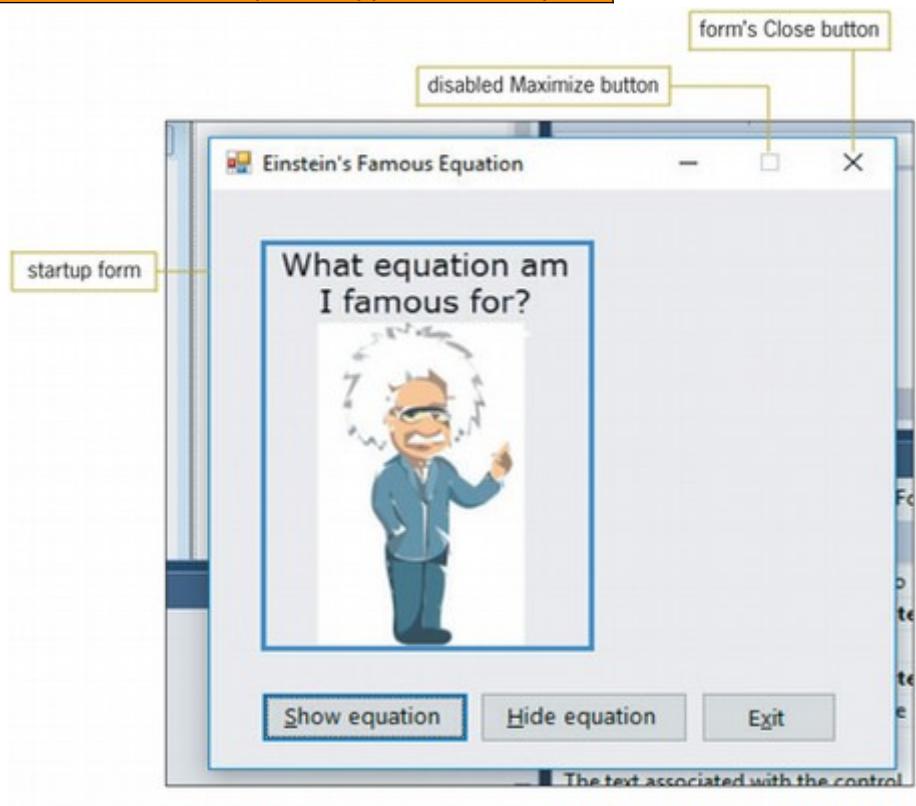


Figure 1-28 Result of starting the Einstein application

### CH1\_A12 - Enter Code and Comments in the Code Editor Window:

- after creating your application's interface, you can begin entering the Visual Basic instructions (code) that tell the controls how to respond to the user's actions ->  
-> those actions - such as clicking, double-clicking, scrolling, etc... - are called **events**

- you tell an object how to respond to an **event** by writing an **event procedure**, which is a set of VB instructions that are processed only when the **event occurs**

- instructions that are processed (executed) by the computer are also called **statements**

- you enter the procedure's code in the **Code Editor** window:

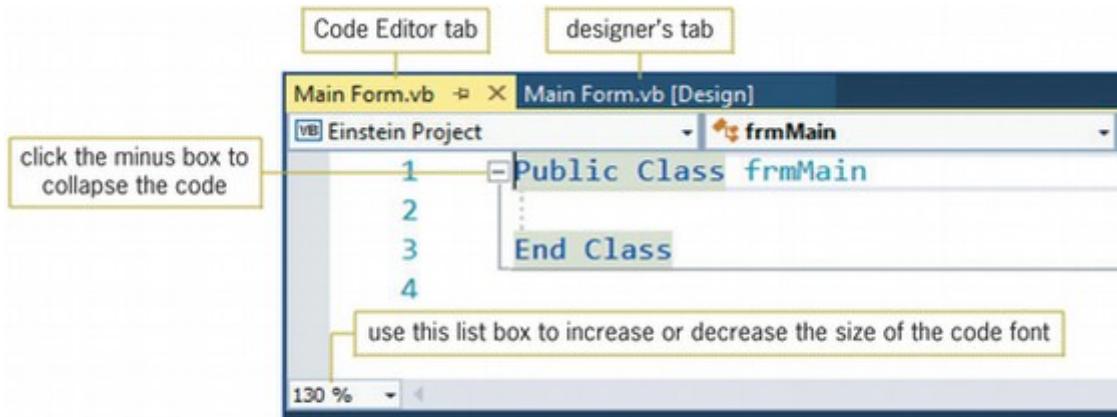
### CH1\_A12.1 - To open the Code Editor Window:

a). press the **F7** key on your keyboard

b). right-click the **form** and then on the context menu click: **View Code F7**

- either way, the **Code Editor** window opens in the IDE **Figure 1-29**

**note:** if the line numbers do not appear in your **Code Editor** window, on the menu bar click **Tools**, choose **Options**, expand the node **Text Editor**, click **Basic**, and select the check box: **Line numbers**, and then click the **OK** button



**Figure 1-29** Code Editor window opened in the IDE

- the **Code Editor** window contains the Class statement, which is used to define a class in Visual Basic - in this case, the class is the **frmMain** form
  - the Class statement begins with the **Public Class frmMain** clause and ends with the **End Class** clause
  - within the Class statement, you enter the **code** to tell the form and its objects how to react to the user's actions
- note:** the keyword **Public** in the Class statement indicates that the class can be used by code defined outside of the class

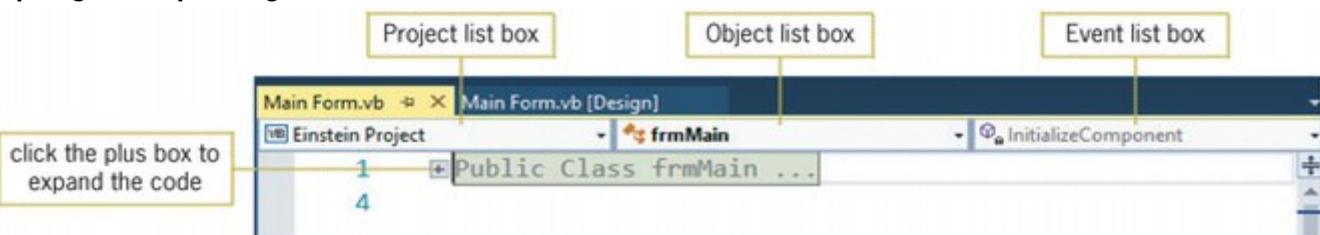
#### CH1\_A12.2 - To collapse and expand a region of code:

- if the **Code Editor** window contains many lines of code, you might want to hide the sections of code that you are not currently working with
- you hide a section (or region) of code by clicking the **minus box** that appears from left to it
- to unhide a region of code, you click the **plus box** that appears from left to the code
- hiding and unhiding the code is also referred to as **collapsing** and **expanding** the code

- 1). click the **minus box** that appears from left to the **Public Class frmMain** clause

- doing this collapses the Class statement, as shown in **Figure 1-29**

- 2). click the **plus box** to expand the code **Figure 1-30**



**Figure 1-30** Code collapsed in the Code Editor window

#### CH1\_A12.3 - To select the btnExit control's Click event:

- as **Figure 1-30** indicates, the **Code Editor** window contains **3** dropdown list boxes:

- 1st **Project list box**: - contains the name of the current project

e.g.: in this case it contains **Einstein Project**

- 2nd **Object list box**: - lists the names of the objects included in the user interface

- 3rd **Event list box**: - lists the events to which the selected object is capable of responding

- 1). click the 2nd **Object** list arrow and then in the list choose **btnExit**
  - 2). click the 3rd **Event** list arrow and then in the list choose **Click**
- a code template for the **btnExit** control's **Click** event procedure appears in the Code Editor window: **Figure 1-31**

```

Main Form.vb*  X Main Form.vb [Design]* 
VB Einstein Project  btnExit  Click
1  Public Class frmMain
2  Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
3
4  End Sub
5  End Class

```

**Figure 1-31** btnExit control's Click event procedure

- the Code Editor provides the code template to help you follow the rules of the Visual Basic language
- the rules of a programming language are called its **syntax**
- the first line in the code template is called the **procedure header**, and begins with the **keywords** **Private Sub**
- the last line **End Sub** in the code template is called the **procedure footer**
- a **keyword** is a word that has a special meaning in a programming language, and it appears in a different color from the rest of the code
- the keyword **Private** indicates that the buttons **Click** event procedure can be used only within the class defined in the current Code Editor window
- the keyword **Sub** is an abbreviation of the term **sub procedure**, which is a block of code that performs a specific task
- following the keyword **Sub** is the name of the **object**, an **underscore**, the name of the **event**, and **parentheses** containing some text:  
-> for now, you do not have to be concerned with the text that appears between the parentheses (**sender As Object, e As EventArgs**)
- after the closing parenthesis is the: **Handles btnExit.Click** -> this clause indicates, that the procedure is associated with the **btnExit** control's **Click** event  
-> it tells the computer to process the procedure only when the **btnExit** control is clicked

**Private Sub btnExit\_Click(sender As Object, e As EventArgs) Handles btnExit.Click**

**Click** = name of the event

**Handles btnExit.Click** =

it tells the computer to process the procedure  
only when the **btnExit** control is clicked

**btnExit** = name of the object

- the code template ends with the **procedure footer**, which contains the keywords **End Sub**
- you enter your Visual Basic instructions at the location of the **insertion point**, which appears between the **Private Sub** and **End Sub** clauses **Figure 1-31**
- the Code Editor automatically indents (odsadí) the lines between the procedure header and footer
- indenting the lines within a procedure makes the instructions easier to read and is a common programming practice

#### CH1\_A12.4 - the Me.Close() Statement:

- the **Me.Close()** statement tells the computer to close the current form
- if the current form is the only form in the application, closing it terminates the entire application

**Me** = keyword that refers to the current form  
**Close** = one of the **methods** available in Visual Basic  
**()** = parentheses are required when calling some Visual Basic **methods**  
- depending on the **method**, it might or might not be empty  
- if you forget them, the Code Editor will enter them for you when you move the insertion point to another line in the Code Editor window

- a **method** is a predefined procedure that you can call (or invoke) when needed

#### basic syntax:

**Me.Close()**

#### CH1\_A12.5 - to code the **btnExit\_Click** procedure:

- you can type the **Me.Close()** statement on your own, or use the Code Editor window's **IntelliSense** feature

1). in the insertion point type: **me.** -> be sure to type the period, but don't press Enter key

-> when you type the period, the **IntelliSense** feature displays a list of properties, methods, and so on, from which you can select

**note:** if the list of choices does not appear, the **IntelliSense** feature might have been turned off on your computer system ->

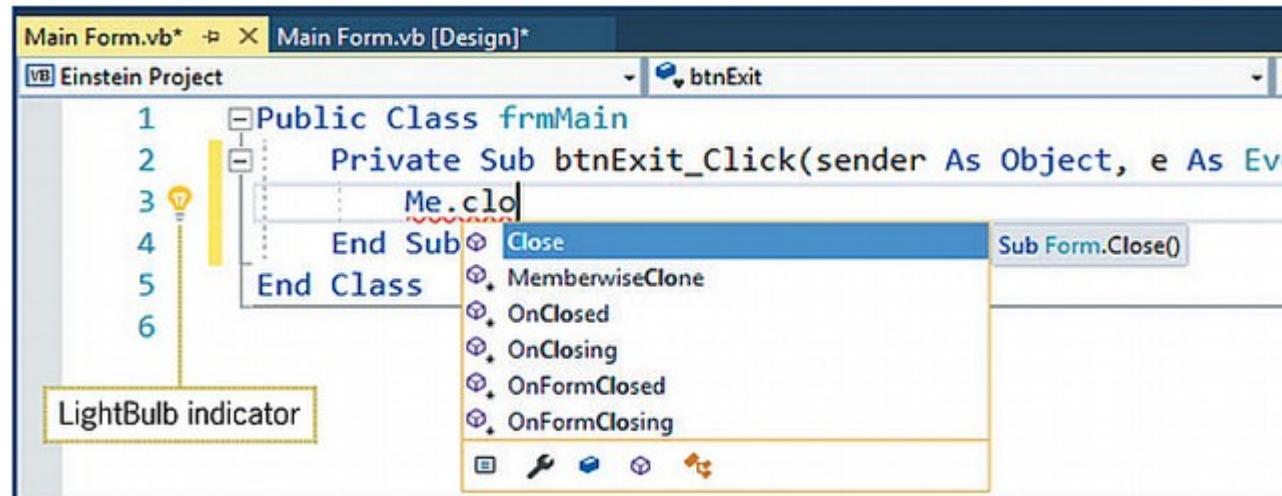
-> to turn it on, on the menu bar click **Tools**, then **Options**, on the left expand the nod **Text Editor** and choose the nod **Basic**, then on the right side locate: **Statement completion** and select the check box **Auto list members**

2). after the **me.** type without space: **clo** -> but don't press Enter key

- the **IntelliSense** feature highlights the **Close** method in the list

**Figure 1-32**

- for now, don't be concerned with the **LightBulb** indicator and the red jagged line -> **squiggle**



**Figure 1-32** List displayed by the IntelliSense feature

3). press **Tab** key to include the **Close** method in the statement and then press **Enter** key

4). save the solution and test it -> choose on of the methods from: **CH1\_A11.2 - Start and End an Application:** -> for example press the **F5** key

- it is a good idea to test a procedure after you have coded it so you will know where to look if an error occurs
- you can test the **Exit** buttons **Click** event procedure by starting the application and then clicking the button
- when the button is clicked, the computer will process the **Me.Close()** statement contained in the procedure

## CH1\_A12.6 - Assignment Statements:

- you can set the object's properties:

a). during **design time**: - you learned earlier in the chapters, how to use the **Properties** window when you are building the interface

- see: **CH1\_A5 - Change the Properties of a Form:**

**CH1\_A5.1** - to display the **form's** properties and the most commonly used:

**CH1\_A8.1** - To add a **picture box** to the Form: the Einstein's Equation application example

**CH1\_A8.2** - To add a **button** to the Form: the Einstein's Equation application example

b). during **run time**: - you do this by using an **assignment statement**, which is one of many different types of Visual Basic instructions ->

-> its purpose is to assign a value to something, such as to the **property** of an object

### basic syntax:

```
object.property = expression
```

**object**            <- (**name**) property of an object  
.  
<- dot member access operator (a period)  
<- used to separate the object name from the property name  
<- indicates, that the **property** is a **member** of the **object**  
**property**        - Visible, Width, Text.....etc.....  
=                    <- **assignment operator**  
**expression**        - can be a **keyword**, a **number**, or a **string literal** (which is defined as zero or more characters enclosed in quotation marks " ")  
                    - it can also be a **calculation**, but about that later - **CH3**

- when the computer processes an assignment statement, it assigns the value of the expression that appears on the right side of the assignment operator to the object and property that appear on the left side of the assignment operator

e.g.1: **picEquation.Visible = False**    <- to the **picEquation**'s **Visible** property assigns the keyword **False**

e.g.2: **lblDue.Width = 120**    <- to the **lblDue**'s **Width** property assigns the number **120**

e.g.3: **txtState.Text = "Ohio"**    <- to the **txtState**'s **Text** property assigns the string literal "**Ohio**"

## CH1\_A12.7 - Comments:

- in each procedure, you enter a **comment** that indicates the procedure's purpose

- a **comment** is a line of text that serves to internally document a program - you can recognize it by the **green colored font**

- you create it in Visual Basic by placing an **apostrophe** ' before the text that represents the comment

- the computer ignores everything that appears after the **apostrophe** ' on that line

- the conventions are to use a space to separate the **apostrophe** ' from the comment text, and begin the comment text with a Capital letter and end it with a period.

## CH1\_A12.8 - To enter a **comment** and **code**: the Einstein's Equation application example

- you will use **assignment statements** and **comments** to code the **btnShow\_Click** and **btnHide\_Click** procedures

1). use the 2nd **Object** list box and 3rd **Event** list box to open the code template for the **btnShow** object's **Click** event

-> in the insertion point type: ' **Show Einstein's equation.** and press **Enter** key

2). on the next line type: **piceq** to highlight **picEquation** in the list and then press **Tab** key

- then type: **.v** to highlight **Visible** in the list and then press **Tab** key

- type: **=t** to highlight **True** in the list and then press **Enter** key

- the statement: **picEquation.Visible = True** appears in the procedure

**CH1\_A12.3...**

**Figure 1-30**

-> **picEquation**

-> **picEquation.Visible**

-> **picEquation.Visible = True**

- 3). save the solution and then start the application  
click the button **Show equation** to display the image stored in the **picEquation** control, and then click the button **Exit**
- 4). on your own, open the code template for the **btnHide** object's **Click** event
  - type: ' Hide Einstein's equation. and press Enter key twice
  - now enter the statement in the procedure: **picEquation.Visible = False**
- 5). save the solution and then start the application  
- click the button **Show equation** to display the image stored in the **picEquation** control, and then click the button **Hide equation** to hide the image
- 6). now use the button's access key to show the image again: press and hold down the Alt key as you tap the letter **s**, and then release the Alt key  
- the **Alt + s** combination tells the computer to process (execute) the statements contained in the button's **Click** event procedure
- 7). use the **Hide equation** button's access key **Alt + h** to hide the image, and then use the **Exit** button's access key **Alt + x** to end the application
- 8). enter additional comments: **Figure 1-35**
  - click immediately before the letter **P** in the **Public Class frmMain** clause, and then press Enter key
  - enter the additional comments shown in **Figure 1-35** and enter there your name and the current date

```

Main Form.vb*  Main Form.vb [Design]*  frmMain  InitializeComponent
Einstein Project
1  ' Name: Einstein Project
2  ' Purpose: Show/hide an equation.
3  ' Programmer: <your name> on <current date>
4
5  Public Class frmMain
6      Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles Me.Close()
7          Me.Close()
8
9      End Sub
10
11     Private Sub btnHide_Click(sender As Object, e As EventArgs) Handles
12         ' Hide Einstein's equation.
13
14         picEquation.Visible = False
15
16     End Sub
17
18     Private Sub btnShow_Click(sender As Object, e As EventArgs) Handles
19         ' Show Einstein's equation.
20
21         picEquation.Visible = True
22
23     End Sub
24 End Class

```

**Figure 1-35** Additional comments entered in the Code Editor window

#### CH1\_A13 - Print an Application's Code and Interface:

- it is a good idea to print a copy of your application's code for future reference
- to print the code, the Code Editor window must be the active (current) window
- you might also want to print a copy of the GUI - you can do this using the Window's Snipping Tool, as long as the designer window is the active window

1). how to print the code: the Code Editor window is currently the active window

-> on the menu bar click **File**, and then click **Print Ctrl+P** to open the Print dialog box

-> if your computer is connected to a printer, select the appropriate one and then click button **Print**, otherwise click the **Cancel** button

- very handy if you wanna save a copy of your code in **.pdf**

2). how to print the GUI: the designer window is currently the active window

a). you might want to use a **Print Screen** key on your keyboard, and using some image editor, just crop the picture

b). you might run your application, and follow the steps in a).

c). on the Windows 10 taskbar click the **Start** button, click **Windows Accessories**, and then click **Snipping Tool**

-> there click the button **New** and hold down your left mouse button as you drag your mouse pointer around the form, and then release the mouse button

-> on the Snipping Tool's menu bar click **File** and then click **Print Ctrl+P**

#### CH1\_A14 - Exit Visual Studio and Run an Executable File:

- you can exit Visual Studio using either the **Close** button on its title bar or the **Exit Alt+F4** option on its **File** menu

- you can run an application's executable file **.exe** by locating the file in the project's: **bin\Debug** folder

1). click **File** and then **Exit Alt+F4**

2). open the folder: **VB2017\Chap01\Partial Einstein Solution\Einstein Project\bin\Debug** and then double-click the file: **My Einstein.exe**

3). test the application to verify that it works correctly

#### Mini-Quiz 1-7:

1. What is an event procedure?

2. What is a keyword?

3. If an application contains only one form, what Visual Basic statement tells the computer to end the application?

4. Write an assignment statement that assigns the keyword **False** to the **btnPrint** control's **Enabled** property.

5. In the Code Editor window, what character designates that the text that follows it, is a comment?

5...an apostrophe ,

4...btnPrint.Enabled = False

3...Me.Close()

2...a word with a special meaning in a programming language

(a sequence of steps the computer should do, when the event occurs)

1...a set of instructions processed only when its associated event occurs

#### CH1\_A15 - Splash Screen - additional topic from Appendix B - template & code

- some applications begin by displaying a **Splash screen** - form that introduces the app

- holds the user's attention while the rest of the application is being read into RAM

- to create: a). you can use VB's **Splash Screen** template, or

b). you can create one from scratch by using a **Windows form**

- to use the VB's **Splash Screen** template:

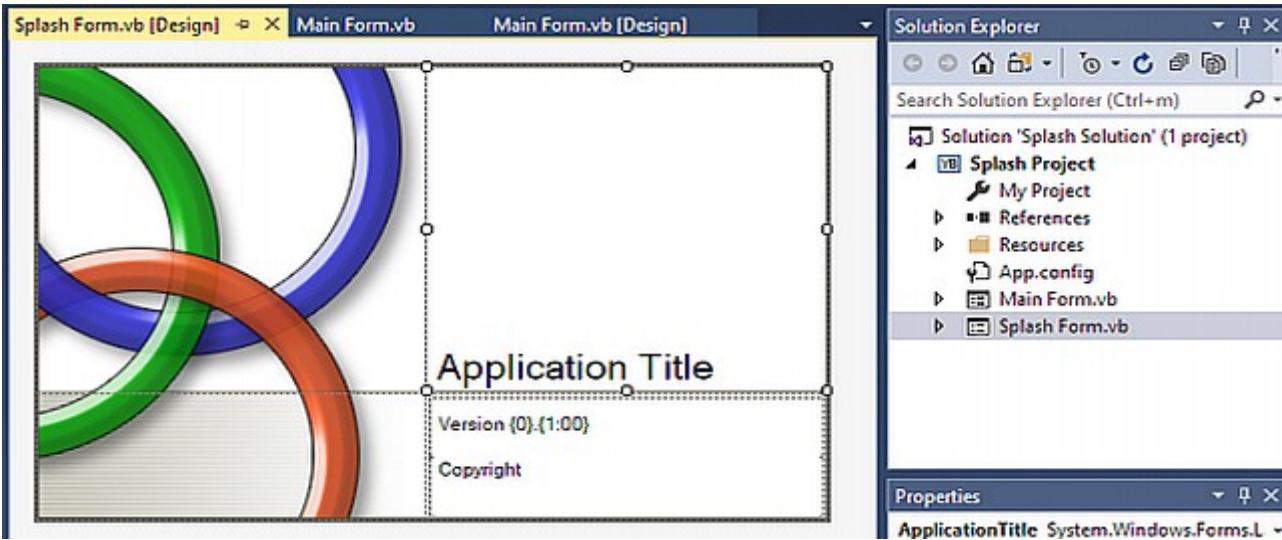
1). open the: ...**VB2017\Chap01\Exercise\Splash Solution\Splash Solution.sln**

2). open the **Solution Explorer** window, and **Designer** window

-> in VS toolbar click: **Project / Add New Item... Ctrl+Shift+A**

-> in the left column: expand the **Common Items** node and then click **Windows Forms**

-> in the middle column: choose **Splash Screen** and change the file's name from **SplashScreen1.vb** to **Splash Form.vb**, then click the button **Add**



-> in the tab **Splash Form.vb [Design]**, on the splash screen form, click the **Application Title** text to display its **Properties**

-> change the **Text** property from "Application Title" to: "**Welcome to my application!**"

-> in the **Solution Explorer** window, right-click **My Project** and then choose **Open** to open the **Project Designer** window

-> in the left column choose the first tab: **Application**

-> change the box **Startup form:** from **Form1** to **frmMain**

-> change the box **Splash screen:** from **(None)** to **Splash\_Form**

-> save the solution, close the **Project Designer** window and start the application

<- the splash screen appears first, then after several seconds it disappears and the **Main Form** appears

-> close the solution

- ERROR: the **Splash Screen** won't show the text "**Welcome to my application!**", instead it shows the **Application name**

### 3). entire code & GUI:

```

1  Public NotInheritable Class Splash_Form
2
3      'TODO: This form can easily be set as the splash screen for the application by going to the "Application" tab
4      ' of the Project Designer ("Properties" under the "Project" menu).
5
6
7      Private Sub Splash_Form_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load
8          'Set up the dialog text at runtime according to the application's assembly information.
9
10         'TODO: Customize the application's assembly information in the "Application" pane of the project
11         ' properties dialog (under the "Project" menu).
12

```

```

13     'Application title
14     If My.Application.Info.Title <> "" Then
15         ApplicationTitle.Text = My.Application.Info.Title
16     Else
17         'If the application title is missing, use the application name, without the extension
18         ApplicationTitle.Text = System.IO.Path.GetFileNameWithoutExtension(My.Application.Info.AssemblyName)
19     End If
20
21     'Format the version information using the text set into the Version control at design time as the
22     'formatting string. This allows for effective localization if desired.
23     'Build and revision information could be included by using the following code and changing the
24     'Version control's designtime text to "Version {0}.{1:00}.{2}.{3}" or something similar. See
25     'String.Format() in Help for more information.
26
27     ' Version.Text = System.String.Format(Version.Text, My.Application.Info.Version.Major, My.Application.Info.Version.Minor,
28     '   My.Application.Info.Version.Build, My.Application.Info.Version.Revision)
29
30     Version.Text = System.String.Format(Version.Text, My.Application.Info.Version.Major, My.Application.Info.Version.Minor)
31
32     'Copyright info
33     Copyright.Text = My.Application.Info.Copyright
34 End Sub
35
36 End Class

```



- original code from the book won't show the text entered in the **Text** property during the design time

<- if I comment/erase the following lines, the **Splash Screen** will show  
 <- the text entered in **Text** property during design time  
 <-  
 <-  
 <-



- with commented/erased lines **14 - 19**, the **Splash Screen** will show your text entered in the **Text** property during the design time

## CH1\_A16 - Timer control - additional topic from Appendix B - code example

- the purpose of a **Timer** Control is to process code at one or more regular intervals
- the length of each interval - specified in **milliseconds** (a millisecond is 1/1000 of a second; there are 1000 milliseconds in a second)
  - entered in the **Timer's** property **Interval**
- the **Timer's** state - determined by its property **Enabled**
  - set by either the Boolean value
    - **True** = running
    - **False** = stopped
- if the **Timer** is:
  - a). running - its **Tick** event occurs each time an interval has elapsed
    - each time the **Tick** event occurs, the computer processes any code contained in the **Tick** event procedure
  - b). stopped - the **Tick** event does not occur
    - and therefore, any code entered in the **Tick** event procedure is not processed

- to use a **Timer** Control:

1). open the: ...VB2017\Chap01\Exercise\Timer Solution\Timer Solution.sln

2). open the **Solution Explorer** window, and **Designer** window

3). display the **Toolbox** window, expand the node **Components** and locate the **Timer** control

-> drag the **Timer** tool to the form and release the mouse button

<- a **Timer** control appears in the component tray located at the bottom of the IDE

4). in the **Timer** control's **Property** window set:

-> (Name) = **tmrBlink**

-> Enabled = **True**

-> Interval = **1000** (1000 milliseconds = 1 second)

5). open the Code Editor window (F7 key)

-> open the code template for the **tmrBlink\_Tick** procedure

<- the lines of code appears:

```
2      Private Sub tmrBlink_Tick(sender As Object, e As EventArgs) Handles tmrBlink.Tick
3
4      End Sub
```

-> type the following assignment statement

<- you will learn about the **Not** logical operator in CH4 - the operator changes **True** to **False** and changes **False** to **True**

```
2      Private Sub tmrBlink_Tick(sender As Object, e As EventArgs) Handles tmrBlink.Tick
3          picSmile.Visible = Not picSmile.Visible
4      End Sub
```

6). save the solution and start the application

<- a blinking picture box appears on the form

-> close the solution

7). entire code and GUI:

```
1  Public Class frmMain
2      Private Sub tmrBlink_Tick(sender As Object, e As EventArgs) Handles tmrBlink.Tick
3          picSmile.Visible = Not picSmile.Visible
4      End Sub
5  End Class
```



**CH1\_A17 - PrintForm Control** - additional topic from **Appendix B** - code example; installing Visual Basic PowerPack

- VB provides the **PrintForm** tool for printing an interface from code
- the tool is contained in the **Visual Basic PowerPacks** section of the **Toolbox**
- if your **Toolbox** does not contain the **PowerPacks** section, the next set of steps will show you how to add the section to your **Toolbox**
- when you drag the **PrintForm** tool to a form, a **Print Form** control appears in the component tray located at the bottom of the IDE
- you can use the control to send the printout
  - a). to a file
  - b). the Print preview window
  - c). directly to the printer

- to use a **Print Form** Control:

1). open the: ...VB2017\Chap01\Exercise\PrintForm Solution\PrintForm Solution.sln

2). open the **Solution Explorer** window, and **Designer** window

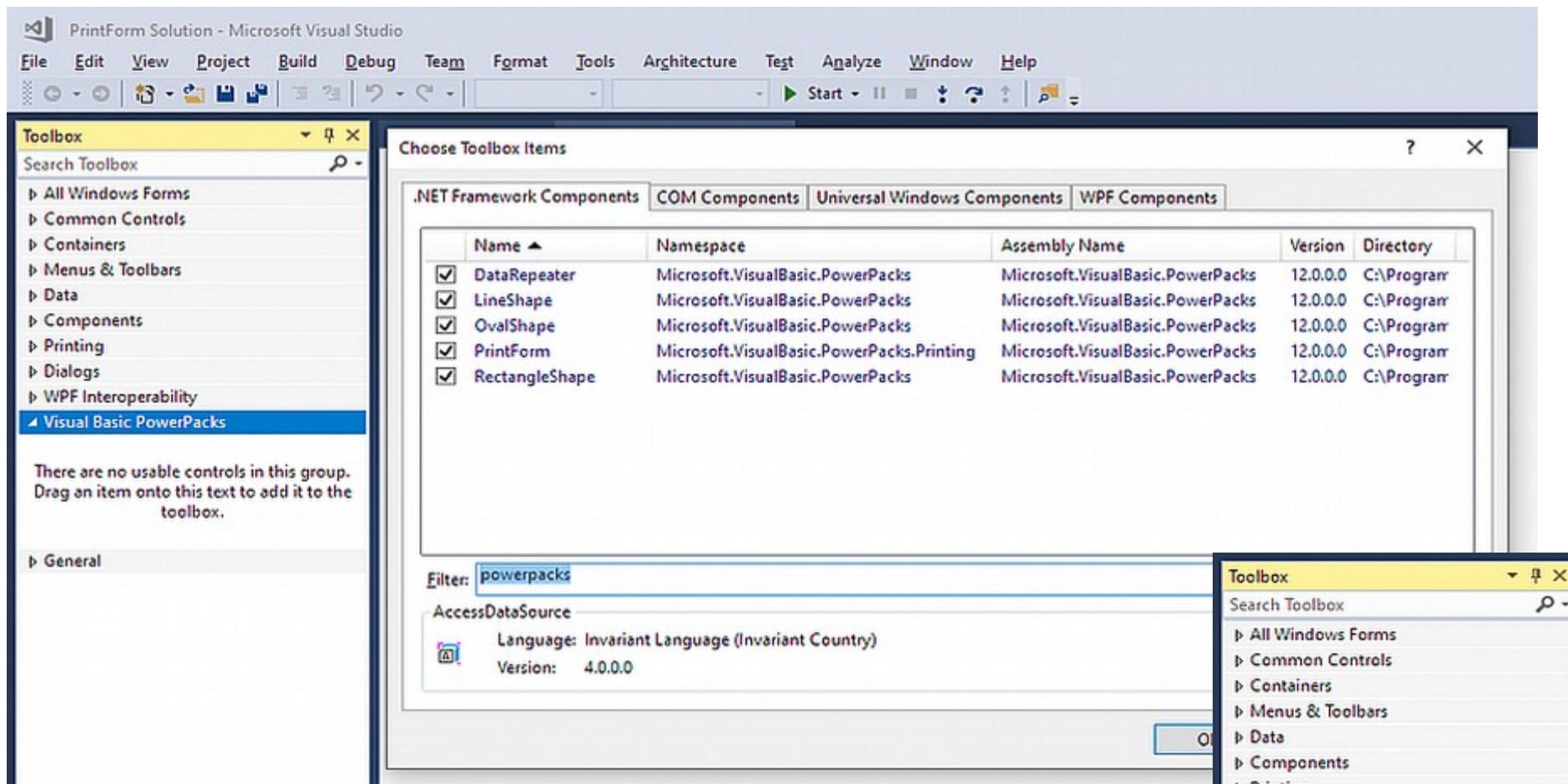
3). display the **Toolbox** window, scroll down to see if it contains the **Visual Basic PowerPacks** section

-> if it does, you can skip to Step 4).

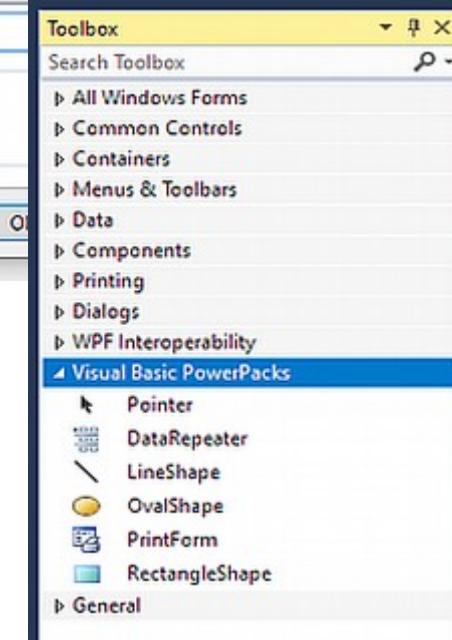
-> otherwise follow the next set of steps to install the **Visual Basic PowerPacks**:

- 1). close the solution and exit Visual Studio
- 2). open the folder **Appendix\_B** and install the **vb\_vbpowerpacks.exe** file
- 3). start **Visual Studio** and open the **PrintForm Solution.sln** file again
- 4). right-click the **Toolbox** window, select the Add Tab, type: **Visual Basic PowerPacks** and press the **Enter** key

- 5). right-click just created tab **Visual Basic PowerPacks**, and click **Choose Items...**
- 6). in the new dialog box **Choose Toolbox Items**, select the tab **.NET Framework Components**
- 7). in the **Filter:** box type: **powerpacks** and select the **PrintForm** control
  - you can also select the other controls like **DataRepeater**, **LineShape**, **OvalShape**, **RectangleShape**, but they are not covered in this appendix
- 8). click the **OK** button to close the dialog box



- 4). in the **Toolbox**, expand the **Visual Basic PowerPacks** node
  - 5). click **PrintForm** and drag the control to the form
- <- a Print Form control appears in the Component tray
- > in the Properties window, set the: **PrintAction = PrintToPreview**



6). open the **Code Editor** window, and then:

-> open the code template for the **btnPrint\_Click** procedure and enter the following code:

```
6      Private Sub btnPrint_Click(sender As Object, e As EventArgs) Handles btnPrint.Click
7          ' Print the sales receipt without the button
8          btnPrint.Visible = False
9          btnExit.Visible = False
10         PrintForm1.Print()
11         btnPrint.Visible = True
12         btnExit.Visible = True
13     End Sub
```

7). save the solution and then start and test the application

-> click the button **Print**

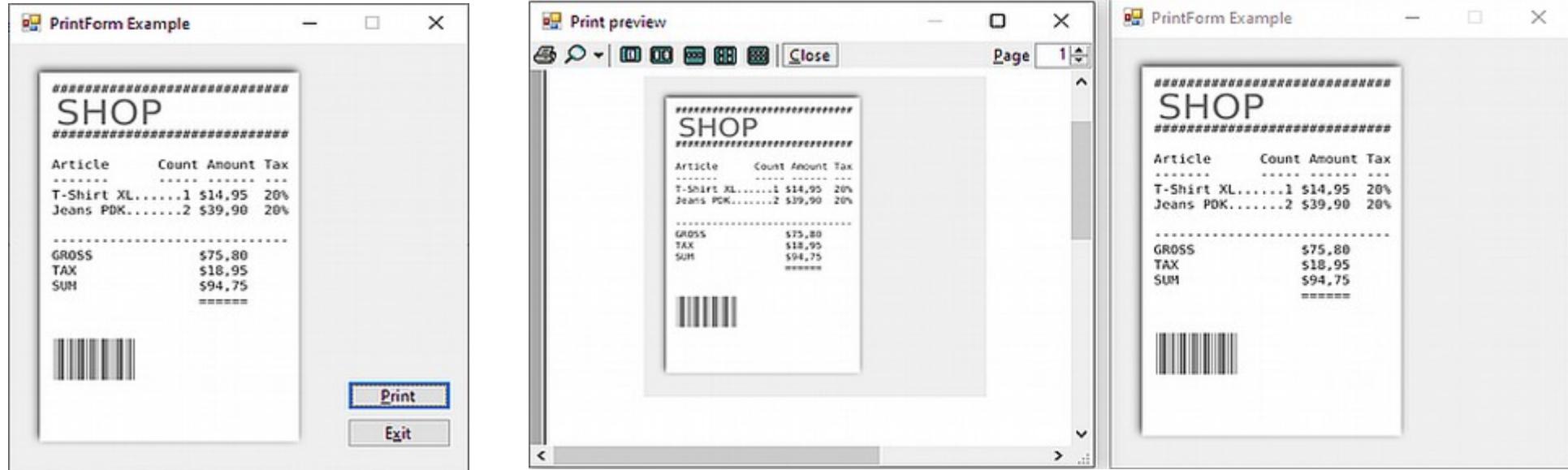
<- a printout of the interface appears in the **Print preview** window

-> click the **Zoom** button's list arrow and then choose **75%**

8). you won't need to print the sales receipt, so click the **Close** button, exit the the application and close the solution

9). entire code and GUI:

```
1  Public Class frmMain
2      Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
3          Me.Close()
4      End Sub
5
6      Private Sub btnPrint_Click(sender As Object, e As EventArgs) Handles btnPrint.Click
7          ' Print the sales receipt without the button
8          btnPrint.Visible = False
9          btnExit.Visible = False
10         PrintForm1.Print()
11         btnPrint.Visible = True
12         btnExit.Visible = True
13     End Sub
14
15 End Class
```



## CH1\_Summary

- programs are the step-by-step instructions that tell a computer how to perform a task
- programmers use various programming languages to communicate with the computer
- programmers are responsible for translating a problem's solution into instructions that the computer can understand
- programmers rigorously test a program before releasing the program to the user
  
- an object-oriented programming language (OOP), such as Visual Basic, enables programmers to use objects like buttons, etc... to accomplish a program's goal
  - an object is anything that can be seen, touched, or used
- every object in an OOP is instantiated-created from a class, which is a pattern that tells the computer how the object should look and behave
  - an object is referred to as an instance of the class
  - each object has a set of properties that determines its appearance and behavior
  
- the 4 windows you use most often when designing your application's GUI:
  - **designer** window, of course
  - **Toolbox** window
  - **Solution Explorer** window
  - **Properties** window
  
- each tool in the Toolbox represents a class
  
- Windows applications in Visual Basic are composed of **solutions**, **projects**, and **files**
- you enter your application's program instructions in the Code Editor window
- an object's name, which is entered in its **(Name)** property, can be used to refer to the object in code

- an access key allows the user to select a control by pressing the **Alt** key in combination with the access key (ampersand & precede it)
- when you start a Visual Basic application, the computer automatically creates an executable file that can be run outside of the IDE -> the file's name ends with **.exe**
- you tell an object how to respond to the user's actions by writing an **event procedure**, which can contain comments and Visual Basic statements
- during run time, you can use an **assignment statement** to assign a value to an object's property
- all comments in the Code Editor window begin with an apostrophe

### **CH1\_Key Terms:**

- **Application** - a GUI with its program instructions
- **Assignment operator** - the equal sign (**=**) in an assignment statement
- **Assignment statement** - an instruction that assigns a value to something, such as to the property of an object
- **Camel case** - used when entering object names (Name)
  - the practice of entering the object's 3-char ID in lowercase and then capitalizing the first letter of each subsequent word in the name (**lblShowThis**)
- **Class** - a pattern that the computer uses to create (instantiate) an object
- **Class definition** - a block of code that specifies (or defines) an object's appearance and behavior
- **Code** - program instructions
- **Code Editor window** - where you enter the program instructions-code for your application
- **Coding** - the process of translating a solution into a language that the computer can understand
- **Comment** - a line of text that serves to internally document a program, begins with an apostrophe '
- **Computer program** - the directions given to computer
- **Controls** - objects like a picture boxes, buttons, etc... added to a form
- **Design time** - occurs when you are building an interface
- **Design window** - used to create an application's GUI
- **Dot member access operator** - the period (.) used to indicate a hierarchy
- **Event procedure** - a set of VB instructions that tell an object how to respond to an event
- **Events** - actions to which an object can respond; examples include clicking and double-clicking
- **Executable file** - a file that can be run outside of the IDE; the filename ends with the **.exe**
- **Form** - the foundation for the user interface in a Windows Forms app; also called a **Windows Form object**
- **Form file** - a file that contains the code associated with a Windows form
- **Graphical user interface - GUI**, what the user sees and interacts with while your program is running
- **Integrated development environment - IDE**, an environment that contains all of the tools and features you need to create, run and test your programs
- **Instance** - an object created (instantiated) from a class
- **Instantiated** - the process of creating an object from a class
- **Keyword** - a word that has a special meaning in a programming language
- **Me.Close() statement** - tells the computer to close the current form
- **Method** - a predefined procedure that you can call (invoke) when needed
- **Name property** - assigns a name to an object (**Name**) ; the name can be used to refer to the object in code
- **Namespace** - a block of memory cells inside the computer; contains the code that defines a group of related class
- **Object** - anything that can be seen, touched, or used
- **Object-oriented programming language - OOP**, a programming language that allows the programmer to use objects to accomplish a program's goal
- **Picture box - (pic)**, used to display an image
- **PrintForm tool** - a tool for printing an interface from code; included in Visual Basic's Power Packs
- **Procedure header / footer** - the first / last line in a procedure
- **Program** - the directions given to computers; also called a **computer program**

- **Programmers** - the people who write computer program
- **Programming** - the process of giving a mechanism the directions to accomplish a task
- **Programming language** - languages used to communicate with a computer
- **Properties** - the attributes that determine an object's appearance and behavior
- **Properties window** - lists an object's attributes (properties)
- **Reference control** - the first control selected in a group of controls; this is the control whose size and / or location you want the other selected controls to match
- **Run time** - occurs while an app is running
- **Solution Explorer** - displays a list of the projects contained in the current solution and the items contained in each project
- **Source files** - files that contain program instructions; in VB, the names of source files end with **.vb**
- **Startup form** - the form that appears automatically when an application is started
- **Statements** - VB instructions that are processed (executed) by the computer
- **String literal** - zero or more characters enclosed in quotation marks " "
- **Sub procedure** - a block of code that performs a specific task
- **Syntax** - the rules of a programming language
- **Timer control** - used to process code at one or more regular intervals, measured in milliseconds; the code is entered in the control's Tick event procedure
- **Toolbox window** - contains the tools used when creating an interface (each tool represents a class); referred to more simply as the toolbox

The End, Konec, Ende, Fine, Basta

plan a Windows Forms app; windows standards for lbl, btn, graphics, fonts, colors; Access keys - Exit; Tab order - TabIndex; create a Planning Chart; design GUI using Windows standards; add a (lbl) to the form; add a (txt) to the form

## CH2\_FOCUS ON THE CONCEPTS LESSON

- CH2\_F1** - Planning a Windows Forms application: example with Restaurant Tip application
- CH2\_F2** - Windows standards for interfaces: lbl, btn, graphics, fonts, color
- CH2\_F3** - Access keys - Alt keys: E&xit = Exit
- CH2\_F4** - Tab order - TabIndex: example with Restaurant Tip application

## CH2\_APPLY THE CONCEPTS LESSON

- CH2\_A1** - Create a Planning Chart for a Windows Forms application: example with Jacobsons Furniture store application
- CH2\_A2** - Design an interface using the Windows standards: example with Jacobsons Furniture store application
- CH2\_A3** - Add a label control to the form (lbl)
- CH2\_A4** - Add a text box to the form (txt)
- CH2\_A5** - Set the Tab Order via Designer window

## CH2\_Summary

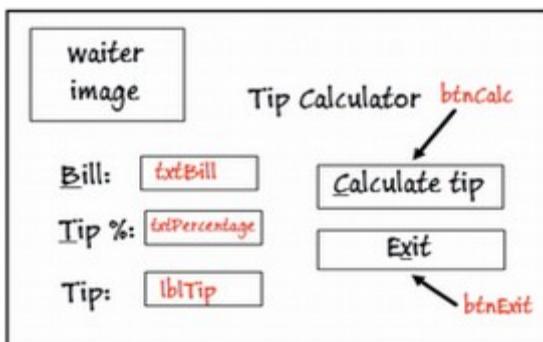
## CH2\_Key Terms

## CH2\_FOCUS ON THE CONCEPTS LESSON:

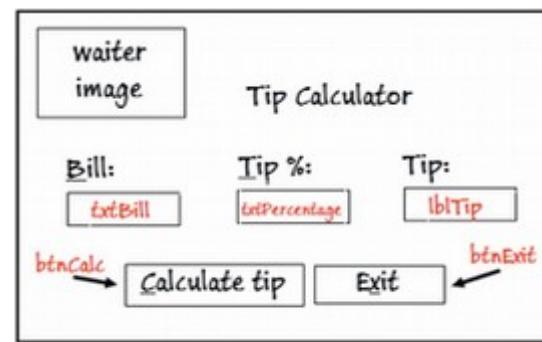
### CH2\_F1 - Planning a Windows Forms application: example with Restaurant Tip application

- |                                         |                                                       |
|-----------------------------------------|-------------------------------------------------------|
| - 1...what should it do?                | <- to calculate and display a server's tip            |
| - 2...what the user must provide?       | <- the bill amount and the tip percentage             |
| - 3...what the app must provide?        | <- the tip amount & button to end the app             |
| - 4...how 2. and 3. will work together? | <- section How? in Planning Chart                     |
| - 5...draw a sketch of the GUI          | <- make it easy and understandable                    |
|                                         | <- controls: - vertical arrangement <b>Figure 2-4</b> |
|                                         | - horizontal arrangement <b>Figure 2-5</b>            |

| Planning Chart for the Restaurant Tip app                         |                                                                                                                               |
|-------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------|
| Purpose: Calculate and display a server's tip.                    |                                                                                                                               |
| <u>User-provided:</u><br>1. bill amount<br>2. tip percentage      | <b>How?</b><br>- user will enter in <code>txtBill</code><br>- user will enter in <code>txtPercentage</code>                   |
| <u>App-provided:</u><br>1. tip<br>2. button for ending<br>the app | - <code>btnCalc_Click</code> will calculate & display in <code>lblTip</code><br>- <code>btnExit_Click</code> will end the app |



**Figure 2-4** Vertical arrangement of the interface



**Figure 2-5** Horizontal arrangement of the interface

## CH2\_F2 - Windows standards for Interfaces: lbl, btn, graphics, fonts, color

- familiar look will make it easy for users
- related controls should be grouped together using either white (empty) space or one of the tools found in the **Containers** section of the **Toolbox** ->
  - > like **GroupBox**, **Panel**, **TableLayoutPanel** tool... - more in Chapter 4
- place related controls close to each other
- maintain a consistent maring from the edges of the form
- minimize the number of different margins appearing in the interface, so it is more easy for the user to scan the info

### - Guidelines for identifying labels, buttons; including graphics; selecting fonts:

- lbl:**
  - text should be meaningful
  - 1-3 words in one line above, or to the left of the control it identifies
  - an identifying label should end with a colon (:), which distinguishes it from other text
  - capitalize only the **first** letter in the first word and in any words that are customarily (obvykle, tradičně) capitalized
- btn:**
  - are identified by the text that appears in the button's face
  - the text is often referred to as the button's **caption**
  - meaningful, 1-3 words in one line, capitalization
  - if stacked vertically, use the same height and width (**Figure 2-4**)
  - if horizontally, use the same height, but widths may vary if necessary (**Figure 2-5**)
  - in a **group** of buttons, the **most** commonly **used** one typically appears **first**
- graphics:**
  - human eye is attracted to pictures before text, so use graphics sparingly (střídme)
  - do not distract, use it maybe for personal touch?
- fonts:**
  - use only one font type (typically Segoe UI used by Microsoft, or Liberation Mono for Red Hat)
  - max 2 sizes and avoid italics and underlining - otherwise difficult to read
  - bold should be limited to titles, headings and key items that you want to emphasize
  - you have to buy a licence to use a font from Microsoft, or other creators !!!**
  - use only free fonts, but make sure they are really free, not any restriction included !!!**
  - free typefaces : Arvo (s padkama), Source Sans Pro (bezpadkovy), Audrey (elegant), ....**
  - free typefaces :**
- color:**
  - human eye is attracted to colour before BW
  - it is a good practice to use B, W, G first and then add color if you have a good reason to do so
  - keep in mind
    - people with color blindness / confusion could have a trouble distinguishing (rozeznání) colors
      - color is very subjective, it may look pretty to you, but others could find it hideous
      - a color may have a different meaning in a different culture
  - usually, use black text on white-off white-light gray background - because dark text on light background is the easiest to read
  - dark background is hard on the eyes / light-colored text can appear blurry (rozmazaně)
  - if you wanna include colours in a GUI, limit the number to 3 and be sure they complement each other
  - although color can be used to identify an important element in the interface - you should never use it as the only means of identification, include an id label for the colored control

## CH2\_F3 - Access keys - Alt keys: E&xit = Exit

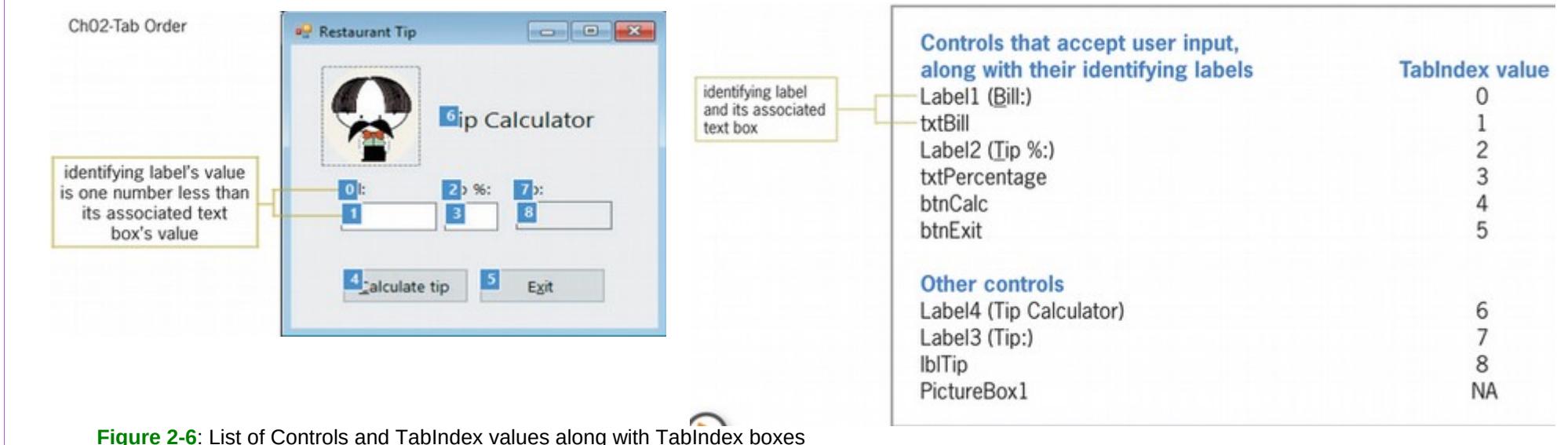
- are not case sensitive
- if can't see, press Alt for temp show
- should assign to each control that can accept user input
- except **OK** and **Cancel** button - which typically do not have access keys in Windows applications
- meaningful letter - the first letter, consonant, vowel, number

## CH2\_F4 - Tab order - TabIndex: example with Restaurant Tip application

designated window active, then – **Menu / View / Tab Order**

- Property / **TabIndex** - but not all controls have it (a **PictureBox**...)
- each control's **TabIndex** property contains a number, beginning with 0, that represents the order in which the control was added to the form
- TabIndex value determine the **tab order**, in which each control receives the **focus** when either pressed **Tab** or **Access key**
- when a **control** has the **focus** (blinking cursor for **txt** input, blue rectangle for **btn** input) , it can accept user **input**
- most times you will need to reset the TabIndex values - first make a list of the controls in an order you want them
- identifying label's value in one number less than its associated text box's value

e.g.: Restaurant Tip Application    Figure 2-6



- You can view those small blue boxes if designer window is active, via : **Menu / View / Tab Order**

## CH2\_APPLY THE CONCEPTS LESSON:

### CH2\_A1 - Create a planning chart for a Windows Forms application: example with Jacobsons Furniture store application

e.g.: Jacobson Furniture store

- for every purchase made at the store, cashiers manually calculate the amount of sales tax to charge the customer as well as the total amount the customer owes
- you will create an app what will computerize both calculations

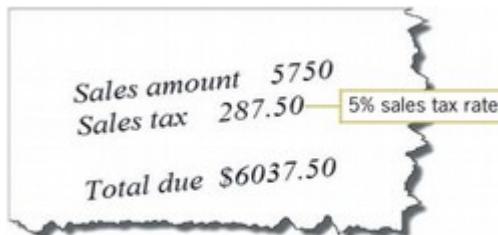


Figure 2-7 Sample of manual calculations

- you should plan a Windows Forms app before you begin creating it:

#### Step #1: identify the app's purpose

- it needs to calculate and display the sales tax and total due amounts.

#### Step #2: identify the items that the user must provide to accomplish app's purpose

- the cashier needs to provide only one item, the sales amount.

#### Step #3: identify the items that the app must provide

- 5 percent sales tax
- total due
- button to end the app

#### Step #4: determine how the user and the app will provide their respective items

- the user provided item (sales amount) will be entered in a text box named `txtSales`
- the app will use a button named `btnCalc` to calculate 5 percent sales tax and total due items
  - the button will also be responsible for displaying the result of the calculations in two label controls named `lblTax` and `lblTotal`
  - the button will perform these tasks when it is clicked, so you will code its **Click event procedure**
- the app will also provide a button for the user to end the app
  - `btnExit`
- see **Figure 2-9** for completed Planning Chart

### Planning Chart for the Jacobson Furniture application

Purpose: Calculate and display the sales tax and total due amounts.

How?

User-provided

1. sales amount

Application-provided

1. 5% sales tax
2. total due
3. button for ending the application

Figure 2-8 Planning Chart showing the results of the first three planning steps

### Planning Chart for the Jacobson Furniture application

Purpose: Calculate and display the sales tax and total due amounts.

How?

User-provided

1. sales amount

user will enter in `txtSales`

Application-provided

1. 5% sales tax
2. total due
3. button for ending the application

`btnCalc_Click` will calculate and display in `lblTax`  
`btnCalc_Click` will calculate and display in `lblTotal`  
`btnExit_Click` will end the application

Figure 2-9 Completed Planning Chart for the Jacobson Furniture application

CONTINUE NEXT PAGE:

## CH2\_A2 - Design an interface using the Windows standards: example with Jacobsons Furniture store application

CONTINUE FROM PREVIOUS PAGE:

Step #5: draw a sketch for the GUI :

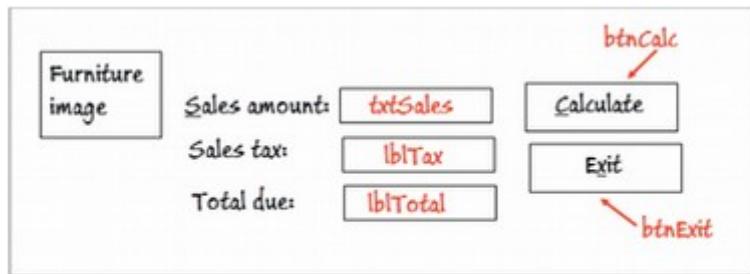


Figure 2-10 Vertical arrangement of the Jacobson Furniture interface

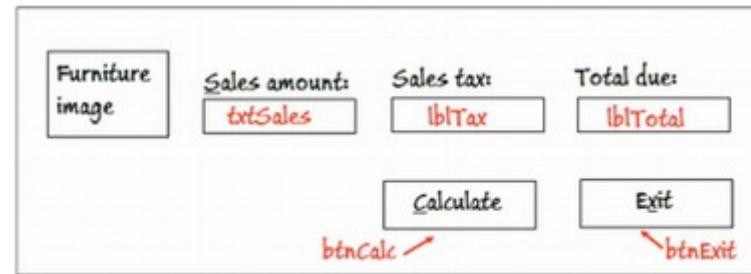


Figure 2-11 Horizontal arrangement of the Jacobson Furniture application

my own personal jesus:  
- make it idiot proof  
- not every shop assistant knows IT, so make it more easy then in a pictures, in a way like - Enter the amount:, tax; customer will pay...

## CH2\_A3 - add a Label Control to the form: (lbl)

- the purpose of a **label** control is to display text the user is not allowed to edit
- used to:
  - identify other controls <- typically: Autosize = True, BorderStyle = None, Text = info for other control
  - display the result of calculations <- typically: Autosize = False, BorderStyle = FixedSingle, Text nothing
- the most commonly used properties:
  - AutoSize**      - enable/disable auto sizing  
                  -> **identifying** labels = **True** typically  
                  -> labels that **display output** = **False** typically
  - BackColor**      - label's background color
  - BorderStyle**      - appearance of the label's border  
                  -> **identifying** labels = **None** typically  
                  -> labels that **display output** = **FixedSingle** typically
  - Font**      - font to use for text
  - ForeColor**      - color of the text inside the label
  - (Name)**      - **lbl+MeaningfulName**
  - Text**      - text that appears inside the label  
                  -> if identifies another control, that can accept user input, include access key (**&**)
  - TextAlign**      - position of the text inside the label

- homework : open VB2017/Chap02/Jacobson Solution and add missing stuff - page 061

## CH2\_A4 - add a Text Box to the form: (txt)

- the purpose of a **text box** is to provide an area in the form where the user can enter data
- the most commonly used properties:

|                        |                                                                                                                                                                      |
|------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>BackColor</b>       | - text box's background color                                                                                                                                        |
| <b>CharacterCasing</b> | - specify whether the text should remain as typed or converted to either UPPERCASE or lowercase                                                                      |
| <b>Font</b>            | - no comment                                                                                                                                                         |
| <b>ForeColor</b>       | - font colour                                                                                                                                                        |
| <b>(Name)</b>          | - txt+MeaningfulName                                                                                                                                                 |
| <b>MaxLength</b>       | - max number of characters                                                                                                                                           |
| <b>Multiline</b>       | - whether the text box can span more than one line<br>-> handy when using <b>txt</b> as an output box instead of <b>Ibl</b>                                          |
| <b>PasswordChar</b>    | - specify the character to display when entering a password                                                                                                          |
| <b>ReadOnly</b>        | - whether the text can be edited<br>-> <b>True</b> when using <b>txt</b> as an output box instead of <b>Ibl</b>                                                      |
| <b>ScrollBars</b>      | - used with a Multiline; appearance of scroll bars on the text box<br>-> <b>True</b> when using <b>txt</b> as an output box instead of <b>Ibl</b>                    |
| <b>TabStop</b>         | - whether the text box <b>can</b> receive the focus when user presses the <b>Tab</b><br>-> <b>False</b> when using <b>txt</b> as an output box instead of <b>Ibl</b> |
| <b>Text</b>            | - get or set the text that appears inside the text box                                                                                                               |

## CH2\_A5 - set the Tab Order via Designer Window:

- order in which each control receives the focus when the user either presses the **Tab** key or employs an **access** key
- determined by the **TabIndex** values of the controls in the GUI
- when Designer window active, go to: **Menu / View / Tab Order**
  - when shown, change numbers via small plus
  - when done, press **Esc** key or again Menu/View/Tab Order

## CH2\_Summary:

- to plan a Windows Forms app, perform the following 5 steps:
  1. what should it do ?
  2. what the user must provide ?
  3. what the app must provide ?
  4. how 2. and 3. will work together ?
  5. draw a sketch of GUI
- use a **Label** control to only display text, what can't be edited
- use a **Text Box** control, in which user can enter data
- don't use Access keys - Alt keys - & keys, where user can't enter data
- don't forget to use the **Tab Order**
- **Ibl** that display data:
  - property **BorderStyle** on **FixedSingle** - single border around
  - **AutoSize** property to **False**, otherwise the border will change its size
- **Ibl** that only identify:
  - **BorderStyle** on **None**
- **txt** input:
  - default **BorderStyle** on **Fixed3D**, same like **btn** --- rule for anything that accepts user input

## **CH2\_Key Terms:**

**Access key:** allows the user to select the object using the Alt key in combination with the underlined character

**Focus:** indicates that a control is ready to accept user input

**Label control:** used to display that user is not allowed to edit

**Sentence capitalization:** used to capitalize only the first letter in the first word and words customarily capitalized

**Tab order:** the order in which each control receives the focus when **Tab** or **access key** is pressed

**Text box:** a control that provides an area in the form for the user to enter data

# Coding with Variables, Named Constants & Calculations

pseudocode & flowchart; Variables scope level: **For, Dim, Static, Private**; Named Constants scope level: **Const, Private Const**; Arithmetic Operators (#\$1-6): **^ - \* / \ Mod + -**; TryParse method; ToString method; Option statements; InputBox function; event procedures **txt\_TextChanged** and **txt\_Enter**

| <u>scope from smallest:</u> |                    |                      |
|-----------------------------|--------------------|----------------------|
|                             | <i>variable</i>    | <i>constant</i>      |
| 1. Block-level              | <b>For</b>         |                      |
| 2. Procedure-level          | <b>Dim, Static</b> | <b>Const</b>         |
| 3. Class-level              | <b>Private</b>     | <b>Private Const</b> |
| 4. Form-level               | <b>Friend</b>      |                      |

## CH3\_FOCUS ON THE CONCEPT LESSON

- CH3\_F1 - Pseudocode and Flowchart: beneficial planning tools for your code
- CH3\_F2 - Main Memory of a Computer: basic info & reserving/declaring a memory location: **variables & named constants**
- CH3\_F3 - Variable: declaring a variable memory location for your data: syntax & example
- CH3\_F3.1 - Data types for your **variables & named constants**: my tab with a detailed info
- CH3\_F4 - TryParse method: converts **string** into **numeric** data type - syntax & example
- CH3\_F5 - Arithmetic Operators (#\$1-6): **^ - \* / \ Mod + -**
- CH3\_F6 - Assigning a Value to an **existing Variable**:
- CH3\_F7 - ToString method and it's format specifiers **C** or **F** or **N** or **P**: converts **numbers** into **strings** - syntax & e.g.
- CH3\_F8 - **Option statement** - above all in a code - including explanation: **Option Explicit** & **Option Strict** & **Option Infer**
- CH3\_F9 - Named Constant declaring a constant memory location for your data: syntax & example

## CH3\_APPLY THE CONCEPT LESSON

- CH3\_A1 - determine a memory location's **Scope** and **Lifetime**: **Procedure-level** vs **Class-level**
- CH3\_A2 - use a **Procedure-Level Variables**: the **Dim** statement
- CH3\_A3 - use a **Procedure-Level Named Constants**: the **Const** statement
- CH3\_A4 - use a **Class-Level Variable**: the **Private** statement
- CH3\_A5 - use a **Procedure-Level Variable** with a **Class-Level scope**: the **Static** statement
- CH3\_A6 - use a **Class-Level Named Constant**: the **Private Const** statement
- CH3\_A7 - Professionalize Your Application's Interface: event procedures **txt\_TextChanged** and **txt\_Enter**
- CH3\_A8 - InputBox function - additional topic from **Appendix B** - interact with user while app is running - syntax & example

**CH3\_Summary**

**CH3\_Key Terms**

**CH3\_Exercises**

## CH3\_FOCUS ON THE CONCEPTS LESSON :

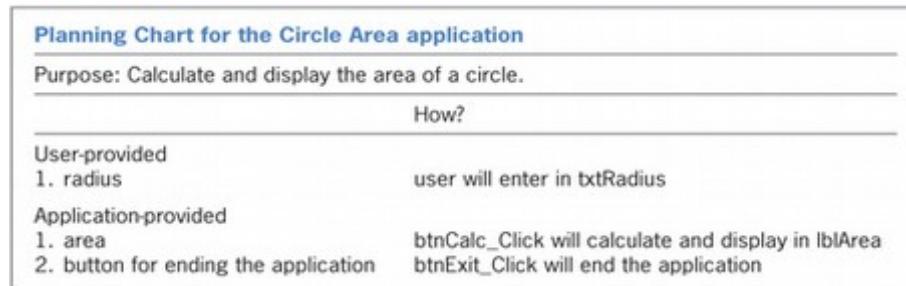
### CH3\_F1 - Pseudocode and Flowchart: beneficial planning tools for your code

- you learned in Chapter 2 how to plan a **Windows Forms** app, design and build **GUI** - now how to code it?

#### Circle Area Application example:

- 1). it is important to plan a procedure before you begin coding it:

**Figure 3-1:** Planning Chart and user interface for the Circle Area application



- 2). many programmers use **planning tools** such as **pseudocode** and **flowcharts** - pick one of them, no need to use both

#### Pseudocode

- uses short phrases to describe the steps a procedure must take to accomplish its goal.

#### btnCalc\_Click

1. declare variables
2. convert **txtRadius.Text** property to a number and store in a variable
3. calculate the area by multiplying pi by the radius squared (3.14159 for pi)
4. display the area in **lblArea** (converts number to string)

#### btnExit\_Click

end the application

#### Flowchart

- uses standardized symbols to illustrate a procedure's steps

##### symbols:

- an oval:** - called **start/stop** symbol
  - indicate the beginning and end of the flowchart
- a rectangle:** - called **process** symbols
  - represent tasks; like ending the app, making calculations
- a parallelogram:** - called **input/output** symbol
  - represent input tasks - like getting info from the user
  - and output tasks – such as displaying info
- arrows:** - called **flowlines**

1. declare variables

#### CH3\_F3 - Variable:

2. convert Text property to number

#### CH3\_F4 - TryParse method:

3. calculate

#### CH3\_F5 - Arithmetic Operators

#### CH3\_F6 - Assigning a Value to an existing Variable:

4. convert and display number to Text property

#### CH3\_F7 - ToString method and it's format specifiers

- because the procedure will need to temporarily store values in the Main Memory of the computer, we will begin with that concept in **CH3\_F2**

## CH3\_F2 - Main Memory of a Computer: basic info & reserving/declaring a memory location: variables & named constants

- random access memory - **RAM**
- when your computer is on, it uses **RAM** to temporarily store the items that are currently being used - so the processor can quickly access them
- in mentioned example items like:
  - **OS Windows**
  - **application programs** (Visual Studio, or or the Circle Area application)
  - **data** (such as the radius entered in the **txtRadius** control, or the area calculated by the application)

- when the computer is turned OFF, the contents of RAM are erased - but but OS files and application program files still reside on your HD

- each **memory** location in **RAM** has a **unique** address
  - like **lockers** in school with different numbers
  - can be different sizes and hold different types of items
  - **but can store** only 1 item at a time !

- some of the locations in **RAM** are **automatically** filled with data while you use your comp.

- e.g.: - when you start the **Circle Area application**, each program instruction is placed in a memory location, where it awaits processing
- similarly, when you enter the number 10 in the **txtRadius** control during run time, the computer stores your entry in a memory location
  - your application can access the contents of the memory location by referring to the control's Text property, like this: **txtRadius.Text**

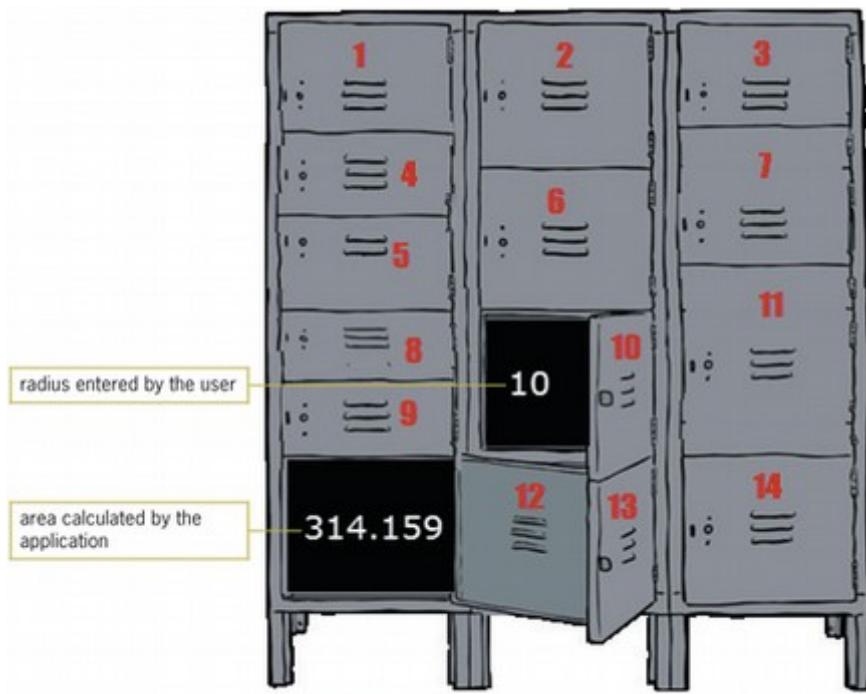


Figure 3-3 Locker illustration

- programmers can also **reserve** memory locations for their own use when coding a procedure -> - more commonly referred to as **declaring a memory location**
- there are **two** types of memory locations that a programmer can declare:

- 1). **Variables:** **Private, Dim, Static, For...Next**
- 2). **Named constants:** **Private Const, Const**

### CH3\_F3 - Variable: declaring a variable memory location for your data: syntax & example

- a variable is a computer memory location, where a programmer can **temporarily** store an item or data while an application is running
  - the memory location is called a variable because its contents can change (vary) during run time
  - examples of data stored in variables:
    - all of the user-provided items
    - result of any calculation made by the application
  - **storing** the data in a **variable** allows the programmer to:
    - **control** the preciseness (přesnost) of the data and
    - **verify** that the data meets certain requirements and
    - **save** the data for later use within the app's code
- code runs more efficiently, because computer can process data **data** stored in **variable** much **faster** than it can process data stored in the **property** of a **control**

- most times you gonna use the **Dim** statement to declare a variable within the procedure that needs it
- **Dim** = dimension (size of something), which is how programmers in the 1960 referred to the process of allocating the computer's memory
- the statement assigns:
  - **a name** - use a reasonable name what you would be able to recognize even after 150 years
  - **a data type** - see **Selecting an Appropriate Data Type**:
  - **an initial value** - if there is no value mentioned, it's automatically 0, false, nothing

e.g.: `Dim intQuantity As Integer` declares a variable whose:

- name is **intQuantity**,
- data type is **Integer**,
- and initial value is the number **0**

- when the computer processes the **Dim statement**, it sets aside a section in its **RAM** and attaches the name **intQuantity** to it
- an instruction within the procedure can then use the **name** to access the memory location to:
  - perhaps to store a different value in it, or
  - use its current value in a calculation, or
  - simply display its value
- 2 data type systems:
  - dynamic: **PHP, RUBY**; automatically chooses a data type, it can be changed anytime
    - smaller code, but can include mistakes therefore crash
  - static: **VB**; once declared data type, it can't be changed into different one
    - bigger code, but better control

### CH3\_F3.1 - Data types for your variables & named constants: my tab with a detailed info

- data type determines the variable's size, which is the amount of memory it consumes
- each data type is a class, which means that each is a pattern from which 1 or more objects - in this case, variables - can be created (instantiated)

| VB alias:              | ID convention + identifier type character (from da book) |     |    |                | Value range:                     | Memory                             | Example / Notes:                                                                                |
|------------------------|----------------------------------------------------------|-----|----|----------------|----------------------------------|------------------------------------|-------------------------------------------------------------------------------------------------|
|                        | forced literal types                                     |     |    |                |                                  |                                    |                                                                                                 |
|                        | wo                                                       | co  | go |                |                                  |                                    |                                                                                                 |
| Integers<br>celá čísla | SByte                                                    |     | -  | -/+ very short | -128 -> +127 (signed = -/+)      | 1 byte                             |                                                                                                 |
|                        | Byte                                                     |     | -  | + very short   | 0 -> 255 (unsigned = only +)     | 1 byte                             |                                                                                                 |
|                        | Short                                                    |     | S  | -/+ short      | -32 768 -> +32 767 (signed= -/+) | 2 bytes                            |                                                                                                 |
|                        | UShort                                                   |     | US | + short        | 0 -> 65 535 (unsigned = only +)  | 2 bytes                            |                                                                                                 |
|                        | Integer                                                  | int | %  | I              | -/+ normal                       | -/+ 2,1 bil (signed = -/+)         | 4 bytes<br><code>Dim intQuantity As Integer = 265</code><br><code>Dim intQuantity% = 265</code> |
|                        | UInteger                                                 |     |    | UI             | + normal                         | 0 -> 4,2 bil (unsigned = only +)   | 4 bytes                                                                                         |
|                        | Long                                                     |     | &  | L              | -/+ long                         | -/+ 9 kurva hodně (signed = -/+)   | 8 bytes                                                                                         |
|                        | ULong                                                    |     |    | UL             | + long                           | 0 -> 18 kurva hodně (unsigned = +) | 8 bytes                                                                                         |

- většinou používá Integers, máš-li však více proměnných v poli (kolekci), pak řešíš paměťový nároky a vybereš ideální

|                             |         |     |   |                |                                   |                              |                                                                                                            |
|-----------------------------|---------|-----|---|----------------|-----------------------------------|------------------------------|------------------------------------------------------------------------------------------------------------|
| Decimals<br>desetinná čísla | Single  | !   | F | -/+ very short | -/+ 7 digits decimal number       | 4 bytes                      | <code>Dim singleDue As Single = 3.14F</code><br><code>Dim singleDue! = 3.14F</code>                        |
|                             | Double  | dbl | # | R              | -/+ long<br>! VB implicit !       | -/+ 15 digits decimal number | 8 bytes<br><code>Dim dblTotalDue As Double = 3.14159</code><br><code>Dim dblTotalDue# = 3.14159</code>     |
|                             | Decimal | dec | @ | D              | -/+ very long<br>financial apps ! | -/+ 29 digits decimal number | 16 bytes<br><code>Dim decSales As Decimal = 256.678999D</code><br><code>Dim decSales@ = 256.678999D</code> |

!!! every decimal number in VB is automatically treated as a Double, therefore you gonna have to use the force, if you desire another type, like Single or Decimal !!!

|        |         |     |                  |                             |                                   |                                                                                                        |
|--------|---------|-----|------------------|-----------------------------|-----------------------------------|--------------------------------------------------------------------------------------------------------|
| Others | Char    | C   | single character | U+0000 -> U+ffff (unsigned) | 2 bytes                           | <code>Dim charA As Char = "A"C</code>                                                                  |
|        | Boolean | bln |                  | logical value               | False / True                      | 2 bytes<br><code>Dim blnIsStoned As Boolean = True</code>                                              |
|        | String  | str | \$               | text                        | 0 -> 2 billion Unicode characters | 32bit=<br>64bit=<br><code>Dim strName As String = "text"</code><br><code>Dim strName\$ = "text"</code> |

- if the **initial value (=)** is omitted (most times), computer stores a **default** value in the variable:
  - numeric data types default value is **0**
  - Boolean default value is **False**
  - String default value is **nothing** (no data at all)
- notes: E=10, E+2=10<sup>2</sup>, E-2=10<sup>-2</sup> ---> 3.56E+2 = 356

### Circle Area Application example: continue 1.

- the **btnCalc\_Click** procedure in the Circle Area application will require 2 variables:
  - one for the **radius** (because it is a user-provided item that will be used in a **calculation**)
  - one for the **area** (because it is a **result** of a calculation made by the application)

#### To begin coding the **btnCalc\_Click** procedure:

- so, I'm gonna choose a **Double** data type for both variables

- for **radius** number entered by the user I'm gonna  
choose a name **dblRadius** and  
and for **area** result I will go with **dblArea**

1). open the Circle Solution.sln

2). in a Code Editor window locate procedure: **btnCalc\_Click**

3). click the blank line immediately above the **End Sub** clause and  
then enter the following 2 Dim statements:

```
Dim dblRadius As Double  
Dim dblArea As Double
```

4). place your mouse on the **green squiggle** to see a warning message

- in this case it shows, that the **dblRadius** and **dblArea** variable has not been used yet
- it will disappear when you include the variable name in another statement within the procedure

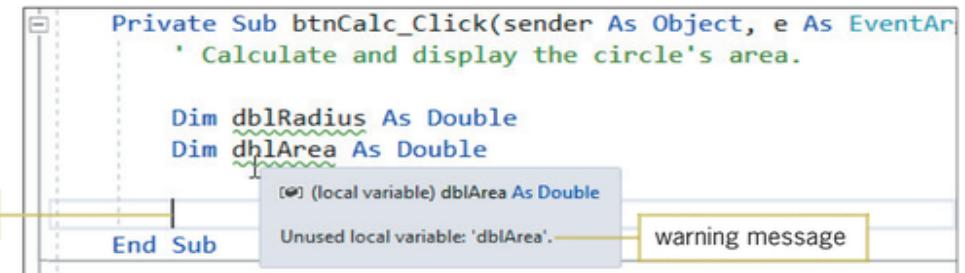


Figure 3-7 Dim statements entered in the **btnCalc\_Click** procedure

### CH3\_F4 - TryParse method: converts string into numeric data type - syntax & example

`numericDataType.TryParse(WhatToConvert, Into)`

- all data entered by the user in **Text box** are stored in the control's **Text** property and are treated as a **String** !!!

**!!! therefore it can't be used as is in a calculation !!!**

- it must be converted to its numeric equivalent (viz Step 2 in **Pseudocode & Flowchart**)
- One way of converting a **String** to a **number** is to use **TryParse** method

- every numeric data type in VB has a **TryParse** method that converts a string to that particular data type

syntax:

```
numericDataType.TryParse(string, numericVariable)
```

- **TryParse** method is a member of the **numericDataType** class
- **( )** = the method's arguments
- **string** argument = what you wanna convert - either the **Text** property of a control or a name of a **String** variable
- **numericVariable** argument = name of a numeric variable where the **TryParse** method can store the number
  - must have the same data type as specified in the **numericDataType** --->
  - > must have been declared the same in its **Dim** statement

e.g.1: `Double.TryParse(txtPaid.Text, dblPaid)` <- example with a **property** of a **Text Box** only

- if the string contained in the `txtPaid.Text` property can be converted into number, it will do so,  
otherwise it stores the number **0** in the variable

e.g.2: `Integer.TryParse(strQuantity, intQuantity)` <- example with already **declared variable** - but the variable must have been the same data type  
as the one in **numericDataType**

- if the string contained in the `strQuantity` variable can be converted into number, it will do so
- otherwise it stores the number **0** in the variable

e.g.3: `Double.TryParse(15.65, dblPay)`

- gonna store 15.65 into variable `dblPay`

e.g.4: `Double.TryParse("ABC", dblPay)`

- will store the number **0**, because ABC can not be converted to a Double number

e.g.5: - how the **TryParse** method converts various strings:

| string:           | "62" | -9 | " 33 " | "12.55" | "-4.23" | "1,457" | "\$5" | "7%" | "122a" | "1 345" | empty space                 |
|-------------------|------|----|--------|---------|---------|---------|-------|------|--------|---------|-----------------------------|
| Double.TryParse:  | 62   | -9 | 33     | 12.55   | -4.23   | 1457    | 0     | 0    | 0      | 0       | 0                           |
| Decimal.TryParse: | 62   | -9 | 33     | 12.55   | -4.23   | 1457    | 0     | 0    | 0      | 0       | 0                           |
| Integer.TryParse: | 62   | -9 | 33     | 0       | 0       | 0       | 0     | 0    | 0      | 0       | 0                           |
|                   |      |    |        |         |         |         |       |      |        |         | ↳ space within-NO           |
|                   |      |    |        |         |         |         |       |      |        |         | ↳ letter-NO                 |
|                   |      |    |        |         |         |         |       |      |        |         | ↳ sign-NO                   |
|                   |      |    |        |         |         |         |       |      |        |         | ↳ comma-ONLY decimals       |
|                   |      |    |        |         |         |         |       |      |        |         | ↳ leading/trailing space-OK |
|                   |      |    |        |         |         |         |       |      |        |         | ↳ negative-OK               |

#### Circle Area Application example: continue 2.

- within the insertion point (under Dim statements) enter the following **TryParse** method:

```
Double.TryParse(txtRadius.Text, dblRadius)
```

- my example code will look like this:

```
Dim dblRadius As Double
Dim dblArea As Double
Double.TryParse(txtRadius.Text, dblRadius)
```

#### CH3\_F5 - Arithmetic Operators (#\$1-6): ^ - \* / \ Mod + -

- you gonna use an **arithmetic** expression, containing arithmetic **operators**

#\$ = precedence = the order in which the computer performs the operation in an expression

| #\$ | operator: | operation:                                                                                                         | example:                                                                      | additional info:                                                                                                                          |
|-----|-----------|--------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------|
| 1   | ^         | exponentiation (umocňování)                                                                                        | $7^2 = 49$<br>$3.14159 * dblRadius^2$                                         | - like $7^2 = 49$<br>- $(dblRadius * dblRadius) * 3.14159$                                                                                |
| 2   | -         | negation (negace) (- hyphen)                                                                                       | $3 = -3$                                                                      | <b>unary</b> operator (jednosložkový) = requires only 1 operand                                                                           |
| 3   | * , /     | multiplication , division                                                                                          | $6 / 2 + 4 = 7$<br>$6 / (2 + 4) = 1$<br>$40 * 2 / 8 = 10$                     | - $6 / 2 = 3$ and $3 + 4 = 7$<br>- operations within <b>parentheses</b> performed <b>first</b><br>the order comes from left               |
| 4   | \         | integer division without decimal<br>(celočíselné dělení beze zbytku)                                               | $11 \ 3 = 3$                                                                  | - divides two only <b>integers</b> and returns the result also as <b>integer</b><br>- $11 / 3 = 3.67$                                     |
| 5   | Mod       | modulus = remainder (zbytek po dělení)<br>commonly used to determine if<br>the number is <b>even</b> or <b>odd</b> | $26 \text{ Mod } 5 = 1$<br>$28 \text{ Mod } 2 = 0$<br>$29 \text{ Mod } 2 = 1$ | - divides two <b>numbers</b> (can be decimals), returns the remainder<br>- an <b>even</b> number<br>- an odd number (nederlands - uneven) |
| 6   | + , -     | addition , subtraction (- hyphen)                                                                                  | decSales - decDiscount<br>$40 - 30 + 10 = 20$                                 | <b>binary</b> operators (dvousložkový) = requires 2 operands<br>the order comes from left                                                 |

- **can't** contain commas or special characters like \$, %, ... (Veerle: 1,500 vs 1500 moje)..., dblTotal\*15% vs dblTotal \* 0.15

- **can** contain **numeric variables** (dblSalary \* 0.04)

### CH3\_F6 - Assigning a Value to an existing Variable:

- the expression's data type should be the same as the variable's data type

syntax: `variable = expression`      - numbers, string literals, object properties, variables, keywords, arithmetic operators

e.g.:

- `intYear = 2019`
- `strCity = "Nymburk"`
- `strState = txtState.Text`      -> the string contained in the `txtState` control's `Text` property **assigns** to the `strState` variable
- `dblNewPay = dblCurrentPay * 1.1`      -> in VB a number with a decimal place is automatically treated as a **Double**
- `dblRate = 0.25`      -> converts the **Double** number 0.03 to **Decimal** and then assigns the result to the `decRate` variable
- `decRate = 0.03D`

**D** = literal type character, what forces to assume (nabýt) a data type other then the one its form indicates

#### Circle Area Application example: continue 3.

- you are now ready to code Step 3 in the `btnCalc_Click` procedure's pseudocode:

- in the blank line below the:

`Double.TryParse(txtRadius.Text, dblRadius)`

- enter the following assignment statement:

`dblArea = 3.14159 * dblRadius ^ 2`

## CH3\_F7 - ToString method and its format specifiers C or F or N or P: converts numbers into strings - syntax & e.g.

### Circle Area Application example: continue 4.

- the last step (number 4.) is the **btnCalc\_Click** procedure's pseudocode displays the circle's area in the **lblArea** control

- you do this using an assignment statement that assigns the number stored in the **dblArea** variable to the **lblArea.Text** property - **BUT !**
- **BUT** you first **must** convert the number **back to a string** - because the **Text** property stores only strings !!!
- you can use the **ToString** method for this purpose - all of the numeric data types have this method
- this method also allows you to format the value
  - = specify the number of decimal places
  - = can display the special characters such as £, \$, %, €, thousands separator (Veerle)...etc
- **ToString** method formats a copy of the value stored in the numeric variable and then returns the result as a string

syntax:

```
numericVariable.ToString("CFNPxx")
```

( ) - optional

**CFNP** = format specifier

- single alphabetic character, isn't case sensitive -> **c, f, n, p**

**xx** = optional precision specifier

- **0 -> 99**

- number of digits in the result, does not round the number

note: **ToString("C2")** <- reads Currency settings from your Computer !!! <- my default has been **XDR**, so it showed **XDR** sign

e.g.: - **lblTotal.Text = dblTotal.ToString**

- **lblAge.Text = intAge.ToString("N2")**

- **lblPay.Text = lblPay.Text & intTerm.ToString & " years: " & dblPay.ToString("C2") & ControlChars.NewLine**

| wocotigo    | format specifier | precision specifier              | result                                                                           | example                                                                                                                  |
|-------------|------------------|----------------------------------|----------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------|
| Currency    | <b>C</b>         | number of decimal digits         | a currency value<br>!!! it reads the <b>sign</b> from your computer settings !!! | - <b>intSales = 75000</b><br><b>lblSalex.Text = intSales.ToString("C2")</b><br><b>&gt; result string : "\$75,000.00"</b> |
| Fixed-point | <b>F</b>         | number of decimal digits         | integer & decimal with opt. negative sign                                        |                                                                                                                          |
| Number      | <b>N</b>         | desired number of decimal places | integer & decimal, group & decimal separator with opt. negative sign             | - <b>decTotal = 4599.639D</b><br><b>lblTotal.Text = decTotal.ToString(N2)</b><br><b>&gt; result string : "4,599.64"</b>  |
| Percent     | <b>P</b>         | desired number of decimal places | number * 100 and displayed with % symbol                                         | - <b>dblRate = 0.15</b><br><b>lblRate.Text = dblRate.ToString("P0")</b><br><b>&gt; result string : "15 %"</b>            |

### Circle Area Application example: continue 5.

- below the calculation statement, enter the following assignment statement:

```
lblArea.Text = dblArea.ToString("N2")
```

## CH3\_F8 - Option statement - above all in a code - including explanation: Option Explicit & Option Strict & Option Infer

- if a project contains more than one form, the 3 mentioned **Option** statements should be entered in **each form's** Code Editor window

- the best is to enter it in **Code Editor**, rather than in **Designer window** or **Options dialog box** ---> more easy to control it

- you enter the statement **above** the **Public Class** clause in the Code Editor window:

**Option Explicit On** = flag any undeclared variables

- a variable's declaration statement is important because it allows you to control the variable's data type, to ensure that all of the variables in your code have been declared, VB provides a statement that tells the Code Editor to **flag any undeclared** variables

**Option Strict On** = eliminates **implicit type conversion** problems and the computer uses the rules listed below

- **implicit type conversion** = automatic data type conversion when mistake is made

- you learned that the data type of the value assigned to a memory location should be the same as the data type of the memory location itself

- but if the value's data type does **not** match, the computer uses a process called **implicit type conversion** to fit the memory location

- for example: **Dim dblLength As Double = 9** - the computer converts **integer** 9 to the **Double** 9.0 before storing

- when a value is converted into larger number or nr. with greater precision - the value is said to be **promoted** (Integer to Double.....)

- if you inadvertently (neúmyslně) assign a **Double** to a memory that can store only **Integer**, it rounds the number and bin the rest - the value is said to be **demoted**

- **demotion** could lead to the incorrect calculations !

the rules listed:

### 1. Strings will not be implicitly converted to numbers

incorrect: **dblRadius = txtRadius.Text**

solution: **Double.TryParse(txtRadius.Text, dblRadius)**

<- the Code Editor will issue the warning message like  
"Option Strict On disallows implicit conversions from 'String' to 'Double'"

### 2. Numbers will not be implicitly converted to strings

incorrect: **lblArea.Text = dblArea**

solution: **lblArea.Text = dblArea.ToString("N2")**

<- the Code Editor will issue an appropriate warning message

### 3. Wider data types will not be implicitly demoted to narrower data type

incorrect: **decRate = 0.05**

solution: **decRate = 0.05D**

<- the Code Editor will issue an appropriate warning message

### 4. Narrower data types will be implicitly promoted to wider data type

correct: **dblAverage = dblTotal / intNum**

- computer will implicitly promote the **Integer** stored in the **intNum** variable to **Double** before dividing it into contents of the **dblTotal** variable  
- the result: a **Double** number will be assigned to the **dblAverage** variable

- also handy:

**Option Infer OFF**

= every variable is declared with a data type

= the statement tells the computer not to infer (or assume) a memory location's data type based on the data assigned to it

### Circle Area Application example: continue 6.

- above the Public class enter: **Option Explicit On**

**Option Strict On**

**Option Infer Off**

### CH3\_F9 - Named Constant declaring a constant memory location for your data: syntax & example

- in addition to declaring **variables**, you can also declare **named** constants
- like a variable is a memory location inside the computer
  - BUT it cannot be changed while the app is running**
- make code more self-documenting and easier to modify - because they allow you to use meaningful words in place of values that are less clear
- e.g.: **dblPI** is more meaningful than the number 3.14159
- once you create a named constant, you then can use its **meaningful name** - instead of its less clear value
- another advantage: if the value changes in the future, you will need to modify only the **named constant's declaration statement !!!**
  - rather than all of the program statements that use the value

syntax: **Const constantName As dataType = expression**

e.g. 1: **Const dblPI As Double = 3.14159**  
e.g. 2: **Const intMAX\_SPEED As Integer = 172**  
e.g. 3: **Const strTITLE As String = "Biggus Dikkus"**  
e.g. 4: **Const decRATE As Decimal = 0.05D**

<- don't forget D - change implicit Double to Decimal !

#### Circle Area Application example: continue 7. (Figure 3-18)

- to use a named constant:
  1. above the first **Dim** statement type: **Const dblPI As Double = 3.14159**
  2. in the **calculation** change **3.14159** to **dblPI**
  3. save and test, enter the value **5,5** and the result should be **95,03**

```
1 ' Name:           Circle Project
2 ' Purpose:        Calculate and display the circle's area.
3 ' Programmer:     <your name> on <current date>
4
5 Option Explicit On
6 Option Strict On
7 Option Infer Off
8
9 Public Class frmMain
10    Private Sub btnCalc_Click(sender As Object, e As EventArgs)
11        ' Calculate and display the circle's area.
12
13        Const dblPI As Double = 3.14159
14        Dim dblRadius As Double
15        Dim dblArea As Double
16
17        Double.TryParse(txtRadius.Text, dblRadius)
18        dblArea = dblPI * dblRadius ^ 2
19        lblArea.Text = dblArea.ToString("N2")
20
21    End Sub
22
23    Private Sub btnExit_Click(sender As Object, e As EventArgs)
24        Me.Close()
25    End Sub
26 End Class
```

Figure 3-18 Circle Area application's code

### homework p.93 - You Do It 1:

- create an app. Add a text box, a label, and a button.
- the button's Click event procedure should store the contents of the text box in a Double variable named **dblCost**
- It then should display the variable's contents in the label
- enter the 3 Option statements.

```
Option Explicit On
Option Strict On
Option Infer Off
Public Class frmMain
    Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
        Me.Close()
    End Sub

    Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
        Dim dblCost As Double
        Double.TryParse(txtCost.Text, dblCost)
        lblCost.Text = dblCost.ToString("N2")
    End Sub
End Class
```

## CH3\_APPLY THE CONCEPTS LESSON:

### CH3\_A1 - determine a memory location's **Scope** and **Lifetime**: Procedure-level vs Class-level

- every variable (**Dim**) and named constant (**Const**) has a:
  - name
  - data type
  - initial value
- but also:
  - **scope** = where the declared memory location can be used in an code (sféra platnosti)
  - **lifetime** = how long **Dim / Const** remains in RAM
  - **namespace scope** <- can lead to errors and should be avoided if possible - not covered in the book
- the scope and lifetime are **determined** by where you declare the memory location in your code
- memory locations declared in:
  - **procedure**
    - used mostly !!! (I think for considering RAM consumption)
    - have procedure scope
    - called: **procedure-level variables** = local variables with local scope  
= procedure scope
    - procedure-level named constants** = local named constant
    - can be used only:
      1. if you declare it first and then
      2. within the procedure, where you declared it
    - removed from RAM when the procedure ends
    - e.g. in Circle Area app: - declared in the **btnCalc\_Click** procedure: - **dblRadius** and **dblArea** variables  
- **dblPI** named constant
  - **form class's declaration section**
    - (but outside of any procedures)
    - = the area between the form's **Public Class** and **End Class** clauses (line 9-26)
    - = typically appear immediately after the **Public Class frmMain** (after the line 9)
    - can be used by all of the procedures
    - same lifetime as the application = they remain in the RAM until the app ends
    - have class scope
    - called: **class-level variables** **(Private)**  
**class-level named constants** **(Private Const)**
    - like **btnExit**

### CH3\_A2 - use a Procedure-Level Variables: the Dim statement

- most of the variables in your codes (fewer unintentional errors occur in app when the variables are declared using the minimum scope needed)
- typically declared at the beginning of a procedure

#### example : *The Commission Calculator application:*

- illustrates the use of **procedure-level** variables
- as the GUI in **Figure 3-19** indicates, the app displays the amount of a salesperson's commission
- the commission is calculated by multiplying the salesperson's sales by the appropriate commission rate: either 8% or 10% (`* 0.08D` ; `* 0.1D`)
- **Figure 3-20** shows the Click event procedures for the 8% rate and 10% rate buttons
- the comments in the figure indicate the purpose of each line of code
- when each procedure ends, its **procedure-level variables** are removed from RAM
- **the variables will be created again the next time the user clicks the button**
- notice that both procedures declare a variable named `decSales`
  - = each procedure creates its own variable when invoked
  - = each procedure also destroys its own variable when the procedure ends

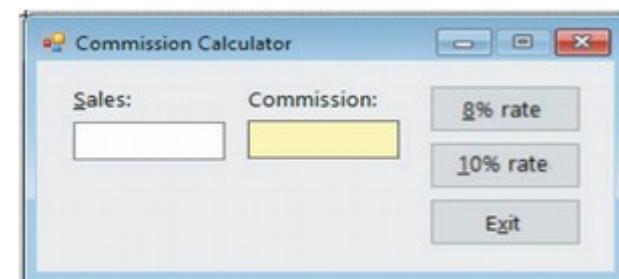


Figure 3-19 User interface for the Commission Calculator application

```
14 Private Sub btnRate10_Click(sender As Object, e As EventArgs)
15     ' Calculates and displays a 10% commission.
16
17     ' Declare two procedure-level variables that can
18     ' be used only within the btnRate10_Click procedure.
19     Dim decSales As Decimal
20     Dim decComm10 As Decimal
21
22     ' Convert the txtSales.Text property to Decimal. Store
23     ' the result in the procedure-level decSales variable.
24     Decimal.TryParse(txtSales.Text, decSales)
25
26     ' Multiply the value in the procedure-level decSales
27     ' variable by the Decimal number 0.1D. Assign the
28     ' result to the procedure-level decComm10 variable.
29     decComm10 = decSales * 0.1D
30
31     ' Convert the value in the procedure-level decComm10
32     ' variable to String. Assign the result to the
33     ' lblCommission.Text property.
34     lblCommission.Text = decComm10.ToString("C2")
35
36 End Sub
```

```
38 Private Sub btnRate8_Click(sender As Object, e As EventArgs)
39     ' Calculates and displays an 8% commission.
40
41     ' Declare two procedure-level variables that can
42     ' be used only within the btnRate8_Click procedure.
43     Dim decSales As Decimal
44     Dim decComm8 As Decimal
45
46     ' Convert the txtSales.Text property to Decimal. Store
47     ' the result in the procedure-level decSales variable.
48     Decimal.TryParse(txtSales.Text, decSales)
49
50     ' Multiply the value in the procedure-level decSales
51     ' variable by the Decimal number 0.08D. Assign the
52     ' result to the procedure-level decComm8 variable.
53     decComm8 = decSales * 0.08D
54
55     ' Convert the value in the procedure-level decComm8
56     ' variable to String. Assign the result to the
57     ' lblCommission.Text property.
58     lblCommission.Text = decComm8.ToString("C2")
59
60 End Sub
```

Figure 3-20 Click event procedures using procedure-level variables

homework: p.96-97

- 1.) open the **Commission Solution.sln** and code the **Click** event procedures
- 2.) test: enter 40000, try 8% and result should be: \$3,200.00
- 3.) test: enter letter a, try 10% and result should be \$0.00
- 4.) test: enter 25000, try 10% and result should be \$2,500.00
- 5.) test: enter \$900, try 10% and result should be \$0.0

### CH3\_A3 - use a Procedure-Level Named Constants: the **Const** statement

- are also declared at the beginning of a procedure
- also can be used only after their declaration
- also same lifetime as the procedure in which they are declared

homework: p.98

- 1.) open the Commission Solution.sln
- 2.) locate the **btnRate10\_Click** procedure and make the modifications shaded in **Figure 3-22**
- 3.) locate the **btnRate8\_Click** procedure and make the modifications shaded in **Figure 3-23**

```
17     ' Declare a procedure-level named constant.  
18     Const decRATE10 As Decimal = 0.1D  
19     ' Declare two procedure-level variables that can  
20     ' be used only within the btnRate10_Click procedure.  
21     Dim decSales As Decimal  
22     Dim decComm10 As Decimal  
23  
24     ' Convert the txtSales.Text property to Decimal. Store  
25     ' the result in the procedure-level decSales variable.  
26     Decimal.TryParse(txtSales.Text, decSales)  
27  
28     ' Multiply the value in the procedure-level decSales  
29     ' variable by the Decimal named constant decRATE10. Assign the  
30     ' result to the procedure-level decComm10 variable.  
31     decComm10 = decSales * decRATE10
```

Figure 3-22 btnRate10\_Click procedure using a procedure-level named constant

```
42     ' Declare a procedure-level named constant.  
43     Const decRATE8 As Decimal = 0.08D  
44     ' Declare two procedure-level variables that can  
45     ' be used only within the btnRate8_Click procedure.  
46     Dim decSales As Decimal  
47     Dim decComm8 As Decimal  
48  
49     ' Convert the txtSales.Text property to Decimal. Store  
50     ' the result in the procedure-level decSales variable.  
51     Decimal.TryParse(txtSales.Text, decSales)  
52  
53     ' Multiply the value in the procedure-level decSales  
54     ' variable by the Decimal named constant decRATE8. Assign the  
55     ' result to the procedure-level decComm8 variable.  
56     decComm8 = decSales * decRATE8
```

Figure 3-23 btnRate8\_Click procedure using a procedure-level named constant

**CH3\_A4 - use a Class-Level Variable: the **Private** statement****Private** variableName **As** dataType = expression

- declared immediately after the **Public Class** clause, **outside** of any procedure
- declared using **Private** keyword instead of **Dim**, what could be used as well, but **Private** is recommended to make it more obvious to anyone reading the code
- retain (ponechat) their values and remain in the computer's RAM until the app ends
- can be accessed by any of the procedures entered in the window ->
  - > as a result, it can lead to unexpected results when one of the procedures makes an inadvertent (neúmyslný) or incorrect change to the variable's value
- tracking down the errors in a app's code becomes more complicated as the number of procedures having access to the same variable increases
  - therefore, the use of class-level variables should be minimized !
- rather than using this one to accumulate values, you can use a **Static variable**: **CH3\_A5 - Use a Static Variable: the **Static** statement**

**syntax:****Private** variableName **As** dataType = expression

usually used when: - 2 or more procedures in the same form need access to the same variable (you can observe this in **CH5\_F6 - Counters and Accumulators**)  
- a procedure needs to retain a variable's value even after the procedure ends

**e.g.1:** **Private** intQuantity **As Integer****e.g.2:** **Total Scores Accumulator application:** - when the user starts the application:

1. the computer processes the **Private dblTotal As Double** = the statement creates and initializes (to 0) the class-level **dblTotal** variable
  - the variable is created and initialized only once, when the app starts
  - remains in RAM until the app ends
2. each time the user clicks the **Add to total** button,  
the **Dim** statement on Line 17 in the **btnAdd\_Click** procedure creates and initializes a procedure-level variable named **dblScore**
3. the **TryParse** method on Line 20 converts the **txtScore.Text** property do **Double** and stores the result in the **dblScore** variable
4. the assignment statement on Line 21 adds the contents of the procedure-level **dblScore** variable to the content of the class-level **dblTotal** variable
  - at this point, the **dblTotal** contains the sum of all of the scores entered so far
5. the assignment statement on Line 22 converts the **dblTotal** value to String and then assigns the result to the **lblTotal.Text** property
6. when the procedure ends, the computer removes the procedure-level **dblScore** variable from RAM
  - however, it does not remove the class-level **dblTotal** variable; its removed only when the app ends

```
9  Public Class frmMain
10   ' Class-level variable for accumulating scores.
11   Private dblTotal As Double
12
13   Private Sub btnAdd_Click(sender As Object, e As EventArgs)
14     ' Accumulates and displays the scores.
15
16     ' Procedure-level variable for storing a score.
17     Dim dblScore As Double
18
19     ' Accumulate the scores and display the results.
20     Double.TryParse(txtScore.Text, dblScore)
21     dblTotal = dblTotal + dblScore
22     lblTotal.Text = dblTotal.ToString
23   End Sub
```

class-level variable declared in the  
form class's declarations section

procedure-level variable declared  
in the **btnAdd\_Click** procedure

**Figure 3-26** Most of the application's code using a class-level variable

homework p.100:

1. open the **Total Scores Solution-Class-level.sln**
2. declare the class-level **dblTotal** - enter the following lines below **Public Class** clauses:  
`' Class-level variable for accumulating scores.  
Private dblTotal As Double`
3. locate the code template for the **btnAdd\_Click** procedure and enter the **Dim** statement, **TryParse** method, and **2** assignment statements shown in **Figure 3-26** (lines 17, 20, 21, 22)
4. test: - type **89** and click the button, **same** number should appear in the box  
- type **75** and click the button, then number **164** should appear.

### CH3\_A5 - use a Procedure-Level Variable with a Class-Level scope: the **Static** statement

- it is a **procedure-level** variable, that remains in RAM and also retains its value even when its declaring procedure ends like **class-level** variable
- but can be used **only** by the procedure in which it is declared !
- has a narrower, **more restrictive scope = less unintentional errors**

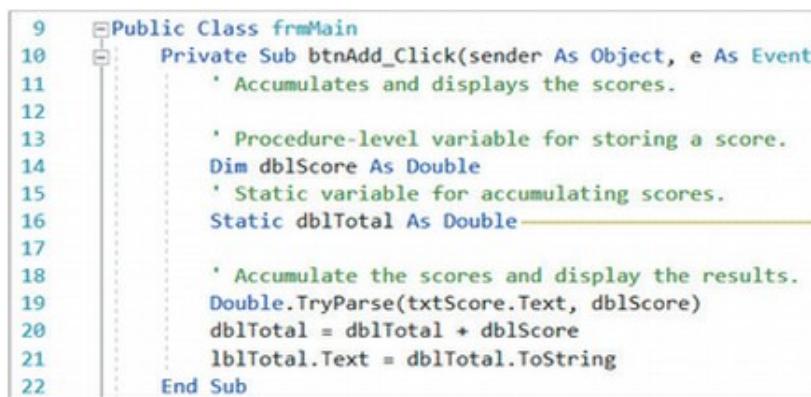
syntax: `Static variableName As dataType = expression`

e.g.1: `Static dblTotal As Double`

e.g.2: `Static intCount As Integer = 1`

e.g.3: **Figure 3-29** - the Total Scores Accumulator application used a **class-level** variable to accumulate the scores

- rather than using a **class-level** variable for that purpose,  
you can also use a **procedure-level static variable**,  
who has a narrower scope,  
therefore lower possibility for an error



```
9  Public Class frmMain
10 Private Sub btnAdd_Click(sender As Object, e As EventArgs)
11     ' Accumulates and displays the scores.
12
13     ' Procedure-level variable for storing a score.
14     Dim dblScore As Double
15     ' Static variable for accumulating scores.
16     Static dblTotal As Double
17
18     ' Accumulate the scores and display the results.
19     Double.TryParse(txtScore.Text, dblScore)
20     dblTotal = dblTotal + dblScore
21     lblTotal.Text = dblTotal.ToString()
22 End Sub
```

**Figure 3-29** Most of the application's code using a static variable

homework p.102:

1. open the **Total Scores Solution-Static.sln**
2. delete the comment and the Private declaration statement entered in the form class's declarations section
3. locate the **btnAdd\_Click** procedure and then click the blank line below the **Dim** statement and enter 2 lines below:  
`' Static variable for accumulating scores.  
Static dblTotal As Double`
4. test: use the following three scores: **89, 75, 100**. the result should be **264**.

### CH3\_A6 - use a Class-Level Named Constant: the **Private Const** statement

**Private Const dblPI As Double = 3.14159**

- earlier you learned, that the use of **class-level variables** should be minimized because they can lead to errors that are difficult to locate in your code
- **class-level named constants** do not have the same problem - simply because their values cannot be changed during run time
- declared immediately after the **Public Class** clause, outside of any procedure

syntax:

```
Private Const constantName As dataType = expression
```

e.g.1: **Private Const dblPI As Double = 3.14159**

e.g.2: **Private Const strCOMPANY As STRING = "Brett bvba."**

e.g.3: **Private Const decMinPay As Decimal = 15.65D**

- the **Circle Area** app that you coded in **Focus** lesson declared a **procedure-level named constant** to represent the value of PI ->

<- for a homework you gonna change the **named constant** to a **class-level** one:

homework p.103: to use a class-level named constant in the **Circle Area** app:

1. open the **Circle Solution-Class-level.sln**
2. locate the **btnCalc\_Click** procedure
3. highlight (select) the entire **Const** statement and -> Ctrl+x
4. below the '**Class-level named constant**' comment -> Ctrl+v
5. insert the keyword **Private** at the beginning of the declaration statement as shown in **Figure 3-31**:
6. test: type **15** in the Radius box, the result should be: **706.86**

```
9  Public Class frmMain
10   ' Class-level named constant.
11   Private Const dblPI As Double = 3.14159
12
13   Private Sub btnCalc_Click(sender As Object, e As EventArgs)
14     ' Calculate and display the circle's area.
15
16   Dim dblRadius As Double
17   Dim dblArea As Double
18
19   Double.TryParse(txtRadius.Text, dblRadius)
20   dblArea = dblPI * dblRadius ^ 2
21   lblArea.Text = dblArea.ToString("N2")
22
23 End Sub
```

Figure 3-31 Class-level named constant entered in the form class's declarations section

### mini quiz 3-8: Write a statement that declares:

1. a **procedure-level named constant** whose value and name are **10.5** and **decPAY**, respectively.
2. a **class-level variable** named **strName**
3. a **procedure-level variable** named **dblNumber**
4. a **procedure-level variable** named **dblSum**. The variable should retain its value until the app ends.

1...Const decPAY As Decimal = 10.5D  
2...Private strName As String  
3...Dim dblNumber As Double  
4...Static dblSum As Double

## CH3\_A7 - Professionalize Your Application's Interface: event procedures `txt_TextChanged` and `txt_Enter`

- you can make your GUI more proffy-looking by coding each Text Box's event procedures:
  - 1. **TextChanged** = clear the previous values on **Label**
  - 2. **Enter** = highlight the previous value on **Text Box**

### 1. coding the **TextChanged** event procedure:

e.g.1:

1. open the folder **Circle Solution-TextChanged and Enter**, start the **Circle Solution.sln**
2. test it and type **2**, then **Calculate**. The result gonna show **12.57**
3. test it and type **25** - notice that the result is still the old one, until you **recalculate**
  - having the previously calculated items on the screen could be misleading to the user -
  - therefore a better approach is to **clear the content** by coding the text box's **TextChange** event procedure
  - a control's **TextChanged** event occurs each time the value in its **Text** property changes

e.g.2:

- you gonna code the `txtRadius_TextChanged` procedure, so that it clears the contents of the **Area** box when the event occurs
- you can clear the contents of a control by assignint the value `String.Empty` to its Text property

1. open the code editor window from previous example and enter the lines following **Figure 3-32**:

```
27 Private Sub txtRadius_TextChanged(sender As Object)
28     ' Clear the Area box.
29
30     lblArea.Text = String.Empty
31
32     End Sub
33 End Class
```

A screenshot of a code editor showing a Visual Basic .NET code snippet. The code defines a class with a single event handler named `txtRadius_TextChanged`. Inside the handler, there is a comment `' Clear the Area box.` followed by an assignment statement `lblArea.Text = String.Empty`. A yellow callout box points to this assignment statement with the text "enter this comment and assignment statement".

Figure 3-32 `txtRadius_TextChanged` event procedure

2. test:
  - type **2** and click **Calculate**. The result is **12.57**.
  - type **25** - you see that the `txtRadius_TextChanged` procedure clears the contents of the Area box.
3. now press the **Tab** key **twice** to send the focus to the **Radius** box. Notice that the insertion point appears at the end of the number 25. ->  
<- but that is not ok. Use the **Enter** event procedure.

### 2. coding the **Enter** event procedure:

- customary (zvyk) in Win apps - when the text box with old text receives the focus, the existing text is highlighted ---> **Enter** Event Procedure
- a text box's **Enter** event occurs when the text box receives the focus
- you gonna code the: `txtRadius_Enter` event procedure so that it selects (highlights) the contents
- you can select the existing text using the **SelectAll** method

- e.g.:
  1. open the code editor window from previous example and enter the lines following **Figure 3-33**:
  2. test it and you will see the change

```
26
27 Private Sub txtRadius_Enter(sender As Object, e As EventArgs)
28     ' Select the existing text.
29
30     txtRadius.SelectAll()
31
32     End Sub
```

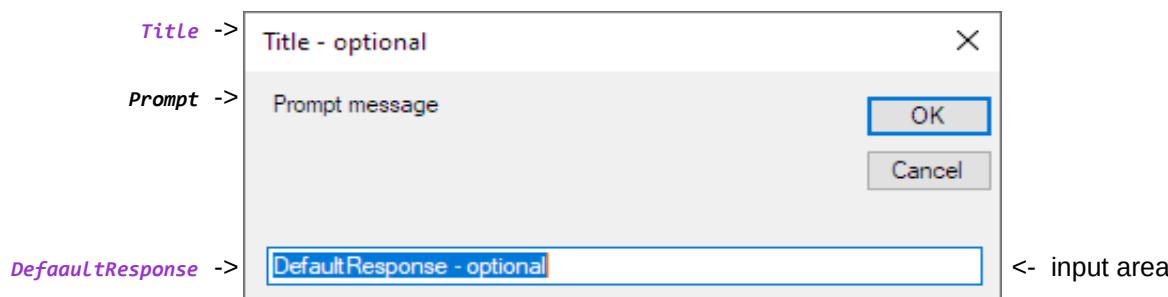
A screenshot of a code editor showing a Visual Basic .NET code snippet. The code defines a class with a single event handler named `txtRadius_Enter`. Inside the handler, there is a comment `' Select the existing text.` followed by a call to the `SelectAll()` method of the `txtRadius` control. A yellow callout box points to this call with the text "enter this comment and SelectAll method".

Figure 3-33 `txtRadius_Enter` event procedure

### CH3\_A8 - InputBox function - additional topic from Appendix B - interact with user while app is running - syntax & example

- the **InputBox** function
  - displays an input dialog box, which is one of the standard dialog boxes available in Visual Basic
  - allows an application to interact with a user while an application is running

e.g.



- the message in the dialog box should prompt the user to enter the appropriate information in the input area
- the value returned by the **InputBox** function depends on the button the user chooses:

- a). **OK** button
  - the function returns the value contained in the input area of the dialog box
  - always treated as a **String**
- b). either **Cancel** or **Close** button
  - the function returns an empty string / zero-length

InputBox Function Interaction syntax:  
As String

**InputBox(Prompt As String, Title As String, DefaultResponse As String, XPos As Integer, YPos As Integer)**

**Prompt** - **As String**  
= an argument; contains the message to display inside the dialog box

**Title** - **optional**  
**As String**  
= an argument; contains the text that appears in the dialog box's title bar  
- if omitted, the project name will be used

**DefaultResponse**  
**As String**  
= an argument; contains the text that appears in the dialog box's input area  
- if omitted, a blank input area appears when the dialog box opens

**XPos** - **optional**  
**As Integer**  
= an argument; specify the dialog box's horizontal position

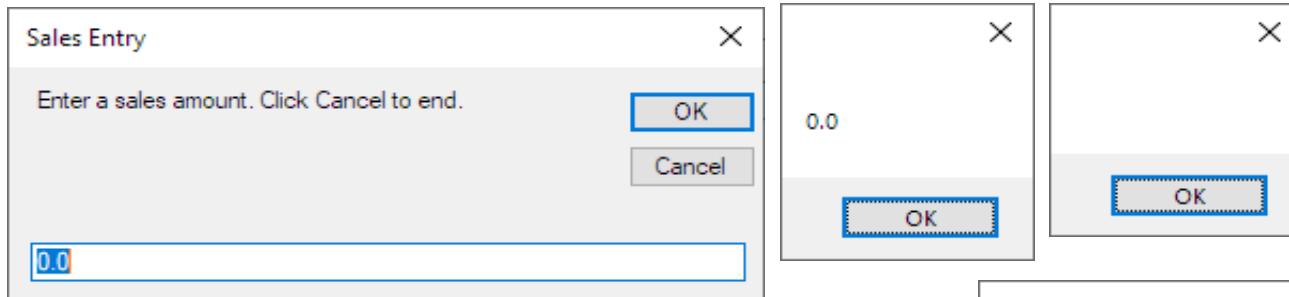
**YPos** - **optional**  
**As Integer**  
= an argument; specify the dialog box's vertical position  
- if both omitted, the dialog box appears centered on the screen = the default values -1 will be used

e.g. 1

```
Dim strSales As String  
strSales = InputBox("Enter a sales amount. Click Cancel to end.", "Sales Entry", "0.0")  
MessageBox.Show(strSales)
```

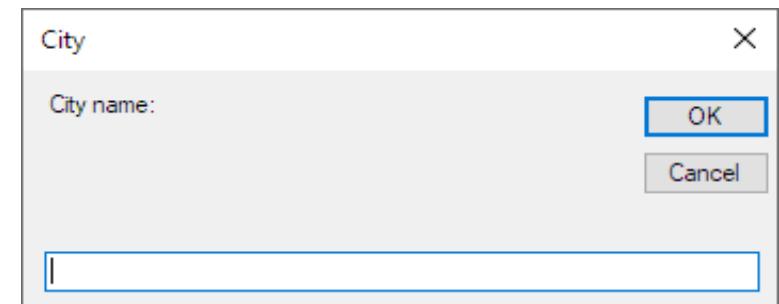
<- when the user closes the dialog box, the assignment statement assigns the function's return value to the **strSales** variable:

- a). if the user closes by **OK** button, the string value in the input area will be assigned to the **strSales** variable
- b). if the user closes by **Cancel**, or **Exit** buttons, an empty string will be assigned to the **strSales** variable



e.g. 2

```
Dim strCity As String  
strCity = InputBox("City name:", "City")  
MessageBox.Show(strCity)
```



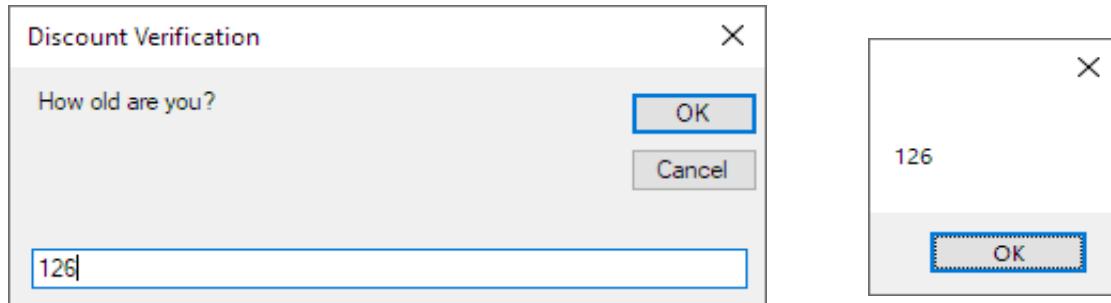
e.g. 3

```
Dim strRate As String  
Const strPROMPT As String = "Enter the discount rate:"  
Const strTITLE As String = "Discount Rate"  
strRate = InputBox(strPROMPT, strTITLE, ".00")  
MessageBox.Show(strRate)
```



e.g. 4

```
Dim intAge As Integer  
Integer.TryParse(InputBox("How old are you?", "Discount Verification"), intAge)  
MessageBox.Show(intAge.ToString())
```



- to use an **InputBox** function - an input dialog box:

1). open the: ...VB2017\Chap03\Exercise\InputBox Solution\InputBox Solution.sln

2). open the **Solution Explorer** window and **Designer** window

3). open the **Code Editor** window, and:

-> open the code template for the **btnGetName\_Click** procedure:

<- the lines of code appears:

```
10     Private Sub btnGetName_Click(sender As Object, e As EventArgs) Handles btnGetName.Click  
11  
12 End Sub
```

-> type the following comment and code:

```
10     Private Sub btnGetName_Click(sender As Object, e As EventArgs) Handles btnGetName.Click  
11         ' Get and display a name.  
12         Dim strName As String  
13         strName = InputBox("Enter your name:", "InputBox Example")  
14         lblName.Text = strName  
15     End Sub
```

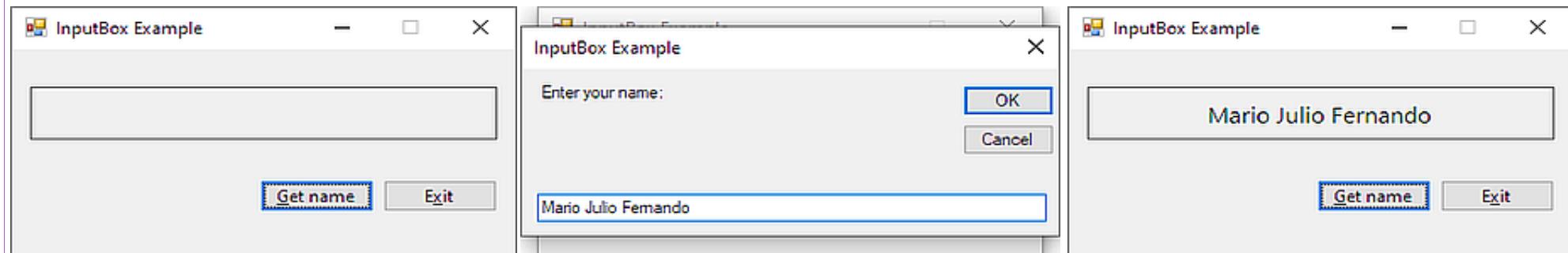
4). save the solution and start and test the application

-> click the **Get name** button

<- the input dialog box opens

-> type your name in the input area of the dialog box and then click the **OK** button

<- your name appears in the **label** control on the form



5). entire code:

```
1  Option Explicit On  
2  Option Strict On  
3  Option Infer Off  
4  
5  Public Class frmMain  
6      Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click  
7          Me.Close()  
8      End Sub
```

```

9
10     Private Sub btnGetName_Click(sender As Object, e As EventArgs) Handles btnGetName.Click
11         ' Get and display a name.
12         Dim strName As String
13         strName = InputBox("Enter your name:", "InputBox Example")
14         lblName.Text = strName
15     End Sub
16 End Class

```

### CH3\_Summary:

- use planning tool as **flowcharts** or **pseudocode**, it makes the process more easy
- variables and named constants are memory locations used to store data
- user-provided items included in a calculation should be stored in **variables**
- **calculations** made by the app should also be stored in **variables**
- in **most** procedures, the **Dim** statement is used to declare a variable
- each **data type** is a Class from which a **variable** can be instantiated
- the value stored in a control's **Text property** is always treated as a **string**
- you can use the **TryParse** method to convert a string to a number
- the arithmetic operators have an order of precedence, but you can use **parentheses** to override the order
- the **data type** of the expression assigned to a **variable** should match the variable's **data type**
- you can use the literal type character **D** to convert a **default Double** number to the **Decimal** data type
- you can use the **ToString** method to convert a numeric value to a string.

The method also allows you to specify the number of decimal places and the special characters to include in the string

- you use the **Const** statement to declare a named constant
- a memory location's scope indicates where the variable or named constant can be used in an app's code
- a memory location's lifetime indicates how long the variable or named constant remains in the RAM
- most of the declarations are done in procedure-level scope
- you should **limit** the use of **class-level variables** to lower the potential errors
- you can use the keyword **Static** to declare a **procedure-level** variable that retains its value even after the declaring procedure ends

- you can set the **Three Options** by 3 different ways:

#### 1. only for current frm = in your code:

- to ensure that all of your variables have been declared, enter the **Option Explicit On** statement above the Public Class clause
- to prevent the computer from making implicit type conversions that may result in a loss of data, enter the **Option Strict On** statement
- to prevent the computer from interfering (motat se do) a variable's data type, enter the **Option Infer Off**

#### 2. for entire Project = in Project Designer window:

- open the Solution that contains the Project. In the Solution Explorer right-click on **My Project** and choose **Open** to open the Project Designer window.
- Click the **Compile** tab and set the **Options**.

#### 3. for all of the Projects you create = in Option dialog box:

- on the menu bar click Tools and Options. When the Options dialog box opens, expand the Projects and Solutions node and click VB Defaults.
- You can set the **Options**.

- a control's **TextChanged** event occurs each time the value in its **Text** property changes (used in **Label** what shows the result of calculation)
- a control's **Enter** event occurs when the control receives the focus (like via Tab key) (used in **Text Box** when entering new data)
  - you can use the **SelectAll** method to select the text contained in a text box

### CH3\_Key Terms:

- **Arguments** - items within parentheses ( ) after the name of a method; represents info that the method needs to perform its task(s)
- **Class scope** - variable's or better named constant's scope of a class-level memory location; can be used by any procedure in the form
- **Class-level named constants** - **Private Const**, a named constant declared in a form class's declaration section just under the **Public Class**;
- **Class-level variables** - **Private**, a variable declared in a form class's declaration section just under the **Public Class**;
- **Const statement** - the statement used to create a named constant
- **Declaring a memory location** - reserving a location in the RAM for use within an app's code
- **Demoted** - opposite to **Promoted**; the process of converting a value from one data type to another that can store only smaller numbers with less precision
- **Dim statement** - the statement used to declare procedure-level variables
- **Enter event** - occurs when a control receives the focus
- **Flowchart** - a planning tool that uses standardized symbols to illustrate the steps a procedure must take to accomplish its purpose
  - try the soft installed as Flowgorithm
- **Flowlines** - the lines connecting the symbols in a flowchart
- **Form class's declarations section** - the area located between the **Public Class** and **End Class** clauses in the Code Editor window;
  - class-level memory locations are declared in this section
- **Format** - specifying the number of decimal places and the special characters to display in a number treated as string
- **Implicit type conversion** - the process by which a value is automatically converted to fit the data type of the memory location to which it is assigned;
  - **Option Strict On**
- **Input / Output symbol** - the parallelogram (rovnoběžník) in a flowchart
- **InputBox function**
  - allows an application to communicate with the user during run time
  - displays an input dialog box that contains: a prompt message, an input area, OK and Cancel buttons
- **Integer division operator** - represented by a backslash \ ; divides two integers and then returns the quotient (podíl) as an integer
- **Lifetime** - indicates how long a variable or named constant remains in the RAM
- **Literal type character** - a character (such as the letter D) appended (přidaný) to a literal for the purpose of forcing the literal to assume a different data type (such as **Decimal**)
- **Modulus operator** - represented by the keyword **Mod**; divides two numbers and then returns the remainder of the division
- **Named constant** - **Const**; **Private Const**; RAM location where programmers can store data that cannot be changed during run time
- **Private keyword** - used to declare class-level memory locations (variables and named constants)
- **Procedure scope** - the scope of a procedure-level memory location (variable or named constant)
  - memory location can be used only by the procedure that declares it
- **Procedure-level named constants** - **Const**; declared in a procedure and have procedure scope
- **Procedure-level variables** - **Dim**; declared in a procedure and have procedure scope
- **Process symbols** - the rectangles in a flowchart
- **Promoted** - opposite to **Demoted**; the process of converting a value from one data type to another that can store either larger numbers or numbers with greater precision
- **Pseudocode** - a planning tool that uses phrases to describe the steps a procedure must take to accomplish its purpose
- **Scope** - (prostor) indicates where a memory location (variable or named constant) can be used in an app's code
- **SelectAll method** - used to select all of the text contained in a text box
- **Start / Stop symbol** - the ovals in a flowchart

- **Static variable** - a procedure-level variable that remains in RAM and also retains its value until the app (rather than its declaring procedure) ends
- **String** - a sequence of characters (numbers, letters, special char, and so on)
- **String.Empty** - the value that represents the empty string in VB
- **TextChange event** - occurs each time the value in a control's Text property changes
- **ToString method** - formats a copy of a number and returns the result as a string
- **TryParse method** - used to convert a string to a specified numeric data type
- **Variable** - RAM location where programmers can temporarily store data, as well as change the data, during run time

**EXERCISE 01.Tip Solution\_introductory** - txt\_TextChanged, txt\_Enter, txt.SelectAll()

**EXERCISE 02.Jacobson Solution\_introductory** - decimal number with "D" - 0.05D, txt\_Enter, txt\_TextChanged, txt.SelectAll()

**EXERCISE 03.Moonbucks Solution\_introductory** - easter egg: If...Then...Else, pic.Visible = True / False  
- decimal number with "D" - 0.15D, txt\_Enter, txt\_TextChanged, txt.SelectAll()

**EXERCISE 04.Tax Solution\_introductory** - decimal number with "D" - 1.35D, txt\_Enter, txt\_TextChanged, txt.SelectAll(),  
lbl.BackColor = Color.Beige

**EXERCISE 05.Orders Solution\_intermediate** - Private... class-level variable, txt\_TextChanged, txt\_Enter, txt.SelectAll()

**EXERCISE 06.Enrolled Solution\_intermediate** - Private... class-level variable, txt\_TextChanged, txt\_Enter, txt.SelectAll()

**EXERCISE 07.Modified Enrolled Solution\_intermediate** - Static... procedure-level variable with class-level scope,  
txt\_TextChanged, txt\_Enter, txt.SelectAll()

**EXERCISE 08.Area Solution\_intermediate** - txt\_TextChanged, txt\_Enter, txt.SelectAll()

**EXERCISE 09.Salary Solution\_intermediate** - Const... procedure-level named constant, txt\_TextChanged, txt\_Enter, txt.SelectAll()

**EXERCISE 10.Kramden Solution\_intermediate** - txt\_TextChanged, txt\_Enter, txt.SelectAll()

**EXERCISE 11.Chopkins Solution\_intermediate** - lbl / Properties: BackColor = Info

- Const... procedure-level named constant, txt\_TextChanged, txt\_Enter, txt.SelectAll()

**EXERCISE 12.Chopkins Solution - Modified\_advanced** - Const... procedure-level named constant, txt\_TextChanged, txt\_Enter, txt.SelectAll(),  
lbl.BackColor = SystemColors.Control, lbl.BackColor = Color.Yellow

**EXERCISE 13.Grade Solution\_advanced** - when the result is subtraction from 100%, I use "1 - :x% = 1 - y%" and using: x.ToString("P1")

- using colon : to separate different instructions in one line

- txt\_TextChanged, txt\_Enter, txt.SelectAll()

- using colon : to separate different instructions in one line

- txt\_TextChanged, txt\_Enter, txt.SelectAll()

    lbl.BackColor = SystemColors.Control, lbl.BackColor = Color.Yellow

**EXERCISE 15.OnYourOwn Solution** - using colon : to separate different instructions in one line

- txt\_TextChanged, txt\_Enter, txt.SelectAll(),

    lbl.BackColor = SystemColors.Control, lbl.BackColor = Color.Yellow

**EXERCISE 16.FixIt Solution** - Const... procedure-level named constant, txt\_TextChanged, txt\_Enter, txt.SelectAll(),  
    lbl.BackColor = SystemColors.Control, lbl.BackColor = Color.Yellow / Color.Red

**Appendix E - Case Projects\_01.Your Special Day Catering-CH 01-03** - using .NET Framework 4.5.2

**Appendix E - Case Projects\_02.Crispies Bagels and Bites-CH 01-03** - using .NET Framework 4.8

In this exercise, you will complete the Restaurant Tip application from Chapter 2's Focus on the Concepts lesson. The application's Planning Chart is shown in Figure 3-34.

- Use either a flowchart or pseudocode to plan the btnCalc\_Click procedure, which should calculate and display a server's tip.
- Open the Tip Solution.sln file contained in the VB2017\Chap03\Tip Solution folder. Enter the three Option statements in the Code Editor window. Use the comments as a guide when coding the btnCalc\_Click procedure. Be sure to use variables in your code. Display the tip with a dollar sign and two decimal places. Save the solution and then start and test the application. (If the restaurant bill and tip percentage are 56 and 20, respectively, the tip is \$11.20.)
- Now professionalize your interface by coding each text box's TextChanged and Enter event procedures. Save the solution and then start and test the application.

### Planning Chart for the Restaurant Tip application

Purpose: Calculate and display a server's tip.

How?

User-provided

1. bill amount
2. tip percentage

user will enter in txtBill  
user will enter in txtPercentage

Application-provided

1. tip
2. button for ending the application

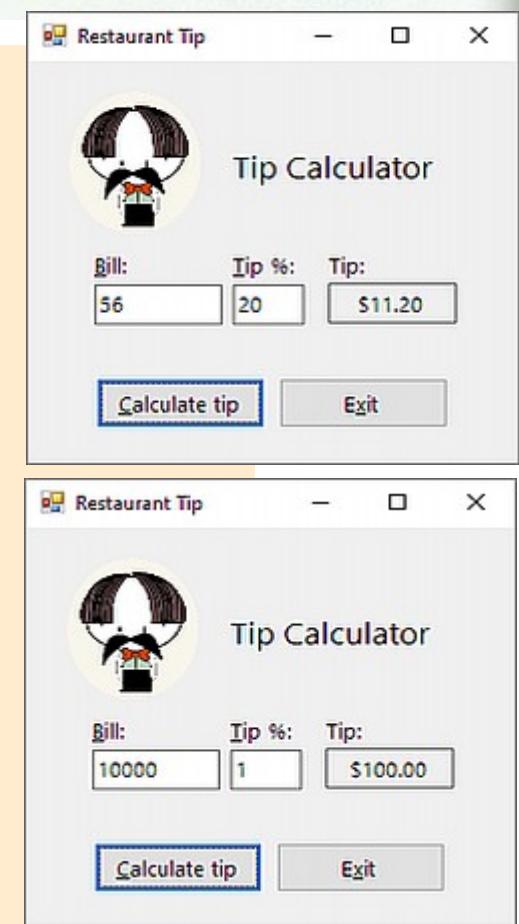
btnCalc\_Click will calculate and display in lblTip  
btnExit\_Click will end the application

Figure 3-34 Planning Chart for Exercise 1

```

1  ' Name:      Tip Project
2  ' Purpose:    Calculate and display a server's tip.
3  ' Programmer: <your name> on <current date>
4  Option Explicit On
5  Option Strict On
6  Option Infer Off
7
8  Public Class frmMain
9      Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
10         ' Calculates and displays a server's tip.
11         Dim decBill As Decimal
12         Dim decPercentage As Decimal
13         Dim decTip As Decimal
14         Decimal.TryParse(txtBill.Text, decBill)
15         Decimal.TryParse(txtPercentage.Text, decPercentage)
16
17         ' Calculate the tip:
18         decTip = decBill * decPercentage / 100
19
20         ' Display the tip with a dollar sign and two decimal places:
21         lblTip.Text = decTip.ToString("C2")
22     End Sub
23
24     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
25         Me.Close()
26     End Sub

```



```

27      Private Sub txtBill_TextChanged(sender As Object, e As EventArgs) Handles txtBill.TextChanged
28          lblTip.Text = String.Empty
29      End Sub
30
31
32      Private Sub txtBill_Enter(sender As Object, e As EventArgs) Handles txtBill.Enter
33          txtBill.SelectAll()
34      End Sub
35
36      Private Sub txtPercentage_TextChanged(sender As Object, e As EventArgs) Handles txtPercentage.TextChanged
37          lblTip.Text = String.Empty
38      End Sub
39
40      Private Sub txtPercentage_Enter(sender As Object, e As EventArgs) Handles txtPercentage.Enter
41          txtPercentage.SelectAll()
42      End Sub
43  End Class

```

## Chap03\Exercisel

### EXERCISE 02.Jacobson Solution\_introductory

- decimal number with "D" - 0.05D, txt\_Enter, txt\_TextChanged, txt.SelectAll()

In this exercise, you will complete the Jacobson Furniture application from Chapter 2's Apply the Concepts lesson. The application's Planning Chart is shown in Figure 3-35.

- Use either a flowchart or pseudocode to plan the btnCalc\_Click procedure, which should calculate and display both a 5% sales tax and the total due.
- Open the Jacobson Solution.sln file contained in the VB2017\Chap03\Jacobson Solution folder. Enter the three Option statements in the Code Editor window. Code the btnCalc\_Click procedure using variables and a named constant. Display the sales tax with a comma (if necessary) and two decimal places. Display the total due with a comma (if necessary), a dollar sign and two decimal places. Save the solution and then start and test the application. (If the sales amount is 500, the sales tax and total due are 25.00 and \$525.00, respectively.)
- Now professionalize your interface by coding the text box's TextChanged and Enter event procedures. Save the solution and then start and test the application.

#### Planning Chart for the Jacobson Furniture application

Purpose: Calculate and display the sales tax and total due amounts.

How?

User-provided

1. sales amount

user will enter in txtSales

Application-provided

1. 5% sales tax

btnCalc\_Click will calculate and display in lblTax

2. total due

btnCalc\_Click will calculate and display in lblTotal

3. button for ending the application

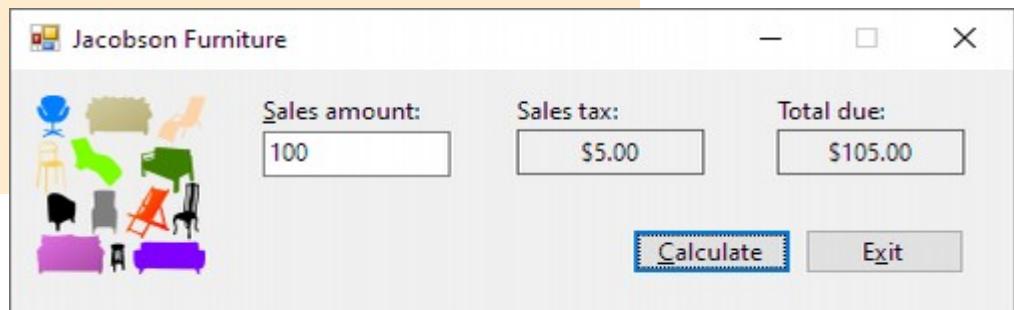
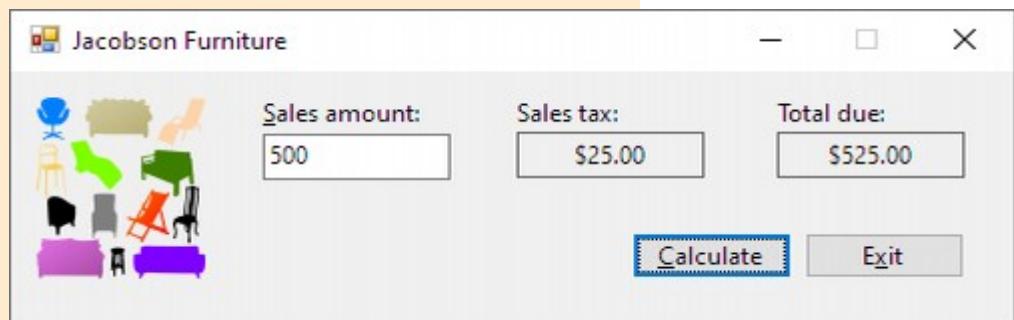
btnExit\_Click will end the application

Figure 3-35 Planning Chart for Exercise 2

```

1  ' Name:      Jacobson Solution
2  ' Purpose:   Calculate and display the sales tax and total due.
3  ' Programmer: <your name> on <current date>
4  Option Explicit On
5  Option Strict On
6  Option Infer Off
7
8  Public Class frmMain
9    Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
10     Me.Close()
11 End Sub
12
13  Private Sub txtSales_TextChanged(sender As Object, e As EventArgs) Handles txtSales.TextChanged
14    lblTax.Text = String.Empty
15    lblTotal.Text = String.Empty
16 End Sub
17
18  Private Sub txtSales_Enter(sender As Object, e As EventArgs) Handles txtSales.Enter
19    txtSales.SelectAll()
20 End Sub
21
22  Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
23    ' declare named constant on 5%; and variables:
24    Const decSalesTax5 As Decimal = 0.05D
25    Dim decSales As Decimal
26    Dim decTax As Decimal
27    Dim decTotal As Decimal
28
29    ' TryParse input value into number:
30    Decimal.TryParse(txtSales.Text, decSales)
31
32    ' do the math:
33    decTax = decSales * decSalesTax5
34    decTotal = decSales * decSalesTax5 + decSales
35
36    ' convert numbers to strings:
37    lblTax.Text = decTax.ToString("C2")
38    lblTotal.Text = decTotal.ToString("C2")
39  End Sub
40 End Class

```



## Chap03\Exercise1

### EXERCISE 03.Moonbucks Solution\_introductory

- easter egg: `If...Then...Else, pic.Visible = True / False`
- decimal number with "D" - `0.15D`, `txt_Enter`, `txt_TextChanged`, `txt.SelectAll()`

In this exercise, you will complete the Moonbucks application that you created in Exercise 2 in Chapter 2.

- Use Windows to copy the Moonbucks Solution folder from the VB2017\Chap02 folder to the VB2017\Chap03 folder. Open the Moonbucks Solution.sln file contained in the VB2017\Chap03\Moonbucks Solution folder. The application's Planning Chart is shown in Figure 3-36.
- Use either a flowchart or pseudocode to plan the `btnCalc_Click` procedure. Enter the three Option statements in the Code Editor window. Code the procedure using variables and a named constant. Display the calculated amounts with a comma (if necessary) and two decimal places. Save the solution and then start and test the application. (If the current sales amount is 2300.75, the increase and projected sales amounts are 230.08 and 2,530.83, respectively.)
- Now professionalize your interface by coding the text box's `TextChanged` and `Enter` event procedures. Save the solution and then start and test the application.

#### Planning Chart for the Moonbucks application

Purpose: Calculate and display the increase and projected sales amounts.

How?

User-provided

1. current sales amount

user will enter in `txtCurrent`

Application-provided

1. 10% increase amount

`btnCalc_Click` will calculate and display in `lblIncrease`

2. projected sales amount

`btnCalc_Click` will calculate and display in `lblProjected`

3. button for ending the application

`btnExit_Click` will end the application

Figure 3-36 Planning Chart for Exercise 3

```
1  Option Explicit On
2  Option Strict On
3  Option Infer Off
4
5  Public Class frmMain
6      Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
7          ' declare the var & named const:
8          Const decInterest10 As Decimal = 0.1D
9          Dim decCurrent As Decimal
10         Dim decIncrease As Decimal
11         Dim decProjected As Decimal
12
13         ' convert input text into number:
14         Decimal.TryParse(txtCurrent.Text, decCurrent)
15
16         ' assign declared var and do the math:
17         decIncrease = decCurrent * decInterest10
18         decProjected = decCurrent + decIncrease
19
20         ' convert numbers to strings:
21         lblIncrease.Text = decIncrease.ToString("C2")
22         lblProjected.Text = decProjected.ToString("C2")
23     End Sub
24
```

```

25  Private Sub btnEgg_Click(sender As Object, e As EventArgs) Handles btnEgg.Click
26      ' easter egg i have created: but how to hide picEgg -> I know s.s.:
27      If picEgg.Visible = True Then
28          picEgg.Visible = False
29      Else
30          picEgg.Visible = True
31      End If
32  End Sub
33
34  Private Sub txtCurrent_TextChanged(sender As Object, e As EventArgs) Handles txtCurrent.TextChanged
35      ' clear the old results on labels:
36      lblIncrease.Text = String.Empty
37      lblProjected.Text = String.Empty
38  End Sub
39
40  Private Sub txtCurrent_Enter(sender As Object, e As EventArgs) Handles txtCurrent.Enter
41      ' highlight the existing text on text box:
42      txtCurrent.SelectAll()
43  End Sub
44
45  Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
46      Me.Close()
47  End Sub
48 End Class

```



## EXERCISE 04.Tax Solution\_introductory

- decimal number with "D" - 1.35D, txt\_Enter, txt\_TextChanged, txt.SelectAll(),  
**lbl.BackColor = Color.Beige**

In this exercise, you will complete the Property Tax application that you created in Exercise 3 in Chapter 2.

- a. Use Windows to copy the Tax Solution folder from the VB2017\Chap02 folder to the VB2017\Chap03 folder. Open the Tax Solution.sln file contained in the VB2017\Chap03\Tax Solution folder. You created a Planning Chart for this application in Chapter 2. Enter the three Option statements in the Code Editor window. One of the buttons in the interface should calculate and display the property tax; use either

a flowchart or pseudocode to plan the button's Click event procedure. Code the procedure using variables and a named constant. Display the tax with a comma (if necessary), a dollar sign and two decimal places. Save the solution and then start and test the application. (If the assessed value is 87650, the tax is \$1,183.28.)

- b. Now professionalize your interface by coding the text box's TextChanged and Enter event procedures. Save the solution and then start and test the application.

```

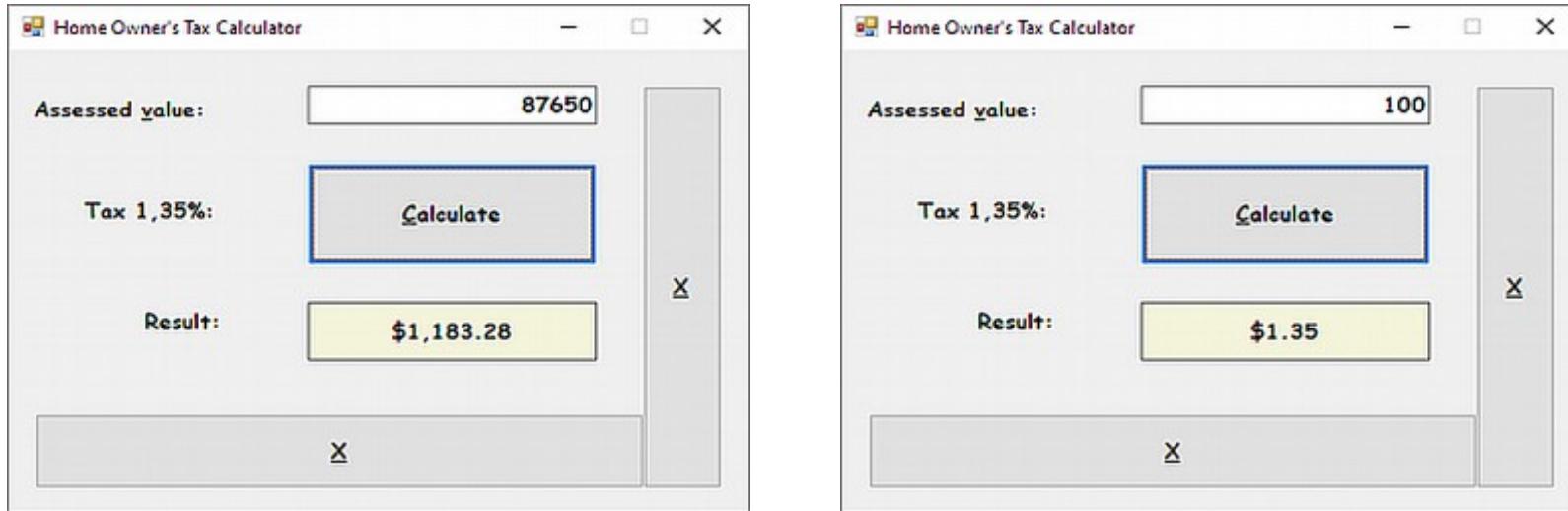
1  Option Explicit On
2  Option Strict On
3  Option Infer Off
4
5  Public Class frmMain
6      Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
7          Me.Close()
8      End Sub
9
10     Private Sub btnExit2_Click(sender As Object, e As EventArgs) Handles btnExit2.Click
11         Me.Close()
12     End Sub
13
14     Private Sub btnCalculate_Click(sender As Object, e As EventArgs) Handles btnCalculate.Click
15         lblResult.BackColor = Color.Beige
16
17         ' declare the var & named constant:
18         Const decTax135 As Decimal = 1.35D
19         Dim decValue As Decimal
20         Dim decResult As Decimal
21
22         ' convert input text into number:
23         Decimal.TryParse(txtValue.Text, decValue)
24
25         ' do the math:
26         decResult = decValue * decTax135 / 100
27
28         ' convert result to string:
29         lblResult.Text = decResult.ToString("C2")
30     End Sub
31

```

```

32     Private Sub txtValue_Enter(sender As Object, e As EventArgs) Handles txtValue.Enter
33         ' highlight old value in text box:
34         txtValue.SelectAll()
35     End Sub
36
37     Private Sub txtValue_TextChanged(sender As Object, e As EventArgs) Handles txtValue.TextChanged
38         ' clear the old result on label:
39         lblResult.Text = String.Empty
40     End Sub
41 End Class

```



## Chap03\Exercise1

### EXERCISE 05.Orders Solution\_intermediate

- **Private**... class-level variable, **txt\_TextChanged**, **txt\_Enter**, **txt.SelectAll()**

- Open the Orders Solution.sln file contained in the VB2017\Chap03\Orders Solution folder. The interface provides a button for adding the number ordered to the total ordered, and a button for subtracting the number ordered from the total ordered. Code the application using a class-level variable. Be sure to code the text box's TextChanged and Enter event procedures. Save the solution and then start the application. Test the application by adding any two numbers, then subtracting any three numbers, and then adding any four numbers. (If you add 7 and 5, and then subtract 2, 1, and 6, and then add 10, 7, 4, and 5, the total ordered will be 29.)

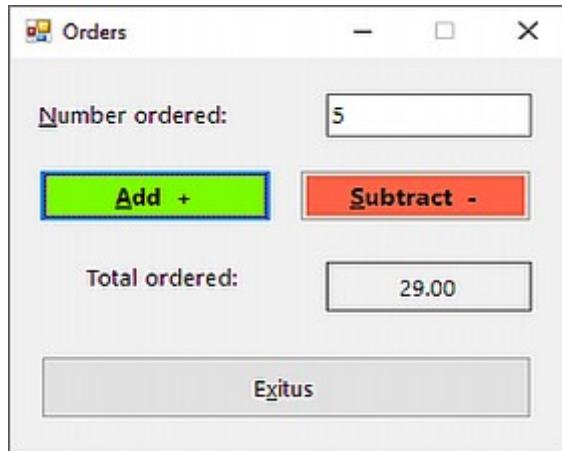
```
1  ' Name:      Orders Project
2  ' Purpose:    Calculate and display the total numbered ordered.
3  ' Programmer: <your name> on <current date>
4  Option Explicit On
5  Option Strict On
6  Option Infer Off
7
8  Public Class frmMain
9      ' declare the class-level variable, what can be used by both buttons:
10     Private intTotal As Integer
11
12     Private Sub btnAdd_Click(sender As Object, e As EventArgs) Handles btnAdd.Click
13         ' button + formula.
14         ' declare the + variable:
15         Dim intAdd As Integer
16
17         ' convert txt into number for calculation:
18         Integer.TryParse(txtNumOrdered.Text, intAdd)
19
20         ' do the math:
21         intTotal = intTotal + intAdd
22
23         ' convert number into string:
24         lblTotal.Text = intTotal.ToString("N")
25     End Sub
26
27     Private Sub btnSubtract_Click(sender As Object, e As EventArgs) Handles btnSubtract.Click
28         ' button - formula.
29         ' declare the var for -:
30         Dim intSubtract As Integer
31
32         ' convert input text into number:
33         Integer.TryParse(txtNumOrdered.Text, intSubtract)
34
35         ' do the math:
36         intTotal = intTotal - intSubtract
37
38         ' convert number into string:
39         lblTotal.Text = intTotal.ToString("N")
40     End Sub
41
42     Private Sub txtNumOrdered_Enter(sender As Object, e As EventArgs) Handles txtNumOrdered.Enter
43         txtNumOrdered.SelectAll()
44     End Sub
```

```

45
46     'Private Sub txtNumOrdered_TextChanged(sender As Object, e As EventArgs) Handles txtNumOrdered.TextChanged
47     ' if I use this, the previous result would be erased, what I don't like
48     'lblTotal.Text = String.Empty
49     'End Sub
50
51     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
52         Me.Close()
53     End Sub
54 End Class

```

test:  $7 + 5 - 2 - 1 - 6 + 10 + 7 + 4 + 5 = 29$



## Chap03\Exercise1

### EXERCISE 06.Enrolled Solution\_intermediate

- **Private**... class-level variable, **txt\_TextChanged**, **txt\_Enter**, **txt.SelectAll()**

6. Open the Enrolled Solution.sln file contained in the VB2017\Chap03\Enrolled Solution folder. The application provides a text box and a button for updating the total number of enrollees. Code the application using a class-level variable. Be sure to code the text box's TextChanged and Enter event procedures. Save the solution and then start and test the application.

```

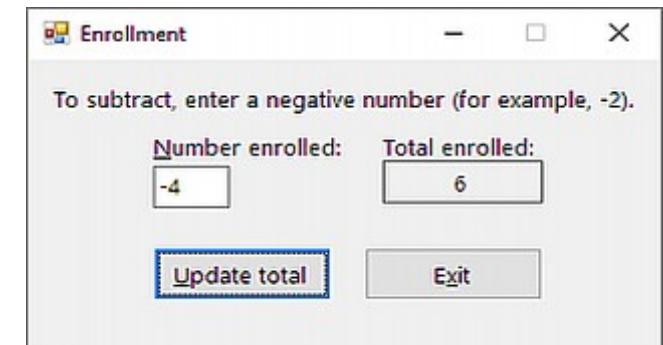
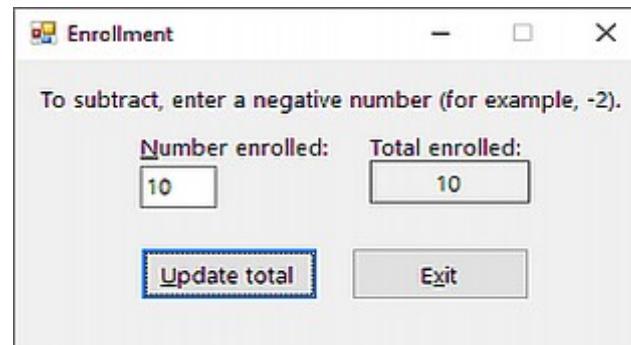
1  ' Name:      Enrolled Project
2  ' Purpose:   Calculate and display the total numbered enrolled.
3  ' Programmer: <your name> on <current date>
4  Option Explicit On
5  Option Strict On
6  Option Infer Off

```

```

7
8  Public Class frmMain
9      ' declare a class-level variable (in this case a Static would be better):
10     Private intTotalEnrolled As Integer
11
12     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
13         Me.Close()
14     End Sub
15
16     Private Sub btnUpdate_Click(sender As Object, e As EventArgs) Handles btnUpdate.Click
17         ' declare a procedure-level var for input number:
18         Dim intNumEnrolled As Integer
19
20         ' convert input text into number for calculation:
21         Integer.TryParse(txtNumEnrolled.Text, intNumEnrolled)
22
23         ' do the math:
24         intTotalEnrolled = intTotalEnrolled + intNumEnrolled
25
26         ' convert calculation into string displayed in label:
27         lblTotalEnrolled.Text = intTotalEnrolled.ToString("N0")
28     End Sub
29
30     Private Sub txtNumEnrolled_TextChanged(sender As Object, e As EventArgs) Handles txtNumEnrolled.TextChanged
31         ' clear the old result:
32         lblTotalEnrolled.Text = String.Empty
33     End Sub
34
35     Private Sub txtNumEnrolled_Enter(sender As Object, e As EventArgs) Handles txtNumEnrolled.Enter
36         ' highlight the text when focus:
37         txtNumEnrolled.SelectAll()
38     End Sub
39 End Class

```



- **Static**... procedure-level variable with class-level scope,  
txt\_TextChanged, txt\_Enter, txt.SelectAll()

7. In this exercise, you modify the Enrollment application from Exercise 6. Use Windows to make a copy of the Enrolled Solution folder. Rename the folder Modified Enrolled Solution. Open the Enrolled Solution.sln file contained in the VB2017\Chap03\Modified Enrolled Solution folder. Code the application without using a class-level variable. Save the solution and then start and test the application.

```
1  ' Name:      Modified Enrolled Project
2  ' Purpose:    Calculate and display the total numbered enrolled.
3  ' Programmer: <your name> on <current date>
4  Option Explicit On
5  Option Strict On
6  Option Infer Off
7
8  Public Class frmMain
9    ' Mod:
10   ' declare a class-level variable (in this case a Static would be better):
11   'Private intTotalEnrolled As Integer
12
13  Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
14    Me.Close()
15  End Sub
16
17  Private Sub btnUpdate_Click(sender As Object, e As EventArgs) Handles btnUpdate.Click
18    ' Mod: declare a procedure-level var with class-level scope:
19    Static intTotalEnrolled As Integer
20
21    ' declare a procedure-level var for input number:
22    Dim intNumEnrolled As Integer
23
24    ' convert input text into number for calculation:
25    Integer.TryParse(txtNumEnrolled.Text, intNumEnrolled)
26
27    ' do the math:
28    intTotalEnrolled = intTotalEnrolled + intNumEnrolled
29
30    ' convert calculation into string displayed in label:
31    lblTotalEnrolled.Text = intTotalEnrolled.ToString("N0")
32  End Sub
33
```

```

34      Private Sub txtNumEnrolled_TextChanged(sender As Object, e As EventArgs) Handles txtNumEnrolled.TextChanged
35          lblTotalEnrolled.Text = String.Empty
36          ' clear the old result:
37      End Sub
38
39      Private Sub txtNumEnrolled_Enter(sender As Object, e As EventArgs) Handles txtNumEnrolled.Enter
40          txtNumEnrolled.SelectAll()
41          ' highlight the text when focus
42      End Sub
43  End Class

```

## Chap03\Exercise1

### EXERCISE 08.Area Solution\_intermediate

- txt\_TextChanged, txt\_Enter, txt.SelectAll()

In this exercise, you will complete the Rectangle Area application that you created in Exercise 4 in Chapter 2.

- Use Windows to copy the Area Solution folder from the VB2017\Chap02 folder to the VB2017\Chap03 folder. Open the Area Solution.sln file contained in the VB2017\Chap03\Area Solution folder. You created a Planning Chart for this application in Chapter 2. Enter the three Option statements in the Code Editor window. One of the buttons in the interface should calculate and display the

rectangle's area in both square feet and square yards; use either a flowchart or pseudocode to plan the button's Click event procedure. Code the procedure using variables. Display the calculated amounts with a comma (if necessary) and one decimal place. Save the solution and then start and test the application.

- Now professionalize your interface by coding each text box's TextChanged and Enter event procedures. Save the solution and then start and test the application.

```

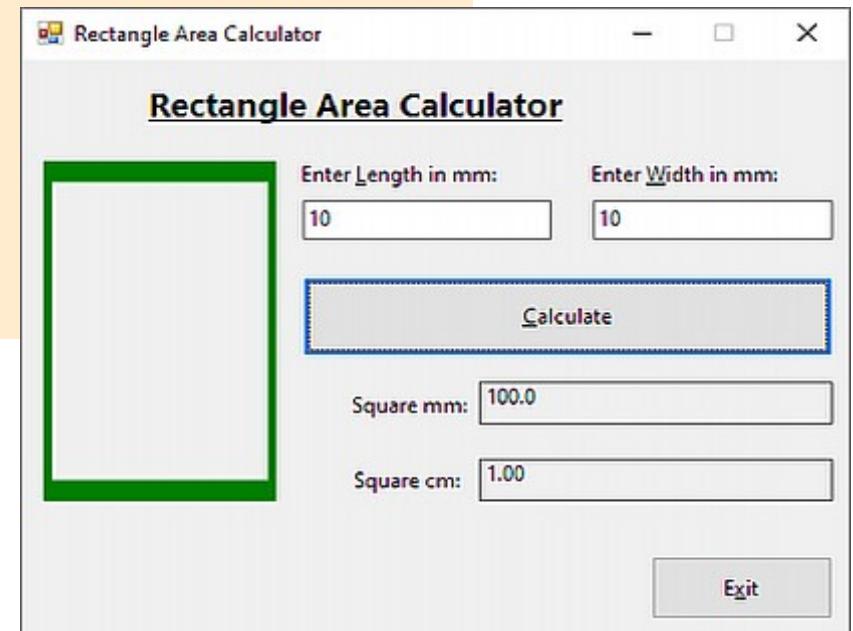
1  Option Explicit On
2  Option Strict On
3  Option Infer Off
4
5  Public Class frmMain
6      Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
7          Me.Close()
8      End Sub
9
10     Private Sub txtLength_TextChanged(sender As Object, e As EventArgs) Handles txtLength.TextChanged
11         ' clear the old results in labels:
12         lblSquareMm.Text = String.Empty
13         lblSquareCm.Text = String.Empty
14     End Sub
15

```

```

16  Private Sub txtWidth_TextChanged(sender As Object, e As EventArgs) Handles txtWidth.TextChanged
17      ' clear the old results in labels:
18      lblSquareMm.Text = String.Empty
19      lblSquareCm.Text = String.Empty
20  End Sub
21
22  Private Sub txtLength_Enter(sender As Object, e As EventArgs) Handles txtLength.Enter
23      ' highlight old text in text box Length:
24      txtLength.SelectAll()
25  End Sub
26
27  Private Sub txtWidth_Enter(sender As Object, e As EventArgs) Handles txtWidth.Enter
28      ' highlight old text in text box Width:
29      txtWidth.SelectAll()
30  End Sub
31
32  Private Sub btnCalculate_Click(sender As Object, e As EventArgs) Handles btnCalculate.Click
33      Dim singleLength As Single
34      Dim singleWidth As Single
35      Dim singleSquareMm As Single
36      Dim singleSquareCm As Single
37
38      ' convert txt into numbers used in calculations:
39      Single.TryParse(txtLength.Text, singleLength)
40      Single.TryParse(txtWidth.Text, singleWidth)
41
42      ' do the math:
43      singleSquareMm = singleLength * singleWidth
44      singleSquareCm = singleSquareMm / 100
45
46      ' convert numbers into strings:
47      lblSquareMm.Text = singleSquareMm.ToString("N1")
48      lblSquareCm.Text = singleSquareCm.ToString("N2")
49  End Sub
50 End Class

```



In this exercise, you will complete the Raises and New Salaries application that you created in Exercise 5 in Chapter 2.

- a. Use Windows to copy the Salary Solution folder from the VB2017\Chap02 folder to the VB2017\Chap03 folder. Open the Salary Solution.sln file contained in the VB2017\Chap03\Salary Solution folder. You created a Planning Chart for this application in Chapter 2.
- b. Enter the three Option statements in the Code Editor window. One of the buttons in the interface should calculate and display the two raises and the two

new salaries; use either a flowchart or pseudocode to plan the button's Click event procedure. Code the procedure using variables and named constants. Display the calculated amounts with a comma (if necessary), a dollar sign and two decimal places.

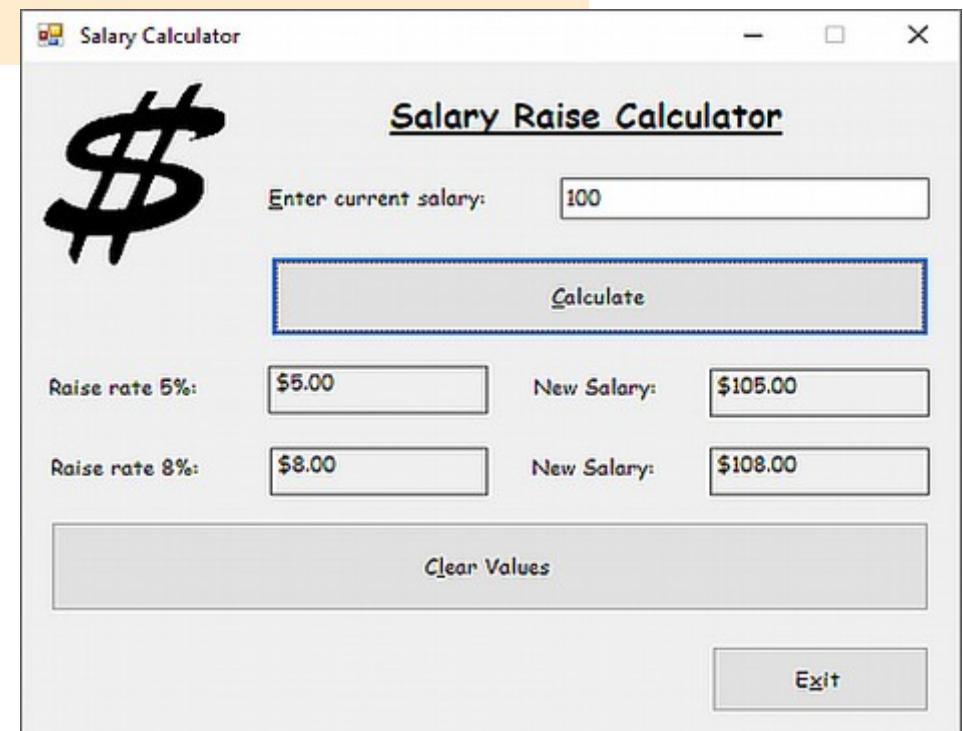
- c. One of the buttons in the interface should clear the user input and calculated amounts from the screen. Code the button's Click event procedure appropriately.
- d. Code the text box's Enter and TextChanged event procedures.
- e. Save the solution and then start and test the application.

```
1  Option Explicit On
2  Option Strict On
3  Option Infer Off
4
5  Public Class frmMain
6      Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
7          ' declare const and var:
8          Const dblRate5 As Double = 0.05
9          Const dblRate8 As Double = 0.08
10         Dim dblSalary As Double
11         Dim dblRaise5 As Double
12         Dim dblSalary5 As Double
13         Dim dblRaise8 As Double
14         Dim dblSalary8 As Double
15
16         ' convert text input into number:
17         Double.TryParse(txtSalary.Text, dblSalary)
18
19         ' do the math:
20         dblRaise5 = dblSalary * dblRate5
21         dblSalary5 = dblRaise5 + dblSalary
22         dblRaise8 = dblSalary * dblRate8
23         dblSalary8 = dblRaise8 + dblSalary
24
25         ' convert calculations into strings displayed:
26         lblRaise5.Text = dblRaise5.ToString("C2")
27         lblSalary5.Text = dblSalary5.ToString("C2")
28         lblRaise8.Text = dblRaise8.ToString("C2")
29         lblSalary8.Text = dblSalary8.ToString("C2")
30     End Sub
```

```

31
32     Private Sub txtSalary_TextChanged(sender As Object, e As EventArgs) Handles txtSalary.TextChanged
33         lblRaise5.Text = String.Empty
34         lblRaise8.Text = String.Empty
35         lblSalary5.Text = String.Empty
36         lblSalary8.Text = String.Empty
37     End Sub
38
39     Private Sub txtSalary_Enter(sender As Object, e As EventArgs) Handles txtSalary.Enter
40         txtSalary.SelectAll()
41     End Sub
42
43     Private Sub btnClear_Click(sender As Object, e As EventArgs) Handles btnClear.Click
44         txtSalary.Text = String.Empty
45         lblRaise5.Text = String.Empty
46         lblRaise8.Text = String.Empty
47         lblSalary5.Text = String.Empty
48         lblSalary8.Text = String.Empty
49     End Sub
50
51     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
52         Me.Close()
53     End Sub
54 End Class

```



In this exercise, you will complete the Kramden Inc. application that you created in Exercise 6 in Chapter 2.

- a. Use Windows to copy the Kramden Solution folder from the VB2017\Chap02 folder to the VB2017\Chap03 folder. Open the Kramden Solution.sln file contained in the VB2017\Chap03\Kramden Solution folder. You created a Planning Chart for this application in Chapter 2. Enter the three Option statements in the Code Editor window. One of the buttons in the interface

should calculate and display the total cost of the expense allowances; use either a flowchart or pseudocode to plan the button's Click event procedure. Code the procedure using variables. Display the total cost with a comma (if necessary), a dollar sign and two decimal places. Save the solution and then start and test the application.

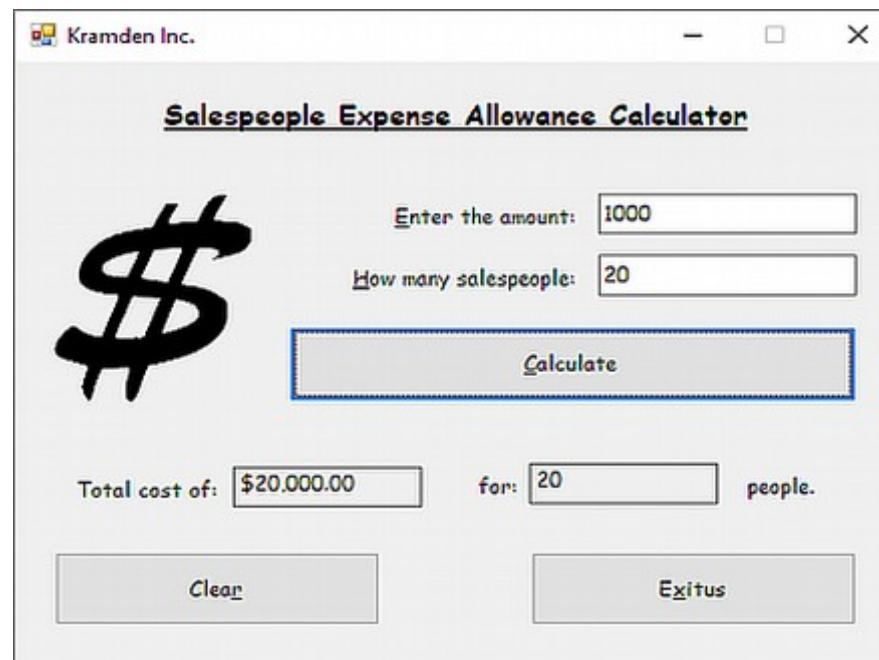
- b. Now professionalize your interface by coding each text box's TextChanged and Enter event procedures. Save the solution and then start and test the application.

```
1  Option Explicit On
2  Option Strict On
3  Option Infer Off
4
5  Public Class frmMain
6      Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
7          Me.Close()
8      End Sub
9
10     Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
11         Dim dblAmount As Double
12         Dim intPeople As Integer
13         Dim dblTtlCost As Double
14         Dim intTtlPeople As Integer
15
16         Double.TryParse(txtAmount.Text, dblAmount)
17         Integer.TryParse(txtPeople.Text, intPeople)
18
19         dblTtlCost = dblAmount * intPeople
20         intTtlPeople = intPeople
21
22         lblTtlCost.Text = dblTtlCost.ToString("C2")
23         lblTtlPeople.Text = intTtlPeople.ToString("N0")
24     End Sub
25
26     Private Sub txtAmount_TextChanged(sender As Object, e As EventArgs) Handles txtAmount.TextChanged
27         lblTtlCost.Text = String.Empty
28         lblTtlPeople.Text = String.Empty
29     End Sub
30 
```

```

31  Private Sub txtPeople_TextChanged(sender As Object, e As EventArgs) Handles txtPeople.TextChanged
32      lblTtlCost.Text = String.Empty
33      lblTtlPeople.Text = String.Empty
34  End Sub
35
36  Private Sub txtAmount_Enter(sender As Object, e As EventArgs) Handles txtAmount.Enter
37      txtAmount.SelectAll()
38  End Sub
39
40  Private Sub txtPeople_Enter(sender As Object, e As EventArgs) Handles txtPeople.Enter
41      txtPeople.SelectAll()
42  End Sub
43
44  Private Sub btnClear_Click(sender As Object, e As EventArgs) Handles btnClear.Click
45      txtAmount.Text = String.Empty
46      txtPeople.Text = String.Empty
47      lblTtlCost.Text = String.Empty
48      lblTtlPeople.Text = String.Empty
49  End Sub
50 End Class

```



- 1b1 / Properties: **BackColor = Info**

- **Const**... procedure-level named constant, `txt_TextChanged`, `txt_Enter`, `txt.SelectAll()`

Create a Windows Forms application. Use the following names for the project and solution, respectively: Chopkins Project and Chopkins Solution. Save the application in the VB2017\Chap03 folder. Change the form file's name to Main Form.vb. Change the form's name to frmMain. Create the interface shown in Figure 3-37. The interface contains six labels, three text boxes, and two buttons. The application calculates and displays the total number of packs ordered and the total price of the order. The prices of a 12 pack, a 5 pack, and a 2 pack are \$14.99, \$6.99, and \$2.50, respectively. Use variables and named constants in your code. Enter the three Option statements in the Code Editor window. The total sales amount should be displayed with a comma (if necessary), a dollar sign and two decimal places. Code each text box's TextChanged and Enter event procedures. Save the solution and then start and test the application.

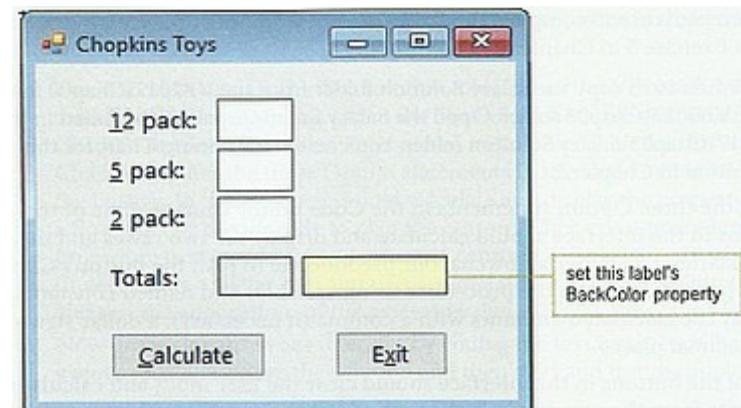


Figure 3-37 User interface for Exercise 11

```

1  Option Explicit On
2  Option Strict On
3  Option Infer Off
4
5  Public Class frmMain
6      Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
7          Const dbl12price As Double = 14.99
8          Const dbl5price As Double = 6.99
9          Const dbl2price As Double = 2.5
10         Dim int12 As Integer
11         Dim int5 As Integer
12         Dim int2 As Integer
13         Dim intTtlPacks As Integer
14         Dim dblTtlPrice As Double
15
16         Integer.TryParse(txt12.Text, int12)
17         Integer.TryParse(txt5.Text, int5)
18         Integer.TryParse(txt2.Text, int2)
19
20         intTtlPacks = int12 + int5 + int2
21         dblTtlPrice = (int12 * dbl12price) + (int5 * dbl5price) + (int2 * dbl2price)
22
23         lblTtlPacks.Text = intTtlPacks.ToString("N0")
24         lblTtlPrice.Text = dblTtlPrice.ToString("C2")
25     End Sub
26

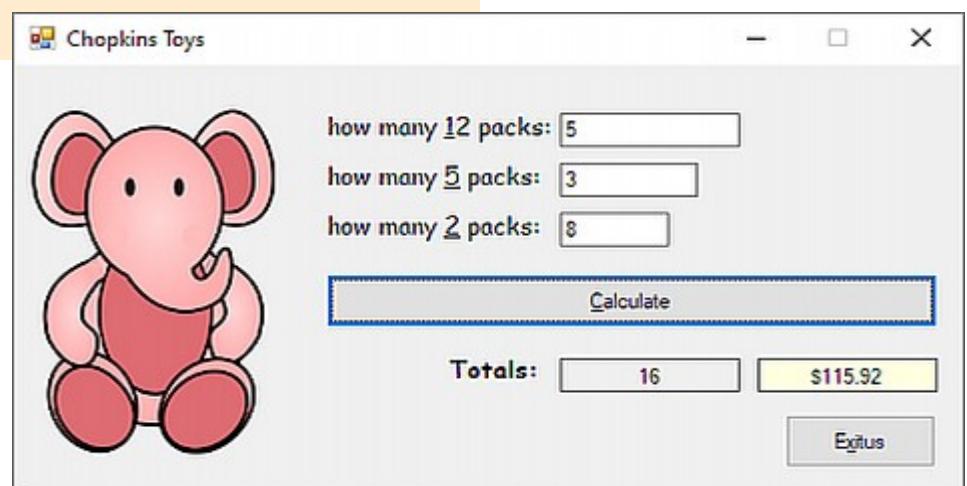
```

```

27  Private Sub txt12_TextChanged(sender As Object, e As EventArgs) Handles txt12.TextChanged
28      lblTtlPacks.Text = String.Empty
29      lblTtlPrice.Text = String.Empty
30  End Sub
31
32  Private Sub txt2_TextChanged(sender As Object, e As EventArgs) Handles txt2.TextChanged
33      lblTtlPacks.Text = String.Empty
34      lblTtlPrice.Text = String.Empty
35  End Sub
36
37  Private Sub txt5_TextChanged(sender As Object, e As EventArgs) Handles txt5.TextChanged
38      lblTtlPacks.Text = String.Empty
39      lblTtlPrice.Text = String.Empty
40  End Sub
41
42  Private Sub txt12_Enter(sender As Object, e As EventArgs) Handles txt12.Enter
43      txt12.SelectAll()
44  End Sub
45
46  Private Sub txt2_Enter(sender As Object, e As EventArgs) Handles txt2.Enter
47      txt2.SelectAll()
48  End Sub
49
50  Private Sub txt5_Enter(sender As Object, e As EventArgs) Handles txt5.Enter
51      txt5.SelectAll()
52  End Sub
53
54  Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
55      Me.Close()
56  End Sub
57 End Class

```

test:  $5 * 14.99 = 74.95$   
 $3 * 6.99 = 20.97$   
 $8 * 2.50 = 20$   
**16            115.92**



## Chap03\\_Exercise1

### EXERCISE 12.Chopkins Solution - Modified\_advanced

- **Const**... procedure-level named constant, `txt_TextChanged`, `txt_Enter`, `txt.SelectAll()`,  
`lbl.BackColor = SystemColors.Control`, `lbl.BackColor = Color.Yellow`

In this exercise, you modify the Chopkins Toys application from Exercise 11. Use Windows to make a copy of the Chopkins Solution folder. Rename the copy Modified Chopkins Solution. Open the Chopkins Solution.sln file contained in the Modified Chopkins Solution folder. Modify the interface as shown in Figure 3-38. The interface will now display the sale totals for each of the different packs. For example, if the customer purchased five 12 packs, the label that appears next to the associated text box should display 74.95 ( $5 * 14.99$ ). Modify the `btnCalc_Click` procedure appropriately. The procedure should also allow the user to enter the shipping charge, which should be added to the total sale amount. Save the solution and then start and test the application.



```
1  Public Class frmMain
2      Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
3          Me.Close()
4      End Sub
5
6      Private Sub txt12_TextChanged(sender As Object, e As EventArgs) Handles txt12.TextChanged
7          lblTtlPacks.Text = String.Empty
8          lblTtlPrice.Text = String.Empty
9          lbl12priceTtl.Text = String.Empty
10         lblTtlPrice.BackColor = SystemColors.Control
11         lblTtlPacks.BackColor = SystemColors.Control
12         lbl12priceTtl.BackColor = SystemColors.Control
13     End Sub
14
15     Private Sub txt2_TextChanged(sender As Object, e As EventArgs) Handles txt2.TextChanged
16         lblTtlPacks.Text = String.Empty
17         lblTtlPrice.Text = String.Empty
18         lbl2priceTtl.Text = String.Empty
19         lblTtlPrice.BackColor = SystemColors.Control
20         lblTtlPacks.BackColor = SystemColors.Control
21         lbl2priceTtl.BackColor = SystemColors.Control
22     End Sub
23
```

```
24  Private Sub txt5_TextChanged(sender As Object, e As EventArgs) Handles txt5.TextChanged
25      lblTtlPacks.Text = String.Empty
26      lblTtlPrice.Text = String.Empty
27      lbl5priceTtl.Text = String.Empty
28      lblTtlPrice.BackColor = SystemColors.Control
29      lblTtlPacks.BackColor = SystemColors.Control
30      lbl5priceTtl.BackColor = SystemColors.Control
31  End Sub
32
33  Private Sub txt12_Enter(sender As Object, e As EventArgs) Handles txt12.Enter
34      txt12.SelectAll()
35  End Sub
36
37  Private Sub txt2_Enter(sender As Object, e As EventArgs) Handles txt2.Enter
38      txt2.SelectAll()
39  End Sub
40
41  Private Sub txt5_Enter(sender As Object, e As EventArgs) Handles txt5.Enter
42      txt5.SelectAll()
43  End Sub
44
45  Private Sub txtShipping_Enter(sender As Object, e As EventArgs) Handles txtShipping.Enter
46      txtShipping.SelectAll()
47  End Sub
48
49  Private Sub txtShipping_TextChanged(sender As Object, e As EventArgs) Handles txtShipping.TextChanged
50      lblTtlPacks.Text = String.Empty
51      lblTtlPrice.Text = String.Empty
52  End Sub
53
54  Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
55      Const dbl12price As Double = 14.99
56      Const dbl5price As Double = 6.99
57      Const dbl2price As Double = 2.5
58      Dim int12 As Integer
59      Dim dbl12priceTtl As Double
60      Dim int5 As Integer
61      Dim dbl5priceTtl As Double
62      Dim int2 As Integer
63      Dim dbl2priceTtl As Double
64      Dim dblShipping As Double
65      Dim intTtlPacks As Integer
66      Dim dblTtlPrice As Double
67
```

```

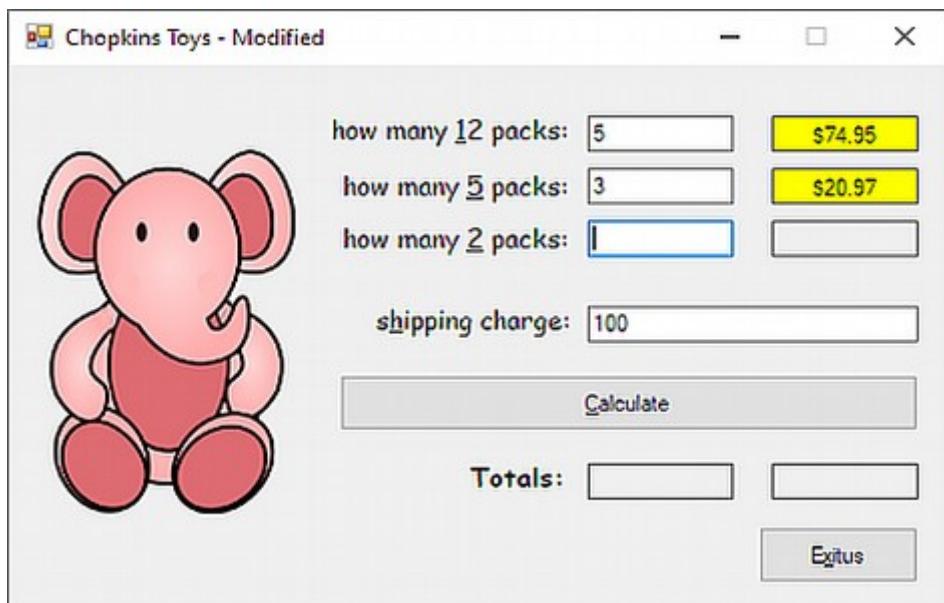
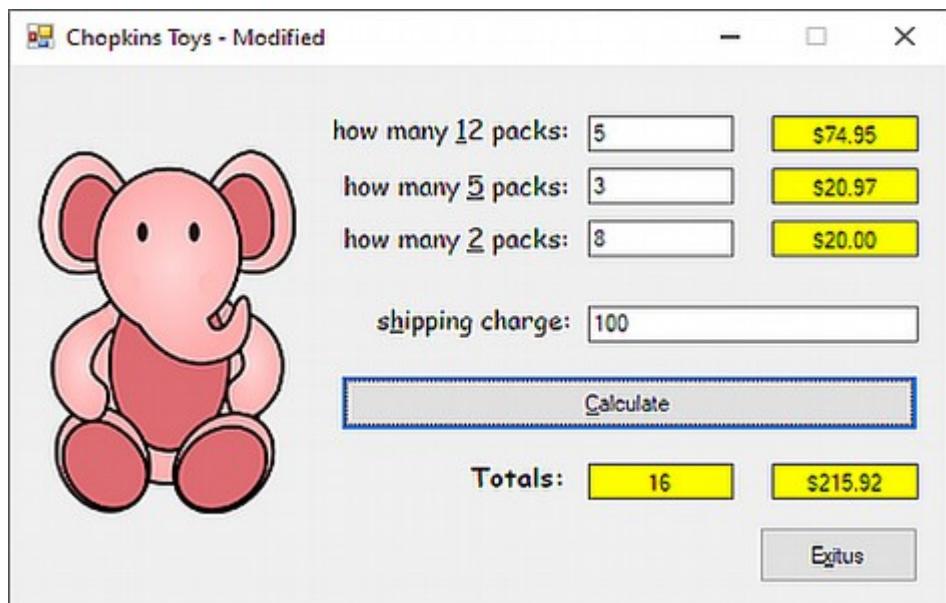
68     Integer.TryParse(txt12.Text, int12)
69     Integer.TryParse(txt5.Text, int5)
70     Integer.TryParse(txt2.Text, int2)
71     Double.TryParse(txtShipping.Text, dblShipping)

72
73     dbl12priceTtl = dbl12price * int12
74     dbl5priceTtl = dbl5price * int5
75     dbl2priceTtl = dbl2price * int2
76     intTtlPacks = int12 + int5 + int2
77     dblTtlPrice = dblShipping + (int12 * dbl12price) + (int5 * dbl5price) + (int2 * dbl2price)

78
79     lbl12priceTtl.Text = dbl12priceTtl.ToString("C2")
80     lbl5priceTtl.Text = dbl5priceTtl.ToString("C2")
81     lbl2priceTtl.Text = dbl2priceTtl.ToString("C2")
82     lblTtlPacks.Text = intTtlPacks.ToString("N0")
83     lblTtlPrice.Text = dblTtlPrice.ToString("C2")

84
85     lbl12priceTtl.BackColor = Color.Yellow
86     lbl2priceTtl.BackColor = Color.Yellow
87     lbl5priceTtl.BackColor = Color.Yellow
88     lblTtlPacks.BackColor = Color.Yellow
89     lblTtlPrice.BackColor = Color.Yellow
90 End Sub
91 End Class

```



## Chap03\Exercise1

### EXERCISE 13.Grade Solution\_advanced

- when the result is subtraction from 100%, I use "1 -":  $x\% = 1 - y\%$  and using: `x.ToString("P1")`
- using colon : to separate different instructions in one line
- `txt_TextChanged, txt_Enter, txt.SelectAll()`

In this exercise, you will complete the Grade Percentages application that you created in Exercise 7 in Chapter 2.

- a. Use Windows to copy the Grade Solution folder from the VB2017\Chap02 folder to the VB2017\Chap03 folder. Open the Grade Solution.sln file contained in the VB2017\Chap03\Grade Solution folder. You created a Planning Chart for this application in Chapter 2. Enter the three Option statements in the Code Editor window. One of the buttons in the interface should calculate and display the percentage of students receiving a grade of P (for Pass) and the percentage of students receiving a grade of F (for Fail). Code the button's Click event procedure using variables. Use Integer

variables for the numbers of P and F grades. Use Double variables for the percentages. Display the percentages with a percent sign and one decimal place. Save the solution and then start and test the application. (When testing the application with invalid data, the percentages may display as NaN, which stands for Not a Number. The message is a result of dividing a number by 0. In Chapter 4, you will learn how to use a selection structure to prevent this error message.)

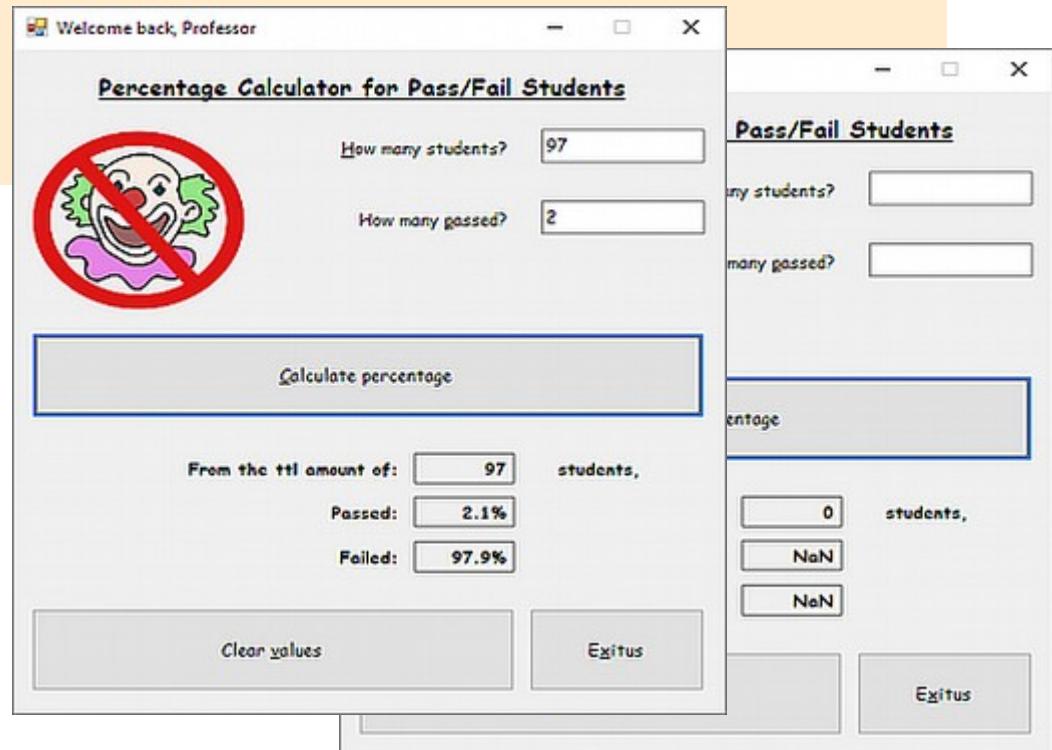
- b. Now professionalize your interface by coding each text box's TextChanged and Enter event procedures. Save the solution and then start and test the application.

```
1  Option Explicit On
2  Option Strict On
3  Option Infer Off
4
5  Class frmMain
6      Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
7          Me.Close()
8      End Sub
9
10     Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
11         Dim intStudents As Integer : Dim intPassed As Integer : Dim dblTtlPassed As Double
12         Integer.TryParse(txtStudents.Text, intStudents)
13         Integer.TryParse(txtPassed.Text, intPassed)
14
15         dblTtlPassed = intPassed / intStudents
16         ' had to drink 2 haine and hear _Happy.bsl to figure out da beautiful "1-" 55555:
17         dblTtlFailed = 1 - dblTtlPassed
18
19         lblTtlPassed.Text = dblTtlPassed.ToString("P1")
20         lblTtlFailed.Text = dblTtlFailed.ToString("P1")
21         lblTtlStudents.Text = intStudents.ToString("N0")
22     End Sub
23
24     Private Sub txtStudents_Enter(sender As Object, e As EventArgs) Handles txtStudents.Enter
25         'cistirna #2_Local highlight:
26         txtStudents.SelectAll()
27     End Sub
28
```

```

29     Private Sub txtPassed_Enter(sender As Object, e As EventArgs) Handles txtPassed.Enter
30         'cistirna #2_Local highlight:
31         txtPassed.SelectAll()
32     End Sub
33
34     Private Sub btnClear_Click(sender As Object, e As EventArgs) Handles btnClear.Click
35         txtStudents.Text = String.Empty
36         txtPassed.Text = String.Empty
37         lblTtlStudents.Text = String.Empty
38         lblTtlPassed.Text = String.Empty
39         lblTtlFailed.Text = String.Empty
40     End Sub
41
42     Private Sub txtStudents_TextChanged(sender As Object, e As EventArgs) Handles txtStudents.TextChanged
43         'cistirna #1_Global labels:
44         lblTtlFailed.Text = String.Empty
45         lblTtlPassed.Text = String.Empty
46         lblTtlStudents.Text = String.Empty
47     End Sub
48
49     Private Sub txtPassed_TextChanged(sender As Object, e As EventArgs) Handles txtPassed.TextChanged
50         'cistirna #1_Global labels:
51         lblTtlFailed.Text = String.Empty
52         lblTtlPassed.Text = String.Empty
53         lblTtlStudents.Text = String.Empty
54     End Sub
55 End Class

```



## EXERCISE 14.Sales Solution\_advanced

- using colon : to separate different instructions in one line
- `txt_TextChanged, txt_Enter, txt.SelectAll(),  
lbl.BackColor = SystemColors.Control, lbl.BackColor = Color.Yellow`

In this exercise, you will complete the Sales application that you created in Exercise 8 in Chapter 2.

- a. Use Windows to copy the Sales Solution folder from the VB2017\Chap02 folder to the VB2017\Chap03 folder. Open the Sales Solution.sln file contained in the VB2017\Chap03\Sales Solution folder. You created a Planning Chart for this application in Chapter 2. Enter the three Option statements in the Code Editor window. One of the buttons in the interface should calculate and display the percentage of the total sales made by each of

the three salespeople. Code the button's Click event procedure using Double variables. Display the percentages with a percent sign and one decimal place. Save the solution and then start and test the application. (When testing the application with invalid data, the percentages may display as NaN, which stands for Not a Number. The message is a result of dividing a number by 0. In Chapter 4, you will learn how to use a selection structure to prevent this error message.)

- b. Now professionalize your interface by coding each text box's TextChanged and Enter event procedures. Save the solution and then start and test the application.

```

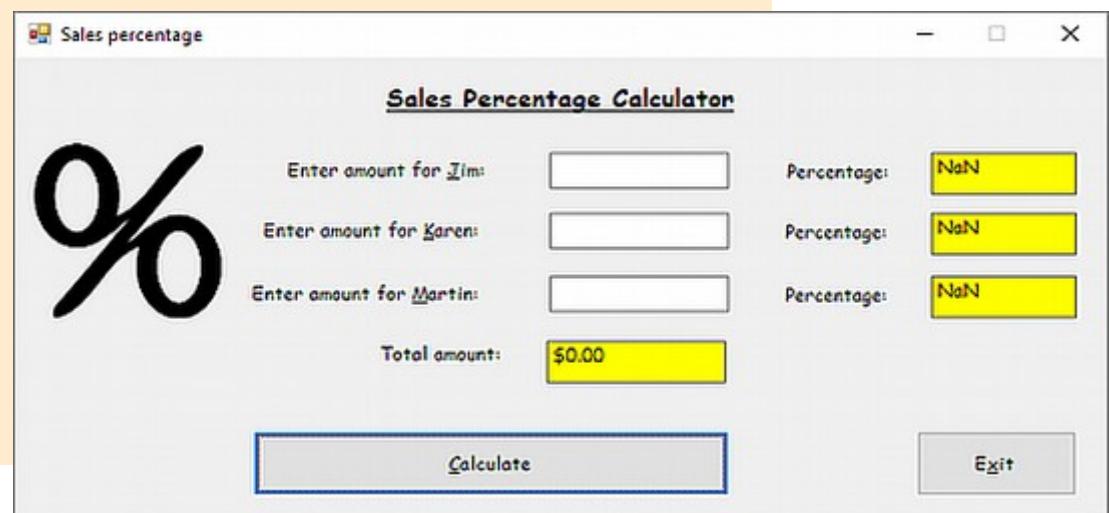
1  Option Explicit On
2  Option Strict On
3  Option Infer Off
4
5  Public Class frmMain
6      Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
7          Me.Close()
8      End Sub
9
10     Private Sub txtJim_TextChanged(sender As Object, e As EventArgs) Handles txtJim.TextChanged
11         lblTotal.Text = String.Empty : lblTotal.BackColor = SystemColors.Control
12         lblPEJim.Text = String.Empty : lblPEJim.BackColor = SystemColors.Control
13         lblPEKaren.Text = String.Empty : lblPEKaren.BackColor = SystemColors.Control
14         lblPEMartin.Text = String.Empty : lblPEMartin.BackColor = SystemColors.Control
15     End Sub
16
17     Private Sub txtKaren_TextChanged(sender As Object, e As EventArgs) Handles txtKaren.TextChanged
18         lblTotal.Text = String.Empty : lblTotal.BackColor = SystemColors.Control
19         lblPEKaren.Text = String.Empty : lblPEKaren.BackColor = SystemColors.Control
20         lblPEJim.Text = String.Empty : lblPEJim.BackColor = SystemColors.Control
21         lblPEMartin.Text = String.Empty : lblPEMartin.BackColor = SystemColors.Control
22     End Sub
23
24     Private Sub txtMartin_TextChanged(sender As Object, e As EventArgs) Handles txtMartin.TextChanged
25         lblTotal.Text = String.Empty : lblTotal.BackColor = SystemColors.Control
26         lblPEMartin.Text = String.Empty : lblPEMartin.BackColor = SystemColors.Control
27         lblPEJim.Text = String.Empty : lblPEJim.BackColor = SystemColors.Control
28         lblPEKaren.Text = String.Empty : lblPEKaren.BackColor = SystemColors.Control
29     End Sub

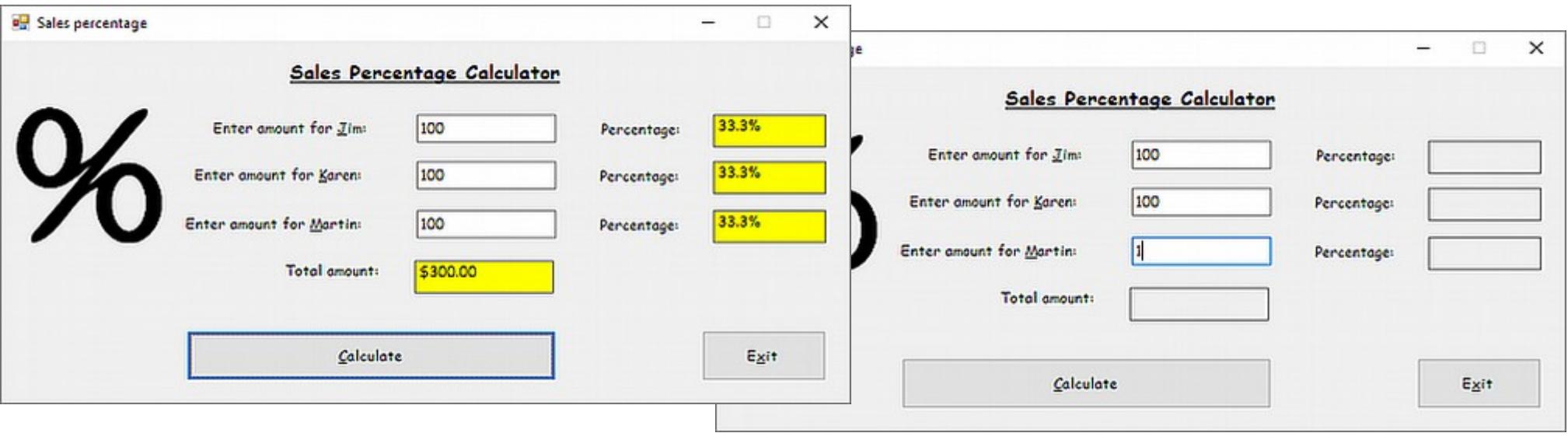
```

```

30
31     Private Sub txtJim_Enter(sender As Object, e As EventArgs) Handles txtJim.Enter
32         txtJim.SelectAll()
33     End Sub
34
35     Private Sub txtKaren_Enter(sender As Object, e As EventArgs) Handles txtKaren.Enter
36         txtKaren.SelectAll()
37     End Sub
38
39     Private Sub txtMartin_Enter(sender As Object, e As EventArgs) Handles txtMartin.Enter
40         txtMartin.SelectAll()
41     End Sub
42
43     Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
44         Dim dblJim As Double
45         Dim dblKaren As Double
46         Dim dblMartin As Double
47         Dim dblTotal As Double
48         Dim dblPEJIm As Double
49         Dim dblPEKKaren As Double
50         Dim dblPEMartin As Double
51
52         Double.TryParse(txtJim.Text, dblJim)
53         Double.TryParse(txtKaren.Text, dblKaren)
54         Double.TryParse(txtMartin.Text, dblMartin)
55
56         dblTotal = dblJim + dblKaren + dblMartin
57         dblPEJIm = dblJim / dblTotal
58         dblPEKKaren = dblKaren / dblTotal
59         dblPEMartin = dblMartin / dblTotal
56
60
61         lblTotal.Text = dblTotal.ToString("C2")
62         lblPEJIm.Text = dblPEJIm.ToString("P1")
63         lblPEKKaren.Text = dblPEKKaren.ToString("P1")
64         lblPEMartin.Text = dblPEMartin.ToString("P1")
65
66         lblTotal.BackColor = Color.Yellow
67         lblPEJIm.BackColor = Color.Yellow
68         lblPEKKaren.BackColor = Color.Yellow
69         lblPEMartin.BackColor = Color.Yellow
70     End Sub
71 End Class

```





## Chap03\Exercise1

### EXERCISE 15.OnYourOwn Solution

- using colon : to separate different instructions in one line

- txt\_TextChanged, txt\_Enter, txt.SelectAll(),

lbl.BackColor = SystemColors.Control, lbl.BackColor = Color.Yellow

Create a Windows Forms application. Use the following names for the project and solution, respectively: OnYourOwn Project and OnYourOwn Solution. Save the application in the VB2017\Chap03 folder. Plan and design an application of your choice. The only requirement is that you must follow the minimum guidelines listed in Figure 3-39. Before starting the application, be sure to verify the name of the startup form. Save the solution and then start and test the application.

1. The user interface must contain a minimum of two text boxes, four labels, and two buttons. One of the buttons must be an Exit button.
2. The interface can include a picture box, but this is not a requirement.

3. The interface must follow the GUI design guidelines summarized in Figure 2-20 in Chapter 2. The guidelines are also summarized in Appendix A.
4. Objects that are either coded or referred to in code should be named appropriately.
5. The Code Editor window must contain comments, the three Option statements, at least two variables, at least one named constant, at least two assignment statements, and the Me.Close() statement. The application must perform at least one calculation.
6. Each text box's TextChanged and Enter event procedures should be coded.

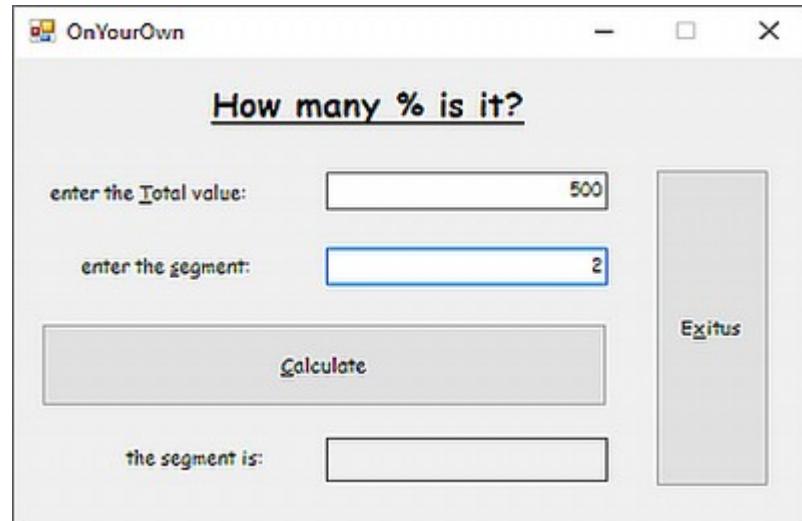
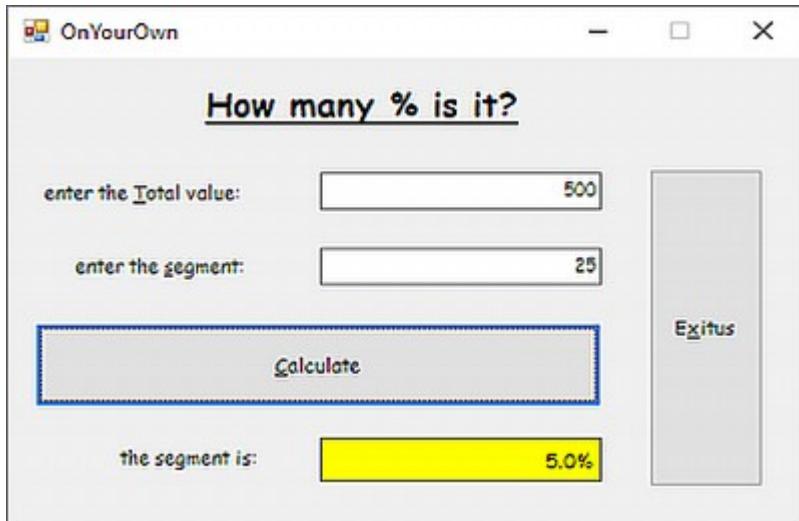
Figure 3-39 Guidelines for Exercise 15

```

1  ' Name:    % Calculator
2  ' Purpose: From 1.entered Total value calculate the % of the amount 2.entered
3  ' Coder:   poakamandizi
4  Option Explicit On
5  Option Strict On
6  Option Infer Off
7

```

```
8  Public Class frmMain
9      Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
10         Me.Close()
11     End Sub
12
13     Private Sub txtTotal_TextChanged(sender As Object, e As EventArgs) Handles txtTotal.TextChanged
14         lblShow.Text = String.Empty : lblShow.BackColor = SystemColors.Control
15     End Sub
16
17     Private Sub txtSegment_TextChanged(sender As Object, e As EventArgs) Handles txtSegment.TextChanged
18         lblShow.Text = String.Empty : lblShow.BackColor = SystemColors.Control
19     End Sub
20
21     Private Sub txtTotal_Enter(sender As Object, e As EventArgs) Handles txtTotal.Enter
22         txtTotal.SelectAll()
23     End Sub
24
25     Private Sub txtSegment_Enter(sender As Object, e As EventArgs) Handles txtSegment.Enter
26         txtSegment.SelectAll()
27     End Sub
28
29     Private Sub btnCalculate_Click(sender As Object, e As EventArgs) Handles btnCalculate.Click
30         ' _F3_declare variables & named constant:
31         ' sorry, i don't have any use for a named constant :)
32         Dim dblTotal As Double
33         Dim dblSegment As Double
34         Dim dblShow As Double
35
36         ' _F4_convert input txt into nr:
37         Double.TryParse(txtTotal.Text, dblTotal)
38         Double.TryParse(txtSegment.Text, dblSegment)
39
40         ' do the math:
41         dblShow = dblSegment / dblTotal
42
43         ' convert nr into str:
44         ' P1 converts number into % (*100)
45         lblShow.Text = dblShow.ToString("P1")
46
47         ' what.which = expression:
48         lblShow.BackColor = Color.Yellow
49     End Sub
50 End Class
```



## Chap03\ Exercise1

### EXERCISE 16.FixIt Solution

- **Const**... procedure-level named constant, `txt_TextChanged`, `txt_Enter`, `txt.SelectAll()`,  
`lbl.BackColor = SystemColors.Control`, `lbl.BackColor = Color.Yellow / Color.Red`

16. Open the VB2017\Chap03\FixIt Solution\FixIt Solution.sln file. The application should calculate and display the area of a rectangular floor (in square feet) and the total cost of tiling the floor. Open the Code Editor window and correct the errors in the code. Save the solution and then start and test the application.

```
1  ' Name:      FixIt Project
2  ' Purpose:    Displays area and total cost.
3  ' Programmer: PoaKamaNdizi on Today
4
5  Option Explicit On
6  Option Strict On
7  Option Infer Off
8
```

```
9  Public Class frmMain
10     Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
11         ' Calculates and displays area and total cost.
12
13         ' original:
14         'Const dblSQ_FT_PRICE As Double = 5.99      ' <- correct, but I don't like the name
15         Dim dblLength As Double
16         Dim dblWidth As Double
17         Dim dblArea As Double
18         ' my fix:
19         Const dblPrice As Double = 5.99           ' <- easy name
20         Dim dblTotalCost As Double                 ' <- has been missing
21
22         ' original:
23         'dblLength = txtLength.Text          ' <- missing Function: TryParse
24         'dblWidth = txtWidth.Text          ' <- missing Function: TryParse
25         ' my fix:
26         Double.TryParse(txtLength.Text, dblLength)
27         Double.TryParse(txtWidth.Text, dblWidth)
28
29         dblArea = dblLength * dblWidth
30
31         ' original:
32         'dblTotalCost = dblArea + dblSQ_FT_PRICE
33         ' my fix:
34         dblTotalCost = dblArea * dblPrice      ' <- dblTotalCost delcaration had been missing; changed const name
35
36         ' original:
37         'lblArea.Text = dblArea.("N2")        ' <- missing Function: ToString
38         'lblTotal.Text = dblTotalCost("C2")    ' <- missing Function: ToString
39         ' my fix:
40         lblArea.Text = dblArea.ToString("N2")
41         lblTotal.Text = dblTotalCost.ToString("C2")
42
43         ' my extra colors:
44         lblArea.BackColor = Color.Yellow
45         lblTotal.BackColor = Color.Red
46         'what.which = expression
47     End Sub
48
49     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
50         Me.Close()
51     End Sub
52
```

```

53     Private Sub txtLength_Enter(sender As Object, e As EventArgs) Handles txtLength.Enter
54         ' Select existing text.
55
56         ' original:
57         'txtLength.SelectedText      ' <- wrong keyword
58         ' my fix:
59         txtLength.SelectAll()      ' <- correct keyword
60     End Sub
61
62     Private Sub txtLength_TextChanged(sender As Object, e As EventArgs) Handles txtLength.TextChanged
63         ' Clear calculated results.
64
65         ' original:
66         'lblArea.Text = Empty      ' <- missing String.
67         'lblTotal.Text = Empty     ' <- missing String.
68         ' my fix:
69         lblArea.Text = String.Empty
70         lblTotal.Text = String.Empty
71         ' my extra colors:
72         lblArea.BackColor = SystemColors.Control
73         lblTotal.BackColor = SystemColors.Control
74     End Sub
75
76     Private Sub txtWidth_Enter(sender As Object, e As EventArgs) Handles txtWidth.Enter
77         ' Select existing text.
78         txtWidth.SelectAll()
79     End Sub
80
81     Private Sub txtWidth_TextChanged(sender As Object, e As EventArgs) Handles txtWidth.TextChanged
82         ' Clear calculated results.
83         lblArea.Text = String.Empty
84         lblTotal.Text = String.Empty
85         ' my extra colors:
86         lblArea.BackColor = SystemColors.Control
87         lblTotal.BackColor = SystemColors.Control
88     End Sub
89 End Class

```



## Your Special Day Catering (Chapters 1–3)

Create an application for Your Special Day Catering. The interface should allow the user to enter the customer ID, the bride's name, the groom's name, and the date of the wedding reception. It should also allow the user to enter the number of beef dinners, the number of chicken dinners, and the number of vegetarian dinners ordered for the reception. The interface should display the total number of dinners ordered, the subtotal (which is the total price of the order without sales tax), the sales tax, and the total price of the order with sales tax. Each dinner costs \$26.75, and the sales tax rate is 5%. Include an image in the interface. (You can find many different images on the Open Clip Art Library Web site at [openclipart.org](http://openclipart.org).)

```
1  ' Name:      Your Special Day Catering.
2  ' Purpose:    Calculate and display price for wedding dinner.
3  ' Programmer: Me on just now. CH 01-03 so i make it simple
4  Option Explicit On
5  Option Strict On
6  Option Infer Off
7
8  Public Class FrmMain
9      Private Sub btnCalculate_Click(sender As Object, e As EventArgs) Handles btnCalculate.Click
10         ' each dinner costs $26.75 = Const decDinner D; sales tax rate = 5% = 0.05D -> Const decTax5 D
11         ' txt1ID; txt2BrideNam; txt3GroomName; txt4Date; txt5Beef; txt6Chicken = int6Chicken; ' txt7Vege = int7Vege
12         ' int1Dinners -> lbl1Dinners; decSubtotal -> lbl2Subtotal; decTotal -> lbl3Total
13
14         Const decOneDinner As Decimal = 26.75D : Const decTax5 As Decimal = 0.05D
15         Dim int5Beef As Integer : Dim int6Chicken As Integer : Dim int7Vege As Integer
16         Dim int1Dinners As Integer : Dim decSubtotal As Decimal : Dim decTotal As Decimal
17
18         Integer.TryParse(txt5Beef.Text, int5Beef)
19         Integer.TryParse(txt6Chicken.Text, int6Chicken)
20         Integer.TryParse(txt7Vege.Text, int7Vege)
21
22         int1Dinners = int5Beef + int6Chicken + int7Vege
23         decSubtotal = int1Dinners * decOneDinner
24         decTotal = (decSubtotal * decTax5) + decSubtotal
25
26         lbl1Dinners.Text = int1Dinners.ToString
27         lbl2Subtotal.Text = decSubtotal.ToString("C2")
28         lbl3Total.BackColor = Color.Red
29         lbl3Total.Text = decTotal.ToString("C2")
30     End Sub
31
```

```

32     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
33         Me.Close()
34     End Sub
35
36     ' accept only numbers & backspace key:
37     Private Sub txt567_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txt5Beef.KeyPress,
38                                     txt6Chicken.KeyPress, txt7Vege.KeyPress
39         If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
40             e.Handled = True
41         End If
42     End Sub
43
44     ' clear calculations when input values change:
45     Private Sub txt567_TextChanged(sender As Object, e As EventArgs) Handles txt5Beef.TextChanged,
46                                     txt6Chicken.TextChanged, txt7Vege.TextChanged
47         lbl1Dinners.Text = Nothing
48         lbl2Subtotal.Text = String.Empty
49         lbl3Total.Text = Nothing
50         lbl3Total.BackColor = SystemColors.Control
51     End Sub
52 End Class

```

01.Your Special Day Catering

1. Customer ID:

2. Bride's name:

3. Groom's name:

4. Date of wedding reception:   
(DD/MM/YYYY)

5. Number of dinners (á \$26.75):      Beef:   
                                                Chicken:   
                                                 Vegetarian:

Dinners:  Subtotal:  Total price:



Total:  $26.75 * 0.05 = 1.3375$      $1.3375 + 26.75 = \mathbf{28.0875}$

01.Your Special Day Catering

1. Customer ID:

2. Bride's name:

3. Groom's name:

4. Date of wedding reception:   
(DD/MM/YYYY)

5. Number of dinners (á \$26.75):      Beef:   
                                                Chicken:   
                                                 Vegetarian:

Dinners:  Subtotal:  Total price:

Subtotal:  $134 * 26.75 = 3584.5$   
Total:  $3584.5 * 0.05 = 179.225$      $3584.5 + 179.225 = \mathbf{3763.725}$



Subtotal:  $134 * 26.75 = 3584.5$   
Total:  $3584.5 * 0.05 = 179.225$      $3584.5 + 179.225 = \mathbf{3763.725}$

## Crispies Bagels and Bites (Chapters 1–3)

Create an application for Crispies Bagels and Bites. The interface should allow the salesclerk to enter the number of bagels, donuts, and cups of coffee a customer orders. Bagels are 99¢, donuts are 75¢, and coffee is \$1.20 per cup. The application should calculate and display the subtotal (which is the total price of the order without sales tax), the sales tax, and the total due. The sales tax rate is 6%. Include an image in the interface. (You can find many different images on the Open Clip Art Library Web site at [openclipart.org](http://openclipart.org).)

```
1  ' Name:      Bakery Project
2  ' Purpose:    Calculate and display customers order
3  ' Programmer: Me on just now_using .NET Framework 4.8
4  Option Explicit On
5  Option Strict On
6  Option Infer Off
7
8  Public Class frmMain
9      Private Sub btnCalculate_Click(sender As Object, e As EventArgs) Handles btnCalculate.Click
10         ' Decimal constants: bagel = 0.99D; donut = 0.75D; coffee = 1.2D; Sales Tax = 6% = 0.06D
11         ' txt1Bagel; txt2Donut; txt3Coffee;  lbl1Subtotal; lbl2SalesTax; lbl3Total
12
13         Const dec1Bagel As Decimal = 0.99D          ' price for 1 Bagel = $ 0.99
14         Const dec2Donut As Decimal = 0.75D          ' price for 1 Donut = $ 0.75
15         Const dec3Coffe As Decimal = 1.2D           ' price for 1 Coffee = $ 1.20
16         Const decSalesTax6 As Decimal = 0.06D        ' Sales tax = 6% = 0.06
17         Dim decSubtotal As Decimal : Dim decSalesTax As Decimal : Dim decTotal As Decimal
18
19         Dim int1Bagel As Integer : Dim int2Donut As Integer : Dim int3Coffee As Integer
20         Integer.TryParse(txt1Bagel.Text, int1Bagel)
21         Integer.TryParse(txt2Donut.Text, int2Donut)
22         Integer.TryParse(txt3Coffee.Text, int3Coffee)
23
24         decSubtotal = (int1Bagel * dec1Bagel) + (int2Donut * dec2Donut) + (int3Coffee * dec3Coffe)
25         decSalesTax = decSubtotal * decSalesTax6
26         decTotal = decSubtotal + decSalesTax
27
28         lbl1Subtotal.Text = decSubtotal.ToString("C2")
29         lbl2SalesTax.Text = decSalesTax.ToString("C2")
30         lbl3Total.Text = decTotal.ToString("C2") : lbl3Total.BackColor = Color.Red
31     End Sub
32
```

```
33  Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
34      Me.Close()
35  End Sub
36
37  ' allow only integers & Backspace key:
38  Private Sub txt123_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txt1Bagel.KeyPress, txt2Donut.KeyPress, txt3Coffee.KeyPress
39      If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
40          e.Handled = True
41      End If
42  End Sub
43
44  ' when inputs change -> clear calculations:
45  Private Sub txt123_TextChanged(sender As Object, e As EventArgs) Handles txt1Bagel.TextChanged, txt2Donut.TextChanged,
46   txt3Coffee.TextChanged
47      lbl1Subtotal.Text = Nothing
48      lbl2SalesTax.Text = String.Empty
49      lbl3Total.Text = Nothing : lbl3Total.BackColor = SystemColors.Control
50  End Sub
51
52  Private Sub txt1Bagel_Enter(sender As Object, e As EventArgs) Handles txt1Bagel.Enter
53      txt1Bagel.SelectAll()
54  End Sub
55
56  Private Sub txt2Donut_Enter(sender As Object, e As EventArgs) Handles txt2Donut.Enter
57      txt2Donut.SelectAll()
58  End Sub
59
60  Private Sub txt3Coffie_Enter(sender As Object, e As EventArgs) Handles txt3Coffee.Enter
61      txt3Coffee.SelectAll()
62  End Sub
63
64  Private Sub txt1Bagel_MouseClick(sender As Object, e As MouseEventArgs) Handles txt1Bagel.MouseClick
65      txt1Bagel.SelectAll()
66  End Sub
67
68  Private Sub txt2Donut_MouseClick(sender As Object, e As MouseEventArgs) Handles txt2Donut.MouseClick
69      txt2Donut.SelectAll()
70  End Sub
71
72  Private Sub txt3Coffie_MouseClick(sender As Object, e As MouseEventArgs) Handles txt3Coffee.MouseClick
73      txt3Coffee.SelectAll()
74  End Sub
75 End Class
```

02.Crispies Bagels and Bites



Customer order:

|                                                 |            |                                                         |
|-------------------------------------------------|------------|---------------------------------------------------------|
| <u>Bagels:</u> <input type="text" value="10"/>  | (á \$0.99) | <u>Subtotal:</u> <input type="text" value="\$ 29.40"/>  |
| <u>Donuts:</u> <input type="text" value="10"/>  | (á \$0.75) | <u>Salex tax:</u> <input type="text" value="\$ 1.76"/>  |
| <u>Coffees:</u> <input type="text" value="10"/> | (á \$1.20) | <u>Total due:</u> <input type="text" value="\$ 31.16"/> |

**Calculate**    **Exit**

$$\begin{aligned}10 * .99 &= 9.9 \\10 * .75 &= 7.5 \\10 * 1.2 &= 12 \\&= 29.40 \\ \text{Subtotal} \\ \text{Sales Tax (6\%)}: 29.40 * 0.06 &= 1.764 \\ \text{Total: } 29.4 + 1.764 &= 31.164\end{aligned}$$

02.Crispies Bagels and Bites



Customer order:

|                                                |            |                                                        |
|------------------------------------------------|------------|--------------------------------------------------------|
| <u>Bagels:</u> <input type="text" value="1"/>  | (á \$0.99) | <u>Subtotal:</u> <input type="text" value="\$ 0.99"/>  |
| <u>Donuts:</u> <input type="text" value="0"/>  | (á \$0.75) | <u>Salex tax:</u> <input type="text" value="\$ 0.06"/> |
| <u>Coffees:</u> <input type="text" value="0"/> | (á \$1.20) | <u>Total due:</u> <input type="text" value="\$ 1.05"/> |

**Calculate**    **Exit**

02.Crispies Bagels and Bites



Customer order:

|                                                |            |                                                 |
|------------------------------------------------|------------|-------------------------------------------------|
| <u>Bagels:</u> <input type="text" value="10"/> | (á \$0.99) | <u>Subtotal:</u> <input type="text" value=""/>  |
| <u>Donuts:</u> <input type="text" value="10"/> | (á \$0.75) | <u>Salex tax:</u> <input type="text" value=""/> |
| <u>Coffees:</u> <input type="text" value="0"/> | (á \$1.20) | <u>Total due:</u> <input type="text" value=""/> |

**Calculate**    **Exit**

# The Selection Structure & operators & (chk) & (rad) & KeyPress

selection structures: **If...Then...Else...ElseIf** & **Select Case...To...Is**; nested selection structures;  
 operators: arithmetic & comparison & logical/boolean; string comparisons: methods **ToUpper** & **ToLower** & **Trim**; Check Box (chk);  
 Radio Button (rad); GroupBox tool; event procedure **txt\_KeyPress** & **e.KeyChar** & **e.Handled = True** & **ControlChars.Back**;  
 arithmetic assignment operators **+=** etc...; using **TryParse** and its return **Boolean** value to validate numbers;  
 4 common errors with selection structures; **If...Then...Else** s.s. example: swapping numeric values

## 3 basic control structures:

- **1. Sequence structure** = CH1, CH2, CH3
- **2. Selection structure** = If...Then...else, Select Case
- **3. Repetition structure** = For...Next, Loop

## Operators precedence:

- |           |                      |                                               |
|-----------|----------------------|-----------------------------------------------|
| #\$ 7:    | comparison operators | - can override with ()<br><, <=, >, >=, =, <> |
| #\$ 8-11: | logical operators    | Not, AndAlso, OrElse, Xor                     |

## CH4\_FOCUS ON THE CONCEPTS LESSON

- CH4\_F1 - 3 basic **control Structures** - **Sequence** structure, **Selection** structure, **Repetition** structure
- CH4\_F2 - **If...Then...Else** selection structure statement:
- CH4\_F3 - **Comparison Operators** (#\$ = 7): =, >, >=, <, <=, <>
- CH4\_F4 - **Logical - Boolean Operators** (#\$8-11): **Not**, **And**, **AndAlso**, **Or**, **OrElse**, **Xor**
- CH4\_F5 - **Summary of Operators**: **arithmetic** (#\$1-6), **comparison** (#\$7), **logical** (#\$8-11)
- CH4\_F6 - **String Comparisons**: methods **ToUpper** & **ToLower** & **Trim**
- CH4\_F7 - **Nested** selection structures aka s.s. **Nested Inside** other s.s.
- CH4\_F8 - **If...Then...Else** s.s.: **Multiple-Alternative** aka **Extended** selection structures using **ElseIf**
- CH4\_F9 - **Select Case** selection structure statement:
- CH4\_F9.1 - **Select Case** s.s. - specifying a range of values using keywords: **To**, **Is**

## CH4\_APPLY THE CONCEPTS LESSON

- CH4\_A1 - **Check Box (chk)** : add it to a Form
- CH4\_A2 - **Check Box (chk)** : code a **GUI** that contains it
- CH4\_A2.1 - **Check Box (chk)** : **chk\_CheckedChanged** event
- CH4\_A3 - **Radio Button (rad)** : add it to a Form
- CH4\_A4 - **Radio Button (rad)** : code a **GUI** that contains it
- CH4\_A4.1 - **Radio Button (rad)** : **rad\_CheckedChanged** event
- CH4\_A4.2 - **Radio Button (rad)** : **rad.Checked** compare using s.s. **Select Case** vs **If...Then...Else**
- CH4\_A5 - group Objects using a **GroupBox** tool/control located at: **Toolbox / Containers / GroupBox**
- CH4\_A6 - **Professionalize your app's GUI**: code the event procedure: **txt\_KeyPress** to restrict user from entering specified characters using s.s. & constant **ControlChars.Back**
- CH4\_A7 - **Professionalize your code** using **arithmetic assignment operators (=)**: **+=**, **-=**, **\*=**, **/=**
- CH4\_A8 - using **TryParse** to validate data - additional topic from **Appendix B** - using a return Boolean value to validate numbers
- CH4\_A9 - 4 **common errors** made when writing **Selection Structures** - additional topic from **Appendix B**
- CH4\_A10 - **If...Then...Else** selection structure example: swapping numeric values - additional topic from **Appendix B**

**CH4\_Summary**

**CH4\_Key Terms**

**CH4\_Exercises**

## CH4\_FOCUS ON THE CONCEPTS LESSON :

### CH4\_F1 - 3 basic control Structures - Sequence structure, Selection structure, Repetition structure

- every procedure in an application is written using one or more of **3 basic control structures:**

- **1. Sequence structure**
- **2. Selection structure**
- **3. Repetition structure**

= processed in the order they are written; same like CH1, CH2, CH3  
= If...Then...Else, Select Case  
= Do...Loop, For...Next -> CH4

- control structure = control the order in which a procedure's instructions are processed

**1. Sequence structure:** Me.Close()      lblShow.BackColor = SystemColors.Control      lblShow.Text = String.Empty

- the procedures in previous CH1-CH3 used the sequence structure only
- when one of the procedures was invoked during run time, the PC processed its instructions in the order they appeared in the procedure - sequentially
- every procedure you write will contain the sequence structure

**2. Selection structure:** IF...THEN...ELSE statement (instruction); SELECT CASE...TO...IS

- many times a procedure will need to use the selection structure
- it tells the PC that it needs to make a decision before it can select the next instruction to process
- the decision is based on a condition specified at the beginning of the selection structure, and the next instruction to process is based on the result of that decision

#### Conditions:

- the condition in a selection structure must be phrased so that it evaluates to an answer of either **True** or **False**
- **a single-alternative** selection structure requires 1 or more actions to be taken only when its condition evaluates to **True**
- **a dual-alternative** requires:
  - 1 set of instructions to be followed only when the condition is **True**
  - different set of instructions only when it is **False**
- the instructions to follow when the condition evaluates to true are called the **true path**
  - begins with the instruction immediately below the IF and ends with either ELSE (if there is one) or the END IF
- the instructions to follow when the condition evaluates to false are called the **false path**
  - begins with the instruction immediately below the ELSE and ends with the END IF

Single-alternative = only **True**  
Dual-alternative = **True & False**

**Figure 4-1 & Figure 4-2** - shows examples of 2 different types:  
- single-alternative  
- dual-alternative  
- in the example pseudocode and flowchart is **condition** shaded:  
- the **diamond** in a flowchart **Figure 4-2** is called **decision symbol** because it is used to represent the selection structure's condition (decision)

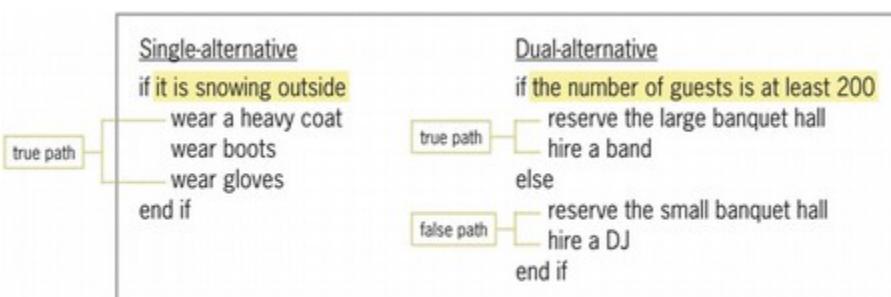


Figure 4-1 Examples of selection structures written in pseudocode

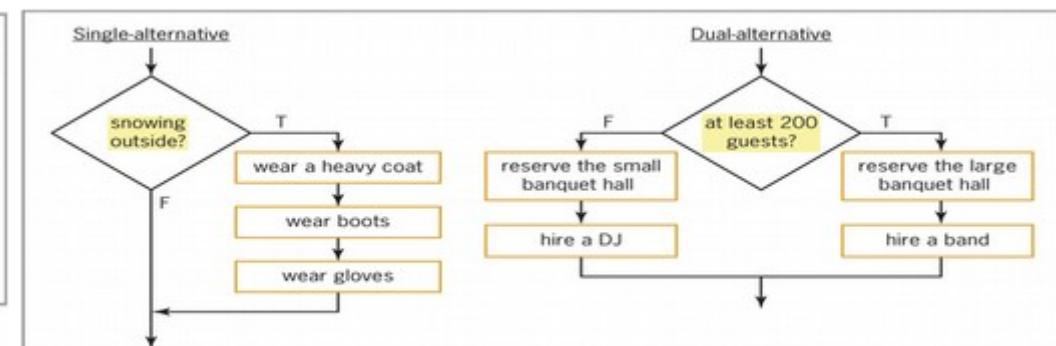


Figure 4-2 Examples of selection structures drawn in flowchart form

## CH4\_F2 - If...Then...Else selection structure statement:

- used for coding single-alternative & dual-alternative **2. Selection structure**

### basic syntax:

```
If condition Then (Enter)  
    true path  
Else  
    false path  
End If
```

**condition** - must be a Boolean expression    **True / False**

- in addition to providing the condition,  
the programmer must provide the statements to be processed in the **True** path and optionally in the **False** path
- set of statements contained in path is referred to as a statement block (terminated by an **Else**, **End If**, **Loop**, **Next**)
- you will learn about the **Loop** & **Next** clauses in Chapter 5
- can contain: variables, named constants
- can contain: literals, properties, methods, keywords,
- can contain: CH3 - **arithmetic operators**, F3 - **comparison operators**, F4 - **logical operators**

evaluated **first**

evaluated **second**

evaluated **third**

- in my IDE: don't forget to tap Enter or it can show an Error !!!

## CH4\_F3 - Comparison Operators (#\$ = 7): =, >, >=, <, <=, <>

= relational operators

#\$ = 7      - evaluated AFTER any **arithmetic** operators !!!

| #\$ = 7 | operator: | operation:               | example:                                                        |
|---------|-----------|--------------------------|-----------------------------------------------------------------|
|         | =         | equal to                 | If blnIsStoned = True Then      - same like If blnIsStoned Then |
|         | >         | greater than             | If decHours > 40 Then                                           |
|         | >=        | greater than or equal to | If decMax >= 75.65D Then                                        |
|         | <         | less than                | If intOnHand < intOrder Then                                    |
|         | <=        | less than or equal to    | If dblTotal <= 999.99 Then                                      |
|         | <>        | not equal to             | If strContinue <> "N" Then                                      |

- do not have an order of precedence (unlike arithmetic operators), the computer evaluates from LEFT to RIGHT

- evaluated after any arithmetic operators:

|            |                   |                     |
|------------|-------------------|---------------------|
| Example 1: | expression:       | 14 / 2 < 15 - 2 * 3 |
|            | 1. division       | 14 / 2 = 7          |
|            | 2. multiplication | 2 * 3 = 6           |
|            | 3. subtraction    | 15 - 6              |
|            | 4. < comparison   | 7 < 9               |
|            | answer :          | <b>True</b>         |

|            |                         |                    |
|------------|-------------------------|--------------------|
| Example 2: | expression:             | 6 * 2 + 3 >= 5 * 4 |
|            | 1. multiplication left  | 6 * 2 = 12         |
|            | 2. multiplication right | 5 * 4 = 20         |
|            | 3. addition             | 12 + 3 = 15        |
|            | 4. >= comparison        | 15 >= 20           |
|            | answer :                | <b>False</b>       |

### Example 3: single-alternative selection structure

#### Total Due Application

- the app displays the total amount a customer owes, which is based on the number of items purchased
- each item costs \$8.50, however the customer receives a 10% discount when the number of items purchased is at least 5
- to code and then test the app:
  1. open: ..VB2017\Chap04\Exercise\Total Due Solution\Total Due Solution.sln
  2. in a code editor window locate the: **btnCalc\_Click** procedure  
and enter the single-alternative selection structure  
shown in **Figure 4-6**
- notice that the Code editor automatically indents (*odsadit*)  
the instructions in the **True** path for you
- the 2 statements in the selection structure's **True** path will  
be processed only when the number purchased, which is  
stored in the intPurchased variable, is at least 5
- "at least" means equal to or greater than
- the 2 instructions will be skipped over when the number  
purchased is less than 5
- 3. test:
  - enter 3, the result should be \$25.50 (3\*8.50)
  - enter 7, the result should be \$53.55 (7\*8.50-5.95 discount)

```
10  Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles
11    ' Calculate and display the total due.
12
13    Const dblUNIT_PRICE As Double = 8.5
14    Const dblDISCOUNT_RATE As Double = 0.1
15    Dim intPurchased As Integer
16    Dim dblTotalDue As Double
17    Dim dblDiscount As Double
18
19    Integer.TryParse(txtPurchased.Text, intPurchased)
20
21    dblTotalDue = intPurchased * dblUNIT_PRICE
22    ' If the number purchased is at least 5, give the discount.
23    If intPurchased >= 5 Then
24      dblDiscount = dblTotalDue * dblDISCOUNT_RATE
25      dblTotalDue = dblTotalDue - dblDiscount
26    End If
27
28    lblTotalDue.Text = dblTotalDue.ToString("C2")
29  End Sub
```

Figure 4-6 Single-alternative selection structure entered in the procedure

### Example 4: - dual-alternative selection structure - Comparison Operator Example:

#### Net Income / Loss Application

- the app calculates either a company's net income or its net loss
- it displays the net income using a **black font** and  
net loss using a **red font**

1. open: ..VB2017\Chap04\Exercise\Net Solution\Net Solution.sln
2. locate the **btnCalc\_Click** procedure and enter the  
dual-alternative selection structure shown in **Figure 4-9**
- notice that the Code Editor automatically indents the  
instructions in both paths for you
- the statement in the selection structure's **True path** will be  
processed only when the decNet variable's value is  
less than 0
- otherwise (when variable's value is equal to or greater than 0)  
the statement in the **False path** will be processed
3. test:
  - type 15000 and 9500, Calculate, result is \$5,500.00
  - type 15000 and 15750, Calculate, result is -\$750.00  
or (\$750.00) depending on System settings !

```
10  Private Sub btnCalc_Click(sender As Object, e As EventArgs)
11    ' Calculate and display net income or net loss.
12
13    Dim decIncome As Decimal
14    Dim decExpenses As Decimal
15    Dim decNet As Decimal
16
17    Decimal.TryParse(txtIncome.Text, decIncome)
18    Decimal.TryParse(txtExpenses.Text, decExpenses)
19
20    decNet = decIncome - decExpenses
21    ' Change font color to red if it's a net loss;
22    ' otherwise, change font color to black.
23    If decNet < 0 Then
24      lblNet.ForeColor = Color.Red
25    Else
26      lblNet.ForeColor = Color.Black
27    End If
28    lblNet.Text = decNet.ToString("C2")
29  End Sub
```

Figure 4-9 Dual-alternative selection structure entered in the procedure

## CH4\_F4 - Logical - Boolean Operators (#\$8-11): Not, And, AndAlso, Or, OrElse, Xor

#\$ = 8-11 - evaluated AFTER any arithmetic and comparison operators !!!

- can be also included in an IF...THEN...ELSE statement's condition
- evaluated ALWAYS after any arithmetic and comparison operators in an expression
- except for the Not operator, all of the logical operators allow you to combine 2 or more conditions, called subconditions, into one compound condition
- this book uses only the Not, AndAlso, OrElse operators, but you should familiarize yourself with the And, Or, Xor operators because you may encounter them when modifying another programmer's code
- if you are unsure whether to use AndAlso or OrElse in an If clause, test the selection structure using both operators because only the correct operator will give you the expected result (see You Do It 3)

#\$ = precedence = the order in which the computer performs the operation in an expression

| #\$ | book operator | - operation info             | Truth table:     |                  |
|-----|---------------|------------------------------|------------------|------------------|
|     | old operator  | If = exempli gratia = id est | A=subcondition 1 | B=subcondition 2 |

|   |                          |                                             |                  |                 |                    |
|---|--------------------------|---------------------------------------------|------------------|-----------------|--------------------|
| 8 | Not                      | - reverses the TRUTH-value of the condition | <- can't combine | condition value | NOT condition val. |
|   | If Not blnSenior Then... | (same like If blnSenior = False Then...)    |                  | True            | FALSE              |

= condition evaluates to TRUE when blnSenior is FALSE

|   |                                                      |                                                                                                                    |                       |       |                 |             |
|---|------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------|-----------------------|-------|-----------------|-------------|
| 9 | And                                                  | - all subconditions must be TRUE to evaluate TRUE                                                                  | <- compound condition | A:    | B:              | A AndAlso B |
|   | AndAlso                                              | - checks both conditions, not efficient                                                                            |                       | True  | True            | TRUE        |
|   |                                                      | - same, except performs short-circuit evaluation, i.e.:                                                            |                       | True  | False           | FALSE       |
|   |                                                      | - checks the first condition and if it is a FALSE, it doesn't check Second                                         |                       | False | (not evaluated) | FALSE       |
|   | If dblRate > 0 AndAlso dblRate < 0.15 Then...        | = the compound condition evaluates to TRUE when value in the dblRate is between mentioned numbers                  |                       | False | False           | FALSE       |
|   | If strCode = "1" AndAlso decSales > 4999.99D Then... | = TRUE when strCode contains the string "1" and at the same time the value in the decSales is greater than 4999.99 |                       |       |                 |             |

|    |                                                     |                                                                           |                       |       |                 |            |
|----|-----------------------------------------------------|---------------------------------------------------------------------------|-----------------------|-------|-----------------|------------|
| 10 | Or                                                  | - only 1 condition needs to be TRUE to evaluate TRUE                      | <- compound condition | A:    | B:              | A OrElse B |
|    | OrElse                                              | - checks both conditions, not efficient                                   |                       | True  | (not evaluated) | TRUE       |
|    |                                                     | - same, except performs short-circuit evaluation, i.e.:                   |                       | False | True            | TRUE       |
|    |                                                     | - checks the first condition and if it is a TRUE, it doesn't check Second |                       | True  | True            | TRUE       |
|    | If strCode = "1" OrElse decSales > 4999.99D Then... | = TRUE when strCode=1 or when decSales is greater than 4999.99            |                       | False | False           | FALSE      |

|    |     |                                                                    |                       |       |       |         |
|----|-----|--------------------------------------------------------------------|-----------------------|-------|-------|---------|
| 11 | Xor | - one and only 1 of the subconditions can be TRUE to evaluate TRUE | <- compound condition | A:    | B:    | A Xor B |
|    |     | If strCoupon1 = "USE" Xor strCoupon2 = "USE" Then...               |                       | True  | True  | FALSE   |
|    |     | = TRUE when only one of the variables contains the string USE      |                       | True  | False | TRUE    |
|    |     |                                                                    |                       | False | True  | TRUE    |
|    |     |                                                                    |                       | False | False | FALSE   |

## Logical Operator Example: Gross Pay Calculator Application

example of using: **AndAlso** & **OrElse**

- the Gross Pay Calculator app calculates and displays an employee's weekly gross pay, given the number of hours worked and the hourly pay rate
- the number of hours worked must be greater than 0 but less than or equal to 40
- if the number of hours worked is not valid, the app should display N/A (for Not Available)

1. open: ..VB2017\Chap04\_Exercise\Gross Solution\Gross Solution.sln
2. locate the **btnAndAlso\_Click** procedure and click the **blank line** above the **End Sub** clause
3. enter the dual-alternative selection structure shown in **Figure 4-14**
  - the statements in the selection structure's **True path** will be processed only when the value in the **dblHours** variable is  $>$  then 0 and, at the same time,  $\leq$  to 40
  - the statement in the **False path** will be processed either when the value is less than or equal to 0 or when it is greater than 40

```
10 Private Sub btnAndAlso_Click(sender As Object, e As EventArgs)
11     ' Calculate and display weekly gross pay.
12
13     Dim dblHours As Double
14     Dim dblRate As Double
15     Dim dblGross As Double
16
17     Double.TryParse(txtHours.Text, dblHours)
18     Double.TryParse(txtRate.Text, dblRate)
19
20     If dblHours > 0 AndAlso dblHours <= 40 Then
21         dblGross = dblHours * dblRate
22         lblGross.Text = dblGross.ToString("C2")
23     Else
24         lblGross.Text = "N/A"
25     End If
26 End Sub
```

Figure 4-14 Dual-alternative selection structure entered in the btnAndAlso procedure

4. locate the **btnOrElse\_Click** procedure and click the **blank line** above the **End Sub** clause
5. enter the dual-alternative selection structure shown in **Figure 4-15**
  - the statement in the selection structure's **True path** will be processed either when the value in the **dblHours** variable is  $<$  then or  $=$  to 0 or when it is  $>$  then 40
  - the statements in the **False path** will be processed only when the value is  $>$  then 0 and, at the same time,  $<$  then or  $=$  to 40
6. test the **btnAndAlso\_Click** procedure using a valid and invalid numbers:
  - type **10** and **8** and click **Calculate-AndAlso** button. **Result = \$80.00**
  - type **43** and **8** and click **Calculate-AndAlso** button. **Result = N/A**
7. test the **btnOrElse\_Click** procedure using a valid and invalid numbers:
  - type **10** and **8** and click **Calculate-AndAlso** button. **Result = \$80.00**
  - type **43** and **8** and click **Calculate-AndAlso** button. **Result = N/A**

```
28 Private Sub btnOrElse_Click(sender As Object, e As EventArgs)
29     ' Calculate and display weekly gross pay.
30
31     Dim dblHours As Double
32     Dim dblRate As Double
33     Dim dblGross As Double
34
35     Double.TryParse(txtHours.Text, dblHours)
36     Double.TryParse(txtRate.Text, dblRate)
37
38     If dblHours <= 0 OrElse dblHours > 40 Then
39         lblGross.Text = "N/A"
40     Else
41         dblGross = dblHours * dblRate
42         lblGross.Text = dblGross.ToString("C2")
43     End If
44 End Sub
```

Figure 4-15 Dual-alternative selection structure entered in the btnOrElse procedure

### You Do It 3: Logical Operator Example: Gross Pay Calculator Application

example of using: **AndAlso** & **OrElse**

- if you are unsure whether to use **AndAlso** or **OrElse** in an **If** clause, test the selection structure using both operators because only the correct operator will give you the expected result

1. copy & rename folder : ..VB2017\Chap04\\_Exercise\Gross Solution into **You Do It 3 Solution**
2. open: **Gross Solution.sln**
3. in the **btnAndAlso\_Click** procedure, change **AndAlso** to **OrElse**
4. test the app, type **10** and **8**, click the **Calculate-AndAlso** button
5. what appears in the **Gross pay** box? Is this value correct? **YES**
6. test the app, change the number of hours worked to **43** and then click the **Calculate-AndAlso** button
7. what appears in the **Gross pay** box? Is this value correct? **NO**
8. Exit, in the **btnAndAlso\_Click** procedure, change back **OrElse** to **AndAlso**
  
9. in the **btnOrElse\_Click** procedure, change **OrElse** to **AndAlso**
10. test the app, type **10** and **8**, click the **Calculate-OrElse** button
11. what appears in the **Gross pay** box? Is this value correct? **YES**
12. test the app, change the number of hours worked to **43** and then click the **Calculate-OrElse** button
13. what appears in the **Gross pay** box? Is this value correct? **NO**
14. Exit, in the **btnOrElse\_Click** procedure, change back **AndAlso** to **OrElse**
15. Finish

### CH4\_F5 - Summary of Operators: arithmetic (#\$1-6), comparison (#\$7), logical (#\$8-11)

arithmetic - comparison - logical  
1 - 6      7      8 - 11

= global precedence number of operators I have learned so far

|            | <b>#\$</b> | <b>operator</b>                                  | <b>operation</b>                                                | <b>example</b>                                                             | <b>info</b>                                                                                                                            |
|------------|------------|--------------------------------------------------|-----------------------------------------------------------------|----------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------|
| arithmetic | 1.         | <b>^</b>                                         | exponentiation                                                  | <code>dblPI * dblRadius ^</code>                                           | - like $3.14159 * dblRadius^2$                                                                                                         |
|            | 2.         | <b>-</b>                                         | negation                                                        | $8 = -8$                                                                   | - unary operator (requires only one operand)                                                                                           |
|            | 3.         | <b>*</b> , <b>/</b>                              | multiplication, division                                        | $2 * 4 / 2 = 4$                                                            | - same #\$, left to right                                                                                                              |
|            | 4.         | <b>\</b>                                         | integer division                                                | $11 \ 3 = 3$                                                               | - divides two only integers and returns only integer                                                                                   |
|            | 5.         | <b>Mod</b>                                       | modulus arithmetic                                              | $25 \text{ Mod } 2 = 1; 26 \text{ Mod } 2 = 0$                             | - used to find out if number is odd or even                                                                                            |
|            | 6.         | <b>+, -</b>                                      | addition, subtraction                                           | $40 - 30 + 10 = 20$                                                        | - same #\$, left to right                                                                                                              |
| comparison | 7.         | <b>=, &gt;, &gt;=,<br/>&lt;, &lt;=, &lt;&gt;</b> | comparison                                                      | <b>If</b> <code>dblNumber &gt;= 22 Then...</code>                          |                                                                                                                                        |
| logical    | 8.         | <b>Not</b>                                       | reverses the TRUTH value ( <b>can't combine</b> )               | <b>If Not</b> <code>blnStoned Then...</code>                               | - same like: If <code>blnStoned = False Then...</code>                                                                                 |
|            | 9.         | <b>AndAlso, And</b>                              | all subconditions<br><b>must</b> be TRUE                        | <b>If</b> <code>dblA &gt; 0 AndAlso<br/>dblA &lt; 100 Then...</code>       | - TRUE if <code>dblA</code> is between 0 and 100<br>- <b>AndAlso</b> - if the 1 <sup>st</sup> is FALSE, it won't check 2 <sup>nd</sup> |
|            | 10.        | <b>OrElse, Or</b>                                | only <b>one</b> of the subconditions<br><b>needs</b> to be TRUE | <b>If</b> <code>blnCode = True OrElse<br/>decSale &gt; 150D Then...</code> | - True if only one condition is True<br>- <b>OrElse</b> - if the 1 <sup>st</sup> is TRUE, it won't check 2 <sup>nd</sup>               |
|            | 11.        | <b>Xor</b>                                       | only <b>one</b> of the subconditions<br><b>can</b> be TRUE      | <b>If</b> <code>intAge &gt; 65 Xor<br/>strPension = "Yes" Then...</code>   | - one or other or other or other....must be TRUE to evaluate TRUE                                                                      |

you can use parentheses to override the order of precedence

|                              |                                           |
|------------------------------|-------------------------------------------|
| - Global precedence example: | $30 > 75 / 3$ <b>AndAlso</b> $5 < 10 * 2$ |
| <b>evaluation steps:</b>     | <b>result:</b>                            |
| 1. $75 / 3 = 25$             | $30 > 25$ <b>AndAlso</b> $5 < 10 * 2$     |
| 2. $10 * 2 = 20$             | $30 > 25$ <b>AndAlso</b> $5 < 20$         |
| 3. $30 > 25 = \text{True}$   | true <b>AndAlso</b> $5 < 20$              |
| 4. $5 < 20 = \text{True}$    | true <b>AndAlso</b> true                  |
| 5. True <b>AndAlso</b> True  | true                                      |

|                                                                         |       |
|-------------------------------------------------------------------------|-------|
| - Mini-Quiz 4-5:                                                        |       |
| - evaluate the following expressions:                                   |       |
| 1. $6 / 2 + 7 - 5 > 4$ <b>AndAlso</b> $7 > 3$                           | true  |
| 2. $8 < 12 / 2$ <b>OrElse</b> $4 > 9 \text{ Mod } 3 * 4$                | false |
| 3. $13 \text{ Mod } 3 * 2 \leq 1$ <b>AndAlso</b> $9 \setminus 4 \leq 2$ | true  |

## CH4\_F6 - String Comparisons: methods ToUpper & ToLower & Trim

- **ToUpper & ToLower methods changes only Temp copy !!!**
- **Trim method changes only Temp copy !!!**
- case sensitive, id est the uppercase version of a letter and its lowercase counterpart are not interchangeable
- each character on the computer keyboard is stored using a different **Unicode** value in the computer's RAM
- **Unicode** = universal coding scheme for characters, and it assigns a unique value to each character used in the written language of the world
- more about Unicode: unicode.org

- example: - the Unicode value for the uppercase letter K is **004B**  
                   - the Unicode value for the lowercase letter k is **006B**  
                   - since both Unicode values are not the same, the condition "K" = "k" evaluates to **FALSE**

- example: - the condition **strState = "Ky"** will evaluate to **TRUE** only when the **strState** variable contains the two letters "Ky"  
                   - it will evaluate to **False** when the variable contains anything other then Ky, such as KY, kY, ky, Ks, La.....and so on

- string comparisons that involve user input can be problematic because the user might enter the string using any combination of uppercase and lowercase letters
- you can avoid the comparison problem by using either the: **ToUpper** method or the: **ToLower** method to temporarily convert the string what can be compared

**basic syntax:**

**string.ToUpper**  
**string.ToLower**

e.g.1: **If strState.ToUpper = "KY" Then...**

- temporarily converts the contents of the **strState** variable to uppercase and then compares the result with the uppercase letters KY

e.g.2: **If strName1.ToLower = strName2.ToLower Then...**

- temporarily converts the contents of the **strName1** & **strName2** variables to lowercase and then compares both results

e.g.3: **lblState.Text = strState.ToUpper**

- temporarily converts the contents of the **strState** variable to uppercase and then assigns the result to the **lblState.Text** property

e.g.4: **strName = strName.ToUpper**

**txtState.Text = txtState.Text.ToLower**

- temporarily changes the contents of the **strName** variable to uppercase and

temporarily changes the contents of the **txtState.Text** property to lowercase

- in each syntax, **string** is usually either the name of a **String variable** or the **Text property** of an object
- both methods copy the contents of the **string** to a temp location in RAM
- the methods convert the temp string to the appropriate case (if necessary) and then return the temporary string
- keep in mind that the **ToUpper & ToLower** methods do **NOT change** the contents of the **original string**; they change only the copy stored in **temp** location
- in addition, the ToUpper & ToLower methods affect only letters of the alphabet (because they are the only characters that have uppercase and lowercase form)
- when using the ToUpper method, be sure that everything you are comparing is UPPERCASE, otherwise the comparison will not evaluate correctly
- likewise when using the ToLower method - be sure that everything you are comparing is lowercase

### basic syntax:

**string.Trim**

- when the users are typing a value in a **txt**, they sometimes inadvertently include one or more space characters at either the beginning or the end of the entry and if the control's Text property is subsequently (následně) used in a string comparison, the extra space will cause the comparison to evaluate incorrectly
- the **Trim** method can appear either before or after the **ToUpper** method

e.g.1: **If txtState.Text.Trim.ToUpper = "LA" Then...**

- id est:
- 1. copies the Text property's value to a temp location in RAM, then
  - 2. removes any leading and trailing spaces from the copy, then
  - 3. converts the copy to uppercase, then
  - 4. compares the result to the string "LA"

**txtState.Text  
.Trim  
.ToUpper  
= "LA" Then**

e.g.2: **strCity = txtCity.Text.Trim.ToUpper**

- id est:
- 1. copies the Text property's value to a temp location in RAM, then
  - 2. removes any leading and trailing spaces from the copy, then
  - 3. converts the copy to uppercase, then
  - 4. assigns the result to the strVariable

e.g.3: **txtName.Text = txtName.Text.Trim** = assignment statement removes extra space characters from the original string

- id est:
- 1. copies the Text property's value to a temp location in RAM, then
  - 2. removes any leading and trailing spaces from the copy, then
  - 3. assigns the result to the Text property

- in the syntax, **string** is usually either the name of a String variable or the Text property of an object

- the **Trim** method copies the contents of the string to a temp location in RAM

- it then removes any leading and trailing spaces from the copy and returns a string with the appropriate characters removed

- the **Trim** method does **NOT remove** any characters from the original string, but to do then:

- you would need to use the assignment statement like:

e.g.3: **txtName.Text = txtName.Text.Trim**

### String Comparison example: Shipping app

- a customer placing an order online can get free shipping by entering the FREESHIP code during checkout

- without the code, the shipping is \$5

- look at the 3 ways of writing the IF...Then...Else statements:

e.g.1: - using the Trim and ToUpper methods in the condition:

```
If txtCode.Text.Trim.ToUpper = "FREESHIP" Then
    intShipping = 0
Else
    intShipping = 5
End If
```

e.g.2: - using the Trim and ToLower methods in the condition:

```
If txtCode.Text.Trim.ToLower = "freeship" Then
    intShipping = 0
Else
    intShipping = 5
End If
```

e.g.3: - assigning the result of the Trim and ToUpper methods to a variable:

```
strCode = txtCode.Text.Trim.ToUpper
If strCode = "FREESHIP" Then
    intShipping = 0
Else
    intShipping = 5
End If
```

- 1. open the ..VB2017\Chap04\\_Exercise\Shipping Solution\Shipping Solution.sln
- 2. in a code editor locate the **btnDisplay\_Click** procedure, and enter the dual-alternative selection structure

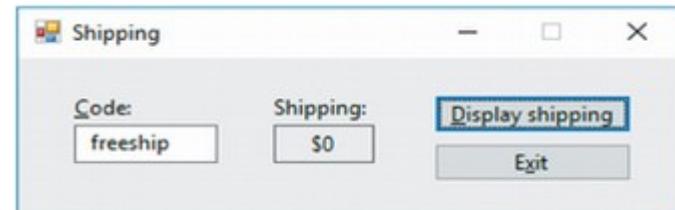
**Figure 4-22**

```

10  Private Sub btnDisplay_Click(sender As Object, e As Eve
11      ' Determine and display the shipping charge.
12
13      Dim intShipping As Integer
14
15      If txtCode.Text.Trim.ToUpper = "FREESHIP" Then
16          intShipping = 0
17      Else
18          intShipping = 5
19      End If
20      lblShipping.Text = intShipping.ToString("C0")
21  End Sub

```

**Figure 4-22** Selection structure using the Trim and ToUpper methods

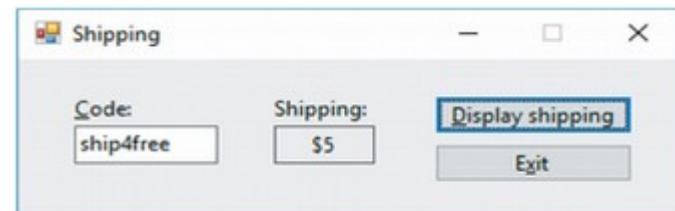


**Figure 4-23** Result of entering the valid free shipping code

- 3. test it:
  1. press the **SPACEBAR** twice, type **freeship**, press **SPACEBAR** 3x, and click the Display shipping button **Figure 4-23**
  - the result is correct = the shipping charge is \$0
  - entered valid free shipping code

2. change the contents of the Code box to **ship4free**, and click the Display shipping button **Figure 4-24**
- the result is incorrect = the shipping charge is \$5
- entered invalid free shipping code

- 4. exit



**Figure 4-24** Result of entering an invalid code

#### CH4\_F7 - Nested selection structures aka s.s. **Nested Inside** other s.s.

- both paths (true path, false path) in a selection structure can include instructions that declare variables, perform calculations, and so on...
- both can also include other selection structures - inner selection structure = **Nested Selection Structure**

example: pseudocode & flowchart for the **btnDisplay\_Click** procedure

in the Voter Eligibility app **Figure 4-25 & Figure 4-26**

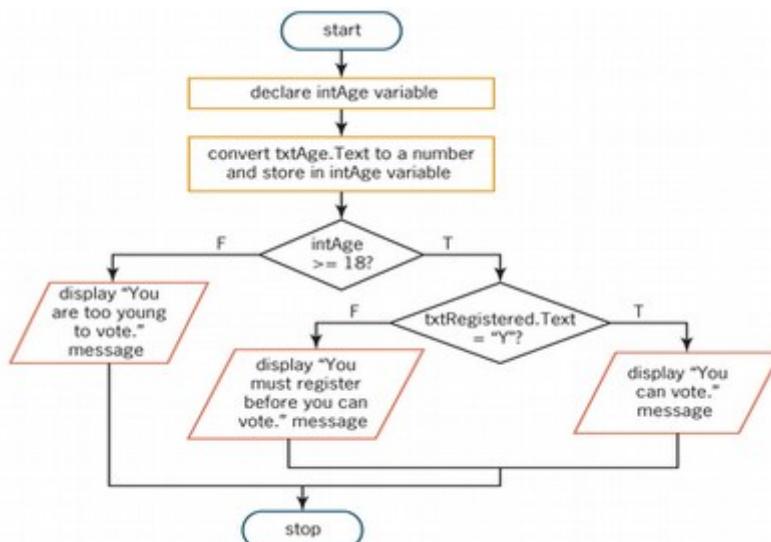
- the procedure contains:
  1. an **outer** s.s.: **intAge >= 18**
  2. a **nested** s.s.: **txtRegistered.Text = "Y"**
- the s.s. determine whether a person can vote and then display 1 of 3 messages
- the appropriate msg depends on the person's age and voter registration status
- 1. if the person is younger than 18, the **OUTER** s.s. **false path** displays:  
" You are too young to vote " , otherwise:
- if the person is at least 18, the **NESTED** s.s. displays 1 of 2 messages
- 2. if the person is registered, the **NESTED** s.s. **true path** displays:  
" You can vote " , otherwise:
- 3. if the person is not registered, the **NESTED** s.s. **false path** displays:  
" You must register before you can vote "

- after the appropriate msg is displayed, the **NESTED** & **OUTER** s.s. end
- notice that the **NESTED** selection structure is processed only when the **OUTER** structure's condition evaluates to **True**
- as you can see, determining the person's voter reg. status is important only after the age is determined i.e.:
  - decision regarding the age is considered the Primary decision
  - depending on status of Primary is registration status as Secondary decision
- **OUTER** s.s. = Primary decision    **NESTED** s.s. = Secondary decision
- even small procedures can be written in more than one way: **Figure 4-26**
- an **outer** s.s.: **intAge < 18**
- neither version of the procedure is better than the other

**Figure 4-25:** Pseudocode and flowchart for the `btnDisplay_Click` procedure

#### `btnDisplay_Click`

1. declare intAge variable for the age
2. convert txtAge.Text property to a number and store it in the intAge variable
3. if the value in the intAge variable is at least 18
  - a. if txtRegistered.Text contains "Y"
    - b. display "You can vote." message
  - b. else
    - c. display "You must register before you can vote." message
- d. end if
- e. else
  - f. display "You are too young to vote." message
- g. end if



```

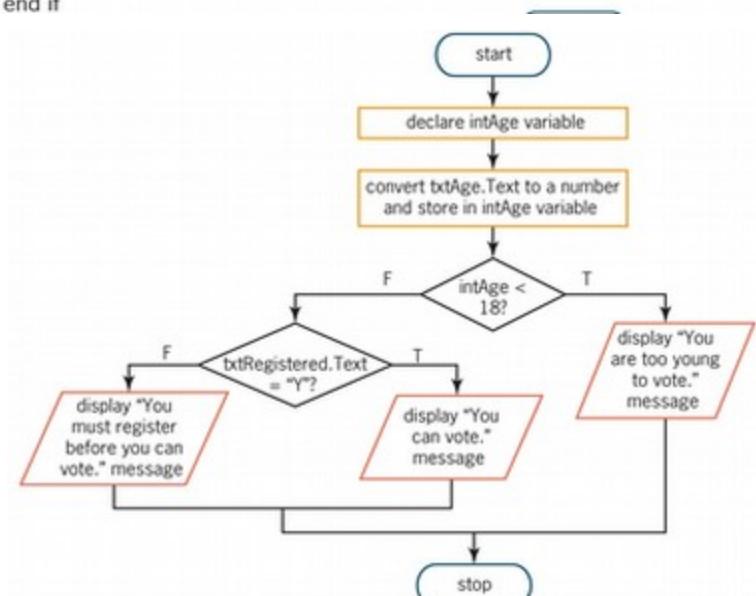
11 Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles
12   ' Displays a message indicating whether a person can vote.
13
14   Dim intAge As Integer
15
16   Integer.TryParse(txtAge.Text, intAge)
17   ' Determine age.
18   If intAge >= 18 Then
19     ' Determine registration status.
20     If txtRegistered.Text.Trim.ToUpper = "Y" Then
21       lblMsg.Text = "You can vote."
22     Else
23       lblMsg.Text = "You must register before you can vote."
24     End If
25   Else
26     lblMsg.Text = "You are too young to vote."
27   End If
28 End Sub
  
```

- nested in the outer selection structure's **true** path

**Figure 4-26:** another version of the [Figure 4-25](#)

#### `btnDisplay_Click`

1. declare intAge variable for the age
2. convert txtAge.Text property to a number and store it in the intAge variable
3. if the value in the intAge variable is less than 18
  - a. display "You are too young to vote." message
- b. else
  - c. if txtRegistered.Text contains "Y"
    - d. display "You can vote." message
  - e. else
    - f. display "You must register before you can vote." message
- g. end if



```

11 Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles
12   ' Displays a message indicating whether a person can vote.
13
14   Dim intAge As Integer
15
16   Integer.TryParse(txtAge.Text, intAge)
17   ' Determine age.
18   If intAge < 18 Then
19     lblMsg.Text = "You are too young to vote."
20   Else
21     ' Determine registration status.
22     If txtRegistered.Text.Trim.ToUpper = "Y" Then
23       lblMsg.Text = "You can vote."
24     Else
25       lblMsg.Text = "You must register before you can vote."
26     End If
27   End If
28 End Sub
  
```

- nested in the outer selection structure's **false** path

- 1. open the ..VB2017\Chap04\\_Exercise\Voter Slution\Voter Solution.sln
- 2. in a code editor locate the **btnDisplay\_Click** procedure, and enter the s.s. of your choice from the examples **Figure 4-25** and **Figure 4-26**
- 3. test: - enter 27 -> Yes      btn Display = "You can vote"  
- enter 27 -> No      btn Display = "You must register before you can vote"  
- enter 17 ->      btn Display = "You are too young to vote"

#### You Do It 5: test the **OUTER & NESTED** selection structure

- 1. open the ..VB2017\Chap04\\_Exercise\You Do it 5 Slution\You Do it 5 Solution.sln
- 2. in the code editor locate the **btnDisplay\_Click** procedure
  - the procedure should display the shipping charge, which is based on the info shown below
  - complete the procedure by entering the appropriate **OUTER** and **NESTED** selection structures
  - info: - purchase amount at least \$100: \$4.99 shipping  
- purchase amount less than \$100: \$10.99 shipping  
- FREESHIP shipping code: free shipping on any purchase amount

my code:

```
If txtCode.Text.Trim.ToUpper = "FREESHIP" Then
    dblShip = 0
Else
    If dblPurchase >= 100 Then
        dblShip = 4.99
    Else
        dblShip = 10.99
    End If
End If
```

#### CH4\_F8 - **If...Then...Else** s.s: **Multiple-Alternative aka Extended** selection structures using **ElseIf**

- some procedures require a selection structure that can choose from several different alternatives

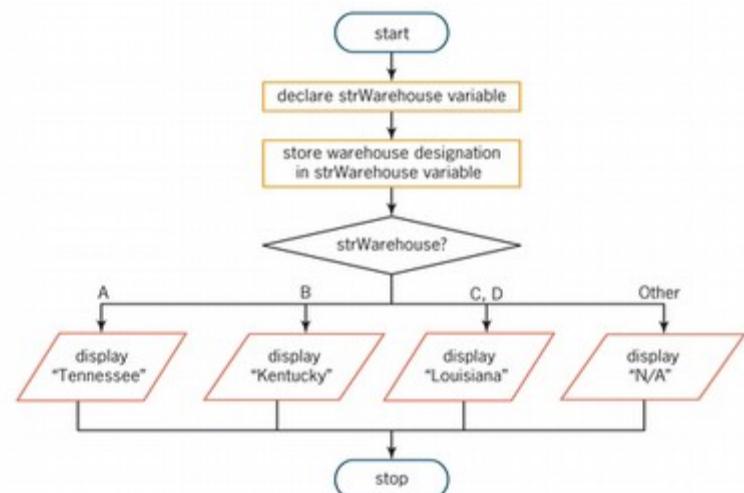
e.g.: **Figure 4-29** & **Figure 4-30**

- shows the pseudocode & flowchart for a procedure that uses a multiple-alternative (extended) selection structure to display a message
- the appropriate msg to display is based on the warehouse designation entered by the user
- the diamond in the flowchart represents the condition in the multiple-alternative s.s., the same shape like the condition in single/dual alternative selection structure - BUT: this one has several flowlines leading out of the symbol
  - each flowline represents a possible path and
  - must be marked appropriately, indicating the values necessary for the path to be chosen

**Figure 4-30** shows 2 versions; both use If...Then...Else statement

- both produce the same result, but **Version 2** provides a more convenient way

1. declare strWarehouse variable for the warehouse designation  
2. store the warehouse designation in the strWarehouse variable  
3. if the value in the strWarehouse variable is one of the following:  
 A      display "Tennessee"  
 B      display "Kentucky"  
 C, D    display "Louisiana"  
 else     display "N/A"  
 end if



**Figure 4-29** Pseudocode and flowchart containing a multiple-alternative selection structure

### Version 1

```
14      Dim strWarehouse As String  
15  
16  
17      If strWarehouse = "A" Then  
18          lblLocation.Text = "Tennessee"  
19      Else  
20          If strWarehouse = "B" Then  
21              lblLocation.Text = "Kentucky"  
22          Else  
23              If strWarehouse = "C" OrElse strWarehouse = "D" Then  
24                  lblLocation.Text = "Louisiana"  
25              Else  
26                  lblLocation.Text = "N/A"  
27              End If  
28          End If  
29      End If
```

you get here when the code is not "A"

you get here when the code is not "A" and not "B"

you get here when the code is not "A", "B", "C", or "D"

three End If clauses are required

### Version 2

```
14      Dim strWarehouse As String  
15  
16  
17      If strWarehouse = "A" Then  
18          lblLocation.Text = "Tennessee"  
19      ElseIf strWarehouse = "B" Then  
20          lblLocation.Text = "Kentucky"  
21      ElseIf strWarehouse = "C" OrElse strWarehouse = "D" Then  
22          lblLocation.Text = "Louisiana"  
23      Else  
24          lblLocation.Text = "N/A"  
25      End If
```

only one End If clause is required

Figure 4-30 Two versions of the code containing a multiple-alternative selection structure

- 1. open the: VB2017\Chap04\\_Exercise\Warehouse Solution-If\Warehouse Solution.sln
- 2. in a code editor locate the btnDisplay\_Click procedure, and enter the selection structure shown in Version 2 in Figure 4-30
- 3. test it: type b, a, d, k....

You Do It 6: test by Redbullself the **Multiple-alternative** selection structure:

- 1. create an app named You Do It 6
- 2. add a text box, a label, and a button
- 3. button's Click event procedure should display either price of a ticket or N/A
- 4. ticket price is based on the code entered in the text box:  
1,2 = \$15; 3 = \$25; 4,5 = \$35.

- I tried to use a different Framework 4.6.2 then the default one:  
.....its ok it seems

my code:

```
Option Explicit On
Option Strict On
Option Infer Off

Public Class frmMain
    Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
        Me.Close()
    End Sub

    Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
        'Dim strCode As String = strCode.Trim
        'that is a test if i can do it
        'Dim strCode As String
        'strCode = strCode.Text.Trim 'not sure
        'but why to convert? No NEED TO, Redby
        txtCode.Text = txtCode.Text.Trim
        If txtCode.Text = "1" OrElse txtCode.Text = "2" Then
            lblDisplay.Text = "$15"
        ElseIf txtCode.Text = "3" Then
            lblDisplay.Text = "$25"
        ElseIf txtCode.Text = "4" OrElse txtCode.Text = "5" Then
            lblDisplay.Text = "$35"
        Else
            lblDisplay.Text = "N/A"
        End If
    End Sub

    Private Sub txtCode_Enter(sender As Object, e As EventArgs) Handles txtCode.Enter
        txtCode.SelectAll()
    End Sub

    Private Sub txtCode_TextChanged(sender As Object, e As EventArgs) Handles txtCode.TextChanged
        lblDisplay.Text = String.Empty
    End Sub
End Class
```

#### CH4\_F9 - Select Case selection structure statement:

- when a multiple-alternative s.s. has many paths from which to choose, it is often simpler and clearer to code the s.s. using the SELECT CASE statement

##### basic syntax:

```
Select Case selectorExpression
    Case expressionList1
        instructions for the first Case
    Case expressionList2, expressionList3
        instructions for the second Case
    Case expressionListXYZ
        instructions for the XYZ Case
    Case Else
        instructions for when the selectorExpression
        does not match any of expressionLists
End Select
```

- *selectorExpression* =can contain any combination of: -----
  - variables
  - named constants
  - literals
  - keywords
  - functions
  - methods
  - operators
  - properties
- *Case* =can have as many as necessary
- *expressionList* =each Case clause must contain
  - =can include 1 or more expressions-(use comma) - it needs to match only one of the expressions
  - =data type must be compatible with the data type of the selectorExpression
- *Case Else* =if includes, must be the last

- when processing, the computer compares the values from the **BEGINNING** (or from the **LEFT**) and if matches, it **SKIPS** the others
- if the **selectorExpression** matches a value in **more** than **ONE** Case clause, only the instructions in the **FIRST** match's Case clause are **processed**
- the data type of the **expressions** must be **compatible** with the data type of the **selectorExpression**

use the Select Case in the Warehouse Location app:

Figure 4-32

- 1. open the ..VB2017\Chap04\\_Exercise\Warehouse Slution-Select Case\Warehouse Solution.sln
- 2. in a code editor locate the **btnDisplay\_Click** procedure, and enter the Select Case statement shown in **Figure 4-32**
- 3. test it: type b, a, d, k...

Figure 4-32 example from: Figure 4-30

```
14 Dim strWarehouse As String
15
16 strWarehouse = txtDesignation.Text.Trim.ToUpper
17 Select Case strWarehouse
18     Case "A"
19         lblLocation.Text = "Tennessee"
20     Case "B"
21         lblLocation.Text = "Kentucky"
22     Case "C", "D"
23         lblLocation.Text = "Louisiana"
24     Case Else
25         lblLocation.Text = "N/A"
26 End Select
```

- just to compare with a **If...Then...Else** statement

```
14 Dim strWarehouse As String
15
16 strWarehouse = txtDesignation.Text.Trim.ToUpper
17 If strWarehouse = "A" Then
18     lblLocation.Text = "Tennessee"
19 ElseIf strWarehouse = "B" Then
20     lblLocation.Text = "Kentucky"
21 ElseIf strWarehouse = "C" orElse strWarehouse = "D" Then
22     lblLocation.Text = "Louisiana"
23 Else
24     lblLocation.Text = "N/A"
25 End If
26
```

#### CH4\_F9.1 - Select Case s.s. - specifying a range of values using keywords: To, Is

- in addition to specifying one or more discrete values in a Case clause, you can also specify a range of values (like 1 through 4; or values > then 10;...)
- use the keywords either:
  - **To** = use when you know both the smaller and higher values in the range
  - **Is** = when you know only one end of the range (either the smaller or higher end)

##### basic syntax:

```
Case smallest value in the range      To      largest value in the range  
or  
Case Is  comparisonOperator   value
```

- *comparisonOperator* = **#\$ 7:** <, <=, >, >=, =, <>

- be sure to test your code thoroughly because the computer will not display an error when you switch values in **To** - it will just give an incorrect result

e.g.: The ABC Corporation's price chart:

| quantity ordered: | price per item: |
|-------------------|-----------------|
| less than 1       | \$0             |
| 1 - 5             | \$25            |
| 6 - 10            | \$23            |
| more than 10      | \$20            |

```
Select Case intQuantity
    Case 1 To 5
        intPrice = 25
    Case 6 To 10
        intPrice = 23
    Case Is > 10
        intPrice = 20
    Case Else
        intPrice = 0
End Select
```

= 1, 2, 3, 4, 5 - from lower to upper value in a range  
= 6, 7, 8, 9, 10 - from lower to upper value in a range  
= greater than 10; you can also write: **Case Is >=11**  
- **Is** keyword always used in combination with one of the **comparison operators**  
= any other unspecified values

code and then test the ABC Corporation app:

- 1. open the ..VB2017\Chap04\\_Exercise\ABC Slution\ABC Solution.sln
- 2. in a code editor locate the **btnDisplay\_Click** procedure, and enter the Select Case statement shown above
- 3. test it: type 9 and it should show \$23.....test other numbers too, my dear

### You Do It 7: test by yourself the Select Case statement with specifying a range:

- 1. create an app named You Do It 7
- 2. add a text box, a label, and a button
- 3. button's Click event procedure should display:  
either price of a ticket or N/A
- 4. ticket price is based on the code entered in the text box:  
 $1,2 = \$15; 3 = \$25; 4,5 = \$35; \text{other} = \text{N/A}$

- my code without čistírna:

```
Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
    Dim intCode As Integer
    'Dim intDisplay As Integer
    txtCode.Text = txtCode.Text.Trim
    Integer.TryParse(txtCode.Text, intCode)

    Select Case intCode
        Case 1 To 2
            lblDisplay.Text = "$15"
        Case Is = 3
            lblDisplay.Text = "$25"
        Case 4 To 5
            lblDisplay.Text = "$35"
        Case Else
            lblDisplay.Text = "N/A"
    End Select
End Sub
```

### CH4\_APPLY THE CONCEPTS LESSON:

#### CH4\_A1 - Check Box (chk) : add it to a Form

- CheckBox tool in the toolbox to add a check box control to a form
- provides an option to select or not to select
- if more than one, the option offered by each must be unique & unrelated to any other check box option
- each check box is independent from any other check box, i.e. user can select any number of them on a form at the same time

- most commonly used properties:

|                |                                                                                                                                  |
|----------------|----------------------------------------------------------------------------------------------------------------------------------|
| <b>Checked</b> | = selected or unselected                                                                                                         |
| <b>Font</b>    | = font to use for text                                                                                                           |
| <b>(Name)</b>  | = <b>chk</b>                                                                                                                     |
| <b>Text</b>    | = text that appears in the check box<br>= use <b>Sentence</b> capitalization<br>= include a unique access key ( <b>ALT</b> key ) |

**chkXYZ.Checked**

- when a **chk** is selected, its Checked property contains the Boolean value **TRUE**; otherwise **FALSE**

- you can use a **selection structure** to determine the value in a **check box's Checked property** and then take the appropriate action based on the result

## CH4\_A2 - Check Box (chk) : code a GUI that contains it

e.g. Seminars app using chk: **Figure 4-37 & Figure 4-38**

- the app offers the user 3 different seminars
- each seminar occurs on a different day, so the user can select any of them or none

to complete the **btnCalc\_Click** procedure:

1. ...Exercise\Seminars Solution-CheckBox\ Seminars Solution.sln
2. locate the **btnCalc\_Click** procedure and
  - the procedure will use 3 single-alternative s.s. to determine which (if any) of the 3 check boxes are selected

- if a s.s's condition evaluates to TRUE, the instruction in its true path will add the seminar's fee to the amount due

- enter the 3 s.s. shown in **Figure 4-38**

3. test it: click the Health (\$125) chk

**- NOTICE** that \$0 still appears in the Amount due box, which could be misleading -  
- you will fix this problem in the next section

4. test it: click any of them, Calculate, deselect any, Calculate.....

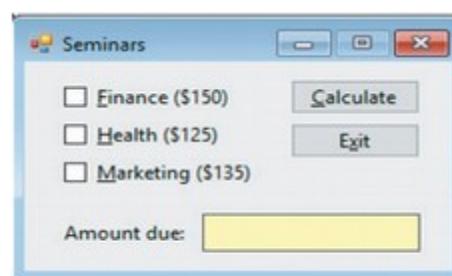


Figure 4-37 Seminars application using check boxes

```
10 Private Sub btnCalc_Click(sender As Object, e As EventArgs)
11     ' Calculate and display the amount due.
12
13     Dim intAmountDue As Integer
14
15     ' Determine which (if any) check boxes are selected
16     ' and add the associated fee to the amount due.
17     If chkFinance.Checked = True Then
18         intAmountDue = intAmountDue + 150
19     End If
20
21     If chkHealth.Checked = True Then
22         intAmountDue = intAmountDue + 125
23     End If
24
25     If chkMarketing.Checked = True Then
26         intAmountDue = intAmountDue + 135
27     End If
28
29     lblAmountDue.Text = intAmountDue.ToString("C0")
30 End Sub
```

Figure 4-38 Selection structures entered in the btnCalc\_Click procedure

## CH4\_A2.1 - Check Box (chk) : chk\_CheckedChanged event

- a CheckedChanged event occurs when the value in its Checked property changes,
  - i.e. when user selects a check box, because doing so changes the chk box's Checked property from FALSE to TRUE
  - deselecting a chk box changes its Checked property from TRUE to FALSE - thereby (čímž) invoking its CheckedChanged event
- in the next set of steps, you will code each chk's CheckedChanged procedure to clear the contents of lbl's when the event occurs:

e.g.: code each **chk**'s CheckedChanged procedure: **Figure 4-39**

1. ...Exercise\Seminars Solution-CheckBox\ Seminars Solution.sln

2. enter the assignment statements in all 3 procedures: (e.g. **chkHealth\_CheckedChanged**)
- choose **chkFinance / CheckedChanged** and enter  
**lblAmountDue.Text = String.Empty**

3. test it

```
36 Private Sub chkFinance_CheckedChanged(sender As Object, e As EventArgs)
37     lblAmountDue.Text = String.Empty
38
39 End Sub
40
41 Private Sub chkHealth_CheckedChanged(sender As Object, e As EventArgs)
42     lblAmountDue.Text = String.Empty
43
44 End Sub
45
46 Private Sub chkMarketing_CheckedChanged(sender As Object, e As EventArgs)
47     lblAmountDue.Text = String.Empty
48
49 End Sub
```

enter the assignment statement in these CheckedChanged procedures

Figure 4-39 Assignment statements entered in the CheckedChanged procedures

#### CH4\_A3 - Radio Button (rad) : add it to a Form

- RadioButton tool in the toolbox to add a radio button control to a form
- allow you to limit the user only one choice from a group of related but mutually exclusive options
- the group must contain at least 2 rad - because the only way to deselect a rad is to select a different one
- each rad in an interface should be labeled so the user knows the choice it represents

- most commonly used properties:

|                |                                                  |
|----------------|--------------------------------------------------|
| <b>Checked</b> | = selected or unselected                         |
| <b>Font</b>    | = font to use for text                           |
| <b>(Name)</b>  | = <b>rad</b>                                     |
| <b>Text</b>    | = text that appears in the radio button          |
|                | = use <b>Sentence</b> capitalization             |
|                | = include a unique access key ( <b>ALT</b> key ) |

- same like in chk - when a rad is selected, its Checked property contains the Boolean value **TRUE**; otherwise **FALSE**
- you can use a selection structure to determine the value in a check box's Checked property and then take the appropriate action based on the result
- it is customary in Win apps to have one of the rad in a group already selected = called the **default radio button**
  - either first one or
  - the user's most likely choice

#### CH4\_A4 - Radio Button (rad) : code a GUI that contains it

- you will code a different version of the Seminars app, then in the CH4\_A2
- this version uses rad (rather than chk) to offer the user 3 different seminar options
- each seminar in this version actually occurs at the same time, so the user will be allowed to select only 1 of the options
- when the user clicks the Calculate button, the btn's Click event procedure will determine which rad is selected and then assign & display the appropriate amount due

e.g.: to begin completing this version of the app: **Figure 4-41 & Figure 4-42**

1. ... Exercise\Seminars Solution-RadioButton-If\Seminars Solution.sln

2. first, you gonna designate a default rad - in a designer window click

the Finance rad and set its Checked property to **TRUE**

(a colored dot appears inside the circle - **Figure 4-41**)

3. open the code editor window and locate the **btnCalc\_Click** proced.

- the procedure will use a multiple-alternative s.s. to determine which of the three rad is selected

- enter the s.s. shown in: **Figure 4-42**

4. test it: click Health (\$125) - computer selects the Health rad as it deselects the Finance

**- NOTICE that \$150 still appears in the Amount due box, which could be misleading - you will fix this problem in the next section**

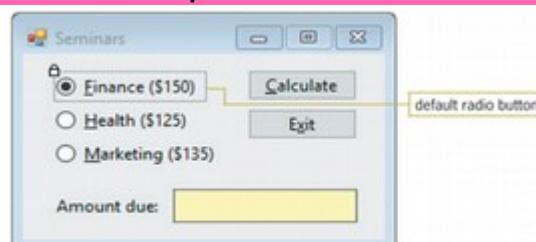


Figure 4-41 Default radio button in the group

```
10  Private Sub btnCalc_Click(sender As Object, e As EventArgs
11      ' Display the amount due.
12
13      Dim intAmountDue As Integer
14
15      ' Determine which radio button is selected
16      ' and assign the associated fee.
17      If radFinance.Checked = True Then
18          intAmountDue = 150
19      ElseIf radHealth.Checked = True Then
20          intAmountDue = 125
21      ElseIf radMarketing.Checked = True Then
22          intAmountDue = 135
23      End If
24
25      lblAmountDue.Text = intAmountDue.ToString("C0")
26  End Sub
```

Figure 4-42 Selection structure entered in the **btnCalc\_Click** procedure

enter this multiple-alternative selection structure

#### CH4\_A4.1 - Radio Button (rad) : rad\_CheckedChanged event

- a radio button's **CheckedChanged** event occurs when the value in its **Checked** property changes
- e.g. when user selects a **rad**, its **Checked** property changes from **FALSE** to **TRUE**, invoking its **CheckedChanged** event
- in addition, the **Checked** property of the previously selected **rad** in the group changes from **TRUE** to **FALSE**, thereby invoking that **rad**'s **CheckedChanged** event
- in the next set of steps, you will code each **rad**'s **CheckedChanged** procedure to clear the contents of **lbl**'s when the event occurs:

e.g.: code each **rad**'s **CheckedChanged** procedure:

Figure 4-43

- its actually same like **chk** čistírna

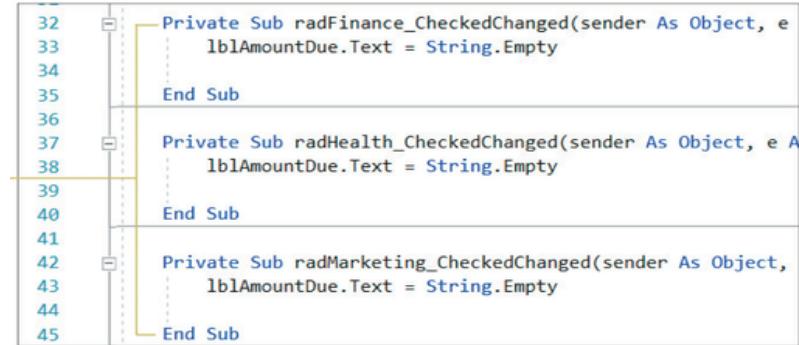
1. ...Exercise\Seminars Solution-RadioButton-If\Seminars Solution.sln

2. enter the assignment statements in all 3 procedures:

(radFinance, radHealth, radMarketing)

- choose **radFinance / CheckedChanged** and enter:  
**lblAmountDue.Text = String.Empty**

3. test it



```
32 Private Sub radFinance_CheckedChanged(sender As Object, e As EventArgs)
33     lblAmountDue.Text = String.Empty
34 End Sub
35
36 Private Sub radHealth_CheckedChanged(sender As Object, e As EventArgs)
37     lblAmountDue.Text = String.Empty
38 End Sub
39
40 Private Sub radMarketing_CheckedChanged(sender As Object, e As EventArgs)
41     lblAmountDue.Text = String.Empty
42 End Sub
43
44
45
```

Figure 4-43 Assignment statements entered in the three CheckedChanged procedures

#### CH4\_A4.2 - Radio Button (rad) : rad.Checked compare using s.s. **Select Case** vs **If...Then...Else**

- you gonna use the **Select Case** statement, rather then **If...Then...Else** statement, to code the **btnCalc\_Click** procedure

e.g.: to use the **Select Case** statement in the **btnCalc\_Click** procedure:

Figure 4-44

1. ...Exercise\Seminars Solution-RadioButton-Select Case\Seminars Solution.sln

2. open the code editor window, locate the **btnCalc\_Click** procedure and enter the **Select Case** statement:

Figure 4-44: **Select Case** statement

< compare >

```
Dim intAmountDue As Integer
Select Case True
    Case radFinance.Checked
        intAmountDue = 150
    Case radHealth.Checked
        intAmountDue = 125
    Case radMarketing.Checked
        intAmountDue = 135
End Select
lblAmountDue.Text = intAmountDue.ToString("C0")
```

Figure 4-42: **If...Then...Else** statement

```
Dim intAmountDue As Integer
If radFinance.Checked = True Then
    intAmountDue = 150
ElseIf radHealth.Checked = True Then
    intAmountDue = 125
ElseIf radMarketing.Checked = True Then
    intAmountDue = 135
End If
lblAmountDue.Text = intAmountDue.ToString("C0")
```

or

If radFinance.Checked Then

## CH4\_A5 - group Objects using a **GroupBox** tool/control located at: **Toolbox / Containers / GroupBox**

- **Toolbox / Containers / GroupBox** tool
- a **GroupBox** serves as a container for other controls and is typically used to visually separate related controls from other controls on the **form**
- you can include an identifying **lbl** on a **GroupBox** by setting the **Text** property,
  - labeling is optional, but if you do - use Sentence capitalization

### **- a **GroupBox** and its controls are treated as 1 unit =>**

=> therefore, when you move or delete a **GroupBox**, the controls inside are also moved or deleted

- the **Seminars** app that you completed in the previous section contained 1 group of **rads**

**- if you need to include 2 groups of rads in an interface, at least 1 of the groups must be placed within a container, such as a **GroupBox** =>**

=> otherwise, the rads are considered to be in the same group and only 1 can be selected at any one time

e.g. : to use a **GroupBox** to group radio buttons: **Figure 4-45**

1. ... Exercise\RadioButton Solution\RadioButton Solution.sln
2. open the **Designer** window - the GUI contains 2 sets of **rads**:
  - the 1st set offers 2 choices : Coffee or Tea
  - the 2nd set offers 3 choices : Small, Medium or Large
3. just **Test** it - click each button and **notice** that **only 1** of them can be **selected** at any one time
4. Exit & come back to Designer window.
5. go to the **Toolbox** window, expand **Containers**, and **drag GroupBox** tool to the upper-left corner of the form and
  - set the group's Text property to: **Type of the drink**
6. drag the **Coffie & Tea rads** into the GroupBox: **Figure 4-45**

**- the GUI now has 2 independent groups of radio buttons**

7. adjust the size of the GroupBox as needed, thirak
8. **before** coding and testing, you better **specify** the **default** button in each group:
  - set the first choices: **Properties : Checked : True**
    - black dot will appear as a response
9. test it - notice, that **rads** in each group works independently from the other one
10. on your own, add another group box named **Size**
11. test and exit

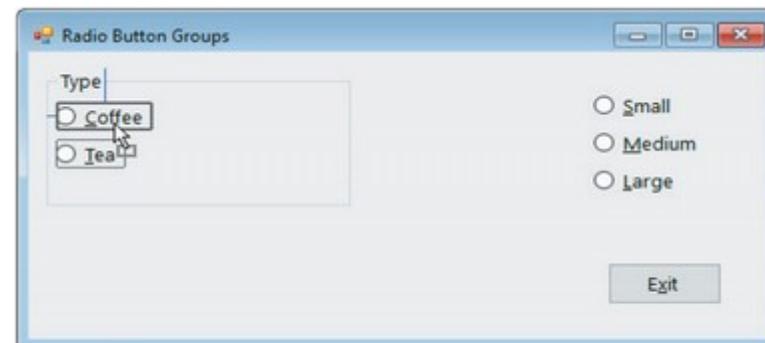


Figure 4-45 Selected controls being dragged into the Type group box

### Mini-Quiz 4-8:

1. If a check box is not selected, what value is contained in its **Checked** property?
2. How do you specify the default radio button?
3. Radio button (rad) & Check box (chk) controls have a **CheckedChanged** event that is invoked when a change is made to their **Checked** property. True or False?
4. A form contains 6 radio buttons. 3 of them are contained in a **GroupBox**. How many of the radio buttons in the GUI can be selected at the same time?

4.2

3. True

2. Properties : Checked = True

1. False

**CH4\_A6 - Professionalize your app's GUI: code the event procedure: txt\_KeyPress to restrict user from entering specified characters using s.s. & constant `ControlChars.Back`**

`txt_KeyPress` = an event, when a key is pressed on a keyboard

`e.KeyChar` = character press on keyboard

`e.Handled` = forbid to use

- in some apps, you can make the GUI more professional-looking by coding a Text Box's KeyPress event procedure to prevent the control from accepting inappropriate characters
- e.g.: if a txt requires a numeric entry, you can tell its KeyPress event procedure to accept only the numbers entered by the user, and to ignore all other characters
- NOT every txt needs a KeyPress event procedure - only when you need to limit the user's entry to specific characters

- a txt's KeyPress event occurs each time the user presses a key while the TextBox has the focus

- as indicated in **Figure 4-47**, the procedure associated with the

KeyPress event has **2 parameters**, which appear within the **parentheses ()** in the procedure header:

- `sender`
- `e`

- a parameter represents info that is passed to the procedure when the event occurs

- when the KeyPress event occurs, a character corresponding to the pressed key is sent to the event's `e` parameter

**- e.g. when the user presses the period (.) while entering data into a txt, the TextBox's KeyPress event occurs and a period is sent to the event's parameter**

- similarly, when the Shift key along with a letter is pressed, the uppercase version of the letter is sent to the `e` parameter

- to prevent a TextBox from accepting an inappropriate character, you FIRST use the `e` parameter's KeyChar property to determine the pressed key

- KeyChar = key character

- SECOND you use the `e` parameter's Handled property to CANCEL the key if it is an inappropriate one

- you CANCEL the key by setting the Handled property to TRUE, like this: `e.Handled = True`

- KeyPress automatically allows the use of the DELETE key for editing

e.g.: using the KeyChar and Handled properties in a TextBox's KeyPress event procedure:

**Figure 4-48**

- e.g.1:

- the condition in s.s. compares the contents of the KeyChar property with a dollar sign \$

- if the condition evaluates to TRUE, the `e.Handled = True` instruction in the s.s.'s TRUE PATH cancels the \$ key before it is entered in the txtSales control

- e.g. 2:

- you can use this s.s. to allow the txtAge control to accept only numbers and the BACKSPACE key (which is used for editing)

- you refer to the BACKSPACE key on your keyboard using Visual Basic's `ControlChar.Back` constant

e.g.1:

```
Private Sub txtSales_KeyPress(sender As Object, e As KeyPress)
    'Prevent the text box from accepting the dollar sign.
    If e.KeyChar = "$" Then
        e.Handled = True
    End If
End Sub
```

e.g.2:

```
Private Sub txtAge_KeyPress(sender As Object, e As KeyPress)
    'Accept only numbers & the BACKSPACE key.
    If (e.KeyChar < "0" orElse e.KeyChar > "9") AndAlso
        e.KeyChar <> ControlChar.Back Then
        e.Handled = True
    End If
End Sub
```

e.g. To code the **txtPurchased\_KeyPress** procedure:

1. open the: ..VB2017\Chap04\Exercise\Total Due Solution-KeyPress\Total Due Solution.sln
2. in a Code editor window open the template for the **txtPurchased\_KeyPress** procedure and enter the code shown in: **Figure 4-49**
3. test it: try typing a letter, a period, \$ - you will not be able to do so
  - type 12 and click the Calculate button - the answer should be \$91.80

**Figure 4-49:**

```
39  Private Sub txtPurchased_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtPurcha...
40      'Allow the TextBox to accept only numbers and the BACKSPACE key.
41
42      If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
43          e.Handled = True
44      End If
45  End Sub
```

#### CH4\_A7 - Professionalize your code using arithmetic assignment operators(=): +=, -=, \*=, /=

+=, -=, \*=, /= instead of intA = intA +,-,\*,/ = prostě zkrátíš opakujícíse

- in addition to the standard arithmetic operators that you learned about in Chapter 3, Visual Basic provides several **arithmetic assignment operators**
- you can use them to abbreviate an assignment statement that contains an arithmetic operator
- however, the assignment statement **MUST HAVE** the following format, in which **variableName** on both sides of the equal sign is the **NAME** of the **SAME** variable:

**variableName = variableName arithmeticOperator value**

- e.g.: you can use the addition assignment operator (=+) to abbreviate the statement: **intNum = intNum + 1** as follows: **intNum += 1**

- both statements tell the computer to add the number 1 to the contents of the **intNum** variable and then store the result in the variable

**syntax:** **variableName arithmeticAssignmentOperator value**      - **arithmeticAssignmentOperator** = **arithmetic operator** with =

**operators:**

**example:**

**abbreviation of:**

|           |                                |                                          |
|-----------|--------------------------------|------------------------------------------|
| <b>+=</b> | <b>intNum += 1</b>             | <b>intNum = intNum + 1</b>               |
| <b>--</b> | <b>decPrice -- decDiscount</b> | <b>decPrice = decPrice - decDiscount</b> |
| <b>*=</b> | <b>dblPrice *= 1.05</b>        | <b>decPrice = decPrice * 1.05</b>        |
| <b>/=</b> | <b>dblNum /= 2</b>             | <b>dblNum = dblNum / 2</b>               |

- it is a common syntax error to include a space between the **operator** and **=**, so be careful Redby

e.g.: to use an assignment operator in the **btnCalc\_Click** procedure:

1. open the: ..VB2017\Chap04\Exercise\Total Due Solution-KeyPress\Total Due Solution.sln
2. locate the line nr. 25: **dblTotalDue = dblTotalDue - dblDiscount**
3. modify the statement into: **dblTotalDue -= dblDiscount**
4. test it: enter the dragon number 10, the result should be \$76.50

#### Mini-Quiz 4-9:

1. which property of the KeyPress procedure's **e** parameter is used to determine the key pressed by the user?
2. which property of the KeyPress procedure's **e** parameter is used to cancel the key pressed by the user?
3. every TextBox's KeyPress event procedure needs to be coded. True or False?
4. to CANCEL a key, you assign what value to the **e** parameter's Handled property?
5. write a statement that uses an arithmetic assignment operator to add the number 1 to the: **intAge** variable

1. e.KeyChar
2. e.Handled = True
3. false
4. True
5. intAge += 1

#### CH4\_A8 - using TryParse to validate data - additional topic from Appendix B - using a return Boolean value to validate numbers

- in Chapter 3, you learned how to use the **TryParse** method to convert a **string** to a **number** of a specified data type

> CH3\_F4 - TryParse method: converts **string** into **numeric** data type - syntax & example  
>  
e.g. **numericDataType.TryParse(string, numericVariable)**  
**Integer.TryParse(txtAge.Text, intAge)**

- recall that:
  - if the conversion is successful, the **TryParse** method stores the number in the variable specified in the method's **numericVariable** argument
  - otherwise / if the conversion is not successful, the **TryParse** method stores the number **0** in the variable

- what you did not learn in CH3\_F4 -
  - in addition to storing a number in the variable, the **TryParse** method also returns a **Boolean value**, that indicates whether the conversion was:
    - a). successful = **True** or
    - b). unsuccessful = **False**
  - you can assign the value returned by the **TryParse** method to a **Boolean** variable as shown in the syntax and example
  - you can then use a selection structure to take the appropriate action based on the result of the conversion

#### TryParse method and its Return Value syntax:

**booleanVariable = numericDataType.TryParse(string, numericVariable)**

e.g.

```
Dim blnIsValid As Boolean
Dim intAge As Integer

blnIsValid = Integer.TryParse(InputBox("How old are you?"), intAge)

If blnIsValid = True Then
    MessageBox.Show("Valid")
Else
    MessageBox.Show("Invalid")
End If
```

| value entered in InputBox: | result of assignment statement in |              |
|----------------------------|-----------------------------------|--------------|
|                            | intAge:                           | blnIsValid:  |
| 12                         | 12                                | <b>True</b>  |
| 25.7                       | 0                                 | <b>False</b> |
| Ab                         | 0                                 | <b>False</b> |
| 25%                        | 0                                 | <b>False</b> |
|                            | 0                                 | <b>False</b> |

- the **TryParse** method in the assignment statement will attempt to convert the string entered in the **InputBox** to a number that has the **Integer** data type
- if the conversion is:
  - a). successful: - the method stores the **Integer** in the **intAge** variable and also
    - returns the Boolean value **True**
  - b). not successful: - the method stores the number **0** in the **intAge** variable and also
    - returns the Boolean value **False**

- to use the **Boolean** value returned by the **TryParse** method:

1). open the: ...VB2017\Chap04\Exercise\TryParse Solution\TryParse Solution.sln

2). open the **Solution Explorer** window and **Designer** window

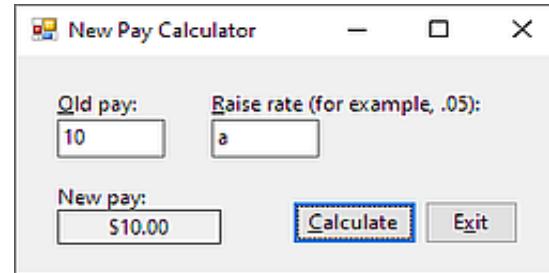
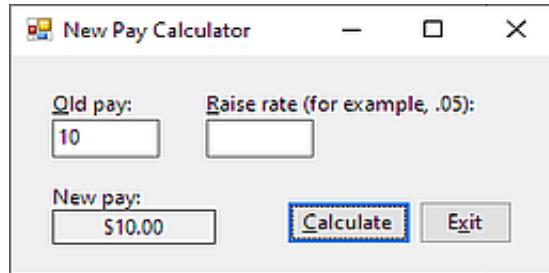
3). open the **Code Editor** window, and locate the **btnCalc\_Click** procedure

<- before modifying the code to use the **Boolean** value returned by the **TryParse** method, you will observe how the procedure currently works, so:

-> start and test the application:

1). -> in the **Old pay:** box type **10**, leave the **Raise rate** box empty and then click the **Calculate** button

<- rather than alerting the user that the **Raise rate** box is empty, the procedure displays in the **New pay:** box the old pay amount of **\$10.00**



2). -> in the **Old pay:** box type **10**, and in the **Raise rate** box type **a** and then click the **Calculate** button

<- even though the **Raise rate** is invalid, the procedure displays in the **New pay:** box the old pay amount of **\$10.00**

-> close the application

4). in the **Code Editor** window, locate and modify the **btnCalc\_Click** procedure:

```
1  ' Name:      New Pay Project
2  ' Purpose:    Calculate an employee's new pay.
3  ' Programmer: <your name> on <current date>
4
5  Option Explicit On
6  Option Strict On
7  Option Infer Off
8
9  Public Class frmMain
10 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
11     Me.Close()
12 End Sub
13
14 Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
15     ' Calculates and displays the new pay.
16
17     Dim dblOld As Double
18     Dim dblRate As Double
19     Dim dblNew As Double
```

```

20
21     ' Convert the input to numbers:
22     Double.TryParse(txtOld.Text, dblOld)
23     Double.TryParse(txtRate.Text, dblRate)
24
25     ' Calculate and display the new pay:
26     dblNew = dblOld + dblOld * dblRate
27     lblNew.Text = dblNew.ToString("C2")
28 End Sub
29 End Class

```

```

20     Dim blnOldOk As Boolean
21     Dim blnRateOk As Boolean
22
23     ' Convert the input to numbers:
24     blnOldOk = Double.TryParse(txtOld.Text, dblOld)
25     blnRateOk = Double.TryParse(txtRate.Text, dblRate)
26
27     ' Determine whether the conversions were successful:
28     If blnOldOk AndAlso blnRateOk Then
29         ' Calculate and display the new pay:
30         dblNew = dblOld + dblOld * dblRate
31         lblNew.Text = dblNew.ToString("C2")
32     Else
33         lblNew.Text = "N/A"
34     End If
35 End Sub
36 End Class

```

5). save the solution and then start & test the application:

1). -> in the Old pay: box type **10** and leave the Raise rate box empty and then click the Calculate button:

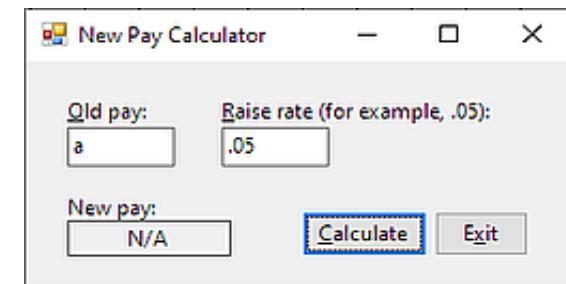
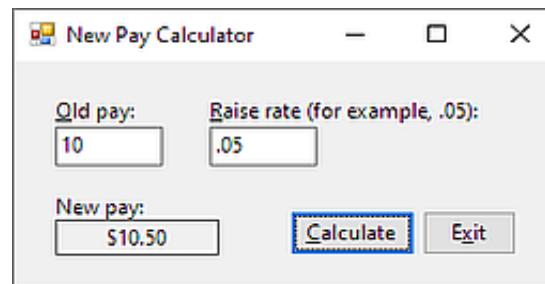
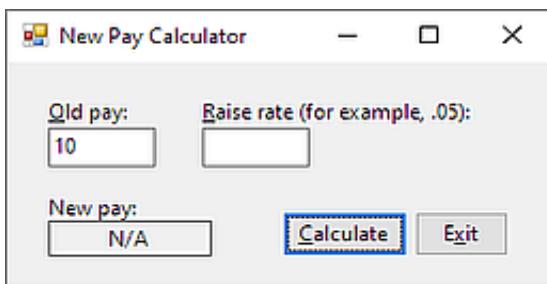
<- because nothing was entered in Raise rate box, the **blnRateOk** evaluates to **False**, so the New pay: box displays **N/A** message, which is correct

2). -> in the Old pay: box type **10** and in the Raise rate box type **.05** and then click the Calculate button:

<- the procedure displays in the New pay: box **\$10.50** as the new pay amount, which is correct

3). -> in the Old pay: box type **a** and in the Raise rate box type **.05** and then click the Calculate button:

<- because the user entry **a** won't much the converted data type **Double**, the **blnOldOk** evaluates to **False**, so the New pay: box displays **N/A** message



## CH4\_A9 - 4 common errors made when writing Selection Structures - additional topic from Appendix B

- it is easier to understand the listed **errors** by viewing them in a procedure, so here are the errors alongside the examples:

- error 1:** - using a compound condition rather than a nested selection structure
- error 2:** - reversing the decisions in the outer and nested selection structures
- error 3:** - using an unnecessary nested selection structure
- error 4:** - including an unnecessary comparison in a condition

- Example 1 & pseudocode e.g. error 1:**
- Example 1 & pseudocode e.g. error 2:**
- Example 1 & pseudocode e.g. error 3:**
- Example 2 & pseudocode e.g. error 4:**

### Example 1 - imagine a procedure that displays the daily fee for renting a car

- the daily fee is **\$55**
  - however, there is an additional charge for renting a **luxury** car:
    - the additional charge, either **\$20** or **\$30**, depends on whether the customer belongs to the **Car Rental Club**
- <- notice that the car's classification determines whether the renter is charged an additional amount:
- if the car is classified as a **luxury** vehicle, then whether the customer is a club **member** determines the appropriate additional amount
  - in this case, the decision regarding the car's classification is the **primary** decision,  
while the decision regarding the customer's membership status is the **secondary** decision

### Example 1 & pseudocode - correct

```
1. daily fee = 55
2. if luxury car
    if club member
        add 20 to the daily fee
    else
        add 30 to the daily fee
    end if
    end if (luxury car)
3. display the daily fee
```

- <- the selection structures indicate that a hierarchy exists between the car classification and the membership decisions
- <- the decision regarding the car classification must be made first
- <- the membership decision is necessary only when the car classification decision evaluates to **True**

### Example 1 & pseudocode e.g. error 1:

```
1. daily fee = 55
2. if luxury car and club member
    add 20 to the daily fee
else
    add 30 to the daily fee
end if
3. display the daily fee
```

- using a compound condition rather than a nested selection structure
- <- the compound condition indicates that only club members who are renting a luxury car are charged **\$20** extra;  
everyone else is charged **\$30** extra
- <- if a customer is renting a standard vehicle, this selection structure will **incorrectly** charge an additional **\$30**

**Example 1 & pseudocode e.g. error 2:**

```

1. daily fee = 55
2. if club member
    if luxury car
        add 20 to the daily fee
    else
        add 30 to the daily fee
    end if
end if (club member)
3. display the daily fee

```

- reversing the decisions in the outer and nested selection structures
- <- the selection structures indicate that a hierarchy exists between the car classification and the membership decisions
- <- the decision regarding the membership must be made first
- <- the car classification decision is necessary only when the membership decision evaluates to **True**
- <- these selection structures will incorrectly charge a club member renting a standard vehicle an extra **\$30**, and and it will not charge anything extra to a nonmember renting a luxury vehicle

**Example 1 & pseudocode e.g. error 3:**

```

1. daily fee = 55
2. if luxury car
    if club member
        add 20 to the daily fee
    else
        if nonmember
            add 30 to the daily fee
        end if
    end if
end if
3. display the daily fee

```

- using an unnecessary nested selection structure
- <- the third selection structure is unnecessary because the second selection structure's condition already determines the membership status
- <- although these selection structures produce the correct results, they do so less efficiently

**Example 2 - imagine a procedure that displays the price of an item**

- the item price is based on the quantity purchased, as shown in the tab below:

| Quantity purchased:     | Price per item: |
|-------------------------|-----------------|
| less than or equal to 0 | \$0.00          |
| 1 - 99                  | \$9.50          |
| 100 or more             | \$7.75          |

**Example 2 & pseudocode - correct**

```

1. if quantity <= 0
    price = 0
else
    if quantity < 100
        price = 9.50
    else
        price = 7.75
    end if
end if
2. display the price

```

**Example 2 & pseudocode e.g. error 4:**

```

1. if quantity <= 0
    price = 0
else
    if quantity > 0 and quantity < 100
        price = 9.50
    else
        price = 7.75
    end if
end if
2. display the price

```

- including an unnecessary comparison in a condition

<- although the selection structures produce the correct results, they do so in a less efficient manner than the ones shown above

<- unnecessary comparison because the quantity has to be greater than 0 for this nested structure to be processed

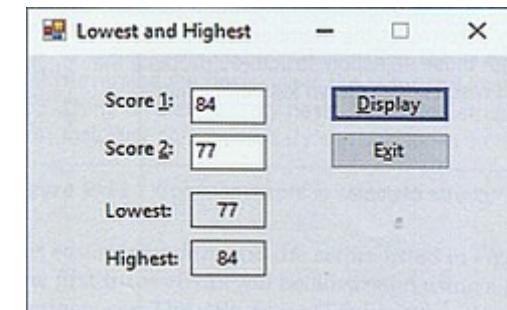
#### CH4\_A10 - If...Then...Else selection structure example: swapping numeric values - additional topic from Appendix B

- an example shows:
  - a sample run of an application that displays the lowest and highest of two scores entered by the user
  - it also shows the code entered in the `btnDisplay_Click` procedure:

- the condition in the `If` clause compares the contents of the `intScore1` variable with the contents of the `intScore2` variable:

- if the value in the `intScore1` variable is greater than the value in the `intScore2` variable, the condition evaluates to `True` and and 3 instructions in the `If...Then...Else` statement's `True` path swap both values
- swapping the values places the smaller number in the `intScore1` variable and places the larger number in the `intScore2` variable
- if the condition evaluates to `False`, on the other hand, the `True` path instructions are skipped over because the `intScore1` variable already contains a number that is smaller than (or possibly equal to) the number stored in the `intScore2` variable

```
14  Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
15      ' Display the lowest and highest scores.
16
17      Dim intScore1 As Integer
18      Dim intScore2 As Integer
19      Dim intTemp As Integer
20
21      Integer.TryParse(txtScore1.Text, intScore1)
22      Integer.TryParse(txtScore2.Text, intScore2)
23
24      ' Swap scores (if necessary):
25      If intScore1 > intScore2 Then
26          intTemp = intScore1
27          intScore1 = intScore2
28          intScore2 = intTemp
29      End If
30
31      ' Display lowest and highest scores:
32      lblLow.Text = intScore1.ToString
33      lblHigh.Text = intScore2.ToString
34  End Sub
```



- the first instruction in the `If...Then...Else` statement's `True` path assigns the `intScore1` variable's value to the `intTemp` variable

<- if you do not store that value in the `intTemp` variable, it will be lost when the computer processes the next statement `intScore1 = intScore2`, which replaces the contents of the `intScore1` variable with the contents of the `intScore2` variable

- finally, the `intScore2 = intTemp` instruction assigns the `intTemp` variable's value to the `intScore2` variable and this completes the swap

- illustration of the swapping concept, assuming the user enters the numbers 84 and 77 in the **Score 1** and **Score 2** boxes:

|                                                                                                               | intScore1 | intScore2 | intTemp |
|---------------------------------------------------------------------------------------------------------------|-----------|-----------|---------|
| values stored in the variables immediately before the <code>intTemp = intScore1</code> statement is processed | 84        | 77        | 0       |
| result of the <code>intTemp = intScore1</code> statement                                                      | 84        | 77        | 84      |
| result of the <code>intScore1 = intScore2</code> statement                                                    | 77        | 77        | 84      |
| result of the <code>intScore2 = intTemp</code> statement                                                      | 77        | 84        | 84      |

## CH4\_Summary

### **Selection structures:**

- you can use the IF...THEN...ELSE statement to code single- & dual-alternative selection structure
- you can use the Comparison operators to compare 2 values (=, >, >=, <, <=, <>)
- to create a Compound condition, use the Logical operators and the TRUTH tables (Not, AndAlso, And, OrElse, Or, Xor)
- the order of precedence when evaluating an expression that contains operators:
  - ## 1-6: arithmetic operators (^, -, \*, /, \, Mod, +-)
  - ## 7: comparison operators (<, <=, =, >, >=, <>)
  - ## 8-11: logical operators (Not, AndAlso And, OrElse Or, Xor)
- to temporarily convert a string to either uppercase or lowercase, use the ToUpper and ToLower methods
- to remove leading and trailing spaces from a string, use the Trim method
- to create a s.s. that evaluates both a primary and a secondary decision, place (nest) the secondary decision's s.s.entirely within either the TRUE or FALSE path of the primary decision's selection structure
- to code a multiple-alternative (Extended) s.s. you can use either If...Then...Else statement (ElseIf) or the Select Case statement
- to specify a range of values in a Select Case statement's Case clause, use the:
  - To keyword when you know both higher and lower values in the range
  - Is keyword when you know only one end of the range (used in combination with one of the Comparison operators: =, >, >=, <, <=, <>)

### **CheckBox & RadioButton tools:**

- use the CheckBox tool (chk) to allow the user to select any number of choices
- use the RadioButton tool (rad) to limit the user to only 1 choice in a group
  - (to include 2 groups of RadioButtons on a form, at least 1 of them must be placed within a container, such as a **GroupBox tool**)
- to determine whether a chk or rad is selected or unselected, use the control's Checked property
  - (the property will contain the Boolean value TRUE if the control is selected; otherwise FALSE)
- control's CheckedChanged event procedure is used to process code when the value in the Checked property changes
- use the GroupBox tool to group controls together by dragging them into the box and to include an optional label, set the Text property

### **profi GUI: TextBox tool - forbit to use a specified keys:**

- to allow a TextBox (txt) to accept only certain keys, code the txt's KeyPress event procedure:
  - e.KeyChar property stores the key the user pressed
  - e.Handled = True statement cancels the key pressed by the user

### **profi code: abbreviate Dim's:**

- you can abbreviate some assignment statements using the arithmetic statement operators (+=, -=, \*=, /=)

## CH4\_Key Terms :

- **And operator** - Logical operator, same as AndAlso but less efficient because it does not perform a short-circuit evaluation
- **AndAlso operator** - Logical operator, performs a short-circuit evaluation
  - when used to combine 2 subconditions, the result compound condition evaluates to TRUE only when both subconditions are TRUE
  - evaluates to FALSE only when one or both of the subconditions are FALSE

- **Arithmetic assignment operators** - used to abbreviate some assignment statements, e.g. A += 3.14 (A = A + 3.14); (+=, -=, \*=, /=)
- **Boolean / Logical operators** - #\\$8-11 - operators used to combine 2 or more subconditions into one compound condition (Not, AndAlso And, OrElse Or, Xor)
- **Check box (chk)** - used in a GUI to offer the user one or more independent and nonexclusive (nevylučujíci se) choices
- **Check box's Checked property** - contains a Boolean value that indicates whether the chk is selected (True) or not selected (False)
- **Check box's CheckedChanged event** - occurs when the value in the chk's Checked property changes
- **Comparison / Relational operators** - #\\$7 - operators used to compare values in an expression (<, <=, >, >=, =; <>)
- **Condition** - specifies the decision that the computer needs to make - must be phrased so that it evaluates to either True or False
- **ControlChars.Back constant** - the Visual Basic constant that represents the Backspace key on your keyboard
- **Decision symbol** - the diamond in a flowchart - used to represent the condition in a selection structure
- **Default Radio button (rad)** - the rad that is automatically selected when the app is started
- **Dual-alternative selection structures** - a s.s. that requires one set of instructions to be followed only when the structure's condition evaluates to True and  
and a different set of instructions to be followed only when the structure's condition evaluates to False
- **Extended / Multiple-alternative selection structure** - s.s. that contain several alternatives - can be coded using either If...Then...Else or Select Case statements
- **False path** - contains the instructions to be processed when a selection structure's condition evaluates to False
- **Group box** - a control (tool) that is used to contain other controls - instantiated using the Toolbox / Containers / GroupBox
- **Handled property** - a property of the KeyPress event procedure's **e** parametr (when assigned the value True, it cancels the key pressed by the user)
- **If...Then...Else statement** - used to code single- / dual- / multiple-alternative selection structures
- **KeyChar property** - a property of the KeyPress event procedure's **e** parameter (stores the character associated with the key pressed by the user)
- **KeyPress event** - occurs each time the user presses a key while a control has the focus
- **Logical / Boolean operators** - #\\$8-11 - operators used to combine 2 or more subconditions into one compound condition (Not, AndAlso And, OrElse Or, Xor)
- **Multiple-alternative / Extended selection structure** - s.s. that contain several alternatives - can be coded using either If...Then...Else or Select Case statements
- **Nested selection structure** - a s.s. that is wholly contained (nested) within either the True path or False path of another selection structure
- **Not operator** - Logical operator, reverses the truth-value of a condition
- **Or operator** - Logical operator, same as the OrElse but less efficient because it does not perform a short-circuit evaluation
- **OrElse operator**
  - Logical operator, performs a short-circuit evaluation
    - when used to combine 2 subconditions, the result compound condition evaluates to TRUE when at least one of them is TRUE
    - evaluates to FALSE only when both of the subconditions are FALSE
- **Parameter** - an item contained within parentheses () in a procedure header - stores information passed to the procedure when the procedure is invoked (sender, e)
- **Radio buttons (rad)** - controls (tools) used to limit the user to only one choice from a group of related but mutually exclusive options
- **Radio button's CheckedChanged event** - occurs when the value in the rad's Checked property changes
- **Radio button's Checked property** - contains a Boolean value that indicates whether the rad is selected (True) or not selected (False)
- **Relational / Comparison operators** - #\\$7 - operators used to compare values in an expression (<, <=, >, >=, =; <>)
- **Select Case statement** - used to code a multiple-alternative / Extended selection structure
- **Selection structure**
  - 2/3 basic control structures, tells the computer to make a decision based on some condition and  
and then select the appropriate action (If...Then...Else; Select Case statements)
- **Sequence structure**
  - 1/3 basic control structures, directs the computer to process a procedure's instructions sequentially,  
id est: in the order they appear in the procedure (e.g. lblShow.Text = String.Empty)
- **Short-circuit evaluation**
  - more efficient than standard - way to evaluate 2 subconditions connected by either the AndAlso or OrElse operator
    - AndAlso - if subcondition1 is False, it does not evaluate subcondition2 anymore (both must be True)
    - OrElse - if subcondition1 is True, it does not evaluate subcondition2 anymore (at least 1 must be True)
- **Single-alternative selection structure** - a s.s. that requires a special set of actions to be performed only when the structure's condition evaluates to True
- **Statement block** - used in selection structure - the set of statements terminated by an Else or End If
- **TryParse method**
  - converts a string to a number of a specific data type
  - returns a Boolean value that indicates whether the conversion was successful - True, or unsuccessful - False

- **ToLower method** - temporarily converts a string to lowercase
- **ToUpper method** - temporarily converts a string to UPPERCASE
- **Trim method** - removes any leading and trailing spaces from a string
- **True path** - contains the instructions to be processed when a selection structure's condition evaluates to True
- **Truth tables** - summarize how the computer evaluates Logical operators in an expression
- **Unicode** - the universal coding scheme that assigns a unique numeric value to each character used in the written languages of the world ([unicode.org](http://unicode.org))
- **Xor operator**
  - Logical operator, used to combine 2 subconditions. The resulting compound condition evaluates to:
  - True when only 1 of the subconditions is True
  - False when both subconditions are either True or False

**EXERCISE 01.States Capitals Solution\_introductory** - rad.Checked, **ElseIf**, rad\_CheckedChanged

**EXERCISE 02.Hales Solution\_introductory** - **Select Case**, **Case** x, y, rad.Checked, chk.Checked, chk\_CheckedChanged, rad.CheckedChanged

**EXERCISE 03.Baxters Solution\_introductory** - chk.Checked, txt\_TextChanged, txt\_Enter, chk\_CheckedChanged, txt\_KeyPress, e.KeyChar, **ControlChars.Back**

**EXERCISE 04.Hales Solution-Select Case\_intermediate** - I used **Select Case** in EXERCISE 02 already, sorry

**EXERCISE 05.Lorenzo Solution\_intermediate** - **ElseIf**, txt\_Enter, txt\_TextChanged, txt\_KeyPress, e.KeyChar, **ControlChars.Back**

**EXERCISE 06.Cars Solution\_intermediate** - txt\_Enter, txt\_TextChanged, txt\_KeyPress, e.KeyChar, **ControlChars.Back**  
**EXERCISE 07.Wedding Solution\_intermediate** - arithmetic operator \ integer division, **ElseIf**, **Mod**, txt\_Enter, txt\_TextChanged, txt\_KeyPress, e.KeyChar, **ControlChars.Back**

**EXERCISE 08.Wedding Solution-Modified\_intermediate** - arithmetic operator \ integer division, **ElseIf**, **Mod**, txt\_Enter, txt\_TextChanged, txt\_KeyPress, e.KeyChar, **ControlChars.Back**

**EXERCISE 09.Software Solution\_intermediate** - **Select Case**, rad.Checked, rad\_CheckedChanged, rad\_CheckedChanged

**EXERCISE 10.Williams Solution\_intermediate** - **Select Case**, rad.Checked, chk.Checked, rad\_CheckedChanged, chk\_CheckedChanged

**EXERCISE 11.Canton Solution\_advanced** - decimal numbers -> number + D -> 3720D  
- **Select Case**, **Case** ... **To** ..., **Case Is** ..., **Case Else**, chk.Checked, chk\_CheckedChanged, txt\_TextChanged, txt\_Enter, txt\_KeyPress, e.KeyChar, **ControlChars.Back**

**EXERCISE 12.Genatone Solution\_advanced** - rad.Checked, **ElseIf**, **AndAlso**, txt\_Enter, txt\_TextChanged, rad\_CheckedChanged, txt\_KeyPress, e.KeyChar, **ControlChars.Back**

**EXERCISE 13.Jacket Haven Solution\_advanced** - decimal numbers -> number + D -> 45.99D  
- **ElseIf** chk.Checked = False **AndAlso** int > 1 Then  
- txt\_Enter, txt\_TextChanged, txt\_KeyPress, e.KeyChar, **ControlChars.Back**

**EXERCISE 14.Grade Solution-Modified\_advanced** - when the result is subtraction from 100%, I use "1 -" : x% = 1 - y% and using: x.ToString("P1")  
**ElseIf**, **AndAlso**, **OrElse**, txt\_KeyPress, e.KeyChar, **ControlChars.Back**, txt\_Enter, txt\_TextChanged

**EXERCISE 15.Sales Solution-Modified\_advanced** - using colon : to separate different instructions in one line  
txt\_Enter, txt\_KeyPress, e.KeyChar, **ControlChars.Back**, txt\_TextChanged, lbl.BackColor = **SystemColors.Control**, ToString("P2")

**EXERCISE 16.OnYourOwn Solution** - rad.Checked, **ElseIf**, chk.Checked, txt\_Enter, txt\_TextChanged, rad\_CheckedChanged, chk\_CheckedChanged, txt\_KeyPress, e.KeyChar, **ControlChars.Back**

**EXERCISE 17.FixIt Solution** - **Select Case**, **Case Is**, txt\_Enter, txt\_TextChanged, txt\_KeyPress, e.KeyChar, **ControlChars.Back**

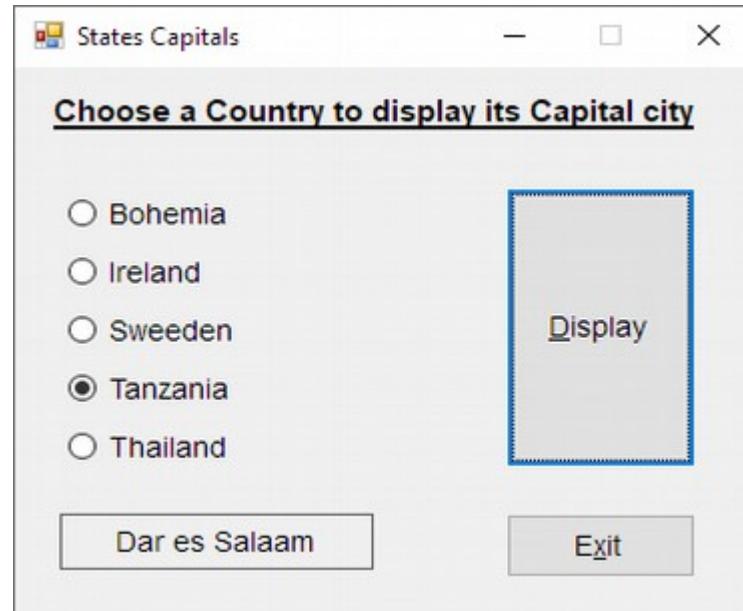
1. Create a Windows Forms application. Use the following names for the project and solution, respectively: States Capitals Project and States Capitals Solution. Save the application in the VB2017\Chap04 folder. Change the form file's name to Main Form.vb. Change the form's name to frmMain. Add five radio buttons, two labels, and two buttons to the form. One of the buttons should be an Exit button; the other should be a Display button. Each radio button's Text property should contain the name of a different state; choose any five state names. The Display button should display (in one of the labels) the name of the capital associated with the selected state name. Code the button's Click event procedure using the If...Then...Else statement. Be sure to code each radio button's CheckedChanged procedure. Save the solution and then start and test the application.

```
1 Option Explicit On
2 Option Strict On
3 Option Infer Off
4
5 Public Class frmMain
6     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
7         Me.Close()
8     End Sub
9
10    Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
11        If radBohemia.Checked = True Then
12            lblDisplay.Text = "Prague"
13        ElseIf radIreland.Checked = True Then
14            lblDisplay.Text = "Dublin"
15        ElseIf radSweeden.Checked = True Then
16            lblDisplay.Text = "Stockholm"
17        ElseIf radTanzania.Checked = True Then
18            lblDisplay.Text = "Dar es Salaam"
19        ElseIf radThailand.Checked = True Then
20            lblDisplay.Text = "Bangkok"
21        End If
22    End Sub
23
24    Private Sub radBohemia_CheckedChanged(sender As Object, e As EventArgs) Handles radBohemia.CheckedChanged
25        lblDisplay.Text = String.Empty
26    End Sub
```

```

27
28     Private Sub radIreland_CheckedChanged(sender As Object, e As EventArgs) Handles radIreland.CheckedChanged
29         lblDisplay.Text = String.Empty
30     End Sub
31
32     Private Sub radSweeden_CheckedChanged(sender As Object, e As EventArgs) Handles radSweeden.CheckedChanged
33         lblDisplay.Text = String.Empty
34     End Sub
35
36     Private Sub radTanzania_CheckedChanged(sender As Object, e As EventArgs) Handles radTanzania.CheckedChanged
37         lblDisplay.Text = String.Empty
38     End Sub
39
40     Private Sub radThailand_CheckedChanged(sender As Object, e As EventArgs) Handles radThailand.CheckedChanged
41         lblDisplay.Text = String.Empty
42     End Sub
43 End Class

```



Create a Windows Forms application. Use the following names for the project and solution, respectively: Hales Project and Hales Solution. Save the application in the VB2017\Chap04 folder. Change the form file's name to Main Form.vb. Change the form's name to frmMain. Create the interface shown in Figure 4-56. The interface contains a check box, a group box, four radio buttons, two labels, and two buttons. Be sure to set the tab order. The prices of the comforters sold at Hales Department Store, as well as the shipping fee, are included in Figure 4-56. The shipping fee is charged only when the customer is not taking advantage of the store pickup option. The Display cost button should determine the comforter's price and whether to charge a shipping fee. The button should display the cost of the comforter (which might include the shipping fee) with a dollar sign and two decimal places. Be sure to code the CheckedChanged procedures for the radio buttons and check box. Save the solution and then start and test the application. (The cost for a Queen comforter that will be picked up at the store is \$49.99; if it is shipped, the cost is \$54.99.)

```

1  Option Explicit On
2  Option Strict On
3  Option Infer Off
4
5  Public Class frmMain
6      Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
7          Dim dblCost As Double
8          Const intShipping As Integer = 5
9
10         Select Case True
11             Case radTwin.Checked
12                 dblCost = 39.99 + intShipping
13             Case radFull.Checked, radQueen.Checked
14                 dblCost = 49.99 + intShipping
15             Case radKing.Checked
16                 dblCost = 69.99 + intShipping
17         End Select
18
19         If chkPickUp.Checked = True Then
20             dblCost -= intShipping
21         End If
22
23         lblCost.Text = dblCost.ToString("C2")
24     End Sub

```

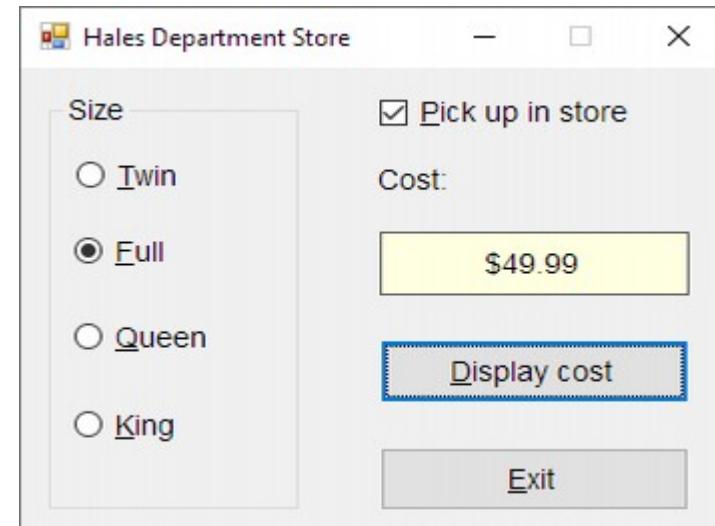
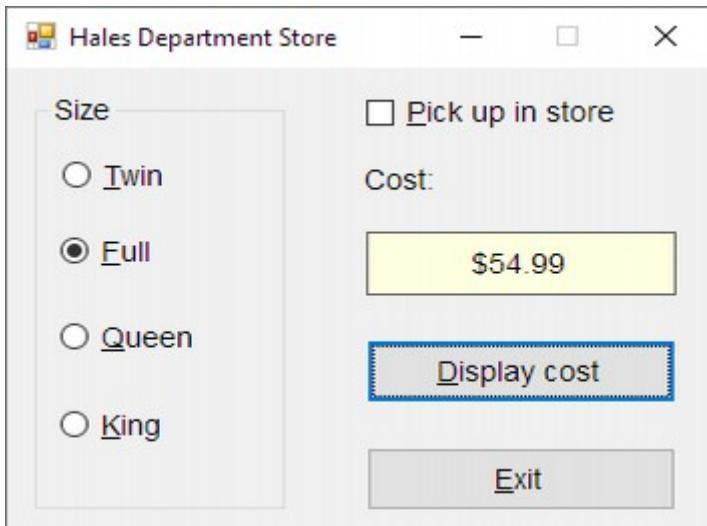


Figure 4-56 Interface, prices, and shipping fee for Exercise 2

```

25
26     Private Sub chkPickUp_CheckedChanged(sender As Object, e As EventArgs) Handles chkPickUp.CheckedChanged
27         lblCost.Text = String.Empty
28     End Sub
29
30     Private Sub radFull_CheckedChanged(sender As Object, e As EventArgs) Handles radFull.CheckedChanged
31         lblCost.Text = String.Empty
32     End Sub
33
34     Private Sub radKing_CheckedChanged(sender As Object, e As EventArgs) Handles radKing.CheckedChanged
35         lblCost.Text = String.Empty
36     End Sub
37
38     Private Sub radQueen_CheckedChanged(sender As Object, e As EventArgs) Handles radQueen.CheckedChanged
39         lblCost.Text = String.Empty
40     End Sub
41
42     Private Sub radTwin_CheckedChanged(sender As Object, e As EventArgs) Handles radTwin.CheckedChanged
43         lblCost.Text = String.Empty
44     End Sub
45
46     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
47         Me.Close()
48     End Sub
49 End Class

```



Create a Windows Forms application. Use the following names for the project and solution, respectively: Baxters Project and Baxters Solution. Save the application in the VB2017\Chap04 folder. Change the form file's name to Main Form.vb. Change the form's name to frmMain. Create the interface shown in Figure 4-57. The interface contains a text box, two check boxes, three labels, and two buttons. Be sure to set the tab order. Baxters (an online retailer) sells an 18-volt cordless drill for \$25.99. Employees of Baxters receive a 10% discount on their total order, excluding shipping. The shipping fee is a flat rate of \$9.99; however, if a customer has a shipping coupon, the shipping fee is \$4.99. Code the application. Be sure to code the CheckedChanged procedures for the two check boxes. Also code the Enter, KeyPress, and TextChanged procedures for the text box. Save the solution and then start and test the application. (The total due for two drills with an employee discount and a \$4.99 shipping coupon is \$51.77.)

```

1   ' item price = $25.99
2   ' shipping = $9.99, but with a coupon = $4.99
3   ' employee = -10% on items, excluding shipping
4   ' test: 1 item only = 25.99 + 9.99 = 35.98
5 Option Explicit On
6 Option Strict On
7 Option Infer Off
8 Public Class frmMain
9     Private Sub txtOrdered_TextChanged(sender As Object, e As EventArgs) Handles txtOrdered.TextChanged
10        lblTotal.Text = String.Empty
11    End Sub
12
13    Private Sub txtOrdered_Enter(sender As Object, e As EventArgs) Handles txtOrdered.Enter
14        txtOrdered.SelectAll()
15    End Sub
16
17    Private Sub chkDiscount_CheckedChanged(sender As Object, e As EventArgs) Handles chkDiscount.CheckedChanged
18        lblTotal.Text = String.Empty
19    End Sub
20
21    Private Sub chkShippingCoupon_CheckedChanged(sender As Object, e As EventArgs) Handles chkShippingCoupon.CheckedChanged
22        lblTotal.Text = String.Empty
23    End Sub
24
25    Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
26        Me.Close()
27    End Sub
28

```

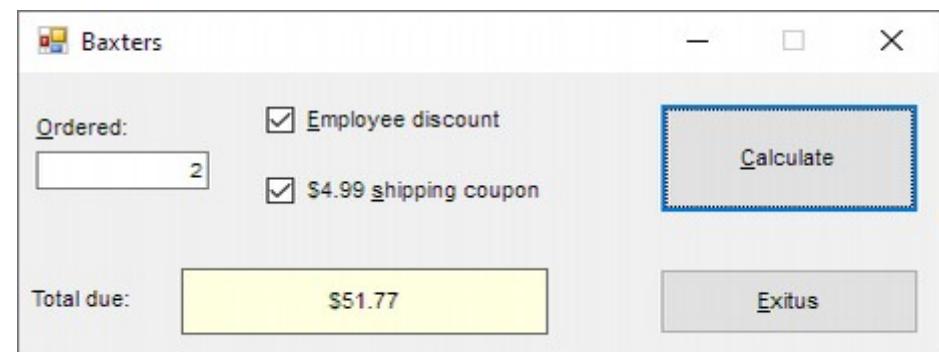
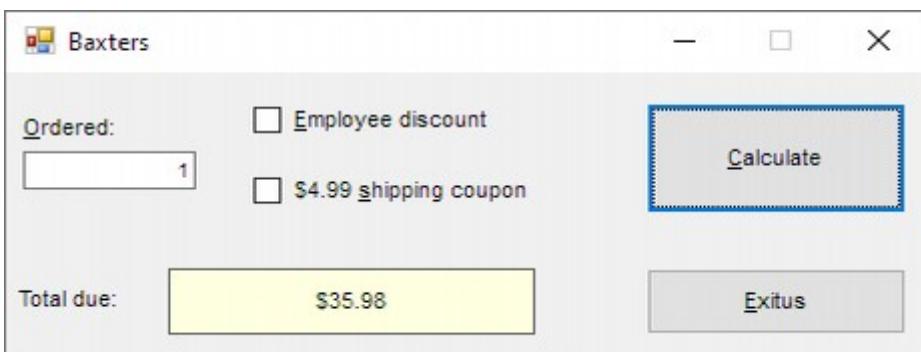


Figure 4-57 Interface for Exercise 3

```

29      Private Sub txtOrdered_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtOrdered.KeyPress
30          If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
31              e.Handled = True
32          End If
33      End Sub
34
35      Private Sub btnCalculate_Click(sender As Object, e As EventArgs) Handles btnCalculate.Click
36          Const dblItem As Double = 25.99           ' price for an item
37          Const dblShipping As Double = 9.99        ' standard shipping = +$9.99
38          Const dblShippingCoupon As Double = 4.99   ' Shipping coupon = not +$9.99, but +$4.99
39          Const dblDiscount As Double = 0.1         ' employee discount = -10% on items only, excluding shipping
40          Dim intOrdered As Integer
41          Dim dblPrice As Double
42          Dim dblTotal As Double
43          Integer.TryParse(txtOrdered.Text, intOrdered)
44
45          dblPrice = intOrdered * dblItem
46
47          If chkDiscount.Checked = True Then
48              dblPrice -= (dblPrice * dblDiscount)
49          End If
50
51          If chkShippingCoupon.Checked = True Then
52              dblTotal = dblPrice + dblShippingCoupon
53          Else
54              dblTotal = dblPrice + dblShipping
55          End If
56
57          lblTotal.Text = dblTotal.ToString("C2")
58      End Sub
59  End Class

```



Lorenzo's is having a BoGoHo (Buy One, Get One Half Off) sale. The store manager wants an application that allows the salesclerk to enter the prices of two items. The half off should always be applied to the item that has the lowest price. The application should calculate and display the total amount the customer owes as well as the amount he or she saved. For example, if the two items cost \$24.99 and \$10.00, the half off is applied to the \$10.00 item. The total owed is \$29.99 and the savings is \$5.00. Display the calculated amounts in label controls, and display them with a dollar sign and two decimal places.

- Create a Windows Forms application. Use the following names for the project and solution, respectively: Lorenzo Project and Lorenzo Solution. Save the

application in the VB2017\Chap04 folder. Change the form file's name to MainForm.vb. Change the form's name to frmMain. Create an appropriate interface and then code the application.

- Save the solution and then start and test the application.
- Now professionalize your application's interface as follows: The calculated amounts should be removed from the interface when a change is made to the contents of a text box in the interface. The contents of each text box should be selected when the text box receives the focus. Each text box should accept only numbers, the period, and the Backspace key. Save the solution and then start and test the application.

```

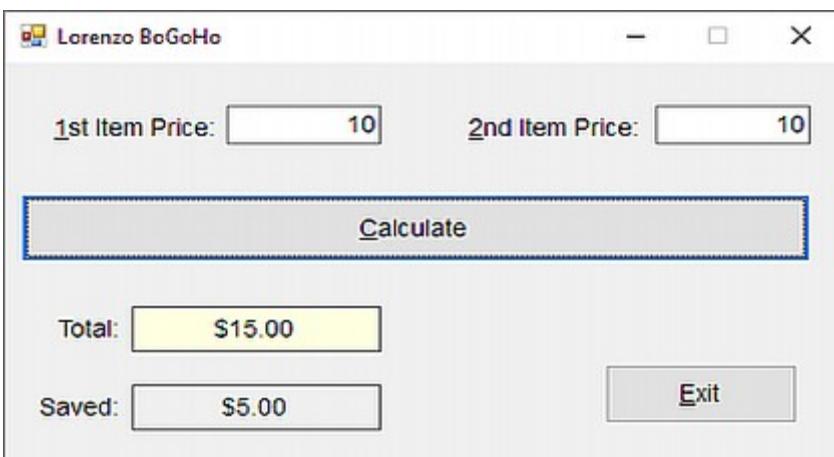
1  Option Explicit On
2  Option Strict On
3  Option Infer Off
4  Public Class frmMain
5      Private Sub txtPrice1_Enter(sender As Object, e As EventArgs) Handles txtPrice1.Enter
6          txtPrice1.SelectAll()
7      End Sub
8
9      Private Sub txtPrice2_Enter(sender As Object, e As EventArgs) Handles txtPrice2.Enter
10         txtPrice2.SelectAll()
11     End Sub
12
13     Private Sub txtPrice1_TextChanged(sender As Object, e As EventArgs) Handles txtPrice1.TextChanged
14         lblSaved.Text = String.Empty
15         lblTotal.Text = String.Empty
16     End Sub
17
18     Private Sub txtPrice2_TextChanged(sender As Object, e As EventArgs) Handles txtPrice2.TextChanged
19         lblSaved.Text = String.Empty
20         lblTotal.Text = String.Empty
21     End Sub
22
23     Private Sub btnCalculate_Click(sender As Object, e As EventArgs) Handles btnCalculate.Click
24         Dim dblPrice1 As Double
25         Dim dblPrice2 As Double
26         Dim dblTotal As Double
27         Dim dblSaved As Double
28         Double.TryParse(txtPrice1.Text, dblPrice1)
29         Double.TryParse(txtPrice2.Text, dblPrice2)

```

```

30
31     If dblPrice1 >= dblPrice2 Then
32         dblTotal = dblPrice1 + dblPrice2 - (dblPrice2 / 2)
33         dblSaved = dblPrice2 / 2
34     ElseIf dblPrice1 < dblPrice2 Then
35         dblTotal = dblPrice1 + dblPrice2 - (dblPrice1 / 2)
36         dblSaved = dblPrice1 / 2
37     End If
38
39     lblTotal.Text = dblTotal.ToString("C2")
40     lblSaved.Text = dblSaved.ToString("C2")
41 End Sub
42
43 Private Sub txtPrice1_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtPrice1.KeyPress
44     If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> "." AndAlso e.KeyChar <> ControlChars.Back Then
45         e.Handled = True
46     End If
47 End Sub
48
49 Private Sub txtPrice2_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtPrice2.KeyPress
50     If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> "." AndAlso e.KeyChar <> ControlChars.Back Then
51         e.Handled = True
52     End If
53 End Sub
54
55 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
56     Me.Close()
57 End Sub
58 End Class

```



Patti Garcia owns two cars, referred to as Car 1 and Car 2. She wants to drive one of the cars to her vacation destination, but she's not sure which one (if any) would cost her the least amount in gas. Create a Windows Forms application. Use the following names for the project and solution, respectively: Car Project and Car Solution. Save the application in the VB2017\Chap04 folder. Change the form file's name to MainForm.vb. Change the form's name to frmMain.

- The application's interface should provide text boxes for Patti to enter the following five items: the total miles she will drive, Car 1's miles per gallon (mpg), Car 2's miles per gallon (mpg), Car 1's cost per gallon of gas, and Car 2's cost per gallon of gas. (The cost per gallon of gas must be entered separately for each car because one car uses regular gas and the other uses premium gas.) The interface should display the total cost of the gas if she takes Car 1 and the total cost of the gas if she takes Car 2; display both amounts with a dollar sign and two decimal places. It should also display the car she should take (either Car 1 or Car 2) and approximately how much she will save by taking that car (show the savings with a dollar sign and no decimal places). If the total cost of gas would be the same for

- both cars, Patti should take Car 1 because that is her favorite car.
- The three text boxes that get the trip miles and the miles per gallon should accept only numbers and the Backspace key. The two text boxes that get the cost per gallon should accept only numbers, the period, and the Backspace key.
  - When a text box receives the focus, its existing text should be selected.
  - The output should be cleared when a change is made to the contents of a text box in the interface.
  - Test the application using 1200, 28, 35, 1.97, and 2.09 as the trip miles, Car 1's mpg, Car 2's mpg, Car 1's cost per gallon, and Car 2's cost per gallon, respectively. (Hint: The total costs for Car 1 and Car 2 are \$84.43 and \$71.66, respectively. By taking Car 2, Patti will save approximately \$13.)
  - Now, change Car 1's mpg to 35. Also, change Car 2's mpg to 28. Which car should Patti take, and how much (approximately) will she save?
  - Next, change Car 1's mpg and cost per gallon to 28 and 2.09, respectively. Which car should Patti take, and how much (approximately) will she save?

```

1  ' changed US localization to EU: gallons to Litres, $ to €, mpg to L / km
2  Option Explicit On
3  Option Strict On
4  Option Infer Off
5
6  Public Class frmMain
7      Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
8          Me.Close()
9      End Sub
10
11     Private Sub btnCalculate_Click(sender As Object, e As EventArgs) Handles btnCalculate.Click
12         Dim dblKM As Double           ' input: (txtKM) = how many km you gonna drive?
13         Dim dblKM100Car1 As Double    ' input: (txtKM100Car1) = how many km/100L
14         Dim dblKM100Car2 As Double    ' input: (txtKM100Car2) = how many km/100L
15         Dim dblLiter1 As Double       ' input: (txtLiter1) = price for 1L car 1
16         Dim dblLiter2 As Double       ' input: (txtLiter2) = price for 1L car 2
17         Dim dblCar1Drink As Double    ' price for 1L car 1
18         Dim dblCar2Drink As Double    ' price for 1L car 2
19         Dim dblCostCar1 As Double     ' output: (lblCostCar1) = ttl car 1
20         Dim dblCostCar2 As Double     ' output: (lblCostCar2) = ttl car 2
21         Dim dblSaves As Double        ' output: (lblSaves) = higher price - lower price
22         'lblChoose.Text = "Car 1" / "Car 2"
23

```

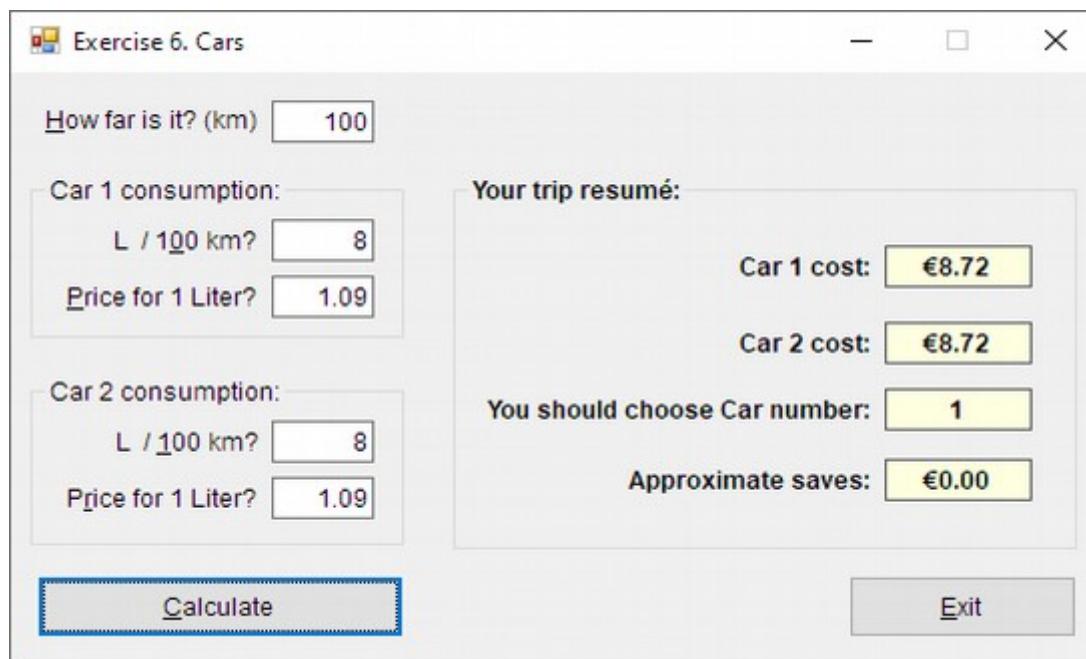
```
24     Double.TryParse(txtKM.Text, dblKM)
25     Double.TryParse(txtKM100Car1.Text, dblKM100Car1)
26     Double.TryParse(txtKM100Car2.Text, dblKM100Car2)
27     Double.TryParse(txtLiter1.Text, dblLiter1)
28     Double.TryParse(txtLiter2.Text, dblLiter2)
29
30     'do the math:
31     dblCar1Drink = (dblKM100Car1 * dblLiter1) / 100      'price for 1L Car1
32     dblCar2Drink = (dblKM100Car2 * dblLiter2) / 100      'price for 1L Car2
33
34     dblCostCar1 = dblKM * dblCar1Drink
35     dblCostCar2 = dblKM * dblCar2Drink
36
37     If dblCostCar1 <= dblCostCar2 Then
38         lblChoose.Text = "1"
39         dblSaves = dblCostCar2 - dblCostCar1
40     Else
41         lblChoose.Text = "2"
42         dblSaves = dblCostCar1 - dblCostCar2
43     End If
44
45     lblCostCar1.Text = "€" & dblCostCar1.ToString("N2")
46     lblCostCar2.Text = "€" & dblCostCar2.ToString("N2")
47     lblSaves.Text = "€" & dblSaves.ToString("N2")
48
49     End Sub
50
51     Private Sub txtKM_Enter(sender As Object, e As EventArgs) Handles txtKM.Enter
52         txtKM.SelectAll()
53     End Sub
54
55     Private Sub txtKM100Car1_Enter(sender As Object, e As EventArgs) Handles txtKM100Car1.Enter
56         txtKM100Car1.SelectAll()
57     End Sub
58
59     Private Sub txtKM100Car2_Enter(sender As Object, e As EventArgs) Handles txtKM100Car2.Enter
60         txtKM100Car2.SelectAll()
61     End Sub
62
63     Private Sub txtLiter1_Enter(sender As Object, e As EventArgs) Handles txtLiter1.Enter
64         txtLiter1.SelectAll()
65     End Sub
66
67     Private Sub txtLiter2_Enter(sender As Object, e As EventArgs) Handles txtLiter2.Enter
68         txtLiter2.SelectAll()
69     End Sub
```

```
68
69     Private Sub txtKM_TextChanged(sender As Object, e As EventArgs) Handles txtKM.TextChanged
70         lblChoose.Text = String.Empty
71         lblCostCar1.Text = String.Empty
72         lblCostCar2.Text = String.Empty
73         lblSaves.Text = String.Empty
74     End Sub
75
76     Private Sub txtKM100Car1_TextChanged(sender As Object, e As EventArgs) Handles txtKM100Car1.TextChanged
77         lblCostCar1.Text = String.Empty
78         lblChoose.Text = String.Empty
79         lblSaves.Text = String.Empty
80     End Sub
81
82     Private Sub txtLiter1_TextChanged(sender As Object, e As EventArgs) Handles txtLiter1.TextChanged
83         lblCostCar1.Text = String.Empty
84         lblChoose.Text = String.Empty
85         lblSaves.Text = String.Empty
86     End Sub
87
88     Private Sub txtKM100Car2_TextChanged(sender As Object, e As EventArgs) Handles txtKM100Car2.TextChanged
89         lblCostCar2.Text = String.Empty
90         lblChoose.Text = String.Empty
91         lblSaves.Text = String.Empty
92     End Sub
93
94     Private Sub txtLiter2_TextChanged(sender As Object, e As EventArgs) Handles txtLiter2.TextChanged
95         lblCostCar2.Text = String.Empty
96         lblChoose.Text = String.Empty
97         lblSaves.Text = String.Empty
98     End Sub
99
100    Private Sub txtKM_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtKM.KeyPress
101        If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> "." AndAlso e.KeyChar <> ControlChars.Back Then
102            e.Handled = True
103        End If
104    End Sub
105
106    Private Sub txtKM100Car1_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtKM100Car1.KeyPress
107        If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> "." AndAlso e.KeyChar <> ControlChars.Back Then
108            e.Handled = True
109        End If
110    End Sub
111
```

```

112  Private Sub txtKM100Car2_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtKM100Car2.KeyPress
113      If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> "." AndAlso e.KeyChar <> ControlChars.Back Then
114          e.Handled = True
115      End If
116  End Sub
117
118  Private Sub txtLiter1_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtLiter1.KeyPress
119      If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> "." AndAlso e.KeyChar <> ControlChars.Back Then
120          e.Handled = True
121      End If
122  End Sub
123
124  Private Sub txtLiter2_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtLiter2.KeyPress
125      If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> "." AndAlso e.KeyChar <> ControlChars.Back Then
126          e.Handled = True
127      End If
128  End Sub
129 End Class

```

Exercise 6. Cars

|                                          |                                    |
|------------------------------------------|------------------------------------|
| How far is it? (km)                      | <input type="text" value="100"/>   |
| Car 1 consumption:                       |                                    |
| L / 100 km?                              | <input type="text" value="8"/>     |
| Price for 1 Liter?                       | <input type="text" value="1.09"/>  |
| Car 2 consumption:                       |                                    |
| L / 100 km?                              | <input type="text" value="8"/>     |
| Price for 1 Liter?                       | <input type="text" value="1.09"/>  |
| Your trip resumé:                        |                                    |
| Car 1 cost:                              | <input type="text" value="€8.72"/> |
| Car 2 cost:                              | <input type="text" value="€8.72"/> |
| You should choose Car number:            | <input type="text" value="1"/>     |
| Approximate saves:                       | <input type="text" value="€0.00"/> |
| <input type="button" value="Calculate"/> |                                    |
| <input type="button" value="Exit"/>      |                                    |

- arithmetic operator \ integer division, **ElseIf**, **Mod**, **txt\_Enter**, **txt\_TextChanged**, **txt\_KeyPress**, **e.KeyChar**, **ControlChars.Back**

7. Open the Wedding Solution.sln file contained in the VB2017\Chap04\Wedding Solution folder. The application should display the number of round tables needed to seat only the guests at a wedding reception. (In other words, the bridal party does not need to be included in this calculation.) Each round table can accommodate a maximum of 8 guests. When the text box receives the focus, its existing text should be selected. The text box should accept only numbers and the Backspace key. The output should be cleared when a change is made to the contents of the text box. Code the application. Save the solution and then start and test the application. (If the number of guests is 235, the number of required tables is 30.)

```
1  ' Name:      Wedding Project
2  ' Purpose:    Displays the number of tables needed.
3  ' Programmer: <your name> on <current date>
4
5 Option Explicit On
6 Option Strict On
7 Option Infer Off
8
9 Public Class frmMain
10    Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
11        'Calculate and display the number of required tables.
12        '1 table = 8 guests
13        Dim intGuests As Integer
14        Dim intRound As Integer
15        Const intTable As Integer = 8
16        Integer.TryParse(txtGuests.Text, intGuests)
17
18        If intGuests = 0 Then
19            intRound = 0
20        ElseIf intGuests <= 8 Then
21            intRound = 1
22        ElseIf intGuests Mod intTable = 0 Then
23            intRound = intGuests \ intTable
24        Else
25            intRound = intGuests \ intTable + 1
26        End If
27
28        lblRound.Text = intRound.ToString("N0")
29    End Sub
```

```

30
31     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
32         Me.Close()
33     End Sub
34
35     Private Sub txtGuests_Enter(sender As Object, e As EventArgs) Handles txtGuests.Enter
36         txtGuests.SelectAll()
37     End Sub
38
39     Private Sub txtGuests_TextChanged(sender As Object, e As EventArgs) Handles txtGuests.TextChanged
40         lblRound.Text = String.Empty
41     End Sub
42
43     Private Sub txtGuests_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtGuests.KeyPress
44         If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
45             e.Handled = True
46         End If
47     End Sub
48 End Class

```



## Chap04\Exercisel

### EXERCISE 08.Wedding Solution-Modified\_intermediate

- arithmetic operator \ integer division, **ElseIf**, **Mod**, **txt\_Enter**, **txt\_TextChanged**, **txt\_KeyPress**, **e.KeyChar**, **ControlChars.Back**

In this exercise, you modify the application from Exercise 7. The modified application will display the number of rectangular tables needed to seat the bridal party as well as the number of round tables required for the guests. Each rectangular table can accommodate a maximum of 10 people. As in Exercise 7, a maximum of 8 guests can fit at each round table. Use Windows to make a copy of the Wedding Solution folder. Rename the copy Modified Wedding Solution. Open the Wedding Solution.sln file contained in the Modified Wedding Solution folder. In addition to entering the number of guests, the interface should now allow the user to also enter the number of people

in the bridal party. Modify the interface by including three additional labels and a text box, and then reset the tab order. Make the appropriate modifications to the code. Be sure to code the new text box's Enter, KeyPress, and TextChanged procedures. Also, be sure to modify the txtGuests\_TextChanged procedure. Save the solution and then start and test the application. (If the numbers of guests and bridal party members are 130 and 15, respectively, the numbers of required round and rectangular tables are 17 and 2, respectively.)

```
1 Option Explicit On
2 Option Strict On
3 Option Infer Off
4
5 Public Class frmMain
6     Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
7         'Calculate and display the number of required tables.
8         Dim intGuests As Integer
9         Dim intRound As Integer
10        Const intTablesRound As Integer = 8
11
12        Dim intBridal As Integer
13        Dim intRect As Integer
14        Const intTablesRect As Integer = 10
15
16        Integer.TryParse(txtGuests.Text, intGuests)
17        Integer.TryParse(txtBridal.Text, intBridal)
18
19        ' Round Tables calculation:
20        If intGuests = 0 Then
21            intRound = 0
22        ElseIf intGuests <= 8 Then
23            intRound = 1
24        ElseIf intGuests Mod intTablesRound = 0 Then
25            intRound = intGuests \ intTablesRound
26        Else
27            intRound = intGuests \ intTablesRound + 1
28        End If
29
30        ' Rectangular Tables calculation:
31        If intBridal = 0 Then
32            intRect = 0
33        ElseIf intBridal <= 10 Then
34            intRect = 1
35        ElseIf intBridal Mod intTablesRect = 0 Then
36            intRect = intBridal \ intTablesRect
37        Else
38            intRect = intBridal \ intTablesRect + 1
39
40        End If
41        lblRound.Text = intRound.ToString("N0")
42        lblRect.Text = intRect.ToString("N0")
43    End Sub
44
```

```

45      Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
46          Me.Close()
47      End Sub
48
49      Private Sub txtGuests_Enter(sender As Object, e As EventArgs) Handles txtGuests.Enter
50          txtGuests.SelectAll()
51      End Sub
52
53      Private Sub txtGuests_TextChanged(sender As Object, e As EventArgs) Handles txtGuests.TextChanged
54          lblRound.Text = String.Empty
55      End Sub
56
57      Private Sub txtGuests_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtGuests.KeyPress
58          If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
59              e.Handled = True
60          End If
61      End Sub
62
63      Private Sub txtBridal_Enter(sender As Object, e As EventArgs) Handles txtBridal.Enter
64          txtBridal.SelectAll()
65      End Sub
66
67      Private Sub txtBridal_TextChanged(sender As Object, e As EventArgs) Handles txtBridal.TextChanged
68          lblRect.Text = String.Empty
69      End Sub
70
71      Private Sub txtBridal_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtBridal.KeyPress
72          If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
73              e.Handled = True
74          End If
75      End Sub
76  End Class

```

Seating - Modified

|                                                                              |                                 |
|------------------------------------------------------------------------------|---------------------------------|
| Guests:                                                                      | Round tables needed:            |
| <input type="text" value="130"/>                                             | <input type="text" value="17"/> |
| Bridal people:                                                               | Rectangular tables needed:      |
| <input type="text" value="15"/>                                              | <input type="text" value="2"/>  |
| <input type="button" value="Calculate"/> <input type="button" value="Exit"/> |                                 |

Seating - Modified

|                                                                              |                                |
|------------------------------------------------------------------------------|--------------------------------|
| Guests:                                                                      | Round tables needed:           |
| <input type="text" value="48"/>                                              | <input type="text" value="6"/> |
| Bridal people:                                                               | Rectangular tables needed:     |
| <input type="text" value="20"/>                                              | <input type="text" value="2"/> |
| <input type="button" value="Calculate"/> <input type="button" value="Exit"/> |                                |

Seating - Modified

|                                                                              |                                |
|------------------------------------------------------------------------------|--------------------------------|
| Guests:                                                                      | Round tables needed:           |
| <input type="text" value="49"/>                                              | <input type="text" value="7"/> |
| Bridal people:                                                               | Rectangular tables needed:     |
| <input type="text" value="21"/>                                              | <input type="text" value="3"/> |
| <input type="button" value="Calculate"/> <input type="button" value="Exit"/> |                                |

Software Haven sells a software package that is available in three editions. The application should display the price of the edition a customer wants to purchase. The retail prices for the Ultimate, Professional, and Student editions are \$899.99, \$599.99, and \$99.99, respectively. Some customers may have a coupon worth 10% off the price of the Ultimate edition, while others may have a coupon worth 20% off the price of the Student edition. Create a Windows Forms application. Use the following names for the project and solution, respectively: Software Project and Software Solution. Save the application in the VB2017\Chap04 folder. Change the form file's name to Main Form.vb. Change the form's name to frmMain. Create the interface shown in Figure 4-58. The interface contains a group box, six radio buttons, two labels, and two buttons. Be sure to set the tab order. Code the application. Be sure to code each radio button's CheckedChanged procedure. Save the solution and then start and test the application.



```

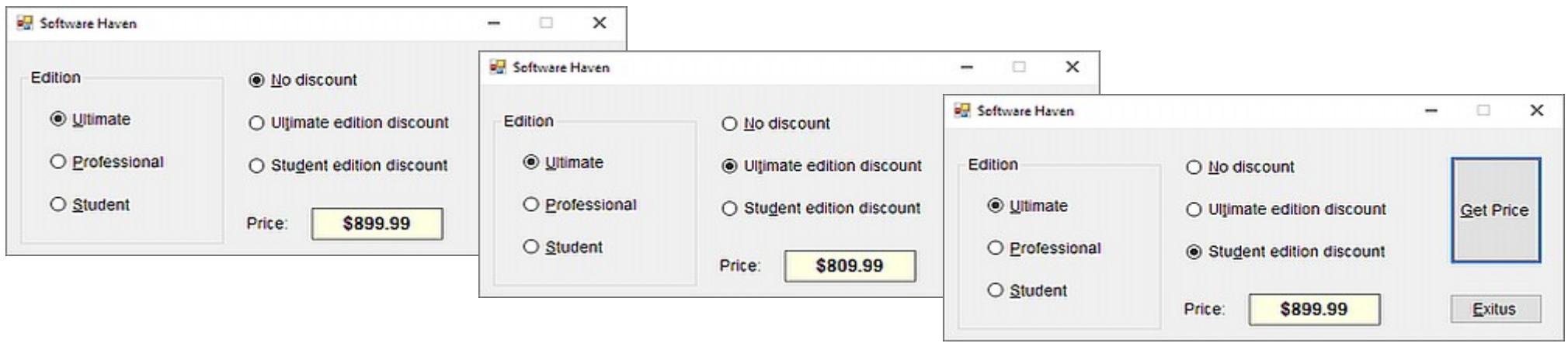
1  Option Explicit On
2  Option Strict On
3  Option Infer Off
4  Public Class frmMain
5      Private Sub btnPrice_Click(sender As Object, e As EventArgs) Handles btnPrice.Click
6          Const dblUlti As Double = 899.99
7          Const dblPro As Double = 599.99
8          Const dblStu As Double = 99.99
9          Const dbldiscUlti As Double = 0.1
10         Const dbldiscStu As Double = 0.2
11         Dim dblPrice As Double
12
13         Select Case True
14             Case rad5discUlti.Checked AndAlso rad1ulti.Checked
15                 dblPrice = dblUlti - dblUlti * dbldiscUlti
16             Case rad6discStu.Checked AndAlso rad3stu.Checked
17                 dblPrice = dblStu - (dblStu * dbldiscStu)
18             Case rad1ulti.Checked
19                 dblPrice = dblUlti
20             Case rad2pro.Checked
21                 dblPrice = dblPro
22             Case rad3stu.Checked
23                 dblPrice = dblStu
24         End Select
25         lblPrice.Text = dblPrice.ToString("C2")
26     End Sub

```

```

27
28     Private Sub rad1ulti_CheckedChanged(sender As Object, e As EventArgs) Handles rad1ulti.CheckedChanged
29         lblPrice.Text = String.Empty
30     End Sub
31
32     Private Sub rad2pro_CheckedChanged(sender As Object, e As EventArgs) Handles rad2pro.CheckedChanged
33         lblPrice.Text = String.Empty
34     End Sub
35
36     Private Sub rad3stu_CheckedChanged(sender As Object, e As EventArgs) Handles rad3stu.CheckedChanged
37         lblPrice.Text = String.Empty
38     End Sub
39
40     Private Sub rad4discNo_CheckedChanged(sender As Object, e As EventArgs) Handles rad4discNo.CheckedChanged
41         lblPrice.Text = String.Empty
42     End Sub
43
44     Private Sub rad5discUlti_CheckedChanged(sender As Object, e As EventArgs) Handles rad5discUlti.CheckedChanged
45         lblPrice.Text = String.Empty
46     End Sub
47
48     Private Sub rad6discStu_CheckedChanged(sender As Object, e As EventArgs) Handles rad6discStu.CheckedChanged
49         lblPrice.Text = String.Empty
50     End Sub
51
52     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
53         Me.Close()
54     End Sub
55 End Class

```



Williams Cable Company wants an application that displays a customer's monthly cable bill, which is based on the information shown in Figure 4-59. Create a Windows Forms application. Use the following names for the project and solution, respectively: Williams Project and Williams Solution. Save the application in the VB2017\Chap04 folder.

Change the form file's name to Main Form.vb. Change the form's name to frmMain. Create an appropriate interface using radio buttons for the different packages and check boxes for the additional features. Be sure to set the tab order. Code the application. Be sure to code the CheckedChanged procedures for the radio buttons and check boxes. Save the solution and then start and test the application.

| Packages                 | Monthly charge (\$) |
|--------------------------|---------------------|
| Basic                    | 24.99               |
| Silver                   | 42.99               |
| Gold                     | 84.99               |
| Diamond                  | 99.99               |
| Additional features      | Monthly charge (\$) |
| Cinematic movie channels | 9.50                |
| HBI movie channels       | 9.50                |
| Showtimer movie channels | 10.50               |
| Local stations           | 6.00                |

```

1  Option Explicit On
2  Option Strict On
3  Option Infer Off
4
5  Public Class frmMain
6      Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
7          Me.Close()
8      End Sub
9
10     Private Sub chk1cinnematic_CheckedChanged(sender As Object, e As EventArgs) Handles chk1cinnematic.CheckedChanged
11         lblShow.Text = String.Empty
12     End Sub
13
14     Private Sub chk2hbi_CheckedChanged(sender As Object, e As EventArgs) Handles chk2hbi.CheckedChanged
15         lblShow.Text = String.Empty
16     End Sub
17
18     Private Sub chk3showtime_CheckedChanged(sender As Object, e As EventArgs) Handles chk3showtime.CheckedChanged
19         lblShow.Text = String.Empty
20     End Sub
21
22     Private Sub chk4local_CheckedChanged(sender As Object, e As EventArgs) Handles chk4local.CheckedChanged
23         lblShow.Text = String.Empty
24     End Sub
25
26     Private Sub rad1basic_CheckedChanged(sender As Object, e As EventArgs) Handles rad1basic.CheckedChanged
27         lblShow.Text = String.Empty
28     End Sub

```

```
29
30     Private Sub rad2silver_CheckedChanged(sender As Object, e As EventArgs) Handles rad2silver.CheckedChanged
31         lblShow.Text = String.Empty
32     End Sub
33
34     Private Sub rad3gold_CheckedChanged(sender As Object, e As EventArgs) Handles rad3gold.CheckedChanged
35         lblShow.Text = String.Empty
36     End Sub
37
38     Private Sub rad4diamond_CheckedChanged(sender As Object, e As EventArgs) Handles rad4diamond.CheckedChanged
39         lblShow.Text = String.Empty
40     End Sub
41
42     Private Sub btnCalculate_Click(sender As Object, e As EventArgs) Handles btnCalculate.Click
43         Const dbl1 As Double = 9.5
44         Const dbl2 As Double = 9.5
45         Const dbl3 As Double = 10.5
46         Const dbl4 As Double = 6
47         Dim dblShow As Double
48
49         Select Case True
50             Case rad1basic.Checked
51                 dblShow = 24.99
52             Case rad2silver.Checked
53                 dblShow = 42.99
54             Case rad3gold.Checked
55                 dblShow = 84.99
56             Case rad4diamond.Checked
57                 dblShow = 99.99
58         End Select
59
60         If chk1cinematic.Checked Then
61             dblShow += dbl1
62         End If
63
64         If chk2hbi.Checked Then
65             dblShow += dbl2
66         End If
67
68         If chk3showtime.Checked Then
69             dblShow += dbl3
70         End If
71
```

```

72     If chk4local.Checked Then
73         dblShow += dbl4
74     End If
75
76     lblShow.Text = dblShow.ToString("C2")
77 End Sub
78 End Class

```

This screenshot shows the Windows application interface for the Williams Cable Company. On the left, under 'Packages', the 'Gold (\$84.99)' radio button is selected. On the right, under 'Additional features', the 'Cinematic movie channels (\$9.50)', 'HBI movie channels (\$9.50)', 'Showtimer movie channels (\$10.50)', and 'Local stations (\$6.00)' checkboxes are all unselected. At the bottom, the 'Calculate' button is highlighted in blue, and the 'Monthly Cable Bill:' label shows '\$84.99'.

This screenshot shows the same application after changes. Under 'Additional features', the 'Cinematic movie channels (\$9.50)', 'HBI movie channels (\$9.50)', 'Showtimer movie channels (\$10.50)', and 'Local stations (\$6.00)' checkboxes are now checked. The 'Calculate' button is still highlighted in blue, and the 'Monthly Cable Bill:' label shows '\$120.49'.

## Chap04\Exercise1

### EXERCISE 11.Canton Solution\_advanced

- decimal numbers -> number + D -> 3720D

- `Select Case, Case ... To ..., Case Is ..., Case Else, chk.Checked, chk_CheckedChanged, txt_TextChanged, txt_Enter, txt_KeyPress, e.KeyChar, ControlChars.Back`

Each salesperson at Canton Inc. receives a commission based on the amount of his or her sales. The commission rates and additional payment amounts are shown in Figure 4-60. Create a Windows Forms application. Use the following names for the project and solution, respectively: Canton Project and Canton Solution. Save the application in the VB2017\Chap04 folder. Change the form file's name to Main Form.vb. Change the form's name to frmMain. Create the interface shown in Figure 4-60. The text box should accept only numbers, the period, and the Backspace key, and its text should be selected when it receives the focus. Calculate

the commission, any additional amount, and the total due only when the sales amount is greater than 0; otherwise, display \$0.00 as the commission, additional amount, and total due. The calculated amounts should be cleared when a change is made to any of the input items. Save the solution and then start and test the application. (The total due for a salesperson who has been with the company for 11 years and whose sales are \$13,000 is \$2,010.00.)

The screenshot shows the 'Canton Inc.' application window. It includes input fields for 'Sales:', 'Commission only:', and 'Additional amount:', and checkboxes for 'Over 10 years' and 'Traveling'. Buttons for 'Calculate' and 'Exit' are also present. Below the window is a table detailing commission rates:

| Sales (\$)                                                                                                      | Commission                                |
|-----------------------------------------------------------------------------------------------------------------|-------------------------------------------|
| 1-5,999.99                                                                                                      | 10% of sales                              |
| 6,000-29,999.99                                                                                                 | \$600 plus 13% of the sales over 6,000    |
| 30,000 and over                                                                                                 | \$3,720 plus 14% of the sales over 30,000 |
| Additional                                                                                                      |                                           |
| \$500 if the salesperson has worked at the company for over 10 years and the sales amount is at least \$10,000. |                                           |
| \$700 if the salesperson travels.                                                                               |                                           |

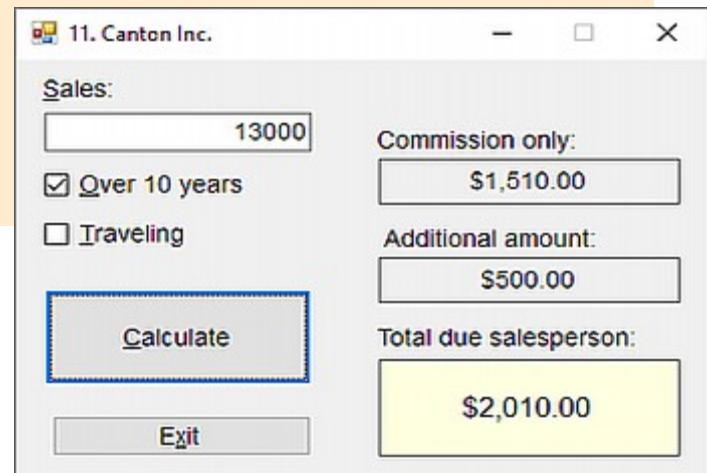
```
1 Option Explicit On
2 Option Strict On
3 Option Infer Off
4 ' sales: 1 - 5999.99      -> commission = 10% of all sales
5 ' sales: 6000 - 29999.99 -> commission = 13% of sales over 6000
6 '                                     = + 600 bonus
7 ' sales: 30000 and over   -> commission = 14% of sales over 30000
8 '                                     = + 3720 bonus
9 ' additional: if over 10 years & sales >= 10000 = + 500
10 '           if travel          = + 700
11 ' test: 13000, 11 years -> commission = 2010.00
12
13 Public Class frmMain
14     Private Sub txtSales_Enter(sender As Object, e As EventArgs) Handles txtSales.Enter
15         txtSales.SelectAll()
16     End Sub
17
18     Private Sub txtSales_TextChanged(sender As Object, e As EventArgs) Handles txtSales.TextChanged
19         lblAdd.Text = String.Empty
20         lblCom.Text = String.Empty
21         lblTotal.Text = String.Empty
22     End Sub
23
24     Private Sub chkOver10_CheckedChanged(sender As Object, e As EventArgs) Handles chkOver10.CheckedChanged
25         lblAdd.Text = String.Empty
26         lblCom.Text = String.Empty
27         lblTotal.Text = String.Empty
28     End Sub
29
30     Private Sub chkTraveling_CheckedChanged(sender As Object, e As EventArgs) Handles chkTraveling.CheckedChanged
31         lblAdd.Text = String.Empty
32         lblTotal.Text = String.Empty
33     End Sub
34
35     Private Sub txtSales_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtSales.KeyPress
36         If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> "." AndAlso e.KeyChar <> ControlChars.Back Then
37             e.Handled = True
38         End If
39     End Sub
40
41     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
42         Me.Close()
43     End Sub
44
```

```

45  Private Sub btnCalculate_Click(sender As Object, e As EventArgs) Handles btnCalculate.Click
46      Dim decSales As Decimal           ' input txtSales
47      Dim decCom As Decimal
48      Dim decAdd As Decimal
49      Dim decTotal As Decimal
50      Const decBonus1 As Decimal = 600D
51      Const decBonus2 As Decimal = 3720D
52      Decimal.TryParse(txtSales.Text, decSales)
53
54      ' Commission only:
55      Select Case decSales
56          Case 1D To 5999.99D
57              decCom = decSales * 0.1D
58          Case 6000D To 29999.99D
59              decCom = ((decSales - 6000) * 0.13D) + decBonus1
60          Case Is >= 30000D
61              decCom = ((decSales - 30000D) * 0.14D) + decBonus2
62          Case Else
63              decCom = 0
64      End Select
65
66      ' additional:
67      If decCom > 0 AndAlso chkOver10.Checked AndAlso decSales >= 10000D Then
68          decAdd += 500D
69      End If
70
71      If decCom > 0 AndAlso chkTraveling.Checked Then
72          decAdd += 700D
73      End If
74
75      decTotal = decCom + decAdd
76      lblCom.Text = decCom.ToString("C2")
77      lblAdd.Text = decAdd.ToString("C2")
78      lblTotal.Text = decTotal.ToString("C2")
79  End Sub
80 End Class

```

test: if sales \$13000 + 11 years  
 $13000 - 6000 = 7000$   
 $7000 * 13\% = 910$   
 $910 + 600 = 1510$   
 $1510 + 500 = 2010$



In this exercise, you create an application for Genatone Inc. The application displays the price of an order based on the number of units ordered and the customer's status (either wholesaler or retailer). The price per unit is shown in Figure 4-61. Create a Windows Forms application. Use the following names for the project and solution, respectively: Genatone Project and Genatone Solution. Save the application in the VB2017\Chap04 folder. Change the form file's name to Main Form.vb. Change the form's name to frmMain. Create a suitable interface. Use radio buttons to determine the customer's status. Code the application. Save the solution and then start and test the application. (A wholesaler ordering 70 units will pay \$910.00, while a retailer will pay \$1,260.00.)

| Wholesaler      |                     | Retailer        |                     |
|-----------------|---------------------|-----------------|---------------------|
| Number of units | Price per unit (\$) | Number of units | Price per unit (\$) |
| 1-50            | 15                  | 1-25            | 22                  |
| 51-150          | 13                  | Over 25         | 18                  |
| Over 150        | 10                  |                 |                     |

Figure 4-61 Pricing chart for Exercise 12

```

1  Option Explicit On
2  Option Strict On
3  Option Infer Off
4  Public Class frmMain
5      Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
6          Me.Close()
7      End Sub
8
9      Private Sub txtHowMany_Enter(sender As Object, e As EventArgs) Handles txtHowMany.Enter
10         txtHowMany.SelectAll()
11     End Sub
12
13     Private Sub txtHowMany_TextChanged(sender As Object, e As EventArgs) Handles txtHowMany.TextChanged
14         lblPrice.Text = String.Empty
15     End Sub
16
17     Private Sub rad1whole_CheckedChanged(sender As Object, e As EventArgs) Handles rad1whole.CheckedChanged
18         lblPrice.Text = String.Empty
19     End Sub
20
21     Private Sub rad2reta_CheckedChanged(sender As Object, e As EventArgs) Handles rad2reta.CheckedChanged
22         lblPrice.Text = String.Empty
23     End Sub
24
25     Private Sub txtHowMany_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtHowMany.KeyPress
26         If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
27             e.Handled = True
28         End If
29     End Sub

```

```

30
31     Private Sub btnCalculate_Click(sender As Object, e As EventArgs) Handles btnCalculate.Click
32         Dim intHowMany As Integer
33         Dim intPrice As Integer
34         Dim intADD As Integer
35         Integer.TryParse(txtHowMany.Text, intHowMany)
36
37         If rad1whole.Checked Then
38             If intHowMany > 0 AndAlso intHowMany <= 50 Then
39                 intADD = 15
40             ElseIf intHowMany > 50 AndAlso intHowMany <= 150 Then
41                 intADD = 13
42             ElseIf intHowMany > 150 Then
43                 intADD = 10
44             Else
45                 intADD = 0
46             End If
47         End If
48
49         If rad2reta.Checked Then
50             If intHowMany > 0 AndAlso intHowMany <= 25 Then
51                 intADD = 22
52             ElseIf intHowMany > 25 Then
53                 intADD = 18
54             Else
55                 intADD = 0
56             End If
57         End If
58
59         intPrice = intHowMany * intADD
60         lblPrice.Text = intPrice.ToString("C2")
61     End Sub
62 End Class

```



## EXERCISE 13.Jacket Haven Solution\_advanced

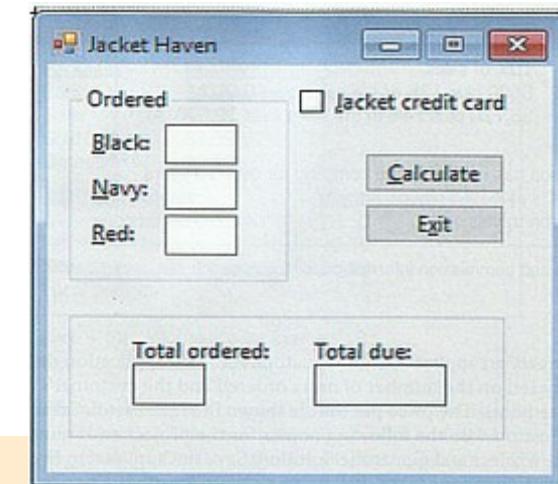
- decimal numbers -> number + D -> 45.99D
- **ElseIf** chk.Checked = **False AndAlso** int > 1 **Then**
- txt\_Enter, txt\_TextChanged, txt\_KeyPress, e.KeyChar, ControlChars.Back

13. Create a Windows Forms application. Use the following names for the project and solution, respectively: Jacket Haven Project and Jacket Haven Solution. Save the application in the VB2017\Chap04 folder. Change the form file's name to Main Form.vb. Change the form's name to frmMain. Create the interface shown in Figure 4-62. The interface contains a check box, seven labels, three text boxes, two group boxes, and two buttons. Be sure to set the tab order. The black jackets at Jacket Haven are the most popular and cost \$45.99; the navy and red jackets cost \$39.99. Customers are given a 10% discount when using their Jacket credit card to pay for an order. Customers who do not use the Jacket credit card to pay for the order receive a 5% discount on the purchase of two or more jackets. Save the solution and then start and test the application.

```

1 Option Explicit On
2 Option Strict On
3 Option Infer Off
4
5 Public Class frmMain
6     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
7         Me.Close()
8     End Sub
9
10    Private Sub txt1_Enter(sender As Object, e As EventArgs) Handles txt1.Enter
11        txt1.SelectAll()
12    End Sub
13
14    Private Sub txt1_TextChanged(sender As Object, e As EventArgs) Handles txt1.TextChanged
15        lbl1ttlOrdered.Text = String.Empty
16        lbl2ttlDue.Text = String.Empty
17    End Sub
18
19    Private Sub txt2_Enter(sender As Object, e As EventArgs) Handles txt2.Enter
20        txt2.SelectAll()
21    End Sub
22
23    Private Sub txt2_TextChanged(sender As Object, e As EventArgs) Handles txt2.TextChanged
24        lbl1ttlOrdered.Text = String.Empty
25        lbl2ttlDue.Text = String.Empty
26    End Sub

```



```
27
28     Private Sub txt3_Enter(sender As Object, e As EventArgs) Handles txt3.Enter
29         txt3.SelectAll()
30     End Sub
31
32     Private Sub txt3_TextChanged(sender As Object, e As EventArgs) Handles txt3.TextChanged
33         lbl1ttlOrdered.Text = String.Empty
34         lbl2ttlDue.Text = String.Empty
35     End Sub
36
37     Private Sub chkCard_CheckedChanged(sender As Object, e As EventArgs) Handles chkCard.CheckedChanged
38         lbl2ttlDue.Text = String.Empty
39     End Sub
40
41     Private Sub txt1_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txt1.KeyPress
42         If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
43             e.Handled = True
44         End If
45     End Sub
46
47     Private Sub txt2_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txt2.KeyPress
48         If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
49             e.Handled = True
50         End If
51     End Sub
52
53     Private Sub txt3_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txt3.KeyPress
54         If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
55             e.Handled = True
56         End If
57     End Sub
58
59     Private Sub btnCalculate_Click(sender As Object, e As EventArgs) Handles btnCalculate.Click
60         ' change to Decimal, please
61         Dim int1 As Integer
62         Dim int2 As Integer
63         Dim int3 As Integer
64         Dim int1ttlOrdered As Integer
65         Dim dec2ttlDue As Decimal
66         Const dec1 As Decimal = 45.99D
67         Const dec2 As Decimal = 39.99D
68         Const dec3 As Decimal = 39.99D
69         Dim decTTL13 As Decimal
```

```

70     Integer.TryParse(txt1.Text, int1)
71     Integer.TryParse(txt2.Text, int2)
72     Integer.TryParse(txt3.Text, int3)

73
74     int1ttl1Ordered = int1 + int2 + int3
75     decTTL13 = (int1 * dec1) + (int2 * dec2) + (int3 * dec3)

76
77     If chkCard.Checked Then
78         dec2ttl1Due = decTTL13 - (decTTL13 * 0.1D)
79     ElseIf chkCard.Checked = False AndAlso int1ttl1Ordered > 1 Then
80         dec2ttl1Due = decTTL13 - (decTTL13 * 0.05D)
81     Else
82         dec2ttl1Due = decTTL13
83     End If

84
85     lbl1ttl1Ordered.Text = int1ttl1Ordered.ToString("N0")
86     lbl2ttl1Due.Text = dec2ttl1Due.ToString("C2")
87 End Sub
88 End Class

```

13. Jacket Haven

|                                             |                                       |
|---------------------------------------------|---------------------------------------|
| Amount of:                                  |                                       |
| Rainbow jackets:                            | <input type="text" value="5"/>        |
| Skull jackets:                              | <input type="text" value="3"/>        |
| Naked jackets:                              | <input type="text" value="8"/>        |
| Total ordered:                              | <input type="text" value="16"/>       |
| Total due:                                  | <input type="text" value="\$636.35"/> |
| <input type="checkbox"/> Jacket credit card |                                       |
| <b>Calculate</b>                            | <b>Exitus</b>                         |

$45.99 * 5 = 229.95$   
 $39.99 * 3 = 119.97$   
 $39.99 * 8 = 319.92$   
 ttl: 16 669.84  
 - if no credit card, but 2 or more, then - 5%  
 $669.84 * 0.05 = 33.492$   
 $669.84 - 33.492 = 636.35$

13. Jacket Haven

|                                                        |                                       |
|--------------------------------------------------------|---------------------------------------|
| Amount of:                                             |                                       |
| Rainbow jackets:                                       | <input type="text" value="5"/>        |
| Skull jackets:                                         | <input type="text" value="3"/>        |
| Naked jackets:                                         | <input type="text" value="8"/>        |
| Total ordered:                                         | <input type="text" value="16"/>       |
| Total due:                                             | <input type="text" value="\$602.86"/> |
| <input checked="" type="checkbox"/> Jacket credit card |                                       |
| <b>Calculate</b>                                       | <b>Exitus</b>                         |

$45.99 * 5 = 229.95$   
 $39.99 * 3 = 119.97$   
 $39.99 * 8 = 319.92$   
 ttl: 16 669.84  
 - if credit card, then - 10%  
 $669.84 * 0.1 = 66.984$   
 $669.84 - 66.984 = 602.86$

- when the result is subtraction from 100%, I use "1 -":  $x\% = 1 - y\%$  and using: `x.ToString("P1")`  
`ElseIf`, `AndAlso`, `OrElse`, `txt_KeyPress`, `e.KeyChar`, `ControlChars.Back`, `txt_Enter`,  
`txt_TextChanged`

14. In this exercise, you modify the Grade application from Exercise 13 in Chapter 3. Use Windows to copy the Grade Solution folder from the VB2017\Chap03 folder to the VB2017\Chap04 folder. Open the Grade Solution.sln file contained in the VB2017\Chap04\Grade Solution folder. Start the application and then click the Calculate button. The percentages display as NaN, which stands for Not a Number. The NaN message is a result of dividing a number by 0. Use a selection structure to display 0.0% rather than NaN as the percentages. Save the solution and then start and test the application. Now professionalize the interface by allowing the text boxes to accept only numbers and the Backspace key. Save the solution and then start and test the application.

```
1 Option Explicit On
2 Option Strict On
3 Option Infer Off
4 Class frmMain
5     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
6         Me.Close()
7     End Sub
8
9     Private Sub txtPassed_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtPassed.KeyPress
10        If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
11            e.Handled = True
12        End If
13    End Sub
14
15    Private Sub txtStudents_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtStudents.KeyPress
16        If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
17            e.Handled = True
18        End If
19    End Sub
20
21    Private Sub txtStudents_Enter(sender As Object, e As EventArgs) Handles txtStudents.Enter
22        txtStudents.SelectAll()
23    End Sub
24
25    Private Sub txtPassed_Enter(sender As Object, e As EventArgs) Handles txtPassed.Enter
26        txtPassed.SelectAll()
27    End Sub
```

```
28
29     Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
30         Dim intStudents As Integer
31         Dim intPassed As Integer
32         Dim dblTtlPassed As Double
33         Dim dblTtlFailed As Double
34         Integer.TryParse(txtStudents.Text, intStudents)
35         Integer.TryParse(txtPassed.Text, intPassed)
36
37         '1st mod: when no numbers -> use a S.S. to display 0.0% rather than NaN (Not a Number)
38         ' original:
39         'dblTtlPassed = intPassed / intStudents
40         ' had to drink 2 haine and hear _Happy.bsl to firuge out da beautiful "1-" 55555:
41         'dblTtlFailed = 1 - dblTtlPassed
42
43         If intPassed = 0 AndAlso intStudents = 0 OrElse intStudents < intPassed Then
44             dblTtlPassed = 0
45             dblTtlFailed = 0
46         ElseIf intPassed = 0 AndAlso intStudents > 0 Then
47             dblTtlPassed = 0
48             dblTtlFailed = intStudents / 100
49         ElseIf intStudents = 0 AndAlso intPassed > 0 Then
50             intStudents = 0
51             dblTtlFailed = 0
52             dblTtlPassed = 0
53         Else
54             dblTtlPassed = intPassed / intStudents
55             ' "1 -" because percentage calculation !!!:
56             dblTtlFailed = 1 - dblTtlPassed
57         End If
58         lblTtlPassed.Text = dblTtlPassed.ToString("P1")
59         lblTtlFailed.Text = dblTtlFailed.ToString("P1")
60         lblTtlStudents.Text = intStudents.ToString("N0")
61     End Sub
62
63     Private Sub btnClear_Click(sender As Object, e As EventArgs) Handles btnClear.Click
64         txtStudents.Text = String.Empty
65         txtPassed.Text = String.Empty
66         lblTtlStudents.Text = String.Empty
67         lblTtlPassed.Text = String.Empty
68         lblTtlFailed.Text = String.Empty
69     End Sub
70 
```

```

71  Private Sub txtStudents_TextChanged(sender As Object, e As EventArgs) Handles txtStudents.TextChanged
72      'cistirna #1_Global labels:
73      lblTtlFailed.Text = String.Empty
74      lblTtlPassed.Text = String.Empty
75      lblTtlStudents.Text = String.Empty
76  End Sub
77
78  Private Sub txtPassed_TextChanged(sender As Object, e As EventArgs) Handles txtPassed.TextChanged
79      'cistirna #1_Global labels:
80      lblTtlFailed.Text = String.Empty
81      lblTtlPassed.Text = String.Empty
82      lblTtlStudents.Text = String.Empty
83  End Sub
84 End Class

```

**Welcome back, Professor**

Percentage Calculator for Pass/Fail Students



How many students?

How many passed?

**Calculate percentage**

From the ttl amount of:  students,  
 Passed:   
 Failed:

**Clear values** **Exitus**

**Welcome back, Professor**

Percentage Calculator for Pass/Fail Students



How many students?

How many passed?

**Calculate percentage**

From the ttl amount of:  students,  
 Passed:   
 Failed:

**Clear values** **Exitus**

- using colon : to separate different instructions in one line

```
txt_Enter, txt_KeyPress, e.KeyChar, ControlChars.Back, txt_TextChanged,  
lbl.BackColor = SystemColors.Control, ToString("P2")
```

15. In this exercise, you modify the Sales application from Exercise 14 in Chapter 3. Use Windows to copy the Sales Solution folder from the VB2017\Chap03 folder to the VB2017\Chap04 folder. Open the Sales Solution.sln file contained in the VB2017\Chap04\Sales Solution folder. Start the application and then click the Calculate button. The percentages display as NaN, which stands for Not a Number. The NaN message is a result of dividing a number by 0. Use a selection structure to display 0.0% rather than NaN as the percentages. Save the solution and then start and test the application. Now professionalize the interface by allowing the text boxes to accept only numbers, the period, and the Backspace key. Save the solution and then start and test the application.

```
1 Option Explicit On  
2 Option Strict On  
3 Option Infer Off  
4 Public Class frmMain  
5     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click  
6         Me.Close()  
7     End Sub  
8  
9     Private Sub txtJim_TextChanged(sender As Object, e As EventArgs) Handles txtJim.TextChanged  
10        lblTotal.Text = String.Empty : lblTotal.BackColor = SystemColors.Control  
11        lblPEJim.Text = String.Empty : lblPEJim.BackColor = SystemColors.Control  
12        lblPEKaren.Text = String.Empty : lblPEKaren.BackColor = SystemColors.Control  
13        lblPEMartin.Text = String.Empty : lblPEMartin.BackColor = SystemColors.Control  
14    End Sub  
15  
16    Private Sub txtKaren_TextChanged(sender As Object, e As EventArgs) Handles txtKaren.TextChanged  
17        lblTotal.Text = String.Empty : lblTotal.BackColor = SystemColors.Control  
18        lblPEKaren.Text = String.Empty : lblPEKaren.BackColor = SystemColors.Control  
19        lblPEJim.Text = String.Empty : lblPEJim.BackColor = SystemColors.Control  
20        lblPEMartin.Text = String.Empty : lblPEMartin.BackColor = SystemColors.Control  
21    End Sub  
22  
23    Private Sub txtMartin_TextChanged(sender As Object, e As EventArgs) Handles txtMartin.TextChanged  
24        lblTotal.Text = String.Empty : lblTotal.BackColor = SystemColors.Control  
25        lblPEMartin.Text = String.Empty : lblPEMartin.BackColor = SystemColors.Control  
26        lblPEJim.Text = String.Empty : lblPEJim.BackColor = SystemColors.Control  
27        lblPEKaren.Text = String.Empty : lblPEKaren.BackColor = SystemColors.Control  
28    End Sub  
29
```

```
30  Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
31      Dim dblJim As Double
32      Dim dblKaren As Double
33      Dim dblMartin As Double
34      Dim dblTotal As Double
35      Dim dblPEJim As Double
36      Dim dblPEKaren As Double
37      Dim dblPEMartin As Double
38
39      Double.TryParse(txtJim.Text, dblJim)
40      Double.TryParse(txtKaren.Text, dblKaren)
41      Double.TryParse(txtMartin.Text, dblMartin)
42
43      'original:
44      dblTotal = dblJim + dblKaren + dblMartin
45      'dblPEJim = dblJim / dblTotal
46      'dblPEKaren = dblKaren / dblTotal
47      'dblPEMartin = dblMartin / dblTotal
48
49      '1st mod:
50      If dblJim = 0 Then
51          dblPEJim = 0
52      Else dblPEJim = dblJim / dblTotal
53      End If
54
55      If dblKaren = 0 Then
56          dblPEKaren = 0
57      Else dblPEKaren = dblKaren / dblTotal
58      End If
59
60      If dblMartin = 0 Then
61          dblPEMartin = 0
62      Else dblPEMartin = dblMartin / dblTotal
63      End If
64
65      lblTotal.Text = dblTotal.ToString("C2")
66      lblPEJim.Text = dblPEJim.ToString("P2")
67      lblPEKaren.Text = dblPEKaren.ToString("P2")
68      lblPEMartin.Text = dblPEMartin.ToString("P2")
69      lblTotal.BackColor = Color.Yellow
70      lblPEJim.BackColor = Color.Yellow
71      lblPEKaren.BackColor = Color.Yellow
72      lblPEMartin.BackColor = Color.Yellow
73  End Sub
```

```

74
75     Private Sub txtJim_Enter(sender As Object, e As EventArgs) Handles txtJim.Enter
76         txtJim.SelectAll()
77     End Sub
78
79     Private Sub txtKaren_Enter(sender As Object, e As EventArgs) Handles txtKaren.Enter
80         txtKaren.SelectAll()
81     End Sub
82
83     Private Sub txtMartin_Enter(sender As Object, e As EventArgs) Handles txtMartin.Enter
84         txtMartin.SelectAll()
85     End Sub
86
87     Private Sub txtJim_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtJim.KeyPress
88         If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> "." AndAlso e.KeyChar <> ControlChars.Back Then
89             e.Handled = True
90         End If
91     End Sub
92
93     Private Sub txtKaren_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtKaren.KeyPress
94         If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> "." AndAlso e.KeyChar <> ControlChars.Back Then
95             e.Handled = True
96         End If
97     End Sub
98
99     Private Sub txtMartin_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtMartin.KeyPress
100        If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> "." AndAlso e.KeyChar <> ControlChars.Back Then
101            e.Handled = True
102        End If
103    End Sub
104 End Class

```

**Sales Percentage Calculator**

|                                          |                                     |                                     |                                    |
|------------------------------------------|-------------------------------------|-------------------------------------|------------------------------------|
| Enter amount for Jim:                    | <input type="text"/>                | Percentage:                         | <input type="text" value="0.00%"/> |
| Enter amount for Karen:                  | <input type="text"/>                | Percentage:                         | <input type="text" value="0.00%"/> |
| Enter amount for Martin:                 | <input type="text"/>                | Percentage:                         | <input type="text" value="0.00%"/> |
| Total amount:                            | <input type="text" value="\$0.00"/> |                                     |                                    |
| <input type="button" value="Calculate"/> |                                     | <input type="button" value="Exit"/> |                                    |

**Sales Percentage Calculator**

|                                          |                                       |                                     |                                     |
|------------------------------------------|---------------------------------------|-------------------------------------|-------------------------------------|
| Enter amount for Jim:                    | <input type="text" value="100"/>      | Percentage:                         | <input type="text" value="40.00%"/> |
| Enter amount for Karen:                  | <input type="text" value="100"/>      | Percentage:                         | <input type="text" value="40.00%"/> |
| Enter amount for Martin:                 | <input type="text" value="50"/>       | Percentage:                         | <input type="text" value="20.00%"/> |
| Total amount:                            | <input type="text" value="\$250.00"/> |                                     |                                     |
| <input type="button" value="Calculate"/> |                                       | <input type="button" value="Exit"/> |                                     |

## EXERCISE 16.OnYourOwn Solution

- rad.Checked, ElseIf, chk.Checked, txt\_Enter, txt\_TextChanged, rad\_CheckedChanged,  
chk\_CheckedChanged, txt\_KeyPress, e.KeyChar, ControlChars.Back

Create a Windows Forms application. Use the following names for the project and solution, respectively: OnYourOwn Project and OnYourOwn Solution. Save the application in the VB2017\Chap04 folder. Plan and design an application of your choice. The only requirement is that you must follow the minimum guidelines listed in Figure 4-63. Before starting the application, be sure to verify the name of the startup form. Save the solution and then start and test the application.

1. The user interface must contain a minimum of one text box, three labels, two radio buttons, one check box, and two buttons. One of the buttons must be an Exit button.
2. The interface can include a picture box, but this is not a requirement.

3. The interface must follow the GUI design guidelines summarized in Figure 2-20 in Chapter 2 and in Figure 4-52 in Chapter 4. (The guidelines are also listed in Appendix A.)
4. Objects that are either coded or referred to in code should be named appropriately.
5. The Code Editor window must contain comments, the three Option statements, at least two variables, at least two assignment statements, the Me.Close() statement, and at least one selection structure. The application must perform at least one calculation.
6. Every text box on the form should have its TextChanged and Enter event procedures coded. At least one of the text boxes should have its KeyPress event procedure coded.
7. Every radio button and check box should have its CheckedChanged event procedure coded.

```

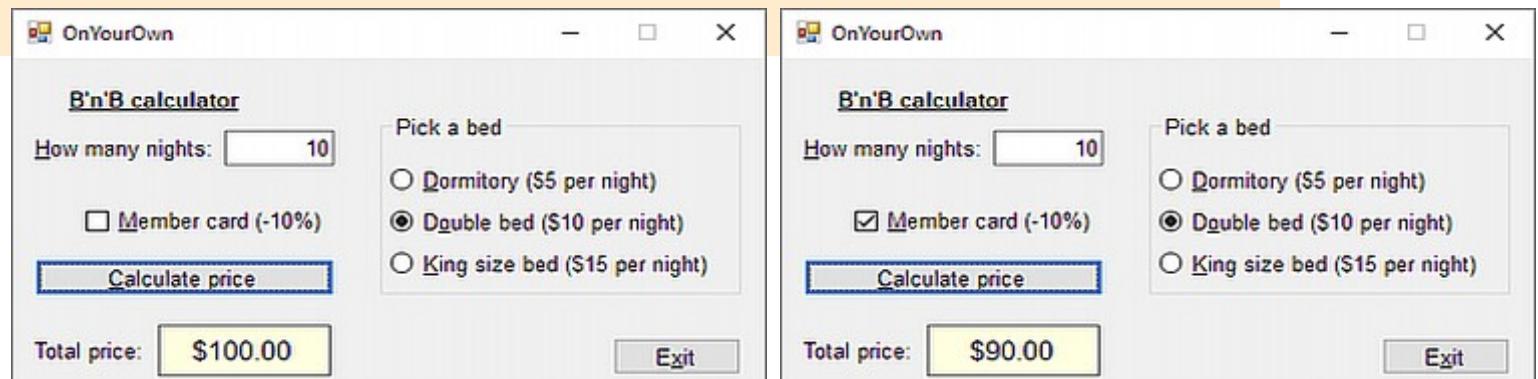
1  Option Explicit On
2  Option Strict On
3  Option Infer Off
4
5  Public Class frmMain
6      Private Sub btnCalculate_Click(sender As Object, e As EventArgs) Handles btnCalculate.Click
7          Dim intNights As Integer
8          Dim dblTotal As Double
9          Dim intBed As Integer
10         Integer.TryParse(txtNights.Text, intNights)
11
12         If rad1.Checked Then
13             intBed = 5
14         ElseIf rad2.Checked Then
15             intBed = 10
16         ElseIf rad3.Checked Then
17             intBed = 15
18         End If
19
20         If chkMember.Checked Then
21             dblTotal = intNights * intBed - ((intNights * intBed) * 0.1)
22         Else dblTotal = intNights * intBed
23         End If
24
25         lblTotal.Text = dblTotal.ToString("C2")
26     End Sub
27

```

```

28     Private Sub txtNights_Enter(sender As Object, e As EventArgs) Handles txtNights.Enter
29         txtNights.SelectAll()
30     End Sub
31
32     Private Sub txtNights_TextChanged(sender As Object, e As EventArgs) Handles txtNights.TextChanged
33         lblTotal.Text = String.Empty
34     End Sub
35
36     Private Sub chkMember_CheckedChanged(sender As Object, e As EventArgs) Handles chkMember.CheckedChanged
37         lblTotal.Text = String.Empty
38     End Sub
39
40     Private Sub rad1_CheckedChanged(sender As Object, e As EventArgs) Handles rad1.CheckedChanged
41         lblTotal.Text = String.Empty
42     End Sub
43
44     Private Sub rad2_CheckedChanged(sender As Object, e As EventArgs) Handles rad2.CheckedChanged
45         lblTotal.Text = String.Empty
46     End Sub
47
48     Private Sub rad3_CheckedChanged(sender As Object, e As EventArgs) Handles rad3.CheckedChanged
49         lblTotal.Text = String.Empty
50     End Sub
51
52     Private Sub txtNights_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtNights.KeyPress
53         If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
54             e.Handled = True
55         End If
56     End Sub
57
58     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
59         Me.Close()
60     End Sub
61 End Class

```



## Chap04\Exercise1

### EXERCISE 17.FixIt Solution

- **Select Case, Case Is, txt\_Enter, txt\_TextChanged, txt\_KeyPress, e.KeyChar, ControlChars.Back**

The purpose of this exercise is to demonstrate the importance of testing an application thoroughly. Open the FixIt Solution.sln file contained in the VB2017\Chap04\FixIt Solution folder. The application displays a shipping charge that is based on the total price entered by the user, as shown in Figure 4-64. Start the application and then test it by clicking the Display shipping button. Notice that the Shipping charge box contains \$13, which is not correct. Now, test the application using the following total prices: 100, 501, 1500, 500.75, 30, 1000.33, and 2000. Here too, notice that the application does not always display the correct shipping charge. (More specifically, the shipping charge for two of the seven total prices is incorrect.) Open the Code Editor window and correct the errors in the code. Save the solution and then start and test the application.

```
1  ' Name:      FixIt Project
2  ' Purpose:    Displays a shipping charge based on a total price
3  ' Programmer: <your name> on <current date>
4
5  Option Explicit On
6  Option Strict On
7  Option Infer Off
8
9  Public Class frmMain
10     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
11         Me.Close()
12     End Sub
13
14     Private Sub txtTotal_Enter(sender As Object, e As EventArgs) Handles txtTotal.Enter
15         txtTotal.SelectAll()
16     End Sub
17
18     Private Sub txtTotal_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtTotal.KeyPress
19         ' accept only numbers, the period, and the Backspace key
20         If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> "." AndAlso e.KeyChar <> ControlChars.Back Then
21             e.Handled = True
22         End If
23     End Sub
24
25     Private Sub txtTotal_TextChanged(sender As Object, e As EventArgs) Handles txtTotal.TextChanged
26         lblShipping.Text = String.Empty
27     End Sub
```

| Total price                          | Shipping |
|--------------------------------------|----------|
| Less than \$1                        | \$ 0     |
| At least \$1 but less than \$100     | \$13     |
| At least \$100 but less than \$501   | \$10     |
| At least \$501 but less than \$1,001 | \$ 7     |
| At least \$1,001                     | \$ 5     |

Figure 4-64 Shipping charges for Exercise 17

```
28
29     Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
30         'displays a shipping charge
31         Dim dblTotal As Double
32         Dim intShipping As Integer
33         Double.TryParse(txtTotal.Text, dblTotal)
34
35         ' original:
36         ' wrong with: 0, 500.75, 1000.33
37         'Select Case dblTotal
38             'Case Is < 100
39             'intShipping = 13
40             'Case 100 To 500
41             'intShipping = 10
42             'Case 501 To 1000
43             'intShipping = 7
44             'Case Else
45             'intShipping = 5
46         'End Select
47
48         ' fixed:
49         Select Case dblTotal
50             Case Is < 1
51                 intShipping = 0
52             Case Is < 100
53                 intShipping = 13
54             Case Is < 501
55                 intShipping = 10
56             Case Is < 1001
57                 intShipping = 7
58             Case Is >= 1001
59                 intShipping = 5
60         End Select
61
62         lblShipping.Text = intShipping.ToString("C0")
63     End Sub
64 End Class
```

original:

Shipping Charge

Total price:

Shipping charge:

**Display shipping** **Exit**

Shipping Charge

Total price:

Shipping charge:

**Display shipping** **Exit**

Shipping Charge

Total price:

Shipping charge:

**Display shipping** **Exit**

fixed:

Shipping Charge

Total price:

Shipping charge:

**Display shipping** **Exit**

Shipping Charge

Total price:

Shipping charge:

**Display shipping** **Exit**

Shipping Charge

Total price:

Shipping charge:

**Display shipping** **Exit**

## Filmore's Fast Food (Chapters 1–4)

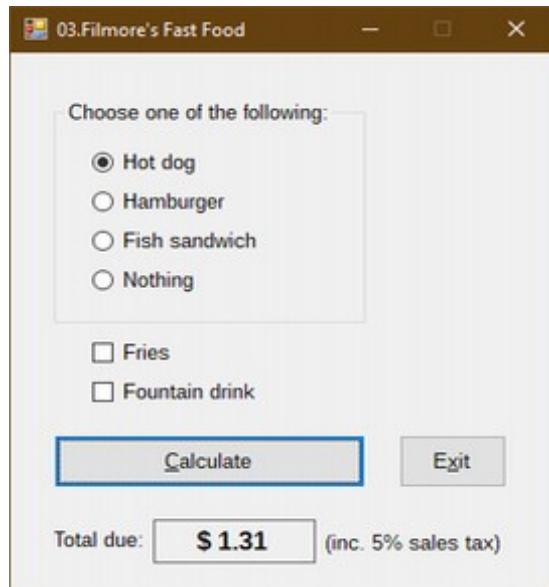
Create an application for Filmore's Fast Food restaurant. The restaurant sells hot dogs for \$1.25, hamburgers for \$2.50, fish sandwiches for \$2.97, fries for \$0.99, and fountain drinks for \$1.49. Create an interface that allows the user to enter a customer's order. The customer can choose only one of the following: a hot dog, a hamburger, or a fish sandwich. However, he or she can also order only fries, only a fountain drink, or both fries and a fountain drink. The application should calculate and display the total due, which should include a 5% sales tax. (For example, if the customer orders only fries, the total due is \$1.04. If he or she orders a hot dog and a drink, the total due is \$2.88.)

```
1  ' Name:      Filmore's Fast Food.
2  ' Purpose:    Calculate and display customer's order.
3  ' Programmer: me on just now.
4  Option Explicit On
5  Option Strict On
6  Option Infer Off
7
8  Public Class frmMain
9      Private Sub btnCalculate_Click(sender As Object, e As EventArgs) Handles btnCalculate.Click
10         ' rad1HotDog = 1.25 or rad2Hamburger = 2.50 or rad3Fish = 2.97 or rad4Nothing; chk1Fries = 0.99, chk2Drink = 1.49
11
12         Const decSalesTax5 As Decimal = 0.05D : Dim decSub As Decimal : Dim decTotal As Decimal
13
14         If rad1HotDog.Checked Then
15             decSub = 1.25D
16         End If
17
18         If rad2Hamburger.Checked Then
19             decSub = 2.5D
20         End If
21
22         If rad3Fish.Checked Then
23             decSub = 2.97D
24         End If
25
26         If rad4Nothing.Checked Then
27             decSub = 0
28         End If
29
30         If chk1Fries.Checked Then
31             decSub += 0.99D
32         End If
```

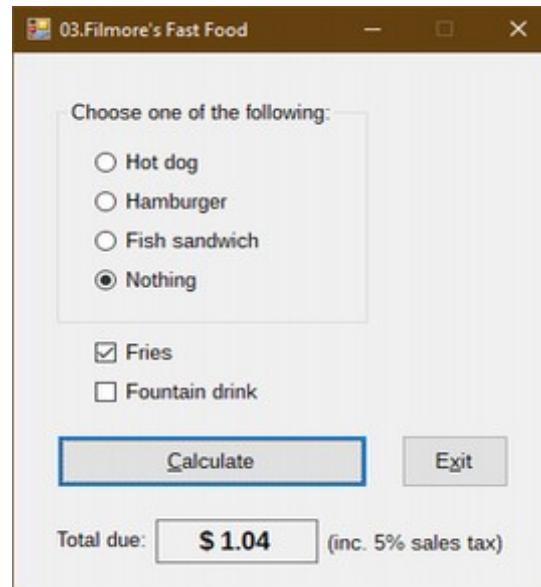
```

33
34     If chk2Drink.Checked Then
35         decSub += 1.49D
36     End If
37
38     decTotal = decSub * decSalesTax5 + decSub
39     lblTotal.Text = decTotal.ToString("C2")
40 End Sub
41
42 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
43     Me.Close()
44 End Sub
45
46 Private Sub chkrad_CheckedChanged(sender As Object, e As EventArgs) Handles chk1Fries.CheckedChanged, chk2Drink.CheckedChanged,
47     rad1HotDog.CheckedChanged, rad2Hamburger.CheckedChanged, rad3Fish.CheckedChanged, rad4Nothing.CheckedChanged
48     lblTotal.Text = Nothing
49 End Sub
50 End Class

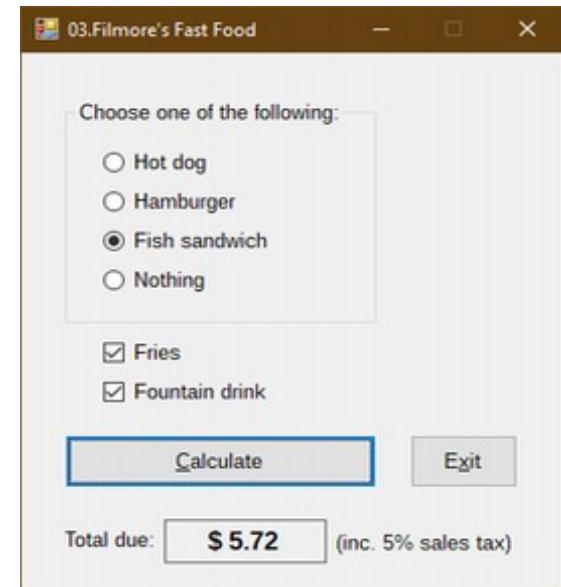
```



1. only hotdog: = 1.25  
 $1.25 * 0.05 = 0.0625$   
 total: = **1.3125**



2. only fries: = 0.99  
 $0.99 * 0.05 = 0.0495$   
 total: = **1.0395**



3. fish sandwich = 2.97  
 fries = 0.99  
 drink = 1.49  
 $= 5.45$   
 $5.45 * 0.05 = 0.2725$   
 total: = **5.7225**

info: hot dog = 1.25 or hamburger = 2.50 or fish sandwich = 2.97, fries = 0.99, drink = 1.49, sales tax 5% = 0.05

Totales mandzáres - finito tutti - basta mucho

Pretest loops: **Do While/Do Until...Loop** & counter-controlled loop **For...Next**; Posttest loop: **Do...Loop While/Loop Until**; Nested; Counters & Accumulators; String's concatenation operator & **ListBox** (1st); **Financial Class**; default buttons: **AcceptButton** & **CancelButton**; Sys constants: **ControlChars.Back**, **ControlChars.NewLine**, **ControlChars.Tab**;

## CH5\_MY CHEAT SHEET

## CH5\_FOCUS ON THE CONCEPTS LESSON

- CH5\_F1 - Repetition Structure / Loop - basic info
- CH5\_F2 - Pretest loop: **Do While/Do Until...Loop** statement
- CH5\_F3 - String's concatenation operator: & (zřetězení), and constant: **ControlChars.NewLine**
- CH5\_F4 - Pretest loop: **Do While/Do Until...Loop** statement's **Infinite loops** = mistake by the programmer
- CH5\_F5 - Posttest loop: **Do ... Loop While/Loop Until** statement
- CH5\_F5.1 - Posttest loop: **Do ... Loop Until** e.g.: [30.Fibonacci Solution\\_EXERCISE 15](#)
- CH5\_F6 - Counters and Accumulators basic info & e.g.
- CH5\_F7 - Pretest loop: **For...As dataType/... = ...To...Step ... Next** statement - specific counter-controlled loop
- CH5\_F8 - Pretest loop: create counter-controlled loop - compare **Do...Loop** statement vs **For...Next** statement
- CH5\_F9 - Pretest loop: **For...Next** statement - specific counter-controlled loop - Flowcharting
- CH5\_F9.1 - Pretest loop: **For...Next** statement - create Counter and Accumulator - [07.You Do It 3 Solution](#)

## CH5\_APPLY THE CONCEPTS LESSON

- CH5\_A1 - Pretest loop: **Do...Loop** statement - use a **Loop**, **Counter**, and **Accumulator** - e.g. [08.Projected Sales Solution](#)
- CH5\_A1.1 - Pretest loop: **For...Next** statement - use a **Loop**, **Counter**, and **Accumulator** - e.g. [09.Projected Sales Solution-ForNext](#)
- CH5\_A2 - **ListBox** - add a (1st) to a **Form**: basic info & basic properties
- CH5\_A2.1 - **ListBox** - add items using the **String Collection editor** in Properties - e.g. [10.ListBox Solution](#)
- CH5\_A2.2 - **ListBox** - the property **Sorted** - e.g. [10.ListBox Solution](#)
- CH5\_A2.3 - **ListBox** - the **SelectedItem** and **SelectedIndex** properties - when **SelectionMode = One** - e.g. [10.ListBox Solution](#)
- CH5\_A2.3.1 - **ListBox** - the **SelectedItems** and **SelectedIndices** properties - when **SelectionMode = MultiSimple / MultiExtended** - [31.Multi Solution\\_EXERCISE 16](#)
- CH5\_A2.4 - **ListBox** - the **SelectedValueChanged** and **SelectedIndexChanged** events - e.g. [10.ListBox Solution](#)
- CH5\_A3.1 - **ListBox** - Items Collection: method **Add** to add items - e.g. [11.ListBox Items Solution](#)
- CH5\_A3.2 - **ListBox** - Items Collection: method **Insert** to add item at a desired position during run time
- CH5\_A3.3 - **ListBox** - Items Collection: method **Remove** to remove item during run time
- CH5\_A3.4 - **ListBox** - Items Collection: method **RemoveAt** to remove item during run time
- CH5\_A3.5 - **ListBox** - Items Collection: property **Count** to get the number of items - e.g. [11.ListBox Items Solution](#)
- CH5\_A3.6 - **ListBox** - Items Collection: method **Clear** to remove all of the items - e.g. [11.ListBox Items Solution](#)
- CH5\_A4 - **Financial Class** - introduction: index of **Financial**. methods
- CH5\_A4.1 - **Financial Class** - calculate a periodic payment using **Financial.Pmt** method
- CH5\_A4.2 - **Financial.Pmt** method & **ListBox & Loop** to calculate monthly mortgage payment - e.g. [13.Payment Solution](#)
- CH5\_A4.3 - **Financial.FV** method **research** - calculate the future value of an annuity e.g.: [Appendix E - Case Projects\\_04.Savings Calculator-CH 01-05](#)
- CH5\_A5 - Nested repetition structures / loops = outer loop contains nested/inner loop - basic info
- CH5\_A5.1 - Nested repetition structures / loops = outer loop contains nested/inner loop - e.g. using 2x Pretest loop **For...Next**, e.g. [14.Savings Solution](#)
- CH5\_A6 - a caution about **Real numbers** = decimals: **As Double** less accurate than **As Decimal** -> can cause wrong result, e.g. [15.Savings Solution-Caution](#)
- CH5\_A7 - professionalize your application's interface: the **default buttons** set in **frmMain**'s Properties: **AcceptButton** = Enter key, **CancelButton** = Esc key

## CH5\_Key Terms

## CH5\_Exercises

## CH5\_MY CHEAT SHEET

|                                                                                                                                                                                                                                                                                                                                                                                                                                        |                                                                                                                                                                  |                                                                                                            |                                                               |                |     |                    |             |                |         |  |       |  |               |                                                                                                                                                                                                                                                                                                                                                                                                                                |  |                   |                                     |                      |                                     |                  |                                               |
|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------|----------------|-----|--------------------|-------------|----------------|---------|--|-------|--|---------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|-------------------|-------------------------------------|----------------------|-------------------------------------|------------------|-----------------------------------------------|
| F2                                                                                                                                                                                                                                                                                                                                                                                                                                     | <pre>Do While + condition or Do Until + condition     + loop body instructions Loop</pre>                                                                        | <b>pretest loop</b> with <u>looping</u> condition<br><b>pretest loop</b> with <u>loop exit</u> condition   | <pre>Do While intNum &lt;= 5 Do Until intNum &gt; 5</pre>     |                |     |                    |             |                |         |  |       |  |               |                                                                                                                                                                                                                                                                                                                                                                                                                                |  |                   |                                     |                      |                                     |                  |                                               |
| F5                                                                                                                                                                                                                                                                                                                                                                                                                                     | <pre>Do     + loop body instructions Loop While + condition or Loop Until + condition</pre>                                                                      | <b>posttest loop</b> with <u>looping</u> condition<br><b>posttest loop</b> with <u>loop exit</u> condition | <pre>Loop While intNum &lt;= 5 Loop Until intNum &gt; 5</pre> |                |     |                    |             |                |         |  |       |  |               |                                                                                                                                                                                                                                                                                                                                                                                                                                |  |                   |                                     |                      |                                     |                  |                                               |
| F7                                                                                                                                                                                                                                                                                                                                                                                                                                     | <pre>For newCounter As dataType/existingVariable = startValue To endValue Step stepValue     ...LoopBody instructions Next newCounterName/existingVariable</pre> |                                                                                                            | <b>pretest loop</b> - specific counter-controlled loop        |                |     |                    |             |                |         |  |       |  |               |                                                                                                                                                                                                                                                                                                                                                                                                                                |  |                   |                                     |                      |                                     |                  |                                               |
| <u>scope</u> from smallest:<br><table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 50%;">variable</td> <td style="width: 50%;">constant</td> </tr> <tr> <td>1. Block-level</td> <td>For</td> </tr> <tr> <td>2. Procedure-level</td> <td>Dim, Static</td> </tr> <tr> <td>3. Class-level</td> <td>Private</td> </tr> <tr> <td></td> <td>Const</td> </tr> <tr> <td></td> <td>Private Const</td> </tr> </table> |                                                                                                                                                                  | variable                                                                                                   | constant                                                      | 1. Block-level | For | 2. Procedure-level | Dim, Static | 3. Class-level | Private |  | Const |  | Private Const | <b>some of the System Constants:</b><br><table style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 50%;">ControlChars.Back</td> <td style="width: 50%;">-&gt; the <b>Backspace</b> key = erase</td> </tr> <tr> <td>ControlChars.NewLine</td> <td>-&gt; the <b>Enter</b> key = next line</td> </tr> <tr> <td>ControlChars.Tab</td> <td>-&gt; the <b>Tab</b> key = align the information</td> </tr> </table> |  | ControlChars.Back | -> the <b>Backspace</b> key = erase | ControlChars.NewLine | -> the <b>Enter</b> key = next line | ControlChars.Tab | -> the <b>Tab</b> key = align the information |
| variable                                                                                                                                                                                                                                                                                                                                                                                                                               | constant                                                                                                                                                         |                                                                                                            |                                                               |                |     |                    |             |                |         |  |       |  |               |                                                                                                                                                                                                                                                                                                                                                                                                                                |  |                   |                                     |                      |                                     |                  |                                               |
| 1. Block-level                                                                                                                                                                                                                                                                                                                                                                                                                         | For                                                                                                                                                              |                                                                                                            |                                                               |                |     |                    |             |                |         |  |       |  |               |                                                                                                                                                                                                                                                                                                                                                                                                                                |  |                   |                                     |                      |                                     |                  |                                               |
| 2. Procedure-level                                                                                                                                                                                                                                                                                                                                                                                                                     | Dim, Static                                                                                                                                                      |                                                                                                            |                                                               |                |     |                    |             |                |         |  |       |  |               |                                                                                                                                                                                                                                                                                                                                                                                                                                |  |                   |                                     |                      |                                     |                  |                                               |
| 3. Class-level                                                                                                                                                                                                                                                                                                                                                                                                                         | Private                                                                                                                                                          |                                                                                                            |                                                               |                |     |                    |             |                |         |  |       |  |               |                                                                                                                                                                                                                                                                                                                                                                                                                                |  |                   |                                     |                      |                                     |                  |                                               |
|                                                                                                                                                                                                                                                                                                                                                                                                                                        | Const                                                                                                                                                            |                                                                                                            |                                                               |                |     |                    |             |                |         |  |       |  |               |                                                                                                                                                                                                                                                                                                                                                                                                                                |  |                   |                                     |                      |                                     |                  |                                               |
|                                                                                                                                                                                                                                                                                                                                                                                                                                        | Private Const                                                                                                                                                    |                                                                                                            |                                                               |                |     |                    |             |                |         |  |       |  |               |                                                                                                                                                                                                                                                                                                                                                                                                                                |  |                   |                                     |                      |                                     |                  |                                               |
| ControlChars.Back                                                                                                                                                                                                                                                                                                                                                                                                                      | -> the <b>Backspace</b> key = erase                                                                                                                              |                                                                                                            |                                                               |                |     |                    |             |                |         |  |       |  |               |                                                                                                                                                                                                                                                                                                                                                                                                                                |  |                   |                                     |                      |                                     |                  |                                               |
| ControlChars.NewLine                                                                                                                                                                                                                                                                                                                                                                                                                   | -> the <b>Enter</b> key = next line                                                                                                                              |                                                                                                            |                                                               |                |     |                    |             |                |         |  |       |  |               |                                                                                                                                                                                                                                                                                                                                                                                                                                |  |                   |                                     |                      |                                     |                  |                                               |
| ControlChars.Tab                                                                                                                                                                                                                                                                                                                                                                                                                       | -> the <b>Tab</b> key = align the information                                                                                                                    |                                                                                                            |                                                               |                |     |                    |             |                |         |  |       |  |               |                                                                                                                                                                                                                                                                                                                                                                                                                                |  |                   |                                     |                      |                                     |                  |                                               |

### ListBox (1st)

- property: **SelectionMode** = None / One / MultiSimple / MultiExtended
- property: **Sorted** = True / False - dictionary leftmost order: numbers -> lowercase -> UPPERCASE

|                                                      |                                                                                         |
|------------------------------------------------------|-----------------------------------------------------------------------------------------|
| <code>objectListBox.Items.Add(item)</code>           | -> method <b>add</b> item using method, mostly entered in a <b>Load event procedure</b> |
| <code>objectListBox.Items.Insert(index, item)</code> | -> method <b>add</b> item at a desired position during run time                         |
| <code>objectListBox.Items.Remove(item)</code>        | -> method <b>remove</b> item during run time                                            |
| <code>objectListBox.Items.RemoveAt(index)</code>     | -> method <b>remove</b> indexed item during run time                                    |
| <code>objectListBox.Items.Clear()</code>             | -> method <b>clear</b> all items during run time                                        |

|      |                                                 |
|------|-------------------------------------------------|
| e.g. | <code>1stNames.Items.Add("Hubert")</code>       |
| e.g. | <code>1stNames.Items.Insert(3, "Hubert")</code> |
| e.g. | <code>1stNames.Items.Remove("Hubert")</code>    |
| e.g. | <code>1stNames.Items.RemoveAt(3)</code>         |
| e.g. | <code>1stNames.Items.Clear()</code>             |

```
lblItems.Text = 1stNames.Items.Count.ToString
1stNames.SelectedItem = "Hubert"
1stNames.SelectedIndex = 3
lblItem.Text = 1stNames.SelectedItem.ToString
lblIndex.Text = 1stNames.SelectedIndex.ToString
```

-> property stores number of items  
 -> property select item - when property **SelectionMode** = One  
 -> property select index - when property **SelectionMode** = One  
 -> assign selected item during run time - when property **SelectionMode** = One  
 -> assign selected indexed item during run time - when property **SelectionMode** = One

```
For a As Integer = 0 To 1stNames.SelectedItems.Count - 1
    lblItems.Text += 1stNames.SelectedItems.Item(a).ToString & ControlChars.NewLine
    lblIndexes.Text += 1stNames.SelectedIndexes.Item(a).ToString & ControlChars.NewLine
Next a
```

- when property **SelectionMode** = MultiSimple / MultiExtended  
 -> assign all selected items  
 -> assign all selected indexes

```
Private Sub 1stNames_SelectedIndexChanged(sender...
Private Sub 1stNames_SelectedValueChanged(sender...
```

-> event what to do when selected different index during run time  
 -> event what to do when select different item during run time

## CH5\_FOCUS ON THE CONCEPTS LESSON

### CH5\_F1 - Repetition Structure / Loop - basic info

- coders use the **repetition structure**, referred to more simply as a **loop**, when they need the computer to repeatedly process one or more program instructions
- the **loop** contains a condition that controls whether the instructions are repeated
- in many programming languages, the condition can be phrased in **1 of 2 ways**:

- 1). it can either specify the requirement for **repeating** the instructions or
  - = **looping condition**, because it indicates when the computer should continue “looping” through the instructions
    - e.g.: Leave your car’s windshield wipers on **while it is raining.**
    - e.g.: Listen **while the speaker is talking.**
- 2). specify the requirement for **not repeating** them
  - = **loop exit condition**, because it tells the computer when to exit (or stop) the loop
    - e.g.: Leave your car’s windshield wipers on **until it stops raining.**
    - e.g.: Listen **until the speaker stops talking.**

- every **looping condition** has an opposing **loop exit condition**

- the condition in a loop can appear at:

- 1) either the **top** of the loop     = **pretest loop**     - may never be processed
  - condition is evaluated **before** the instructions within the loop are processed
- 2) or the **bottom** of the loop     = **posttest loop**     - will always be processed at least once
  - condition is evaluated **after** the instructions within the loop are processed

- the difference between a **pretest loop** and **posttest loop** is that the instructions in a **posttest loop** will always be processed at least once, whereas the instruction in a **pretest loop** may never be processed

- e.g. if the **intNum** variable used in the e.g. **Figure 5-2** was initialized to 6 (rather than to 1):
  - the conditions in both **pretest loops** would prevent the instructions in those loops from being processed
  - the instructions in both **posttest loops**, on the other hand, would be processed once before the conditions were evaluated the first time

- **the Visual Basic provides 3 different statements for coding loops:**
- |                            |            |
|----------------------------|------------|
| - <b>Do ... Loop</b>       | <b>CH5</b> |
| - <b>For ... Next</b>      | <b>CH5</b> |
| - <b>For Each ... Next</b> | <b>CH8</b> |

### CH5\_F2 - Pretest loop: Do While/Do Until...Loop statement

- looping / loop exit     - may never be processed     = infinite loop = mistake by Coder

- condition is evaluated **before** the instructions within the loop are processed

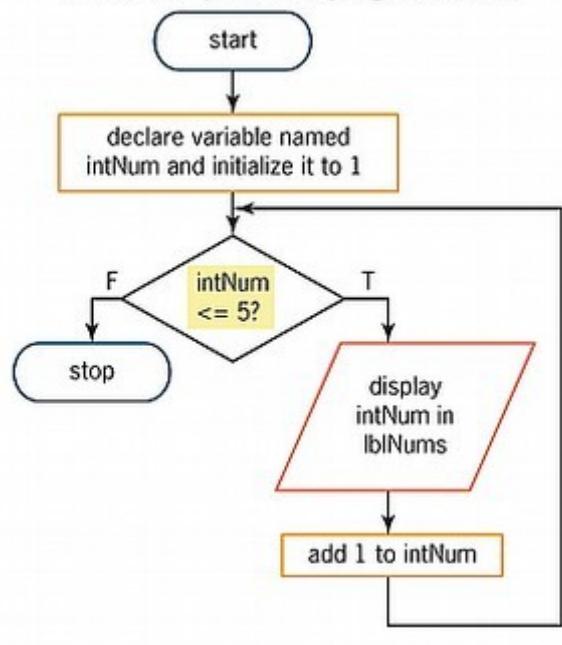
- syntax:
- |                                                                                                             |                                                                                                        |
|-------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------|
| <b>Do While</b> + condition<br>or<br><b>Do Until</b> + condition<br>+ loop body instructions<br><b>Loop</b> | = looping condition (as long as the condition is <b>TRUE</b> )                                         |
|                                                                                                             | = loop exit condition (until the condition becomes <b>TRUE</b> )                                       |
|                                                                                                             | = to be processed either while the condition is <b>TRUE</b> or until the condition becomes <b>TRUE</b> |

- can contain: **variables, named constants, literals, properties, methods, keywords, operators**     (like the condition in an **If...Then...Else** statement)
- must evaluate to a **Boolean value**     (like the condition in an **If...Then...Else** statement)
- the condition is evaluated with each repetition of the loop and determines whether the computer processes the loop body

e.g. Figure 5-2 & Figure 5-3 & Figure 5-4: pseudocode & flowchart & code: display the numbers 1 through 5 in a label control:

(the loop will stop when the value in intNum is 6)

### Pretest loop with looping condition



### Pretest loop with looping condition:

1. declare variable named intNum and initialize it to 1
2. repeat while **intNum <= 5**  
display intNum in lblNums  
add 1 to intNum  
end repeat while

**Dim intNum As Integer = 1**

**Do While intNum <= 5** <- specifies when to continue

**lblNums.Text = lblNums.Text & intNum.ToString & " "**

**intNum += 1** &= concatenation operator

**Loop**

### Pretest loop with loop exit condition:

1. declare variable named intNum and initialize it to 1
2. repeat until **intNum > 5**  
display intNum in lblNums  
add 1 to intNum  
end repeat while

**Dim intNum As Integer = 1**

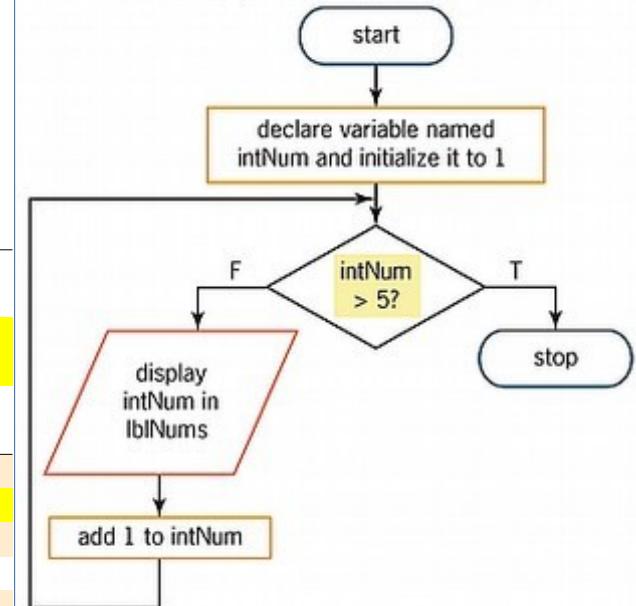
**Do Until intNum > 5** <- specifies when to stop

**lblNums.Text = lblNums.Text & intNum.ToString & " "**

**intNum += 1** &= concatenation operator

**Loop**

### Pretest loop with loop exit condition



<- result of using either of:

- Pretest loop with looping condition or
- Pretest loop with loop exit condition

- the loop will stop when the value in intNum is 6

+ 5x Space characters

e.g.: To code and then test the **Do...Loop Pretest loop** application:

1. open the ...VB2017\Chap05\Exercise01.Do Loop Pretest Solution\Do Loop Pretest Solution.sln
2. in a Code editor window locate the: **btnDisplay\_Click** procedure

- enter the Do...Loop Pretest statement by your choice (there are 5x Space characters between the quotation marks)

3. save & test

- it will display the numbers from 1 to 5 in the label control

4. in a **Dim** statement change the **1** to **6**

- save & test

- no numbers appear in the label control because:

- 1). if you used: **Do While intNum <= 5**
  - the computer skipped over the instructions in the loop body because the **intNum <= 5** looping condition evaluated to **FALSE**
- 2). if you used: **Do Until intNum > 5**
  - the instructions in the loop body were skipped over because the **intNum > 5** loop exit condition evaluated to **TRUE**

5. change back 6 to 1

### Mini-Quiz 5-1

1. Using the **While** keyword, write a **Do** clause that processes the loop body as long as the value in the **intAge** variable is greater than **21**.
2. Rewrite the **Do** clause from Question 1 using the **Until** keyword.
3. Using the **While** keyword, write a **Do** clause that processes the loop body as long as the value in the **dblScore** variable is greater than **0** and, at the same time, less than or equal to **100**.
4. Using the **Until** keyword, write a **Do** clause that stops the loop when the value in the **strContinue** variable contains the letter **N** (in either uppercase or lowercase).

```
1...Do While intAge > 21  
2...Do Until intAge <= 21  
3...Do While dblScore > 0 AndAlso dblScore <= 100  
4...Do Until strContinue.ToUpper().Trim = "N"
```

### CH5\_F3 - String's concatenation operator: & (zřetězení), and constant: **ControlChars.NewLine**

- you use the **concatenation operator**, which is the ampersand - **&**, to concatenate strings (connect or link together)
- **ControlChars.NewLine** = constant which advances the insertion point to the next line

e.g. Concatenating Strings: **(Figure 5-5)**

| variables                                                                                                           | content |
|---------------------------------------------------------------------------------------------------------------------|---------|
| <pre>Dim strCity As String = "Atlanta"<br/>Dim strState As String = "GA"<br/>Dim intSalary As Integer = 42500</pre> |         |

concatenated string:

|                                                                                                                                                                                                                                                              |                                                                                                                                  |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------|
| <pre>strCity &amp; strState<br/>strCity &amp; " " &amp; strState<br/>strCity &amp; ", " &amp; strState<br/>"She lives in " &amp; strCity &amp; "."<br/>"Salary: " &amp; intSalary.ToString("C0")<br/>strCity &amp; ControlChars.NewLine &amp; strState</pre> | <p>result:</p> <p>AtlantaGA<br/>Atlanta GA<br/>Atlanta, GA<br/>She lives in Atlanta.<br/>Salary: \$42,500<br/>Atlanta<br/>GA</p> |
|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------|

You Do It 1:

1. create an app named You Do It 1
2. add a **lbl** and **btn**
3. the **btn** should use a **pretest loop** and the **concatenation operator** to display the following numbers in the **lbl**: **1, 3, 5, 7**.

```
Dim intNum As Integer = 1  
Do While intNum <= 7  
    lblShow.Text = lblShow.Text & intNum.ToString & " "  
    intNum += 2  
Loop
```

### CH5\_F4 - Pretest loop: **Do While/Do Until...Loop** statement's Infinite loops = mistake by the programmer

- a loop that has no way to end
- is many times created when the **loop body** does **not contain** an instruction that will make either:
  - the looping condition evaluate to **FALSE** or **(Do While)**
  - the loop exit condition evaluate to **TRUE** **(Do Until)**
- you can STOP an infinitive loop:
  - 1). by clicking on Menu Bar / Debug / Stop Debugging
  - 2). via click on the **red square** (Stop Debugging button) on the Standard toolbar

e.g. to create and then stop an infinitive loop:

1. open the ...VB2017\Chap05\_Exercise03.Infinite Pretest Loop Solution\Infinite Pretest Loop Solution.sln
2. locate the **btnDisplay\_Click** procedure
  - turn the **intNum += 1** assignment statement into a comment by inserting an apostrophe before it
    - notice that that the **loop body** no longer provides an **instruction** that can **change** the value stored in the **intNum** variable from its initial value of **1** to a value that can stop the loop - in this case **6**
3. save and start the app
  - click the **Display** button, wait a few seconds and then click the **Exit** button
    - notice that the **Exit** button does **not** respond to its Click event
  - click the **Display** button again
    - no response either

Neither button responds because the Do...Loop statement is in an infinitive loop =

= the loop has no way to STOP because without the **intNum += 1** statement, the **intNum** variable's value remains at **1**

4. click the **Stop Debugging** button (the red square) to stop the loop

5. remove the apostrophe from the assignment statement

6. save and exit

#### CH5\_F5 - Posttest loop: Do ... Loop While/Loop Until statement

syntax:

```
Do
    + loop body instructions
Loop While + condition
or
Loop Until + condition
```

- looping/loop exit - will always be processed at least once
- condition is evaluated **after** the instructions within the loop are processed

= looping condition (as long as the condition is **TRUE**)

= loop exit condition (until the condition becomes **TRUE**)

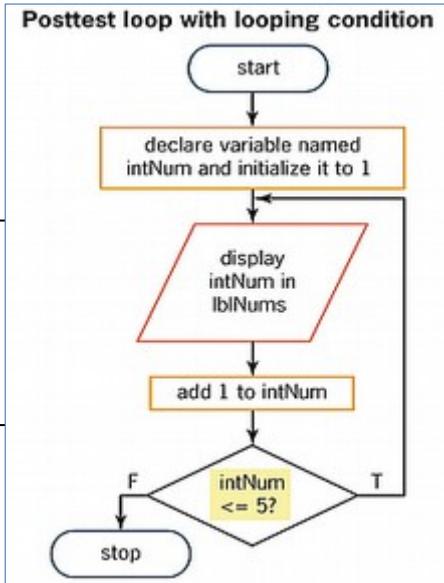
- can contain: **variables, named constants, literals, properties, methods, keywords, operators**
- must evaluate to a **Boolean value** (like the condition in an **If...Then...Else** statement)
- the condition is evaluated with each repetition of the loop and determines whether the computer processes the loop body
- use **Posttest Loop** rather than **Pretest Loop**

e.g. **Figure 5-2 & Figure 5-3**: pseudocode & flowchart & code: display the numbers 1 through 5 in a label control:

#### Posttest loop with looping condition:

```
1. declare variable named intNum and initialize it to 1
2. repeat
    display intNum in lblNums
    add 1 to intNum
    end repeat while intNum <= 5      <- bottom of the loop
Dim intNum As Integer = 1
Do
    lblNums.Text = lblNums.Text & intNum.ToString & " "
    intNum += 1
Loop While intNum <= 5
```

& = concatenation operator ->  
specifies when to continue ->



### Posttest loop with loop exit condition:

1. declare variable named intNum and initialize it to 1
  2. repeat
    - display intNum in lblNums
    - add 1 to intNum
    - end repeat   **until intNum > 5**
- < bottom of the loop

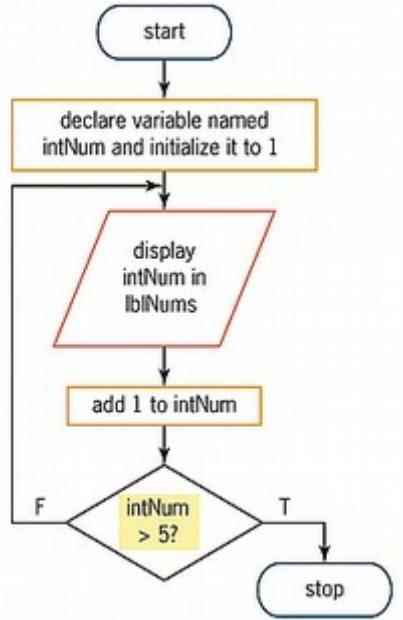
```
Dim intNum As Integer = 1
Do
    lblNums.Text = lblNums.Text & intNum.ToString & " "
    intNum += 1
Loop Until intNum > 5
```

& = concatenation operator ->  
specifies when to stop ->



<- result of using either of:   **- Posttest loop with looping condition or - Posttest loop with loop exit condition**  
 - the loop will stop when the value in **intNum** is **6**  
 + 5x Space characters

### Posttest loop with loop exit condition



e.g.: To code and then test the **Do...Loop Posttest loop** application:

1. open the ...VB2017\Chap05\\_Exercise\04.Do Loop Posttest Solution\Do Loop Posttest Solution.sln
2. in a Code editor window locate the: **btnDisplay\_Click** procedure
  - enter the Do...Loop Posttest statement by your choice (there are 5x Space characters between the quotation marks)
3. save & test
  - it will display the numbers from 1 to 5 in the label control
4. in a **Dim** statement change the **1** to **6**
  - save & test
- the number **6** appears in the label control because:
  - the Posttest Loop's condition is not evaluated until after the instructions in the loop are processed the first time
5. change back **6** to **1**

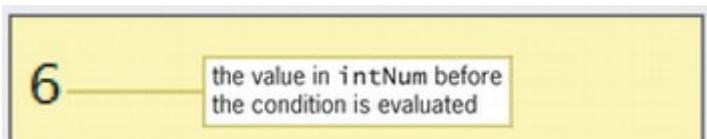


Figure 5-7

### Mini-Quiz 5-2

1. Using the **While** keyword, write a **Loop** clause that processes the loop body as long as the value in the **intAge** variable is greater than **21**.
2. Rewrite the **Loop** clause from Question 1 using the **Until** keyword.
3. Using the **While** keyword, write a **Loop** clause that processes the loop body as long as the value in the **dblScore** variable is greater than **0** and, at the same time, less than or equal to **100**.
4. Using the **Until** keyword, write a **Loop** clause that stops the loop when the value in the **strContinue** variable contains the letter **N** (in either uppercase or lowercase).

1... Loop While intAge > 21

2... Loop Until intAge <= 21

3... Loop While dblScore > 0 AndAlso dblScore <= 100

4... Loop Until strContinue.ToUpper = "N"

**F2**

```
Do While + condition or
Do Until + condition
    + loop body instructions
Loop
```

pretest loop with looping condition  
 pretest loop with loop exit condition

```
Do While intNum <= 5
Do Until intNum > 5
```

**F5**

```
Do
    + loop body instructions
Loop While + condition or
Loop Until + condition
```

posttest loop with looping condition  
 posttest loop with loop exit condition

```
Loop While intNum <= 5
Loop Until intNum > 5
```

### You Do It 2:

1. create an app named You Do It 2
2. add a **lbl** and **btn**
3. the **btn** should use a **posttest loop** and the **concatenation operator** to display the following numbers in the **lbl**: **1, 3, 5, 7.**

```
Dim intNum As Integer = 1
Do
    lblShow.Text = lblShow.Text & intNum.ToString & " "
    intNum += 2
Loop While intNum <= 7
```

### CH5\_F5.1 - Posttest loop: Do ... Loop Until e.g.: 30.Fibonacci Solution\_EXERCISE 15

- Chap05\\_Exercise\30.Fibonacci Solution\_EXERCISE 15

```
Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
' Displays the first 10 Fibonacci numbers: 1, 1, 2, 3, 5, 8, 13, 21, 34, 55:
    Dim A As Integer = 0
    Dim B As Integer = 1
    Dim C As Integer

    Do
        C = A + B
        A = B
        B = C
        lblNumbers.Text += A.ToString & ", "
    Loop Until A = 55
End Sub
```

## CH5\_F6 - Counters and Accumulators basic info & e.g.

- some procedures require you to calculate a **subtotal**, a **total**, or an **average**
- you make these calculations using a **Counter**, an **Accumulator**, or both
- a **Counter** is:
  - a numeric variable used for counting something, such as the number of employees paid in a week
  - the **intNum** variable in e.g. 1 in **Figure 5-8** is a **counter** because it keeps track of the number of times the loop instructions are repeated

e.g. 1:

**Figure 5-8**

**Counter**

```
Dim intNum As Integer = 1
Do While intNum <= 5
    lblNums.Text = lblNums.Text & intNum.ToString & "      "
    intNum += 1
Loop
```

<- initializes the **Counter** before the loop

<- updates the **Counter** within the loop

- an **Accumulator** is:
  - a numeric variable used for accumulating (**adding together**) something, such as the total dollar amount of a week's payroll (výplatnice)
  - **info after EXERCISES: don't use a Dim, but Static -> otherwise it would erase its value each time !!!**
  - the **dblTotal** variable in e.g. 2 in **Figure 5-8** is an **accumulator** because it adds together the scores entered by the user
  - the **btnAdd\_Click** procedure is from CH3 - Total Scores apps (static/class level) <- class level variable = **Static**

e.g. 2:

**Figure 5-8**

**Accumulator**

```
Private Sub btnAdd_Click(sender As Object, e As EventArgs) Handles btnAdd.Click
    'Accumulates and displays the scores.
    Dim dblScore As Double
    ' Procedure-level variable for storing a score.
    Static dblTotal As Double
    ' Static variable for accumulating scores:
    Double.TryParse(txtScore.Text, dblScore)
    dblTotal = dblTotal + dblScore
    lblTotal.Text = dblTotal.ToString
    ' Accumulate the scores and display the results.
End Sub
```

<- initializes the **Accumulator**

<- updates the **Accumulator**

- **Counters** and **Accumulators** must be first initialized and then updated :

- 1). initialization:
  - **Counters** are initialized to either 0 or 1, depending on the value required by the procedure's code :
    - > e.g.1, **Dim** statement initializes the **intNum** counter variable to **1**
  - **Accumulators** are, in most cases, initialized to 0 :
    - > e.g.2, the **Static** statement initializes the **dblTotal** accumulator variable to **0**
- 2). update:
  - by either adding (+) a number to their value = **incrementing**
  - or subtracting (-) a number from their value = **decrementing**
  - the number can be either positive or negative, integer or decimal
  - a **Counter** is always updated by a constant amount - typically the number **1**
    - > e.g.1, the **intNum += 1** statement updates the **intNum** counter variable by **1**
    - unlike the **intNum** variable's initialization statement, which appears above the loop, its update statement is entered in the body of the loop - this is because the variable needs to be updated each time the loop instructions are processed
  - an **Accumulator** on the other hand, is usually updated by an amount that varies, and it is usually incremented rather than decremented
    - > e.g.2, the **dblTotal = dblTotal + dblScore** statement updates the **dblTotal** accumulator variable by the current value stored in the **dblScore** variable

- Game programs make extensive (rozsáhlý) use of Counters and Accumulators

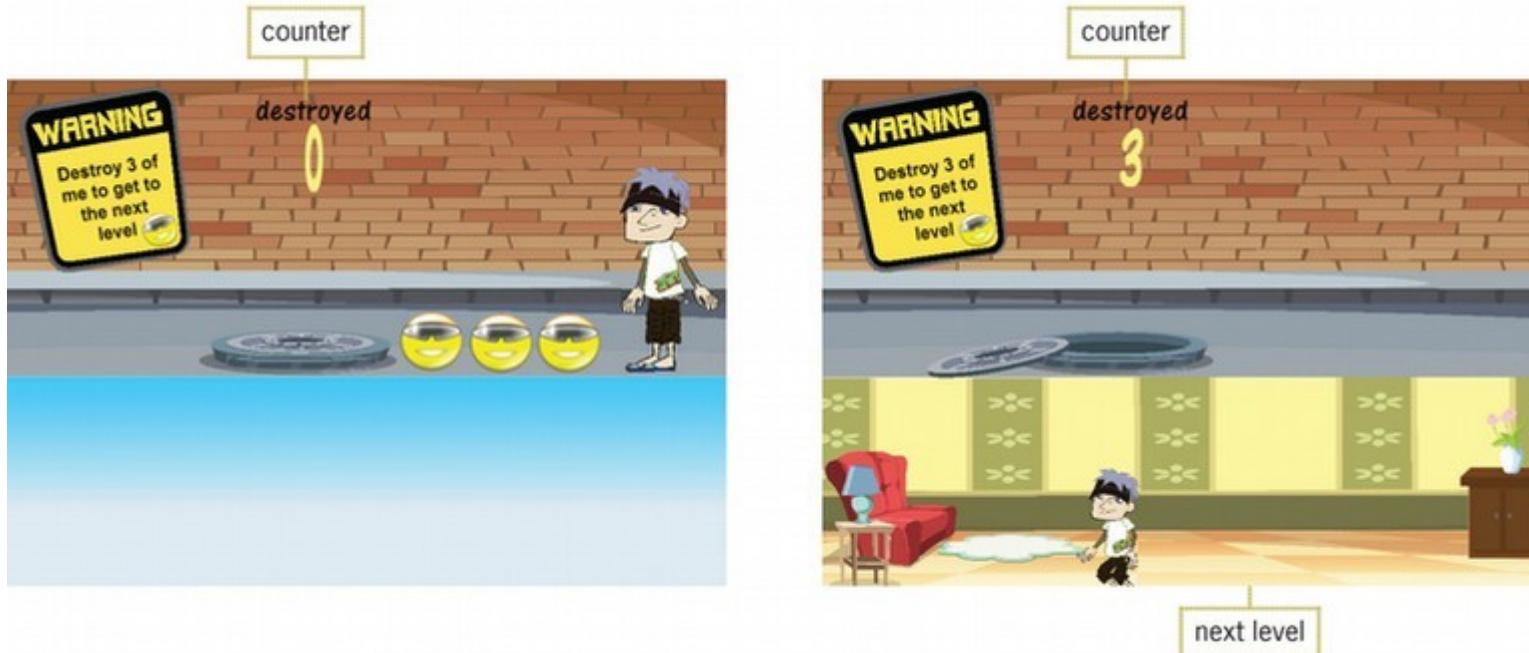
- **Figure 5-9** - the partial game program, for example, uses a counter to keep track of the number of smiley faces that Eddie destroys

- after he destroys 3 smiley faces and then jumps through the manhole, he advances to the next level in the game, as shown in the figure:

1. initialize destroyed counter to **0**
2. repeat **while** destroyed counter is **less than 3**
  - jump on smiley face to destroy it
  - add **1** to destroyed counter
  - end repeat while**
3. jump into manhole to advance to the next level

```
Static intDestroyed As Integer = 0  
Do While intDestroyed < 3  
  
    intDestroyed +=1  
Loop
```

```
Static intDestroyed As Integer = 0  
Do  
  
    intDestroyed +=1  
Loop While intDestroyed < 3
```



### Mini-Quiz 5-3

1. Write an assignment statement that increments the **intRegistered** counter variable by **2**. Use an arithmetic assignment operator.
2. Write an assignment statement that decrements the **intNum** counter variable by **5**. Use an arithmetic assignment operator.
3. Write an assignment statement that increments the **decTotal** variable by the value in the **decRegion** variable. Use an arithmetic assignment operator.
4. Which of the following is true?
  - a. Counters and Accumulators must be initialized.
  - b. Counters and Accumulators must be updated.
  - c. Counters are usually updated by a constant amount.
  - d. All of the above.

1...intRegistered +=2  
2...intNum -= 5 Or intNum += -5  
3...decTotal = decTotal + decRegion Or decTotal += decRegion  
4...d - all of the above

## CH5\_F7 - Pretest loop: For...As dataType/... = ...To...Step ... Next statement - specific counter-controlled loop

- the **For...Next** statement can be used to code only a specific type of **pretest loop**, called a **counter-controlled loop**

- unlike the **Do...Loop** statement, which can be used to code both:

- **pretest loop** - **Do While/Do Until...Loop**
- **posttest loop** - **Do...Loop While/Loop Until**

-> **specific counter-controlled pretest loop**: **For...Next** = loop whose processing is controlled by a **counter**

- used when you want the computer to process the loop instructions a **precise number of times**

- although you can also use the **Do...Loop** statement, the **For...Next** statement provides a more compact and convenient way of writing that type of loop

-> **counter** = name of a numeric variable that the computer will use to keep track of/count the number of times the loop body instructions are processed

syntax:

```
For newCounter As dataType/existingVariable = startValue To endValue Step stepValue  
    ...loopBody instructions  
Next newCounterName/existingVariable
```

**Step** = optional

- if omitted, default value **1** is used

- technically, you do not need to specify the name of the counter variable in the **Next** clause, but it is highly recommended for easier **understandability**

counter's variable declaration:

- when deciding where to declare the counter's variable, keep in mind that if the variable is needed only by the **For...Next** loop,  
then is better coding practice to declare it there ->

-> as mentioned in **CH3**, fewer unintentional errors occur in applications when the variables are declared using the **minimum scope needed**

scope from smallest:

|                    | variable    | constant      |
|--------------------|-------------|---------------|
| 1. Block-level     | For         | -             |
| 2. Procedure-level | Dim, Static | Const         |
| 3. Class-level     | Private     | Private Const |

a). in the **For** clause: **For newCounter As dataType = ...** e.g.1 & e.g.2

- variable has **block scope** - the smallest scope = can be used only within the **For...Next** loop

b). in the **Dim** statement: **For existingVariable = ...** e.g.3

- variable has **procedure scope** = can be used within the entire procedure

- should be declared **only** when its value is required by statements outside the **For...Next** loop in the procedure

startValue, endValue:

- numeric - integer/decimal and positive/negative - items in the **For** clause control the number of times the loop body is processed

- the **startValue** and **endValue** tell the computer where to begin and end counting

stepValue:

- tells the computer how much to count by - i.e. how much to add to the counter variable each time the loop body is processed

- if omitted, the **Step** = **1** will be used

- if the **stepValue** is:  
- a). a **positive** number, the loop **ends** when the counter's value is **greater** than the **endValue**  
- b). a **negative** number, the loop **ends** when the counter's value is **less** than the **endValue**

| if the <b>stepValue</b> is a: | the <b>loopBody</b> is processed when the: | the loop ends when the:              |
|-------------------------------|--------------------------------------------|--------------------------------------|
| a) (+) <b>positive number</b> | counter's value <b>&lt;= endValue</b>      | counter's value <b>&gt; endValue</b> |
| b) (-) <b>negative number</b> | counter's value <b>&gt;= endValue</b>      | counter's value <b>&lt; endValue</b> |

processing tasks:

1. - if the counter is declared in the **For** clause, the variable is created and then initialized to the startValue
  - if the counter is declared in the **Dim** statement, it is just initialized to the startValue
  - the initialization task is done only once, at the beginning of the loop
2. - the counter's value is compared with the endValue to determine whether the loop should end:
  - if the stepValue is:
    - a) a **positive number**, the comparison is: **counter's value > endValue**
    - b) a **negative number**, the comparison is: **counter's value < endValue**
  - notice that the computer evaluated the loop condition before processing the instructions within the loop = **Pretest Loop**
3. - if the comparison from task 2. evaluates to:
  - 3a) **FALSE** : the loop body instructions are processed and then Task 4 is performed
  - 3b) **TRUE** : the loop ends and processing continues with the statement following the **Next** clause
4. - is performed only when the comparison from **Task 2** evaluates to **FALSE**
  - in this task, the stepValue is added to the counter's value, and then **Tasks 2, 3 and 4** are repeated until the loop condition evaluates to **TRUE**

e.g.1 **For intNum As Integer = 1 To 5**

```
    lblNums.Text = lblNums.Text & intNum.ToString & "      "  
    Next intNum
```

<- 5x Space character

output: 1 2 3 4 5

- the startValue is **1**, the endValue is **5**, and the stepValue (which is omitted) is **1**
- those values tell the computer to **start** counting at **1** and, **counting by 1, stop** at **5**
- the computer will process the instructions in loop body five times
- when the loop ends, the value in the **intNum** variable will be **6** because that is the first integer that is greater than the loop's endValue of **5**

e.g.2 **For intNum As Integer = 5 To 1 Step -1**

```
    lblNums.Text = lblNums.Text & intNum.ToString & "      "  
    Next intNum
```

<- 5x Space character

output: 5 4 3 2 1

e.g.3 **Dim dblRate As Double**

```
For dblRate = 0.05 To 0.1 Step 0.01  
    lblRates.Text = lblRates.Text & dblRate.ToString("P0") & ControlChars.NewLine  
    Next dblRate
```

output: 5%

6%

7%

8%

9%

10%

e.g.4 to code and then test the **For...Next** application: **06.For Next Solution**

1. open the ...VB2017\Chap05\Exercise06.For Next Solution\For Next Solution.sln and in a Code editor window locate the: **btnDisplay\_Click** procedure

-> click the blank line above the **End Sub** clause and then enter the **For...Next** statement shown in e.g.1

```
For intNum As Integer = 1 To 5  
    lblNums.Text = lblNums.Text & intNum.ToString & "      "  
    Next intNum
```

2. save and test it - the **Display** button should show the numbers from **1** to **5** as in the **Figure 5-11**:

3. now change the startValue in the **For** clause from **1** to **6**, and test it

<- notice that no numbers appear in the **lbl** control

- because the startValue is greater than the endValue, the pretest loop's condition evaluates to **TRUE**  
and the computer does not process the instructions in the loop body

4. change back **6** to **1**

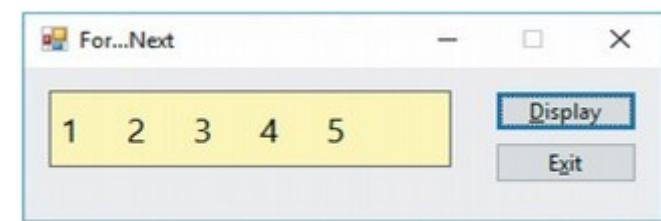


Figure 5-11 Output displayed by the For...Next statement

## CH5\_F8 - Pretest loop: create counter-controlled loop - compare Do...Loop statement vs For...Next statement

- you can code a **counter-controlled loop** by using either the **Do...Loop** statement or **For...Next** statement

a). **Do...Loop** statement:

```
Dim intNum As Integer = 1
Do While intNum <= 5
    lblNums.Text = lblNums.Text & intNum.ToString & "    "
    intNum += 1
Loop
```

### CH5\_F2 - Pretest loop: Do While/Do Until...Loop statement

<- declares and initializes the **counter** variable  
<- compares the **counter** variable  
<- updates the **counter** variable

- processing task: **1. + 2.**  
- processing task: **3.**

- processing task: **4.**

b). **For...Next** statement:

```
For intNum As Integer = 1 To 5
    lblNums.Text = lblNums.Text & intNum.ToString & "    "
Next intNum
```

### CH5\_F7 - Pretest loop: For...As dataType/... = ...To...Step ... Next statement - specific counter-controlled loop

<- declares, initializes, compares, updates the **counter** variable

- processing task: **1.- 4.**

- conclusion: as indicated, the **For...Next** statement is more convenient way of coding this type of loop, because:

- a). **Do...Loop** statement: - you must include statements to declare, initialize, and update the counter variable and  
- you also must include the appropriate comparison in the **Do** clause
- b). **For...Next** statement: - the **For** clause handles declaration, initialization, comparison, and update tasks

## CH5\_F9 - Pretest loop: For...Next statement - specific counter-controlled loop - Flowcharting

### Example 1:

- the counter variable's initialization task is entered in a rectangle,
- its comparison task is in a diamond,
- its update task is in another rectangle

### Example 2:

- a hexagon is used to represent the tasks performed by the **For** clause
- the counter variable's **name** and the **stepValue** are placed at the top and bottom of the hexagon
- the **startValue** and **endValue** are placed on the left and right side
- notice that a greater than sign (**>**) precedes the **endValue**
- the **>** sign indicates that the loop will end when the counter variable's value is greater than 5

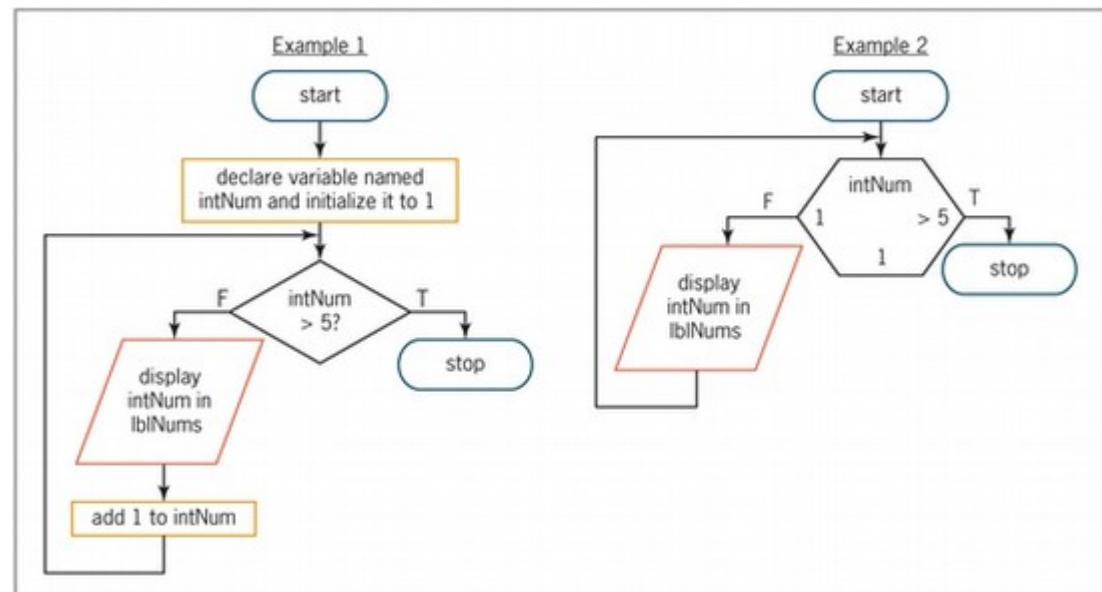


Figure 5-13 Flowcharts for a For...Next loop

#### Mini-Quiz 5-4

1. Write a **For** clause that repeats the loop instructions **10** times. Use **intX** as the counter variable's name, and declare the variable in the **For** clause
2. If the loop's **For** clause is: **For intX = 2 To 8 Step 2**, how many times will the loop instructions be processed?
3. What value will be in the **intX** variable when the loop in **Question 2** ends?

3...10 because its the "over" value  
2...4 times - 2 4 6 8  
1...For intX As Integer = 1 To 10

#### CH5\_F9.1 - Pretest loop: For...Next statement - create Counter and Accumulator - 07.You Do It 3 Solution

1. create an app named You Do It 3
2. add **2x lbl** and **btn**
3. the **btn**'s Click event procedure should display:
  - in one **lbl**: the number of integers from **14** to **23**
  - in other **lbl**: the **sum** of those integers
4. code the procedure using the **For...Next** statement
5. the procedure should display:
  - in **lbl1** - **10x** numbers (Counter)
  - in **lbl2** - **185** (Accumulator)

```
Static intTotal As Integer
For int1 As Integer = 14 To 23
    lbl1.Text += int1.ToString & " "
    intTotal += int1
    lbl2.Text = intTotal.ToString
Next int1
```

<- initializes the **Accumulator**  
<- declares, initializes, compares, updates the **Counter** variable  
<- display the **Counter**  
<- updates the **Accumulator**  
<- display the **Accumulator**

## APPLY THE CONCEPTS LESSON

### CH5\_A1 - Pretest loop: Do...Loop statement - use a Loop, Counter, and Accumulator - e.g. 08.Projected Sales Solution

- Figure 5-14: e.g.: the GUI and pseudocode for the Project Sales application (Do While...Loop)

- using the current sales amount entered by the user and a 3% annual growth rate, the app will calculate the number of years required for a company's projected sales to reach at least \$150,000
- it will also calculate the projected sales amount at that time
- both calculated amounts will be displayed in the **lblProjSales** control

#### **btnCalc\_Click procedure's pseudocode:**

1. declare dblGROWTH\_RATE constant and initialize it to 0.03
2. declare variables (dblSales, accumulator, dblIncrease, intYears counter)
3. convert txtCurrentSales.Text to a number and store in dblSales
4. repeat while dblSales < 150000
  - dblIncrease = dblSales \* dblGROWTH\_RATE
  - add dblIncrease to dblSales
  - add 1 to intYears
- end repeat while
5. display intYears and dblSales in lblProjSales

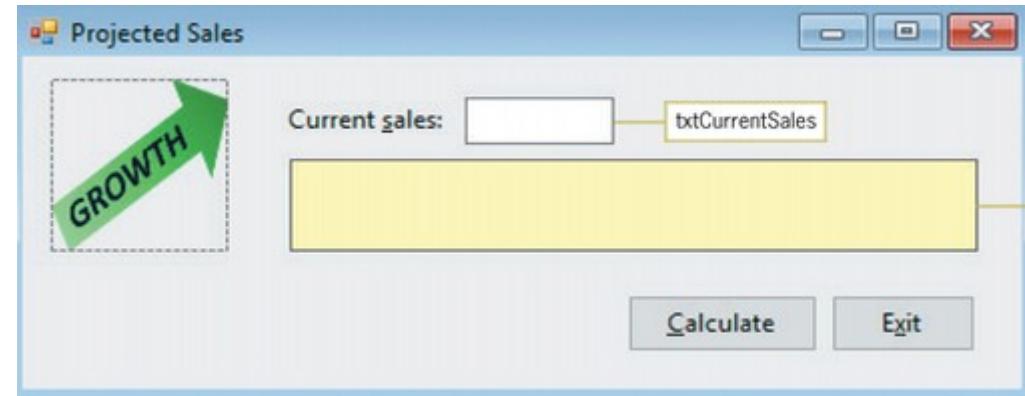


Figure 5-14

e.g.: To code and then test the Projected Sales application (Do While...Loop):

1. open the ...VB2017\Chap05\Exercise\08.Projected Sales Solution\Projected Sales Solution.sln
2. in a Code Editor window locate the **btnCalc\_Click** procedure:
  - the **first 3 steps** in the procedure's pseudocode have already been coded for you
  - the **Dim** statements that declare the **accumulator** and **counter** variables initialize both memory locations to **0**
3. click the blank line above the **End Sub** clause
  - **step 4** in the pseudocode is a loop that repeats its instructions as long as (or while) the value in the **dblSales** variable is less than **150000**
  - the loop will **stop** when the **dblSales** variable's value is **greater than or equal** to **150000**
  - **type:** **Do While dblSales < 150000** and press **Enter**
  - the **first** instruction in the loop calculates the sales increase for the current year
  - the **second** instruction adds the sales increase to the **dblSales** variable
  - the **third** instruction increments the **intYears** variable by **1**
  - the **second and third** instructions **update** the **accumulator** and **counter** variables
4. enter the loop body instructions:

```
21    Do While dblSales < 150000
22        dblIncrease = dblSales * dblGROWTH_RATE
23        dblSales += dblIncrease
24        intYears += 1
25    Loop
26    End Sub
```

5. after the loop has completed its processing, the procedure will display both the number of years and the projected sales in the **lblProjSales** control  
 - insert a blank line between the **Loop** and **End Sub** clause, and then enter the assignment statement:

```

25      Loop
26      lblProjSales.Text = "Projected sales " &
27          intYears.ToString & " years from now: " &
28          dblSales.ToString("C0")
29
30  End Sub

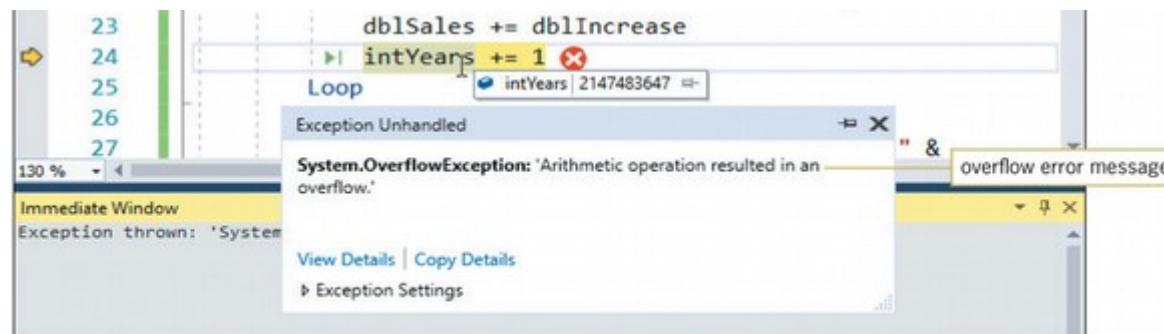
```

6. start the app and: **(Figure 5-17)**

- type: **75000** in the Current sales box and then click the **Calculate** button
- the result in the label should be: **Projected sales 24 years from now: \$152,460**
- id est:
  - It will take 24 years for the company's projected sales to exceed \$150,000
  - and at that time, the projected sales will be \$152,460

7. delete the contents of the Current sales box and then click the **Calculate** button

- after a short period of time, a run time error occurs and the error message box shown in **Figure 5-18** appears on the screen
- place your mouse pointer on **intYears**, as shown in the figure:

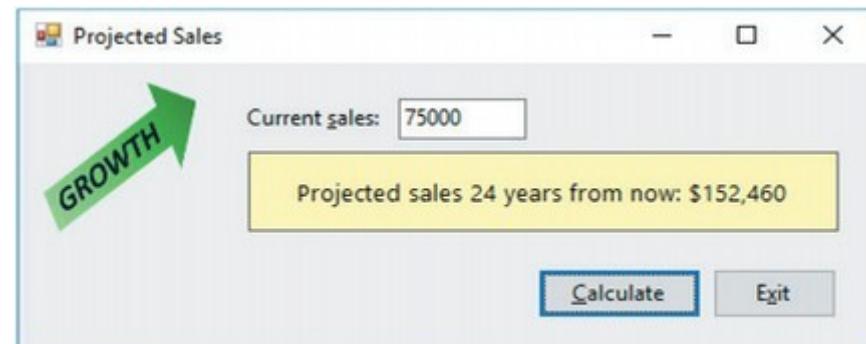


**Figure 5-18** Screen showing the error message box

- the error message informs you that an arithmetic operation - in this case, **adding 1 to the intYears variable** - resulted in an overflow
- an **overflow error**
  - occurs when the value assigned to a memory location is too large for the location's data type
  - similar to trying to fill a 0.2L glass with 1L of water
  - in this case, the **intYears** variable already contains the largest value that can be stored in an Integer variable (2,147,483,647 -CH3)
    - > therefore, when the **intYears += 1** statement attempts to increase the variable's value by 1, an overflow error occurs

#### - but why does the **intYears** variable contain **2,147,483,647**?

- because you **didn't** provide an **initial** value for the current sales amount, the **dblSales** variable contains the value assigned to it by the **TryParse** method: **0**
- the loop's condition (**dblSales < 150000**) evaluates to **TRUE** the first time it is processed - therefore, the computer processes the loop's instructions
- **none** of the instructions in the loop body **change** the value in the **dblSales** variable to anything other than **0** ->
- > as a result, the loop's condition **always** evaluates to **TRUE** and the computer continues to process the loop's instructions, creating an **infinite loop**
- the **intYears += 1** statement continues to increase the value in the **intYears** variable by **1** until the value reaches the largest amount an **Integer** variable can store
- at that point, the **intYears += 1** statement causes an **overflow error**, which is the only reason the loop **stopped**



**Figure 5-17** Sample run of the Projected Sales application

- in the next set of steps, you will modify the loop's condition so that it processes the loop body only when the current sales are **greater** than **0** and, at the same time, **less** than **150000**:

e.g. to **modify** and then test the **Projected Sales** application:

1. click the **Stop Debugging** button (**the red square**)
2. change the condition in the **Do While** clause to a compound condition, as indicated below on line **21**

```

15 Const dblGROWTH_RATE As Double = 0.03
16 Dim dblSales As Double 'Used as an accumulator.
17 Dim dblIncrease As Double
18 Dim intYears As Integer 'Used as a counter.
19
20 Double.TryParse(txtCurrentSales.Text, dblSales)
21 Do While dblSales > 0 AndAlso dblSales < 150000
    dblIncrease = dblSales * dblGROWTH_RATE
    dblSales += dblIncrease
    intYears += 1
25 Loop
26 lblProjSales.Text = "Projected sales " &
    intYears.ToString & " years from now: " &
28     dblSales.ToString("C0")
29
30 End Sub

```

<- compound condition

3. save and test:

<- notice that **no overflow** error occurs, when click the **Calculate** button

#### CH5\_A1.1 - Pretest loop: **For...Next** statement - use a **Loop**, **Counter**, and **Accumulator** - e.g. 09.Projected Sales Solution-ForNext

- **Figure 5-20:** e.g.: the GUI and pseudocode for the **Project Sales** application (**For...Next**)

- in this version of the Projected Sales application, the **btnCalc\_Click** procedure will display the projected sales amount for each of 4 years, beginning with 2019

#### **btnCalc\_Click** procedure's pseudocode:

1. declare dblGROWTH\_RATE constant and initialize it to 0.03
2. declare variables (dblSales, accumulator, dblIncrease)
3. convert txtCurrentSales.Text to a number and store in dblSales
4. repeat for intYears from 2019 to 2022 in increments of 1
  - dblIncrease = dblSales \* dblGROWTH\_RATE
  - add dblIncrease to dblSales
  - in lblProjSales display: intYears, dblSales, and a blank line
- end repeat while

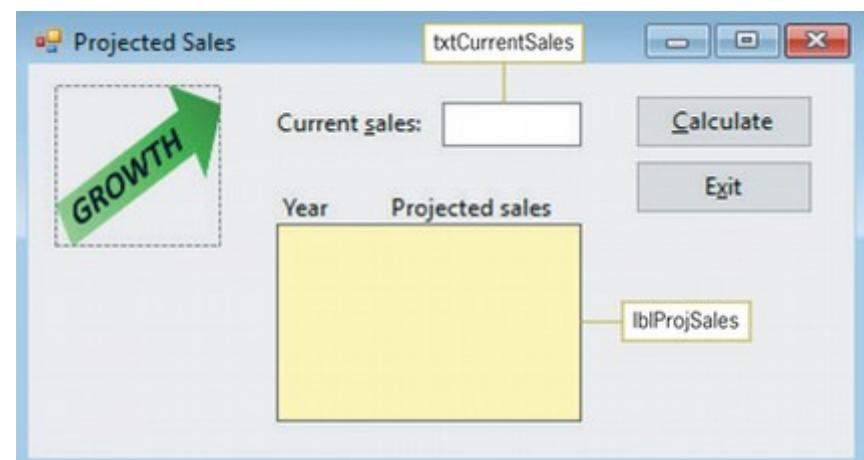


Figure 5-20

e.g.: To code and then test the **Projected Sales** application (**For...Next**):

1. open the ...VB2017\Chap05\Exercise09.Projected Sales Solution-ForNext\Projected Sales Solution.sln
2. in a Code Editor window locate the **btnCalc\_Click** procedure:
  - the **first 3 steps** in the procedure's pseudocode have already been coded for you
  - the first **Dim** statement initializes the **accumulator** variable to **0**
3. click the blank line above the **End Sub** clause and then enter the **For...Next** statement shown in **Figure 5-21**:

```
15 Const dblGROWTH_RATE As Double = 0.03
16 Dim dblSales As Double 'Used as an accumulator.
17 Dim dblIncrease As Double
18
19 Double.TryParse(txtCurrentSales.Text, dblSales)
20 For intYears As Integer = 2019 To 2022
21     dblIncrease = dblSales * dblGROWTH_RATE
22     dblSales += dblIncrease
23     lblProjSales.Text = lblProjSales.Text &
24         intYears.ToString & "      " &
25         dblSales.ToString("C0") & ControlChars.NewLine
26 Next intYears
```

<- 10 space characters

4. save and test: type **86500**, the result should be:
- |      |          |
|------|----------|
| 2019 | \$89,095 |
| 2020 | \$91,768 |
| 2021 | \$94,521 |
| 2022 | \$97,357 |

#### CH5\_A2 - ListBox - add a (lst) to a Form: basic info & basic properties

- you can use the **ListBox** tool to add a list box - **(lst)** to a GUI
- it displays a list of items from which the user can select: **0** items, **1** item, or **multiple** items ->
  - > the **number** of items that can be selected at any one time is determined by the **SelectionMode** property, which is typically left as its default value : **1**
- the most commonly used properties of a **ListBox**:

**Font** = specify the font to use for text

**(Name)** **Ist**

**SelectedIndex** = **get** or **set** the **index** of the selected item

**SelectedItem** = **get** or **set** the **value** of the selected item

**SelectionMode** = indicate whether the user can select **0** items, **1** item, or **multiple** items at a time; default = **One**

- can be: **None / One / MultiSimple / MultiExtended**

- **MultiSimple** = multiple selection only with a **Ctrl** or **Shift** key

- **MultiExtended** = multiple selection without a need of keys

**Sorted** = specify whether the items in the list should appear in the order they are entered **or** in dictionary leftmost like sorted order

- can be: **True / False**

- VB sorts the items in **dictionary leftmost order**: numbers -> lowercase -> **UPPERCASE**

**CH5\_A2.3 - ListBox - the SelectedItem and SelectedIndex properties ...**

**CH5\_A2.3 - ListBox - the SelectedItem and SelectedIndex properties ...**

**CH5\_A2.3.1 - ListBox - the SelectedItems and SelectedIndices ...**

**CH5\_A2.3.1 - ListBox - the SelectedItems and SelectedIndices ...**

**CH5\_A2.2 - ListBox - the property Sorted - e.g. 10.ListBox Solution**

- although you can make a **Ist** any **size** you want, you should follow the Windows standard, which is to display **at least 3** items but **no more than 8** items at a time
- if you have more items than can fit into the **Ist**, the control automatically displays a **scroll bar** for viewing the complete list of items
- you should use a **lbl** control to provide **keyboard access (Alt key)** to the **Ist** with a **TabIndex** property set to 1 number less than the **Ist's TabIndex**

### CH5\_A2.1 - ListBox - add items using the String Collection editor in Properties - e.g. 10.ListBox Solution

- the items in a **Ist** belong to a collection called the **Items collection**

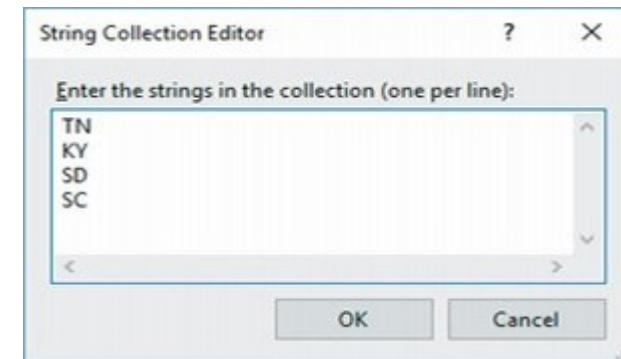
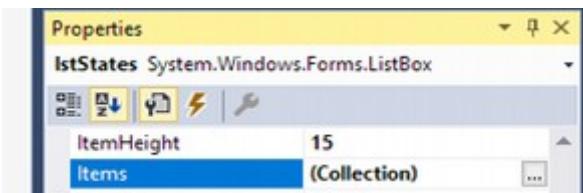
- > **Items collection:**
  - is a group of individual objects treated as one unit
  - each item in the **Item collection** is identified by a unique number -> an **index**
  - the **1st** item in the collection (which is also the first item in the list box) has an **index = 0**
  - the **2nd** item has an **index =1**, and so on

- the **String Collection editor** provides an easy way to add items to the **Items collection**

e.g. to use the **String Collector editor**: open the ...VB2017\Chap05\Exercise\10.ListBox Solution\ListBox Solution.sln

-> click the **IstStates** ListBox and then in the **Properties** window click **Items**

<- notice that **(Collection)** appears in the property's setting box



-> click the ...**(ellipsis)** button and then enter the **4 state IDs** shown:

-> click **OK** button, save and test

<- notice the 4 state IDs appear in the same order as they do in the **String Collection editor**:

**TN, KY, SD, SC**

### CH5\_A2.2 - ListBox - the property **Sorted** - e.g. 10.ListBox Solution

- typically, **Ist** items are either arranged:

- by **use**, with the most used entries appearing first in the list
- or in **ascending** order

- VB sorts the items in **dictionary leftmost order**: **numbers -> lowercase -> UPPERCASE**

e.g. to sort the items in the **IstStates** control: ...VB2017\Chap05\Exercise\10.ListBox Solution\ListBox Solution.sln

-> set the **IstState** control's property **Sorted = TRUE**

-> save and test

<- notice: if the property **Sorted = True**, then the state ID will appear in following order: **KY; SC, SD, TN**

<- it also means that the numbers **1, 2, 3, 10** would appear in the following order: **1, 10, 2, 3**

### CH5\_A2.3 - ListBox - the **SelectedItem** and **SelectedIndex** properties - when **SelectionMode = One** - e.g. 10.ListBox Solution

- as mentioned earlier, a **Ist**'s **SelectionMode** property determines the number of items the user can select at any one time

- if a **Ist** allows the user to make only one selection, it is customary in Windows applications to have one of the **Ist** items already selected when the GUI appears

- the **selected item**, called the **default list box item**, should be either the item selected most frequently or the first item in the list

- you can use either:

- the **SelectedItem** property or
- the **SelectedIndex** property

e.g. **IstStates.SelectedItem = "SD"**

<- selects the **SD** item in the **IstStates** control

e.g. **IstStates.SelectedIndex = 0**

<- selects the first item in the **IstStates** control

- in most cases, you enter the appropriate code in the form's **Load event** procedure

- a form's **Load event** occurs when the application is started and the form is displayed the first time

e.g. to select the first item in the **Ist**: ...VB2017\Chap05\Exercise10.ListBox Solution\ListBox Solution.sln

1. open the Code Editor window

- in the Object list box (middle drop-down menu) click on **frmMain Events** and in the Event list box (right drop-down menu) choose **Load**
- in a line 21 enter the comment: 'Select the first item in the **IstStates** control
- in a line 22 enter the code: **lstStates.SelectedIndex = 0**

The screenshot shows the Visual Studio Code Editor with the Main Form.vb [Design] tab selected. The code editor displays the following VB.NET code:

```
Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
    'Select the first item in the lstStates control
    lstStates.SelectedIndex = 0
End Sub
End Class
```

2. save and test

<- notice that the first item in the list is selected

- you can also use either the **SelectedItem** property or the **SelectedIndex** property to determine which item (if any) is selected in a **Ist**

- if an item is selected:
  - the **SelectedItem** = selected item's **value**
  - the **SelectedIndex** = selected item's **index**
- when no item is selected:
  - the **SelectedItem** = **Nothing**
  - the **SelectedIndex** = **-1**

e.g. to display the selected item and its index: ...VB2017\Chap05\Exercise10.ListBox Solution\ListBox Solution.sln

1. locate the **btnSelected\_Click** procedure

- click the blank line above the **End Sub** clause and then enter the assignment statements:

```
14     Private Sub btnSelected_Click(sender.....)
15         'Display the selected item and selected index.
16
17         lblItem.Text = lstStates.SelectedItem.ToString
18         lblIndex.Text = lstStates.SelectedIndex.ToString
19
20     End Sub
```

2. save and test:

- click the **Selected** button:
  - **KY** and **0** appear in the Item and Index boxes, respectively
- click **SD** in the list box
  - do not be concerned that **KY** and **0** still appear in the GUI - you will fix this problem in the next section
- click the **Selected** button:
  - **SD** and **2** appear in the Item and Index boxes, respectively

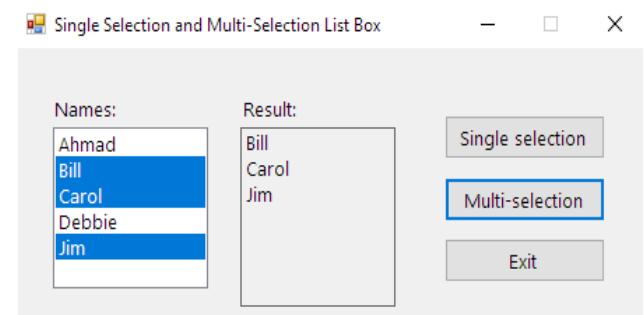
## CH5\_A2.3.1 - ListBox - the SelectedItems and SelectedIndices properties - when SelectionMode = MultiSimple / MultiExtended - 31.Multi Solution\_EXERCISE 16

- info after EXERCISES: ...Chap05\Exercise31.Multi Solution\_EXERCISE 16

```
Private Sub btnMulti_Click(sender As Object, e As EventArgs) Handles btnMulti.Click
    lblResult.Text = String.Empty

    'lstNames.Items.Count.ToString      <- number of ALL items_COUNT
    'lstNames.SelectedItems.Count.ToString <- number of SELECTED items_COUNT
    'lstNames.SelectedItems             <- the COLLECTION of selected items_ITEMS
    'lstNames.SelectedIndices          <- the INDICES of items_INDICES (PL from INDEX)

    For a As Integer = 0 To lstNames.SelectedItems.Count - 1
        lblResult.Text += lstNames.SelectedItems.Item(a).ToString & ControlChars.NewLine
    Next a
End Sub
```



## CH5\_A2.4 - ListBox - the SelectedValueChanged and SelectedIndexChanged events - e.g. 10.ListBox Solution

- the event occurs each time either the user or a statement selects an item in a **Ist**
- in the next set of steps, you will use the **IstStates\_SelectedIndexChanged** procedure to clear the contents of the Item and Index boxes

e.g. to code the **SelectedIndexChanged** procedure: ...VB2017\Chap05\Exercise10.ListBox Solution\ListBox Solution.sln

1. open the code template for the **IstStates\_SelectedIndexChanged** procedure and then enter the comment and 2 assignment statements:

```
29     Private Sub IstStates_SelectedIndexChanged(sender.....
30         'Clear the contents of the Item and Index boxes.
31
32         lblItem.Text = String.Empty
33         lblIndex.Text = String.Empty
34
35     End Sub
```

2. save and test:

- click the **Selected** button - KY and 0 appear in the Item and Index boxes, respectively
- click **SD** in the **Ist**:
  - the **IstStates\_SelectedIndexChanged** procedure clears the contents of the Item and Index boxes, respectively

### Mini-Quiz 5-5

1. The items in a **Ist** (list box) belong to which collection?
2. When a list box's **Sorted** property is set to **TRUE**, in what order will the following items appear: Banana, Apple, Watermelon?
3. When a list box's **Sorted** property is set to **TRUE**, in what order will the following items appear: 35, 40, 3, 4?
4. Which **2** properties can be used to select a default item in a **Ist** (list box)?

...the Items collection  
2...Apple, Banana, Watermelon  
3...3, 4, 35, 40 by my logic, but the true is: 3, 35, 4, 40 -> because the Leftmost rule!!!  
4...SelectedItem and SelectedIndex

### CH5\_A3.1 - ListBox - Items Collection: method Add to add items - e.g. 11.ListBox Items Solution

- to add items to **Ist**, you can use:
  - a). **String Collection Editor**
  - b). method **Add**

### CH5\_A2.1 - ListBox - add items using the String Collection editor in Properties ...

- e.g. rather than typing the numbers **1** through **25** in the **String Collection Editor**, you can use the **Add** method along with a **Loop** to add the numbers to the **Ist**
- in most cases, you enter the **Add** method in a form's **Load event procedure** because you typically want the **Ist** to display its values when the form **first** appears

syntax: `objectListBox.Items.Add(item)` <- mostly entered in a **Load event procedure**

e.g.1

```
1stAnimals.Items.Add("Dog")
1stAnimals.Items.Add("Cat")
1stAnimals.Items.Add("Horse")
```

<- to the **1stNames** control adds the items: **Dog, Cat, Horse**

e.g.2

```
For intCode As Integer = 100 To 105
    1stCodes.Items.Add(intCode)
Next intCode
```

<- to the **1stCodes** control adds the items: **100, 101, 102, 103, 104, 105**

e.g.3

```
For decRate As Decimal = 0.1D To 0.3D Step 0.1D
    1stRates.Items.Add(decRate.ToString("P0"))
Next decRate
```

<- to the **1stRates** control adds the items: **10%, 20%, 30%**

e.g.4 to use the **Add** method: open the ...VB2017\Chap05\Exercise\11.ListBox Items Solution\ListBox Items Solution.sln

-> in a Code Editor window locate the **frmMain\_Load** procedure

-> click the blank line above the **End Sub** clause and then enter the **For...Next** loop and assignment statement:

```
26     Private Sub frmMain_Load(sender.....)
27         'Add items to the list box and select the first item.
28
29         For intNum As Integer = 1 To 25
30             1stNums.Items.Add(intNum)
31         Next intNum
32         1stNums.SelectedIndex = 0
33
34     End Sub
```

<- declares variable **intNum** with values **1, 2, ..., 25**

<- into **1stNums** adds **intNum** values

<- selects the value in first position with Index number of 0

-> save and test

<- notice the **frmMain\_Load** procedure adds the numbers from **1** through **25** to the list box and then selects the **first** number: **1**

### CH5\_A3.2 - ListBox - Items Collection: method **Insert** to add item at a desired position during run time

- allows you to **add** an item at a desired position in a **Ist** during run time

syntax: `objectListBox.Items.Insert(index, item)`      *index* = index of the item

*item* = the item's value

e.g.

```
Private Sub btnInsert_Click(sender As Object, e As EventArgs) Handles btnInsert.Click
    1stNames.Items.Insert(3, "Brett, Martin")      '_mycode_should add on 4th position my name
End Sub
```

### CH5\_A3.3 - ListBox - Items Collection: method Remove to remove item during run time

- allows you to **remove** an item from a **Ist** during run time

syntax: `objectListBox.Items.Remove(item)`      *item* = the item's value

e.g.

```
Private Sub btnRemove_Click(sender As Object, e As EventArgs) Handles btnRemove.Click
    lstNames.Items.Remove("Brett, Martin")           '_mycode_should remove my name from a lst
End Sub
```

### CH5\_A3.4 - ListBox - Items Collection: method RemoveAt to remove item during run time

- like the **Remove** method, it also allows you to **remove** an item from a **Ist** while an application is running, **however**
- however you specify the item's **index** rather than its value

syntax: `objectListBox.Items.RemoveAt(index)`      *index* = index of the item

e.g.

```
Private Sub btnRemoveAt_Click(sender As Object, e As EventArgs) Handles btnRemoveAt.Click
    lstNames.Items.RemoveAt(1)                      '_mycode_should remove 2nd name
End Sub
```

### CH5\_A3.5 - ListBox - Items Collection: property Count to get the number of items - e.g. 11.ListBox Items Solution

- stores an integer number that represents the number of **Ist** items

e.g. to use the **Count** property: ...VB2017\Chap05\Exercise\11.ListBox Items Solution\ListBox Items Solution.sln

-> locate the **btnCount\_Click** procedure  
-> click the blank line above the **End Sub** clause and then enter following assignment statement:

```
16      Private Sub btnCount_Click(sender As Object, e.....)
17          'Display the number of list box items.
18
19          lblItems.Text = lstNums.Items.Count.ToString
20      End Sub
```

-> save and test: click the button **Count**      <- the number **25** appears in the **Items** box

### CH5\_A3.6 - ListBox - Items Collection: method Clear to remove all of the items - e.g. 11.ListBox Items Solution

- for clearing (removing) the items from a **Ist**

syntax: `objectListBox.Items.Clear()`

e.g. to use the **Clear** method: ...VB2017\Chap05\Exercise\11.ListBox Items Solution\ListBox Items Solution.sln

-> in the **btnClear\_Click** procedure click the blank line above the **End Sub** clause and then enter the **2** statements:

```
10      Private Sub btnClear_Click(sender As Object, e.....)
11          'Clear the list box items.
12          lstNums.Items.Clear()
13          lblItems.Text = String.Empty
14      End Sub
```

<- from the **lstNums** removes all the **Items**

<- clears the content of the **lblItems**

-> save and test:

-> click the **Count** button

-> click the **Clear** button

-> click the **Count** button again

<- notice the **Items** box displays the number **25**

<- the method **Clear** removes all the items

<- notice the **Items** box displays the number **0**

### Mini-Quiz 5-6

1. Write an **Add** method that adds the contents of the **decPrice** variable to the **IstPrices** control
2. Write a statement that assigns the number of items in the **IstPrices** control to the **intNumPrices** variable
3. Write a statement that removes the items from the **IstPrices** control

```
1...IstPrices.Items.Add(decPrice)
2...intNumPrices = IstPrices.Items.Count
3...IstPrices.Items.Clear()
```

### CH5\_A4 - Financial Class - introduction: index of **Financial**. methods

- VB's **Financial class** contains many methods that your applications can use to perform financial calculations
- all of the methods return the result of their calculation as a **Double** number

#### **Financial**. methods:

|                                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                       |
|-----------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| calculate depreciation            | <b>Financial.DDB(cost, Salvage, Life, Period, Factor)</b><br>- the depreciation of an asset for a specific time period using the double-declining balance method or some other method you specify<br><b>Financial.SLN(cost, Salvage, Life)</b><br>- the straight-line depreciation of an asset for a single period<br><b>Financial.SYD(cost, Salvage, Life, Period)</b><br>- the sum-of-years digits depreciation of an asset for a specified period                                                  |
| calculate future value            | <b>Financial.FV(Rate, Nper, Pmt, PV, Due)</b><br>- the future value of an annuity based on periodic, fixed payments and a fixed interest rate                                                                                                                                                                                                                                                                                                                                                         |
| calculate interest rate           | <b>Financial.Rate(NPer, Pmt, PV, FV, Due, Guess)</b><br>- the interest rate per period for an annuity                                                                                                                                                                                                                                                                                                                                                                                                 |
| calculate internal rate of return | <b>Financial.IRR(valueArray[], Guess)</b><br>- the internal rate of return for a series of periodic cash flows (payments and receipts)<br><b>Financial.MIRR(ValueArray[], FinanceRate, ReinvestRate)</b><br>- the modified internal rate of returns for a series of periodic cash flows (payments and receipts)                                                                                                                                                                                       |
| calculate number of periods       | <b>Financial.NPer(Rate, Pmt, PV, FV, Due)</b><br>- the number of periods for an annuity based on periodic fixed payments and a fixed interest rate                                                                                                                                                                                                                                                                                                                                                    |
| calculate payments                | <b>Financial.IPmt(Rate, Per, NPer, PV, FV, Due)</b><br>- the interest payment for a given period of an annuity based on periodic, fixed payments and a fixed interest rate<br><b>Financial.Pmt(Rate, NPer, PV, FV, Due)</b><br>- the payment for an annuity based on periodic, fixed payments and a fixed interest rate<br><b>Financial.PPmt(Rate, Per, NPer, PV, FV, Due)</b><br>- the principal payment for a given period of an annuity based on periodic fixed payments and a fixed interest rate |
| calculate present value           | <b>Financial.NPV(Rate, ValueArray[])</b><br>- the net present value of an investment based on a series of periodic cash flows (payments and receipts) and a discount rate<br><b>Financial.PV(Rate, NPer, Pmt, FV, Due)</b><br>- the present value of an annuity based on periodic, fixed payments to be paid in the future and a fixed interest rate                                                                                                                                                  |

|                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                           |                     |                                                                                                                                                                                                                                                                                                                                      |
|--------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Cost</b>        | - the initial cost of the item                                                                                                                                                                                                                                                                                                                                                                                                                            | <b>Period</b>       | - the period of which asset depreciation is calculated                                                                                                                                                                                                                                                                               |
| <b>Due</b>         | <ul style="list-style-type: none"> <li>- object of type DueDate that specifies when payments are due</li> <li>- this argument must be either: <ul style="list-style-type: none"> <li>- <u>DueDate.EndOfPeriod</u> -&gt; if payments are due at the end of the payment period</li> <li>or <u>DueDate.BegOfPeriod</u> -&gt; if payments are due at the beginning of the period</li> <li>- if omitted, DueDate.EndOfPeriod is assumed</li> </ul> </li> </ul> | <b>Pmt</b>          | <ul style="list-style-type: none"> <li>- the payment to be made each period</li> <li>- payments usually contain principal and interest that doesn't change over the life of the annuity</li> </ul>                                                                                                                                   |
| <b>Factor</b>      | <ul style="list-style-type: none"> <li>- the rate at which the balance declines</li> <li>- if omitted, 2 (double-declining method) is assumed</li> </ul>                                                                                                                                                                                                                                                                                                  | <b>PV</b>           | <ul style="list-style-type: none"> <li>- the present value (or lump sum, of value today) of a series of future payments or receipts</li> <li>- e.g.: when you borrow money to buy a car, the loan amount is the present value to the lender of the monthly car payments you will make</li> <li>- if omitted, 0 is assumed</li> </ul> |
| <b>FinanceRate</b> | - the interest rate paid as the cost of financing                                                                                                                                                                                                                                                                                                                                                                                                         | <b>Rate</b>         | <ul style="list-style-type: none"> <li>- the interest rate per period</li> <li>- e.g.: if you get a car loan at an annual percentage rate (APR) of 10% and make monthly payments, the rate per period is 0.1/12, or 0.0083</li> </ul>                                                                                                |
| <b>FV</b>          | <ul style="list-style-type: none"> <li>- the future value or cash balance you want after you've made the final payment</li> <li>- e.g.: the future value of a loan is \$0 because that's its value after the final payment, however, if you want to save \$50,000 over 18 years for your child's education, then \$50,000 is the future value</li> <li>- if omitted, 0 is assumed</li> </ul>                                                              | <b>ReinvestRate</b> | - the interest rate received on gains from cash reinvestment                                                                                                                                                                                                                                                                         |
| <b>Guess</b>       | <ul style="list-style-type: none"> <li>- object specifying value you estimate will be returned by IRR</li> <li>- if omitted, Guess is 0.1 (10%)</li> </ul>                                                                                                                                                                                                                                                                                                | <b>Salvage</b>      | - the value of the asset at the end of its useful life                                                                                                                                                                                                                                                                               |
| <b>Life</b>        | - the length of useful life of the asset                                                                                                                                                                                                                                                                                                                                                                                                                  | <b>ValueArray</b>   | <ul style="list-style-type: none"> <li>- array of Double[] specifying cash flow values</li> <li>- the array must contain at least one negative value (a payment) and one positive value (a receipt)</li> </ul>                                                                                                                       |
| <b>NPer</b>        | <ul style="list-style-type: none"> <li>- the total number of payment periods in the annuity</li> <li>- e.g.: if you make monthly payments on a 4 year car loan, your loan has a total of 48 (or 4x12) payment periods</li> </ul>                                                                                                                                                                                                                          |                     |                                                                                                                                                                                                                                                                                                                                      |
| <b>Per</b>         | - the payment period in the range 1 through Nper                                                                                                                                                                                                                                                                                                                                                                                                          |                     |                                                                                                                                                                                                                                                                                                                                      |

**anuity** - a series of fixed cash payments made over a period of time, can be a loan (e.g. home mortgage) or an investment (e.g. a monthly savings plan)

**asset** - (aktyva, jmění) - possessions, valuables

**discount rate** - (diskontní sazba) - interest rate used in **DCF** - discounted cash flow analysis to determine the present value of future cash flows

**interest rate** - (úrok) - the amount charged, expressed as a % of principal, by a lender to a borrower for the use of assets

**internal rate of return / IRR** - (vnitřní míra návratnosti) - used in capital budgeting to estimate the profitability of potential investments

**principal** - the loan amount

**principal payment** - a payment made to pay back all loan or part of a loan, rather than to pay interest on the loan

**single period / short term / one-off** - within 3 months

**straight-line depreciation** - the default method used to recognize the carrying amount of a fixed asset evenly over its useful life

**sum-of-years digits depreciation / SYD** - used to accelerate the recognition of depreciation (devalvace);

- most of the depreciation associated with an asset is recognized in the first few years of its useful life

**term** - the number of years the borrower has to pay off the loan (doba trvání smlouvy / pojistky / půjčky)

## CH5\_A4.1 - Financial Class - calculate a periodic payment using `Financial.Pmt` method

- the Monthly Payment application, which you will code in the next section, uses the `Financial.Pmt` method to calculate a monthly mortgage payment
- all of the methods return the result of their calculation as a **Double** number

syntax:

```
Financial.Pmt(Rate, NPer, PV, FV, Due)
```

- arguments:
- Rate** interest **rate** per period
  - NPer** number of **periods** -> total number of payment periods in the term
  - PV** present value of the loan (the principal)
    - the **Rate** and **NPer** arguments must be expressed using the same units (month, year):
    - i.e.: If **Rate** is a monthly interest rate, then **NPer** must specify the number of monthly payments
    - i.e.: If **Rate** is an annual interest rate, then **NPer** must specify the number of annual payments

e.g.1

```
Financial.Pmt(0.05, 3, 9000)
```

- calculates the annual payment for a loan of **\$9,000** for **3** years with a **5%** interest rate
- Rate is 0.05, NPer is 3, PV is 9000
- the annual payment returned by the method (rounded to the nearest cent) is - 3304.88
- you can change the negative number to a positive by preceding the method with the negation operator :  
**-Financial.Pmt(0.05, 3, 9000)**

e.g.2

```
-Financial.Pmt(0.03 / 12, 15 * 12, 125000)
```

- calculates the monthly payment for a loan of **\$125,000** for **15** years with a **3%** interest rate
- Rate is 0.03 / 12, NPer is 15 \* 12, PV is 125000
- the monthly payment returned by the method (rounded to the nearest cent and expressed as a positive number) is 863.23
- the Rate and NPer arguments are expressed in monthly terms rather than in annual terms ->
  - > you change an annual rate to a monthly rate by dividing the annual rate by 12
  - > you change the term from years to months by multiplying the number of years by 12

### Mini-Quiz 5-7

1. Write the `Financial.Pmt` method that calculates the monthly payment (expressed as a positive number) for a loan of \$25,000 for 10 years with a 4% annual interest rate.
2. Write the `Financial.Pmt` method that calculates the monthly payment (expressed as a positive number) for a loan of \$130,000 for 30 years with a 2% annual interest rate.
3. The `Financial.Pmt` method returns a number of which data type?

You Do It 4: 12.

1. create an app named You Do It 4
2. add a **lbl** and **btn**
3. the **btn**'s Click event procedure should calculate the monthly payment for a loan of \$50,000 for 3 years with a 5% annual interest rate, and then display the result in the **lbl**
  - display the monthly payment as a positive number with a dollar sign and 2 decimal places
  - test: the monthly payment is \$1,498.54

```
Dim dblShow As Double  
dblShow = -Financial.Pmt(0.05 / 12, 3 * 12, 50000)  
lblShow.Text = dblShow.ToString("C2")
```

```
1...-Financial.Pmt(0.04 / 12, 10 * 12, 25000)  
2...-Financial.Pmt(0.02 / 12, 30 * 12, 130000)  
3...-Double
```

## CH5\_A4.2 - Financial.Pmt method & ListBox & Loop to calculate monthly mortgage payment - e.g. 13.Payment Solution

- the Monthly Payment application will display the monthly payments on a mortgage loan, using **terms** of 15, 20, 25, and 30 years
- the **term** is the number of years the borrower has to pay off the loan (doba trvání smlouvy/pojistky/půjčky)
- the user will enter the **loan amount**, called the **principal**, in a **TextBox**
- the user will select the **interest rate** from a **Ist** that contains rates ranging from **2.0%** to **7.0%** in increments of **0.5%**

e.g. to code and then test the **Monthly Payment** app:

...VB2017\Chap05\Exercise\13.Payment Solution\Payment Solution.sln

- > in the Code Editor window, which already contains the code for many of the app's procedures
  - > locate the **frmMain\_Load** procedure and click the blank line above the **End Sub** clause
  - > enter the **For...Next** loop and assignment statement shown below
  - > be sure to change the **Next** clause to **Next dblRates**

```
11     Private Sub frmMain_Load(sender As Object, e.....  
12         'Fill list box with rates and select 3.0 rate.  
13  
14         For dblRates As Double = 2 To 7 Step 0.5  
15             1stRates.Items.Add(dblRates.ToString("N1"))  
16         Next dblRates  
17         1stRates.SelectedItem = "3.0"  
18  
19     End Sub
```

- > save and test:

- > verify that the **Ist** contains the appropriate rates and that the **3.0** rate is selected
- > exit

- > locate the **btnCalc\_Click** procedure and click the blank line above the **End Sub** clause

- > first, you will enter a **TryParse** method that assigns the rate selected in the **Ist** to **dblRate** variable
- > row **29**: enter the **TryParse** method
- > row **30**: next, you will convert the rate to its decimal equivalent by dividing it by 100
- > row **32**: before calculating and displaying the monthly payments, you will clear any previous payments from the **lblPay** control
- > row **33-39**: finally, you will enter a **For...Next** loop that calculates and displays the monthly payments for terms of 15, 20, 25, and 30 years

```
21     Private Sub btnCalc_Click(sender As Object, e.....  
22         'Display the monthly mortgage payment.  
23  
24         Dim intPrincipal As Integer  
25         Dim dblRate As Double  
26         Dim dblPay As Double  
27  
28         Integer.TryParse(txtPrincipal.Text, intPrincipal)  
29         Double.TryParse(1stRates.SelectedItem.ToString, dblRate)  
30         dblRate = dblRate / 100  
31  
32         lblPay.Text = String.Empty
```

<- or **dblRate /= 100**

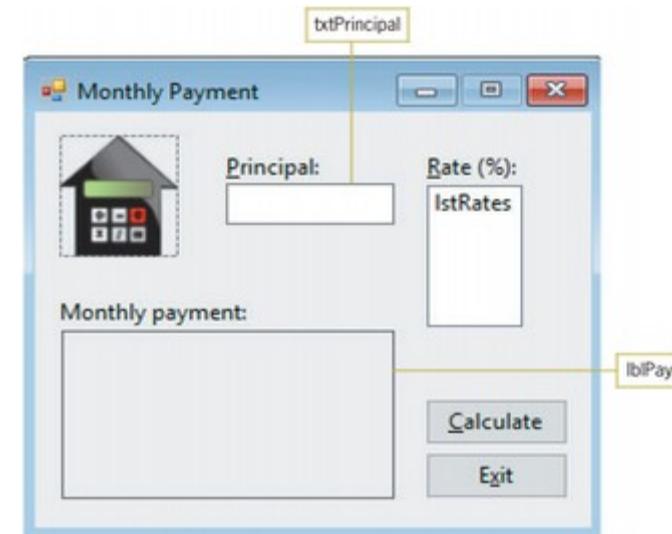


Figure 5-33 Monthly Payment application's interface

```

33     For intTerm As Integer = 15 To 30 Step 5
34         dblPay = -Financial.Pmt(dblRate / 12,
35                                 intTerm * 12, intPrincipal)
36         lblPay.Text = lblPay.Text & intTerm.ToString &
37             " years: " & dblPay.ToString("C2") &
38             ControlChars.NewLine
39     Next intTerm
40 End Sub

```

-> open the code template for the **lstRates\_SelectedIndexChanged** procedure  
 -> you will have this procedure clear the contents of the **lblPay** control  
 when a change is made to the rate selected in the **lstRates** control

```

62 Private Sub lstRates_SelectedIndexChanged(sender As Object, e.....)
63     lblPay.Text = String.Empty
64 End Sub

```

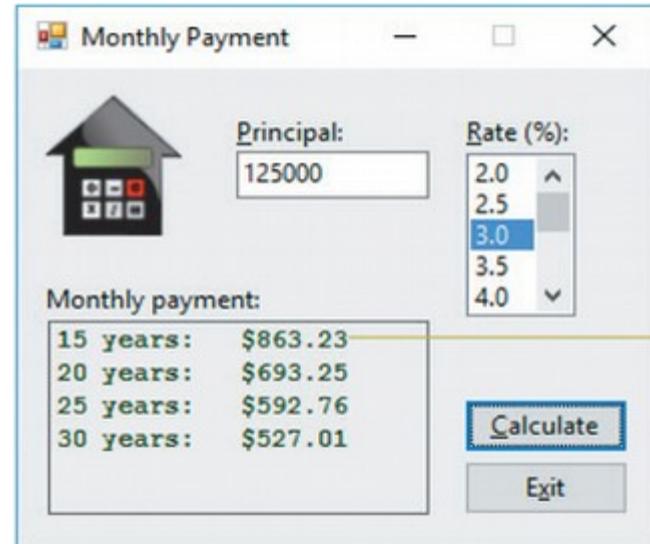
-> save and test:  
 -> in the **Principal** box type: **125000** and then click the **Calculate** button  
 <- the monthly mortgage payments appear in the interface, as shown in **Figure 5-37**:

entire code:

```

11 Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
12     ' Fill list box with rates and select 3.0 rate.
13
14     For dblRates As Double = 2 To 7 Step 0.5
15         lstRates.Items.Add(dblRates.ToString("N1"))
16     Next dblRates
17     lstRates.SelectedItem = "3.0"
18
19 End Sub
20
21 Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
22     ' Display the monthly mortgage payment.
23
24     Dim intPrincipal As Integer
25     Dim dblRate As Double
26     Dim dblPay As Double
27
28     Integer.TryParse(txtPrincipal.Text, intPrincipal)
29     Double.TryParse(lstRates.SelectedItem.ToString, dblRate)
30     dblRate /= 100
31
32     lblPay.Text = String.Empty

```



**Figure 5-37**

```
33     For intTerm As Integer = 15 To 30 Step 5
34         dblPay = -Financial.Pmt(dblRate / 12,
35                                 intTerm * 12, intPrincipal)
36         lblPay.Text = lblPay.Text & intTerm.ToString &
37             " years: " & dblPay.ToString("C2") &
38             ControlChars.NewLine
39     Next intTerm
40 End Sub
41
42 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
43     Me.Close()
44 End Sub
45
46 Private Sub txtPrincipal_Enter(sender As Object, e As EventArgs) Handles txtPrincipal.Enter
47     txtPrincipal.SelectAll()
48 End Sub
49
50 Private Sub txtPrincipal_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtPrincipal.KeyPress
51     ' Accept only numbers and the Backspace key.
52
53     If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
54         e.Handled = True
55     End If
56 End Sub
57
58 Private Sub txtPrincipal_TextChanged(sender As Object, e As EventArgs) Handles txtPrincipal.TextChanged
59     lblPay.Text = String.Empty
60 End Sub
61
62 Private Sub lstRates_SelectedIndexChanged(sender As Object, e As EventArgs) Handles lstRates.SelectedIndexChanged
63     lblPay.Text = String.Empty
64 End Sub
```

- **Financial.FV** method is used to calculate the future value of an annuity based on periodic, fixed payments and a fixed interest rate

syntax:

**Financial.FV(Rate, NPer, Pmt, PV, Due)**

**As Double**

- Rate** = interest rate per period (period must be of same units)
  - e.g. if you get a car loan at an annual percentage rate (APR) of 10% and make monthly payments, the rate per period is  $0.1 / 12$  (0.0083)
- NPer** = total number of payment periods in the annuity (period must be of same units)
  - e.g. if you make monthly payments on a 4 year car loan, your loan has a total of  $4 * 12$  (48) payment periods
- Pmt** = payment to be made each period (period must be of same units)
  - payments usually contain principal and interest that doesn't change over the life of the annuity
- PV**
  - optional, if omitted, **0** is assumed
  - the present value (or lump sum, of value today) of a series of future payments or receipts
  - when calculating savings, use **0**
  - e.g. when you borrow money to buy a car, the loan amount is the present value to the lender of the monthly car payments you will make
- Due**
  - optional, if omitted, **DueDate.EndOfPeriod** is assumed
  - object of type **DueDate** that specifies when payments are due
  - argument must be either:
    - DueDate.EndOfPeriod** = if payments are due at the end of the payment period
    - DueDate.BegOfPeriod** = if payments are due at the beginning of the period

- **compound interest & compounding period:** (source: <http://www.math.hawaii.edu/~ramsey/CompoundInterest.html>)

- bank deposits, over time, usually have **compound interest**
- interest is computed on an account such as savings account or a checking account and the interest is added to the account
- because the interest is added to the account, the interest itself earns interest during the next time period for computing interest -> the interest compounds
- the time interval between the occasions at which interest is added to the account is called **the compounding period**
- compounding period can be: daily, monthly, quarterly (trimester), semiannually, annually

| Compounding period  |                     |                    |                                                                |                                                                |
|---------------------|---------------------|--------------------|----------------------------------------------------------------|----------------------------------------------------------------|
| descriptive adverb: | fraction of 1 year: | calculation:       | e.g. with a nominal annual rate 6% interest rate for 1 period: | e.g. with a nominal annual rate 3% interest rate for 1 period: |
| daily               | $1/365 *$           | $(\% / 100) / 365$ | $(6 / 100) / 365 = 0.06 / 365 = \mathbf{0.000164384}$          | $(3 / 100) / 365 = 0.03 / 365 = \mathbf{0.000082191780}$       |
| monthly             | $1/12$              | $(\% / 100) / 12$  | $(6 / 100) / 12 = 0.06 / 12 = \mathbf{0.005}$                  | $(3 / 100) / 12 = 0.03 / 12 = \mathbf{0.0025}$                 |
| quarterly           | $1/4$               | $(\% / 100) / 4$   | $(6 / 100) / 4 = 0.06 / 4 = \mathbf{0.015}$                    | $(3 / 100) / 4 = 0.03 / 4 = \mathbf{0.0075}$                   |
| semiannually        | $1/2$               | $(\% / 100) / 2$   | $(6 / 100) / 2 = 0.06 / 2 = \mathbf{0.03}$                     | $(3 / 100) / 2 = 0.03 / 2 = \mathbf{0.015}$                    |
| annually            | $1$                 | $(\% / 100) / 1$   | $(6 / 100) / 1 = 0.06 / 1 = \mathbf{0.06}$                     | $(3 / 100) / 1 = 0.03 / 1 = \mathbf{0.03}$                     |

\* ignoring leap years, which have 366 days

## Savings Calculator (Chapters 1–5)

Research Visual Basic's Financial.FV (Future Value) method. Create an application that allows the user to enter the amount a customer plans to deposit in a savings account each month, and whether the money will be deposited at either the beginning or the end of the month. The application should calculate and display the value of the account at the end of 5 years, 10 years, 15 years, 20 years, and 25 years. The interest rate is 3% and is compounded monthly.

- using .NET Framework 4.8

```
1  ' Name:      Savings Calculator CH 01-05.
2  ' Purpose:    Research Financial.FV (Future Value) method and use it to calculate and display savings.
3  ' Programmer: me on just now.
4  Option Explicit On
5  Option Strict On
6  Option Infer Off
7
8  Public Class frmMain
9      Private Sub btnCalculate_Click(sender As Object, e As EventArgs) Handles btnCalculate.Click
10         ' Calculate and display the value of the account at the end of: 5, 10, 15, 20, 25 years.
11
12         ' Interest rate = 3% = 0.03 and is compounded monthly ->
13         ' -> id est: interest is added to the principal every month, so in the next period,
14         ' the interest is computed from the principal + interest, etc...
15         ' Because the interest is added to the account, the interest itself earns interest during the next time period for computing interest.
16
17         ' Financial.FV(Rate, Nper, Pmt, PV, Due) As Double - PV & Due are optional
18         ' Rate = interest rate per period (if annually 10% then the rate per period is 0.1/12 = 0.0083) -> 3% compounded monthly, so 0.0025.
19         ' Nper = total number of payment periods in the annuity (number of years * 12)                                -> 12 * intCounter(5, 10, 15, 20, 25).
20         ' Pmt = payment to be made each period   -> here user input
21         ' PV = optional, present value of a series of future payments                                     -> here 0, because its saving.
22         ' Due = optional, when payments are due: - either DueDate.EndOfPeriod = if payments are due at the end of the period.
23         '   DueDate.BegOfPeriod = if payments are due at the beginning of the period.
24
25         ' input: txtDeposit; output: txtBalance; rad1Beginning, rad2End
26
27         Dim dblDeposit As Double          ' input.
28         Dim dblBalance As Double         ' output.
29         Const dblRate3 As Double = 0.03   ' 3% = 0.03. 0.03 / 12 = 0.0025 monthly compounded
30         Double.TryParse(txtDeposit.Text, dblDeposit)
31
32         txtBalance.Text = Nothing       ' clear output.
33
```

```

34     If dblDeposit > 0 Then      ' s.s. to check a valid input.
35         For intCounter As Integer = 5 To 25 Step 5
36
37             If rad1Beginning.Checked Then
38                 ' be aware to use the same units - months, years...
39                 dblBalance = -Financial.FV(dblRate3 / 12, 12 * intCounter, dblDeposit, 0, DueDate.BegOfPeriod)
40             Else
41                 dblBalance = -Financial.FV(dblRate3 / 12, 12 * intCounter, dblDeposit, 0, DueDate.EndOfPeriod)
42             End If
43
44             ' output:
45             txtBalance.Text &= intCounter.ToString & ControlChars.Tab & dblBalance.ToString("C2") & ControlChars.NewLine
46
47         Next intCounter
48
49         ' select input when operation terminates.
50         txtDeposit.SelectAll()
51
52     Else      ' if the input number is invalid.
53         MessageBox.Show("Please enter a valid number.", "Savings Calculator", MessageBoxButtons.OK, MessageBoxIcon.Exclamation)
54     End If
55
56 End Sub
57
58 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
59     Me.Close()
60 End Sub
61
62 Private Sub txtDeposit_Enter(sender As Object, e As EventArgs) Handles txtDeposit.Enter
63     txtDeposit.SelectAll()
64 End Sub
65
66 ' allow only decimal numbers and Backspace key:
67 Private Sub txtDeposit_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtDeposit.KeyPress
68     If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> "." AndAlso e.KeyChar <> ControlChars.Back Then
69         e.Handled = True
70     End If
71 End Sub
72
73 Private Sub txtDeposit_TextChanged(sender As Object, e As EventArgs) Handles txtDeposit.TextChanged
74     txtBalance.Text = Nothing
75 End Sub
76

```

```

77      Private Sub rad12_CheckedChanged(sender As Object, e As EventArgs) Handles rad1Beginning.CheckedChanged, rad2End.CheckedChanged
78          txtBalance.Text = Nothing
79      End Sub
80
81      Private Sub txtDeposit_MouseClick(sender As Object, e As MouseEventArgs) Handles txtDeposit.MouseClick
82          txtDeposit.SelectAll()
83      End Sub
84  End Class

```

The screenshot shows the '4.Savings' application window. In the 'Deposit:' field, the value '100' is entered. Below it, the note '(Interest rate 3% compounded monthly)' is displayed. Under the 'Beginning of the month' radio button, which is selected, the 'Calculate' button is highlighted with a blue border. The 'Exit' button is unhighlighted. At the bottom, a table titled 'Years: Balance:' lists values for years 5, 10, 15, 20, and 25.

| Years | Balance      |
|-------|--------------|
| 5     | \$ 6,480.83  |
| 10    | \$ 14,009.08 |
| 15    | \$ 22,754.01 |
| 20    | \$ 32,912.28 |
| 25    | \$ 44,712.28 |

| no  | year | future value<br>FV | interest  | effective<br>rate |
|-----|------|--------------------|-----------|-------------------|
| 1   |      | <b>100.25</b>      | 0.25      | 0.25 %            |
| 2   |      | <b>200.75</b>      | 0.75      | 0.375 %           |
| 3   | 1/4  | <b>301.50</b>      | 1.50      | 0.501 %           |
| 4   |      | <b>402.51</b>      | 2.51      | 0.627 %           |
| 5   |      | <b>503.76</b>      | 3.76      | 0.753 %           |
| 6   | 1/2  | <b>605.27</b>      | 5.27      | 0.879 %           |
| 7   |      | <b>707.04</b>      | 7.04      | 1.005 %           |
| 8   |      | <b>809.05</b>      | 9.05      | 1.132 %           |
| 9   | 3/4  | <b>911.33</b>      | 11.33     | 1.258 %           |
| 10  |      | <b>1,013.85</b>    | 13.85     | 1.385 %           |
| 11  |      | <b>1,116.64</b>    | 16.64     | 1.513 %           |
| 12  | 1    | <b>1,219.68</b>    | 19.68     | 1.64 %            |
| 60  | 5    | <b>6,480.83</b>    | 480.83    | 8.014 %           |
| 120 | 10   | <b>14,009.08</b>   | 2,009.08  | 16.742 %          |
| 180 | 15   | <b>22,754.01</b>   | 4,754.01  | 26.411 %          |
| 240 | 20   | <b>32,912.28</b>   | 8,912.28  | 37.134 %          |
| 300 | 25   | <b>44,712.28</b>   | 14,712.28 | 49.041 %          |

The screenshot shows the '4.Savings' application window. In the 'Deposit:' field, the value '100' is entered. Below it, the note '(Interest rate 3% compounded monthly)' is displayed. Under the 'End of the month' radio button, which is selected, the 'Calculate' button is highlighted with a blue border. The 'Exit' button is unhighlighted. At the bottom, a table titled 'Years: Balance:' lists values for years 5, 10, 15, 20, and 25.

| Years | Balance      |
|-------|--------------|
| 5     | \$ 6,464.67  |
| 10    | \$ 13,974.14 |
| 15    | \$ 22,697.27 |
| 20    | \$ 32,830.20 |
| 25    | \$ 44,600.78 |

## CH5\_A5 - Nested repetition structures / loops = outer loop contains nested/inner loop - basic info

- like selection structures, **repetition structures** can be nested - i.e. you can place one loop called **nested/inner** within another loop called the **outer** loop
- both loops can be:
  - **pretest loops**      **Do While/Do Until...Loop, For...Next**
  - **posttest loops**      **Do...Loop While/Loop Until**
  - **pretest loop** and **posttest loop** = mixed

e.g. a clock uses **Nested Loops** to keep track of the time

- for simplicity, consider a clock's minute and second hands only
- the second hand on a clock moves one position, clockwise, for every second that has elapsed
- after the second hand moves 60 positions, the minute hand moves one position, also clockwise
- the second hand then begins its journey around the clock again

- the logic used by a clock's minute and second hands:

```
repeat for minutes from 0 to 59      <- start outer loop
    repeat for seconds from 0 to 59   <- start nested loop
        move second hand 1 position, clockwise    <- action for nested loop
    end repeat for seconds           <- end nested loop
    move minute hand 1 position, clockwise    <- action for outer loop
end repeat for minutes             <- end outer loop
```

- my code, i think its correct

```
For intMinute As Integer = 0 To 59 Step 1
    For intSecond As Integer = 0 To 59 Step 1
        intSecondHand +=1
    Next intSecond
    intMinuteHand +=1
Next intMinute
```

- as the figure indicates, an **outer** loop controls the **minute** hand, while the **inner/nested** loop controls the **second** hand

- notice that the entire nested loop is contained within the outer loop -> this must be **TRUE** for the loop to be nested and for it to work correctly
- the next iteration (opakování, iterace) of the outer loop (which controls the minute hand) occurs only after the nested loop (which controls the second hand) has finished processing

## CH5\_A5.1 - Nested repetition structures / loops = outer loop contains nested/inner loop - e.g. using 2x Pretest loop For...Next, e.g. 14.Savings Solution

- the **Savings Account application**, which you will code in this section, displays the balance in a savings account at the end of each of 5 years, based on an initial deposit and rates from 3% to 7%

- the formula for calculating the balance is:  $\text{deposit} * (1 + \text{rate})^{\text{year}}$

pseudocode for btnCalc\_Click procedure:

```
1. declare dblDeposit and dblBalance variables
2. convert txtDeposit.Text to a number and store in dblDeposit variable
3. display Rate, Year, and Balance column headings in txtBalance
4. repeat for dblRate from 3% to 7% in increments of 1%
   display dblRate
   repeat for intYear from 1 to 5 in increments of 1
      dblBalance = dblDeposit * (1 + dblRate) ^ intYear
      display intYear and dblBalance in txtBalance
   end repeat for intYear
end repeat for dblRate
```

<- outer loop starts

<- nested loop starts

<- nested loop ends

<- outer loop ends

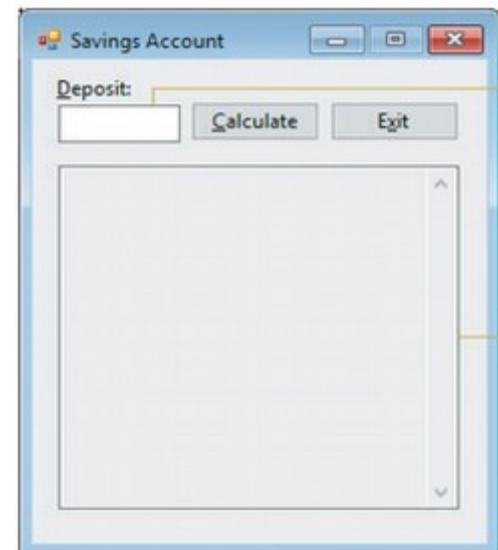
- as the pseudocode indicates, the **btnCalc\_Click** procedure requires **2 loops**, one **nested** within the other:

- the **outer loop** controls the rates, which range from 3% to 7% in increments of 1%
- the **inner loop** controls the years, which range from 1 to 5 in increments of 1

- when displaying the information in the **txtBalance** control, the procedure will use the **ControlChars.Tab** constant, which represents the **Tab** key, to align the information

- the **txtBalance** control's **Properties**:
  - AutoCompleteCustomSource **(Collection)**
  - BorderStyle **Fixed3D**
  - Cursor **IBeam** (cursor like for a text)
  - Lines **String[] Array**
  - Multiline **True**
  - ReadOnly **True**
  - ScrollBars **Vertical**

GUI:



e.g. to code and then test the **Nested loop** using 2xPretest loop **For...Next**: ...VB2017\Chap05\Exercise14.Savings Solution\Savings Solution.sln

-> in the Code Editor window locate the **btnCalc\_Click** procedure

- the first 3 steps in the pseudocode have already been coded for you

-> click the blank line above the **End Sub** clause and then enter the **outer** and **nested** loops shown below:

```
12     Private Sub btnCalc_Click(sender As Object, e.....  
13         'Calculate account balances for each of five years  
14         ' using rates from 3% to 7% in increments of 1%.  
15  
16         Dim dblDeposit As Double  
17         Dim dblBalance As Double  
18  
19         Double.TryParse(txtDeposit.Text, dblDeposit)  
20  
21         txtBalance.Text = "Rate" & ControlChars.Tab &  
22             "Year" & ControlChars.Tab & "Balance" &  
23             ControlChars.NewLine  
24  
25         'Calculate and display account balances:  
26         For dblRate As Double = 0.03 To 0.07 Step 0.01  
27             txtBalance.Text = txtBalance.Text &  
28                 dblRate.ToString("P0") & ControlChars.NewLine  
29             For intYear As Integer = 1 To 5  
30                 dblBalance = dblDeposit * (1 + dblRate) ^ intYear  
31                 txtBalance.Text = txtBalance.Text &  
32                     ControlChars.Tab & intYear.ToString &  
33                     ControlChars.Tab & dblBalance.ToString("C2") &  
34                     ControlChars.NewLine  
35             Next intYear  
36         Next dblRate  
37     End Sub
```

<- **ControlChars.Tab** = Tab key

<- **ControlChars.NewLine** = Enter key

<- if optional **Step** is omitted, number **1** is used

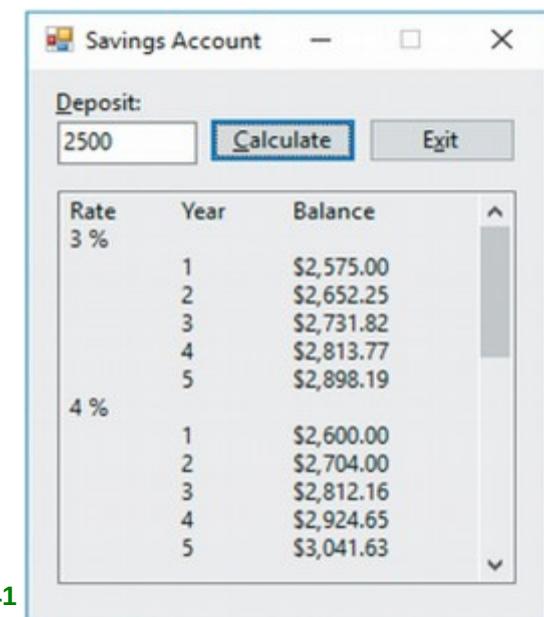


Figure 5-41

-> save and test: in the **Deposit** box type: **2500** and then click the **Calculate** button

- see **Figure 5-41**: Account balances shown in the interface

- scroll the **txt** to verify that it contains the account balances for each of the 5 rates

## CH5\_A6 - a caution about **Real numbers** = decimals: **As Double** less accurate than **As Decimal** -> can cause wrong result, e.g. 15.Savings Solution-Caution

- not all **real numbers**, which are numbers with a decimal place, can be stored **precisely** in the RAM
- many can be stored only as an **approximation** (přiblžení), which may lead to unexpected results when 2 real numbers are compared with each other

e.g. sometimes a **Double** number that is the result of a calculation does not compare **precisely** with the same number expressed as a **literal**

- this is why it is so important to test your application's code thoroughly
- solution:
  - a). increase the literal's value slightly      e.g. instead of **0.06** use **0.0600001**
  - b). use the **Decimal** data type

- in the next set of steps, you will observe how the comparison problem would affect the **Savings Account application** from the previous section:

e.g. to modify the **Savings Account application** from the previous section: ...VB2017\Chap05\Exercise\15.Savings Solution-Caution\Savings Solution.sln

-> copy the ...VB2017\Chap05\Exercise\14.Savings Solution and then rename the copy to: 15.Savings Solution-Caution

-> open the ...VB2017\Chap05\Exercise\15.Savings Solution-Caution\Savings Solution.sln

-> open the Code Editor window:

- you will modify the application, so that it uses rates from **3%** to **6%**, rather than to **7%**

-> change: 14                          ' using rates from 3% to 7% in increments of 1%.

-> for: 14                          ' using rates from 3% to 6% in increments of 1%.

-> change: 26                          For dblRate As Double = 0.03 To 0.07 Step 0.01

-> for: 26                          For dblRate As Double = 0.03 To 0.06 Step 0.01

-> save and test:

-> click the **Calculate** button and then scroll to the bottom of the **txtBalance** control

<- notice: as shown in **Figure 5-42**, the information associated with the **6%** rate is **missing** from the control

| 5 % |        |  |
|-----|--------|--|
| 1   | \$0.00 |  |
| 2   | \$0.00 |  |
| 3   | \$0.00 |  |
| 4   | \$0.00 |  |
| 5   | \$0.00 |  |

**Figure 5-42**  
**txtBalance** control with the  
6% rate information missing

- consider why the loop that controls the rates **failed** to display the **6%** rate information:

- recall that the **For** clause in that loop tells the computer to **stop** processing the loop instructions when the value in the **dblRate** variable is greater than **0.06**

26                          For dblRate As Double = 0.03 To 0.06 Step 0.01

- the fact that the **6%** rate information did not display indicates that when the **For** clause compares the variable's value with the literal **0.06**,  
the value is viewed as **greater** than the literal -> this causes the loop to end prematurely

- **to fix this problem you can either:** a). increase the literal's value slightly - you can use **0.0600001**

26                          For dblRate As Double = 0.03 To 0.0600001 Step 0.01

b). use the **Decimal** data type for the loop that controls the **rates**

26                          For decRate As Decimal = 0.03D To 0.06D Step 0.01D

- don't forget to change all **dblRate** for **decRate** (lines **26, 28, 30, 36**)

## CH5\_A7 - professionalize your application's interface: the **default buttons** set in **frmMain**'s Properties: **AcceptButton** = Enter key, **CancelButton** = Esc key

- as you already know, you can select a **btn** during run time either by:
    - 1). clicking it by mouse or
    - 2). pressing the **Enter** key when the **btn** has the focus
  - if you make a **btn** the **default button**, you can also select it by pressing the **Enter** key even when the button does not have the focus
  - when a **btn** is selected, the computer processes the code contained in the **btn\_Click** event procedure
  - a GUI does not have to have a **default button**
  - if one is used, it should be the **btn** that is most often selected by the user, except in cases
    - where the tasks performed by the **btn** are both **destructive** and **irreversible**
  - e.g.: - a **btn** that deletes information should not be designated as the **default button** unless the application provides a way for the information to be restored
  - if you assign a **default button** in a GUI, it typically is the **first btn** when the buttons are positioned horizontally, or stacked vertically
  - a form can have only **one** default button
- you specify the default button (if any) by setting the **form's property**:
- **AcceptButton** property for **Enter** key
  - **CancelButton** property for **Esc** key

e.g. to designate a **default button** in the **Savings Account application** from the previous section:

- > open the ...VB2017\Chap05\Exercises\14.Savings Solution\Savings Solution.sln
- > display the **frmMain**'s properties in the **Properties** window
  - > set the **AcceptButton** property to **btnCalc**
  - > set the **CancelButton** property to **btnExit**

<- notice a darkened border now appears around the **Calculate** button

- > save and test:
  - > in the **Principal** box type **5000** and then press **Enter** key
    - <- the computer processes the code contained in the **btnCalc\_Click** procedure, and the savings account balances appear in the **txtBalance** control
  - > press anytime the **Esc** key and the application will **Exit**

### Mini-Quiz 5-8

1. Only the **For...Next** statement can be used to code a **nested** loop. True or False?
2. Which of a **text box**'s properties determines whether the **txt** can accept either one or multiple lines of text?
3. Which **constant** is used to represent the **Tab** key?
4. Which of a **form**'s properties specifies the **default button** (if any)?

1...False, any loop can be used  
2...Multiline = True/False  
3...ControlChars.Tab  
4...AcceptButton, CancelButton

## CH5\_Key Terms

- & - the **ampersand** - the concatenation (zřetězení) operator in VB = like string + string
- **AcceptButton property** - a property of a form - used to designate the default button that can be selected by pressing the **Enter** key
- **Accumulator** - a numeric variable used for accumulating something = adding together
- **Add method** - the Items collection's method used to add an item to a **Ist**
- **Block scope** - scope of the variable declared in a **For...Next** statement's **For** clause
  - indicates that the variable can be used only within the **For...Next** loop
- **CancelButton property** - a property of a form - used to designate the default button that can be selected by pressing the **Esc** key
- **Clear method** - the Items collection's method used to clear/remove items from a **Ist**
- **Collection** - a group of individual objects treated as one unit
- **Concatenation operator** - & - the **ampersand** - used to concatenate strings = like string + string
- **ControlChars.NewLine constant** - advances the insertion point to the next line = like Enter in a text document
- **ControlChars.Tab constant** - represents the Tab key = align the information
- **Count property** - **Ist** property - stores an integer that represents the number of items in the Items collection
- **Counter** - a numeric variable used for counting something
- **Counter-controlled loop** - a loop whose processing is controlled by a counter
  - the loop body will be processed a precise number of times
- **Decrementing** - decreasing a value; *lat.opozitum* = Incrementing
- **Default button** - the **btn** that can be selected by pressing the **Enter** key even when the button does not have the focus
  - designated by setting the form's **AcceptButton** property
- **Default List Box item** - the item automatically selected in a **Ist** when the application is started and the GUI appears
  - Items collection's **Add** method entered in a **Load event procedure**
- **Dictionary order** - numbers are sorted before letters, and a lowercase letter is sorted before its uppercase equivalent
- **Do...Loop statement** - a VB statement that can be used to code both pretest loops and posttest loops
- **Endless loop** - a loop whose instructions are processed indefinitely - also called an infinite loop
- **Financial. method** - used for financial calculations
- **Financial.Pmt method** - used to calculate the periodic payment on either a loan or an investment
- **For...Next statement** - a VB statement that is used to code a specific type of pretest loop, called a **counter-controlled loop**
- **Incrementing** - increasing a value; *lat.opozitum* = Decrementing
- **Infinite loop** - another name for an endless loop
- **Items collection** - the collection of items in a **Ist**
- **List box - Ist** - a control used to display a list of items from which the user can select 0, 1, or multiple items
- **Load event** - an event associated with a form - occurs when the application is started and the form is displayed the first time
- **Loop** - another name for the repetition structure
- **Loop exit condition** - the requirement that must be met for the computer to *stop* processing the loop body instructions
- **Looping condition** - the requirement that must be met for the computer to *continue* processing the loop body instructions
- **Multiline property** - a property of a **txt** - indicates whether the text box can accept and display either one or multiple lines of text
- **Overflow error** - occurs when the value assigned to a memory location is too large for the location's data type
- **Posttest loop** - a loop whose condition is evaluated *after* the instructions in its loop body are processed
- **Pretest loop** - a loop whose condition is evaluated *before* the instructions in its loop body are processed
- **ReadOnly property** - a property of a **txt** - specifies whether or not the user can change the contents of the text box
- **Real numbers** - numbers with a decimal place
- **Repetition structure** - the control structure used to repeatedly process one or more program instructions - also called a **loop**
- **ScrollBars property** - a property of a **txt** - specifies whether the text box has no scroll bars / a horizontal / a vertical / both horizontal and vertical

- **SelectedIndex property** - can be used to select an item in a **Ist** and also to determine which item is selected - stores the index of the selected item
- **SelectedIndexChanged event** - occurs when an item is selected in a **Ist**
- **SelectedItem property** - can be used to select an item in a **Ist** and also to determine which item is selected - stores the value of the selected item
- **SelectedValueChanged event** - occurs when an item is selected in a **Ist**
- **SelectionMode property** - determines the number of items that can be selected in a **Ist** - 0, 1, multiple
- **Sorted property** - specifies whether the **Ist** items should appear in the order they are entered in the **Ist** or in sorted order
- **String Collector Editor** - provides an easy way to add items to a **Ist**'s Items collection

|                                                                        |                                                                                                                                                                                                                                                                   |  |
|------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|
| CH5_Exercises                                                          | Chap05\Exercises                                                                                                                                                                                                                                                  |  |
| 16.Payment Solution-DoLoop_EXERCISE 1_introductory                     | - frmMain_Load, Do Until...Loop, lst.Items.Add, -Financial.Pmt, ControlChars, txt_Enter, txt_KeyPress, txt_TextChanged, lst_SelectedIndexChangedChanged                                                                                                           |  |
| 17.State Capital Solution_EXERCISE 2_introductory                      | - Private Sub lst_SelectedIndexChangedChanged, ElseIf, Select Case, lst.SelectedIndex                                                                                                                                                                             |  |
| 18.Modified 14.Savings Solution_EXERCISE 3_introductory                | - ControlChars.Tab, .NewLine, .Back, For...To...Step...Next, Do Until...Loop, txt_Enter, txt_TextChanged, txt_KeyPress                                                                                                                                            |  |
| 19.Modified 08.Projected Sales Solution_EXERCISE 4_introductory        | - Do While...AndAlso...Loop, txt_Enter, txt_KeyPress, txt_TextChanged                                                                                                                                                                                             |  |
| 20.Modified 09_Projected Sales Solution-DoLoop_EXERCISE 5_introductory | - Do Until...Loop, ControlChars.NewLine, .Back, e.KeyChar, txt_Enter, txt_KeyPress, txt_TextChanged                                                                                                                                                               |  |
| 21.Numbers Solution_EXERCISE 6_introductory                            | - For...To...Next, Do Until...Loop, Do...Loop Until, lst.Items.Add, txt_KeyPress, e.KeyChar, ControlChars.Back, txt_Enter, txt_TextChanged, lst.Items.Clear                                                                                                       |  |
| 22.Modified 21.Numbers Solution_EXERCISE 7_intermediate                | - For...To...Next, Do Until...Loop, Do...Loop Until, lst.Items.Add, e.KeyChar, ControlChars.Back, txt_Enter, txt_TextChanged, lst.Items.Clear                                                                                                                     |  |
| 23.Multiplication Solution_EXERCISE 8_intermediate                     | - Do Until...Loop, ControlChars.NewLine, ControlChars.Back, For...To...Next, txt_Enter, txt_TextChanged, txt_KeyPress, e.KeyChar                                                                                                                                  |  |
| 24.Discount Solution_EXERCISE 9_intermediate                           | - frmMain_Load, For...To...Step...Next, lst.SelectedIndex, txt_TextChanged, txt_KeyPress, ControlChars.Back, e.KeyChar, lst_SelectedIndexChangedChanged                                                                                                           |  |
| 25.Modified 13.Payment Solution_EXERCISE 10_intermediate               | - frmMain_Load, For...To...Step...Next, lst.Items.Add, lst.SelectedItem, -Financial.Pmt, txt_Enter, txt_KeyPress, e.KeyChar, ControlChars.Back, txt_TextChanged, lst_SelectedIndexChangedChanged, lst_SelectedValueChanged                                        |  |
| 26.Mills Solution_EXERCISE 11_advanced                                 | - frmMain_Load, For...To...Next, lst.Items.Add, Class-level accumulator                                                                                                                                                                                           |  |
| 27.GPA Solution_EXERCISE 12_advanced                                   | - frmMain_Load, For...To...Next, lst.Items.Add, lst.SelectedIndex, Select Case, rad.Checked                                                                                                                                                                       |  |
| 28.General Solution_EXERCISE 13_advanced                               | - ControlChars.NewLine, txt_Enter, txt_KeyPress, e.KeyChar, ControlChars.Back                                                                                                                                                                                     |  |
| 29.Canton Solution_EXERCISE 14_advanced                                | - frmMain_Load, For...To...Next, lst.Items.Add, lst.SelectedIndex, lst.Items.Clear, txt_Enter, ControlChars.Tab, ControlChars.NewLine, Financial.DDB, Financial.SYD, txt_TextChanged, txt_KeyPress, e.KeyChar, ControlChars.Back, lst_SelectedIndexChangedChanged |  |
| 30.Fibonacci Solution_EXERCISE 15_advanced                             | - Do...Loop Until                                                                                                                                                                                                                                                 |  |
| 31.Multi Solution_EXERCISE 16_advanced                                 | - lst Properties: SelectionMode = None/One/MultiSimple/MultiExtended; Sorted = True                                                                                                                                                                               |  |
|                                                                        | - frmMain_Load, lst.Items.Add, lst.SelectedItems.Count, For...To...Next, lst.SelectedItems.Item(), ControlChars.NewLine                                                                                                                                           |  |
| 32.Items Collection Solution_EXERCISE 17_advanced                      | - frmMain_Load, lst.Items.Add, lst.Items.Count, lst.Items.Insert, lst.Items.Remove, lst.Items.RemoveAt                                                                                                                                                            |  |
| 33.Savings Solution 14-Advanced_EXERCISE 18_advanced                   | - ControlChars.Tab, ControlChars.NewLine, For...To...Step...Next, txt_Enter, txt_TextChanged, txt_KeyPress, e.KeyChar, ControlChars.Back                                                                                                                          |  |
| 34.Salary Solution_EXERCISE 19_advanced                                | - txt Properties: MaxLength = 5, For...To...Step...Next, ToString("F2"), txt_TextChanged, ControlChars.Tab, .NewLine, .Back, txt_Enter, txt_KeyPress, e.KeyChar                                                                                                   |  |
| 35.OnYourOwn Solution_EXERCISE 20                                      | - frmMain_Load, lst.Items.Add, lst.SelectedIndex, ElseIf, Do...Loop Until, txt_TextChanged, txt_Enter, txt_KeyPress, e.KeyChar, ControlChars.Back, .NewLine, lst_SelectedIndexChangedChanged                                                                      |  |
| 36.FixIt Solution_EXERCISE 21                                          | - Do Until...Loop, ControlChars.NewLine                                                                                                                                                                                                                           |  |

## Chap05\\_Exercise1

### 16.Payment Solution-DoLoop\_EXERCISE 1\_introductory

- frmMain\_Load, Do Until...Loop, lst.Items.Add, -Financial.Pmt, ControlChars, txt\_Enter, txt\_KeyPress, txt\_TextChanged, lst\_SelectedIndexChanged

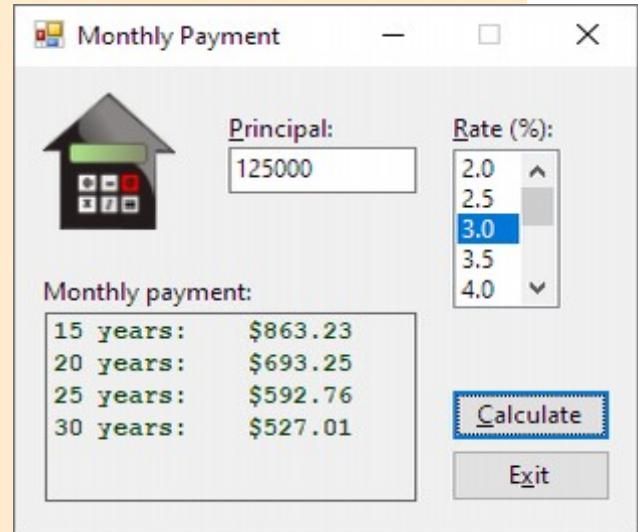
- In this exercise, you modify the Monthly Payment application from this chapter's Apply lesson. Use Windows to make a copy of the Payment Solution folder. Rename the copy Payment Solution-DoLoop. Open the Payment.sln file contained in the Payment Solution-DoLoop folder. Change the For...Next statements in the frmMain\_Load and btnCalc\_Click procedures to Do...Loop statements. Save the solution and then start and test the application. (If you test the application using 125000 as the principal and 3.0 as the rate, the payments should be identical to those shown earlier in Figure 5-37.)

```
1  ' Name:      Payment Project - DoLoop - Exercise 1.
2  ' Purpose:    Display the monthly mortgage payments for terms of 15, 20, 25, and 30 years.
3  ' Programmer: <your name> on <current date>
4
5  Option Explicit On
6  Option Strict On
7  Option Infer Off
8
9  Public Class frmMain
10
11     Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
12         ' Fill list box with rates and select 3.0 rate.
13
14         ' Exercise 1: Change the For...Next statement to Do...Loop statement:
15         'For dblRates As Double = 2 To 7 Step 0.5
16         'lstRates.Items.Add(dblRates.ToString("N1"))
17         'Next dblRates
18
19         Dim dblRates As Double = 2D
20         Do Until dblRates > 7D
21             lstRates.Items.Add(dblRates.ToString("N1"))
22             dblRates += 0.5D
23         Loop
24
25         lstRates.SelectedItem = "3.0"
26     End Sub
27
28     Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
29         ' Display the monthly mortgage payment.
30         Dim intPrincipal As Integer
31         Dim dblRate As Double
32         Dim dblPay As Double
33         Integer.TryParse(txtPrincipal.Text, intPrincipal)
```

```

34     Double.TryParse(lstRates.SelectedItem.ToString(), dblRate)
35     dblRate /= 100
36     lblPay.Text = String.Empty
37
38     ' Exercise 1: Change the For...Next statement to Do...Loop statement:
39     'For intTerm As Integer = 15 To 30 Step 5
40     'dblPay = -Financial.Pmt(dblRate / 12,
41     'intTerm * 12, intPrincipal)
42     'lblPay.Text = lblPay.Text & intTerm.ToString() &
43     '    " years: " & dblPay.ToString("C2") &
44     'ControlChars.NewLine
45     'Next intTerm
46
47     Dim intTerm As Integer = 15
48     Do Until intTerm > 30
49         dblPay = -Financial.Pmt(dblRate / 12, intTerm * 12, intPrincipal)
50         lblPay.Text = lblPay.Text & intTerm.ToString() & " years: " & dblPay.ToString("C2") & ControlChars.NewLine
51         intTerm += 5
52     Loop
53 End Sub
54
55 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
56     Me.Close()
57 End Sub
58
59 Private Sub txtPrincipal_Enter(sender As Object, e As EventArgs) Handles txtPrincipal.Enter
60     txtPrincipal.SelectAll()
61 End Sub
62
63 Private Sub txtPrincipal_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtPrincipal.KeyPress
64     ' Accept only numbers and the Backspace key.
65     If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
66         e.Handled = True
67     End If
68 End Sub
69
70 Private Sub txtPrincipal_TextChanged(sender As Object, e As EventArgs) Handles txtPrincipal.TextChanged
71     lblPay.Text = String.Empty
72 End Sub
73
74 Private Sub lstRates_SelectedIndexChanged(sender As Object, e As EventArgs) Handles lstRates.SelectedIndexChanged
75     lblPay.Text = String.Empty
76 End Sub
77 End Class

```

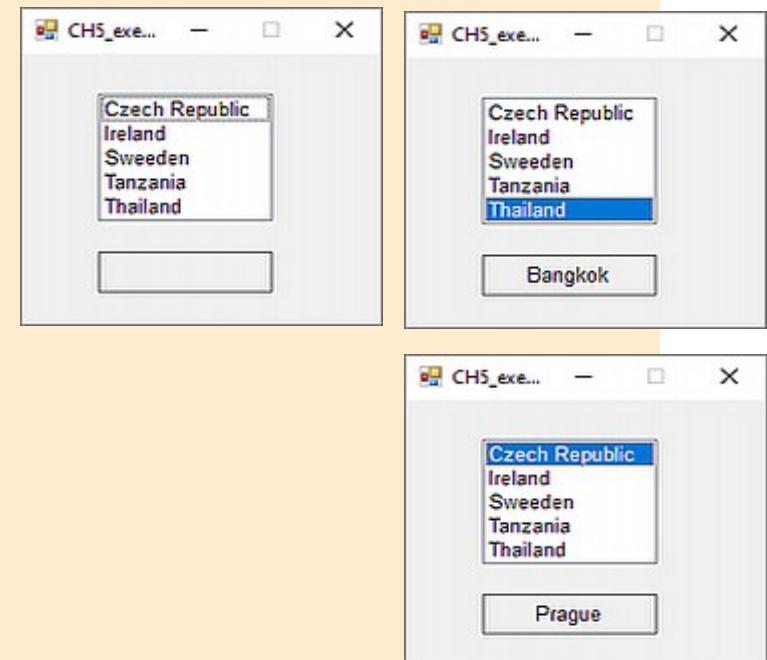


2. Create a Windows Form application. Use the following names for the project and solution, respectively: State Capital Project and State Capital Solution. Save the application in the VB2017\Chap05 folder. Add any five state names to a list box. When the user clicks a name in the list box, the list box's SelectedIndexChanged procedure should display the name of the state's capital in a label control. Save the solution and then start and test the application.

```

1  'add 5 States in a lst. each items SelectedIndexChanged should present its Capitals:
2  Option Explicit On
3  Option Strict On
4  Option Infer Off
5
6  Public Class frmMain
7      Private Sub lstStates_SelectedIndexChanged(sender As Object, e As EventArgs) Handles lstStates.SelectedIndexChanged
8          'If lstStates.SelectedIndex = 0 Then
9          '    lblCapital.Text = "Prague"
10         'ElseIf lstStates.SelectedIndex = 1 Then
11         '    lblCapital.Text = "Dublin"
12         'ElseIf lstStates.SelectedIndex = 2 Then
13         '    lblCapital.Text = "Stockholm"
14         'ElseIf lstStates.SelectedIndex = 3 Then
15         '    lblCapital.Text = "Dar Es Salaam"
16         'ElseIf lstStates.SelectedIndex = 4 Then
17         '    lblCapital.Text = "Bangkok"
18         'End If
19         ' ok, its working. i will try another code
20
21         Select Case lstStates.SelectedIndex
22             Case 0
23                 lblCapital.Text = "Prague"
24             Case 1
25                 lblCapital.Text = "Dublin"
26             Case 2
27                 lblCapital.Text = "Stockholm"
28             Case 3
29                 lblCapital.Text = "Dar Es Salaam"
30             Case 4
31                 lblCapital.Text = "Bangkok"
32         End Select
33         ' ok, its wotking too, juchuuuuuu
34     End Sub
35 End Class

```



3. In this exercise, you modify the Savings Account application from this chapter's Apply lesson. Use Windows to make a copy of the Savings Solution folder. Rename the copy Modified Savings Solution. Open the Savings Solution.sln file contained in the Modified Savings Solution folder. In the btnCalc\_Click procedure, change the For...Next statement that controls the years to a Do...Loop statement. Save the solution and then start and test the application. (If you test the application using 2500 as the deposit, the account balances should be identical to those shown earlier in Figure 5-41. Verify that the txtBalance control contains the 7% rate information.)

```

1  ' Name:      Modified Savings Project - CH5 exercise 3: change For...Next statement that controls the
2  '          years to a Do...Loop statement
3  ' Purpose:    Display a savings account balance for each of 5 years using rates
4  '          from 3% to 7% in increments of 1%.
5  Option Explicit On
6  Option Strict On
7  Option Infer Off
8
9  Public Class frmMain
10     Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
11         ' Calculate account balances for each of five years
12         ' using rates from 3% to 7% in increments of 1%.
13
14         Dim dblDeposit As Double
15         Dim dblBalance As Double
16         Double.TryParse(txtDeposit.Text, dblDeposit)
17
18         txtBalance.Text = "Rate" & ControlChars.Tab & "Year" & ControlChars.Tab & "Balance" &
19                         ControlChars.NewLine
20
21         ' Calculate and display account balances.
22         For dblRate As Double = 0.03 To 0.07 Step 0.01
23             txtBalance.Text = txtBalance.Text & dblRate.ToString("P0") & ControlChars.NewLine
24
25             ' CH5 Exercise 3: change For...Next statement that controls the years to a Do...Loop statement:
26             'For intYear As Integer = 1 To 5
27             'dblBalance = dblDeposit * (1 + dblRate) ^ intYear
28             'txtBalance.Text = txtBalance.Text &
29             'ControlChars.Tab & intYear.ToString &
30             'ControlChars.Tab & dblBalance.ToString("C2") &
31             'ControlChars.NewLine
32             'Next intYear

```

```

33
34     Dim intYear As Integer = 1
35     Do Until intYear > 5
36         dblBalance = dblDeposit * (1 + dblRate) ^ intYear
37         txtBalance.Text = txtBalance.Text & ControlChars.Tab & intYear.ToString & ControlChars.Tab &
38                         dblBalance.ToString("C2") & ControlChars.NewLine
39         intYear += 1
40     Loop
41     'juchuuuu it works
42     Next dblRate
43 End Sub
44
45 Private Sub txtDeposit_Enter(sender As Object, e As EventArgs) Handles txtDeposit.Enter
46     txtDeposit.SelectAll()
47 End Sub
48
49 Private Sub txtDeposit_TextChanged(sender As Object, e As EventArgs) Handles txtDeposit.TextChanged
50     txtBalance.Text = String.Empty
51 End Sub
52
53 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
54     Me.Close()
55 End Sub
56
57 Private Sub txtDeposit_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtDeposit.KeyPress
58     ' Allows the text box to accept only numbers, the period, and the Backspace key.
59     If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> "." AndAlso e.KeyChar <> ControlChars.Back Then
60         e.Handled = True
61     End If
62 End Sub
63 End Class

```

| Rate | Year | Balance    |
|------|------|------------|
| 3%   | 1    | \$2,575.00 |
|      | 2    | \$2,652.25 |
|      | 3    | \$2,731.82 |
|      | 4    | \$2,813.77 |
|      | 5    | \$2,898.19 |
| 4%   | 1    | \$2,600.00 |
|      | 2    | \$2,704.00 |
|      | 3    | \$2,812.16 |
|      | 4    | \$2,924.65 |
|      | 5    | \$3,041.63 |

| Rate | Year | Balance    |
|------|------|------------|
| 6%   | 1    | \$2,650.00 |
|      | 2    | \$2,809.00 |
|      | 3    | \$2,977.54 |
|      | 4    | \$3,156.19 |
|      | 5    | \$3,345.56 |
| 7%   | 1    | \$2,675.00 |
|      | 2    | \$2,862.25 |
|      | 3    | \$3,062.61 |
|      | 4    | \$3,276.99 |
|      | 5    | \$3,506.38 |

4. In this exercise, you modify one of the Projected Sales applications from this chapter's Apply lesson. Use Windows to make a copy of the Projected Sales Solution folder. Rename the copy Modified Projected Sales Solution. Open the Projected Sales Solution.sln file contained in the Modified Projected Sales Solution folder. Rather than using \$150,000 as the sales goal, the user should be able to enter any sales goal. Modify the interface (using a text box for entering the sales goal) and code as needed. Be sure to reset the tab order. Also be sure to code the new text box's KeyPress, Enter, and TextChanged event procedures. Save the solution and then start and test the application. (If the current sales and sales goal are 50000 and 125000, respectively, it will take the company 31 years to reach \$125,004 in sales.)

```

1  'Modified Projected Sales Project:
2  ' - rather than using $150,000 as the sales goal, the user should be able to enter any Sales goal
3  ' - modify the interface (using a text box for entering the sales goal) and code as needed
4  ' - be sure to reset the tab order
5  ' - also be sure to code the new text box's: KeyPress, Enter, TextChanged event procedures
6  'test: if the current sales is 50000, sales goal is 125000, it will take company 31 years to reach $125,004 in sales.
7  ' Purpose:      Display the number of years required for a company's projected sales
8  '                  to reach a specific amount and the projected sales at that time.
9  Option Explicit On
10 Option Strict On
11 Option Infer Off
12
13 Public Class frmMain
14     Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
15         ' Calculate and display the number of years
16         ' and the projected sales.
17         Const dblGROWTH_RATE As Double = 0.03
18         Dim dblSales As Double ' Used as an accumulator.
19         Dim dblGoal As Double ' _mod_Sales Goal entered by the user, upper limit value
20         Dim dblIncrease As Double
21         Dim intYears As Integer ' Used as a counter.
22         Double.TryParse(txtCurrentSales.Text, dblSales)
23         Double.TryParse(txtSalesGoal.Text, dblGoal) ' _mod_
24
25         'Do While dblSales > 0 AndAlso dblSales < 150000 _original
26         Do While dblSales > 0 AndAlso dblSales < dblGoal 'juchuuuu its wotking
27             dblIncrease = dblSales * dblGROWTH_RATE
28             dblSales += dblIncrease
29             intYears += 1
30         Loop
31
32         lblProjSales.Text = "Projected sales " & intYears.ToString & " years from now: " & dblSales.ToString("C0")

```

```

33     End Sub

34

35     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
36         Me.Close()
37     End Sub

38

39     Private Sub txtCurrentSales_Enter(sender As Object, e As EventArgs) Handles txtCurrentSales.Enter
40         txtCurrentSales.SelectAll()
41     End Sub

42

43     Private Sub txtCurrentSales_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtCurrentSales.KeyPress
44         ' Accept only numbers and the Backspace key.
45         If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
46             e.Handled = True
47         End If
48     End Sub

49

50     Private Sub txtCurrentSales_TextChanged(sender As Object, e As EventArgs) Handles txtCurrentSales.TextChanged
51         lblProjSales.Text = String.Empty
52     End Sub

53

54     Private Sub txtSalesGoal_Enter(sender As Object, e As EventArgs) Handles txtSalesGoal.Enter
55         txtSalesGoal.SelectAll()
56     End Sub

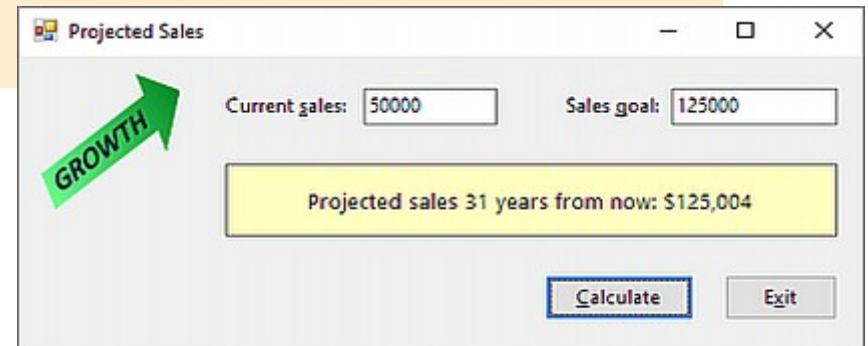
57

58     Private Sub txtSalesGoal_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtSalesGoal.KeyPress
59         'accept only numbers and the Backspace key:
60         If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
61             e.Handled = True
62         End If
63     End Sub

64

65     Private Sub txtSalesGoal_TextChanged(sender As Object, e As EventArgs) Handles txtSalesGoal.TextChanged
66         lblProjSales.Text = String.Empty
67     End Sub
68 End Class

```



- Do Until...Loop, ControlChars.NewLine, .Back, e.KeyChar,  
txt\_Enter, txt\_KeyPress, txt\_TextChanged

In this exercise, you modify one of the Projected Sales applications from this chapter's Apply lesson. Use Windows to make a copy of the Projected Sales Solution-ForNext folder. Rename the copy Projected Sales Solution-DoLoop. Open the Projected Sales

Solution.sln file contained in the Projected Sales Solution-DoLoop folder. Change the For...Next statement in the btnCalc\_Click procedure to a Do...Loop statement. Save the solution and then start and test the application. (If the current sales amount is 86500, the projected sales amounts should be identical to those shown earlier in Figure 5-22.)

```

1  ' Name:      Projected Sales Project
2  ' Purpose:   Display the number of years required for a company's projected sales
3  '
4  '           to reach a specific amount and the projected sales at that time.
5  ' Programmer: <your name> on <current date>
6  'Modified : in the btnCalc_Click change the For...Next statement to a Do...Loop statement
7  'test: if the current sales amount is 86500, the projected s.a. will be same as Figure 5-22
8  Option Explicit On
9  Option Strict On
10 Option Infer Off
11
12 Public Class frmMain
13     Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
14         ' Calculate and display the number of years and the projected sales.
15         Const dblGROWTH_RATE As Double = 0.03
16         Dim dblSales As Double ' Used as an accumulator.
17         Dim dblIncrease As Double
18         Dim intYears As Integer = 2019 '_mod_
19         Double.TryParse(txtCurrentSales.Text, dblSales)
20
21         Do Until intYears > 2022 '_mod_:
22             dblIncrease = dblSales * dblGROWTH_RATE
23             dblSales += dblIncrease
24             lblProjSales.Text = lblProjSales.Text & intYears.ToString & "          " & dblSales.ToString("C0") &
25                         ControlChars.NewLine
26             intYears += 1
27             'juchuuuu it works
28             Loop
29             'For intYears As Integer = 2019 To 2022
30             'dblIncrease = dblSales * dblGROWTH_RATE
31             'dblSales += dblIncrease
32             'lblProjSales.Text = lblProjSales.Text & intYears.ToString & "          " & dblSales.ToString("C0") &
33                         ControlChars.NewLine
34             'Next intYears
35     End Sub

```

```

35
36     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
37         Me.Close()
38     End Sub
39
40     Private Sub txtCurrentSales_Enter(sender As Object, e As EventArgs) Handles txtCurrentSales.Enter
41         txtCurrentSales.SelectAll()
42     End Sub
43
44     Private Sub txtCurrentSales_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtCurrentSales.KeyPress
45         ' Accept only numbers and the Backspace key.
46         If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
47             e.Handled = True
48         End If
49     End Sub
50
51     Private Sub txtCurrentSales_TextChanged(sender As Object, e As EventArgs) Handles txtCurrentSales.TextChanged
52         lblProjSales.Text = String.Empty
53     End Sub
54 End Class

```

| Year | Projected sales |
| --- | --- |
| 2019 | \$89,095 |
| 2020 | \$91,768 |
| 2021 | \$94,521 |
| 2022 | \$97,357 |

Open the Numbers Solution.sln file contained in the VB2017\Chap05\Numbers Solution folder. The application allows the user to display a list of numbers in a list box. The user will enter the start and end values for these numbers in the two text boxes.

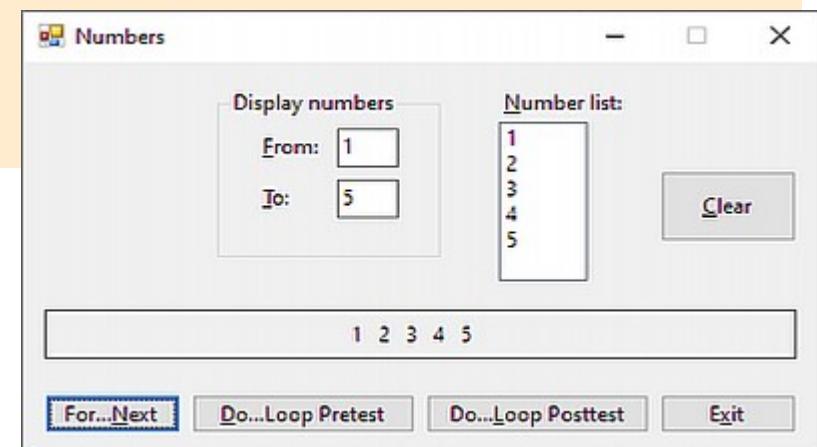
- a. Open the Code Editor window. Locate the `btnForNext_Click` procedure and then click the blank line above its `End Sub` clause. Use the `For...Next` statement to display the range of numbers indicated by the user. For example, if the user enters the numbers 1 and 5 in the From and To boxes, respectively, the statement should display the numbers 1 through 5 in the list box. Display the numbers only when the value in the From box is less than the value in the To box; otherwise, leave the list box empty. Save the solution and then start and test the procedure.
  - b. Now, locate the `btnDoPretest_Click` procedure and then click the blank line above its `End Sub` clause. Use a pretest `Do...Loop` statement to display the range of numbers indicated by the user. Display the numbers only when the value in the From box is less than the value in the To box; otherwise, leave the list box empty. Save the solution and then start and test the procedure.
  - c. Finally, locate the `btnDoPosttest_Click` procedure and then click the blank line above its `End Sub` clause. Use a posttest `Do...Loop` statement to display the range of numbers indicated by the user. Display the numbers only when the value in the From box is less than the value in the To box; otherwise, leave the list box empty. Save the solution and then start and test the procedure.

```
25  Private Sub btnDoPretest_Click(sender As Object, e As EventArgs) Handles btnDoPretest.Click
26      ' Display a list of numbers.
27      Dim intFrom As Integer
28      Dim intTo As Integer
29      Integer.TryParse(txtFrom.Text, intFrom)
30      Integer.TryParse(txtTo.Text, intTo)
31      lstNumbers.Items.Clear()
32      lblNumbers.Text = String.Empty                                'mycode
33
34      Do Until intFrom > intTo                                    'mycode
35          lstNumbers.Items.Add(intFrom)                            'mycode
36          lblNumbers.Text = lblNumbers.Text & intFrom.ToString & "    "   'mycode
37          intFrom += 1   'mycode
38      Loop   'mycode
39  End Sub
40
41  Private Sub btnDoPosttest_Click(sender As Object, e As EventArgs) Handles btnDoPosttest.Click
42      ' Display a list of numbers.
43      Dim intFrom As Integer
44      Dim intTo As Integer
45      Integer.TryParse(txtFrom.Text, intFrom)
46      Integer.TryParse(txtTo.Text, intTo)
47      lstNumbers.Items.Clear()
48      lblNumbers.Text = String.Empty                                'mycode
49
50      If intTo > intFrom Then                                     'mycode
51          Do  'mycode
52              lstNumbers.Items.Add(intFrom)
53              lblNumbers.Text = lblNumbers.Text & intFrom.ToString & "    "   'mycode
54              intFrom += 1   'mycode
55          Loop Until intFrom > intTo                           'mycode
56      End If  'mycode
57  End Sub
58
59  Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
60      Me.Close()
61  End Sub
62
63  Private Sub txtFrom_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtFrom.KeyPress
64      ' Accept only numbers and the Backspace key.
65      If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
66          e.Handled = True
67      End If
68  End Sub
```

```

69
70     Private Sub txtTo_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtTo.KeyPress
71         ' Accept only numbers and the Backspace key.
72
73         If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
74             e.Handled = True
75         End If
76     End Sub
77
78     Private Sub txtFrom_Enter(sender As Object, e As EventArgs) Handles txtFrom.Enter
79         txtFrom.SelectAll()
80     End Sub
81
82     Private Sub txtTo_Enter(sender As Object, e As EventArgs) Handles txtTo.Enter
83         txtTo.SelectAll()
84     End Sub
85
86     Private Sub txtFrom_TextChanged(sender As Object, e As EventArgs) Handles txtFrom.TextChanged
87         lstNumbers.Items.Clear()
88         lblNumbers.Text = String.Empty
89     End Sub
90
91     Private Sub txtTo_TextChanged(sender As Object, e As EventArgs) Handles txtTo.TextChanged
92         lstNumbers.Items.Clear()
93         lblNumbers.Text = String.Empty
94     End Sub
95
96     'my extra button btnClear:
97     Private Sub btnClear_Click(sender As Object, e As EventArgs) Handles btnClear.Click
98         txtFrom.Text = String.Empty
99         txtTo.Text = String.Empty
100        lstNumbers.Items.Clear()
101        lblNumbers.Text = String.Empty
102    End Sub
103 End Class

```



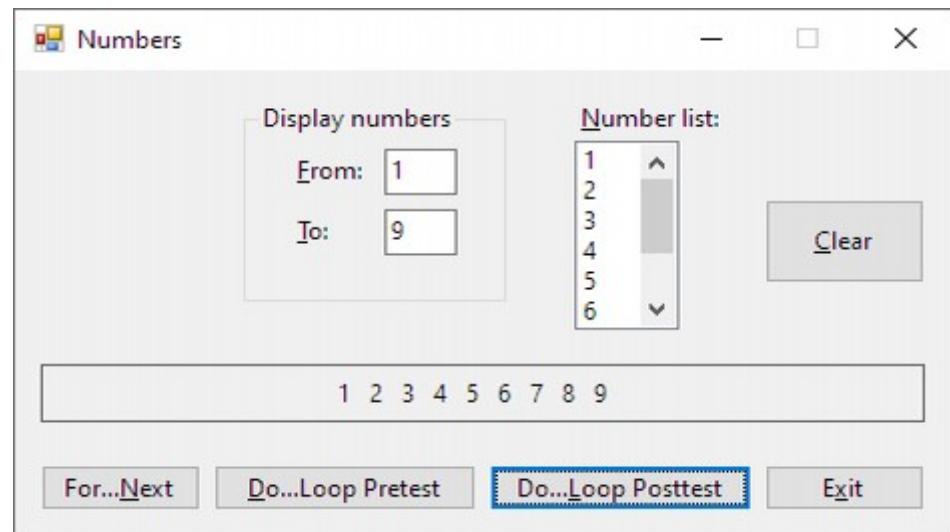
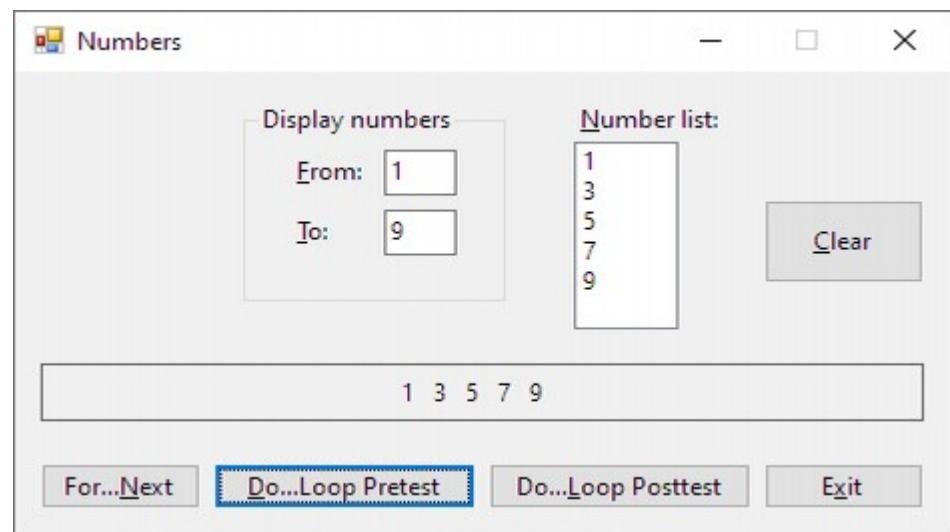
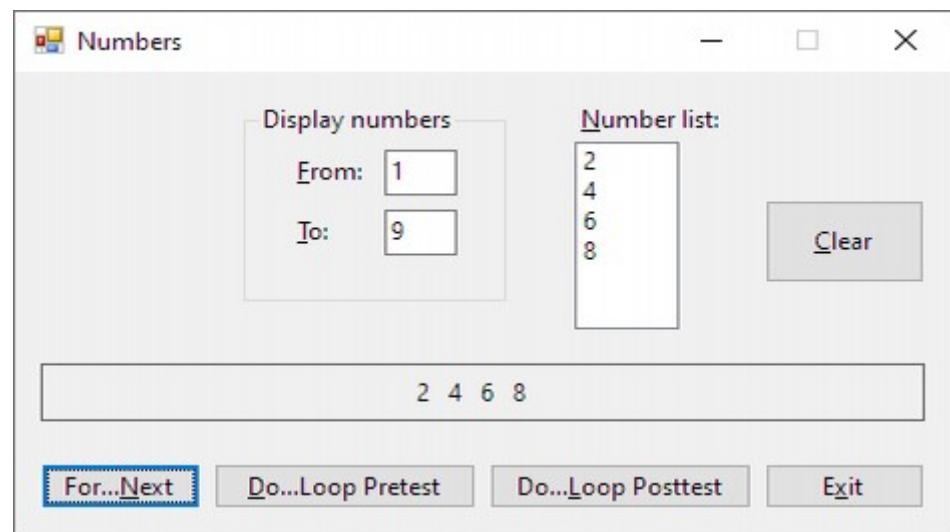
- **For...To...Next**, **Do Until...Loop**, **Do...Loop Until**, **lst.Items.Add**,  
**e.KeyChar**, **ControlChars.Back**, **txt\_Enter**, **txt\_TextChanged**, **lst.Items.Clear**  
**txt\_KeyPress**, **Mod**

In this exercise, you modify the Numbers application from Exercise 6. Use Windows to make a copy of the Numbers Solution folder. Rename the copy Modified Numbers Solution. Open the Numbers Solution.sln file contained in the Modified Numbers Solution folder.

- a. Open the Code Editor window. Modify the `btnForNext_Click` procedure to display only the even numbers within the range specified by the user. (If necessary, review the arithmetic operators listed in Figure 3-10 in Chapter 3.) Save the solution and then start and test the procedure. (If the From and To boxes contain 1 and 9, respectively, the procedure should display the numbers 2, 4, 6, and 8.)
  - b. Now, modify the `btnDoPretest_Click` procedure to display only the odd numbers within the range specified by the user. Save the solution and then start and test the procedure. (If the From and To boxes contain 1 and 9, respectively, the procedure should display the numbers 1, 3, 5, 7, and 9.)

```
28  Private Sub btnDoPretest_Click(sender As Object, e As EventArgs) Handles btnDoPretest.Click
29      '2. Display a list of numbers - only the ODD numbers
30      Dim intFrom As Integer
31      Dim intTo As Integer
32      Integer.TryParse(txtFrom.Text, intFrom)
33      Integer.TryParse(txtTo.Text, intTo)
34      lstNumbers.Items.Clear()
35      lblNumbers.Text = String.Empty                                'mycode
36
37      Do Until intFrom > intTo                                    'mycode
38          If intFrom Mod 2 = 1 Then                                'mycode
39              lstNumbers.Items.Add(intFrom)                         'mycode
40              lblNumbers.Text = lblNumbers.Text & intFrom.ToString & "    "   'mycode
41          End If   'mycode
42          intFrom += 1   'mycode
43      Loop   'mycode
44  End Sub
45
46  Private Sub btnDoPosttest_Click(sender As Object, e As EventArgs) Handles btnDoPosttest.Click
47      ' Display a list of numbers.
48      Dim intFrom As Integer
49      Dim intTo As Integer
50      Integer.TryParse(txtFrom.Text, intFrom)
51      Integer.TryParse(txtTo.Text, intTo)
52      lstNumbers.Items.Clear()
53      lblNumbers.Text = String.Empty                                'mycode
54
55      If intTo > intFrom Then                                    'mycode
56          Do   'mycode
57              lstNumbers.Items.Add(intFrom)
58              lblNumbers.Text = lblNumbers.Text & intFrom.ToString & "    "   'mycode
59              intFrom += 1   'mycode
60          Loop Until intFrom > intTo                           'mycode
61      End If   'mycode
62  End Sub
63
64  Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
65      Me.Close()
66  End Sub
67
```

```
68  Private Sub txtFrom_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtFrom.KeyPress
69      ' Accept only numbers and the Backspace key.
70      If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
71          e.Handled = True
72      End If
73  End Sub
74
75  Private Sub txtTo_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtTo.KeyPress
76      ' Accept only numbers and the Backspace key.
77      If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
78          e.Handled = True
79      End If
80  End Sub
81
82  Private Sub txtFrom_Enter(sender As Object, e As EventArgs) Handles txtFrom.Enter
83      txtFrom.SelectAll()
84  End Sub
85
86  Private Sub txtTo_Enter(sender As Object, e As EventArgs) Handles txtTo.Enter
87      txtTo.SelectAll()
88  End Sub
89
90  Private Sub txtFrom_TextChanged(sender As Object, e As EventArgs) Handles txtFrom.TextChanged
91      lstNumbers.Items.Clear()
92      lblNumbers.Text = String.Empty
93  End Sub
94
95  Private Sub txtTo_TextChanged(sender As Object, e As EventArgs) Handles txtTo.TextChanged
96      lstNumbers.Items.Clear()
97      lblNumbers.Text = String.Empty
98  End Sub
99
100 'my extra button btnClear:
101 Private Sub btnClear_Click(sender As Object, e As EventArgs) Handles btnClear.Click
102     txtFrom.Text = String.Empty
103     txtTo.Text = String.Empty
104     lstNumbers.Items.Clear()
105     lblNumbers.Text = String.Empty
106 End Sub
107 End Class
```



Open the Multiplication Solution.sln file contained in the VB2017\Chap05\Multiplication Solution folder. Code the application to display a multiplication table similar to the one shown in Figure 5-47. Use the For...Next statement in the btnForNext\_Click procedure, and use the Do...Loop statement in the btnDoLoop\_Click procedure. Save the solution and then start and test the application.

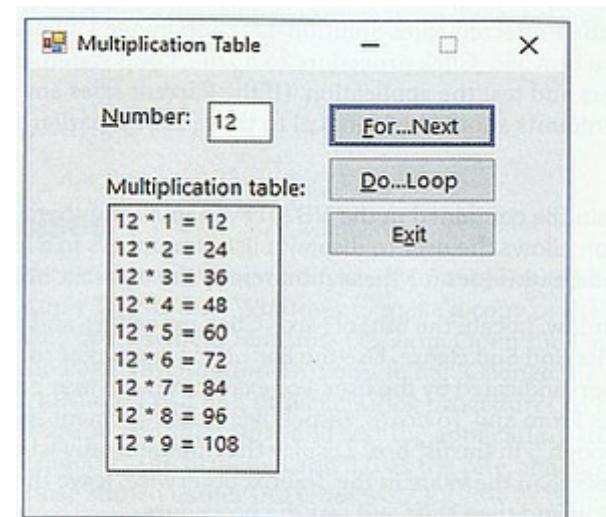


Figure 5-47 Sample multiplication table for Exercise 8

```

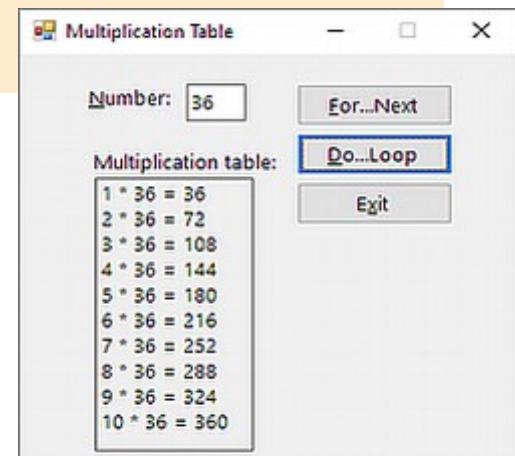
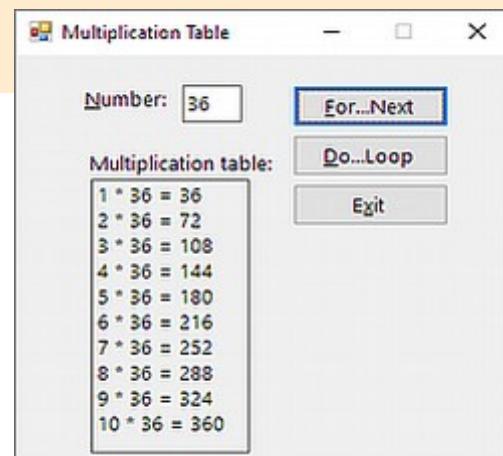
1  ' Name:      Multiplication Project
2  ' Purpose:    Displays a multiplication table that shows the multiplicand, multiplier, and product.
3  ' Programmer: <your name> on <current date>
4  ' Notes: i think this Exercise should show the benefit of the Do....Loop in this example,
5  '         rather than For...Next what seems more complicated
6  Option Explicit On
7  Option Strict On
8  Option Infer Off
9
10 Public Class frmMain
11     Private Sub btnDoLoop_Click(sender As Object, e As EventArgs) Handles btnDoLoop.Click
12         ' Displays a multiplication table. txtNumber, lblTable
13         Dim intNumber As Integer                      ' mycode
14         Dim intCount As Integer = 1                   ' mycode
15         Integer.TryParse(txtNumber.Text, intNumber)   ' mycode
16         lblTable.Text = String.Empty                  ' mycode_cistirna for repeated click
17
18         Do Until intCount > 10                      ' loop for counter 1 till 10
19             lblTable.Text = lblTable.Text & intCount.ToString & " * " & intNumber & " = " & (intNumber * intCount) &
20                 ControlChars.NewLine
21             intCount += 1                             ' mycode_loop for counter !!!!
22         Loop
23     End Sub

```

```

24
25     Private Sub btnForNext_Click(sender As Object, e As EventArgs) Handles btnForNext.Click
26         ' Displays a multiplication table. txtNumber, lblTable
27         Dim intNumber As Integer   ' mycode
28         Integer.TryParse(txtNumber.Text, intNumber)                      ' mycode
29         lblTable.Text = String.Empty                                     ' mycode
30
31         For intCount As Integer = 1 To 10                                ' Counter 1 till 10
32             lblTable.Text = lblTable.Text & intCount.ToString & " * " & intNumber & " = " & (intNumber * intCount) &
33                         ControlChars.NewLine
34         Next intCount   ' Counter 1 till 10
35     End Sub
36
37     Private Sub txtNumber_Enter(sender As Object, e As EventArgs) Handles txtNumber.Enter
38         txtNumber.SelectAll()
39     End Sub
40
41     Private Sub txtNumber_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtNumber.KeyPress
42         ' Allow only numbers and the Backspace key.
43         If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
44             e.Handled = True
45         End If
46     End Sub
47
48     Private Sub txtNumber_TextChanged(sender As Object, e As EventArgs) Handles txtNumber.TextChanged
49         lblTable.Text = String.Empty
50     End Sub
51
52     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
53         Me.Close()
54     End Sub
55 End Class

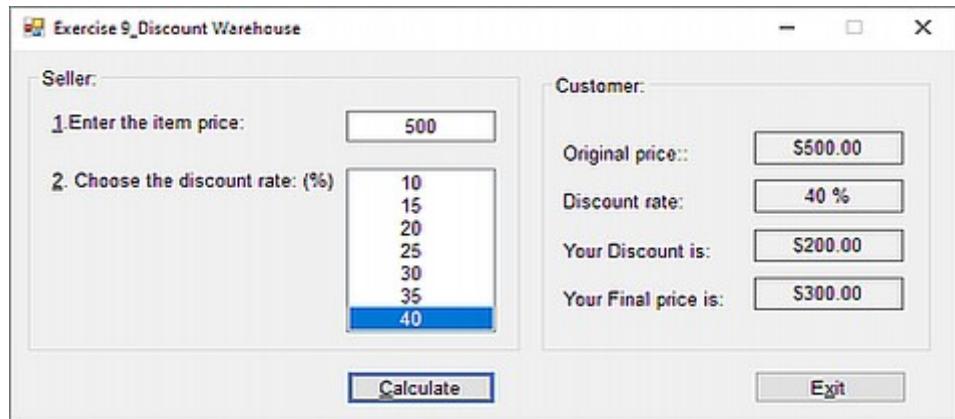
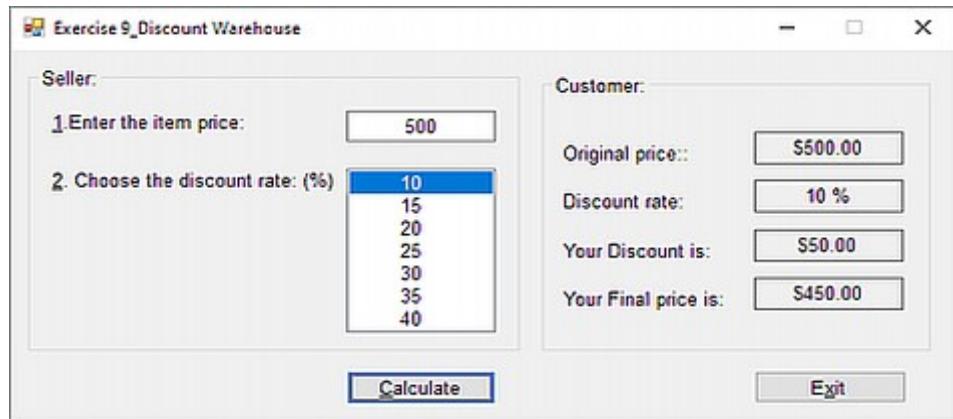
```



9. In this exercise, you create an application for Discount Warehouse. Create a Windows Forms application. Use the following names for the project and solution, respectively: Discount Project and Discount Solution. Save the application in the VB2017\Chap05 folder. The interface should allow the user to enter an item's original price and its discount rate. The discount rates should range from 10% through 40% in increments of 5%. Use a text box for entering the original price, and use a list box for entering the discount rates. The application should display the amount of the discount and also the discounted price. The button that calculates and displays the discount and discounted price should be the default button. Save the solution and then start and test the application.

```
1  'txtPrice = original price entered by the user
2  '      decPrice As Decimal
3  'lstRate = discount rate 10% till 40% step 5%: user can choose only 1
4  'btnCalculate = default button
5  'lbl1Price = original price entered by the user - ToString("C2")
6  'lbl2Rate = discount rate - selected item in lstRate
7  'lbl3Discount = shows the amount of the discount: original price * % choosed in lst
8  'lbl4NewPrice = discounted price: original price -(original price * %)
9  'btnCalculate = default button
10 Option Explicit On
11 Option Strict On
12 Option Infer Off
13 Public Class frmMain
14     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
15         Me.Close()
16     End Sub
17
18     Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
19         For dblRates As Double = 10 To 40 Step 5
20             lstRate.Items.Add("      " & dblRates.ToString("N0")) '!!! add loop values into lst !!!
21         Next dblRates
22         lstRate.SelectedIndex = 3
23     End Sub
24
```

```
25  Private Sub btnCalculate_Click(sender As Object, e As EventArgs) Handles btnCalculate.Click
26      Dim dbl1Price As Double
27      Dim dbl3Discount As Double
28      Dim dbl4NewPrice As Double
29      Dim dblRate As Double
30      Double.TryParse(txtPrice.Text, dbl1Price)
31      Double.TryParse(lstRate.SelectedItem.ToString, dblRate) '!!! transform SelectedItem into dblRate !!!
32
33      'do the math:
34      dbl4NewPrice = dbl1Price - (dbl1Price * (dblRate / 100))
35      dbl3Discount = dbl1Price - dbl4NewPrice
36
37      lbl1Price.Text = dbl1Price.ToString("C2")
38      lbl2Rate.Text = dblRate.ToString("N0") & " %"
39      lbl3Discount.Text = dbl3Discount.ToString("C2")
40      lbl4NewPrice.Text = dbl4NewPrice.ToString("C2")
41  End Sub
42
43  Private Sub txtPrice_Enter(sender As Object, e As EventArgs) Handles txtPrice.Enter
44      txtPrice.SelectAll()
45  End Sub
46
47  Private Sub txtPrice_TextChanged(sender As Object, e As EventArgs) Handles txtPrice.TextChanged
48      'clear all values:
49      lbl1Price.Text = String.Empty
50      lbl2Rate.Text = String.Empty
51      lbl3Discount.Text = String.Empty
52      lbl4NewPrice.Text = String.Empty
53  End Sub
54
55  Private Sub txtPrice_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtPrice.KeyPress
56      If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back AndAlso e.KeyChar <> "." Then
57          e.Handled = True
58      End If
59  End Sub
60
61  Private Sub lstRate_SelectedIndexChanged(sender As Object, e As EventArgs) Handles lstRate.SelectedIndexChanged
62      'clear all values:
63      lbl1Price.Text = String.Empty
64      lbl2Rate.Text = String.Empty
65      lbl3Discount.Text = String.Empty
66      lbl4NewPrice.Text = String.Empty
67  End Sub
68 End Class
```



## Chap05\Exercise1

### 25.Modified 13.Payment Solution\_EXERCISE 10\_intermediate

```
- frmMain_Load, For...To...Step...Next, lst.Items.Add, lst.SelectedItem,
- Financial.Pmt, txt_Enter, txt_KeyPress, e.KeyChar, ControlChars.Back,
txt_TextChanged, lst_SelectedIndexChanged, lst_SelectedValueChanged
```

10. In this exercise, you modify the Monthly Payment application from this chapter's Apply lesson. Use Windows to make a copy of the Payment Solution folder. Rename the copy Modified Payment Solution. Open the Payment Solution.sln file contained in the Modified Payment Solution folder. Modify the interface to allow the user to select the term (15, 20, 25 or 30) from a list box. Make any other modifications you deem necessary to the interface. Be sure to reset the tab order. Also be sure to select a default item in the new list box. The Calculate button should now display only the monthly payment corresponding to the selected term. Save the solution and then start and test the application. (If the principal, rate, and term are 125000, 3.0, and 30, respectively, the monthly payment is \$527.01.)

```

1  ' Name:      Modified 13.Payment Project
2  ' Purpose:    Display the monthly mortgage payments for terms of 15, 20, 25, and 30 years.
3  ' Programmer: <your name> on <current date>
4  'MOD: 1).let user select the term 15/20/25/30y from a lst
5  '       2). the Calculate btn should display only the monthly payment corresponding to the selected term
6  'TEST: 125000, 3.0, 30 years = $527.01
7  Option Explicit On

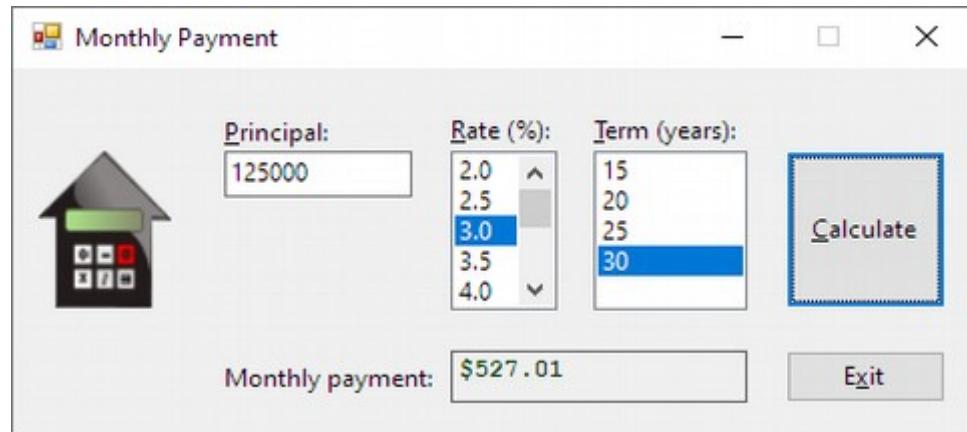
```

```
8 Option Strict On
9 Option Infer Off
10
11 Public Class frmMain
12     Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
13         ' Fill lstRates with rates_dblRates and select 3.0 rate: _dblRate
14
15         For dblRates As Double = 2 To 7 Step 0.5
16             lstRates.Items.Add(dblRates.ToString("N1"))
17         Next dblRates
18         lstRates.SelectedItem = "3.0"
19
20         'Fill lstTerms with terms_intTerms and select 30 years: _intTerm
21         For intTerms As Integer = 15 To 30 Step 5                      'my code
22             lstTerms.Items.Add(intTerms.ToString)                          'my code
23         Next intTerms   'my code
24         lstTerms.SelectedItem = "30"                                'my code
25     End Sub
26
27     Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
28         ' Display the monthly mortgage payment.
29
30         Dim intPrincipal As Integer
31         Dim dblRate As Double
32         Dim dblPay As Double
33         Dim intTerm As Integer                               'my code
34         Integer.TryParse(txtPrincipal.Text, intPrincipal)
35         Double.TryParse(lstRates.SelectedItem.ToString, dblRate)
36         Integer.TryParse(lstTerms.SelectedItem.ToString, intTerm)      'my code
37         dblRate /= 100
38
39         lblPay.Text = String.Empty
40
41         'For intTerm As Integer = 15 To 30 Step 5
42         dblPay = -Financial.Pmt(dblRate / 12, intTerm * 12, intPrincipal)
43         'lblPay.Text = lblPay.Text & intTerm.ToString & " years: " &
44         'dblPay.ToString("C2") & ControlChars.NewLine
45         'Next intTerm
46         lblPay.Text = dblPay.ToString("C2")
47     End Sub
48
49     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
50         Me.Close()
51     End Sub
```

```

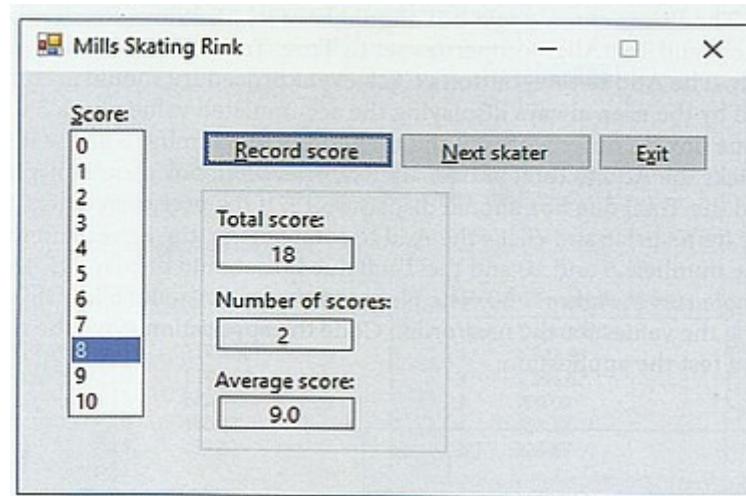
52
53     Private Sub txtPrincipal_Enter(sender As Object, e As EventArgs) Handles txtPrincipal.Enter
54         txtPrincipal.SelectAll()
55     End Sub
56
57     Private Sub txtPrincipal_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtPrincipal.KeyPress
58         ' Accept only numbers and the Backspace key.
59         If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
60             e.Handled = True
61         End If
62     End Sub
63
64     Private Sub txtPrincipal_TextChanged(sender As Object, e As EventArgs) Handles txtPrincipal.TextChanged
65         lblPay.Text = String.Empty
66     End Sub
67
68     Private Sub lstRates_SelectedIndexChanged(sender As Object, e As EventArgs) Handles lstRates.SelectedIndexChanged
69         lblPay.Text = String.Empty
70     End Sub
71
72     Private Sub lstTerms_SelectedIndexChanged(sender As Object, e As EventArgs) Handles lstTerms.SelectedIndexChanged
73         lblPay.Text = String.Empty
74     End Sub
75 End Class

```



Create a Windows Forms application. Use the following names for the project and solution, respectively: Mills Project and Mills Solution. Save the application in the VB2017\Chap05 folder. Mills Skating Rink holds a weekly ice-skating competition. Competing skaters must perform a two-minute program in front of a panel of judges. The number of judges varies from week to week. At the end of a skater's program, each judge assigns a score of 0 through 10 to the skater. The manager of the ice rink wants you to create an application that allows him to enter each judge's score for a specific skater. The application should calculate and display the skater's average score. It should also display the skater's total score and the number of scores entered. Figure 5-48 shows a sample run of the application, assuming the manager entered

two scores: 10 and 8. (You enter a score by selecting it from the list box and then clicking the Record score button.) Code the application. Save the solution and then start and test the application.



```

1  ' Exercise 11 (advanced):
2  ' The application should calculate and display the skater's average score.
3  ' It should also display the skater's total score and the number of scores entered.
4  Option Explicit On
5  Option Strict On
6  Option Infer Off
7  Public Class frmMain
8      ' because of the class-level scoop, i can erase value of the variables, when next skater:
9      Private int1Total As Integer
10     Private int2NumberOfScores As Integer
11
12     Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
13         For intScores As Integer = 0 To 10
14             lstScore.Items.Add(intScores)
15         Next intScores
16     End Sub
17
18     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
19         Me.Close()
20     End Sub
21

```

```

22  Private Sub btnRecord_Click(sender As Object, e As EventArgs) Handles btnRecord.Click
23      Dim intScore As Integer 'declare the Score selected in lstScore
24      Dim dbl3Average As Double
25      Integer.TryParse(lstScore.SelectedItem.ToString, intScore) 'convert intScores from lstScore into intScore
26
27      ' accumulator - !!! Class-level is important because Dim erases each time you click !!!:
28      int1Total += intScore
29
30      int2NumberOfScores += 1 'accumulator to count number of entries
31      dbl3Average = int1Total / int2NumberOfScores
32
33      lbl1Total.Text = int1Total.ToString("N0")
34      lbl2NumberOfScores.Text = int2NumberOfScores.ToString("N0")
35      lbl3Average.Text = dbl3Average.ToString("N2")
36  End Sub
37
38  Private Sub btnNext_Click(sender As Object, e As EventArgs) Handles btnNext.Click
39      lbl1Total.Text = String.Empty
40      lbl2NumberOfScores.Text = String.Empty
41      lbl3Average.Text = String.Empty
42      int1Total = 0
43      int2NumberOfScores = 0
44  End Sub
45 End Class

```

**Mills Skating Rink\_26...EXERCISE 11**

Score:

- 0
- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10

Total score:

Number of scores:

Average score:

**Record score**

**Mills Skating Rink\_26...EXERCISE 11**

Score:

- 0
- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10

Total score:

Number of scores:

Average score:

**Record score**

**Mills Skating Rink\_26...EXERCISE 11**

Score:

- 0
- 1
- 2
- 3
- 4
- 5
- 6
- 7
- 8
- 9
- 10

Total score:

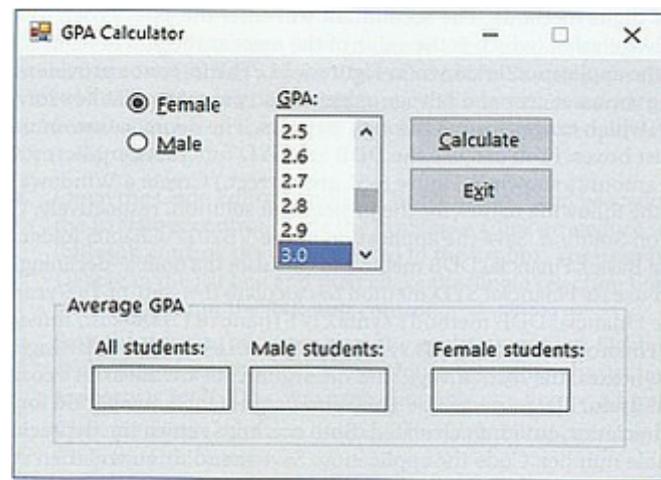
Number of scores:

Average score:

**Next skater**

**Exitus**

In this exercise, you create an application that allows the user to enter the gender (either F or M) and GPA for any number of students. The application should calculate the average GPA for all students, the average GPA for male students, and the average GPA for female students. Create a Windows Forms application. Use the following names for the project and solution, respectively: GPA Project and GPA Solution. Save the application in the VB2017\Chap05 folder. The application's interface is shown in Figure 5-49. The list box should list GPAs from 1.0 through 4.0 in increments of 0.1 (e.g., 1.0, 1.1, 1.2, 1.3, etc.). Code the application. Save the solution and then start and test the application.



```

1  '_Average
2  'Application allows the user to enter the gender - either Female or Male and GPA for any number of students.
3  'GPA = Grade Point Average - used in US for Bachelor's or Master's degrees....for more info see attached web page
4  'the application should calculate the Average GPA for:
5  ' 1.) all students
6  ' 2.) only male students
7  ' 3.) only female students
8  ' see Figure 5-49 as a guide for a GUI
9  '_lst GPA should list values: 1.0 through 4.0 in increments of 0.1
10 Option Strict On
11 Option Explicit On
12 Option Infer Off
13
14 Public Class frmMain
15     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
16         Me.Close()
17     End Sub
18
19     Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
20         For decGPAs As Decimal = 1D To 4D Step 0.1D
21             lstGPA.Items.Add(decGPAs.ToString("N1"))
22         Next decGPAs
23         lstGPA.SelectedIndex = 0
24     End Sub

```

```

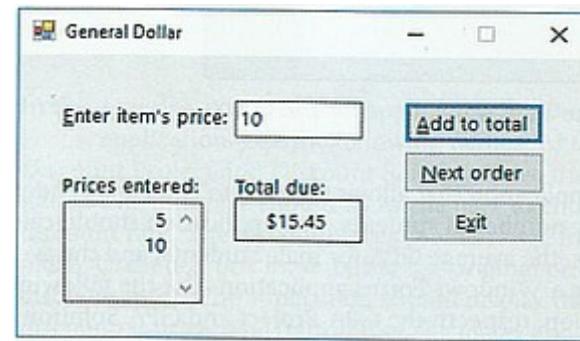
25
26     Private Sub btnCalculate_Click(sender As Object, e As EventArgs) Handles btnCalculate.Click
27         Static dec1All As Decimal
28         Static dec2Male As Decimal
29         Static dec3Female As Decimal
30         Dim decGPA As Decimal
31         Decimal.TryParse(lstGPA.SelectedItem.ToString, decGPA)
32
33         Select Case True
34             Case radFemale.Checked
35                 dec3Female += decGPA
36                 dec1All += decGPA
37             Case radMale.Checked
38                 dec2Male += decGPA
39                 dec1All += decGPA
40         End Select
41
42         lbl1All.Text = dec1All.ToString("N1")
43         lbl2Male.Text = dec2Male.ToString("N1")
44         lbl3Female.Text = dec3Female.ToString("N1")
45     End Sub
46 End Class

```

The application displays three different sets of input data for calculating average GPA:

- Set 1 (Left):** GPA dropdown shows 1.0. All students: 1.0, Male students: 0.0, Female students: 1.0.
- Set 2 (Middle):** GPA dropdown shows 4.0. All students: 5.0, Male students: 4.0, Female students: 1.0.
- Set 3 (Right):** GPA dropdown shows 2.6. All students: 7.6, Male students: 6.6, Female students: 1.0.

Create a Windows Forms application. Use the following names for the project and solution, respectively: General Project and General Solution. Save the application in the VB2017\Chap05 folder. A sample run of the application is shown in Figure 5-50. The interface allows the user to enter an item's price, which should be displayed in the Prices entered text box. The Prices entered text box should have its Multiline, ReadOnly, ScrollBars, TabStop, and TextAlign properties set to True, True, Vertical, False, and Right, respectively. The Add to total button's Click event procedure should accumulate the prices entered by the user, always displaying the accumulated value plus a 3% sales tax in the Total due box. In other words, if the user enters the number 5 as the item's price and then clicks the Add to total button, the Prices entered box should display the number 5 and the Total due box should display \$5.15. If the user then enters the number 10 as the item's price and clicks the Add to total button, the Prices entered box should display the numbers 5 and 10 and the Total due box should display \$15.45, as shown in the sample run in Figure 5-50. The Next order button should allow the user to start accumulating the values for the next order. Code the application. Save the solution and then start and test the application.



```

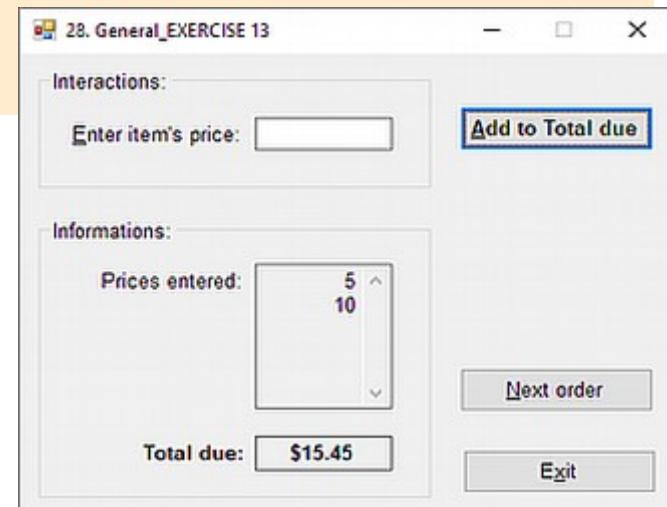
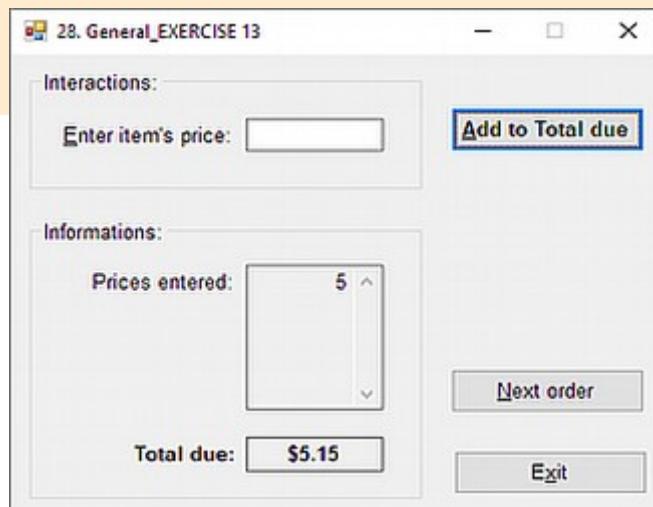
1  '28. General dollar_exercise 13_advanced
2  '-the interface allows the user to enter an item's price, which should be displayed in the "Prices Entered" TextBox.
3  ' - TextBox settings: Multiline = True, ReadOnly = True, ScrollBars = Vertical,
4  '                         TabStop = False (will be excluded from a Tab key circle), TextAlign = Right.
5  '-the "Add to Total" btn's Click event procedure should accumulate the prices entered by the user,
6  '                         always displaying the accumulated value plus a 3% sales tax in the "Total due" lbl.
7  'i.e. - if the user enters "5" as the item's price, the "Prices entered" (txt)
8  '       should display "5", and the "Total due" (lbl) should display $5.15.
9  ' - if the user then enters "10", the "Prices entered" (txt)
10 '      should display "5 and 10" and the "Total due" (lbl) should display "$15.45".
11 '-the "Next order" btn should allow the user to start accumulating the values for the next order.
12
13 Option Explicit On
14 Option Strict On
15 Option Infer Off
16
17 Public Class frmMain
18     ' important as Class-level - the values can be totally cleared when implementing another button for another calculations:
19     Private dec1_EnterItemPrice As Decimal
20     'Private dec2_PricesEntered As Decimal           ' no need to be a class-level
21     Private dec3_TotalDue As Decimal
22

```

```

23  Private Sub btnAdd_Click(sender As Object, e As EventArgs) Handles btnAdd.Click
24      Decimal.TryParse(txtEnter.Text, dec1_EnterItemPrice)
25      Const decSalesTax As Decimal = 0.03D
26
27      dec3_TotalDue += dec1_EnterItemPrice + (dec1_EnterItemPrice * decSalesTax)
28
29      lblTotal.Text = dec3_TotalDue.ToString("C2")
30
31      txtPricesEntered.Text += dec1_EnterItemPrice.ToString & ControlChars.NewLine
32      txtEnter.Text = String.Empty           'cleares the entry after clicking the Add btn
33  End Sub
34
35  Private Sub btnNext_Click(sender As Object, e As EventArgs) Handles btnNext.Click    ' clean all the values:
36      txtPricesEntered.Text = String.Empty
37      dec3_TotalDue = 0
38      lblTotal.Text = String.Empty
39  End Sub
40
41  Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
42      Me.Close()
43  End Sub
44
45  Private Sub txtEnter_Enter(sender As Object, e As EventArgs) Handles txtEnter.Enter
46      txtEnter.SelectAll()
47  End Sub
48
49  Private Sub txtEnter_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtEnter.KeyPress
50      If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back AndAlso e.KeyChar <> "." Then
51          e.Handled = True
52      End If
53  End Sub
54 End Class

```



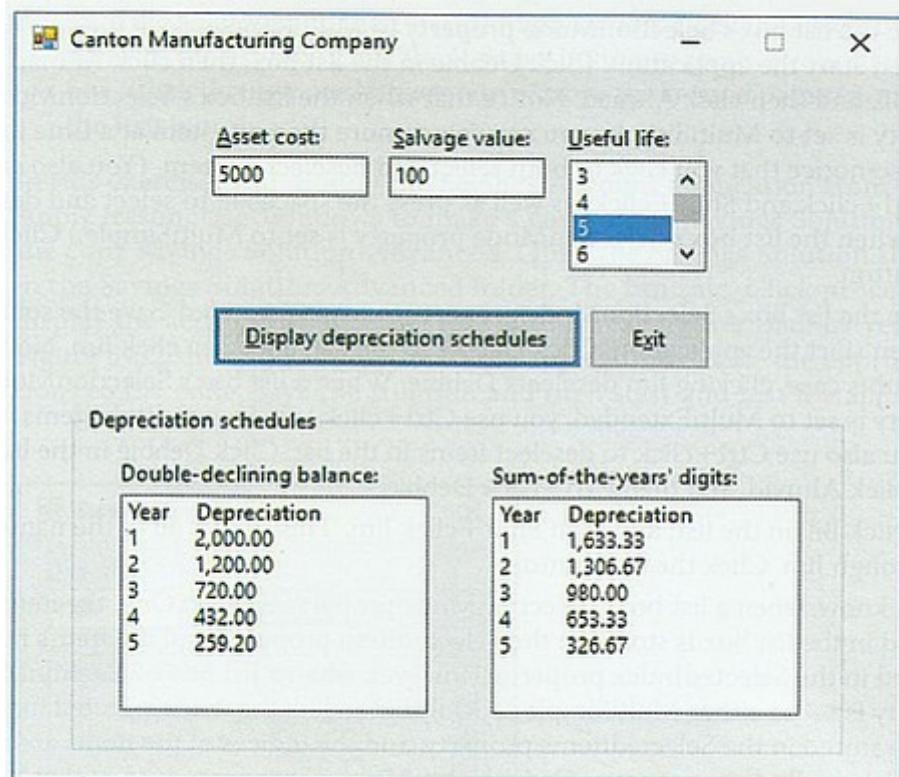
## Chap05\Exercise1

### 29.Canton Solution\_EXERCISE 14\_advanced

```
- frmMain_Load, For...To...Next, lst.Items.Add, lst.SelectedIndex, lst.Items.Clear, txt_Enter,
ControlChars.Tab, ControlChars.NewLine, Financial.DDB, Financial.SYD, txt_TextChanged,
txt_KeyPress, e.KeyChar, ControlChars.Back, lst_SelectedIndexChanged
```

The accountant at Canton Manufacturing Company wants you to create an application that calculates an asset's annual depreciation using the double-declining balance and sum-of-the-years' digits methods. The accountant will enter the asset's cost, useful life (in years), and salvage value (which is the value of the asset at the end of its useful life). A sample run of the application is shown in Figure 5-51. The interface provides text boxes for entering the asset cost and salvage value. It also provides a list box for selecting the useful life, which ranges from 3 through 20 years. The depreciation amounts are displayed in list boxes. (You can use the DDB and SYD functions in Microsoft Excel to verify that the amounts shown in Figure 5-51 are correct.) Create a Windows Forms application. Use the following names for the project and solution, respectively: Canton Project and Canton Solution. Save the application in the VB2017\Chap05 folder.

You can use Visual Basic's Financial.DDB method to calculate the double-declining balance depreciation, and use its Financial.SYD method to calculate the sum-of-the-years' digits depreciation. The Financial.DDB method's syntax is `Financial.DDB(cost, salvage, life, period)`. The Financial.SYD method's syntax is `Financial.SYD(cost, salvage, life, period)`. In both syntaxes, the `cost`, `salvage`, and `life` arguments are the asset's cost, salvage value, and useful life, respectively. The `period` argument is the period for which you want the depreciation amount calculated. Both methods return the depreciation amount as a Double number. Code the application. Save the solution and then start and test the application.



```
1  'user: - Asset cost = txtAssetCost = dblAssetCost
2  '      - Salvage value = txtSalvage = dblSalvage
3  '      - Useful life = lstUsefullLife = 3 through 20 years = For intUsefulLife -> Load
4  Option Explicit On
5  Option Strict On
6  Option Infer Off
7
8  Public Class frmMain
9    Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
10   For intUsefulLife As Integer = 3 To 20
11     lstUsefullLife.Items.Add(intUsefulLife.ToString)
12   Next intUsefulLife
13   lstUsefullLife.SelectedIndex = 2
14 End Sub
```

```
15
16  Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
17      Me.Close()
18  End Sub
19
20  Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
21      Dim dblAssetCost As Double
22      Dim dblSalvage As Double
23      Dim intLife As Integer      'takes the Selected value from lstUsefulLife = intUsefulLife
24      Dim dblDDB As Double       'Financial.DDB = (Cost, Salvage, Life, Period)
25      Dim dblSYD As Double       'Financial.SYD = (Cost, Salvage, Life, Period)
26      Double.TryParse(txtAssetCost.Text, dblAssetCost)
27      Double.TryParse(txtSalvage.Text, dblSalvage)
28      Integer.TryParse(lstUsefulLife.SelectedItem.ToString, intLife)
29      lstDDB.Items.Clear()
30      lstSYD.Items.Clear()
31
32      lstDDB.Items.Add("Year:" & ControlChars.Tab & "Depreciation:" & ControlChars.NewLine)
33      lstSYD.Items.Add("Year:" & ControlChars.Tab & "Depreciation:" & ControlChars.NewLine)
34
35      For intYears As Integer = 1 To intLife
36          dblDDB = Financial.DDB(dblAssetCost, dblSalvage, intLife, intYears)
37          dblSYD = Financial.SYD(dblAssetCost, dblSalvage, intLife, intYears)
38
39          lstDDB.Items.Add(intYears & ControlChars.Tab & dblDDB.ToString("N2") & ControlChars.NewLine)
40          lstSYD.Items.Add(intYears & ControlChars.Tab & dblSYD.ToString("N2") & ControlChars.NewLine)
41      Next intYears
42  End Sub
43
44  Private Sub txtAssetCost_Enter(sender As Object, e As EventArgs) Handles txtAssetCost.Enter
45      txtAssetCost.SelectAll()
46  End Sub
47
48  Private Sub txtSalvage_Enter(sender As Object, e As EventArgs) Handles txtSalvage.Enter
49      txtSalvage.SelectAll()
50  End Sub
51
52  Private Sub txtAssetCost_TextChanged(sender As Object, e As EventArgs) Handles txtAssetCost.TextChanged
53      lstDDB.Items.Clear()
54      lstSYD.Items.Clear()
55  End Sub
56
57  Private Sub txtSalvage_TextChanged(sender As Object, e As EventArgs) Handles txtSalvage.TextChanged
58      lstDDB.Items.Clear()
```

```

59         1stSYD.Items.Clear()
60     End Sub
61
62     Private Sub txtAssetCost_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtAssetCost.KeyPress
63         If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back AndAlso e.KeyChar <> "." Then
64             e.Handled = True
65         End If
66     End Sub
67
68     Private Sub txtSalvage_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtSalvage.KeyPress
69         If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back AndAlso e.KeyChar <> "." Then
70             e.Handled = True
71         End If
72     End Sub
73
74     Private Sub 1stUsefulLife_SelectedIndexChanged(sender As Object, e As EventArgs) Handles 1stUsefulLife.SelectedIndexChanged
75         1stDDB.Items.Clear()
76         1stSYD.Items.Clear()
77     End Sub
78 End Class

```

29.Canton Manufacturing Company\_exercise 14

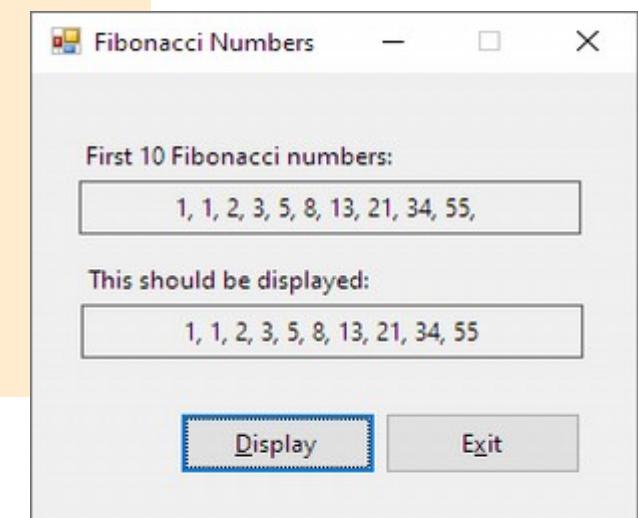
|                                                                                                      |                                                         |                                                                         |                                                                         |
|------------------------------------------------------------------------------------------------------|---------------------------------------------------------|-------------------------------------------------------------------------|-------------------------------------------------------------------------|
| <u>Asset cost:</u>                                                                                   | <input type="text" value="5000"/>                       | <u>Depreciation schedules</u>                                           |                                                                         |
| <u>Salvage value:</u>                                                                                | <input type="text" value="100"/>                        | <u>Double-declining balance:</u>                                        | <u>Sum-of-the-year's digits:</u>                                        |
| <u>Useful life (years):</u>                                                                          | <input type="text" value="5"/><br>3    4    5    6    7 | Year: 1 2 3 4 5<br>Depreciation: 2,000.00 1,200.00 720.00 432.00 259.20 | Year: 1 2 3 4 5<br>Depreciation: 1,633.33 1,306.67 980.00 653.33 326.67 |
| <input type="button" value="Display depreciation schedules"/><br><input type="button" value="Exit"/> |                                                         |                                                                         |                                                                         |

15. Open the Fibonacci Solution.sln file contained in the VB2017\Chap05\Fibonacci Solution folder. The application should display the first 10 Fibonacci numbers: 1, 1, 2, 3, 5, 8, 13, 21, 34, and 55. Notice that beginning with the third number in the series, each Fibonacci number is the sum of the prior two numbers. For example, 2 is the sum of 1 plus 1, 3 is the sum of 1 plus 2, 5 is the sum of 2 plus 3, and so on. Display the numbers in the lblNumbers control. Code the btnDisplay\_Click procedure. Save the solution and then start and test the application.

```

1  ' Name:      30.Fibonacci Project_EXERCISE 15
2  ' Purpose:    Displays the first 10 Fibonacci numbers.
3  'The application should display the first 10 Fibonacci numbers:
4  'Each Fibonacci number is the sum of the prior two numbers
5  Option Explicit On
6  Option Strict On
7  Option Infer Off
8
9  Public Class frmMain
10     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
11         Me.Close()
12     End Sub
13
14     Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
15         ' Displays the first 10 Fibonacci numbers.
16         Dim A As Integer = 0      ' tosemdal
17         Dim B As Integer = 1      ' tosemdal
18         Dim C As Integer        ' tosemdal
19
20         ' Couldn't make it, the solution is from some Forum about VB.NET
21         Do
22             C = A + B          ' tosemdal
23             A = B              ' tosemnedal
24             B = C              ' tosemnedal
25             lblNumbers.Text += A.ToString & ", "
26             Loop Until A = 55
27             lblNumbersCorrect.Text = "1, 1, 2, 3, 5, 8, 13, 21, 34, 55"
28         End Sub
29     End Class

```



- **1st Properties:** **SelectionMode** = None/One/MultiSimple/MultiExtended; **Sorted** = True  
 - **frmMain\_Load**, **1st.Items.Add**, **1st.SelectedItems.Count**, **For...To...Next**,  
**1st.SelectedItems.Item()**, **ControlChars.NewLine**

In this exercise, you learn how to create a list box that allows the user to select more than one item at a time. Open the Multi Solution.sln file contained in the VB2017\Chap05\Multi Solution folder. The interface contains a list box named **lstNames**. The list box's **Sorted** and **SelectionMode** properties are set to **True** and **One**, respectively.

- a. Open the Code Editor window. The **frmMain\_Load** procedure adds five names to the **lstNames** control. Code the **btnSingle\_Click** procedure so that it displays, in the **lblResult** control, the item selected in the list box. For example, if the user clicks Debbie in the list box and then clicks the Single selection button, the name Debbie should appear in the **lblResult** control.
- b. Save the solution and then start the application. Click Debbie in the list box, then click Ahmad, and then click Bill. Notice that when the list box's **SelectionMode** property is set to **One**, you can select only one item at a time in the list.
- c. Click the Single selection button. The name Bill appears in the **lblResult** control. Click the Exit button.
- d. Change the list box's **SelectionMode** property to **MultiSimple**. Save the solution and then start the application. Click Debbie in the list box, then click Ahmad, then click Bill, and then click Ahmad. Notice that when the list box's **SelectionMode** property is set to **MultiSimple**, you can select more than one item at a time in the list. Also notice that you click to both select and deselect an item. (You also can use **Ctrl+click** and **Shift+click**, as well as press the spacebar, to select and deselect items when the list box's **SelectionMode** property is set to **MultiSimple**.) Click the Exit button.

- e. Change the list box's **SelectionMode** property to **MultiExtended**. Save the solution and then start the application. Click Debbie in the list, and then click Jim. Notice that in this case, clicking Jim deselects Debbie. When a list box's **SelectionMode** property is set to **MultiExtended**, you use **Ctrl+click** to select multiple items in the list. You also use **Ctrl+click** to deselect items in the list. Click Debbie in the list, then **Ctrl+click** Ahmad, and then **Ctrl+click** Debbie.
- f. Next, click Bill in the list, and then **Shift+click** Jim. This selects all of the names from Bill through Jim. Click the Exit button.
- g. As you know, when a list box's **SelectionMode** property is set to **One**, the item selected in the list box is stored in the **SelectedItem** property, and the item's index is stored in the **SelectedIndex** property. However, when a list box's **SelectionMode** property is set to either **MultiSimple** or **MultiExtended**, the items selected in the list box are stored in the **SelectedItems** property, and the indices of the items are stored in the **SelectedIndices** property. Code the **btnMulti\_Click** procedure so that it first clears the contents of the **lblResult** control. The procedure should then display the selected names (which are stored in the **SelectedItems** property) on separate lines in the **lblResult** control.
- h. Save the solution and then start the application. Click Ahmad in the list box, and then **Shift+click** Jim. Click the Multi-selection button. The five names should appear on separate lines in the **lblResult** control. Click the Exit button.

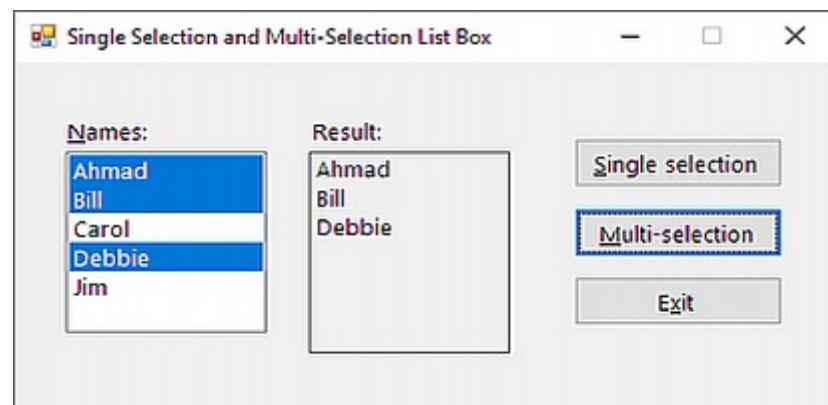
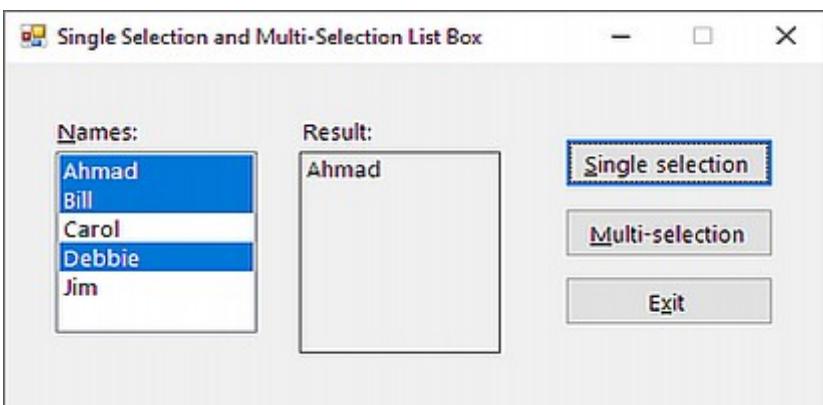
```

1  ' Name:      Multi Project
2  ' Purpose:   Demonstrates single selection and multi-selection list boxes.
3  Option Explicit On
4  Option Strict On
5  Option Infer Off
6
7  Public Class frmMain
8      Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
9          Me.Close()
10     End Sub
11 
```

```

12  Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
13      ' Fills the list box with values.
14      lstNames.Items.Add("Debbie")
15      lstNames.Items.Add("Bill")
16      lstNames.Items.Add("Jim")
17      lstNames.Items.Add("Ahmad")
18      lstNames.Items.Add("Carol")
19  End Sub
20
21  Private Sub btnSingle_Click(sender As Object, e As EventArgs) Handles btnSingle.Click
22      lblResult.Text = String.Empty
23
24      If lstNames.SelectedItems.Count = 0 Then      ' prevents an error when nothing is selected
25          lblResult.Text = String.Empty           ' prevents an error when nothing is selected
26      Else
27          lblResult.Text = lstNames.SelectedItem.ToString
28      End If
29  End Sub
30
31  Private Sub btnMulti_Click(sender As Object, e As EventArgs) Handles btnMulti.Click
32      lblResult.Text = String.Empty
33      'lstNames.Items.Count                  <- number of ALL items      COUNT
34      'lstNames.SelectedItems.Count          <- number of SELECTED items    COUNT
35      'lstNames.SelectedItems                <- the COLLECTION of selected items ITEMS
36      'lstNames.SelectedIndices            <- the INDICES of items       INDEXES
37
38      For a As Integer = 0 To lstNames.SelectedItems.Count - 1
39          lblResult.Text += lstNames.SelectedItems.Item(a).ToString & ControlChars.NewLine
40      Next a
41  End Sub
42 End Class

```



## Chap05\Exercise1

### 32.Items Collection Solution\_EXERCISE 17\_advanced

- frmMain\_Load, lst.Items.Add, lst.Items.Count, lst.Items.Insert,  
lst.Items.Remove, lst.Items.RemoveAt

In this exercise, you learn how to use the Items collection's Insert, Remove, and RemoveAt methods. Open the Items Collection Solution.sln file contained in the VB2017\Chap05\Items Collection Solution folder.

- a. The Items collection's Insert method allows you to add an item at a desired position in a list box during run time. The Insert method's syntax is `object.Items.Insert(position, item)`, where *position* is the index of the item. Code the btnInsert\_Click procedure so it adds your name as the fourth item in the list box. (Recall that the first item in a list box has an index of 0.)
- b. The Items collection's Remove method allows you to remove an item from a list box during run time. The Remove method's syntax is `object.Items.Remove(item)`, where *item* is the item's value. Code the btnRemove\_Click procedure so it removes your name from the list box.

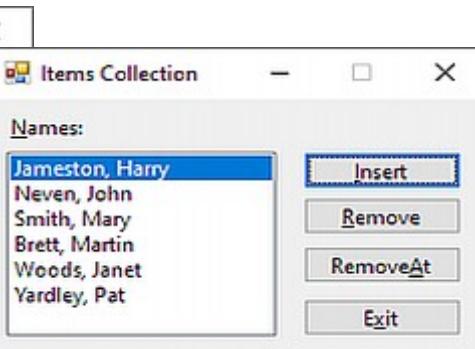
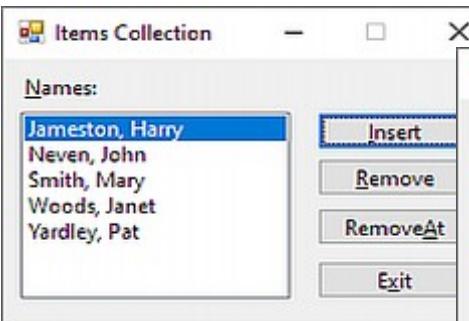
- c. Like the Remove method, the Items collection's RemoveAt method also allows you to remove an item from a list box while an application is running. However, in the RemoveAt method, you specify the item's index rather than its value. The RemoveAt method's syntax is `object.Items.RemoveAt(index)`, where *index* is the item's index. Code the btnRemoveAt\_Click procedure so it removes the second name from the list box.
- d. Save the solution and then start the application. Click the Insert button to add your name to the list box. Click the Remove button to remove your name from the list box. Click the RemoveAt button to remove the second name from the list box. Click the Exit button.

```
1  ' Name:      32.ListBox Items Project_EXERCISE 17
2  ' Purpose:    Demonstrate the Items collection's Insert, Remove, and RemoveAt methods.
3  Option Explicit On
4  Option Strict On
5  Option Infer Off
6
7  Public Class frmMain
8      Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
9          ' Fills the list box with values and then selects the first value.
10         lstNames.Items.Add("Jameston, Harry")
11         lstNames.Items.Add("Neven, John")
12         lstNames.Items.Add("Smith, Mary")
13         lstNames.Items.Add("Woods, Janet")
14         lstNames.Items.Add("Yardley, Pat")
15         lstNames.SelectedIndex = 0
16     End Sub
17
18     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
19         Me.Close()
20     End Sub
21 
```

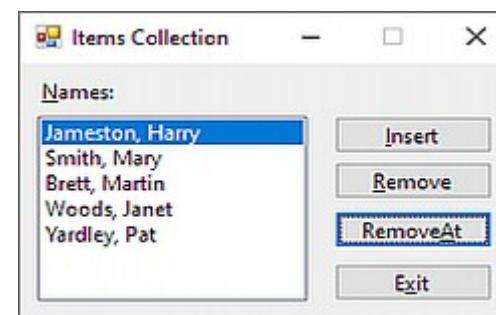
```

22  Private Sub btnInsert_Click(sender As Object, e As EventArgs) Handles btnInsert.Click
23      ' Adds a name at a specified position in the list box.
24      ' Items collection's "Insert" method allows you to add an item at a desired position in a lst during run time
25      '_syntax: object.Items.Insert(position index, item)
26
27      If lstNames.Items.Count > 3 Then          ' if there is more than 3 names
28          lstNames.Items.Insert(3, "Brett, Martin")    ' will add my name on the 4th position (index 3)
29      Else
30          lstNames.Items.Insert(lstNames.Items.Count, "Brett, Martin")
31      End If
32  End Sub
33
34  Private Sub btnRemove_Click(sender As Object, e As EventArgs) Handles btnRemove.Click
35      ' Removes a specified name from the list box.
36      ' Items collection's "Remove" method allows you to remove an item from a lst during run time
37      '_syntax: object.Items.Remove(item)
38
39      lstNames.Items.Remove("Brett, Martin")           ' will remove defined last added name from a lst
40  End Sub
41
42  Private Sub btnRemoveAt_Click(sender As Object, e As EventArgs) Handles btnRemoveAt.Click
43      ' Removes any name from a specified position in the list box.
44      ' Items collection's "RemoveAt" method also allows you to remove an item from a lst during run time.
45      ' However, you specify the item's index rather than its value.
46      '_syntax: object.Items.RemoveAt(index)
47
48      If lstNames.Items.Count > 1 Then          ' if there is more than 1 name, then
49          lstNames.Items.RemoveAt(1)            ' will remove 2nd name
50      End If
51  End Sub
52 End Class

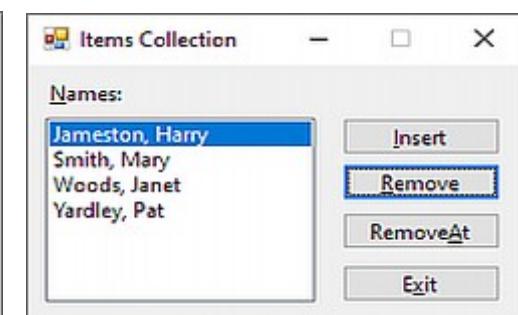
```



inserts my name on the 4th position



removes any 2nd name



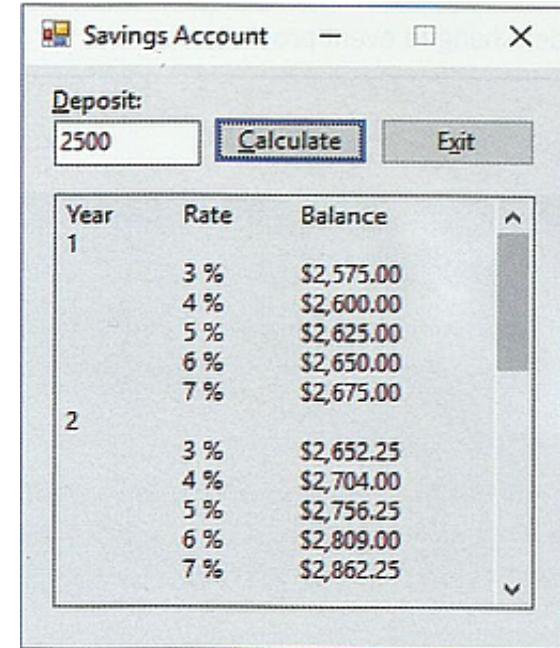
removes defined name

## Chap05\\_Exercise1

### 33.Savings Solution 14-Advanced\_EXERCISE 18\_advanced

- ControlChars.Tab, ControlChars.NewLine, For...To...Step...Next, txt\_Enter,  
txt\_TextChanged, txt\_KeyPress, e.KeyChar, ControlChars.Back

In this exercise, you modify the Savings Account application from this chapter's Apply lesson. Use Windows to make a copy of the Savings Solution folder. Rename the copy Savings Solution-Advanced. Open the Savings Solution.sln file contained in the Savings Solution-Advanced folder. The btnCalc\_Click procedure should now display the account balances by rate within year (rather than by year within rate). Figure 5-52 shows a sample run of the application. Make the appropriate modifications to the code. Save the solution and then start and test the application.



```
1  ' Name:      33.Savings Project-Advanced_EXERCISE 18
2  ' Purpose:    Display a savings account balance
3  '
4  '           for each of 5 years using rates
5  '           from 3% to 7% in increments of 1%.
6  ' Mod: display the account balances by rate within year (rather than by year within rate)
7  Option Explicit On
8  Option Strict On
9  Option Infer Off
10
11 Public Class frmMain
12     Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
13         ' Calculate account balances for each of five years using rates from 3% to 7% in increments of 1%.
14         Dim dblDeposit As Double
15         Dim dblBalance As Double
16         Dim dblDeposit2 As Double
17         Dim dblBalance2 As Double
18         Double.TryParse(txtDeposit.Text, dblDeposit)
19         Double.TryParse(txtDeposit.Text, dblDeposit2)
```

```
19
20     ' Header_original for txtBalance:
21     txtBalance.Text = "Rate" & ControlChars.Tab & "Year" & ControlChars.Tab & "Balance" & ControlChars.NewLine
22
23     ' Calculate and display account balances in txtBalance:
24     For dblRate As Double = 0.03 To 0.07 Step 0.01
25         txtBalance.Text = txtBalance.Text & dblRate.ToString("P0") & ControlChars.NewLine
26         For intYear As Integer = 1 To 5
27             dblBalance = dblDeposit * (1 + dblRate) ^ intYear
28             txtBalance.Text = txtBalance.Text & ControlChars.Tab & intYear.ToString & ControlChars.Tab &
29                             dblBalance.ToString("C2") & ControlChars.NewLine
30         Next intYear
31     Next dblRate
32
33     'Header_modified for txtBalance2:
34     txtBalance2.Text = "Year" & ControlChars.Tab & "Rate" & ControlChars.Tab & "Balance" & ControlChars.NewLine
35
36     'Calculate and display account balances in txtBalance2:
37     For intYear2 As Integer = 1 To 5
38         txtBalance2.Text = txtBalance2.Text & intYear2.ToString & ControlChars.NewLine
39         For dblRate2 As Double = 0.03 To 0.07 Step 0.01
40             dblBalance2 = dblDeposit2 * (1 + dblRate2) ^ intYear2
41             txtBalance2.Text = txtBalance2.Text & ControlChars.Tab & dblRate2.ToString("P0") & ControlChars.Tab &
42                             dblBalance2.ToString("C2") & ControlChars.NewLine
43         Next dblRate2
44     Next intYear2
45 End Sub
46
47 Private Sub txtDeposit_Enter(sender As Object, e As EventArgs) Handles txtDeposit.Enter
48     txtDeposit.SelectAll()
49 End Sub
50
51 Private Sub txtDeposit_TextChanged(sender As Object, e As EventArgs) Handles txtDeposit.TextChanged
52     txtBalance.Text = String.Empty
53     txtBalance2.Text = String.Empty
54 End Sub
55
56 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
57     Me.Close()
58 End Sub
59
60 Private Sub txtDeposit_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtDeposit.KeyPress
61     ' Allows the text box to accept only numbers, the period, and the Backspace key.
62
```

```

63     If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> "." AndAlso e.KeyChar <> ControlChars.Back Then
64         e.Handled = True
65     End If
66 End Sub
67 End Class

```

Savings Account

Deposit:

|      |                  |             |
|------|------------------|-------------|
| 2500 | <u>Calculate</u> | <u>Exit</u> |
|------|------------------|-------------|

| Rate | Year | Balance    | Year | Rate | Balance    |
|------|------|------------|------|------|------------|
| 3%   | 1    | \$2,575.00 | 1    | 3%   | \$2,575.00 |
|      | 2    | \$2,652.25 |      | 4%   | \$2,600.00 |
|      | 3    | \$2,731.82 |      | 5%   | \$2,625.00 |
|      | 4    | \$2,813.77 |      | 6%   | \$2,650.00 |
|      | 5    | \$2,898.19 |      | 7%   | \$2,675.00 |
| 4%   | 1    | \$2,600.00 | 2    | 3%   | \$2,652.25 |
|      | 2    | \$2,704.00 |      | 4%   | \$2,704.00 |
|      | 3    | \$2,812.16 |      | 5%   | \$2,756.25 |
|      | 4    | \$2,924.65 |      | 6%   | \$2,809.00 |
|      | 5    | \$3,041.63 |      | 7%   | \$2,862.25 |
| 5%   | 1    | \$2,625.00 | 3    | 3%   | \$2,731.82 |
|      | 2    | \$2,756.25 |      | 4%   | \$2,812.16 |
|      | 3    | \$2,894.06 |      | 5%   | \$2,894.06 |
|      | 4    | \$3,038.77 |      | 6%   | \$2,977.54 |
|      | 5    | \$3,190.70 |      | 7%   | \$3,062.61 |
| 6%   | 1    | \$2,650.00 | 4    | 3%   | \$2,813.77 |
|      | 2    | \$2,809.00 |      | 4%   | \$2,924.65 |
|      | 3    | \$2,977.54 |      | 5%   | \$3,038.77 |
|      | 4    | \$3,156.19 |      | 6%   | \$3,156.19 |
|      | 5    | \$3,345.56 |      | 7%   | \$3,276.99 |
| 7%   | 1    | \$2,675.00 | 5    | 3%   | \$2,898.19 |
|      | 2    | \$2,862.25 |      | 4%   | \$3,041.63 |
|      | 3    | \$3,062.61 |      | 5%   | \$3,190.70 |
|      | 4    | \$3,276.99 |      | 6%   | \$3,345.56 |
|      | 5    | \$3,506.38 |      | 7%   | \$3,506.38 |

original modified

Savings Account

Deposit:

|    |                  |             |
|----|------------------|-------------|
| 10 | <u>Calculate</u> | <u>Exit</u> |
|----|------------------|-------------|

| Rate | Year | Balance | Year | Rate | Balance |
|------|------|---------|------|------|---------|
| 3%   | 1    | \$10.30 | 1    | 3%   | \$10.30 |
|      | 2    | \$10.61 |      | 4%   | \$10.40 |
|      | 3    | \$10.93 |      | 5%   | \$10.50 |
|      | 4    | \$11.26 |      | 6%   | \$10.60 |
|      | 5    | \$11.59 |      | 7%   | \$10.70 |
| 4%   | 1    | \$10.40 | 2    | 3%   | \$10.61 |
|      | 2    | \$10.82 |      | 4%   | \$10.82 |
|      | 3    | \$11.25 |      | 5%   | \$11.03 |
|      | 4    | \$11.70 |      | 6%   | \$11.24 |
|      | 5    | \$12.17 |      | 7%   | \$11.45 |
| 5%   | 1    | \$10.50 | 3    | 3%   | \$10.93 |
|      | 2    | \$11.03 |      | 4%   | \$11.25 |
|      | 3    | \$11.58 |      | 5%   | \$11.58 |
|      | 4    | \$12.16 |      | 6%   | \$11.91 |
|      | 5    | \$12.76 |      | 7%   | \$12.25 |
| 6%   | 1    | \$10.60 | 4    | 3%   | \$11.26 |
|      | 2    | \$11.24 |      | 4%   | \$11.70 |
|      | 3    | \$11.91 |      | 5%   | \$12.16 |
|      | 4    | \$12.62 |      | 6%   | \$12.62 |
|      | 5    | \$13.38 |      | 7%   | \$13.11 |
| 7%   | 1    | \$10.70 | 5    | 3%   | \$11.59 |
|      | 2    | \$11.45 |      | 4%   | \$12.17 |
|      | 3    | \$12.25 |      | 5%   | \$12.76 |
|      | 4    | \$13.11 |      | 6%   | \$13.38 |
|      | 5    | \$14.03 |      | 7%   | \$14.03 |

original modified

## Chap05\Exercise1

### 34.Salary Solution\_EXERCISE 19\_advanced

- txt Properties: **MaxLength = 5**, **For...To...Step...Next**, **ToString("F2")**, **txt\_TextChanged**, **ControlChars.Tab**, **.NewLine**, **.Back**, **txt\_Enter**, **txt\_KeyPress**, **e.KeyChar**

19. Create a Windows Forms application. Use the following names for the project and solution, respectively: Salary Project and Salary Solution. Save the application in the VB2017\Chap05 folder. At the beginning of every year, you receive a raise on your previous year's salary. Create an application that displays the amount of your annual raises and also your new salaries for the next five years, using raise rates of 1.5%, 2%, 2.5%, and 3%. Create a suitable interface. Use a text box to enter your current salary. Code the application. Save the solution and then start and test the application.

```
1  ' At the beginning of every year, you receive a raise on your previous year's salary.
2  ' Display the amount of your annual raises and also your new salaries for the next 5 years,
3  '     using raise rates of: 1.5%, 2%, 2.5%, 3%.
4  ' - use a text box to enter your current salary.
5
6  ' txtSalary -> intSalary
7  ' years 1 - 5 -> intYears
8  ' rate 1.5% - 3% Step 0.5% -> dblRate
9  ' intNewSalary = ___calculation___ <- BUT each year has a changed value calculated from
10 '                 a previous one !!!! viz Fibonacci !!!
11 ' txtShow
12 Option Explicit On
13 Option Strict On
14 Option Infer Off
15
16 Public Class frmMain
17     Private Sub btnCalculate_Click(sender As Object, e As EventArgs) Handles btnCalculate.Click
18         Dim dblSalary As Double
19         Dim dblRaise As Double
20         Dim dblNewSalary As Double
21         Double.TryParse(txtSalary.Text, dblSalary)
22         ' header:
23         txtShow.Text = " Years:" & ControlChars.Tab & "Rate:" & ControlChars.Tab & "Raise:" & ControlChars.Tab &
24             " " & "New Salary:" & ControlChars.NewLine
25
```

```

26      ' Previous solution example, i think to use the template:
27      'For intYear2 As Integer = 1 To 5
28      '    txtBalance2.Text = txtBalance2.Text & intYear2.ToString & ControlChars.NewLine
29      '    For dblRate2 As Double = 0.03 To 0.07 Step 0.01
30      '        dblBalance2 = dblDeposit2 * (1 + dblRate2) ^ intYear2
31      '        txtBalance2.Text = txtBalance2.Text & ControlChars.Tab & dblRate2.ToString("P0") & ControlChars.Tab &
32      '                                dblBalance2.ToString("C2") & ControlChars.NewLine
33      '    Next dblRate2
34  'Next intYear2
35
36  For intYear As Integer = 1 To 5
37      txtShow.Text = txtShow.Text & intYear.ToString & ControlChars.NewLine
38
39      For decRate As Decimal = 0.015D To 0.03D Step 0.005D
40          'dblBalance2 = dblDeposit2 * (1 + dblRate2) ^ intYear2
41          dblNewSalary = dblSalary * (1 + decRate) ^ intYear
42          dblRaise = dblNewSalary - dblSalary
43
44          txtShow.Text = txtShow.Text & ControlChars.Tab & decRate.ToString("P1") & ControlChars.Tab & "$" &
45              dblRaise.ToString("F2") & ControlChars.Tab & "    " & dblNewSalary.ToString("C2") &
46              ControlChars.NewLine
47          ' long time had a problem, when input 5 digits and more -> formatting did not align the numbers,
48          ' until i tried: "$" & dblRaise.ToString("F2") instead of: dblRaise.ToString("C2"),
49          ' even "N2" won't show the desired formatting - WHY?
50      Next decRate
51  Next intYear
52 End Sub
53
54 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
55     Me.Close()
56 End Sub
57
58 Private Sub txtSalary_Enter(sender As Object, e As EventArgs) Handles txtSalary.Enter
59     txtSalary.SelectAll()
60 End Sub
61
62 Private Sub txtSalary_TextChanged(sender As Object, e As EventArgs) Handles txtSalary.TextChanged
63     txtShow.Text = String.Empty
64 End Sub
65

```

```

66  Private Sub txtSalary_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtSalary.KeyPress
67      If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
68          e.Handled = True
69      End If
70  End Sub
71 End Class

```

34.Salary\_EXERCISE 19

Enter your annual salary:

Calculate Exit

| Years: | Rate: | Raise: | New Salary: |
|--------|-------|--------|-------------|
| 1      | 1.5%  | \$0.15 | \$10.15     |
|        | 2.0%  | \$0.20 | \$10.20     |
|        | 2.5%  | \$0.25 | \$10.25     |
|        | 3.0%  | \$0.30 | \$10.30     |
| 2      | 1.5%  | \$0.30 | \$10.30     |
|        | 2.0%  | \$0.40 | \$10.40     |
|        | 2.5%  | \$0.51 | \$10.51     |
|        | 3.0%  | \$0.61 | \$10.61     |
| 3      | 1.5%  | \$0.46 | \$10.46     |
|        | 2.0%  | \$0.61 | \$10.61     |
|        | 2.5%  | \$0.77 | \$10.77     |
|        | 3.0%  | \$0.93 | \$10.93     |
| 4      | 1.5%  | \$0.61 | \$10.61     |
|        | 2.0%  | \$0.82 | \$10.82     |
|        | 2.5%  | \$1.04 | \$11.04     |
|        | 3.0%  | \$1.26 | \$11.26     |
| 5      | 1.5%  | \$0.77 | \$10.77     |
|        | 2.0%  | \$1.04 | \$11.04     |
|        | 2.5%  | \$1.31 | \$11.31     |
|        | 3.0%  | \$1.59 | \$11.59     |

34.Salary\_EXERCISE 19

Enter your annual salary:

Calculate Exit

| Years: | Rate: | Raise:    | New Salary: |
|--------|-------|-----------|-------------|
| 1      | 1.5%  | \$900.00  | \$60,900.00 |
|        | 2.0%  | \$1200.00 | \$61,200.00 |
|        | 2.5%  | \$1500.00 | \$61,500.00 |
|        | 3.0%  | \$1800.00 | \$61,800.00 |
| 2      | 1.5%  | \$1813.50 | \$61,813.50 |
|        | 2.0%  | \$2424.00 | \$62,424.00 |
|        | 2.5%  | \$3037.50 | \$63,037.50 |
|        | 3.0%  | \$3654.00 | \$63,654.00 |
| 3      | 1.5%  | \$2740.70 | \$62,740.70 |
|        | 2.0%  | \$3672.48 | \$63,672.48 |
|        | 2.5%  | \$4613.44 | \$64,613.44 |
|        | 3.0%  | \$5563.62 | \$65,563.62 |
| 4      | 1.5%  | \$3681.81 | \$63,681.81 |
|        | 2.0%  | \$4945.93 | \$64,945.93 |
|        | 2.5%  | \$6228.77 | \$66,228.77 |
|        | 3.0%  | \$7530.53 | \$67,530.53 |
| 5      | 1.5%  | \$4637.04 | \$64,637.04 |
|        | 2.0%  | \$6244.85 | \$66,244.85 |
|        | 2.5%  | \$7884.49 | \$67,884.49 |
|        | 3.0%  | \$9556.44 | \$69,556.44 |

- frmMain\_Load, lst.Items.Add, lst.SelectedIndex, ElseIf, Do...Loop Until, txt\_TextChanged,  
txt\_Enter, txt\_KeyPress, e.KeyChar, ControlChars.Back, .NewLine, lst\_SelectedIndexChanged

Create a Windows Forms application. Use the following names for the project and solution, respectively: OnYourOwn Project and OnYourOwn Solution. Save the application in the VB2017\Chap05 folder. Plan and design an application of your choice. The only requirement is that you must follow the minimum guidelines listed in Figure 5-53. Before starting the application, be sure to verify the name of the startup form. Save the solution and then start and test the application.

1. The user interface must contain a minimum of one text box, three labels, one list box, and two buttons. One of the buttons must be an Exit button.
2. The interface can include a picture box, but this is not a requirement.

3. The interface must follow the GUI design guidelines summarized in Figure 2-20 in Chapter 2, in Figure 4-52 in Chapter 4, and in Figure 5-45 in Chapter 5. The guidelines are also listed in Appendix A.
4. Objects that are either coded or referred to in code should be named appropriately.
5. The Code Editor window must contain comments, the three Option statements, at least two variables, at least two assignment statements, at least one loop, and the Me.Close() statement. The application must perform at least one calculation.
6. Every text box on the form should have its TextChanged and Enter event procedures coded. At least one of the text boxes should have its KeyPress event procedure coded.
7. The list box should have its SelectedIndexChanged event procedure coded.

```

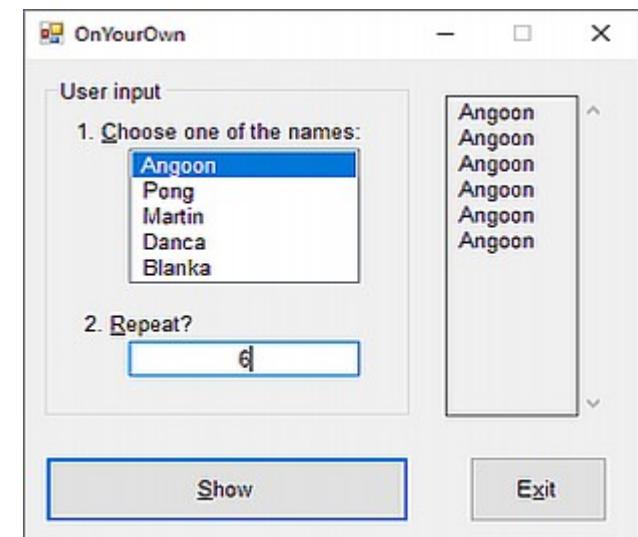
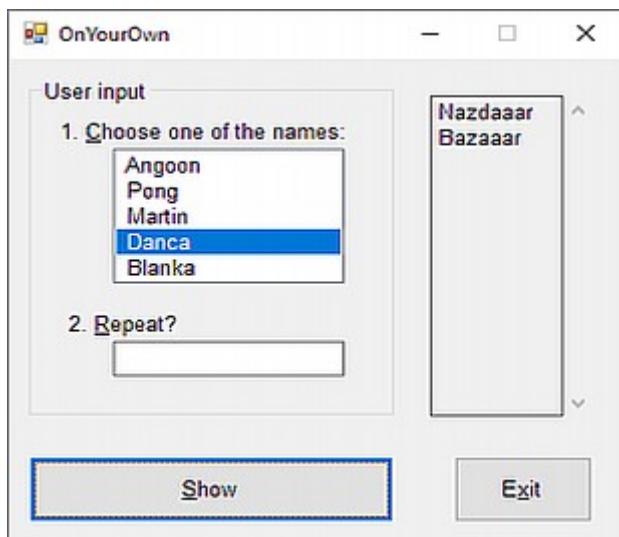
1  Option Explicit On
2  Option Strict On
3  Option Infer Off
4
5  Public Class frmMain
6      Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
7          lstNames.Items.Add(" Angoon")
8          lstNames.Items.Add(" Pong")
9          lstNames.Items.Add(" Martin")
10         lstNames.Items.Add(" Danca")
11         lstNames.Items.Add(" Blanka")
12         lstNames.SelectedIndex = 1
13     End Sub
14
15     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
16         Me.Close()
17     End Sub
18
19     Private Sub txtNumber_Enter(sender As Object, e As EventArgs) Handles txtNumber.Enter
20         txtNumber.SelectAll()
21         txtShow.Text = String.Empty
22     End Sub
23
24     Private Sub txtNumber_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtNumber.KeyPress
25         If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
26             e.Handled = True
27         End If
28     End Sub

```

```

29
30     Private Sub btnShow_Click(sender As Object, e As EventArgs) Handles btnShow.Click
31         Dim intRepeat As Integer
32         Integer.TryParse(txtNumber.Text, intRepeat)
33
34         If intRepeat = 0 Then                      ' easter egg
35             txtShow.Text = "Nazdaar Bazaaar"
36         ElseIf intRepeat = 666 Then                ' easter egg
37             txtShow.Text = "Your wish for your soul"
38         ElseIf intRepeat <> 0 OrElse intRepeat <> 666 Then
39
40             Do
41                 txtShow.Text = txtShow.Text & lstNames.SelectedItem.ToString & ControlChars.NewLine
42                 intRepeat = intRepeat - 1
43             Loop Until intRepeat = 0
44         End If
45     End Sub
46
47     Private Sub txtNumber_TextChanged(sender As Object, e As EventArgs) Handles txtNumber.TextChanged
48         txtShow.Text = String.Empty
49     End Sub
50
51     Private Sub lstNames_SelectedIndexChanged(sender As Object, e As EventArgs) Handles lstNames.SelectedIndexChanged
52         txtShow.Text = String.Empty
53     End Sub
54 End Class

```

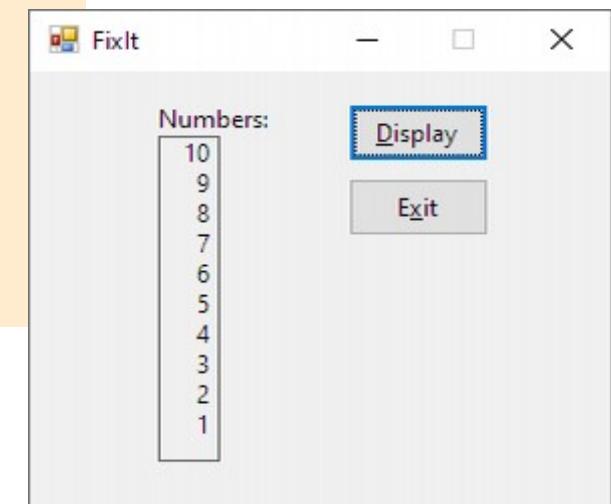


21. Open the FixIt Solution.sln file contained in the VB2017\Chap05\FixIt Solution folder. Open the Code Editor window and review the existing code. Start and then test the application. Correct any errors in the code.

```

1  ' Name:      FixIt Project
2  ' Purpose:    Displays the numbers 10 through 1.
3  ' Programmer: <your name> on <current date>
4
5  Option Explicit On
6  Option Strict On
7  Option Infer Off
8
9  Public Class frmMain
10     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
11         Me.Close()
12     End Sub
13
14     Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
15         ' Displays the numbers 10, 9, 8, 7, 6, 5, 4, 3, 2, and 1.
16         lblNums.Text = Nothing      ' my fix
17         Dim intNum As Integer = 10
18
19         'Do While intNum < 10      ' original, won't display anything
20         Do Until intNum = 0        ' my fix
21             lblNums.Text = lblNums.Text & intNum.ToString & ControlChars.NewLine
22             intNum -= 1
23         Loop
24     End Sub
25 End Class

```



# Event & Sub & Function Procedures

Event-handling Sub Procedure (Event Procedure) - more in detail & multiple objects & multiple events;  
 Independent Sub Procedure (Sub Procedure) + Calling Statement - no parameter & passing variable: by Value - **ByVal** & by Reference - **ByRef**;  
 Function Procedure (Function) + Calling Statement; rounding numbers - method **Math.Round**; ComboBox tool (**cbo**); **frmMain\_FormClosing**;  
**MessageBox.Show** method & return value **DialogBox**

## CH6\_FOCUS ON THE CONCEPTS LESSON

- CH6\_F1 - Event-Handling Sub Procedures aka Event Procedures: more in a detail & multiple objects and events associated
- CH6\_F1.1 - How to connect multiple objects and events to the same procedure: modify the **Monthly Payment application** example + how to break a line of code
- CH6\_F2 - Independent Sub Procedures (1st in code) + Calling Statements (2nd in code):
- CH6\_F2.1 - No Parameter/Arguments example: History Grade Application
- CH6\_F3 - Passing Information to a Procedure & passing a variable by value & passing a variable by reference
- CH6\_F3.1 - Passing Variables by Value example: **Gross Pay Application** -> **ByVal** keyword
- CH6\_F3.2 - You Do It 1: Passing Variables by Value -> **ByVal** keyword
- CH6\_F3.3 - Passing Variables by Reference example: **Concert Tickets Application** -> **ByRef** keyword
- CH6\_F4 - Rounding Numbers - method **Math.Round**
- CH6\_F4.1 - to round the Subtotal and Discount amount example: **Concert Tickets Application** -> **Math.Round** method
- CH6\_F4.2 - You Do It 2: Passing Variables by Address -> **ByRef** keyword
- CH6\_F5 - Function Procedures + Calling Statements, vs event procedures & Sub procedures
- CH6\_F5.1 - Function example: code the **Concert Tickets Application** -> **Function** procedure
- CH6\_F5.2 - Sub procedure vs Function procedure : **Concert Tickets Application** examples to compare
- CH6\_F5.3 - You Do IT 3: using a Function

## CH6\_APPLY THE CONCEPTS LESSON

- CH6\_A1 - Add a Combo box to the form: **ComboBox** tool
- CH6\_A1.1 - to modify the **Monthly Payment application** - replace **ListBox** for a **ComboBox**:
- CH6\_A2 - Add Items to a **ComboBox** and Select a Default Item: Cerruti Company Payroll application example 01
- CH6\_A3 - Code a **Combo Box's KeyPress Event Procedure**: Cerruti Company Payroll application example 02
- CH6\_A4 - Create an Event-Handling Sub Procedure to clear all labels when user input changes: Cerruti Company Payroll application example 03
- CH6\_A5 - Calculate Federal Withholding Tax - **FWT**: Cerruti Company Payroll application example 04
- CH6\_A6 - Invoke an Independent Sub Procedure and a Function (btnCalc\_Click procedure): Cerruti Company Payroll application example 05
- CH6\_A7 - Create an Independent Sub Procedure: **GetSingleFwt** Sub procedure for Single employee: Cerruti Company Payroll application example 06
- CH6\_A8 - Create a Function: **GetMarriedFwt** function for Married employee: Cerruti Company Payroll application example 07
- CH6\_A9 - Validate an Application's Code: Cerruti Company Payroll application example 08
- CH6\_A10 - Professionalize Your Application's Interface: **Message Box** - **MessageBox.Show** method syntax, and its return value: **DialogResult**.
- CH6\_A10.1 - Message Box - **MessageBox.Show** method - confirmation to kill the Cerruti Company Payroll application example 09
- CH6\_A11 - Cerruti Company Payroll application - completed code & GUI with items example 10

## CH6\_Summary

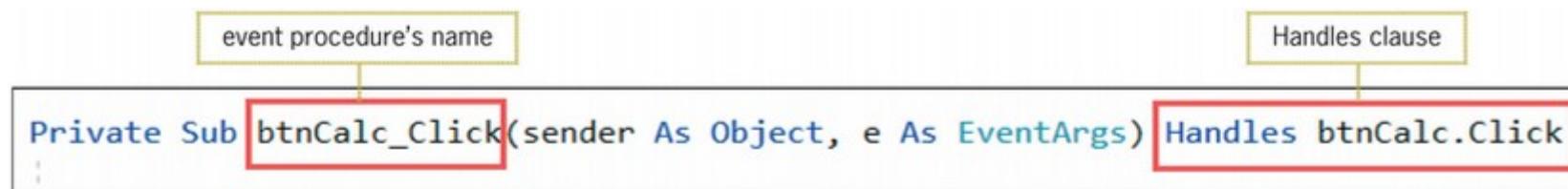
## CH6\_Key Terms

## CH6\_Exercises

## CH6\_FOCUS ON THE CONCEPTS LESSON

### CH6\_F1 - Event-Handling Sub Procedures aka Event Procedures: more in a detail & multiple objects and events associated

- procedures that are processed only when a specific event occurs are called **event-handling Sub procedures**, or more simply, **event procedures**
- all of the procedures that you coded in the previous Chapters were **event procedures**
- the **Handles clause** in an event procedure's header indicates the **object** and **event** associated with the procedure
- the **Handles clause** in **Figure 6-1** indicates that the procedure is associated with the **Click event** of the **btnCalc control** ->
- > as a result, the procedure will be processed when the control's **Click event occurs**



**Figure 6-1** btnCalc\_Click procedure header

- by default, an **event procedure's name** is composed of the object's name followed by an underscore and the event's name -> `btnCalc_Click`
- however, you can change the name to almost anything you like as long as the name follows the same rules for naming variables ->
- > unlike variable names, however, **procedure names** are usually entered using **Pascal case**
- it is a common practice to begin the name with a verb (sloveso)

- **Pascal case** = you capitalize the First letter in the name and the First letter of each subsequent word in the name
  - = subset (podmnožina) of camelCase

e.g.:  
- `userAccount` = camelCase      - used for **variables**  
- `UserAccount` = PascalCase      - used for **Procedure names, Class names, Constructors**

- you can associate a **procedure** with more than one object and **event** as long as each **event** contains the same parameters in its **procedure header**
- as you learned in Chapter 4, a parametr represents information that is passed to the event procedure when the event occurs
- to associate multiple objects and **events** with a procedure, you list each object and event in the procedure's **Handles clause**
- you separate the **object** and **event** with a period, like this: `object.event`
- you separate each `object.event` with a comma, like this: `object.event, object.event, object.event .....`

### CH6\_F1.1 - How to connect multiple objects and **events** to the same **procedure**: modify the **Monthly Payment application** example + how to break a line of code

- how to connect multiple objects and events to the same procedure -> doing this tells the computer to process the procedure when any of the events occur
- in the next set of steps, you will open the **Monthly Payment application** from Chapter 5
- the application's code contains two event procedures that perform the same task ->
  - > `txtPrincipal_TextChanged` and `IstRates_SelectedIndexChanged` perform the same task of clearing the `lblPay.Text` property (čistírny)
- you will change one of the **event procedure's names** and then associate both events with that **procedure**
- when doing this, the **Handles clause** will be rather long and, depending on the size of the font used in your Code Editor window, you might not be able to view the entire statement without scrolling the window ->
- > fortunately, the Code Editor allows you to break a line of code into two or more physical lines as long as the break comes either:

a). before a closing parenthesis )

b). or after one of the following: a comma, an opening parenthesis, an operator

- if you want to break a line of code anywhere else, you will need to use the line continuation character -> an underscore \_, that is immediately preceded (předcházet) by a space, and it must also appear at the end of a physical line of code

1). open the: ...VB2017\Chap06\_Exercise\01.Payment Solution\Payment Solution.sln

2). open the Code Editor window and locate the two procedures: - **txtPrincipal\_TextChanged** and  
- **IstRates\_SelectedIndexChanged**

**notice:** both procedures have the same parameters **sender** and **e** and both perform the same task of clearing the **lblPay.Text** property

- rather than having the same line of code in two procedures, you can create **one procedure** and associate the **two events** with it

3). first, you will change the **txtPrincipal\_TextChanged** procedure's name ->

-> in the procedure's header, change **txtPrincipal\_TextChanged** to: **ClearPay**

4). now, you will use the line continuation character to break the assignment statement before the **Handles clause** ->

-> in the keyword **Handles** click immediately before the letter **H**, and type **\_** (an underscore)

- be sure, there is a space between the ending parenthesis ")" and the underscore "\_"

-> then press **Enter** to move the **Handles** clause to the next line in the procedure

5). the **ClearPay** procedure is already associated with the **TextChanged** event, so you just need to associate it with the **SelectedIndexChanged** event ->

-> modify the **Handles** clause as shown in line **59** (add: ", **IstRates.SelectedIndexChanged**")

```
58     Private Sub ClearPay(sender As Object, e As EventArgs) _
59         Handles txtPrincipal.TextChanged, IstRates.SelectedIndexChanged
60
61         lblPay.Text = String.Empty
62     End Sub
```

6). next, delete the **IstRates\_SelectedIndexChanged** procedure from the Code Editor window (lines **64 -> 66**)

7). save the solution and test:

- click the button **Calculate** -> the Monthly payment box shows: **\$0.00** as the monthly payment

- then, in the Principal box type: **3** -> the **ClearPay** procedure, which is associated with the text box's **TextChanged** event, clears the Monthly payment box

- click the button **Calculate** and then in the list box choose **2.0** -> the **ClearPay** procedure, which is associated with the list box's **SelectedIndexChanged** event, clears the Monthly payment box

8). exit and close all

### Mini-Quiz 6-1

1. What is the line continuation character?

2. Without using the line continuation character, you can break a line of code immediately before a comma. True or False?

3. Write a **Handles clause** that associates a procedure with the **TextChanged events** for the **txtFirst** and **txtSecond** control.

3...Handles txtFirst.TextChanged, txtSecond.TextChanged  
2...False  
1...an underscore \_

## CH6\_F2 - Independent Sub Procedures (1st in code) + Calling Statements (2nd in code):

- procedures that are processed only when a specific event occurs are called **event-handling Sub procedures**, or more simply, **event procedures**
- all of the procedures that you coded in the previous Chapters were **event procedures**
- **independent Sub procedure** = Sub procedure that is not connected to any **object** and **event**
  - is processed only when a statement in your code **calls it** or **invokes it** (dovolávat se, odvolávat se, uplatňovat, použít)
  - does not return a value after performing its task
- many of the reasons that programmers use independent Sub procedures:
  - 1). Avoid duplicating code:**
    - when different sections of a program need to perform the same task, you can enter the code in a procedure and then have each section call the procedure to perform its task when needed
  - 2). Modify in only one place:**
    - if the task performed by an independent Sub procedure subsequently (následně, dodatečně) changes, you need to make the modification in only the procedure rather than in all of the sections that use the procedure
  - 3). Make procedures easier to code and understand:**
    - if an event procedure performs many tasks, you can prevent the procedure's code from getting unwieldy (nepraktický, neohrabaný) and difficult to understand by assigning some of the tasks to one or more independent Sub procedures -> doing this makes the event procedure easier to code because it allows you to concentrate on one small piece of the code at a time
  - 4). Allow a team of programmers to code the application:**
    - independent Sub procedures are used extensively (četně, rozsáhle) in large and complex applications, which typically are written by a team of programmers
    - the team will break up the application's code into small and manageable tasks, and then assign some of the tasks to different team members to be coded as independent Sub procedures -> doing this allows more than one programmer to work on the application at the same time, decreasing the time it takes to complete the application

syntax of an **independent Sub procedure**:

```
Private Sub ProcedureName(parameterList)
    statements...
End Sub
```

- **ProcedureName** -> use **PascalCase**
- **parameterList** = the received items are called **parameters**

syntax of a **calling statement**:

```
ProcedureName(argumentList)
```

- **argumentList** = the passed items are called **arguments**

- you call / invoke an independent Sub procedure using a stand-alone statement, referred to as the **calling statement**, that includes the Sub procedure's name followed by zero or more arguments that are separated by commas and enclosed in parentheses
- the **arguments**:
  - represent information that the statement must pass to the Sub procedure in order for the procedure to perform its task
  - can be a **literal**, a **named constant**, a **keyword**, or like in most cases: a **variable**
- an independent Sub procedure can be entered anywhere between the **Public Class** and **End Class** clauses in the Code Editor window, but it must be entered outside of any other procedure
- in this book, the independent Sub procedures will be entered above the first event procedure

an **independent Sub procedure's autopsy** & a **calling statement** autopsy:

1. most times, the **header** begins with the keyword **Private**, which indicates that the procedure can be used only within the class that defines it
2. the rules for **naming** are the same as those for naming event-handling Sub procedures -> **PascalCase**

-> typically begins with a verb

-> the name should indicate the task the procedure performs -> e.g. a good name for a Sub procedure that calculates an employee's net pay is **CalcNet**

CH6\_F1 - ...

3. following the procedure name in the procedure header is a set of **parentheses** that contains an optional **parameterList**, which lists the data type and name of one or more **parameters**

-> a **parameter** is simply a memory location - more specifically, it is a **variable**

-> each parameter has procedure scope, which means it can be used only by the procedure in whose **parameterList** it appears

-> the variable will be removed from memory when the procedure ends

4. each parameter stores an item of data that it receives from the **calling statement's argumentList**

-> the number of arguments should agree with the number of parameters

- do not be concerned if you **do not understand** everything in the e.g. -> the Sub procedures and calling statements will be explained further in the following sections

e.g. 1: **no parameters/arguments:**

- if the **parameterList** does not contain any parameters, then an empty set of parentheses follows the procedure name in the **calling statement**

```
Private Sub FillListBox()
    lstNumbers.Items.Clear()
    For intNumber = 1 To 10
        lstNumbers.Items.Add(intNumber)
    Next intNumber
End Sub
...
FillListBox()
```

independent Sub procedure

e.g. 2: **one parameter/argument passed by value:**

- if the **parameterList** contains **one** parameter, then the **argumentList** should have **one** argument

```
Private Sub ListSalaries(ByVal intEnd As Integer)
    lstSalaries.Items.Clear()
    For intSalary = 45000 To intEnd Step 5000
        lstSalaries.Items.Add(intSalary)
    Next intSalary
End Sub
...
ListSalaries(intMaxSalary)
    Or
ListSalaries(65000)
```

independent Sub procedure

calling statement

either one of these statements could be used to call the procedure

e.g. 3: **three parameters/arguments -> two passed by value and one passed by reference:**

- similarly like in e.g. 2, a procedure header that contains three parameters, requires three arguments in the calling statement

```
Private Sub CalcNet(ByVal intMoneyIn As Integer, ByVal intMoneyOut As Integer, ByRef intDifference As Integer)
    intDifference = intMoneyIn - intMoneyOut
End Sub
...
CalcNet(intIncome, intExpenses, intNet)
```

independent Sub procedure

calling statement

- in addition to having the same number of arguments as parameters, the data type and order/position of each argument should agree with the data type and order/position of its corresponding parameter -> this is necessary because when the procedure is called, the computer associates the first argument with the first parameter, the second argument with the second parameter, and so on

#### CH6\_F2.1 - No Parameter/Arguments example: History Grade Application

- in this section, you will code an application that creates and calls two independent Sub procedures (neither of the procedures requires any parameters or arguments)
- the **History Grade application** displays a student's grade for either a History 101 course or a History 201 course
- the grade is based on the total points the students earned in the course
- in the next set of steps, you will use two independent Sub procedures to display the appropriate grade

1). open the: ...VB2017\Chap06\Exercise02.History Solution\History Solution.sln

2). open the Code Editor window and locate the **ClearGrade** event-handling procedure

**notice:** that the procedure is associated with the text box's **TextChanged** event and also with the **CheckChanged** events for the two radio buttons:

```
Private Sub ClearGrade(sender As Object, e As EventArgs) Handles txtPoints.TextChanged, radHis101.CheckedChanged, radHis201.CheckedChanged
    lblGrade.Text = String.Empty
End Sub
```

3). scroll to the top of the Code Editor window and then click the blank line immediately below the comment: ' Independent Sub procedures.

-> enter the **DisplayGrade101** procedure: (you will enter the first independent Sub procedure)

```
9  Public Class frmMain
10   ' Independent Sub procedures:
11   Private Sub DisplayGrade101()
12       ' Display the grade for History 101.
13
14       Dim intPoints As Integer
15       Integer.TryParse(txtPoints.Text, intPoints)
16
17       Select Case intPoints
18           Case Is >= 90
19               lblGrade.Text = "A"
20           Case Is >= 80
21               lblGrade.Text = "B"
22           Case Is >= 70
23               lblGrade.Text = "C"
24           Case Is >= 60
25               lblGrade.Text = "D"
26           Case Else
27               lblGrade.Text = "F"
28       End Select
29   End Sub
```

4). click the blank line immediately below the procedure's **End Sub** clause and then press **Enter**

-> enter the DisplayGrade201 procedure: (you will enter the second independent Sub procedure)

```
29      End Sub  
30  
31  Private Sub DisplayGrade201()  
32      ' Display the grade for History 201.  
33  
34      Dim intPoints As Integer  
35      Integer.TryParse(txtPoints.Text, intPoints)  
36  
37      If intPoints >= 75 Then  
38          lblGrade.Text = "P"  
39      Else  
40          lblGrade.Text = "F"  
41      End If  
42  End Sub  
43  
44  Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
```

5). next, you will enter the code to **call** each procedure: (you will enter the call statements for the first and second independent Sub procedure)

-> locate the **btnDisplay\_Click** procedure and then click the blank line immediately above its **End Sub** clause and enter the selection structure from line **47**:

**note:** you can also use **radHis101.Checked = True** as the condition in line **47**

```
44  Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click  
45      ' Calls independent Sub procedures to display a grade.  
46  
47      If radHis101.Checked Then  
48          DisplayGrade101()  
49      Else  
50          DisplayGrade201()  
51      End If  
52  End Sub
```

6). save the solution and test the application:

-> the radio button **History 101** is already selected, so in the text box **Total points** type: **83** and then click the button **Display**

wocogo:

1. the computer processes the **btnDisplay\_Click** procedure, which contains a selection structure:

```
44  Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click  
45      ' Calls independent Sub procedures to display a grade.  
46  
47      If radHis101.Checked Then  
48          DisplayGrade101()  
49      Else
```

```
50      DisplayGrade201()
51  End If
52 End Sub
```

2. the structure's condition evaluates to True, so the computer processes the statement **DisplayGrade101()** entered in the True path

```
11  Private Sub DisplayGrade101()
12    ' Display the grade for History 101.
13
14    Dim intPoints As Integer
15    Integer.TryParse(txtPoints.Text, intPoints)
16
17    Select Case intPoints
18      Case Is >= 90
19        lblGrade.Text = "A"
20      Case Is >= 80
21        lblGrade.Text = "B"
22      Case Is >= 70
23        lblGrade.Text = "C"
24      Case Is >= 60
25        lblGrade.Text = "D"
26      Case Else
27        lblGrade.Text = "F"
28    End Select
29 End Sub
```

- at this point, the computer temporarily leaves the procedure **btnDisplay\_Click** to process the code in the procedure **DisplayGrade101** ->  
-> that procedure displays the letter **B** in the **Grade** box, and the computer processes to the procedure's **End Sub** clause, which ends the procedure and ->  
-> the computer then returns to the **btnDisplay\_Click** procedure to finish processing its code  
3. in this case, only remaining to be processed is the **btnDisplay\_Click** procedure's **End Sub** clause, which ends the procedure

-> click the radio button **History 201** and then click the button **Display**

wocogo:

1. the computer processes the **btnDisplay\_Click** procedure, which contains a selection structure

```
44  Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
45    ' Calls independent Sub procedures to display a grade.
46
47    If radHis101.Checked Then
48      DisplayGrade101()
49    Else
50      DisplayGrade201()
51    End If
52 End Sub
```

2. in this case, the selection structure's condition evaluates to False, so the computer processes the statement **DisplayGrade201()** entered in the False path

```
31  Private Sub DisplayGrade201()
```

```

32     ' Display the grade for History 201.
33
34     Dim intPoints As Integer
35     Integer.TryParse(txtPoints.Text, intPoints)
36
37     If intPoints >= 75 Then
38         lblGrade.Text = "P"
39     Else
40         lblGrade.Text = "F"
41     End If
42 End Sub

```

- here too, the computer temporarily leaves the procedure **btnDisplay\_Click**, but this time to **process** the code in the procedure **DisplayGrade201** ->
- > that procedure displays the letter **P** in the **Grade box**, and the computer processes to the procedure's **End Sub** clause, which ends the procedure and ->
- > the computer then returns to the **btnDisplay\_Click** procedure to finish processing its code

3. in this case, only remaining to be processed is the **btnDisplay\_Click** procedure's **End Sub** clause, which ends the procedure

7). on your own, continue testing the application using various total points for each History course, and when you are finished testing, click the button **Exit**

### Mini-Quiz 6-2

1. What is an item within parentheses () in a procedure header called?
2. If the first parameter has the **Integer** data type and the second parameter has the **Double** data type, what data type should the first and second arguments have?
3. Write a statement that invokes the **DisplayCompanyName** procedure, which has no parameters.

3...DisplayCompanyName().

2...same like parameters: 1st argument: Integer, 2nd argument: Double.

1...a (parameter).

### CH6\_F3 - Passing Information to a Procedure & passing a variable **by value** & passing a variable **by reference**

- as mentioned earlier, information is **passed** to a procedure through the **calling statement's** (**argumentList**)
- an **argument** can be a **literal**, a **named constant**, a **keyword**, or like in most cases: a **variable**
- each **variable** has both ->
  - a **value** and
  - a **unique address** that represents its location in the RAM
- Visual Basic allows you to **pass** to the **receiving procedure** either ->
 

|                                                                     |                                                            |                                  |
|---------------------------------------------------------------------|------------------------------------------------------------|----------------------------------|
| a). a copy of the variable's value or<br>b). the variable's address | = <b>passing by value</b><br>= <b>passing by reference</b> | <b>= ByVal</b><br><b>= ByRef</b> |
|---------------------------------------------------------------------|------------------------------------------------------------|----------------------------------|
- the method you choose - **by value** or **by reference** - depends on whether you want the receiving procedure to have access to the variable in memory ->
  - > keep in mind that when you give the receiving procedure access to the variable, it can change the variable's contents (**ByRef**)
- although the idea of passing information **by value** and **by reference** may sound confusing at first, it is a concept with which you are already familiar ??????????

#### a) passing a variable **by value** -> **ByVal**, where can NOT change the contents:

- to pass a variable **by value**, you include the keyword **ByVal** before the name of its corresponding parameter in the **receiving procedure's parameterList**
- when you pass a variable **by value**, the computer passes a **copy** of the variable's contents to the **receiving procedure** ->
- > when only a copy of the contents is passed, the **receiving procedure** is **not** given access to the variable in memory -> therefore, it **cannot change** the value stored inside the variable
- you should pass a variable **by value** when the **receiving procedure** needs to **know** the variable's contents but does **not** need to **change** the contents

### b) passing a variable by reference/address in RAM -> **ByRef**, where can change the contents:

- instead of passing a copy of a variable's value to a procedure, you can pass the variable's **address**, which is its location in RAM
- as you learned earlier, passing a variable's address is referred to as passing by reference -> **ByRef** keyword
- it gives the receiving procedure **access** to the variable being passed
- you pass a variable **ByRef** when you want the receiving procedure to change the contents of the variable
- to pass a variable by reference in VB, you include the keyword **ByRef** before the name of the corresponding parameter in the receiving procedure's header
- the **ByRef** keyword tells the calling statement to pass the variable's address rather than a copy of its contents

### CH6 F3.1 - Passing Variables **by Value** example: **Gross Pay Application** -> **ByVal** keyword

- in the next set of steps, you will complete the **Gross Pay application**, which uses two independent Sub procedures to display the gross pay for employees who are paid either weekly or twice per month
- employees who are paid weekly receive 52 paychecks per year and employees who are paid twice per month receive 24 paychecks per year
- both procedures will contain one parameter in their **parameterList**, and the parameters will receive information that is passed **by value**
- for the procedures to perform their assigned tasks, they will need their calling statements to pass them the employee's annual salary

1). open: ...VB2017\Chap06\Exercise03.Gross Solution\Gross Solution.sln

2). open the Code Editor window and locate the **ClearGross** event-handling procedure

notice: that the procedure is associated with the text box's **TextChanged** event and also with the **CheckChanged** events for the two radio buttons:

```
Private Sub ClearGross(sender As Object, e As EventArgs) Handles txtSalary.TextChanged, radWeekly.CheckedChanged, radTwicePerMonth.CheckedChanged
    lblGross.Text = String.Empty
End Sub
```

3). scroll to the top of the Code Editor window and then click the blank line immediately below the ' **Independent Sub procedure**

-> and enter the two procedures:

```
9   Public Class frmMain
10  ' Independent Sub procedures:
11  Private Sub DisplayWeekly(ByVal intWeekly As Integer)
12      ' Display the weekly gross pay.
13
14      Dim dblPay As Double
15      dblPay = intWeekly / 52
16      lblGross.Text = dblPay.ToString("C2")
17  End Sub
18
19  Private Sub DisplayTwicePerMonth(ByVal intTwicePerMonth As Integer)
20      ' Display the twice per month gross pay.
21
22      Dim dblPay As Double
23      dblPay = intTwicePerMonth / 24
24      lblGross.Text = dblPay.ToString("C2")
25  End Sub
```

procedure 1 - DisplayWeekly

procedure 2 - DisplayTwicePerMonth

4). insert a blank line immediately below the line 25 (prostě vodentruj End Sub)

5). next, you will enter the code to **call** each procedure:

-> locate the **btnCalc\_Click** procedure, click the blank line immediately above its **End Sub** clause and enter the code:

**note:** you can also use **radWeekly.Checked = True** as the condition in line 33

```
27      Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
28          ' Calls independent Sub procedures to calculate and display the gross pay.
29
30          Dim intSalary As Integer
31
32          Integer.TryParse(txtSalary.Text, intSalary)
33          If radWeekly.Checked Then
34              DisplayWeekly(intSalary)
35          Else
36              DisplayTwicePerMonth(intSalary)
37          End If
38      End Sub
```

6). save and test:

-> the radio button **Weekly** is already selected, so in the text box **Salary:** type: **42500** and then click the button **Calculate**

wocogo:

1. the statement **DisplayWeekly(intSalary)** in the procedure **btnCalc\_Click** invokes the procedure **DisplayWeekly**, passing it a copy of the value **42500** stored in the variable **intSalary**
2. the procedure **DisplayWeekly** stores the value passed to it in its **intWeekly parameter**
3. the procedure then calculates and displays the weekly gross pay -> **\$817.31**

-> click the radio button **Twice per month** and then click the button **Calculate**

wocogo:

1. the statement **DisplayTwicePerMonth(intSalary)** in the procedure **btnCalc\_Click** invokes the procedure **DisplayTwicePerMonth**, passing it a copy of the value **42500** stored in the variable **intSalary**
2. the procedure **DisplayTwicePerMonth** stores the value passed to it in its **intTwicePerMonth parameter**
3. the procedure then calculates and displays the gross pay for an employee who is paid twice per month -> **\$1,770.83**

7). exit and then close the solution

**autopsy:** two procedure headers & calling statements from Gross Pay application:

1. notice that the data type of the **argument** in each **calling statement** **matches** the data type of its corresponding **parameter** in the **procedure header**
2. notice that the **argument** names do **not** need to be identical to the **parameter** names -> in fact, to avoid confusion, you should **use** different names for an **argument** and its corresponding **parameter**
3. notice that the **calling statement** does **not** indicate whether a variable is being passed **ByVal** or **ByRef** -> to make that determination, you need to look at the **receiving procedure's header** -> line **11** & line **19**

```
9      Public Class frmMain
10          ' Independent Sub procedures:
11          Private Sub DisplayWeekly(ByVal intWeekly As Integer)
12              ' Display the weekly gross pay.
```

<- receiving procedure by value with parameter **intWeekly**

```

13
14      Dim dblPay As Double
15      dblPay = intWeekly / 52
16      lblGross.Text = dblPay.ToString("C2")
17  End Sub
18
19  Private Sub DisplayTwicePerMonth(ByVal intTwicePerMonth As Integer)      <- receiving procedure by value with parameter intTwicePerMonth
20      ' Display the twice per month gross pay.
21
22      Dim dblPay As Double
23      dblPay = intTwicePerMonth / 24
24      lblGross.Text = dblPay.ToString("C2")
25  End Sub
26
27  Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
28      ' Calls independent Sub procedures to calculate and display the gross pay.
29
30      Dim intSalary As Integer
31
32      Integer.TryParse(txtSalary.Text, intSalary)
33      If radWeekly.Checked Then
34          DisplayWeekly(intSalary)           <- calling statement with (argument)    procedure 1 - DisplayWeekly
35      Else
36          DisplayTwicePerMonth(intSalary)   <- calling statement with (argument)    procedure 2 - DisplayTwicePerMonth
37      End If
38  End Sub

```

### CH6\_F3.2 - You Do It 1: Passing Variables by Value -> **ByVal** keyword

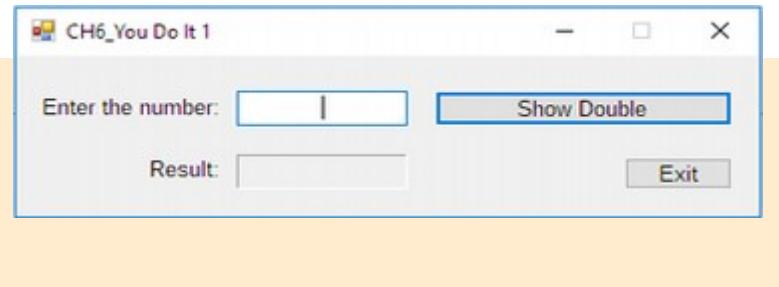
1. create an application named **You Do It 1** and save it in the **VB2017\Chap06\Exercisel04.You Do It 1 Solution**
2. add a text box, a label, and a button to the form
3. the button's **Click** event procedure should assign the text box value to a Double variable and then pass a copy of the variable's value to an independent Sub procedure named **ShowDouble**
4. the **ShowDouble** procedure should multiply the value it receives by 2 and then display the result in the label control
5. code the button's **Click** event procedure and the **ShowDouble** procedure
6. save, start, and test your application

#### - my design & code:

```

1  Option Explicit On
2  Option Strict On
3  Option Infer Off
4
5  Public Class frmMain
6      Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
7          Me.Close()

```



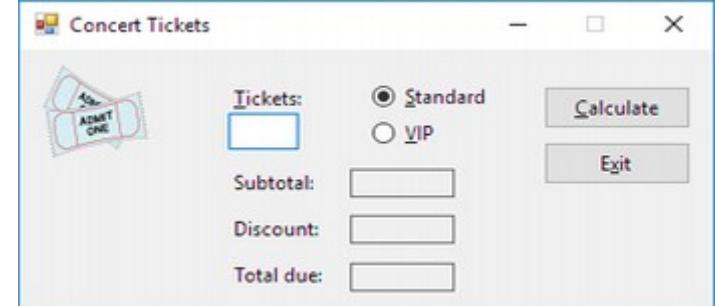
```

8      End Sub
9
10     Private Sub txtEnterTheNumber_Enter(sender As Object, e As EventArgs) Handles txtEnterTheNumber.Enter
11         txtEnterTheNumber.SelectAll()
12     End Sub
13
14     Private Sub txtEnterTheNumber_TextChanged(sender As Object, e As EventArgs) Handles txtEnterTheNumber.TextChanged
15         lblResult.Text = String.Empty
16     End Sub
17
18     Private Sub txtEnterTheNumber_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtEnterTheNumber.KeyPress
19         If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
20             e.Handled = True
21         End If
22     End Sub
23
24     Private Sub ShowDouble(ByVal dblNumber As Double) <- receiving procedure by value with parameter dblNumber
25         Dim dblDouble As Double
26         dblDouble = dblNumber * 2
27         lblResult.Text = dblDouble.ToString("N0")
28     End Sub
29
30     Private Sub btnShow_Click(sender As Object, e As EventArgs) Handles btnShow.Click
31         Dim dblEnterTheNumber As Double
32         Double.TryParse(txtEnterTheNumber.Text, dblEnterTheNumber)
33         ShowDouble(dblEnterTheNumber) <- calling statement with (argument)
34     End Sub
35 End Class

```

### CH6\_F3.3 - Passing Variables by Reference example: Concert Tickets Application -> **ByRef** keyword

- in this section, you will complete the **Concert Tickets application**, which uses an independent Sub procedure named **CalcDiscount** to calculate a customer's discount when purchasing specific numbers of tickets
- in order to calculate the discount, the procedure will need to have its calling statement pass it two values:
  - 1. the number of tickets purchased and
  - 2. the cost of purchasing the tickets before any discount is applied
- it will also need the calling statement to pass it the address of a memory location that it can use to store the discount
- in order to accept the information passed to it, the procedure will need to have 3 **parameters** in its **parameterList**:
  - > the first 2 **parameters** will tell the calling statement to pass its first 2 **arguments by value** (just a copy of a variable, can NOT change its value)
  - > the third **parameter** will tell the calling statement to pass its third **argument by reference** (address in RAM, can change its value)



1).

open: ...VB2017\Chap06\Exercise05.Concert Solution-Sub\Concert Solution.sln

2). open the Code Editor window and locate the **ClearLabels** event-handling procedure

**notice:** that the procedure is associated with the text box's TextChanged event and also with the each radio button's CheckChanged event:

```
Private Sub ClearLabels(sender As Object, e As EventArgs) Handles txtTickets.TextChanged, radStandard.CheckedChanged, radVIP.CheckedChanged
    lblSubtotal.Text = String.Empty
    lblDiscount.Text = String.Empty
    lblTotalDue.Text = String.Empty
End Sub
```

3). locate the procedure **btnCalc\_Click** -> missing from the procedure is the statement that invokes the **CalcDiscount** independent Sub procedure

- but before entering the calling statement, you will enter the **CalcDiscount** procedure's code:

-> scroll to the top of the Code Editor window and then click the blank line immediately below the comment: ' **Independent Sub procedure**

-> and enter the procedure CalcDiscount:

```
9  Public Class frmMain
10   ' Independent Sub procedures:
11   Private Sub CalcDiscount(ByVal intNum As Integer,
12                           ByVal dblBeforeDiscount As Double,
13                           ByRef dblDisc As Double)
14     Select Case intNum
15       Case Is >= 10
16         dblDisc = dblBeforeDiscount * 0.1
17       Case Is >= 5
18         dblDisc = dblBeforeDiscount * 0.05
19       Case Else
20         dblDisc = 0
21     End Select
22   End Sub
```

wocogo:

<- 1st parameter - the number of tickets  
<- 2nd parameter - the cost of a ticket before any discount  
<- 3rd parameter - address of a variable where the discount is stored  
  
<- 10 or more tickets  
<- 10% discount  
<- 5 or more tickets  
<- 5% discount  
<- less than 5 tickets  
<- 0% discount

wocogo:

- the procedure gives a 10% discount for purchases of at least 10 tickets, a 5% discount for purchases of 5 through 9 tickets, and no discount for purchases of less than 5 tickets
- the first parameter will accept value that represent the number of tickets, the second parameter represent the cost of a ticket before any discount, and the third parameter will accept the address of a variable where the discount can be stored

4). now, click the blank line below the comment ' **Use a procedure to calculate the discount** in the procedure **btnCalc\_Click**

- the number of tickets is stored in the variable **intTickets**

- the cost of the tickets before any discount is stored in the variable **dblSubtotal**

- the procedure **btnCalc\_Click** needs the procedure **CalcDiscount** to store the discount in the variable **dblDiscount**

-> type the following calling statement and then press Enter:

```
30   Dim intTickets As Integer
31   Dim dblSubtotal As Double
32   Dim dblDiscount As Double
```

```

33     Dim dblTotalDue As Double
...
44     ' Use a procedure to calculate the discount.
45     CalcDiscount(intTickets, dblSubtotal, dblDiscount)

```

#### wocogo:

<- the variable **intTickets** **lends** its value to **intNum** (read only),  
 in the independent Sub procedure **CalcDiscount** -> **ByVal** keyword  
 <- the variable **dblSubtotal** **lends** its value to **dblBeforeDiscount** (read only),  
 in the independent Sub procedure **CalcDiscount** -> **ByVal** keyword  
 <- the variable **dblDiscount** **gives** its address to **dblDisc** (read & write),  
 and it can change the value of **dblDiscount**  
 in the **CalcDiscount** independent Sub procedure -> **ByRef** keyword

5). save the solution and then test the application;

1. -> in the **Tickets:** box type: **10**, the radio button is chosen to **Standard** and then click the button **Calculate**:

- the Subtotal = 625.00, the Discount = 62.50, and the Total Due = 562.50.

2. -> change the number of tickets to **2**, click the radio button **VIP** and then click the button **Calculate**:

- the Subtotal = 205.50, the Discount = 0.00, and the Total Due = 205.50.

3. -> change the number of tickets to **7**, click the radio button **Standard** and then click the button **Calculate**:

- the Subtotal = 437.50, the Discount = 21.88, and the Total Due = **415.63** <- notice that the total due amount is off by **1** penny -> it should be **415.62**

-> it is a result of rounding performed by the **ToString** method -> you will learn how to fix this problem in the next section:

**CH6\_F4 - Rounding...**

6). click the button **Exit**

#### autopsy: procedure header and calling statement from Concert Tickets application:

- notice that the number, data type, and order (position) of the **arguments** in the **calling statement** **matches** the number, data type, and order (position) of the **parameters** in the **procedure** header
- notice that the names of the **arguments** are **NOT** identical to the names of their corresponding **parameters** -> as mentioned earlier, it is best to use **different** names for an **argument** and its **parameter**
- notice that the **calling statement** does **not** indicate whether a variable is being passed **ByVal** or **ByRef** -> to make that determination, you need to look at the **receiving procedure's** header -> line **11** & line **12** & line **13**

```

9   Public Class frmMain
10    ' Independent Sub procedures:
11    Private Sub CalcDiscount(ByVal intNum As Integer,
12                           ByVal dblBeforeDiscount As Double,
13                           ByRef dblDisc As Double)
...
30    Dim intTickets As Integer
31    Dim dblSubtotal As Double
32    Dim dblDiscount As Double
33    Dim dblTotalDue As Double
...
44    ' Use a procedure to calculate the discount.
45    CalcDiscount(intTickets, dblSubtotal, dblDiscount)

```

#### wocogo:

<- independent Sub procedure (receiving procedure)  
 <- **ByVal** indicates, that the **parameter** is receiving a **copy** of a value  
 <- **ByVal** indicates, that the **parameter** is receiving a **copy** of a value  
 <- **ByRef** indicates, that the **parameter** is receiving an **address** of a variable

#### <- calling statement

<- the variable **intTickets** **lends** its value to **intNum** (read only),  
 in the independent Sub procedure **CalcDiscount** -> **ByVal** keyword  
 <- the variable **dblSubtotal** **lends** its value to **dblBeforeDiscount** (read only),  
 in the independent Sub procedure **CalcDiscount** -> **ByVal** keyword

<- the variable **dblDiscount** **gives** its address to **dblDisc** (read & write),  
and it can change the value of **dblDiscount**  
in the **CalcDiscount** independent Sub procedure -> **ByRef** keyword

## CH6\_F4 - Rounding Numbers - method **Math.Round**:

- e.g.: VB2017\Chap06\Exercise\05.Concert Solution-Sub\Concert Solution.sln

...  
5). save the solution and then test the application;

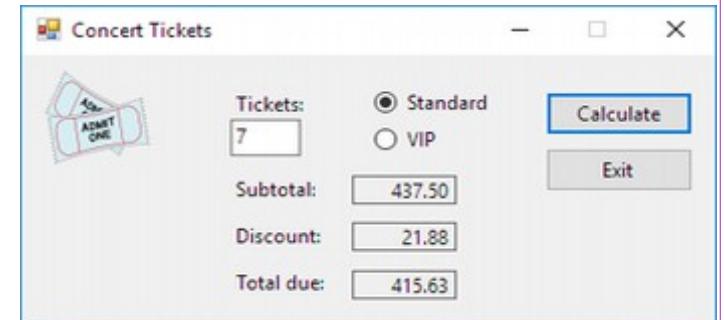
...  
3. -> change the number of tickets to 7, click the radio button **Standard** and then click the button **Calculate**:

- the Subtotal = 437.50, the Discount = 21.88, and the Total Due = **415.63** <- notice that the total due amount is off by 1 penny -> it should be **415.62**  
-> it is a result of rounding performed by the **ToString** method -> you will learn how to fix this problem in the next section:

**CH6\_F4 - Rounding...**

- the penny-off error shown in this example is the result of the  
rounding performed by the **ToString** method:

| Calculation:                | result stored<br>in RAM: | displayed by the<br><b>ToString</b> method: |
|-----------------------------|--------------------------|---------------------------------------------|
| Subtotal: $7 * 62.50$       | 437.5                    | 437.50                                      |
| Discount: $437.5 * 0.05$    | <b>21.875</b>            | <b>21.88</b>                                |
| Total due: $437.5 - 21.875$ | <b>415.625</b>           | <b>415.63</b>                               |



- to fix the penny-off error, you will use VB's method: **Math.Round()** to round the Subtotal and Discount amounts before they are used in the Total due calculation

syntax:

**Math.Round(value, digits)**

**value** = numeric expression

**digits** = optional, integer indicating decimal places

- if omitted, the method returns an integer

e.g.:

**Math.Round(3.235, 2)**      the result = **3.24**

**Math.Round(6.517, 1)**      the result = **6.5**

**Math.Round(8.99)**      the result = **9**

## CH6\_F4.1 - to round the Subtotal and Discount amount example: Concert Tickets Application -> **Math.Round** method

1). open: ...VB2017\Chap06\Exercise\05.Concert Solution-Sub\Concert Solution.sln

2). open the Code Editor window and locate the procedure **btnCalc\_Click**

-> enter the two assignment statements just before the actual calculation:

```

25      Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
26          ' Display subtotal, discount, and total due.
...
44          ' Use a procedure to calculate the discount.
45          CalcDiscount(intTickets, dblSubtotal, dblDiscount)

```

```

46     ' Calculate the total due.
47     dblSubtotal = Math.Round(dblSubtotal, 2)
48     dblDiscount = Math.Round(dblDiscount, 2)
49     dblTotalDue = dblSubtotal - dblDiscount
50

```

3). save the solution and test the application:

-> in the **Tickets:** box type 7, click the radio button **Standard** and then click the button **Calculate**:

- the Subtotal = 437.50, the Discount = 21.88, and the Total Due = **415.62** <- notice that the total due box shows the correct amount

4). click the button **Exit**, close the Code Editor window and then close the solution

### Mini-Quiz 6-3

1. Write the **parameterList** for a procedure that receives a Decimal value followed by the address of a Decimal variable.  
Use **decSales** and **decBonus** as the parameter names.
2. Write a calling statement that invokes an independent Sub procedure named **CalcBonus**, passing it the value stored in the **decFebSales** variable and the address of the **decFebBonus** variable.
3. Write a statement that rounds the value stored in the **dblRate** variable to one decimal place and then assigns the result to the variable.

```

1...ByVal decSales As Decimal, ByRef decBonus As Decimal
2...CalcBonus(decSales, decBonus)
3...dblRate = Math.Round(dblRate, 1)

```

### CH6\_F4.2 - You Do It 2: Passing Variables by Address -> **ByRef** keyword

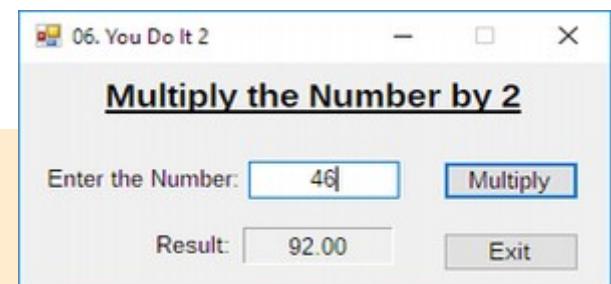
1. create an application named **You Do It 2** and save it in the **VB2017\Chap06\Exercise06.You Do It 2 Solution**
2. add a text box, a label, and a button to the form
3. the button's **Click** event procedure should assign the text box value to a Double variable
4. it should then invoke an independent Sub procedure named **CalcDouble**, passing the procedure the Double variable's address
5. the **CalcDouble** procedure should multiply the contents of the Double variable \* 2
6. after the **CalcDouble** procedure ends, the button's **Click** event procedure should display the contents of the Double variable in the label
7. code the button's **Click** event procedure and the **CalcDouble** procedure
8. save, start, and test your application

- my design & code:

```

1  Option Explicit On
2  Option Strict On
3  Option Infer Off
4
5  Public Class frmMain
6      Private Sub CalcDouble(ByVal dblNumber2 As Double, ByRef dblResult2 As Double)
7          dblResult2 = dblNumber2 * 2
8      End Sub
9      Private Sub btnMultiply_Click(sender As Object, e As EventArgs) Handles btnMultiply.Click
10         Dim dblNumber As Double
11         Dim dblResult As Double
12         Double.TryParse(txtEnterTheNumber.Text, dblNumber)

```



<- receiving procedure by address

```

13
14     CalcDouble(db1Number, db1Result)                                     <- calling statement with (arguments)
15
16     lb1Result.Text = db1Result.ToString("N")
17 End Sub
18
19 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
20     Me.Close()
21 End Sub
22
23 Private Sub txtEnterTheNumber_Enter(sender As Object, e As EventArgs) Handles txtEnterTheNumber.Enter
24     txtEnterTheNumber.SelectAll()
25 End Sub
26
27 Private Sub txtEnterTheNumber_TextChanged(sender As Object, e As EventArgs) Handles txtEnterTheNumber.TextChanged
28     lb1Result.Text = String.Empty
29 End Sub
30
31 Private Sub txtEnterTheNumber_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtEnterTheNumber.KeyPress
32     If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
33         e.Handled = True
34     End If
35 End Sub
36 End Class

```

#### CH6\_F5 - Function Procedures + Calling Statements, vs event procedures & Sub procedures:

- in addition to creating **Event procedures** and **Sub procedures** in VB, you can also create **Function procedures**

##### 1. event-handling Sub procedure

- aka **event procedure**
- is processed only when a **specific event occurs** (like **Click**, **TextChanged**, etc...)
- all of the procedures that you coded in the previous Chapters were **event procedures**
- e.g.:    **Private Sub btnExit\_Click...**

##### 2. independent Sub procedure

- aka **Sub procedure**
- = Sub procedure that is **not** connected to any **object** and **event**
- does **not** return a value after performing its assigned task
- is processed only when a **calling statement** in your code **calls it** or **invokes it**:

  1. receiving procedure syntax:    **Private Sub ProcedureName(parameterList)**
  2. calling statement syntax        **ProcedureName(argumentList)**

- e.g.:

```

6      Private Sub CalcDouble(ByVal db1Number2 As Double, ByRef db1Result2 As Double)
7          db1Result2 = db1Number2 * 2
8      End Sub
...
14      CalcDouble(db1Number, db1Result)

```

### 3. function procedure - aka **function**

- return a value after performing its assigned task

syntax:

```
Private Function ProcedureName(parameterList) As dataType  
    statements...  
    Return expression  
End Function
```

- **ProcedureName** -> use **PascalCase**
- **parameterList** = parameters (**ByVal**, **ByRef**)
- **As dataType** = data type of the return value

e.g.1: 

```
Private Function GetNewPay(ByVal dblOld As Double) As Double  
    ' Increases current pay by 2% and returns new pay.  
    Dim dblNew As Double  
    dblNew = dblOld * 1.02  
    Return dblNew  
End Function
```

<- returns the **dblNew** variable's value to the statement that invoked the function

e.g.2: 

```
Private Function GetNewPay(ByVal dblOld As Double) As Double  
    ' Increases current pay by 2% and returns new pay.  
    Return dblOld * 1.02  
End Function
```

<- calculates and returns the new pay to the statement that invoked the function

- unlike a **Sub procedure**, a **function**'s header and footer contain the keyword **Function**, rather than the keyword **Sub**
- a **function**'s header also includes the **As dataType** section, which specifies the data type of the value the **function** will return
- like a **Sub procedure**, a function can receive information passed to it either **by value**, or **by reference** (**ByVal**, **ByRef**)
- the information it receives is listed in its **parameterList**
- as are **Sub procedures**, **functions** are entered anywhere between the **Public Class** and **End Class** clauses in the Code Editor window, but it must be entered outside of any other procedure
- in this book, the **functions** will be entered above the first event procedure, same like **independent Sub procedures**
- like **Sub procedure** names, **function** names are entered using **PascalCase** and typically begin with a verb
- the name should indicate the task the **function** performs -> e.g. the **GetNewPay** name used in the example indicates that each **function** returns a new pay amount
- the value is returned by the **Return statement**, which typically is the last statement within a **function**
  - syntax: **Return expression** - **expression** = represents the one and only value that will be returned to the statement that invoked the **function**
    - the data type must agree with the data type specified in the **As dataType** section of the header
- you call / invoke a **function** by including the **function's name** and **arguments** (if any) in a statement <- like **Sub procedure**
- the number, data type, and position of the arguments should agree with the number, data type, and position of the **function's parameters** <- like **Sub procedure**
- in most cases, the statement that invokes a **function** assigns the **function's return value** to a variable - however, it also may use the return value in a calculation or simply display the return value

e.g.1: assigns the return value to a variable: **dblNewPay = GetNewPay(dblPay)**  
or  
**dblPay = GetNewPay(dblPay)**

e.g.2: uses the return value in a calculation: `dblNewWeekly = GetNewPay(db1Pay) * 40`

e.g.3: displays the return value: `lblNewPay.Text = GetNewPay(db1Pay).ToString("C2")`

#### CH6\_F5.1 - Function example: code the Concert Tickets Application -> Function procedure

- in the next set of steps, you will code the Concert Tickets application from the previous section using a function named **GetDiscount**, (rather than a Sub procedure named **CalcDiscount**) to determine the appropriate discount amount

1). open: ...VB2017\Chap06\Exercise07.Concert Solution-Function\Concert Solution.sln

2). locate the procedure **btnCalc\_Click** -> missing from the procedure is the statement that invokes the **GetDiscount** function

- for the function to perform its task, it needs to know the number of tickets purchased and the cost of purchasing the tickets before any discount is applied
- those values are stored in the **intTickets** and **dblSubtotal** variables
- the function will not need to change the values stored in those variables, so you will pass the variables **by value** -> **ByVal**
- the **btnCalc\_Click** procedure will assign the function's **return** value to the **dblDiscount** variable

-> click the blank line below the comment: '**Use a function to get the discount.**'

-> and type the following assignment statement and press Enter:

```
32      ' Use a function to get the discount.  
33      dblDiscount = GetDiscount(intTickets, dblSubtotal)
```

3). now, you will enter the **GetDiscount** function's code:

-> scroll to the top of the Code Editor window and then click the blank line below the comment: '**Function procedure.**'

-> enter the **GetDiscount** function:

```
9      Public Class frmMain  
10     ' Function procedure.  
11     Private Function GetDiscount(ByVal intNum As Integer,  
12                               ByVal dblBeforeDiscount As Double) As Double  
13     Dim dblDisc As Double  
14     Select Case intNum  
15       Case Is >= 10  
16         dblDisc = dblBeforeDiscount * 0.1  
17       Case Is >= 5  
18         dblDisc = dblBeforeDiscount * 0.05  
19       Case Else  
20         dblDisc = 0  
21     End Select  
22  
23     Return dblDisc  
24 End Function
```

- the function's parameters will accept values that represent the number of tickets and the cost of the tickets before any discount

- notice that, unlike the **CalcDiscount** Sub procedure, the **GetDiscount** function does not need the calling statement to pass it the address of a variable in which to store the discount -> this is because the function returns the discount to the statement that invoked it, which is the:

`dblDiscount = GetDiscount(intTickets, dblSubtotal)` assignment statement in the **btnCalc\_Click** procedure ->

-> that assignment statement assigns the **function's return** value to the **dblDiscount** variable

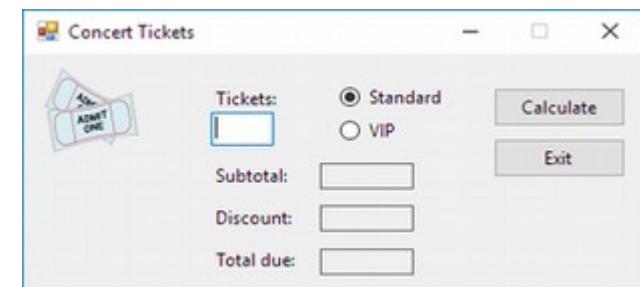
4). save the solution and then start and test the application:

-> type **10** in the **Ticket** box and then click the button **Calculate**:  
- the Subtotal = 625.00, Discount = 62.50, Total due = 562.50

-> change the number of tickets to **2**, choose the radio button **VIP** and then click the button **Calculate**:  
- the Subtotal = 205.50, Discount = 0.00, Total due = 205.50

-> change the number of tickets to **7**, choose the radio button **Standard** and then click the button **Calculate**:  
- the Subtotal = 437.50, Discount = 21.88, Total due = 415.62

5). click the **Exit** button, close the Code Editor window and then close the solution.



-->

CH6\_F5.2 - **Sub** procedure vs **Function** procedure : **Concert Tickets Application** examples to compare:

CH6\_F3.3 - ... ...05.Concert Solution-Sub\Concert Solution.sln

```
9  Public Class frmMain
10   ' Independent Sub procedures:
11   Private Sub CalcDiscount(ByVal intNum As Integer,
12                           ByVal dblBeforeDiscount As Double,
13                           ByRef dblDisc As Double)
14     Select Case intNum
15       Case Is >= 10
16         dblDisc = dblBeforeDiscount * 0.1
17       Case Is >= 5
18         dblDisc = dblBeforeDiscount * 0.05
19       Case Else
20         dblDisc = 0
21     End Select
22   End Sub
...
44   ' Use a procedure to calculate the discount.
45   CalcDiscount(intTickets, dblSubtotal, dblDiscount)
46   ' Calculate the total due.
47   dblSubtotal = Math.Round(dblSubtotal, 2)
48   dblDiscount = Math.Round(dblDiscount, 2)
49   dblTotalDue = dblSubtotal - dblDiscount
```

CH6\_F5.1 - ... ...07.Concert Solution-Function\Concert Solution.sln

```
9  Public Class frmMain
10   ' Function procedure.
11   Private Function GetDiscount(ByVal intNum As Integer,
12                               ByVal dblBeforeDiscount As Double) As Double
13     Dim dblDisc As Double
14     Select Case intNum
15       Case Is >= 10
16         dblDisc = dblBeforeDiscount * 0.1
17       Case Is >= 5
18         dblDisc = dblBeforeDiscount * 0.05
19       Case Else
20         dblDisc = 0
21     End Select
22
23   Return dblDisc
24 End Function
...
45   ' Use a function to get the discount.
46   dblDiscount = GetDiscount(intTickets, dblSubtotal)
47
48   ' Calculate the total due.
49   dblSubtotal = Math.Round(dblSubtotal, 2)
50   dblDiscount = Math.Round(dblDiscount, 2)
51   dblTotalDue = dblSubtotal - dblDiscount
```

- notice that, unlike the **CalcDiscount** Sub procedure, the **GetDiscount** function does not need the calling statement to pass it the address of a variable in which to store the discount -> this is because the function returns the discount to the statement that invoked it, which is the:

**dblDiscount = GetDiscount(intTickets, dblSubtotal)** assignment statement in the **btnCalc\_Click** procedure ->

-> that assignment statement assigns the **function's return** value to the **dblDiscount** variable

## Mini-Quiz 6-4

1. A function named **GetBonus** receives a **Double** value and returns a **Double** value. Write the **function** header, using **dblSales** as the parameter name.
2. Write the **Return statement** for the **GetBonus** **function** from Question 1. The statement should return the value stored in its **dblBonus** variable.
3. Write a statement that invokes the **GetBonus** **function** from Question 1, passing it the value stored in the **dblFebSales** variable. The statement should assign the return value to the **dblFebBonus** variable.

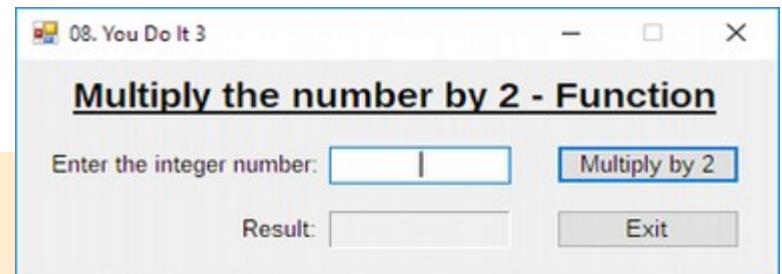
```
1...Private Function GetBonus(ByVal dblSales As Double) As Double  
2...    Return dblBonus  
3...    dblBonus = GetBonus(dblFebSales)
```

## CH6 F5.3 - You Do IT 3: using a Function

1. create an application named **You Do It 3** and save it in the **VB2017\Chap06\Exercise08.You Do It 3 Solution**
2. add a text box, a label, and a button to the form
3. the button's **Click** event procedure should assign the text box value to a **Double** var. and then pass a copy of the variable's value to a **function** named **GetDouble**
4. the **GetDouble** **function** should multiply the value it receives by **2** and then **return** the result to the button's **Click** event procedure, which
5. should assign the **returned** value to the **Double** variable and then display the contents of the **Double** variable in the label
6. code the button's **Click** event procedure and the **GetDouble** function
7. save the solution and then start and test the application

### - my design & code:

```
1  Option Explicit On  
2  Option Strict On  
3  Option Infer Off  
4  Public Class frmMain  
5      Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click  
6          Me.Close()  
7      End Sub  
8  
9      Private Sub txtNumber_Enter(sender As Object, e As EventArgs) Handles txtNumber.Enter  
10         txtNumber.SelectAll()  
11     End Sub  
12  
13     Private Sub txtNumber_TextChanged(sender As Object, e As EventArgs) Handles txtNumber.TextChanged  
14         lblResult.Text = String.Empty  
15     End Sub  
16  
17     Private Sub txtNumber_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtNumber.KeyPress  
18         If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then  
19             e.Handled = True  
20         End If  
21     End Sub  
22     Private Function GetDouble(ByVal dblNum2 As Double) As Double  
23         Dim dblResult2 As Double  
24         dblResult2 = dblNum2 * 2
```



```

25         Return dblResult2
26     End Function
27     Private Sub btnMultiply_Click(sender As Object, e As EventArgs) Handles btnMultiply.Click
28         Dim dblNum As Double
29         Dim dblResult As Double
30         Double.TryParse(txtNumber.Text, dblNum)
31
32         dblResult = GetDouble(dblNum)
33         lblResult.Text = dblResult.ToString("N")
34     End Sub
35 End Class

```

## CH6\_APPLY THE CONCEPTS LESSON

### CH6\_A1 - Add a Combo box to the form: ComboBox tool

- in many interfaces, **combo boxes** are used in place of **list boxes**, and you add a combo box to an interface using **ComboBox** tool in the toolbox
- a **Combo box** is similar to a list box in that it offers the user a list of choices from which to select
- however, unlike a list box:
  - the full list of choices in a combo box can be hidden, allowing you to save space on the form
  - contains a text field, which may or may not be editable by the user

- the most commonly used properties of a **ComboBox**:

|                      |                                                                                                                                                                     |                                                                                                                                                                                                                            |                                                                                                                                                |
|----------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>(Name)</b>        | <b>cbo</b>                                                                                                                                                          |                                                                                                                                                                                                                            |                                                                                                                                                |
| <b>DropDownStyle</b> | - indicate the style of the combo box                                                                                                                               |                                                                                                                                                                                                                            |                                                                                                                                                |
|                      | - choices:                                                                                                                                                          | - a) <b>Simple</b> - looks like <b>Ist</b> , user can enter text<br>- b) <b>DropDown</b> - drop down list of choices, user can enter text<br>- c) <b>DropDownList</b> - drop down list of choices, user can not enter text | -> should use <b>Text</b> property only<br>-> should use <b>Text</b> property only<br>-> can use <b>SelectedItem</b> , or <b>Text</b> property |
|                      | - each style contains a <b>text portion</b> and a <b>list portion</b>                                                                                               |                                                                                                                                                                                                                            |                                                                                                                                                |
|                      | - e.g.s.: shows an example of each style, and it indicates whether the <b>text portion</b> is <u>editable</u> as well as how to <u>view</u> the <b>list portion</b> |                                                                                                                                                                                                                            |                                                                                                                                                |

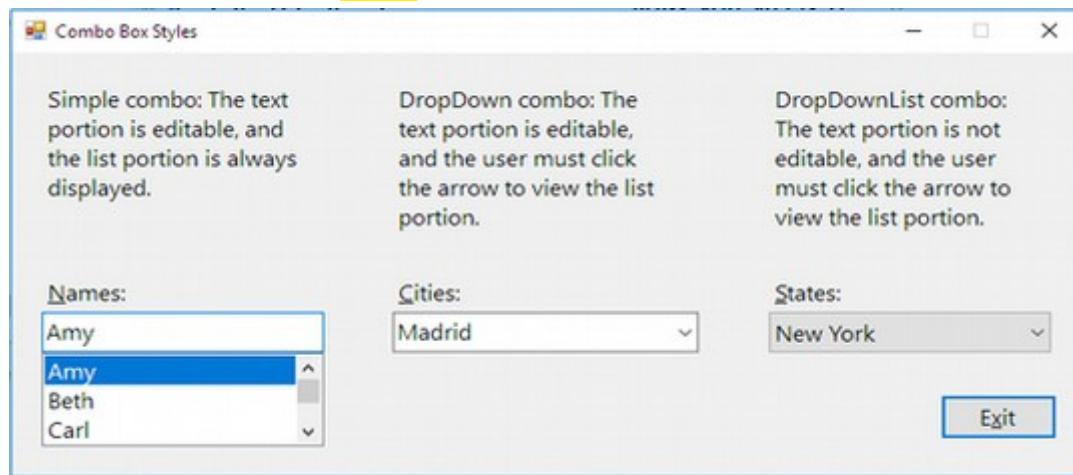


Figure 6-26 ...VB2017\Chap06\Exercise\09.ComboBox Styles Solution

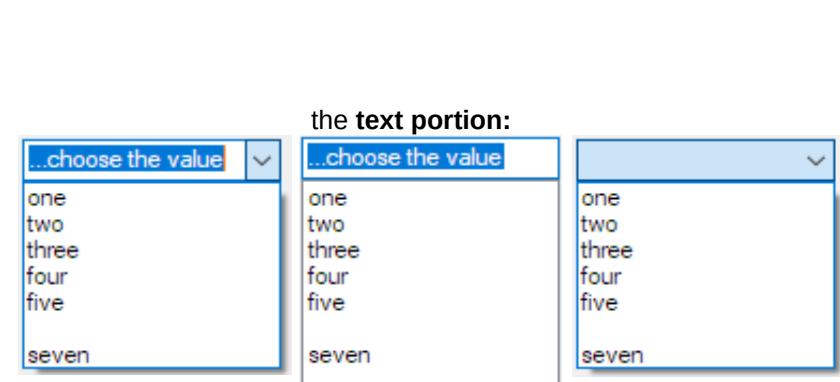


Figure 6-26-1

the list portion

|                      |                                                                                                                                                                   |
|----------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Font</b>          | - specify the font to use for text                                                                                                                                |
| <b>Items</b>         | <b>(Collection)</b> <- String Collection Editor                                                                                                                   |
| <b>SelectedIndex</b> | - as you can with a <b>Ist</b> , you can use either the <b>String Collection Editor</b> or the <b>Item collection's Add method</b> to add an item to a <b>cbo</b> |
| <b>SelectedItem</b>  | - get or set the <b>Index</b> of the selected item                                                                                                                |
| <b>Sorted</b>        | - get or set the <b>Value</b> of the selected item                                                                                                                |
| <b>Text</b>          | - specify whether the items in the list should appear in the order they are entered or<br>in sorted order (dictionary order based on their leftmost characters)   |
|                      | - get or set the value that appears in the <b>text portion</b>                                                                                                    |
|                      | - e.g.: <b>Figure 6-26-1</b> ...choose the value                                                                                                                  |

- you should use a label control to provide keyboard access(ALT key) to the **cbo** **Figure 6-26** -> for the access key to work correctly, you must set the label's **TabIndex** property to a value that is one number less than the **cbo**'s **TabIndex** value

- like the items in a **Ist**, the items in the **list portion** of a **cbo** are either arranged by use, with the most used entries listed first, or sorted in ascending order  
- to sort the items in the **list portion** of a **cbo**, you set the property **Sorted = True**  
- like the first item in a **Ist**, the first item has an **index** of **0**

- as you can with a **Ist**, you can use either the **String Collection Editor** or the **Item collection's Add method** to add an item to a **cbo**

e.g.: code used to fill the **cbos** from **Figure 6-26** with values, and select a default value, which will appear in the **text portion** when the application is started

```

14      Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
15          ' Fills the combo boxes with values and selects the first value in each.
16
17          cboName.Items.Add("Amy")
18          cboName.Items.Add("Beth")
19          cboName.Items.Add("Carl")
20          cboName.Items.Add("Dan")
21          cboName.Items.Add("Jan")
22          cboName.SelectedIndex = 0                                <- select the default value
23
24          cboCity.Items.Add("London")
25          cboCity.Items.Add("Madrid")
26          cboCity.Items.Add("Paris")
27          cboCity.SelectedItem = "Dan"                            <- select the default value
28
29          cboState.Items.Add("Alabama")
30          cboState.Items.Add("Maine")
31          cboState.Items.Add("New York")
32          cboState.Items.Add("South Dakota")
33          cboState.Text = "New York"                           <- select the default value
34      End Sub

```

**notice:** you can use any of the following properties  
to select the default value:

- 1). **SelectedIndex** **(22)**
- 2). **SelectedItem** **(27)**
- 3). **Text** **(33)**

- if no item is selected in a **cbo**:
  - the **Text** property contains the empty string, and
  - the **SelectedItem** property contains the keyword **Nothing**, and
  - the **SelectedIndex** property contains **-1** (negative one)
- if you need to determine the number of items in the **list portion** of a **cbo**, you can use the **Items** collection's **Count** property:  
 syntax: **object.Items.Count**      object = name of the **cbo**
- it is easy to confuse combo box's **SelectedItem** property with its **Text** property:
  - the **SelectedItem** property contains the value of the item selected in the **list portion** of the combo box, whereas
  - the **Text** property contains the value that appears in the **text portion** ->
    - > a value can appear here as a result of the user either: a). selecting an item in the **list portion** of the control, or
    - b). typing an entry in the **text portion** itself
    - c). as a result of a statement that assigns a value to the control's **SelectedIndex**, **SelectedItem**, or **Text** property
- if the **cbo** is:
  - a) **Simple** or **DropDown** style, where the user **CAN type** an entry in the **text portion**, you should use the **Text** property ->
    - > because it contains the value either selected or entered by the user
  - b) **DropDownList** style, where the **text portion** is **NOT editable**, you can use the **SelectedItem**, or **Text** properties interchangeably
- when the value in the **text portion** changes, the combo box's **TextChanged** event occurs

#### **CH6\_A1.1 - to modify the Monthly Payment application - replace **ListBox** for a **ComboBox**:**

- in the next set of steps, you will replace the **Ist** in the Monthly Payment application with a **cbo**
- 1). open: ...VB2017\Chap06\Exercise\10.Payment Solution-ComboBox\Payment Solution.sln
- 2). unlock the controls on the form and click the **IstRates** control and then press **Delete**
  - in the toolbox locate **ComboBox** control and drag it to the form below the label **Rate (%)**
  - size and position the **cbo** to match: **Figure 6-28**
- 3). change its **(Name)** to **cboRates**, and **DropDownStyle** property to **DropDownList**
- 4). lock the controls on the form and then set the **TabIndex** values: **Figure 6-29**
- 5). open the Code Editor window and locate the procedure **frmMain\_Load** (line 11)
  - in the comment (line 12) change **list** to **combo**
  - change both occurrences of **IstRates** in the procedure's code to **cboRates** (lines 15 and 17)
- 6). locate the procedure **btnCalc\_Click** (line 21)
  - in the second TryParse method (line 29) replace **IstRates.SelectedItem.ToString** with **cboRates.Text**
- 7). locate the **ClearPay** procedure (line 58)
  - at the end of the **Handles** clause type: **, cboRate.TextChanged, cboRates.TextChanged**

- 58      **Handles txtPrincipal.TextChanged, cboRates.TextChanged**

- 8). save solution and then start and test the application:
- in the **Principal:** box type **125000** and click the list arrow in the combo box and choose **2.5** and click **Calculate**
  - the result should be:
 

|                  |                 |
|------------------|-----------------|
| <b>15 years:</b> | <b>\$833.49</b> |
| <b>20 years:</b> | <b>\$662.38</b> |
| <b>25 years:</b> | <b>\$560.77</b> |
| <b>30 years:</b> | <b>\$493.90</b> |

- 9). click the **Exit** button, close the Code Editor Window and then close the solution

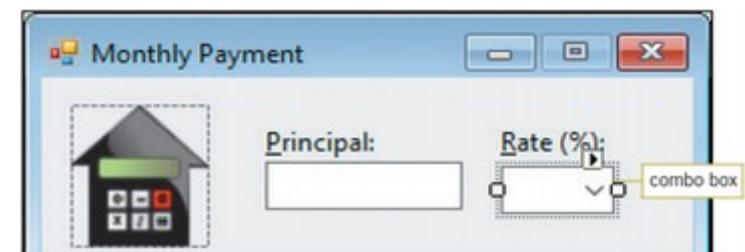


Figure 6-28 Correct location and size of the combo box

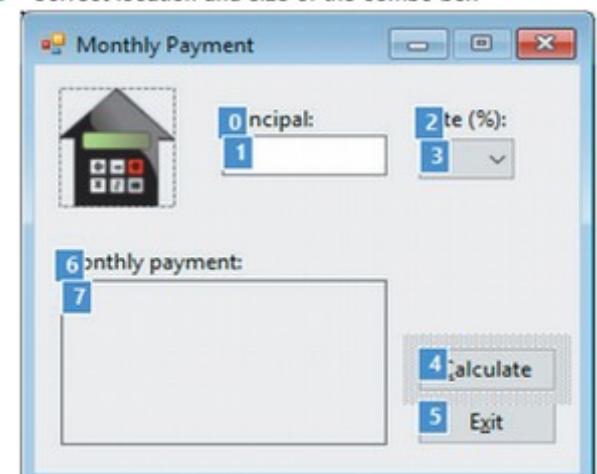


Figure 6-29 Correct TabIndex values

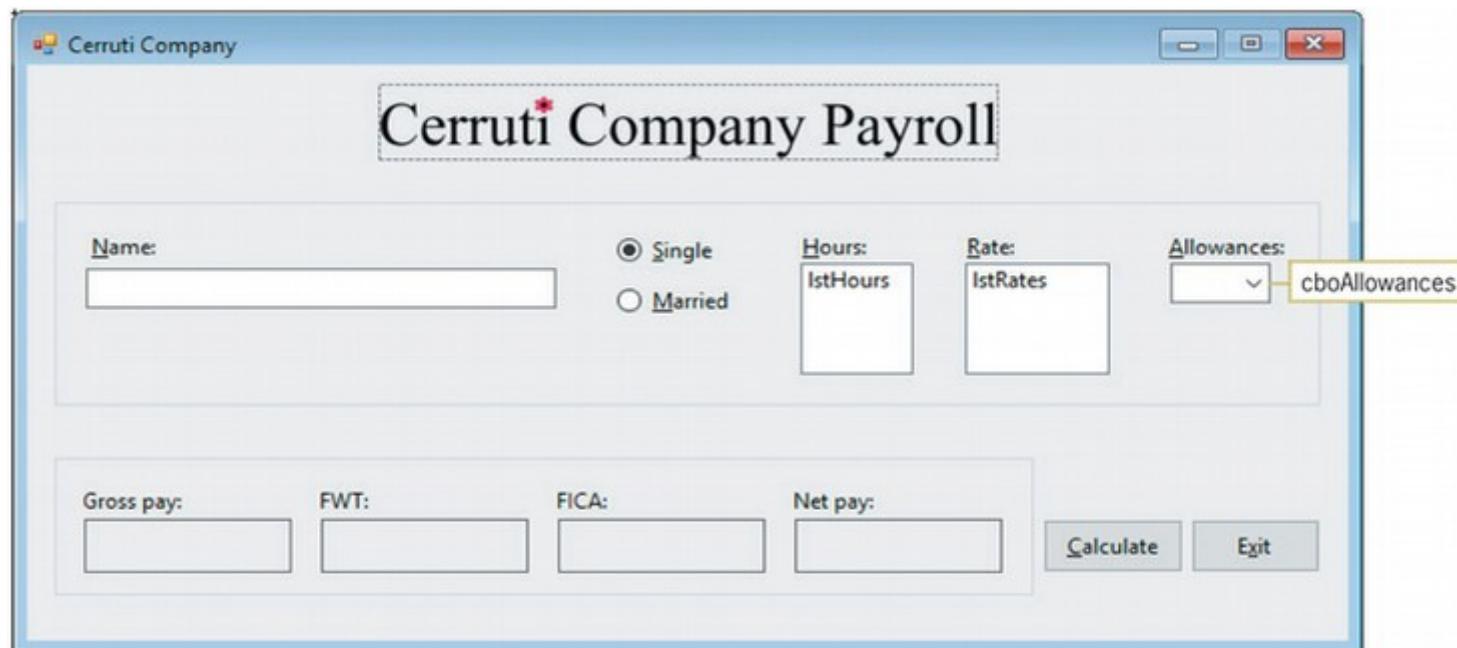
### Mini-Quiz 6-5

1. Which style of combo box has a text portion that is not editable?
2. Which property stores the value either selected in the list portion or typed in the text portion of a combo box?
3. What event occurs when the value in the text portion of a combo box changes?

1...DropDownList combo, looks like list box (others are Single and DropDown combo).  
2...Text property of a combo box.  
3...TextChanged event.

### CH6\_A2 - Add Items to a ComboBox and Select a Default Item: Cerruti Company Payroll application example 01

- in the remaining sections of this lesson, you will code an application for the Cerruti Company, that will calculate an employee's weekly gross pay, federal withholding tax (FWT), Social Security and Medicare (FICA) tax, and net pay



**Figure 6-31** Interface for the Cerruti Company application

- the GUI provides:
  - txt for entering the employee's name: **Name**
  - rad's for entering his or her marital status: **Single/Married**
  - lst for specifying the hours worked: **Hours**
  - lst for specifying the rate of pay: **Rate**
  - cbo allowing the user to either:
    - a). select the number of withholding allowances from the **list portion** of the control,
    - b). or type a number in the **text portion**

1). open: ...VB2017\Chap06\Exercise\11.Cerruti Solution\Cerruti Solution.sln

2). open the Code Editor window (F7) and locate the procedure: **frmMain\_Load**

- the procedure's code adds items to both **ListBoxes** and selects a default item in each one, but

- missing from the procedure is the code to **add** numbers from **0** through **10** to the **cboAllowances** control and then select a default item

-> click the blank line above the **End Sub** clause and then enter the **For...Next** loop and assign. statement: (be sure to change the **Next** clause to **Next intAllow**)

```

66      Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
67          ' Fill list boxes and combo box with values and select a default value in each.
68
69          For dblHours As Double = 0 To 55 Step 0.5
70              lstHours.Items.Add(dblHours.ToString("N1"))
71          Next dblHours
72          lstHours.SelectedItem = "40.0"
73
74          For dblRates As Double = 7.5 To 15.5 Step 0.5
75              lstRates.Items.Add(dblRates.ToString("N2"))
76          Next dblRates
77          lstRates.SelectedItem = "9.50"
78
79          For intAllow As Integer = 0 To 10
80              cboAllowances.Items.Add(intAllow)
81          Next intAllow
82          cboAllowances.SelectedIndex = 0
83
84      End Sub

```

3). save the solution and then start the application

- > in the **Allowances** box just click the list arrow
- the **list portion** of the control displays the numbers from **0** through **10**
- > click the list arrow again to close the list and exit

#### **CH6\_A3 - Code a Combo Box's KeyPress Event Procedure: Cerruti Company Payroll application example 02**

- the **cboAllowances** control is a **DropDown** combo box, which means its text portion is **editable**
- in the next set of steps, you will code the control's **KeyPress** procedure to accept only **numbers** and the **Backspace** key:

1). -> open (create) the code template for the procedure **cboAllowances\_KeyPress** and enter the comment and code:

```

86      Private Sub cboAllowances_KeyPress(sender As Object, e As EventArgs) Handles cboAllowances.KeyPress
87          ' Accept only numbers and the Backspace key.
88
89          If (e.KeyChar < "0" OrElse e.KeyChar > "9") _
90              AndAlso e.KeyChar <> ControlChars.Back Then
91              e.Handled = True
92          End If
93      End Sub

```

2). save the solution and then start the application

- click the **text portion** of the **Allowances** box and then verify that the **text portion** accept only numbers and the Backspace key

3). click the **Exit** button

#### CH6\_A4 - Create an Event-Handling Sub Procedure to clear all labels when user input changes: Cerruti Company Payroll application example 03

- the calculated amounts **gross pay**, **FWT**, **FICA**, and **net pay** should be cleared from the GUI when a change is made to the **employee's name**, **marital status**, **hours worked**, **rate of pay**, or **number of withholding allowances**
- in the next set of steps, you will create an **event-handling Sub procedure** that will be processed when any of these changes occur

1). -> open (create) the code template for the **cboAllowances\_TextChanged** procedure

-> first, replace **cboAllowances\_TextChanged** with **ClearOutput**

-> then, modify the **Handles** clause, and enter the **four** assignment statements:

```
95  Private Sub ClearOutput(sender As Object, e As EventArgs) _
96      Handles cboAllowances.TextChanged, lstHours.SelectedIndexChanged,
97      lstRates.SelectedIndexChanged, radMarried.CheckedChanged,
98      radSingle.CheckedChanged, txtName.TextChanged
99
100     lblGross.Text = String.Empty
101     lblFwt.Text = String.Empty
102     lblFica.Text = String.Empty
103     lblNet.Text = String.Empty
104
105 End Sub
```

<- be sure to type "\_" the line continuation character

<- modify the **Handles** clause

<- modify the **Handles** clause

<- modify the **Handles** clause

<- enter these four assignment statements

2). save the solution

#### CH6\_A5 - Calculate Federal Withholding Tax - FWT: Cerruti Company Payroll application example 04

- the amount of FWT to deduct (odpočítat) from an employee's weekly gross pay is based on his or her weekly taxable wages and **filing status**, which is either Single (including head of household) or Married

- **filing status** - based on marital status and family situation

- 5 possible filing status categories:
  - 1). single individual (never-married, unmarried, legally separated, divorced)
  - 2). married person filing jointly or surviving spouse (you and your spouse can file a joint tax return)
  - 3). married person filing separately (a married couple can choose to file two separate tax returns)
  - 4). head of household (complicated, can maximize your tax savings)
  - 5). qualifying widow(er) with dependent children

- you calculate the weekly taxable wages:    1). by first multiplying the number of withholding allowances by \$77.90 which is the value of a withholding allowance in 2017

2). and then subtracting the result from the weekly gross pay

e.g.1 if your weekly gross pay is **\$500** and you have two withholding allowances, your weekly taxable wages are **\$344.20**     $(500 - 2 \times 77.9) = 344.2$

- you use the weekly taxable wages, along with the filing status and the appropriate weekly FWT table, to determine the amount of tax to withhold

- the weekly tax tables for 2017 are shown in **Figure 6-35**

## FWT Tables Weekly Payroll Period

| Single person (including head of household) |               |                                          |                 |                 | Married person            |               |                                          |       |                 |
|---------------------------------------------|---------------|------------------------------------------|-----------------|-----------------|---------------------------|---------------|------------------------------------------|-------|-----------------|
| If the taxable wages are:                   |               | The amount of income tax to withhold is: |                 |                 | If the taxable wages are: |               | The amount of income tax to withhold is: |       |                 |
| over:                                       | but not over: | Base amount                              | %               | Of excess over: | over:                     | but not over: | Base amount                              | %     | Of excess over: |
| e.g.1                                       | \$44          | \$44                                     | 0               |                 | \$166                     | \$166         | 0                                        | 10%   | \$166           |
|                                             | \$224         | \$224                                    | 0               | 10%             | \$525                     | \$525         | 0                                        | 10%   | \$525           |
|                                             | \$224         | \$774                                    | \$18.00 plus    | 15%             | \$1,626                   | \$1,626       | \$35.90 plus                             | 15%   | \$1,626         |
|                                             | \$774         | \$1,812                                  | \$100.50 plus   | 25%             | \$3,111                   | \$3,111       | \$201.05 plus                            | 25%   | \$1,626         |
|                                             | \$1,812       | \$3,730                                  | \$360.00 plus   | 28%             | \$4,654                   | \$4,654       | \$572.30 plus                            | 28%   | e.g.2           |
|                                             | \$3,730       | \$8,058                                  | \$897.04 plus   | 33%             | \$8,180                   | \$8,180       | \$1,004.34 plus                          | 33%   | \$3,111         |
|                                             | \$8,058       | \$8,090                                  | \$2,325.28 plus | 35%             | \$9,218                   | \$9,218       | \$2,167.92 plus                          | 35%   | \$4,654         |
|                                             | \$8,090       |                                          | \$2,336.48 plus | 39.6%           |                           |               | \$2,531.22 plus                          | 39.6% | \$8,180         |
|                                             |               |                                          |                 |                 |                           |               |                                          |       | \$9,218         |

**Figure 6-35** Weekly FWT tables for the year 2017

- each table contains five columns of information:

- the first two columns list various ranges, also called brackets, of taxable wage amounts:
  - the 1st column **Over** lists the amount that a taxable wage in that bracket must be over
  - the 2nd column **But not over** lists the maximum amount included in the bracket
- the remaining three columns **Base amount**, **Percentage**, **Of excess over** tell you how to calculate the tax for each range

- e.g.1 - assume that your marital status is **Single**, and your weekly wages are **\$500**, so your weekly taxable wages are **\$344.20**:
- before you can calculate the amount of your tax, you need to locate your taxable wages in the first two columns of the Single table:
    - taxable wages of \$344.20 fall within the: \$224 through \$774 bracket
  - after locating the bracket that contains your taxable wages, you then use the remaining three columns in the table to calculate your tax:
    - as the example in **Figure 6-36** indicates, you calculate the tax by:
      - 1). first subtraction \$224 (the amount shown in the **Of excess over** column) from your taxable wages of \$344.20, giving **\$120.20**
      - 2). you then multiply \$120.20 by **15%** (the amount shown in the **Percentage** column), giving **\$18.03**
      - 3). you then add that amount to the amount shown in the **Base amount** column (in this case \$18.00), giving **\$36.03 as your tax**

#1: Calculate the weekly taxable wages:

- if the value of a withholding allowance (constant) in 2017 is: **\$77.90**
  - if a number of withholding allowances is: **2**  $77.90 * 2 = 155.8$
  - if the weekly gross pay is: **\$500**  $500 - 155.8 = 344.2$
- then the weekly taxable wages are: **\$344.20****

#2: Calculate your tax:

- the weekly taxable wages are: **\$344.20**
  - **Of excess over** bracket no.5 in table is: **\$224**  $344.20 - 224 = 120.20$
  - **Percentage** bracket no.4 in table is: **15%**  $120.20 * 0.15 = 18.03$
  - **Base amount** bracket no.3 in table is: **\$201.05 plus**  $18.03 + 18.00 = 36.03$
- then your Federal Withholding Tax (FWT) is: **\$36.03****

e.g.2 - assume that your marital status is **Married**, and your weekly taxable wages are **\$1,700.00**:

#2: Calculate your tax:

- the weekly taxable wages are:

**\$1,700.00**

- Of excess over bracket no.5 in table is:

**\$1,626.00**

$1700 - 1626 = 74$

- Percentage bracket no.4 in table is:

**25%**

$74 * 0.25 = 18.50$

- Base amount bracket no.3 in table is:

**\$18.00 plus**

$18.50 + 201.05 = 219.55$

then your **Federal Withholding Tax (FWT)** is:

**\$219.55**

#### CH6\_A6 - Invoke an Independent Sub Procedure and a Function (btnCalc\_Click procedure): Cerruti Company Payroll application example 05

- to give you practice with creating both: an independent Sub procedure and a function, the Cerutti Company application will use:

- 1). an **independent Sub procedure** to calculate the **FWT** for employees whose marital status is **Single**, but
- 2). a **function** to calculate the **FWT** for employees whose marital status is **Married**

- before creating the Sub procedure and function, you will complete the **btnCalc\_Click** procedure, which will use the **Sub procedure** and **function** to calculate the appropriate tax

1). locate the procedure **btnCalc\_Click** (most of the procedure's code has already been entered for you)

- the procedure declares two **named constants** and eight **variables**

- the constant **dblONE\_ALLOWANCE** is initialized to the value of a withholding allowance in 2017 - \$77.90, therefore **77.9**
- the constant **dblFICA\_RATE** is initialized to the FICA tax rate for 2017 - 7.65%, therefore **0.0765**

```
13      Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
14          ' Calculates and displays gross pay, taxes, and net pay.
15
16          Const dblONE_ALLOWANCE As Double = 77.9
17          Const dblFICA_RATE As Double = 0.0765
18          Dim dblHours As Double
19          Dim dblPayRate As Double
20          Dim intAllowances As Integer
21          Dim dblGross As Double
22          Dim dblTaxable As Double
23          Dim dblFwt As Double
24          Dim dblFica As Double
25          Dim dblNet As Double
26
```

2). - the first two **TryParse** methods store the selected list box items in variables

- the third **TryParse** method stores in a variable the value contained in the combo box's **Text** property -> recall that the **Text** property contains the value either selected in the **list portion**, or entered in the **text portion**

```
27      Double.TryParse(lstHours.SelectedItem.ToString, dblHours)
28      Double.TryParse(lstRates.SelectedItem.ToString, dblPayRate)
29      Integer.TryParse(cboAllowances.Text, intAllowances)
30
```

3). the **selection structure** calculates the employee's gross pay, giving time and a half for any hours worked over 40:

- if employee worked 40 and less hours, the Gross pay will be calculated simply: hours worked \* pay rate

$dblGross = dblHours * PayRate$

- but if employee worked more than 40 hours (overtime), the overtime will be payed plus extra 1.5

dblGross =

$$\begin{aligned} & 40 * \text{dblPayRate} + \\ & (\text{dblHours} - 40) * \text{dblPayRate} * 1.5 \end{aligned}$$

- 1). 40 hours worked \* pay rate, plus
- 2). hours worked overtime \* pay rate \* 1.5

```
31      ' Calculate gross pay.  
32      If dblHours <= 40 Then  
33          dblGross = dblHours * dblPayRate  
34      Else  
35          dblGross = 40 * dblPayRate + (dblHours - 40) * dblPayRate * 1.5  
36      End If  
37
```

4). the **assignment statement** calculates the taxable wages by first multiplying the number of withholdinh allowances by the value of one withholding allowance, and then subtracting the result from the gross pay:

```
38      ' Calculate taxable wages.  
39      dblTaxable = dblGross - (intAllowances * dblONE_ALLOWANCE)  
40
```

5a). -> click the blank line below the comment on line: 41 ' Determine the FWT.

- recall that the appropriate FWT table to use depends on the person's marital status, which the user specifies by selecting either the Single rad, or the Married rad in the interface

-> type the following **If clause** (you can also use `radSingle.Checked = True` as the condition) and press Enter:

```
41      ' Determine the FWT.  
42      If radSingle.Checked Then  
43
```

5b). if the Single rad is selected, the selection structure's true path will call an **independent Sub procedure** named **GetSingleFwt** to calculate the appropriate tax

- to make the calculation, the procedure will need to know:  
1) the employee's taxable wages and  
2) the address of a variable where it can store the tax

-> type the following **calling statement** and then press Enter:

```
43      GetSingleFwt(dblTaxable, dblFwt)  
44
```

<- the **red squiggle** means, that there is no procedure named **GetSingleFwt** yet,  
will disappear when you create the **Sub procedure**

5c). -> type **Else** and press Enter

- if the Married rad is selected, the selection structure's false path will invoke a **function** named **GetMarriedFwt** and assigns its return value to the variable **dblFwt**

- to calculate the FWT, the function will need to know: 1) only the employee's wages

-> type the **assignment statement** shown in the selection structure's false path

```
44      Else  
45      dblFwt = GetMarriedFwt(dblTaxable)  
46      End If
```

<- the **red squiggle** means, that there is no procedure named **GetMarriedFwt** yet,  
will disappear when you create the **function**

- the whole code 5a, 5b, 5c:

```
41      ' Determine the FWT.  
42      If radSingle.Checked Then  
43          GetSingleFwt(dblTaxable, dblFwt)  
44      Else  
45          dblFwt = GetMarriedFwt(dblTaxable)  
46      End If  
47
```

6). below the selection structure is an assignment statement that calculates the FICA tax

- this tax is calculated by multiplying the employee's gross pay by the FICA tax rate constant for year 2017:

```
48      ' Calculate FICA tax.  
49      dblFica = dblGross * dblFICA_RATE
```

7). click the blank line below the comment on line: 50 ' Round gross pay, FWT, and FICA tax.

- > enter the three assignment statements:

- the statements round the: **gross pay, FWT, and FICA tax** to two decimal places before using them in the **net pay** calculation on the line: 56

```
50      ' Round gross pay, FWT, and FICA tax.  
51      dblGross = Math.Round(dblGross, 2)  
52      dblFwt = Math.Round(dblFwt, 2)  
53      dblFica = Math.Round(dblFica, 2)  
  
54  
55      ' Calculate net pay.  
56      dblNet = dblGross - dblFwt - dblFica  
57
```

8). the last statements in the procedure **btnCalc\_Click** display the: **gross pay, FWT, FICA tax, and net pay** in the GUI:

```
58      ' Display calculated amounts.  
59      lblGross.Text = dblGross.ToString("N2")  
60      lblFwt.Text = dblFwt.ToString("N2")  
61      lblFica.Text = dblFica.ToString("N2")  
62      lblNet.Text = dblNet.ToString("N2")  
63
```

#### CH6\_A7 - Create an Independent Sub Procedure: GetSingleFwt Sub procedure for Single employee: Cerruti Company Payroll application example 06

- in this section, you will code the **GetSingleFwt** Sub procedure, which will calculate the FWT for an employee whose marital status is **Single**

- as indicated earlier, the procedure needs **two** items of information from its **calling statement** in the **btnCalc\_Click** procedure:

GetSingleFwt(db1Taxable, db1Fwt)

1) the employee's taxable wages, and

2) the address of a variable where the calculated tax can be stored

db1Taxable (ByVal db1TaxPay)

db1Fwt (ByRef db1FedWthTax)

1). scroll to the top of the Code Editor window and then click the blank line below the comment: 10 ' Independent Sub procedure and function.

- > enter the **GetSingleFwt** procedure header, and the Code Editor will automatically enter the procedure footer **End Sub** clause for you

```
9      Public Class frmMain  
10     ' Calculates and displays gross pay, taxes, and net pay.  
11     Private Sub GetSingleFwt(ByVal db1TaxPay As Double,  
12                               ByRef db1FedWthTax As Double)  
13  
14     End Sub
```

2). for your convenience, you will find the code associated with the **Single FWT table**, in the file: **Single.txt**

-> on the menu click **File**, point to **Open**, and then click **File...**

-> open the folder **Cerruti Project**, click **Single.txt** in the list of filenames, and then click the **Open** button

- the **Single.txt** file appears in a separate window in the IDE

-> on the menu bar click **Edit**, then click **Select All**, then press **Ctrl + c** to copy the selected text and then close the **Single.txt** window

- the insertion point should be in the blank line 13 above the **End Sub** clause

-> press **Ctrl + v** to paste the copied text into the **GetSingleFwt** procedure:

```

9   Public Class frmMain
10    ' Calculates and displays gross pay, taxes, and net pay.
11    Private Sub GetSingleFwt(ByVal dblTaxPay As Double,
12                           ByRef dblFedWthTax As Double)
13      Select Case dblTaxPay
14        Case Is <= 44
15          dblFedWthTax = 0
16        Case Is <= 224
17          dblFedWthTax = 0.1 * (dblTaxPay - 44)
18        Case Is <= 774
19          dblFedWthTax = 18 + 0.15 * (dblTaxPay - 224)
20        Case Is <= 1812
21          dblFedWthTax = 100.5 + 0.25 * (dblTaxPay - 774)
22        Case Is <= 3730
23          dblFedWthTax = 360 + 0.28 * (dblTaxPay - 1812)
24        Case Is <= 8058
25          dblFedWthTax = 897.04 + 0.33 * (dblTaxPay - 3730)
26        Case Is <= 8090
27          dblFedWthTax = 2325.28 + 0.35 * (dblTaxPay - 8058)
28        Case Else
29          dblFedWthTax = 2336.48 + 0.396 * (dblTaxPay - 8090)
30      End Select
31
32    End Sub
33

```

template for calculations:

| Single person (including head of household) |               |                                          |       |                 |
|---------------------------------------------|---------------|------------------------------------------|-------|-----------------|
| If the taxable wages are:                   |               | The amount of income tax to withhold is: |       |                 |
| over:                                       | but not over: | Base amount                              | %     | Of excess over: |
| \$44                                        | \$44          | 0                                        |       | \$44            |
| \$224                                       | \$224         | 0                                        | 10%   | \$224           |
| \$774                                       | \$774         | \$18.00 plus                             | 15%   | \$774           |
| \$1,812                                     | \$1,812       | \$100.50 plus                            | 25%   | \$1,812         |
| \$3,730                                     | \$3,730       | \$360.00 plus                            | 28%   | \$3,730         |
| \$8,058                                     | \$8,058       | \$897.04 plus                            | 33%   | \$8,058         |
| \$8,090                                     | \$8,090       | \$2,325.28 plus                          | 35%   | \$8,090         |
|                                             |               | \$2,336.48 plus                          | 39.6% | \$8,090         |
|                                             |               | no.3                                     | no.4  | no.5            |

Figure 6-35 Weekly FWT tables for the year 2017

- dblTaxPay = weekly taxable wages

$$\text{dblFedWthTax} = \text{no.3} + \text{no.4} * (\text{dblTaxPay} - \text{no.5})$$

#### CH6\_A8 - Create a Function: GetMarriedFwt function for Married employee: Cerruti Company Payroll application example 07

- in this section, you will code the **GetMarriedFwt** function, which will calculate the FWT for an employee whose marital status is **Married**
- as indicated earlier, the procedure needs **only one** item of information from the **statement** that **invokes** it in the **btnCalc\_Click** procedure: ->

1) the employee's taxable wages      **dblTaxable**      (ByVal dblTaxPay)      -> **dblFwt = GetMarriedFwt(dblTaxable)**

- the function will return the calculated FWT to that statement, which will assign the **return value** to the variable **dblFwt**

- 1). click the blank line **33** below the **GetSingleFwt** procedure's **End Sub** clause and then press Enter

-> enter the **GetMarriedFwt** function header, press Enter, and the Code Editor will automatically enter the procedure footer **End Function** clause for you

- the **green squiggle** on the line **36** will disappear when you enter the **Return** statement

```

32   End Sub
33
34   Private Function GetMarriedFwt(ByVal dblTaxPay As Double) As Double
35
36   End Function

```

2). first, you will declare a variable that will store the FWT after it has been calculated: **Dim dblFedWthTax As Double**

- type the following Dim statement and then press Enter twice:

```
35      Dim dblFedWthTax As Double  
36  
37
```

3). you will find the code associated with the **Married FWT table**, in the file: **Married.txt**

-> follow the same procedure like for inserting the code in the **Single.txt**:

- finally, you will enter the **Return statement** that will return the calculated FWT to the statement that invoked the **function** in the procedure **btnCalc\_Click**

-> insert a blank line below the **End Select** clause and then type the **Return statement** in line 55

```
34  Private Function GetMarriedFwt(ByVal dblTaxPay As Double) As Double  
35      Dim dblFedWthTax As Double  
36  
37      Select Case dblTaxPay  
38          Case Is <= 166  
39              dblFedWthTax = 0  
40          Case Is <= 525  
41              dblFedWthTax = 0.1 * (dblTaxPay - 166)  
42          Case Is <= 1626  
43              dblFedWthTax = 35.9 + 0.15 * (dblTaxPay - 525)  
44          Case Is <= 3111  
45              dblFedWthTax = 201.05 + 0.25 * (dblTaxPay - 1626)  
46          Case Is <= 4654  
47              dblFedWthTax = 572.3 + 0.28 * (dblTaxPay - 3111)  
48          Case Is <= 8180  
49              dblFedWthTax = 1004.34 + 0.33 * (dblTaxPay - 4654)  
50          Case Is <= 9218  
51              dblFedWthTax = 2167.92 + 0.35 * (dblTaxPay - 8180)  
52          Case Else  
53              dblFedWthTax = 2531.22 + 0.396 * (dblTaxPay - 9218)  
54      End Select  
55      Return dblFedWthTax  
56  End Function
```

template for calculations:

| Married person            |               |                                          |       |                 |
|---------------------------|---------------|------------------------------------------|-------|-----------------|
| If the taxable wages are: |               | The amount of income tax to withhold is: |       |                 |
| over:                     | but not over: | Base amount                              | %     | Of excess over: |
| \$166                     | \$166         | 0                                        |       | \$166           |
| \$166                     | \$525         | 0                                        | 10%   | \$525           |
| \$525                     | \$1,626       | \$35.90 plus                             | 15%   | \$1,626         |
| \$1,626                   | \$3,111       | \$201.05 plus                            | 25%   | \$3,111         |
| \$3,111                   | \$4,654       | \$572.30 plus                            | 28%   | \$4,654         |
| \$4,654                   | \$8,180       | \$1,004.34 plus                          | 33%   | \$8,180         |
| \$8,180                   | \$9,218       | \$2,167.92 plus                          | 35%   | \$9,218         |
| \$9,218                   |               | \$2,531.22 plus                          | 39.6% |                 |

no.3                    no.4                    no.5

**Figure 6-35** Weekly FWT tables for the year 2017

- dblTaxPay = weekly taxable wages

$$\text{dblFedWthTax} = \text{no.3} + \text{no.4} * (\text{dblTaxPay} - \text{no.5})$$

## CH6\_A9 - Validate an Application's Code: Cerruti Company Payroll application example 08

- you will test the Cerruti Company application twice using the data provided
- the figure also shows the correct amounts for the: **gross pay, taxes (FWT and FICA), and net pay**

**First test:** Carol Swanski, Single, 37.5 hours worked, \$13.50 per hour, one allowance:

Cerruti Company

# Cerruti\* Company Payroll

|                        |                                                                          |                                        |                                           |                              |
|------------------------|--------------------------------------------------------------------------|----------------------------------------|-------------------------------------------|------------------------------|
| Name:<br>Carol Swanski | <input checked="" type="radio"/> Single<br><input type="radio"/> Married | Hours:<br>36.5<br>37.0<br>37.5<br>38.0 | Rate:<br>13.00<br>13.50<br>14.00<br>14.50 | Allowances:<br>1             |
| Gross pay:<br>506.25   | FWT:<br>48.65                                                            | FICA:<br>38.73                         | Net pay:<br>418.87                        | <b>Calculate</b> <b>Exit</b> |

|                          |               |
|--------------------------|---------------|
| <b>gross wages:</b>      | <b>506.25</b> |
| allowance:               | - 77.90       |
| taxable wages:           | 428.35        |
| of excess over:          | - 224.00      |
|                          | 204.35        |
| percentage:              | * 0.15        |
|                          | 30.6525       |
| Base amount:             | + 18.00       |
| <b>FWT (rounded):</b>    | <b>48.65</b>  |
| <br>                     |               |
| <b>FICA tax:</b>         | <b>506.25</b> |
|                          | * 0.0765      |
| rounded to two decimals: | <b>38.73</b>  |
| <br>                     |               |
| <b>Net pay:</b>          | <b>506.25</b> |
|                          | - 48.65       |
|                          | - 38.73       |
|                          | <b>418.87</b> |

**Second test:** Michael Williams, Married, 53.5 hours worked, \$15 per hour, two allowances:

Cerruti Company

# Cerruti\* Company Payroll

|                           |                                                                          |                                        |                                           |                              |
|---------------------------|--------------------------------------------------------------------------|----------------------------------------|-------------------------------------------|------------------------------|
| Name:<br>Michael Williams | <input type="radio"/> Single<br><input checked="" type="radio"/> Married | Hours:<br>53.5<br>54.0<br>54.5<br>55.0 | Rate:<br>14.00<br>14.50<br>15.00<br>15.50 | Allowances:<br>2             |
| Gross pay:<br>903.75      | FWT:<br>69.34                                                            | FICA:<br>69.14                         | Net pay:<br>765.27                        | <b>Calculate</b> <b>Exit</b> |

|                          |               |
|--------------------------|---------------|
| <b>gross wages:</b>      | <b>903.75</b> |
| allowance deduction:     | - 155.80      |
| taxable wages:           | 747.95        |
| of excess over:          | - 525.00      |
|                          | 222.95        |
| percentage:              | * 0.15        |
|                          | 33.4425       |
| Base amount:             | + 35.90       |
| <b>FWT (rounded):</b>    | <b>69.34</b>  |
| <br>                     |               |
| <b>FICA tax:</b>         | <b>903.75</b> |
|                          | * 0.0765      |
| rounded to two decimals: | <b>69.14</b>  |
| <br>                     |               |
| <b>Net pay:</b>          | <b>903.75</b> |
|                          | - 69.34       |
|                          | - 69.14       |
|                          | <b>765.27</b> |

## CH6\_A10 - Professionalize Your Application's Interface: Message Box - `MessageBox`.`Show` method syntax, and its return value: `DialogResult`.

- at times, an application may need to communicate with the user during **run time**
- you can accomplish this task by using Visual Basic's method `MessageBox`.`Show` to display a message box during run time of your application
- the message box contains: text, caption, one or more buttons, icon, and OPTIONAL defaultButton

syntax:

```
MessageBox.Show(text, caption, buttons, icon, defaultButton)
```

**text** - text to display in the message box; use sentence capitalization

**caption** - text to display in the message box's title bar; use book title capitalization

**buttons** - buttons to display in the message box; can be one of the following constants:

`MessageBoxButtons`.`AbortRetryIgnore`

`MessageBoxButtons`.`OK` (default setting)

`MessageBoxButtons`.`OKCancel`

`MessageBoxButtons`.`RetryCancel`

`MessageBoxButtons`.`YesNo`

`MessageBoxButtons`.`YesNoCancel`

**icon** - icon to display in the message box; typically one of the following constants:

`MessageBoxIcon`.`Exclamation` !

`MessageBoxIcon`.`Information` i

`MessageBoxIcon`.`Stop` X

**defaultButton** - button automatically selected when the user presses Enter; can be one of the following constants:

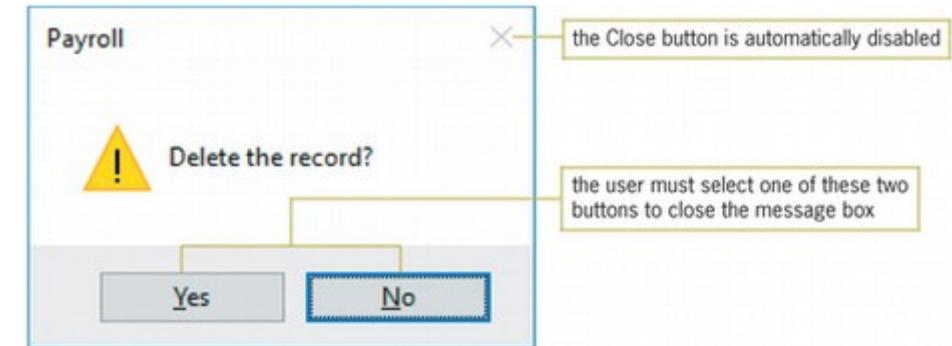
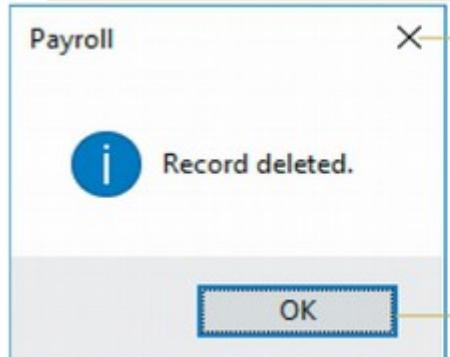
`MessageBoxDefaultButton`.`Button1` (default setting)

`MessageBoxDefaultButton`.`Button2`

`MessageBoxDefaultButton`.`Button3`

**e.g.1:** - used when you are not interested in the value; used for informational purposes only

```
MessageBox.Show("Record deleted.", "Payroll", MessageBoxButtons.OK, MessageBoxIcon.Information)
```



**e.g.2:** - used when the button determines the next task performed by the application

```
MessageBox.Show("Delete the record?", "Payroll", MessageBoxButtons.YesNo, MessageBoxIcon.Exclamation, MessageBoxDefaultButton.Button2)
```

- after displaying the message box, method **MessageBox.Show** waits for the user to choose one of the buttons
- it then closes the message box and returns an **integer** indicating the **button chosen** by the user
- each value is associated with a button that can appear in a message box
- when referring to the method's return value in code, you should use the **DialogResult values** rather than the **integers** because the values make the code more self-documenting and easier to understand

| MessageBox.Show method's return values: |                 |                            |
|-----------------------------------------|-----------------|----------------------------|
| when user chose the button:             | return integer: | return DialogResult value: |
| OK                                      | 1               | DialogResult.OK            |
| Cancel                                  | 2               | DialogResult.Cancel        |
| Abort                                   | 3               | DialogResult.Abort         |
| Retry                                   | 4               | DialogResult.Retry         |
| Ignore                                  | 5               | DialogResult.Ignore        |
| Yes                                     | 6               | DialogResult.Yes           |
| No                                      | 7               | DialogResult.No            |

e.g.3:

```
Dim dlgButton As DialogResult
dlgButton = MessageBox.Show("Delete the record?", "Payroll", MessageBoxButtons.YesNo, MessageBoxIcon.Exclamation, MessageBoxButtons.DefaultButton.Button2)
If dlgButton = DialogResult.Yes Then
    ...instructions to delete the record
End If
```

- the **MessageBox.Show** method's return value is assigned to a **DialogResult** variable named **dlgButton**
- the selection structure in the example compares the contents of the variable with the **DialogResult.Yes** value

e.g.4:

```
If MessageBox.Show("Play another game?", "Math Monster", MessageBoxButtons.YesNo, MessageBoxIcon.Exclamation) = DialogResult.Yes Then
    ...instructions to start another game
Else ' "No" button is selected.
    ...instructions to close the game application
End If
```

- the method's return value is not stored in a variable, instead the method appears in the selection structure's condition, where its return value is compared with the **DialogResult.Yes** value
- the selection structure performs one set of tasks when the user selects the button **Yes** in the message box, and it performs a different set of tasks when the user selects the button **No**
- many programmers document the **Else** portion of the selection structure because it clearly states that it is processed only when the user selects the button **No**

#### CH6\_A10.1 - Message Box - MessageBox.Show method - confirmation to kill the Cerruti Company Payroll application example 09

- a common use for the method **MessageBox.Show** is to verify that the user wants to **exit** an application when he or she clicks the button **Exit**
- the method is typically entered in the form's **FormClosing** event procedure, what occurs when a form is about to be closed
- in most cases, this happens when the computer processes the statement **Me.Close()** in the application's code, however
  - it also occurs when the user clicks the button **Close** on the form's **title bar**
- you prevent the computer from closing a form by setting the **Cancel property** on the **FormClosing** event's **e** parameter to **True** (**e.Cancel = True**)

- in the next set of steps, you will include the method **MessageBox.Show** in the Cerruti Company Payroll application's **FormClosing** event procedure:

1). open: ...VB2017\Chap06\Exercise11.Cerruti Solution\Cerruti Solution.sln

2). open (create) the code template for the **frmMain\_FormClosing** procedure - be sure to open the template **FormClosing** and not the template **FormClosed**

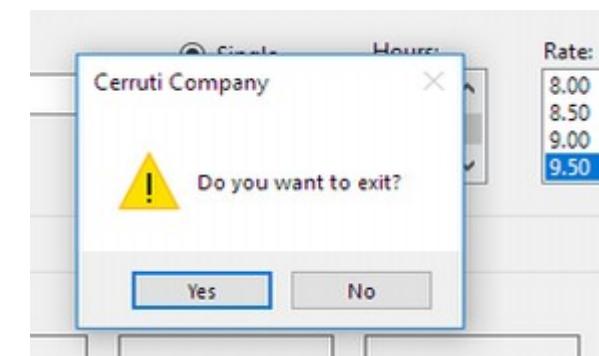
- enter the comments and code:

```
158     Private Sub frmMain_FormClosing(sender As Object, e As FormClosingEventArgs) Handles Me.FormClosing
159         ' Verify that the user wants to kill the application.
160
161         Dim dlgButton As DialogResult
162         dlgButton = MessageBox.Show("Do you want to exit?", 
163                                     "Cerruti Company",
164                                     MessageBoxButtons.YesNo,
165                                     MessageBoxIcon.Exclamation)
166
167         ' If the No button is selected, do not close the form.
168         If dlgButton = DialogResult.No Then
169             e.Cancel = True
170         End If
171     End Sub
172 End Class
```

3). save the solution and then start the application

- > click the button **Exit** --> the message box appears on the screen:
- > click the button **No** --> the form remains on the screen
- > now, on the form's title bar click the button **Close**
- > this time, click the button **Yes** to end the application

4). close the Code Editor window and then close the solution



### Mini-Quiz 6-6

1. What constant is associated with the **Information Icon** in the **MessageBox.Show** method?
2. If the user clicks the button **Cancel** in a message box, the **MessageBox.Show** method returns the integer **2**, which is represented by which **DialogResult** value?
3. When entered in a form's **FormClosing** procedure, what statement prevents the computer from closing the form?

1...**MessageBoxIcon.Information**  
2...**DialogResult.Cancel**  
3...**e.Cancel = True**

## CH6\_A11 - Cerruti Company Payroll application - completed code & GUI with items example 10

```
1      ' Name:          Cerruti Project
2      ' Purpose:       Displays an employee's gross pay, taxes, and net pay.
3      ' Programmer:    <your name> on <current date>
4
5      Option Explicit On
6      Option Strict On
7      Option Infer Off
8
9      Public Class frmMain
10     ' Independent Sub procedure and function.
11     Private Sub GetSingleFwt(ByVal dblTaxPay As Double,
12                               ByRef dblFedWthTax As Double)
13         Select Case dblTaxPay
14             Case Is <= 44
15                 dblFedWthTax = 0
16             Case Is <= 224
17                 dblFedWthTax = 0.1 * (dblTaxPay - 44)
18             Case Is <= 774
19                 dblFedWthTax = 18 + 0.15 * (dblTaxPay - 224)
20             Case Is <= 1812
21                 dblFedWthTax = 100.5 + 0.25 * (dblTaxPay - 774)
22             Case Is <= 3730
23                 dblFedWthTax = 360 + 0.28 * (dblTaxPay - 1812)
24             Case Is <= 8058
25                 dblFedWthTax = 897.04 + 0.33 * (dblTaxPay - 3730)
26             Case Is <= 8090
27                 dblFedWthTax = 2325.28 + 0.35 * (dblTaxPay - 8058)
28             Case Else
29                 dblFedWthTax = 2336.48 + 0.396 * (dblTaxPay - 8090)
30         End Select
31
32     End Sub
33
34     Private Function GetMarriedFwt(ByVal dblTaxPay As Double) As Double
35         Dim dblFedWthTax As Double
36
37         Select Case dblTaxPay
38             Case Is <= 166
39                 dblFedWthTax = 0
40             Case Is <= 525
41                 dblFedWthTax = 0.1 * (dblTaxPay - 166)
42             Case Is <= 1626
43                 dblFedWthTax = 35.9 + 0.15 * (dblTaxPay - 525)
```

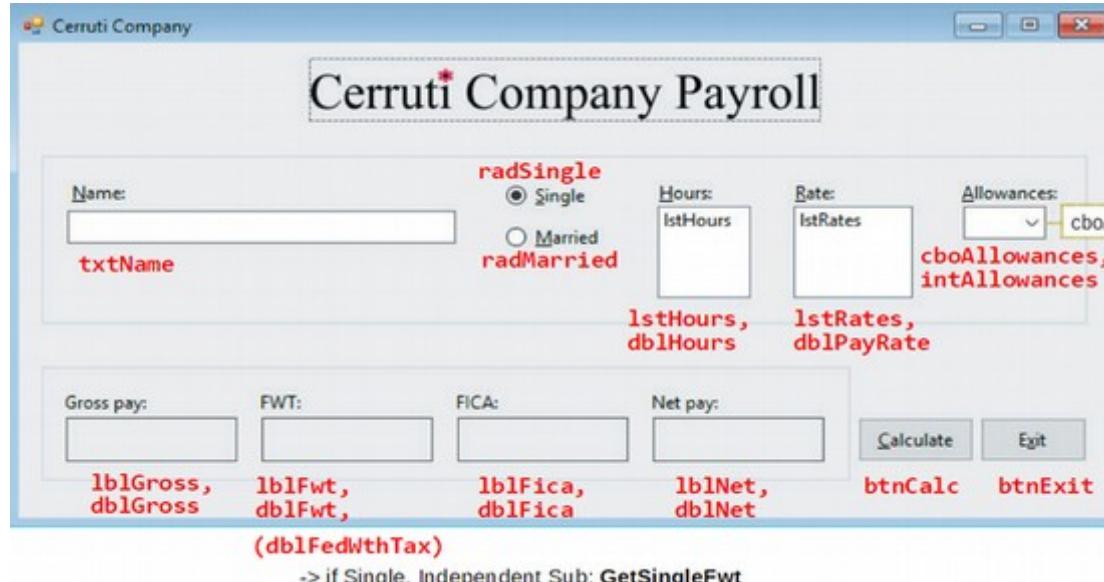
```

44      Case Is <= 3111
45          dblFedWthTax = 201.05 + 0.25 * (dblTaxPay - 1626)
46      Case Is <= 4654
47          dblFedWthTax = 572.3 + 0.28 * (dblTaxPay - 3111)
48      Case Is <= 8180
49          dblFedWthTax = 1004.34 + 0.33 * (dblTaxPay - 4654)
50      Case Is <= 9218
51          dblFedWthTax = 2167.92 + 0.35 * (dblTaxPay - 8180)
52      Case Else
53          dblFedWthTax = 2531.22 + 0.396 * (dblTaxPay - 9218)
54  End Select
55  Return dblFedWthTax
56 End Function
57 Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
58     ' Calculates and displays gross pay, taxes, and net pay.
59
60     Const dblONE_ALLOWANCE As Double = 77.9
61     Const dblFICA_RATE As Double = 0.0765
62     Dim dblHours As Double
63     Dim dblPayRate As Double
64     Dim intAllowances As Integer
65     Dim dblGross As Double
66     Dim dblTaxable As Double
67     Dim dblFwt As Double
68     Dim dblFica As Double
69     Dim dblNet As Double
70
71     Double.TryParse(lstHours.SelectedItem.ToString, dblHours)
72     Double.TryParse(lstRates.SelectedItem.ToString, dblPayRate)
73     Integer.TryParse(cboAllowances.Text, intAllowances)
74
75     ' Calculate gross pay.
76     If dblHours <= 40 Then
77         dblGross = dblHours * dblPayRate
78     Else
79         dblGross = 40 * dblPayRate + (dblHours - 40) * dblPayRate * 1.5
80     End If
81
82     ' Calculate taxable wages.
83     dblTaxable = dblGross - (intAllowances * dblONE_ALLOWANCE)
84
85     ' Determine the FWT.
86     If radSingle.Checked Then
87         GetSingleFwt(dblTaxable, dblFwt)

```

```
88     Else
89         dblFwt = GetMarriedFwt(dblTaxable)
90     End If
91
92     ' Calculate FICA tax.
93     dblFica = dblGross * dblFICA_RATE
94     ' Round gross pay, FWT, and FICA tax.
95     dblGross = Math.Round(dblGross, 2)
96     dblFwt = Math.Round(dblFwt, 2)
97     dblFica = Math.Round(dblFica, 2)
98
99     ' Calculate net pay.
100    dblNet = dblGross - dblFwt - dblFica
101
102    ' Display calculated amounts.
103    lblGross.Text = dblGross.ToString("N2")
104    lblFwt.Text = dblFwt.ToString("N2")
105    lblFica.Text = dblFica.ToString("N2")
106    lblNet.Text = dblNet.ToString("N2")
107 End Sub
108
109 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
110     Me.Close()
111 End Sub
112
113 Private Sub txtName_Enter(sender As Object, e As EventArgs) Handles txtName.Enter
114     txtName.SelectAll()
115 End Sub
116
117 Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
118     ' Fill list boxes and combo box with values and select a default value in each.
119
120     For dblHours As Double = 0 To 55 Step 0.5
121         lstHours.Items.Add(dblHours.ToString("N1"))
122     Next dblHours
123     lstHours.SelectedItem = "40.0"
124
125     For dblRates As Double = 7.5 To 15.5 Step 0.5
126         lstRates.Items.Add(dblRates.ToString("N2"))
127     Next dblRates
128     lstRates.SelectedItem = "9.50"
129
130     For intAllow As Integer = 0 To 10
131         cboAllowances.Items.Add(intAllow)
```

```
132     Next intAllow
133     cboAllowances.SelectedIndex = 0
134
135 End Sub
136
137 Private Sub cboAllowances_KeyPress(sender As Object, e As KeyPressEventArgs) Handles cboAllowances.KeyPress
138     ' Accept only numbers and the Backspace key.
139
140     If (e.KeyChar < "0" OrElse e.KeyChar > "9") _
141         AndAlso e.KeyChar <> ControlChars.Back Then
142         e.Handled = True
143     End If
144 End Sub
145
146 Private Sub ClearOutput(sender As Object, e As EventArgs) _
147     Handles cboAllowances.TextChanged, lstHours.SelectedIndexChanged,
148     lstRates.SelectedIndexChanged, radMarried.CheckedChanged,
149     radSingle.CheckedChanged, txtName.TextChanged
150
151     lblGross.Text = String.Empty
152     lblFwt.Text = String.Empty
153     lblFica.Text = String.Empty
154     lblNet.Text = String.Empty
155
156 End Sub
157
158 Private Sub frmMain_FormClosing(sender As Object, e As FormClosingEventArgs) Handles Me.FormClosing
159     ' Verify that the user wants to kill the application.
160
161     Dim dlgButton As DialogResult
162     dlgButton = MessageBox.Show("Do you want to exit?", 
163                             "Cerruti Company",
164                             MessageBoxButtons.YesNo,
165                             MessageBoxIcon.Exclamation)
166     ' If the No button is selected, do not close the form.
167     If dlgButton = DialogResult.No Then
168         e.Cancel = True
169     End If
170 End Sub
171 End Class
```



## CH6\_Summary

1a. to create an **independent Sub procedure**, use the syntax:

**CH6\_F2 - Independent Sub...**

```
Private Sub ProcedureName(parameterList)
    statements...
End Sub
```

**CH6\_F5.2 - Sub** procedure vs **Function** procedure...

- **ProcedureName** -> use PascalCase
- **parameterList** = the received items are called parameters

1b. you can **call / invoke a Sub procedure** using a stand-alone statement, referred to as the **calling statement**:

**CH6\_F2 - Independent Sub...**

`ProcedureName(argumentList)`

- **argumentList** = the passed items are called arguments

1c. you pass information to a **Sub** or **Function** procedure by including the information in the **calling statement's argumentList**

- in the **parameterList** in the receiving procedure's header, you include the names of variables that will store the information passed to the procedure
- the number, data type, and order (position) of the arguments in the **argumentList** should agree with the number, data type, and order (position) of the parameters in the **parameterList**

1d. you can pass a variable **by value** to a procedure

**CH6\_F3 - Passing...**

**CH6\_F3.1 - ...**

**CH6\_F3.2 - You Do It 1: ...**

- you do this by including the **ByVal** keyword before the parameter name in the receiving procedure's **parameterList**

- because only a copy of the variable's value is passed, the receiving procedure can not access the variable

1e. you can pass a variable **by reference** to a procedure

**CH6\_F3 - Passing...**

**CH6\_F3.3 - ...**

**CH6\_F4.2 - You Do It 2: ...**

- you do this by including the **ByRef** keyword before the parameter name in the receiving procedure's **parameterList**

- because the variable's address is passed, the receiving procedure can change the contents of the variable

2. you can use the method **Math.Round** to round a number to a specified number of decimal places

- the syntax is:

`Math.Round(value, digits)`

**CH6\_F4 - Rounding Numbers - ...**

**CH6\_F4.1 - ...**

**value** = numeric expression

**digits** = optional, integer indicating decimal places

- if omitted, the method returns an integer

3a. to create a **Function** procedure, use the syntax:

CH6\_F5 - Function...

CH6\_F5.1 - Function example...

CH6\_F5.2 - Sub procedure vs Function procedure...

CH6\_F5.3 - You Do IT 3:...

```
Private Function ProcedureName(parameterList) As dataType  
    statements...  
    Return expression  
End Function
```

- **ProcedureName** -> use PascalCase
- **parameterList** = parametres (**ByVal**, **ByRef**)
- **As dataType** = data type of the return value

3b. you **invoke** a **Function** using a statement that assigns the function's return value to a variable, uses the **return** value in a calculation, or displays it

4a. you use the tool **ComboBox** in the toolbox to add a combo box to a form

- the combo box's style is specified in its **property DropDownStyle**

CH6\_A1 - Add a Combo box to the form: ComboBox tool

CH6\_A1.1 - ...

CH6\_A2 - Add Items...

CH6\_A3 -...KeyPress...

4b. to add items to a combo box, you can use either the **String Collection Editor** or the **Items collection**'s method **Add**, same like for **ListBox**: **Ist**

- the Add method's syntax is: **object.Items.Add(item)** object = name of the **cbo**, item = text you want added to the list portion of the control

4c. to automatically sort the items in the list portion of a **cbo**, set the combo box's **property Sorted** to True

4d. to determine the number of items in the list portion of a **cbo**, use the **Items collection**'s property **Count**

- the Count property's syntax is: **object.Items.Count** object = name of the **cbo**

4e. you can use any of the following properties to select a combo box item from code: **SelectedIndex**, **SelectedItem**, or **Text**

4f. to determine the item either selected in the **list portion** or entered in the **text portion**, use the combo box's property **Text**, however

- if the combo box is a **DropDownList** style, you can also use the properties: **SelectedIndex** or **SelectedItem**

4g. to process code when the value in a combo box's **Text** property changes, enter the code in the combo box's event procedure **TextChanged**

5a. you can use the method **MessageBox.Show** to display a message box that contains: text, one or more buttons, and an icon

CH6\_A10 - Professionalize Your Application's Interface: Message Box (MessageBox.Show method)

CH6\_A10.1 - Message Box (MessageBox.Show method) - confirmation to kill the Cerruti Company Payroll application example 09

- the method's syntax: **MessageBox.Show(text, caption, buttons, icon, defaultButton)**

5b. to process code when a form is about to be closed, enter the code in the form's event procedure **FormClosing**

- the **FormClosing event** occurs when:
  - a). the user clicks the button **Close** on a form's title bar, or
  - b). the computer processes the statement: **Me.Close()**

5c. to prevent a form from being closed, set the **Cancel** property of the **FormClosing** event procedure's **e** parameter to **True**      (**e.Cancel = True**)

## CH6 Key Terms

- **Arguments** - the items of information in a **calling statement**; items passed to a **Sub procedure**, **Function**, or method
- **Calling statement** - a statement that invokes / calls an **independent Sub procedure** or **Function procedure**
- **Cancel property** - a property of the **e** parameter in the form's **FormClosing** event procedure; when set to **True**, it prevents the form from closing
- **Combo box (cbo)** - a control that offers the user a list of choices and also has a text field that may or may not be editable
- **DropDownStyle property** - determines the style of a combo box
- **Event procedures / Event-handling Sub procedures** - **Sub** procedures that are processed only when a specific event occurs
- **FormClosing event** - occurs when a form is about to be closed, which can happen as a result of the computer processing the statement **Me.Close()** or
  - or the user clicking the button **Close** on the form's title bar
- **Function / Function procedure** - a procedure that returns a value after performing its assigned task
- **Independent Sub procedures / Sub procedures** - procedures that are not connected to any object and event
  - the procedure is processed only when called / invoked from code
- **Line continuation character** - an underscore **\_** that is immediately preceded by a space and located at the end of a physical line of code in the Code Editor window
- **Math.Round method** - rounds a numeric value to a specific number of decimal places

- **MessageBox.Show method** - displays a message box that contains: **text**, **button / -s**, and an **icon**
  - allows an application to communicate with the user while the application is running
- **Parameters** - memory locations (variables) declared in a procedure header; accepts the information passed to the procedure
- **PascalCase** - used when naming **Sub procedures** and **Functions**
  - the practice of Capitalizing the first letter in the name and the first letter of each subsequent word in the name
- **Passing by reference / ByRef** - refers to the process of passing a variable's address to a procedure so that the value in the variable can be changed
- **Passing by value / ByVal** - refers to the process of passing a copy of a variable's value to a procedure so that the value in the variable can not be changed
- **Return statement** - the Visual Basic statement that returns a **Function**'s value to the statement that invoked the **Function**
- **Sub procedures / Independent Sub procedures** - procedures that are not connected to any object and event
  - the procedure is processed only when called / invoked from code

## 12.History Solution-Functions\_EXERCISE 1\_introductory

- change **Independent Sub** procedures to **Functions - Private Sub** to **Private Function**
- **Private Function...Return, ByVal, Select Case, Case Is, rad.Checked, txt\_Enter, Private Sub** - independent **Sub** procedure, **txt\_KeyPress, ControlChars**

## 13.Modified Gross Solution\_EXERCISE 2\_introductory

- **Private Sub** - independent **Sub** procedure, **ByVal, ByRef, rad.Checked, Private Sub Clear, txt\_KeyPress, ControlChars**

## 14.Gross Solution-Functions\_EXERCISE 3\_introductory

- change **Independent Sub** procedures to **Functions - Private Sub** to **Private Function**
- **Private Function, ByVal, Return, rad.Checked, txt\_Enter, Private Sub Clear, txt\_KeyPress, ControlChars**

## 15.Modified Cerruti Solution\_EXERCISE 4\_introductory

- **Private Function, Return, Private Sub, ByVal, ByRef, rad.Checked, Math.Round, Select Case, Case Is, Case Else, txt\_Enter, frmMain\_Load, For...Step...Next, lst.Items.Add, lst.SelectedItem, cbo.Items.Add, cbo.SelectedIndex, cbo\_KeyPress, ControlChars, Private Sub Clear, frmMain\_FormClosing, dlgButton As DialogResult, MessageBox.Show, DialogResult.No, \_**
- **Private Function, Return, chk.Checked, Private Sub Clear, chk.CheckedChanged**

## 16.Seminars Solution-CheckBox\_EXERCISE 5\_introductory

- **Private Sub, ByRef, Select Case, rad.Checked, Private Sub Clear, rad.CheckedChanged**

## 18.Donut Solution\_EXERCISE 7\_intermediate

- **Private Sub, ByRef, ElseIf, rad.Checked, Private Function, Return, Private Sub Clear, rad.CheckedChanged, frmMain\_FormClosing, dlgButton As DialogResult, MessageBox.Show**

## 19.Mats Solution\_EXERCISE 8\_intermediate

- **Private Function...Return, ElseIf, Private Sub, ByRef, rad.Checked, chk.Checked, Private Sub Clear, rad.CheckedChanged, chk.CheckedChanged, frmMain\_FormClosing, dlgButton As DialogResult, MessageBox.Show, DialogResult.No, e.Cancel**

## 20.Translator Solution\_EXERCISE 9\_intermediate

- **frmMain\_Load, cbo.Items.Add, cbo.SelectedIndex, Private Function...Return, ElseIf, rad.Checked, frmMain\_FormClosing, dlgButton As DialogResult, MessageBox.Show, DialogResult.No, e.Cancel, Private Sub Clear, rad.CheckedChanged**

## 21.Translator Solution-Sub\_EXERCISE 10\_intermediate

- **frmMain\_Load, cbo.Items.Add, cbo.SelectedIndex, Private Sub, ByRef, rad.Checked, ElseIf, frmMain\_FormClosing, dlgButton As DialogResult, MessageBox.Show, DialogResult.No, e.Cancel, Private Sub Clear, rad.CheckedChanged**

## 22.Cable Direct Solution\_EXERCISE 11\_advanced

- **frmMain\_Load, For...To...Next, lst.Items.Add, lst.SelectedIndex, Private Function, dlgButton As DialogResult, MessageBox.Show, ElseIf, AndAlso, Private Sub Clear, rad.CheckedChanged, lst.SelectedIndexChanged, frmMain\_FormClosing, e.Cancel**

## 23.Wallpaper Solution\_EXERCISE 12\_advanced

- **frmMain\_Load, For...To...Next, cbo.Items.Add, cbo.SelectedText, Private Sub, ByVal, ByRef, Math.Round, Private Sub Clear, cbo.TextChanged**

## 24.OnYourOwn Solution\_EXERCISE 13\_not done

- no tengo los náspados

## 25.FixIt Solution\_EXERCISE 14

- **Private Function...Return, ByVal, frmMain\_Load, For...To...Step...Next, cbo.Items.Add, cbo.SelectedIndex, Private Sub Clear, cbo.TextChanged**

- change **Independent Sub** procedures to **Functions - Private Sub** to **Private Function**  
 - **Private Function...Return, ByVal, Select Case, Case Is, rad.Checked, txt\_Enter,**  
**Private Sub** - independent **Sub** procedure, **txt\_KeyPress, ControlChars**

1. In this exercise, you modify the History Grade application from this chapter's Focus lesson. Use Windows to make a copy of the History Solution folder. Rename the copy History Solution-Functions. Open the History Solution.sln file contained in the History Solution-Functions folder. Modify the btnDisplay\_Click procedure so that it uses two functions named GetGrade101 and GetGrade201 to get the appropriate grade; the procedure should then display the grade in the lblGrade control. Change the two independent Sub procedures to functions that return the appropriate grade to the statements that invoke them in the btnDisplay\_Click procedure. Each function should contain a parameter that accepts the total points passed to it. Save the solution and then start and test the application.

```

1  ' Name:      History Project
2  ' Purpose:    Display a student's grade.
3  ' Programmer: <your name> on <current date>
4
5  Option Explicit On
6  Option Strict On
7  Option Infer Off
8
9  Public Class frmMain
10     Private Function GetGrade101(ByVal intPoints2 As Integer) As String
11         'Calculate the grade for History 101.
12         Dim strGrade As String
13
14         Select Case intPoints2
15             Case Is >= 90
16                 strGrade = "A"
17             Case Is >= 80
18                 strGrade = "B"
19             Case Is >= 70
20                 strGrade = "C"
21             Case Is >= 60
22                 strGrade = "D"
23             Case Else
24                 strGrade = "F"
25         End Select
26
27         Return strGrade
28     End Function

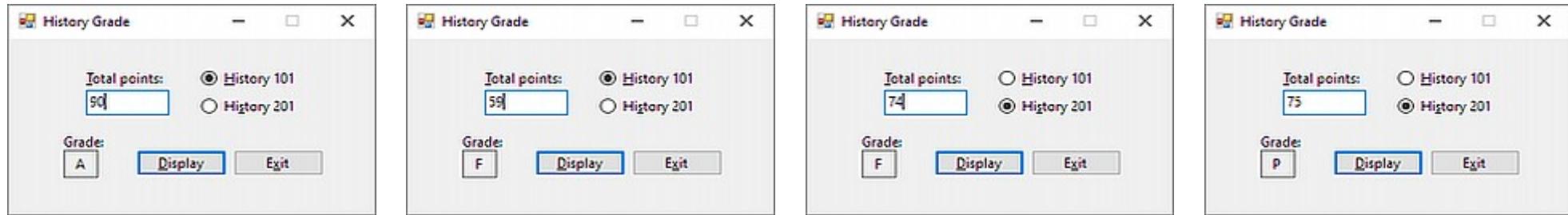
```

```
29
30     Private Function GetGrade201(ByVal intPoints2 As Integer) As String
31         'Calculate the grade for History 201.
32         Dim strGrade As String
33         If intPoints2 >= 75 Then
34             strGrade = "P"
35         Else
36             strGrade = "F"
37         End If
38
39         Return strGrade
40     End Function
41
42     Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
43         'Calls Functions to calculate and send a grade.
44         Dim intPoints As Integer
45         Dim strGrade2 As String
46         Integer.TryParse(txtPoints.Text, intPoints)
47
48         If radHis101.Checked Then
49             strGrade2 = GetGrade101(intPoints)
50
51         Else
52             strGrade2 = GetGrade201(intPoints)
53         End If
54
55         lblGrade.Text = strGrade2
56
57     End Sub
58
59     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
60         Me.Close()
61     End Sub
62
63     Private Sub txtPoints_Enter(sender As Object, e As EventArgs) Handles txtPoints.Enter
64         txtPoints.SelectAll()
65     End Sub
66
67     Private Sub ClearGrade(sender As Object, e As EventArgs) Handles txtPoints.TextChanged,
68                                     radHis101.CheckedChanged, radHis201.CheckedChanged
69
70         lblGrade.Text = String.Empty
71     End Sub
72
```

```

73     Private Sub txtPoints_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtPoints.KeyPress
74         ' Accept only numbers and the Backspace key
75
76         If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
77             e.Handled = True
78         End If
79     End Sub
80
81 End Class

```



### Chap06\Exercise1

#### 13.Modified Gross Solution\_EXERCISE 2\_introductory

- **Private Sub** - independent **Sub** procedure, **ByVal**, **ByRef**, **rad.Checked**,  
**Private Sub** **Clear**, **txt\_KeyPress**, **ControlChars**

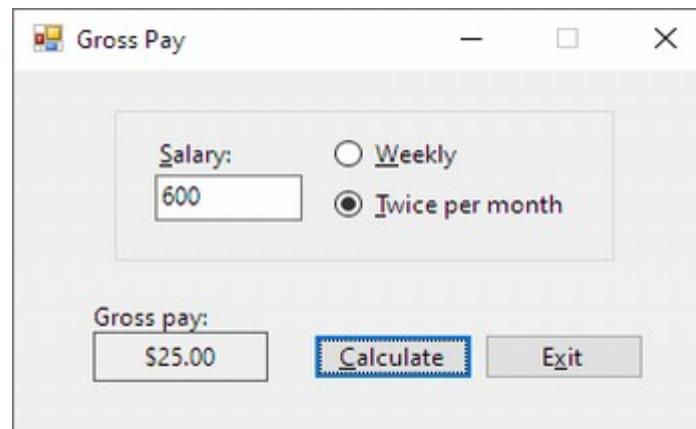
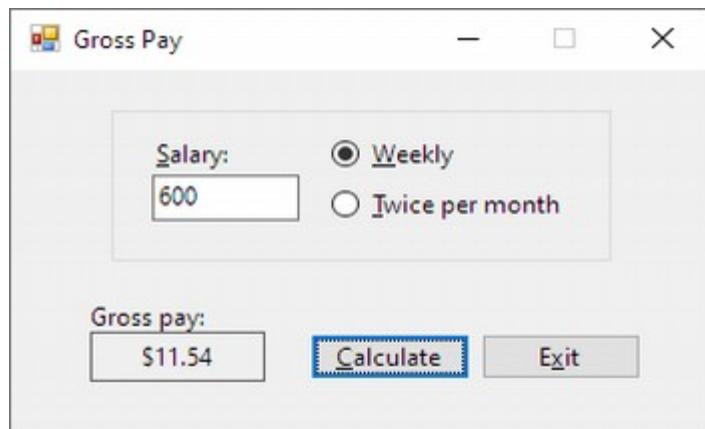
2. In this exercise, you modify the Gross Pay application from this chapter's Focus lesson. Use Windows to make a copy of the Gross Solution folder. Rename the copy Modified Gross Solution. Open the Gross Solution.sln file contained in the Modified Gross Solution folder. Change the names of the two independent Sub procedures to CalcWeekly and CalcTwicePerMonth. Modify the code so that the btnCalc\_Click procedure (rather than the two independent Sub procedures) displays the gross pay in the lblGross control. Save the solution and then start and test the application.

```
1  ' Name:      Gross Project
2  ' Purpose:    Display an employee's gross pay.
3  ' Programmer: <your name> on <current date>
4
5  Option Explicit On
6  Option Strict On
7  Option Infer Off
8
9  Public Class frmMain
10     ' 2x Independent Sub procedures:
11     Private Sub CalcWeekly(ByVal intWeekly As Integer, ByRef dblPay As Double) '_DisplayWeekly
12         'only calculate the weekly gross pay.
13         'Dim dblPay As Double
14         dblPay = intWeekly / 52
15         'lblGross.Text = dblPay.ToString("C2")
16     End Sub
17
18     Private Sub CalcTwicePerMonth(ByVal intTwicePerMonth As Integer, ByRef dblPay As Double) '_DisplayTwicePerMonth
19         'only calculate the twice per month gross pay.
20         'Dim dblPay As Double
21         dblPay = intTwicePerMonth / 24
22         'lblGross.Text = dblPay.ToString("C2")
23     End Sub
24
25     Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
26         ' Calls independent Sub procedures to calculate and display the gross pay.
27         Dim intSalary As Integer
28         Dim dblGross As Double
29         Integer.TryParse(txtSalary.Text, intSalary)
30
31         If radWeekly.Checked Then
32             CalcWeekly(intSalary, dblGross)
33             lblGross.Text = dblGross.ToString("C2")
34         Else
35             CalcTwicePerMonth(intSalary, dblGross)
36             lblGross.Text = dblGross.ToString("C2")
37         End If
38     End Sub
39
40     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
41         Me.Close()
42     End Sub
43
```

```

44  Private Sub txtSalary_Enter(sender As Object, e As EventArgs) Handles txtSalary.Enter
45      txtSalary.SelectAll()
46  End Sub
47
48  Private Sub ClearGross(sender As Object, e As EventArgs) Handles txtSalary.TextChanged, radWeekly.CheckedChanged,
49   radTwicePerMonth.CheckedChanged
50      lblGross.Text = String.Empty
51  End Sub
52
53  Private Sub txtSalary_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtSalary.KeyPress
54      ' Accept only numbers and the Backspace key
55
56      If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
57          e.Handled = True
58      End If
59  End Sub
60 End Class

```



## Chap06\\_Exercise1

### 14.Gross Solution-Functions\_EXERCISE 3\_introductory

- change **Independent Sub** procedures to **Functions - Private Sub** to **Private Function**  
- **Private Function**, **ByVal**, **Return**, **rad.Checked**, **txt\_Enter**, **Private Sub** **Clear**,  
**txt\_KeyPress**, **ControlChars**

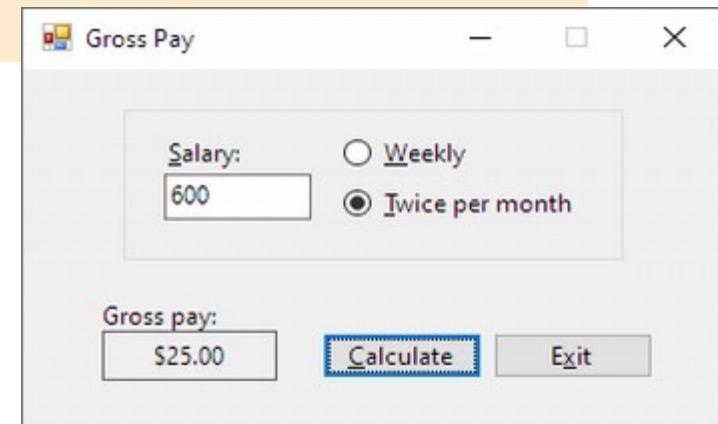
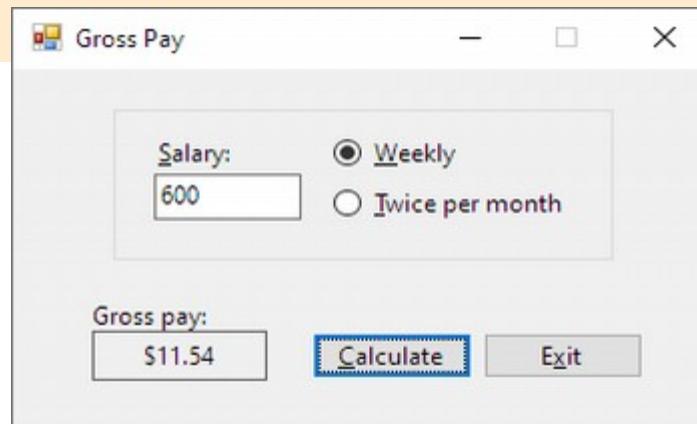
3. In this exercise, you modify the Gross Pay application from this chapter's Focus lesson. Use Windows to make a copy of the Gross Solution folder. Rename the copy Gross Solution-Functions. Open the Gross Solution.sln file contained in the Gross Solution-Functions folder. Change the two independent Sub procedures to functions named GetWeekly and GetTwicePerMonth. Modify the code so that the btnCalc\_Click procedure (rather than the two functions) displays the gross pay in the lblGross control. Save the solution and then start and test the application.

```
1  ' Name:      Gross Project
2  ' Purpose:    Display an employee's gross pay.
3  ' Programmer: <your name> on <current date>
4
5  Option Explicit On
6  Option Strict On
7  Option Infer Off
8
9  Public Class frmMain
10   ' 2x Functions:
11   Private Function GetWeekly(ByVal intWeekly As Integer) As Double
12     'Calculate the weekly gross pay.
13     Dim dblPay As Double
14     dblPay = intWeekly / 52
15     'lblGross.Text = dblPay.ToString("C2")
16     Return dblPay
17   End Function
18
19   Private Function GetTwicePerMonth(ByVal intTwicePerMonth As Integer) As Double
20     ' Display the twice per month gross pay.
21     Dim dblPay As Double
22     dblPay = intTwicePerMonth / 24
23     'lblGross.Text = dblPay.ToString("C2")
24     Return dblPay
25   End Function
26
```

```

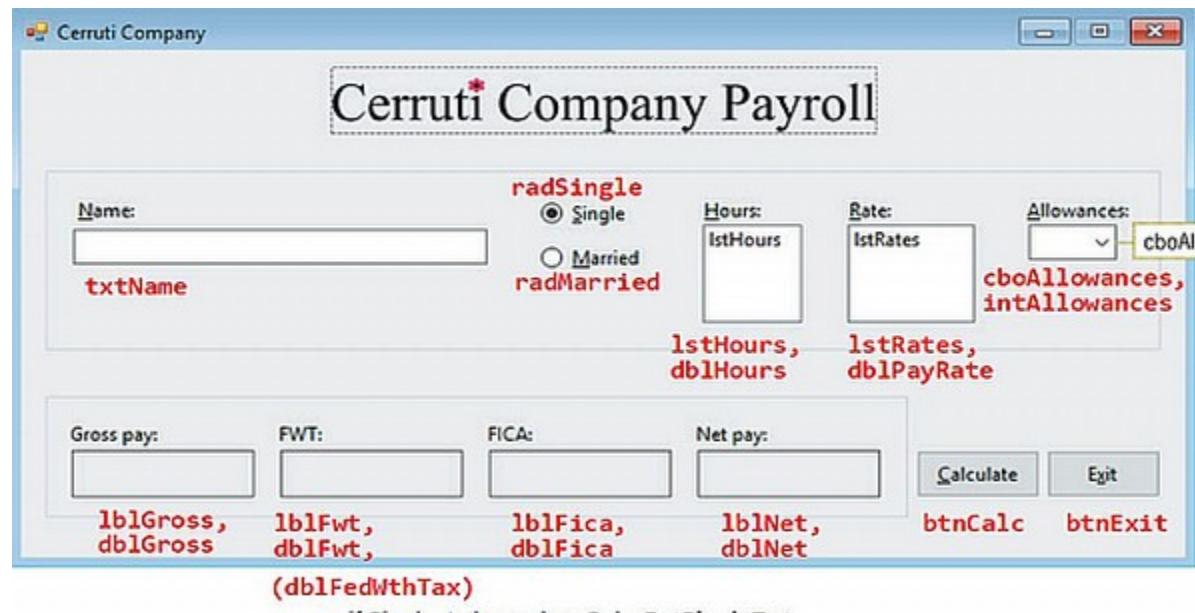
27  Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
28      ' Calls Functions to calculate and display the gross pay.
29      Dim intSalary As Integer
30      Integer.TryParse(txtSalary.Text, intSalary)
31
32      If radWeekly.Checked Then
33          lblGross.Text = GetWeekly(intSalary).ToString("C2")
34      Else
35          lblGross.Text = GetTwicePerMonth(intSalary).ToString("C2")
36      End If
37  End Sub
38
39  Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
40      Me.Close()
41  End Sub
42
43  Private Sub txtSalary_Enter(sender As Object, e As EventArgs) Handles txtSalary.Enter
44      txtSalary.SelectAll()
45  End Sub
46
47  Private Sub ClearGross(sender As Object, e As EventArgs) Handles txtSalary.TextChanged, radWeekly.CheckedChanged,
48   radTwicePerMonth.CheckedChanged
49      lblGross.Text = String.Empty
50  End Sub
51
52  Private Sub txtSalary_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtSalary.KeyPress
53      ' Accept only numbers and the Backspace key
54
55      If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
56          e.Handled = True
57      End If
58  End Sub
59 End Class

```



```
- Private Function, Return, Private Sub, ByVal, ByRef, rad.Checked, Math.Round,
  Select Case, Case Is, Case Else, txt_Enter, frmMain_Load, For...Step...Next,
  lst.Items.Add, lst.SelectedItem, cbo.Items.Add, cbo.SelectedIndex, cbo_KeyPress,
  ControlChars, Private Sub Clear, frmMain_FormClosing, dlgButton As DialogResult,
  MessageBox.Show, DialogResult.No, _
```

4. In this exercise, you modify the Cerruti Company application from this chapter's Apply lesson. Use Windows to make a copy of the Cerruti Solution folder. Rename the copy Modified Cerruti Solution. Open the Cerruti Solution.sln file contained in the Modified Cerruti Solution folder. Change the GetSingleFwt Sub procedure to a function. Then, change the GetMarriedFwt function to a Sub procedure. Make the necessary modifications to the btnCalc\_Click procedure. Save the solution and then start and test the application.



```

1  ' Name:      Cerruti Project
2  ' Purpose:    Displays an employee's gross pay, taxes, and net pay.
3  ' Programmer: <your name> on <current date>
4
5  Option Explicit On
6  Option Strict On
7  Option Infer Off
8
9  Public Class frmMain
10   'Function and Independent Sub procedure:
11   Private Function GetSingleFwt(ByVal dblTaxPay As Double) As Double

```

```

12     Dim dblFedWthTax As Double
13
14     Select Case dblTaxPay
15         Case Is <= 44
16             dblFedWthTax = 0
17         Case Is <= 224
18             dblFedWthTax = 0.1 * (dblTaxPay - 44)
19         Case Is <= 774
20             dblFedWthTax = 18 + 0.15 * (dblTaxPay - 224)
21         Case Is <= 1812
22             dblFedWthTax = 100.5 + 0.25 * (dblTaxPay - 774)
23         Case Is <= 3730
24             dblFedWthTax = 360 + 0.28 * (dblTaxPay - 1812)
25         Case Is <= 8058
26             dblFedWthTax = 897.04 + 0.33 * (dblTaxPay - 3730)
27         Case Is <= 8090
28             dblFedWthTax = 2325.28 + 0.35 * (dblTaxPay - 8058)
29         Case Else
30             dblFedWthTax = 2336.48 + 0.396 * (dblTaxPay - 8090)
31     End Select
32     Return dblFedWthTax
33 End Function
34
35 Private Sub GetMarriedFwt(ByVal dblTaxPay As Double, ByRef dblFedWthTax As Double)
36
37     Select Case dblTaxPay
38         Case Is <= 166
39             dblFedWthTax = 0
40         Case Is <= 525
41             dblFedWthTax = 0.1 * (dblTaxPay - 166)
42         Case Is <= 1626
43             dblFedWthTax = 35.9 + 0.15 * (dblTaxPay - 525)
44         Case Is <= 3111
45             dblFedWthTax = 201.05 + 0.25 * (dblTaxPay - 1626)
46         Case Is <= 4654
47             dblFedWthTax = 572.3 + 0.28 * (dblTaxPay - 3111)
48         Case Is <= 8180
49             dblFedWthTax = 1004.34 + 0.33 * (dblTaxPay - 4654)
50         Case Is <= 9218
51             dblFedWthTax = 2167.92 + 0.35 * (dblTaxPay - 8180)
52         Case Else
53             dblFedWthTax = 2531.22 + 0.396 * (dblTaxPay - 9218)
54     End Select
55 End Sub

```

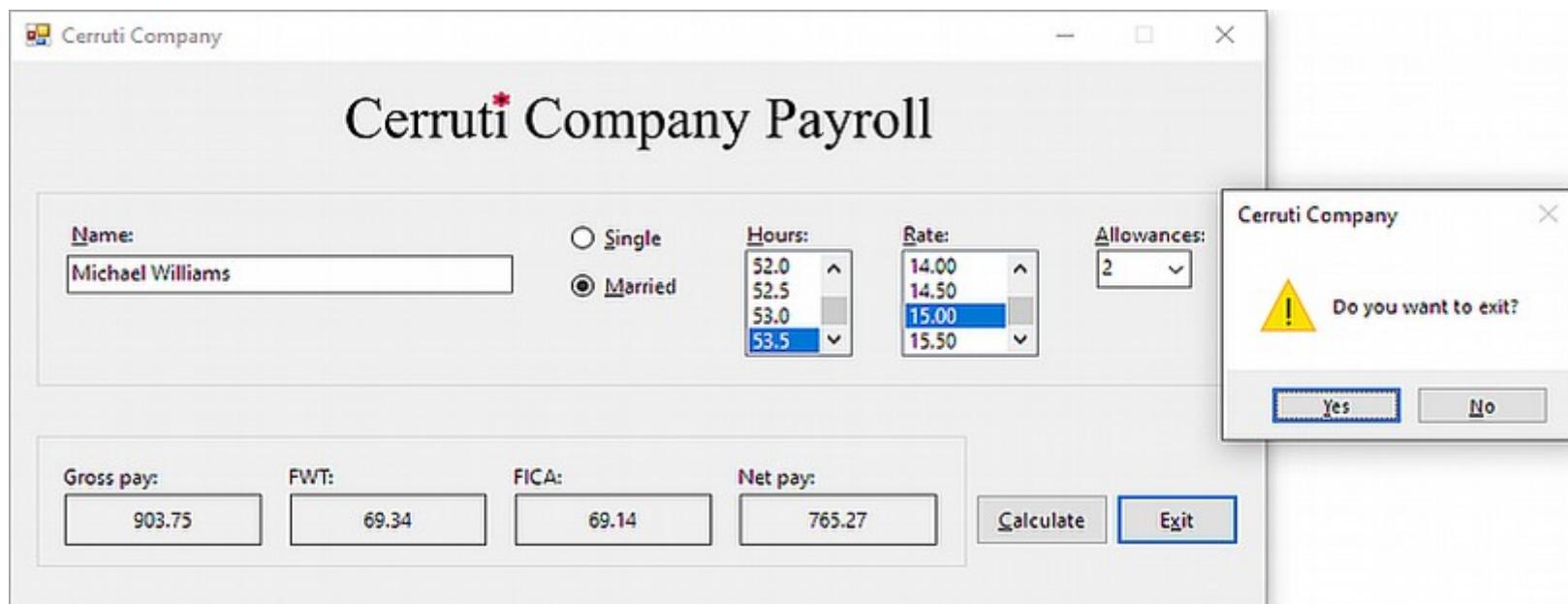
```
56  Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
57      ' Calculates and displays gross pay, taxes, and net pay.
58
59      Const dblONE_ALLOWANCE As Double = 77.9
60      Const dblFICA_RATE As Double = 0.0765
61      Dim dblHours As Double
62      Dim dblPayRate As Double
63      Dim intAllowances As Integer
64      Dim dblGross As Double
65      Dim dblTaxable As Double
66      Dim dblFwt As Double
67      Dim dblFica As Double
68      Dim dblNet As Double
69
70      Double.TryParse(lstHours.SelectedItem.ToString, dblHours)
71      Double.TryParse(lstRates.SelectedItem.ToString, dblPayRate)
72      Integer.TryParse(cboAllowances.Text, intAllowances)
73
74      ' Calculate gross pay.
75      If dblHours <= 40 Then
76          dblGross = dblHours * dblPayRate
77      Else
78          dblGross = 40 * dblPayRate + (dblHours - 40) * dblPayRate * 1.5
79      End If
80
81      ' Calculate taxable wages.
82      dblTaxable = dblGross - (intAllowances * dblONE_ALLOWANCE)
83
84      ' Determine the FWT.
85      If radSingle.Checked Then
86          dblFwt = GetSingleFwt(dblTaxable)
87      Else
88          GetMarriedFwt(dblTaxable, dblFwt)
89      End If
90
91      ' Calculate FICA tax.
92      dblFica = dblGross * dblFICA_RATE
93      ' Round gross pay, FWT, and FICA tax.
94      dblGross = Math.Round(dblGross, 2)
95      dblFwt = Math.Round(dblFwt, 2)
96      dblFica = Math.Round(dblFica, 2)
97
98      ' Calculate net pay.
99      dblNet = dblGross - dblFwt - dblFica
```

```
100      ' Display calculated amounts.
101      lblGross.Text = dblGross.ToString("N2")
102      lblFwt.Text = dblFwt.ToString("N2")
103      lblFica.Text = dblFica.ToString("N2")
104      lblNet.Text = dblNet.ToString("N2")
105
106  End Sub
107
108  Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
109      Me.Close()
110  End Sub
111
112  Private Sub txtName_Enter(sender As Object, e As EventArgs) Handles txtName.Enter
113      txtName.SelectAll()
114  End Sub
115
116  Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
117      ' Fill list boxes and combo box with values and select a default value in each.
118
119      For dblHours As Double = 0 To 55 Step 0.5
120          lstHours.Items.Add(dblHours.ToString("N1"))
121      Next dblHours
122      lstHours.SelectedItem = "40.0"
123
124      For dblRates As Double = 7.5 To 15.5 Step 0.5
125          lstRates.Items.Add(dblRates.ToString("N2"))
126      Next dblRates
127      lstRates.SelectedItem = "9.50"
128
129      For intAllow As Integer = 0 To 10
130          cboAllowances.Items.Add(intAllow)
131      Next intAllow
132      cboAllowances.SelectedIndex = 0
133
134  End Sub
135
136  Private Sub cboAllowances_KeyPress(sender As Object, e As KeyPressEventArgs) Handles cboAllowances.KeyPress
137      ' Accept only numbers and the Backspace key.
138
139      If (e.KeyChar < "0" OrElse e.KeyChar > "9") _
140          AndAlso e.KeyChar <> ControlChars.Back Then
141          e.Handled = True
142      End If
143  End Sub
```

```

144
145     Private Sub ClearOutput(sender As Object, e As EventArgs) _
146         Handles cboAllowances.TextChanged, lstHours.SelectedIndexChanged,
147         lstRates.SelectedIndexChanged, radMarried.CheckedChanged,
148         radSingle.CheckedChanged, txtName.TextChanged
149
150     lblGross.Text = String.Empty
151     lblFwt.Text = String.Empty
152     lblFica.Text = String.Empty
153     lblNet.Text = String.Empty
154
155 End Sub
156
157 Private Sub frmMain_FormClosing(sender As Object, e As FormClosingEventArgs) Handles Me.FormClosing
158     ' Verify that the user wants to kill the application.
159
160     Dim dlgButton As DialogResult
161     dlgButton = MessageBox.Show("Do you want to exit?", "Cerruti Company",
162                               MessageBoxButtons.YesNo, MessageBoxIcon.Exclamation)
163
164     ' If the No button is selected, do not close the form.
165     If dlgButton = DialogResult.No Then
166         e.Cancel = True
167     End If
168 End Sub
169
End Class

```

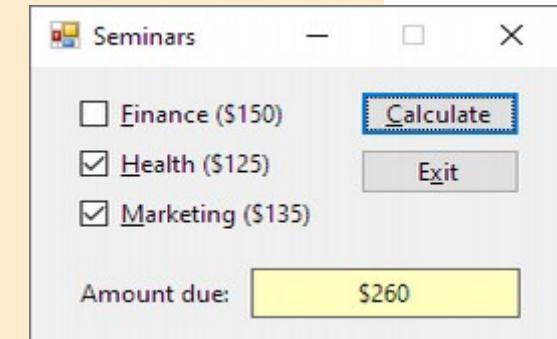
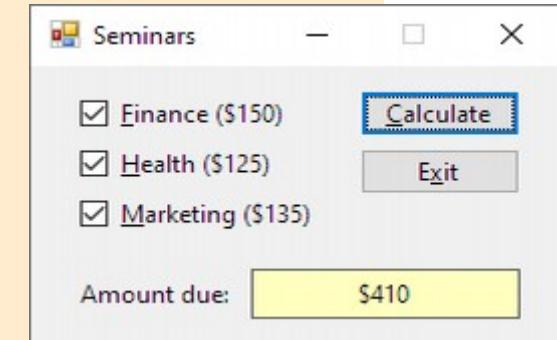


5. In this exercise, you modify one of the Seminars applications from Chapter 4's Apply lesson. Open the Seminars Solution.sln file contained in the Seminars Solution-CheckBox folder. Create an event-handling Sub procedure named ClearAmountDue and associate it with each check box's CheckChanged event. Then, create a function that determines which (if any) check boxes are selected and then adds the associated fee to the total due. The function should return the total due to the statement that invoked it. Also, make the necessary modifications to the btnCalc\_Click procedure's code. Save the solution and then start and test the application.

```

1  ' Name:      Seminars Project
2  ' Purpose:    Calculate and display the amount due.
3  ' Programmer: <your name> on <current date>
4
5  Option Explicit On
6  Option Strict On
7  Option Infer Off
8
9  Public Class frmMain
10     Private Function IsChecked() As Integer
11         Dim intAmount As Integer
12
13         If chkFinance.Checked = True Then
14             intAmount = intAmount + 150
15         End If
16
17         If chkHealth.Checked = True Then
18             intAmount = intAmount + 125
19         End If
20
21         If chkMarketing.Checked = True Then
22             intAmount = intAmount + 135
23         End If
24
25         Return intAmount
26     End Function
27
28     Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
29         ' Calculate and display the amount due.
30         lblAmountDue.Text = IsChecked().ToString("C0")
31     End Sub

```



```

32
33     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
34         Me.Close()
35     End Sub
36
37     Private Sub ClearAmountDue_CheckedChanged(sender As Object, e As EventArgs) Handles chkFinance.CheckedChanged,
38   chkHealth.CheckedChanged, chkMarketing.CheckedChanged
39         lblAmountDue.Text = String.Empty
40         lblAmountDue.Text = String.Empty
41         lblAmountDue.Text = String.Empty
42     End Sub
43
44 End Class

```

## Chap06\Exercise1

### 17.Seminars Solution-RadioButton\_EXERCISE 6\_introductory

- **Private Sub**, **ByRef**, **Select Case**, **rad.Checked**, **Private Sub** **Clear**,  
**rad.CheckedChanged**

6. In this exercise, you modify one of the Seminars applications from Chapter 4's Apply lesson. Open the Seminars Solution.sln file contained in the Seminars Solution-RadioButton folder. Create an event-handling Sub procedure named ClearDue and associate it with each radio button's CheckedChanged event. Then, create a Sub procedure that determines the appropriate fee, which is based on the selected radio button. Also, make the necessary modifications to the btnCalc\_Click procedure's code. Save the solution and then start and test the application.

```

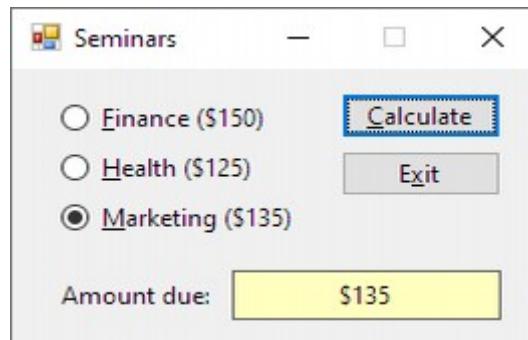
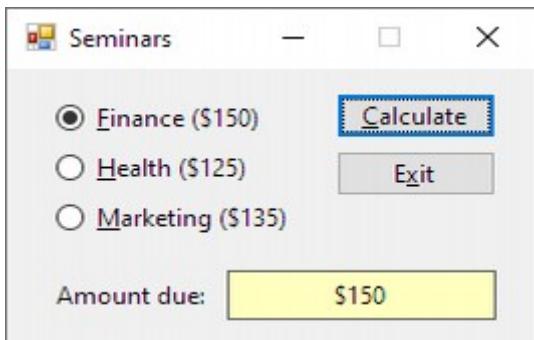
1  ' Name:      Seminars Project
2  ' Purpose:    Display the amount due.
3  ' Programmer: <your name> on <current date>
4
5  Option Explicit On
6  Option Strict On
7  Option Infer Off
8

```

```

9  Public Class frmMain
10     Private Sub Fee(ByRef intAmountDue2 As Integer)
11         Select Case True
12             Case radFinance.Checked
13                 intAmountDue2 = 150
14             Case radHealth.Checked
15                 intAmountDue2 = 125
16             Case radMarketing.Checked
17                 intAmountDue2 = 135
18         End Select
19     End Sub
20
21     Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
22         ' Display the amount due.
23         Dim intAmountDue As Integer
24         Fee(intAmountDue)
25         lblAmountDue.Text = intAmountDue.ToString("C0")
26     End Sub
27
28     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
29         Me.Close()
30     End Sub
31
32     Private Sub ClearDue_CheckedChanged(sender As Object, e As EventArgs) Handles radFinance.CheckedChanged,
33   radHealth.CheckedChanged, radMarketing.CheckedChanged
34         lblAmountDue.Text = String.Empty
35         lblAmountDue.Text = String.Empty
36         lblAmountDue.Text = String.Empty
37     End Sub
38 End Class

```



- Private Sub, ByRef, ElseIf, rad.Checked, Private Function, Return, Private Sub Clear, rad.CheckedChanged, frmMain\_FormClosing, dlgButton As DialogResult, MessageBox.Show

7. The Donut Shoppe sells four varieties of doughnuts: Glazed (\$1.05), Sugar (\$1.05), Chocolate (\$1.25), and Filled (\$1.50). It also sells regular coffee (\$1.50) and cappuccino (\$2.75). The store manager wants you to create an application that displays a customer's subtotal, 6% sales tax, and total due. Create a Windows Forms application. Use the following names for the project and solution, respectively: Donut Project and Donut Solution. Save the application in the VB2017\Chap06 folder. Create the interface shown in Figure 6-57. When coding the application, use one independent Sub procedure to determine the subtotal, which is the total cost without the sales tax. Use a function to determine the sales tax. Use an event-handling Sub procedure to clear the output. Save the solution and then start and test the application.

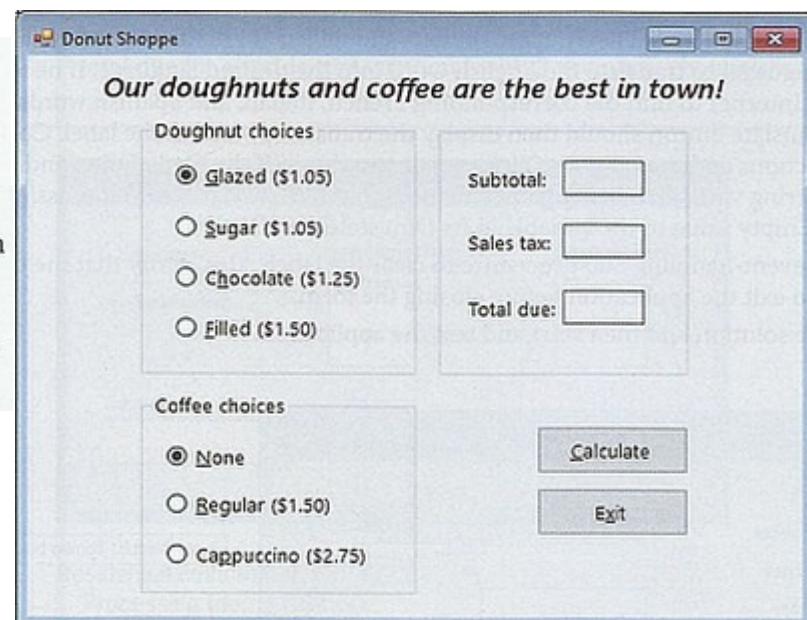


Figure 6-57 Interface for Exercise 7

```

1 Option Explicit On
2 Option Strict On
3 Option Infer Off
4 'use Sub to determine (Total - tax 6%)
5 'use Function to determine Tax (sales tax = 6% = 0.06 from Total)
6 'use event-handling Sub to clear outputs
7 '    doughnuts: Glazed = 1.05, Sugar = 1.05, Chocolate = 1.25, Filled = 1.50
8 '    coffee: none = 0, Regular = 1.50, Cappuccino = 2.75
9
10 Public Class frmMain
11     Public Const dbl6Tax As Double = 0.06
12
13     Private Sub Subtotal(ByRef dblSubtotal2 As Double) 'determine Sales Tax
14         If rad1Glazed.Checked Then
15             dblSubtotal2 += 1.05 - (1.05 * dbl6Tax)
16         ElseIf rad2Sugar.Checked Then
17             dblSubtotal2 += 1.05 - (1.05 * dbl6Tax)
18         ElseIf rad3Chocolate.Checked Then
19             dblSubtotal2 += 1.25 - (1.25 * dbl6Tax)
20         ElseIf rad4Filled.Checked Then
21             dblSubtotal2 += 1.5 - (1.5 * dbl6Tax)
22         End If

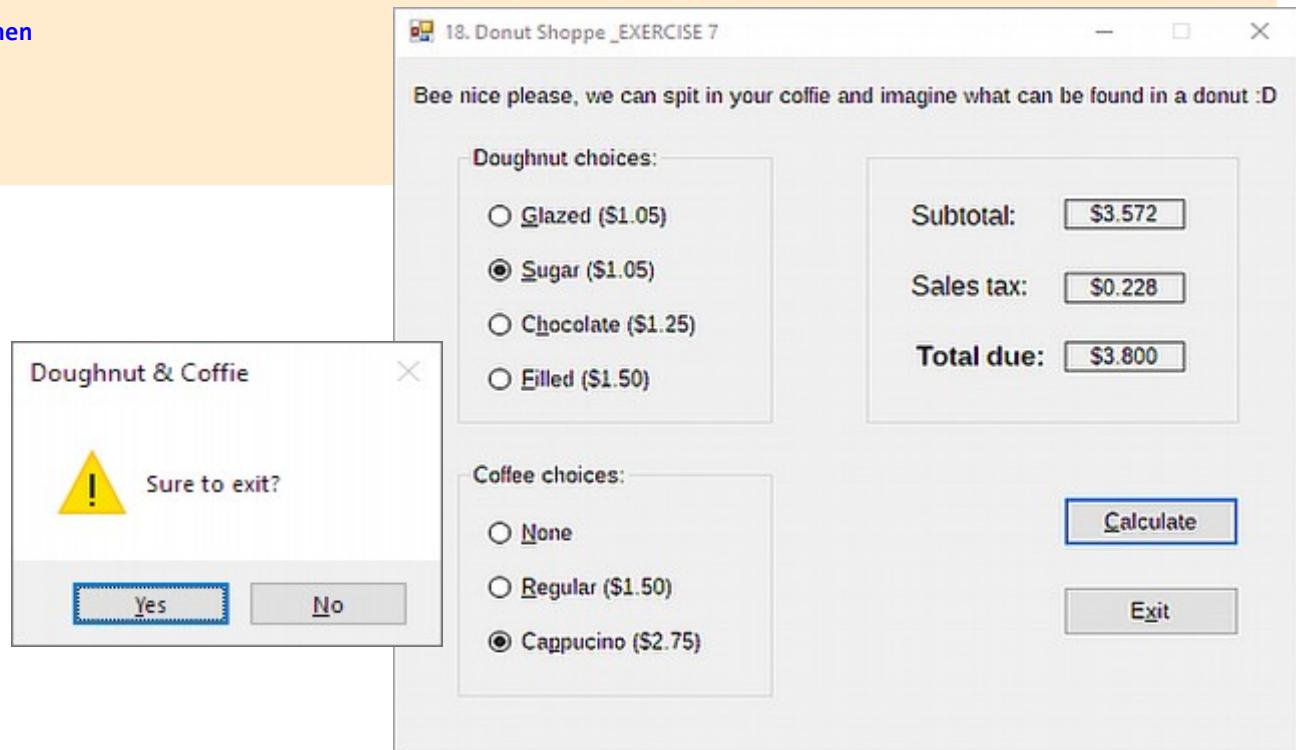
```

```
23
24     If rad5None.Checked Then
25         dblSubtotal2 += 0
26     ElseIf rad6Regular.Checked Then
27         dblSubtotal2 += 1.5 - (1.5 * dbl6Tax)
28     ElseIf rad7Cappuccino.Checked Then
29         dblSubtotal2 += 2.75 - (2.75 * dbl6Tax)
30     End If
31 End Sub
32
33 Private Function Tax() As Double
34     Dim dblTax2 As Double
35
36     If rad1Glazed.Checked Then
37         dblTax2 += 1.05 * dbl6Tax
38     ElseIf rad2Sugar.Checked Then
39         dblTax2 += 1.05 * dbl6Tax
40     ElseIf rad3Chocolate.Checked Then
41         dblTax2 += 1.25 * dbl6Tax
42     ElseIf rad4Filled.Checked Then
43         dblTax2 += 1.5 * dbl6Tax
44     End If
45
46     If rad5None.Checked Then
47         dblTax2 += 0
48     ElseIf rad6Regular.Checked Then
49         dblTax2 += 1.5 * dbl6Tax
50     ElseIf rad7Cappuccino.Checked Then
51         dblTax2 += 2.75 * dbl6Tax
52     End If
53
54     Return dblTax2
55 End Function
56
57 Private Sub btnCalculate_Click(sender As Object, e As EventArgs) Handles btnCalculate.Click
58     Dim dblSubtotal As Double
59     Dim dblTotal As Double
60     Dim dblTax As Double
61
62     Subtotal(dblSubtotal)
63     dblTax = Tax()
64     dblTotal = dblSubtotal + dblTax
65
```

```

66     lbl1Subtotal.Text = dblSubtotal.ToString("C3")
67     lbl2Tax.Text = dblTax.ToString("C3")
68     lbl3Total.Text = dblTotal.ToString("C3")
69 End Sub
70
71 Private Sub ClearOutputs(sender As Object, e As EventArgs) Handles rad1Glazed.CheckedChanged, rad2Sugar.CheckedChanged,
72     rad3Chocolate.CheckedChanged, rad4Filled.CheckedChanged, rad5None.CheckedChanged, rad6Regular.CheckedChanged,
73     rad7Cappuccino.CheckedChanged
74     lbl1Subtotal.Text = String.Empty
75     lbl2Tax.Text = String.Empty
76     lbl3Total.Text = String.Empty
77 End Sub
78
79 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
80     Me.Close()
81 End Sub
82
83 Private Sub frmMain_FormClosing(sender As Object, e As FormClosingEventArgs) Handles Me.FormClosing
84     'verify kill app:
85     Dim dlgButton As DialogResult
86     dlgButton = MessageBox.Show("Sure to exit?", "Doughnut & Coffie", MessageBoxButtons.YesNo, MessageBoxIcon.Exclamation)
87     'if No, dont close:
88     If dlgButton = DialogResult.No Then
89         e.Cancel = True
90     End If
91     End Sub
92 End Class

```



```
- Private Function...Return, ElseIf, Private Sub, ByRef, rad.Checked, chk.Checked,
Private Sub Clear, rad.CheckedChanged, chk.CheckedChanged, frmMain_FormClosing,
dlgButton As DialogResult, MessageBox.Show, DialogResult.No, e.Cancel
```

Mats-R-Us sells three different types of mats: Standard (\$99), Deluxe (\$129), and Premium (\$179). All of the mats are available in blue, red (\$10 extra), and pink (\$15 extra). There is also an extra \$25 charge if the customer wants the mat to be foldable. Create a Windows Forms application. Use the following names for the project and solution, respectively: Mats Project and Mats Solution. Save the application in the VB2017\Chap06 folder. Create the interface shown in Figure 6-58. Use a function to determine the price of the mat before any additional charges. Use a Sub procedure to calculate the total additional charge (if any). Use an event-handling Sub procedure to clear the price. Also, verify that the user wants to exit the application before closing the form. Save the solution and then start and test the application appropriately.

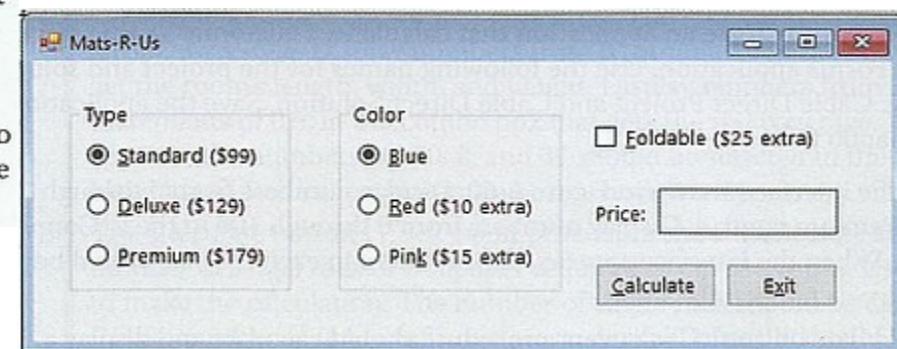


Figure 6-58 Interface for Exercise 8

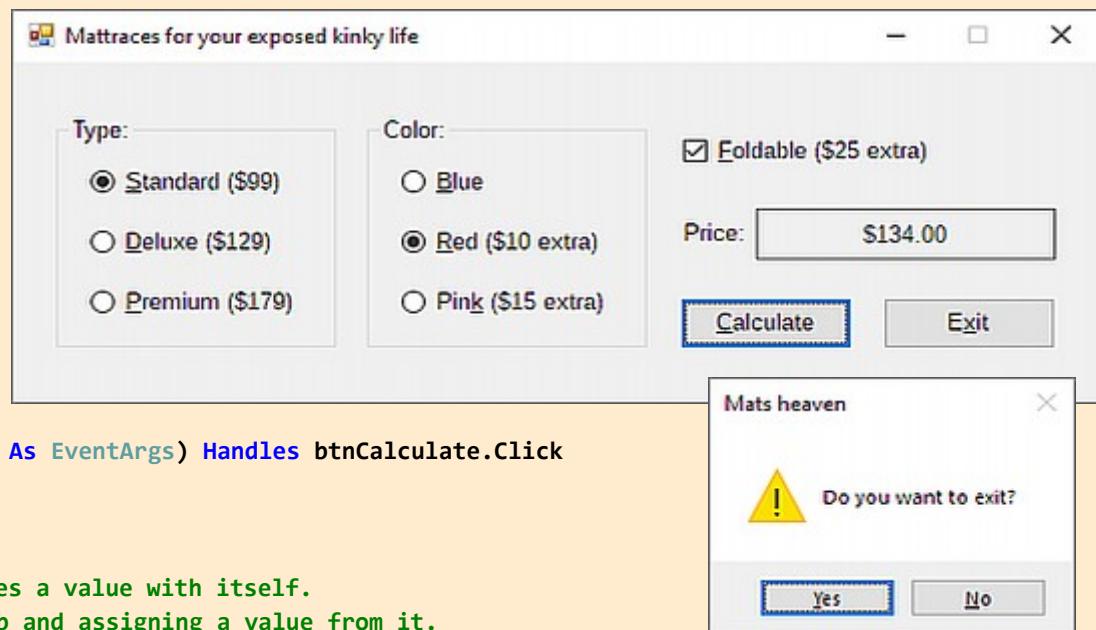
```

1 Option Explicit On
2 Option Strict On
3 Option Infer Off
4 'Function "MatsType", mats type: rad1Standard = 99, rad2Deluxe = 129, rad3Premium = 179.
5 'Sub "AdditionalCharges", ttl additional charges: color: rad4Blue = 0, rad5Red = 10, rad6Pink = 15, + chkFoldable = 25.
6 'event-handling Sub: clear the price.
7 'verify that the user wants to exit.
8
9 Public Class frmMain
10    Private Function MatsType() As Integer
11        Dim intMatsType As Integer
12
13        If rad1Standard.Checked Then
14            intMatsType = 99
15        ElseIf rad2Deluxe.Checked Then
16            intMatsType = 129
17        ElseIf rad3Premium.Checked Then
18            intMatsType = 179
19        End If
20
21        Return intMatsType
22    End Function
23
```

```

24     Private Sub AdditionalCharges(ByRef intAdditional2 As Integer)
25
26         If rad4Blue.Checked Then
27             intAdditional2 += 0
28         ElseIf rad5Red.Checked Then
29             intAdditional2 += 10
30         ElseIf rad6Pink.Checked Then
31             intAdditional2 += 15
32         End If
33         If chkFoldable.Checked Then
34             intAdditional2 += 25
35         Else
36             intAdditional2 += 0
37         End If
38     End Sub
39
40     Private Sub btnCalculate_Click(sender As Object, e As EventArgs) Handles btnCalculate.Click
41         Dim intPrice As Integer
42         Dim intAdditional As Integer
43
44         'MatsType() <- no need, because Function carries a value with itself.
45         AdditionalCharges(intAdditional) 'calling a Sub and assigning a value from it.
46         intPrice = MatsType() + intAdditional
47         lblPrice.Text = intPrice.ToString("C2")
48     End Sub
49
50     Private Sub ClearLabel_CheckedChanged(sender As Object, e As EventArgs) Handles rad1Standard.CheckedChanged,
51                     rad2Deluxe.CheckedChanged, rad3Premium.CheckedChanged, rad4Blue.CheckedChanged, rad5Red.CheckedChanged,
52                     rad6Pink.CheckedChanged, chkFoldable.CheckedChanged
53         lblPrice.Text = String.Empty
54     End Sub
55
56     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
57         Me.Close()
58     End Sub
59
60     Private Sub frmMain_FormClosing(sender As Object, e As FormClosingEventArgs) Handles Me.FormClosing
61         Dim dlgButton As DialogResult
62         dlgButton = MessageBox.Show("Do you want to exit?", "Mats heaven", MessageBoxButtons.YesNo, MessageBoxIcon.Exclamation)
63         If dlgButton = DialogResult.No Then
64             e.Cancel = True
65         End If
66     End Sub
67 End Class

```



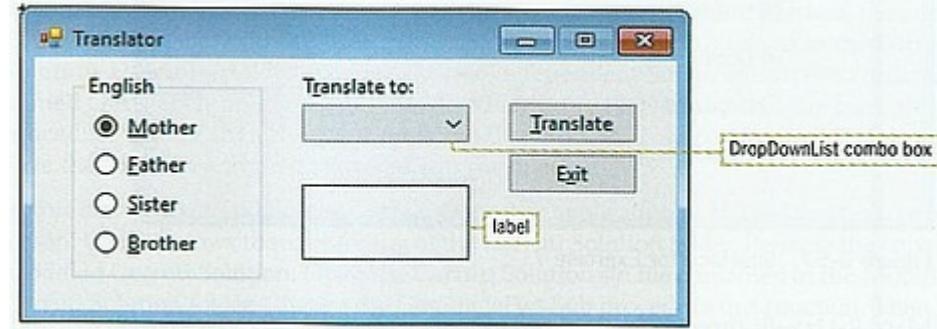
```

- frmMain_Load, cbo.Items.Add, cbo.SelectedIndex, Private Function...Return, ElseIf,
rad.Checked, frmMain_FormClosing, dlgButton As DialogResult, MessageBox.Show,
DialogResult.No, e.Cancel, Private Sub Clear, rad.CheckedChanged

```

Create a Windows Forms application. Use the following names for the project and solution, respectively: Translator Project and Translator Solution. Save the application in the VB2017\Chap06 folder.

- Create the interface shown in Figure 6-59. The combo box should display the following words: French, Italian, and Spanish. When the interface appears, the first item in the combo box should be selected.
- The user will select a radio button in the English group and also select a language from the combo box. The Translate button should use three functions (one for each language) to translate the English word into the desired language. If necessary, use the Internet to find the corresponding French, Italian, and Spanish words. The Translate button should then display the translated word in the label. Code the functions and the button's Click event procedure. (If the Code Editor indicates that a String variable is being passed before it has been assigned a value, assign the String.Empty value to the variable in its Dim statement.)
- Use an event-handling Sub procedure to clear the label. Also, verify that the user wants to exit the application before closing the form.
- Save the solution and then start and test the application.



```

1 Option Explicit On
2 Option Strict On
3 Option Infer Off
4 'the user: select (rad) and select (cbo)
5 'cboTranslateTo = Czech, Dutch, Thai (phonetic) -> the 1st item should be selected
6 'btnTranslate -> should use 3x Function for each language selected
7 '
8     -> and display in lblShow
9 '---- if the Code Editor indicates that a "String variable is being passed before it has been assigned a value",
10 '---- assign the "String.Empty" value to the variable in its Dim Statement.
11 'use event-handling Sub procedure to clear the label
12 'verify that the user wants to exit
13
14 Public Class frmMain
15     Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
16         cboTranslateTo.Items.Add("Czech")
17         cboTranslateTo.Items.Add("Dutch")
18         cboTranslateTo.Items.Add("Thai (phonetic)")
19         cboTranslateTo.SelectedIndex = 0
20     End Sub

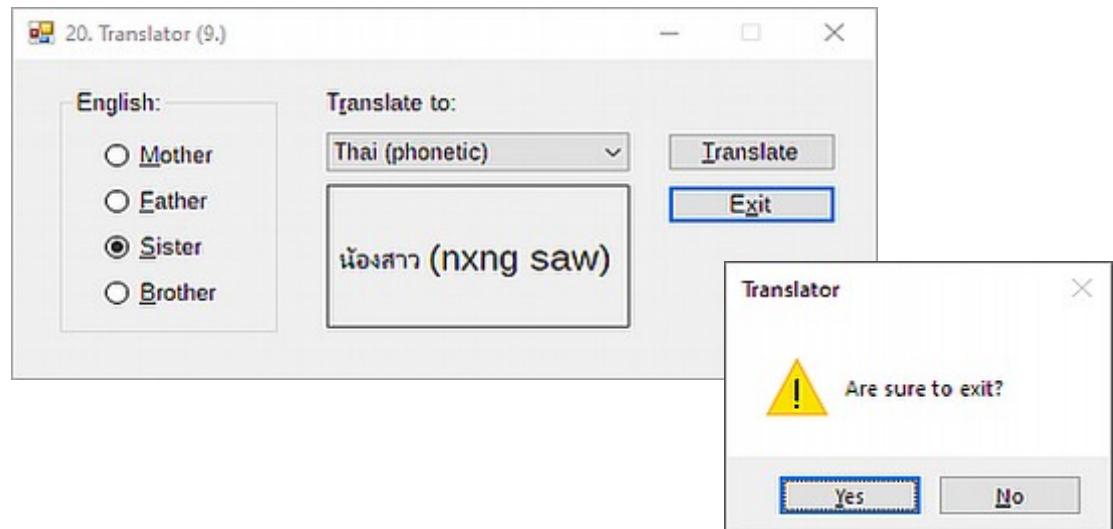
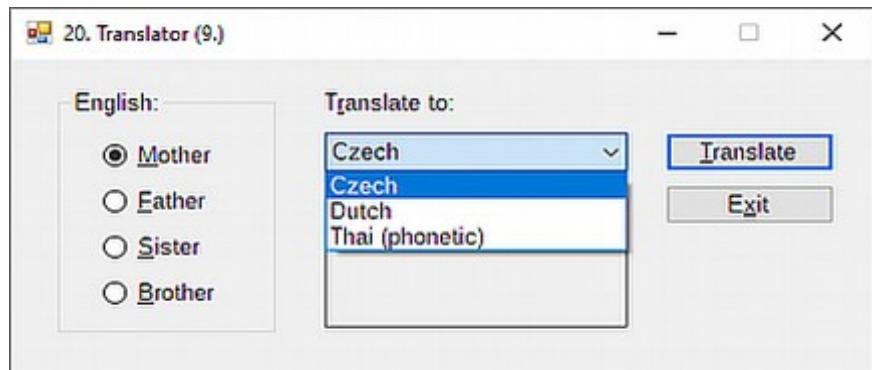
```

```
20
21     Private Function Czech() As String
22         Dim strTranslate As String = String.Empty
23         If rad1Mother.Checked Then
24             strTranslate = "Mamka"
25         ElseIf rad2Father.Checked Then
26             strTranslate = "Otec"
27         ElseIf rad3Sister.Checked Then
28             strTranslate = "Sestra"
29         ElseIf rad4Brother.Checked Then
30             strTranslate = "Bratr"
31         End If
32         Return strTranslate
33     End Function
34
35     Private Function Dutch() As String
36         Dim strTranslate As String = String.Empty
37         If rad1Mother.Checked Then
38             strTranslate = "Moeder"
39         ElseIf rad2Father.Checked Then
40             strTranslate = "Vader"
41         ElseIf rad3Sister.Checked Then
42             strTranslate = "Zus"
43         ElseIf rad4Brother.Checked Then
44             strTranslate = "Broer"
45         End If
46         Return strTranslate
47     End Function
48
49     Private Function Thai() As String
50         Dim strTranslate As String = String.Empty
51         If rad1Mother.Checked Then
52             strTranslate = "แม่ (mae)"
53         ElseIf rad2Father.Checked Then
54             strTranslate = "พ่อ (phx)"
55         ElseIf rad3Sister.Checked Then
56             strTranslate = "น้องสาว (nxng saw)"
57         ElseIf rad4Brother.Checked Then
58             strTranslate = "พี่ชาย (phi chay)"
59         End If
60         Return strTranslate
61     End Function
62
```

```

63      Private Sub btnTranslate_Click(sender As Object, e As EventArgs) Handles btnTranslate.Click
64          If cboTranslateTo.SelectedIndex = 0 Then
65              lblShow.Text = Czech()
66          ElseIf cboTranslateTo.SelectedIndex = 1 Then
67              lblShow.Text = Dutch()
68          ElseIf cboTranslateTo.SelectedIndex = 2 Then
69              lblShow.Text = Thai()
70          End If
71      End Sub
72
73      Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
74          Me.Close()
75      End Sub
76
77      Private Sub frmMain_FormClosing(sender As Object, e As FormClosingEventArgs) Handles Me.FormClosing
78          Dim dlgButton As DialogResult
79          dlgButton = MessageBox.Show("Are sure to exit?", "Translator", MessageBoxButtons.YesNo, MessageBoxIcon.Exclamation)
80          If dlgButton = DialogResult.No Then
81              e.Cancel = True
82          End If
83      End Sub
84
85      Private Sub ClearLabel(sender As Object, e As EventArgs) Handles rad1Mother.CheckedChanged, rad2Father.CheckedChanged,
86          rad3Sister.CheckedChanged, rad4Brother.CheckedChanged, cboTranslateTo.SelectedIndexChanged
87          lblShow.Text = String.Empty
88      End Sub
89  End Class

```



## Chap06\\_Exercise1

### 21.Translator Solution-Sub\_EXERCISE 10\_intermediate

- frmMain\_Load, cbo.Items.Add, cbo.SelectedIndex, **Private Sub**, **ByRef**, rad.Checked, **ElseIf**, frmMain\_FormClosing, dlgButton As DialogResult, MessageBox.Show, DialogResult.No, e.Cancel, **Private Sub** Clear, rad.CheckedChanged

In this exercise, you modify the application created in Exercise 9. Use Windows to make a copy of the Translator Solution folder. Rename the copy Translator Solution-Sub. Open the Translator Solution.sln file contained in the Translator Solution-Sub folder. Change the three functions to Sub procedures. (If the Code Editor indicates that a String variable is being passed before it has been assigned a value, assign the String.Empty value to the variable in its Dim statement.) Save the solution and then start and test the application appropriately.

```
1  Option Explicit On
2  Option Strict On
3  Option Infer Off
4  'MODIFICATION FROM EXERCISE 9: CHANGE FUNCTION FOR SUB
5  'the user: select (rad) and select (cbo)
6  'cboTranslateTo = Czech, Dutch, Thai (phonetic) -> the 1st item should be selected
7  'btnTranslate -> should use 3x Sub for each language selected
8  '           -> and display in lblShow
9  '---- if the Code Editor indicates that a "String variable is being passed before it has been assigned a value",
10 '---- assign the "String.Empty" value to the variable in its Dim Statement.
11 'use event-handling Sub procedure to clear the label
12 'verify that the user wants to exit
13
14 Public Class frmMain
15     Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
16         cboTranslateTo.Items.Add("Czech")
17         cboTranslateTo.Items.Add("Dutch")
18         cboTranslateTo.Items.Add("Thai (phonetic)")
19         cboTranslateTo.SelectedIndex = 0
20     End Sub
21
22     Private Sub Czech(ByRef strTranslate2 As String)
23         If rad1Mother.Checked Then
24             strTranslate2 = "Mamka"
25         ElseIf rad2Father.Checked Then
26             strTranslate2 = "Otec"
27         ElseIf rad3Sister.Checked Then
28             strTranslate2 = "Sestra"
29         ElseIf rad4Brother.Checked Then
30             strTranslate2 = "Bratr"
31         End If
32     End Sub
```

```
33
34     Private Sub Dutch(ByRef strTranslate2 As String)
35         If rad1Mother.Checked Then
36             strTranslate2 = "Moeder"
37         ElseIf rad2Father.Checked Then
38             strTranslate2 = "Vader"
39         ElseIf rad3Sister.Checked Then
40             strTranslate2 = "Zus"
41         ElseIf rad4Brother.Checked Then
42             strTranslate2 = "Broer"
43         End If
44     End Sub
45
46     Private Sub Thai(ByRef strTranslate2 As String)
47         If rad1Mother.Checked Then
48             strTranslate2 = "แม่ (mae)"
49         ElseIf rad2Father.Checked Then
50             strTranslate2 = "พ่อ (phx)"
51         ElseIf rad3Sister.Checked Then
52             strTranslate2 = "น้องสาว (nxng saw)"
53         ElseIf rad4Brother.Checked Then
54             strTranslate2 = "พี่ชาย (phi chay)"
55         End If
56     End Sub
57
58     Private Sub btnTranslate_Click(sender As Object, e As EventArgs) Handles btnTranslate.Click
59         Dim strTranslate As String = String.Empty
60         If cboTranslateTo.SelectedIndex = 0 Then
61             Czech(strTranslate)
62         ElseIf cboTranslateTo.SelectedIndex = 1 Then
63             Dutch(strTranslate)
64         ElseIf cboTranslateTo.SelectedIndex = 2 Then
65             Thai(strTranslate)
66         End If
67         lblShow.Text = strTranslate
68     End Sub
69
70     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
71         Me.Close()
72     End Sub
73
```

```

74  Private Sub frmMain_FormClosing(sender As Object, e As FormClosingEventArgs) Handles Me.FormClosing
75      Dim dlgButton As DialogResult
76      dlgButton = MessageBox.Show("Are sure to exit?", "Translator", MessageBoxButtons.YesNo, MessageBoxIcon.Exclamation)
77      If dlgButton = DialogResult.No Then
78          e.Cancel = True
79      End If
80  End Sub
81
82  Private Sub ClearLabel(sender As Object, e As EventArgs) Handles rad1Mother.CheckedChanged, rad2Father.CheckedChanged,
83      rad3Sister.CheckedChanged, rad4Brother.CheckedChanged, cboTranslateTo.SelectedIndexChanged
84      lblShow.Text = String.Empty
85  End Sub
86 End Class

```

## Chap06\Exercise1

### 22.Cable Direct Solution\_EXERCISE 11\_advanced

- frmMain\_Load, For...To...Next, lst.Items.Add, lst.SelectedIndex, Private Function, dlgButton As DialogResult, MessageBox.Show, ElseIf, AndAlso, Private Sub Clear, rad.CheckedChanged, lst.SelectedIndexChanged, frmMain\_FormClosing, e.Cancel

In this exercise, you create an application that calculates a customer's cable bill. Create a Windows Forms application. Use the following names for the project and solution, respectively: Cable Direct Project and Cable Direct Solution. Save the application in the VB2017\Chap06 folder.

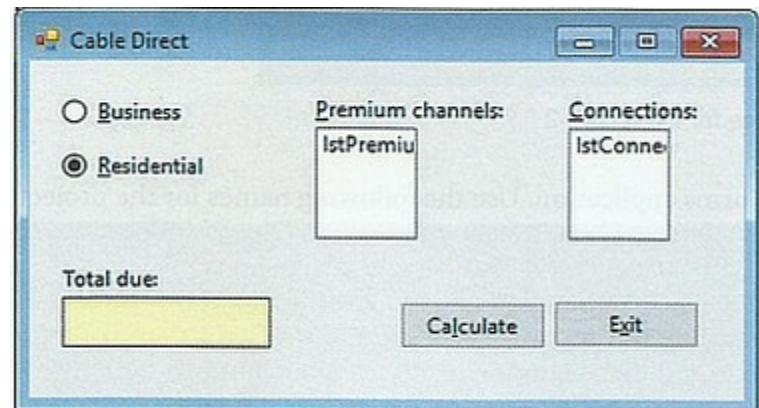
- Create the interface shown in Figure 6-60. Display numbers from 0 through 20 in the lstPremium control. Display numbers from 0 through 100 in the lstConnections control. When the interface appears, the first item in each list box should be selected.
- The Calculate button's Click event procedure should calculate and display a customer's cable bill. The cable rates are included in Figure 6-60. Business customers must have at least one connection. Use two functions: one to calculate and return the total due for business customers, and one to calculate and return the total due for residential customers.
- The form's FormClosing event procedure should verify that the user wants to close the application.
- The total due should be cleared when a change is made to a radio button or list box.
- Save the solution and then start and test the application. (The total due for a business customer with 3 premium channels and 12 connections is \$254.50. The total due for a residential customer with 3 premium channels is \$49.50.)

#### Residential customers:

Processing fee: \$4.50  
 Basic service fee: \$30  
 Premium channels: \$5 per channel

#### Business customers:

Processing fee: \$16.50  
 Basic service fee: \$80 for the first 10 connections; \$4 for each additional connection  
 Premium channels: \$50 per channel for any number of connections



```

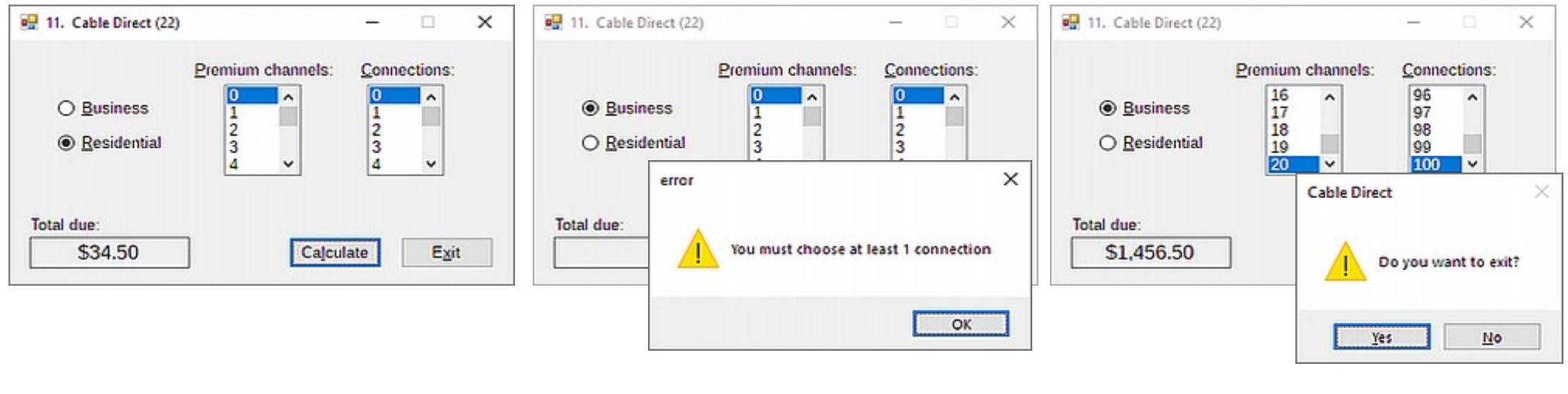
1 Option Explicit On
2 Option Strict On
3 Option Infer Off
4 'lst1Premium = 0 -> 20, SelectedIndex = 0; lst2Connections = 0 -> 100, SelectedIndex = 0
5 'btnCalculate.Click = calculate and display customer's cable bill, using 2x Function:
6 'rad2Residential = Function "fuResidential":
7 '    fuResidential - calculate & return total due
8 '        = 1. Processing fee = $4.50
9 '        = 2. Basic service fee = $30
10 '           = 3. Premium channels: $5 per channel.
11 'rad1Business = Function "fuBusiness":
12 '    fuBusiness - calculate & return total due, AT LEAST 1 CONNECTION !!!
13 '        = 1. Processing fee = $16.50,
14 '        = 2. Basic service fee: $80 for the first 10 connections, $4 for each additional connection
15 '        = 3. Premium channels: $50 per channel for any number of connections.
16 'FormClosing event should verify that the user wants to kill the application
17 'lbl should be cleared when a change is made to a (rad) or (lst)
18 'TEST_1: Business customer, 3 Premium channels, 12 connections = $254.50.
19 'TEST_2: Residential customer, 3 Premium channels = $49.50.
20
21 Public Class frmMain
22     Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
23         For int2Premium As Integer = 0 To 20
24             lst1Premium.Items.Add(int2Premium)
25             lst1Premium.SelectedIndex = 0
26         Next int2Premium
27
28         For int2Connections As Integer = 0 To 100
29             lst2Connections.Items.Add(int2Connections)
30             lst2Connections.SelectedIndex = 0
31         Next int2Connections
32     End Sub
33
34     Private Function fuBusiness() As Double
35         Dim dblTotalDue2 As Double
36         Dim intPremium As Integer
37         Dim intConnections As Integer
38         Integer.TryParse(lst1Premium.SelectedItem.ToString, intPremium)
39         Integer.TryParse(lst2Connections.SelectedItem.ToString, intConnections)
40
41         If intConnections = 0 Then
42             Dim dlgButton As DialogResult
43             dlgButton = MessageBox.Show("You must choose at least 1 connection", "error",
44   MessageBoxButtons.OK, MessageBoxIcon.Exclamation)

```

```

45      ElseIf intConnections > 0 AndAlso intConnections <= 10 Then
46          dblTotalDue2 = 16.5 + 80 + (intPremium * 50)
47      ElseIf intConnections > 10 Then
48          dblTotalDue2 = 16.5 + 80 + ((intConnections - 10) * 4) + (intPremium * 50)
49      End If
50      Return dblTotalDue2
51  End Function
52
53  Private Function fuResidential() As Double
54      Dim dblTotalDue2 As Double
55      Dim intPremium As Integer
56      Integer.TryParse(lst1Premium.SelectedItem.ToString, intPremium)
57      dblTotalDue2 = 4.5 + 30 + (intPremium * 5)
58      Return dblTotalDue2
59  End Function
60
61  Private Sub btnCalculate_Click(sender As Object, e As EventArgs) Handles btnCalculate.Click
62      If rad1Business.Checked Then
63          lblTotalDue.Text = fuBusiness().ToString("C2")
64      Else
65          lblTotalDue.Text = fuResidential().ToString("C2")
66      End If
67  End Sub
68
69  Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
70      Me.Close()
71  End Sub
72
73  Private Sub ClearLabel(sender As Object, e As EventArgs) Handles rad1Business.CheckedChanged, rad2Residential.CheckedChanged,
74   lst1Premium.SelectedIndexChanged, lst2Connections.SelectedIndexChanged
75      lblTotalDue.Text = String.Empty
76  End Sub
77
78  Private Sub frmMain_FormClosing(sender As Object, e As FormClosingEventArgs) Handles Me.FormClosing
79      Dim dlgButton As DialogResult
80      dlgButton = MessageBox.Show("Do you want to exit?", "Cable Direct", MessageBoxButtons.YesNo, MessageBoxIcon.Exclamation)
81      If dlgButton = DialogResult.No Then
82          e.Cancel = True
83      End If
84  End Sub
85 End Class

```



## Chap06\Exercises

### 23. Wallpaper Solution\_EXERCISE 12\_advanced

- frmMain\_Load, For...To...Next, cbo.Items.Add, cbo.SelectedItem, Private Sub, ByVal, ByRef, Math.Round, Private Sub Clear, cbo.TextChanged

In this exercise, you create an application that calculates the number of single rolls of wallpaper required to cover a room. Create a Windows Forms application. Use the following names for the project and solution, respectively: Wallpaper Project and Wallpaper Solution. Save the application in the VB2017\Chap06 folder.

- Create the interface shown in Figure 6-61. The four combo boxes have the DropDownList style. Display numbers from 8 through 30 in the combo boxes that get the room's length, width, and height. Display numbers from 30 through 40, in increments of 0.5, in the combo box that gets the roll coverage. When the interface appears, the numbers 10, 10, 8, and 37 should be selected in the Length, Width, Height, and Roll coverage combo boxes, respectively.
- The Calculate button's Click event procedure should calculate and display the number of single rolls of wallpaper required to cover a room. Use a Sub procedure to make the calculation. The number of single rolls should be displayed as an integer. (If the number of single rolls contains a decimal place, it should be rounded to the next highest integer. For example, 8.2 rolls should be rounded up to 9.)

- The number of rolls should be cleared when a change is made to a combo box.
- Save the solution and then start and test the application. (If the roll coverage is 38.5 square feet and the room's length, width, and height are 14, 20, and 8, respectively, the number of single rolls is 15.)

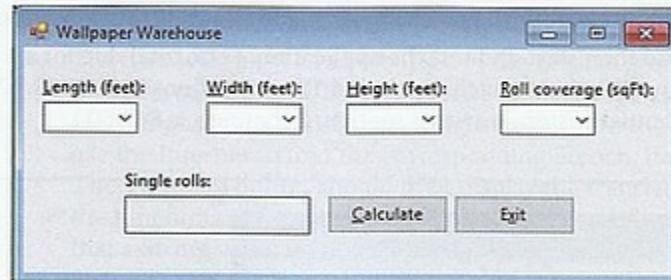


Figure 6-61 Interface for Exercise 12

```

1 Option Explicit On
2 Option Strict On
3 Option Infer Off
4 'I'M FOLLOWING THE ASSIGNED TASK, nothing EXTRA like -only numbers in (cbo), confirm to kill the app, etc...
5 '4x cbo: cbo1Length, cbo2Width, cbo3Height, cbo4RollCoverage
6 ' _____DO NOT DO, BECAUSE THAT WAS NOT THE TASK -> because DropDown, i will limit user entry to: decimal numbers.
7 '     - 1 till 3 to be pre-range: 8 through 30
8 '     - 4 to be pre-range 30 through 40, in increments of 0.5
9 '     ! when GUI appears, it should be pre-selected: 10, 10, 8, 37
10 'btn: - should calculate and display the number of single rolls of wallpapers required to cover a room.
11 '     -> use a Sub procedure to make the calculation
12 '     ! the number of single rolls should be displayed as an INTEGER = do round to the next highest integer (eg. 8.2 round to 9)
13 'lbl: -> should be cleared when a change is made to the cbos
14 'test: if the length = 14, width = 20, height = 8, roll coverage = 38.5 the result should be 15 single rolls.
15 'THAT'S ALL FROM THE TASK, better i won't do more then the task assignment, i feel
16
17 Public Class frmMain
18     Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
19         For IntLoad123 As Integer = 8 To 30
20             cbo1Length.Items.Add(IntLoad123)
21             cbo2Width.Items.Add(IntLoad123)
22             cbo3Height.Items.Add(IntLoad123)
23         Next IntLoad123
24         cbo1Length.SelectedText = "10"
25         cbo2Width.SelectedText = "10"
26         cbo3Height.SelectedText = "8"
27
28         For dblLoad4 As Double = 30 To 40 Step 0.5
29             cbo4RollCoverage.Items.Add(dblLoad4)
30         Next dblLoad4
31         cbo4RollCoverage.SelectedText = "37"
32     End Sub
33
34     Private Sub subCalculation(ByVal dbl1Length2 As Double, ByVal dbl2Width2 As Double, ByVal dbl3Height2 As Double,
35                               ByVal dbl4RollCoverage2 As Double, ByRef dblCalculation2 As Double)
36         '(a*c*2) + (b*c*2) / d
37         dblCalculation2 = ((dbl1Length2 * dbl3Height2 * 2) + (dbl2Width2 * dbl3Height2 * 2)) / dbl4RollCoverage2
38
39     End Sub
40
41     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
42         Me.Close()
43     End Sub

```

```

44
45  Private Sub btnCalculate_Click(sender As Object, e As EventArgs) Handles btnCalculate.Click
46      'send 3x, receive 1x the result
47      Dim dbl1Length As Double
48      Dim dbl2Width As Double
49      Dim dbl3Height As Double
50      Dim dbl4RollCoverage As Double
51      Dim dblCalculation As Double
52      Double.TryParse(cbo1Length.Text, dbl1Length)
53      Double.TryParse(cbo2Width.Text, dbl2Width)
54      Double.TryParse(cbo3Height.Text, dbl3Height)
55      Double.TryParse(cbo4RollCoverage.Text, dbl4RollCoverage)

56
57      subCalculation(dbl1Length, dbl2Width, dbl3Height, dbl4RollCoverage, dblCalculation)
58
59      dblCalculation = Math.Round(dblCalculation + 0.4)
60      lblShow.Text = dblCalculation.ToString
61  End Sub
62
63  Private Sub ClearResult(sender As Object, e As EventArgs) Handles cbo1Length.TextChanged, cbo2Width.TextChanged,
64      cbo3Height.TextChanged, cbo4RollCoverage.TextChanged
65      lblShow.Text = String.Empty
66  End Sub
67 End Class

```

Length (feet): 14    Width (feet): 20    Height (feet): 8    Roll coverage (sqFt): 38.5

Single rolls needed: 15

Length (feet): 10    Width (feet): 10    Height (feet): 8    Roll coverage (sqFt): 37

Single rolls needed: 9

\_notes.txt

```

1  a = length = 14 feet    10
2  b = width = 20 feet     10
3  c = height = 8 feet     8
4  d = roll coverage = 38.5 square feet   37
5  result: the number of single roll is: 15
6
7
8  38.5 * 15 = 577.5
9  (a*c * 2) + (b*c * 2) = 224 + 320 = 544
10 544 / (d)38.5 = 14.129, rounded to next highest integer = 15 (i tried + 0.4, because Math.Round rounds 14.129 to 14!!!)
11
12  ((a*c*2) + (b*c*2)) / d
13
14  320 / 37 = 8.64, rounded = 9

```

## ON YOUR OWN

13. Create a Windows Forms application. Use the following names for the project and solution, respectively: OnYourOwn Project and OnYourOwn Solution. Save the application in the VB2017\Chap06 folder. Plan and design an application of your choice. The only requirement is that you must follow the minimum guidelines listed in Figure 6-62. Before starting the application, be sure to verify the name of the startup form. Save the solution and then start and test the application.

1. The user interface must contain a minimum of one text box, three labels, one combo box, and two buttons. One of the buttons must be an Exit button.
2. The interface can include a picture box, but this is not a requirement.
3. The interface must follow the GUI design guidelines summarized for Chapters 2 through 6 in Appendix A.
4. Objects that are either coded or referred to in code should be named appropriately.
5. The Code Editor window must contain comments, the three Option statements, at least two variables, at least two assignment statements, at least one independent Sub procedure or at least one function, at least one event-handling Sub procedure, and the Me.Close() statement. The application must perform at least one calculation.
6. Every text box on the form should have its TextChanged and Enter event procedures coded. At least one of the text boxes should have its KeyPress event procedure coded.
7. The combo box should have its TextChanged event procedure coded.
8. The form's FormClosing procedure should verify that the user wants to end the application.

Figure 6-62 Guidelines for Exercise 13

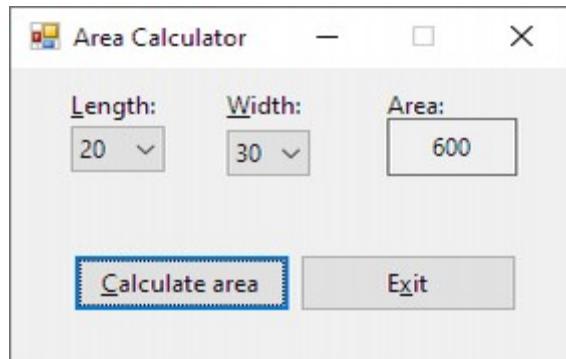
- Private Function...Return, ByVal, frmMain\_Load, For...To...Step...Next, cbo.Items.Add,  
cbo.SelectedIndex, Private Sub Clear, cbo.TextChanged

## FIX IT

14. Open the VB2017\Chap06\FixIt Solution\FixIt Solution.sln file. Start the application. Click 20 in the Length combo box and then click 30 in the Width combo box. Click the Calculate area button, which should display the area of a rectangle having a length of 20 feet and a width of 30 feet. Notice that the application does not display the correct area. Stop the application. Correct the application's code.

```
1  ' Name:      Debug Project
2  ' Purpose:    Calculates and displays the area of a rectangle.
3  ' Programmer: <your name> on <current date>
4
5  Option Explicit On
6  Option Strict On
7  Option Infer Off
8
9  Public Class frmMain
10     Private Function GetArea(ByVal intLen As Integer, ByVal intWid As Integer) As Integer
11         Dim intArea As Integer
12         intArea = intLen * intWid
13         ' Return statement has been missing:
14         Return intArea
15     End Function
16
17     Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
18         ' Uses a function to calculate the area and then displays the area.
19         Dim intLength As Integer
20         Dim intWidth As Integer
21         Dim intArea As Integer
22
23         Integer.TryParse(cboLength.Text, intLength)
24         Integer.TryParse(cboWidth.Text, intWidth)
25
26         intArea = GetArea(intLength, intWidth)
27         lblArea.Text = intArea.ToString
28     End Sub
29
30     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
31         Me.Close()
32     End Sub
```

```
33
34     Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
35         ' Fills the combo boxes with values and selects first item.
36
37         For intNum As Integer = 10 To 40 Step 5
38             cboLength.Items.Add(intNum)
39             cboWidth.Items.Add(intNum)
40         Next intNum
41
42         cboLength.SelectedIndex = 0
43         cboWidth.SelectedIndex = 0
44     End Sub
45
46     Private Sub ClearArea(sender As Object, e As EventArgs) Handles cboLength.TextChanged, cboWidth.TextChanged
47         lblArea.Text = String.Empty
48     End Sub
49 End Class
```



## Mortgage Calculator (Chapters 1–6)

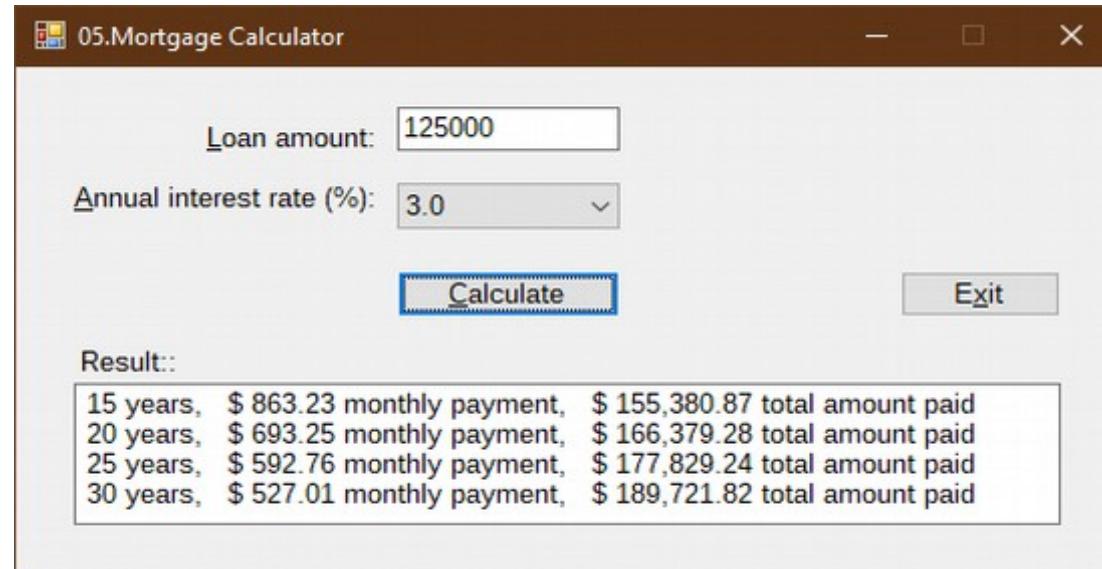
Create an application that calculates and displays four monthly mortgage payments. The application should use the loan amount and annual interest rate provided by the user, along with terms of 15 years, 20 years, 25 years, and 30 years. Use a combo box to get the annual interest rate, which should range from 2% through 8% in increments of 0.5%. The application should also display the total amount paid at the end of 15 years, 20 years, 25 years, and 30 years.

```
1  ' Name:      Mortgage Calculator.
2  ' Purpose:    Calculate and display 4 mortgage payments based on user's loan and annual interest.
3  ' Programmer: Me on just now.
4  Option Explicit On
5  Option Strict On
6  Option Infer Off
7
8  Public Class frmMain
9      Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
10         For dblRate As Double = 2.0 To 8.0 Step 0.5
11             cboRate.Items.Add(dblRate.ToString("N1"))
12         Next dblRate
13         cboRate.SelectedIndex = 6
14         'txtDisplay.Text = "Years:" & ControlChars.Tab & "Monthly payment:" & ControlChars.Tab & "Total payment:"
15     End Sub
16
17     Private Sub btnCalculate_Click(sender As Object, e As EventArgs) Handles btnCalculate.Click
18         ' calculate and display 4 mortgage payments: terms of 15, 20, 25, 30 years.
19         ' also display total amount paid at the end of 15, 20, 25, 30 years.
20         ' annual interest: 2% -> 8% in increments of 0.5%. (cboRate.SelectedIndex = 0 To 12
21
22         ' Financial.Pmt(Rate, Nper, PV, FV, Due) As Double
23         ' Rate = interest rate per period
24         ' Nper = total number of payment periods in the annuity
25         ' PV = present value of a series of future payments; PV = loan amount
26         ' FV = optional, if you borrow money, default FV = 0
27         ' Due = optional, when payment are due, default = DueDate.EndOfPeriod
28
```

```

29      ' txtLoan = user input -> PV
30      ' cboRate = annual interest rate 2 - 8 Step 0.5 (Index 0 - 12)
31      ' txtDisplay = multiline output
32
33      Dim dblLoan As Double : Double.TryParse(txtLoan.Text, dblLoan)
34      Dim dblRate As Double : Double.TryParse(cboRate.SelectedItem.ToString, dblRate)
35      Dim dblMonthly As Double
36      Dim dblTotal As Double
37
38      For intYears As Integer = 15 To 30 Step 5
39          dblMonthly = -Financial.Pmt((dblRate / 100) / 12, intYears * 12, dblLoan)
40          dblTotal = dblMonthly * (intYears * 12)
41
42          txtDisplay.Text &= intYears.ToString & " years, " & dblMonthly.ToString("C2") & " monthly payment, " &
43                          dblTotal.ToString("C2") & " total amount paid" & ControlChars.NewLine
44      Next intYears
45  End Sub
46
47  Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
48      Me.Close()
49  End Sub
50
51  Private Sub txtLoan_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtLoan.KeyPress
52      ' allow only numbers & decimal point & Backspace
53      If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> "." AndAlso e.KeyChar <> ControlChars.Back Then
54          e.Handled = True
55      End If
56  End Sub
57
58  Private Sub InputChange(sender As Object, e As EventArgs) Handles txtLoan.TextChanged, cboRate.SelectedIndexChanged
59      ' clear output when any input changes
60      txtDisplay.Text = Nothing
61  End Sub
62
63  Private Sub txtLoan_MouseClick(sender As Object, e As MouseEventArgs) Handles txtLoan.MouseClick
64      txtLoan.SelectAll()
65  End Sub
66
67  Private Sub txtLoan_Enter(sender As Object, e As EventArgs) Handles txtLoan.Enter
68      txtLoan.SelectAll()
69  End Sub
70 End Class

```



FINITO, THE END, KONEC, BASTA AMIGO

**CH7\_FOCUS ON THE CONCEPTS LESSON**

- CH7\_F1** - Length property of the String: `string.Length`
- CH7\_F1.1** - Length property of the String: `string.Length` example: The Product ID Application:
- CH7\_F2** - Insert method of the String: `string.Insert(startIndex, value)`
- CH7\_F3** - aligning characters in string & columns - methods: `PadLeft` & `PadRight` & function: `String.Space` - additional topic from **Appendix B**
- CH7\_F3.1** - Insert method & `PadLeft` method example: **The Net Pay Application** (02.Net Pay Solution)
- CH7\_F3.2** - **You Do It 1:** Trim method, `PadRight` method, Insert method exercise: 03.You Do It 1 Solution
- CH7\_F4** - Search for a specific sequence of characters: `Contains` and `IndexOf` methods of the String:
  - CH7\_F4.1** - String search: `Contains` method
  - CH7\_F4.2** - String search: `IndexOf` method
  - CH7\_F4.3** - String search: `IndexOf` method example: **The City and State Application** (04.City State Solution)
  - CH7\_F4.4** - **You Do It 2:** `Contains` method and `IndexOf` method exercise: 05.You Do It 2 Solution
- CH7\_F5** - Substring method of the String
  - CH7\_F5.1** - Substring method of the String example: **The Rearrange Name Application** (06.Rearrange Solution)
  - CH7\_F5.2** - **You Do It 3:** Substring method exercise: 07.You Do It 3 Solution
- CH7\_F6** - Character Array: using `Character's index` to access a `Character` in a String
  - CH7\_F6.1** - using `Character's index` to access a `Character` in a String example: **The First Name Application** (08. Name Solution)
  - CH7\_F6.2** - **You Do It 4:** `Character's index` exercise: 09.You Do It 4 Solution
- CH7\_F7** - Remove method of the String:
  - CH7\_F8** - Trim, `TrimStart` and `TrimEnd` methods of the String:
    - CH7\_F8.1** - `TrimEnd` method example: **The Tax Calculator Application** (10. Tax Solution)
- CH7\_F9** - Replace method of the String:
  - CH7\_F10** - Like Operator to compare Strings:
    - CH7\_F10.1** - Like Operator to compare Strings example: **The Inventory Application** (11. Inventory Solution)
    - CH7\_F10.2** - **You Do It 5:** Like operator to compare strings exercise: 12.You Do It 5 Solution

**CH7\_APPLY THE CONCEPTS LESSON**

- CH7\_A1** - Code the **Check Digit Application** - thoroughly step by step : (13.Check Digit Solution)
- CH7\_A2** - Code the **Password Application** - thoroughly step by step : (14.Password Solution)
- CH7\_A3** - Generate Random numbers - object `Random` & method `Next` for integers, or method `NextDouble` for doubles\_additional topic from **Appendix B**
- CH7\_A3.1** - Generate Random Integers example: code the **Guess a Letter Application** (15.Letter Guess Solution) - part 1/3
- CH7\_A4** - use the `Enabled` property of the Control
- CH7\_A5** - use the `Focus` method of the Control
- CH7\_A6** - Generate Random Integers example: code the **Guess a Letter Application** (15.Letter Guess Solution) - part 2/3
- CH7\_A7** - Generate Random Integers example: code the **Guess a Letter Application** (15.Letter Guess Solution) - entire code - part 3/3
- CH7\_A8** - many of the concepts from CH7 example: code the **Guess the Word Game Application** (16.Word Guess Solution) + entire code

**CH7\_Summary:** Check digits, Random integers, Enable property & Focus method of the Control, Concepts used for String manipulations

**CH7\_Key Terms**

**CH7\_Exercises**

## CH7\_FOCUS ON THE CONCEPTS LESSON

### CH7\_F1 - Length property of the String: `string.Length`

- the number of characters contained in a String is stored as an integer in the string's Length property

e.g.: if an application expects the user to enter a seven-digit phone number or a five-digit ZIP code,  
-> you should verify that the user's input contains the required number of characters:

String property Length syntax:

**As Integer**

**string.Length As Integer**

`string` = can be - a String variable,

- a String named constant,

- the Text property of a control, like (cbo) for example

e.g.1

```
Dim intNumChars As Integer  
Dim strCountry As String = "Mexico"  
intNumChars = strCountry.Length
```

<- string **Mexico** contains 6 characters  
<- assigns the number **6** to the variable: **intNumChars**

e.g.2

```
...  
intChars = txtName.Text.Trim.Length
```

<- assigns the number of characters in the: **txtName.Text** property,  
excluding any leading or trailing space characters (**Trim** property), to the variable **intChars**

### CH7\_F1.1 - Length property of the String: `string.Length` example: The Product ID Application:

- you will use the **Length** property in the application: **Product ID**, which displays a listing of the product IDs entered by the user, where each product ID must contain exactly **five** characters:

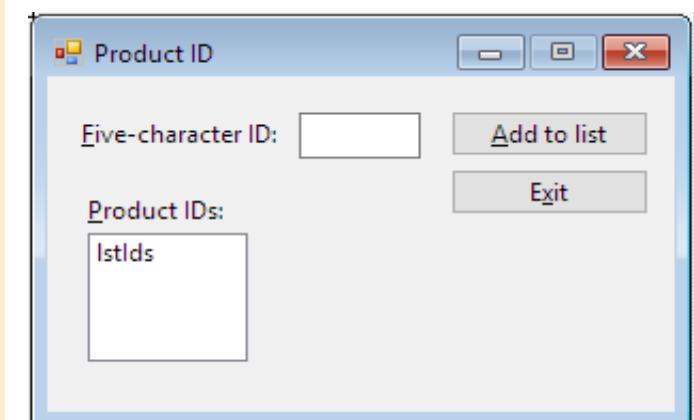
1). open the: ...VB2017\Chap07\\_Exercise\01.Product Solution\Product Solution.sln

2). open the Code Editor window and locate the **btnAdd\_Click** procedure -> you learned about **ToUpper** and **Trim** methods in CH4:

-> modify the **If** clause as shown:

```
18     Private Sub btnAdd_Click(sender As Object, e As EventArgs) Handles btnAdd.Click  
19         ' Add a product ID to a list box.  
20  
21         Dim strID As String  
22  
23         strID = txtId.Text.ToUpper.Trim  
24         If strID.Length = 5 Then  
25             lstIds.Items.Add(strID)  
26         Else  
27             MessageBox.Show("The ID must contain 5 characters.",  
28                             "Product ID", MessageBoxButtons.OK,  
29                             MessageBoxIcon.Information)  
30         End If  
31     End Sub  
32 End Class
```

### CH4\_F6 - String Comparisons: **ToUpper**, **ToLower**, **Trim** methods:



3). save the solution and then start and test the application:

-> type **abcd** as the ID and then click the button **Add to list**

- a message box opens and displays the message: "**The ID must contain 5 characters**", close the message box

-> change the ID to **abcd4** and then click the button **Add to list**

- the button's Click event procedure adds the **ABCD4** ID to the list box (capital letters caused by the method **ToUpper**)

-> test the application using an ID that contains both **leading** and **trailing** spaces (**Trim** method erases them before processing)

4). exit the application and close the solution.

#### CH7\_F2 - Insert method of the String: `string.Insert(startIndex, value)`

- Visual Basic's method **Insert** allows you to insert characters anywhere in a string

- it then returns a string with the appropriate characters inserted

- when processing the method **Insert**, the computer first stores a temporary copy of the **string** in its RAM, then it performs the specified insertion on the copy only ->

-> the **Insert** method does **NOT** affect the original string

String method **Insert** syntax:

**As String**

`string.Insert(startIndex As Integer, value As String) As String`

**string - As String**

= can be - a String variable,  
- a String named constant,  
- the Text property of a control, like (cbo) for example

**startIndex - As Integer**

= argument; an integer that specifies where in the string's copy you want the **value** inserted  
- the integer represents the zero-based starting character position  
- the 1st character in a string has an index of 0, the 2nd character has an index of 1, and so on...

**value - As String**

= specifies what to insert in previously designated position

e.g.1

```
Dim strPhone As String = "111-2222"  
txtPhone.Text = strPhone.Insert(0, "(877) ")
```

<- assigns the string **(877) 111-2222** to the **txtPhone.Text** property

<- the **Insert** method inserts the specified **value** "**(877)** " at the beginning (**startIndex = 0**) of the **strPhone** variable

e.g.2

```
Dim strName As String = "Jess Gonzales"  
strName = strName.Insert(5, "M. ")
```

<- assigns the string **Jess M. Gonzales** to the **strName** variable

<- the **Insert** method inserts the specified **value** "**M.** " at the 6th place (**startIndex = 5**) of the **strName** variable

## CH7\_F3 - aligning characters in string & columns - methods: **PadLeft** & **PadRight** & function: **String.Space** - additional topic from Appendix B

- you can use those Visual Basic's methods to **align** the characters in a string
- the methods do this by **inserting - padding** (vycpáním, obložením) the string with zero or more characters until the string is a specified length ->
  - > each method then returns the padded string
- when processing those methods, the computer first makes a temporary copy of the string in memory, and then it pads the copy only
- method **PadLeft**    - pads the string on the left, which means it inserts the padded characters at the beginning of the string, thereby:
  - right-aligning the characters within the string
  - can be used to align columns of information, as shown in: [e.g.4](#) & [e.g.5](#)
- method **PadRight**    - pads the string on the right, which means it inserts the padded characters at the end of the string, thereby:
  - left-aligning the characters within the string
  - can be used to align columns of information, as shown in: [e.g.4](#) & [e.g.5](#)
- function **Space**    - includes a specific number of space characters in a string

String method PadLeft syntax:

String method PadRight syntax:

**As String**

`string.PadLeft(totalChars As Integer, padCharacter As Char)`

`string.PadRight(totalChars As Integer, padCharacter As Char)`

**As String**

**As String**

String function Strings.Space syntax:

simplified syntax:

**As String**

`Strings.Space(numberOfSpaces As Integer)`

`Space(numberOfSpaces As Integer)`

**As String**

**As String**

**string**    **- As String**

= can be: a String variable; a String named constant; the Text property of a control, like **txt**, **cbo**, **lst**...etc

**totalChars**    **- As Integer**

= argument, an integer that specifies the total number of characters you want the string's copy to contain

**padCharacter**    **- OPTIONAL**

**- As Char (c)**

- if omitted, the default padding character is the space character

= argument; the character that is used to pad the string until the desired number of characters is reached ->  
-> specified in **totalChars**

**- character** = enclosed in quotation marks, followed by letter **c**: **"\*"**c****

**- c** = literal type character converts the string to the **Char** data type

**CH3\_F3.1 - ...appropriate data type....**

**Strings**    **= Module Microsoft.VisualBasic.Strings**

= a type and cannot be used as an expression

**Space**    **= function** (returns a value)

**- As Integer**

= includes a specific number of space characters in a *string*

**e.g.**

`MessageBox.Show("First name is:" & Space(6) & strFirstName)`

e.g.1

```
Dim strNumber As String = "300.99"
txtNum.Text = strNumber.PadLeft(9)
<- assigns the string " 300.99" to the txtNum.Text property
```

<- there are **6** characters in the **strNumber** variable  
 <- so,  $9 - 6 = 3$  space characters at the beginning of the string

e.g.2

```
Dim strFirst As String = "Charles"
strFirst = strFirst.PadRight(15)
<- assigns the string "Charles" to the strFirst variable
```

<- there are **7** characters in the **strFirst** variable  
 <- so,  $15 - 7 = 8$  space characters at the end of the string

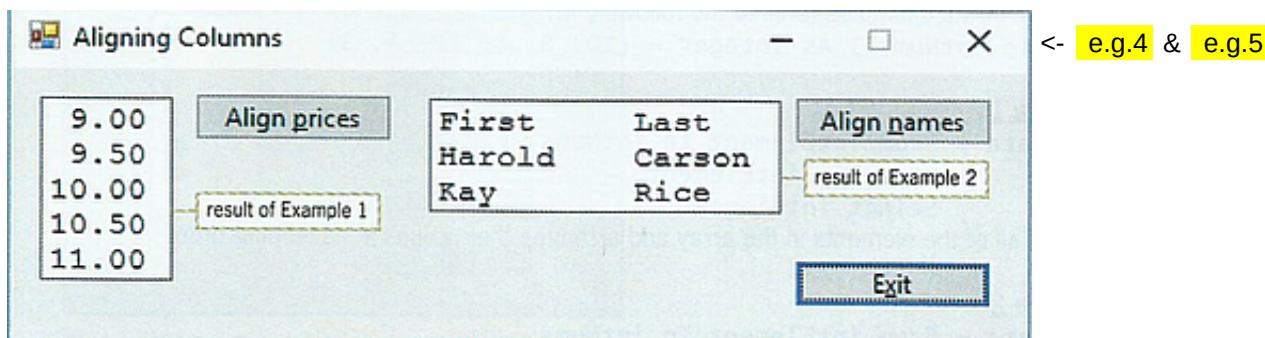
e.g.3

```
Dim dblNet As Double = 633.75
Dim strNet As String
strNet = dblNet.ToString("C2").PadLeft(10, "***c")
lblShow = strNet
<- assigns the string "***$633.75" to the strNet variable
```

**c** = literal type character converts the string to the Char data type

\*\*\*\$633.75

- many companies use this type of formatting on their employee paychecks because it makes it more difficult for someone to change the amount.



e.g.4 - code aligns a column of numbers by the decimal point

```
Dim strPrice As String
lstPrices.Items.Clear()
For dblPrice As Double = 9 To 11 Step 0.5
    strPrice = dblPrice.ToString("N2").PadLeft(5)
    lstPrices.Items.Add(strPrice)
Next dblPrice
```

<- notice that:

**1st** you format each number in the column to ensure that each has the same number of digits to the right of the decimal point -> **ToString("N2")**, so the numbers with a **2** decimal places  
**2nd** you use the **PadLeft** method to insert spaces at the beginning - if necessary - so, this right-aligns the number within the column -> **PadLeft(5)**, where total characters are **5** - so, because each number has the same number of digits to the right of the decimal point, aligning each number on the right will align each by its decimal point

e.g.5 - code shows how you can align the second column of information when the first column contains strings with varying lengths

```
Dim strFirst1 As String = "Harold"
Dim strLast1 As String = "Carson"
Dim strFirst2 As String = "Kay"
Dim strLast2 As String = "Rice"
lstNames.Items.Clear()
lstNames.Items.Add("First" & Strings.Space(5) & "Last")
lstNames.Items.Add(strFirst1.PadRight(10) & strLast1)
lstNames.Items.Add(strFirst2.PadRight(10) & strLast2)
```

<- notice that:

**1st** you use either the **PadRight** or **PadLeft** method to ensure that each string in the 1st column contains the same number of characters -> **PadRight(10)**, where total char is **10**  
**2nd** you concatenate the padded string to the information in the second column - the example also shows how you can use the **Strings.Space** function to include a specific number of space characters in a string

## CH7\_F3.1 - Insert method & PadLeft method example: The Net Pay Application (02.Net Pay Solution)

- the Net Pay application will use the **Insert** and **PadLeft** methods to display an employee's net pay with a **leading dollar sign**, **asterisks**, and **two decimal places**

1). open the: ...VB2017\Chap07\_Exercise\02.Net Pay Solution\Net Pay Solution.sln

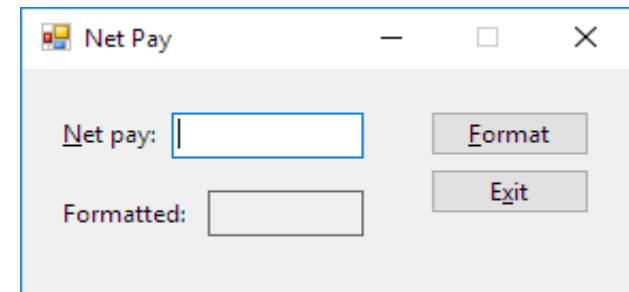
2). open the Code Editor window and locate the **btnFormat\_Click** procedure

- first, the procedure will format the net pay to include **two decimal places**
- then, it will **pad** the net pay with **asterisks** (if necessary) until the net pay contains **10 characters**
- finally, it will **insert** a **dollar sign** at the beginning of the formatted net pay

-> type the 3 assignment statements indicated on lines **22, 24, 26**:

```
14      Private Sub btnFormat_Click(sender As Object, e As EventArgs) Handles btnFormat.Click
15          ' Format the net pay.
16
17          Dim decNet As Decimal
18          Dim strFormatted As String
19
20          Decimal.TryParse(txtNet.Text, decNet)
21          ' Format with two decimal places.
22          strFormatted = decNet.ToString("N2")
23          ' Pad with asterisks until 10 characters.
24          strFormatted = strFormatted.PadLeft(10, "***c")           <- need to force the literal "c", because the method syntax: (As Char - !!!)
25          ' Insert a dollar sign at the beginning.                  string.PadLeft(totalChars As Integer, padCharacter As Char) As String
26          strFormatted = strFormatted.Insert(0, "$")                <- no need to force literal type "c" because the method syntax: (As String)
27
28          lblFormatted.Text = strFormatted
29      End Sub
```

<- **ToString** method > line **22**  
<- **PadLeft** method > line **24**  
<- **Insert** method > line **26**

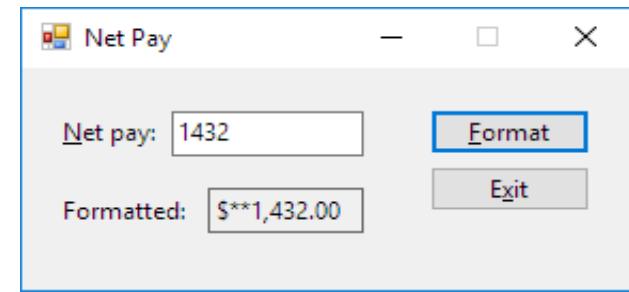


3). save the solution and then start the application:

-> type **1432** as the net pay and then click the button **Format**:

- the button's Click event procedure displays: **\$\*\*1,432.00** in the interface

4). click the button **Exit** and close the Code Editor window and then close the solution



### Mini-Quiz 7-1:

1. Write a statement that assigns the number of characters in the **strZip** variable to the **intNum** variable.
2. Write a statement that uses the **Insert** method to change the contents of the **strState** variable from "Ky" to "Kentucky"
3. Write a statement that uses the **PadRight** method to change the contents of the **strBonus** variable from "100" to "100\$\$\$".

```
1...intNum = strZip.Length.  
2...strState = strState.Insert(1, "entucky")  
3...strBonus = strBonus.PadRight(6, "$")
```

### CH7\_F3.2 - You Do It 1: Trim method, PadRight method, Insert method exercise: 03.You Do It 1 Solution

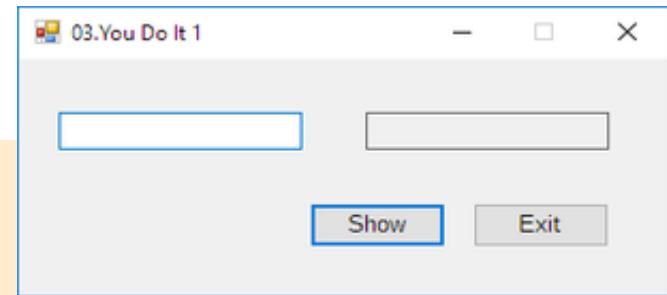
1. create an application named **You Do It 1** and save it in the ...VB2017\Chap07\Exercise\03.You Do It 1 Solution
2. add: a **text box**, a **label**, and a **button** to the form
3. the button's **Click** event procedure should assign the contents of the **text box**, excluding any leading or trailing space characters, to a **String** variable
4. if the variable contains at least two characters, but less than seven characters, the procedure should **insert** a number sign (#) as the second character and then **pad** the variable's value (on the right) with asterisks until the variable contains 10 characters
5. finally, the procedure should display the variable's contents in the **label**
6. code the procedure, save the solution and then start and test the application:

-> if you enter: 1234567, the label will display: 1234567

-> if you enter: 123456, the label will display: 1#23456\*\*\*

my GUI & code:

```
1  Option Explicit On
2  Option Strict On
3  Option Infer Off
4
5  Public Class frmMain
6      Private Sub btnShow_Click(sender As Object, e As EventArgs) Handles btnShow.Click
7          Dim strNumber As String = txtNumber.Text.Trim
8          'strNumber = txtNumber.Text.Trim
9          If strNumber.Length > 1 AndAlso strNumber.Length < 7 Then
10              strNumber = strNumber.Insert(1, "#")
11              strNumber = strNumber.PadRight(10, "*")
12              lblShow.Text = strNumber
13          Else
14              lblShow.Text = strNumber
15          End If
16      End Sub
17
18      Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
19          Me.Close()
20      End Sub
21
22      Private Sub txtNumber_Enter(sender As Object, e As EventArgs) Handles txtNumber.Enter
23          txtNumber.SelectAll()
24      End Sub
25
26      Private Sub txtNumber_TextChanged(sender As Object, e As EventArgs) Handles txtNumber.TextChanged
27          lblShow.Text = String.Empty
28      End Sub
29
30  End Class
```



## CH7\_F4 - Search for a specific sequence of characters: Contains and IndexOf methods of the String:

- to determine whether a string contains a specific sequence of characters, you can use either the **Contains** method, or the **IndexOf** method

### CH7\_F4.1 - String search: Contains method

- performs a case-sensitive search (same like **IndexOf** method), which means the case of the **subString** must match the case of the **string** in order for both to be considered equal (consider usage of the **ToUpper**, or the **ToLower** methods to skip the case-sensitive)
- returns the Boolean value **True** when the **subString** is contained anywhere in the string - otherwise, it returns the Boolean value **False**
- **Contains** method always begins the search with the first character in the string

String method Contains syntax:

**As Boolean**

**string.Contains(subString As String) As Boolean**

**string**      **- As String**

= can be: a String variable, a String named constant, the Text property of a control, like (cbo), (txt)...

**subString**    **- As String**

= argument, represents the sequence of characters for which you are searching -> case-sensitive

e.g.1

```
Dim blnIsContained As Boolean  
txtCityState.Text = "Louisville, KY"  
blnIsContained = txtCityState.Text.Contains("KY")
```

<- the Contains method performs a case-sensitive search

<- assigns **True** to the **blnIsContained** variable because the **subString** "KY" appears in the **txtCityState.Text** property

e.g.2

```
Dim blnIsContained As Boolean  
Dim strCityState As String = "Louisville, KY"  
blnIsContained = strCityState.Contains("Ky")
```

<- the Contains method performs a case-sensitive search

<- assigns **False** to the **blnIsContained** variable because the **subString** "Ky" does not appear in the **strCityState** variable

e.g.3

```
Dim strAddress As String = "75 Main St."  
If strAddress.ToUpper.Contains("MAIN") Then
```

<- the **ToUpper** method will be evaluated before the **Contains** method

<- the condition evaluates to **True** because the **subString** "MAIN" appears in the **strAddress** variable when the variable's contents are temporarily converted to uppercase

### CH7\_F4.2 - String search: IndexOf method

- performs a case-sensitive search (same like **Contains** method), which means the case of the **subString** must match the case of the **string** in order for both to be considered equal (consider usage of the **ToUpper**, or the **ToLower** methods to skip the case-sensitive)
- returns an Integer:
  - a). either **-1**                          -> indicates that the **subString** is not contained in the **string**
  - b). or a number that is greater than or equal to **0**                          -> a number other than **-1** is the character index of the **subString**'s starting position in the **string**
- unless you specify otherwise, the **IndexOf** method starts the search with the first character in the string, but ->
  - > to specify a different starting location, you use the **optional** argument **startIndex**

String method IndexOf syntax:  
As Integer

`string.IndexOf(subString As String, startIndex As Integer) As Integer`

`string`

- As String

= can be: a String variable, a String named constant, the Text property of a control, like (cbo), (txt)...

`subString`

- As String

= argument, represents the sequence of characters for which you are searching -> case-sensitive

`startIndex`

- OPTIONAL

- As Integer

= argument, zero-based starting character position you want to access

- if omitted, the default starting location index is 0 (the first character in the string)

e.g.1

```
Dim intCharIndex As Integer  
Dim strLocation As String = "Dallas, TX"  
intCharIndex = strLocation.IndexOf("TX")
```

<- "T" = character index 8

<- assigns the number 8 to the `intCharIndex` variable because the `subString` "TX" appears in the `strLocation` variable, beginning with the character whose index is 8

e.g.2

```
Dim intCharIndex As Integer  
Dim strLocation As String = "Dallas, TX"  
intCharIndex = strLocation.IndexOf("Tx")
```

<- the `IndexOf` method performs a **case-sensitive** search

<- assigns the number -1 to the `intCharIndex` variable because the `subString` "Tx" does not appear in the `strLocation` variable

e.g.3

```
Dim intCharIndex As Integer  
Dim strAddress As String = "75 Main St."  
intCharIndex = strAddress.ToLower.IndexOf("main", 4)
```

<- character index 4 = a (0-1-2-3-4)

<- the `ToLower` method will be evaluated before the `IndexOf` method

<- assigns the number -1 to the `intCharIndex` variable because the `subString` "main" does not appear in the `strAddress` variable when the search starts with the character whose index is 4 (the letter a)

#### CH7\_F4.3 - String search: `IndexOf` method example: The City and State Application (04.City State Solution)

- the **City and State application** will use the `IndexOf` method to locate the comma contained in a string

1). open the: ...VB2017\Chap07\Exercise\04.City State Solution\City State Solution.sln

2). open the Code Editor window and locate the `btnLocate_Click` procedure:

>- enter the indicated assignment statement on line 30

- btw: you can also include the `startIndex` argument in the `IndexOf` method, like this: (" , ", 0)

```
22      Private Sub btnLocate_Click(sender As Object, e As EventArgs) Handles btnLocate.Click  
23          ' Displays the index of the comma contained in a string.  
24      End Sub
```

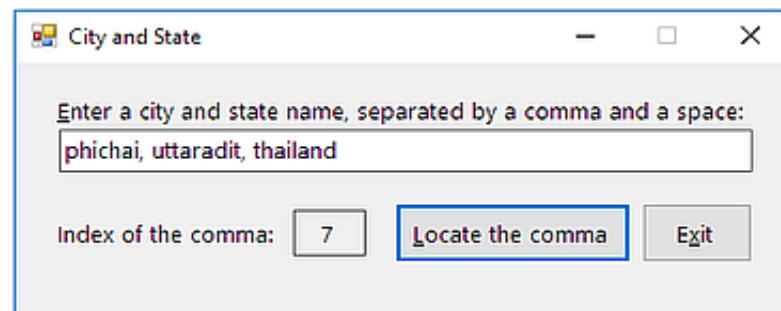
```

25      Dim strCityState As String
26      Dim intIndex As Integer
27
28      strCityState = txtCityState.Text
29      ' Determine the comma's index.
30      intIndex = strCityState.IndexOf(",")
31
32      lblIndex.Text = intIndex.ToString
33  End Sub

```

- 3). save the solution and then start and test the application:

- > type: **Chiang Mai, Thailand** and click the button **Locate the comma** --> Index of the comma = **10**
- > type: **Phichai, Uttaradit, Thailand** and click the button **Locate the comma** --> Index of the comma = **7**, because the default starting location index is **0**
- > type: **Ban Hatapja** and click the button **Locate the comma** --> Index of the comma = **-1**, because the text box does **not** contain a comma



- 4). click the button **Exit**, close the Code Editor window and then close the solution.

#### Mini-Quiz 7-2:

1. Write a statement that uses the **Contains** method to determine whether the **strAddress** variable contains the string: "**Elm St.**" (in uppercase, lowercase, or a combination of uppercase and lowercase). Assign the return value to the **bInlsContained** variable.
2. Write a statement that uses the **IndexOf** method to determine whether the **strAddress** variable contains the string "**Elm St.**" (in uppercase, lowercase, or a combination of uppercase and lowercase). Assign the return value to the **intIndex** variable.
3. What does the **IndexOf** method return when the string does **not** contain the **subString**?
4. What does the **Contains** method return when the string does **not** contain the **subString**?

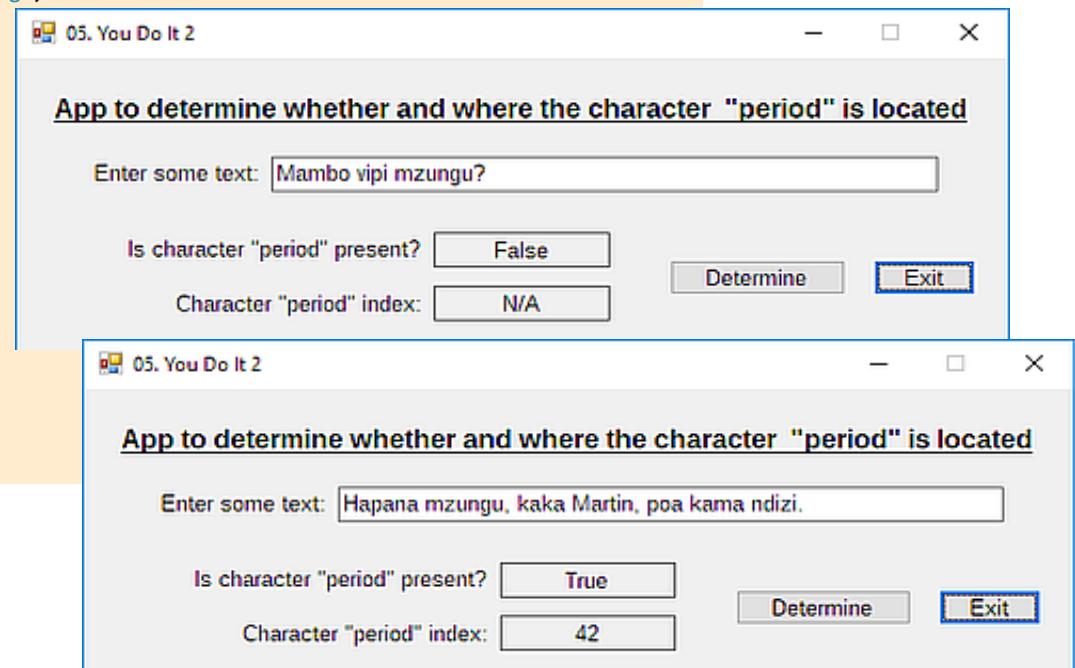
1. bInlsContained = strAddress.ToUpper.Contains("Elm St.")
2. intIndex = strAddress.ToLower.IndexOf("elm st.")
3. IndexOf AS integer, therefore it returns -1
4. Contains AS Boolean, therefore it returns False

#### CH7\_F4.4 - You Do It 2: Contains method and IndexOf method exercise: 05.You Do It 2 Solution

1. create an application named **You Do It 2** and save it in the ...VB2017\Chap07\Exercise\05.You Do It 2 Solution
2. add: a **text box**, two **labels**, and a **button** to the form
3. the button's **Click** event procedure should determine whether a period appears anywhere in the **text box** and then display the result (either **True** or **False**) in the **first label**.
4. if the **text box** contains a period, the procedure should display its index in the **second label**; otherwise it should display: **N/A**.
5. code the procedure, save the solution, start and test the application.

my GUI & code:

```
1  Option Explicit On
2  Option Strict On
3  Option Infer Off
4  Public Class frmMain
5      Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
6          Me.Close()
7      End Sub
8      Private Sub txtenter_Enter(sender As Object, e As EventArgs) Handles txtenter.Enter
9          txtenter.SelectAll()
10     End Sub
11     Private Sub txtenter_TextChanged(sender As Object, e As EventArgs) Handles txtenter.TextChanged
12         lbl1present.Text = String.Empty
13         lbl2index.Text = String.Empty
14     End Sub
15     Private Sub btnDetermine_Click(sender As Object, e As EventArgs) Handles btnDetermine.Click
16         Dim strText As String = txtenter.Text
17         ' Dim bolPresent As Boolean
18         Dim intIndex As Integer
19
20         If txtenter.Text.Contains(".") Then
21             lbl1present.Text = "True"
22             intIndex = txtenter.Text.IndexOf(".")
23             lbl2index.Text = intIndex.ToString
24         Else
25             lbl1present.Text = "False"
26             lbl2index.Text = "N/A"
27         End If
28     End Sub
29 End Class
```



## CH7\_F5 - Substring method of the String:

- Visual Basic provides the **Substring** method for accessing any number of characters in a string

String method **Substring** syntax:  
**As String**

```
string.Substring(startIndex As Integer, Length As Integer) As String
```

|                   |                                                                                                                                                                                                                                    |
|-------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>string</i>     | <b>- As String</b><br>= can be: a String variable, a String named constant, the Text property of a control, like (cbo), (txt)...                                                                                                   |
| <i>startIndex</i> | <b>- As Integer</b><br>= argument, zero-based starting character position you want to access                                                                                                                                       |
| <i>Length</i>     | <b>- OPTIONAL</b><br><b>- As Integer</b><br>- argument, specifies the number of characters you want to access<br>- if omitted, the method returns all characters from the <i>startIndex</i> position through the end of the string |

e.g.1

```
Dim strFirst As String
Dim strLast As String
Dim strFull As String = "Laquisha Jones"
strFirst = strFull.Substring(0, 8)
strLast = strFull.Substring(9)

Or

strLast = strFull.Substring(9, 5)
```

<- "L" = character index 0....."J" = character index 9  
<- assigns the string "Laquisha" to the **strFirst** variable  
<- assigns the string "Jones" to the **strLast** variable  
  
<- assigns the string "Jones" to the **strLast** variable

e.g.2

```
Dim strEmployeeNum As String = "38F45"
strDepartment = strEmployeeNum.Substring(2, 1)
```

<- "F" = character index 2  
<- assigns the string "F" to the **strDepartment** variable

## CH7\_F5.1 - Substring method of the String example: The Rearrange Name Application (06.Rearrange Solution)

- you will use the **Substring** method in the **Rearrange Name application**
- the application's GUI provides a text box for entering a person's first name followed by a space and the person's last name -> e.g.: "**Antonio Vivaldi**"
- the application rearranges the name so that the last name comes first, followed by a comma, a space, and the first name -> e.g.: "**Vivaldi, Antonio**"

- 1). open the: ...VB2017\Chap07\Exercise\06.Rearrange Solution\Rearrange Solution.sln
  - 2). open the Code Editor window and locate the **btnRearrange\_Click** procedure
    - the procedure assigns the name entered by the user, excluding any leading or trailing spaces, to the **strName** variable
  - 3). before you can rearrange the name stored in the variable, you need to separate the first name from the last name -->
    - > to do this, you first search for the space character that appears between the names:
  - 4). if the value in the **intIndex** variable is not **-1**, it means that the **IndexOf** method found a space character in the variable -->
    - > in that case, the **selection structure's True** path should continue rearranging the name; otherwise, its **False** path should display an appropriate message
- 24      intIndex = strName.IndexOf(" ")
- 26      If intIndex <> -1 Then

5). now you can use the value stored in the **intIndex** variable to separate the first name from the last name:

- all of the characters to the left of the space character represent the **first name**, and all of the char. to the right of the space character represent the **last name**

```
28     strFirstName = strName.Substring(0, intIndex)
29     strLastName = strName.Substring(intIndex + 1)
```

6). finally, you will display the rearranges name in the GUI:

```
32     lblRearrange.Text = strLastName & ", " & strFirstName
```

7). save the solution and then start and test the application:

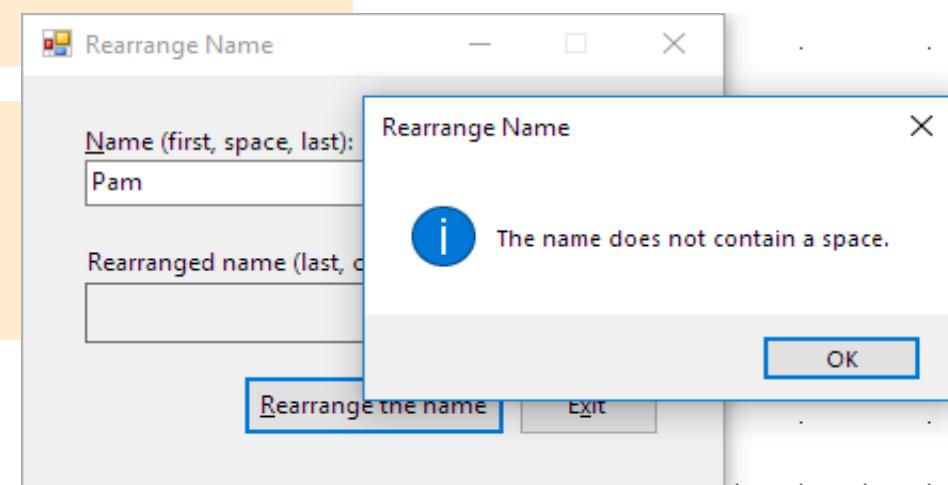
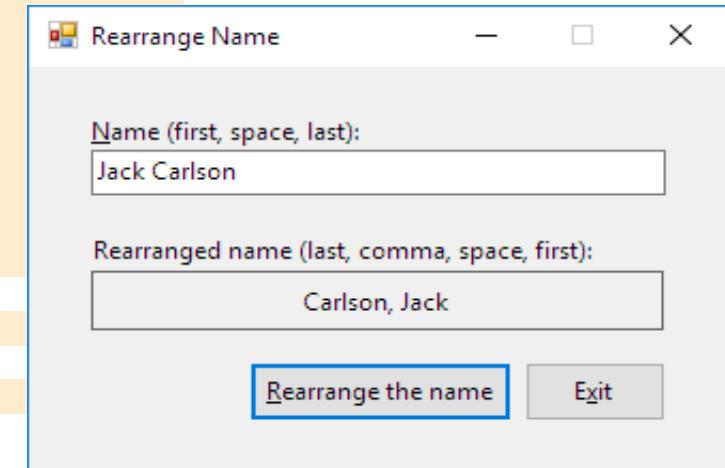
-> type **Jack Carlson** as the name and then click the button **Rearrange the name**

- the result should be: **Carlson, Jack**

-> change the name to **Pam** and then click the button **Rearrange the name**

- the message "**The name does not contain a space.**" appears in a message box

```
14     Private Sub btnRearrange_Click(sender As Object, e As EventArgs) Handles btnRearrange.Click
15         ' Rearranges and then displays a name.
16
17         Dim strName As String
18         Dim strFirstName As String
19         Dim strLastName As String
20         Dim intIndex As Integer
21
22         strName = txtName.Text.Trim
23         ' Search for the space in the name:
24         intIndex = strName.IndexOf(" ")
25
26         If intIndex <> -1 Then
27             ' Separate the first and last names:
28             strFirstName = strName.Substring(0, intIndex)
29             strLastName = strName.Substring(intIndex + 1)
30
31             ' Display last name, comma, space, and the first name:
32             lblRearrange.Text = strLastName & ", " & strFirstName
33
34         Else
35             MessageBox.Show("The name does not contain a space.",
36                             "Rearrange Name", MessageBoxButtons.OK,
37                             MessageBoxIcon.Information)
38         End If
39     End Sub
```

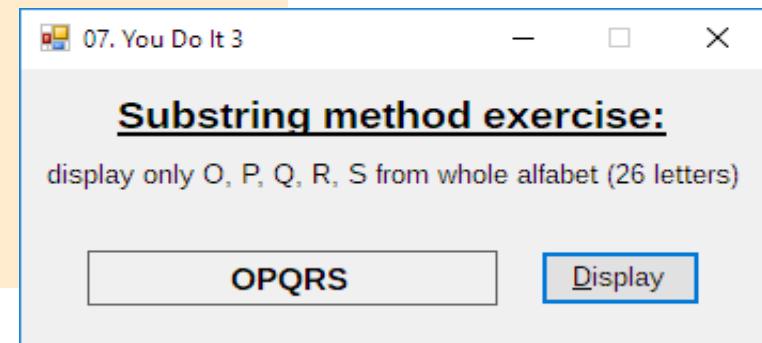


## CH7\_F5.2 - You Do It 3: Substring method exercise: 07.You Do It 3 Solution

1. create an application named **You Do It 3** and save it in the ...VB2017\Chap07\Exercise\07.You Do It 3 Solution
2. add: a **label** and a **button** to the form
3. the button's **Click** event procedure should declare a String variable named **strAlphabet** and initialize it to the **26 uppercase letters of the alphabet**
4. it then should use the **Substring** method to display only the letters: **O, P, Q, R, S** in the label
5. code the procedure, save the solution, start and test the application.

my GUI & code:

```
1  Option Explicit On
2  Option Strict On
3  Option Infer Off
4
5  Public Class frmMain
6      Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
7          lblDisplay.Text = String.Empty
8
9          'declare 26 characters of the alphabet:
10         Dim strAlphabet As String = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
11
12         'show only OPQRS:
13         lblDisplay.Text = strAlphabet.Substring(14, 5)
14     End Sub
15 End Class
```



## CH7\_F6 - Character Array: using Character's index to access a Character in a String

(array = řada, řetězec)

- a string is simply a group of related characters and is commonly referred to as an **array of characters**, or a **character array**
- each character in a character array, or string, has a zero-based unique index that represents its position in the string ->
  - > you can use the index to refer to an individual character in the string
- you do this by using the name of the location (variable, named constant, Text property) where the string is stored, followed by the character's index enclosed in parentheses      e.g. **strAlphabet(0)**

Character's index syntax:  
**As Character**

```
string(Index As Integer) As Character
```

*string*      **- As String**  
= can be: a String variable, a String named constant, the Text property of a control, like (cbo), (txt)...

*Index*      **- As Integer**  
= argument, zero-based character position you want to access

e.g.1

```
Dim strName As String = "Martin"  
strName(0)    -> M  
strName(1)    -> a  
strName(2)    -> r  
strName(3)    -> t  
strName(4)    -> i  
strName(5)    -> n
```

e.g.2

```
Dim strName As String = "Martin"  
lblLetter.Text = strName(0)           <- equivalent to: lblLetter.Text = strName.Substring(0, 1)  
-> assigns the first character in the strName variable (the letter M) to the lblLetter.Text property
```

e.g.3

```
Dim strName As String = "Martin"  
lblLetter.Text = strName(5)           <- equivalent to: lblLetter.Text = strName.Substring(5, 1)  
-> assigns the last character in the strName variable (the letter n) to the lblLetter.Text property
```

- although the character's index can typically be used in place of the Substring method to refer to one character, there is a difference between both :
  - > the character's index returns a value that has the **Char** data type (Character)
  - > whereas the Substring method returns a value that has the **String** data type
- this is important because you cannot use a String method, such as **ToUpper**, on the value returned by using character's index -> you would first need to use the **ToString** method to convert the **Char** value to the String data type

<- e.g. strName(0)

<- e.g. strName.Substring(0, 1)

<- e.g. strName(3).ToString.ToUpper

#### CH7\_F6.1 - using Character's index to access a Character in a String example: The First Name Application (08. Name Solution)

- the First Name application's GUI provides a text box for entering a person's first name
- the List characters button in the GUI uses the index of each character in the name to add the individual characters to a list box

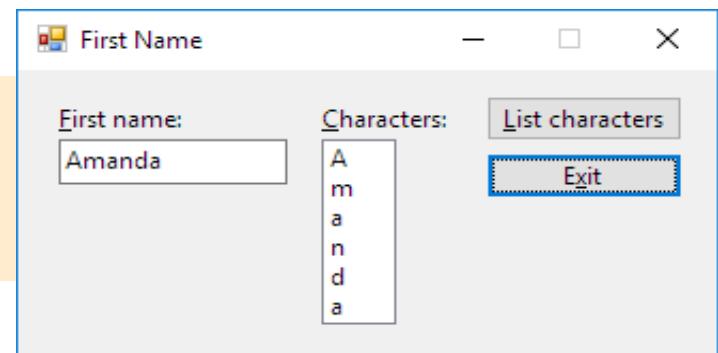
1). open the: ...VB2017\Chap07\\_Exercise\08.Name Solution\Name Solution.sln

2). open the Code Editor window and locate the **btnList\_Click** procedure

- the procedure assigns the name entered by the user, excluding any leading or trailing spaces, to the **strName** variable
- it then clears the contents of the **lstCharacters** list box

-> enter the indicated **For...Next** loop on lines 20 to 22:

```
14      Private Sub btnList_Click(sender As Object, e As EventArgs) Handles btnList.Click  
15          ' Add each character in a string to a list box.  
16          Dim strName As String  
17          strName = txtName.Text.Trim  
18          lstCharacters.Items.Clear()  
19  
20          For intIndex As Integer = 0 To strName.Length - 1  
21              lstCharacters.Items.Add(strName(intIndex))  
22          Next intIndex  
23      End Sub
```



3). save the solution and then start and test the application:

-> type **Amanda** in the First name box and then click the button **List characters**

- each character in the name appears on a separate line in the list box

4). click the button **Exit**, close the Code Editor window and then close the solution.

### Mini-Quiz 7-3:

1. The **strItem** variable contains the string "**XMr**edBQ**".**

Write a statement that uses the **Substring** method to assign the string "**red**" to the **strColor** variable.

2. The **strName** variable contains the string "**Jane H. Doe**".

Write a statement that uses the **Substring** method to assign the string "**H**" to the **strInitial** variable.

3. Rewrite the statement from Question 2 using the **H character's index**.

```
1...strColor = strItem.Substring(2, 3)  
2...strInitial = strName.Substring(5, 1)  
3...strInitial = strName(5)
```

### CH7\_F6.2 - You Do It 4: Character's index exercise: 09.You Do It 4 Solution

1. create an application named **You Do It 4** and save it in the ...VB2017\Chap07\Exercise\09.You Do It 4 Solution

2. add: **two labels** and **two button** to the form

3. create a **class-level** variable named **strLetters** and initialize it to the first 10 uppercase letters of the alphabet (the letters A through J)

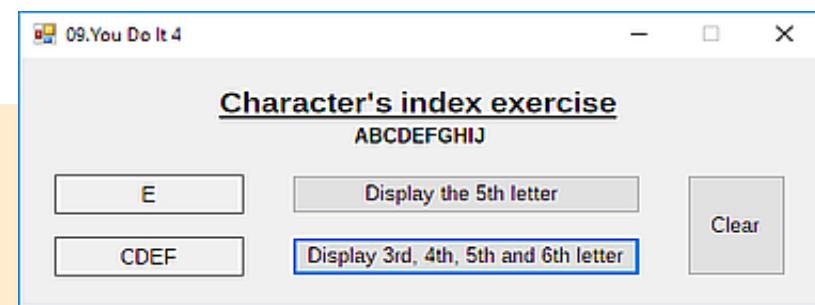
4. the **first button**'s Click event procedure should use the approp. **character's index** to display the **fifth** letter from the **strLetters** variable (letter E) in the **first label**

5. the **second button** should use a **loop** and the appropriate **indexes** to display the letters **CDEF** in the **second label**

6. code the procedures, save the solution and then start and test the application.

my GUI & code:

```
1  Option Explicit On  
2  Option Strict On  
3  Option Infer Off  
4  Public Class frmMain  
5      Public strLetters As String = "ABCDEFGHIJ"  
6  
7      Private Sub btn1_Click(sender As Object, e As EventArgs) Handles btn1.Click  
8          lbl1.Text = strLetters(4)  
9      End Sub  
10  
11      Private Sub btn2_Click(sender As Object, e As EventArgs) Handles btn2.Click  
12          For intA As Integer = 2 To 5  
13              lbl2.Text += strLetters(intA)  
14          Next intA  
15      End Sub  
16  
17      Private Sub btnClear_Click(sender As Object, e As EventArgs) Handles btnClear.Click  
18          lbl1.Text = String.Empty  
19          lbl2.Text = String.Empty  
20      End Sub  
21  End Class
```



## CH7\_F7 - Remove method of the String:

- you can use Visual Basic's **Remove** method to remove a specified number of characters located anywhere in a string
- the method returns a string with the appropriate characters removed
- when processing the method, the computer first stores a temporary copy of the **string** in its main memory
  - it then performs the specified removal on the copy only -> it does not remove any characters from the original **string**

### String method Remove syntax:

**As String**

**string.Remove(startIndex As Integer, Count As Integer) As String**

**string**

#### **- As String**

= can be: a String variable, a String named constant, the Text property of a control, like (cbo), (txt)...

**startIndex**

#### **- As Integer**

= argument, zero-based starting character position you want removed from the copy of the **string**

- to remove only the first character from a string, you use **0** as the **startIndex** and **1** as the **Count**

**Count**

#### **- OPTIONAL**

#### **- As Integer**

- argument, specifies the number of characters you want removed

- to remove only the first character from a string, you use **0** as the **startIndex** and **1** as the **Count**

- if omitted, the method removes all of the characters from the **startIndex** position through the end of the string

e.g.1

```
Dim strCityState As String = "Atlanta, GA"  
txtState.Text = strCityState.Remove(0, 9)  
txtState.Text = strCityState.Remove(0, 1)  
txtState.Text = strCityState.Remove(3, 5)  
txtState.Text = strCityState.Remove(7)  
  
Or:  
txtState.Text = strCityState.Remove(7, 4)
```

<- assigns the string "**GA**" to the **txtState.Text** property  
<- removes only the first character, assigns the string "**lanta, GA**"  
<- removes the 4th through 8th characters (3+5), assigns the string "**AtlGA**"  
<- removes everything from 8th character, assigns the string "**Atlanta**"

<- removes the 8th through 11th characters (7+4), assigns the string "**Atlanta**"

e.g.2

```
Dim strFirst As String = "John"  
strFirst = strFirst.Remove(2, 1)
```

<- assigns the string "**Jon**" to the **strFirst** variable

## CH7\_F8 - Trim, TrimStart and TrimEnd methods of the String:

- in **CH4\_F6**, you learned how to use the **Trim** method to remove space characters from both the beginning and end of a string
- you can also use the **Trim** method to remove other characters, such as the dollar sign or percent sign
- if you need to remove characters from only the beginning of a string, you use the **TrimStart** method, similarly
- if you need to remove characters from only the end of a string, you use the **TrimEnd** method
- when processing the methods, the computer makes a temporary copy of the **string** in memory -> it then removes the characters from the copy only

String method Trim syntax:

String method TrimStart syntax:

String method TrimEnd syntax:

**As String**

```
string.Trim(TrimChars As Char) As String  
string.TrimStart(TrimChars As Char) As String  
string.TrimEnd(TrimChars As Char) As String
```

= removes specified leading & trailing characters

= removes specified leading characters

= removes specified trailing characters

**string**

**- As String**

= can be: a String variable, a String named constant, the Text property of a control, like (cbo), (txt)...

*TrimChars*

**- OPTIONAL**

**- As Char**, therefore need to use forced literal type: **c**

- argument, comma-separated list of characters that you want removed (trimmed)

- if omitted, the default value is the space character (" "c)

e.g.1

```
Dim strPrice As String  
txtPrice.Text = "$$$34$$"  
strPrice = txtPrice.Text.Trim("$"c)
```

<- removes all "\$" characters and assigns the remaining string "34" to the **strPrice** variable

e.g.2

```
Dim strSales As String  
txtSales.Text = " $456.25 "  
strSales = txtSales.Text.Trim("$"c, " "c)
```

<- removes all: "\$" & space characters " ", and assigns the string "456.25" to the **strSales** variable

e.g.3

```
Dim strPay As String = "$$***340.56***$$"  
strPay = strPay.Trim("$"c, "*"c)
```

<- removes all: "\$" & "\*" characters **everywhere** in the string and  
assigns the string "340.56" to the **strPay** variable

e.g.4

```
Dim strPay As String = "$$***340.56***$$"  
strPay = strPay.TrimStart("$"c, "*"c)
```

<- removes all: "\$" & "\*" characters **only** from the **start** of the string and  
assigns the string "340.56\*\*\*\$\$" to the **strPay** variable

e.g.5

```
Dim strPay As String = "$$***340.56***$$"  
strPay = strPay.TrimEnd("$"c, "*"c)
```

<- removes all: "\$" & "\*" characters **only** from the **end** of the string and  
assigns the string "\$\$\*\*\*340.56" to the **strPay** variable

### CH7\_F8.1 - TrimEnd method example: The Tax Calculator Application (10. Tax Solution)

- you will use the **TrimEnd** method in the **Tax Calculator** application, which calculates the amount of sales tax to charge a customer

1). open the: ...VB2017\Chap07\_Exercise\10.Tax Solution\Tax Solution.sln

2). open the Code Editor window and locate the **frmMain\_Load** procedure on lines 26 through 33

- the procedure adds three tax rates to the **IstRates** control, formatting each with a percent sign and no decimal places
- it then selects the first rate in the list

```
26      Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load  
27          ' Fills the list box with tax rates.  
28  
29          For dblRate As Double = 0.02 To 0.04 Step 0.01
```

```

30         1stRates.Items.Add(db1Rate.ToString("P0"))
31     Next dblRate
32     1stRates.SelectedIndex = 0
33 End Sub

```

- 3). -> start the application and type: **100** in the Sales box and then click the button **Calculate**

- the Sales tax box shows: **\$0.00**, which is **incorrect**, so click the button **Exit**

- 4). locate the **btnCalc\_Click** procedure on line **10**

- the assignment statement on line **20** assigns the item selected in the **1stRates** control to the **strRate** variable, and the **TryParse** method on line **21** tries to convert the variable's contents to the **Double** data type <- however, as you learned in CH3\_F4, the **TryParse** method cannot convert a string that contains a **percent sign** <- you can use the **TrimEnd** method to fix this problem

-> type indicated **comment** and **TrimEnd** method:

```

10      Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
11          ' Calculates and displays the sales tax.
12
13          Dim dblSales As Double
14          Dim strRate As String
15          Dim dblRate As Double
16          Dim dblTax As Double
17
18          Double.TryParse(txtSales.Text, dblSales)
19          ' Remove trailing percent sign. (.TrimEnd("%c"))
20          strRate = 1stRates.SelectedItem.ToString.TrimEnd("%c")
21          Double.TryParse(strRate, dblRate)
22          dblTax = dblSales * dblRate / 100
23          lblTax.Text = dblTax.ToString("C2")
24      End Sub

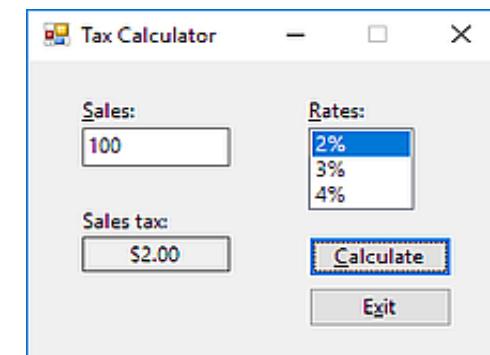
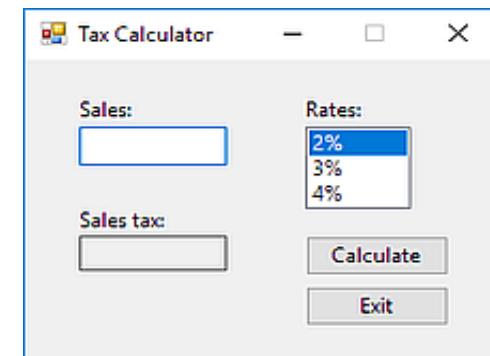
```

- 5). save the solution and then start the application

-> type **100** in the Sales box and then click the button **Calculate**

- the correct sales tax appears in the Sales tax box

- 6). click the button **Exit**, close the Code Editor window and then close the solution



#### Mini-Quiz 7-4:

- The **strAmount** variable contains the string **"1,234"**. Write a statement that uses the **Remove** method to change the variable's content to: **"1234"**.
- The **strTotal** variable contains the string **"\*\*\*75.67"**. Write a statement that uses the **TrimStart** method to change the variable's contents to **"75.67"**.
- If the **strDue** variable contains the string **"\$8.50\$"**, what will the **strDue.TrimStart("\$c")** method return?
- If the **strDue** variable contains the string **"\$8.50\$"**, what will the **strDue.Trim("\$c")** method return?

```

1. strAmount = strAmount.Remove(1, 1)
2. strTotal = strTotal.TrimStart("***")
3. "8.50"
4. "$8.50"

```

### CH7\_F9 - Replace method of the String:

- Visual Basic provides the **Replace** method for replacing a sequence of characters in a string with another sequence of characters
- when processing the method, the computer makes a temporary copy of the **string** in memory and then replaces the characters in the copy only
- the method returns a string with all occurrences of **oldValue** replaced with **newValue**

String method Replace syntax:  
As String

```
string.Replace(oldValue As String, newValue As String) As String
```

string      - As String

= can be: a String variable, a String named constant, the Text property of a control, like (cbo), (txt)...

e.g.1

```
Dim strQuote As String = "To Be Or Not To Be"  
strQuote = strQuote.Replace("To ", "2")
```

<- assigns the string "2Be Or Not 2Be" to the strQuote variable

e.g.2

```
txtCode.Text = "45-6-789"  
strCode = txtCode.Text.Replace("-", "")
```

<- assigns the string "456789" to the strCode variable

### CH7\_F10 - Like Operator to compare Strings:

- the **Like** operator allows you to use pattern-matching characters to determine whether one string is equal to another string
- performs a case-sensitive comparison of the **string** to the **pattern**
- if the **string** matches the **pattern**, the **Like** operator returns the Boolean value **True**, otherwise it returns the Boolean value **False**

String operator Like syntax:  
As Boolean

```
string Like pattern As Boolean
```

string      - As String

= can be: a String variable, a String named constant, the Text property of a control, like (cbo), (txt)...

pattern      - As

= String expression containing one or more of the pattern-matching characters listed:

| pattern-matching characters: | matches in string:                                                                                      | e.g.:                                                                                                                      |
|------------------------------|---------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------|
| ?                            | - any single character                                                                                  | "B?LL" - matches BILL, BELL, BALL...                                                                                       |
| *                            | - zero or more characters                                                                               | "K*" - matches Kentucky, KANSAS, Ky...                                                                                     |
| #                            | - any single digit (0 through 9)                                                                        | "####" - matches 1234, 1000,...                                                                                            |
| [characterList]              | - any single character in the characterList,<br>can also include a range of values using a hyphen -     | "[A5T]" - matches A, 5, or T,<br>"[a-z]" - matches any lowercase letter                                                    |
| [!characterList]             | - any single character not in the characterList,<br>can also include a range of values using a hyphen - | "[!A5T]" - matches any character other than A, 5, or T,<br>"[!a-z]" - matches any character that is not a lowercase letter |

e.g.1

```
If strFirst.ToUpper Like "B?LL" Then
```

- the condition evaluates to **True** when the string stored in the **strFirst** variable and converted to uppercase, begins with the letter **B** followed by one character and then the letters **LL** -> e.g. strings like: "Bill", "Ball", "bell", "bull"...
- otherwise, it evaluates to **False** -> e.g. strings like: "BPL", " BLL", "billy"...

e.g.2

```
If txtState.Text Like "K*" Then
```

- the condition evaluates to **True** when the value in the **txtState** control's Text property begins with the letter **K** followed by zero or more characters  
-> e.g. strings like: "KANSAS", "Ky", "Kentucky"...
- otherwise, it evaluates to **False** -> e.g. strings like: "kansas", "ky", "Bee"...

e.g.3

```
Do While strID Like "###*"
```

- the condition evaluates to **True** when the string stored in the **strID** variable begins with three digits followed by zero or more characters  
-> e.g. strings like: "178", "983Ab"...
- otherwise, it evaluates to **False** -> e.g. strings like: "X34", "34Z5"...

e.g.4

```
If strFirst.ToUpper Like "T[OI]M" Then
```

- the condition evaluates to **True** when the string stored in the **strFirst** variable and converted to uppercase, is either: "**TOM**" or "**TIM**"
- otherwise, it evaluates to **False** -> e.g. strings like: "TAM", "TOMMY"...

e.g.5

```
If strLetter Like "[a-z]" Then
```

- the condition evaluates to **True** when the string stored in the **strLetter** variable is one lowercase letter
- otherwise, it evaluates to **False**

e.g.6

```
For intIndex As Integer = 0 To strInput.Length - 1
    strChar = strInput(intIndex).ToString.ToUpper
    If strChar Like "[!A-Z]" Then
        intNonLetter += 1
    End If
Next intIndex
```

<- notice the exclamation mark ! which stands for **NOT**

- compares each character contained in the **strInput** variable with the uppercase letter of the alphabet, and counts the number of characters that are not letters

e.g.7

```
If strInput Like "*.*" Then
```

- the condition evaluates to **True** when a **period** appears anywhere in the **strInput** variable; otherwise, it evaluates to **False**

e.g.8

```
If strInput.ToUpper Like "[A-Z] [A-Z]##" Then
```

- the condition evaluates to **True** when the value in the **strInput** variable, converted to uppercase, is two letters followed by two numbers
- otherwise, it evaluates to **False**

## CH7\_F10.1 - Like Operator to compare Strings example: The Inventory Application (11. Inventory Solution)

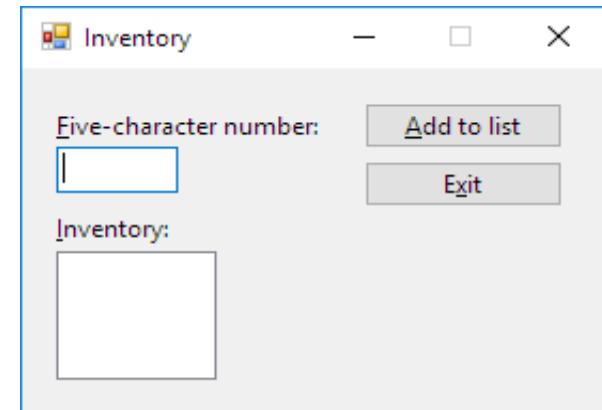
- the **Inventory application** will use the **Like** operator to verify that the inventory number entered by the user consists of three letters followed by two numbers

1). open the: ...VB2017\Chap07\_Exercise\11.Inventory Solution\Inventory Solution.sln

2). open the Code Editor window and locate the **btnAdd\_Click** procedure on line **18**

-> modify the **If** clause on line **24**:

```
18     Private Sub btnAdd_Click(sender As Object, e As EventArgs) Handles btnAdd.Click
19         ' Add an inventory number to a list box.
20
21         Dim strNumber As String
22
23         strNumber = txtNumber.Text.ToUpper.Trim
24         If strNumber Like "[A-Z][A-Z][A-Z]##" Then
25             lstNumbers.Items.Add(strNumber)
26         Else
27             MessageBox.Show("Incorrect inventory number.",
28                             "Inventory", MessageBoxButtons.OK,
29                             MessageBoxIcon.Information)
30         End If
31     End Sub
```



3). save the solution and then start and test the application:

-> first, test the application using an invalid inventory number, so type: **abc2f** as the inventory number and then click the button **Add to list**

<- the message "Incorrect inventory number" appears in a message box. Close the message box.

-> next, test the application using a valid inventory number, so change the inventory number to: **abc23** and then click the button **Add to list**

<- **ABC23** appears in the **Inventory** list box

4). click the button **Exit** and close the Code editor window and then close the solution

### Mini-Quiz 7-5:

1. The **strTotal** variable contains the string **"\*\*\*75.67"**. Write a statement that uses the **Replace** method to change the variable's contents to **"75.67"**.

2. Write an **If** clause that determines whether the **strInput** variable contains two numbers followed by an uppercase letter.

3. Write an **If** clause that determines whether the **strInput** variable contains the dollar sign followed by a number and a letter (in uppercase or lowercase).

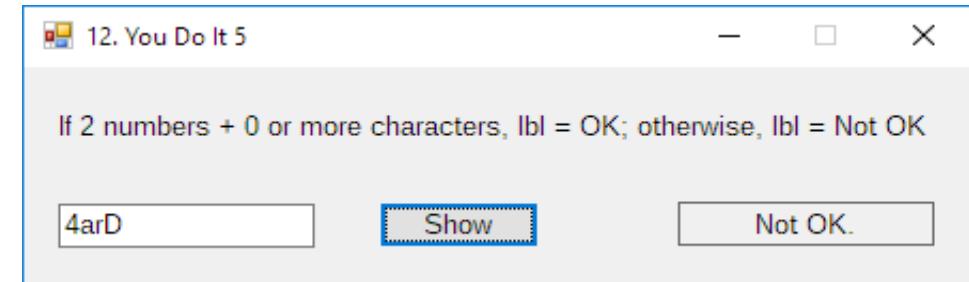
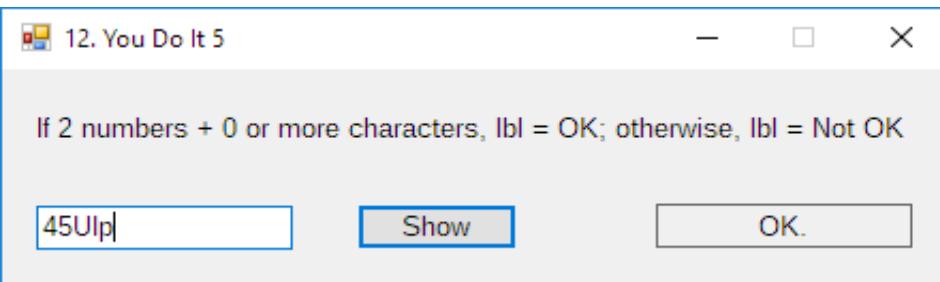
```
1...strTotal = strTotal.Replace("***", "")
2...If strInput Like "##[A-Z]" Then
3...If strInput Like "$#[a-zA-Z]" Then
```

### CH7\_F10.2 - You Do It 5: Like operator to compare strings exercise: 12.You Do It 5 Solution

1. create an application named **You Do It 5** and save it in the ...VB2017\Chap07\Exercise\12.You Do It 5 Solution
2. add: **a text box**, a **label** and a **button** to the form
3. the button's **Click** event procedure should display the message "OK" when the **text box** contains **two numbers** followed by **zero or more characters**; otherwise, it should display the message "**Not OK**"
- 4 display the message in the **label control**
5. code the procedure, save the solution and then start and test the application

my GUI & code:

```
1  Option Explicit On
2  Option Strict On
3  Option Infer Off
4
5  Public Class frmMain
6      Private Sub btnShow_Click(sender As Object, e As EventArgs) Handles btnShow.Click
7          If txtEnter.Text Like "##*" Then
8              lblMessage.Text = "OK."
9          Else
10             lblMessage.Text = "Not OK."
11         End If
12     End Sub
13
14     Private Sub txtEnter_Enter(sender As Object, e As EventArgs) Handles txtEnter.Enter
15         txtEnter.SelectAll()
16     End Sub
17
18     Private Sub txtEnter_TextChanged(sender As Object, e As EventArgs) Handles txtEnter.TextChanged
19         lblMessage.Text = String.Empty
20     End Sub
21 End Class
```



## CH7\_APPLY THE CONCEPTS LESSON

### CH7\_A1 - Code the Check Digit Application - thoroughly step by step : (13.Check Digit Solution)

- a **check digit**:
  - is a digit that is added to either the beginning or typically the end of a number for the purpose of **validating the number's authenticity**
  - used on a credit card number, a bank account number, a product's UPC (Universal Product Code), and a book's ISBN (International Standard Book Number)
- many algorithms for creating a **check digit** have been developed, including this example one, which is used for **13-character ISBNs**:

- Color legend:

Step 1 : DeepPink

Step 2 : Lime

Step 3 : PeachPuff

Step 4 : DeepSkyBlue

Step 5 : DarkGrey

#### 1. Check Digit Application Algorithm

Figure 7-25:

**Step 1:** Starting with the second digit, multiply every other digit by 3, and then total the results.

- i.e.: multiply by 3 the odd indexes 1, 3, 5, 7, 9, and 11 -> 2nd, 4th, 6th, 8th, 10th, and 12th digits, and then add together all of the products

**Step 2:** Add together each of the digits skipped in Step 1.

- these will be the even indexes 0, 2, 4, 6, 8, 10 -> 1st, 3rd, 5th, 7th, 9th, and 11th digits

**Step 3:** Add the sum from Step 1 to the sum from Step 2.

**Step 4:** Divide the sum from Step 3 by 10 and find the remainder. (10 choosed by the coder)

**Step 5:** If the remainder from Step 4 is 0, then the check digit is 0. Otherwise, subtract the remainder from 10, giving the check digit

(10 choosed by the coder)

e.g.:

ISBN without check digit: 978-1-337-10212

ISBN with check digit: 978-1-337-10212-4 calculated as shown here:

| index:         | 0         | 1              | 2    | 3              | 4    | 5              | 6    | 7              | 8    | 9              | 10   | 11             | result:                      | info:                                                                                   | variable:     |
|----------------|-----------|----------------|------|----------------|------|----------------|------|----------------|------|----------------|------|----------------|------------------------------|-----------------------------------------------------------------------------------------|---------------|
| digit:         | even      | odd            | even | odd            | even | odd            | even | odd            | even | odd            | even | odd            |                              | - blabla                                                                                |               |
| ISBN:          | 1st       | 2nd            | 3rd  | 4th            | 5th  | 6th            | 7th  | 8th            | 9th  | 10th           | 11th | 12th           |                              |                                                                                         |               |
| <b>ISBN:</b>   | 9         | 7              | 8    | 1              | 3    | 3              | 7    | 1              | 0    | 2              | 1    | 2              |                              |                                                                                         |               |
| <b>Step 1:</b> | -         | $\frac{7}{*3}$ | -    | $\frac{1}{*3}$ | -    | $\frac{3}{*3}$ | -    | $\frac{1}{*3}$ | -    | $\frac{2}{*3}$ | -    | $\frac{2}{*3}$ | = 48                         | <- every <u>odd index</u> (even number),<br><- multiply by 3, and<br><- add the results | intTotalOdd   |
| <b>Step 2:</b> | 9         | -              | 8    | -              | 3    | -              | 7    | -              | 0    | -              | 1    | -              | = 28                         | <- add all even indices together                                                        | intTotalEven  |
| <b>Step 3:</b> | 9         | 21             | 8    | 3              | 3    | 9              | 7    | 3              | 0    | 6              | 1    | 6              | = 76                         | <- add the sums together                                                                | intGrandTotal |
| <b>Step 4:</b> | 76 Mod 10 |                |      |                |      |                |      |                |      |                |      | = 6            | - number 10 set by the coder | intRemainder                                                                            |               |
| <b>Step 5:</b> | 10 - 6    |                |      |                |      |                |      |                |      |                |      | = 4            | - number 10 set by the coder | intCheckDigit                                                                           |               |

## 2. Check Digit Application GUI

Figure 7-26:

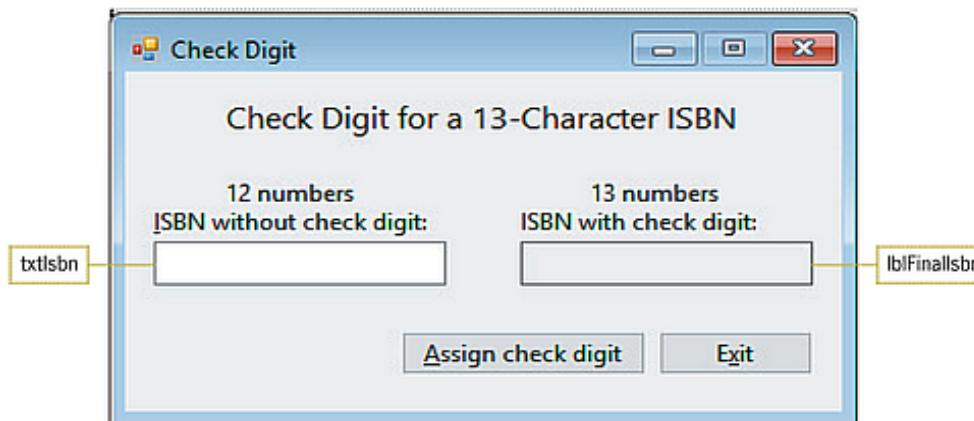


Figure 7-26 Check Digit application's interface

## 3. Check Digit Application code

1). open the: ...VB2017\Chap07\\_Exercise\13.Check Digit Solution\Check Digit Solution.sln

2). open the Code Editor window and locate the **txtIsbn\_KeyPress** procedure on line **41**

- the procedure allows the **text box** to accept only numbers and the Backspace key

```
41      Private Sub txtIsbn_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtIsbn.KeyPress
42          ' Allow only numbers and the Backspace key.
43
44          If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
45              e.Handled = True
46
47      End Sub
```

3). locate the **btnAssign\_Click** procedure on line **10**

- the procedure declares **seven variables**:

```
10      Private Sub btnAssign_Click(sender As Object, e As EventArgs) Handles btnAssign.Click
11          ' Assign a check digit to an ISBN.
12
13          Dim strIsbn As String
14          Dim intDigit As Integer
15          Dim intTotalOdd As Integer
16          Dim intTotalEven As Integer
17          Dim intGrandTotal As Integer
18          Dim intRemainder As Integer
19          Dim intCheckDigit As Integer
```

<- will store the 12 characters entered by a user in the (txtIsbn)  
<- will be used by a TryParse methods that converts each character stored in the strIsbn to an integer  
<- will store the sum from: Step 1 - odd indexes (even numbers) multiply by 3 and add the results  
<- will store the sum from: Step 2 - every even index digit (odd number) add together  
<- will store the sum from: Step 3 - add both sums together  
<- will store the remainder calculated in: Step 4 - both sums Mod 10  
<- will store the check digit calculated in: Step 5

4). before coding the check digit algorithm, you need to verify that the user entered exactly 12 characters in the **txtIsbn** control:

-> modify the selection structure's **If** clause as indicated:

```
21      If txtIsbn.Text.Length = 12 Then  
22          strIsbn = txtIsbn.Text  
23  
24      Else  
25          MessageBox.Show("Please enter 12 numbers.", "Check Digit",  
26                          MessageBoxButtons.OK, MessageBoxIcon.Information)  
27      End If
```

- if the control does not contain exactly 12 characters, the selection structure will display an appropriate message -> notice that the code to display the message has already been entered in the structure's false path

- on the other hand, if the control contains the correct number of characters, the selection structure's true path will assign the characters to the **strIsbn** variable

5). now you can start coding the check digit algorithm, **Step 1:**

- it multiplies every other digit (starting with the 2nd digit) by 3 and then totals the results -> these are the 2nd, 4th, 6th, 8th, 10th, and 12th digits in the ISBN

- in the **strIsbn** variable, however, these digits will have indexes of **1, 3, 5, 7, 9, and 11**

-> you can use a **For...Next** loop and either the **Substring** method or the **character's index** to access the individual character

<- the parentheses in the assignment statement on line **25** are not necessary, because multiplication has a higher precedence than addition - however, they serve to make the statement easier to understand

<- in the **TryParse** method on line **24**, you can use **strIsbn.Substring(intOdd, 1)** instead of **strIsbn(intOdd)**

```
22      strIsbn = txtIsbn.Text  
23      For intOdd As Integer = 1 To 11 Step 2  
24          Integer.TryParse(strIsbn(intOdd), intDigit)  
25          intTotalOdd += (intDigit * 3)  
26      Next intOdd  
27
```

<- instead of: **strIsbn(intOdd)**, you can use: **strIsbn.Substring(intOdd, 1)**  
<- parentheses are not necessary, because multiplication has a higher precedence than addition

6). **Step 2:**

- the second step in the algorithm adds together each of the digits skipped in the algorithm's **Step 1**

- these are the 1st, 3rd, 5th, 7th, 9th, and 11th digits in the ISBN

- in the **strIsbn** variable, these digits will have indexes of **0, 2, 4, 6, 8, and 10**

-> here too, you can use a **For...Next** loop and either the **Substring** method or the **character's index** to access the individual character

<- in the **TryParse** method on line **28**, you can use **strIsbn.Substring(intEven, 1)** instead of **strIsbn(intEven)**

```
26      Next intOdd  
27      For intEven As Integer = 0 To 10 Step 2  
28          Integer.TryParse(strIsbn(intEven), intDigit)  
29          intTotalEven += intDigit  
30      Next intEven  
31  
32
```

<- instead of: **strIsbn(intEven)**, you can use: **strIsbn.Substring(intEven, 1)**

7). **Step 3:**

- the third step in the algorithm adds together the sum from the **Step 1** and the **Step 2** in the algorithm

-> type the following assignment statement on line **32**:

```
32     intGrandTotal = intTotalOdd + intTotalEven  
33
```

#### 8). Step 4:

- the fourth step in the algorithm finds the remainder after dividing the total sum by 10

-> type the following assignment statement on line 33:

```
33     intRemainder = intGrandTotal Mod 10  
34  
35
```

#### 9). Step 5:

- the last step in the check digit algorithm uses the remainder calculated in the previous

**Step 4** to determine the check digit

<- as indicated earlier, if the remainder **is 0**, then the check digit is **0**

<- if the remainder is **not 0**, then the check digit is calculated by subtracting the remainder from the number **10**

-> enter the selection structure indicated:

```
34  
35     If intRemainder <> 0 Then  
36         intCheckDigit = 10 - intRemainder  
37     End If  
38
```

- 10). now that the check digit has been determined, you can append it to the end of the ISBN entered by the user and then display the final ISBN in the **lblFinalIsbn**

-> type the following assignment statement:

```
38     lblFinalIsbn.Text = strIsbn & intCheckDigit.ToString  
39
```

- 11). save the solution and then start the application:

-> first, you will enter an invalid number of characters: type **9781** and then click the button **Assign check digit**

<- the "**Please enter 12 numbers.**" message appears in a message box, so close the message box

-> change the text box entry to: **978133710212** and then click the button **Assign check digit**

<- the button's Click event procedure calculates the appropriate check digit - 4, appends it to the ISBN entered by the user, and then displays the final ISBN

-> on your own, display the final ISBNs for the following books: **978-1-285-86026**, **978-1-285-86019**, **978-1-305-87001**

<- the check digits should be: **8**, **0**, and **7**, respectively

- 12). when you are finished testing, click the button **Exit**, close the Code Editor window and then close the solution

#### 4. Check Digit Application code - completed

```
1     ' Name:      Check Digit Project  
2     ' Purpose:    Assign a check digit to an ISBN.  
3     ' Programmer: <your name> on <current date>  
4  
5     Option Explicit On  
6     Option Strict On  
7     Option Infer Off
```

```

8
9  Public Class frmMain
10     Private Sub btnAssign_Click(sender As Object, e As EventArgs) Handles btnAssign.Click
11         ' Assign a check digit to an ISBN.
12
13         Dim strIsbn As String
14         Dim intDigit As Integer
15         Dim intTotalOdd As Integer
16         Dim intTotalEven As Integer
17         Dim intGrandTotal As Integer
18         Dim intRemainder As Integer
19         Dim intCheckDigit As Integer
20
21         If txtIsbn.Text.Length = 12 Then
22             strIsbn = txtIsbn.Text
23             For intOdd As Integer = 1 To 11 Step 2
24                 Integer.TryParse(strIsbn(intOdd), intDigit)
25                 intTotalOdd += (intDigit * 3)
26             Next intOdd
27             For intEven As Integer = 0 To 10 Step 2
28                 Integer.TryParse(strIsbn(intEven), intDigit)
29                 intTotalEven += intDigit
30             Next intEven
31
32             intGrandTotal = intTotalOdd + intTotalEven
33             intRemainder = intGrandTotal Mod 10
34
35             If intRemainder <> 0 Then
36                 intCheckDigit = 10 - intRemainder
37             End If
38             lblFinalIsbn.Text = strIsbn & intCheckDigit.ToString
39
40         Else
41             MessageBox.Show("Please enter 12 numbers.", "Check Digit",
42                             MessageBoxButtons.OK, MessageBoxIcon.Information)
43         End If
44     End Sub
45
46     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
47         Me.Close()
48     End Sub
49
50     Private Sub txtIsbn_Enter(sender As Object, e As EventArgs) Handles txtIsbn.Enter
51         txtIsbn.SelectAll()

```

-> will store the 12 characters entered by a user in the (txtIsbn)

-> will be used by a TryParse methods that converts each character stored in the strIsbn to an integer

-> will store the sum from: Step 1 - odd indexes (even numbers) multiply by 3 and add the results

-> will store the sum from: Step 2 - every even index digit (odd number) add together

-> will store the sum from: Step 3 - add both sums together

-> will store the remainder calculated in: Step 4 - both sums Mod 10

-> will store the check digit calculated in: Step 5

-> if the user enters exactly 12 digits:

-> strIsbn variable contains the entry from (txtIsbn)

-> start loop to choose every odd index of the even numbers: 2, 4, 6, 8, 10, 12

-> intDigit = every odd index from strIsbn can be now used for the calculations

-> sum of all odd indexes (even numbers), what are multiplicated by 3

-> end loop for odd indexes (even numbers)

-> start loop to choose every even index of the odd number only: 1, 3, 5, 7, 9, 11

-> intDigit = every even index from strIsbn can be now used for the calculations

-> sum of all even indexes (odd numbers)

-> end loop for even indexes (odd numbers)

-> add both sums together

-> both sums added together Mod 10 (10 choosed by the coder)

-> if previous calculation isn't 0, then:

-> subtract the previous calculation from 10 (10 choosed by the coder)

(but if it was 0, then 0 would be the result)

-> lblFinalIsbn = original number entered by the user + calculated check digit

-> if the user enters less or more than 12 digits,

-> the message box appears

-> (btnExit) Click event procedure

-> will close the application

-> when the (txtIsbn) gets a focus -Tab key,

-> the whole previously entered text is selected

```

52     End Sub

53

54     Private Sub txtIsbn_TextChanged(sender As Object, e As EventArgs) Handles txtIsbn.TextChanged
55         lblFinalIsbn.Text = String.Empty
56     End Sub

57

58     Private Sub txtIsbn_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtIsbn.KeyPress
59         ' Allow only numbers and the Backspace key.

60

61         If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
62             e.Handled = True
63         End If
64     End Sub
65 End Class

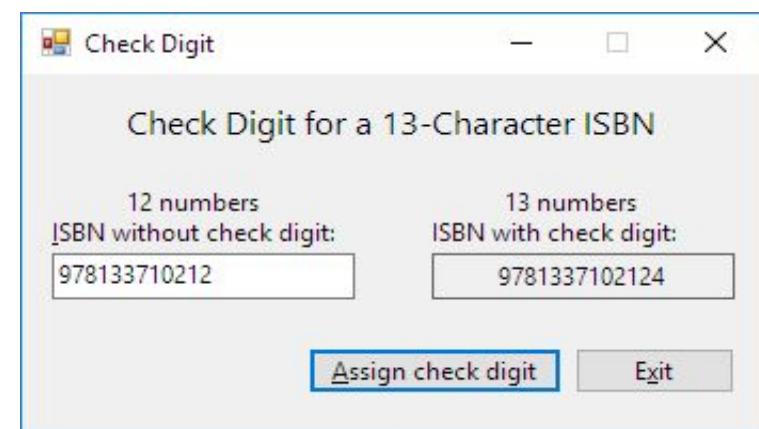
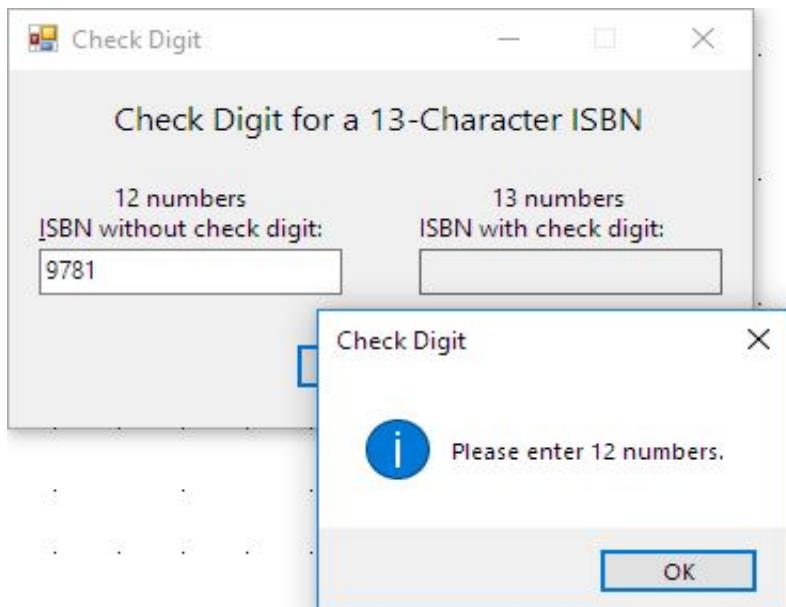
```

<- when the entry in the (txtIsbn) is changed,  
<- the (lblFinalIsbn) is emptied

<- key press event for (txtIsbn)

<- allows to press only 0 - 9, and BackSpace key  
<- otherwise it won't respond

## 5. Test the application



## CH7\_A2 - Code the Password Application - thoroughly step by step : (14.Password Solution)

- Figure 7-33 shows the interface for the **Password application**
- GUI provides a **text box (txtWords)** for the user to enter one or more words
- the button **Create password (btnCreate)** will create a password using **2 steps**:

**Step 1:** using the first letter from each of the words

**Step 2:** it will then insert a number immediately after the first character ->

-> the number will represent the length of the password before the number is inserted

1). open the: ...VB2017\Chap07\_Exercise\14.Password Solution\Password Solution.sln

2). open the Code Editor window and locate the **btnCreate\_Click** procedure on line **10**

- the procedure will use the **strWords** variable to store the user's input,

- and use the **strPassword** variable to store the password,

- and use the **intSpaceIndex** variable to store the indexes of the space characters that separate each word entered by the user

3). first, you will assign the contents of the **txtWords.Text** property, excluding any leading or trailing spaces, to the **strWords** variable:

-> click the blank line above the **If** clause, and type the following assignment statement on line **17** and then press Enter:

```
17 strWords = txtWords.Text.Trim
```

```
18
```

4). next, you will verify that the **strWords** variable is not empty:

-> change the **If** clause as follows:

```
19 If strWords <> String.Empty Then
```

```
19
```

5). the first character in the password should be the first character entered by the user

- you can access the first character using either: **strWords.Substring(0, 1)** or **strWords(0)**

-> click the blank line **21** below the first comment in the selection structure's true path and type the following assignment statement and then press Enter:

```
20 ' Assign the first character as the password:
```

```
21 strPassword = strWords(0)
```

```
22
```

6). now that the first character from the first word has been assigned to the password, you can move on to the next word in the **strWords** variable

- you do this by searching for the space character that separates the first word from the second word

-> click the blank line **24** above the **Do Until** clause, and type the following assignment statement and then press Enter:

- be sure to include a space character between the quotation marks

```
23 ' Search for the first space in the input:
```

```
24 intSpaceIndex = strWords.IndexOf(" ")
```

```
25
```

7). if the **strWords** variable contains only one word, the **IndexOf** method will return **-1** because it did not locate a space character

<- the search can stop at that point

- however, if it does not contain **-1**, the search for spaces should continue until it does

-> change the **Do Until** clause on line **26** as follows:

```
26 Do Until intSpaceIndex = -1
```

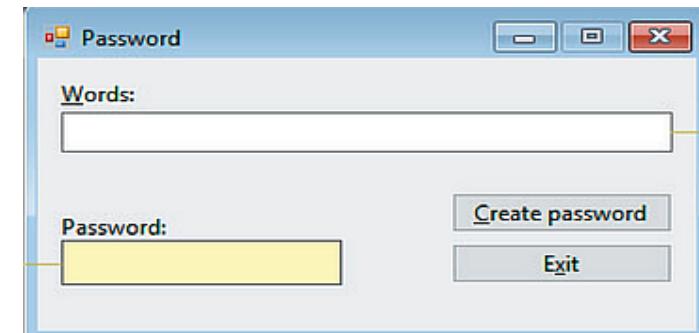


Figure 7-33 Password application's interface

- 8). if the **intSpaceIndex** variable contains a value other than **-1**, it means that a space was located  
 - in that case, you will concatenate the character that appears immediately after the space to the contents of the **strPassword** variable  
 -> click the blank line **29** and then type the assignment statement as follows:

```

 27      ' Concatenate the character that follows
 28      ' the space to the password:
 29      strPassword = strPassword & strWords(intSpaceIndex + 1)

```

- 9). now you can search for the next space character in the **strWords** variable  
 -> click the blank line **31** above the **Loop** clause, and type the assignment statement as follows:

```

 30      ' Search for the next space:
 31      intSpaceIndex = strWords.IndexOf(" ", intSpaceIndex + 1)
 32      Loop

```

- 10). when the loop ends, the **strPassword** variable will contain the first letter from each of the words entered by the user  
 - to create the final password, you just need to insert a number after the first character in the variable  
 - the number represents the length of the password before the number is inserted  
 - after inserting the number, the procedure can display the final password in the **lblPassword** control

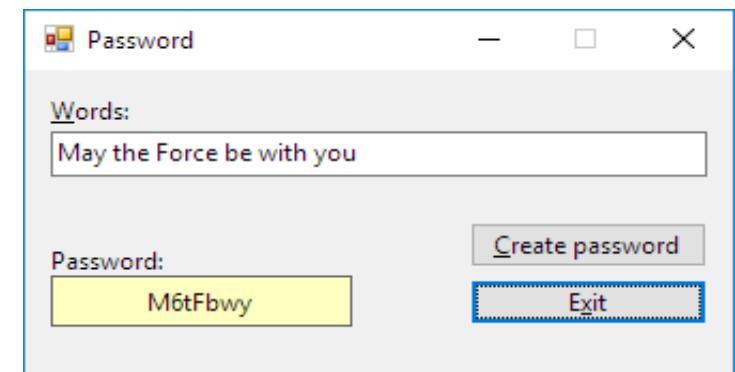
-> type the two additional assignment statements indicated on lines: **35** and **37**

```

 33
 34      ' Insert the number after the first character:
 35      strPassword = strPassword.Insert(1, strPassword.Length.ToString())
 36      ' Display the final password:
 37      lblPassword.Text = strPassword
 38      End If
 39      End Sub

```

- 11). save the solution and then start and test the application:  
 -> type: **May the Force be with you** and then click the button **Create password**  
 -< the **btnCreate\_Click** procedure displays the password  
 - the number **6** is the length of the password without the number



#### Password Application (14.Password Solution) code:

```

 1      ' Name:      Password Project
 2      ' Purpose:   Create a password.
 3      ' Programmer: <your name> on <current date>
 4
 5      Option Explicit On
 6      Option Strict On
 7      Option Infer Off
 8
 9      Public Class frmMain
10      Private Sub btnCreate_Click(sender As Object, e As EventArgs) Handles btnCreate.Click
11          ' Create a password.
12

```

```

13     Dim strWords As String
14     Dim strPassword As String
15     Dim intSpaceIndex As Integer
16
17     strWords = txtWords.Text.Trim
18
19     If strWords <> String.Empty Then
20         ' Assign the first character as the password:
21         strPassword = strWords(0)           <- 5). CH7_F6 - Character Array: using Character's index to access a Character in a String
22
23         ' Search for the first space in the input:
24         intSpaceIndex = strWords.IndexOf(" ")
25
26         Do Until intSpaceIndex = -1
27             ' Concatenate the character that follows
28             ' the space to the password:
29             strPassword = strPassword & strWords(intSpaceIndex + 1)
30             ' Search for the next space:
31             intSpaceIndex = strWords.IndexOf(" ", intSpaceIndex + 1)           <- 8). ->
32         Loop
33
34         ' Insert the number after the first character:
35         strPassword = strPassword.Insert(1, strPassword.Length.ToString)           <- 10). ->
36         ' Display the final password:
37         lblPassword.Text = strPassword
38     End If
39 End Sub
40
41 Private Sub txtWords_Enter(sender As Object, e As EventArgs) Handles txtWords.Enter
42     txtWords.SelectAll()
43 End Sub
44
45 Private Sub txtWords_TextChanged(sender As Object, e As EventArgs) Handles txtWords.TextChanged
46     lblPassword.Text = String.Empty
47 End Sub
48
49 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
50     Me.Close()
51 End Sub
52 End Class

```

<- 2).  
 <- 2).  
 <- 2).

<- 3).  
 <- 3).  
 <- 4).

<- 5).

<- 6). CH7\_F4.2 - String search: IndexOf method

<- 6).  
 <- 7).

<- 8). ->

<- 9). CH7\_F4.2 - String search: IndexOf method

<- 10). ->

<- 10).

## CH7\_A3 - Generate Random numbers - object **Random** & method **Next** for integers, or method **NextDouble** for doubles \_additional topic from Appendix B

- random numbers are used in many computer game programs
- the numbers can be **integers** or **real numbers**, which are numbers with a decimal place = **doubles**
- most programming languages provide a **pseudo-random number generator**, which is a mathematical algorithm that produces a sequence of numbers
  - although the numbers are not completely random, they are sufficiently random for practical purposes
- the **pseudo-random number generator** in Visual Basic is represented by an **object** whose data type is **Random**

1). you create an **object Random** to represent the pseudo-random number generator in your application's code

- you can do this by declaring a variable in a **Dim** statement **As New Random**

2). you can use the **object Random**'s methods:  
- a). **Next** to generate random **integer** number within the specified values  
- b). **NextDouble** to generate random **double** number within the specified values

1). create an object **Random** / make an instance of a **Class Random**

Random object's syntax:  
**As Random**

**Dim** *yourRandomObject* **As New Random**

**As Random**

2a). to generate random **Integer**, where: *minValue* <= *yourRandomObject* < *maxValue*

Random object's method **Next** syntax:  
**As Integer**

**int** *YourVariable* = *yourRandomObject*.**Next**(*minValue As Integer*, *maxValue As Integer*)

**As Integer**

2b1). to generate random **Double**, where: *0.0* <= *yourRandomObject* < *1.0*

Random object's method **NextDouble** syntax:  
**As Double**

**dbl** *YourVariable* = *yourRandomObject*.**NextDouble**

**As Double**

2b2). to generate random **Double**, where: *minValue* <= *yourRandomObject* < *maxValue*

Random object's method **NextDouble** syntax:  
**As Double**

**dbl** *YourVariable* = (*maxValue* - *minValue* + 1) \* *yourRandomObject*.**NextDouble** + *minValue*

**As Double**

**Dim/Static/Private**    - variable declaration rules learned in previous lessons apply, so:  
                          - **Dim** = procedure-level variable, declared in procedure, procedure scope...

*yourRandomObject*    = name of the **Random** object  
**As New**              = creates a **new object instance**  
**Random**              = an object, **Class System.Random**  
                          - represents a pseudo-random number **generator**, which is a device that produces  
                          a sequence of numbers that meet statistical requirements for randomness

|                     |                                                                                                                                                                                                                                                                                                             |
|---------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <i>YourVariable</i> | = previously created variable, whose data type depends on, which data type you wanna use<br>- since <i>yourRandomObject</i> is an instance of an object, you can use it only when assigned to a variable and in this case, you picked to use a function of the object ( <b>.Next</b> ; <b>.NextDouble</b> ) |
| <b>Next</b>         | <b>- As Integer</b><br>- a method, where: <i>minValue</i> $\leq$ <i>yourRandomObject</i> < <i>maxValue</i>                                                                                                                                                                                                  |
| <i>minValue</i>     | = an argument; the inclusive lower bound of the random number returned<br>- must be less than <i>maxValue</i>                                                                                                                                                                                               |
| <i>maxValue</i>     | <b>- As Integer</b><br>= an argument; the exclusive upper bound of the random number returned<br>- must be greater than or equal to <i>minValue</i>                                                                                                                                                         |
| <b>NextDouble</b>   | <b>- As Double</b><br>- a method, where: <b>0.0</b> $\leq$ <i>yourRandomObject</i> < <b>1.0</b>                                                                                                                                                                                                             |

e.g. 2a

```
Dim intNum As Integer
Dim randGen As New Random
intNum = randGen.Next(1, 51)
```

<- the **Dim** statement creates a **Random** object named **randGen**  
 <- the **randGen.Next(1, 51)** expression generates a random integer from **1** through **50**  
 <- the assignment statement assigns the random integer to the **intNum** variable

e.g. 2a

```
Dim intNum As Integer
Dim randGen As New Random
intNum = randGen.Next(-10, 20)
```

<- the **Dim** statement creates a **Random** object named **randGen**  
 <- the **randGen.Next(-10, 20)** expression generates a random integer from **-10** through **19**  
 <- the assignment statement assigns the random integer to the **intNum** variable

e.g. 2b1

```
Dim dblNum As Double
Dim randGen As New Random
dblNum = randGen.NextDouble
```

<- the **Dim** statement creates a **Random** object named **randGen**  
 <- the **randGen.NextDouble** expression generates a random double from **0.0**, but less than **1.0**  
 <- the assignment statement assigns the random double to the **dblNum** variable

e.g. 2b2

```
Dim dblNum As Double
Dim randGen As New Random
dblNum = (25 - 5 + 1) * randGen.NextDouble + 5
```

<- the **Dim** statement creates a **Random** object named **randGen**  
 <- the expression generates a random double number from **5.0**, but less than **25.0**  
 <- the assignment statement assigns the random double to the **dblNum** variable

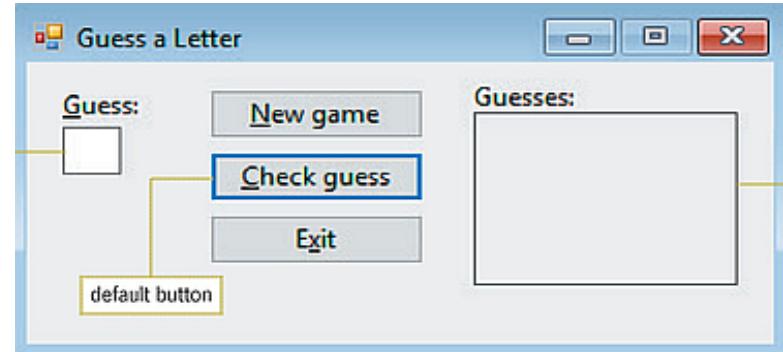
### Mini-Quiz 7-6:

1. Write a **Dim** statement that creates a **Random** object named **randGen**.
2. Using the **Random** object created in Question 1, write a statement that assigns a random integer from **25** to **100 including** to the **intNum** variable.
3. Using the **Random** object created in Question 1, write a statement that assigns a random integer from **2** to **25 excluding** to the **intNum** variable.

```
1... Dim randgen As New Random  
2... intNum = randgen.Next(25, 101)  
3... intNum = randgen.Next(2, 25)
```

### CH7\_A3.1 - Generate Random Integers example: code the Guess a Letter Application (15.Letter Guess Solution) - part 1/3

- you will use random numbers in the **Guess a Letter** application
- the **btnNewGame\_Click** procedure generates a random number and then uses the number to select a letter from the **strALPHABET** constant, which contains the uppercase letters of the alphabet
- the user enters a guess in the **txtGuess** control and then clicks the **Check guess** button
- the **btnCheck\_Click** procedure records the guess in the **lblGuesses** control -> doing this allows the user to view the previous guesses
- the procedure then displays a message indicating whether or not the user's guess is correct



pseudocode for: **btnNewGame\_Click** -> the **New game** button

1. declare **strALPHABET** constant and initialize it to the uppercase letters of the alphabet
2. declare **randGen** variable to store a **Random** object, and declare **intRandNum** variable to store a random number
3. generate a random number and use it to select a letter from **strALPHABET** -> assign the letter to a class-level variable named **strRandLetter**
4. prepare the interface for a new game by clearing **lblGuesses.Text**, enabling **btnCheck**, and sending the focus to **txtGuess**

pseudocode for: **btnCheck\_Click** -> the **Check guess** button

1. declare **strGuess** variable to store user's guess
2. assign **txtGuess.Text**, in uppercase and excluding any leading or trailing spaces, to **strGuess**
3. concatenate **strGuess** with current contents of **lblGuesses**
4. if the letter stored in **strGuess** is the same as the letter stored in class-level **strRandLetter**:
  - display "You guessed the correct letter:" message along with the letter
  - disable **btnCheck**
- else
  - display "Guess again!" message
- end if
5. clear **txtGuess.Text** and send focus to **txtGuess**

1). open the: ...VB2017\Chap07\Exercise\15.Letter Guess Solution\Letter Guess Solution.sln

<- notice that the **btnCheck** button is the default button for **Enter** key in users keyboard

2). open the Code Editor window:

- first, you will create a class-level variable to store the random letter

- a class-level variable is appropriate in this case because the variable will need to be used by two procedures:

- 1st procedure: **btnNewGame\_Click**
- 2nd procedure: **btnCheck\_Click**

-> click the blank line below line **10** and type the following declaration statement and then press Enter:

```
10      ' Class-level variable:  
11      Private strRandLetter As String  
12
```

<- class-level variable will be used by 2 procedures: - **btnNewGame\_Click**  
- **btnCheck\_Click**

### 3). locate the **btnNewGame\_Click** procedure on line **13**

- the procedure on line **16** declares a **String** constant named **strALPHABET** and initializes it to the **26 uppercase letters** of the alphabet

- the procedure will also need two variables: - one to store a **Random** object: **randGen**  
- second to store a random number: **intRandNum**

-> click the blank line below the **Constant** statement and then enter the following two **Dim** statements:

```
Step 1: 16      Const strALPHABET As String = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"  
Step 2: 17      Dim randGen As New Random  
Step 2: 18      Dim intRandNum As Integer
```

<- 26 characters  
<- variable will store a **Random** object  
<- variable will store a random number

### 4). now the procedure can:

- generate the random number

- and then use it to select a letter from the **strALPHABET** constant, using the character's index

**CH7\_A3 - Generate Random Integers - object Random, method Next**

**CH7\_F6 - Character Array: using Character's index to access a Character in a String**

- the letters contained in the constant have indexes of **0** through **25**

- so in the **Random.Next** method you will need to use: - **0** as the **minValue**  
- **26** as the **maxValue**

<- minValue is inclusive  
<- maxValue is exclusive

-> click the blank line below the second comment in the procedure and then enter the following two assignment statements:

```
Step 3: 19      ' Generate a random number and use it to select a letter:  
Step 3: 20      intRandNum = randGen.Next(0, 26)  
Step 3: 21      strRandLetter = strALPHABET(intRandNum)
```

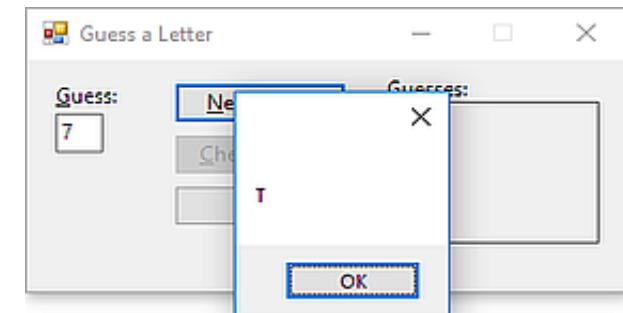
<- **CH7\_A3 - Generate Random Integers - object Random, method Next**

<- **CH7\_F6 - Character Array: using Character's index to access a Character in a String**

### 5). before finishing the procedure's code, you will observe how the code you entered so far works:

-> type the following statement but do not press Enter:

```
22      MessageBox.Show(strRandLetter)
```



result of the: **MessageBox.Show(strRandLetter)**

### 6). save the solution and then start the application

- the **Check guess** button appears dimmed-grayed out because its **Enabled** property is set to **False** in the **Properties** window

<- you will learn about the **Enabled** property in the next section:

**CH7\_A4 - use the Enabled property of the Control:**

### 7). -> click the **New game** button

<- the random letter selected by the **btnNewGame\_Click** procedure appears in a message box

-> close the message box and then click the **New game** button again

<- a different letter appears in the message box

- if the same letter appears, close the message box and then click the **New game** button again

-> close the message box and then click the **Exit** button

8). change the statement to comment

22

MessageBox.Show(strRandLetter)

statement from the procedure and then save the solution

#### CH7\_A4 - use the **Enabled** property of the Control:

- a control's **Enabled** property, which can be set to either **True** or **False**, determines whether the control will respond to the user
- when the property is set to **False**, the control appears dimmed (grayed out) during run time, indicating that it is not available for use
- you can set the **Enabled** property either:
  - in the **Properties** window,
  - or in a procedure's code

control's property **Enabled** syntax:

As Boolean

**object.Enabled = True/False As Boolean**

**object** = (name) of your control

**Enabled** - As Boolean

- property; determines whether the control will respond to the user during run time

e.g.:

btnShow.Enabled = False

<- the control (btnShow) appears dimmed (grayed out) during run time and can't be used

#### CH7\_A5 - use the **Focus** method of the Control:

- you can use the **Focus method** to move the focus to a control during run time

control's method **Focus** syntax:

As Boolean

**object.Focus() As Boolean**

**object** = (name) of the object to which you want the focus sent

**Focus** - method; send the focus during the run time

e.g.:

txtGuess.Focus()

<- the method **Focus** sends the focus during run time to the control (txtGuess)

#### CH7\_A6 - Generate Random Integers example: code the **Guess a Letter Application** (15.Letter Guess Solution) - part 2/3

- to finish coding the **btnNewGame\_Click** procedure:

9). click the blank line above the procedure's **End Sub** clause on line 26 and then enter the additional statements indicated:

step 4:

```
24      ' Clear lblGuesses, enable btnCheck, send focus to txtGuess:  
25      lblGuesses.Text = String.Empty  
26      btnCheck.Enabled = True  
27      txtGuess.Focus()  
28  End Sub
```

<- CH7\_A4 - use the **Enabled** property of the Control:

<- CH7\_A5 - use the **Focus** method of the Control:

10). in the next set of steps, you will code the **btnCheck\_Click** procedure, which pseudocode is shown earlier

-> locate the **btnCheck\_Click** procedure on line 30

- the first two steps in the pseudocode have already been coded:

**step 1:** - the procedure declares a String variable named **strGuess**

```
33    Dim strGuess As String
```

**step 2:** - and then assigns the **txtGuess.Text** property - in uppercase and without any leading or trailing spaces - to the variable

```
35        strGuess = txtGuess.Text.Trim.ToUpper
```

**step 3:** - the third step in the pseudocode concatenates the user's guess with the current contents of the **lblGuesses** control

-> click the blank line below the line 36 with a comment, enter the assignment statement indicated and then press Enter

- be sure to type a space character between the quotation marks

```
36    ' Display guess in lblGuesses:
```

```
37        lblGuesses.Text = lblGuesses.Text & " " & strGuess
```

```
38
```

**step 4:** - the fourth step in the pseudocode is a selection structure whose condition determines whether the user guessed the random letter

- both paths in the structure then display an appropriate message

-> first, delete the apostrophe that appears before the: **If**, **Else**, and **End If** clauses and then change the **If** clause as follows:

```
39    If strGuess = strRandLetter Then
```

- after displaying the appropriate message, the selection structure's true path should disable the **btnCheck** control because there is no reason to continue checking when the user guessed the random letter

-> click the blank line 43 above the **Else** clause and type the following assignment statement and then press Enter:

```
43        btnCheck.Enabled = False
```

```
44
```

**step 5:** - the last step in the pseudocode clears the **txtGuess.Text** property and then sends the focus to the **txtGuess** control

-> click the blank line 49 above the **End Sub** clause and then enter the two additional statements indicated:

```
48    End If
```

```
49        txtGuess.Text = String.Empty
```

```
50        txtGuess.Focus()
```

```
51    End Sub
```

<- CH7\_A5 - use the **Focus** method of the Control:

11). save the solution and then start and test the application:

- > click the **New game** button
- > and type a letter in the **Guess** box and then press Enter
- the **btnCheck\_Click** procedure displays an appropriate message
- > close the message box
- the letter you guessed appears in the **lblGuesses** control
- > if you did not guess the correct letter, keep guessing until you do

**Figure 1** <- notice that the **Check guess** button is dimmed (shaded-out)

<- notice that the **Check guess** button is **not** dimmed (shaded-out) any more

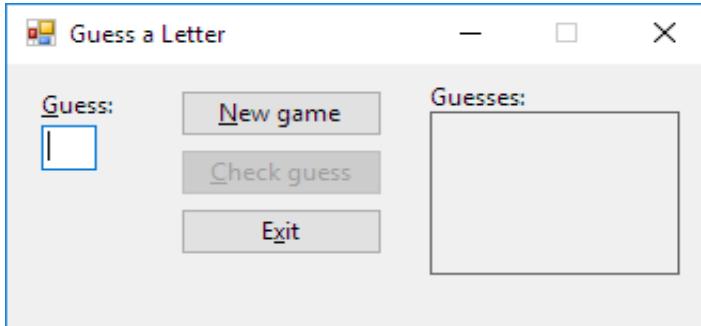
<- recall that the **Check guess** button is the default button

**Figure 2**

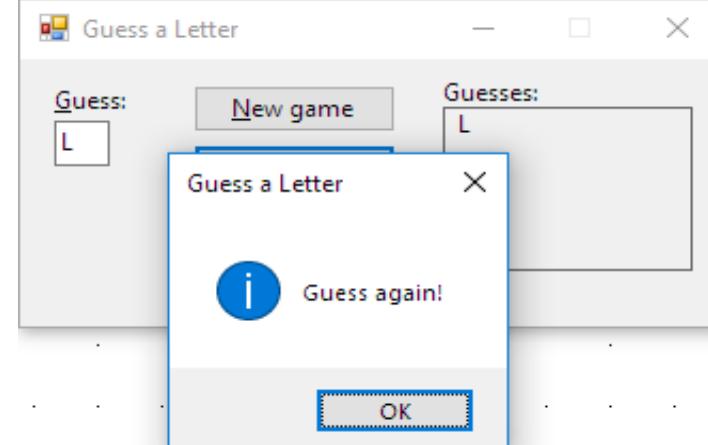
**Figure 3**

**Figure 4** <- notice that the **Check guess** button is dimmed when the user's guess is correct

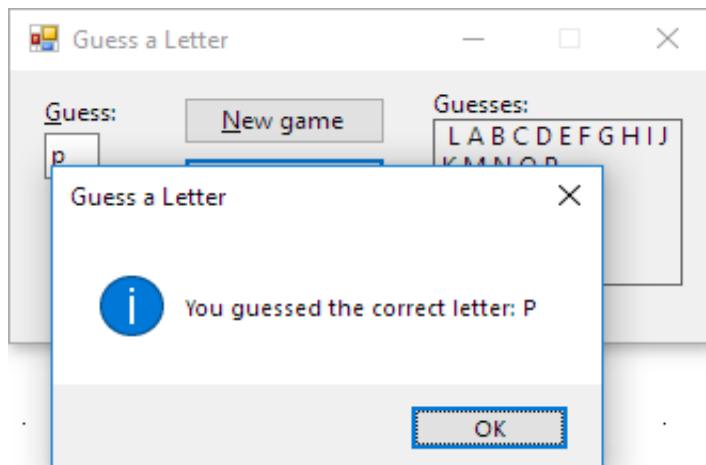
**Figure 1**



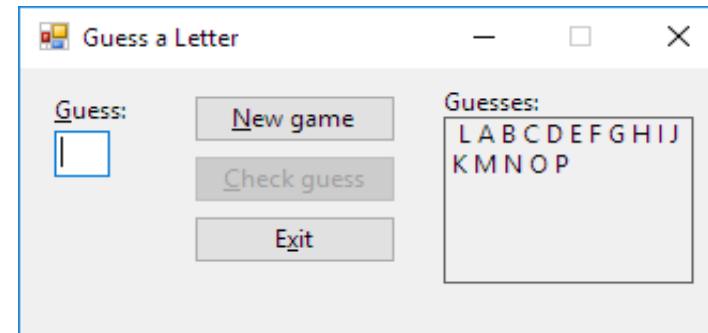
**Figure 2**



**Figure 3**



**Figure 4**



```
1  ' Name:      Letter Guess Project
2  ' Purpose:    Guess a random letter.
3  ' Programmer: <your name> on <current date>
4
5  Option Explicit On
6  Option Strict On
7  Option Infer Off
8
9  Public Class frmMain
10   ' Class-level variable.
11   Private strRandLetter As String
12
13  Private Sub btnNewGame_Click(sender As Object, e As EventArgs) Handles btnNewGame.Click
14   ' Select a random letter and prepare for a new game.
15
16   Const strALPHABET As String = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
17   Dim randGen As New Random
18   Dim intRandNum As Integer
19
20   ' Generate a random number and use it to select a letter:
21   intRandNum = randGen.Next(0, 26)
22   strRandLetter = strALPHABET(intRandNum)
23   'MessageBox.Show(strRandLetter)
24   ' Clear lblGuesses, enable btnCheck, send focus to txtGuess.
25   lblGuesses.Text = String.Empty
26   btnCheck.Enabled = True
27   txtGuess.Focus()
28 End Sub
29
30  Private Sub btnCheck_Click(sender As Object, e As EventArgs) Handles btnCheck.Click
31   ' Determine whether the user guessed the random letter.
32
33   Dim strGuess As String
34
35   strGuess = txtGuess.Text.Trim.ToUpper
36   ' Display guess in lblGuesses:
37   lblGuesses.Text = lblGuesses.Text & " " & strGuess
38
39   If strGuess = strRandLetter Then
40     MessageBox.Show("You guessed the correct letter: " & strGuess,
41                   "Guess a Letter", MessageBoxButtons.OK,
42                   MessageBoxIcon.Information)
```

```

43     btnCheck.Enabled = False
44
45     Else
46         MessageBox.Show("Guess again!", "Guess a Letter",
47                         MessageBoxButtons.OK, MessageBoxIcon.Information)
48     End If
49     txtGuess.Text = String.Empty
50     txtGuess.Focus()
51 End Sub
52
53 Private Sub txtGuess_Enter(sender As Object, e As EventArgs) Handles txtGuess.Enter
54     txtGuess.SelectAll()
55 End Sub
56
57 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
58     Me.Close()
59 End Sub
60 End Class
61

```

### Mini-Quiz 7-7:

1. Write a statement that prevents the **btnCalc** control from responding to the user.
2. Write a statement that reverses the statement from Question 1.
3. Write a statement that moves the insertion point the **txtName** control.

```

1...btnCalc.Enabled = False
2...btnCalc.Enabled = True
3...txtName.Focus()

```

### CH7\_A8 - many of the concepts from CH7 example: code the Guess the Word Game Application (16.Word Guess Solution) + entire code

- in this section, you will use many of the concepts you learned in the Focus lesson to code the **Guess the Word Game** application
- the game requires two players:
  - first, player 1 will enter a **five-letter** word in the **txtWord** control and then click the **New word** button
    - the **btnNewWord\_Click** procedure will display **five hyphens** in the **lblResult** control
    - each hyphen represents a letter in the word
  - next, player 2 will enter a letter in the **txtLetter** control and then click the **Try this letter** button
    - the **btnTryLetter\_Click** procedure will determine whether the letter appears in player 1's word
    - if it does, the procedure will replace the hyphen in the **lblResult** control with the letter
    - when all of the hyphens have been replaced, it means that the user guessed the word
    - at that point, the **btnTryLetter\_Click** procedure will display an appropriate message

**step #1:** set some **Properties** of the controls in the **Designer window**

**step #2:** code the **btnNewWord\_Click** procedure -> the button **New word**

**step #3:** code the **btnTryLetter\_Click** procedure -> the button **Try this letter**

## step #1: set some **Properties** of the controls in the **Designer window**

- in the next set of steps, you will set:
  - the **grpLetter** control's **Enabled** property to **False**  
    <- doing this will disable all of the controls contained in the group box during run time
  - each text box's **MaxLength** property  
    <- which specifies the maximum number of characters the text box will accept
  - the **txtWord** control's **PasswordChar** property  
    <- typically used for text boxes that contain passwords  
    <- hides the user's entry by displaying a replacement character (such as an asterisk) in place of the character the user entered
  - in this case, you will use the property to hide player 1's word from player 2

1). open the: ...VB2017\Chap07\\_Exercise\16.Word Guess Solution\Word Guess Solution.sln

2). open the **Designer window**

<- notice that the **AcceptButton = btnTryLetter** = that the **Try this letter** button is the default button in the interface

3). -> click/choose the **grpLetter** control and set its property **Enabled = False**

-> click/choose the **txtLetter** control and set its property **MaxLength = 1**

-> click/choose the **txtWord** control and then:

-> set its property **MaxLength = 5**

-> set its property **PasswordChar = \*** (an asterisk)

4). save the solution



## step #2: code the **btnNewWord\_Click** procedure -> the button New word

pseudocode: if **txtWord.Text** contains five letters  
    disable **grpWord**  
    enable **grpLetter**  
    display five hyphens in **lblResult**  
    send focus to **txtLetter**  
else  
    display "Please enter 5 letters." message  
end if

1). open the Code Editor window and locate the **btnNewWord\_Click** procedure on line 10

- according to its pseudocode, the procedure will use a selection structure to verify that the **txtWord.Text** property contains five letters
- if it does contain five letters, the structure's **true** path will perform four tasks;
- otherwise, its **false** path will display an appropriate message

<- the **false** path has already been coded for you

-> change the **If** clause, and also enter the four statements in the true path:

```
10      Private Sub btnNewWord_Click(sender As Object, e As EventArgs) Handles btnNewWord.Click
11          ' Determine whether the word contains five letters:
12
13          If txtWord.Text.Trim.ToUpper Like "[A-Z][A-Z][A-Z][A-Z][A-Z]" Then
14              grpWord.Enabled = False
```

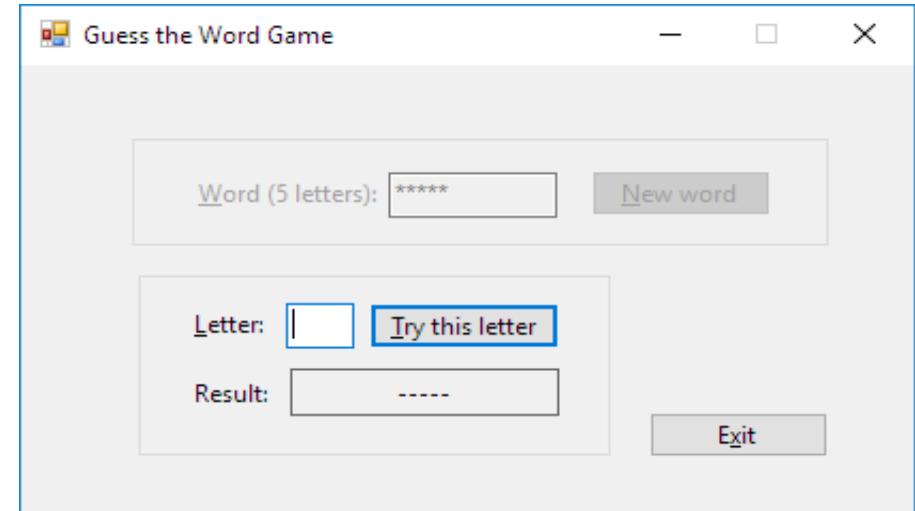
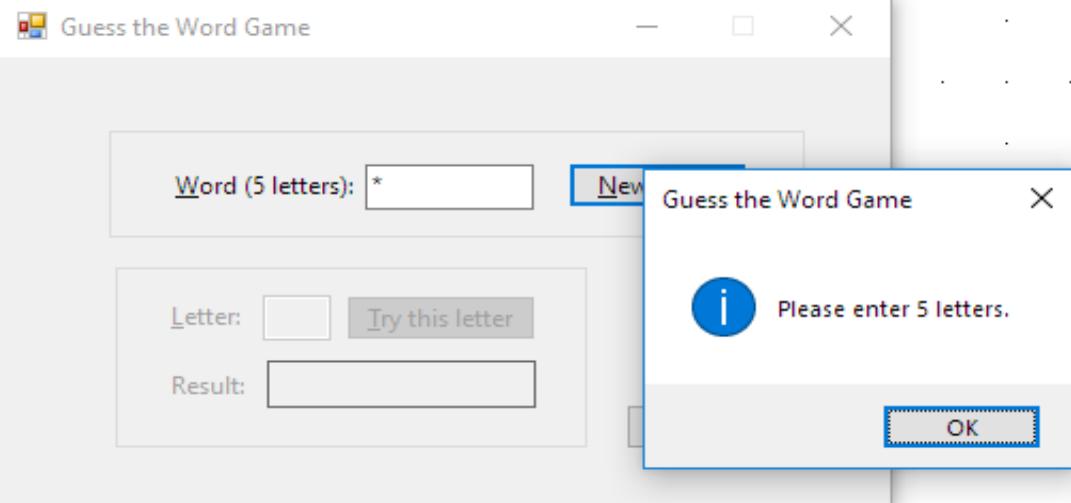
```

15     grpLetter.Enabled = True
16     lblResult.Text = "-----"
17     txtLetter.Focus()
18 Else
19     MessageBox.Show("Please enter 5 letters.", "Guess the Word Game",
20                         MessageBoxButtons.OK, MessageBoxIcon.Information)
21 End If
22 End Sub

```

2). save the solution and then start and test the application:

- > in the Word (5 letters) box type: **b** <- notice that the letter **b** is replaced by an \* asterisk
- > click the button New word <- the **btnNewWord\_Click** procedure displays the message "Please enter 5 letters"
- > close the message box



- > change the entry in the Word (5 letters) box to the word: **basic** and then click the button New word
- <- the **btnNewWord\_Click** procedure performs the four tasks listed in the selection structure's **true** path:

1. the Group box **grpWord** is disabled
2. the Group box **grpLetter** is enabled
3. the label box **lblResult** contains five hyphens
4. the text box **txtLetter** has a focus

```

<- 14             grpWord.Enabled = False
<- 15             grpLetter.Enabled = True
<- 16             lblResult.Text = "-----"
<- 17             txtLetter.Focus()

```

3). click the button **Exit**

### step #3: code the **btnTryLetter\_Click** procedure -> the button **Try this letter**

pseudocode:

**Step 1:** declare **strWord**, **strLetter**, and **strResult** variables  
**Step 2:** assign **txtWord.Text**, in uppercase and excluding any leading or trailing spaces, to **strWord**  
**Step 3:** assign **txtLetter.Text**, in uppercase and excluding any leading or trailing spaces, to **strLetter**  
**Step 4:** assign **lblResult.Text** to **strResult**  
**Step 5:** if **strWord** contains the letter stored in **strLetter**  
    repeat the following for each letter in **strWord**  
        if the current letter in **strWord** is the same as the letter in **strLetter**  
            remove the hyphen from **strResult**  
            insert the letter in **strResult**  
        end if  
    end repeat  
  
    display the contents of **strResult** in **lblResult**  
  
    if **strResult** does not contain any hyphens  
        display "You guessed it:" message along with **strWord**  
        enable **grpWord**  
        disable **grpLetter**  
        clear **lblResult.Text**  
        send focus to **txtWord**  
    end if  
  
else  
    display "Try again!" message  
end if  
  
**Step 6:** clear **txtLetter.Text**

#### 1). locate the **btnTryLetter\_Click** procedure on line 24

- the first four steps in the pseudocode have already been coded for you:

|                |                                                   |                                                                                                                                                                                                                                                    |
|----------------|---------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Step 1:</b> | 27        Dim strWord As String                   | <- contains the word entered by player 1                                                                                                                                                                                                           |
| <b>Step 1:</b> | 28        Dim strLetter As String                 | <- contains the letter entered by player 2                                                                                                                                                                                                         |
| <b>Step 1:</b> | 29        Dim strResult As String                 | <- contains the current contents of the <b>lblResult.Text</b> property                                                                                                                                                                             |
| <b>Step 2:</b> | 30                                                |                                                                                                                                                                                                                                                    |
| <b>Step 3:</b> | 31        strWord = txtWord.Text.Trim.ToUpper     | <- assign <b>txtWord.Text</b> , in uppercase and excluding any leading or trailing spaces, to <b>strWord</b>                                                                                                                                       |
| <b>Step 4:</b> | 32        strLetter = txtLetter.Text.Trim.ToUpper | <- assign <b>txtLetter.Text</b> , in uppercase and excl. any leading or trailing spaces, to <b>strLetter</b>                                                                                                                                       |
|                | 33        strResult = lblResult.Text              | <- assign <b>lblResult.Text</b> to <b>strResult</b><br><- the first time the procedure is processed, the <b>strResult</b> variable will contain the five hyphens assigned to the <b>lblResult</b> control by the <b>btnNewWord_Click</b> procedure |

2). -> delete the ' (apostrophe) from the beginning of the **If**, **Else**, and **End If** clauses on lines **35**, **42**, and **45**

### Step 5:

3). according to Step 5 in the pseudocode, the procedure will use a selection structure to determine whether player 1's word contains player 2's letter  
-> change the **If** clause on line **35** as follows:

|                                                      |                                                                                                              |
|------------------------------------------------------|--------------------------------------------------------------------------------------------------------------|
| 35 <b>If</b> strWord.Contains(strLetter) <b>Then</b> | <- Contains method returns <b>True</b> when the <b>strLetter</b> is contained anywhere in the <b>strWord</b> |
|                                                      | <- CH7_F4 - Search for a specific sequence of characters: <b>Contains</b> and <b>IndexOf</b> methods ...     |
|                                                      | <- CH7_F4.1 - String search: <b>Contains</b> method                                                          |
|                                                      | <- CH7_F4.4 - You Do It 2: <b>Contains</b> meth. and <b>IndexOf</b> meth. exercise: 05.You Do It 2 Solution  |

4). if the letter is not in the word, the selection structure's **False** path will display the "Try again!" message, what is already entered in the procedure:

|                                                                |                                                                                                    |
|----------------------------------------------------------------|----------------------------------------------------------------------------------------------------|
| 42 <b>Else</b>                                                 | <- CH6_A10 - Professionalize Your Application's Interface:<br>Message Box (MessageBox.Show method) |
| 43 <b>MessageBox.Show("Try again!", "Guess the Word Game",</b> |                                                                                                    |
| 44 <b>MessageBoxButtons.OK, MessageBoxIcon.Information)</b>    |                                                                                                    |
| 45 <b>End If</b>                                               |                                                                                                    |

5). on the other hand, if the letter appears in the word, the selection structure's **True** path will use a loop to look at each character in the word:

-> type the following **For** clause on line **37**, press enter and change the automatic **Next** clause on line **39** to: **Next intIndex**

|                                                                          |  |
|--------------------------------------------------------------------------|--|
| 35 <b>If</b> strWord.Contains(strLetter) <b>Then</b>                     |  |
| 36            ' Replace the hyphen(s) in strResult:                      |  |
| 37 <b>For</b> intIndex <b>As Integer</b> = 0 <b>To</b> strWord.Length -1 |  |
| 38                                                                       |  |
| 39 <b>Next</b> intIndex                                                  |  |

6). the loop body will use a selection structure to determine whether the current character in the word matches player 2's letter

- if it does, the structure's **True** path will remove the hyphen from the **strResult** variable and then insert the letter in its place

-> enter the nested selection structure on line **38**:

|                                                                          |                                                                                                      |
|--------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------|
| 36            ' Replace the hyphen(s) in strResult:                      |                                                                                                      |
| 37 <b>For</b> intIndex <b>As Integer</b> = 0 <b>To</b> strWord.Length -1 |                                                                                                      |
| 38 <b>If</b> strWord(intIndex) = strLetter <b>Then</b>                   | <- CH7_F6 - Character Array: using <b>Character's index</b> to access a <b>Character</b> in a String |
| 39 <b>strResult</b> = strResult.Remove(intIndex, 1)                      | <- CH7_F7 - Remove method of the String:                                                             |
| 40 <b>strResult</b> = strResult.Insert(intIndex, strLetter)              | <- CH7_F2 - Insert method of the String: <b>string.Insert(startIndex, value)</b>                     |
| 41 <b>End If</b>                                                         |                                                                                                      |
| 42 <b>Next</b> intIndex                                                  |                                                                                                      |

7). after the loop has finished processing, the outer selection structure's **True** path will display the contents of the **strResult** variable in the **lblResult** control

-> click the blank line **44** below the comment: '**Display the contents of strResult**', type the following assignment statement and then press enter:

|                                                    |
|----------------------------------------------------|
| 43            ' Display the contents of strResult: |
| 44 <b>lblResult.Text = strResult</b>               |
| 45                                                 |

8). next, the outer selection structure's **True** path needs to determine whether the **strResult** variable contains any hyphens

-> click the blank line **47** below the comment: '**Determine whether strResult contains any hyphens:**' on line **46**, and type the following nested **If** clause and then press enter:

```
46      ' Determine whether strResult contains any hyphens:  
47      If strResult.Contains("-") = False Then  
48  
49          End If  
50      Else
```

<- CH7\_F4.1 - String search: Contains method

9). if there are no more hyphens in the variable, it means that player 2 guessed all of the letters in player 1's word

- in that case, the nested selection structure should display the message "**You guessed it:**" along with the word
- it should also:
  - enable the **grpWord** control,
  - disable the **grpLetter** control,
  - clear the **lblResult.Text** property,
  - and send focus to the **txtWord** control

-> enter the five statements indicated:

```
46      ' Determine whether strResult contains any hyphens:  
47      If strResult.Contains("-") = False Then  
48          MessageBox.Show("You guessed it: " & strWord,  
49              "Guess the Word Game",  
50              MessageBoxButtons.OK, MessageBoxIcon.Information)  
51          grpWord.Enabled = True  
52          grpLetter.Enabled = False  
53          lblResult.Text = String.Empty  
54          txtWord.Focus()  
55      End If
```

<- CH6\_A10 - Professionalize Your Application's Interface:  
Message Box (MessageBox.Show method)

## Step 6:

10). the last step in the pseudocode clears the **txtLetter.Text** property

- the code for this step has already been entered above the procedure's **End Sub** clause on line **61**:

```
60      txtLetter.Text = String.Empty  
61  End Sub
```

11). save the solution and then start and test the application:

-> type: **apple** and then click the button **New word**

-> type: **e** and then press Enter

<- the letter **E** appears after the four hyphens in the **Result** box

-> type: **k** and press Enter

<- the message "**Try again!**" appears

-> close the message box, type: **p** and press Enter

<- the **Result** box now shows: **-PP-E**

-> type: **a** and press Enter, then type **I** and press Enter

<- the message "**You guessed it: APPLE**" appears

-> close the message box, and close the Solution

<- recall that the button **Try this letter** is the default button

## Guess the Word Game Application (16.Word Guess Solution) code:

```
1      ' Name:          Word Guess Project
2      ' Purpose:       Guess the word entered by player 1.
3      ' Programmer:    <your name> on <current date>
4
5      Option Explicit On
6      Option Strict On
7      Option Infer Off
8
9      Public Class frmMain
10     Private Sub btnNewWord_Click(sender As Object, e As EventArgs) Handles btnNewWord.Click
11         ' Determine whether the word contains five letters:
12
13         If txtWord.Text.Trim.ToUpper Like "[A-Z][A-Z][A-Z][A-Z][A-Z]" Then
14             grpWord.Enabled = False
15             grpLetter.Enabled = True
16             lblResult.Text = "-----"
17             txtLetter.Focus()
18         Else
19             MessageBox.Show("Please enter 5 letters.", "Guess the Word Game",
20                             MessageBoxButtons.OK, MessageBoxIcon.Information)
21         End If
22     End Sub
23
24     Private Sub btnTryLetter_Click(sender As Object, e As EventArgs) Handles btnTryLetter.Click
25         ' Determine whether player 2 has guessed the word.
26
27         Dim strWord As String
28         Dim strLetter As String
29         Dim strResult As String
30
31         strWord = txtWord.Text.Trim.ToUpper
32         strLetter = txtLetter.Text.Trim.ToUpper
33         strResult = lblResult.Text
34
35         If strWord.Contains(strLetter) Then
36             ' Replace the hyphen(s) in strResult:
37             For intIndex As Integer = 0 To strWord.Length - 1
38                 If strWord(intIndex) = strLetter Then
39                     strResult = strResult.Remove(intIndex, 1)
40                     strResult = strResult.Insert(intIndex, strLetter)
41                 End If
42             Next intIndex
```

```

43     ' Display the contents of strResult:
44     lblResult.Text = strResult
45
46     ' Determine whether strResult contains any hyphens:
47     If strResult.Contains("-") = False Then
48         MessageBox.Show("You guessed it: " & strWord,
49                         "Guess the Word Game",
50                         MessageBoxButtons.OK, MessageBoxIcon.Information)
51         grpWord.Enabled = True
52         grpLetter.Enabled = False
53         lblResult.Text = String.Empty
54         txtWord.Focus()
55     End If
56     Else
57         MessageBox.Show("Try again!", "Guess the Word Game",
58                         MessageBoxButtons.OK, MessageBoxIcon.Information)
59     End If
60     txtLetter.Text = String.Empty
61 End Sub
62
63 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
64     Me.Close()
65 End Sub
66
67 Private Sub txtWord_Enter(sender As Object, e As EventArgs) Handles txtWord.Enter
68     txtWord.SelectAll()
69 End Sub
70
71 Private Sub txtLetter_Enter(sender As Object, e As EventArgs) Handles txtLetter.Enter
72     txtLetter.SelectAll()
73 End Sub
74 End Class

```

### CH7\_Summary: Check digits, Random integers, Enable property & Focus method of the Control, Concepts used for String manipulations

1. Check digits are used for validating numbers, such as credit card numbers, bank account numbers, product UPCs, and book ISBNs

**CH7\_A1 - Code the Check Digit Application - thoroughly step by step : (13.Check Digit Solution)**

& **CH7\_A2 - Code the Password Application -**

- thoroughly step by step : (14.Password Solution)

- 2a. to generate random integers, you first create a Random object to represent the pseudo-random number generator

- 2b. you then use the object's Random.Next method to generate a random integer

- 2c. the method returns a number that is greater than or equal to its minValue argument but less than its maxValue argument

- 2d. refer to the syntax and examples shown earlier in Figure 7-39

**CH7\_A3 - Generate Random Integers - object Random, method Next**

&

**CH7\_A3.1 - Generate Random Integers** example: code the **Guess a Letter Application** (15.Letter Guess Solution) - part 1/3

&

**CH7\_A6 - ... (15.Letter Guess Solution) - part 2/3** & **CH7\_A7 - ... (15.Letter Guess Solution) - entire code - part 3/3**

3a. you disable a control by setting its Enabled property to False either in the Properties window or in a procedure's code

3b. you enable it by setting its Enabled property to True

CH7\_A4 - use the **Enabled** property of the Control:

4. you can send the focus to a control using the control's Focus method

CH7\_A5 - use the **Focus** method of the Control:

5a. a text box's MaxLength property specifies the maximum number of characters that the text box will accept

5b. a text box's PasswordChar property specifies the character to use in place of each character entered in the text box

CH7\_A8 - many of the concepts from CH7 example: code the **Guess the Word Game Application** (16.Word Guess Solution) + entire code

6. summary of the concepts for **String manipulations** covered in this chapter's Focus lesson:

Concept name:

Syntax: , optional

Dim strName As String = "Martin"

More info:

- purpose:

<- result

01 **Character array**

string(Index As Integer) As Character

lblLetter.Text = strName(5)

CH7\_F6

- accesses an individual character in a string based on character's index

<- n

02 **Contains method**

string.Contains(subString As String) As Boolean

blnIsContained = txtName.Text.Contains("Mart")

CH7\_F4.1

- determines whether a string contains a specific sequence of characters

<- True

03 **IndexOf method**

string.IndexOf(subString As String, startIndex As Integer) As Integer

intCharIndex = strName.IndexOf("ti")

CH7\_F4.2

- determines whether a string contains a specific sequence of characters;

<- 3

- returns either -1 (not present) or an integer that indicates the starting position of the character in the string

04 **Insert method**

string.Insert(startIndex As Integer, value As String) As String

txtPhone.Text = strName.Insert(0, "kaka ")

CH7\_F2

- inserts string in specified position in a string

<- kaka Martin

05 **Length property**

string.Length As Integer

intNumChars = strName.Length

CH7\_F1

- stores an integer that represents the number of characters contained in a string

<- 6

06 **Like operator**

string Like pattern As Boolean

If strName.ToUpper Like "M??TIN" Then

CH7\_F10

- uses pattern-matching characters ? \* # [...] [!...] to compare strings

If strInput.ToUpper Like "[A-Z] [A-Z]##" Then

07 **PadLeft method**

string.PadLeft(totalChars As Integer, padCharacter As Char) As String

strName = strName.PadLeft(10, "\*c")

CH7\_F3

- pads the beginning of a string with a character until the string has the specified number of characters;

<- \*\*\*\*Martin

- right-aligns the string

08 **PadRight method**

string.PadRight(totalChars As Integer, padCharacter As Char) As String

strName = strName.PadRight(10, "\*c")

CH7\_F3

- pads the end of a string with a character until the string has the specified number of characters;

<- Martin\*\*\*\*

- left-aligns the string

09 **Remove method**

string.Remove(startIndex As Integer, Count As Integer) As String

txtName.Text = strName.Remove(2, 3)

CH7\_F7

- removes characters from a string, based on character's index

<- Man

txtName.Text = strName.Remove(3)

<- Mar

| Concept name:                                                             | Syntax: , optional                                                                                     | exempli gratia:                                                                                                                                  | More info: |
|---------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------|------------|
| - purpose:                                                                |                                                                                                        | <- result                                                                                                                                        |            |
| 10 <b>Replace method</b>                                                  | <code>string.Replace(oldValue As String, newValue As String) As String</code>                          | <pre>Dim strQuote As String = "To Be Or Not To Be" strQuote = strQuote.Replace("To ", "2") &lt;- 2 Be Or Not 2 Be</pre>                          | CH7_F9     |
| - replaces all occurrences of oldValue in a string with newValue          |                                                                                                        |                                                                                                                                                  |            |
| 11 <b>Space function</b>                                                  | <code>Strings.Space(numberOfSpaces As Integer)</code><br><code>Space(numberOfSpaces As Integer)</code> | <code>As String</code><br><code>As String</code><br>1stNames.Items.Add("First" & Strings.Space(5) & "Last")<br><- First Last                     | CH7_F3     |
| - includes a specific number of space characters in a string              |                                                                                                        |                                                                                                                                                  |            |
| 12 <b>Substring method</b>                                                | <code>string.Substring(startIndex As Integer, Length As Integer) As String</code>                      | <pre>Dim strFull As String = "Laquisha Jones" strFirst = strFull.Substring(0, 8) &lt;- Laquisha strLast = strFull.Substring(9) &lt;- Jones</pre> | CH7_F5     |
| - accesses one or more characters in a string, based on character's index |                                                                                                        |                                                                                                                                                  |            |
| 13 <b>Trim method</b>                                                     | <code>string.Trim(trimChars As Char) As String</code>                                                  | <pre>Dim strPay As String = "\$\$***340.56***\$\$" strPay = strPay.Trim("\$"c, "*"c) &lt;- 340.56</pre>                                          | CH7_F8     |
| - removes all trimChars from both the beginning and the end of a string   |                                                                                                        |                                                                                                                                                  |            |
| 14 <b>TrimEnd method</b>                                                  | <code>string.TrimEnd(trimChars As Char) As String</code>                                               | <pre>strPay = strPay.TrimEnd("\$"c, "*"c) &lt;- \$\$***340.56</pre>                                                                              | CH7_F8     |
| - removes all trimChars from the end of a string                          |                                                                                                        |                                                                                                                                                  |            |
| 15 <b>TrimStart method</b>                                                | <code>string.TrimStart(trimChars As Char) As String</code>                                             | <pre>strPay = strPay.TrimStart("\$"c, "***c) &lt;- 340.56***\$\$</pre>                                                                           | CH7_F8     |
| - removes all trimChars from the beginning of a string                    |                                                                                                        |                                                                                                                                                  |            |

## CH7\_Key Terms

- **Array of characters / Character array** - a string; a group (or array) of related characters
- **Character array / Array of characters** - a string; a group (or array) of related characters
- **Check digit** - a digit that is added to either the beginning or typically the end of a number for the purpose of validating the number's authenticity
  - used on: credit card number, bank account number, UPC - Universal Product Code, ISBN - International Standard Book Number
- **Contains method** - performs a case-sensitive search to determine whether a string contains a specific sequence of characters; returns a Boolean value
- **Enabled property** - used to enable (True) or disable (False) a control during run time
- **Focus method** - moves the focus to a specified control during run time
- **IndexOf method** - performs a case-sensitive search to determine whether a string contains a specific sequence of characters
  - returns either -1 (if the string does not contain the sequence of characters) or
  - or an integer that represents the starting position of the sequence of characters
- **Insert method** - inserts characters anywhere in a string
- **Length property** - stores an integer that represents the number of characters contained in a string
- **Like operator** - uses case-sensitive pattern-matching characters ? # [...] [!]... to determine whether one string is equal to another string
- **MaxLength property** - a property of a text box control
  - specifies the maximum number of characters the control will accept

- **PadLeft method** - right-aligns a string by inserting characters at the beginning of the string
- **PadRight method** - left-aligns a string by inserting characters at the end of the string
- **PasswordChar property** - specifies the character used to hide the user's input
- **Pseudo-random number generator** - a mathematical algorithm that produces a sequence of random numbers
  - in VB, the pseudo-random generator is represented by an object whose data type is **Random**
- **Random object** - represents the pseudo-random number generator in VB
- **Random.Next method** - used to generate a random integer that is greater than or equal to minimum value but less than a maximum value
- **Remove method** - removes a specified number of characters located anywhere in a string
- **Replace method** - replaces a sequence of characters in a string with another sequence of characters
- **Substring method** - used to access any number of characters contained in a string
- **Trim method** - removes characters from both the beginning and the end of a string
- **TrimEnd method** - removes characters from the end of a string
- **TrimStart method** - removes characters from the beginning of a string

|                                                                  |                                                                                                                                                                                                                                              |
|------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 17.Check Digit Solution-Hyphens_EXERCISE 1_introductory          | - txt.Text.Replace.Length, For...Step...Next, Mod, MessageBox.Show,<br>txt_Enter, txt_TextChanged, txt_KeyPress, ControlChars                                                                                                                |
| 18.Modified Password Solution_EXERCISE 2                         | - txt.Text.Trim, <> String.Empty, str.IndexOf, str.Insert, txt_Enter, txt_TextChanged                                                                                                                                                        |
| 19.Validate ISBN Solution_EXERCISE 3                             | - str.Length, For...Step...Next, str.Substring, Mod, str Like str, MessageBox.Show,<br>txt_Enter, txt_TextChanged, txt_KeyPress, ControlChars                                                                                                |
| 20.Password Solution-Spaces_EXERCISE 4                           | - txt.Text.Trim, <> String.Empty, str.IndexOf, Do Until...Loop, Like "[! ]",<br>str.Trim, str.Insert, txt_Enter, txt_TextChanged                                                                                                             |
| 21.Color Solution_EXERCISE 5_introductory                        | - txt.Text.Trim.ToUpper, str.Length, Like, lbl.ForeColor, lbl.BackColor, Color., ElseIf,<br>SystemColors.Control, SystemColors.ControlText, MessageBox.Show,<br>txt.Focus, txt.SelectAll, txt_TextChanged, txt_Enter                         |
| 22.Poor Case Solution_EXERCISE 6_intermediate                    | - txt.Text.Trim, str(index), str.IndexOf, <> -1, str.Substring, str.ToUpper,<br>txt.Focus, txt.SelectAll, str.Trim, txt_Enter, txt_TextChanged                                                                                               |
| 23.Zip Solution_EXERCISE 7_intermediate                          | - input number of characters limitation set in Designer window: txt - Properties - maxLength<br>- Like "[]", txt.Focus, txt.SelectAll, txt_Enter, txt_TextChanged, txt_KeyPress, ControlChars                                                |
| 24.Check Digit Solution-ForNext_EXERCISE 8_intermediate          | - txt.Text.Length, For...Step...Next, Mod, MessageBox.Show,<br>txt_Enter, txt_TextChanged, txt_KeyPress, ControlChars                                                                                                                        |
| 25.Password Solution-Index_EXERCISE 9_intermediate               | - Const, txt.Text.Trim, str(Index), str.ToUpper, str.IndexOf,<br>txt_Enter, txt_TextChanged                                                                                                                                                  |
| 26.Shipping Solution_EXERCISE 10_intermediate                    | - str.ToUpper, ElseIf, Like "#characters", lst.SelectedIndex, MessageBox.Show,<br>frmMain_Load, lst.Items.Add, txt_Enter, txt_TextChanged                                                                                                    |
| 27.Password Solution-Advanced_EXERCISE 11_Advanced               | - txt.Text.Trim, str(index), str.IndexOf, Do Until...Loop, str.ToLower,<br>Like "[A-Z]" - range in uppercase, Like "[a-z]" - range in lowercase,<br>Mod, str.ToUpper, ElseIf, str.Insert, str.Length, txt_TextChanged, txt_Enter             |
| 28.Poor Case Solution-Middle_EXERCISE 12_Advanced_C INFO         | - does not work properly, other version 28b is super                                                                                                                                                                                         |
| 28b.Poor Case Solution-Middle_EXERCISE 12_Advanced_C INFO        | - works for max 3 words<br>- str.IndexOf, str(index), str.Substring, str.Trim, str.ToUpper,<br>txt_Enter, txt_TextChanged                                                                                                                    |
| 29.Poor Case Solution-Hyphenated_EXERCISE 13_Advanced            | - txt.Text.Trim, str(index), str.IndexOf, str.Substring, str.Trim,<br>str.ToUpper, str.TrimEnd, txt_Enter, txt_TextChanged                                                                                                                   |
| 30.Rembrandt Solution_EXERCISE 14_Advanced                       | - Private used as accumulator, str.ToUpper, Select Case, str.Contains, MessageBox.Show,<br>Like "###[characters][characters]" - any 3 numbers + 2 defined characters,<br>txt_KeyPress, ControlChars, Private Sub - independent Sub procedure |
| 31.Addition Solution_EXERCISE 15_Advanced                        | - Private used as global Random integer, Private Sub - independent Sub procedure, btn.Enabled,<br>randGen As New Random, randGen.Next, MessageBox.Show, txt_KeyPress, ControlChars                                                           |
| 32.Validate Number Solution_EXERCISE 16_Advanced                 | - str.Length, For...Step...Next, MessageBox.Show, Mod, txt_TextChanged,<br>txt_KeyPress, ControlChars, frmMain_FormClosing, dlgButton As DialogResult,<br>dlgButton = MessageBox.Show, DialogResult.No, e.Cancel                             |
| 33.OnYourOwn Solution_EXERCISE 17_NOT FINISHED                   | - not done, postponed for müza time                                                                                                                                                                                                          |
| 34.FixIt Solution_EXERCISE 18                                    | - txt.Text.Trim, For...To...Step...Next, str(index), txt_Enter, txt_TextChanged                                                                                                                                                              |
| _Appendix E - Case Projects_06.High Total Game-CH 01-07          | - using .NET Framework 4.8                                                                                                                                                                                                                   |
| _Appendix E - Case Projects_07.Math Practice-CH 01-07            | - using .NET Framework 4.8                                                                                                                                                                                                                   |
| _Appendix E - Case Projects_08.Tax-Deductible Calculator-CH01-07 | - using .NET Framework 4.8                                                                                                                                                                                                                   |

1. In this exercise, you modify the Check Digit application from this chapter's Apply lesson. Use Windows to make a copy of the Check Digit Solution folder. Rename the copy Check Digit Solution-Hyphens. Open the Check Digit Solution.sln file contained in the Check Digit Solution-Hyphens folder.
  - a. Modify the txtIsbn\_KeyPress procedure to allow the user to also enter hyphens.
  - b. Before verifying the length of the ISBN entered by the user, the btnAssign\_Click procedure should assign the ISBN (without any hyphens) to the strIsbn variable. Make the appropriate modifications to the procedure.
  - c. When displaying the ISBN in the lblFinalISBN control, the btnAssign\_Click procedure should insert a hyphen after the third number, the fourth number, the seventh number, and the twelfth number (for example, 978-1-337-10212-4). Make the appropriate modifications to the procedure.
  - d. Save the solution and then start and test the application. (If the user enters 978-1-285-86026, with or without the hyphens, the btnAssign\_Click procedure should display 978-1-285-86026-8.)

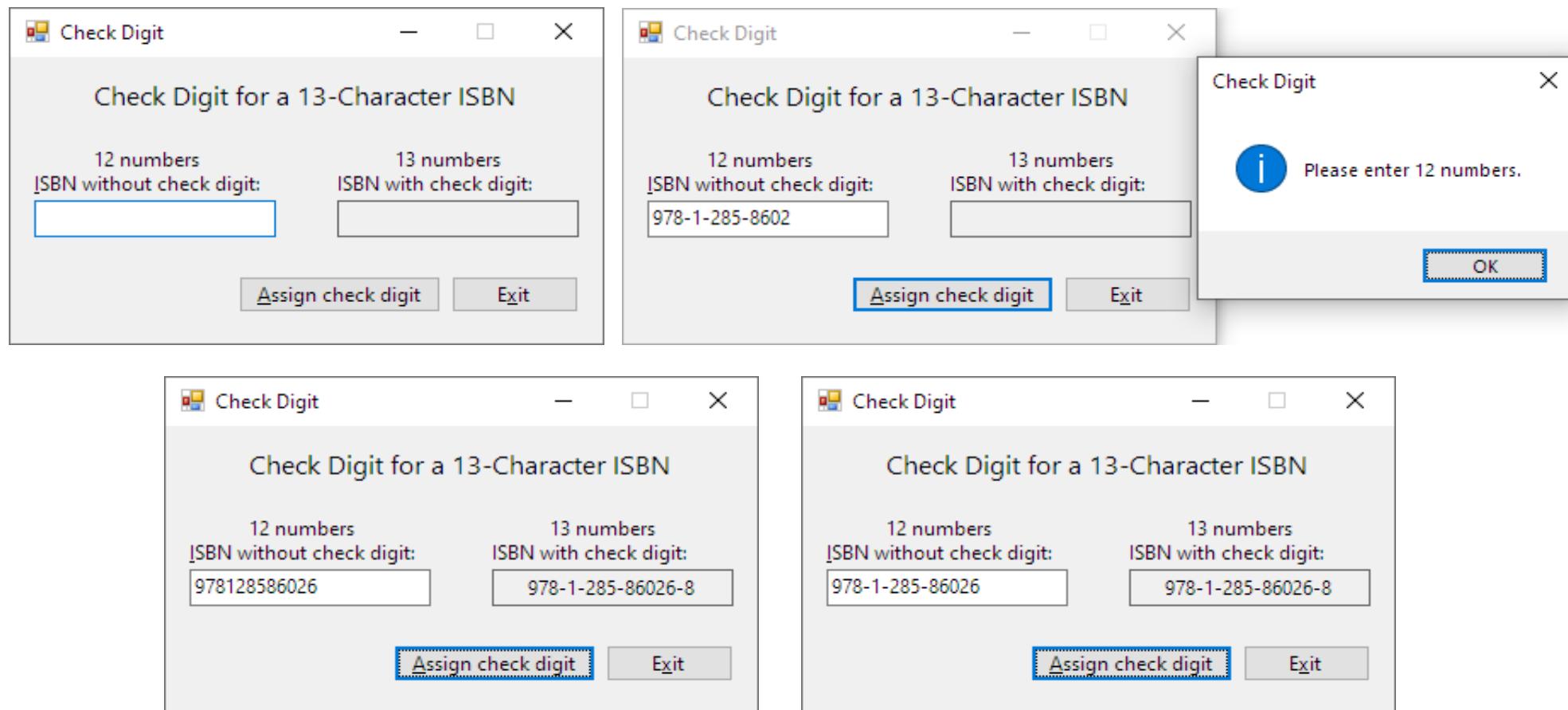
```

1  ' Name:          Check Digit Project
2  ' a). modify the txtIsbn_KeyPress to allow the user to also enter hyphens "-"
3  ' b). before verifying the length of the ISBN entered by the user, the btnAssign_Click should assign the ISBN without any hyphens,
4  '       to strIsbn variable -> TRIM or REPLACE? TRIM i try on line 25
5  ' c). when display ISBN in lblFinalISBN, the btnAssign_Click should insert "-" after the:
6  '       3rd, 4th, 7th, 12th -> 123-4-567-89012-3 -> charIndex of: 3,4,7,12 -> line 42
7  ' d). test: enter with and without:978128586026, 978-1-285-86026, the result: 978-1-285-86026-8
8
9 Option Explicit On
10 Option Strict On
11 Option Infer Off
12
13 Public Class frmMain
14     Private Sub btnAssign_Click(sender As Object, e As EventArgs) Handles btnAssign.Click
15         ' Assign a check digit to an ISBN.
16
17         Dim strIsbn As String
18         Dim intDigit As Integer
19         Dim intTotalOdd As Integer
20         Dim intTotalEven As Integer
21         Dim intGrandTotal As Integer
22         Dim intRemainder As Integer
23         Dim intCheckDigit As Integer
24

```

```
25     'txtIsbn.Text = txtIsbn.Text.Replace("-", "")  
26     If txtIsbn.Text.Replace("-", "").Length = 12 Then  
27         strIsbn = txtIsbn.Text.Replace("-", "")  
28         For intOdd As Integer = 1 To 11 Step 2  
29             Integer.TryParse(strIsbn(intOdd), intDigit)  
30             intTotalOdd += (intDigit * 3)  
31         Next intOdd  
32         For intEven As Integer = 0 To 10 Step 2  
33             Integer.TryParse(strIsbn(intEven), intDigit)  
34             intTotalEven += intDigit  
35         Next intEven  
36  
37         intGrandTotal = intTotalOdd + intTotalEven  
38         intRemainder = intGrandTotal Mod 10  
39  
40         If intRemainder <> 0 Then  
41             intCheckDigit = 10 - intRemainder  
42         End If  
43         ' INSERT - no 3rd, 4th, 7th, 12th -> 123-4-567-89012-3  
44         '   CharIndex: 012-3-456-78901-  
45         'lblFinalIsbn.Text = strIsbn & intCheckDigit.ToString <- original  
46         lblFinalIsbn.Text = strIsbn.Substring(0, 3) & "-" & strIsbn(3) & "-" & strIsbn.Substring(4, 3) &  
47             "--" & strIsbn.Substring(7, 5) & "-" & intCheckDigit.ToString  
48  
49     Else  
50         MessageBox.Show("Please enter 12 numbers.", "Check Digit",  
51                         MessageBoxButtons.OK, MessageBoxIcon.Information)  
52     End If  
53 End Sub  
54  
55 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click  
56     Me.Close()  
57 End Sub  
58  
59 Private Sub txtIsbn_Enter(sender As Object, e As EventArgs) Handles txtIsbn.Enter  
60     txtIsbn.SelectAll()  
61 End Sub  
62  
63 Private Sub txtIsbn_TextChanged(sender As Object, e As EventArgs) Handles txtIsbn.TextChanged  
64     lblFinalIsbn.Text = String.Empty  
65 End Sub  
66  
67 Private Sub txtIsbn_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtIsbn.KeyPress  
68     ' Allow only numbers, the Backspace key, and hyphens - a).
```

```
69
70     If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back AndAlso e.KeyChar <> "-" Then
71         e.Handled = True
72     End If
73 End Sub
74 End Class
```



2. In this exercise, you modify the Password application from this chapter's Apply lesson.

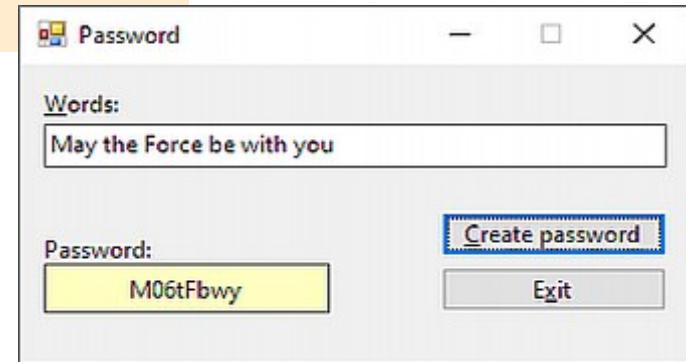
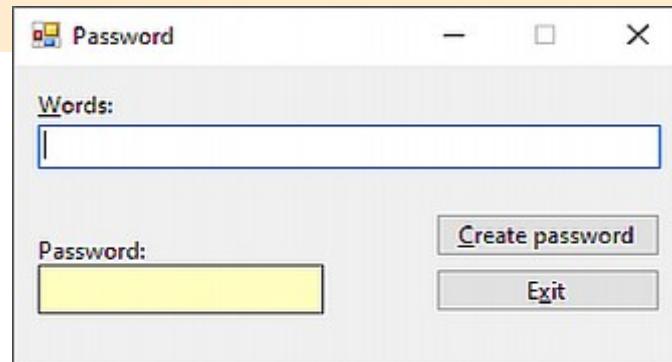
Use Windows to make a copy of the Password Solution folder. Rename the copy Modified Password Solution. Open the Password Solution.sln file contained in the Modified Password Solution folder. The btnCreate\_Click procedure inserts a number immediately after the first character in the password. The number represents the length of the string before the number is inserted. Modify the procedure so that the number always contains two characters. For example, if the length of the string is 6, insert "06" (a zero and the number 6). Save the solution and then start and test the application. (If the user enters "May the Force be with you", the Create password button should display M06tFbwy.)

```
1  'the btnCreate_Click procedure inserts a number immediately after the 1st character in the password
2  'the number represents the LENGTH of the string before the number is inserted
3  '-> modify the procedure so that the number always contains 2 characters
4  '    e.g. if the length of the string is 6, insert "06"
5  'test: if the user enters: "May the Force be with you", the result should be: M06tFbwy
6
7  Option Explicit On
8  Option Strict On
9  Option Infer Off
10
11 Public Class frmMain
12     Private Sub btnCreate_Click(sender As Object, e As EventArgs) Handles btnCreate.Click
13         ' Create a password.
14
15         Dim strWords As String
16         Dim strPassword As String
17         Dim intSpaceIndex As Integer
18
19         strWords = txtWords.Text.Trim
20
21         If strWords <> String.Empty Then
22             ' Assign the first character as the password:
23             strPassword = strWords(0)
24
25             ' Search for the first space in the input:
26             intSpaceIndex = strWords.IndexOf(" ")
27
```

```

28     Do Until intSpaceIndex = -1
29         ' Concatenate the character that follows
30         ' the space to the password:
31         strPassword = strPassword & strWords(intSpaceIndex + 1)
32         ' Search for the next space:
33         intSpaceIndex = strWords.IndexOf(" ", intSpaceIndex + 1)
34     Loop
35
36     ' Insert the 2-digit number after the first character: -> MODIFICATION
37
38     strPassword = strPassword.Insert(1, strPassword.Length.ToString)
39
40     'MODIFICATION:
41     If strPassword.Length < 10 Then
42         strPassword = strPassword.Insert(1, "0")
43     End If
44
45     ' Display the final password:
46     lblPassword.Text = strPassword
47 End If
48 End Sub
49
50 Private Sub txtWords_Enter(sender As Object, e As EventArgs) Handles txtWords.Enter
51     txtWords.SelectAll()
52 End Sub
53
54 Private Sub txtWords_TextChanged(sender As Object, e As EventArgs) Handles txtWords.TextChanged
55     lblPassword.Text = String.Empty
56 End Sub
57
58 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
59     Me.Close()
60 End Sub
61 End Class

```



## Chap07\Exercise1

### 19. Validate ISBN Solution\_EXERCISE 3

- str.Length, For...Step...Next, str.Substring, Mod, str Like str, MessageBox.Show,  
txt\_Enter, txt\_TextChanged, txt\_KeyPress, ControlChars

3. Open the Validate ISBN Solution.sln file contained in the VB2017\Chap07\Validate ISBN Solution folder. The interface provides a text box for entering a 13-character ISBN. The btnValidate\_Click procedure should use the ISBN's check digit, which is the last digit in the number, to determine whether the ISBN is valid. (The check digit algorithm is shown earlier in Figure 7-25.) If the ISBN is valid, the procedure should display the "Valid" message in the lblStatus control; otherwise, it should display the "Not valid" message. Code the procedure. Save the solution and then start and test the application. (If the user enters 9781285860268, the btnValidate\_Click procedure should display the "Valid" message.)

```
1  'GUI provides a txt for entering a 13-char ISBN with a last CheckDigit
2  '-> btnValidate_Click procedure should use the ISBN's check digit to determine whether the ISBN is valid
3  'the check digit algorithm is shown earlier in Figure 7-25
4  '-> if the ISBN is valid, the procedure should display the "Valid" in the lblStatus; otherwise, it should display "Not valid"
5  '-> code the procedure
6  '-> test: enter 9781285860268 -> "Valid"
7  'im testing with a: 978-1-337-10212-4 -> 9781337102124 -> 4 is a check digit!
8
9  Option Explicit On
10 Option Strict On
11 Option Infer Off
12
13 Public Class frmMain
14     Private Sub btnValidate_Click(sender As Object, e As EventArgs) Handles btnValidate.Click
15
16         Dim strUserIsbn As String = txtIsbn.Text                      'txtIsbn.Text entered by the user -> 13-char with check-digit
17         If strUserIsbn.Length = 13 Then
18
19
20             Dim strPick As String
21             Dim intStep1 As Integer
22             Dim intTotalOdd As Integer
23             Dim intStep2 As Integer
24             Dim intTotalEven As Integer
25             Dim intGrandTotal As Integer
```

```
26     Dim intRemainder As Integer
27     Dim intCheckDigit As Integer
28     Dim strCheckDigit As String
29
30     'Step 1: every odd index multiply by 3 and add the results together:
31     For intOddIndex As Integer = 1 To 11 Step 2
32         'MessageBox.Show(intOddIndex.ToString)
33         strPick = strUserIsbn.Substring(intOddIndex, 1)
34         'MessageBox.Show(strStep1)
35         Integer.TryParse(strPick, intStep1)
36         intStep1 = intStep1 * 3
37         intTotalOdd += intStep1
38     Next intOddIndex
39     'the result should be: 48:
40     'MessageBox.Show(intTotalOdd.ToString)
41
42     'Step 2: add all even indices together -> except the last check digit !!!
43     For intEvenIndex As Integer = 0 To 10 Step 2
44         strPick = strUserIsbn.Substring(intEvenIndex, 1)
45         Integer.TryParse(strPick, intStep2)
46         intTotalEven += intStep2
47     Next intEvenIndex
48     'the result should be: 28:
49     'MessageBox.Show(intTotalEven.ToString)
50
51     'Step 3(intGrandTotal): add both sums together:
52     intGrandTotal = intTotalOdd + intTotalEven
53     'the result should be: 76:
54     'MessageBox.Show(intGrandTotal.ToString)
55
56     'Step 4(intRemainder): Step 3 Mod 10    <- 10 is choosed by the coder
57     intRemainder = intGrandTotal Mod 10
58     'the result should be: 6:
59     'MessageBox.Show(intRemainder.ToString)
60
61     'Step 5(intCheckDigit): 10 - Step 4    <- 10 choosed by the coder
62     '- but if the Step 4-intRemainder = 0, then the Step 5 is 0
63     If intRemainder = 0 Then
64         intCheckDigit = 0
65     Else
66         intCheckDigit = 10 - intRemainder
67     End If
68     'the result should be: 4:
69     'MessageBox.Show(intCheckDigit.ToString)
```

```
70      'compare the check digits: strCompare <-> strCheckDigit
71      strCheckDigit = intCheckDigit.ToString
72      strUserIsbn = strUserIsbn.Remove(0, 12)
73
74      'MessageBox.Show(intCheckDigit.ToString)
75
76
77      If strCheckDigit Like strUserIsbn Then
78          lblStatus.Text = "Valid"
79      Else
80          lblStatus.Text = "Not Valid"
81      End If
82
83      Else
84          MessageBox.Show("Please enter 13 character ISBN including a check digit", "Validate ISBN",
85                          MessageBoxButtons.OK, MessageBoxIcon.Exclamation)
86      End If
87
88  End Sub
89  Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
90      Me.Close()
91  End Sub
92
93  Private Sub txtIsbn_Enter(sender As Object, e As EventArgs) Handles txtIsbn.Enter
94      txtIsbn.SelectAll()
95  End Sub
96
97  Private Sub txtIsbn_TextChanged(sender As Object, e As EventArgs) Handles txtIsbn.TextChanged
98      lblStatus.Text = String.Empty
99  End Sub
100
101 Private Sub txtIsbn_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtIsbn.KeyPress
102     ' Allow only numbers and the Backspace key.
103
104     If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
105         e.Handled = True
106     End If
107  End Sub
108
109 End Class
```

- Step 1:** Starting with the second digit, multiply every other digit by 3, and then total the results.  
 - i.e.: multiply by 3 the odd indexes 1, 3, 5, 7, 9, and 11 -> 2nd, 4th, 6th, 8th, 10th, and 12th digits, and then add together all of the products
- Step 2:** Add together each of the digits skipped in Step 1.  
 - these will be the even indexes 0, 2, 4, 6, 8, 10 -> 1st, 3rd, 5th, 7th, 9th, and 11th digits
- Step 3:** Add the sum from Step 1 to the sum from Step 2.
- Step 4:** Divide the sum from Step 3 by 10 and find the remainder. (10 choosed by the coder)
- Step 5:** If the remainder from Step 4 is 0, then the check digit is 0. Otherwise, subtract the remainder from 10, giving the check digit  
 (10 choosed by the coder)

e.g.:

ISBN without check digit: 978-1-337-10212

ISBN with check digit: 978-1-337-10212-4 calculated as shown here:

| index:  | 0         | 1                                                 | 2   | 3                                                | 4   | 5                                                | 6   | 7                                                | 8   | 9                                                | 10   | 11                                               | result:                      | info:                                                                           | variable:     |
|---------|-----------|---------------------------------------------------|-----|--------------------------------------------------|-----|--------------------------------------------------|-----|--------------------------------------------------|-----|--------------------------------------------------|------|--------------------------------------------------|------------------------------|---------------------------------------------------------------------------------|---------------|
| digit:  | 1st       | 2nd                                               | 3rd | 4th                                              | 5th | 6th                                              | 7th | 8th                                              | 9th | 10th                                             | 11th | 12th                                             |                              | - blabla                                                                        |               |
| ISBN:   | 9         | 7                                                 | 8   | 1                                                | 3   | 3                                                | 7   | 1                                                | 0   | 2                                                | 1    | 2                                                |                              |                                                                                 |               |
| Step 1: | -         | $\frac{7}{\begin{array}{l} *3 \\ 21 \end{array}}$ | -   | $\frac{1}{\begin{array}{l} *3 \\ 3 \end{array}}$ | -   | $\frac{3}{\begin{array}{l} *3 \\ 9 \end{array}}$ | -   | $\frac{1}{\begin{array}{l} *3 \\ 3 \end{array}}$ | -   | $\frac{2}{\begin{array}{l} *3 \\ 6 \end{array}}$ | -    | $\frac{2}{\begin{array}{l} *3 \\ 6 \end{array}}$ | = 48                         | < every odd index (even number),<br><- multiply by 3, and<br><- add the results | intTotalOdd   |
| Step 2: | 9         | -                                                 | 8   | -                                                | 3   | -                                                | 7   | -                                                | 0   | -                                                | 1    | -                                                | = 28                         | <- add all even indices together                                                | intTotalEven  |
| Step 3: | 9         | 21                                                | 8   | 3                                                | 3   | 9                                                | 7   | 3                                                | 0   | 6                                                | 1    | 6                                                | = 76                         | <- add the sums together                                                        | intGrandTotal |
| Step 4: | 76 Mod 10 |                                                   |     |                                                  |     |                                                  |     |                                                  |     |                                                  |      | = 6                                              | - number 10 set by the coder | intRemainder                                                                    |               |
| Step 5: | 10 - 6    |                                                   |     |                                                  |     |                                                  |     |                                                  |     |                                                  |      | = 4                                              | - number 10 set by the coder | intCheckDigit                                                                   |               |

ISBN

13-character ISBN: 9781337102124

Validate ISBN

Valid/Not valid: Valid

Exit

ISBN

13-character ISBN: 9781337102126

Validate ISBN

Valid/Not valid: Not Valid

Exit

ISBN

13-character ISBN: 97813

Validate ISBN

**!** Please enter 13 character ISBN including a check digit

OK

- txt.Text.Trim, <> String.Empty, str.IndexOf, Do Until...Loop, Like "[! ]", str.Trim, str.Insert, txt\_Enter, txt\_TextChanged

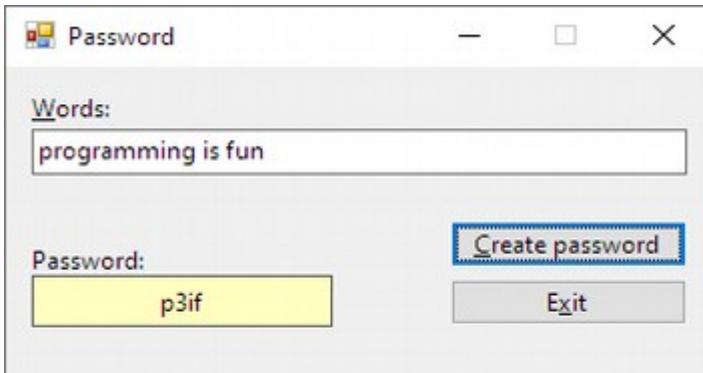
4. In this exercise, you modify the Password application from this chapter's Apply lesson. Use Windows to make a copy of the Password Solution folder. Rename the copy Password Solution-Spaces. Open the Password Solution.sln file contained in the Password Solution-Spaces folder.
  - a. Start the application. Type the following three words, using two spaces (rather than one space) to separate each word: programming is fun. Click the Create password button. The Password box displays p5 followed by a space, the letter i, a space and the letter f. Click the Exit button.
  - b. Open the Code Editor window and locate the btnCreate\_Click procedure. The first instruction in the loop concatenates the character that follows the space. Modify the loop's code so that it performs the concatenation only when the character is not a space. Save the solution and then start the application. Use the information in Step a to test the application. This time, the Password box should display p3if.

```
1 Option Explicit On
2 Option Strict On
3 Option Infer Off
4
5 Public Class frmMain
6     Private Sub btnCreate_Click(sender As Object, e As EventArgs) Handles btnCreate.Click
7         ' Create a password.
8
9         Dim strWords As String
10        Dim strPassword As String
11        Dim intSpaceIndex As Integer
12
13        strWords = txtWords.Text.Trim
14
15        If strWords <> String.Empty Then
16            ' Assign the first character as the password:
17            strPassword = strWords(0)
18
19            ' Search for the first space in the input:
20            intSpaceIndex = strWords.IndexOf(" ")
21
22            Do Until intSpaceIndex = -1
23                ' Concatenate the character that follows
24                ' the space to the password:
25                'MODIFY that it concatenates only when a character is NOT a SPACE: string Like "[! ]" As Boolean
26                'strPassword = strPassword & strWords(intSpaceIndex + 1)    <- ORIGINAL
```

```

27
28     If strWords(intSpaceIndex + 1) Like "[! ]" Then
29         strPassword = strPassword & strWords(intSpaceIndex + 1)
30     Else
31         strWords = strWords.Trim()
32     End If
33
34     ' Search for the next space:
35     intSpaceIndex = strWords.IndexOf(" ", intSpaceIndex + 1)
36 Loop
37
38     ' Insert the number after the first character:
39     strPassword = strPassword.Insert(1, strPassword.Length.ToString)
40     ' Display the final password:
41     lblPassword.Text = strPassword
42 End If
43 End Sub
44
45 Private Sub txtWords_Enter(sender As Object, e As EventArgs) Handles txtWords.Enter
46     txtWords.SelectAll()
47 End Sub
48
49 Private Sub txtWords_TextChanged(sender As Object, e As EventArgs) Handles txtWords.TextChanged
50     lblPassword.Text = String.Empty
51 End Sub
52
53 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
54     Me.Close()
55 End Sub
56 End Class

```



## Chap07\Exercise1

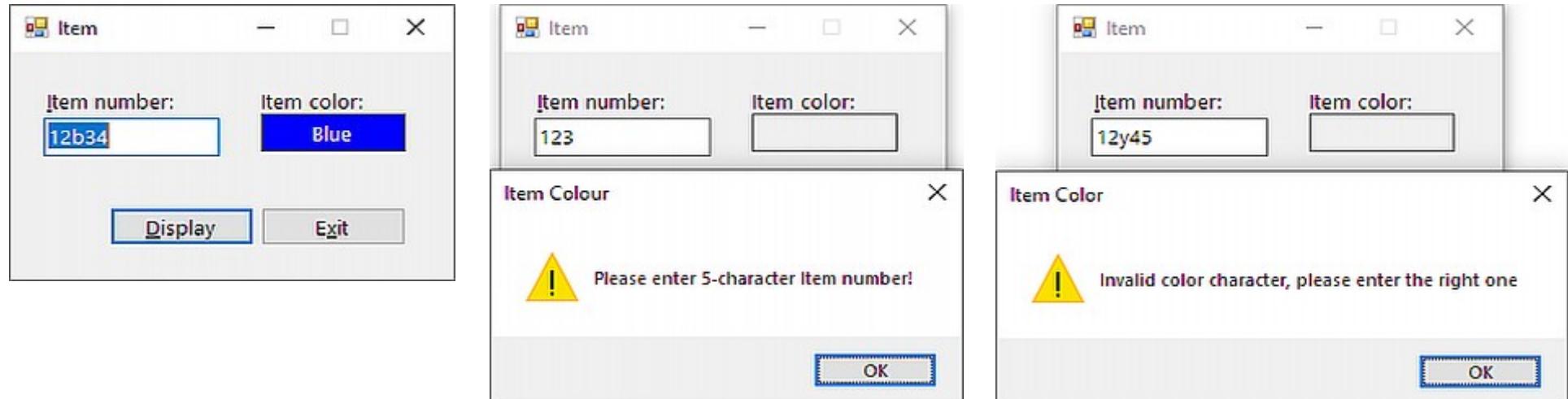
### 21. Color Solution\_EXERCISE 5\_introductory

```
- txt.Text.Trim.ToUpper, str.Length, Like, lbl.ForeColor, lbl.BackColor, Color., ElseIf,
SystemColors.Control, SystemColors.ControlText, MessageBox.Show,
txt.Focus, txt.SelectAll, txt_TextChanged, txt_Enter
```

5. Open the Color Solution.sln file contained in the VB2017\Chap07\Color Solution folder. The btnDisplay\_Click procedure should display the color of the item whose item number is entered by the user. All item numbers contain exactly five characters. All items are available in four colors: blue, green, red, and white. The third character in the item number indicates the item's color, as follows: B or b indicates Blue, G or g indicates Green, R or r indicates Red, and W or w indicates White. The procedure should display an appropriate error message if the item number does not contain exactly five characters. It should also display an appropriate message if the third character is not one of the valid color characters. Before ending, the procedure should send the focus to the txtItem control. Code the procedure. Save the solution and then start the application. Test the application using the following invalid item numbers: 123, 12345, 123456, and 12Y45. Then, test it using the following valid item numbers: 12b34, abr73, n6gtn, and 12w87.

```
1  '-> btnDisplay_Click procedure should display the color of the item whose item number is entered by the user
2  '- all item numbers contain exactly 5 characters
3  '- all items are available in 4 colours: BLUE, GREEN, RED, and WHITE
4  '- the 3RD character in the item number indicates the item's color: b/B = BLUE, g/G = GREEN, r/R = RED, w/W = WHITE
5  '-> if item number does not contain 5 characters -> appropriate error message
6  '-> if 3RD character is not one of the valid color characters -> appropriate error message
7  '-> before ending, the procedure should SEND THE FOCUS to the txtItem control
8  '-> code the procedure
9  '-> test invalid numbers: 123, 12345, 123456, 12Y45
10 '-> test valid numbers: 12b34, abr73, n6gtn, 12w87
11 Option Explicit On
12 Option Strict On
13 Option Infer Off
14 Public Class frmMain
15     Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
16         ' txtItem; lblColor
17
18         Dim strItem As String = txtItem.Text.Trim.ToUpper
19
20         If strItem.Length = 5 Then    ' exactly 5-character Item number:
21             If strItem(2) Like "B" Then
22                 lblColor.ForeColor = Color.White
23                 lblColor.BackColor = Color.Blue
24                 lblColor.Text = "Blue"
```

```
25      ElseIf strItem(2) Like "G" Then
26          lblColor.ForeColor = Color.White
27          lblColor.BackColor = Color.Green
28          lblColor.Text = "Green"
29      ElseIf strItem(2) Like "R" Then
30          lblColor.ForeColor = Color.White
31          lblColor.BackColor = Color.Red
32          lblColor.Text = "Red"
33      ElseIf strItem(2) Like "W" Then
34          lblColor.BackColor = Color.White
35          lblColor.Text = "White"
36      Else
37          MessageBox.Show("Invalid color character, please enter the right one", "Item Color",
38                          MessageBoxButtons.OK, MessageBoxIcon.Exclamation)
39          txtItem.Focus()
40          txtItem.SelectAll()
41      End If
42      txtItem.Focus()
43      txtItem.SelectAll()
44
45      Else ' less/more than 5-character Item number:
46          MessageBox.Show("Please enter 5-character Item number!", "Item Colour",
47                          MessageBoxButtons.OK, MessageBoxIcon.Exclamation)
48          txtItem.Focus()
49          txtItem.SelectAll()
50      End If
51  End Sub
52
53  Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
54      Me.Close()
55  End Sub
56
57  Private Sub txtItem_TextChanged(sender As Object, e As EventArgs) Handles txtItem.TextChanged
58      lblColor.Text = String.Empty
59      lblColor.BackColor = SystemColors.Control
60      lblColor.ForeColor = SystemColors.ControlText
61  End Sub
62
63  Private Sub txtItem_Enter(sender As Object, e As EventArgs) Handles txtItem.Enter
64      txtItem.SelectAll()
65  End Sub
66
67 End Class
```



## Chap07\Exercises

### 22. Proper Case Solution\_EXERCISE 6\_intermediate

```
- txt.Text.Trim, str(index), str.IndexOf, <> -1, str.Substring, str.ToUpper,  
txt.Focus, txt.SelectAll, str.Trim, txt_Enter, txt_TextChanged
```

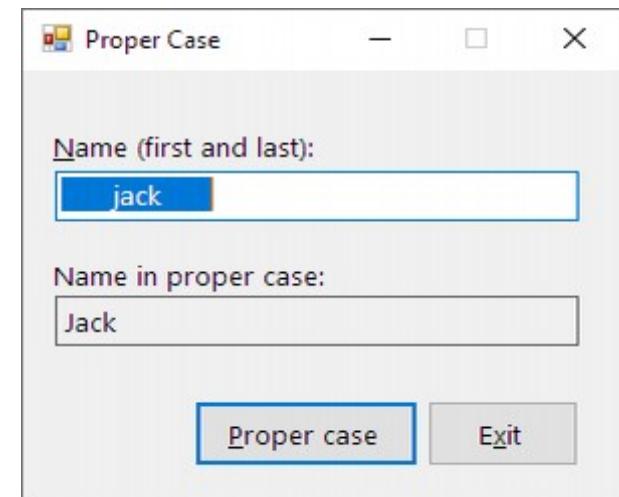
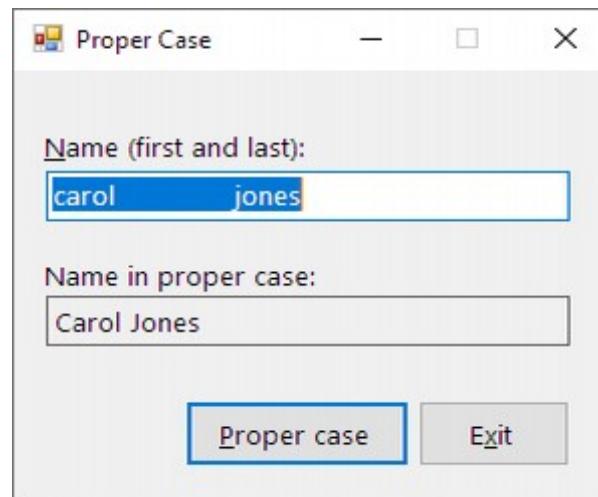
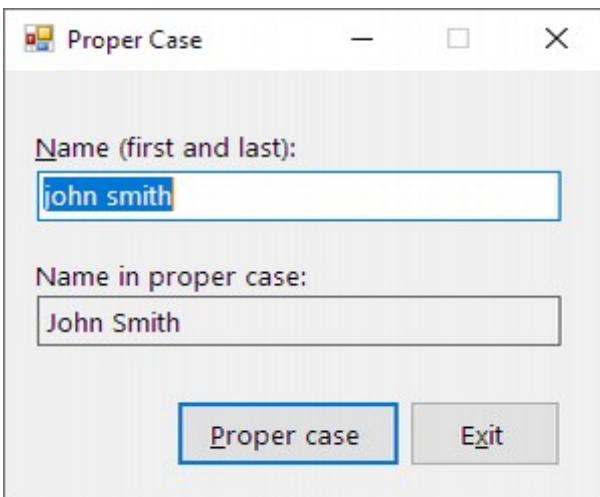
6. Open the Proper Case Solution.sln file contained in the VB2017\Chap07\Proper Case Solution folder. The interface provides a text box for entering a person's first and last names. The btnProper\_Click procedure should display the first and last names in the proper case. In other words, the first and last names should begin with an uppercase letter and the remaining letters in each name should be lowercase. If the user enters only one name, display the name in proper case. Be sure the btnProper\_Click procedure works correctly if the user inadvertently enters more than one space between the first and last names. After displaying the name, the procedure should send the focus to the txtName control. Code the procedure. Save the solution and then start and test the application. (If the user enters "john smith" as the name, the application should display "John Smith". If the user enters "carol" followed by three spaces and then "jones", the application should display "Carol Jones". If the user enters three spaces followed by "jack" and another three spaces, the application should display "Jack".)

```
1  'GUI provides a txt for entering a person's 1st and last names
2  '->btnProper_Click procedure should display the 1st and last names in the proper case =
3  '
4  '           = should begin With an UPPERCASE letter followed by the lowercase letters
5  '-> if the user enteres only one name, display the name in proper case
6  '-> be sure the procedure works correctly if the user inadvertently enteres more than one space between the 1st and last names
7  '-> after displaying the name, the procedure should send the focus to the txtName control (Focus & SelectAll)
8  '->code the procedure, save and test:
9  '-> test: "john smith" => "John Smith"; "carol    jones" => "Carol Jones"; "    jack    " => "Jack"
10 Option Explicit On
11 Option Strict On
12 Option Infer Off
13
14 Public Class frmMain
15     Private Sub btnProper_Click(sender As Object, e As EventArgs) Handles btnProper.Click
16         'txtName; lblName
17         Dim strName As String = txtName.Text.Trim          ' removes any leading and trailing spaces
18         Dim intSpaceIndex As Integer                      ' Index of the Space char for the 2nd name
19         Dim str1 As String
20         Dim str1a As String = String.Empty
21         Dim str2 As String
22         Dim str2a As String
23         Dim str2b As String
24
25         ' fill the variable with a 1st character from the 1st name:
26         str1 = strName(0)
27
28         ' fill variable with an index number of a Space character, searching from the 2nd char(1) of the word: (-1 = there is none)
29         intSpaceIndex = strName.IndexOf(" ", 1)
30
31         If intSpaceIndex <> -1 Then      ' if there is a second name after a Space character:
32
33             ' 1st name:
34             For Int1b As Integer = 1 To intSpaceIndex - 1
35                 str1a += strName(Int1b)
36             Next Int1b
37
38             ' 2nd name:
39             str2 = strName.Substring(intSpaceIndex + 1)
40             'MessageBox.Show(str2)
41             str2 = str2.Trim
42
43             ' 2nd name, first letter in Uppercase:
44             str2a = str2(0)
45             str2b = str2.Substring(1)
```

```

45
46     lblName.Text = str1.ToUpper & str1a & " " & str2a.ToUpper & str2b
47
48     txtName.Focus()
49     txtName.SelectAll()
50
51     Else ' If there is only 1 name (If intSpaceIndex = -1 = there is none):
52         lblName.Text = str1.ToUpper & strName.Substring(1)
53         txtName.Focus()
54         txtName.SelectAll()
55     End If
56 End Sub
57
58 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
59     Me.Close()
60 End Sub
61
62 Private Sub txtName_Enter(sender As Object, e As EventArgs) Handles txtName.Enter
63     txtName.SelectAll()
64 End Sub
65
66 Private Sub txtName_TextChanged(sender As Object, e As EventArgs) Handles txtName.TextChanged
67     lblName.Text = String.Empty
68 End Sub
69
70 End Class

```



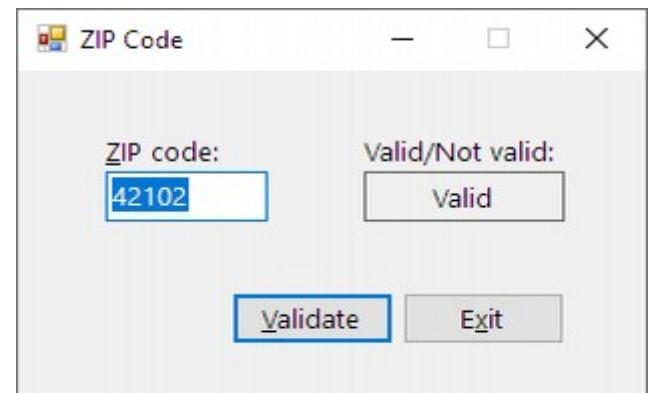
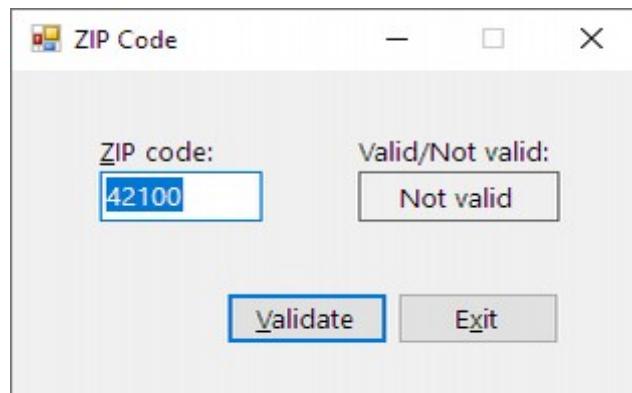
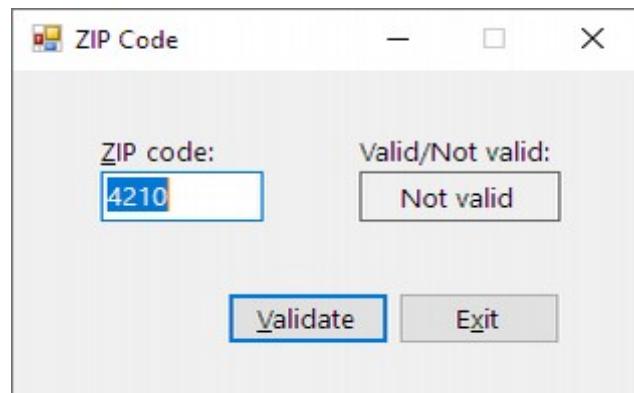
- input number of characters limitation set in Designer window: txt - Properties - maxLength  
 - Like "[ ]", txt.Focus, txt.SelectAll, txt\_Enter, txt\_TextChanged, txt\_KeyPress, ControlChars

7. Open the Zip Solution.sln file contained in the VB2017\Chap07\Zip Solution folder file. The btnDisplay\_Click procedure should validate the ZIP code entered by the user. To be valid, the first four digits in the ZIP code must be 4210, and the last digit must be 2, 3, or 4. Use one selection structure along with the Like operator to validate the ZIP code. Display the "Valid" message if the ZIP code is valid; otherwise, display the "Not valid" message. After displaying the message, the procedure should send the focus to the txtZip control. Code the procedure. Save the solution and then start and test the application.

```

1  ' ->btnDisplay_Click procedure should validate the 5-digit ZIP code entered by the user
2  ' -> to be valid: digit 1234 must be 4210, digit 5 must be 2/3/4
3  ' ->use 1 selection structure along with the Like operator to validate the ZIP code
4  ' ->in lbl display "Valid" or "Not valid"
5  ' ->then the procedure should send the focus to the txtZip control
6  Option Explicit On
7  Option Strict On
8  Option Infer Off
9
10 Public Class frmMain
11     Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
12         'txtZip; lblStatus
13
14         ' _I set in Designer window: txtZip - Properties - maxLength = 5 !!! -> therefore can enter only 5 characters
15
16         Dim strZip As String = txtZip.Text
17
18         If strZip Like "4210[234]" Then
19             lblStatus.Text = "Valid"
20             txtZip.Focus()
21             txtZip.SelectAll()
22         Else
23             lblStatus.Text = "Not valid"
24             txtZip.Focus()
25             txtZip.SelectAll()
26         End If
27     End Sub
28     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
29         Me.Close()
30     End Sub
31 
```

```
32      Private Sub txtZip_Enter(sender As Object, e As EventArgs) Handles txtZip.Enter
33          txtZip.SelectAll()
34      End Sub
35
36      Private Sub txtZip_TextChanged(sender As Object, e As EventArgs) Handles txtZip.TextChanged
37          lblStatus.Text = String.Empty
38      End Sub
39
40      Private Sub txtZip_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtZip.KeyPress
41          ' Allow only numbers and the Backspace key:
42          If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
43              e.Handled = True
44          End If
45      End Sub
46  End Class
```



8. In this exercise, you modify the Check Digit application from this chapter's Apply lesson. Use Windows to make a copy of the Check Digit Solution folder. Rename the copy Check Digit Solution-ForNext. Open the Check Digit Solution.sln file contained in the Check Digit Solution-ForNext folder. Delete the `intGrandTotal = intTotalOdd + intTotalEven` statement from the `btnAssign_Click` procedure. Also delete the two `Dim` statements that declare the `intTotalOdd` and `intTotalEven` variables. Modify the procedure to use one `For...Next` loop (rather than two `For...Next` loops) to calculate the grand total. Save the solution and then start and test the application. (If the user enters 978128586026, the `btnAssign_Click` procedure should display 9781285860268.)

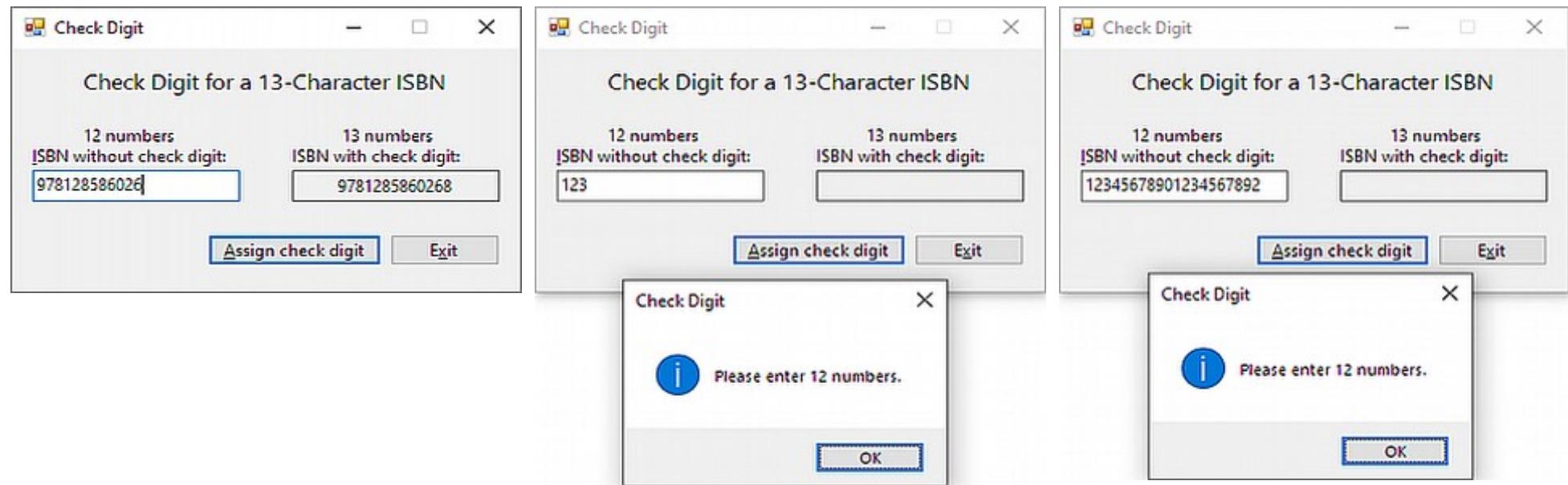
```
1  'modify the procedure to use: one For...Next loop rather than 2 For...Next loops to calculate the grand total
2  '->remove: - Dim declaration statements intTotalOdd & intTotalEven
3  '           - intGrandTotal = intTotalOdd + intTotalEven
4  '-> test: 978-1-285-86026 = 978-1-285-86026-8 <- adds the check digit 8
5  Option Explicit On
6  Option Strict On
7  Option Infer Off
8
9  Public Class frmMain
10     Private Sub btnAssign_Click(sender As Object, e As EventArgs) Handles btnAssign.Click
11         ' Assign a check digit to an ISBN.
12
13         Dim strIsbn As String
14         'Dim intTotalOdd As Integer
15         'Dim intTotalEven As Integer
16         Dim intOneStep As Integer
17         Dim strOddIndex As String
18         Dim intOddIndex As Integer
19         Dim strEvenIndex As String
20         Dim intEvenIndex As Integer
21
22         Dim intGrandTotal As Integer
23         Dim intRemainder As Integer
24         Dim intCheckDigit As Integer
25
```

```
26     If txtIsbn.Text.Length = 12 Then
27         strIsbn = txtIsbn.Text
28
29         For intIndex As Integer = 0 To 10 Step 2
30             strOddIndex = strIsbn(1 + intIndex)
31             Integer.TryParse(strOddIndex, intOddIndex)
32
33             strEvenIndex = strIsbn(intIndex)
34             Integer.TryParse(strEvenIndex, intEvenIndex)
35
36             intOneStep += (3 * intOddIndex) + intEvenIndex
37         Next intIndex
38
39         intGrandTotal += intOneStep
40
41         'For intOdd As Integer = 1 To 11 Step 2          ' original
42         'Integer.TryParse(strIsbn(intOdd), intDigit)      ' original
43         'intTotalOdd += (intDigit * 3)                     ' original
44         'Next intOdd                                     ' original
45
46         'For intEven As Integer = 0 To 10 Step 2          ' original
47         'Integer.TryParse(strIsbn(intEven), intDigit)      ' original
48         'intTotalEven += intDigit                         ' original
49         'Next intEven                                    ' original
50
51         'intGrandTotal = intTotalOdd + intTotalEven        ' original
52
53         intRemainder = intGrandTotal Mod 10
54
55         If intRemainder <> 0 Then
56             intCheckDigit = 10 - intRemainder
57         End If
58         lblFinalIsbn.Text = strIsbn & intCheckDigit.ToString
59
60     Else
61         MessageBox.Show("Please enter 12 numbers.", "Check Digit",
62                         MessageBoxButtons.OK, MessageBoxIcon.Information)
63     End If
64 End Sub
65
66 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
67     Me.Close()
68 End Sub
69
```

```

70      Private Sub txtIsbn_Enter(sender As Object, e As EventArgs) Handles txtIsbn.Enter
71          txtIsbn.SelectAll()
72      End Sub
73
74      Private Sub txtIsbn_TextChanged(sender As Object, e As EventArgs) Handles txtIsbn.TextChanged
75          lblFinalIsbn.Text = String.Empty
76      End Sub
77
78      Private Sub txtIsbn_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtIsbn.KeyPress
79          ' Allow only numbers and the Backspace key.
80
81          If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
82              e.Handled = True
83          End If
84      End Sub
85  End Class

```



- Const, txt.Text.Trim, str(Index), str.ToUpper, str.IndexOf,  
txt\_Enter, txt\_TextChanged

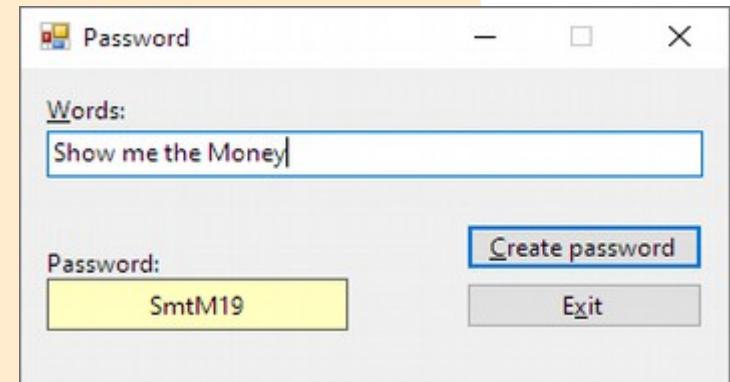
9. In this exercise, you modify the Password application from this chapter's Apply lesson. Use Windows to make a copy of the Password Solution folder. Rename the copy Password Solution-Index. Open the Password Solution.sln file contained in the Password Solution-Index folder. In the btnCreate\_Click procedure, declare a constant named strALPHABET and initialize it to the 26 uppercase letters of the alphabet. Then, rather than inserting the length of the password, the procedure should insert a number that represents the position of the password's first character in the alphabet. For example, if the first character in the password is the letter A, the procedure should insert the number 1. Similarly, if the first character is the letter Z, the procedure should insert the number 26. Insert the number after the last character in the password (rather than after the first character). Save the solution and then start and test the application. (If the user enters "show me the money", the Create password button should display smtm19.)

```
1  '->1). in btnCreate_Click declare a constant strALPHABET = 26 uppercase letters of the alphabet
2  '->2). then, rather than inserting the length of the password, insert a number that represents the position
3  '      of the password's 1st character in the alphabet
4  '      <-character array => string(Index As Integer) As Character
5  '      e.g. - if the 1st char in the password is the letter "A", insert the number 1
6  '              - if the 1st char in the password is the letter "Z", insert the number 26
7  '->3). rather than after the 1st char (index 1), insert the number after the last character in the password
8  '->test: "Show me the Money" = SmtM19
9  Option Explicit On
10 Option Strict On
11 Option Infer Off
12
13 Public Class frmMain
14     Private Sub btnCreate_Click(sender As Object, e As EventArgs) Handles btnCreate.Click
15         ' Create a password.
16         Dim strWords As String
17         Dim strPassword As String
18         Dim intSpaceIndex As Integer
19         'modifications:
20         Const strALPHABET As String = "ABCDEFGHIJKLMNOPQRSTUVWXYZ" ' ->1. 26 letters, index = 0 to 25
21         Dim strErsteLetter As String
22         Dim intErsteIndex As Integer
23
24         strWords = txtWords.Text.Trim
25
```

```

26     If strWords <> String.Empty Then
27         ' Assign the first character as the password:
28         strPassword = strWords(0)
29         strErsteLetter = strWords(0)                               'pick the first letter
30         strErsteLetter = strErsteLetter.ToUpper
31         intErsteIndex = strALPHABET.IndexOf(strErsteLetter)
32         intErsteIndex = intErsteIndex + 1
33         'MessageBox.Show(intErsteIndex.ToString)
34
35         ' Search for the first space in the input:
36         intSpaceIndex = strWords.IndexOf(" ")
37
38         Do Until intSpaceIndex = -1
39             ' Concatenate the character that follows the space to the password:
40             strPassword = strPassword & strWords(intSpaceIndex + 1)
41             ' Search for the next space:
42             intSpaceIndex = strWords.IndexOf(" ", intSpaceIndex + 1)
43         Loop
44
45         ' Insert the number after the first character:
46         'strPassword = strPassword.Insert(1, strPassword.Length.ToString)
47
48         ' Display the final password:
49         lblPassword.Text = strPassword & intErsteIndex.ToString
50     End If
51 End Sub
52
53 Private Sub txtWords_Enter(sender As Object, e As EventArgs) Handles txtWords.Enter
54     txtWords.SelectAll()
55 End Sub
56
57 Private Sub txtWords_TextChanged(sender As Object, e As EventArgs) Handles txtWords.TextChanged
58     lblPassword.Text = String.Empty
59 End Sub
60
61 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
62     Me.Close()
63 End Sub
64 End Class

```



## Chap07\Exercise1

### 26.Shipping Solution\_EXERCISE 10\_intermediate

```
- str.ToUpper, ElseIf, Like "##characters", lst.SelectedIndex, MessageBox.Show,  
frmMain_Load, lst.Items.Add, txt_Enter, txt_TextChanged
```

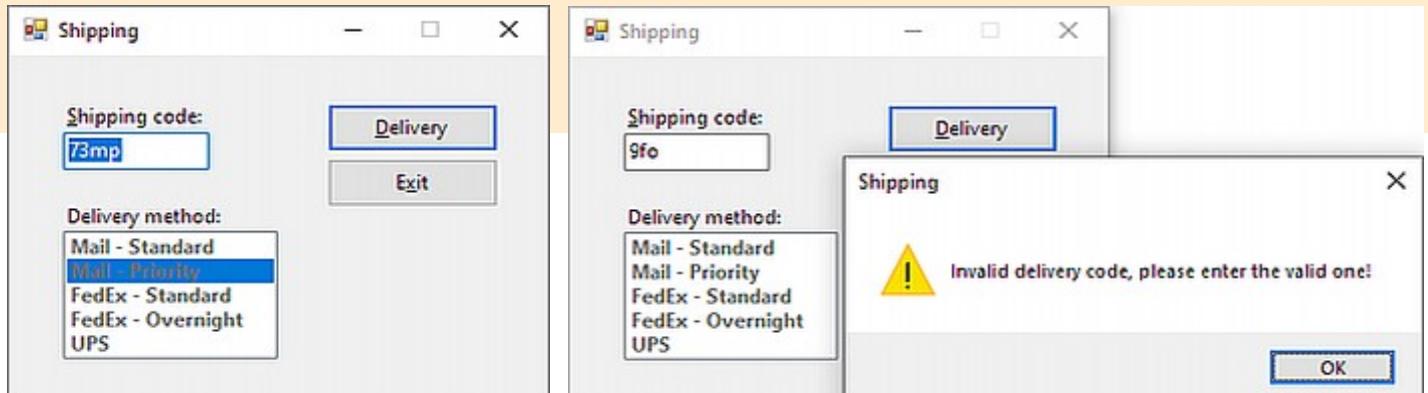
10. Open the Shipping Solution.sln file contained in the VB2017\Chap07\Shipping Solution folder. The interface provides a text box for entering a shipping code, which should consist of two numbers followed by either one or two letters. The letter(s) represent the delivery method, as follows: MS represents Mail – Standard, MP represents Mail – Priority, FS represents FedEx – Standard, FO represents FedEx – Overnight, and U represents UPS. The btnDelivery\_Click procedure should use the Like operator to determine the delivery method to select in the list box. For example, if the shipping code is 73mp, the procedure should select the Mail – Priority item in the list box. The procedure should display an appropriate message when the shipping code is not valid. Code the procedure. Save the solution and then start the application. Test the application using the following valid codes: 73mp, 34fs, 88FO, 12u, and 34ms. Then, test it using the following invalid codes: 9fo and 78hs.

```
1  '-(txtCode) for entering a shipping code: ##?? => 2x number & 1/2 letters  
2  '-?? => represents the delivery method: MS = Mail-Standard, MP = Mail-Priority, FS = FedEx-Standard, FO = FedEx-Overnight, U = UPS  
3  '->btnDelivery_Click should use the Like operator to determine the delivery method to select in the (lstDelivery)  
4  '-> e.g. if the shipping code is 73mp, the procedure should select the: "Mail - Priority" item in the (lstDelivery)  
5  '->the procedure should display an appropriate message when the shipping code is not valid  
6  '->Test it: valid codes = 73mp, 34fs, 88FO, 12u, 34ms. invalid codes = 9fo, 78hs  
7  
8  Option Explicit On  
9  Option Strict On  
10 Option Infer Off  
11  
12 Public Class frmMain  
13     Private Sub btnDelivery_Click(sender As Object, e As EventArgs) Handles btnDelivery.Click  
14         Dim strCode As String = txtCode.Text  
15         strCode = strCode.ToUpper  
16  
17         If strCode Like "##MS" Then  
18             lstDelivery.SelectedIndex = 0  
19             txtCode.SelectAll()  
20         ElseIf strCode Like "##MP" Then  
21             lstDelivery.SelectedIndex = 1  
22             txtCode.SelectAll()  
23         ElseIf strCode Like "##FS" Then  
24             lstDelivery.SelectedIndex = 2  
25             txtCode.SelectAll()  
26         ElseIf strCode Like "##FO" Then  
27             lstDelivery.SelectedIndex = 3  
28             txtCode.SelectAll()
```

```

29      ElseIf strCode Like "##U" Then
30          lstDelivery.SelectedIndex = 4
31          txtCode.SelectAll()
32      Else
33          MessageBox.Show("Invalid delivery code, please enter the valid one!", "Shipping",
34                         MessageBoxButtons.OK, MessageBoxIcon.Exclamation)
35          txtCode.Focus()
36          txtCode.SelectAll()
37      End If
38
39  End Sub
40
41  Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
42      ' Fills the list box with values.
43
44      lstDelivery.Items.Add("Mail - Standard")
45      lstDelivery.Items.Add("Mail - Priority")
46      lstDelivery.Items.Add("FedEx - Standard")
47      lstDelivery.Items.Add("FedEx - Overnight")
48      lstDelivery.Items.Add("UPS")
49  End Sub
50
51  Private Sub txtCode_Enter(sender As Object, e As EventArgs) Handles txtCode.Enter
52      txtCode.SelectAll()
53  End Sub
54
55  Private Sub txtCode_TextChanged(sender As Object, e As EventArgs) Handles txtCode.TextChanged
56      ' Clears the list box selection.
57      lstDelivery.Enabled = False
58      lstDelivery.SelectedIndex = -1
59  End Sub
60  Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
61      Me.Close()
62  End Sub
63
64 End Class

```



- txt.Text.Trim, str(index), str.IndexOf, Do Until...Loop, str.ToLower,  
 Like "[A-Z]" - range in uppercase, Like "[a-z]" - range in lowercase,  
 Mod, str.ToUpper, ElseIf, str.Insert, str.Length, txt\_TextChanged, txt\_Enter

11. In this exercise, you modify the Password application from this chapter's Apply lesson. Use Windows to make a copy of the Password Solution folder. Rename the copy Password Solution-Advanced. Open the Password Solution.sln file contained in the Password Solution-Advanced folder. Before inserting the number, the btnCreate\_Click procedure should alternate the case of each letter in the password. If the first character is lowercase, the procedure should change it to uppercase; it should then change the second letter to lowercase, the third letter to uppercase, and so on. For example, if the password is abcd, the procedure should change it to AbCd. On the other hand, if the first character is uppercase, the procedure should change it to lowercase and then alternate the case of the following letters. For example, if the password is Abcd, the procedure should change it to aBcD. Modify the procedure's code. Save the solution and then start and test the application. (If the user enters "May the Force be with you", the procedure should display m6TfBwY. If the user enters "may the Force be with you", the procedure should display M6tFbWy.)

```

1  'Modification:
2  '>before inserting the number, the btnCreate_Click procedure should alternate the case of each letter in the password
3  '  if the 1st char is lowercase, the procedure should change it to uppercase, it should then change the 2nd letter to lowercase...
4  '    e.g. if the password is "abcd", the procedure should change it to "AbCd"
5  '  on the other hand - if the 1st char is uppercase, the procedure should change it to lowercase and then
6  '      alternate the Case Of the following letters
7  '    e.g. if the password is "Abcd" the procedure should change it to "aBcD"
8  '>test: "May the Force be with you" => "m6TfBwY"
9  '      "may the Force be with you" => "M6tFbWy"
10
11 Option Explicit On
12 Option Strict On
13 Option Infer Off
14
15 Public Class frmMain
16     Private Sub btnCreate_Click(sender As Object, e As EventArgs) Handles btnCreate.Click
17         ' Create a password.
18
19         Dim strWords As String
20         Dim strPassword As String = String.Empty
21         Dim intSpaceIndex As Integer
22         Dim strTest As String = String.Empty
23         Dim strNewPassword As String = String.Empty
24         Dim strEven As String = String.Empty
25         Dim strOdd As String = String.Empty

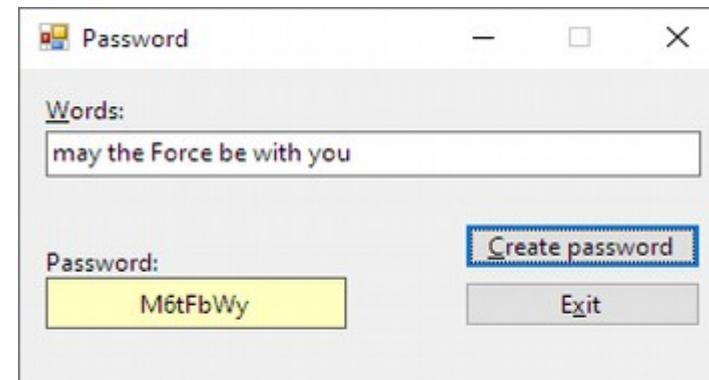
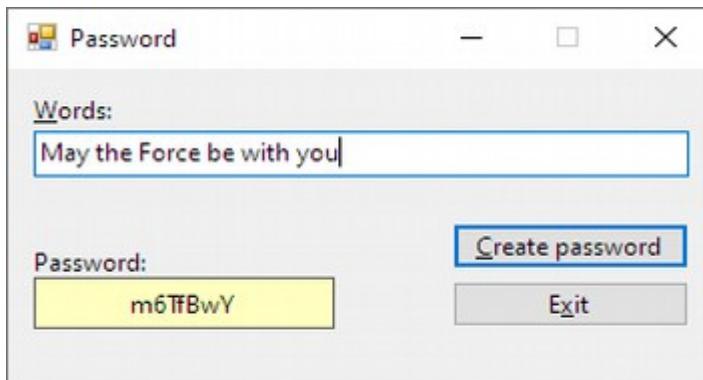
```

```
26  
27     strWords = txtWords.Text.Trim  
28  
29     If strWords <> String.Empty Then  
30  
31         strPassword = strWords(0)           ' Assign the first character as the password:  
32         intSpaceIndex = strWords.IndexOf(" ")    ' Search for the first space in the input  
33  
34         Do Until intSpaceIndex = -1  
35             ' Concatenate the character that follows the space to the password:  
36             strPassword = strPassword & strWords(intSpaceIndex + 1)  
37  
38             intSpaceIndex = strWords.IndexOf(" ", intSpaceIndex + 1)      ' Search for the next space  
39     Loop  
40  
41     ' MODIFICATION: alternate each letter in the password:  
42     For intIndex As Integer = 0 To strPassword.Length - 1  
43         strTest = strTest & intIndex.ToString  
44  
45         If strPassword(0) Like "[A-Z]" Then          'if the 1st(0) char is UPPERCASE  
46             If intIndex Mod 2 = 0 Then  
47                 strEven = strPassword(intIndex)  
48                 strEven = strEven.ToLower  
49                 strNewPassword += strEven  
50             ElseIf intIndex Mod 2 = 1 Then  
51                 strOdd = strPassword(intIndex)  
52                 strOdd = strOdd.ToUpper  
53                 strNewPassword += strOdd  
54             End If  
55  
56             ElseIf strPassword(0) Like "[a-z]" Then      'if the 1st(0) char is lowercase  
57                 If intIndex Mod 2 = 0 Then  
58                     strEven = strPassword(intIndex)  
59                     strEven = strEven.ToUpper  
60                     strNewPassword += strEven  
61                 ElseIf intIndex Mod 2 = 1 Then  
62                     strOdd = strPassword(intIndex)  
63                     strOdd = strOdd.ToLower  
64                     strNewPassword += strOdd  
65                 End If  
66             End If  
67             '0 Mod2 = 0; 1 Mod 2 = 1  
68     Next intIndex  
69
```

```

70     strPassword = strNewPassword
71
72     ' Insert the number after the first character:
73     strPassword = strPassword.Insert(1, strPassword.Length.ToString)
74
75     ' Display the final password:
76     lblPassword.Text = strPassword
77   End If
78 End Sub
79
80 Private Sub txtWords_Enter(sender As Object, e As EventArgs) Handles txtWords.Enter
81   txtWords.SelectAll()
82 End Sub
83
84 Private Sub txtWords_TextChanged(sender As Object, e As EventArgs) Handles txtWords.TextChanged
85   lblPassword.Text = String.Empty
86 End Sub
87
88 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
89   Me.Close()
90 End Sub
91 End Class

```



12. In this exercise, you modify the Proper Case application from Exercise 6. If necessary, complete Exercise 6. Then, use Windows to make a copy of the Proper Case Solution folder. Rename the copy Proper Case Solution-Middle. Open the Proper Case Solution.sln file contained in the Proper Case Solution-Middle folder. Modify the application to allow the user to also enter his or her middle name or middle initial. The btnProper\_Click procedure should display the full name, which might include a middle name or middle initial, in proper case. Save the solution and then start and test the application. (If the user enters "john smith" as the name, the application should display "John Smith". If the user enters "john thomas smith", the application should display "John Thomas Smith". If the user enters "carol g. jones", the application should display "Carol G. Jones". If the user enters two spaces, "pam", two spaces, "grace", three spaces, "darwin", and two spaces, the application should display "Pam Grace Darwin".)

- does not work properly, other version **28b** is super

## 22.Poor Case Solution\_EXERCISE 6\_intermediate

6. Open the Proper Case Solution.sln file contained in the VB2017\Chap07\Proper Case Solution folder. The interface provides a text box for entering a person's first and last names. The btnProper\_Click procedure should display the first and last names in the proper case. In other words, the first and last names should begin with an uppercase letter and the remaining letters in each name should be lowercase. If the user enters only one name, display the name in proper case. Be sure the btnProper\_Click procedure works correctly if the user inadvertently enters more than one space between the first and last names. After displaying the name, the procedure should send the focus to the txtName control. Code the procedure. Save the solution and then start and test the application. (If the user enters "john smith" as the name, the application should display "John Smith". If the user enters "carol" followed by three spaces and then "jones", the application should display "Carol Jones". If the user enters three spaces followed by "jack" and another three spaces, the application should display "Jack".)

```

1  'GUI provides a txt for entering a person's 1st and last names
2  '->btnProper_Click procedure should display the 1st and last names in the proper case = should begin with an UPPERCASE letter
3  '      followed by the lowercase letters
4  '-> if the user enters only one name, display the name in proper case
5  '-> be sure the procedure works correctly if the user inadvertently enters more than one space between the 1st and last names
6  '-> after displaying the name, the procedure should send the focus to the txtName control (Focus & SelectAll)
7  '->code the procedure, save and test:
8  '-> test: "john smith" => "John Smith"; "carol    jones" => "Carol Jones"; "    jack    " => "Jack"
9
10 'Middle - MODIFICATION:
11 'modify the app to allow the user to also enter middle name or middle initial
12 'btnProper_Click procedure should display the full name, which might include a middle name or middle initial, in proper case
13 'test: "thomas edison" => "Thomas Edison"; "thomas alva edison" => "Thomas Alva Edison"; "carol g. jones" => "Carol G. Jones"
14 'test: " pam grace darwin " => "Pam Grace Darwin"
15 ' WORKS FOR ANY NUMBER OF NAMES, BUT:
16 '      - ONLY WITH 1 SPACE CHAR BETWEEN THE NAMES &
17 '      - 1ST CHAR IN UPPER MAKES ANY OTHER SAME CHARS ALSO TO UPPER
18 ' NOT FINISHED, IM GONNA TRY OTHER VERSION
19
20 Option Explicit On
21 Option Strict On
22 Option Infer Off
23
24 Public Class frmMain
25     Private Sub btnProper_Click(sender As Object, e As EventArgs) Handles btnProper.Click
26         'txtName; lblName
27         Dim strName As String = txtName.Text.Trim      ' removes any leading and trailing spaces

```

```

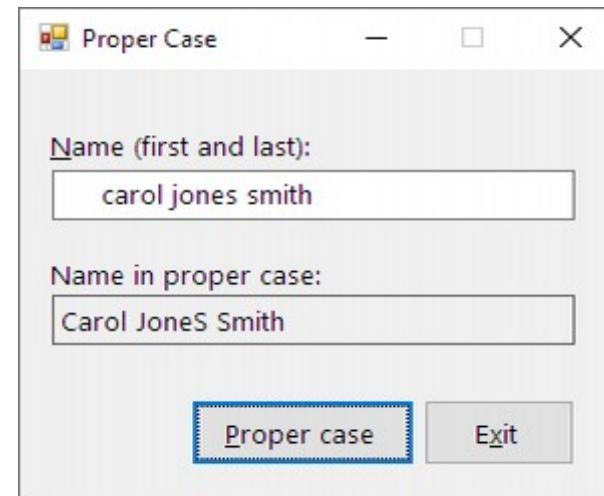
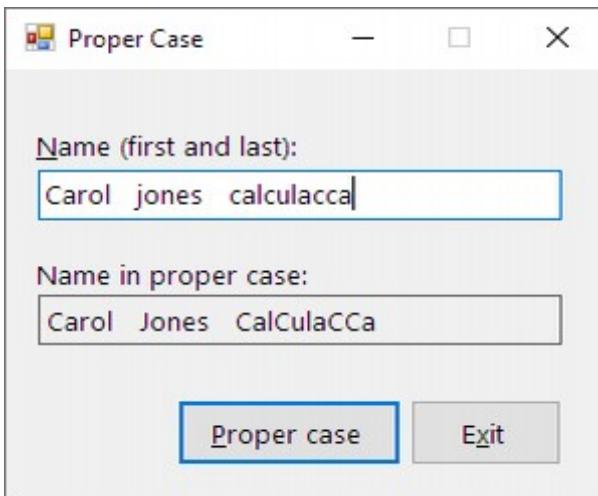
28     Dim strSingleName As String           ' used only for a single name
29     Dim charSpace As String = " "
30     Dim intSpace As Integer
31     Dim strFirstChar As String = String.Empty
32     Dim strFirstCharOld As String = String.Empty
33     Dim strFirstNameChar As String
34     Dim strFirstName As String
35     Dim intX As Integer
36
37     Dim CharsSpace As String = " "
38
39     If strName <> String.Empty Then          ' if there is any character entered:
40         If strName.Contains(charSpace) = False Then      ' no have Space char => if there is only 1 word:
41             strSingleName = strName(0)                      ' 1st character what will be TOUPPER
42             lblName.Text = strSingleName.ToUpper & strName.Substring(1)    ' 1st char TOUPPER & the rest of the word normal
43             txtName.Focus()
44             txtName.SelectAll()
45         Else   ' my playground => if there is any Space char:
46
47             ' inadvertent spaces between the words solution => ?erase any additional spaces in the beginning?
48             ' solution for only 1 space character between the names:
49
50             'For CountChar As Integer = 2 To 20
51             'CharsSpace = charSpace
52             'Next CountChar
53             'strName = strName.Replace(charSpace & " ", charSpace)
54             'strName = strName.Replace(" ", charSpace)
55
56             intSpace = strName.IndexOf(charSpace)
57             strFirstNameChar = strName(0)
58             strFirstNameChar = strFirstNameChar.ToUpper
59             strFirstName = strName.Substring(1, intSpace - 1)
60
61             Do Until intSpace = -1
62                 ' what should happen when the Index of (charSpace) is located:
63                 strFirstCharOld = strName(intSpace + 1)
64                 strFirstChar = strName(intSpace + 1)                  ' takes the 1st char after the (charSpace)
65                 strFirstChar = strFirstChar.ToUpper
66                 strName = strName.Replace(strFirstCharOld, strFirstChar)
67
68                 intSpace = strName.IndexOf(charSpace, intSpace + 1)      ' Search for the next space
69                 strFirstChar = String.Empty
70                 strFirstCharOld = String.Empty
71             Loop

```

```

72
73     intX = strName.IndexOf(charSpace)
74     strName = strName.Substring(intX + 1)
75     lblName.Text = strFirstNameChar & " " & strName
76
77     End If
78 Else      ' if there is no character entered:
79     MessageBox.Show("Please enter a name", "Proper Case Application", MessageBoxButtons.OK,
80                         MessageBoxIcon.Exclamation)
81     txtName.Focus()
82 End If
83 End Sub
84
85 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
86     Me.Close()
87 End Sub
88
89 Private Sub txtName_Enter(sender As Object, e As EventArgs) Handles txtName.Enter
90     txtName.SelectAll()
91 End Sub
92
93 Private Sub txtName_TextChanged(sender As Object, e As EventArgs) Handles txtName.TextChanged
94     lblName.Text = String.Empty
95 End Sub
96
97 End Class

```



- works for max 3 words

- str.IndexOf, str(index), str.Substring, str.Trim, str.ToUpper,  
txt\_Enter, txt\_TextChanged

```

1  'GUI provides a txt for entering a person's 1st and last names
2  ' btnProper_Click procedure should display the 1st and last names in the proper case, i.e:
3  '   should begin with an UPPERCASE letter followed by the lowercase letters
4  ' if the user enters only one name, display the name in proper case
5  ' be sure the procedure works correctly if the user inadvertently enters more than one space between the 1st and last names
6  ' after displaying the name, the procedure should send the focus to the txtName control (Focus & SelectAll)
7  ' code the procedure, save and test:
8  '-> test: "john smith" => "John Smith"; "carol jones" => "Carol Jones"; " jack " => "Jack"
9
10 'Middle - MODIFICATION:
11 ' modify the app to allow the user to also enter middle name or middle initial
12 ' btnProper_Click procedure should display the full name, which might include a middle name or middle initial, in proper case
13 ' test: "thomas a. edison" => "Thomas A. Edison"; " pam grace darwin " => "Pam Grace Darwin"
14
15 Option Explicit On
16 Option Strict On
17 Option Infer Off
18
19 Public Class frmMain
20     Private Sub btnProper_Click(sender As Object, e As EventArgs) Handles btnProper.Click
21         'txtName; lblName
22         Dim strName As String = txtName.Text.Trim           ' removes any leading and trailing spaces
23         Dim intSpaceIndex As Integer                      ' Index of the space for the 2nd name
24         Dim str1a As String                                ' 1st word, 1st character will be ToUpper
25         Dim str1b As String = String.Empty                ' 1st word, 2nd->end of the word
26         ' int1c = loop for str1b
27         Dim str2 As String                                ' 2nd word = everything after the 1st Space char
28         Dim str2a As String                              ' 2nd word, 1st character will be ToUpper
29         Dim str2b As String = String.Empty                ' 2nd word, 2nd->end of the word
30         ' int2c = loop for str2b
31         Dim str3 As String                                ' 3rd word = everything after the 2nd Space char
32         Dim str3a As String                              ' 3rd word, 1st character will be ToUpper
33         Dim str3b As String                ' 3rd word, 2nd->end of the word
34
35         intSpaceIndex = strName.IndexOf(" ", 1)          ' find 1st Index for the Space character
36
37         str1a = strName(0)                                ' 1st word: 1st char

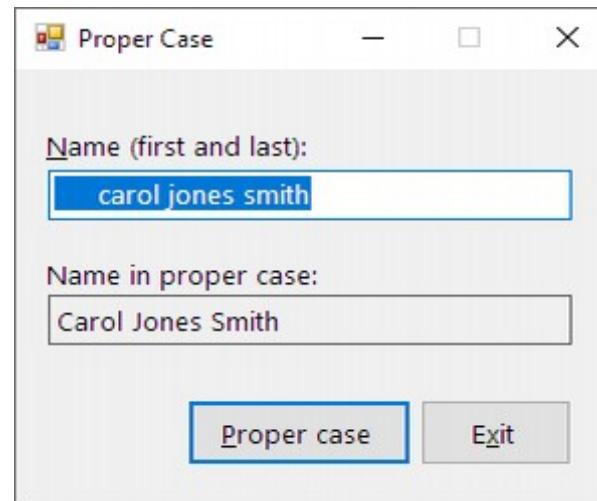
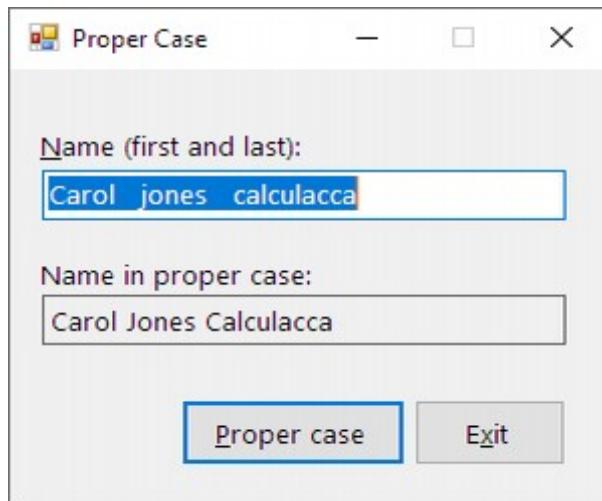
```

```

38
39     If intSpaceIndex <> -1 Then           ' if there is more words:
40
41         For Int1c As Integer = 1 To intSpaceIndex - 1   ' 1st word: 2nd char->end of the word
42             str1b += strName(Int1c)                      ' 1st word: 2nd char->end of the word
43         Next Int1c                                     ' 1st word: 2nd char->end of the word
44
45         str2 = strName.Substring(intSpaceIndex + 1)      ' 2nd word: everything after the 1st Space char
46         str2 = str2.Trim                                ' 2nd word: Trim Space characters
47         str2a = str2(0)                                 ' 2nd word: 1st character only
48         intSpaceIndex = str2.IndexOf(" ", 1)           ' 2nd word: NEW Space Index -> IMPORTANT
49
50     If intSpaceIndex <> -1 Then           ' if there is a 3rd word:
51
52         For int2c As Integer = 1 To intSpaceIndex - 1   ' 2nd word: 2nd char->end of the word
53             str2b += str2(int2c)                        ' 2nd word: 2nd char->end of the word
54         Next int2c                                    ' 2nd word: 2nd char->end of the word
55
56         str3 = str2.Substring(intSpaceIndex + 1)      ' 3rd word: everything after the 1st Space char
57         str3 = str3.Trim                            ' 3rd word: Trim Space characters
58         str3a = str3(0)                             ' 3rd word: 1st character only
59         str3b = str3.Substring(1)                   ' 3rd word: 2nd->rest of the word
60
61         lblName.Text = str1a.ToUpper & str1b & " " & str2a.ToUpper & str2b & " " & str3a.ToUpper & str3b
62         txtName.Focus()
63         txtName.SelectAll()
64
65     Else   ' if there is not a 3rd word:
66
67         str2b = str2.Substring(1)
68         lblName.Text = str1a.ToUpper & str1b & " " & str2a.ToUpper & str2b
69         txtName.Focus()
70         txtName.SelectAll()
71     End If
72
73     Else   ' if there is not more words:
74
75         lblName.Text = str1a.ToUpper & strName.Substring(1)
76         txtName.Focus()
77         txtName.SelectAll()
78     End If
79 End Sub
80
81 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
82     Me.Close()
83 End Sub

```

```
82  
83     Private Sub txtName_Enter(sender As Object, e As EventArgs) Handles txtName.Enter  
84         txtName.SelectAll()  
85     End Sub  
86  
87     Private Sub txtName_TextChanged(sender As Object, e As EventArgs) Handles txtName.TextChanged  
88         lblName.Text = String.Empty  
89     End Sub  
90  
91 End Class
```



## Chap07\Exercise1

### 29.Poor Case Solution-Hyphenated\_EXERCISE 13\_Advanced

- txt.Text.Trim, str(index), str.IndexOf, str.Substring, str.Trim,  
str.ToUpper, str.TrimEnd, txt\_Enter, txt\_TextChanged

13. In this exercise, you modify the Proper Case application from Exercise 6. If necessary, complete Exercise 6. Then, use Windows to make a copy of the Proper Case Solution folder. Rename the copy Proper Case Solution-Hyphenated. Open the Proper Case Solution.sln file contained in the Proper Case Solution-Hyphenated folder. Start the application. Type "carol mason-smith" (without the quotes) as the name and then click the Proper case button. The btnProper\_Click procedure displays "Carol Mason-smith". Click the Exit button. Modify the procedure to display hyphenated names in proper case; for example, the procedure should display "Carol Mason-Smith". Save the solution and then start and test the application. Be sure the application works correctly if the user inadvertently enters "carol mason-" (notice the hyphen at the end of the entry).

### 22.Poor Case Solution\_EXERCISE 6\_intermediate

6. Open the Proper Case Solution.sln file contained in the VB2017\Chap07\Proper Case Solution folder. The interface provides a text box for entering a person's first and last names. The btnProper\_Click procedure should display the first and last names in the proper case. In other words, the first and last names should begin with an uppercase letter and the remaining letters in each name should be lowercase. If the user enters only one name, display the name in proper case. Be sure the btnProper\_Click procedure works correctly if the user inadvertently enters more than one space between the first and last names. After displaying the name, the procedure should send the focus to the txtName control. Code the procedure. Save the solution and then start and test the application. (If the user enters "john smith" as the name, the application should display "John Smith". If the user enters "carol" followed by three spaces and then "jones", the application should display "Carol Jones". If the user enters three spaces followed by "jack" and another three spaces, the application should display "Jack".)

```
1  ' GUI provides a txt for entering a person's 1st and last names
2  ' MOD: Proper Case Solution-Hyphenated modification:
3  ' MOD:-> type "carol mason-smith" => the procedure displays: "Carol Mason-smith", so modify te procedure to
4  ' display hyphenated names is proper case: "Carol Mason-Smith"
5  ' MOD:-> be sure the application works correctly if the user inadvertently enters "carol mason-".
6  Option Explicit On
7  Option Strict On
8  Option Infer Off
9
10 Public Class frmMain
11     Private Sub btnProper_Click(sender As Object, e As EventArgs) Handles btnProper.Click
12         'txtName; lblName
13         Dim strName As String = txtName.Text.Trim           ' removes any leading and trailing spaces
14         Dim intSpaceIndex As Integer                      ' Index of the space for the 2nd name
15         Dim str1a As String
16         Dim str1b As String = String.Empty
17         Dim str2 As String
18         Dim str2a As String
19         Dim str2b As String = String.Empty
20         Dim intHyphen As Integer      ' MOD
21         Dim str3 As String
22         Dim str3a As String
23         Dim str3b As String
```

```

24
25     str1a = strName(0)                                ' 1st name: 1st character
26
27     intSpaceIndex = strName.IndexOf(" ", 1)           ' find the 1st index for the Space character
28
29     If intSpaceIndex <> -1 Then                      ' If there are 2 names:
30
31         For Int1c As Integer = 1 To intSpaceIndex - 1   ' 1st name: 2nd->rest of the name until Space character
32             str1b += strName(Int1c)                      ' 1st name: 2nd->rest of the name until Space character
33         Next Int1c                                     ' 1st name: 2nd->rest of the name until Space character
34
35         str2 = strName.Substring(intSpaceIndex + 1)      ' 2nd name: everything after the 1st Space character
36         str2 = str2.Trim                                ' 2nd name: Trim any additional Space characters
37         intHyphen = str2.IndexOf("-", 1)                 ' 2nd name: find the 1st Index of the Hyphen
38         str2a = str2(0)                                 ' 2nd name: 1st Index
39
40         If intHyphen <> -1 Then
41
42             For int2c As Integer = 1 To intHyphen - 1    ' 2nd name: 2nd->rest of the name until Hyphen character_MOD
43                 str2b += str2(int2c)                      ' 2nd name: 2nd->rest of the name until Hyphen character_MOD
44             Next int2c                                  ' 2nd name: 2nd->rest of the name until Hyphen character_MOD
45             str2b = str2.Substring(1)                   ' 2nd name: 2nd->rest of the word
46
47             str3 = str2.Substring(intHyphen + 1)
48             str3 = str3.Trim
49
50             If str3 = String.Empty Then                ' 3rd name: if there is nothing after the hyphen:
51                 lblName.Text = str1a.ToUpper & str1b & " " & str2a.ToUpper & str2b
52                 txtName.Focus()
53                 txtName.SelectAll()
54             Else                                      ' 3rd name: if there is something after the hyphen:
55                 str3a = str3(0)
56                 str3b = str3.Substring(1)
57                 str2b = str2b.TrimEnd                  ' fix additional Space characters
58                 lblName.Text = str1a.ToUpper & str1b & " " & str2a.ToUpper & str2b & "-" & str3a.ToUpper & str3b
59                 txtName.Focus()
60                 txtName.SelectAll()
61             End If
62         Else
63             str2b = str2.Substring(1)
64             lblName.Text = str1a.ToUpper & str1b & " " & str2a.ToUpper & str2b
65             txtName.Focus()
66             txtName.SelectAll()
67         End If

```

```

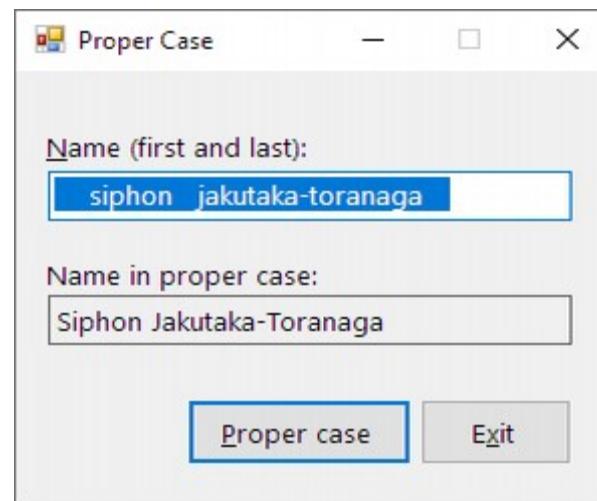
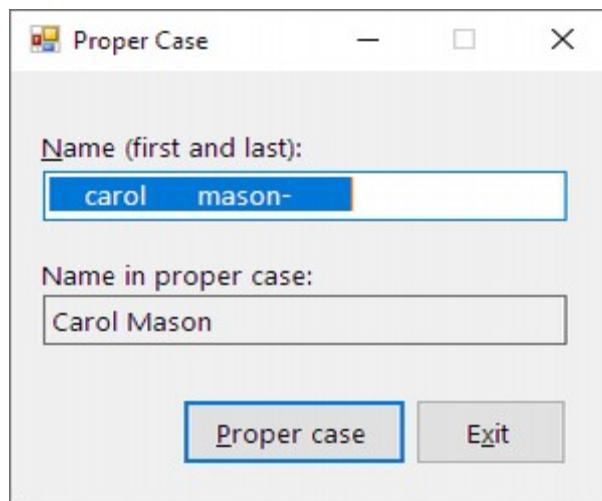
68     Else                                ' If there is only 1 name:
69         lblName.Text = str1a.ToUpper & strName.Substring(1)
70         txtName.Focus()
71         txtName.SelectAll()
72     End If
73 End Sub

74
75 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
76     Me.Close()
77 End Sub

78
79 Private Sub txtName_Enter(sender As Object, e As EventArgs) Handles txtName.Enter
80     txtName.SelectAll()
81 End Sub

82
83 Private Sub txtName_TextChanged(sender As Object, e As EventArgs) Handles txtName.TextChanged
84     lblName.Text = String.Empty
85 End Sub
86
87 End Class

```



- **Private** used as accumulator, **str.ToUpper**, **Select Case**, **str.Contains**, **MessageBox.Show**, **Like "###[characters][characters]"** - any 3 numbers + 2 defined characters, **txt\_KeyPress**, **ControlChars**, **Private Sub** - independent Sub procedure

14. Each salesperson at Rembrandt Auto-Mart is assigned an ID number that consists of five characters. The first three characters are numbers. The fourth character is a letter: either the letter N if the salesperson sells new cars or the letter U if the salesperson sells used cars. The fifth character is also a letter: either the letter F if the salesperson is a full-time employee or the letter P if the salesperson is a part-time employee. Create a Windows Forms application. Use the following names for the project and solution, respectively: Rembrandt Project and Rembrandt Solution. Save the application in the VB2017\Chap07 folder. Create the interface shown in Figure 7-55. Make the Calculate button the default button. The application should allow the sales manager to enter the ID and the number of cars sold for as many salespeople as needed. The **btnCalc\_Click** procedure should display the total number of cars sold by each of the following four categories of employees: full-time employees, part-time employees, employees selling new cars, and employees selling used cars. Code the application. Save the solution and then start and test the application.

```

1 Option Explicit On
2 Option Strict On
3 Option Infer Off
4 ' each salesperson is assigned a 5-character ID number: ?-?-?-N/U-F/P
5 ' 1-3 = some numbers
6 ' 4 = N/U : N = salesperson sells New cars, U = salesperson sells Used cars
7 ' 5 = F/P : F = full-time employee, P = part-time employee
8 ' the app should allow the sales manager to enter the ID and the number of cars sold for
9 ' as many salespeople as needed
10 ' display the total number of cars sold by each of the 4 categories
11 ' btnCalculate, btnClear, btnExit, txt1ID, txt2Sold
12 ' lbl1FullTime, lbl2PartTime, lbl3NewCar, lbl4UsedCar
13 Public Class frmMain
14     Private int1FullTime As Integer      ' F =lbl1FullTime
15     Private int2PartTime As Integer      ' P =lbl2PartTime
16     Private int3NewCar As Integer        ' N =lbl3NewCar
17     Private int4UsedCar As Integer       ' U =lbl4UsedCar
18
19     Private Sub btnCalculate_Click(sender As Object, e As EventArgs) Handles btnCalculate.Click
20         Dim str1ID As String = txt1ID.Text           ' =txt1ID (MaxLength = 5)
21         Dim int2Sold As Integer                    ' =txt2Sold ->only numbers: Private Sub txt2Sold_KeyPress...
22         Integer.TryParse(txt2Sold.Text, int2Sold)

```

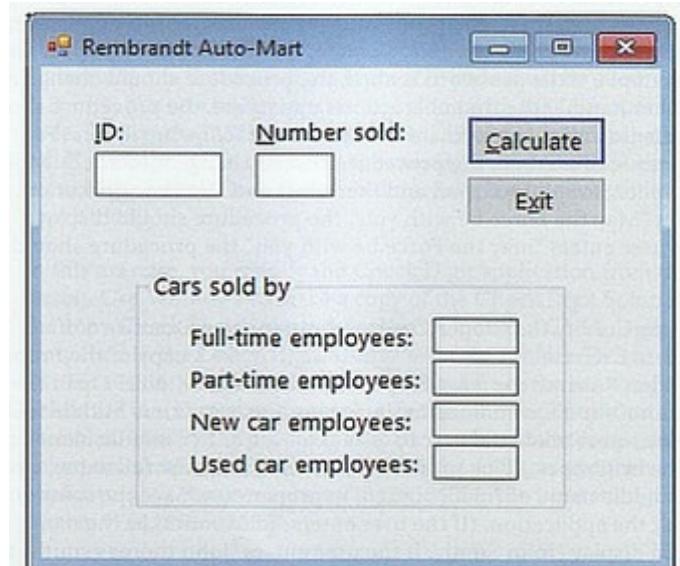


Figure 7-55 Interface for Exercise 14

```
23
24     If str1ID.ToUpper Like "###[NU][FP]" Then ' if the user entry matches the predefined conditions:
25
26         Select Case True
27             Case str1ID.ToUpper.Contains("N")
28                 int3NewCar += int2Sold
29             Case str1ID.ToUpper.Contains("U")
30                 int4UsedCar += int2Sold
31         End Select
32
33         Select Case True
34             Case str1ID.ToUpper.Contains("F")
35                 int1FullTime += int2Sold
36             Case str1ID.ToUpper.Contains("P")
37                 int2PartTime += int2Sold
38         End Select
39
40         lbl1FullTime.Text = int1FullTime.ToString
41         lbl2PartTime.Text = int2PartTime.ToString
42         lbl3NewCar.Text = int3NewCar.ToString
43         lbl4UsedCar.Text = int4UsedCar.ToString
44         txt1ID.Focus()
45         txt1ID.SelectAll()
46
47     Else      ' if the user entry won't match the predefined conditions:
48         MessageBox.Show("Please enter the valid ID number", "Rembrandt Auto-Mart",
49                         MessageBoxButtons.OK, MessageBoxIcon.Exclamation)
50         txt1ID.Focus()
51         txt1ID.SelectAll()
52     End If
53 End Sub
54
55 Private Sub btnClear_Click(sender As Object, e As EventArgs) Handles btnClear.Click
56     int1FullTime = 0
57     int2PartTime = 0
58     int3NewCar = 0
59     int4UsedCar = 0
60     txt1ID.Text = String.Empty
61     txt2Sold.Text = String.Empty
62     lbl1FullTime.Text = String.Empty
63     lbl2PartTime.Text = String.Empty
64     lbl3NewCar.Text = String.Empty
65     lbl4UsedCar.Text = String.Empty
66 End Sub
```

```

67
68     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
69         Me.Close()
70     End Sub
71
72     Private Sub txt2Sold_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txt2Sold.KeyPress
73         If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
74             e.Handled = True
75         End If
76     End Sub
77
78     Private Sub Ent(sender As Object, e As EventArgs) Handles txt1ID.Enter, txt2Sold.Enter
79         txt1ID.SelectAll()
80         txt2Sold.SelectAll()
81     End Sub
82 End Class

```

The image displays three screenshots of a Windows application window titled "Rembrandt Auto-Mart". The window contains a label "Please enter:" followed by two text boxes: "Employee ID:" and "Number sold:". To the right of these are two buttons: "Calculate" and "Clear All". Below the text boxes is a message box titled "Cars sold by" containing four text boxes for "Full-time employees", "Part-time employees", "New car employees", and "Used car employees".

- Screenshot 1:** The "Employee ID:" text box contains "123nj". A modal dialog box titled "Rembrandt Auto-Mart" with a yellow warning icon shows the message "Please enter the valid ID number".
- Screenshot 2:** The "Employee ID:" text box now contains "123nl". The modal dialog box is no longer visible.
- Screenshot 3:** The "Employee ID:" text box now contains "123np". The modal dialog box is no longer visible.

- **Private** used as global **Random integer**, **Private Sub** - independent **Sub** procedure, **btn.Enabled**, **randGen As New Random**, **randGen.Next**, **MessageBox.Show**, **txt\_KeyPress**, **ControlChars**

15. Open the Addition Solution.sln file contained in the VB2017\Chap07\Addition Solution folder. The btnNew\_Click procedure is responsible for generating two random integers from 0 to 10 (including 10) and displaying them in the lblNum1 and lblNum2 controls. The btnCheck\_Click procedure is responsible for determining whether the user's answer, which is entered in the txtAnswer control, is correct. If the answer is correct, the procedure should display an appropriate message and then clear the problem and answer from the interface. If the answer is not correct, the procedure should display an appropriate message and then allow the user to answer the addition problem again. Code both procedures. Include any other code that will professionalize the interface. Save the solution and then start and test the application.

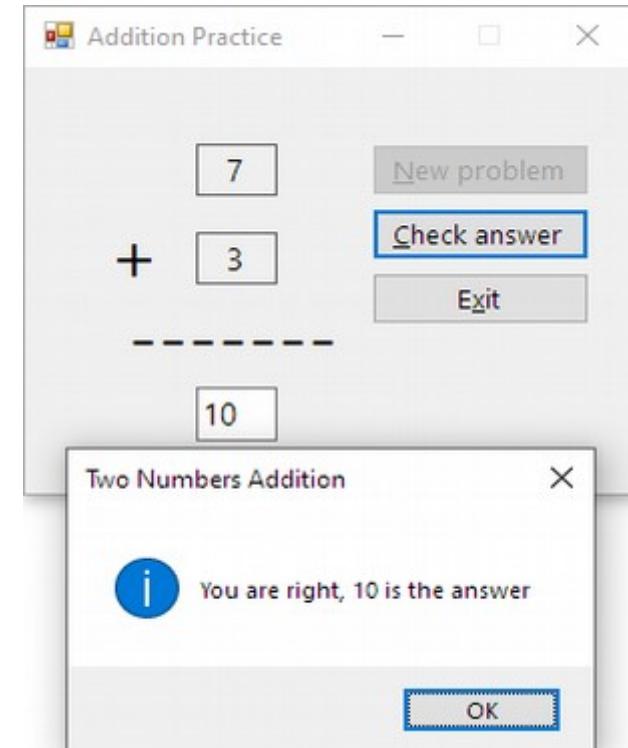
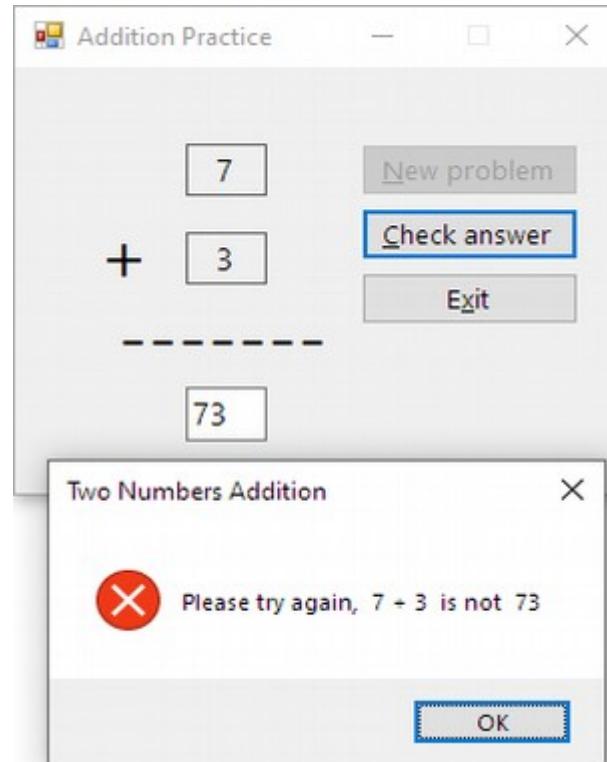
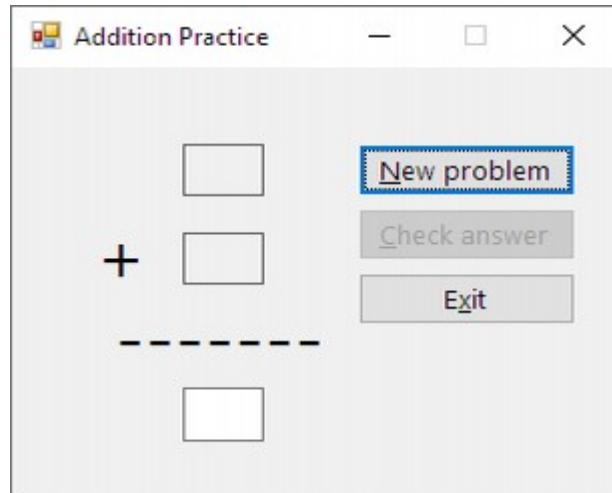
```

1 Option Explicit On
2 Option Strict On
3 Option Infer Off
4
5 ' btnNew_Click procedure is responsible for generating two random integers from 0 to 10 included,
6 ' and displaying them in the lblNum1 and lblNum2 controls
7
8 ' user enters his answer in: txtAnswer control
9
10 ' btnCheck_Click procedure is responsible for determining whether the user's answer is correct
11 ' - if correct, then display an appropriate message and clear problem and answer from GUI
12 ' - if not correct, then display an appropriate message and then allow the user to answer the
13 ' addition problem again
14
15 ' code both procedures and include any other code that will professionalize GUI
16 ' txtAnswer, lblNum1, lblNum2
17
18 Public Class frmMain
19     Private intNum1 As Integer      ' 1st global Random integer, lblNum1
20     Private intNum2 As Integer      ' 2nd global Random integer, lblNum2
21     Private intNum3 As Integer      ' 1st + 2nd to check with the user's answer
22
23     Private Sub RandomInteger()      ' i have used an Independent Sub Procedure to generate a random integer
24         Dim randGen As New Random
25         intNum1 = randGen.Next(0, 11)   ' random integer between 0-10
26         intNum2 = randGen.Next(0, 11)   ' other random integer between 0-10
27         ' the procedure ONLY say "Hi, I am an Independent Procedure, do what you desire"
28     End Sub
29

```

```
30  Private Sub btnNew_Click(sender As Object, e As EventArgs) Handles btnNew.Click
31
32      RandomInteger()           ' helooo, waaaake uuuuup RandomInteger and give me all your values
33
34      intNum3 = intNum1 + intNum2
35
36      lblNum1.Text = intNum1.ToString
37      lblNum2.Text = intNum2.ToString
38      'MessageBox.Show(intNum3.ToString)      ' shows the addition of 2 independent random integers
39
40      btnNew.Enabled = False
41      btnCheck.Enabled = True
42      txtAnswer.Focus()
43 End Sub
44
45 Private Sub btnCheck_Click(sender As Object, e As EventArgs) Handles btnCheck.Click
46     ' txtAnswer = user's input
47     Dim intAnswer As Integer
48     Integer.TryParse(txtAnswer.Text, intAnswer)
49
50     If intNum3 = intAnswer Then
51         MessageBox.Show("You are right, " & intAnswer.ToString & " is the answer", "Two Numbers Addition",
52                         MessageBoxButtons.OK, MessageBoxIcon.Information)
53     btnNew.Enabled = True
54     btnCheck.Enabled = False
55     lblNum1.Text = String.Empty
56     lblNum2.Text = String.Empty
57     txtAnswer.Text = String.Empty
58     btnNew.Focus()
59 Else
60     MessageBox.Show("Please try again, " & lblNum1.Text & " + " & lblNum2.Text & " is not " &
61                     intAnswer.ToString, "Two Numbers Addition", MessageBoxButtons.OK, MessageBoxIcon.Error)
62     txtAnswer.SelectAll()
63 End If
64 End Sub
65
66 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
67     Me.Close()
68 End Sub
69
70 Private Sub txtAnswer_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtAnswer.KeyPress
71     If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
72         e.Handled = True
73     End If
```

```
74      End Sub  
75  
76  End Class
```



```
- str.Length, For...Step...Next, MessageBox.Show, Mod, txt_TextChanged,
  txt_KeyPress, ControlChars, frmMain_FormClosing, dlgButton As DialogResult,
  dlgButton = MessageBox.Show, DialogResult.No, e.Cancel
```

16. Open the Validate Number Solution.sln file contained in the VB2017\Chap07\Validate Number Solution folder. The interface provides a text box for entering a 9-digit number. The btnValidate\_Click procedure should use the algorithm and example shown in Figure 7-56 to validate the user's entry. The procedure should display a message indicating whether the entry is or is not valid. Code the procedure. Include any other code that will professionalize the interface. Save the solution and then start and test the application.

#### Algorithm

- Starting with the second digit, multiply every other digit by 2. (These will be the second, fourth, sixth, and eighth digits.)
- If a product from Step 1 is greater than 9, sum the two digits in the product. For example, if the product is 12, add the 1 to the 2, giving 3.
- Add together the results from Steps 1 and 2 and each of the digits skipped in Step 1. (The skipped digits will be the first, third, fifth, seventh, and ninth digits.)
- Divide the sum from Step 3 by 10 and find the remainder. If the remainder is 0, then the number is valid.

Number: 631620176

|         |          |   |   |             |   |          |   |             |   |                          |
|---------|----------|---|---|-------------|---|----------|---|-------------|---|--------------------------|
| Step 1: | 6        | 3 | 1 | 6           | 2 | 0        | 1 | 7           | 6 |                          |
|         | *        |   |   | *           |   | *        |   | *           |   |                          |
|         | 2        |   |   | 2           |   | 2        |   | 2           |   |                          |
|         | <u>6</u> |   |   | <u>12</u>   |   | <u>0</u> |   | <u>14</u>   |   |                          |
| Step 2: |          |   |   | $1 + 2 = 3$ |   |          |   | $1 + 4 = 5$ |   |                          |
| Step 3: | 6        | 6 | 1 | 3           | 2 | 0        | 1 | 5           | 6 | 30                       |
| Step 4: |          |   |   |             |   |          |   |             |   | $30 \text{ Mod } 10 = 0$ |

the 0 indicates that  
the number is valid

```

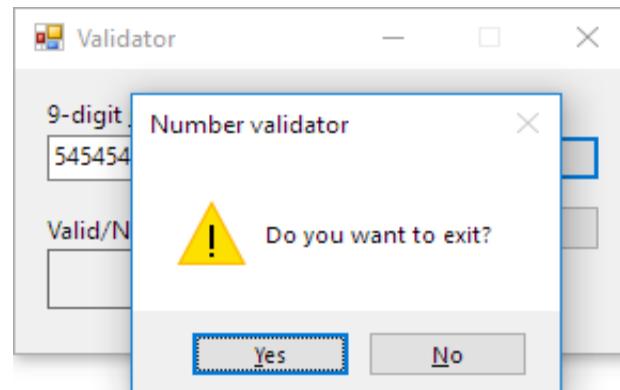
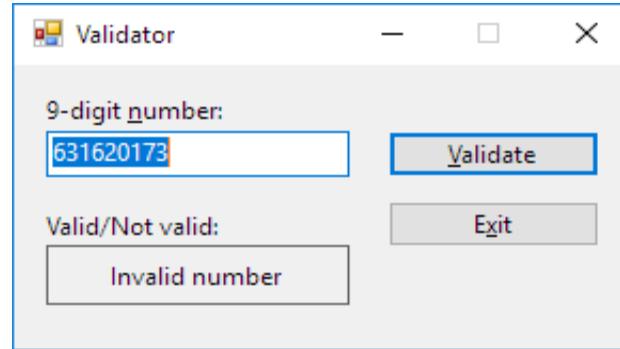
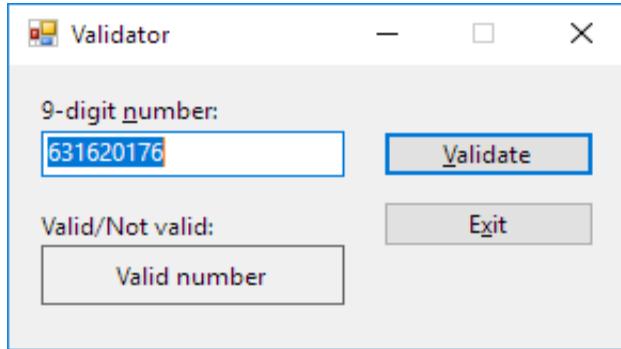
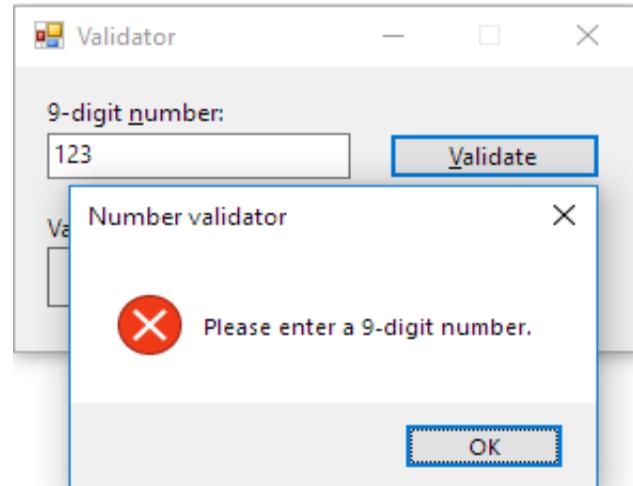
1  '1. every ODD digit (even index) add together: 6+1+2+1+6 ==16
2  '2. every EVEN digit (odd index) multiply by 2: 3*2=6, 6*2=12, 0*2=0, 7*2=14           =6  (=32)
3  '2b. if a product from step 2. is greater than 9, sum the two digits: 12:1+2=3, 14:1+4=5  =8  (==14)
4  '3. add together the results from step 1. and step 2: 16+14=30
5  '4. divide the sum from step 3 by 10 and find the remainder: 30 Mod 10
6  '5. if the remainder is 0, then the number is valid
7  ' txtNumber (MaxLength property = 9), lblStatus
8  ' 631620176
9  Option Explicit On
10 Option Strict On
11 Option Infer Off
12
13 Public Class frmMain
14     Private Sub btnValidate_Click(sender As Object, e As EventArgs) Handles btnValidate.Click
15         Dim strNumber As String = txtNumber.Text
16         Dim intDigit As Integer          ' will be used as a single digit picked from both: even index & odd index
  
```

```

17     Dim intTwoDigits As Integer      ' 2b. 2 digits
18     Dim intSumOdd As Integer        ' 1. will add together all ODD digits (even indexes)
19     Dim intSumEven As Integer       ' 2. EVEN digits *2, but if 2 digits then add them together
20     Dim intTotalSum As Integer      ' 3. results from 1. & 2. add together 16+14=30
21     Dim intRemainder As Integer     ' 4. intTotalSum Mod 10
22
23     If strNumber.Length = 9 Then    ' user entry must have 9 digits to continue
24
25         For intEvenIndex As Integer = 0 To 8 Step 2          ' 1.even indexes 0-2-4-6-8
26             ' 1.from strNumber pick character with index of intEvenIndex and transform it to a number named intDigit
27             Integer.TryParse(strNumber(intEvenIndex), intDigit)
28             intSumOdd += intDigit                                ' 1.odd numbers
29         Next intEvenIndex                                     ' 1.loop
30         'MessageBox.Show(intSumOdd.ToString)                   ' 1. 6+1+2+1+6 = 16 .ok
31
32         For intOddIndex As Integer = 1 To 7 Step 2           ' 2.odd indexes
33             ' 2.from strNumber pick character with index of intOddIndex and transform it to a number named intDigit
34             Integer.TryParse(strNumber(intOddIndex), intDigit)
35             'MessageBox.Show(intDigit.ToString)                  ' 2. 3, 6, 0, 7 .ok
36             intDigit = intDigit * 2
37
38             If intDigit > 9 Then                               ' 2. if intDigit has a 2 digits
39                 'MessageBox.Show(intMultiplied.ToString)        ' 2. 12, 14 .ok
40                 intTwoDigits = (intDigit - 10) + 1
41                 'MessageBox.Show(intTwoDigits.ToString)        ' 2. 3, 5 .ok
42                 intSumEven += intTwoDigits
43             Else
44                 intSumEven += intDigit
45             End If
46
47         Next intOddIndex
48         'MessageBox.Show(intSumEven.ToString)                  ' 2. =14 .ok
49
50         intTotalSum = intSumOdd + intSumEven                ' 3.
51         'MessageBox.Show(intTotalSum.ToString)                  ' 3. =30 .ok
52
53         intRemainder = intTotalSum Mod 10                  ' 4. =0 .ok
54         'MessageBox.Show(intRemainder.ToString)
55
56         If intRemainder = 0 Then
57             lblStatus.Text = "Valid number"
58             txtNumber.Focus()
59             txtNumber.SelectAll()
60

```

```
61      Else
62          lblStatus.Text = "Invalid number"
63          txtNumber.Focus()
64          txtNumber.SelectAll()
65      End If
66
67      Else
68          MessageBox.Show("Please enter a 9-digit number.", "Number validator",
69                          MessageBoxButtons.OK, MessageBoxIcon.Error)
70          txtNumber.Focus()
71          txtNumber.SelectAll()
72          lblStatus.Text = String.Empty
73      End If
74
75  End Sub
76
77  Private Sub txtNumber_TextChanged(sender As Object, e As EventArgs) Handles txtNumber.TextChanged
78      lblStatus.Text = String.Empty
79  End Sub
80
81  Private Sub txtNumber_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtNumber.KeyPress
82      If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
83          e.Handled = True
84      End If
85  End Sub
86
87  Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
88      Me.Close()
89  End Sub
90
91
92  Private Sub frmMain_FormClosing(sender As Object, e As FormClosingEventArgs) Handles Me.FormClosing
93      ' Verify that the user wants to kill the application.
94
95      Dim dlgButton As DialogResult
96      dlgButton = MessageBox.Show("Do you want to exit?", "Number validator",
97                                  MessageBoxButtons.YesNo, MessageBoxIcon.Exclamation)
98
99      ' If the "No" button is selected, do not close the form:
100     If dlgButton = DialogResult.No Then
101         e.Cancel = True
102     End If
103  End Sub
104
105 End Class
```



## Chap07\Exercise1

### 33.OnYourOwn Solution\_EXERCISE 17\_NOT FINISHED

- not done, postponed for müza time

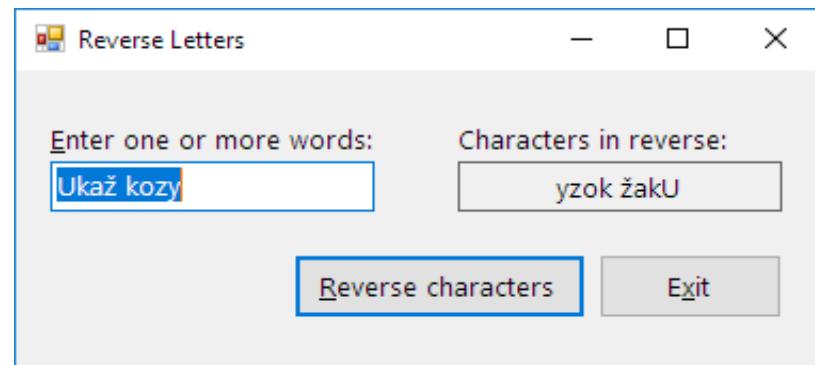
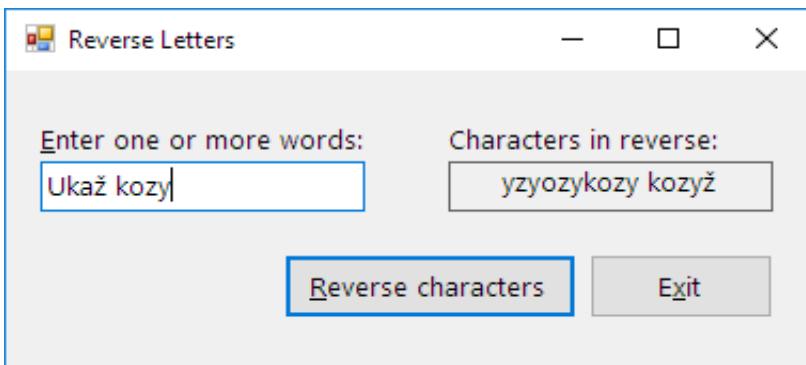
17. Create a Windows Forms application. Use the following names for the project and solution, respectively: OnYourOwn Project and OnYourOwn Solution. Save the application in the VB2017\Chap07 folder. Plan and design an application of your choice. The only requirement is that you must follow the minimum guidelines listed in Figure 7-57. Before starting the application, be sure to verify the name of the startup form. Save the solution and then start and test the application.

1. The user interface must contain a minimum of one text box, three labels, and two buttons. One of the buttons must be an Exit button.
2. The interface can include a picture box, but this is not a requirement.
3. The interface must follow the GUI design guidelines summarized for Chapters 2 through 6 in Appendix A.
4. Objects that are either coded or referred to in code should be named appropriately.
5. The Code Editor window must contain comments, the three Option statements, at least two variables, at least two assignment statements, at least two of the concepts covered in the Focus lesson, and the Me.Close() statement.
6. Every text box on the form should have its TextChanged and Enter event procedures coded. At least one of the text boxes should have its KeyPress event procedure coded.

18. Open the VB2017\Chap07\FixIt Solution\FixIt Solution.sln file. The interface provides a text box for the user to enter one or more words. The btnReverse\_Click procedure should display the characters in reverse order. In other words, if the user enters the words "Show me the money", the procedure should display "yenom eht em wohS". Start and test the application. Notice that the application is not working properly. Correct the application's code.

```
1  ' ...VB2017\Chap07\_Exercise\34.FixIt Solution_EXERCISE 18\
2  ' GUI provides a (txtWord) for the user to enter one or more words
3  ' btnReverse_Click procedure should display the characters in reverse order
4  ' e.g. "Show me the money" should be "yenom eht em wohS"
5  ' notice that the application is not working properly, so correct the code please.
6
7  Option Explicit On
8  Option Strict On
9  Option Infer Off
10
11 Public Class frmMain
12     Private Sub btnReverse_Click(sender As Object, e As EventArgs) Handles btnReverse.Click
13
14         lblReverse.Text = String.Empty                                '<- fix : erases previous results
15         ' Displays the characters in reverse order.
16
17         ' Dim strWord As String   '- original #1
18         'strWord = txtWord.Text.Trim                                     '- original #2
19         Dim strWord As String = txtWord.Text.Trim                      '<- fix #1#2 - not really needed, though
20         Dim strReverse As String                                       '<- fix #4
21
22         'For intIndex As Integer = strWord.Length To 0 Step -1        <- original #3
23         For intIndex As Integer = strWord.Length - 1 To 0 Step -1      '<- fix #3 : Length vs Index
24
25             'lblReverse.Text = lblReverse.Text & strWord.Substring(intIndex)    '<- original #4
26             strReverse = strWord(intIndex)                                     '<- fix #4
27             lblReverse.Text = lblReverse.Text & strReverse                     '<- fix #4
28
29             Next intIndex
30             txtWord.Focus()   '<- fix
31             txtWord.SelectAll()
32     End Sub
```

```
33
34     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
35         Me.Close()
36     End Sub
37
38     Private Sub txtWord_Enter(sender As Object, e As EventArgs) Handles txtWord.Enter
39         txtWord.SelectAll()
40     End Sub
41
42     Private Sub txtWord_TextChanged(sender As Object, e As EventArgs) Handles txtWord.TextChanged
43         lblReverse.Text = String.Empty
44     End Sub
45 End Class
```



## High Total Game (Chapters 1–7)

The High Total game requires two players. The application's interface should allow the user to enter each player's name. When the user clicks a button, the button's Click event procedure should generate two random numbers for player 1 and two random numbers for player 2. The random numbers should be in the range of 1 through 20, inclusive. The procedure should display the four numbers in the interface. It should also total the numbers for each player and then display both totals in the interface. If both totals are the same, the application should display the message "Tie". If player 1's total is greater than player 2's total, it should display the message "*player 1's name* won". If player 2's total is greater than player 1's total, it should display the message "*player 2's name* won". The application should keep track of the number of times player 1 wins, the number of times player 2 wins, and the number of ties. The interface should also include a button that allows the user to reset the counters and interface for a new game.

```
1  ' Name:      High Total Game.
2  ' Purpose:    Game for 2 players, where the higher sum of 2 random numbers wins.
3  ' Programmer: Me on just now.
4  Option Explicit On
5  Option Strict On
6  Option Infer Off
7
8  Public Class frmMain
9      ' total each player's 2 random integers (1-20 inclusive)
10     ' compare totals and display results, either: "[player 1's/2's name] won" / "Tie"
11     ' keep track of winners and ties
12     ' include btn to Reset counters and GUI for a new game
13
14     ' Class-level variables used as Counters:
15     Private intP1Wins As Integer : Private intP2Wins As Integer : Private intTies12 As Integer
16
17     Private Sub btnGenerate_Click(sender As Object, e As EventArgs) Handles btnGenerate.Click
18         ' txtP1Name, lblP1one, lblP1two, lblP1Sum, lblP1Wins
19         ' txtP2Name, lblP2one, lblP2two, lblP2Sum, lblP2Wins
20         ' lblTies1, lblTies2
21         ' lblResult
22
23         lblResult.Text = Nothing
24
```

```

25     ' 4 random numbers:
26     Dim intNum1 As Integer : Dim intNum2 As Integer : Dim intNum3 As Integer : Dim intNum4 As Integer : Dim randGen As New Random
27     intNum1 = randGen.Next(1, 21) : intNum2 = randGen.Next(1, 21) : intNum3 = randGen.Next(1, 21) : intNum4 = randGen.Next(1, 21)
28
29     ' Sum:
30     Dim intP1Sum As Integer = intNum1 + intNum2 : Dim intP2Sum As Integer = intNum3 + intNum4
31
32     ' s.s. to determine the winner:
33     Select Case True
34         Case intP1Sum > intP2Sum
35             lblResult.Text = txtP1Name.Text & " won"
36             intP1Wins += 1
37         Case intP1Sum < intP2Sum
38             lblResult.Text = txtP2Name.Text & " won"
39             intP2Wins += 1
40         Case Else
41             lblResult.Text = "Tie"
42             intTies12 += 1
43     End Select
44
45     lblP1one.Text = intNum1.ToString : lblP1two.Text = intNum2.ToString : lblP1Sum.Text = intP1Sum.ToString
46     lblP2one.Text = intNum3.ToString : lblP2two.Text = intNum4.ToString : lblP2Sum.Text = intP2Sum.ToString
47     lblP1Wins.Text = intP1Wins.ToString : lblP2Wins.Text = intP2Wins.ToString
48     lblTies1.Text = intTies12.ToString : lblTies2.Text = intTies12.ToString
49 End Sub
50
51 Private Sub btnReset_Click(sender As Object, e As EventArgs) Handles btnReset.Click
52     ' clear Counters and Labels:
53     txtP1Name.Text = Nothing : txtP2Name.Text = Nothing
54     lblP1one.Text = Nothing : lblP1two.Text = Nothing : lblP1Sum.Text = Nothing : intP1Wins = 0 : lblP1Wins.Text = Nothing
55     lblP2one.Text = Nothing : lblP2two.Text = Nothing : lblP2Sum.Text = Nothing : intP2Wins = 0 : lblP2Wins.Text = Nothing
56     lblResult.Text = Nothing : intTies12 = 0 : lblTies1.Text = Nothing : lblTies2.Text = Nothing
57 End Sub
58
59 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
60     Me.Close()
61 End Sub
62 End Class

```

06.High Total Game

1st players name: Chulio Mario Fernando

2nd players name: Yo

**Generate 2 random numbers for each player**

Player 1      Player 2

|    |   |    |
|----|---|----|
| 9  | + | 12 |
| =  |   |    |
| 21 |   |    |

|    |   |   |
|----|---|---|
| 7  | + | 5 |
| =  |   |   |
| 12 |   |   |

**Chulio Mario Fernando won**

Total wins: 1

Ties: 0

Total wins: 0

Ties: 0

**Reset**   **Exit**

06.High Total Game

1st players name: Chulio Mario Fernando

2nd players name: Yo

**Generate 2 random numbers for each player**

Player 1      Player 2

|    |   |   |
|----|---|---|
| 12 | + | 7 |
| =  |   |   |
| 19 |   |   |

|    |   |   |
|----|---|---|
| 18 | + | 1 |
| =  |   |   |
| 19 |   |   |

**Tie**

Total wins: 7

Ties: 1

Total wins: 3

Ties: 1

**Reset**   **Exit**

## Math Practice (Chapters 1–7)

Create an application that can be used to practice adding, subtracting, multiplying, and dividing numbers. The application should display a math problem on the screen and then allow the student to enter the answer and also verify that the answer is correct. The application should give the student as many chances as necessary to answer the problem correctly. The math problems should use random integers from 1 through 20, inclusive. The subtraction problems should never ask the student to subtract a larger number from a smaller one. The division problems should never ask the student to divide a smaller number by a larger number. Also, the answer to the division problems should always result in a whole number. The application should keep track of the number of correct and incorrect responses made by the student. The interface should include a button that allows the user to reset the counters for a different student.

```
1  ' Name:      Math Practice.
2  ' Purpose:
3  ' Programmer: Me on just now.
4  ' lbl1RanNum1 = intRanNum1, lbl3RanNum2 = intRanNum2
5  ' Counters: lbl1Correct = int1Correct, lbl2Wrong = int2Wrong
6  Option Explicit On
7  Option Strict On
8  Option Infer Off
9
10 Public Class frmMain
11     ' Global variables:
12     Private intRanNum1 As Integer : Private intRanNum2 As Integer
13     Private dblResult As Double : Private intAnswer As Integer
14     Private charSign As Char
15     Private int1Correct As Integer : Private int2Wrong As Integer    ' counters
16
17     Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load    ' ok
18         cboProblem.Items.Add("Adding")      ' + index = 0    charSign = "+"c
19         cboProblem.Items.Add("Subtracting") ' - index = 1
20         cboProblem.Items.Add("Multiplying") ' * index = 2
21         cboProblem.Items.Add("Dividing")    ' / index = 3
22         cboProblem.SelectedIndex = 0
23         lbl1Correct.Text = "0" : lbl2Wrong.Text = "0"
24     End Sub
```

```

25
26     Private Sub btnVerify_Click(sender As Object, e As EventArgs) Handles btnVerify.Click      ' Compare input and Calculation:
27         Integer.TryParse(txtAnswer.Text, intAnswer)      ' user input
28
29     If txtAnswer.Text <> String.Empty Then
30
31         If intAnswer <> dblResult Then
32             int2Wrong += 1      ' Update Counter + 1
33             txtAnswer.Focus()
34             txtAnswer.SelectAll()
35             MessageBox.Show(" Wrong!      " & intRanNum1.ToString & " " & charSign & " " & intRanNum2.ToString & " ≠ " &
36                             intAnswer.ToString, "Math Practice", MessageBoxButtons.OK, MessageBoxIcon.Exclamation)
37         Else
38             int1Correct += 1      ' Update Counter +1
39             MessageBox.Show(" Correct!      " & intRanNum1.ToString & " " & charSign & " " & intRanNum2.ToString & " = " &
40                             intAnswer.ToString, "Math Practice", MessageBoxButtons.OK, MessageBoxIcon.Information)
41             RandomNumbers()      ' Call the Independent Sub: 2x Random Number Generator
42         End If
43     Else
44         txtAnswer.Focus()
45         MessageBox.Show("Please enter your answer.", "Math Practice", MessageBoxButtons.OK, MessageBoxIcon.Exclamation)
46     End If
47
48     ' output counters:
49     lbl1Correct.Text = int1Correct.ToString
50     lbl2Wrong.Text = int2Wrong.ToString
51 End Sub
52
53     Private Sub cboProblem_SelectedIndexChanged(sender As Object, e As EventArgs) Handles cboProblem.SelectedIndexChanged
54         Select Case True
55             Case cboProblem.SelectedIndex = 0
56                 lbl2Sign.Text = "+"
57             Case cboProblem.SelectedIndex = 1
58                 lbl2Sign.Text = "-"
59             Case cboProblem.SelectedIndex = 2
60                 lbl2Sign.Text = "*"
61             Case cboProblem.SelectedIndex = 3
62                 lbl2Sign.Text = "/"
63         End Select
64         RandomNumbers()      ' Call the Independent Sub: 2x Random Number Generator
65     End Sub
66

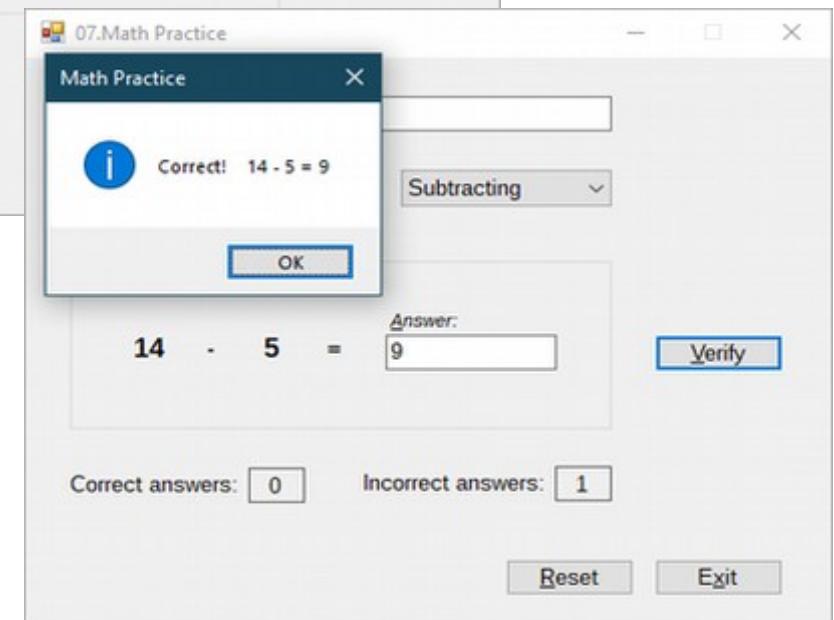
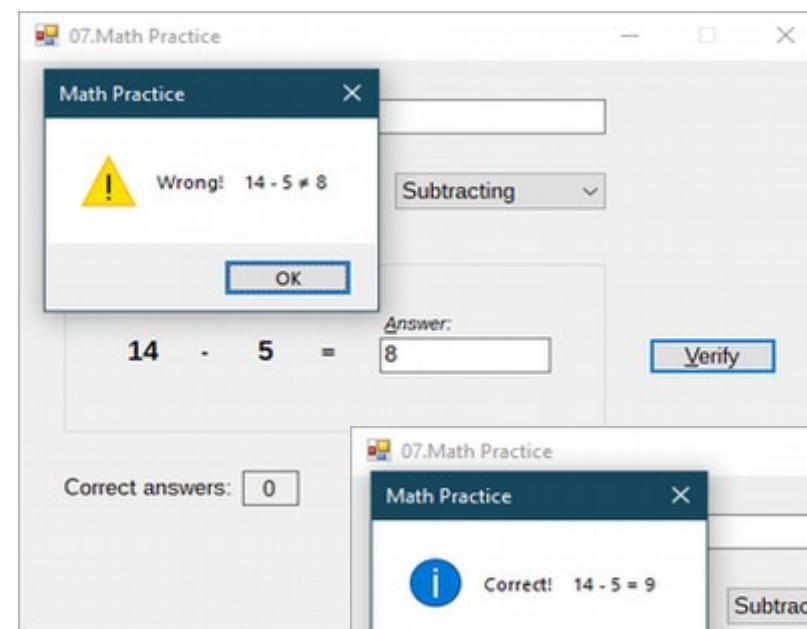
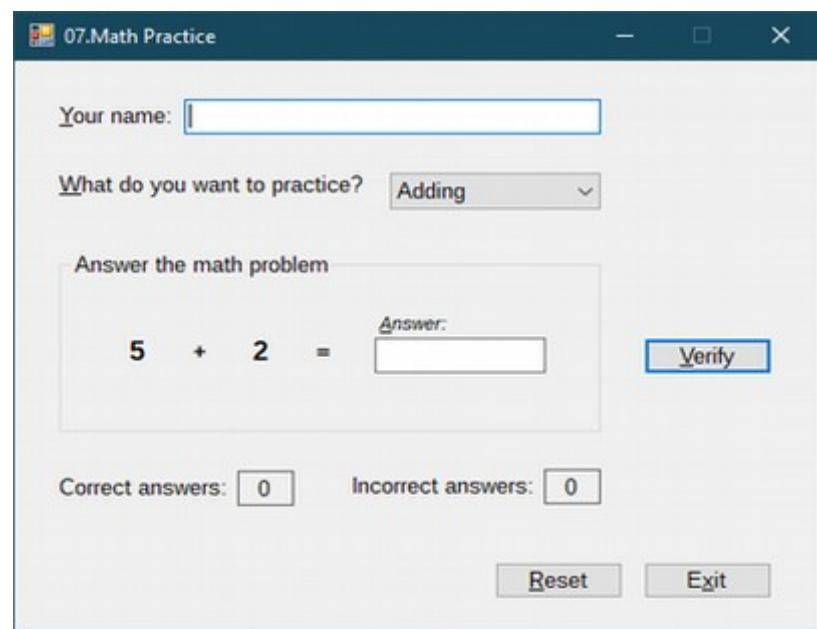
```

```
67     Private Sub RandomNumbers()    ' Independent Sub: 2x Random Number Generator
68
69         lbl1RanNum1.Text = Nothing : lbl3RanNum2.Text = Nothing : txtAnswer.Text = Nothing
70
71         Dim ranGen As New Random : intRanNum1 = ranGen.Next(1, 21) : intRanNum2 = ranGen.Next(1, 21)
72         Dim intTemp As Integer
73
74         ' Swap numbers, so the 1st is always bigger:
75         If intRanNum1 < intRanNum2 Then
76             intTemp = intRanNum1 : intRanNum1 = intRanNum2 : intRanNum2 = intTemp
77         End If
78
79         Select Case True
80             Case cboProblem.SelectedIndex = 0      ' ok
81                 charSign = "+"c
82                 dblResult = intRanNum1 + intRanNum2
83
84             Case cboProblem.SelectedIndex = 1
85                 charSign = "-"c
86                 dblResult = intRanNum1 - intRanNum2
87
88             Case cboProblem.SelectedIndex = 2
89                 charSign = "*"c
90                 dblResult = intRanNum1 * intRanNum2
91
92             Case cboProblem.SelectedIndex = 3
93                 charSign = "/"c
94                 If intRanNum1 Mod intRanNum2 <> 0 Then
95                     RandomNumbers()    ' Call the Independent Sub: 2x Random Number Generator
96                 Else
97                     dblResult = intRanNum1 / intRanNum2
98                 End If
99             End Select
100
101         lbl1RanNum1.Text = intRanNum1.ToString : lbl3RanNum2.Text = intRanNum2.ToString : txtAnswer.Focus()
102     End Sub
103
104     Private Sub btnReset_Click(sender As Object, e As EventArgs) Handles btnReset.Click
105         ' Clear everything:
106         txtName.Text = Nothing : txtAnswer.Text = Nothing
107         int1Correct = Nothing : lbl1Correct.Text = Nothing
108         int2Wrong = Nothing : lbl2Wrong.Text = Nothing
109         txtName.Focus()
110     End Sub
```

```

111
112     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click    'ok
113         Me.Close()
114     End Sub
115
116     Private Sub txtAnswer_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtAnswer.KeyPress    'ok
117         ' allow the user to enter only integers and use a Backspace key:
118         If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
119             e.Handled = True
120         End If
121     End Sub
122 End Class

```



## Tax-Deductible Calculator (Chapters 1–7)

Create an interface that provides text boxes for entering the following business expenses: lodging, travel, meals, and entertainment. Lodging and travel are 100% tax deductible; meals and entertainment are only 50% tax deductible. The application should calculate and display the total

expenses, the amount that is tax deductible, and the percentage that is tax deductible. Each text box should accept only numbers, one period, and the Backspace key. (Notice that each text box should not allow the user to enter more than one period.)

```
1  ' Name:      Tax-Deductible Calculator.
2  ' Purpose:    Calculate and display business expenses.
3  ' Programmer: me on today.
4  Option Explicit On
5  Option Strict On
6  Option Infer Off
7
8  Public Class frmMain
9      Private Sub btnCalculate_Click(sender As Object, e As EventArgs) Handles btnCalculate.Click
10         ' inputs: txt1Lodging = dec1Lodging, txt2Travel = dec2Travel, txt3Meals = dec3Meals, txt4Entertainment = dec4Entertainment
11         Dim dec1Lodging As Decimal : Dim dec2Travel As Decimal : Dim dec3Meals As Decimal : Dim dec4Entertainment As Decimal
12         Decimal.TryParse(txt1Lodging.Text, dec1Lodging) : Decimal.TryParse(txt2Travel.Text, dec2Travel)
13         Decimal.TryParse(txt3Meals.Text, dec3Meals) : Decimal.TryParse(txt4Entertainment.Text, dec4Entertainment)
14
15         ' outputs: lbl5Total, lbl6DeduAmount, lbl7DeduPercent
16         Dim dec5Total As Decimal : Dim dec6DeduAmount As Decimal : Dim dec7DeduPercent As Decimal
17
18         ' math:
19         dec5Total = dec1Lodging + dec2Travel + dec3Meals + dec4Entertainment
20         dec6DeduAmount = dec1Lodging + dec2Travel + (dec3Meals / 2) + (dec4Entertainment / 2)
21         dec7DeduPercent = dec6DeduAmount / dec5Total
22
```

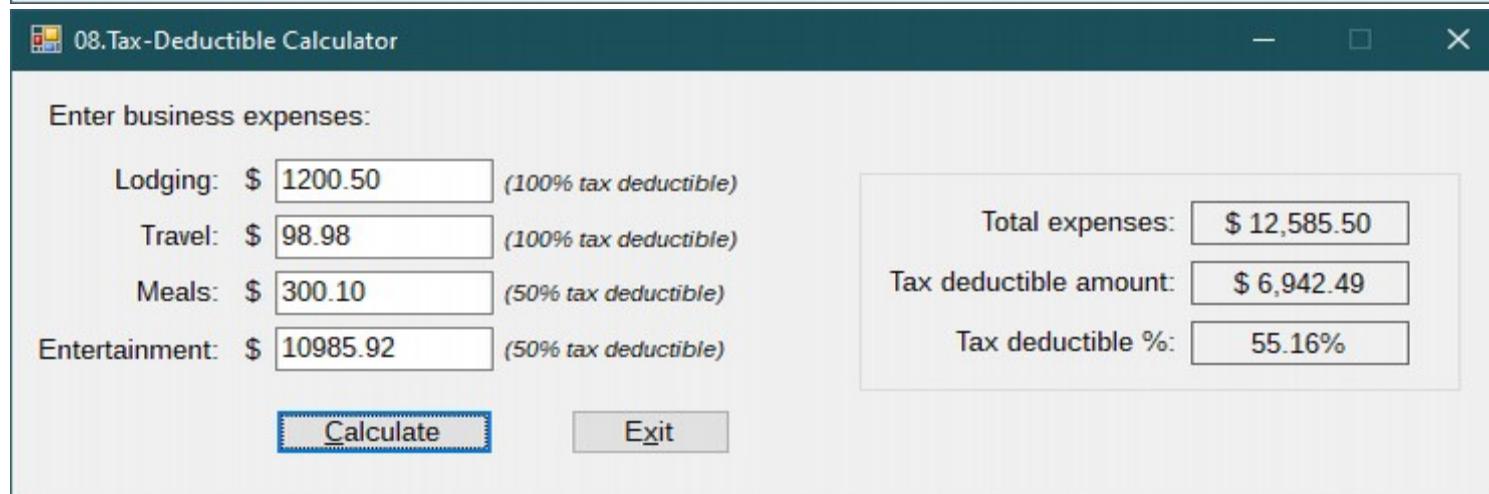
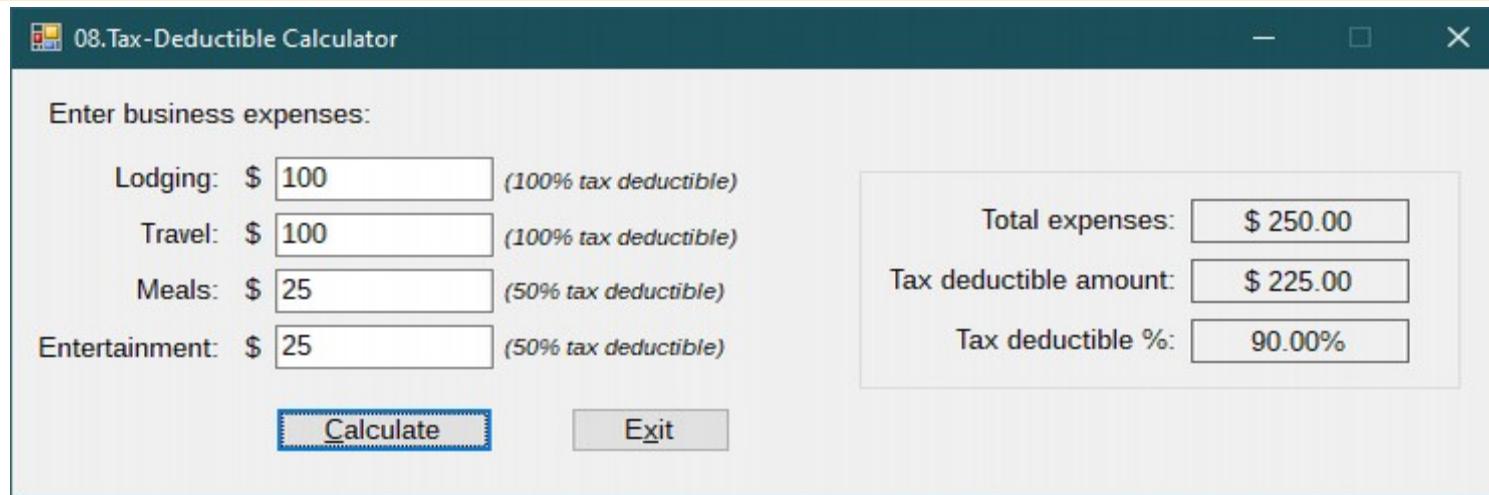
```
23     ' display calculations:
24     lbl5Total.Text = dec5Total.ToString("C2") : lbl6DeduAmount.Text = dec6DeduAmount.ToString("C2")
25     lbl7DeduPercent.Text = dec7DeduPercent.ToString("P2")
26 End Sub
27
28 Private Sub txt1Lodging_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txt1Lodging.KeyPress
29     ' allow only numbers, 1x decimal point, and Backspace key (each txt should not allow the user to enter more than 1 period):
30     If txt1Lodging.Text Like "*.*" Then
31         If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
32             e.Handled = True
33         End If
34     Else
35         If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back AndAlso e.KeyChar <> "." Then
36             e.Handled = True
37         End If
38     End If
39 End Sub
40
41 Private Sub txt2Travel_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txt2Travel.KeyPress
42     ' allow only numbers, 1x decimal point, and Backspace key (each txt should not allow the user to enter more than 1 period):
43     If txt2Travel.Text Like "*.*" Then
44         If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
45             e.Handled = True
46         End If
47     Else
48         If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back AndAlso e.KeyChar <> "." Then
49             e.Handled = True
50         End If
51     End If
52 End Sub
53
54 Private Sub txt3Meals_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txt3Meals.KeyPress
55     ' allow only numbers, 1x decimal point, and Backspace key (each txt should not allow the user to enter more than 1 period):
56     If txt3Meals.Text Like "*.*" Then
57         If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
58             e.Handled = True
59         End If
60     Else
61         If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back AndAlso e.KeyChar <> "." Then
62             e.Handled = True
63         End If
64     End If
65 End Sub
66
```

```
67  Private Sub txt4Entertainment_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txt4Entertainment.KeyPress
68      ' allow only numbers, 1x decimal point, and Backspace key (each txt should not allow the user to enter more than 1 period):
69      If txt4Entertainment.Text Like "*.*" Then
70          If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
71              e.Handled = True
72          End If
73      Else
74          If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back AndAlso e.KeyChar <> "." Then
75              e.Handled = True
76          End If
77      End If
78  End Sub
79
80  Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
81      Me.Close()
82  End Sub
83
84  Private Sub txt1234_TextChanged(sender As Object, e As EventArgs) Handles txt1Lodging.TextChanged, txt2Travel.TextChanged,
85   txt3Meals.TextChanged, txt4Entertainment.TextChanged
86      lbl5Total.Text = Nothing : lbl6DeduAmount.Text = Nothing : lbl7DeduPercent.Text = Nothing
87  End Sub
88
89  Private Sub txt1Lodging_Enter(sender As Object, e As EventArgs) Handles txt1Lodging.Enter
90      txt1Lodging.SelectAll()
91  End Sub
92
93  Private Sub txt2Travel_Enter(sender As Object, e As EventArgs) Handles txt2Travel.Enter
94      txt2Travel.SelectAll()
95  End Sub
96
97  Private Sub txt3Meals_Enter(sender As Object, e As EventArgs) Handles txt3Meals.Enter
98      txt3Meals.SelectAll()
99  End Sub
100
101 Private Sub txt4Entertainment_Enter(sender As Object, e As EventArgs) Handles txt4Entertainment.Enter
102     txt4Entertainment.SelectAll()
103 End Sub
104
105 Private Sub txt1Lodging_MouseClick(sender As Object, e As MouseEventArgs) Handles txt1Lodging.MouseClick
106     txt1Lodging.SelectAll()
107 End Sub
108
```

```

109  Private Sub txt2Travel_MouseClick(sender As Object, e As MouseEventArgs) Handles txt2Travel.MouseClick
110      txt2Travel.SelectAll()
111  End Sub
112
113  Private Sub txt3Meals_MouseClick(sender As Object, e As MouseEventArgs) Handles txt3Meals.MouseClick
114      txt3Meals.SelectAll()
115  End Sub
116
117  Private Sub txt4Entertainment_MouseClick(sender As Object, e As MouseEventArgs) Handles txt4Entertainment.MouseClick
118      txt4Entertainment.SelectAll()
119  End Sub
120 End Class

```



## CH8\_FOCUS ON THE CONCEPTS LESSON

- CH8\_F1 - [Array](#) of variables ([Array](#)) introduction: One-Dimensional and Two-Dimensional [Arrays](#)
- CH8\_F2.1 - One-Dimensional [Array](#): introduction
- CH8\_F2.2 - declaring One-Dimensional [Array](#) - version 1 & version 2 autopsy
- CH8\_F2.3 - storing data in a [1D Array](#) autopsy
- CH8\_F2.4 - determing the number of elements in a [1D Array](#): [Array](#)'s [Length](#) property
- CH8\_F2.5 - determing the highest [subscript](#) in a [1D Array](#): property [Length](#) -1 or method [GetUpperBound\(0\)](#)
- CH8\_F2.6 - You Do It 1: [1D Array](#)'s [Lengt](#) property vs [GetUpperBound](#) method exercise: 01.You Do It 1 Solution
- CH8\_F2.7 - Traversing a [1D Array](#) = look at each element using [loop](#)
- CH8\_F2.8 - Traversing [1D Array](#) example: coding the [Harry Potter Characters Application](#) (02.Potter Solution)
- CH8\_F3.1 - [For Each...Next](#) loop statement - great for collection/array with unknown therefore variable number of elements
- CH8\_F3.2 - using [For Each...Next](#) loop example: coding the [Harry Potter Characters Application](#) (03.Potter Solution-ForEachNext)
- CH8\_F3.3 - Traversing [1D Array](#): [Do...Loop](#) vs [For...Next](#) vs [For Each...Next](#) code example with [Harry Potter Application](#):
- CH8\_F3.4 - You Do It 2: Traversing [1D Array](#): [Do...Loop](#) vs [For...Next](#) vs [For Each...Next](#) exercise: 04.You Do It 2 Solution
- CH8\_F4.1 - Calculating the Average [1D Array](#) value: calculate the average price of a company's stock example: 05.Waterson Solution-Average
- CH8\_F4.2 - You Do It 3: calculating the Total [1D Array](#) value using [Do...Loop](#) & [For...Next](#) & [For Each...Next](#) exercise: 06.You Do It 3 Solution
- CH8\_F5.1 - finding the Highest [1D Array](#) value: display the highest stock price and number of days example: 07.Waterson Solution-Highest
- CH8\_F5.2 - You Do It 4: finding the lowest [1D Array](#) value using [For...Next](#) statement exercise: 08.You Do It 4 Solution
- CH8\_F6.1 - Sorting a [1D Array](#): [Array.Sort\(\)](#) & [Array.Reverse\(\)](#) methods
- CH8\_F6.2 - Sorting a [1D Array](#): [Array.Sort\(\)](#) & [Array.Reverse\(\)](#) methods example: 09.Continents Solution
- CH8\_F7.1 - Two-Dimensional [Array](#): introduction
- CH8\_F7.2 - declaring [2D Array](#) - version 1 & version 2 autopsy
- CH8\_F7.3 - storing data in a [2D Array](#) autopsy
- CH8\_F7.4 - determing the highest [subscripts](#) in a [2D Array](#): method [GetUpperBound\(0\)](#) for row and [GetUpperBound\(1\)](#) for column
- CH8\_F7.5 - Traversing a [2D Array](#) = look at each element using loops: [Do...Loop](#) vs [For...Next](#) vs [For Each...Next](#) code examples
- CH8\_F7.6 - Traversing a [2D Array](#) example using [Do...Loop](#) & [For...Next](#) & [For Each...Next](#): coding the [Months Application](#) (10.Months Solution)
- CH8\_F7.7 - Totaling the values stored in a [2D Array](#) example: coding the [Jenko Booksellers Application](#) (11.Jenko Solution)
- CH8\_F7.8 - You Do It 5: Totaling [2D Array](#) value: [Do...Loop](#) vs [For...Next](#) vs [For Each...Next](#) exercise: 12.You Do It 5 Solution

## CH8\_APPLY THE CONCEPTS LESSON

- CH8\_A1.1 - Associate a 1D **Array** with a (1st) i.e. associate **Array** elements with **Items Collection**, for they have a several **commonalities**
- CH8\_A1.2 - Associate a 1D **Array** with a (1st) example: coding the **Presidents and Vice Presidents Application** (13.Presidents Solution)
- CH8\_A2.1 - create **1D Array Accumulator** and **1D Array Counter** associated with (1st) example: **Chocolate Bars Application** (14.Warren Solution)
- CH8\_A2.2 - **You Do It 6: 1D Array Accumulator & Counter** associated with (1st) using **For Each...Next & For...Next & Do...Loop**
- CH8\_A3.1 - create **2x Parallel 1D Arrays** introduction & example
- CH8\_A3.2 - create **2x Parallel 1D Arrays** example: coding the **Paper Warehouse Application** (16.Paper Solution-Parallel)
- CH8\_A4.1 - search **2D Array** introduction & example vs. **2x Parallel 1D Arrays**
- CH8\_A4.2 - search **2D Array** instead of **2x Parallel 1D Arrays** example: coding the **Paper Warehouse Application** (17.Paper Solution-TwoDim)
- CH8\_A5 - my addendum from EXERCISES: **Array** statement **ReDim** - 37.ReDim Solution\_EXERCISE 20\_advanced
- CH8\_A6 - using **LINQ** - Language-Integrated Query [dotaz] - to retrieve **values** from **Array** + e.g. **40.Linq Array Solution** - additional topic from **Appendix B**
- CH8\_A6.1 - using **LINQ's Aggregate operators** to retrieve a **single value** from an **Array** + **41.Linq Aggregate Array Solution** - additional topic from **Appendix B**
- CH8\_A7 - **Structure** - composite of variables like **array**, but with a combination of a different data types - additional topic from **Appendix B**
- CH8\_A7.1 - compare passing variables to a procedure: **several separated** vs. **only 1 structure** - **42.Norbert Solution** - additional topic from **Appendix B**
- CH8\_A7.2 - **array of structure** variables = each **array element** contains **member** variables created within **Structure** - **43.Paper Solution-Structure - Appendix B** & comparison with 2x parallel **1D array** used in **16.Paper Solution-Parallel**, and with **1x 2D array** used in **17.Paper Solution-TwoDim**

**CH8\_Summary:** **1D Array**, **For Each...Next** loop statement, **Array.Sort** & **Array.Reverse** methods, **2D Array**, **Parallel 1D Arrays / 2D Array**

**CH8\_Key Terms**

**CH8\_Exercises**

## CH8\_FOCUS ON THE CONCEPTS LESSON

### CH8\_F1 - Array of variables (Array) introduction: One-Dimensional and Two-Dimensional Arrays

- all of the variables you have used so far have been **simple variables**, also called a **scalar variables** - unrelated to any other variable in memory
- **scalar** = skalár, skalární, fyzikální veličina charakterizovaná jen velikostí
- at times, however, you will encounter applications in which some of the variables **are related to each other** ->
  - > in those applications, it is easier and more efficient to treat the **related variables as a group** as **array of variables** as **array of elements** as **Array**
- e.g.: - you might use an array of 50 variables to store the population of each U.S. state
- e.g.: - you might use an array of 8 variables to store the sales made in each of your company's 8 sales regions
- after your application enters the data into an array, it can use the data as many times as necessary without having to enter the data again
- e.g.: - your company's sales application can use the sales amounts stored in an array to calculate the total company sales and the percentage that each region contributed to the total sales
  - it can also use the sales amounts in the array either to calculate the average sales amount or to simply display the sales made in a specific region
- as you will learn in this lesson, the variables in an array can be used just like any other variables:
  - you can assign values to them,
  - use them in calculations,
  - display their contents, and so on...
- the most commonly used arrays in business applications are **one-dimensional** and **two-dimensional**
- the variables in an array are stored in consecutive (následný, postupný) locations in the RAM
- each variable in an array has the same name and data type

### CH8\_F2.1 - One-Dimensional Array: introduction

- the variables in an **Array** are stored in consecutive (následný, postupný) locations in the RAM
- each variable in an **Array** has the same name and data type
- you distinguish one variable from another variable in the same **Array** by using a **unique number => subscript**
- **subscript**: - always an Integer
  - indicates the variable's zero-based position in the array
  - is assigned by the computer when the array is created in RAM
  - the **1st** variable is assigned a **subscript of 0**, the **2nd** a **subscript of 1**, and so on...
- you refer to each variable in an **Array**:
  - #1. by the **Array's name** and
  - #2. by the **variable's subscript**, which is specified in a set of parentheses () immediately following the **Array** name

e.g.

- a **1D Array** named **intSales** that contains 4 variables, related that each stores the annual sales made by one of a company's 4 salespeople
  - **intSales(0)** -> 25500 (read as "intSales sub zero") <- you use **intSales(0)** to refer to the **1st** variable in the **intSales** array
  - **intSales(1)** -> 46750 (read as "intSales sub one") ...
  - **intSales(2)** -> 21000 (read as "intSales sub two") ...
  - **intSales(3)** -> 35500 (read as "intSales sub three") <- you use **intSales(3)** to refer to the **4th** variable in the **intSales** array

syntax version 1:  
**As Array**

**Dim/Static/Private arrayName(highestSubscript As Integer) As dataType**

**As Array**

declaring 1D Array:

syntax version 2:  
**As Array**

**Dim/Static/Private arrayName() As dataType = {initialValues}**

**As Array**

|                           |                                                                                                                                                                                                                                                                                                                                                                                |
|---------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Dim/Static/Private</b> | - variable declaration rules learned in previous lessons apply, so:<br>- <b>Dim</b> = procedure-level array, declared in procedure, procedure scope<br>- <b>Static</b> = procedure-level array, declared in procedure, class-level scope (used for accumulators)<br>- <b>Private</b> = class-level array, declared in the form class's declarations section, class-level scope |
| <b>arrayName</b>          | = version 1: one-dimensional array variable, with automatic initialization to <b>Nothing</b>                                                                                                                                                                                                                                                                                   |
| <b>arrayName()</b>        | = version 2: one-dimensional array variable, with specified initialization values                                                                                                                                                                                                                                                                                              |
| <b>highestSubscript</b>   | - <b>As Integer</b><br>= specifies the zero-based highest subscript in the array<br>- i.e.: an array whose <b>highestSubscript</b> = 2 will contain <b>3 elements</b>                                                                                                                                                                                                          |
| <b>As dataType</b>        | = type of data the array variables (referred to as <b>elements</b> ) will store                                                                                                                                                                                                                                                                                                |
| <b>{initialValues}</b>    | = comma-separated list of values you want assigned to the array elements                                                                                                                                                                                                                                                                                                       |

**Syntax version 1:** - the computer automatically initializes each array element when the **Array** is created - similar to scalar variables initialization - to **default value**  
 - elements in **string** data type **Array**: - automatically initialized using the keyword **Nothing**  
     - same like: **...As String = Nothing** or **...As String = String.Empty**  
 - elements in **numeric** data type **Array**: - automatically initialized to **0**  
     - same like: **...As Integer = Nothing** or **...As Integer = 0**  
 - elements in **Boolean** data type **Array**: - automatically initialized using the Boolean keyword **False**  
     - same like: **...As Boolean = Nothing** or **...As Boolean = False**  
 - **Nothing** = **default value** for any data type  
 - variables initialized to **Nothing** do not actually contain the word **Nothing**, rather they contain **no data** at all

**Syntax version 2:** - rather than having the computer use a **default value** to initialize each **Array** element, you can use this syntax to specify each element's initial value when the **Array** is declared  
 - assigning initial values to an **Array** is often referred to as **populating the Array**  
 - you list the initial values in the **initialValues** section of the syntax, using commas „,“ to separate the values, and enclose them in braces {}  
 <- notice that this syntax does not include the **highestSubscript** argument - instead, an empty set of parentheses follows the array name  
 - the computer automatically calculates the highest subscript based on the number of values listed in the **initialValues** section  
 - zero-based subscript in a **1D Array**, therefore the highest subscript is always one number less than the number of values listed in the **initialValues** section

e.g.1 - ver. 1:

**Dim strWarehouse(2) As String**

<- same like: **...As String = Nothing**, or **...As String = String.Empty**

<- declares a three-element procedure-level array named **strWarehouse** => strWarehouse(0), strWarehouse(1), strWarehouse(2)  
 <- each element is automatically initialized using the keyword **Nothing**

e.g.2 - ver.1:

```
Static intTotals(4) As Integer
```

<- same like: ...As Integer = Nothing, or ...As Integer = 0  
<- declares a static, five-element procedure-level array named **intTotals** => intTotals(0), intTotals(1), intTotals(2), intTotals(3), intTotals(4)  
<- each element is automatically initialized to 0

e.g.3 - ver.1:

```
Dim blnIsTrue(3) As Boolean
```

<- same like: ...As Boolean = Nothing, or .....As Boolean = False  
<- declares a four-element procedure-level array named **blnIsTrue** => blnIsTrue(0), blnIsTrue(1), blnIsTrue(2), blnIsTrue(3)  
<- each element is automatically initialized to False

e.g.4: ver. 2:

```
Private strNames() As String = {"Harry Potter", "Ron Weasley", "Hermione Granger", "Lord Voldemort"}
```

<- declares and initializes a four-element class-level array named **strNames**, where:  
strNames(0) = "Harry Potter", strNames(1) = "Ron Weasley", strNames(2) = "Hermione Granger", strNames(3) = "Lord Voldemort"

e.g.5: ver. 2:

```
Dim intSale() As Integer = {25500, 46750, 21000, 35500}
```

<- declares and initializes a four-element procedure-level array named **intSale**, where:  
intSale(0) = 25500, intSale(1) = 46750, intSale(2) = 21000, intSale(3) = 35500

### CH8\_F2.3 - storing data in a 1D Array autopsy

- after an **Array** is declared, you can use another statement to store a different value in an **Array** element / **Array** variable / element

e.g.1

```
Dim strCity(5) As String
```

<- declares a 6-element array variable

```
strCity(0) = "Nashville"
```

<- assigns the string "Nashville" to the first element in the **strCity** array

e.g.2

```
12 Dim intNumbers(4) As Integer
```

<- declares a 5-element numeric array variable

```
13 For intX As Integer = 1 To 5
```

```
14     intNumbers(intX - 1) = intX ^ 2
```

```
15 Next intX
```

<- note

<- assigns the **squares** of the numbers from **1** through **5** to the **intNumbers** array in a way that:

- intNumbers(0) = 1      <= intNumbers (1 -1) =  $1^2$
- intNumbers(1) = 4      <= intNumbers (2 -1) =  $2^2$
- intNumbers(2) = 9      <= intNumbers (3 -1) =  $3^2$
- intNumbers(3) = 16     <= intNumbers (4 -1) =  $4^2$
- intNumbers(4) = 25     <= intNumbers (5 -1) =  $5^2$

-> note    <- if you use **Option Strict On**, the error will occur:    "Option Strict On disallows implicit conversions from 'Double' to 'Integer'."

- why ?: because math operator **^** is a numeric **Double** data type operator

- fix1: exclude the expression **Option Strict On** from your code

- fix2: **14                intNumbers(intX - 1) = intX \* intX**

- fix3: **12                Dim dblNumbers(4) As Double**

**13                For intX As Integer = 1 To 5**

**14                dblNumbers(intX - 1) = intX ^ 2**

**15                Next intX**

e.g.3

```
Dim intNumbers(4) As Integer  
Dim intSub As Integer  
Do While intSub < 5  
    intNumbers(intSub) = 100  
    lblShow.Text = lblShow.Text & intNumbers(intSub) & " "  
    intSub += 1  
    'lblShow.Text = lblShow.Text & intNumbers(intSub) & " "  
Loop  
'lblShow.Text = lblShow.Text & intNumbers(intSub) & " "  
<- assigns the number 100 to each element in the intNumbers array
```

<- displays: "100 100 100 100 100" in the lblShow

<- any other position would lead to an error:



e.g.4

```
Dim dblPrice() As Double = {45, 25, 56.99, 33, 75}  
dblPrice(1) *= 1.25  
or  
dblPrice(1) = dblPrice(1) * 1.25
```

<- multiplies the contents of the 2nd element in the dblPrice array by 1.25 and then assigns the result to the element

e.g.5

```
Dim dblRates(9) As Double  
Double.TryParse(txtRate.Text, dblRates(2))  
<- assigns either the value entered in the txtRate control (converted to Double) or the number 0 to the 3rd element in the dblRates array
```

#### CH8\_F2.4 - determining the number of elements in a 1D Array: Array's Length property

- the number of elements in a **1D Array** is stored as an **integer** in the **Array's Length** property
- don't forget that the value in the **Length** property is **one number more than the highest subscript**

**1D Array's property Length syntax:**  
**As Integer**

**arrayName.Length As Integer**

number of elements **As Integer**

e.g.

```
Dim strNames(3) As String  
Dim intNumElements As Integer  
intNumElements = strNames.Length
```

<- highest subscript = 3, therefore Length property = 4

<- assigns the number 4 to the intNumElements variable

## CH8\_F2.5 - determining the highest subscript in a 1D Array: property Length -1 or method GetUpperBound(0)

- the highest subscript in a one-dimensional **Array** is always one number less than the number of **Array** elements, therefore you can determine it by the **Array's Length** property **-1** or you can use a method **GetUpperBound**
- **GetUpperBound** method
  - returns an Integer that represents the highest subscript in the specified dimension of the **Array**
  - when used with a **1D Array**, the specified dimension is always **0** and appears between parentheses after the method

**1D Array's** method **GetUpperBound** syntax:  
**As Integer**

**arrayName.GetUpperBound(dimension As Integer) As Integer**

highest subscript **As Integer**

**GetUpperBound**  
**(dimension)**

- returns **As Integer**
- = gets the index of the last element of the specified dimension in the array
- **As Integer**
- = zero-based dimension of the array whose upper bound needs to be determined
- specified dimension for a **one-dimensional** array is always **0**

e.g.

```
Dim strNames(3) As String  
Dim intHighestSub As Integer  
intHighestSub = strNames.GetUpperBound(0)  
<- assigns the number 3 to the intHighestSub variable
```

<- specified dimension for a **one-dimensional** array is always **0**

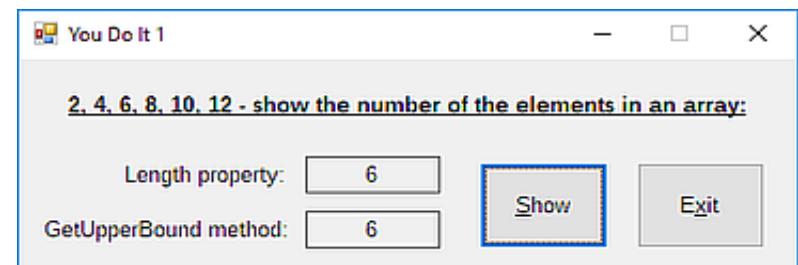
### Mini-Quiz 8-1:

1. Write a statement that declares a procedure-level one-dimensional array named **intOrders** containing **15** elements.
2. Write a statement that assigns the number **150** to the **3rd** element in the **intOrders** array.
3. Write a statement that assigns the number of elements in the **intOrders** array to the **intNum** variable.
4. Write a statement that assigns the **highest** subscript in the **intOrders** array to the **intLastSub** variable.

```
1...Dim intOrders(14) As Integer  
2...intOrders(2) = 150  
3...intNum = intOrders.Length Or  
3...intNum = intOrders.GetUpperBound(0) + 1  
4...intLastSub = intOrders.GetUpperBound(0) Or  
4...intLastSub = intOrders.Length - 1
```

## CH8\_F2.6 - You Do It 1: 1D Array's Length property vs GetUpperBound method exercise: 01.You Do It 1 Solution

1. create an application named **You Do It 1** and save it in the ...VB2017\Chap08\Exercise01.**You Do It 1** Solution
2. add: **two labels**, and a **button** to the form
3. the button's **Click** event procedure should declare and initialize an **integer array** named **intNums**
  - > use the following numbers to initialize the array: **2, 4, 6, 8, 10, 12**
4. the procedure should display the number of array elements in both label controls in a way that:
  - > use the **Length** property for one of the labels and
  - > use the **GetUpperBound** method for the other label
5. code the procedure, save the solution and then start and test the application.



## my GUI & code:

```
1  Public Class frmMain
2      Private Sub btnShow_Click(sender As Object, e As EventArgs) Handles btnShow.Click
3          Dim intNums() As Integer = {2, 4, 6, 8, 10, 12}
4          lbl1Length.Text = intNums.Length                      ' the number of array elements = 6
5          lbl2GetUpperBound.Text = intNums.GetUpperBound(0) + 1 ' the number of array elements = 6
6      End Sub
7      Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
8          Me.Close()
9      End Sub
10     End Class
```

## CH8\_F2.7 - Traversing a 1D Array = look at each element using loop

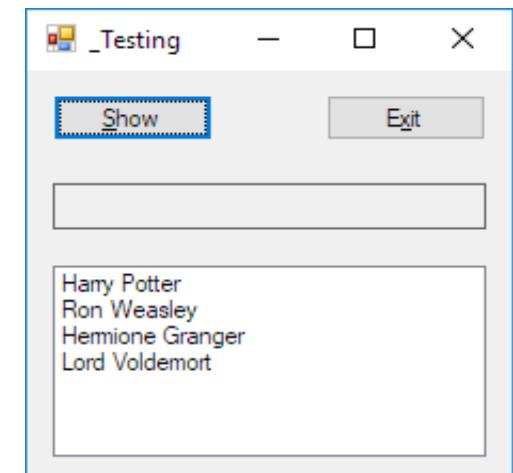
- at times, you may need to **traverse** an **Array** = to **look at each array element**, one by one, from first to the last element, using a **loop**

### e.g.1: For...Next loop

```
Dim strNames() As String = {"Harry Potter", "Ron Weasley", "Hermione Granger", "Lord Voldemort"}
Dim intHighSub As Integer = strNames.GetUpperBound(0)                                <- Same like = strNames.Length - 1
For intSub As Integer = 0 To intHighSub
    lstNames.Items.Add(strNames(intSub))
Next intSub
<- the loop instructions display each element's value in the lstNames control
```

### e.g.2: Do...Loop loop

```
Dim strNames() As String = {"Harry Potter", "Ron Weasley", "Hermione Granger", "Lord Voldemort"}
Dim intHighSub As Integer = strNames.Length - 1                                     <- same like = strNames.GetUpperBound(0)
Dim intSub As Integer
Do While intSub <= intHighSub
    lstNames.Items.Add(strNames(intSub))
    intSub += 1
Loop
<- the loop instructions display each element's value in the lstNames control
```



## CH8\_F2.8 - Traversing 1D Array example: coding the Harry Potter Characters Application (02.Potter Solution)

### 1). open the: ...VB2017\Chap08\Exercise\02.Potter Solution\Potter Solution.sln

- GUI contains a list box, three labels, and a button; open the Code Editor window and:

### 2). locate the **lstNames\_SelectedIndexChanged** procedure

- the procedure assigns the item selected in the list box to the **lblSelection.Text** property

```
- Private Sub lstNames_SelectedIndexChanged(sender As Object, e As EventArgs) Handles lstNames.SelectedIndexChanged
-     lblSelection.Text = lstNames.SelectedItem.ToString
- End Sub
```

3). locate the **frmMain\_Load** procedure on line **6**

-> enter the array declaration statement in lines **9** through **10**

```
6      Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
7          ' Fills the list box with array values and then selects the first item.
8
9          Dim strNames() As String = {"Harry Potter", "Ron Weasley", "Hermione Granger",
10             "Lord Voldemort", "Albus Dumbledore"}
11
12
13      End Sub
```

4). the procedure will fill the list box **lstNames** with the values stored in the **strNames()** array, so:

-> in the line **12** enter the loop of your choice -> either **For...Next** or **Do...Loop**, using either **GetUpperBound** method or **Length** property

- use the info from: **CH8\_F2.7 - Traversing a 1D Array** = look at each element using **loop**

```
12      Dim intHighSub As Integer = strNames.GetUpperBound(0)
13      For intSub As Integer = 0 To intHighSub
14          lstNames.Items.Add(strNames(intSub))
15      Next intSub
16
17      End Sub
```

or

```
12      Dim intHighSub As Integer = strNames.Length - 1
13      Dim intSub As Integer
14      Do While intSub <= intHighSub
15          lstNames.Items.Add(strNames(intSub))
16          intSub += 1
17      Loop
18
19      End Sub
```

5). finally, the procedure will select the **1st** item in the list box **lstNames**:

-> above the **End Sub** clause enter the following assignment statement:

```
    -
        lstNames.SelectedIndex = 0
    End Sub
```

6). save the solution and then start the application:

- the **frmMain\_Load** procedure creates and initializes the **strNames** array on lines **9** through **10**

- the loop then adds the contents of each array element to the **lstNames** control in lines **13** through **15**

- the loop stops when the **intSub** variable contains the number **5**, which is one number more than the highest subscript in the array in line **12**

- the last statement in the **frmMain\_Load** procedure invokes the list box's **SelectedIndexChanged** event, whose procedure displays the selected item in line **17**

-> click each name in the list box, one at a time, to verify that the application works correctly.

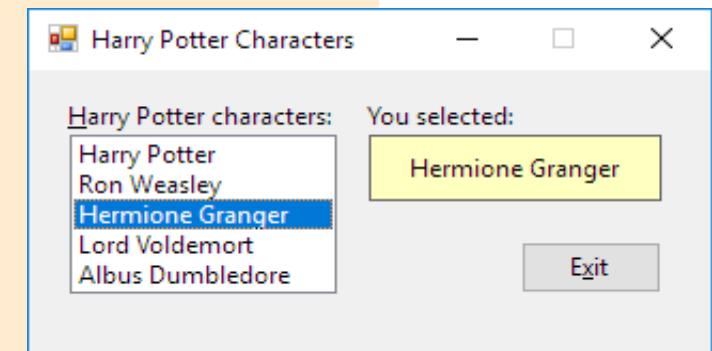
7). completed code & GUI:

```
1  Option Explicit On
2  Option Strict On
3  Option Infer Off
```

```

4
5  Public Class frmMain
6      Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
7          ' Fills the list box with array values and then selects the first item.
8
9      Dim strNames() As String = {"Harry Potter", "Ron Weasley", "Hermione Granger",
10             "Lord Voldemort", "Albus Dumbledore"}
11
12     Dim intHighSub As Integer = strNames.GetUpperBound(0)
13     For intSub As Integer = 0 To intHighSub
14         lstNames.Items.Add(strNames(intSub))
15     Next intSub
16
17     lstNames.SelectedIndex = 0
18
19 End Sub
20
21 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
22     Me.Close()
23 End Sub
24
25 Private Sub lstNames_SelectedIndexChanged(sender As Object, e As EventArgs) Handles lstNames.SelectedIndexChanged
26     lblSelection.Text = lstNames.SelectedItem.ToString()
27 End Sub
28 End Class

```



### CH8\_F3.1 - For Each...Next loop statement - great for collection/array with unknown therefore variable number of elements

- for coding a loop you can use:
  - Do...Loop, or
  - For...Next, or
  - For Each...Next**
- CH5 - The Repetition Structure**
- CH5 - The Repetition Structure**
- used when dealing with a collection/group/array with unknown-variable number of elements
- handy for one-dimensional array
  
- a **For...Next** Statement works well when you can associate each iteration (opakování, iterace) of a loop with a control variable and determine that variable's initial and final values. However, when you are dealing with a collection / group / array, the concept of initial and final values isn't meaningful, and you don't necessarily know how many elements the collection has. In this kind of case, a **For Each...Next** loop is often a better choice.
  
- the **For Each...Next**:
  - provides a convenient way of coding a loop whose instructions you want processed for **each element** in a group, such as for **each variable** in an array
  - **advantage:**
    - your code doesn't need to keep track of the array subscript
    - even doesn't need to know the number of array elements
  - **disadvantage:**
    - unlike the loop instructions in a **Do...Loop** or **For...Next** statement, the instructions can **only read** the array values, they **can't** permanently modify the values

[For Each...As...In...Next](#) loop statement syntax:

```
For Each elementVariable As dataType In collection
    ...loop body instructions...
Next elementVariable
```

- elementVariable* = name of a variable that the computer can use to keep track of each element in the *collection*  
- although you don't need to specify it, doing so is highly recommended because it makes your code clearer and easier to understand
- dataType* - **As data type**  
= *elementVariable*'s data type  
- must be same as *collection*'s data type  
- has a block scope, therefore recognized only by the instructions within the loop
- collection* = array variable / group of related variables

e.g.

```
Dim strNames() As String = {"Harry Potter", "Ron Weasley", "Hermione Granger", "Lord Voldemort", "Albus Dumbledore"}

For Each strElement As String In strNames
    lstNames.Items.Add(strElement)
Next strElement
```

#### CH8\_F3.2 - using [For Each...Next](#) loop example: coding the Harry Potter Characters Application (03.Potter Solution-ForEachNext)

1). open the: ...VB2017\Chap08\Exercise\03.Potter Solution-ForEachNext\Potter Solution.sln

2). locate the frmMain\_Load procedure on line 6

-> enter the code for [For Each...Next](#) loop statement shown on lines 11 through 13

```
11      For Each strElement As String In strNames
12          lstNames.Items.Add(strElement)
13      Next strElement
```

3). save the solution and then start the application:

-> click each name in the list box, one at a time, to verify that the application works correctly.

### CH8\_F3.3 - Traversing 1D Array: Do...Loop vs For...Next vs For Each...Next code example with Harry Potter Application:

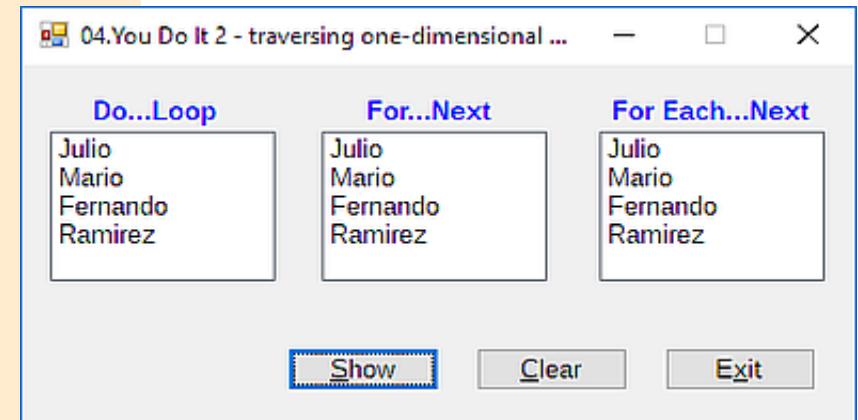
|                                                                                                                      |                                                       |                                           |
|----------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------|-------------------------------------------|
| Dim strNames() As String = {"Harry Potter", "Ron Weasley", "Hermione Granger", "Lord Voldemort", "Albus Dumbledore"} |                                                       |                                           |
| Dim intHighSub As Integer = strNames.Length - 1                                                                      | Dim intHighSub As Integer = strNames.GetUpperBound(0) | For Each strElement As String In strNames |
| Dim intSub As Integer                                                                                                | For intSub As Integer = 0 To intHighSub               | lstNames.Items.Add(strElement)            |
| Do While intSub <= intHighSub                                                                                        | lstNames.Items.Add(strNames(intSub))                  | Next strElement                           |
| lstNames.Items.Add(strNames(intSub))                                                                                 |                                                       | For Each...Next statement                 |
| intSub += 1                                                                                                          |                                                       |                                           |
| Loop                                                                                                                 | Next intSub                                           |                                           |
| Do...Loop statement                                                                                                  | For...Next statement                                  |                                           |

### CH8\_F3.4 - You Do It 2: Traversing 1D Array: Do...Loop vs For...Next vs For Each...Next exercise: 04.You Do It 2 Solution

1. create an application named **You Do It 2** and save it in the ...VB2017\Chap08\Exercise04.You Do It 2 Solution
2. add: a **button**, and 3x **list boxes** to the form
3. the button's **Click** event procedure should declare and initialize a **one-dimensional String array**, using any four names to initialize the array
4. the procedure should display the array elements in all three list boxes in a way that:
  - > **1st** list box should display the contents of the array using the **Do...Loop** statement
  - > **2nd** list box should display the contents of the array using the **For...Next** statement
  - > **3rd** list box should display the contents of the array using the **For Each...Next** statement
5. code the procedure, save the solution and then start and test the application.

my GUI & code:

```
1  Option Explicit On
2  Option Strict On
3  Option Infer Off
4  Public Class frmMain
5      Private Sub btnShow_Click(sender As Object, e As EventArgs) Handles btnShow.Click
6          lst1DoLoop.Items.Clear()
7          lst2ForNext.Items.Clear()
8          lst3ForEachNext.Items.Clear()
9
10         Dim strNames() As String = {"Julio", "Mario", "Fernando", "Ramirez"}
11         Dim intMaxIndexA As Integer = strNames.Length - 1           ' =3
12         Dim intMaxIndexB As Integer = strNames.GetUpperBound(0)     ' =3
13         Dim intIndex As Integer
14
15         ' Do...Loop loop:
16         Do While intMaxIndexA >= 0
17             lst1DoLoop.Items.Add(strNames(intIndex))
18             intIndex += 1
19             intMaxIndexA -= 1
20         Loop
```



```

21      ' For...Next loop:
22      For intCount As Integer = 0 To intMaxIndexB
23          lst2ForNext.Items.Add(strNames(intCount))
24      Next intCount
25
26      ' For Each...Next loop:
27      For Each strElement As String In strNames
28          lst3ForEachNext.Items.Add(strElement)
29      Next strElement
30
31  End Sub
32
33  Private Sub btnClear_Click(sender As Object, e As EventArgs) Handles btnClear.Click
34      lst1DoLoop.Items.Clear()
35      lst2ForNext.Items.Clear()
36      lst3ForEachNext.Items.Clear()
37  End Sub
38  Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
39      Me.Close()
40  End Sub
41
42 End Class

```

#### CH8\_F4.1 - Calculating the Average 1D Array value: calculate the average price of a company's stock example: 05.Waterson Solution-Average

- to calculate the average of the values stored in an **Array**, you first total the values and then divide the total by the number of **Array** elements
- in the next set of steps, you code an application that calculates the average price of a company's stock

1). open the: ...VB2017\Chap08\Exercise\05.Waterson Solution-Average\Waterson Solution.sln

2). open the Code Editor window

- the **Private** statement on line 8 in the form's class's declarations section stores the company's 10-day stock prices in a class-level array named **dblPrices**

```

7      ' Class-level array:
8      Private dblPrices() As Double = {85.7, 89.5, 91, 99, 97.5, 96, 96.8, 96.8, 96, 99}

```

- a class-level array is appropriate in this case because two procedures will need access to the array -> **frmMain\_Load** and **btnCalc\_Click**

3). locate the **frmMain\_Load** procedure on line 10

- the procedure will use the **For Each...Next** statement to fill the list box with the prices stored in the array

-> enter the code for **For Each...Next** loop statement shown on lines 13 through 15

```

13      For Each dblStockPrice As Double In dblPrices
14          lstPrices.Items.Add(dblStockPrice.ToString("N2"))
15      Next dblStockPrice

```

4). locate the **btnCalc\_Click** procedure on line **18**

- the procedure declares two variables:

- variable named **dblTotal** will be used to accumulate the prices stored in the array
- variable named **dblAvg** will store the average price

-> on line **24** enter one of the examples for accumulating the array values

- each example uses a loop to add each array element's value to the **dblTotal** variable

| e.g. 1 - Do...Loop statement:                                                                                                                                                                        | e.g. 2 - For...Next statement:                                                                                                                                                         | e.g. 3 - For Each...Next statement:                                                                               |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------|
| <pre>Dim intHighSub As Integer = dblPrices.Length - 1 Dim intSub As Integer ' accumulate stock prices: Do While intSub &lt;= intHighSub     dblTotal += dblPrices(intSub)     intSub += 1 Loop</pre> | <pre>Dim intHighSub As Integer = dblPrices.GetUpperBound(0) ' ' accumulate stock prices: For intSub As Integer = 0 To intHighSub     dblTotal += dblPrices(intSub) ' Next intSub</pre> | <pre>' accumulate stock prices: For Each dblDay As Double In dblPrices     dblTotal += dblDay ' Next dblDay</pre> |

- <- notice:
- **Do...Loop** and **For...Next** statements need you to specify the highest array subscript using either **Length** property or **GetUpperBound** method
    - but not the **For Each...Next** statement
  - **Do...Loop** and **For...Next** statements must keep track of the array subscripts
    - but this task is not necessary in the **For Each...Next** statement
- when each loop has finished processing, the **dblTotal** variable contains the sum of the **10** stock prices stored in the **dblPrices** array, the number **947.3**

5). next, the procedure will calculate the average price by dividing the value stored in the **dblTotal** variable by the number of array elements

- then it will display the average price

-> enter the following comment and assignment statements:

```
' Calculate and display the average:
dblAvg = dblTotal / dblPrices.Length
lblAvg.Text = dblAvg.ToString("C2"))
End Sub
```

6). save the solution and then start the application

- the **Average price** should be: **\$94.73**

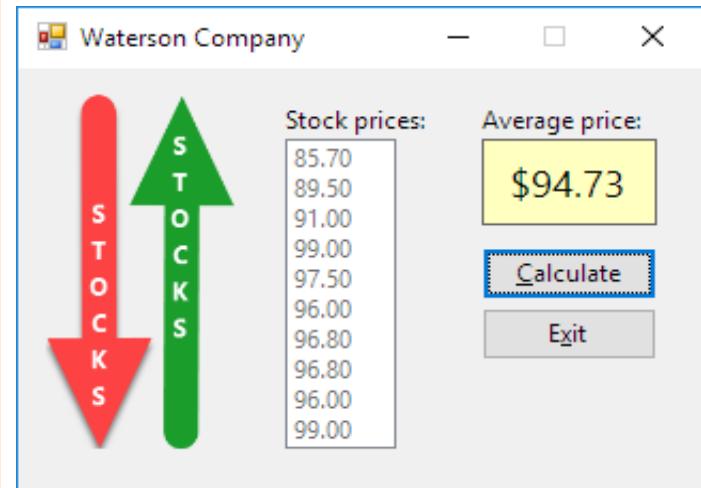
7). completed code & GUI:

```
1  Option Explicit On
2  Option Strict On
3  Option Infer Off
4
5  Public Class frmMain
6
7      ' Class-level array:
8      Private dblPrices() As Double = {85.7, 89.5, 91, 99, 97.5, 96, 96.8, 96.8, 96, 99}
9
```

```

10  Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
11      ' Fills list box with prices.
12
13      For Each dblStockPrice As Double In dblPrices
14          lstPrices.Items.Add(dblStockPrice.ToString("N2"))
15      Next dblStockPrice
16  End Sub
17
18  Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
19      ' Calculates and displays the average stock price.
20
21      Dim dblTotal As Double
22      Dim dblAvg As Double
23
24      ' accumulate stock prices:
25      For Each dblDay As Double In dblPrices
26          dblTotal += dblDay
27      Next dblDay
28
29      ' Calculate and display the average:
30      dblAvg = dblTotal / dblPrices.Length
31      lblAvg.Text = dblAvg.ToString("C2")
32  End Sub
33
34  Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
35      Me.Close()
36  End Sub
37 End Class

```



### Mini-Quiz 8-2:

1. In order to access each element in an array, the **Do...Loop** statement needs to know the highest subscript in the array. True or False?
2. In order to access each element in an array, the **For...Next** statement needs to know the highest subscript in the array. True or False?
3. In order to access each element in an array, the **ForEach...Next** statement needs to know the highest subscript in the array. True or False?
4. When traversing an array, which of the following statements doesn't need to keep track of the individual array subscripts?
  - a). **Do...Loop**
  - b). **For...Next**
  - c). **ForEach...Next**

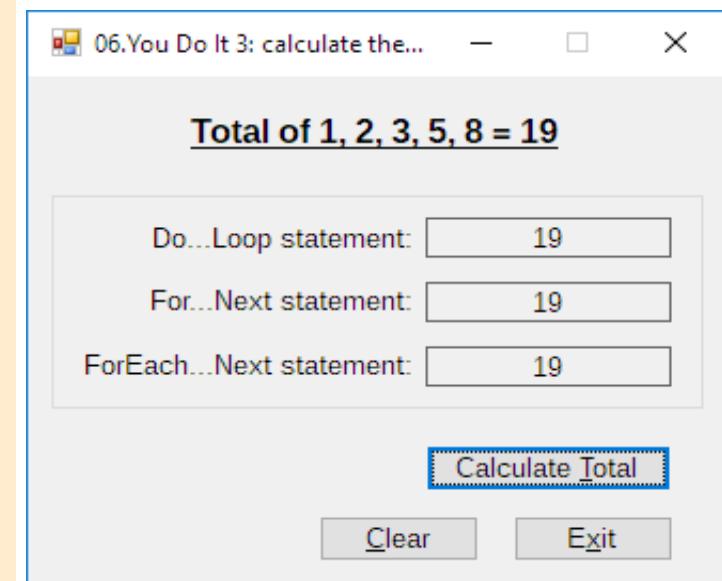
1...True  
2...True  
3...False  
4...C). ForEach...Next

## CH8\_F4.2 - You Do It 3: calculating the Total 1D Array value using Do...Loop & For...Next & For Each...Next exercise: 06.You Do It 3 Solution

1. create an application named **You Do It 3** and save it in the ...VB2017\Chap08\Exercise\06.You Do It 3 Solution
2. add: a **button**, and **3x labels** to the form
3. the button's **Click** event procedure should declare and initialize a **one-dimensional Integer array**, using any five integers to initialize the array
4. the procedure should total the five integers and then display the result in three labels in a way that:
  - > **1st** label should display the calculated total using the **Do...Loop** statement
  - > **2nd** label should display the calculated total using the **For...Next** statement
  - > **3rd** label should display the calculated total using the **ForEach...Next** statement
5. code the procedure, save the solution and then start and test the application.

my GUI & code:

```
1  Option Explicit On
2  Option Strict On
3  Option Infer Off
4
5  Public Class frmMain
6
7      Private Sub btnClear_Click(sender As Object, e As EventArgs) Handles btnClear.Click
8          lbl1DoLoop.Text = String.Empty
9          lbl2ForNext.Text = String.Empty
10         lbl3ForEachNext.Text = String.Empty
11     End Sub
12
13    Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
14        Me.Close()
15    End Sub
16
17    Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
18        lbl1DoLoop.Text = String.Empty
19        lbl2ForNext.Text = String.Empty
20        lbl3ForEachNext.Text = String.Empty
21
22        Dim intArray() As Integer = {1, 2, 3, 5, 8}
23
24        ' Do...Loop:
25        Dim intTotal1 As Integer
26        Dim intCount1a As Integer
27        Dim intCount1b As Integer = intArray.Length - 1
28        Do While intCount1a <= intCount1b
29            intTotal1 += intArray(intCount1a)
30            lbl1DoLoop.Text = intTotal1.ToString
31            intCount1a += 1
32        Loop
33
```



```

34     ' For...Next:
35     Dim intTotal2 As Integer
36     Dim intCount2 As Integer = intArray.GetUpperBound(0)
37     For intElement As Integer = 0 To intCount2
38         intTotal2 += intArray(intElement)
39         lbl2ForNext.Text = intTotal2.ToString
40     Next intElement
41
42     ' ForEach...Next:
43     Dim intTotal3 As Integer
44     For Each intElement As Integer In intArray
45         intTotal3 += intElement
46         lbl3ForEachNext.Text = intTotal3.ToString
47     Next intElement
48
49     End Sub
50 End Class

```

#### CH8\_F5.1 - finding the Highest 1D Array value: display the highest stock price and number of days example: 07.Waterson Solution-Highest

- in this section, you will code a different application for the **Waterson Company**
- rather than displaying the average stock price, this application displays the **highest** stock price and the **number** of days the stock closed at the price
- <- when searching an **Array** for the highest (or lowest) value, it is a common coding practice to **initialize** the variable to the value stored in the **1st Array element**

##### pseudocode for the **btnDisplay\_Click** procedure:

- step 1:** declare **intLastSub** variable and initialize it to **last** array subscript
- step 2:** declare **dblHighest** variable and initialize it to the price stored in the **1st** array element
- step 3:** declare **intDays** counter variable and initialize it to **1**
- step 4:** repeat for each element in the **dblPrices** array, starting with the **2nd** element
- step 4.1:**
- if the current element's price is equal to the price stored in the **dblHighest**
    - add **1** to the **intDays** counter variable
  - else
    - if the current element's price is greater than the price stored in **dblHighest**
      - assign the current element's price to **dblHighest**
      - assign **1** to the **intDays** counter variable
    - end if
  - end if

end if

end repeat

- step 5:** display the highest price: **dblHighest** in **lblHighest** and number of days: **intDays** in **lblDays**

1). open the: ...VB2017\Chap08\Exercise\07.Waterson Solution-Highest\Waterson Solution.sln

2). open the Code Editor window

- the **Private** statement on line **8** in the form's class's declarations section creates and initializes the class-level **dblPrices** array
- a class-level array is appropriate in this case because two procedures will need access to the array: **frmMain\_Load** and **btnDisplay\_Click**

```
7      ' Class-level array:  
8      Private dblPrices() As Double = {85.7, 89.5, 91, 99, 97.5, 96, 96.8, 96.8, 96, 99}
```

3). locate the **frmMain\_Load** procedure on line 10 - it fills the list box with the array values

```
10     Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load  
11         ' Fills list box with prices:  
12  
13         For Each dblStockPrice As Double In dblPrices  
14             lstPrices.Items.Add(dblStockPrice.ToString("N2"))  
15         Next dblStockPrice  
16     End Sub
```

4). locate the **btnDisplay\_Click** procedure on line 18

- the procedure already contains the code for the last step in the pseudocode, which is to display the highest price and the number of days:

**step 5:** display the highest price: **dblHighest** in **lblHighest** and number of days: **intDays** in **lblDays**

```
-     lblHighest.Text = dblHighest.ToString("C2")  
-     lblDays.Text = intDays.ToString()
```

**step 1:** declare **intLastSub** variable and initialize it to **last** array subscript

-> click the blank line 22 below the comment '**Declare and initialize variables:**' and type the following declaration statement:

```
21     ' Declare and initialize variables:  
22     Dim intLastSub As Integer = dblPrices.GetUpperBound(0)
```

**step 2:** declare **dblHighest** variable and initialize it to the price stored in the **1st** array element

<- when searching an array for the highest (or lowest) value, it is a common coding practice to initialize the variable to the value stored in the **1st** array element

-> type the following declaration statement:

```
23     Dim dblHighest As Double = dblPrices(0)
```

**step 3:** declare **intDays** counter variable and initialize it to **1**

- the variable will keep track of the number of elements (days) whose stock price matches the value stored in the **dblHighest** variable

<- the procedure will initialize the variable to **1** because, at this point, only the **1st** element contains the price currently stored in the **dblHighest** variable

-> type the following declaration statement:

```
24     Dim intDays As Integer = 1
```

**step 4:** repeat for each element in the **dblPrices** array, starting with the **2nd** element

- next, the procedure will use the **For...Next** statement to traverse the **2nd** element through the **last** element in the array

- each element's value will be compared - one at a time - to the value stored in the **dblHighest** variable

<- you don't need to look at the **1st** element because its value is already contained in the **dblHighest** variable

-> enter the following **For...Next** loop clause:

```
26     For intSub As Integer = 1 To intLastSub  
27  
28         Next intSub
```

**step 4.1:** if the current element's price is equal to the price stored in the **dblHighest**...

- the loop contains s.s. whose condition determines whether the price stored in the current array element is equal to the price stored in the **dblHighest** variable
  - if both prices are equal, the selection structure's **true** path adds **1** to the **intDays** counter variable
  - if both prices aren't equal, the selection structure's **false** path determines whether the price stored in the current array element is greater than the price stored in the **dblHighest** variable
    - if the price in the current array element is greater than the price in the **dblHighest** variable, the s.s. should assign the higher value to the **dblHighest**
    - it should also reset the number of days counter to **1** because - at this point - only the current element contains that price

-> enter the following selection structure and assignment statements on line **27** through **32**:

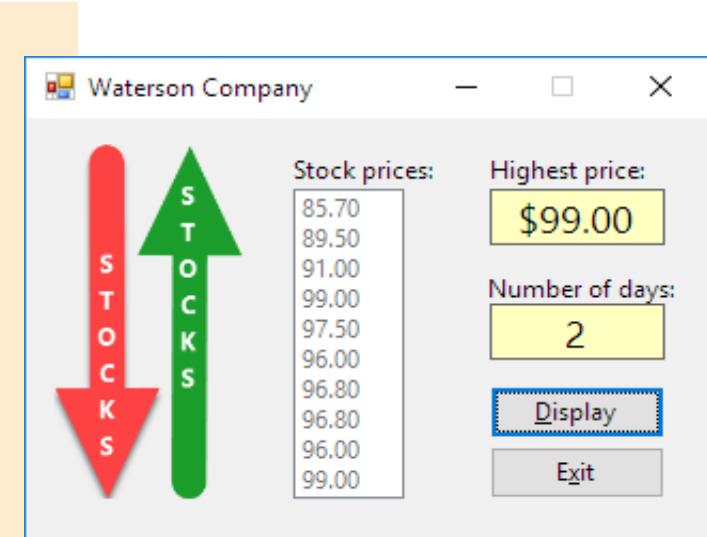
```
26      For intSub As Integer = 1 To intLastSub
27          If dblPrices(intSub) = dblHighest Then
28              intDays += 1
29          ElseIf dblPrices(intSub) > dblHighest Then
30              dblHighest = dblPrices(intSub)
31              intDays = 1
32          End If
33      Next intSub
```

**5).** save the solution and then start and test the application:

-> click the button **Display** - the highest stock price: **\$99.00** and the number of days the stock closed at that price: **2** appear in the GUI

**6).** completed code & GUI:

```
1  Option Explicit On
2  Option Strict On
3  Option Infer Off
4
5  Public Class frmMain
6
7      ' Class-level array.
8      Private dblPrices() As Double = {85.7, 89.5, 91, 99, 97.5, 96, 96.8, 96.8, 96, 99}
9
10     Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
11         ' Fills list box with prices.
12
13         For Each dblStockPrice As Double In dblPrices
14             lstPrices.Items.Add(dblStockPrice.ToString("N2"))
15         Next dblStockPrice
16     End Sub
17
18     Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
19         ' Displays the highest stock price and the number of days at that price.
20
21         ' Declare and initialize variables:
22         Dim intLastSub As Integer = dblPrices.GetUpperBound(0)
23         Dim dblHighest As Double = dblPrices(0)
24         Dim intDays As Integer = 1
25
```



```

26     For intSub As Integer = 1 To intLastSub
27         If dblPrices(intSub) = dblHighest Then
28             intDays += 1
29         ElseIf dblPrices(intSub) > dblHighest Then
30             dblHighest = dblPrices(intSub)
31             intDays = 1
32         End If
33     Next intSub
34
35     lblHighest.Text = dblHighest.ToString("C2")
36     lblDays.Text = intDays.ToString
37 End Sub
38
39 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
40     Me.Close()
41 End Sub
42 End Class

```

#### CH8\_F5.2 - You Do It 4: finding the lowest 1D Array value using For...Next statement exercise: 08.You Do It 4 Solution

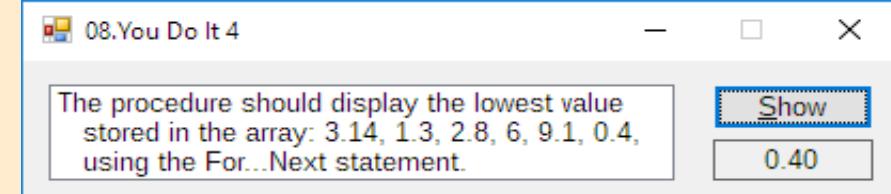
1. create an application named **You Do It 4** and save it in the ...VB2017\Chap08\Exercise\08.You Do It 4 Solution
2. add: a **button**, and a **label** to the form
3. the button's **Click** event procedure should declare and initialize a **one-dimensional Double array**, using any six numbers to initialize the array
4. the procedure should display the lowest value stored in the array using the **For...Next** statement
5. code the procedure, save the solution and then start and test the application.

my GUI & code:

```

1  Option Strict On
2  Option Explicit On
3  Option Infer Off
4  Public Class frmMain
5      Private Sub btnShow_Click(sender As Object, e As EventArgs) Handles btnShow.Click
6          Dim dblArray() As Double = {3.14, 1.3, 2.8, 6, 9.1, 0.4}
7          Dim dblLowest As Double = dblArray(0)
8
9          For intCount As Integer = 1 To dblArray.GetUpperBound(0)
10             If dblArray(intCount) <= dblLowest Then
11                 dblLowest = dblArray(intCount)
12             End If
13         Next intCount
14         lblShow.Text = dblLowest.ToString("N2")
15     End Sub
16 End Class

```



## CH8\_F6.1 - Sorting a 1D Array: `Array.Sort()` & `Array.Reverse()` methods

- you can use the method `Array.Sort` to sort the values in a one-dimensional `Array` in ascending order
- you can use the method `Array.Reverse` to just reverse the order of the `Array` values stored in a one-dimensional `Array`
- to sort the values in descending order:
  - #1: use the `Array.Sort()` method to sort the values in ascending order, then
  - #2: use the `Array.Reverse()` method to reverse the sorted values

One-Dimensional array's method `Array.Sort` syntax:

`Array.Sort(arrayName As Array)`

array variables / array elements / elements

One-Dimensional array's method `Array.Reverse` syntax:

`Array.Reverse(arrayName As Array)`

array variables / array elements / elements

- `.Sort` = sorts the elements in an entire one-dimensional `Array` in ascending order
- `.Reverse` = just reverses the order of the elements in an entire one-dimensional `Array`
- `arrayName`
  - `As Array`
  - = any previously declared `Array`

e.g.1

```
Dim intScores() As Integer = {78, 90, 75, 83}  
Array.Sort(intScores)
```

<- sorts the contents of the array in ascending order, as follows: **75, 78, 83, 90**.

e.g.2

```
Dim intScores() As Integer = {78, 90, 75, 83}  
Array.Reverse(intScores)
```

<- just reverses the order of the array contents, so: **83, 75, 90, 78**.

e.g.3

```
Dim intScores() As Integer = {78, 90, 75, 83}  
Array.Sort(intScores)  
Array.Reverse(intScores)
```

<- sorts the contents of the array in ascending order, as follows: **75, 78, 83, 90**.

<- just reverses the previous order, so the result is a descending order: **90, 83, 78, 75**.

## CH8\_F6.2 - Sorting a 1D Array: `Array.Sort()` & `Array.Reverse()` methods example: 09.Continents Solution

- you will use the `Array.Sort` and `Array.Reverse` methods to finish coding this application
- the application stores the names of the seven continents in a one-dimensional array named `strContinents`
- it then allows the user to display the names in a list box, in either ascending or descending order

1). open the: ...VB2017\Chap08\Exercise09.Continents Solution\Continents Solution.sln

2). open the Code Editor window and notice that **form class**'s declaration section contains two `Private` statements:

- the 1st statement declares the `strContinents` array and initializes it to the names of the seven continents
- the 2nd statement declares the `intLastSub` variable and initializes it to the last subscript in the array
- the array and variable were declared as class-level memory locations because both need to be accessed by two procedures:

`btnAscending_Click` and `btnDescending_Click`

```
5  Public Class frmMain  
6      ' Class-level array and variable:  
7      Private strContinents() As String = {"North America", "Africa", "South America", "Antarctica", "Australia", "Asia", "Europe"}  
8      Private intLastSub As Integer = strContinents.GetUpperBound(0)
```

3). -> locate the **btnAscending\_Click** and **btnDescending\_Click** procedures and notice that both already contains:

- statement, which clears the contents of the list box:
- loop that displays the array contents in the list box:

```
1stContinents.Items.Clear()  
For intSub As Integer = 0 To intLastSub  
    1stContinents.Items.Add(strContinents(intSub))  
Next intSub
```

-> locate the **btnAscending\_Click** procedure and just under the **1stContinents.Items.Clear()** statement enter the **Sort** method:

```
13     1stContinents.Items.Clear()  
14     Array.Sort(strContinents)
```

-> locate the **btnDescending\_Click** procedure and just under the **1stContinents.Items.Clear()** statement enter the **Sort** and **Reverse** methods:

```
24     1stContinents.Items.Clear()  
25     Array.Sort(strContinents)  
26     Array.Reverse(strContinents)
```

4). save the solution and then start and test the application:

- > click the button **Ascending order** to display the names in ascending order
- > click the button **Descending order** to display the names in descending order



5). click the button **Exit**, close the Code Editor window and then close the solution.

6). completed code:

```
1  Option Explicit On  
2  Option Strict On  
3  Option Infer Off  
4  
5  Public Class frmMain  
6      ' Class-level array and variable:  
7      Private strContinents() As String = {"North America", "Africa", "South America", "Antarctica", "Australia", "Asia", "Europe"}  
8      Private intLastSub As Integer = strContinents.GetUpperBound(0)  
9  
10     Private Sub btnAscending_Click(sender As Object, e As EventArgs) Handles btnAscending.Click  
11         ' Sorts the array values in ascending order.  
12     End Sub
```

```

13     1stContinents.Items.Clear()
14     Array.Sort(strContinents)
15
16     For intSub As Integer = 0 To intLastSub
17         1stContinents.Items.Add(strContinents(intSub))
18     Next intSub
19 End Sub
20
21 Private Sub btnDescending_Click(sender As Object, e As EventArgs) Handles btnDescending.Click
22     ' Sorts the array values in descending order.
23
24     1stContinents.Items.Clear()
25     Array.Sort(strContinents)
26     Array.Reverse(strContinents)
27
28     For intSub As Integer = 0 To intLastSub
29         1stContinents.Items.Add(strContinents(intSub))
30     Next intSub
31 End Sub
32
33 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
34     Me.Close()
35 End Sub
36
37 End Class

```

### CH8\_F7.1 - Two-Dimensional Array: introduction

- as mentioned earlier, the most commonly used **Arrays** in business applications are **one-dimensional** and **two-dimensional**

- **1D Array** = like a **column** of variables-elements in RAM

- **2D Array** = like a table, where variables-elements are in **columns and rows**

- you can determine the number of elements by multiplying the number of its rows by the number of its columns  
e.g. an array that has 4 rows and 3 columns, contains 12 elements

- each element is identified by a **unique combination of 2 subscripts** that the computer assigns to the **element** when the array is created

- **2 subscripts** specify the element's **row** and **column** zero-based positions in the **Array** in a way that:

- all of the elements in the **1st** row have a **row subscript of 0**, elements in the **2nd** row have a **row subscript of 1**, and so on...

- all of the elements in the **1st** column have a **column subscript of 0**, elements in the **2nd** column have a **column subscript of 1**, and so on...

- **you refer to each element:** - #1. array's **name**

- #2. element's **row** zero-based **subscript**

- #3. element's **column** zero-based **subscript**

**e.g.:** strGrammy(1, 3)

<- e.g.: strGrammy

<- e.g.: 1

<- e.g.: 3

<- **element** located in the **2nd** row, **4th** column

<- read: "strGrammy sub 1 comma 3"

- elements contained in the 2D Array strGrammy :

```
Private strGrammy(,) As String = {{"2015", "57th", "Morning Phase", "Beck"},  
                                  {"2016", "58th", "1989", "Taylor Swift"},  
                                  {"2017", "59th", "25", "Adele"}}}
```

e.g.

therefore:

- strGrammy(0, 0) = "2015"
- strGrammy(0, 2) = "Morning Phase"
- strGrammy(1, 3) = "Taylor Swift"
- strGrammy(2, 1) = "59th"

|      | column0             | column1             | column2                      | column3                     |
|------|---------------------|---------------------|------------------------------|-----------------------------|
| row0 | 0, 0<br><b>2015</b> | 0, 1<br><b>57th</b> | 0, 2<br><b>Morning Phase</b> | 0, 3<br><b>Beck</b>         |
| row1 | 1, 0<br><b>2016</b> | 1, 1<br><b>58th</b> | 1, 2<br><b>1989</b>          | 1, 3<br><b>Taylor Swift</b> |
| row2 | 2, 0<br><b>2017</b> | 2, 1<br><b>59th</b> | 2, 2<br><b>25</b>            | 2, 3<br><b>Adele</b>        |

## CH8\_F7.2 - declaring 2D Array - version 1 & version 2 autopsy

syntax version 1:  
**As Array**

```
Dim/Static/Private arrayName(highestRowSub As Integer, highestColumnSub As Integer) As dataType
```

**As Array**

declaring 2D Array:

syntax version 2:  
**As Array**

```
Dim/Static/Private arrayName(,) As dataType = {{rowInitialValues}, {rowInitialValues}, ...}
```

**As Array**

|                                                                                                                                                                                    |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Dim/Static/Private</b><br><b>arrayName</b><br><b>arrayName(,)</b><br><b>highestRowSub</b><br><br><b>highestColumnSub</b><br><br><b>As dataType</b><br><b>{rowInitialValues}</b> | <ul style="list-style-type: none"> <li>- variable declaration rules learned in previous lessons apply, so:</li> <li>- <b>Dim</b> = procedure-level array, declared in procedure, procedure scope</li> <li>- <b>Static</b> = procedure-level array, declared in procedure, class-level scope (used for accumulators)</li> <li>- <b>Private</b> = class-level array, declared in the form class's declarations section, class-level scope</li> </ul> <p>= version 1: two-dimensional array variable, with automatic initialization to <b>Nothing</b></p> <p>= version 2: two-dimensional array variable, with specified initialization values</p> <ul style="list-style-type: none"> <li>- <b>As Integer</b><br/>= specifies the zero-based highest row subscript in the array<br/>- i.e.: an array whose <b>highestRowSub = 2</b> will contain <b>3 rows</b></li> <li>- <b>As Integer</b><br/>= specifies the zero-based highest column subscript in the array<br/>- i.e.: an array whose <b>highestColumnSub = 2</b> will contain <b>3 columns</b></li> <li>- type of data the array variables (referred to as <b>elements</b>) will store</li> <li>- comma-separated list of values you want assigned to the row array elements</li> <li>- composed of column values separated by comma</li> <li>- e.g.: {{"r0c0", "r0c1", "r0c2"}, {"r1c0", "r1c1", "r1c2"}}</li> </ul> |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

e.g.1 - ver. 1:

```
Dim strStateCapitals(49, 1) As String
```

<- declares a **50-row, 2-column** procedure-level array named **strStateCapitals** => strStateCapitals(0, 0), strStateCapitals(0, 1),

strStateCapitals(1, 0), strStateCapitals(1, 1) ... ... ... strStateCapitals(49, 0), strStateCapitals(49, 1)

<- each element is automatically initialized using the keyword **Nothing**

<- same like: ...**As String** = **Nothing**, or ...**As String** = **String.Empty**

e.g.2 - ver. 1:

```
Static intNumSold(5, 4) As Integer
```

<- same like: ...As Integer = Nothing, or ...As Integer = 0  
<- declares a static, 6-row, 5-column procedure-level array named **intNumSold** => intNumSold(0, 0) ... ... ... intNumSold(5, 4)  
<- each element is automatically initialized using the keyword **Nothing**

e.g.3 - ver. 2:

```
Private strGrammy(,) As String = {{"2015", "57th", "Morning Phase", "Beck"},  
                                   {"2016", "58th", "1989", "Taylor Swift"},  
                                   {"2017", "59th", "25", "Adele"}}}
```

<- declares and initializes a 3-row, 4-column class-level array named **strGrammy** => strGrammy(0, 0) = "2015" ... strGrammy(3, 4) = "Adele"

e.g.4 - ver. 2:

```
Private dblSales(,) As Double = {{75.33, 9.65},  
                                  {23.55, 6.89},  
                                  {4.5, 89.3},  
                                  {100.67, 38.92}}
```

<- declares and initializes a 4-row, 2-column class-level array named **dblSales** => dblSales(0, 0) = 75.33 ... ... ... dblSales(4, 2) = 38.92

### CH8\_F7.3 - storing data in a 2D Array autopsy

- after an array is declared, you can use another statement to store a different value in an array element / array variable / element

e.g.1

```
Dim strStateCapitals(49, 1) As String  
strStateCapitals(0, 0) = "AL"  
strStateCapitals(0, 1) = "Montgomery"
```

<- declares a 50-row, 2-column procedure-level array named **strStateCapitals**  
<- assigns the string "AL" to the element located in 1st row, 1st column in the **strStateCapitals** array  
<- assigns the string "Montgomery" to the element located in 1st row, 2nd column

e.g.2

```
Static intNumSold(5, 4) As Integer  
For intRow As Integer = 0 To 5  
    For intColumn As Integer = 0 To 4  
        intNumSold(intRow, intColumn) += 1  
    Next intColumn  
Next intRow
```

<- declares a static, 6-row, 5-column procedure-level array named **intNumSold**

<- adds the number 1 to the contents of each element in the **intNumSold** array

e.g.3

```
Dim dblSales(,) As Double = {{75.33, 9.65}, {23.55, 6.89}, {4.5, 89.3}, {100.67, 38.92}}  
<- declares and initializes a 4-row, 2-column class-level array named dblSales => dblSales(0, 0) = 75.33 ... ... ... dblSales(4, 2) = 38.92  
Dim intRow As Integer  
Dim intCol As Integer  
Do While intRow <= 3  
    intCol = 0  
    Do While intCol <= 1  
        dblSales(intRow, intCol) *= 1.1  
        intCol += 1  
    Loop  
    intRow += 1  
Loop  
MessageBox.Show(dblSales(0, 0).ToString)
```

<- multiplies each element in the **dblSales** array by 1.1  
and stores the result in the element

<- 75.33 \* 1.1 = 82.863

e.g.4

```
Dim dblSales(,) As Double = {{75.33, 9.65}, {23.55, 6.89}, {4.5, 89.3}, {100.67, 38.92}}
    <- declares and initializes a 4-row, 2-column class-level array named dblSales => dblSales(0, 0) = 75.33 ... ... ... dblSales(4, 2) = 38.92
    dblSales(2, 1) *= 0.07
    Or
    dblSales(2, 1) = dblSales(2, 1) * 0.07
    <- multiplies the value contained in the 3rd row, 2nd column in the dblSales array by 0.07 and then assigns the result to the element
```

e.g.5

```
Double.TryParse(txtSales.Text, dblSales(0, 0))
    <- assigns either the value entered in the txtSales control converted to Double, or the number 0 to the element
        located in the 1st row, 1st column in the dblSales array
```

#### CH8\_F7.4 - determining the highest subscripts in a 2D Array: method **GetUpperBound(0)** for row and **GetUpperBound(1)** for column

- the zero-based highest subscript is always one number less than the number of array elements
- you can use the method **GetUpperBound(0)**:
  - **1D Array** - determine the highest **subscript**
  - **2D Array** - determine the highest **row subscript**
- you can use the method **GetUpperBound(1)**:
  - **2D Array** - determine the highest **column subscript**

2D Array - determine the highest **row subscript** method **GetUpperBound** syntax:  
highest **row subscript** **As Integer**

**arrayName.GetUpperBound(0) As Integer**

highest **row subscript As Integer**

2D Array - determine the highest **column subscript** method **GetUpperBound** syntax:  
highest **column subscript** **As Integer**

**arrayName.GetUpperBound(1) As Integer**

highest **column subscript As Integer**

**GetUpperBound**    - returns **As Integer**

- = gets the index of the last element of the specified dimension in the array
- (0) = zero-based **row** dimension of the array whose upper bound needs to be determined
- (1) = zero-based **column** dimension of the array whose upper bound needs to be determined

e.g.

```
Dim strOrders(10, 3) As String
Dim intHighestRowSub As Integer
Dim intHighestColumnSub As Integer
intHighestRowSub = strOrders.GetUpperBound(0)
intHighestColumnSub = strOrders.GetUpperBound(1)
```

- <- assigns the number **10** to the **intHighestRowSub** variable
- <- assigns the number **3** to the **intHighestColumnSub** variable

#### Mini-Quiz 8-3:

1. Write a statement that declares a class-level two-dimensional array named **intOrders**, containing **5 rows** and **3 columns**.
2. Write a statement that assigns the number **150** to the element located in the: **4th row, 3rd column** in the **intOrders** array.
3. Write a statement that assigns the number of **rows** in the **intOrders** array to the **intRows** variable.
4. Write a statement that assigns the highest **column subscript** in the **intOrders** array to the **intLastColSub** variable.

```
1...Private intOrders(4, 2) As Integer
2...intOrders(3, 2) = 150
3...intRows = intOrders.GetUpperBound(0) + 1
4...intLastColSub = intOrders.GetUpperBound(1)
```

## CH8\_F7.5 - Traversing a 2D Array = look at each element using loops: Do...Loop vs For...Next vs For Each...Next code examples

- traverse = look at each array element, one by one, from first to the last element
- recall that you use 1 loop to traverse a 1D Array
- to traverse a 2D Array: - typically use 2 loops: an outer loop and a nested loop - one keeps track of the row subscript, other keeps track of the column subscript
  - can use either For...Next or Do...Loop statements, or their combination
  - rather than using 2 loops, you can also use 1 loop For Each...Next statement, but:
    - recall that the instructions in a For Each...Next loop can only read the values = can't permanently modify them => therefore can't be used to fill elements with a data, but only for traversing

e.g.: 3 loops that traverse the **strMonths** array, displaying each element's value in the **lstMonths** control: (you can combine Do...Loop and For...Next)

<- note: the result of Do...Loop with 1st loop Column is different from Do...Loop with 1st loop Row and For...Next and For Each...Next

| <pre> Dim strMonths() As String = {{"Jan", "31"}, {"Feb", "28 or 29"}, {"Mar", "31"}, {"Apr", "30"}}  Dim intHighRow As Integer = strMonths.GetUpperBound(0) Dim intHighCol As Integer = strMonths.GetUpperBound(1) Dim intRow As Integer Dim intCol As Integer  Do While intRow &lt;= intHighRow     intCol = 0     Do While intCol &lt;= intHighCol         lstMonths.Items.Add(strMonths(intRow, intCol))         intCol += 1     Loop     intRow += 1 Loop </pre> | <pre> Dim intHighRow As Integer = strMonths.GetUpperBound(0) Dim intHighCol As Integer = strMonths.GetUpperBound(1)  For intRow As Integer = 0 To intHighRow     For intCol As Integer = 0 To intHighCol         lstMonths.Items.Add(strMonths(intRow, intCol))     Next intCol Next intRow </pre>                                                                          | <pre> For Each strElement As String In strMonths     lstMonths.Items.Add(strElement) Next strElement </pre>                                                                                                                                                                                                                                                                                                                               |  |         |         |      |             |            |      |             |                  |      |             |            |      |             |            |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--|---------|---------|------|-------------|------------|------|-------------|------------------|------|-------------|------------|------|-------------|------------|
| <pre> Do While intCol &lt;= intHighCol     intRow = 0     Do While intRow &lt;= intHighRow         lstMonths.Items.Add(strMonths(intRow, intCol))         intRow += 1     Loop     intCol += 1 Loop </pre>                                                                                                                                                                                                                                                            | <p>-&gt; <u>different output</u>:</p> <ul style="list-style-type: none"> <li>- <u>Do...Loop</u> with 1st loop <u>Column</u> and 2nd loop <u>Row</u>:</li> <li>- shows: Jan, Feb, Mar, Apr, 31, 28 or 29, 31, 30</li> <li>- like: r0c0, r0c1, r1c0, r1c1, r2c0, r2c1, r3c0, r3c1</li> <li>- i.e. based on <u>ascending columns</u> and then <u>ascending rows</u></li> </ul> | <table border="1" data-bbox="1689 897 2048 1354"> <thead> <tr> <th></th> <th>column0</th> <th>column1</th> </tr> </thead> <tbody> <tr> <td>row0</td> <td>0, 0<br/>Jan</td> <td>0, 1<br/>31</td> </tr> <tr> <td>row1</td> <td>1, 0<br/>Feb</td> <td>1, 1<br/>28 or 29</td> </tr> <tr> <td>row2</td> <td>2, 0<br/>Mar</td> <td>2, 1<br/>31</td> </tr> <tr> <td>row3</td> <td>3, 0<br/>Apr</td> <td>3, 1<br/>30</td> </tr> </tbody> </table> |  | column0 | column1 | row0 | 0, 0<br>Jan | 0, 1<br>31 | row1 | 1, 0<br>Feb | 1, 1<br>28 or 29 | row2 | 2, 0<br>Mar | 2, 1<br>31 | row3 | 3, 0<br>Apr | 3, 1<br>30 |
|                                                                                                                                                                                                                                                                                                                                                                                                                                                                       | column0                                                                                                                                                                                                                                                                                                                                                                     | column1                                                                                                                                                                                                                                                                                                                                                                                                                                   |  |         |         |      |             |            |      |             |                  |      |             |            |      |             |            |
| row0                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | 0, 0<br>Jan                                                                                                                                                                                                                                                                                                                                                                 | 0, 1<br>31                                                                                                                                                                                                                                                                                                                                                                                                                                |  |         |         |      |             |            |      |             |                  |      |             |            |      |             |            |
| row1                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | 1, 0<br>Feb                                                                                                                                                                                                                                                                                                                                                                 | 1, 1<br>28 or 29                                                                                                                                                                                                                                                                                                                                                                                                                          |  |         |         |      |             |            |      |             |                  |      |             |            |      |             |            |
| row2                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | 2, 0<br>Mar                                                                                                                                                                                                                                                                                                                                                                 | 2, 1<br>31                                                                                                                                                                                                                                                                                                                                                                                                                                |  |         |         |      |             |            |      |             |                  |      |             |            |      |             |            |
| row3                                                                                                                                                                                                                                                                                                                                                                                                                                                                  | 3, 0<br>Apr                                                                                                                                                                                                                                                                                                                                                                 | 3, 1<br>30                                                                                                                                                                                                                                                                                                                                                                                                                                |  |         |         |      |             |            |      |             |                  |      |             |            |      |             |            |

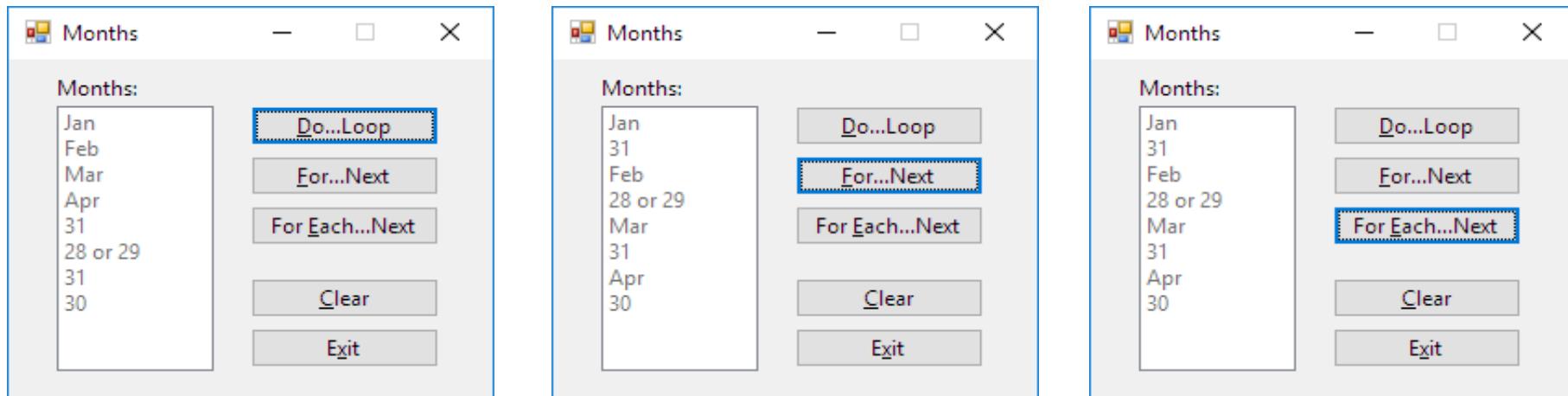
1). open the: ...VB2017\Chap08\Exercise\10.Months Solution\Months Solution.sln

- i modified the original example from only one loop into all 3 loops, so follow me :)

-> enter the code for each looping system - shown just previously - into appropriate procedures

2). start and test the application by clicking the buttons:

- <- **notice that:** - **Do...Loop** shows the result as: **Jan, Feb, Mar, Apr, 31, 28 or 29, 31, 30**  
 - **For...Next** shows the result as: **Jan, 31, Feb, 28 or 29, Mar, 31, Apr, 30**  
 - **For Each...Next** shows the result same as **For...Next**: **Jan, 31, Feb, 28 or 29, Mar, 31, Apr, 30**



3). completed code:

```

1  ' Display the contents of a 2D array in a list box, using 3 different loops:
2  Option Explicit On
3  Option Strict On
4  Option Infer Off
5
6  Public Class frmMain
7
8      Private strMonths() As String = {{"Jan", "31"}, {"Feb", "28 or 29"}, {"Mar", "31"}, {"Apr", "30"}}
9
10     Private Sub btnDoLoop_Click(sender As Object, e As EventArgs) Handles btnDoLoop.Click
11         Dim intHighRow As Integer = strMonths.GetUpperBound(0)
12         Dim intHighCol As Integer = strMonths.GetUpperBound(1)
13         Dim intRow As Integer
14         Dim intCol As Integer
15
16         lstMonths.Items.Clear()
17

```

```
18     Do While intCol <= intHighCol
19         intRow = 0 ' without this line it shows only: Jan, Feb, Mar, Apr
20         Do While intRow <= intHighRow
21             lstMonths.Items.Add(strMonths(intRow, intCol))
22             intRow += 1
23         Loop
24         intCol += 1
25     Loop
26 End Sub
27
28 Private Sub btnForNext_Click(sender As Object, e As EventArgs) Handles btnForNext.Click
29     Dim intHighRow As Integer = strMonths.GetUpperBound(0)
30     Dim intHighCol As Integer = strMonths.GetUpperBound(1)
31
32     lstMonths.Items.Clear()
33
34     For intRow As Integer = 0 To intHighRow
35         For intCol As Integer = 0 To intHighCol
36             lstMonths.Items.Add(strMonths(intRow, intCol))
37         Next intCol
38     Next intRow
39 End Sub
40
41 Private Sub btnForEachNext_Click(sender As Object, e As EventArgs) Handles btnForEachNext.Click
42
43     lstMonths.Items.Clear()
44
45     For Each strElement As String In strMonths
46         lstMonths.Items.Add(strElement)
47     Next strElement
48 End Sub
49
50 Private Sub btnClear_Click(sender As Object, e As EventArgs) Handles btnClear.Click
51     lstMonths.Items.Clear()
52 End Sub
53 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
54     Me.Close()
55 End Sub
56
57 End Class
```

## CH8\_F7.7 - Totaling the values stored in a 2D Array example: coding the Jenko Booksellers Application (11.Jenko Solution)

- in this section, you will finish coding the **Jenko Booksellers Application**, which displays the total sales made in the company's 3 stores
- the sales amounts are stored in a 2D array that has 3 rows and 2 columns, where:
  - each **row** contains the sales amounts for one of the 3 stores
  - **1st column** contains the sales of paperback books
  - **2nd column** contains the sales of hardcover books

1). open the: ...VB2017\Chap08\Exercise\11.Jenko Solution\Jenko Solution.sln

2). open the Code Editor window and locate the **btnCalc\_Click** procedure

- first, the procedure will declare and initialize a **2D Array** to store the sales amounts:

-> enter the array declaration statement shown on line **14**:

- the procedure will also declare a variable that it can use to accumulate the sales amounts stored in the array

-> enter the scalar variable declaration statement shown on line **15**:

```
12      ' Displays the total sales:  
13  
14      Dim intSales(,) As Integer = {{1500, 2535}, {2300, 3675}, {1850, 2475}}  
15      Dim intTotal As Integer  
16
```

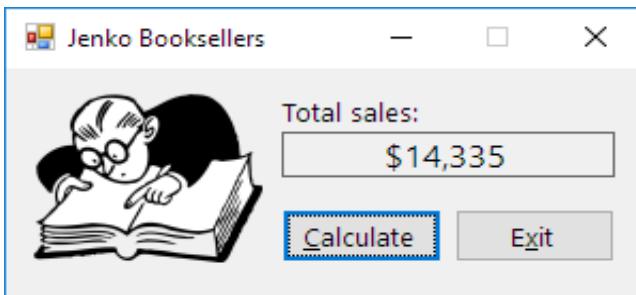
3). the procedure will use a **For Each...Next** loop to total the values stored in the array and it will then display the total sales in the **lblTotal** control

-> enter the loop and assignment statement:

```
16  
17      For Each intElement As Integer In intSales  
18          intTotal += intElement  
19      Next intElement  
20  
21      lblTotal.Text = intTotal.ToString("C0")  
22
```

4). save the solution and then start and test the application

-> click the button **Calculate** - the total sales amount is **\$14,335**



5). completed code:

```
1   ' Totaling the values stored in a 2D Array:  
2   Option Explicit On  
3   Option Strict On  
4   Option Infer Off  
5  
6   Public Class frmMain  
7       Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click  
8           Me.Close()  
9       End Sub  
10  
11      Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click  
12          ' Displays the total sales.  
13  
14          Dim intSales(,) As Integer = {{1500, 2535}, {2300, 3675}, {1850, 2475}}  
15          Dim intTotal As Integer  
16  
17          For Each intElement As Integer In intSales  
18              intTotal += intElement  
19          Next intElement  
20  
21          lblTotal.Text = intTotal.ToString("C0")  
22  
23      End Sub  
24  End Class
```

**CH8\_F7.8 - You Do It 5: Totaling 2D Array value: Do...Loop vs For...Next vs For Each...Next exercise: 12.You Do It 5 Solution**

1. create an application named **You Do It 5** and save it in the ...VB2017\Chap08\Exercise\12.You Do It 5 Solution
2. add: a **button**, and **3x labels** to the form
3. the button's **Click** event procedure should declare and initialize a **2D integer Array** that contains **3 rows** and **2 columns**, using any **6 integers** to initialize the array
4. the procedure should total the **6 integers** and then display the result in three labels in a way that:
  - > **1st** label should display the calculated total using the **Do...Loop** statements
  - > **2nd** label should display the calculated total using the **For...Next** statements
  - > **3rd** label should display the calculated total using the **ForEach...Next** statement
5. code the procedure, save the solution and then start and test the application.

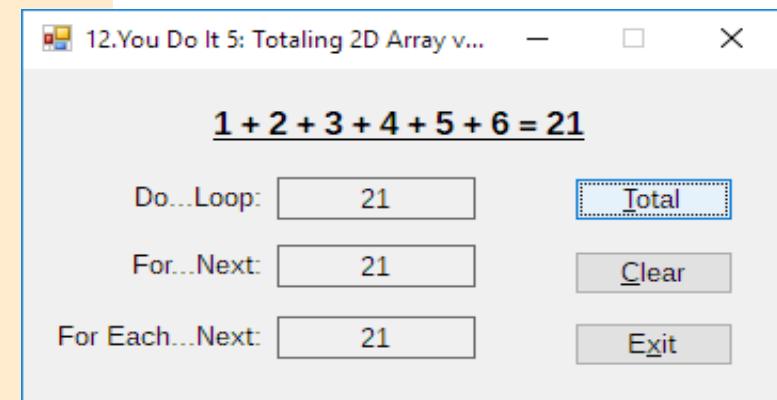
my GUI & code:

```
1   ' 12.You Do It 5: Totaling 2D Array value using Do...Loop vs For...Next vs For Each Next:  
2   Option Explicit On  
3   Option Strict On  
4   Option Infer Off  
5
```

```

6  Public Class frmMain
7      Private Sub btnTotal_Click(sender As Object, e As EventArgs) Handles btnTotal.Click
8
9          Dim intNumbers(,) As Integer = {{1, 2}, {3, 4}, {5, 6}}
10
11         ' 1.Do...Loop x2:
12         Dim intTotal1 As Integer
13         Dim intMaxRow As Integer = intNumbers.GetUpperBound(0)
14         Dim intMaxCol As Integer = intNumbers.GetUpperBound(1)
15         Dim intRow As Integer
16         Dim intCol As Integer
17
18         ' 1.1. Row as 1st: 1+2+3+4+5+6
19         Do While intRow <= intMaxRow
20             intCol = 0
21             Do While intCol <= intMaxCol
22                 intTotal1 += intNumbers(intRow, intCol)
23                 lbl1DoLoop.Text = intTotal1.ToString
24                 intCol += 1
25             Loop
26             intRow += 1
27         Loop
28
29         ' 1.2. Column as 1st: 1+3+5+2+4+6
30         Do While intCol <= intMaxCol
31             intRow = 0
32             Do While intRow <= intMaxRow
33                 intTotal1 += intNumbers(intRow, intCol)
34                 lbl1DoLoop.Text = intTotal1.ToString
35                 intRow += 1
36             Loop
37             intCol += 1
38         Loop
39
40         ' 2.For...Next x2:
41         Dim intTotal2 As Integer
42         For intRowArray As Integer = 0 To intNumbers.GetUpperBound(0)
43             For intColumnArray As Integer = 0 To intNumbers.GetUpperBound(1)
44                 intTotal2 += intNumbers(intRowArray, intColumnArray)
45                 lbl2ForNext.Text = intTotal2.ToString
46                 Next intColumnArray
47             Next intRowArray
48
49         ' 3.For Each...Next x1:

```



```
50     Dim intTotal3 As Integer
51         For Each intElement As Integer In intNumbers
52             intTotal3 += intElement
53             lbl3ForEachNext.Text = intTotal3.ToString
54         Next intElement
55
56     End Sub
57     Private Sub btnClear_Click(sender As Object, e As EventArgs) Handles btnClear.Click
58         lbl1DoLoop.Text = String.Empty
59         lbl2ForNext.Text = String.Empty
60         lbl3ForEachNext.Text = String.Empty
61     End Sub
62
63     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
64         Me.Close()
65     End Sub
66
67 End Class
```

### Mini-Quiz 8-4:

1. The code to traverse a **2D Array** using the **Do...Loop** statement requires 2 loops. True or False?
  2. The code to traverse a **2D Array** using the **For...Next** statement requires 2 loops. True or False?
  3. The code to traverse a **2D Array** using the **For Each...Next** statement requires 2 loops. True or False?
  4. When using 2 loops to traverse an **Array**, you can use the **For...Next** statement to code the outer loop and use the **For Each...Next** statement to code the nested loop. True or False?

三

CH8 APPLY THE CONCEPTS LESSON

**CH8\_A1.1** - Associate a **1D Array** with a **(1st)** i.e. associate **Array elements** with **Items Collection**, for they have a several **commonalities**

- it isn't uncommon for coders to associate the **items** in a **(List)** with the values stored in an **Array**
  - this is because the items in a **(List)** belong to a collection = Items collection, and collections and arrays have several things in **common**:

- 1...each is a group of individual objects treated as one unit**

- 2...each individual object in the group is identified by a **unique number**:** - called an **Index** when referring to a **collection**  
- called a **subscript** when referring to an **array**

- 3...both are zero-based**

- these commonalities allow you to associate the list box items and array elements by their positions within their respective groups

- to associate **(Ist)** with **Array**: 1). add the appropriate items to the **(Ist)**  
2). store each item's related value in its corresponding position in the **Array**

e.g.: `strVPs(1stPresidents.SelectedIndex)`

- relationship between the items in the **1stPresidents** control and the elements in a **1D Array** named **strVPs**:

e.g.

| <b>1stPresidents</b> | <- index / subscript -> | <b>strVPs</b>   | <b>1D Array</b> |
|----------------------|-------------------------|-----------------|-----------------|
| George Washington    | 0                       | John Adams      |                 |
| George Bush          | 1                       | Dan Quayle      |                 |
| Bill Clinton         | 2                       | Albert Gore     |                 |
| George W. Bush       | 3                       | Richard Cheney  |                 |
| Barack Obama         | 4                       | Joseph R. Biden |                 |
| Donald J. Trump      | 5                       | Mike Pence      |                 |

e.g.: `lblVicePres.Text = strVPs(1stPresidents.SelectedIndex)`

### CH8\_A1.2 - Associate a **1D Array** with a **(1st)** example: coding the **Presidents and Vice Presidents Application** (13.Presidents Solution)

- the application displays the list with the names of presidents
- by selecting any name, the label should display the appropriate vice president

<- **(1st)** items collection

<- **1D Array** elements

1). open the: ...VB2017\Chap08\Exercise\13.Presidents Solution\Presidents Solution.sln

2). open the Code Editor window

-> notice that the **form class**'s declaration section contains a **Private** statement that declares and initializes the **strVPs Array**:

```
8     ' Class-level array:
9     Private strVPs() As String = {"John Adams", "Dan Quayle", "Albert Gore", "Richard Cheney", "Joseph R. Biden", "Mike Pence"}
```

-> notice that **frmMain\_Load** procedure adds the names to the **1stPresidents** control and then selects the 1st name in the list:

```
11    Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
12        ' Fill list box with names of presidents and select first name:
13
14        1stPresidents.Items.Add("George Washington")
15        1stPresidents.Items.Add("George Bush")
16        1stPresidents.Items.Add("Bill Clinton")
17        1stPresidents.Items.Add("George W. Bush")
18        1stPresidents.Items.Add("Barack Obama")
19        1stPresidents.Items.Add("Donald J. Trump")
20        1stPresidents.SelectedIndex = 0
21    End Sub
```

3). locate the procedure **1stPresidents\_SelectedIndexChanged** on line **23**

- when the user selects a president's name in the list box, the procedure should display the appropriate vice president's name in the **Vice President** box

<- you can use the **index** of the **selected item** to access the appropriate name from the **1D Array**: **strVPs**

-> type the assignment statement shown on line **26**:

```
23    Private Sub 1stPresidents_SelectedIndexChanged(sender As Object, e As EventArgs) Handles 1stPresidents.SelectedIndexChanged
24        ' Display associated name from strVPs array:
25
26        lblVicePres.Text = strVPs(1stPresidents.SelectedIndex)
27    End Sub
```

4). save the solution and then start and test the application

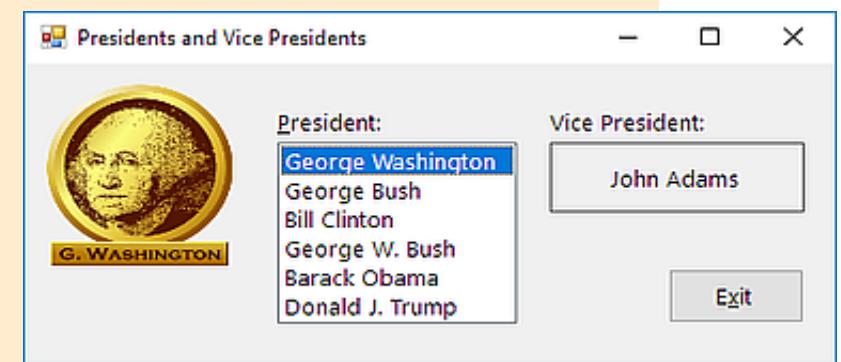
- the 1st item in the list box - George Washington is already selected, and the name of his vice president - John Adams appears in the **Vice President** box

-> verify that the application displays the appropriate vice president's name for the remaining list box items:

George Bush = Dan Quayle, Bill Clinton = Albert Gore, George W. Bush = Richard Cheney, Barack Obama = Joseph R. Biden, Donald J. Trump = Mike Pence

5). completed code & GUI:

```
1  ' Displays the name of a president's vice president.
2  Option Explicit On
3  Option Strict On
4  Option Infer Off
5
6  Public Class frmMain
7
8      ' Class-level array:
9      Private strVPs() As String = {"John Adams", "Dan Quayle", "Albert Gore", "Richard Cheney", "Joseph R. Biden", "Mike Pence"}
10
11     Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
12         ' Fill list box with names of presidents and select first name:
13
14         lstPresidents.Items.Add("George Washington")
15         lstPresidents.Items.Add("George Bush")
16         lstPresidents.Items.Add("Bill Clinton")
17         lstPresidents.Items.Add("George W. Bush")
18         lstPresidents.Items.Add("Barack Obama")
19         lstPresidents.Items.Add("Donald J. Trump")
20
21         lstPresidents.SelectedIndex = 0
22
23     Private Sub lstPresidents_SelectedIndexChanged(sender As Object, e As EventArgs) Handles lstPresidents.SelectedIndexChanged
24         ' Display associated name from strVPs array:
25
26         lblVicePres.Text = strVPs(lstPresidents.SelectedIndex)
27
28
29     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
30         Me.Close()
31
32     End Sub
33
34 End Class
```



## CH8\_A2.1 - create 1D Array Accumulator and 1D Array Counter associated with (1st) example: Chocolate Bars Application (14.Warren Solution)

- 1D Arrays are often used to:
  - accumulate related values <- commonly referred to as **accumulator Array**
  - count related values <- commonly referred to as **counter Array**

- the **Warren School application**, which you code next, uses an **accumulator Array** to keep track of the number of candy bars sold by each student
- GUI provides:
  - (**1stCandy**) for selecting the candy type
  - (**txtSold**) for entering the number sold by a student
  - (**btnAdd**) will accumulate the numbers sold and then display the totals by candy type

1). open the: ...VB2017\Chap08\Exercise\14.Warren Solution\Warren Solution.sln

2). open the Code Editor window

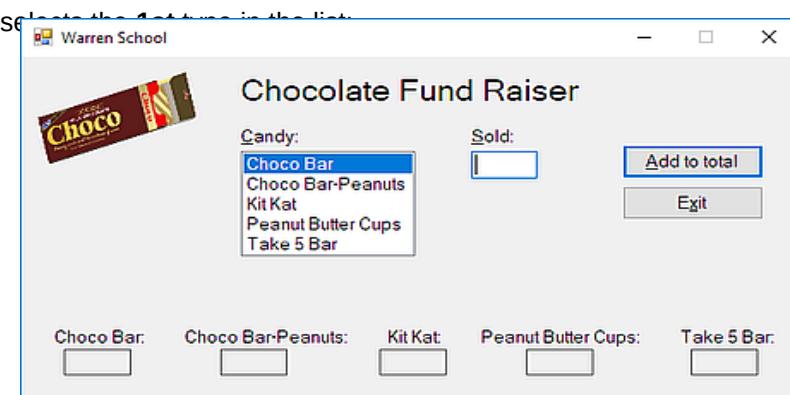
> notice the **txtSold\_KeyPress** procedure:

- the procedure allows the (**txtSold**) to accept only:
  - numbers
  - hyphen <- necessary in case the user needs to make a **correction** to the amount sold, by **subtracting** a number from the amount sold
  - Backspace key

```
x     Private Sub txtSold_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtSold.KeyPress
x         ' Accept only numbers, the hyphen, and the Backspace:
x
x         If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> "-" AndAlso e.KeyChar <> ControlChars.Back Then
x             e.Handled = True
x         End If
x     End Sub
```

> notice that the **frmMain\_Load** procedure fills the (**1stCandy**) with 5 types of candy and then selects the first value.

```
x     Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
x         ' Fills the list box with values and then selects the first value:
x
x         1stCandy.Items.Add("Choco Bar")
x         1stCandy.Items.Add("Choco Bar-Peanuts")
x         1stCandy.Items.Add("Kit Kat")
x         1stCandy.Items.Add("Peanut Butter Cups")
x         1stCandy.Items.Add("Take 5 Bar")
x
x         1stCandy.SelectedIndex = 0
x     End Sub
```



3). locate the **btnAdd\_Click** procedure on line 17:

- the procedure will declare a 1D **Accumulator Array** named **intCandy** with 5 elements, where each element:
  - will correspond to a candy type listed in (**1stCandy**)
  - will be used to accumulate the sales of its corresponding item in the (**1stCandy**)

> below the comment on line 20, enter the **Static** declaration statement:

```
20     ' Declare array and variable:
21     Static intCandy(4) As Integer
```

<- the **intCandy 1D accumulator Array** will need to retain its values until the application ends, so to accomplish this you can either:  
a). declare it in the **form class**'s declaration section using: **Private intCandy(4) As Integer** to make it a **class-level Array**  
b). declare it in the **btnAdd\_Click** procedure using: **Static intCandy(4) As Integer** to make it a **static procedure-level Array**

- in addition to the **Array**, the procedure will use an **Integer** variable to store the amount sold in **intSold**
- the procedure needs to convert the **txtSold.Text** property to **Integer** and store the result in the **intSold** variable, using **TryParse** method  
-> type the following declaration statement and **TryParse** method on lines **22 & 23**:

```
22      Dim intSold As Integer
23      Integer.TryParse(txtSold.Text, intSold)
```

- 4). the procedure will use the **index** of the item selected in the (**lstCandy**) to update the appropriate **Array** element

-> below the comment on line **25**, enter the following assignment statement:

```
25      ' Update array value:
26      intCandy(lstCandy.SelectedIndex) += intSold
```

- 5). finally, the procedure will display the **Array** values in the GUI

-> below the comment on line **28**, enter the following **5** assignment statements:

```
28      ' Display array values:
29      lblChocoBar.Text = intCandy(0).ToString
30      lblChocoBarPeanuts.Text = intCandy(1).ToString
31      lblKitKat.Text = intCandy(2).ToString
32      lblPeanutButCups.Text = intCandy(3).ToString
33      lblTake5Bar.Text = intCandy(4).ToString
34
35      txtSold.Focus()
```

- 6). save the solution and then start and test the application:

-> in the **Sold** box type: **100** and then press **Enter** to select the default button **Add to total**

<- the number **100** appears in the **Choco Bar** box

-> in the **Candy** box select **Kit Kat**, and in the **Sold** box change the **100** to **45** and then press **Enter**

<- the number **45** appears in the **Kit Kat** box

-> now, in the **Sold** box change the number **45** to **-6** and then press **Enter**

<- the number **39** appears in the **Kit Kat** box

-> record the following 3 candy sales: **36** of the **Peanut Butter Cups**, **10** of the **Take 5 Bar**, **2** of the **Choco Bar-Peanut**

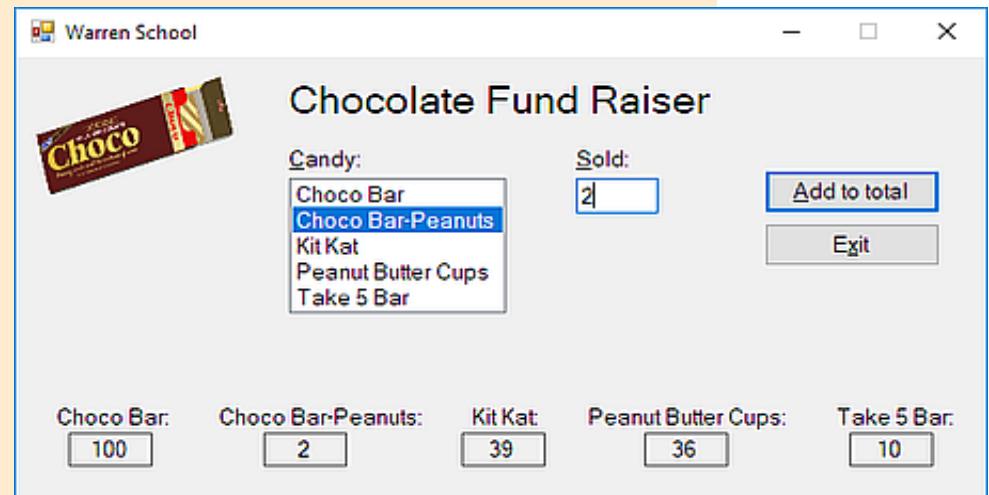
- 7). completed code & GUI:

```
1  Option Explicit On
2  Option Strict On
3  Option Infer Off
4
5  Public Class frmMain
6      Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
7          ' Fills the list box with values and then selects the first value:
8
```

```

9      lstCandy.Items.Add("Choco Bar")
10     lstCandy.Items.Add("Choco Bar-Peanuts")
11     lstCandy.Items.Add("Kit Kat")
12     lstCandy.Items.Add("Peanut Butter Cups")
13     lstCandy.Items.Add("Take 5 Bar")
14
15     lstCandy.SelectedIndex = 0
16
17  End Sub
18
19
20  Private Sub btnAdd_Click(sender As Object, e As EventArgs) Handles btnAdd.Click
21      ' Adds the amount sold to the appropriate total:
22
23      ' Declare array and variable:
24      Static intCandy(4) As Integer
25      Dim intSold As Integer
26      Integer.TryParse(txtSold.Text, intSold)
27
28      ' Update array value:
29      intCandy(lstCandy.SelectedIndex) += intSold
30
31      ' Display array values:
32      lblChocoBar.Text = intCandy(0).ToString
33      lblChocoBarPeanuts.Text = intCandy(1).ToString
34      lblKitKat.Text = intCandy(2).ToString
35      lblPeanutButCups.Text = intCandy(3).ToString
36      lblTake5Bar.Text = intCandy(4).ToString
37
38      txtSold.Focus()
39
40  End Sub
41
42  Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
43      Me.Close()
44
45
46  Private Sub txtSold_Enter(sender As Object, e As EventArgs) Handles txtSold.Enter
47      txtSold.SelectAll()
48
49
50  Private Sub txtSold_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtSold.KeyPress
51      ' Accept only numbers, the hyphen, and the Backspace:
52
53      If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> "-" AndAlso e.KeyChar <> ControlChars.Back Then
54          e.Handled = True
55
56      End If
57
58  End Sub

```



```

53
54     Private Sub ClearOutput(sender As Object, e As EventArgs) Handles txtSold.TextChanged, lstCandy.SelectedIndexChanged
55         lblChocoBar.Text = String.Empty
56         lblChocoBarPeanuts.Text = String.Empty
57         lblKitKat.Text = String.Empty
58         lblPeanutButCups.Text = String.Empty
59         lblTake5Bar.Text = String.Empty
60     End Sub
61 End Class

```

### Mini-Quiz 8-5:

1. Write a statement that updates the **2nd element** in the **1D Array intOrders** by the value stored in the **intSold** variable.
2. Write a statement that subtracts the number **10** from the **1st element** in the **1D Array intOrders**.
3. The items in the **1D Array intOrders** are associated with the items listed in the **lstProducts** control.

Write a statement that assigns the array value associated with the selected list box item to the **intQuantity** variable.

```

1...intOrders(1) += intSold
2...intOrders(0) -= 10 or intOrders(0) += -10
3...intQuantity = intOrders(lstProducts.SelectedIndex)

```

### CH8\_A2.2 - You Do It 6: 1D Array Accumulator & Counter associated with (lst) using For Each...Next & For...Next & Do...Loop

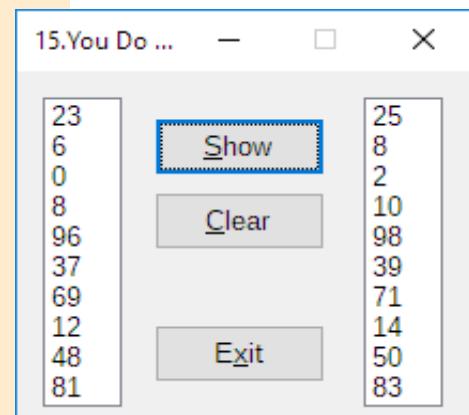
1. create an application named **You Do It 6** and save it in the ...VB2017\Chap08\Exercise15.You Do It 6 Solution
2. add: **2x (lst)** and a (**btn**) to the form
3. the buttons **Click** event procedure should declare and initialize a **1D integer Array**, using any **10** integers to initialize the **Array**
4. the procedure should use the **For Each...Next** statement to display the contents of the **Array** in the **1st (lst)**
5. the procedure should then use the **For...Next** statement to increase each **Array** element's value by **2**
6. finally, it should use the **Do...Loop** statement to display the updated results in the **2nd (lst)**

my GUI & code:

```

1  ' 15.You Do It 6: 1D Accumulator & Counter associated with (lst) using For Each...Next & For...Next & Do...Loop
2  Option Explicit On
3  Option Strict On
4  Option Infer Off
5
6  Public Class frmMain
7      Private Sub btnShow_Click(sender As Object, e As EventArgs) Handles btnShow.Click
8          lst1.Items.Clear()
9          lst2.Items.Clear()
10
11         ' 3: declare 1D Array and initialize it to any 10 integers:
12         Dim intArray() As Integer = {23, 6, 0, 8, 96, 37, 69, 12, 48, 81}
13
14         ' 4: display the Array contents in 1st (lst) using For Each...Next:
15         For Each intElement As Integer In intArray

```



```

16         lst1.Items.Add(intElement)
17     Next intElement
18
19     ' 5: increase each Array element by 2 using For...Next:
20     For intCounter As Integer = 0 To intArray.GetUpperBound(0)
21         intArray(intCounter) += 2
22     Next intCounter
23
24     ' 6: display the updated Array in 2nd (lst) using Do...Loop:
25     Dim intCount As Integer
26     Do While intCount <= intArray.GetUpperBound(0)
27         lst2.Items.Add(intArray(intCount))
28         intCount += 1
29     Loop
30
31 End Sub
32
33 Private Sub btnClear_Click(sender As Object, e As EventArgs) Handles btnClear.Click
34     lst1.Items.Clear()
35     lst2.Items.Clear()
36 End Sub
37
38 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
39     Me.Close()
40 End Sub
41 End Class

```

### CH8\_A3.1 - create 2x Parallel 1D Arrays introduction & example

- knowing that all of the variables in an **Array** must **have the same data type**,  
how do you store a price list composed of a product ID (**string**) and price (**number**) in an **Array**?

- one solution is to use: **2x Parallel 1D Arrays**: (other solution is to use **String 2D Array**) [CH8\\_A4.1 ...](#)

e.g.

| strIds()  |        | <- index -> |  | dblPrices() |              |
|-----------|--------|-------------|--|-------------|--------------|
|           |        | 0           |  | 8.99        | dblPrices(0) |
| strIds(0) | "A45G" | 1           |  | 12.99       | dblPrices(1) |
| strIds(1) | "J63Y" | 2           |  | 5.99        | dblPrices(2) |
| strIds(2) | "M93K" | 3           |  | 13.5        | dblPrices(3) |
| strIds(3) | "C20P" | 4           |  | 7.25        | dblPrices(4) |
| strIds(4) | "F77T" |             |  |             |              |

- a **String 1D Array** to store the IDs and **Double 1D Array** to store the prices
- the **Arrays** are **parallel** because each element in the **strIds Array** corresponds to the element located in the same position in the **dblPrices Array**
  - e.g.1: the price of item **A45G** is **8.99** -> **strIds(0)**, **dblPrices(0)**
  - e.g.2: the price of item **M93K** is **5.99** -> **strIds(2)**, **dblPrices(2)**
- to determine an item's price, you locate the item's ID in the **strIds Array** and then view its corresponding element in the **dblPrices Array**

## CH8\_A3.2 - create 2x Parallel 1D Arrays example: coding the Paper Warehouse Application (16.Paper Solution-Parallel)

- you will use the **two parallel Arrays** from previous example in the **Paper Warehouse application**
- the application will search the **strIds Array** for the **product ID** entered by the user and then display the corresponding **price** from the **dblPrices Array**

pseudocode for the **btnGet\_Click** procedure:

- step 1:** declare **strSearchId** variable to store the ID entered in the **txtId.Text** property  
**step 2:** declare **intSub** variable to keep track of **Array** subscripts  
**step 3:** assign **txtId.Text** property to **strSearchId**  
**step 4:** loop that searches each element in the **strIds Array**, stopping either when the end of the **Array** is reached or when the **ID** is located in the **Array**  
    repeat until all of the elements in the **strIds Array** have been searched or the search **ID** is located in the **Array**  
        **add 1** to **intSub** so the loop can search the next element in the **strIds Array**  
    end repeat  
**step 5:** selection structure whose condition determines why the loop ended by looking at the value in the **intSub** variable  
**step 5.1:** if the search **ID** was located in the **strIds Array**  
    **display the price**, which is contained in the same location in the **dblPrices Array**, in **lblPrice**  
    else  
        **display "ID not found." message** in a message box  
    end if

1). open the: ...VB2017\Chap08\Exercise\16.Paper Solution-Parallel\Paper Solution.sln

2). open the Code Editor window

-> notice that the code to declare **two parallel 1D Arrays** is already entered in the form's declaration section:

```
5  Public Class frmMain
6      ' Declare parallel arrays:
7      Private strIds() As String = {"A45G", "J63Y", "M93K", "C20P", "F77T"}
8      Private dblPrices() As Double = {8.99, 12.99, 5.99, 13.5, 7.25}
```

3). locate the **btnGet\_Click** procedure on line 10

-> notice that the procedure already contains code for:   **step 1:**   **step 2:**   **step 3:**

```
10     Private Sub btnGet_Click(sender As Object, e As EventArgs) Handles btnGet.Click
11         ' Displays an item's price.
12
13         Dim strSearchId As String
14         Dim intSub As Integer
15
16         strSearchId = txtId.Text.Trim.ToUpper
17
```

**step 1:** declare **strSearchId** variable to store the ID entered in the **txtId.Text** property

**step 2:** declare **intSub** variable to keep track of **Array** subscripts

**step 3:** assign **txtId.Text** property to **strSearchId**

**step 4:** loop that searches each element in the **strIds Array**, stopping either when the end of the **Array** is reached or when the **ID** is located in the **Array**

-> enter the loop:

```
18         ' Search the strIds array until the end of the array or the ID is found:
19         Do Until intSub = strIds.Length OrElse strIds(intSub) = strSearchId
20             intSub += 1
21         Loop
```

**step 5:** selection structure whose condition determines why the loop ended by looking at the value in the **intSub** variable

- if the loop ended because it located the **ID** in the **strIds Array**, the **intSub** variable's value will be less than the **Array's length**
- if the loop ended because it reached the end of the **strIds Array** without locating the **ID**, the **intSub** variable's value will be equal to the **Array's length**
  - <- recall that an **Array's length** is always **1 number more** than its **last subscript**

-> enter the following **If** clause:

```
22  
23     If intSub < strIds.Length Then  
24  
25     End If
```

**step 5.1** - if the selection structure's condition evaluates to **True**, it means that the **ID** was located in the **strIds Array**

-> therefore the condition should display in the **lblPrice** control the **price** located in the same position in the **dblPrices Array**

- if the selection structure's condition evaluates to **False**, it means that the **ID** wasn't located in the **strIds Array**

-> therefore the condition should display the message: "ID not found."

-> enter additional code for the **If** clause:

```
23     If intSub < strIds.Length Then  
24         lblPrice.Text = dblPrices(intSub).ToString("C2")  
25     Else  
26         MessageBox.Show("ID not found.", "Paper Warehouse", MessageBoxButtons.OK, MessageBoxIcon.Information)  
27     End If
```

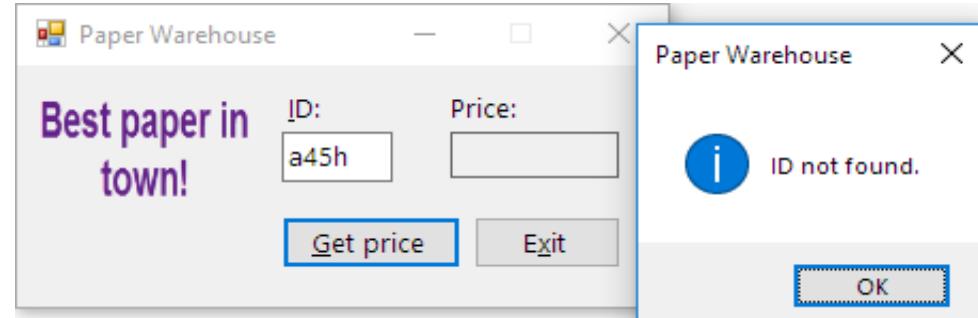
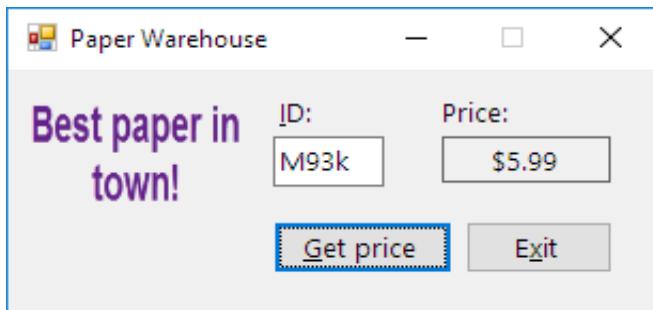
4). save the solution and then start and test the application:

-> in the **ID** box type: **M93k** and then click the button **Get price**

-> the **Price** box should show: **\$5.99**

-> in the **ID** box type: **a45h** and then click the button **Get price**

-> the Message box "ID not found" should appear



5). completed code:

```
1  Option Explicit On  
2  Option Strict On  
3  Option Infer Off  
4
```

```
5  Public Class frmMain
6      ' Declare parallel arrays:
7      Private strIds() As String = {"A45G", "J63Y", "M93K", "C20P", "F77T"}
8      Private dblPrices() As Double = {8.99, 12.99, 5.99, 13.5, 7.25}
9
10     Private Sub btnGet_Click(sender As Object, e As EventArgs) Handles btnGet.Click
11         ' Displays an item's price.
12
13         Dim strSearchId As String
14         Dim intSub As Integer
15
16         strSearchId = txtId.Text.Trim.ToUpper
17
18         ' Search the strIds array until the end of the array or the ID Is found:
19         Do Until intSub = strIds.Length OrElse strIds(intSub) = strSearchId
20             intSub += 1
21         Loop
22
23         If intSub < strIds.Length Then
24             lblPrice.Text = dblPrices(intSub).ToString("C2")
25         Else
26             MessageBox.Show("ID not found.", "Paper Warehouse", MessageBoxButtons.OK, MessageBoxIcon.Information)
27         End If
28
29     End Sub
30
31     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
32         Me.Close()
33     End Sub
34
35     Private Sub txtId_Enter(sender As Object, e As EventArgs) Handles txtId.Enter
36         txtId.SelectAll()
37     End Sub
38
39     Private Sub ClearPrice(sender As Object, e As EventArgs) Handles txtId.TextChanged
40         lblPrice.Text = String.Empty
41     End Sub
42 End Class
```

## CH8\_A4.1 - search 2D Array introduction & example vs. 2x Parallel 1D Arrays

- in the previous section, you used **2x Parallel 1D Arrays** to code the **Paper Warehouse application**:
  - a **String 1D Array** for the item **IDs** and a **Double 1D Array** for the corresponding **prices**
- instead of storing the price list in **2x Parallel 1D Arrays**, you can store it in a **2D Array**, with:
  - 1st column storing the **IDs** and the 2nd column storing the **prices**
  - **BUT**: you need to treat the **prices** as **Strings** because all of the data in an **Array** must have the same data type

- String 2D Array vs 2x Parallel 1D Array example with Paper Warehouse Application:

| e.g. | String 2D Array |        |          | String 1D Array |             | Double 1D Array |             |              |
|------|-----------------|--------|----------|-----------------|-------------|-----------------|-------------|--------------|
|      | strItems(,)     |        | strIds() |                 | <- index -> |                 | dblPrices() |              |
|      | strItems(0, 0)  | "A45G" | "8.99"   | strItems(0, 1)  |             | 0               | 8.99        | dblPrices(0) |
|      | strItems(1, 0)  | "J63Y" | "12.99"  | strItems(1, 1)  |             | 1               | 12.99       | dblPrices(1) |
|      | strItems(2, 0)  | "M93K" | "5.99"   | strItems(2, 1)  |             | 2               | 5.99        | dblPrices(2) |
|      | strItems(3, 0)  | "C20P" | "13.50"  | strItems(3, 1)  |             | 3               | 13.5        | dblPrices(3) |
|      | strItems(4, 0)  | "F77T" | "7.25"   | strItems(4, 1)  |             | 4               | 7.25        | dblPrices(4) |
|      | IDs             | prices |          | IDs             |             |                 | prices      |              |

## CH8\_A4.2 - search 2D Array instead of 2x Parallel 1D Arrays example: coding the Paper Warehouse Application (17.Paper Solution-TwoDim)

- you will use **2D Array** instead of **two parallel 1D Arrays** from previous example in the **Paper Warehouse application**

pseudocode for the **btnGet\_Click** procedure:

- step 1:** declare **strSearchId** variable to store the ID entered in the **txtId.Text** property  
**step 2:** declare **intRow** variable to keep track of the **Array's row subscripts**  
**step 3:** assign **txtId.Text** property to **strSearchId**  
**step 4:** loop that searches each element/row in the 1st column in the **strItems 2D Array**,  
stopping either when all of the elements/rows in the 1st column have been searched or when the **ID** is located in the 1st column  
repeat until all of the elements-rows in the **strItems Array's 1st column** have been searched or the search **ID** is located in the 1st column  
    **add 1** to **intRow** so the loop can search the next element-row in the **Array's 1st column**  
end repeat  
**step 5:** selection structure whose condition determines why the loop ended by looking at the value in the **intRow** variable  
**step 5.1:** if the search **ID** was located in the **Array's 1st column**  
        **display the price**, which is contained in the same **row** as the **ID**, but in the 2nd column, in **lblPrice**  
    else  
        **display "ID not found." message** in a message box  
    end if

1). open the: ...VB2017\Chap08\Exercise\17.Paper Solution-TwoDim\Paper Solution.sln

2). open the Code Editor window

-> notice that the code to declare **2D Array** is already entered in the form's declaration section:

```
5  Public Class frmMain
6      ' Declare two-dimensional array:
7      Private strItems(,) As String = {{"A45G", "8.99"}, {"J63Y", "12.99"}, {"M93K", "5.99"}, {"C20P", "13.50"}, {"F77T", "7.25"}}
```

### 3). locate the **btnGet\_Click** procedure on line 9

-> notice that the procedure already contains code for:

step 1: step 2: step 3:

```
8  
9      Private Sub btnGet_Click(sender As Object, e As EventArgs) Handles btnGet.Click  
10     ' Displays an item's price.  
11  
12     Dim strSearchId As String  
13     Dim intRow As Integer  
14  
15     strSearchId = txtId.Text.Trim.ToUpper  
16
```

- step 1: declare **strSearchId** variable to store the ID entered in the **txtId.Text** property  
step 2: declare **intRow** variable to keep track of the **Array**'s row **subscripts**  
step 3: assign **txtId.Text** property to **strSearchId**

#### step 4: loop that searches each element/row in the 1st column in the **strItems 2D Array**,

stopping either when all of the elements/rows in the 1st column have been searched or when the **ID** is located in the 1st column

-> enter the loop:

```
17     ' Search the first column for the ID. Continue searching until the end of the first column Or the ID Is found:  
18     Do Until intRow > strItems.GetUpperBound(0) orElse strItems(intRow, 0) = strSearchId  
19         intRow += 1  
20     Loop  
21
```

#### step 5: selection structure whose condition determines why the loop ended by looking at the value in the **intRow** variable

- if the loop ended because it located the **ID** in **Array**'s 1st column, the **intRow** variable's value will be less or equal to the highest row **subscript**
- if the loop ended because it reached the end of the **Array**'s 1st column without locating the **ID**,  
the **intRow** variable's value will be greater than the highest row **subscript**

-> enter the following **IF** clause:

```
22     If intRow <= strItems.GetUpperBound(0) Then  
23  
24     End If
```

#### step 5.1 - if the selection structure's condition evaluates to **True**, it means that the **ID** was located in the 1st column of the **Array**

-> therefore the condition should display in the **lblPrice** control the **price** contained in the same row as the **ID**, but in the 2nd column in the **Array**

- if the selection structure's condition evaluates to **False**, it means that the **ID** wasn't located in the **strItems 2D Array**

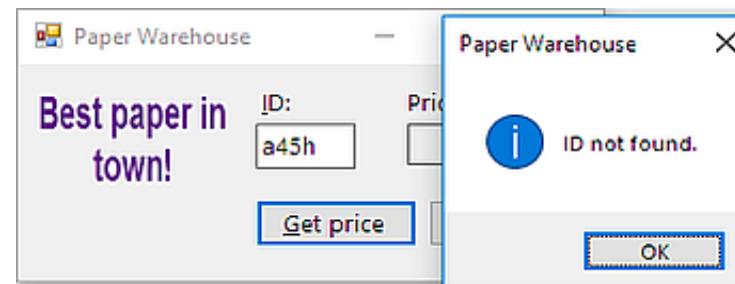
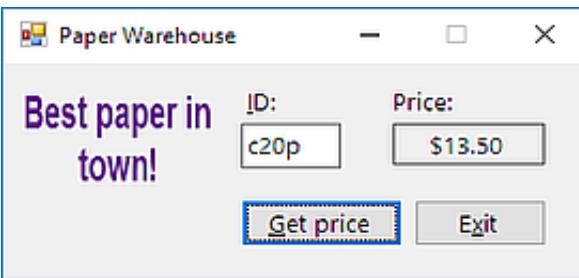
-> therefore the condition should display the message: "**ID not found.**"

-> enter additional code for the **IF** clause:

```
22     If intRow <= strItems.GetUpperBound(0) Then  
23         lblPrice.Text = "$" & strItems(intRow, 1)  
24     Else  
25         MessageBox.Show("ID not found.", "Paper Warehouse", MessageBoxButtons.OK, MessageBoxIcon.Information)  
26     End If
```

4). save the solution and then start and test the application:

- > in the **ID** box type: **c20p** and then click the button **Get price**  
-> the **Price** box should show: **\$13.50**
- > in the **ID** box type: **a45h** and then click the button **Get price**  
-> the Message box "ID not found" should appear



5). completed code:

```
1  Option Explicit On
2  Option Strict On
3  Option Infer Off
4
5  Public Class frmMain
6      ' Declare two-dimensional array:
7      Private strItems(,) As String = {{"A45G", "8.99"}, {"J63Y", "12.99"}, {"M93K", "5.99"}, {"C20P", "13.50"}, {"F77T", "7.25"}}
8
9      Private Sub btnGet_Click(sender As Object, e As EventArgs) Handles btnGet.Click
10         ' Displays an item's price.
11
12         Dim strSearchId As String
13         Dim intRow As Integer
14
15         strSearchId = txtId.Text.Trim.ToUpper
16
17         ' Search the first column for the ID. Continue searching until the end of the first column Or the ID Is found:
18         Do Until intRow > strItems.GetUpperBound(0) OrElse strItems(intRow, 0) = strSearchId
19             intRow += 1
20         Loop
21
22         If intRow <= strItems.GetUpperBound(0) Then
23             lblPrice.Text = "$" & strItems(intRow, 1)
24         Else
25             MessageBox.Show("ID not found.", "Paper Warehouse", MessageBoxButtons.OK, MessageBoxIcon.Information)
26         End If
27
28     End Sub
```

```
29
30     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
31         Me.Close()
32     End Sub
33
34     Private Sub txtId_Enter(sender As Object, e As EventArgs) Handles txtId.Enter
35         txtId.SelectAll()
36     End Sub
37
38     Private Sub ClearPrice(sender As Object, e As EventArgs) Handles txtId.TextChanged
39         lblPrice.Text = String.Empty
40     End Sub
41 End Class
```

## CH8\_A5 - my addendum from EXERCISES: **Array** statement **ReDim** - 37.ReDim Solution\_EXERCISE 20\_advanced

- **ReDim** statement:
  - **reallocates** (přerozdělit) storage space for an **Array** variable
  - releases the existing **Array** and creates a new **Array** with the same **Rank** = number of Dimensions
  - used to size or resize a dynamic array that has already been formally declared by using a **Private**, **Public**, or **Dim** statement with empty parentheses (without dimension subscripts)
- usage:
  - **a)** if you have a large **Array** and you no longer need some of its elements, **ReDim** can free up RAM by reducing the **Array** size
  - **b)** on the other hand, if your **Array** needs more elements, **ReDim** can add them
- rules:
  - **a)** the **ReDim** statement is intended only for **Arrays**; it's not valid on scalar variables (variables that contain only a single value), collections, or structures
  - **b)** note that if you declare a variable to be of type **Array**, the **ReDim** statement doesn't have sufficient type information to create the new **Array**
  - **c)** you can use **ReDim** only at procedure level -> the declaration context for the variable must be a procedure, it can't be a source file, a namespace, an interface, a class, a structure, a module, or a block
  - **d) Multiple variables:** - you can resize several **Array** variables in the same declaration statement and specify the **arrayName** and **boundList** parts for each variable; multiple variables are separated by commas
  - **e) Array bounds:** - each entry in **boundList** can specify the lower and upper bounds of that Dimension
    - the lower bound is always **0**
    - the upper bound is the highest possible index value for that dimension, not the length of the dimension (**GetUpperBound +1**)
    - the Index for each Dimension can vary from **0** through its upper bound value
    - the number of Dimensions in **boundList** must match the original number of Dimensions = Rank of the **Array**
  - **f) Data Types:** - the **ReDim** statement cannot change the data type of an **Array** variable or its elements
  - **g) Initialization:** - the **ReDim** statement cannot provide new initialization values for the **Array** elements
  - **h) Rank:** - the **ReDim** statement cannot change the Rank of the **Array** = the number of Dimensions
  - **i) optional Preserve keyword:** - if you use **Preserve optional** keyword, you can resize only the **last** Dimension of the **Array**
    - for every other Dimension, you must specify the bound of the existing **Array**
      - if your **Array** has only **1D**, you can resize that Dimension and still preserve all the contents of the **Array**,
      - e.g. because you are changing the last and only Dimension
      - however, if your Array has 2 or more Dimensions, you can change the size of only the last Dimension
    - initialization without **Preserve**: - if you do not specify **Preserve**, **ReDim** initializes the elements of the new **Array** by using the default value for their data type
    - initialization with **Preserve**: - if you specify **Preserve**, VB copies the elements from the existing **Array** to the new **Array**
  - **j) Properties:** - you can use **ReDim** on a property that holds an **Array** of values

**Array** statement's **ReDim** syntax:  
**As Array**

**ReDim Preserve arrayName(boundList As Integer), arrayName(boundList As Integer), ...**

**As Array**

**Preserve** - **OPTIONAL**

= modifier used to preserve the data in the existing **Array** when you change the size of only the last dimension

**arrayName** = name of already declared **Array** variable

**boundList** - **As Integer**

= list of bounds of each Dimension of the redefined **Array**

e.g.1

```
Dim intArray(10, 10, 10) As Integer  
  
ReDim Preserve intArray(10, 10, 20)  
  
ReDim Preserve intArray(10, 10, 15)  
  
ReDim intArray(10, 10, 10)
```

<- the **Dim** statement creates a new **Array** with **3 Dimensions**, where each declared with a bound of **10**

<- increases the size of the last Dimension without losing any existing data

<- decreases the size of the last Dimension with partial data loss

<- decreases the size of the last Dimension back to its original value and reinitializes all the **Array** elements

e.g.2

### 37.ReDim Solution\_EXERCISE 20\_advanced

Research the VB **ReDim** statement:

- What is the purpose of the **Array ReDim** statement?
- What is the purpose of the **Preserve** keyword?

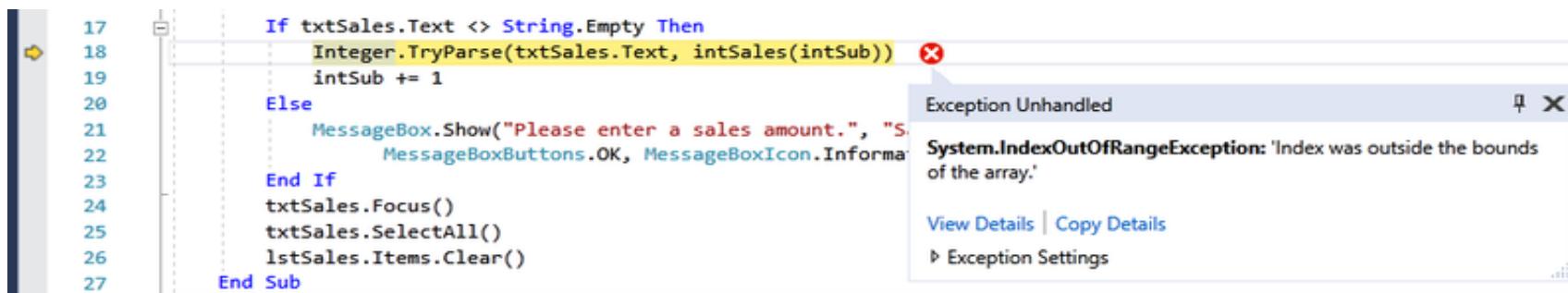
1). open the: ...VB2017\Chap08\Exercise\37.ReDim Solution\_EXERCISE20\_advanced\ReDim Solution.sln

2). -> open the Code Editor window and notice the **Array** declaration statement in the form class's declaration section:

```
10      Private intSales() As Integer = {}
```

3). -> start the application and in the **Sales** box type: **25**, and then click the button **Add to array**

<- an error message box informs you that the computer encountered an error when trying to process the **TryParse** method in the **btnAdd\_Click** procedure:



-> stop the application

4). -> modify the **btnAdd\_Click** procedure so that it can store any number of sales amounts in the **Array**

<- hint: use the **ReDim** statement to add an element to the **Array** before the **TryParse** method is processed

5). save the solution and then start and test the application:

-> in the **Sales** box type: **25**, press **Enter** key and then click the button **Display array**

<- the sales amount appears in the list box

-> now, use the button **Add to array** to add the following sales amounts, one at a time: **700, 550, 800** and then click the button **Display array**

<- the four sales amounts appear in the list box: **25, 700, 550, 800**.

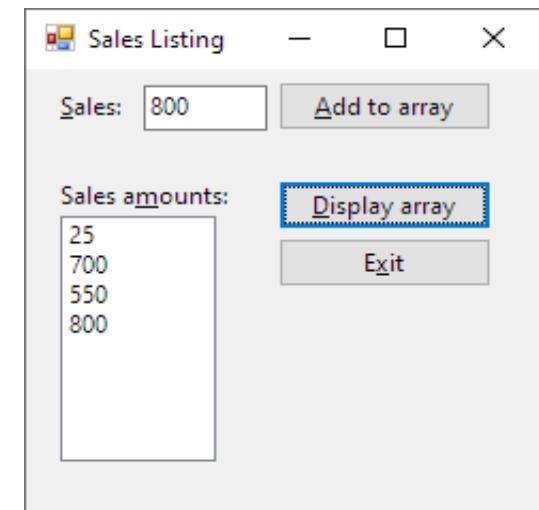
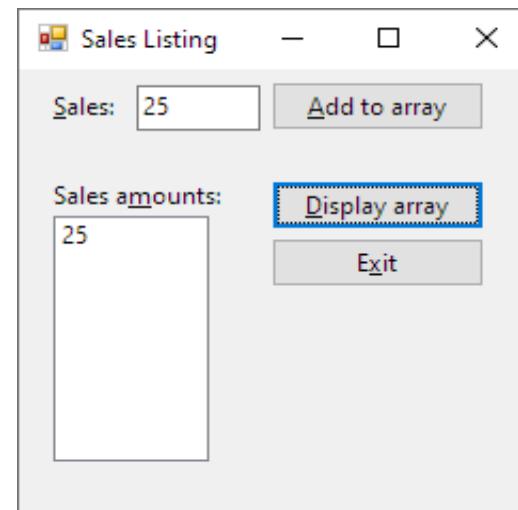
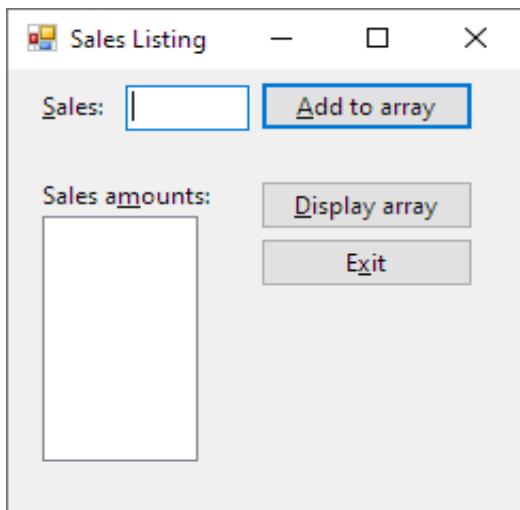
6). modified & fixed code:

```
1  ' Name:      ReDim Project
2  ' Purpose:    Add sales to an array and then display the array.
3  ' Programmer: <your name> on <current date>
4
5  Option Explicit On
6  Option Strict On
7  Option Infer Off
8
9  Public Class frmMain
10     Private intSales() As Integer = {}
11
12     Private Sub btnAdd_Click(sender As Object, e As EventArgs) Handles btnAdd.Click
13         ' Adds a sales amount to the array.
14
15         Static intSub As Integer
16
17         If txtSales.Text <> String.Empty Then
18
19             ' my line, without it an ERROR from the next line
20             ' Preserve = keep previous values in the Array
21             ReDim Preserve intSales(intSub)
22             ' _gonna research the Array statement more and gonna include it into my notes
23
24             ' an ERROR if ReDim Preserve missing:
25             Integer.TryParse(txtSales.Text, intSales(intSub))
26             intSub += 1
27         Else
28             MessageBox.Show("Please enter a sales amount.", "Sales Listing",
29                             MessageBoxButtons.OK, MessageBoxIcon.Information)
30         End If
31         txtSales.Focus()
32         txtSales.SelectAll()
33         lstSales.Items.Clear()
34     End Sub
35
36     Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
37         ' Displays the sales amounts stored in an array.
38
39         lstSales.Items.Clear()
40         For intSub As Integer = 0 To intSales.GetUpperBound(0)
41             lstSales.Items.Add(intSales(intSub).ToString)
42         Next intSub
43     End Sub
```

```

44
45     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
46         Me.Close()
47     End Sub
48
49     Private Sub txtSales_Enter(sender As Object, e As EventArgs) Handles txtSales.Enter
50         txtSales.SelectAll()
51     End Sub
52
53     Private Sub txtSales_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtSales.KeyPress
54         ' Accept only numbers and the Backspace key.
55
56         If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
57             e.Handled = True
58         End If
59     End Sub
60
61     Private Sub txtSales_TextChanged(sender As Object, e As EventArgs) Handles txtSales.TextChanged
62         lstSales.Items.Clear()
63     End Sub
64 End Class

```



LINQ = built into Visual Basic is a query language called **Language-Integrated Query**, or more simply **LINQ**

- query = dotaz; prověřit dotazováním; vyptávat se; zeptat se...
- a query language allows you to retrieve specific informations from variety of data sources, such as:
  - arrays
  - collections
  - databases

**Notice:**

- be sure to set **Option Infer On**, otherwise an errors can occur like:

for more info see: **CH3\_F8 - Option Statement - above all in a code:**

- **BC30209** Option Strict On requires all variable declarations to have an As clause

e.g. `Dim MyData = From intElement In intNums`

- **BC30574** Option Strict On disallows late binding

e.g.

`For intIndexes As Integer = 0 To MyData.Count - 1`

e.g.

`1stNames.Items.Add(MyData(intIndexes))`

**LINQ / Language-Integrated Query:**

retrieving informations from an **Array** basic syntax:

**As Array**

```
Dim yourQueryVariable = From yourElement In yourArray
optional Where some condition
optional Order By yourElement Ascending/Descending
Select yourElement
```

**As Array**

|                          |                                                                                                                                     |
|--------------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| <b>yourQueryVariable</b> | - <b>As Array</b><br>= range variable / array                                                                                       |
| <b>From</b>              | = keyword; specifies a collection and a range variable to use in a query                                                            |
| <b>yourElement</b>       | - <b>As Integer</b><br>= local range variable                                                                                       |
| <b>In</b>                | = keyword; specifies the group that the range variable is to traverse in a query                                                    |
| <b>yourArray</b>         | - <b>As Array</b><br>= previously created array                                                                                     |
| <b>Where</b>             | - <b>optional</b><br>= keyword; specifies the filtering condition for a range variable in a query                                   |
| some condition           | = specifies the filtering condition for a range variable in a query                                                                 |
| <b>Order By</b>          | - <b>optional</b><br>= keyword; specifies the sort order for columns in a query                                                     |
|                          | - can be followed by either the <b>Ascending</b> or the <b>Descending</b> keyword                                                   |
| <b>Ascending</b>         | - if keyword omitted, default will be used<br>= keyword; default; specifies the sort order for an <b>Order By</b> clause in a query |
| <b>Descending</b>        | - the smallest element will appear first<br>= keyword; specifies the sort order for an <b>Order By</b> clause in a query            |
|                          | - the largest element will appear first                                                                                             |
| <b>Select</b>            | = keyword; specifies which columns to include in the result of a query                                                              |

e.g. <- my example for testing; selects all of the elements in the **array** **intNums**, arranges their values in ascending order, and shows them in a **lstNames**

```
Dim intNums() As Integer = {10, 8, 5, 12, 7, 3}
Dim MyData = From intElement In intNums
    Order By intElement
    Select intElement
For intIndexes As Integer = 0 To MyData.Count - 1
    lstNames.Items.Add(MyData(intIndexes))
Next intIndexes
```

OR

```
For Each intIndexes As Integer In intNums
    lstNames.Items.Add(intIndexes)
Next intIndexes
```

e.g.1 <- an **array** named **data** contains all of the elements in the **array** **intNums**, and arranges their values in ascending order

```
Dim data = From intElement In intNums
    Order By intElement
    Select intElement
```

e.g.2 <- an **array** named **data** contains all of the elements in the **array** **intNums**, and arranges their values in descending order

```
Dim data = From intElement In intNums
    Order By intElement Descending
    Select intElement
```

e.g.3 <- an **array** named **data** contains only selected **array** elements in **intNums** that contain a value that is greater than **8**

```
Dim data = From intElement In intNums
    Where intElement > 8
    Select intElement
```

e.g.4 <- an **array** named **data** contains only selected **array** elements in **intNums** that contain an **even** number and arranges their values in **ascending** order

```
Dim data = From intElement In intNums
    Where intElement Mod 2 = 0
    Order By intElement
    Select intElement
```

- using **Language-Integrated Query / LINQ** to retrieve information from an **array**

1). open the: ...VB2017\Chap08\Exercise40.Linq Array Solution\Linq Array Solution.sln

2). open the **Main Form.vb**, Designer window, and Code Editor window

**Notice:**

7    Option Infer On ' Option Infer must be set to On when using LINQ.

3). locate the **btnEx1\_Click** procedure and above the statement **lstNums.Items.Clear** enter the code from

e.g.1

```
9  Public Class frmMain
10     ' Class-level array:
11     Private intNums() As Integer = {10, 8, 5, 12, 7, 3}
12
13     Private Sub btnEx1_Click(sender As Object, e As EventArgs) Handles btnEx1.Click
14         ' Display array values in ascending order.
15         Dim data = From intElement In intNums
16             Order By intElement
17             Select intElement
18
19         lstNums.Items.Clear
20         For Each intValue As Integer In data
21             lstNums.Items.Add(intValue.ToString.PadLeft(4))
22         Next intValue
23     End Sub
```

4). locate the **btnEx2\_Click** procedure and above the statement **lstNums.Items.Clear** enter the code from

e.g.2

```
24    Private Sub btnEx2_Click(sender As Object, e As EventArgs) Handles btnEx2.Click
25        ' Display array values in descending order.
26        Dim data = From intElement In intNums
27            Order By intElement Descending
28            Select intElement
29
30        lstNums.Items.Clear
```

5). locate the **btnEx3\_Click** procedure and above the statement **lstNums.Items.Clear** enter the code from

e.g.3

```
35    Private Sub btnEx3_Click(sender As Object, e As EventArgs) Handles btnEx3.Click
36        ' Display array values that are greater than 8.
37        Dim data = From intElement In intNums
38            Where intElement > 8
39            Select intElement
40
41        lstNums.Items.Clear
```

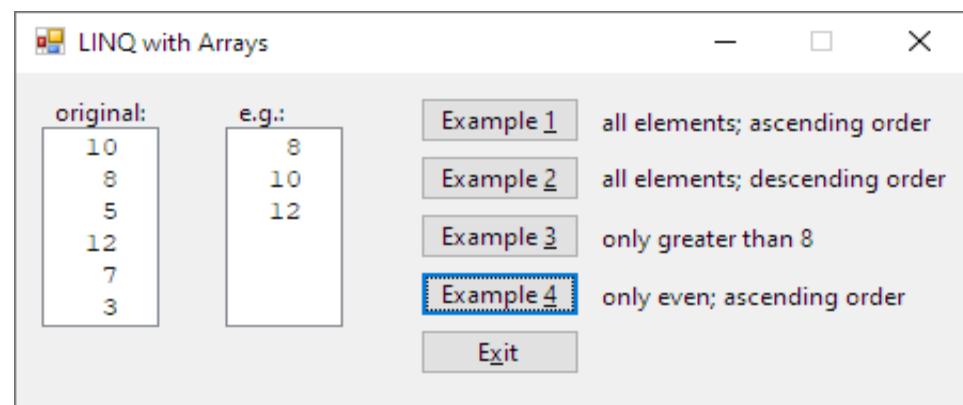
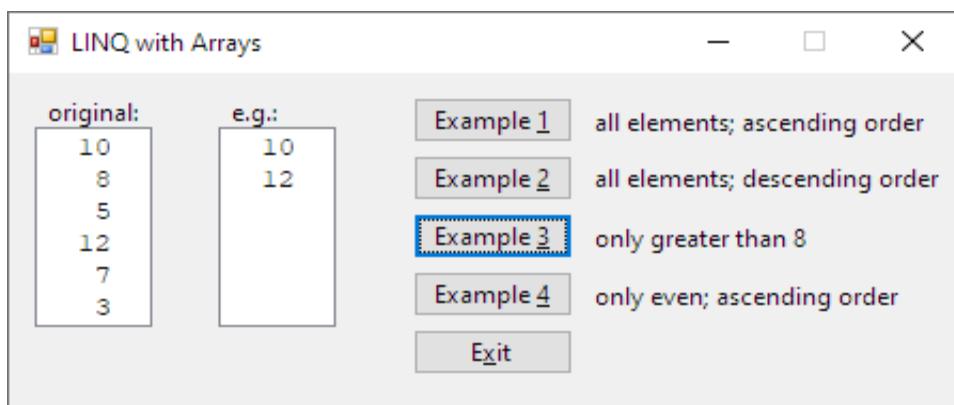
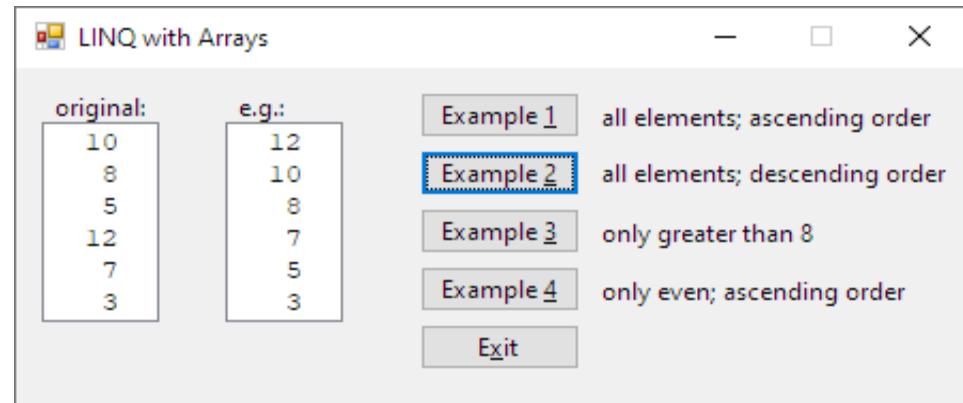
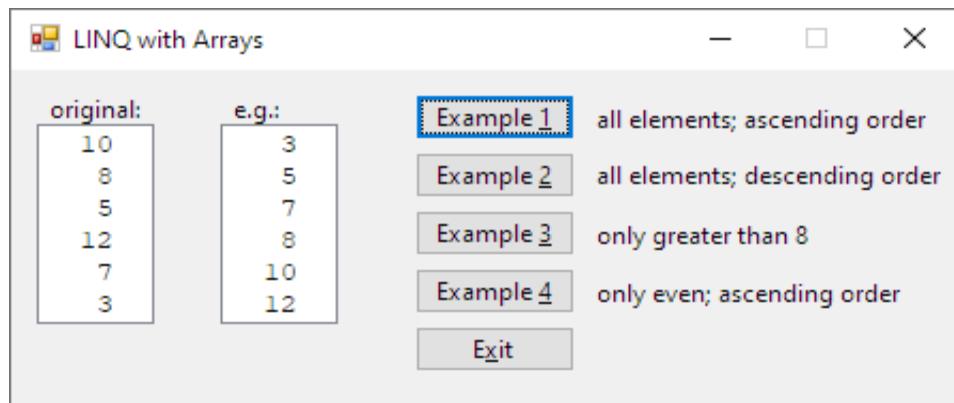
6). locate the **btnEx4\_Click** procedure and above the statement **lstNums.Items.Clear** enter the code from

e.g.4

```
46    Private Sub btnEx4_Click(sender As Object, e As EventArgs) Handles btnEx4.Click
47        ' Display array values that are even numbers and arrange them in ascending order.
48        Dim data = From intElement In intNums
49            Where intElement Mod 2 = 0
50            Order By intElement
51            Select intElement
52
53        lstNums.Items.Clear
```

7). save the solution and start & test the application

> click the buttons to see the results:



8). the entire code: ..VB2017\Chap08\Exercise\40.Linq Array Solution\LinQ Array Solution.sln

```
1  ' Name:      LinQ Array Project
2  ' Purpose:    Demonstrate LINQ queries with an array.
3  ' Programmer: <your name> on <current date>
4
5  Option Explicit On
6  Option Strict On
7  Option Infer On ' Option Infer must be set to On when using LINQ.
8
9  Public Class frmMain
10    ' Class-level array:
11    Private intNums() As Integer = {10, 8, 5, 12, 7, 3}
12
```

```
13  Private Sub btnEx1_Click(sender As Object, e As EventArgs) Handles btnEx1.Click
14      ' Display array values in ascending order.
15      Dim data = From intElement In intNums
16          Order By intElement
17          Select intElement
18      lstNums.Items.Clear()
19      For Each intValue As Integer In data
20          lstNums.Items.Add(intValue.ToString.PadLeft(4))
21      Next intValue
22  End Sub
23
24  Private Sub btnEx2_Click(sender As Object, e As EventArgs) Handles btnEx2.Click
25      ' Display array values in descending order.
26      Dim data = From intElement In intNums
27          Order By intElement Descending
28          Select intElement
29      lstNums.Items.Clear()
30      For Each intValue As Integer In data
31          lstNums.Items.Add(intValue.ToString.PadLeft(4))
32      Next intValue
33  End Sub
34
35  Private Sub btnEx3_Click(sender As Object, e As EventArgs) Handles btnEx3.Click
36      ' Display array values that are greater than 8.
37      Dim data = From intElement In intNums
38          Where intElement > 8
39          Select intElement
40      lstNums.Items.Clear()
41      For Each intValue As Integer In data
42          lstNums.Items.Add(intValue.ToString.PadLeft(4))
43      Next intValue
44  End Sub
45
46  Private Sub btnEx4_Click(sender As Object, e As EventArgs) Handles btnEx4.Click
47      ' Display array values that are even numbers and arrange them in ascending order.
48      Dim data = From intElement In intNums
49          Where intElement Mod 2 = 0
50          Order By intElement
51          Select intElement
52      lstNums.Items.Clear()
53      For Each intValue As Integer In data
54          lstNums.Items.Add(intValue.ToString.PadLeft(4))
55      Next intValue
56  End Sub
```

```

57
58     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
59         Me.Close()
60     End Sub
61
62     ' My extra:
63     Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
64         For Each intOriginal As Integer In intNums
65             lstOriginal.Items.Add(intOriginal.ToString.PadLeft(4))
66         Next intOriginal
67     End Sub
68 End Class

```

### CH8\_A6.1 - using LINQ's Aggregate operators to retrieve a single value from an Array + 41.Linq Aggregate Array Solution - additional topic from Appendix B

- LINQ / Language-Integrated Query also provides several **Aggregate** [souhrnný] operators, that you can use when querying an array

- an **Aggregate operator** returns a single value from a group of values:

**Average** = returns the average of the values in the group / array e.g.3

**Count** = returns the number of results according the filtering condition specified in **Where** e.g.4

**Max** = returns the highest value in the group / array e.g.2

**Min** = returns the smallest value in the group / array

**Sum** = returns the sum of the values in the group / array e.g.1

#### LINQ / Language-Integrated Query:

using **Aggregate operator** to retrieve a single value from an array syntax:

As Number

```

Dim yourAggregatedVariable As dataType | yourAggregatedVariable = Aggregate yourElement In yourArray
optional Where some condition
Select yourElement Into aggregateOperator

```

**yourAggregatedVariable** = will contain a single value from a group of values, depending on **aggregateOperator** used

**As dataType** - mind **yourAggregatedVariable** data type and **yourArray** data type

**Aggregate** = keyword; applies an aggregation function to a sequence

**yourElement** = local range variable

**In** = keyword; specifies the group that the range variable is to traverse in a query

**yourArray** - **As Array**

**Where** - **optional**

= keyword; specifies the filtering condition for a range variable in a query

= specifies the filtering condition for a range variable in a query

**Select** = keyword; specifies which columns to include in the result of a query

**Into** = keyword; specifies an identifier that can serve as a reference to the results of a join or grouping subexpression

**aggregateOperator** = returns a single value from a group of values; **Average**, **Count**, **Max**, **Min**, **Sum**

e.g. <- calculates the total of the values in the array dblNums and assigns the result to the variable dblTotal = 34.48

```
Dim dblNums() As Double = {10.5, 8.5, 5.5, 9.98}
Dim dblTotal As Double
dblTotal = Aggregate number In dblNums
    Select number Into Sum
```

e.g.1 <- calculates the total of the values in the array intNums and assigns the result to the variable intTotal = 45

```
Private intNums() As Integer = {10, 8, 5, 12, 7, 3}
...
Dim intTotal As Integer = Aggregate number In intNums
    Select number Into Sum
```

e.g.2 <- finds the highest value in the array intNums and assigns the result to the variable intHighest = 12

```
Private intNums() As Integer = {10, 8, 5, 12, 7, 3}
...
Dim intHighest As Integer = Aggregate number In intNums
    Select number Into Max
```

e.g.3 <- calculates the average of the values in the array intNums and assigns the result to the variable dblAvg = 7.5

```
Private intNums() As Integer = {10, 8, 5, 12, 7, 3}
...
Dim dblAvg As Double = Aggregate number In intNums
    Select number Into Average
```

e.g.4 <- counts the number of odd numbers in the array intNums and assigns the result to the variable intCountOdd = 3 (5, 7, 3)

```
Private intNums() As Integer = {10, 8, 5, 12, 7, 3}
...
Dim intCountOdd As Integer = Aggregate number In intNums
    Where number Mod 2 = 1
        Into Count or Select number Into Count
```

- the Count operator is the only operator that does not need the Select clause - but nothing happened, when have

- using LINQ (Language-Integrated Query) **Aggregate** operators:

1). open the: ...VB2017\Chap08\Exercise\41.Linq Aggregate Array Solution\.Linq Aggregate Array Solution.sln

2). open the **Main Form.vb**, Designer window, and Code Editor window

Notice: 7 Option Infer Off

3). locate the **btnEx1\_Click** procedure and above the assignment statement enter the code from e.g.1

```
9  Public Class frmMain
10     ' Class-level array:
11     Private intNums() As Integer = {10, 8, 5, 12, 7, 3}
12
13     Private Sub btnEx1_Click(sender As Object, e As EventArgs) Handles btnEx1.Click
14         ' Display the sum of the array values.
15         Dim intTotal As Integer = Aggregate number In intNums
16                         Select number Into Sum
17         lblResult.Text = "Sum: " & intTotal.ToString
18     End Sub
19
```

4). locate the **btnEx2 Click** procedure and above the assignment statement enter the code from e.g.2

```
20 Private Sub btnEx2_Click(sender As Object, e As EventArgs) Handles btnEx2.Click
21     ' Display the highest value in the array.
22     Dim intHighest As Integer = Aggregate number In intNums
23                     Select number Into Max
24     lblResult.Text = "Highest number: " & intHighest.ToString
25 End Sub
26
```

5). locate the **btnEx3\_Click** procedure and above the assignment statement enter the code from

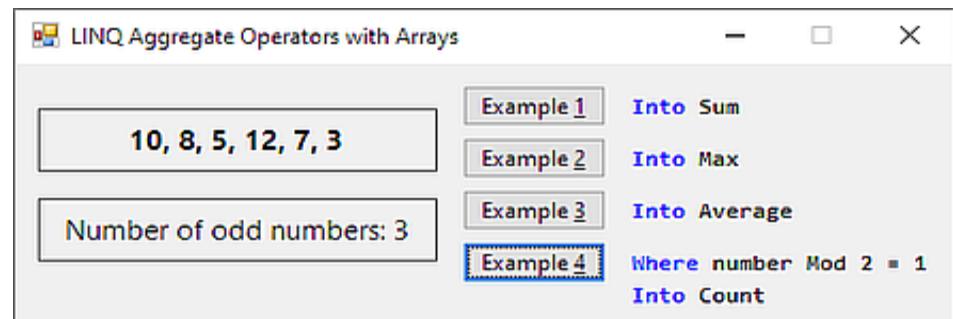
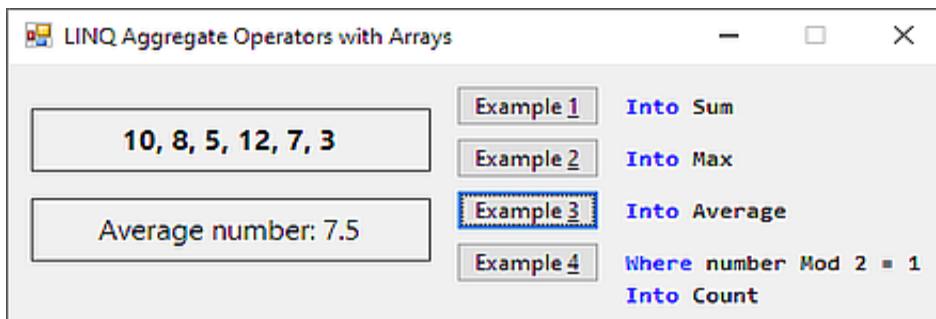
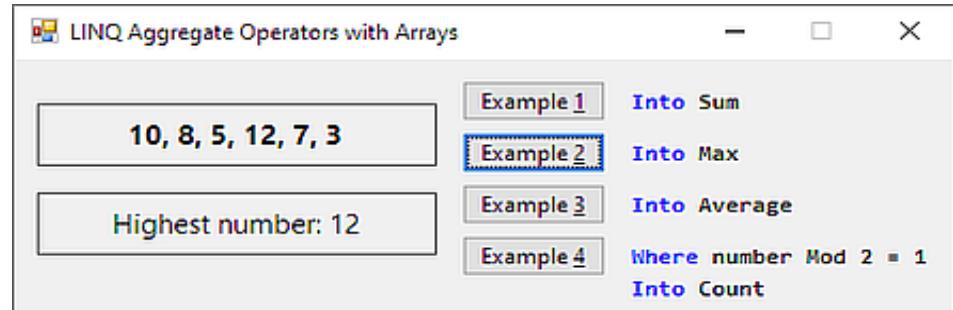
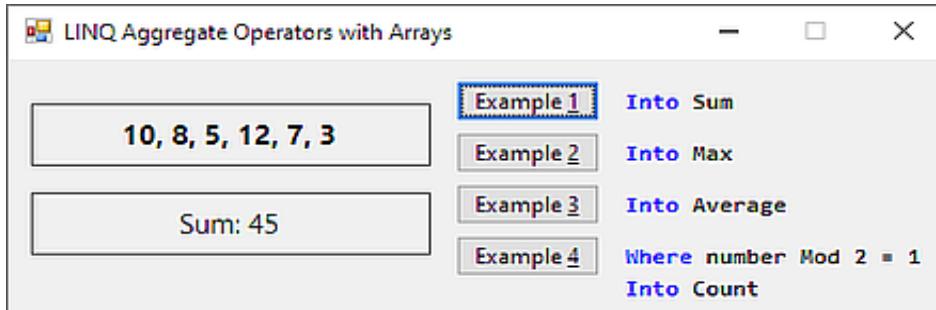
```
27 Private Sub btnEx3_Click(sender As Object, e As EventArgs) Handles btnEx3.Click
28     ' Display the average value contained in the array.
29     Dim dblAvg As Double = Aggregate number In intNums
30             Select number Into Average
31     lblResult.Text = "Average number: " & dblAvg.ToString
32 End Sub
33
```

6). locate the **btnEx4 Click** procedure and above the assignment statement enter the code from e.g.4

```
34     Private Sub btnEx4_Click(sender As Object, e As EventArgs) Handles btnEx4.Click
35         ' Display the number of odd numbers contained in the array.
36         Dim intCountOdd As Integer = Aggregate number In intNums
37                         Where number Mod 2 = 1
38                         Into Count
39         lblResult.Text = "Number of odd numbers: " & intCountOdd.ToString
40     End Sub
```

7). save the solution and start & test the application

> click the buttons to see the results:



8). the entire code: ...VB2017\Chap08\\_Exercise\41.Linq Aggregate Array Solution\Linq Aggregate Array Solution.sln

```
1  ' Name:      Linq Aggregate Array Project
2  ' Purpose:    Demonstrate LINQ aggregate operators with an array.
3  ' Programmer: <your name> on <current date>
4
5  Option Explicit On
6  Option Strict On
7  Option Infer Off
8
9  Public Class frmMain
10   ' Class-level array:
11   Private intNums() As Integer = {10, 8, 5, 12, 7, 3}
12
13  Private Sub btnEx1_Click(sender As Object, e As EventArgs) Handles btnEx1.Click
14      ' Display the sum of the array values.
15      Dim intTotal As Integer = Aggregate number In intNums
16          Select number Into Sum
17      lblResult.Text = "Sum: " & intTotal.ToString
18  End Sub
19
```

```

20  Private Sub btnEx2_Click(sender As Object, e As EventArgs) Handles btnEx2.Click
21      ' Display the highest value in the array.
22      Dim intHighest As Integer = Aggregate number In intNums
23          Select number Into Max
24      lblResult.Text = "Highest number: " & intHighest.ToString
25  End Sub
26
27  Private Sub btnEx3_Click(sender As Object, e As EventArgs) Handles btnEx3.Click
28      ' Display the average value contained in the array.
29      Dim dblAvg As Double = Aggregate number In intNums
30          Select number Into Average
31
32  End Sub
33
34  Private Sub btnEx4_Click(sender As Object, e As EventArgs) Handles btnEx4.Click
35      ' Display the number of odd numbers contained in the array.
36      Dim intCountOdd As Integer = Aggregate number In intNums
37          Where number Mod 2 = 1
38          Into Count
39      lblResult.Text = "Number of odd numbers: " & intCountOdd.ToString
40  End Sub
41
42  End Class

```

### CH8\_A7 - Structure - composite of variables like **array**, but with a combination of different data types - additional topic from Appendix B

- the data types used in previous chapters, such as **Integer**, **Double**, **String** etc. are built into Visual Basic language, and creates a **simple / scalar** variables
- you can use them to create a **group of related variables**:
  - **array variable** - where all of the **array member** variables must be of **same data type** **As Array**
  - **structure variable** - where you can combine items of different data types **As Structure**
- **Structure**: = generalization of the user-defined data type -> **UDT**
  - = associates one or more items with each other (therefore **members**) & with the **structure** itself -> declared as *yourMemberVariable*
  - the **structures** you create are composed [složený] of **members** that are defined between the **Structure** and **End Structure** clauses
  - you can combine data items of a different types to create a **structure**
  - when you declare a **structure**, it becomes a composite data type, and you can declare variables of that type
  - data types created by the **Structure** statement are referred to as **user-defined data types** or **structures**
  - useful when you want a single variable to hold several related pieces of information with a different data types
  - e.g.** - you might want to keep together: an employee's id, first name, second name, and salary
    - you could use several variables for this information,
    - or you could define a **structure** and use it for a single variable: **employee** e.g. 1
    - the advantage of the **structure** becomes apparent when you have many employees and therefore many instances of the variable
- programmers use **structure variables** when they need to pass a group of related items to a procedure for further processing -> **CH8\_A7.1**
  - > this is because it is easier to pass one **structure** variable rather than many individual variables
  - when you pass a **structure** variable to a procedure, all of its **members** are passed automatically
- programmers also use structure variables to store related items in an array, even when the members have different data types **CH8\_A7.2**

## How to create and use a Structure

= a compound variable; a composite of several variables with a different data types

### Step 1

= declare a **Structure** and only define its **member variables** in: *yourMemberVariable*

### Step 2

= declare a **structure variables** / declare an **array** of **structure variables**

- after entering the **Structure** statement in the Code Editor window, you can use the **structure** to declare a variable

- variables declared using a **structure** are often referred to as **structure variables**: *yourStructureVariable*

- **structure variable** then contains **member variables** defined in: **Step 1**

### Step 3

= use your **structure variables** / **array** of **structure variables** declared in: **Step 2** ,

and its **member variables** defined in: **Step 1**

**Step 1** = declare a **Structure** and only define its **member variables** in: *yourMemberVariable*

[Structure statement & defining its member variables syntax:](#)

```
Structure YourStructureName
    Public yourMemberVariable As dataType
    Public yourMemberVariable As dataType
    ...
End Structure
```

**As Structure / user-defined data type**

= composite / compound data type since the **members** can be a different data type

**Step 2** = declare a **structure variables** / declare an **array** of **structure variables**

a. [structure variable syntax:](#)

**As Structure**

```
Dim / Private yourStructureVariable As YourStructureName
```

b. [array of structure variables syntax:](#)

**As Array** of **structure** variables

```
Dim/Static/Private arrayOfStructureName(highestSubscript As Integer) As YourStructureName
```

**Step 3** = use your **structure variables** / **array** of **structure variables** declared in: **Step 2** , and its **member variables** defined in: **Step 1**

a. [use your structure variable syntax:](#)

**As** *yourMemberVariable* **data type**

```
... yourStructureVariable.yourMemberVariable ...
```

**As** *yourMemberVariable* **data type**

b. [use your array of structure variables syntax:](#)

**As** *yourMemberVariable* **data type**

```
... arrayOfStructureName(Subscript).yourMemberVariable ...
```

**As** *yourMemberVariable* **data type**

|                                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
|-------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Structure &amp; End Structure</b>            | = statement, keyword<br>- typically entered in the <b>form class</b> 's declarations section, which is the area in the Code Editor window between the <b>Public Class</b> and <b>End Class</b> clauses<br>- declares the name of a <b>structure</b> and introduces the definition of the variables, properties, events, and procedures that make up the <b>structure</b><br>- the <b>Structure</b> statement allows the programmer to group related items into one unit - a <b>structure</b><br><b>- notes:</b> - the <b>Structure</b> statement merely defines the <b>structure members</b> in <i>yourStructureMemberVariables</i><br>- <b>it does not reserve any memory locations inside the computer</b><br>-> you reserve memory locations by declaring a <b>structure</b> variable / <b>array</b> of <b>structure</b> variables in: <b>Step 2</b> |
| <i>YourStructureName</i>                        | = name of your <b>structure</b> , usually entered using Pascal case                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |
| <b>Public</b>                                   | = keyword; specifies that one or more declared programming elements have no access restrictions                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                         |
| <i>yourMemberVariable</i>                       | = referred to as <b>member</b> of your <b>structure</b> , named in: <i>YourStructureName</i><br>- can be: variables, constants, or procedures<br>- however, in most cases, the <b>members</b> will be variables; such variables are referred to as <b>member variables</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |
| <b>As dataType</b>                              | = identifies the type of data the <b>member variable</b> will store<br>- can be any of the standard data types available in Visual Basic<br>- it can also be another <b>structure</b> = <b>user-defined</b> data type                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
| <i>yourStructureVariable</i>                    | = name of your <b>structure variable</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <i>arrayOfStructureName</i>                     | - contains all of the <b>member variables</b> defined in: <b>Step 1</b> as: <i>yourMemberVariable</i><br>= <b>1D array</b> of <b>Structure</b> variables                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                |
| <i>highestSubscript</i>                         | - each <b>array</b> element contains all of the <b>Structure</b> 's member variables defined in: <b>Step 1</b><br><b>- As Integer</b><br>= specifies the zero-based highest subscript in the array<br>- i.e.: an array whose <b>highestSubscript = 2</b> will contain <b>3 elements</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| <i>yourStructureVariable.yourMemberVariable</i> | - in code, you refer to the entire <b>structure variable</b> by its name defined in: <i>yourStructureVariable</i><br>- you refer to a <b>member variable</b> by preceding its name with the name of the <b>structure variable</b> in which it is defined<br>- you use the <b>dot</b> member access operator to separate them<br>- the <b>member variables</b> in a <b>structure variable</b> can be used just like any other variables:<br>- you can assign values to them, use them in calculations, display their contents, and so on                                                                                                                                                                                                                                                                                                                 |

**Step 1** = declare a **Structure** and only define its **member variables** in: *yourMemberVariable*

e.g. 1    **Structure Employee**  
           Public strId As String  
           Public strFirst As String  
           Public strLast As String  
           Public dblPay As Double  
       **End Structure**

<- declares a **Structure** named **Employee**  
   <- only defines a string **member variable** named **strId**  
   <- only defines a string **member variable** named **strFirst**  
   <- only defines a string **member variable** named **strLast**  
   <- only defines a double **member variable** named **dblPay**

**Step 2** = declare a **structure variables** / declare an **array** of **structure variables**

e.g. 2a    **Dim hourly As Employee**    <- declares a procedure-level **Employee structure variable** named **hourly**  
                   <- the **hourly structure variable** now contains 4 **member variables**:  
                   **hourly.strId, hourly.strFirst, hourly.strLast, hourly.dblPay**

e.g. 2b    **Private salaried As Employee**    <- declares a class-level **Employee structure variable** named **salaried**  
                   <- the names of the **member variables** within the **salaried structure variable** are:  
                   **salaried.strId, salaried.strFirst, salaried.strLast, salaried.dblPay**

e.g. 2c    **Dim arrayHourly(3) As Employee**    <- declares a procedure-level **array** of **structure** variables with 4 elements  
                   <- the names of the **member variables** within the **array of structure** variables are:  
                   from: **arrayHourly(0).strId** till: **arrayHourly(3).dblPay** - 4 x 4 = **16 unique member variables**

**Step 3** = use your **structure variables** / **array** of **structure variables** declared in: **Step 2**, and its **member variables** defined in: **Step 1**

e.g. 3a    **hourly.strId = "employee123"**  
           **hourly.strFirst = "Caroline"**  
           **hourly.strLast = "Hotentot"**  
           **hourly.dblPay = 17.35**  
           ...  
           **hourly.dblPay \*= 1.05**    <- multiplies the contents of the **hourly.dblPay** member variable by **1.05** and then assigns the result to it

OR:  
     **hourly.dblPay = hourly.dblPay \* 1.05**  
           <- in code, you refer to the entire **structure variable** by its name - in this case, **hourly**  
           <- you refer to a **member variable** by preceding its name with the name of the **structure variable** in which it is defined  
           <- you use the **dot** member access operator (a period) to separate the **structure variable's name** from the **member variable's name**, like this:  
                   **hourly.strId, hourly.strFirst, hourly.strLast, hourly.dblPay**  
           <- the **dot** member access operator indicates that **strId, strFirst, strLast, and dblPay** are **members** of the **hourly structure variable**

e.g. 3b    **lblSalary.Text = salaried.dblPay.ToString("C2")**    <- formats the value contained in the **salaried.dblPay** member variable and then displays the result in the **lblSalary** control

e.g. 3c    **arrayHourly(0).strId = "employee123"**  
           **arrayHourly(1).strId = "employee456"**  
           **arrayHourly(3).dblPay = 16.63**

## CH8\_A7.1 - compare passing variables to a procedure: **several separated** vs. **only 1 structure** - 42.Norbert Solution - additional topic from Appendix B

- the sales manager at **Norbert Pool & Spa Depot** wants you to create an application that determines the amount of water required to fill a rectangular pool
- to perform this task, the application will need to calculate the volume [objem] of the pool
- you calculate the volume by multiplying the pool's length by its width and then multiplying the result by the pool's depth  $a * b * c$
- assuming the length, width, and depth are measured in feet, this gives you the volume in cubic feet
- to determine the number of gallons of water, you multiply the number of cubic feet by **7.48** because there are **7.48** gallons in one cubic foot

- passing **several** separated variables vs. passing only **1 structure** variable to a procedure e.g.:

1). open the: ...VB2017\Chap08\Exercise\42.Norbert Solution\Norbert Solution.sln

2). open the **Main Form.vb**, Designer window, and Code Editor window

3). start and test the application:

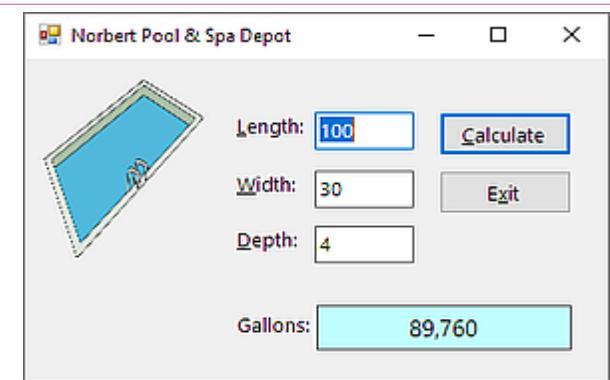
- > type **100** in the **Length:** box, **30** in the **Width:** box, and **4** in the **Depth:** box
- > click the button **Calculate**
- <- the required number of gallons appears in the interface: **89.760**
- > click the button **Exit** to end the application

4). open the Code Editor window

**notice:** the code without a **structure** for the **btnCalc\_Click** procedure and **GetGallons** function:

1. - the procedure **btnCalc\_Click** calls the **function GetGallons**, passing it 3 variables: **dblPoolLength**, **dblPoolWidth**, **dblPoolDepth** by value
2. - the function **GetGallons** uses the values to calculate the number of gallons required to fill the pool
3. - the function **GetGallons** returns the number of gallons as a **Double** number to the procedure, which assigns the value to the **dblGallons** variable

```
12     Private Function GetGallons(ByVal dblLen As Double, ByVal dblWid As Double, ByVal dblDep As Double) As Double
13         ' Calculates and returns the number of gallons.
14
15         Const dblGAL_PER_CUBIC_FOOT As Double = 7.48
16
17         Return dblLen * dblWid * dblDep * dblGAL_PER_CUBIC_FOOT
18     End Function
19
20     Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
21         ' Display the number of gallons.
22
23         Dim dblPoolLength As Double
24         Dim dblPoolWidth As Double
25         Dim dblPoolDepth As Double
26         Dim dblGallons As Double
27
28         Double.TryParse(txtLength.Text, dblPoolLength)
29         Double.TryParse(txtWidth.Text, dblPoolWidth)
30         Double.TryParse(txtDepth.Text, dblPoolDepth)
31
32         dblGallons = GetGallons(dblPoolLength, dblPoolWidth, dblPoolDepth)
```



<- **GetGallons** function, 1.

<- 2.

<- **btnCalc\_Click** procedure

<- 1. & 3.

```

33
34     lblGallons.Text = dblGallons.ToString("N0")
35     txtLength.Focus()
36 End Sub

```

## 5). use a **Structure** in the application -> modify the existing application step by step: #1: - #6:

- note:**
- a more convenient way of coding the application is to use a **structure** to group together the **input items**: **length**, **width**, and **depth**
  - it is logical to group the **3** items, because they are related -> each represents **1** of the **3** dimensions of a rectangular pool
  - a descriptive name for the **structure** would be **Dimensions**

**notice:** the code with a **structure** for the **btnCalc\_Click** procedure and **GetGallons** function:

1. - the procedure **btnCalc\_Click** calls the **function GetGallons**, passing it only **1 structure** variable (and its members): **poolSize** by value
2. - the function **GetGallons** uses the values contained in the **structure** variable to calculate the number of gallons required to fill the pool
3. - the function **GetGallons** returns the number of gallons as a **Double** number to the procedure, which assigns the value to the **dblGallons** variable

### 5.1). locate the the **form class's** declarations section

#1: - first, you will declare the **Structure** in the form class's declarations section

-> enter the following **Structure** statement below:

```

9  Public Class frmMain
10
11  Structure Dimensions
12      Public dblLength As Double
13      Public dblWidth As Double
14      Public dblDepth As Double
15  End Structure
16

```

#1:

### 5.2). locate the **btnCalc\_Click** procedure

#2: - the procedure will use a **structure** variable - rather than **3** separate variables - to store the input items  
 -> on the lines: **28 - 31** replace the first **3 Dim** statements with the **structure** variable **Dim** statement

#3: - next, you will store each input item in its corresponding **member** in the **structure** variable  
 -> on the lines: **34 - 39** in the 3 TryParse methods, change:

|                      |      |                           |
|----------------------|------|---------------------------|
| <b>dblPoolLength</b> | for: | <b>poolSize.dblLength</b> |
| <b>dblPoolWidth</b>  | for: | <b>poolSize.dblWidth</b>  |
| <b>dblPoolDepth</b>  | for: | <b>poolSize.dblDepth</b>  |

#4: - instead of sending **3** separate variables to the function **GetGallons**, the procedure now needs to send only **1** variable: the **structure** variable **poolSize**  
 -> on the lines: **41 - 42** change the statement that invokes the **GetGallons** function,

|       |                                                                           |
|-------|---------------------------------------------------------------------------|
| from: | <b>dblGallons = GetGallons(dblPoolLength, dblPoolWidth, dblPoolDepth)</b> |
| to:   | <b>dblGallons = GetGallons(poolSize)</b>                                  |

- note:**
- when you pass a **structure** variable to a procedure, all of its **members** are passed automatically
  - although passing **1 structure** variable rather than **3** separate variables in this example may not seem like a huge advantage, consider the convenience of passing **1 structure** variable rather than **10 separate** variables

```

25  Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
26      ' Display the number of gallons.
27
28      'Dim dblPoolLength As Double
29      'Dim dblPoolWidth As Double
30      'Dim dblPoolDepth As Double
31      Dim poolSize As Dimensions      ' structure variable
32      Dim dblGallons As Double
33
34      'Double.TryParse(txtLength.Text, dblPoolLength)
35      'Double.TryParse(txtWidth.Text, dblPoolWidth)
36      'Double.TryParse(txtDepth.Text, dblPoolDepth)
37      Double.TryParse(txtLength.Text, poolSize.dblLength)
38      Double.TryParse(txtWidth.Text, poolSize.dblWidth)
39      Double.TryParse(txtDepth.Text, poolSize.dblDepth)
40
41      'dblGallons = GetGallons(dblPoolLength, dblPoolWidth, dblPoolDepth)
42      dblGallons = GetGallons(poolSize)
43

```

#2:

#3:

#4:

&lt;- 1. &amp; 3.

### 5.3). locate the **GetGallons** function

#5: - the function will now receive **1 structure** variable named **Dimensions** created in: #1: , rather than **3 separate Double** variables  
 - like the **Double** variables, the **structure** variable **Dimensions** will be passed by value, because the function does not need to change any **member's** value  
 -> on the lines: 17 - 18 in the function header,  
 replace the 3 parameters: **Private Function GetGallons(ByVal dblLen As Double, ByVal dblWid As Double, ByVal dblDep As Double) As Double**  
 for: **Private Function GetGallons(ByVal pool As Dimensions) As Double**

#6: - the function will now use the **members** of the **structure** variable named **poolSize** to calculate the number of gallons  
 -> on the lines: 23 - 24 change the **Return** statement:

from: **Return dblLen \* dblWid \* dblDep \* dblGAL\_PER\_CUBIC\_FOOT**  
 to: **Return pool.dblLength \* pool.dblWidth \* pool.dblDepth \* dblGAL\_PER\_CUBIC\_FOOT**

```

17  'Private Function GetGallons(ByVal dblLen As Double, ByVal dblWid As Double, ByVal dblDep As Double) As Double
18  Private Function GetGallons(ByVal pool As Dimensions) As Double
19      ' Calculates and returns the number of gallons.
20
21      Const dblGAL_PER_CUBIC_FOOT As Double = 7.48
22
23      'Return dblLen * dblWid * dblDep * dblGAL_PER_CUBIC_FOOT
24      Return pool.dblLength * pool.dblWidth * pool.dblDepth * dblGAL_PER_CUBIC_FOOT
25
26  End Function

```

#5:

&lt;- 1.

#6:

&lt;- 2.

6). test the modified code: start the application and:

-> type **100** in the **Length:** box, **30** in the **Width:** box, and **4** in the **Depth:** box

-> click the button **Calculate**

<- the required number of gallons appears in the interface: **89.760**

-> click the button **Exit** to end the application

7). the entire code: ...VB2017\Chap08\Exercise\42.Norbert Solution\Norbert Solution.sln

**compare:** the code without a **structure** for the **btnCalc\_Click** procedure and **GetGallons** function:

1. - the procedure **btnCalc\_Click** calls the **function GetGallons**, passing it 3 variables: **dblPoolLength**, **dblPoolWidth**, **dblPoolDepth** by value
2. - the function **GetGallons** uses the values to calculate the number of gallons required to fill the pool
3. - the function **GetGallons** returns the number of gallons as a **Double** number to the procedure, which assigns the value to the **dblGallons** variable

**with:** the code with a **structure** for the **btnCalc\_Click** procedure and **GetGallons** function:

1. - the procedure **btnCalc\_Click** calls the **function GetGallons**, passing it only 1 **structure** variable (and its members): **poolSize** by value
2. - the function **GetGallons** uses the values contained in the **structure** variable to calculate the number of gallons required to fill the pool
3. - the function **GetGallons** returns the number of gallons as a **Double** number to the procedure, which assigns the value to the **dblGallons** variable

```
1  ' Name:      Norbert Project
2  ' Purpose:    Display the number of gallons needed to fill a rectangular pool.
3  ' Programmer: <your name> on <current date>
4
5  Option Explicit On
6  Option Strict On
7  Option Infer Off
8
9  Public Class frmMain
10
11     Structure Dimensions
12         Public dblLength As Double
13         Public dblWidth As Double
14         Public dblDepth As Double
15     End Structure
16
17     'Private Function GetGallons(ByVal dblLen As Double, ByVal dblWid As Double, ByVal dblDep As Double) As Double
18     Private Function GetGallons(ByRef pool As Dimensions) As Double
19         ' Calculates and returns the number of gallons.
20
21         Const dblGAL_PER_CUBIC_FOOT As Double = 7.48
22
23         'Return dblLen * dblWid * dblDep * dblGAL_PER_CUBIC_FOOT
24         Return pool.dblLength * pool.dblWidth * pool.dblDepth * dblGAL_PER_CUBIC_FOOT
25
26     End Function
27
```

1.  
1.

2.  
2.

```

28  Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
29      ' Displays the number of gallons.
30
31      'Dim dblPoolLength As Double
32      'Dim dblPoolWidth As Double
33      'Dim dblPoolDepth As Double
34      Dim poolSize As Dimensions      ' structure variable
35      Dim dblGallons As Double
36
37      'Double.TryParse(txtLength.Text, dblPoolLength)
38      'Double.TryParse(txtWidth.Text, dblPoolWidth)
39      'Double.TryParse(txtDepth.Text, dblPoolDepth)
40      Double.TryParse(txtLength.Text, poolSize.dblLength)
41      Double.TryParse(txtWidth.Text, poolSize.dblWidth)
42      Double.TryParse(txtDepth.Text, poolSize.dblDepth)
43
44      'dblGallons = GetGallons(dblPoolLength, dblPoolWidth, dblPoolDepth)
45      dblGallons = GetGallons(poolSize)
46
47      lblGallons.Text = dblGallons.ToString("N0")
48      txtLength.Focus()
49  End Sub
50
51  Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
52      Me.Close()
53  End Sub
54
55  Private Sub txtDepth_Enter(sender As Object, e As EventArgs) Handles txtDepth.Enter
56      txtDepth.SelectAll()
57  End Sub
58
59  Private Sub txtLength_Enter(sender As Object, e As EventArgs) Handles txtLength.Enter
60      txtLength.SelectAll()
61  End Sub
62
63  Private Sub txtWidth_Enter(sender As Object, e As EventArgs) Handles txtWidth.Enter
64      txtWidth.SelectAll()
65  End Sub
66
67  Private Sub CancelKeys(sender As Object, e As KeyPressEventArgs) Handles txtLength.KeyPress, txtWidth.KeyPress, txtDepth.KeyPress
68      ' Allow only numbers, the period, and the Backspace key.
69
70      If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> "." AndAlso e.KeyChar <> ControlChars.Back Then
71          e.Handled = True

```

1. & 3.

1. & 3.

```

72      End If
73  End Sub
74
75  Private Sub ClearGallons(sender As Object, e As EventArgs) Handles txtLength.TextChanged, txtWidth.TextChanged, txtDepth.TextChanged
76      lblGallons.Text = String.Empty
77  End Sub
78 End Class

```

### CH8\_A7.2 - array of structure variables = each array element contains member variables created within Structure - 43.Paper Solution-Structure - Appendix B

& comparison with 2x parallel 1D array used in 16.Paper Solution-Parallel, and with 1x 2D array used in 17.Paper Solution-TwoDim

- another advantage of using a **Structure** is that a **structure** variable can be stored in an **array**, even when its **members** have different data types
- = you can declare **array As** previously created **Structure**, so each **array** element can use any of the **member** variables created within the **Structure**

- recapitulation from: **CH8\_A7 - Structure** - composite of variables like **array**, but with a combination of a different data types - additional topic from **Appendix B**

#### Step 2 = declare a **structure variables** / declare an **array** of **structure variables**

- b. array of structure variables syntax:  
**As Array** of **structure** variables

**Dim/Static/Private** *arrayOfStructureName(highestSubscript As Integer) As YourStructureName*

#### Step 3 = use your **structure variables** / **array** of **structure variables** declared in: **Step 2**, and its **member variables** defined in: **Step 1**

- b. use your array of structure variables syntax:  
**As** *yourMemberVariable* **data type**

*... arrayOfStructureName(Subscript).yourMemberVariable ...*

**As** *yourMemberVariable* **data type**

- use an **array of structure** variables in the application: **43.Paper Solution-Structure**

- the **Paper Warehouse** application from previous examples can be used to illustrate this concept: **16.Paper Solution-Parallel**, and **17.Paper Solution-TwoDim**

- the application displays the **price** corresponding to the item **ID** entered by the user as:

|             |      |       |      |       |      |
|-------------|------|-------|------|-------|------|
| item ID:    | A45G | J63Y  | M93K | C20P  | F77T |
| price (\$): | 8.99 | 12.99 | 5.99 | 13.50 | 7.25 |

- you coded the application in **2 ways**:

- 1). you used **2 parallel 1D arrays**: - **16.Paper Solution-Parallel**
  - first **1D array** having the **String** data type to keep an **item ID**
  - second **1D array** having the **Double** data type to keep the item **price**

**CH8\_A3.1 - create 2x Parallel 1D Arrays** introduction & example

**CH8\_A3.2 - create 2x Parallel 1D Arrays** example: coding the **Paper Warehouse Application** (16.Paper Solution-Parallel)

- 2). you used only **one 2D array**: - **17.Paper Solution-TwoDim**
  - only one **2D array** having the **String** data type to keep both of the info

**CH8\_A4.1 - search 2D Array** introduction & example vs. **2x Parallel 1D Arrays**

**CH8\_A4.2 - search 2D Array** instead of **2x Parallel 1D Arrays** example: coding the **Paper Warehouse Application** (17.Paper Solution-TwoDim)

**notice:** - there are many different ways of solving the same problem

- in this appendix, you will code the application using a **1D array** of **structure** variables,  
where each **structure** variable will contain **2 member** variables:
  - a **String** variable for the **item ID**, and
  - a **Double** variable for the **price**
- > **priceList(0) - priceList(4)**
- > **.strId**
- > **.dblPrice**

- compare the different solutions for the **Paper Warehouse** application:

|                    |              | <b>16.Paper Solution-Parallel</b>                             | <b>17.Paper Solution-TwoDim</b>          | <b>43.Paper Solution-Structure</b>                                                                                            |
|--------------------|--------------|---------------------------------------------------------------|------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------|
|                    |              | - 2x parallel <b>1D arrays</b><br><b>As String, As Double</b> | - 1x <b>2D array</b><br><b>As String</b> | - 1x <b>1D array</b> + 2x <b>structure</b> members<br><b>As Structure</b> + <b>As String, As Structure</b> + <b>As Double</b> |
| <b>item ID:</b>    | <b>A45G</b>  | <b>strIds(0)</b>                                              | <b>strItems(0, 0)</b>                    | <b>priceList(0).strId</b>                                                                                                     |
| <b>price (\$):</b> | <b>8.99</b>  | <b>dblPrices(0)</b>                                           | <b>strItems(0, 1)</b>                    | <b>priceList(0).dblPrice</b>                                                                                                  |
| <b>item ID:</b>    | <b>J63Y</b>  | <b>strIds(1)</b>                                              | <b>strItems(1, 0)</b>                    | <b>priceList(1).strId</b>                                                                                                     |
| <b>price (\$):</b> | <b>12.99</b> | <b>dblPrices(1)</b>                                           | <b>strItems(1, 1)</b>                    | <b>priceList(1).dblPrice</b>                                                                                                  |
| <b>item ID:</b>    | <b>M93K</b>  | <b>strIds(2)</b>                                              | <b>strItems(2, 0)</b>                    | <b>priceList(2).strId</b>                                                                                                     |
| <b>price (\$):</b> | <b>5.99</b>  | <b>dblPrices(2)</b>                                           | <b>strItems(2, 1)</b>                    | <b>priceList(2).dblPrice</b>                                                                                                  |
| <b>item ID:</b>    | <b>C20P</b>  | <b>strIds(3)</b>                                              | <b>strItems(3, 0)</b>                    | <b>priceList(3).strId</b>                                                                                                     |
| <b>price (\$):</b> | <b>13.50</b> | <b>dblPrices(3)</b>                                           | <b>strItems(3, 1)</b>                    | <b>priceList(3).dblPrice</b>                                                                                                  |
| <b>item ID:</b>    | <b>F77T</b>  | <b>strIds(4)</b>                                              | <b>strItems(4, 0)</b>                    | <b>priceList(4).strId</b>                                                                                                     |
| <b>price (\$):</b> | <b>7.25</b>  | <b>dblPrices(4)</b>                                           | <b>strItems(4, 1)</b>                    | <b>priceList(4).dblPrice</b>                                                                                                  |

### 16.Paper Solution-Parallel

```
Private strIds() As String = {"A45G", "J63Y", "M93K", "C20P", "F77T"}  
Private dblPrices() As Double = {8.99, 12.99, 5.99, 13.5, 7.25}
```

### 17.Paper Solution-TwoDim

```
Private strItems(,) As String = {{"A45G", "8.99"}, {"J63Y", "12.99"}, {"M93K", "5.99"}, {"C20P", "13.50"}, {"F77T", "7.25"}}
```

### 43.Paper Solution-Structure

```
Structure ProductInfo  
  Public strId As String  
  Public dblPrice As Double  
End Structure
```

&

```
Private priceList(4) As ProductInfo
```

&

```
priceList(0).strId = "A45G"  
priceList(0).dblPrice = 8.99  
priceList(1).strId = "J63Y"  
priceList(1).dblPrice = 12.99  
priceList(2).strId = "M93K"  
priceList(2).dblPrice = 5.99  
priceList(3).strId = "C20P"  
priceList(3).dblPrice = 13.5  
priceList(4).strId = "F77T"  
priceList(4).dblPrice = 7.25
```

1). open the: ...VB2017\Chap081\_Exercise43.Paper Solution-Structure\Paper Solution.sln

2). open the **Main Form.vb**, Designer window, and Code Editor window

3). locate the the **form Class**'s declarations section

#1: - first, you will declare the **ProductInfo structure**, in the form **Class**'s declaration section, which will contain **2 members**:

- one for the item ID and -> **strId**
- one for the price -> **dblPrice**

-> enter the following **Structure** statement below the comment line **10**:

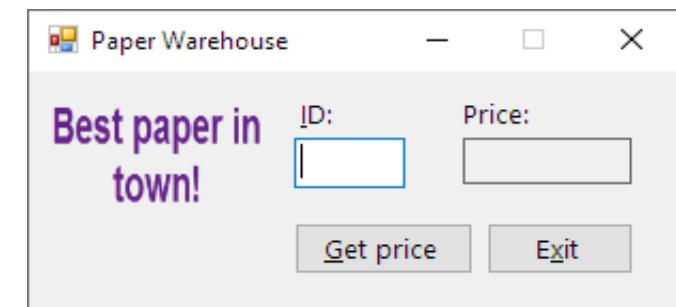
#2: - the procedure will store the price list in a **1D array** of **ProductInfo structure** variables

-> enter the following **1D array** declaration statement below the comment line **16**:

```
9  Public Class frmMain
10   ' Declare structure:
11   Structure ProductInfo
12     Public strId As String
13     Public dblPrice As Double
14   End Structure
15
16   ' Declare array of structure variables:
17   Private priceList(4) As ProductInfo
18
```

#1:

#2:



**note:** - keep in mind that each **element** in the **array** is a **structure** variable, and each **structure** variable contains **2 member** variables

4). the **frmMain\_Load** procedure will be responsible for storing the **5 IDs** and **5 prices** in **1D array**: **priceList**

#3: -> open/create the form's **Load** event procedure **frmMain\_Load** and enter the following comment and assignment statements:

```
-      Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
-        ' Fill array with IDs and prices:
-        priceList(0).strId = "A45G"
-        priceList(0).dblPrice = 8.99
-        priceList(1).strId = "J63Y"
-        priceList(1).dblPrice = 12.99
-        priceList(2).strId = "M93K"
-        priceList(2).dblPrice = 5.99
-        priceList(3).strId = "C20P"
-        priceList(3).dblPrice = 13.5
-        priceList(4).strId = "F77T"
-        priceList(4).dblPrice = 7.25
-      End Sub
```

#3:

## 5). locate the `btnGet_Click` procedure

- #4: - the procedure will use a **loop** to search each **element** in the **array**,  
comparing the value contained in the current **element's strId** member with the value stored in the **strSearchId** variable  
- the **loop** should **stop** searching either: a). when the **ID** is found, or  
b). when the end of the **array** is reached or  
-> enter the following **Do...Loop** under the comment line 25

```
19     Private Sub btnGet_Click(sender As Object, e As EventArgs) Handles btnGet.Click
20         Dim strSearchId As String
21         Dim intSub As Integer           ' = 0, will be used as a counter for loop
22
23         strSearchId = txtId.Text.Trim.ToUpper
24
25         ' Search the priceList array until the end of the array or the ID is found:
26         Do Until intSub = priceList.Length OrElse priceList(intSub).strId = strSearchId
27             intSub += 1
28         Loop
29
```

#4:

- #5: - the procedure will use a **selection structure** to determine why the **loop** ended and then take the appropriate action  
- if the value contained in the **intSub** variable is:  
a). less than the number of array elements: -> `priceList(intSub).strId = strSearchId`  
- the loop ended because the ID was located in the array  
-> in that case, the **selection structure's True** path should display the corresponding price in the **lblPrice** control  
  
b). equal to the number of array elements: -> `intSub = priceList.Length`  
- the loop ended because no mach has been found  
-> in that case, the **selection structure's False** path should display an appropriate message

-> below the **Loop**, enter the **selection structure**:

```
30         If intSub < priceList.Length Then
31             lblPrice.Text = priceList(intSub).dblPrice.ToString("C2")
32         Else
33             MessageBox.Show("ID not found", "Paper Warehouse", MessageBoxButtons.OK, MessageBoxIcon.Information)
34         End If
35     End Sub
```

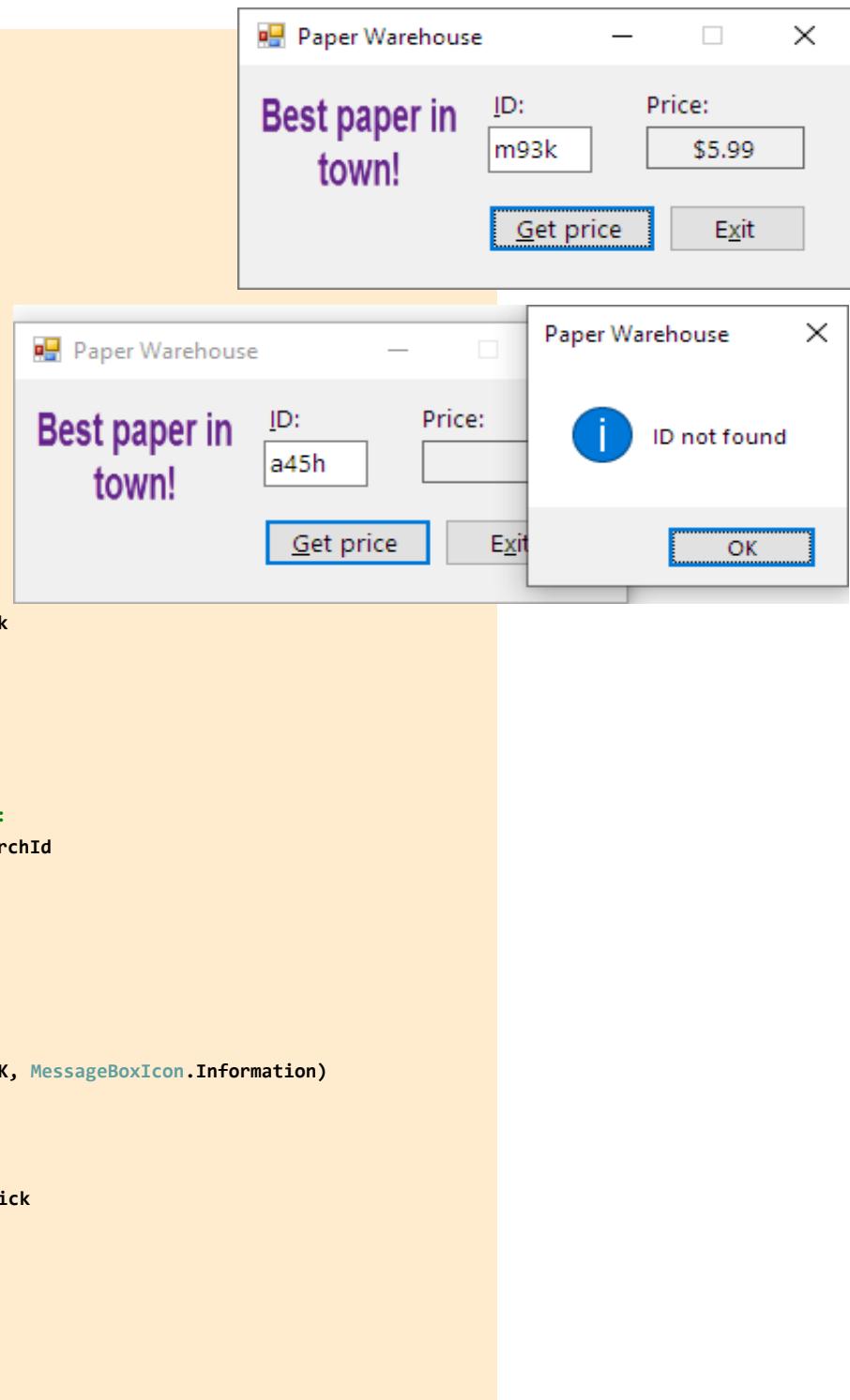
#5:

## 6). save the solution and start & test the application

- > in the **ID:** box type: **m93k** and then click the **Get price** button  
<- in the **Price** box appears: **\$5.99**  
-> in the **ID:** box type: **a45h** and then click the **Get price** button  
<- the "**ID not found**" message appears in a message box

7). entire code for the Paper Warehouse application using an array of structure variables:

```
1  ' Name:      Paper Project
2  ' Purpose:    Displays the price of an item.
3  ' Programmer: <your name> on <current date>
4
5  Option Explicit On
6  Option Strict On
7  Option Infer Off
8
9  Public Class frmMain
10   ' Declare structure:
11   Structure ProductInfo
12     Public strId As String
13     Public dblPrice As Double
14   End Structure
15
16   ' Declare array of structure variables:
17   Private priceList(4) As ProductInfo
18
19   Private Sub btnGet_Click(sender As Object, e As EventArgs) Handles btnGet.Click
20     Dim strSearchId As String
21     Dim intSub As Integer           ' = 0, will be used as a counter for Loop
22
23     strSearchId = txtId.Text.Trim.ToUpper
24
25     ' Search the priceList array until the end of the array or the ID is found:
26     Do Until intSub = priceList.Length OrElse priceList(intSub).strId = strSearchId
27       intSub += 1
28     Loop
29
30     If intSub < priceList.Length Then
31       lblPrice.Text = priceList(intSub).dblPrice.ToString("C2")
32     Else
33       MessageBox.Show("ID not found", "Paper Warehouse", MessageBoxButtons.OK, MessageBoxIcon.Information)
34     End If
35   End Sub
36
37   Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
38     Me.Close()
39   End Sub
40
41   Private Sub txtId_Enter(sender As Object, e As EventArgs) Handles txtId.Enter
42     txtId.SelectAll()
43   End Sub
```



```

44
45     Private Sub ClearPrice(sender As Object, e As EventArgs) Handles txtId.TextChanged
46         lblPrice.Text = String.Empty
47     End Sub
48
49     Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
50         ' Fill array with IDs and prices:
51         priceList(0).strId = "A45G"
52         priceList(0).dblPrice = 8.99
53         priceList(1).strId = "J63Y"
54         priceList(1).dblPrice = 12.99
55         priceList(2).strId = "M93K"
56         priceList(2).dblPrice = 5.99
57         priceList(3).strId = "C20P"
58         priceList(3).dblPrice = 13.5
59         priceList(4).strId = "F77T"
60         priceList(4).dblPrice = 7.25
61     End Sub
62 End Class

```

## CH8\_Summary: 1D Array, For Each...Next loop statement, Array.Sort & Array.Reverse methods, 2D Array, Parallel 1D Arrays / 2D Array

**1a. 1D Array:** (often used to either accumulate or count related values)

**1b.** you declare a **1D Array** using either of the syntax (version 1, version 2)

syntax version 1:  
As Array

Dim/Static/Private *arrayName(highestSubscript As Integer)* As *dataType*

As Array

- the *highestSubscript* argument in Version 1 is an Integer that specifies the highest subscript in the **Array**

- using Version 1's syntax, the computer automatically initializes the **Array** elements

e.g. Dim strWarehouse(2) As String

syntax version 2:  
As Array

Dim/Static/Private *arrayName()* As *dataType* = {*initialValues*}

As Array

- the *initialValues* section in Version 2 is a list of values separated by **commas** and enclosed in braces

- the values are used to initialize each element in the **Array**

e.g. Private strNames() As String = {"Harry Potter", "Ron Weasley", "Hermione Granger", "Lord Voldemort"}

**1c.** you refer to an element in **1D Array** using the **Array**'s name followed by the element's subscript specified in a set of parentheses following the **Array** name

e.g. intSales(3)

<- read as "intSales sub three"

**1d.** examples of statements that you can use to store data in a **1D Array** include an assignment statement and the **TryParse** method:

```
Dim dblPrice() As Double = {45, 25, 56.99, 33, 75}  
dblPrice(1) *= 1.25  
OR  
dblPrice(1) = dblPrice(1) * 1.25
```

```
Dim dblRates(9) As Double  
Double.TryParse(txtRate.Text, dblRates(2))
```

```
Dim intNumbers(4) As Integer  
Dim intSub As Integer  
Do While intSub < 5  
    intNumbers(intSub) = 100  
    lblShow.Text = lblShow.Text & intNumbers(intSub) & " "  
    intSub += 1  
Loop
```

```
Dim dblNumbers(4) As Double  
For intX As Integer = 1 To 5  
    dblNumbers(intX - 1) = intX ^ 2  
Next intX
```

```
Dim strCity(5) As String  
strCity(0) = "Nashville"
```

**1e.** to determine the number of elements, you can:

- use **Array's Length** property, or
  - e.g. `intNumberOfElements = ArrayName.Length`
- add the number **1** to the value returned by the **Array's GetUpperBound** method
  - e.g. `intNumberOfElements = ArrayName.GetUpperBound(0) + 1`

**1f.** to determine the highest subscript, you can:

- use the **Array's GetUpperBound** method with the number **0** as the specified dimension, or
  - e.g. `intHighestSubscript = ArrayName.GetUpperBound(0)`
- subtract the number **1** from the value stored in the **Array's Length** property
  - e.g. `intHighestSubscript = ArrayName.Length - 1`

**1g.** you use a **loop** to traverse (or to look at) each element in a **1D Array** (**Do...Loop**, **For...Next**, **For Each...Next**)

**2. For Each...Next** loop statement provides a convenient way to process one or more instructions for each element in a group

**CH8\_F3.1 - For Each...Next** loop statement - great for collection/array with unknown therefore variable number of elements

**CH8\_F3.3 - Traversing 1D Array: Do...Loop vs For...Next vs For Each...Next** code example with Harry Potter Application:

```
For Each eElementVariable As dataType In collection  
    ...loop body instructions...  
Next eElementVariable
```

**3a.** **Array.Sort** method sorts a **1D Array**'s values in **ascending** order

**3b.** **Array.Reverse** method **reverses** the order of the values stored in a **1D Array**

**3c.** to sort the values in descending order:

- #1. you first use the **Array.Sort** method to sort the values in **ascending** order and
- #2. then use the **Array.Reverse** method to **reverse** the methods

```
Array.Sort(arrayName As Array)
```

```
Array.Reverse(arrayName As Array)
```

**4a. 2D Array:**

**4b.** to declare a **2D Array**, use either of the syntax versions (version 1, version 2)

syntax version 1: `Dim/Static/Private arrayName(highestRowSub As Integer, highestColumnSub As Integer) As dataType` **As Array**

- the **highestRowSub** and **highestColumnSub** arguments are integers that specify the highest **row** and **column** subscripts in the **Array**
- using **Version 1**'s syntax, the computer automatically initializes the **Array** elements
  - e.g. `Dim strStateCapitals(49,1) As String`

syntax version 2:

**As Array**

**Dim/Static/Private** *arrayName*(,) **As** *dataType* = {{*rowInitialValues*}, {*rowInitialValues*},...}

**As Array**

- the **rowinitialValues** section is a list of values separated by commas and enclosed in braces
- you include a separate **rowinitialValues** section for each **row** in the array
- each **rowinitialValues** section should contain the **same** number of values as there are **columns** in the **Array**

e.g. **Private** dblSales(,) **As Double** = {{75.33, 9.65}, {23.55, 6.89}, {4.5, 89.3}, {100.67, 38.92}}

4c. you refer to an element using the **Array**'s name followed by a set of parentheses that contains the element's row subscript, a comma, and its column subscript

e.g. dblSales(3, 1)

<- read as "dblSales sub three comma one"

4d. examples of statements that you can use to store data in a **2D Array** include an assignment statement and the **TryParse** method:

```
Dim dblSales(,) As Double = {{75.33, 9.65}, {23.55, 6.89}, {4.5, 89.3}, {100.67, 38.92}}
Dim intRow As Integer
Dim intCol As Integer
Do While intRow <= 3
    intCol = 0
    Do While intCol <= 1
        dblSales(intRow, intCol) *= 1.1
        intCol += 1
    Loop
    intRow += 1
Loop
```

```
Dim strStateCapitals(49, 1) As String
strStateCapitals(0, 0) = "AL"
strStateCapitals(0, 1) = "Montgomery"
```

**Double.TryParse**(txtSales.Text, dblSales(0, 0))

```
Dim dblSales(,) As Double = {{75.33, 9.65}, {23.55, 6.89}, {4.5, 89.3}, {100.67, 38.92}}
dblSales(2, 1) *= 0.07
```

```
Static intNumSold(5, 4) As Integer
For intRow As Integer = 0 To 5
    For intColumn As Integer = 0 To 4
        intNumSold(intRow, intColumn) += 1
    Next intColumn
Next intRow
```

4e. to determine the highest row subscript, use the **Array**'s **GetUpperBound** method with the number **0** as the specified dimension

e.g. intHighestRowSubscript = *ArrayName*.GetUpperBound(0)

4f. to determine the highest column subscript, use the **Array**'s **GetUpperBound** method with the number **1** as the specified dimension

e.g. intHighestColumnSubscript = *ArrayName*.GetUpperBound(1)

4g. you traverse (or look at) each element in a **2D Array** using either one or two loops:

- one loop is all that is required when using the **For Each...Next** statement
- two loops are required when using the **Do...Loop** or **For...Next** statements

5. to associate the items in a **list box** (in other words, the **Items collection**) with the elements in an **Array**, add the appropriate items to the list box and then store each item's related value in its corresponding position in the **Array**

- each item's index in the list box should be the same as its subscript in the **Array**

**CH8\_A1.1 - Associate a 1D Array with a (lst): Array elements with Items Collection**, for they have a several **commonalities**

e.g.: **lblVicePres.Text** = strVPs(*lstPresidents*.SelectedIndex)

| <b>1stPresidents</b> |
|----------------------|
| George Washington    |
| George Bush          |
| Bill Clinton         |
| George W. Bush       |
| Barack Obama         |
| Donald J. Trump      |

<- index / subscript ->

| <b>strVPs</b>   |
|-----------------|
| John Adams      |
| Dan Quayle      |
| Albert Gore     |
| Richard Cheney  |
| Joseph R. Biden |
| Mike Pence      |

**1D Array**

6. to create **Parallel 1D Arrays**, create two or more **1D Arrays**, or instead use only one **2D Array**

- when assigning values to the **Arrays**, be sure that the values stored in each element in the **1st Array** corresponds

to the values stored in the same elements in the other **Arrays**

e.g.

**String 1D Array**

| strIds()  |        |
|-----------|--------|
| strIds(0) | "A45G" |
| strIds(1) | "J63Y" |
| strIds(2) | "M93K" |
| strIds(3) | "C20P" |
| strIds(4) | "F77T" |
| IDs       |        |

**Double 1D Array**

| <- index -> | dblPrices() |              |
|-------------|-------------|--------------|
| 0           | 8.99        | dblPrices(0) |
| 1           | 12.99       | dblPrices(1) |
| 2           | 5.99        | dblPrices(2) |
| 3           | 13.5        | dblPrices(3) |
| 4           | 7.25        | dblPrices(4) |
| prices      |             |              |

**String 2D Array**

| strItems(,,)   |        |         |
|----------------|--------|---------|
| strItems(0, 0) | "A45G" | "8.99"  |
| strItems(1, 0) | "J63Y" | "12.99" |
| strItems(2, 0) | "M93K" | "5.99"  |
| strItems(3, 0) | "C20P" | "13.50" |
| strItems(4, 0) | "F77T" | "7.25"  |
| IDs            |        | prices  |

## CH8 Key Terms

- **Accumulator arrays** - arrays whose elements are used to accumulate (add together) values
- **Aggregate operator** - an operator that returns a single value from a group of values; e.g. of aggregate operators in **LINQ** include **Average**, **Count**, **Max**, **Min**, **Sum**
- **Array.Reverse method** - reverses the order of the values stored in a 1D array
- **Array.Sort method** - sorts the values stored in a 1D array in ascending order
- **Array** - a group of related variables that have the same name and data type and are stored in consecutive locations in the RAM
- **Counter arrays** - arrays whose elements are used for counting something
- **Elements** - the variables in an array
- **For Each...Next statement** - used to code a loop whose instructions should be processed for each element in a group
- **GetUpperBound method** - returns an integer that represents the highest subscript in a specified dimension of an array
  - when used with a 1D array, the dimension is 0
  - when used with a 2D array, the dimension is: - 0 for the row subscript
    - 1 for the column subscript
- **Language-Integrated Query / Query language / LINQ** - the query language built into VB, used to retrieve a specific information from a variety of data sources
- **Length property** - one of the properties of a 1D array; stores an integer that represents the number of array elements
- **LINQ / Language-Integrated Query / Query language** - the query language built into VB, used to retrieve a specific information from a variety of data sources
- **Member variables** - the variables contained in a Structure
- **One-dimensional array** - an array whose elements are identified by a unique subscript
- **Parallel one-dimensional arrays** - two or more 1D arrays whose elements are related by their subscripts (positions) in the arrays
- **Populating the array** - refers to the process of initializing the elements in an array
- **Query language / LINQ / Language-Integrated Query** - the query language built into VB, used to retrieve a specific information from a variety of data sources
- **Scalar variable** - another name for a simple variable
- **Simple variable** - a variable that is unrelated to any other variable in the computer's RAM; also called a scalar variable
- **Structure statement** - used to create user-defined data types called structures
- **Structure variables** - variables declared using a Structure as the data type
- **Structures / User-defined data types** - data types created by the Structure statement
  - allow the programmer to group related items into one unit
- **Subscript** - a unique integer that identifies the position of an element in an array

- **Two-dimensional array** - an array made up of rows and columns
  - each element has the same name and data type and is identified by a unique combination of two subscripts ->
    - > a row subscript and a column subscript
- **User-defined data types / Structures** - data types created by the Structure statement
  - allow the programmer to group related items into one unit

**CH8\_Exercises****Chap08\ Exercise1****18.Waterson Solution-Highest-DoLoop\_EXERCISE 1\_introductory**

- 1D array, change original loop: **For...Next** for the loop: **Do While...Loop**, **frmMain\_Load**, **For Each...Next**, **GetUpperBound(0)**, **Do While...Loop**, **ElseIf**

**19.Electricity Solution\_EXERCISE 2\_introductory**

- 1D array, **For Each...Next** loop

**20.Gross Pay Solution\_EXERCISE 3\_introductory**

- class-level 1D array, **lst.SelectedIndex**, **lst.SelectedIndexChanged**, **txt.TextChanged**, **txt.Enter**, **txt.KeyPress**

**21.CityState Solution\_EXERCISE 4\_introductory**

- 2x parallel 1D array, **lst.Items.Add**, **Do While...Loop**

**22.Professor Juarez Solution\_EXERCISE 5\_intermediate**

- 2x parallel 1D array, **GetUpperBound(0)**, **Do While...Loop**, **lst.Items.Add**, **lst.SelectedIndexChanged**

**23.Waterson Solution-Lowest\_EXERCISE 6\_intermediate**

- class-level 1D array, **GetUpperBound(0)**, **frmMain\_Load**, **For Each...Next**, **lst.Items.Add**

**24.Professor Juarez Solution-TwoDim\_EXERCISE 7\_intermediate**

- 2x parallel 1D array, **GetUpperBound(0)**, vs 2D array, **GetUpperBound(1)**  
**lst.Items.Add**, **Do While...Loop**, **lst.Items.Add**, **lst.SelectedIndexChanged**

**25.Calories Solution\_EXERCISE 8\_intermediate**

- 1D array, **Math.Round** method, **For Each...Next**, **ElseIf**

**26.Computer Solution\_EXERCISE 9\_intermediate**

- **Static 2D array** used as an accumulator, **lst.SelectedIndex**, **lst.Items.Add**, **txt\_Enter**, **txt\_KeyPress**, **frmMain\_Load**

**27.Schneider Solution\_EXERCISE 10\_advanced**

- 2x parallel 1D array, **GetUpperBound(0)**, **MessageBox.Show**, **txt\_KeyPress**, **txt\_TextChanged**

**28.Schneider Solution-TwoDim\_EXERCISE 11\_advanced**

- 2D array, **MessageBox.Show**, **txt\_KeyPress**, **txt\_TextChanged**

**29.Kraston Solution\_EXERCISE 12\_advanced**

- class-level 2D array, **txt\_KeyPress**, **txt\_TextChanged**, **txt\_Enter**

**30.Shipping Depot Solution\_EXERCISE 13\_advanced**

- 2D array 4x3, **GetUpperBound(0)**, **For Each...Next**

**31.Modified Shipping Depot Solution\_EXERCISE 14\_advanced**

- 2D array 3x4 instead of 2D array 4x3, **GetUpperBound(1)**, **For Each...Next**

**32.Lottery Solution\_EXERCISE 15\_advanced**

- 1D accumulator array, random integer: **intRandGen As New Random**, method **intRandGen.Next()**

**33.Gas Prices Solution\_EXERCISE 16\_advanced\_NOT FINISHED**

- does not make any sense to me, therefore not finished

**34.Organic Solution\_EXERCISE 17\_advanced**

- 2D array 6x3, 1D array, **GetUpperBound(0)**, **For Each...Next**

**35.Bindy Solution\_EXERCISE 18\_advanced**

- 2D Double array 2x10, 1D String array (10), **Select Case**, **rad.Checked**, **lst.Items.Insert**, **ControlChars.Tab**, **If Is...Then**, **Private Sub...**, **rad.CheckedChanged**

**36.Adaline Solution\_EXERCISE 19\_advanced**

- 1D array, 2D array, **For Each...Next**, **ElseIf**, **SelectedIndexChanged**, **lst.Items.Add**, **GetUpperBound(0)**, **frmMain\_Load**, **lst.Items.Add**, **Select Case**, **Private Sub...**

**37.ReDim Solution\_EXERCISE 20\_advanced**

- 1D accumulator array, **ReDim Preserve**, **MessageBox.Show**, **txt\_Enter**, **txt\_KeyPress**, **txt\_TextChanged**

**38.OnYourOwn Solution\_EXERCISE 21\_**

- no idea, not done

**39.FixIt Solution\_EXERCISE 22**

- 2D array 5x2, **GetUpperBound(0)**

- 1D array, change original loop: **For...Next** for the loop: **Do While...Loop**, **frmMain\_Load**, **For Each...Next**, **GetUpperBound(0)**, **Do While...Loop**, **ElseIf**

1. In this exercise, you modify one of the Waterson Company applications from this chapter's Apply lesson. Use Windows to make a copy of the Waterson Solution-Highest folder. Rename the copy Waterson Solution-Highest-DoLoop. Open the Waterson Solution.sln file contained in the Waterson Solution-Highest-DoLoop folder. Change the For...Next statement in the btnDisplay\_Click procedure to the Do...Loop statement. Save the solution and then start and test the application.

```
1  Option Explicit On
2  Option Strict On
3  Option Infer Off
4  ' change For...Next to Do...Loop
5
6  Public Class frmMain
7
8      ' Class-level array.
9      Private dblPrices() As Double = {85.7, 89.5, 91, 99, 97.5, 96, 96.8, 96.8, 96, 99}
10
11     Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
12         ' Fills list box with prices.
13
14         For Each dblStockPrice As Double In dblPrices
15             lstPrices.Items.Add(dblStockPrice.ToString("N2"))
16         Next dblStockPrice
17     End Sub
18
19     Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
20         ' Displays the highest stock price and the number of days at that price.
21
22         ' Declare and initialize variables:
23         Dim intLastSub As Integer = dblPrices.GetUpperBound(0)    ' = 9
24         Dim dblHighest As Double = dblPrices(0)
25         Dim intDays As Integer = 1
26
27         ' Do...Loop instead of For...Next:
28         Dim intSub As Integer = 1
```

```

30     ' original:
31     'For intSub As Integer = 1 To intLastSub
32     ' If dblPrices(intSub) = dblHighest Then
33     '     intDays += 1
34     ' ElseIf dblPrices(intSub) > dblHighest Then
35     '     dblHighest = dblPrices(intSub)
36     '     intDays = 1
37     ' End If
38     'Next intSub
39
40     Do While intSub <= intLastSub
41         If dblPrices(intSub) = dblHighest Then
42             intDays += 1
43         ElseIf dblPrices(intSub) > dblHighest Then
44             dblHighest = dblPrices(intSub)
45             intDays = 1
46         End If
47         intSub += 1
48     Loop
49
50     lblHighest.Text = dblHighest.ToString("C2")
51     lblDays.Text = intDays.ToString
52 End Sub
53
54 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
55     Me.Close()
56 End Sub
57 End Class

```

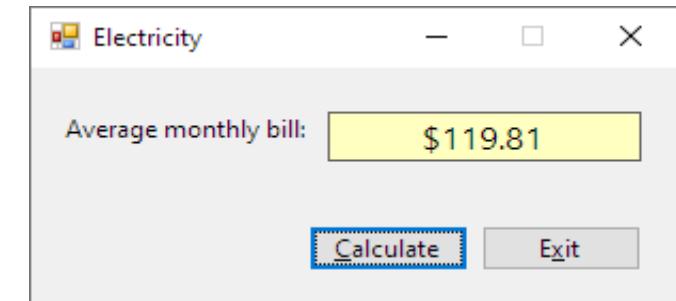


2. Open the Electricity Solution.sln file contained in the VB2017\Chap08\Electricity Solution folder. Open the Code Editor window and locate the btnCalc\_Click procedure. Each of the 12 numbers in the array declaration statement represents the cost of electricity for a month. The procedure should use the For Each...Next statement to calculate the average monthly cost for electricity. Display the average with a dollar sign and two decimal places in the lblAvg control. Code the procedure. Save the solution and then start and test the application. (The average is \$119.81.)

```

1  Option Explicit On
2  Option Strict On
3  Option Infer Off
4  ' 12 elements = 12 monthly costs for electricity
5  ' use For Each...Next to calculate the average monthly cost for electricity
6  ' display in lblAvg: $ and 2 decimal places
7  ' the result should be: $119.81
8
9  Public Class frmMain
10
11     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
12         Me.Close()
13     End Sub
14
15     Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
16         ' Display the average monthly cost.
17
18         Dim dblMonthly() As Double = {141.71, 156.75, 179.25, 141.71, 130.19, 115.05,
19                           95.65, 86.78, 85.45, 79.99, 98.45, 126.78}
20
21         Dim dblAvg As Double
22         Dim dblTotal As Double
23
24         For Each dblElement As Double In dblMonthly
25             dblTotal += dblElement
26         Next
27
28         dblAvg = dblTotal / dblMonthly.Length
29         lblAvg.Text = dblAvg.ToString("C2")
30
31     End Sub
32
33 End Class

```



## Chap08\Exercise1

### 20.Gross Pay Solution\_EXERCISE 3\_introductory

- class-level 1D array, lst.SelectedIndex, lst.SelectedIndexChanged, txt.TextChanged, txt.Enter, txt.KeyPress

3. Open the Gross Pay Solution.sln file contained in the VB2017\Chap08\Gross Pay Solution folder. The interface provides a text box for entering the number of hours an employee worked. It also provides a list box for selecting the employee's pay code. The btnCalc\_Click procedure should display the gross pay, using the number of hours worked and the pay rate corresponding to the selected code. The pay codes and rates are listed in Figure 8-47. Employees working more than 40 hours receive time and a half for the hours worked over 40. Code the application. Use a class-level array to store the pay rates. Save the solution and then start and test the application.

| Pay code | Pay rate |
|----------|----------|
| P23      | 10.50    |
| P56      | 12.50    |
| F45      | 14.25    |
| F68      | 15.75    |
| F96      | 17.65    |

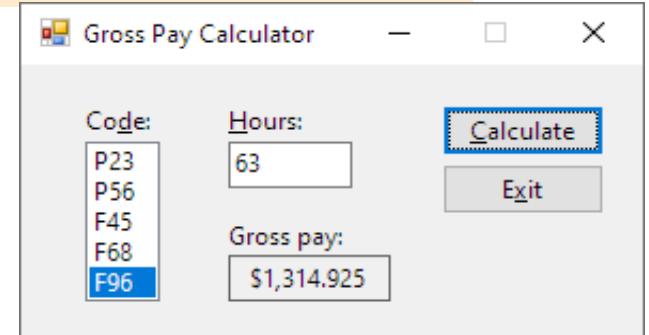
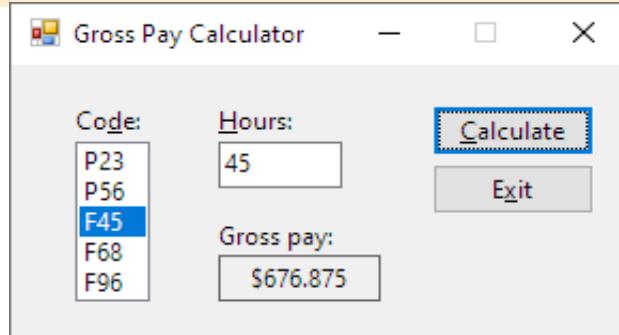
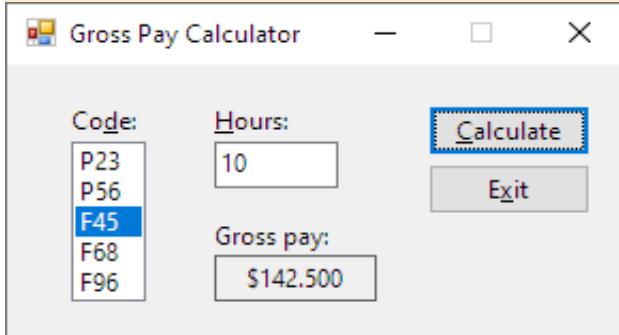
Figure 8-47 Pay codes and rates for Exercise 3

```
1  Option Explicit On
2  Option Strict On
3  Option Infer Off
4  ' display the gross pay using user input in (txtHours) and selected pay code in (lstCodes)
5  ' if more than 40 hours, those extra hours time and a half
6  ' use class-level array to store the pay rates (codes)
7
8  Public Class frmMain
9
10     Private dblPayRate() As Double = {10.5, 12.5, 14.25, 15.75, 17.65}
11
12     Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
13         Dim dblGross As Double          ' final calculation
14         Dim dblExtraHours As Double    ' more than 40 hours worked
15         Dim dblHours As Double        ' input from user: hours worked
16         Double.TryParse(txtHours.Text, dblHours)
17
18         If dblHours > 40 Then
19             dblExtraHours = dblHours - 40
20             dblGross = (40 * dblPayRate(lstCodes.SelectedIndex)) + ((dblExtraHours * 1.5) * dblPayRate(lstCodes.SelectedIndex))
21         Else
22             dblGross = dblHours * dblPayRate(lstCodes.SelectedIndex)
23         End If
24         ' test1: 45 hours,F45 (14.25):
25         ' 40*14.25 = 570
26         ' 5*1.5 = 7.5
27         ' 7.5*14.25 = 106.875
28         ' 570+106.875 = 676.875
29     End Sub
```

```

30      ' test2: 63 hours, F96 (17.65):
31      ' 40*17.65 = 706
32      ' 23*1.5 = 34.5
33      ' 34.5*17.65 = 608.925
34      ' 706+608.925 = 1314.925
35
36      lblGross.Text = dblGross.ToString("C3")
37  End Sub
38
39  Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
40      ' Selects the first pay code in the list box.
41      lstCodes.SelectedIndex = 0
42  End Sub
43
44  Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
45      Me.Close()
46  End Sub
47
48  Private Sub ClearGross(sender As Object, e As EventArgs) Handles lstCodes.SelectedIndexChanged, txtHours.TextChanged
49      lblGross.Text = String.Empty
50  End Sub
51
52  Private Sub txtHours_Enter(sender As Object, e As EventArgs) Handles txtHours.Enter
53      txtHours.SelectAll()
54  End Sub
55
56  Private Sub txtHours_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtHours.KeyPress
57      ' Accept only numbers, the period, and the Backspace key.
58      If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> "." AndAlso e.KeyChar <> ControlChars.Back Then
59          e.Handled = True
60      End If
61  End Sub
62
63 End Class

```

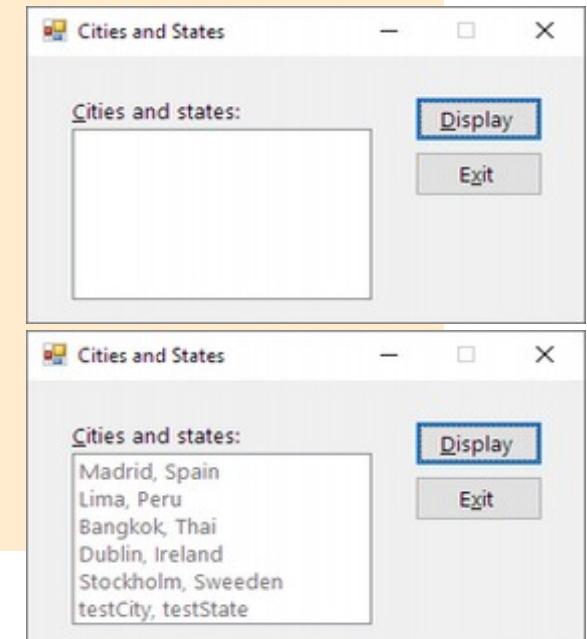


4. Open the CityState Solution.sln file contained in the VB2017\Chap08\CityState Solution folder. Open the Code Editor window. Locate the btnDisplay\_Click procedure. The procedure should declare and initialize two parallel one-dimensional arrays named strStates and strCities. Use the names of any five states to initialize the strStates array. Initialize the strCities array with the capital of its corresponding state in the strStates array. The procedure should display the contents of the arrays in the list box, using the following format: the city name followed by a comma, a space, and the state name. (Be sure to clear the list box before displaying the names.) Save the solution and then start and test the application.

```

1  Option Explicit On
2  Option Strict On
3  Option Infer Off
4
5  Public Class frmMain
6
7      Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
8          ' declare and initialize 2x Parallel 1D Arrays: strStates() & strCities() with any 5 corresponding States and their Capitals
9          ' display the contents of the arrays in (lstCitiesAndStates) in format like: strCities, strStates
10         ' be sure to clear the lst before displaying the names
11
12         Dim strStates() As String = {"Spain", "Peru", "Thailand", "Ireland", "Sweden", "testState"}
13         Dim strCities() As String = {"Madrid", "Lima", "Bangkok", "Dublin", "Stockholm", "testCity"}
14         lstCitiesAndStates.Items.Clear()
15         Dim intCounter As Integer = 0
16
17         Do While intCounter < strStates.Length
18             lstCitiesAndStates.Items.Add(strCities(intCounter) & ", " & strStates(intCounter))
19             intCounter += 1
20         Loop
21
22     End Sub
23     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
24         Me.Close()
25     End Sub
26 End Class

```



- 2x parallel 1D array, GetUpperBound(0), Do While...Loop, lst.Items.Add, lst.SelectedIndexChanged

5. Open the Professor Juarez Solution.sln file contained in the VB2017\Chap08\Professor Juarez Solution folder.
  - a. Open the Code Editor window and locate the btnDisplay\_Click procedure. The procedure declares and initializes two parallel one-dimensional arrays named strNames and strGrades. Code the procedure to display the names of students who have earned the grade selected in the lstGrades control. It should also display the number of students who have earned that grade.
  - b. The first item in the lstGrades control should be selected when the interface appears. Code the appropriate procedure.
  - c. The contents of the lstNames and lblNumber controls should be cleared when a different grade is selected in the lstGrades control. Code the appropriate procedure.
  - d. Save the solution and then start and test the application.

```

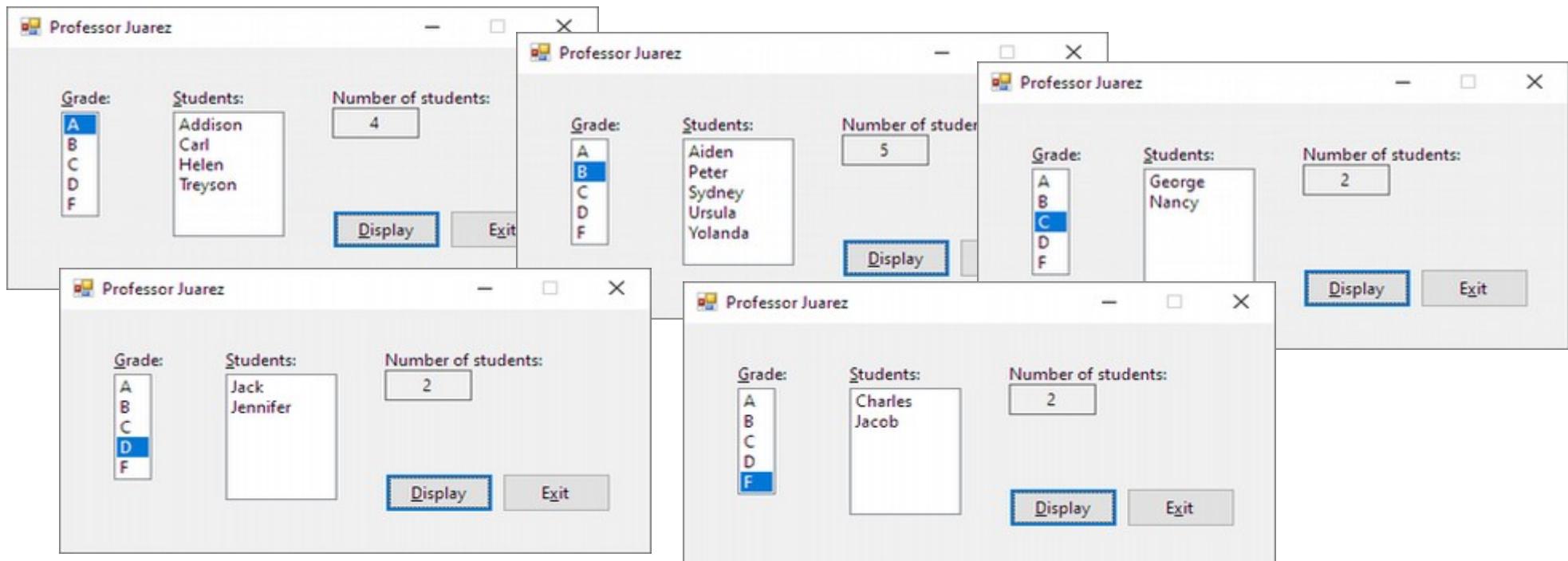
1 Option Explicit On
2 Option Strict On
3 Option Infer Off
4
5 Public Class frmMain
6   ' 2x Parallel 1D Arrays strNames() & strGrades()
7   ' from (lstGrades) select the grade strGrades() & display corresponding names in (lstNames) from strNames() and number in (lblNumber)
8   ' 1st item in (lstGrades) should be selected when application starts
9   ' when a different item in (lstGrades) is selected, clear (lstNames) & lblNumber
10  ' (lstGrades) = A B C D F, strGrades() = strNames() -> (lstNames)
11
12  Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
13    ' Display the names and number of students earning a specific grade.
14    lstNames.Items.Clear()
15    lblNumber.Text = Nothing
16
17    Dim strNames() As String = {"Helen", "Peter", "Yolanda", "Carl", "Jennifer", "Charles", "Addison", "Aiden", "Treysen",
18                           "Sydney", "Jacob", "Nancy", "George", "Ursula", "Jack"}
19    Dim strGrades() As String = {"A", "B", "B", "A", "D", "F", "A", "B", "A", "B", "F", "C", "C", "B", "D"}
20    Dim intIndex As Integer = Nothing
21    Dim intCounter As Integer = Nothing
22
23    Do While intIndex <= strGrades.GetUpperBound(0) ' 15 elements, indexes 0-14
24      'MessageBox.Show(intIndex.ToString()) ' 0-14 ok
25
26      If lstGrades.SelectedItem Is strGrades(intIndex) Then
27        lstNames.Items.Add(strNames(intIndex))
28        intCounter += 1
29      End If

```

```

30
31     intIndex += 1
32     Loop
33
34     lblNumber.Text = intCounter.ToString
35 End Sub
36
37 Private Sub lstGrades_SelectedIndexChanged(sender As Object, e As EventArgs) Handles lstGrades.SelectedIndexChanged
38     lstNames.Items.Clear()
39     lblNumber.Text = Nothing
40
41 End Sub
42
43 Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
44     lstGrades.SelectedIndex = 0
45 End Sub
46
47 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
48     Me.Close()
49 End Sub
50
51 End Class

```



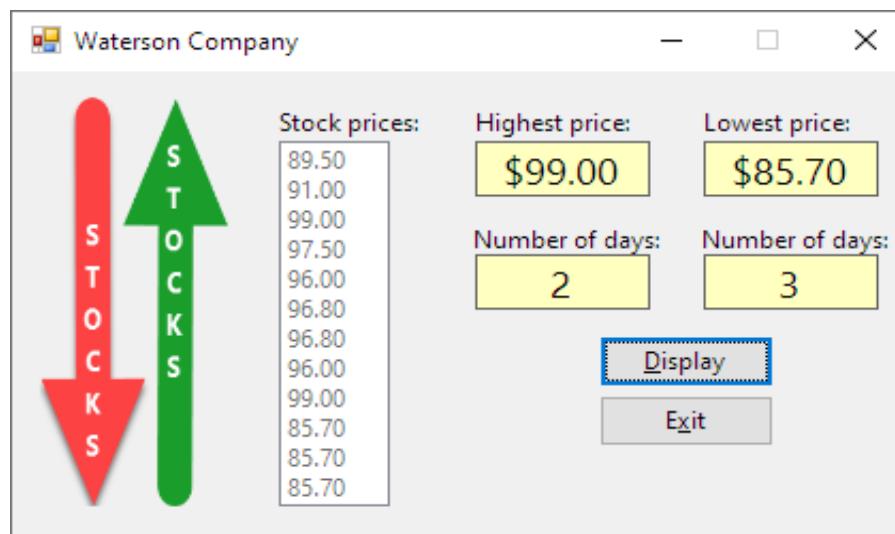
6. In this exercise, you modify one of the Waterson Company applications from this chapter's Apply lesson. Use Windows to make a copy of the Waterson Solution-Highest folder. Rename the copy Waterson Solution-Lowest. Open the Waterson Solution.sln file contained in the Waterson Solution-Lowest folder. The application should now display the lowest price and the number of days the stock closed at that price. Make the appropriate modifications to the interface as well as to the comments and code in the Code Editor window. Save the solution and then start and test the application.

```
1 Option Explicit On
2 Option Strict On
3 Option Infer Off
4 ' MOD: instead of highest price & number of days, display the lowest price and number of days
5 ' result should be: 85.7 & 3
6 Public Class frmMain
7
8     ' Class-level array.
9     Private dblPrices() As Double = {89.5, 91, 99, 97.5, 96, 96.8, 96.8, 96, 99, 85.7, 85.7, 85.7}
10
11    Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
12        ' Fills list box with prices.
13
14        For Each dblStockPrice As Double In dblPrices
15            lstPrices.Items.Add(dblStockPrice.ToString("N2"))
16        Next dblStockPrice
17    End Sub
18
19    Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
20        ' Displays the highest stock price and the number of days at that price.
21
22        ' Declare and initialize variables:
23        Dim intLastSub As Integer = dblPrices.GetUpperBound(0)
24        Dim dblHighest As Double = dblPrices(0)
25        Dim intDays As Integer = 1
26
27        For intSub As Integer = 1 To intLastSub
28            If dblPrices(intSub) = dblHighest Then
29                intDays += 1
30            ElseIf dblPrices(intSub) > dblHighest Then
31                dblHighest = dblPrices(intSub)
32                intDays = 1
33            End If
34        Next intSub
```

```

35
36     lblHighest.Text = dblHighest.ToString("C2")
37     lblDays.Text = intDays.ToString
38
39     ' MOD: display the lowest & number of days:
40     Dim dblLowest As Double = dblPrices(0)
41     Dim intDaysLowest As Integer
42
43     For Each dblElements As Double In dblPrices
44         If dblElements < dblLowest Then
45             dblLowest = dblElements
46             intDaysLowest = 1
47         ElseIf dblElements = dblLowest Then
48             intDaysLowest += 1
49         End If
50     Next dblElements
51
52     lblLowest.Text = dblLowest.ToString("C2")
53     lblDaysLowest.Text = intDaysLowest.ToString
54
55 End Sub
56
57 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
58     Me.Close()
59 End Sub
60 End Class

```



## Chap08\Exercise1

### 24. Professor Juarez Solution-TwoDim\_EXERCISE 7\_intermediate

- 2x parallel **1D array**, **GetUpperBound(0)**, vs **2D array**, **GetUpperBound(1)**  
**lst.Items.Add**, **Do While...Loop**, **lst.Items.Add**, **lst.SelectedIndexChanged**

7. In this exercise, you modify the Professor Juarez application from Exercise 5. Use Windows to make a copy of the Professor Juarez Solution folder. Rename the copy Professor Juarez Solution-TwoDim. Open the Professor Juarez Solution.sln file contained in the Professor Juarez Solution-TwoDim folder. Change the two parallel arrays to a two-dimensional array named **strStudents**, and then make the appropriate modifications to the **btnDisplay\_Click** procedure's code. Save the solution and then start and test the application.

## Chap08\Exercise1

### 22. Professor Juarez Solution\_EXERCISE 5\_intermediate

- 2x parallel **1D array**, **GetUpperBound(0)**, **Do While...Loop**, **lst.Items.Add**, **lst.SelectedIndexChanged**

5. Open the Professor Juarez Solution.sln file contained in the VB2017\Chap08\Professor Juarez Solution folder.
- a. Open the Code Editor window and locate the **btnDisplay\_Click** procedure. The procedure declares and initializes two parallel one-dimensional arrays named **strNames** and **strGrades**. Code the procedure to display the names of students who have earned the grade selected in the **lstGrades** control. It should also display the number of students who have earned that grade.
  - b. The first item in the **lstGrades** control should be selected when the interface appears. Code the appropriate procedure.
  - c. The contents of the **lstNames** and **lblNumber** controls should be cleared when a different grade is selected in the **lstGrades** control. Code the appropriate procedure.
  - d. Save the solution and then start and test the application.

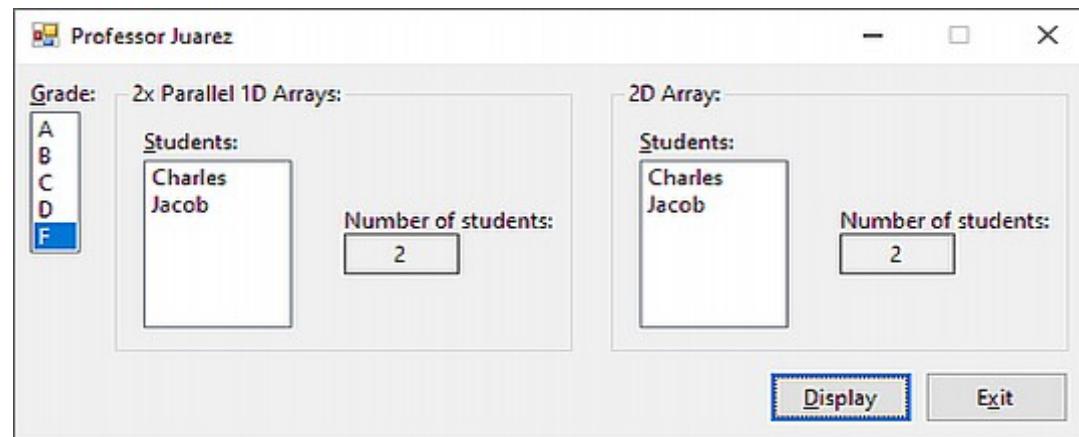
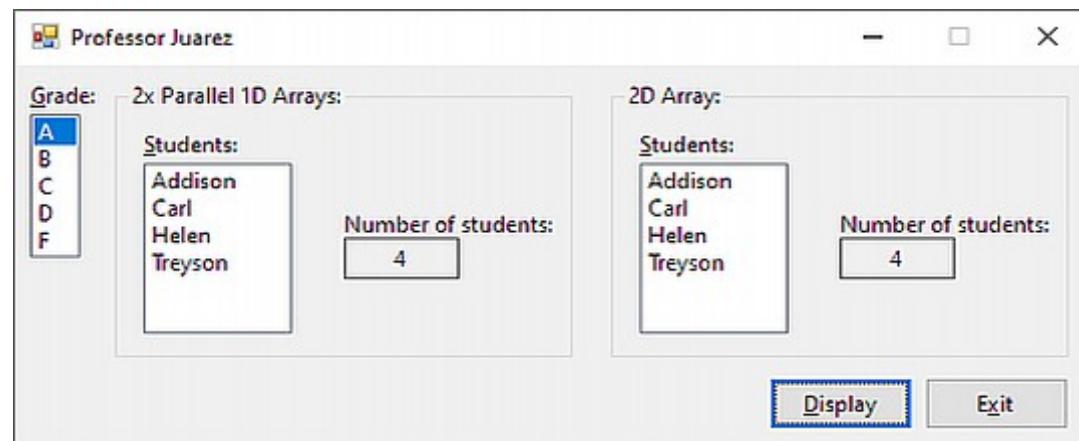
```
1 Option Explicit On
2 Option Strict On
3 Option Infer Off
4
5 Public Class frmMain
6     ' 2x Parallel 1D Arrays strNames() & strGrades()
7     ' from (lstGrades) select the grade strGrades() & display corresponding names in (lstNames) from strNames() and number in (lblNumber)
8     ' 1st item in (lstGrades) should be selected when application starts
9     ' when a different item in (lstGrades) is selected, clear (lstNames) & lblNumber
10    ' (lstGrades) = A B C D F, strGrades() = strNames() -> (lstNames)
11    ' MOD: change 2x Parallel 1D Arrays to 2D Array named strStudents(,) and make the code changes to make it work
12
13    Private Sub Cleaning()
14        lstNames.Items.Clear()
15        lstNames2D.Items.Clear()
16        lblNumber.Text = Nothing
17        lblNumber2D.Text = Nothing
18    End Sub
```

```
19
20  Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
21      ' Display the names and number of students earning a specific grade.
22      Cleaning()
23
24      Dim strNames() As String = {"Helen", "Peter", "Yolanda", "Carl", "Jennifer", "Charles", "Addison",
25          "Aiden", "Treyson", "Sydney", "Jacob", "Nancy", "George", "Ursula", "Jack"}
26      Dim strGrades() As String = {"A", "B", "B", "A", "D", "F", "A",
27          "B", "A", "B", "F", "C", "C", "B", "D"}
28      Dim intIndex As Integer = Nothing
29      Dim intCounter As Integer = Nothing
30
31      Do While intIndex <= strGrades.GetUpperBound(0) ' 15 elements, indexes 0-14
32          'MessageBox.Show(intIndex.ToString)    ' 0-14 ok
33          If lstGrades.SelectedItem Is strGrades(intIndex) Then
34              lstNames.Items.Add(strNames(intIndex))
35              intCounter += 1
36          End If
37          intIndex += 1
38      Loop
39      lblNumber.Text = intCounter.ToString
40
41      ' MOD: change 2x Parallel 1D Arrays to 2D Array named strStudents(,) and make the code changes to make it work
42      'lstNames2D, lblNumber2D
43      Dim strStudents(,) As String = {{"A", "B", "B", "A", "D", "F", "A", "B", "A", "B", "F", "C", "C", "B", "D"}, 
44          {"Helen", "Peter", "Yolanda", "Carl", "Jennifer", "Charles", "Addison", "Aiden", "Treyson",
45          "Sydney", "Jacob", "Nancy", "George", "Ursula", "Jack"}}
46      Dim intRowIndex As Integer = Nothing
47      Dim intCounter2D As Integer = Nothing
48
49      Do While intRowIndex <= strStudents.GetUpperBound(1)
50          'MessageBox.Show(intRowIndex.ToString)    ' 0-14 ok
51
52          If lstGrades.SelectedItem Is strStudents(0, intRowIndex) Then
53              lstNames2D.Items.Add(strStudents(1, intRowIndex))
54              intCounter2D += 1
55          End If
56
57          intRowIndex += 1
58      Loop
59
60      lblNumber2D.Text = intCounter2D.ToString
61
62  End Sub
```

```

63
64      Private Sub lstGrades_SelectedIndexChanged(sender As Object, e As EventArgs) Handles lstGrades.SelectedIndexChanged
65          Cleaning()
66
67      End Sub
68
69      Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
70          lstGrades.SelectedIndex = 0
71      End Sub
72
73      Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
74          Me.Close()
75      End Sub
76
77  End Class

```



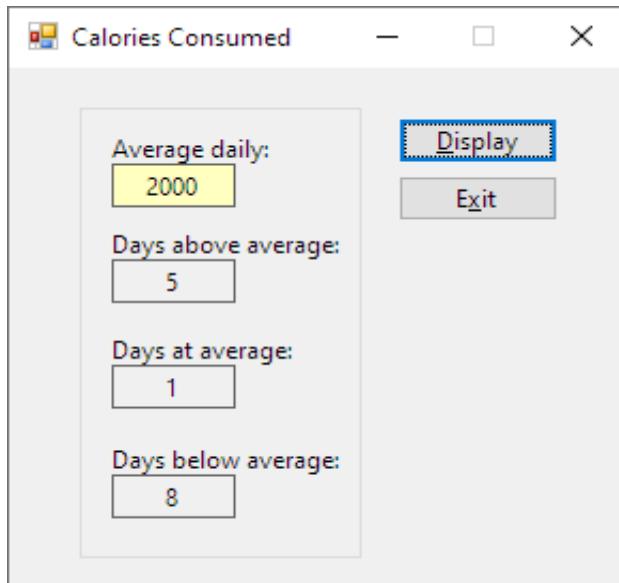
8. Open the Calories Solution.sln file contained in the VB2017\Chap08\Calories Solution folder. Open the Code Editor window and locate the btnDisplay\_Click procedure. The procedure declares and initializes a one-dimensional array named `intCalories`. The array stores the numbers of daily calories consumed. The procedure should calculate and display the average number of calories consumed; use the `Math.Round` method to round the average to an integer. (You learned about the `Math.Round` method in Chapter 6.) The procedure should also display the number of days in which the daily calories were greater than the average, the number of days in which the daily calories were the same as the average, and the number of days in which the daily calories were less than the average. Code the procedure. Save the solution and then start and test the application.

```
1  Option Explicit On
2  Option Strict On
3  Option Infer Off
4  ' 1D intCalories() stores the numbers of daily calories consumed
5  ' (lblAvg) = calculate & display average number of calories consumed, using Math.Round method to round the average to an integer
6  ' display number of days when daily calories were:
7  '     (lblAboveAvg) = greater than the average
8  '     (lblAtAvg) = the same as the average
9  '     (lblBelowAvg) = less than the average
10
11 Public Class frmMain
12     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
13         Me.Close()
14     End Sub
15
16     Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
17         ' Display calorie information.
18
19         Dim intCalories() As Integer = {2250, 1920, 2125, 2505, 2010, 1950, 1825, 2025, 2000,
20                         1820, 1990, 1950, 1875, 1750} ' 27 995/14 = 1999.6428
21         Dim intSuma As Integer
22         Dim dblAvg As Double          ' 2000
23         Dim intDaysAboveAvg As Integer ' 5
24         Dim intDaysAtAvg As Integer    ' 1
25         Dim intDaysBelowAvg As Integer ' 8
26
27         For Each intElement As Integer In intCalories
28             intSuma += intElement
29         Next
```

```

30
31     ' Average daily calories: 27995/14 = 1999.6428
32     dblAvg = intSuma / intCalories.Length
33     dblAvg = Math.Round(dblAvg, 0)    ' ==2000
34
35     ' Days:
36     For Each intElement As Integer In intCalories
37         If intElement > dblAvg Then
38             intDaysAboveAvg += 1
39         ElseIf intElement < dblAvg Then
40             intDaysBelowAvg += 1
41         Else
42             intDaysAtAvg += 1
43         End If
44     Next
45
46     lblAvg.Text = dblAvg.ToString
47     lblAboveAvg.Text = intDaysAboveAvg.ToString
48     lblAtAvg.Text = intDaysAtAvg.ToString
49     lblBelowAvg.Text = intDaysBelowAvg.ToString
50
51 End Sub
End Class

```



## Chap08\Exercise1

### 26.Computer Solution\_EXERCISE 9\_intermediate

- **Static 2D array** used as an accumulator, `lst.SelectedIndex`, `lst.Items.Add`, `txt_Enter`, `txt_KeyPress`, `frmMain_Load`

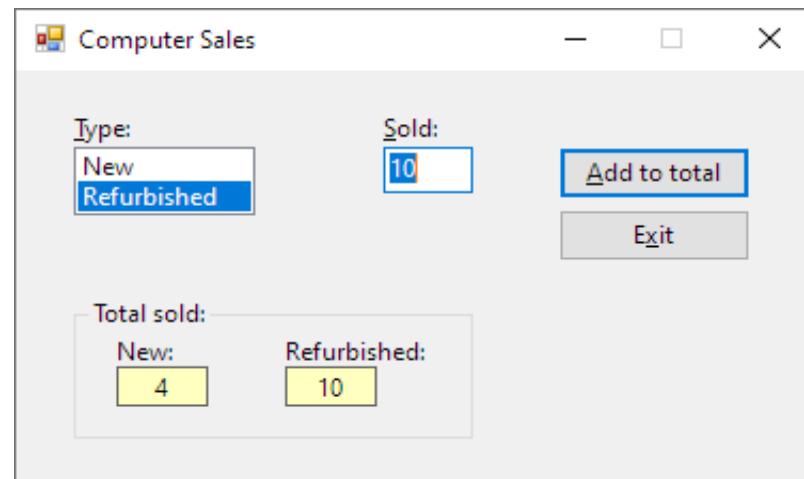
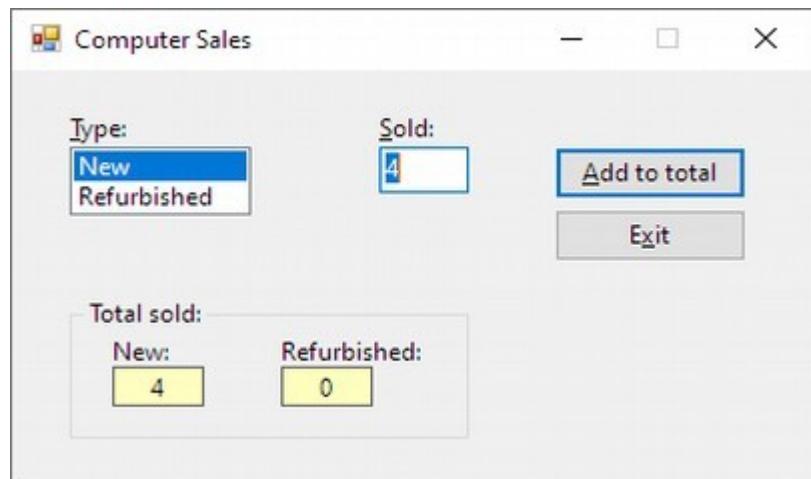
9. Open the Computer Solution.sln file contained in the VB2017\Chap08\Computer Solution folder. The interface allows the user to enter the number of either new or refurbished computers sold. Open the Code Editor window and locate the `btnAdd_Click` procedure. The procedure should use an array to accumulate the numbers sold by type. It also should display (in the labels) the total number sold for each type. Before the procedure ends, it should send the focus to the `txtSold` control and also select the control's existing text. Code the procedure. Save the solution and then start and test the application.

```
1  Option Explicit On
2  Option Strict On
3  Option Infer Off
4  ' choose type in (lstTypes) and enter the number in (txtSold) of sold computers
5  ' btnAdd_Click should use an array to accumulate the numbers sold by type & display totals in labels: lblNew & lblRefurbished
6  ' before the procedure ends, send the focus to the txtSold & select existing text
7
8  Public Class frmMain
9      Private Sub btnAdd_Click(sender As Object, e As EventArgs) Handles btnAdd.Click
10         ' Adds the amount sold to the appropriate total.
11
12         Static intTotal(1, 0) As Integer
13         Dim intSold As Integer
14         Integer.TryParse(txtSold.Text, intSold)
15
16         If lstTypes.SelectedIndex = 0 Then ' lblNew
17             intTotal(0, 0) += intSold
18         Else ' lblRefurbished
19             intTotal(1, 0) += intSold
20         End If
21
22         lblNew.Text = intTotal(0, 0).ToString
23         lblRefurbished.Text = intTotal(1, 0).ToString
24
25         txtSold.Focus()
26         txtSold.SelectAll()
27     End Sub
28
29     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
30         Me.Close()
31     End Sub
```

```

32
33     Private Sub txtSold_Enter(sender As Object, e As EventArgs) Handles txtSold.Enter
34         txtSold.SelectAll()
35     End Sub
36
37     Private Sub txtSold_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtSold.KeyPress
38         ' Accept only numbers and the Backspace key.
39
40         If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
41             e.Handled = True
42         End If
43     End Sub
44
45     'Private Sub ClearLabels(sender As Object, e As EventArgs) Handles txtSold.TextChanged, lstTypes.SelectedIndexChanged
46     '    lblNew.Text = String.Empty
47     '    lblRefurbished.Text = String.Empty
48     'End Sub
49     Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
50         ' Fill the list box and select the first item.
51
52         lstTypes.Items.Add("New")
53         lstTypes.Items.Add("Refurbished")
54         lstTypes.SelectedIndex = 0
55     End Sub
56
57 End Class

```



10. In this exercise, you code the Professor Schneider application, which displays a grade based on the number of points entered by the user. The number of points should always be less than or equal to 500. The grading scale is shown in Figure 8-48. Open the Schneider Solution.sln file contained in the VB2017\Chap08\Schneider Solution folder. Store the minimum points and grades in two parallel one-dimensional arrays named `intMins` and `strGrades`. The `btnDisplay_Click` procedure should use the number of points entered by the user to search the `intMins` array and then display the corresponding grade from the `strGrades` array. If the user enters a number that is greater than 500, the procedure should display an appropriate message and then display N/A as the grade. Code the application. Save the solution and then start and test the application.

| Minimum points | Maximum points | Grade |
|----------------|----------------|-------|
| 0              | 299            | F     |
| 300            | 349            | D     |
| 350            | 414            | C     |
| 415            | 464            | B     |
| 465            | 500            | A     |

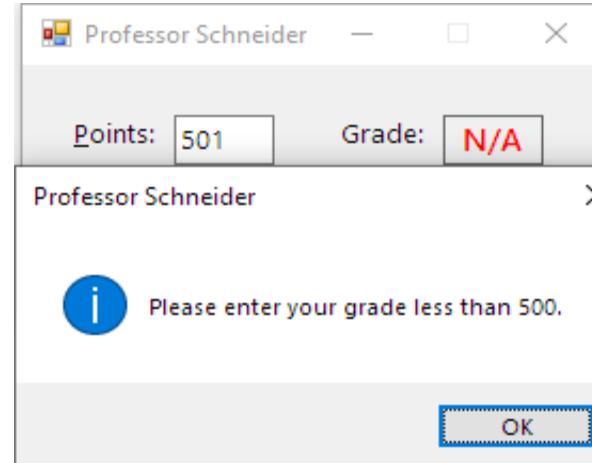
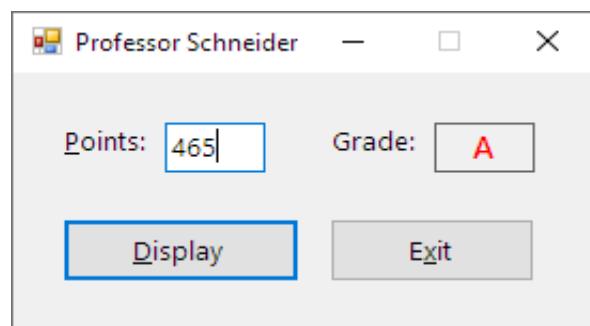
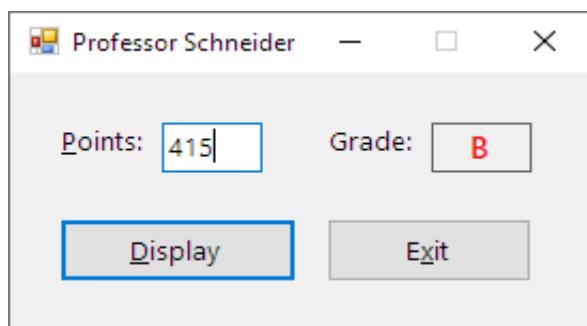
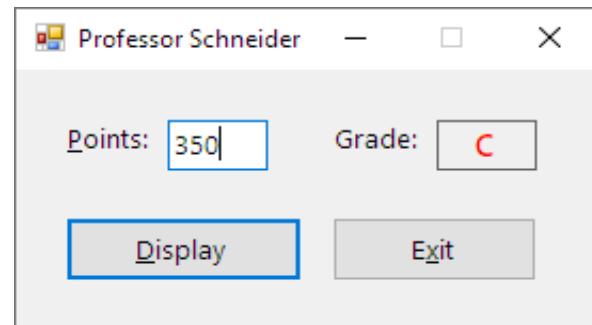
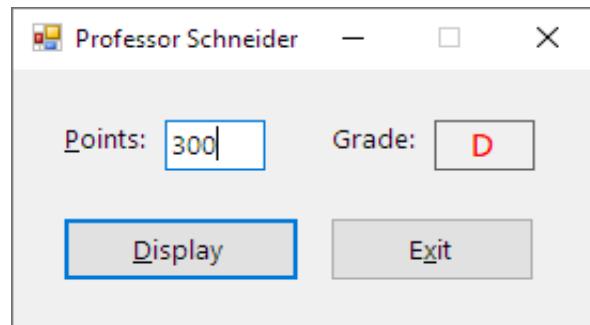
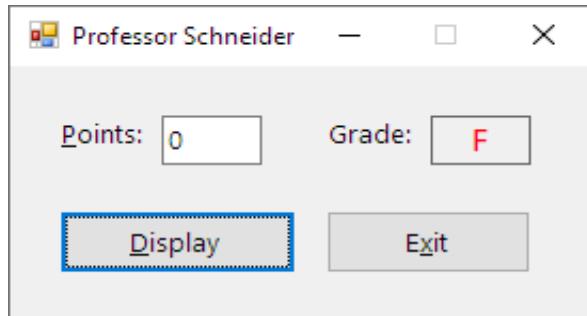
Figure 8-48 Grading scale for Exercise 10

```

1  Option Explicit On
2  Option Strict On
3  Option Infer Off
4  ' app will display a grade based on number of points entered by user:
5  ' user input points in (txtPoints), app will display a grade ABCDF on (lblGrade)
6  ' store the minimum points & grades in 2x Parallel 1D Arrays named: intMins & strGrades
7  ' F=0-299; D=300-349; C=350-414; B=415-464, A=465-500
8  ' btnDisplay_Click should use the number of points to search the intMins() and display corresponding grade from strGrades()
9  ' if more than 500: display the appropriate msg & N/A in (lblGrade)
10 Public Class frmMain
11     Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
12         'Dim intMins() As Integer = {0, 300, 350, 415, 465, 500}
13         'Dim strGrades() As String = {"F", "D", "C", "B", "A", "N/A"}
14         Dim intPoints As Integer
15         Integer.TryParse(txtPoints.Text, intPoints)
16
17         ' like this I can write for all of them, BUT CAN I MAKE IT LOOP??????
18         'If intPoints > intMins(5) Then
19             'MessageBox.Show("Please enter your grade less than 500.", "Professor Schneider", MessageBoxButtons.OK, MessageBoxIcon.Information)
20             'lblGrade.Text = "N/A"
21             'ElseIf intPoints >= intMins(0) AndAlso intPoints < intMins(1) Then
22                 'lblGrade.Text = strGrades(0)
23             'ElseIf intPoints >= intMins(1) AndAlso intPoints < intMins(2) Then
24                 'lblGrade.Text = strGrades(1)
25             'ElseIf intPoints >= intMins(2) AndAlso intPoints < intMins(3) Then
26                 'lblGrade.Text = strGrades(2)

```

```
27     'ElseIf intPoints >= intMins(3) AndAlso intPoints < intMins(4) Then
28     'lblGrade.Text = strGrades(3)
29     'ElseIf intPoints >= intMins(4) AndAlso intPoints <= intMins(5) Then
30     'lblGrade.Text = strGrades(4)
31     'End If
32
33     Dim intMins2() As Integer = {0, 300, 350, 415, 465}
34     Dim strGrades2() As String = {"F", "D", "C", "B", "A"}
35
36     If intPoints > 500 Then
37         lblGrade.Text = "N/A"
38         MessageBox.Show("Please enter your grade less than 500.", "Professor Schneider", MessageBoxButtons.OK, MessageBoxIcon.Information)
39     Else
40         For intIndex As Integer = 0 To intMins2.GetUpperBound(0)
41             If intPoints >= intMins2(intIndex) Then
42                 lblGrade.Text = strGrades2(intIndex)
43             End If
44             Next intIndex
45         End If
46     End Sub
47
48     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
49         Me.Close()
50     End Sub
51
52     Private Sub txtPoints_Enter(sender As Object, e As EventArgs) Handles txtPoints.Enter
53         txtPoints.SelectAll()
54     End Sub
55
56     Private Sub txtPoints_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtPoints.KeyPress
57         ' Accept only numbers and the Backspace key.
58
59         If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
60             e.Handled = True
61         End If
62     End Sub
63
64     Private Sub txtPoints_TextChanged(sender As Object, e As EventArgs) Handles txtPoints.TextChanged
65         lblGrade.Text = String.Empty
66     End Sub
67
68 End Class
```



## Chap08\Exercise1

### 28.Schneider Solution-TwoDim\_EXERCISE 11\_advanced

- 2D array, `MessageBox.Show`, `txt_KeyPress`, `txt_TextChanged`

11. In this exercise, you modify the Professor Schneider application from Exercise 10. Use Windows to make a copy of the Schneider Solution folder. Rename the copy Schneider Solution-TwoDim. Open the Schneider Solution.sln file contained in the Schneider Solution-TwoDim folder. Change the two parallel arrays to a two-dimensional array named `strGradeInfo`, and then make the appropriate modifications to the `btnDisplay_Click` procedure's code. Save the solution and then start and test the application.

## Chap08\Exercise1

### 27.Schneider Solution\_EXERCISE 10\_advanced

- 2x parallel 1D array, `GetUpperBound(0)`, `MessageBox.Show`, `txt_KeyPress`, `txt_TextChanged`

10. In this exercise, you code the Professor Schneider application, which displays a grade based on the number of points entered by the user. The number of points should always be less than or equal to 500. The grading scale is shown in Figure 8-48. Open the Schneider Solution.sln file contained in the VB2017\Chap08\Schneider Solution folder. Store the minimum points and grades in two parallel one-dimensional arrays named `intMins` and `strGrades`. The `btnDisplay_Click` procedure should use the number of points entered by the user to search the `intMins` array and then display the corresponding grade from the `strGrades` array. If the user enters a number that is greater than 500, the procedure should display an appropriate message and then display N/A as the grade. Code the application. Save the solution and then start and test the application.

| Minimum points | Maximum points | Grade |
|----------------|----------------|-------|
| 0              | 299            | F     |
| 300            | 349            | D     |
| 350            | 414            | C     |
| 415            | 464            | B     |
| 465            | 500            | A     |

Figure 8-48 Grading scale for Exercise 10

```
1 Option Explicit On
2 Option Strict On
3 Option Infer Off
4 ' app will display a grade based on number of points entered by user:
5 ' user input points in (txtPoints), app will display a grade ABCDF on (lblGrade)
6 ' *store the minimum points & grades in 2x Parallel 1D Arrays named: intMins & strGrades
7 'MOD: change 2x Parallel 1D Array to a 2D Array named strGradeInfo
8 ' F=0-299; D=300-349; C=350-414; B=415-464, A=465-500
9 ' *btnDisplay_Click should use the number of points to search the intMins() and display corresponding grade from strGrades()
10 'MOD: modify the btnDisplay_Click
11 ' if more than 500: display the appropriate msg & N/A in (lblGrade)
12 Public Class frmMain
13     Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
14         txtPoints.SelectAll()
15
16         Dim intPoints As Integer
17         Integer.TryParse(txtPoints.Text, intPoints)
```

```

18
19     Dim strGradeInfo(,) As String = {{"0", "300", "350", "415", "465"}, {"F", "D", "C", "B", "A"}}
20     Dim intGradeInfo As Integer
21
22     If intPoints > 500 Then
23         lblGrade.Text = "N/A"
24         MessageBox.Show("Please enter the valid points between 0 and 500", "Professor Schneider",
25                         MessageBoxButtons.OK, MessageBoxIcon.Information)
26     Else
27         For intIndex As Integer = 0 To 4
28             Integer.TryParse(strGradeInfo(0, intIndex), intGradeInfo)
29
30             If intPoints >= intGradeInfo Then
31                 lblGrade.Text = strGradeInfo(1, intIndex)
32             End If
33         Next intIndex
34     End If
35 End Sub
36
37 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
38     Me.Close()
39 End Sub
40
41 Private Sub txtPoints_Enter(sender As Object, e As EventArgs) Handles txtPoints.Enter
42     txtPoints.SelectAll()
43 End Sub
44
45 Private Sub txtPoints_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtPoints.KeyPress
46     ' Accept only numbers and the Backspace key.
47
48     If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
49         e.Handled = True
50     End If
51 End Sub
52
53 Private Sub txtPoints_TextChanged(sender As Object, e As EventArgs) Handles txtPoints.TextChanged
54     lblGrade.Text = String.Empty
55 End Sub
56
57 End Class

```

12. Open the Kraston Solution.sln file contained in the VB2017\Chap08\Kraston Solution folder. The btnDisplay\_Click procedure should display a shipping charge that is based on the number of items a customer orders. The order amounts and shipping charges are listed in Figure 8-49. Store the minimum order amounts and shipping charges in a class-level two-dimensional array. Display the appropriate shipping charge with a dollar sign and two decimal places. Code the btnDisplay\_Click procedure. Save the solution and then start and test the application.

| Minimum order | Maximum order | Shipping charge |
|---------------|---------------|-----------------|
| 0             | 0             | N/A             |
| 1             | 5             | 10.99           |
| 6             | 10            | 7.99            |
| 11            | 20            | 3.99            |
| 21            | No maximum    | 0               |

Figure 8-49 Order amounts and shipping charges for Exercise 12

```

1  Option Explicit On
2  Option Strict On
3  Option Infer Off
4  ' Display a shipping charge based on the number of Items a customer orders (txtOrdered):
5  ' store the Minimum order amounts & Shipping charges in 2D Class-level Array
6  ' display the appropriate shipping charge with a $ and 2 decimal places in (lblShipping)
7  '0-0= N/A; 1-5= 10.99; 6-10= 7.99; 11-20= 3.99; 21-no maximum= 0.
8
9  Public Class frmMain
10     Private dblArray As Double(,) = {{1, 6, 11, 21}, {10.99, 7.99, 3.99, 0}}
11
12     Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
13         Dim intOrdered As Integer
14         Integer.TryParse(txtOrdered.Text, intOrdered)
15
16         If intOrdered = 0 Then
17             lblShipping.Text = "N/A"
18         Else
19             For intIndex As Integer = 0 To dblArray.GetUpperBound(1)
20                 'MessageBox.Show(intIndex.ToString) ' shows: 0-1-2-3, because 4 columns, therefore correct
21                 ' if you use GetUpperBound(0), then it shows: 0-1, because 2 rows, therefore wrong
22                 If intOrdered >= dblArray(0, intIndex) Then
23                     lblShipping.Text = dblArray(1, intIndex).ToString("C2")
24                 End If
25
26             Next intIndex
27         End If
28     End Sub
29

```

```

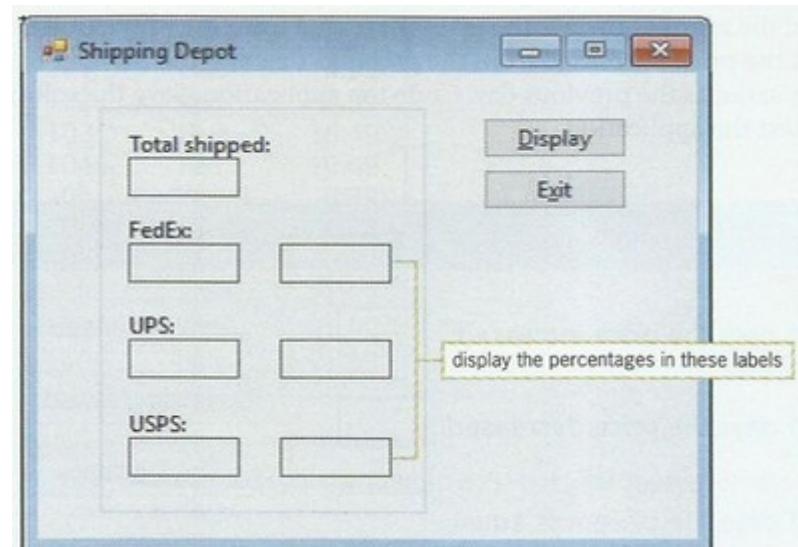
30      Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
31          Me.Close()
32      End Sub
33
34      Private Sub txtordered_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtOrdered.KeyPress
35          ' Accept numbers and the Backspace key.
36
37          If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
38              e.Handled = True
39          End If
40      End Sub
41
42      Private Sub txtordered_TextChanged(sender As Object, e As EventArgs) Handles txtOrdered.TextChanged
43          lblShipping.Text = String.Empty
44      End Sub
45
46      Private Sub txtOrdered_Enter(sender As Object, e As EventArgs) Handles txtOrdered.Enter
47          txtOrdered.SelectAll()
48      End Sub
49
50  End Class

```

|                                 |                                      |
|---------------------------------|--------------------------------------|
| <input type="text" value="0"/>  | <input type="text" value="N/A"/>     |
| <input type="text" value="1"/>  | <input type="text" value="\$10.99"/> |
| <input type="text" value="5"/>  | <input type="text" value="\$10.99"/> |
| <input type="text" value="0"/>  | <input type="text" value="\$7.99"/>  |
| <input type="text" value="10"/> | <input type="text" value="\$7.99"/>  |
| <input type="text" value="11"/> | <input type="text" value="\$3.99"/>  |
| <input type="text" value="20"/> | <input type="text" value="\$3.99"/>  |
| <input type="text" value="21"/> | <input type="text" value="\$0.00"/>  |

13. Create a Windows Forms application. Use the following names for the project and solution, respectively: Shipping Depot Project and Shipping Depot Solution. Save the application in the VB2017\Chap08 folder. The Shipping Depot store ships packages by FedEx, UPS, and USPS. Create the interface shown in Figure 8-50; the interface contains a group box, 11 labels, and two buttons. The Display button's Click event procedure should declare a two-dimensional array that contains four rows (one for each week) and three columns (one for each shipper). Initialize the array using the data shown in Figure 8-50. The procedure should display the total number of packages shipped, the total shipped by FedEx, the total shipped by UPS, and the total shipped by USPS. It should also display the percentage of the total number shipped by each of the different shippers. Display the percentages with a percent sign and no decimal places. Code the application. Save the solution and then start and test the application.

| Week | FedEx | UPS | USPS |
|------|-------|-----|------|
| 1    | 225   | 216 | 150  |
| 2    | 199   | 225 | 215  |
| 3    | 230   | 200 | 225  |
| 4    | 150   | 175 | 200  |



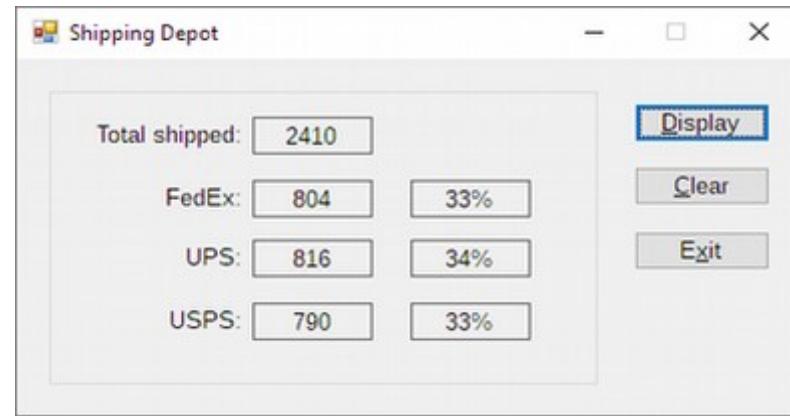
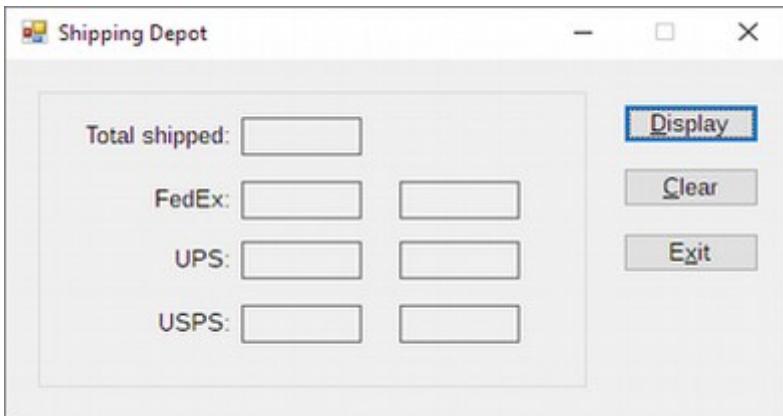
```

1 Option Explicit On
2 Option Strict On
3 Option Infer Off
4 ' The Shipping Depot store ships packages by FedEx, UPS and USPS
5 '--> create a GUI shown in a picture (groupbox, 11 labels, 2 buttons(I made 3)
6 '--> btnDisplay_Click event procedure should:
7   '--> declare 2D Array that contains 4 rows (one for each week) & 3 columns (one for each shipper)
8   '--> initialize the Array using the data shown in Figure 8-50
9   '--> display the total: --> number of packages shipped ... lbl0Total ... int0Total
10   '--> shipped by FedEx ... lbl1FedEx ... int1FedEx
11   '--> shipped by UPS ... lbl2UPS ... int2UPS
12   '--> shipped by USPS ... lbl3USPS ... int3USPS
13   '--> display % of total number with a % sign and no decimal places: --> shipped by FedEx ... lbl4FedExPerc ...
14   '--> shipped by UPS ... lbl5UPSPerc ...
15   '--> shipped by USPS ... lbl6USPSPerc ...
16 Public Class frmMain
17     Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
18       Dim int2DInfo() As Integer = {{225, 216, 150}, {199, 225, 215}, {230, 200, 225}, {150, 175, 200}} ' 4 weeks, 3 shippers
19       ' ttl= 2410, ttl FedEx= 804(33.36%), ttl UPS= 816(33.86%); ttl USPS= 790(32.78%) <- ok

```

```
20
21      Dim int0Total As Integer
22      Dim int1FedEx As Integer
23      Dim int2UPS As Integer
24      Dim int3USPS As Integer
25      Dim dbl4FedExPerc As Double
26      Dim dbl5UPS As Double
27      Dim dbl6USPS As Double
28
29      ' int0Total, lbl0Total:
30      For Each intElements As Integer In int2Dinfo
31          int0Total += intElements
32          lbl0Total.Text = int0Total.ToString ' = 2410 ok
33      Next intElements
34
35      ' int1-3, lbl1-3:
36      For intRow As Integer = 0 To int2Dinfo.GetUpperBound(0) ' 0-1-2-3 ok
37          int1FedEx += int2Dinfo(intRow, 0)
38          int2UPS += int2Dinfo(intRow, 1)
39          int3USPS += int2Dinfo(intRow, 2)
40      Next intRow
41      lbl1FedEx.Text = int1FedEx.ToString
42      lbl2UPS.Text = int2UPS.ToString
43      lbl3USPS.Text = int3USPS.ToString
44
45      ' %:
46      dbl4FedExPerc = int1FedEx / int0Total
47      lbl4FedExPerc.Text = dbl4FedExPerc.ToString("P0") ' 33% ok
48
49      dbl5UPS = int2UPS / int0Total
50      lbl5UPSPerc.Text = dbl5UPS.ToString("P0") ' 34% ok
51
52      dbl6USPS = int3USPS / int0Total
53      lbl6USPSPerc.Text = dbl6USPS.ToString("P0") '33% ok
54
55  End Sub
56
57  Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
58      Me.Close()
59  End Sub
```

```
61      Private Sub btnClear_Click(sender As Object, e As EventArgs) Handles btnClear.Click
62          lbl0Total.Text = Nothing
63          lbl1FedEx.Text = Nothing
64          lbl2UPS.Text = Nothing
65          lbl3USPS.Text = Nothing
66          lbl4FedExPerc.Text = Nothing
67          lbl5UPSPerc.Text = Nothing
68          lbl6USPSPerc.Text = Nothing
69      End Sub
70  End Class
```



14. In this exercise, you modify the Shipping Depot application from Exercise 13. Use Windows to make a copy of the Shipping Depot Solution folder. Rename the copy Modified Shipping Depot Solution. Open the Shipping Depot Solution.sln file contained in the Modified Shipping Depot Solution folder. Open the Code Editor window and locate the Display button's Click event procedure. Change the four-row, three-column array to a three-row (one for each shipper), four-column (one for each week) array. Make the necessary modifications to the procedure's code.

### 30.Shipping Depot Solution\_EXERCISE 13\_advanced

13. Create a Windows Forms application. Use the following names for the project and solution, respectively: Shipping Depot Project and Shipping Depot Solution. Save the application in the VB2017\Chap08 folder. The Shipping Depot store ships packages by FedEx, UPS, and USPS. Create the interface shown in Figure 8-50; the interface contains a group box, 11 labels, and two buttons. The Display button's Click event procedure should declare a two-dimensional array that contains four rows (one for each week) and three columns (one for each shipper). Initialize the array using the data shown in Figure 8-50. The procedure should display the total number of packages shipped, the total shipped by FedEx, the total shipped by UPS, and the total shipped by USPS. It should also display the percentage of the total number shipped by each of the different shippers. Display the percentages with a percent sign and no decimal places. Code the application. Save the solution and then start and test the application.

| Week | FedEx | UPS | USPS |
|------|-------|-----|------|
| 1    | 225   | 216 | 150  |
| 2    | 199   | 225 | 215  |
| 3    | 230   | 200 | 225  |
| 4    | 150   | 175 | 200  |

```

1 Option Explicit On
2 Option Strict On
3 Option Infer Off
4 ' The Shipping Depot store ships packages by FedEx, UPS and USPS
5 '--> create a GUI shown in a picture (groupbox, 11 labels, 2 buttons(I made 3)
6 '--> btnDisplay_Click event procedure should:
7   '--> declare 2D Array that contains 4 rows (one for each week) & 3 columns (one for each shipper)
8   '--> initialize the Array using the data shown in Figure 8-50
9   '--> display the total: --> number of packages shipped ... lbl0Total ... int0Total
10   '--> shipped by FedEx ... lbl1FedEx ... int1FedEx
11   '--> shipped by UPS ... lbl2UPS ... int2UPS
12   '--> shipped by USPS ... lbl3USPS ... int3USPS
13   '--> display % of total number with a % sign and no decimal places: --> shipped by FedEx ...lbl4FedExPerc ...
14   '--> shipped by UPS ...lbl5UPSPerc ...
15   '--> shipped by USPS ...lbl6USPSPerc ...
16 'MOD: locate the btnDisplay_Click event procedure and:
17   '--> change 2D Array from 4x3 into 3x4 (each row for each shipper & each column for each week)
18   '--> make the necessary modifications to the procedure's code

```

```

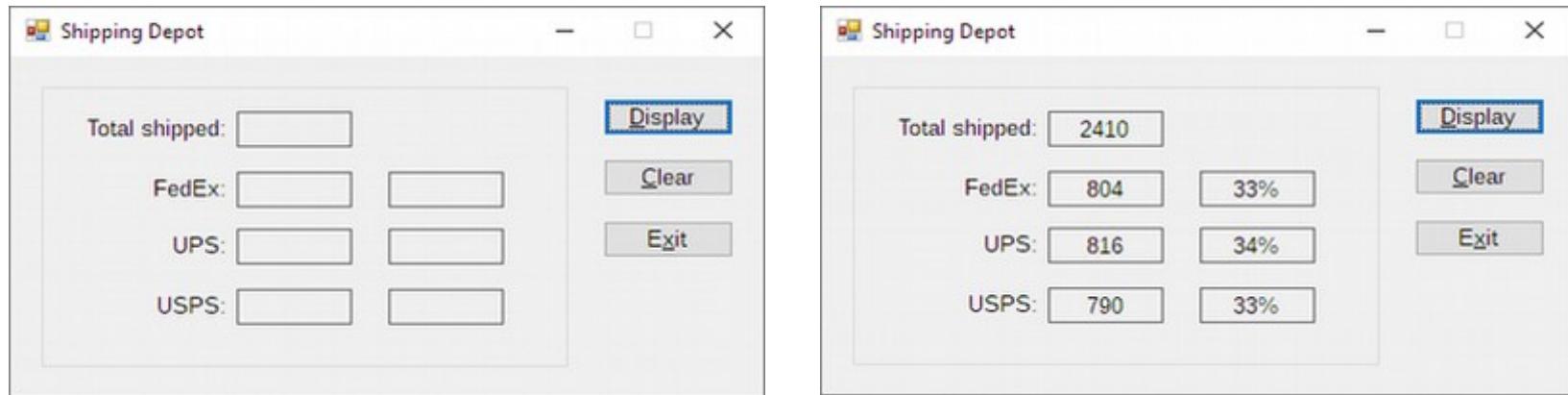
19  Public Class frmMain
20      Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
21          Dim int2DinfoMod() As Integer = {{225, 199, 230, 150}, {216, 225, 200, 175}, {150, 215, 225, 200}} ' 3 shippers, 4 weeks
22          ' ttl= 2410, ttl FedEx= 804(33.36%), ttl UPS= 816(33.86%); ttl USPS= 790(32.78%) <- ok
23
24          Dim int0Total As Integer
25          Dim int1FedEx As Integer
26          Dim int2UPS As Integer
27          Dim int3USPS As Integer
28          Dim dbl4FedExPerc As Double
29          Dim dbl5UPS As Double
30          Dim dbl6USPS As Double
31
32          'MOD: int0Total, lbl0Total:
33          For Each intElements As Integer In int2DinfoMod
34              int0Total += intElements
35              lbl0Total.Text = int0Total.ToString ' = 2410 ok
36          Next intElements
37
38          'MOD: int1-3, lbl1-3:
39          For intCol As Integer = 0 To int2DinfoMod.GetUpperBound(1) ' 0-1-2-3 ok
40              int1FedEx += int2DinfoMod(0, intCol)
41              int2UPS += int2DinfoMod(1, intCol)
42              int3USPS += int2DinfoMod(2, intCol)
43          Next intCol
44
45          lbl1FedEx.Text = int1FedEx.ToString
46          lbl2UPS.Text = int2UPS.ToString
47          lbl3USPS.Text = int3USPS.ToString
48
49          ' %:
50          dbl4FedExPerc = int1FedEx / int0Total
51          lbl4FedExPerc.Text = dbl4FedExPerc.ToString("P0") ' 33% ok
52
53          dbl5UPS = int2UPS / int0Total
54          lbl5UPSPerc.Text = dbl5UPS.ToString("P0") ' 34% ok
55
56          dbl6USPS = int3USPS / int0Total
57          lbl6USPSPerc.Text = dbl6USPS.ToString("P0") ' 33% ok
58
59      End Sub
60

```

```

61      Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
62          Me.Close()
63      End Sub
64
65      Private Sub btnClear_Click(sender As Object, e As EventArgs) Handles btnClear.Click
66          lbl0Total.Text = Nothing
67          lbl1FedEx.Text = Nothing
68          lbl2UPS.Text = Nothing
69          lbl3USPS.Text = Nothing
70          lbl4FedExPerc.Text = Nothing
71          lbl5UPSPerc.Text = Nothing
72          lbl6USPSPerc.Text = Nothing
73      End Sub
74  End Class

```



## Chap08\Exercise1

### 32.Lottery Solution\_EXERCISE 15\_advanced

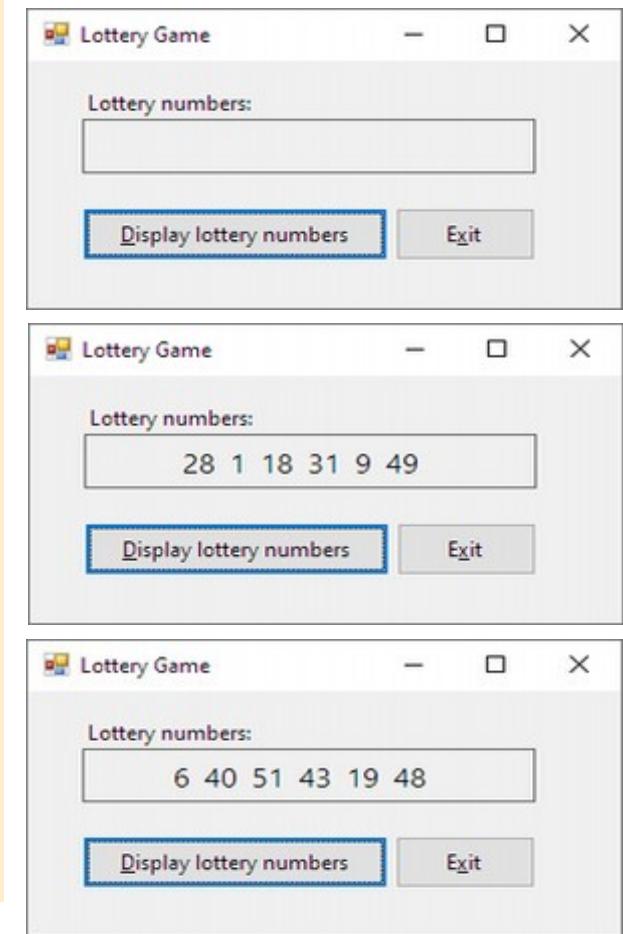
- 1D accumulator array, random integer: `intRandGen As New Random`, method `intRandGen.Next()`

15. In this exercise, you code an application that generates and displays six unique random numbers for a lottery game. Each lottery number can range from 1 through 54 only. Open the Lottery Solution.sln file contained in the VB2017\Chap08\Lottery Solution folder. The `btnDisplay_Click` procedure should display six unique random numbers in the interface. Code the application. Save the solution and then start and test the application.

```

1 Option Explicit On
2 Option Strict On
3 Option Infer Off
4 ' you code an app that generates & displays 6 UNIQUE RANDOM numbers for a lottery game
5 ' each lottery number can range: 1-54 only
6 ' btnDisplay_Click procedure should display in GUI
7
8 ' lblLottery
9 'CH7_A3 info about Random, 31_15 Exercise
10
11 Public Class frmMain
12
13     'Private intAccumulator(5) As Integer
14
15     Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
16         ' Generates and displays six unique random numbers from 1 through 54.
17
18         lblLottery.Text = Nothing
19
20         Dim intAccumulator(5) As Integer
21         Dim intRandGen As New Random
22         Dim intRandomNumber As Integer
23
24         For intCounter As Integer = 0 To 5
25             intRandomNumber = intRandGen.Next(1, 55)
26             For intCheck As Integer = 0 To intCounter
27                 If intRandomNumber <> intAccumulator(intCheck) Then
28                     intAccumulator(intCounter) = intRandomNumber
29                 Else
30                     intRandomNumber = intRandGen.Next(1, 55)
31                 End If
32             Next intCheck
33             lblLottery.Text &= intAccumulator(intCounter).ToString & " "
34         Next intCounter
35
36     End Sub
37
38     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
39         Me.Close()
40     End Sub
41
42 End Class

```



## Chap08\Exercise

### 33.Gas Prices Solution\_EXERCISE 16\_advanced\_NOT FINISHED

- does not make any sense to me, therefore not finished

Create a Windows Forms application. Use the following names for the project and solution, respectively: Gas Prices Project and Gas Prices Solution. Save the application in the VB2017\Chap08 folder. Create the interface shown in Figure 8-51. The application should declare a class-level Double array that contains 30 elements. Each element will store the daily price of a gallon of gas. Initialize the first 10 elements using the following values: 2.25, 2.25, 2.24, 2.15, 2.05, 1.97, 2.25, 2.87, 2.5, and 2.4. Use your own values to initialize the remaining 20 elements. The application should display the following items: the number of days the price increased from the previous day, the number of days the price decreased from the previous day, and the number of days the price stayed the same as the previous day. Code the application. Save the solution and then start and test the application.

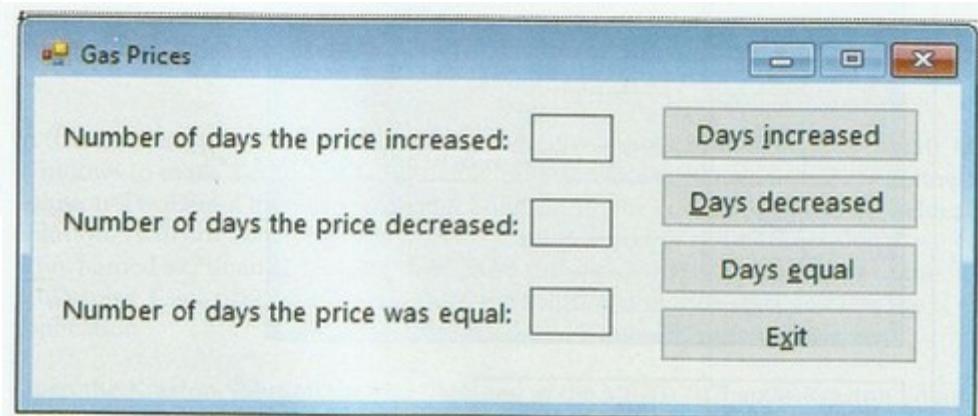
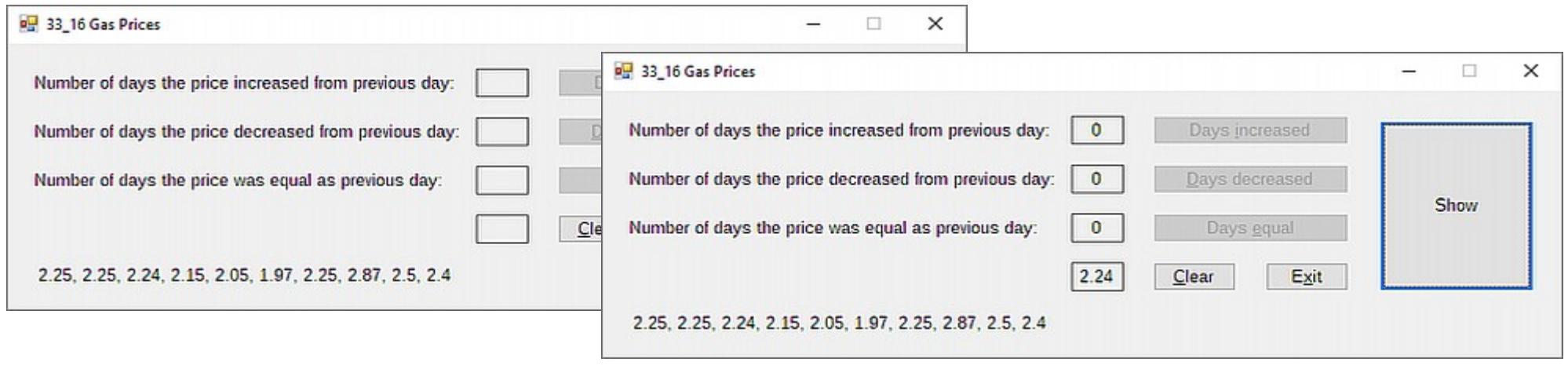


Figure 8-51 Interface for Exercise 16

```
1  Option Explicit On
2  Option Strict On
3  Option Infer Off
4  ' the Application should display the following values:
5  ' -> number of days the price increased from the previous day,
6  ' -> number of days the price decreased from the previous day,
7  ' -> number of days the price stayed the same as the previous day
8  ' -> declare a Class-level Double 1D Array that contains 30 elements
9  ' each element will store the daily price of a galon of gas
10 ' -> initialize: - first 10 elements using the following values: 2.25, 2.25, 2.24, 2.15, 2.05, 1.97, 2.25, 2.87, 2.5, 2.4
11 '           - remaining 20 elements using your own values
12 Public Class frmMain
13
14     Private dblPrices() As Double = {2.25, 2.25, 2.24, 2.15, 2.05, 1.97, 2.25, 2.87, 2.5, 2.4}
15     Private dblActualPrice() As Double
16     Private intCounterIndex As Integer = 0
17     Private dbl4Counter As Double = dblPrices(0)           ' just to show the Actual value flushed from the dblPrices() Array
18
19     Private int1Increased As Integer = 0
20     Private int2Decreased As Integer = 0
21     Private int3Equal As Integer = 0
22
```

```
23     Private Sub btn6Show_Click(sender As Object, e As EventArgs) Handles btn6Show.Click
24
25         If intCounterIndex > dblPrices.GetUpperBound(0) Then
26             intCounterIndex = 0                                ' loop when reached the end of an Array
27             int1Increased = 0
28             int2Decreased = 0
29             int3Equal = 0
30         Else
31
32             Select Case True
33                 Case dbl4Counter > dblPrices(intCounterIndex)
34                     For Each dblElement As Double In dblPrices
35                         If dblElement > dbl4Counter Then
36                             int1Increased += 1
37                         End If
38                         Next dblElement
39                     End Select
40
41             intCounterIndex += 1
42             dbl4Counter = dblPrices(intCounterIndex)
43         End If
44
45         lbl1Increased.Text = int1Increased.ToString
46         lbl2Decreased.Text = int2Decreased.ToString
47         lbl3Equal.Text = int3Equal.ToString
48         lbl4Counter.Text = dbl4Counter.ToString      ' just to show the Actual value flushed from the dblPrices() Array
49     End Sub
50
51     Private Sub btn4Clear_Click(sender As Object, e As EventArgs) Handles btn4Clear.Click
52         lbl1Increased.Text = Nothing
53         lbl2Decreased.Text = Nothing
54         lbl3Equal.Text = Nothing
55         lbl4Counter.Text = Nothing
56     End Sub
57
58     Private Sub btn5Exit_Click(sender As Object, e As EventArgs) Handles btn5Exit.Click
59         Me.Close()
60     End Sub
61
62 End Class
```



## Chap08\Exercisel

### 34.Organic Solution EXERCISE 17 advanced

- 2D array 6x3, 1D array, GetUpperBound(0), For Each...Next

17. The sales manager at Organic Market wants you to create an application that displays the total sales made in each of three regions: the U.S., Canada, and Mexico. The application should also display the total company sales as well as the percentage that each region contributed to the total sales. Display the sales amounts with a dollar sign and no decimal places. Display the percentages with a percent sign and no decimal places. The sales amounts for six months are shown in Figure 8-52. Create a Windows Forms application. Use the following names for the project and solution, respectively: Organic Project and Organic Solution. Save the application in the VB2017\Chap08 folder. Create a suitable interface and then code the application. Store the sales amounts in a two-dimensional array. Save the solution and then start and test the application.

| Month | U.S. sales (\$) | Canada sales (\$) | Mexico sales (\$) |
|-------|-----------------|-------------------|-------------------|
| 1     | 120,000         | 90,000            | 65,000            |
| 2     | 190,000         | 85,000            | 64,000            |
| 3     | 175,000         | 80,000            | 71,000            |
| 4     | 188,000         | 83,000            | 67,000            |
| 5     | 125,000         | 87,000            | 65,000            |
| 6     | 163,000         | 80,000            | 64,000            |

Figure 8-52 Sales amounts for Exercise 17

```

1 Option Explicit On
2 Option Strict On
3 Option Infer Off
4 ' create an app that displays: - the total sales made in each of three regions: US, Canada, Mexico -> no decimal places, $
5 ' - total Company sales -> no decimal places, $
6 ' - percentage that each region contributed to the total sales -> no decimal places, %
7 ' store the sales amounts (Figure 8-52) in a 2D Array (6 months; US, Canada, Mexico)

```

```

8  Public Class frmMain
9      Private Sub btnShow_Click(sender As Object, e As EventArgs) Handles btnShow.Click
10
11         ' all of the amounts are originally *1000:
12         Dim dblSales(,) As Double = {{120, 90, 65}, {190, 85, 64}, {175, 80, 71}, {188, 83, 67}, {125, 87, 65}, {163, 80, 64}}
13         ' ttl US = 961,000; ttl Canada = 505,000; ttl Mexico = 396,000
14
15         Dim dblAmounts(8) As Double
16
17         ' ttls $:
18         For intRowIndexes As Integer = 0 To dblSales.GetUpperBound(0)
19             For intCounter As Integer = 0 To 2
20                 dblAmounts(intCounter + 1) += dblSales(intRowIndexes, intCounter)      ' same like: *
21             Next intCounter
22             'dblAmounts(1) += dblSales(intRowIndexes, 0) ' 1_ttl US = 961           ' same like: *
23             'dblAmounts(2) += dblSales(intRowIndexes, 1) ' 2_ttl Canada = 505        ' same like: *
24             'dblAmounts(3) += dblSales(intRowIndexes, 2) ' 3_ttl Mexico = 396       ' same like: *
25         Next intRowIndexes
26
27         ' 4_ttl Company: 1,862
28         For Each dblElement As Double In dblSales
29             dblAmounts(4) += dblElement
30         Next dblElement
31
32         For intCounter1 As Integer = 5 To 7
33             dblAmounts(intCounter1) = (dblAmounts(intCounter1 - 4) / dblAmounts(4))      ' same like: **
34             dblAmounts(8) += dblAmounts(intCounter1)                                     ' same like: ***
35         Next intCounter1
36
37         ' perc_US= 51.6%; perc_Canada= 27.1%; perc_Mexico = 21.3%
38         'dblAmounts(5) = (dblAmounts(1) / dblAmounts(4))                           ' same like: **
39         'dblAmounts(6) = (dblAmounts(2) / dblAmounts(4))                           ' same like: **
40         'dblAmounts(7) = (dblAmounts(3) / dblAmounts(4))                           ' same like: **
41
42         'dblAmounts(8) = dblAmounts(5) + dblAmounts(6) + dblAmounts(7)            ' same like: ***
43
44         lbl1Ttl_US.Text = (1000 * dblAmounts(1)).ToString("C0")
45         lbl2Ttl_Canada.Text = (1000 * dblAmounts(2)).ToString("C0")
46         lbl3Ttl_Mexico.Text = (1000 * dblAmounts(3)).ToString("C0")
47         lbl4Ttl_Company.Text = (1000 * dblAmounts(4)).ToString("C0")
48         lbl5Perc_US.Text = dblAmounts(5).ToString("P0")
49         lbl6Perc_Canada.Text = dblAmounts(6).ToString("P0")
50         lbl7Perc_Mexico.Text = dblAmounts(7).ToString("P0")
51         lbl8Perc_Company.Text = dblAmounts(8).ToString("P0")

```

```

52
53     End Sub
54
55     Private Sub btnClear_Click(sender As Object, e As EventArgs) Handles btnClear.Click
56         lbl1Ttl_US.Text = String.Empty
57         lbl2Ttl_Canada.Text = String.Empty
58         lbl3Ttl_Mexico.Text = String.Empty
59         lbl4Ttl_Company.Text = String.Empty
60         lbl5Perc_US.Text = String.Empty
61         lbl6Perc_Canada.Text = String.Empty
62         lbl7Perc_Mexico.Text = String.Empty
63         lbl8Perc_Company.Text = String.Empty
64     End Sub
65
66     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
67         Me.Close()
68     End Sub
69
70 End Class

```

The screenshot shows the application window titled "Organic Market". It contains four groups of text boxes for "U.S.", "Canada", "Mexico", and "Company sales", each with a "Total sales:" label and a "Percentage:" label next to it. Below these groups are three buttons: "Show", "Clear", and "Exit".

The screenshot shows the application window titled "Organic Market" after the "Show" button has been clicked. The "U.S." group now displays "Total sales: \$961,000" and "Percentage: 52%". The "Canada" group displays "Total sales: \$505,000" and "Percentage: 27%". The "Mexico" group displays "Total sales: \$396,000" and "Percentage: 21%". The "Company sales" group displays "Total sales: \$1,862,000" and "Percentage: 100%". The "Show" button is highlighted with a blue border.

| <u>month</u> | <u>US sales</u> | <u>Canada sales</u> | <u>Mexico sales</u> |
|--------------|-----------------|---------------------|---------------------|
| 1            | 120,000         | 90,000              | 65,000              |
| 2            | 190,000         | 85,000              | 64,000              |
| 3            | 175,000         | 80,000              | 71,000              |
| 4            | 188,000         | 83,000              | 67,000              |
| 5            | 125,000         | 87,000              | 65,000              |
| 6            | 163,000         | 80,000              | 64,000              |
|              | 961,000         | 505,000             | 396,000             |
|              |                 |                     | <b>1,862,000</b>    |
|              |                 |                     | <b>100.00%</b>      |

18. Create a Windows Forms application. Use the following names for the project and solution, respectively: Bindy Project and Bindy Solution. Save the application in the VB2017\Chap08 folder. Bindy Enterprises sells the 10 items listed in Figure 8-53. The figure also contains the major tasks that the application needs to perform. Create a suitable interface and then code the application. Save the solution and then start and test the application.

#### Tasks

- 1). When the user selects the ID from a list box, the application should display the ID's color and price.
- 2). When the user selects the color from a list box, the application should display the IDs and prices of all items available in that color.
- 3). When the user enters a price in a text box, the application should display the IDs, colors, and prices of items selling at or below that price.

| ID  | Color | Price |
|-----|-------|-------|
| 101 | Blue  | 4.99  |
| 102 | Red   | 4.99  |
| 103 | Blue  | 10.49 |
| 104 | Red   | 10.49 |
| 105 | White | 6.79  |
| 106 | Red   | 6.79  |
| 107 | Blue  | 6.79  |
| 108 | Black | 21.99 |
| 109 | White | 21.99 |
| 110 | Blue  | 21.99 |

```

1 Option Explicit On
2 Option Strict On
3 Option Infer Off
4
5 Public Class frmMain
6   Private dbl2DIndex() As Double = {{101, 102, 103, 104, 105, 106, 107, 108, 109, 110},
7                               {4.99, 4.99, 10.49, 10.49, 6.79, 6.79, 6.79, 21.99, 21.99, 21.99}}
8   Private str1DColors() As String = {"Blue", "Red", "Blue", "Red", "White", "Red", "Blue", "Black", "White", "Blue"}
9
10  Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
11    For intCounter As Integer = 0 To dbl2DIndex.GetUpperBound(1)      ' 10x
12      lst1_ID.Items.Add(dbl2DIndex(0, intCounter))
13    Next intCounter
14
15    lst2_Color.Items.Add("Black")      ' SelectedIndex = 0
16    lst2_Color.Items.Add("Blue")       ' SelectedIndex = 1
17    lst2_Color.Items.Add("Red")        ' SelectedIndex = 2
18    lst2_Color.Items.Add("White")      ' SelectedIndex = 3
19  End Sub
20
21  Private Sub btnShow_Click(sender As Object, e As EventArgs) Handles btnShow.Click
22    lst0_Show.Items.Clear()
  
```

```

23
24     Select Case True
25         Case rad1_ID.Checked  '= (rad1_ID) - display ID's color & price:_OK
26             For intIndex As Integer = 0 To 9
27                 If lst1_ID.SelectedIndex = intIndex Then
28                     'MessageBox.Show(intIndex.ToString)    ' shows an index of selected item_OK
29                     lst0_Show.Items.Insert(0, dbl2DIndex(0, intIndex) & ControlChars.Tab & str1DCOLORS(intIndex) & ControlChars.Tab &
30   dbl2DIndex(1, intIndex).ToString("C2"))
31                 End If
32             Next intIndex
33
34         Case rad2_Color.Checked  '= (rad2_Color) - display ID's & prices in the same color:_OK
35             ' Black = 1x: 108 = $21.99.
36             ' Blue = 4x: 101 = $4.99...103 = $10.49...107 = $6.79...110 = $21.99.
37             ' Red = 3x: 102 = $4.99...104 = $10.49...106 = $6.79.
38             ' White = 2x: 105 = $6.79...109 = $21.99.
39
40             For intCounter As Integer = 0 To dbl2DIndex.GetUpperBound(1)
41                 ' lst2: BETA_working OK
42                 If lst2_Color.SelectedItem Is str1DCOLORS(intCounter) Then
43                     lst0_Show.Sorted = True
44                     lst0_Show.Items.Insert(0, dbl2DIndex(0, intCounter) & ControlChars.Tab & str1DCOLORS(intCounter) & ControlChars.Tab &
45   dbl2DIndex(1, intCounter).ToString("C2"))
46                 End If
47             Next intCounter
48
49             ' lst2: ALFA_working, but not the best one (i will try it more easy in BETA version)
50             'ForintColorIndex As Integer = 0 To 3
51             'If lst2_Color.SelectedIndex = intColorIndex Then
52                 'lst0_Show.Items.Insert(0, lst2_Color.SelectedIndex)      ' shows Index of selected color from lst2_Color_OK
53             'If lst2_Color.SelectedIndex = 0 Then
54                 '0 = "Black" = str1DCOLORS index = 7
55                 lst0_Show.Items.Insert(0, dbl2DIndex(0, 7) & ControlChars.Tab & str1DCOLORS(7) & ControlChars.Tab & dbl2DIndex(1, 7).ToString("C2"))
56             'ElseIf lst2_Color.SelectedIndex = 1 Then
57                 '1 = "Blue" = str1DCOLORS index = 0, 2, 6, 9
58                 lst0_Show.Items.Insert(0, dbl2DIndex(0, 0) & ControlChars.Tab & str1DCOLORS(0) & ControlChars.Tab & dbl2DIndex(1, 0).ToString("C2"))
59                 lst0_Show.Items.Insert(1, dbl2DIndex(0, 2) & ControlChars.Tab & str1DCOLORS(2) & ControlChars.Tab & dbl2DIndex(1, 2).ToString("C2"))
60                 lst0_Show.Items.Insert(2, dbl2DIndex(0, 6) & ControlChars.Tab & str1DCOLORS(6) & ControlChars.Tab & dbl2DIndex(1, 6).ToString("C2"))
61                 lst0_Show.Items.Insert(3, dbl2DIndex(0, 9) & ControlChars.Tab & str1DCOLORS(9) & ControlChars.Tab & dbl2DIndex(1, 9).ToString("C2"))
62             'ElseIf lst2_Color.SelectedIndex = 2 Then
63                 '2 = "Red" = str1DCOLORS index = 1, 3, 5
64                 lst0_Show.Items.Insert(0, dbl2DIndex(0, 1) & ControlChars.Tab & str1DCOLORS(1) & ControlChars.Tab & dbl2DIndex(1, 1).ToString("C2"))
65                 lst0_Show.Items.Insert(1, dbl2DIndex(0, 3) & ControlChars.Tab & str1DCOLORS(3) & ControlChars.Tab & dbl2DIndex(1, 3).ToString("C2"))
66                 lst0_Show.Items.Insert(2, dbl2DIndex(0, 5) & ControlChars.Tab & str1DCOLORS(5) & ControlChars.Tab & dbl2DIndex(1, 5).ToString("C2"))

```

```

67         'ElseIf lst2_Color.SelectedIndex = 3 Then
68         ''3 = "White" = str1DCOLORS index = 4, 8
69         'lst0_Show.Items.Insert(0, dbl2DIndex(0, 4) & ControlChars.Tab & str1DCOLORS(4) & ControlChars.Tab & dbl2DIndex(1, 4).ToString("C2"))
70         'lst0_Show.Items.Insert(1, dbl2DIndex(0, 8) & ControlChars.Tab & str1DCOLORS(8) & ControlChars.Tab & dbl2DIndex(1, 8).ToString("C2"))
71         'End If
72         'End If
73         'Next intColorIndex
74
75     Case rad3_EnterThePrice.Checked    '_OK
76         Dim dblEnterThePrice As Double
77         Double.TryParse(txtEnterThePrice.Text, dblEnterThePrice)
78
79         For intCounter As Integer = 0 To dbl2DIndex.GetUpperBound(1)
80             If dblEnterThePrice >= dbl2DIndex(1, intCounter) Then
81                 lst0_Show.Sorted = True
82                 lst0_Show.Items.Insert(0, dbl2DIndex(0, intCounter) & ControlChars.Tab & str1DCOLORS(intCounter) & ControlChars.Tab &
83   dbl2DIndex(1, intCounter).ToString("C2"))
84             End If
85             Next intCounter
86             txtEnterThePrice.SelectAll()
87         End Select
88     End Sub
89
90     Private Sub txtEnterThePrice_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtEnterThePrice.KeyPress
91         If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> "." AndAlso e.KeyChar <> ControlChars.Back Then
92             e.Handled = True
93         End If
94     End Sub
95
96     Private Sub InputChanged(sender As Object, e As EventArgs) Handles lst1_ID.SelectedIndexChanged, lst2_Color.SelectedIndexChanged,
97   txtEnterThePrice.TextChanged
98         lst0_Show.Items.Clear()
99     End Sub
100
101    Private Sub rad1_ID_CheckedChanged(sender As Object, e As EventArgs) Handles rad1_ID.CheckedChanged
102        lst1_ID.Enabled = True
103        lst2_Color.Enabled = False
104        txtEnterThePrice.Enabled = False
105        lst2_Color.SelectedItem = Nothing
106        txtEnterThePrice.Text = Nothing
107        lst0_Show.Items.Clear()
108        lst1_ID.Focus()
109    End Sub
110

```

```

111  Private Sub rad2_Color_CheckedChanged(sender As Object, e As EventArgs) Handles rad2_Color.CheckedChanged
112      lst2_Color.Enabled = True
113      lst1_ID.Enabled = False
114      txtEnterThePrice.Enabled = False
115      lst1_ID.SelectedItem = Nothing
116      txtEnterThePrice.Text = Nothing
117      lst0_Show.Items.Clear()
118      lst2_Color.Focus()
119  End Sub
120
121  Private Sub rad3_EnterThePrice_CheckedChanged(sender As Object, e As EventArgs) Handles rad3_EnterThePrice.CheckedChanged
122      txtEnterThePrice.Enabled = True
123      lst1_ID.Enabled = False
124      lst2_Color.Enabled = False
125      lst1_ID.SelectedItem = Nothing
126      lst2_Color.SelectedItem = Nothing
127      lst0_Show.Items.Clear()
128      txtEnterThePrice.Focus()
129  End Sub
130
131  Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
132      Me.Close()
133  End Sub
134
135 End Class

```

The screenshot shows a Windows application with two forms. The top form has three radio buttons: 'Select ID', 'Select Color', and 'Enter price'. The 'Select ID' radio button is selected, and its list view shows items 101 through 110, with item 107 highlighted. The 'Select Color' radio button is also selected, and its list view shows 'Black', 'Blue', 'Red', and 'White', with 'Blue' highlighted. The 'Enter price' radio button is unselected. Both forms have 'Show' and 'Exit' buttons.

| ID  | Color | Price  |
|-----|-------|--------|
| 101 |       |        |
| 102 |       |        |
| 103 |       |        |
| 104 |       |        |
| 105 |       |        |
| 106 |       |        |
| 107 | Blue  | \$6.79 |
| 108 |       |        |
| 109 |       |        |
| 110 |       |        |

| ID  | Color | Price   |
|-----|-------|---------|
| 101 | Blue  | \$4.99  |
| 102 | Red   | \$4.99  |
| 103 | Blue  | \$10.49 |
| 104 | Red   | \$10.49 |
| 105 | White | \$6.79  |
| 106 | Red   | \$6.79  |
| 107 | Blue  | \$6.79  |
| 108 | Black | \$21.99 |
| 109 | White | \$21.99 |
| 110 | Blue  | \$21.99 |

**Bindy Enterprise**

Select ID:     Select Color:     Enter price:

|     |       |    |
|-----|-------|----|
| 101 | Black | 11 |
| 102 | Blue  |    |
| 103 | Red   |    |
| 104 | White |    |
| 105 |       |    |
| 106 |       |    |
| 107 |       |    |
| 108 |       |    |
| 109 |       |    |
| 110 |       |    |

**Your search:**

| ID: | Color: | Price:  |
|-----|--------|---------|
| 101 | Blue   | \$4.99  |
| 102 | Red    | \$4.99  |
| 103 | Blue   | \$10.49 |
| 104 | Red    | \$10.49 |
| 105 | White  | \$6.79  |
| 106 | Red    | \$6.79  |
| 107 | Blue   | \$6.79  |

**info:**

|     |       |         |
|-----|-------|---------|
| 101 | Blue  | \$4.99  |
| 102 | Red   | \$4.99  |
| 103 | Blue  | \$10.49 |
| 104 | Red   | \$10.49 |
| 105 | White | \$6.79  |
| 106 | Red   | \$6.79  |
| 107 | Blue  | \$6.79  |
| 108 | Black | \$21.99 |
| 109 | White | \$21.99 |
| 110 | Blue  | \$21.99 |

Show    Exit

## Chap08\Exercise1

### 36. Adaline Solution\_EXERCISE 19\_advanced

- 1D array, 2D array, For Each...Next, ElseIf, SelectedIndexChanged, lst.Items.Add, GetUpperBound(0), frmMain\_Load, lst.Items.Add, Select Case, Private Sub ...  
*(continued)*

19. Create a Windows Forms application. Use the following names for the project and solution, respectively: Adaline Project and Adaline Solution. Save the application in the VB2017\Chap08 folder. Create the interface shown in Figure 8-54. The figure also contains the major tasks that the application needs to perform. Code the application. Save the solution and then start and test the application.

**Adaline Inc.**

Salespeople:  All  Jacob Schmidt  
 Joe Smith  Rex Parker  
 Sue Chen  Sue Perez  
 Suman Patel

Years:  Both  2018  2019

Years:  2018  2019    Status:  Highest  Lowest

Sales:

Sold by:

Figure 8-54 Interface and information for Exercise 19 (continues)

**Tasks**

- 1). Declare a class-level Integer array to store the sales amounts for the following salespeople. You can create either a two-row, six-column array or a six-row, two-column array.

| Salespeople   | 2018 Sales (\$) | 2019 Sales (\$) |
|---------------|-----------------|-----------------|
| Jacob Schmidt | 4000            | 7200            |
| Joe Smith     | 4000            | 5000            |
| Rex Parker    | 2500            | 6500            |
| Sue Chen      | 4000            | 7200            |
| Sue Perez     | 3900            | 6000            |
| Suman Patel   | 3600            | 7000            |

- 2). Regarding the controls in the first group box: The Display sales button should display the sales amount associated with the items selected in the Salespeople and Years list boxes. For example, if All and Both are selected, the button should display the total of all the sales stored in the array (\$60,900). If All and 2019 are selected, the button should display only the sales amounts for the year 2019 (\$38,900). If Sue Chen and Both are selected, the button should display the total sales made by Sue Chen for both years (\$11,200). If Sue Chen and 2018 are selected, the button should display only the amount Sue Chen sold in 2018 (\$4,000).
- 3). Regarding the controls in the second group box: The Sales and names button should display the sales amount and names associated with the items selected in the Years and Status list boxes. For example, if 2018 and Highest are selected, the button should display the highest sales amount made in 2018 (\$4,000) and the name(s) of each salesperson who made that sales amount (Jacob Schmidt, Joe Smith, and Sue Chen).

Figure 8-54 Interface and information for Exercise 19

```

1 Option Explicit On
2 Option Strict On
3 Option Infer Off
4
5 'declare a Class-level Integer 2D Array to store amounts for the following salespeople.
6 ' - you can create either a 2-row & 6-column or 6-row & 2-column 2D Array.
7 'Salespeople:      2018 Sales ($):      2019 Sales ($):      Total:
8 ' All           22,000            38,900            60,900
9 ' Jacob Schmidt  4,000             7,200             11,200
10 ' Joe Smith     4,000             5,000             9,000
11 ' Rex Parker    2,500             6,500             9,000
12 ' Sue Chen      4,000             7,200             11,200
13 ' Sue Perez     3,900             6,000             9,900
14 ' Suman Patel   3,600             7,000             10,600
15
16 'GroupBox1: (btn1_DisplaySales) should:
17 ' display in (lbl1_Sales) the sales amount associated with the items selected in (lst1a_Salespeople) & (lst1b_Years)
18 ' - e.g.1: if (All) & (Both) are selected, (lbl1_Sales) should display the Total of all sales stored in 2D Array = $60,900
19 ' - e.g.2: if (All) & (2019) are selected, (lbl1_Sales) should display only the sales amounts for the year 2019 = $38,900
20 ' - e.g.3: if (Sue Chen) & (Both) are selected, (lbl1_Sales) should display the Total sales made by Sue Chen for both years = $11,200
21 ' - e.g.4: if (Sue Chen) & (2018) are selected, (lbl1_Sales) should display only the amount Sue Chen sold in 2018 = $4000
22
23 'GroupBox2: (btn2_SalesAndNames) should:
24 ' display the sales amount in (lbl2_Sales) and names in (lst2c_SoldBy) associated with the items selected in (lst2a_Years) & (lst2b_Status)
25 ' - e.g.1: if (2018) & (Highest) are selected:
26 '     (lbl2_Sales) should display the highest sales amount made in 2018 = $4,000 &
27 '     (lst2c_SoldBy) should display the name of each salesperson who made that sales amount = Jacob Schmidt, Joe Smith, Sue Chen
28
29 Public Class frmMain
30
31     ' {2018, 2019}:
32     Private intAmounts2D(,) As Integer = {{4000, 7200}, {4000, 5000}, {2500, 6500}, {4000, 7200}, {3900, 6000}, {3600, 7000}}
33     ' idea for _Load and selecting and answering: - added "All" & changed indexes:
34     Private strNames() As String = {"All", "Jacob Schmidt", "Joe Smith", "Rex Parker", "Sue Chen", "Sue Perez", "Suman Patel"}
35     ' ideaaa for _Load and selecting:
36     Private strYears() As String = {"Both", "2018", "2019"}
37
38     Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
39         ' GroupBox1:
40         ' (lst1a_Salespeople) from: Private str2_Names()
41         For Each strName As String In strNames
42             lst1a_Salespeople.Items.Add(strName)
43         Next strName
44         lst1a_Salespeople.SelectedIndex = 0

```

```

45
46      ' (lst1b_Years) from: Private strYears() = {"Both", "2018", "2019"}
47      For Each strYear As String In strYears
48          lst1b_Years.Items.Add(strYear)
49      Next strYear
50      lst1b_Years.SelectedIndex = 0
51
52      ' GroupBox2:
53      ' (lst2a_Years) from: Private strYears() = {"Both", "2018", "2019"} - but only indexes 1&2
54      lst2a_Years.Items.Add(strYears(1))
55      lst2a_Years.Items.Add(strYears(2))
56      lst2a_Years.SelectedIndex = 0
57
58      ' (lst2b_Status)
59      lst2b_Status.Items.Add("Highest")
60      lst2b_Status.Items.Add("Lowest")
61      lst2b_Status.SelectedIndex = 0
62 End Sub
63
64
65 Private Sub btn1_DisplaySales_Click(sender As Object, e As EventArgs) Handles btn1_DisplaySales.Click
66     ' lst1a_Salespeople -> index 0(all) to index 6
67     ' lst1b_Years -> 0=all, 1=2018, 2=2019
68     Dim intNumbers As Integer = Nothing
69
70     If lst1a_Salespeople.SelectedIndex = 0 Then                      ' all Salesperson:
71         Select Case True
72             Case lst1b_Years.SelectedIndex = 0                         ' all in 2018 + 2019:
73                 For Each intElement As Integer In intAmounts2D
74                     intNumbers += intElement
75                 Next intElement
76                 lbl1_Sales.Text = intNumbers.ToString("C0")
77             Case lst1b_Years.SelectedIndex = 1                         ' all in 2018:
78                 For intCounter As Integer = 0 To 5
79                     intNumbers += intAmounts2D(intCounter, 0)
80                 Next intCounter
81                 lbl1_Sales.Text = intNumbers.ToString("C0")
82             Case lst1b_Years.SelectedIndex = 2                         ' all in 2019:
83                 For intCounter As Integer = 0 To 5
84                     intNumbers += intAmounts2D(intCounter, 1)
85                 Next intCounter
86                 lbl1_Sales.Text = intNumbers.ToString("C0")
87         End Select
88

```

```

89      Else                                ' 1->6 each Salesperson:
90          For intCounter As Integer = 1 To 6
91              If lst1a_Salespeople.SelectedIndex = intCounter Then
92                  Select Case True
93                      Case lst1b_Years.SelectedIndex = 0                  ' in 2018 + 2019:
94                          intNumbers = intAmounts2D(intCounter - 1, 0) + intAmounts2D(intCounter - 1, 1)
95                          lbl1_Sales.Text = intNumbers.ToString("C0")
96                      Case lst1b_Years.SelectedIndex = 1                  ' in 2018:
97                          intNumbers = intAmounts2D(intCounter - 1, 0)
98                          lbl1_Sales.Text = intNumbers.ToString("C0")
99                      Case lst1b_Years.SelectedIndex = 2                  ' in 2019:
100                         intNumbers = intAmounts2D(intCounter - 1, 1)
101                         lbl1_Sales.Text = intNumbers.ToString("C0")
102                 End Select
103
104             End If
105         Next intCounter
106     End If
107 End Sub
108
109
110 Private Sub btn2_SalesAndNames_Click(sender As Object, e As EventArgs) Handles btn2_SalesAndNames.Click
111     Dim intColumn As Integer = Nothing
112     Dim intNumber As Integer = Nothing
113     lst2c_SoldBy.Items.Clear()
114
115     Select Case True
116
117         ' year 2018:
118         Case lst2a_Years.SelectedIndex = 0
119             intColumn = 0
120             intNumber = intAmounts2D(0, 0)
121
122             ' highest in 2018: fill number (lbl2_Sales)_ok
123             If lst2b_Status.SelectedIndex = 0 Then
124                 For intCounter As Integer = 0 To intAmounts2D.GetUpperBound(0)
125                     If intNumber < intAmounts2D(intCounter, intColumn) Then
126                         intNumber = intAmounts2D(intCounter, intColumn)
127                     End If
128                 Next intCounter
129                 lbl2_Sales.Text = intNumber.ToString("C0")
130             End If

```

```
131             ' lowest in 2018: fill number (lbl2_Sales)_ok
132         ElseIf lst2b_Status.SelectedIndex = 1 Then
133             For intCounter As Integer = 0 To intAmounts2D.GetUpperBound(0)
134                 If intNumber > intAmounts2D(intCounter, intColumn) Then
135                     intNumber = intAmounts2D(intCounter, intColumn)
136                 End If
137             Next intCounter
138             lbl2_Sales.Text = intNumber.ToString("C0")
139         End If
140
141         ' year 2019:
142     Case lst2a_Years.SelectedIndex = 1
143         intColumn = 1
144         intNumber = intAmounts2D(0, 1)
145
146         ' highest in 2019: fill number (lbl2_Sales)_ok
147         If lst2b_Status.SelectedIndex = 0 Then
148             For intCounter As Integer = 0 To intAmounts2D.GetUpperBound(0)
149                 If intNumber < intAmounts2D(intCounter, intColumn) Then
150                     intNumber = intAmounts2D(intCounter, intColumn)
151                 End If
152             Next intCounter
153             lbl2_Sales.Text = intNumber.ToString("C0")
154
155         ' lowest in 2019: fill number (lbl2_Sales)_ok
156         ElseIf lst2b_Status.SelectedIndex = 1 Then      ' lowest in 2019
157             For intCounter As Integer = 0 To intAmounts2D.GetUpperBound(0)
158                 If intNumber > intAmounts2D(intCounter, intColumn) Then
159                     intNumber = intAmounts2D(intCounter, intColumn)
160                 End If
161             Next intCounter
162             lbl2_Sales.Text = intNumber.ToString("C0")
163         End If
164     End Select
165
166         ' global: fill names (lst2c_SoldBy)_ok
167     For intCounter2 As Integer = 0 To 5
168         If intAmounts2D(intCounter2, intColumn) = intNumber Then
169             lst2c_SoldBy.Items.Add(strNames(intCounter2 + 1))
170         End If
171     Next intCounter2
172
173 End Sub
174
```

```

175  Private Sub GroupBox1SelectedIndex(sender As Object, e As EventArgs) Handles lst1a_Salespeople.SelectedIndexChanged,
176      lst1b_Years.SelectedIndexChanged
177      lbl1_Sales.Text = Nothing
178  End Sub
179
180  Private Sub GroupBox2SelectedIndex(sender As Object, e As EventArgs) Handles lst2a_Years.SelectedIndexChanged,
181      lst2b_Status.SelectedIndexChanged
182      lbl2_Sales.Text = Nothing
183      lst2c_SoldBy.Items.Clear()
184  End Sub
185
186 End Class

```

The application window has four main sections:

- Salespeople:** A dropdown menu containing "All", "Jacob Schmidt", "Joe Smith", "Rex Parker", "Sue Chen", "Sue Perez", and "Suman Patel".
- Years:** A dropdown menu containing "Both", "2018", and "2019".
- Status:** A dropdown menu containing "Highest" and "Lowest".
- Sold by:** An empty text input field.

Below these sections are three buttons:

- Sales:** An empty text input field.
- Display sales**: A button.
- Sales and names**: A button.

**Screenshot 1 (Top Left):** All dropdown menus show "Both", "Both", and "Highest". The "Display sales" button is highlighted.

**Screenshot 2 (Top Right):** The "Years" dropdown shows "Both", "2018", and "2019". The "Status" dropdown shows "Highest" and "Lowest". The "Sold by" field is empty. The "Sales" field contains "\$60,900". The "Display sales" button is highlighted.

**Screenshot 3 (Bottom Left):** The "Years" dropdown shows "Both", "2018", and "2019". The "Status" dropdown shows "Highest" and "Lowest". The "Sold by" field shows "Joe Smith". The "Sales" field contains "\$6,000". The "Sales and names" button is highlighted.

**Screenshot 4 (Bottom Right):** The "Salespeople" dropdown shows "All", "Jacob Schmidt", "Joe Smith", "Rex Parker", "Sue Chen", "Sue Perez", and "Suman Patel". The "Years" dropdown shows "Both", "2018", and "2019". The "Status" dropdown shows "Highest" and "Lowest". The "Sold by" field shows "Jacob Schmidt" and "Sue Chen". The "Sales" field contains "\$7,200". The "Sales and names" button is highlighted.

## Chap08\\_Exercise1

### 37.ReDim Solution\_EXERCISE 20\_advanced

- 1D accumulator array, ReDim Preserve, MessageBox.Show, txt\_Enter, txt\_KeyPress, txt\_TextChanged

- note: more info about Array ReDim statement will be included within' my notes from CH8

CH8\_A5 - my addendum from EXERCISES: Array statement ReDim - 37.ReDim Solution\_EXERCISE 20\_advanced

20. Research the Visual Basic ReDim statement. What is the purpose of the statement? What is the purpose of the Preserve keyword?
- Open the ReDim Solution.sln contained in the VB2017\Chap08\ReDim Solution folder. Open the Code Editor window and locate the array declaration statement in the form class's declarations section. Start the application. Type 25 in the Sales box and then click the Add to array button. An error message box opens and informs you that the computer encountered an error when trying to process the TryParse method in the btnAdd\_Click procedure. Stop the application.
  - Modify the btnAdd\_Click procedure so that it can store any number of sales amounts in the array. (Hint: Use the ReDim statement to add an element to the array before the TryParse method is processed.)
  - Save the solution and then start the application. Type 25 in the Sales box and then press Enter. Click the Display array button . The sales amount appears in the list box.
  - Now, use the Add to array button to add the following sales amounts, one at a time: 700, 550, and 800. Click the Display array button. The four sales amounts appear in the list box.

The screenshot shows the Microsoft Visual Studio IDE. On the left, the code editor displays a portion of a Visual Basic module:

```
17     If txtSales.Text <> String.Empty Then
18
19     Integer.TryParse(txtSales.Text, intSales(intSub))
20     intSub += 1
21
22     Else
23         MessageBox.Show("Please enter a sales amount.", "S
24             MessageBoxButtons.OK, MessageBoxIcon.Informa
25
26     End If
27     txtSales.Focus()
28     txtSales.SelectAll()
29     lstSales.Items.Clear()
End Sub
```

Line 19 contains a syntax error: `Integer.TryParse(txtSales.Text, intSales(intSub))`. A yellow exclamation mark icon is positioned next to the word `Integer` in the code editor. To the right of the code editor, an 'Exception Unhandled' dialog box is displayed:

Exception Unhandled

System.IndexOutOfRangeException: 'Index was outside the bounds of the array.'

View Details | Copy Details

► Exception Settings

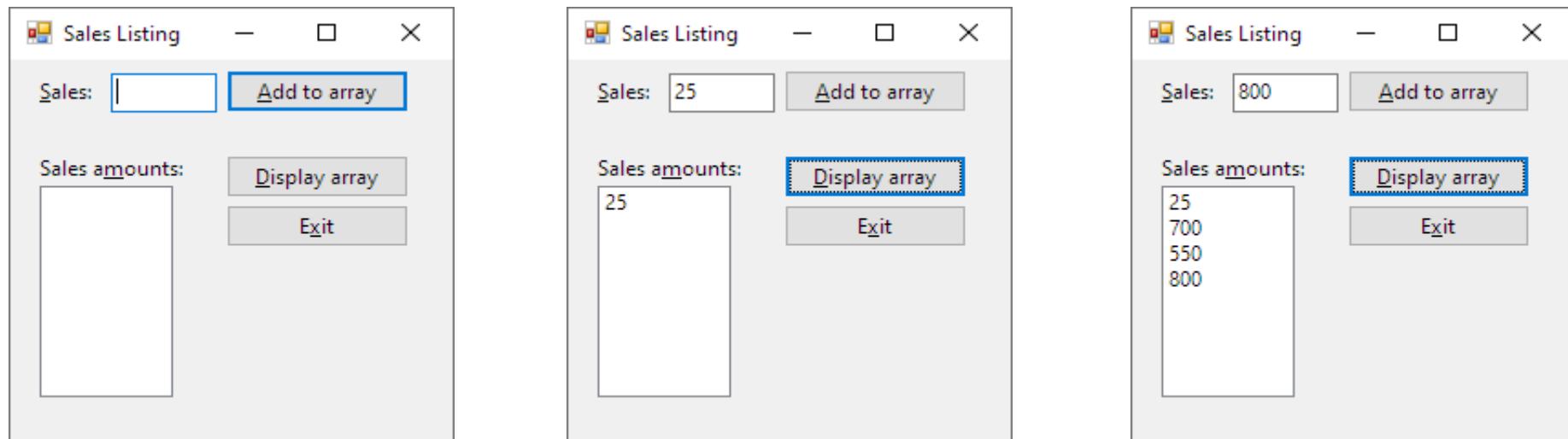
```
1  ' Name:      ReDim Project
2  ' Purpose:    Add sales to an array and then display the array.
3  ' Programmer: <your name> on <current date>
4
5  Option Explicit On
6  Option Strict On
7  Option Infer Off
8
9  Public Class frmMain
```

```
10     Private intSales() As Integer = {}
11
12     Private Sub btnAdd_Click(sender As Object, e As EventArgs) Handles btnAdd.Click
13         ' Adds a sales amount to the array.
14
15         Static intSub As Integer
16
17         If txtSales.Text <> String.Empty Then
18             ReDim Preserve intSales(intSub) ' Preserve = keep previous values in the array
19             Integer.TryParse(txtSales.Text, intSales(intSub))
20             intSub += 1
21         Else
22             MessageBox.Show("Please enter a sales amount.", "Sales Listing",
23                             MessageBoxButtons.OK, MessageBoxIcon.Information)
24         End If
25         txtSales.Focus()
26         txtSales.SelectAll()
27         lstSales.Items.Clear()
28     End Sub
29
30     Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
31         ' Displays the sales amounts stored in an array.
32
33         lstSales.Items.Clear()
34         For intSub As Integer = 0 To intSales.GetUpperBound(0)
35             lstSales.Items.Add(intSales(intSub).ToString())
36         Next intSub
37     End Sub
38
39     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
40         Me.Close()
41     End Sub
42
43     Private Sub txtSales_Enter(sender As Object, e As EventArgs) Handles txtSales.Enter
44         txtSales.SelectAll()
45     End Sub
46
47     Private Sub txtSales_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtSales.KeyPress
48         ' Accept only numbers and the Backspace key.
49
50         If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
51             e.Handled = True
52         End If
53     End Sub
```

```

54
55     Private Sub txtSales_TextChanged(sender As Object, e As EventArgs) Handles txtSales.TextChanged
56         lstSales.Items.Clear()
57     End Sub
58 End Class

```



## Chap08\Exercise1

### 38.OnYourOwn Solution\_EXERCISE 21

- no idea, not done

Create a Windows Forms application. Use the following names for the project and solution, respectively: OnYourOwn Project and OnYourOwn Solution. Save the application in the VB2017\Chap08 folder. Plan and design an application of your choice. The only requirement is that you must follow the minimum guidelines listed in Figure 8-55. Before starting the application, be sure to verify the name of the startup form. Save the solution and then start and test the application.

1. The user interface must contain a minimum of one text box, three labels, and two buttons. One of the buttons must be an Exit button.
2. The interface can include a picture box, but this is not a requirement.
3. The interface must follow the GUI design guidelines summarized for Chapters 2 through 8 in Appendix A.
4. Objects that are either coded or referred to in code should be named appropriately.
5. The Code Editor window must contain comments, the three Option statements, at least two variables, at least two assignment statements, at least two of the concepts covered in the Focus lesson, and the Me.Close() statement.
6. Every text box on the form should have its TextChanged and Enter event procedures coded.

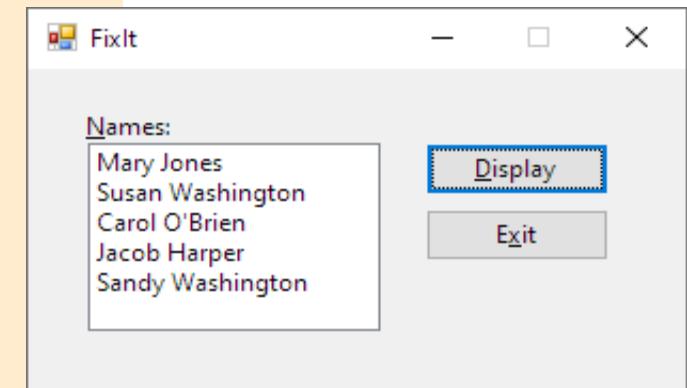
Figure 8-55 Guidelines for Exercise 21

```

1  ' Name:      FixIt Project
2  ' Purpose:    Displays the contents of an array in a list box.
3  Option Explicit On
4  Option Strict On
5  Option Infer Off
6
7  Public Class frmMain
8
9      Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
10         ' Displays the first and last names in a list box.
11
12         ' original wrong:
13         'Dim strNames() As String....
14         ' MyFix:
15         Dim strNames() As String = {"Mary", "Jones"}, {"Susan", "Washington"}, {"Carol", "O'Brien"},
16                         {"Jacob", "Harper"}, {"Sandy", "Washington"}
17
18         ' Clear list box and then display first and last names separated by a space.
19         lstNames.Items.Clear()
20
21         ' original wrong:
22         'For intX As Integer = 1 To strNames.GetUpperBound(0)
23         'lstNames.Items.Add(strNames(0, intX) & " " & strNames(1, intX))
24         'Next intX
25         ' MyFix:
26         For intX As Integer = 0 To strNames.GetUpperBound(0)
27             lstNames.Items.Add(strNames(intX, 0) & " " & strNames(intX, 1))
28         Next intX
29
30     End Sub
31
32     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
33         Me.Close()
34     End Sub
35
36 End Class

```

Open the VB2017\Chap08\FixIt Solution\FixIt Solution.sln file. Open the Code Editor window and locate the btnDisplay\_Click procedure. The first column in the array contains first names, and the second column contains last names. The procedure should add each person's first and last names, separated by a space character, to the list box. Correct the syntax error(s) in the existing code. Save the solution and then start the application. Click the Display button. If the btnDisplay\_Click procedure is not working correctly, stop the application and correct the procedure's code.



- in addition to getting data from the keyboard and sending data to the computer screen, an application can also read data from and write data to a file on a disk
- in this chapter's Focus on the Concepts lesson, you will learn how to create and use a special type of file, called a sequential access file
- you will use sequential access files in the applications coded in this chapter's Apply the Concepts lesson
- the Apply lesson also covers the creation and coding of menus

#### CH9\_FOCUS ON THE CONCEPTS LESSON

- CH9\_F1** - File Handling - Sequential Access Files / Text files introduction: **Namespace System.IO**
- CH9\_F1.1** - **Namespace System.IO** and its **Classes** - basic info including messages(exceptions)
- CH9\_F2** - Sequential Access **Output** Files: **Class StreamWriter** - how to create and use them step by step
- CH9\_F2.1** - **Output Text** Files: **Class StreamWriter** example: code the **Game Show Application** (01.Game Show Solution-TextBox) 1/2
- CH9\_F3** - Sequential Access **Input** Files: **Class StreamReader** - how to create and use them step by step
- CH9\_F3.1** - **Input Text** Files: **Class StreamReader**: **ReadToEnd** function example: code the **Game Show Application** (01.Game Show Solution-TextBox) 2/2
- CH9\_F3.2** - Sequential Access **Input** Files: **Class StreamReader**: **ReadLine** function example: code the **Game Show Application** (02.Game Show Solution-ListBox)
- CH9\_F4** - **You Do It 1:** Sequential Access **Output & Input** file exercise: 03.You Do It 1 Solution  
-> AppendText, WriteLine, OpenText, ReadToEnd, Peek, ReadLine, Close, \_MouseClick

#### CH9\_APPLY THE CONCEPTS LESSON

- CH9\_A1** - Add a **Menu(mnu)** to a **Form: Toolbox / Menus & Toolbars / ToolStrip** control
- CH9\_A1.1** - **GUI** guidelines for **Menus(mnu)**:
- CH9\_A2** - **Menu** example: **Continents Application** - create Sequential Access File (**.txt**) to fill **Array** with values (04.Continents Solution) 1/4
- CH9\_A2.1** - **Menu** example: **Continents Application** - add **Menus** to the **GUI** (04.Continents Solution) 2/4
- CH9\_A2.2** - **Menu** example: **Continents Application** - code the **Items** on a **Menu** (04.Continents Solution) 3/4
- CH9\_A2.3** - **Menu** example: **Continents Application** - modify a **Menu** (04.Continents Solution) 4/4
- CH9\_A3** - **Accumulate the Values** Stored in a **File** example: **05.Harkins Solution**
- CH9\_A4** - easy way to **Sort** the **Data** contained in a **File** example: **06.States Solution**
- CH9\_A5** - **Professionalize Your Application's Interface:** **confirmation message** when the **file** is written (07.Game Show Solution-Professionalize)

**CH9\_Summary:** Sequential access files, **MenuStrip** tool, easy sorting of Sequential access file, confirmation message

**CH9\_Key Terms**

**CH9\_Exercises**

## CH9\_FOCUS ON THE CONCEPTS LESSON

### CH9\_F1 - File Handling - Sequential Access Files / Text files introduction: Namespace System.IO

- the **IO** that appears in the syntaxes stands for **Input/Output**
- at times, an application may need to **read data from** and **write data to** a **file** on a disk
- a **file**: = collection of data stored in a disk with a specific **name** and **directory path**
  - files to which data is written are called **output files** because the files store the output produced by an application
  - files that are read by the computer are called **input files** because an application uses the data in these files as input
- when a **file** is opened for **reading** or **writing**, it becomes a **stream**
- the data is composed of sequence of characters, which programmers refer to as a **stream of characters/character stream**
- **stream**: = basically the sequence of bytes passing through the communication path
  - 2 main **streams**: a). **Input** stream - used for **reading** data from file = read operation
  - b). **Output** stream - used for **writing** into the file = write operation
- the characters in most **input** and **output** files are both **read** and **written** in consecutive {následný, postupný, nepřetržitý} order, one character at a time, beginning with the 1st character and ending with the last character
- such files are referred to as **sequential access files** because of the manner in which the characters are accessed
- they are also called **text files** because they can be opened and modified by a text editor, such as Notepad

### CH9\_F1.1 - Namespace System.IO and its Classes - basic info including messages(exceptions)

- the **System.IO Namespace** has various **Classes** that are used for performing various operations with **files**, like creating, deleting, closing, reading from, writing to...  
e.g. `Dim outFile As IO.StreamWriter`

| IO Class:                  | description:                                                                                                                                                                                          |
|----------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| BinaryReader               | Reads primitive data types as binary values in a specific encoding.                                                                                                                                   |
| BinaryWriter               | Writes primitive types in binary to a stream and supports writing strings in a specific encoding.                                                                                                     |
| BufferedStream             | Adds a buffering layer to r/w operations on another stream; this class can't be inherited [zdědit].                                                                                                   |
| Directory                  | Exposes static methods for creating, moving and enumerating through directories and subdirectories; can't be inherited [zdědit].                                                                      |
| DirectoryInfo              | Exposes instance methods for creating, moving and enumerating through directories and subdirectories; can't be inherited [zdědit].                                                                    |
| DirectoryNotFoundException | The exception that is thrown when part of a file or directory cannot be found.                                                                                                                        |
| DriveInfo                  | Provides access to information on a drive.                                                                                                                                                            |
| DriveNotFoundException     | The exception that is thrown when trying to access a drive or share that is not available.                                                                                                            |
| EndOfStreamException       | The exception that is thrown when reading is attempted past the end of a stream.                                                                                                                      |
| ErrorEventArgs             | Provides data for the Error event.                                                                                                                                                                    |
| File                       | Provides static methods for the creation, copying, deletion, moving, and opening of a single file, and aids in the creation of <b>IO.FileStream</b> objects.                                          |
| FormatException            | The exception that is thrown when an input file/a data stream that is supposed to conform [vyhovovat] to a certain file format specification is malformed.                                            |
| FileInfo                   | Provides properties and instance methods for the creation, copying, deletion, moving, and opening of files, and aids in the creation of <b>IO.FileStream</b> objects. This class cannot be inherited. |
| FileNotFoundException      | The exception that is thrown when a managed assembly is found but cannot be loaded.                                                                                                                   |
| FileNotFoundException      | The exception that is thrown when an attempt to access a file that does not exist on disk fails.                                                                                                      |

|                                                 |                                                                                                                                                        |
|-------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------|
| <a href="#">FileStream</a>                      | Provides a Stream for a file, supporting both synchronous and asynchronous read and write operations.                                                  |
| <a href="#">FileSystemEventArgs</a>             | Provides data for the directory events: Changed, Created, Deleted.                                                                                     |
| <a href="#">FileSystemInfo</a>                  | Provides the base class for both FileInfo and DirectoryInfo objects.                                                                                   |
| <a href="#">FileSystemWatcher</a>               | Listens to the file system change notifications and raises events when a directory, or file in a directory, changes.                                   |
| <a href="#">InternalBufferOverflowException</a> | The exception thrown when the internal buffer overflows.                                                                                               |
| <a href="#">InvalidDataException</a>            | The exception that is thrown when a data stream is in an invalid format.                                                                               |
| <a href="#">IODescriptionAttribute</a>          | Sets the description visual designers can display when referencing an event, extender, or property.                                                    |
| <a href="#">IOException</a>                     | The exception that is thrown when an I/O error occurs.                                                                                                 |
| <a href="#">MemoryStream</a>                    | Creates a stream whose backing store is memory.                                                                                                        |
| <a href="#">Path</a>                            | Performs operations on String instances that contain file or directory path information.<br>These operations are performed in a cross-platform manner. |
| <a href="#">PathTooLongException</a>            | The exception that is thrown when a path or fully qualified file name is longer than the system-defined maximum length.                                |
| <a href="#">PipeException</a>                   | Thrown when an error occurs within a named pipe.                                                                                                       |
| <a href="#">RenamedEventArgs</a>                | Provides data for the Renamed event.                                                                                                                   |
| <a href="#">Stream</a>                          | Provides a generic view of a sequence of bytes. This is an abstract class.                                                                             |
| <a href="#">StreamReader</a>                    | Implements a <a href="#">IO.TextReader</a> that reads characters from a byte stream in a particular encoding.                                          |
| <a href="#">StreamWriter</a>                    | Implements a <a href="#">IO.TextWriter</a> for writing characters to a stream in a particular encoding.                                                |
| <a href="#">StringReader</a>                    | Implements a TextReader that reads from a string.                                                                                                      |
| <a href="#">StringWriter</a>                    | Implements a TextWriter for writing information to a string. The information is stored in an underlying StringBuilder.                                 |
| <a href="#">TextReader</a>                      | Represents a reader that can read a sequential series of characters.                                                                                   |
| <a href="#">TextWriter</a>                      | Represents a writer that can write a sequential series of characters. This class is abstract.                                                          |
| <a href="#">UnmanagedMemoryAccessor</a>         | Provides random access to unmanaged blocks of memory from managed code.                                                                                |
| <a href="#">UnmanagedMemoryStream</a>           | Provides access to unmanaged blocks of memory from managed code.                                                                                       |

**CH9\_F2 - Sequential Access Output Files: Class StreamWriter** - how to create and use them step by step

## Creating and using an output file: `IO.StreamWriter`

**steps:** e.g.: **description:**

e.g.:

*description:*

## #1 step:

```
Dim/Private outFile As IO.StreamWriter
```

<- use the **StreamWriter Class** to declare a **StreamWriter** variable

### #2 step:

```
outFile = IO.File.CreateText("employee.txt")  
Or  
outFile = IO.File.AppendText("D:\report.txt")
```

<- use either **File Class**'s function to open the file & assign it to the **StreamWriter** variable:

- a). if the file already exists, the computer erases the contents of the file before writing data to it or
- b). new data is written **after** any existing data in the file [append = připojít]

### **#3 step:**

```
outFile.WriteLine("Hello")  
or  
outFile.WriteLine("Hello")
```

<- use either **IO.StreamWriter** Class's sub to write data to the file:

- a). writes a string to the stream immediately after the last character in the stream  
or
- b). writes a **newline** string to the stream followed by a **line terminator** - unlike the **Write** method

#### #4 step:

```
outFile.Close()
```

<- close the output file using [IO.StreamWriter Class](#)'s method **Close()**

**#1 step:** - use the **StreamWriter Class** to declare a **StreamWriter** variable:

**StreamWriter Class's StreamWriter variable syntax:**  
**As IO.StreamWriter**

**Dim/Private streamWriterVariable As IO.StreamWriter**

**As IO.StreamWriter**

**streamWriterVariable** - **As IO.StreamWriter**  
= name of your variable  
**IO** - using prefix **out-** will be easy to recognize an **output** file  
- stands for Input/Output  
- **Namespace System.IO**  
= contains types that allow reading and writing to files and data streams,  
and types that provide basic file and directory support  
**StreamWriter** - **Class System.IO.StreamWriter**  
= implements a **IO.TextWriter** for writing characters to a stream in a  
particular encoding

e.g.

**Dim outFile As IO.StreamWriter**

**#2 step:** 1). use either **File Class**'s functions to open the file: - a). the **CreateText** function or

- b). the **AppendText** function [Append = připojit]

- a). the **CreateText** function = creates or opens a file for writing **UTF-8** encoded text

- if the file already exists, the computer **erases** the contents of the file before writing any data to it

- b). the **AppendText** function = creates a **IO.StreamWriter** that appends **UTF-8** encoded text to an existing file, or to a new file if does not exist  
- new data is written **after** any existing data in the file [Append = připojit]

2). assign the **StreamWriter Class**'s object created by either method to the **StreamWriter** variable from **#1 step:**

**File Class's CreateText method syntax:**  
**As IO.StreamWriter**

**IO.File.CreateText(path As String)**

**As IO.StreamWriter**

**File Class's AppendText method syntax:**  
**As IO.StreamWriter**

**IO.File.AppendText(path As String)**

**As IO.StreamWriter**

**IO** - stands for Input/Output  
- **Namespace System.IO**  
= contains types that allow reading and writing to files and data streams,  
and types that provide basic file and directory support  
**File** - **Class System.IO**  
= Provides static methods for the creation, copying, deletion, moving, and opening of a single file,  
and aids [pomáhá] in the creation of **IO.FileStream** objects.  
**CreateText** - **As IO.StreamWriter**  
- **Function System.IO.File**  
= creates or opens a file for writing **UTF-8** encoded text  
- if the file already exists, the computer **erases** the contents of the file before writing any data to it

**AppendText**

- **As IO.StreamWriter**
- **Function System.IO.File**
- = creates a **IO.StreamWriter** that appends **UTF-8** encoded text to an existing file, or to a new file if the specified file does not exist
- new data is written **after** any existing data in the file

**(path)**

- **As String**
- either name of your file located in project's default folder: **bin\Debug** or
- path where to locate your file + name of your file

e.g.1

```
outFile = IO.File.CreateText("employee.txt")
outFile = IO.File.AppendText("D:\report.txt")
```

e.g.2

<- the computer will search for the **employee.txt** file in the project's default folder: **bin\Debug**

<- specify the folder path **only** when you are sure that it will **not** change

**#3 step:** - use either **IO.StreamWriter Class's** sub to write data to the file:

- **a).** the **Write** sub or **Sub** = won't return a value, unlike **Method/Function**
- **b).** the **WriteLine** sub

- **a).** the **Write** sub = writes a string to the stream **immediately** after the last character in the stream

- **b).** the **WriteLine** sub = writes a **newline** string to the stream followed by a **line terminator** - unlike the **Write** method

[StreamWriter Class](#)'s method [Write](#) syntax:

```
streamWriterVariable.Write(value As String)
```

[StreamWriter Class](#)'s method [WriteLine](#) syntax:

```
streamWriterVariable.WriteLine(value As String)
```

**streamWriterVariable**

- **As IO.StreamWriter**
- = name of your variable

**Write**

- **Sub IO.TextWriter** **Sub** = won't return a value, unlike **Method/Function**
- = writes a string to the stream immediately after the last character

**WriteLine**

- **Sub IO.TextWriter** **Sub** = won't return a value, unlike **Method/Function**
- = writes a newline character after the data - unlike the **Write** method
- = writes a string followed by a line terminator to the text string or stream

**(value)**

- **As String**
- = any character/string you want to add to the stream
- e.g. "Nazdar Bazar"

e.g.1

```
outFile.Write("Hello")
```

Hello|<- the next character will be written immediately after the "o"

e.g.2

```
outFile.WriteLine("Hello")
```

Hello  
|<- the next character will be written in a new line

- #4 step:**
- close the output file using **IO.StreamWriter Class**'s method **Close()**
  - you should always close an output file as soon as you are finished using it - this ensures that the data is saved, and it makes the file available for use elsewhere in the application
  - a **run time error** will occur if a program statement attempts to open a file that is **already open**

**StreamWriter Class's method Close syntax:**

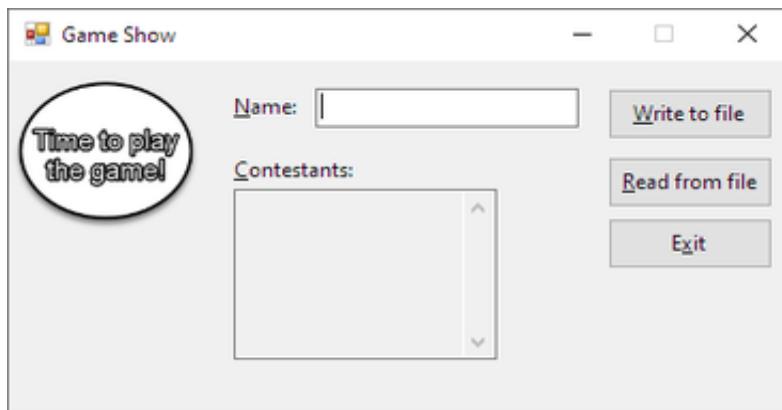
```
streamWriterVariable.Close()
```

|                                                                                                                                                                                                                                                                             |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>streamWriterVariable</b> - <b>As IO.StreamWriter</b><br><b>Close</b> = name of your variable<br>- <b>Sub IO.TextWriter</b> <b>Sub</b> = won't return a value, unlike <b>Method/Function</b><br>= closes the current <b>StreamWriter</b> object and the underlying stream |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

e.g.      **outFile.Close()**

### CH9\_F2.1 - Output Text Files: **Class StreamWriter** example: code the **Game Show Application** (01.Game Show Solution-TextBox) 1/2

- the **Game Show Application** **writes** the names of contestants to a sequential access file named **contestants.txt**
- you will code the button **Write to file** in the next set of steps



- note: the (**txtContestants**) control has properties set like:  
 - **Multiline** = True  
 - **ReadOnly** = True  
 - **ScrollBars** = Vertical

- to code the **btnWrite\_Click** procedure:

1). open the: ...VB2017\Chap09\Exercise\01.Game Show Solution-TextBox\Game Show Solution.sln

- in a Code Editor window locate the **btnWrite\_Click** procedure

2). **#1 step:**

- recall that the **1st** step is to declare a **StreamWriter** variable, so:

-> below the comment line: '**Declare a StreamWriter variable:**' type the following declaration statement and then press **Enter**

```
x      ' Declare a StreamWriter variable:  
x      Dim outFile As IO.StreamWriter  
x
```

### 3). #2 step:

- the **btnWrite\_Click** procedure should add the name entered in the (**txtContestants**) control to the end of the existing names in the file ->
- > therefore, you will need to open the file for append, using the **AppendText** method
- a descriptive name for a file that stores the names of contestants is: **contestants.txt**
- although it is not required, the suffix **.txt** filename extension is commonly used when naming **sequential access files**
- > below the comment line: '**Open the file for append:**' type the following assignment statement and then press **Enter**

```
x      ' Open the file for append:  
x      outFile = IO.File.AppendText("contestants.txt")  
x
```

<- the computer will use the default project's folder: **bin\Debug**

### 4). #3 step:

- each contestant's name should appear on a separate line in the file, so you will use the **WriteLine** method to write each name to the file
- > below the comment line: '**Write the name on a separate line in the file:**' type the following statement and then press **Enter**

```
x      ' Write the name on a separate line in the file:  
x      outFile.WriteLine(txtName.Text.Trim)  
x
```

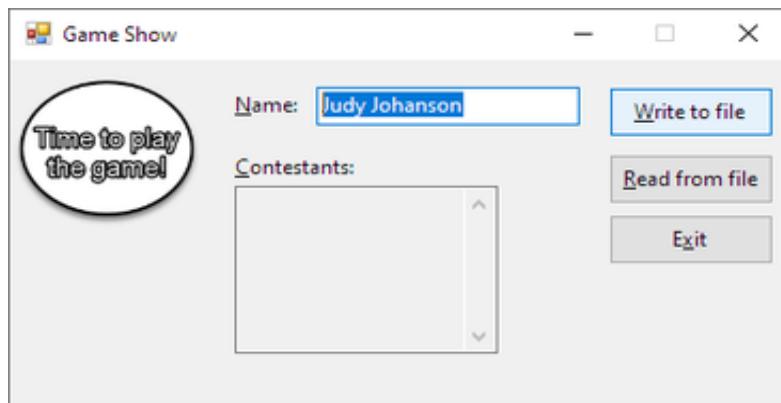
### 5). #4 step:

- the last step from **Creating and using an output file** is to close the file
- important to ensure that the data is saved, but also to make the file available for use elsewhere in the application -> otherwise a runtime error would occur
- > below the comment line: '**Close the file:**' enter the **Close** method

```
x      ' Close the file:  
x      outFile.Close()
```

### 6). save the solution and then start and test the application:

- > in the **Name** box type: **Jess Svarni** and then click the button **Write to file**
- > use the application to write the following 4 names to the file: **Harry Chou, Patti Munez, Sam Sportenski, Judy Johanson**



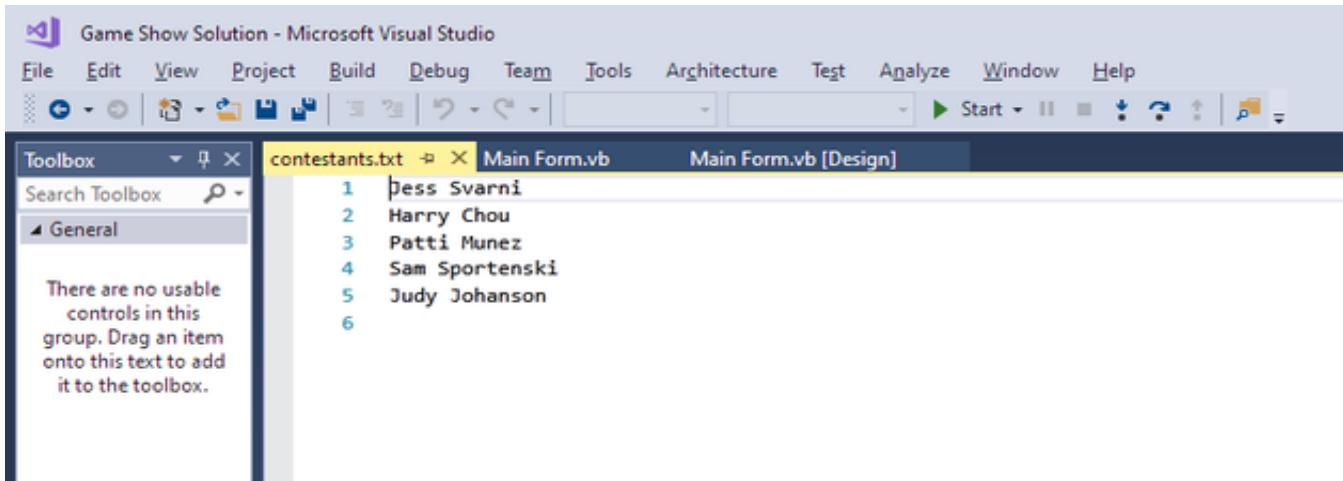
- > click the button **Exit**

7). now, you will open the **contestants.txt** file in Visual Studio to verify its contents

-> on the VS Menu bar click **File**, point to **Open**, and then click **File...**

-> open the project's folder ...bin/Debug and in the list of filenames choose to **Open** your's created file **contestants.txt**

- the new window named **contestants.txt** opens and shows the **5** names contained in the file



8). close the **contestants.txt** window

#### Mini-Quiz 9-1:

1. Write a declaration statement for a procedure-level variable named **outInventory**. The variable will store a **StreamWriter** object.
2. Using the variable from Question 1, write a statement to open a sequential access file named **inventory.txt** for output.
3. Using the variable from Question 1, write a statement that writes the contents of the **strID** variable on a separate line in the file.
4. Write a statement to close the file associated with the variable from Question 1.

```
1...Dim outInventory As IO.StreamWriter
2...outInventory = IO.File.CreateText("inventory.txt")
3...outInventory.WriteLine(strID)
4...outInventory.Close()
```

## Creating and using an input file: **IO.StreamReader**

steps:

e.g.:

description:

**#1 step:**`Dim/Private inFile As IO.StreamReader`<- use the **StreamReader Class** to declare a **StreamReader** variable**#2 step:**`If IO.File.Exists("employee.txt") Then`<- use the **File Class's** function/method **Exists** to determine whether the input file exists**#3 step:**`inFile = IO.File.OpenText("employee.txt")`<- if the function/method **Exists** determines that the file exists,  
use the **IO.StreamReader Class's** function/method **OpenText** to open the file for input and  
assign the created object to the **StreamReader** variable from **#1 step****#4 step:**

```
Dim strLineOfText As String
Do Until inFile.Peek = -1
    strLineOfText = inFile.ReadLine
    ... instructions
Loop
Or
txtReport.Text = inFile.ReadToEnd
```

<- a). to read the file, line by line, use both functions along with a loop:  
1). the **Peek** function/method and then  
2). the **ReadLine** function/method

or

<- b). to read the entire file all at once, use the **ReadToEnd** function**#5 step:**`inFile.Close()`<- close the input file using **IO.StreamReader Class's** method **Close()****#1 step:** - use the **StreamReader Class** to declare a **StreamReader** variable:

**StreamReader Class's StreamReader variable syntax:**  
**As IO.StreamReader**

`Dim/Private streamReaderVariable As IO.StreamReader`**As IO.StreamReader****streamReaderVariable**- **As IO.StreamReader**

= name of your variable

- using prefix **in-** will be easy to recognize an input file

- stands for Input/Output

- **Namespace System.IO**= contains types that allow reading and writing to files and data streams,  
and types that provide basic file and directory support- **Class System.IO**= implements a **IO.TextReader** that reads characters from a byte stream in a  
particular encoding**IO****StreamReader**

e.g.

`Dim inFile As IO.StreamReader`

## #2 step:

- use the **File Class**'s function/method **Exists** to determine whether the input file **exists**
- it is **important** to make this determination because a **run time error** will occur if your code attempts to open an input file that the computer cannot locate

**File Class's Exists** method/function syntax:

As Boolean

**IO.File.Exists(path As String)**

As Boolean

**IO** - stands for Input/Output  
- **Namespace System.IO**  
= contains types that allow reading and writing to files and data streams,  
and types that provide basic file and directory support  
**File** - **Class System.IO.File**  
= provides static methods for the creation, copying, deletion, moving, and opening of a single file,  
and aids in the creation of **IO.FileStream** objects.  
**Exists** - **Function IO.File**  
= determines whether the specified file exists  
**(path)** - **As String**  
- either name of your file located in project's default folder: **bin\Debug** or  
- path where to locate your file + name of your file

e.g.

**If IO.File.Exists("employee.txt") Then ...**

<- the computer will search for the **employee.txt** file in the project's default folder: **bin\Debug**

## #3 step:

- if the function/method **Exists** determines that the file exists, use the **IO.StreamReader Class's** function/method **OpenText** to open the file for input and assign the created object to the **StreamReader** variable from **#1 step**

**StreamReader Class's** function/method **OpenText** syntax:

As **IO.StreamReader**

**IO.File.OpenText(path As String)**

As **IO.StreamReader**

**IO** - stands for Input/Output  
- **Namespace System.IO**  
= contains types that allow reading and writing to files and data streams,  
and types that provide basic file and directory support  
**File** - **Class System.IO.File**  
= provides static methods for the creation, copying, deletion, moving, and opening of a single file, and  
aids in the creation of **IO.FileStream** objects.  
**OpenText** - **Function IO.File**  
= opens an existing UTF-8 encoded text file for reading  
**(path)** - **As String**  
- either the name of your file to be opened for **reading** located in project's default folder: **bin\Debug**  
- or path where to locate your file + name of your file

e.g.

**inFile = IO.File.OpenText("employee.txt")**

#### #4 step:

- a). to read the file, line by line, use both functions along with a loop:
  - 1). the **Peek** function/method and then
  - 2). the **ReadLine** function/method
- b). to read the entire file all at once, use the **ReadToEnd** function

StreamReader Class's function Peek syntax:  
**As Integer**

**streamReaderVariable.Peek() As Integer**

StreamReader Class's function ReadLine syntax:  
**As String**

**streamReaderVariable.ReadLine() As String**

StreamReader Class's function ReadToEnd syntax:  
**As String**

**streamReaderVariable.ReadToEnd() As String**

**streamReaderVariable** - **As IO.StreamReader**  
= name of your variable  
**ReadToEnd** - **As String**  
- **Function IO.StreamReader**  
= reads all characters from the current position to the end of the stream

e.g.1 - example of reading the file, line by line using a loop:

```
Dim strLineOfText As String
Do Until inFile.Peek = -1
    strLineOfText = inFile.ReadLine
    ... instructions
Loop
```

<- **-1** means none available character, i.e. process the loop instructions until the end of the file is reached

e.g.2 - example of reading the entire file all at once:

```
txtReport.Text = inFile.ReadToEnd
```

**#5 step:**

- close the input file using **IO.StreamReader Class**'s method **Close()**
- as you do with an output file, you should always close an input file as soon as you are finished using it ->
  - > this makes the file available for use elsewhere in the application
- a **run time error** will occur if a program statement attempts to open a file that is **already** open

StreamReader Class's method Close syntax:

**streamReaderVariable.Close()**

**streamReaderVariable**

- **As IO.StreamReader**

= name of your variable

**Close**

- **Sub IO.TextWriter**

**Sub** = won't return a value, unlike **Method/Function**

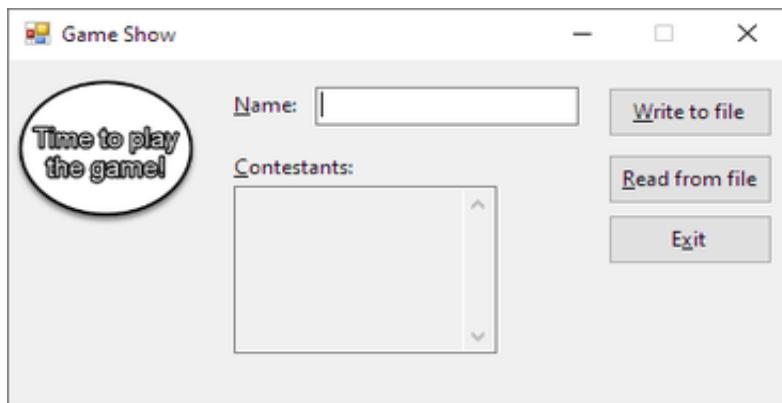
= closes the current **StreamReader** object and the underlying stream

e.g.

**inFile.Close()**

### CH9\_F3.1 - Input Text Files: **Class StreamReader**: ReadToEnd function example: code the **Game Show Application** (01.Game Show Solution-TextBox) 2/2

- button **Read from file** reads the names from the **contestants.txt** file and displays them in the **(txtContestants)** control



- note: the **(txtContestants)** control has properties set like:

- **Multiline** = True
- **ReadOnly** = True
- **ScrollBars** = Vertical

- coding the **btnRead\_Click** procedure:

1). open the: ...VB2017\Chap09\Exercise01.Game Show Solution-TextBox\Game Show Solution.sln

- in a Code Editor window locate the **btnWrite\_Click** procedure

**2). #1 step:**

- recall that the **1st** step is to declare a **StreamReader** variable, so:

-> below the comment line: '**Declare a StreamReader variable:**' type the following declaration statement and then press **Enter**

```
x      ' Declare a StreamReader variable:  
x      Dim inFile As IO.StreamReader  
x
```

### 3). #2 step:

- use the **Exists** function to determine whether the input file exists - in this case the **contestants.txt** file

-> below the comment line: ' Determine whether the file exists: enter the following partial selection structure:

- the structure's **False** path displays an appropriate message if the file does not exists

```
x      ' Determine whether the file exists:  
x      If IO.File.Exists("contestants.txt") Then  
x          ' the True path will be here - open the file for input:  
x  
x      Else  
x          MessageBox.Show("Cannot find the file.", "Game Show", MessageBoxButtons.OK, MessageBoxIcon.Information)  
x      End If  
x  
x  End Sub
```

### 4). #3 step:

- if the **contestants.txt** file exists, the selection structure's **True** path will use the **OpenText** function to open the file for **input**

-> below the comment line: ' the True path will be here - open the file for input: enter the following assignment statement and then press **Enter**

```
x      If IO.File.Exists("contestants.txt") Then  
x          ' the True path will be here - open the file for input:  
x          inFile = IO.File.OpenText("contestants.txt")  
x  
x      Else
```

### 5). #4 step:

- next, the selection structure's **True** path will use the **ReadToEnd** function to read the entire input file, all at once and

- display the file's contents in the **(txtContestants)** control

-> enter the following comment and statement and then press **Enter**:

```
x      If IO.File.Exists("contestants.txt") Then  
x          ' the True path will be here - open the file for input:  
x          inFile = IO.File.OpenText("contestants.txt")  
x          ' Read the file and assign to (txtContestants) control:  
x          txtContestants.Text = inFile.ReadToEnd  
x  
x      Else
```

### 6). #5 step:

- the last step from **Creating and using an input file:** is to **close** the file

- as you do with an output file, you should always close an input file as soon as you are finished using it ->

-> this makes the file available for use elsewhere in the application

- a **run time error** will occur if a program statement attempts to open a file that is **already** open

-> type the following **Close** sub:

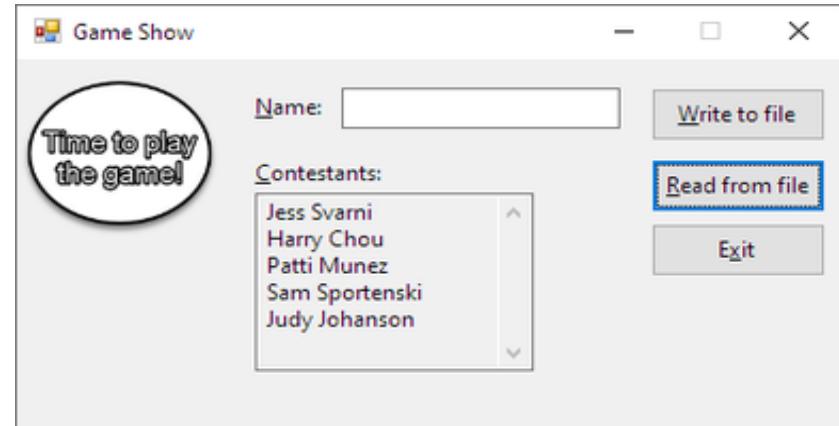
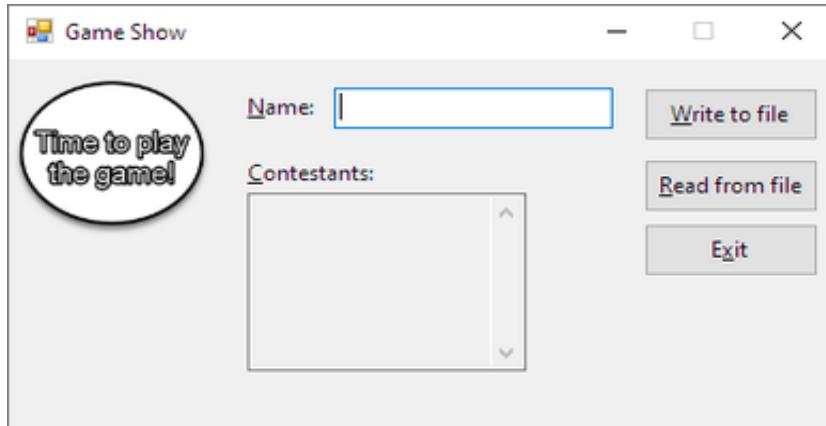
```
x      If IO.File.Exists("contestants.txt") Then  
x          ' the True path will be here - open the file for input:  
x          inFile = IO.File.OpenText("contestants.txt")
```

```
x     ' Read the file and assign to (txtContestants) control:  
x     txtContestants.Text = inFile.ReadToEnd  
x     inFile.Close()  
x Else
```

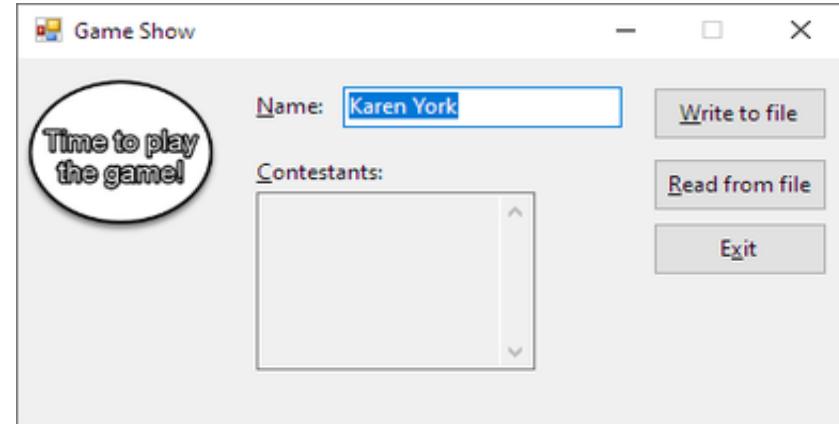
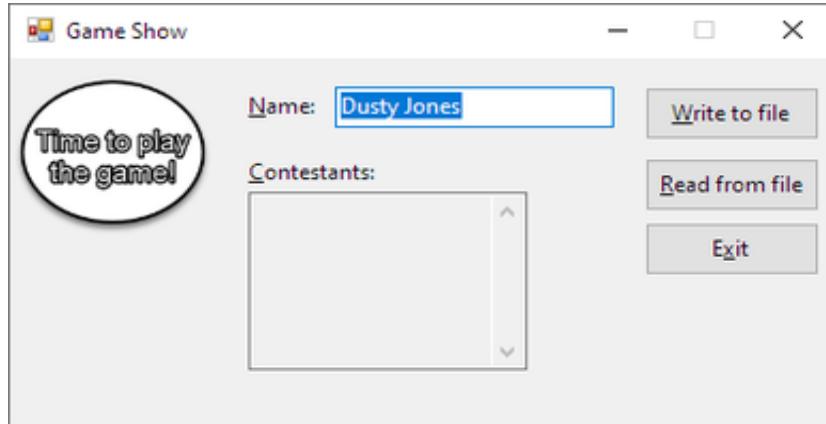
7). save the solution and then start and test the application:

-> click the button **Read from file**

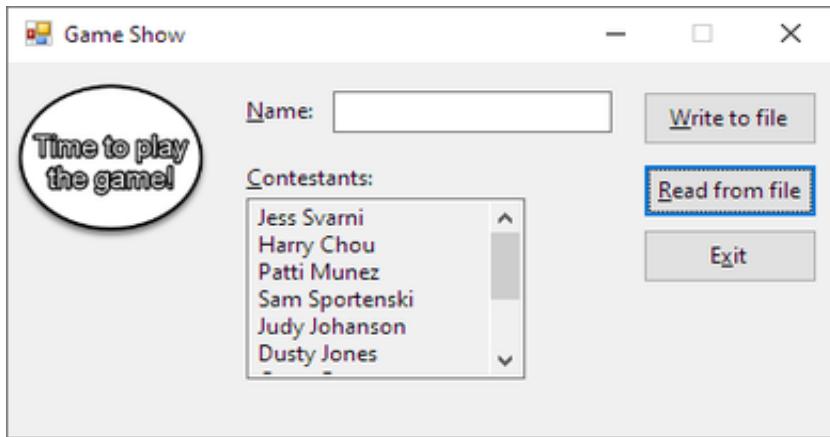
<- the 5 names contained in the **contestants.txt** appear in the **(txtContestants)** control



-> add the following 4 names to the file: **Dusty Jones, Grace Peters, Andrew Chen, Karen York**



-> click the button **Read from file** to display the **9** names in the **(txtContestants)** control:



-> click the button **Exit**

#### 8). testing the selection structure's **False** path

- as you learned in **CH4**, you should always test both paths in a selection structure
- in the next set of steps, you will change the filename in the **If** clause from **contestants.txt** to **contestant.txt** -> doing this will allow you to test the code entered in the selection structure's false path

-> in the selection structure's **If** clause, change:  
for:      **If IO.File.Exists("contestants.txt") Then**  
             **If IO.File.Exists("contestant.txt") Then**

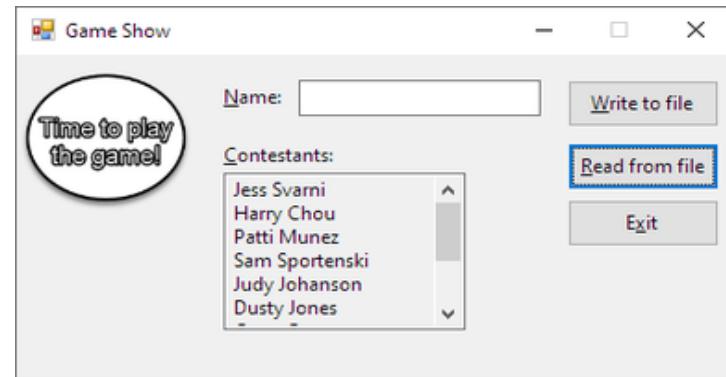
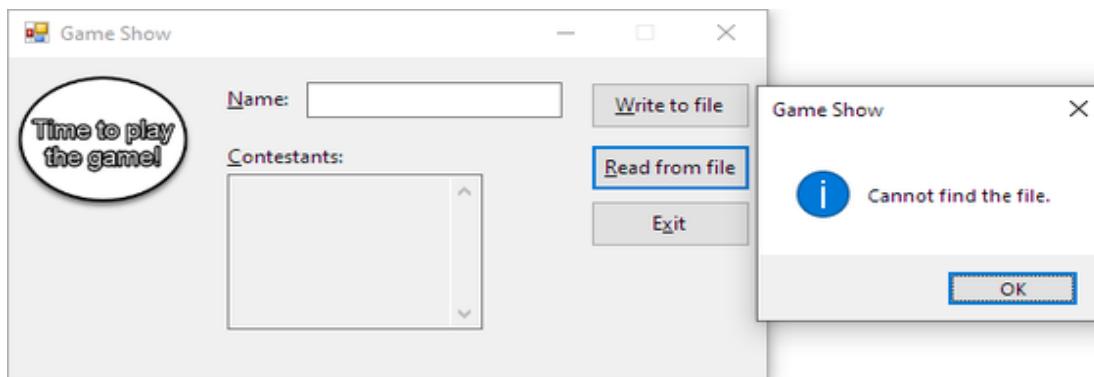
```
x      If IO.File.Exists("contestant.txt") Then
x          ' the True path will be here - open the file for input:
x          inFile = IO.File.OpenText("contestants.txt")
x          ' Read the file and assign to (txtContestants) control:
x          txtContestants.Text = inFile.ReadToEnd
x          inFile.Close()
x      Else
x          MessageBox.Show("Cannot find the file.", "Game Show", MessageBoxButtons.OK, MessageBoxIcon.Information)
x      End If
```

-> save the solution and then start and test the application:

-> click the button **Read from file**

<- because the **contestant.txt** file does not exist, the **Exists** function returns the Boolean value **False**

<- as a result, the instruction in the selection structure's **False** path displays the **Message box**



-> close the **Message box** and then click the button **Exit**

-> in the selection structure's **If** clause, change back:

for: **If IO.File.Exists("contests.txt") Then**

**If IO.File.Exists("contestants.txt") Then**

-> save the solution and then start and test the application

-> click the button **Read from file** to display the **9** names in the **(txtContestants)** control

-> click the button **Exit**, close the Code Editor window and then close the Solution

#### 9). completed code: 01.Game Show Solution-TextBox

```

1  ' Name:      Game Show Project
2  ' Purpose:    Writes names to and read names from a sequential access file.
3  ' Programmer: <your name> on <current date>
4
5  Option Explicit On
6  Option Strict On
7  Option Infer Off
8
9  Public Class frmMain
10     Private Sub btnWrite_Click(sender As Object, e As EventArgs) Handles btnWrite.Click
11         ' Writes a name to a sequential access file.
12
13         ' Declare a StreamWriter variable: (#1 step)
14         Dim outFile As IO.StreamWriter
15
16         ' Open the file for append: (#2 step)
17         outFile = IO.File.AppendText("contestants.txt")
18
19         ' Write the name on a separate line in the file: (#3 step)
20         outFile.WriteLine(txtName.Text.Trim())
21
22         ' Close the file: (#4 step)
23         outFile.Close()

```

```

24
25     ' Clear the Contestants box and then set the focus.
26     txtContestants.Text = String.Empty
27     txtName.Focus()
28 End Sub
29
30 Private Sub btnRead_Click(sender As Object, e As EventArgs) Handles btnRead.Click
31     ' Reads names from a sequential access file and displays them in the interface.
32
33     ' Declare a StreamReader variable:
34     Dim inFile As IO.StreamReader
35
36     ' Determine whether the file exists:
37     If IO.File.Exists("contestants.txt") Then
38         ' the True path will be here - open the file for input:
39         inFile = IO.File.OpenText("contestants.txt")
40         ' Read the file and assign to (txtContestants) control:
41         txtContestants.Text = inFile.ReadToEnd
42         inFile.Close()
43     Else
44         MessageBox.Show("Cannot find the file.", "Game Show", MessageBoxButtons.OK, MessageBoxIcon.Information)
45     End If
46
47 End Sub
48
49 Private Sub txtName_Enter(sender As Object, e As EventArgs) Handles txtName.Enter
50     txtName.SelectAll()
51 End Sub
52
53 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
54     Me.Close()
55 End Sub
56 End Class

```

#### CH9\_F3.2 - Sequential Access Input Files: **Class StreamReader: ReadLine** function example: code the **Game Show Application** (02.Game Show Solution-ListBox)

- in this section, you will code a slightly different version of the **Game Show application**
- this version also reads the names from the **contestants.txt** file, but it displays them in a **(lst)** rather than in a **(txt)**

- coding the **btnRead\_Click** procedure:

**1). open the: ...VB2017\Chap09\Exercise\02.Game Show Solution-ListBox\Game Show Solution.sln**

**> open the Code Editor window and notice that the most of the application has already been coded for you**

## 2). locate the **btnRead\_Click** procedure

- in order to add each name contained in the **contestants.txt** file to the **(1stContestants)**, the procedure will need to read the file, line by line

- as indicated earlier, the procedure can accomplish this task by using a loop along with the Peek and ReadLine functions

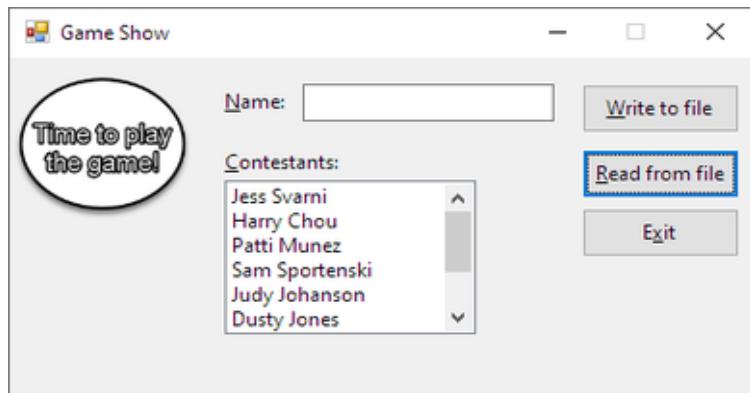
-> below the comment line: '**Process loop instructions until end of file:**' enter the following loop:

```
39      ' Determine whether the file exists:  
40      If IO.File.Exists("contestant.txt") Then  
41          ' Open the file for input:  
42          inFile = IO.File.OpenText("contestants.txt")  
43          ' Process loop instructions until end of file:  
44          Do Until inFile.Peek = -1  
45              1stContestants.Items.Add(inFile.ReadLine)  
46          Loop  
47          inFile.Close()  
48      Else  
49          MessageBox.Show("Cannot find the file.", "Game Show", MessageBoxButtons.OK, MessageBoxIcon.Information)  
50      End If
```

## 3). save the solution and then start and test the application

-> click the button **Read from file**

<- the **btnRead\_Click** procedure adds **9** names contained in the **contestants.txt** file to the **(1stContestants)** control



-> click the button **Exit**, close the Code Editor window and then close the Solution

## 4). completed code: **02.Game Show Solution-ListBox**

```
1  ' Name:        Game Show Project  
2  ' Purpose:     Writes names to and read names from a sequential access file.  
3  ' Programmer:  <your name> on <current date>  
4  
5  Option Explicit On  
6  Option Strict On  
7  Option Infer Off  
8
```

```
9  Public Class frmMain
10     Private Sub btnWrite_Click(sender As Object, e As EventArgs) Handles btnWrite.Click
11         ' Writes a name to a sequential access file.
12
13         ' Declare a StreamWriter variable.
14         Dim outFile As IO.StreamWriter
15
16         ' Open the file for append.
17         outFile = IO.File.AppendText("contestants.txt")
18
19         ' Write the name on a separate line in the file.
20         outFile.WriteLine(txtName.Text.Trim)
21
22         ' Close the file.
23         outFile.Close()
24
25         ' Clear the Contestants box and then set the focus.
26         lstContestants.Items.Clear()
27         txtName.Focus()
28     End Sub
29
30     Private Sub btnRead_Click(sender As Object, e As EventArgs) Handles btnRead.Click
31         ' Reads names from a sequential access file and displays them in the interface.
32
33         ' Declare variable.
34         Dim inFile As IO.StreamReader
35
36         ' Clear previous names from the Contestants box.
37         lstContestants.Items.Clear()
38
39         ' Determine whether the file exists:
40         If IO.File.Exists("contestants.txt") Then
41             ' Open the file for input:
42             inFile = IO.File.OpenText("contestants.txt")
43             ' Process loop instructions until end of file:
44             Do Until inFile.Peek = -1
45                 lstContestants.Items.Add(inFile.ReadLine)
46             Loop
47             inFile.Close()
48         Else
49             MessageBox.Show("Cannot find the file.", "Game Show", MessageBoxButtons.OK, MessageBoxIcon.Information)
50         End If
51     End Sub
52
```

```

53     Private Sub txtName_Enter(sender As Object, e As EventArgs) Handles txtName.Enter
54         txtName.SelectAll()
55     End Sub
56
57     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
58         Me.Close()
59     End Sub
60 End Class

```

### Mini-Quiz 9-2:

1. Write a declaration statement for a procedure-level variable named **inInventory**. The variable will be used to read data from a sequential access file.
2. Write an **If** clause that determines whether the **inventory.txt** file exists.
3. Using the variable from **Q1**, write a statement to open a sequential access file named **inventory.txt** for input.
4. Write a statement that reads a line of text file from the file associated with the variable from **Q1**. Assign the line of text to the **strProduct** variable.
5. Write a **Do** clause that processes the loop instructions until the end of the file is reached. The file is associated with the variable from **Q1**.
6. Write a statement to close the file associated with the variable from **Q1**.
7. Write a statement that reads the entire file associated with the **inFile** variable and assign the file's contents to the **txtDocument** control.

```

1... Dim inInventory As IO.StreamReader
2... If IO.File.Exists("inventory.txt") Then
3...     inInventory = IO.File.OpenText("inventory.txt")
4...     strProduct = inInventory.ReadLine
5... Do Until inInventory.Peek = -1; Do While inInventory.Peek <> -1
6...     inInventory.Close()
7...     txtDocument.Text = inFile.ReadToEnd

```

### CH9\_F4 - You Do It 1: Sequential Access Output & Input file exercise: 03.You Do It 1 Solution

-> AppendText, WriteLine, OpenText, ReadToEnd, Peek, ReadLine, Close, \_MouseClick

1. create an application named **You Do It 1** and save it in the ...VB2017\Chap09\Exercise\03.You Do It 1 Solution
2. add: **3x text box**, and **3x button** to the form
  - the user will enter a number in the **1st text box** and then click the **1st button**
  - set the **2nd** and the **3rd** text boxes: **Multiline = True**, **ReadOnly = True**, **ScrollBars = Vertical**.
3. the **1st button**'s Click event procedure should write the number on a separate line in a sequential access file
4. the **2nd button**'s Click event procedure should read the entire sequential access file all at once and display the file's contents in the **2nd text box**
5. the **3rd button**'s Click event procedure should read the sequential access file, line by line, and display the file's contents in the **3rd text box**
  - display each number on a separate line
6. code the procedure, save the solution and then start and test the application:
  - use the **1st button** to write any six numbers to the file and then:
    - use the **2nd** and **3rd buttons** to display the contents of the file in their respective text boxes
7. close the solution

## my GUI & code:

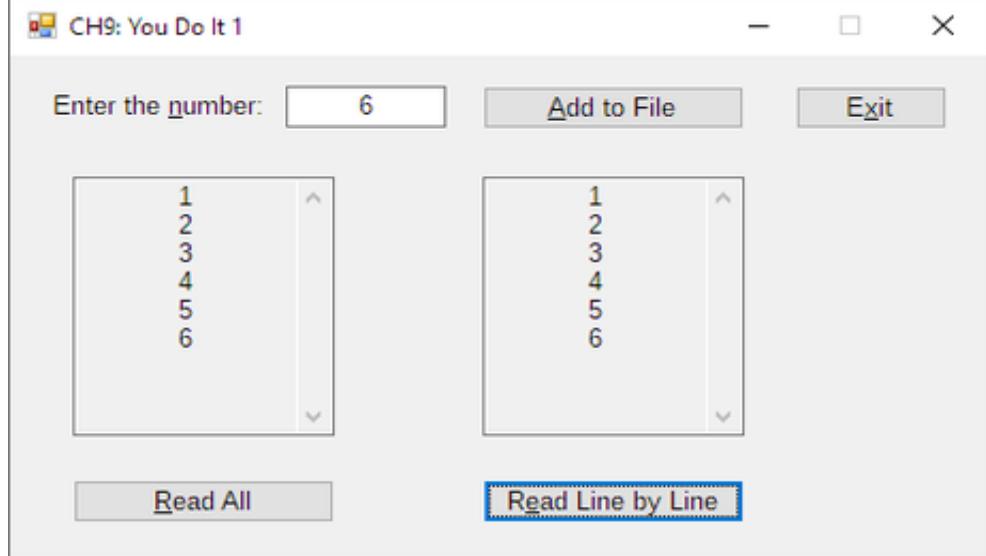
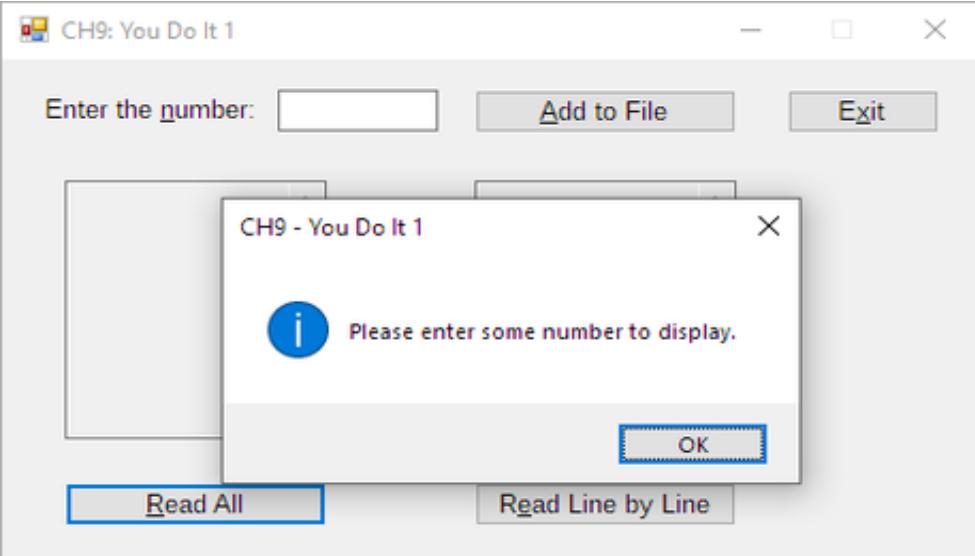
```
1  Option Explicit On
2  Option Strict On
3  Option Infer Off
4  ' allow the user to enter an integer numbers, who are then written into a text file: Sequential Output File
5  ' display the text file content: 1). read the file all at once: Sequential Input File
6  '           2). read the file line by line: Sequential Input File
7  ' btn1Add: txt1AddNumber = intNumbers
8  ' btn2ReadAll: txt2ReadAll.Text
9  ' btn3ReadLineByLine: txt3ReadLineByLine.Text
10 ' MY EXTRA: - when the app exits, it erases all the content of the Sequential Output File
11 '             - app displays the message, if there is no entry in the Sequential Output File and user wants to display the content
12 '             - when user click the left mouse in the input text box, the app selects all the previously entered content
13 '             - clear the output text boxes when user entry changes
14 Public Class frmMain
15     Private outFile As IO.StreamWriter
16     Private inFile As IO.StreamReader
17     Private Sub btn1Add_Click(sender As Object, e As EventArgs) Handles btn1Add.Click
18         txt1AddNumber.Focus()
19         txt1AddNumber.SelectAll()
20
21         ' Sequential Access Output:
22         Dim intNumbers As Integer
23         Integer.TryParse(txt1AddNumber.Text, intNumbers)
24         outFile = IO.File.AppendText("numbers.txt")
25         outFile.WriteLine(intNumbers)
26         outFile.Close()
27     End Sub
28
29     Private Sub btn2ReadAll_Click(sender As Object, e As EventArgs) Handles btn2ReadAll.Click
30         Empty()
31         ' Sequential Access Input: Read entire file all at once:
32         If IO.File.Exists("numbers.txt") Then
33             inFile = IO.File.OpenText("numbers.txt")
34             txt2ReadAll.Text = inFile.ReadToEnd()
35             inFile.Close()
36         Else
37             MessageBox.Show("Cannot find the file.", "CH9 - You Do It 1", MessageBoxButtons.OK, MessageBoxIcon.Information)
38         End If
39     End Sub
40
41     Private Sub btn3ReadLineByLine_Click(sender As Object, e As EventArgs) Handles btn3ReadLineByLine.Click
42         Empty()
43         ' Sequential Access Input: Read the file line by line:
```

```
44     txt3ReadLineByLine.Text = Nothing
45     If IO.File.Exists("numbers.txt") Then
46         inFile = IO.File.OpenText("numbers.txt")
47         Do Until inFile.Peek = -1
48             txt3ReadLineByLine.Text += inFile.ReadLine & ControlChars.NewLine
49         Loop
50         inFile.Close()
51     Else
52         MessageBox.Show("Cannot find the file.", "CH9 - You Do It 1", MessageBoxButtons.OK, MessageBoxIcon.Information)
53     End If
54 End Sub
55
56 Private Sub Empty()
57     ' in a case the file has no characters at all:
58     inFile = IO.File.OpenText("numbers.txt")
59     If inFile.ReadToEnd = String.Empty Then
60         MessageBox.Show("Please enter some number to display.", "CH9 - You Do It 1", MessageBoxButtons.OK, MessageBoxIcon.Information)
61     End If
62     inFile.Close()
63 End Sub
64
65 Private Sub txt1AddNumber_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txt1AddNumber.KeyPress
66     ' accept only Numbers and Backspace key
67     If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
68         e.Handled = True
69     End If
70 End Sub
71
72 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
73     ' erases all content when exit:
74     If IO.File.Exists("numbers.txt") Then
75         outFile = IO.File.CreateText("numbers.txt")
76     End If
77     Me.Close()
78 End Sub
79
80 Private Sub txt1AddNumber_MouseClick(sender As Object, e As MouseEventArgs) Handles txt1AddNumber.MouseClick
81     ' when you click the left mouse in entry text box:
82     txt1AddNumber.SelectAll()
83 End Sub
84
85 Private Sub txt1AddNumber_TextChanged(sender As Object, e As EventArgs) Handles txt1AddNumber.TextChanged
86     ' clear the text boxes when user entry changes:
87     txt2ReadAll.Text = Nothing
```

```

88      txt3ReadLineByLine.Text = Nothing
89  End Sub
90 End Class

```

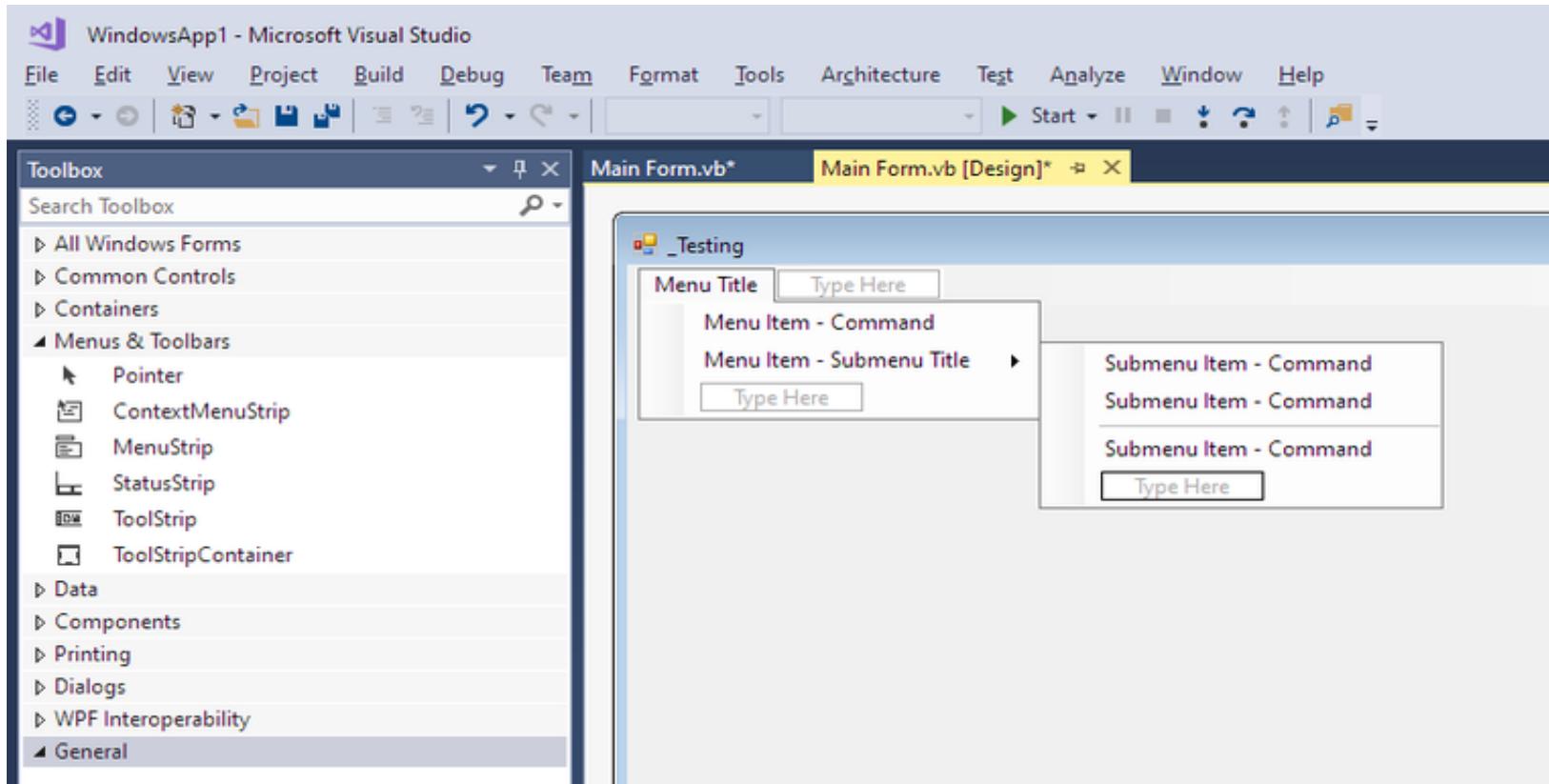


## CH9\_APPLY THE CONCEPTS LESSON

### CH9\_A1 - Add a Menu(mnu) to a Form: Toolbox / Menus & Toolbars / ToolStrip control

- Designer window's **Toolbox / Menus & Toolbars** section contains: **Pointer, ContextMenuStrip, ToolStrip, StatusStrip, ToolStrip, ToolStripContainer Classes**
- the **MenuStrip** tool:
  - used for instantiating a **Menu Strip Control**
  - the control allows you to add 1 or more **menus** to a Windows form
  - each **menu** contains a **menu title**, which appears on the menu bar at the top of the form
  - when you click a **menu title**, its corresponding menu opens and displays a list of options, called **menu items**
  - the **menu items** can be:
    - **commands** - such as Open, Save, Exit...etc
    - **separator bars** - used to visually group together related items on a menu or submenu
    - **submenu titles**
  - as in all Windows applications:
    - clicking a **command** on a menu executes the command
    - clicking a **submenu** title opens an additional menu of options, referred to as a **submenu items**
  - although you can create many levels of **submenus**, it is best to **use only one level** in your application because including too many layers of **submenus** can **confuse** the user
  - each **menu element** is considered an object, and each has a set of properties associated with it
  - the most commonly used **properties** for a **menu element** are:
 

|               |                                                          |
|---------------|----------------------------------------------------------|
| <b>(Name)</b> | = used to refer to the menu element in code              |
|               | - 3-char ID is <b>(mnu)</b>                              |
| <b>Text</b>   | = stores the menu element's caption = text the user sees |
|               | - indicates the purpose of the menu element              |
|               | - e.g.: <b>Edit, Save As, Copy, Exit...</b>              |



#### CH9\_A1.1 - GUI guidelines for Menus(mnu):

- 1). **menu Title:**
  - caption should be a single word with only **1st** letter capitalized
  - caption should have a unique access / Alt key

e.g. File  
e.g. File
- 2). **menu Item:** = **Submenu**
  - unlike the captions for **Menu Titles**, it typically consist of **1 - 3** words using book title capitalization
  - caption should have a unique access / Alt key within its menu
  - if a **menu Item** requires additional information from the user:
    - place 3x period ... at the end of the item's caption, or
    - place 3x period within an ellipsis (...) at the end of the item's caption

e.g. Enter the Dragon  
e.g. Enter the Dragon
- 3). the **Menus** included in your application should follow the standard Windows conventions:
  - e.g. if your application uses a **File menu**, it should be the **1st** menu on the menu bar
  - **File menus** typically contain commands for **Opening**, **Saving**, and **Printing** files, as well as **Exiting** the application
  - commonly used **menu items** should be assigned **shortcut keys**:
    - appearing to the right of a menu item
    - allowing the user to select the item without opening the menu

e.g. File / Save Ctrl+S = Save command on a File menu  
e.g. Edit / Copy Ctrl+C = Copy command on an Edit menu
- use a **separator bar** to separate groups of related menu items

## CH9\_A2 - Menu example: Continents Application - create Sequential Access File (.txt) to fill **Array** with values (04.Continents Solution) 1/4

- the **Continents Application** that you coded in **CH8: ...VB2017\Chap08\Exercise09.Continents Solution** stored the names of the 7 continents in an **Array**
- the application's GUI provided 3 buttons: adding the name to a list box - in either ascending or descending order, and exiting the application
- in this section, you will code a different version of the application:
  - instead of initializing the **Array** in its declaration statement, the **Array** will get its values from a sequential access file named **continents.txt**
  - sequential access files are often used to fill **Arrays** with values
  - in addition, the GUI in this chapter will use **Menus** rather than **buttons**
  - instead of using a **StreamWriter** object and the **WriteLine** method to create the **continents.txt** file, you can use the **New File** option on the **File** menu

- creating the **continents.txt** file:

### 1). open the: ...VB2017\Chap09\Exercise04.Continents Solution\Continents Solution.sln

- > on the Visual Studio Menu bar click **File / New File...** and from the left tab **General** choose **Text File** located on the right and click the button **Open**
- <- an empty file appears in the new tab window named: **TextFile1.txt**
- > on the Visual Studio Menu bar click **File / Save TextFile1.txt As...**
- > open the Continents Project's folder **bin\Debug** and change the name in the **File name:** box to **continents** and then click the button **Save**
- > into the file enter the 7 continent names:

The screenshot shows a Microsoft Notepad window with the title bar 'continents.txt'. The content of the window is as follows:

```
1 North America
2 Africa
3 South America
4 Antarctica
5 Australia
6 Asia
7 Europe
8
```

- > save the solution and then close the **continents.txt** tab window

## CH9\_A2.1 - Menu example: Continents Application - add **Menus** to the GUI (04.Continents Solution) 2/4

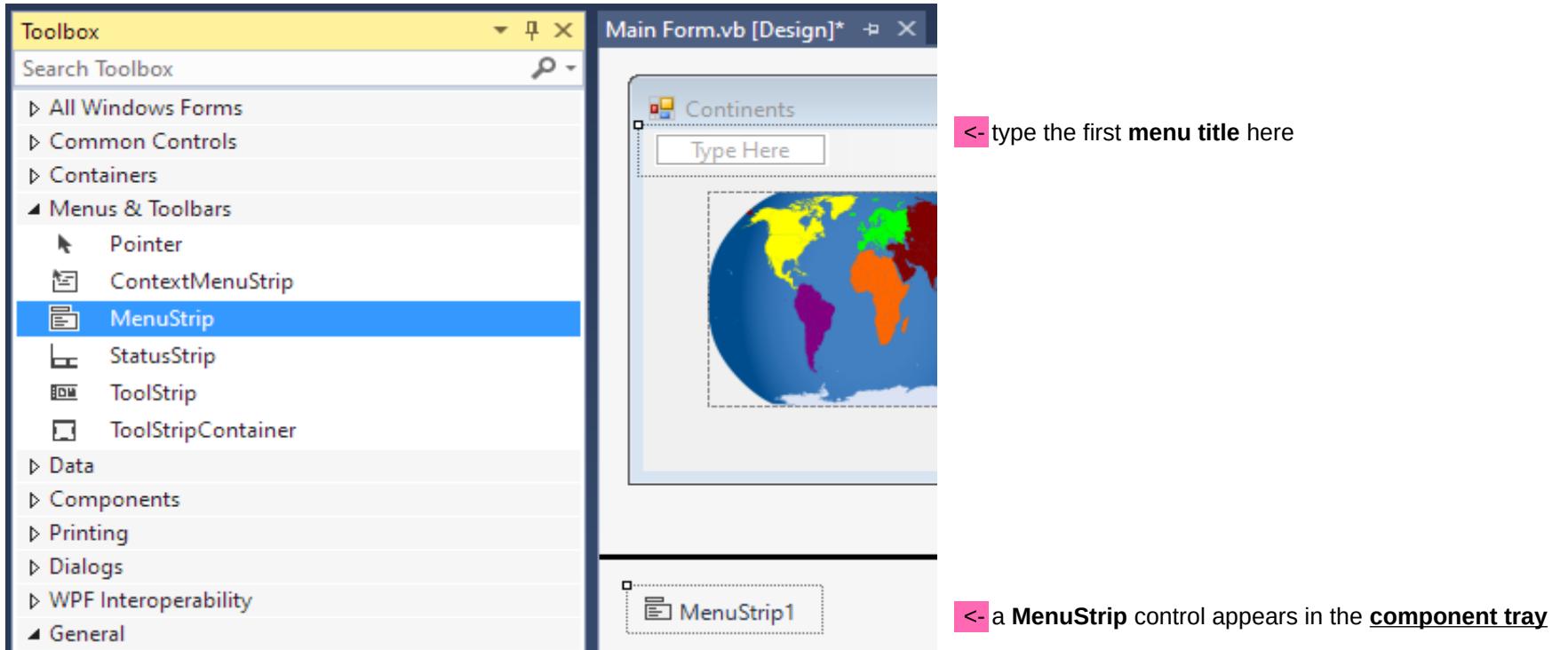
- in the next set of steps, you will add **2 menus** to the form: **File menu** and **Sort menu**:
- the **File menu** will contain an **Exit** option
- the **Sort menu** will contain **2 options: Ascending and Descending**

- adding **2 menus** to the form:

### 1). open the: ...VB2017\Chap09\Exercise04.Continents Solution\Continents Solution.sln

### 2). in the next set of steps, you will add **2 menus** to the form: **File menu** and **Sort menu**:

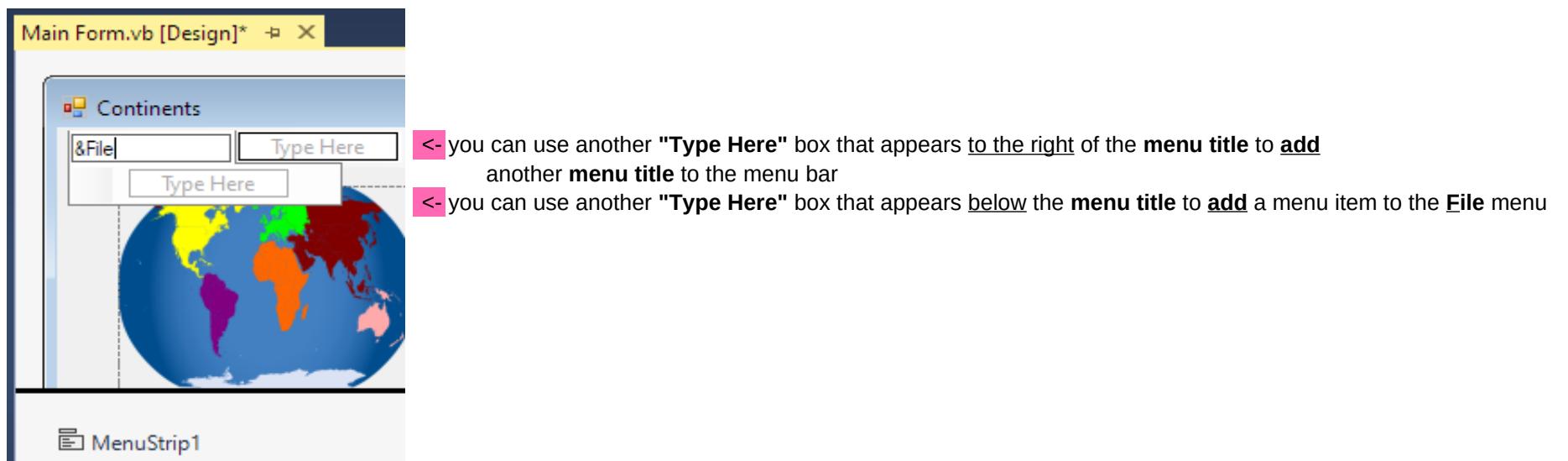
- > expand **Designer** window's **Toolbox / Menus & Toolbars** node, click the **MenuStrip** tool and drag your mouse pointer to the form and then release the mouse
  - don't worry about the exact location
- <- the **MenuStrip1** control appears in the **component tray** at the bottom of the IDE, and the words: "**Type Here**" appear in a box below the form's title bar:



-> on the **menu bar** click the "Type Here" box and then type: **&File**

<- you can use another "Type Here" box that appears below the **menu title** to add a menu item to the **File** menu

<- you can use another "Type Here" box that appears to the right of the **menu title** to add another **menu title** to the menu bar



-> press **Enter** to confirm the name, click the **File** menu title to choose it and then you will set the **Properties**:

-> scroll the **Properties** window until you see the **Text** property, which contains: **&File**

-> now, scroll to the top, click the **(Name)** property and enter **mnuFile** for reference in your code

<- the three character ID for menus is **mnu**

3). now, you will make a **menu item** of the **File** menu, the command **Exit**:

-> click the "Type Here" box that appears below the **File** menu title, type: **E&xit** and then press **Enter**

-> change its **(Name)** property to **mnuFileExit**

4). next, you will add a **Sort** menu to the form:

-> click the "Type Here" box that appears to the right of the **File** menu title, type: **&Sort** and then press **Enter**

-> change its **(Name)** property to **mnuSort**

5). the **Sort** menu will have 2 options: **Ascending** order and **Descending** order and in addition to their access / Alt keys, these options will also have shortcut keys

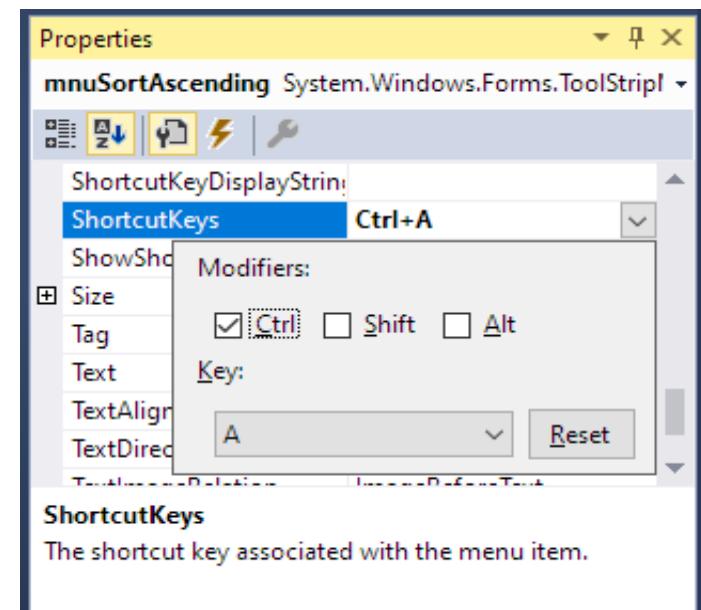
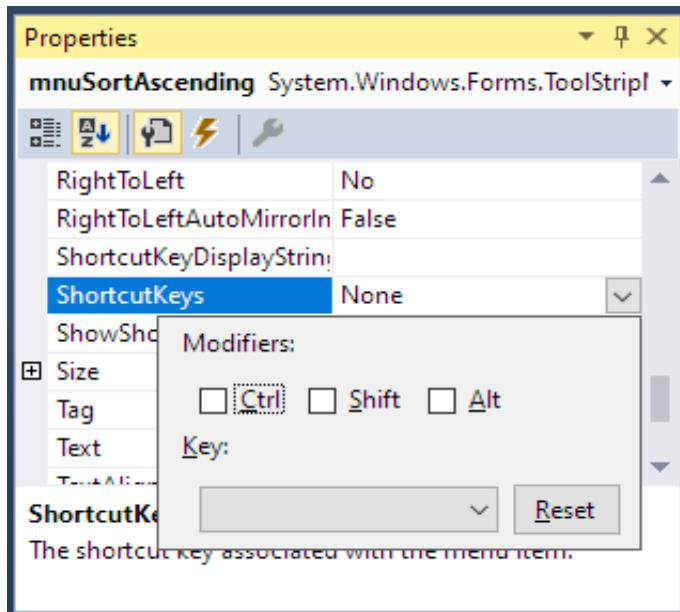
-> click the "Type Here" box that appears below the **Sort** menu title, type: **&Ascending** and then press **Enter**

-> change its **(Name)** property to **mnuSortAscending**

- now, you will add the shortcut key to the menu item / command **Ascending**:

-> click the **ShortcutKeys** in the **Properties** list and then click the **list arrow** in the Settings box

<- a box opens and allows you to specify a modifier and a key:



<- in this case, the modifier and key will be **Ctrl** and **A**

-> click the **Ctrl** check box to select it, and then click the **list arrow** that appears in the **Key:** combo box to select the **A** key and then press **Enter**

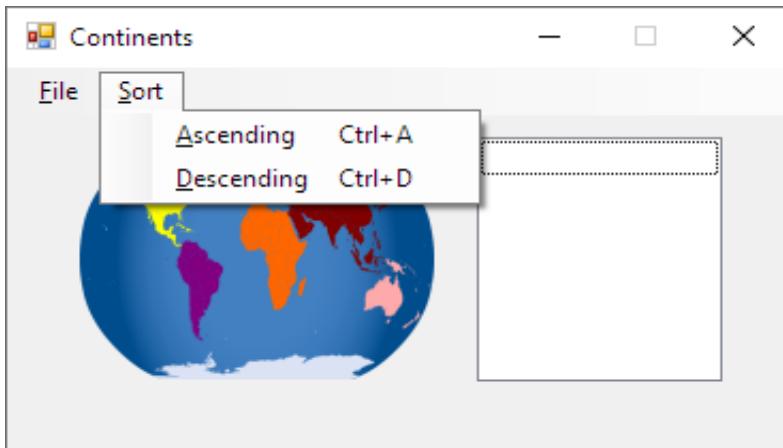
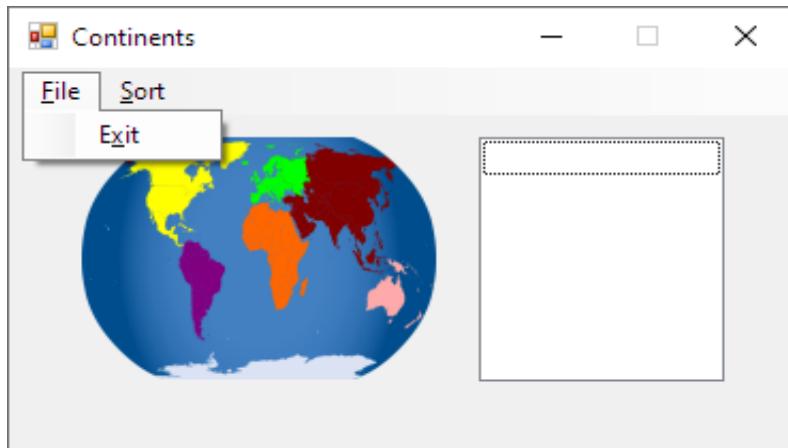
<- your shortcut key combination **Ctrl+A** appears: - in the **ShortcutKeys** property and also

- to the right of the **Ascending** menu item

-> on your own, add the **Descending** menu item to the **Sort** menu, name it **mnuSortDescending** with the **Ctrl+D** as the **ShortcutKey**

6). test your menus:

- > click the form's title bar **Continents** to close the menu, save the solution and then start the application
- > click **File** on the menu bar -> the menu opens and offers an **Exit** option
- > move your mouse pointer to the **Sort** menu title, which displays 2 options: **Ascending Ctrl+A** and **Descending Ctrl+D**



- > click the form's **Close** button to end the application

**CH9\_A2.2 - Menu example: Continents Application - code the Items on a Menu (04.Continents Solution) 3/4**

- now, that you have completed the GUI for the **Continents Application**, you can begin coding it

1). open the: ...VB2017\Chap09\Exercise\04.Continents Solution\Continents Solution.sln

- > open the Code Editor window and locate the form class's declaration section
- <- notice that the **declaration statements** and **Sub procedure** has already been entered for you:

```
9  Public Class frmMain
10     ' Class-level array and variable:
11     Private strContinents(6) As String
12     Private intLastSub As Integer = strContinents.GetUpperBound(0)
13
14     Private Sub AddToListBox()
15         ' Add array values to list box and select first value.
16
17         lstContinents.Items.Clear()
18         For intSub As Integer = 0 To intLastSub
19             lstContinents.Items.Add(strContinents(intSub))
20         Next intSub
21         lstContinents.SelectedIndex = 0
22     End Sub
```

<- declare a 7-element, class-level **Array** named **strContinents**  
<- declare a class-level variable named **intLastSub** and initializes it to the highest subscript in the **Array**  
<- **Independent Sub procedure** named **AddToListBox()**  
 - more about **Independent Sub procedures** can be found in: **CH6\_F2**  
 - when it is invoked, the procedure will:  
<- clear the **Items** from the (**lstContinents**) control  
<- add each **Array** value to the control  
  
<- select the 1st value in the list

2). locate the **frmMain\_Load** procedure

- the procedure will:
  - 1... fill the **Array** with the values stored in the **continents.txt** file that you created earlier, then
  - 2... call the **AddToListBox()** Independent Sub procedure to add the names to the **(lstContinents)** control

-> enter the lines of code indicated:

```
24      Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
25          ' Fill array with values and then add values to list box.
26
27          ' Declare StreamReader variable:
28          Dim inFile As IO.StreamReader
29
30          ' Fill array with values from a sequential access file:
31          If IO.File.Exists("continents.txt") Then
32              inFile = IO.File.OpenText("continents.txt")
33              For intSub As Integer = 0 To intLastSub
34                  strContinents(intSub) = inFile.ReadLine
35              Next intSub
36              inFile.Close()
37              ' Add array values to list box:
38              AddToListBox()
39          Else
40              MessageBox.Show("Cannot find the file.", "Continents", MessageBoxButtons.OK, MessageBoxIcon.Information)
41          End If
42      End Sub
43
```

3). save the solution and start the application:

- the **frmMain\_Load** procedure adds the 7 names to the list box

<- notice that the names appear in the same order as they do in both the **continents.txt** file and the **strContinents()** **Array**



-> click the form's **Close** button to end the application

4). next, you will code and test the **Exit** option on the **File** menu

-> open the code template for the **mnuFileExit\_Click** procedure and type: **Me.Close()**

```
44     Private Sub mnuFileExit_Click(sender As Object, e As EventArgs) Handles mnuFileExit.Click  
45         Me.Close()  
46     End Sub  
47 End Class
```

-> save the solution and then start and test the application: click **File** on the application's menu bar and then click **Exit** to end the application

5). next, you will code the **Ascending** option on the **Sort** menu

-> open the code template for the **mnuSortAscending\_Click** procedure

- the procedure will sort the **Array** values in **ascending** order and then call the **AddToListBox()** procedure to add the values to the **(1stContinents)** control

-> enter the lines of code indicated:

```
47  
48     Private Sub mnuSortAscending_Click(sender As Object, e As EventArgs) Handles mnuSortAscending.Click  
49         ' Adds the array values in ascending order to the list box.  
50  
51         Array.Sort(strContinents)  
52         AddToListBox()  
53     End Sub  
54 End Class
```

6). next, you will code the **Descending** option on the **Sort** menu

-> open the code template for the **mnuSortDescending\_Click** procedure

- the procedure will sort the **Array** values in **ascending** order, **reverse** the values and then call the **AddToListBox()** procedure to add the values to the **(1stContinents)** control

-> enter the lines of code indicated:

```
54  
55     Private Sub mnuSortDescending_Click(sender As Object, e As EventArgs) Handles mnuSortDescending.Click  
56         ' Adds the array values in descending order to the list box.  
57  
58         Array.Sort(strContinents)  
59         Array.Reverse(strContinents)  
60         AddToListBox()  
61     End Sub  
62 End Class
```

7). save the solution and then start and test the application

-> click **Sort** on the menu bar and then click **Descending**

<- the continents names appear in descending order

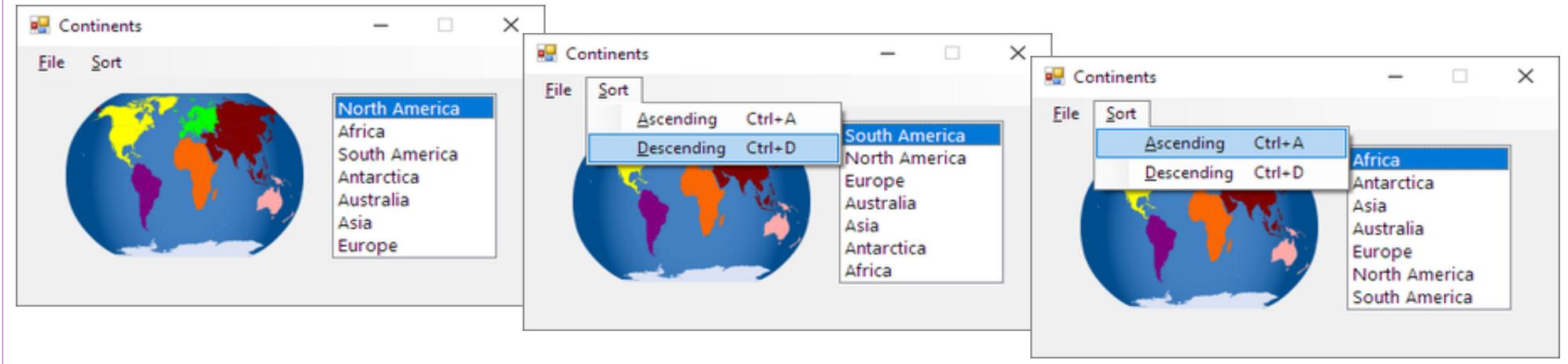
-> click **Sort** on the menu bar and then click **Ascending** to display the names in ascending order

<- notice that once you select an option from the **Sort** menu, GUI does not provide a way to return the names to their original order ->

-> you will fix this in the next section:

8). now, you will verify that the **shortcut keys** work correctly:

- > press **Ctrl+d** to sort the names in **descending** order
- > press **Ctrl+a** to sort the names in **ascending** order
- > press **Alt+f** to open the **File** menu and then tap the letter **x** to select the **Exit** option, which ends the application



#### CH9\_A2.3 - Menu example: Continents Application - modify a Menu (04.Continents Solution) 4/4

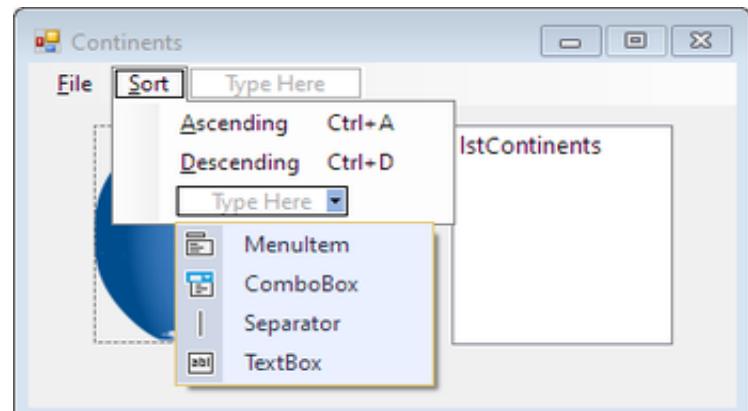
- in this section, you will **add** another option to the **Sort** menu
- the option will allow the user to display the continent names in their original order, which is the order they were entered in the sequential access file **continents.txt**

- modify a **menu**:

1). open the: ...VB2017\Chap09\Exercise\04.Continents Solution\Continents Solution.sln

2). add the separator bar and submenu item to the **Sort** menu:

- > click the **Main Form.vb [Design]** tab to return to the **designer window**
- > click **Sort**, place your mouse pointer on the "Type Here" box **below** **Descending** option, and then click the **list arrow** in the box to expand **Drop-down list**



- > click **Separator** to add a separator bar - horizontal line below the **Descending** option
- > click the "Type Here" box **below** the separator bar, type: &Original Order and press **Enter**
- > change the item's properties: (**Name**) to **mnuSortOriginal** and Shortcut key to **Ctrl+O**
- > click the form's title bar to close the menu

3). coding the **Original Order** submenu item:

- > click the **Main Form.vb** tab to return to the **Code Editor** window
- the **mnuSortOriginal\_Click** procedure will need to perform the same tasks as the **frmMain\_Load** procedure performs
- rather than having the same code in both procedures, you will enter the code in an **Independent Sub procedure** that both procedures can invoke when needed
- > click the blank line above the **Private Sub AddToListBox()** procedure header, press **Enter**, type the following procedure header and then press **Enter**

```
13  
14     Private Sub FillArrayAndListBox()  
15  
16 End Sub
```

- > locate the **frmMain\_Load** procedure, select all of the lines between the procedure header and procedure footer and press **Ctrl+x** to cut the lines
- > click the blank line below the **FillArrayListBox()** procedure header and press **Ctrl+v** to paste the lines in the procedure:

```
14     Private Sub FillArrayAndListBox()  
15         ' Fill array with values and then add values to list box.  
16  
17         ' Declare StreamReader variable:  
18         Dim inFile As IO.StreamReader  
19  
20         ' Fill array with values from a sequential access file:  
21         If IO.File.Exists("continents.txt") Then  
22             inFile = IO.File.OpenText("continents.txt")  
23             For intSub As Integer = 0 To intLastSub  
24                 strContinents(intSub) = inFile.ReadLine  
25             Next intSub  
26             inFile.Close()  
27             ' Add array values to list box:  
28             AddToListBox()  
29         Else  
30             MessageBox.Show("Cannot find the file.", "Continents", MessageBoxButtons.OK, MessageBoxIcon.Information)  
31         End If  
32     End Sub
```

- > now, click the blank line below the **frmMain\_Load** procedure header and enter the following lines of code:

```
44     Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load  
45         ' Use a Sub procedure to fill array and list box.  
46         FillArrayAndListBox()  
47     End Sub
```

- > open the code template for the **mnuSortOriginal\_Click** procedure and enter the following lines of code:

```
68     Private Sub mnuSortOriginal_Click(sender As Object, e As EventArgs) Handles mnuSortOriginal.Click  
69         ' Adds the array values in original order to the list box.  
70         FillArrayAndListBox()  
71     End Sub  
72 End Class
```

4). save the solution and then start and test the application:

-> click **Sort** and then click **Descending**

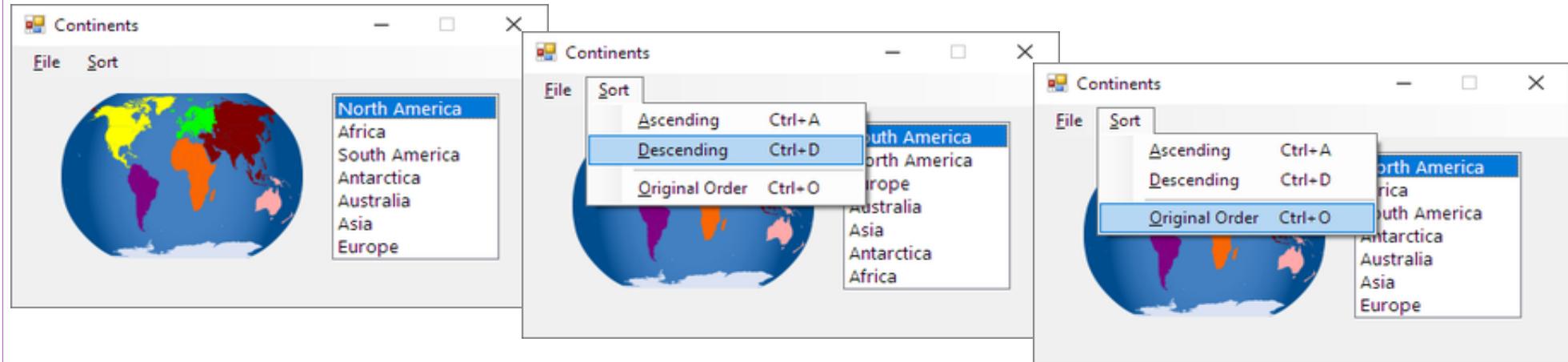
<- the continent names appear in descending order

-> click **Sort** and then click **Original Order**

<- the continent names appear in original order

-> test your shortcuts **Ctrl+A**, **Ctrl+D**, **Ctrl+O**

-> when finished testing, click **File** and then click **Exit**, close the Code Editor window and then close the solution.



5). completed code: 04.Continents Solution

```
1  ' Name:      Continents Project
2  ' Purpose:    Displays the names of the continents in either ascending, descending, or original order.
3  ' Programmer: <your name> on <current date>
4
5  Option Explicit On
6  Option Strict On
7  Option Infer Off
8
9  Public Class frmMain
10   ' Class-level array and variable:
11   Private strContinents(6) As String
12   Private intLastSub As Integer = strContinents.GetUpperBound(0)
13
14   Private Sub FillArrayAndListBox()
15     ' Fill array with values and then add values to list box.
16
17     ' Declare StreamReader variable:
18     Dim inFile As IO.StreamReader
19
20     ' Fill array with values from a sequential access file:
21     If IO.File.Exists("continents.txt") Then
22       inFile = IO.File.OpenText("continents.txt")
```

```
23     For intSub As Integer = 0 To intLastSub
24         strContinents(intSub) = inFile.ReadLine
25     Next intSub
26     inFile.Close()
27     ' Add array values to list box:
28     AddToListBox()
29 Else
30     MessageBox.Show("Cannot find the file.", "Continents", MessageBoxButtons.OK, MessageBoxIcon.Information)
31 End If
32 End Sub
33
34 Private Sub AddToListBox()
35     ' Add array values to list box and select first value.
36
37     lstContinents.Items.Clear()
38     For intSub As Integer = 0 To intLastSub
39         lstContinents.Items.Add(strContinents(intSub))
40     Next intSub
41     lstContinents.SelectedIndex = 0
42 End Sub
43
44 Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
45     ' Use a Sub procedure to fill array and list box.
46     FillArrayAndListBox()
47 End Sub
48
49 Private Sub mnuFileExit_Click(sender As Object, e As EventArgs) Handles mnuFileExit.Click
50     Me.Close()
51 End Sub
52
53 Private Sub mnuSortAscending_Click(sender As Object, e As EventArgs) Handles mnuSortAscending.Click
54     ' Adds the array values in ascending order to the list box.
55
56     Array.Sort(strContinents)
57     AddToListBox()
58 End Sub
59
60 Private Sub mnuSortDescending_Click(sender As Object, e As EventArgs) Handles mnuSortDescending.Click
61     ' Adds the array values in descending order to the list box.
62
63     Array.Sort(strContinents)
64     Array.Reverse(strContinents)
65     AddToListBox()
66 End Sub
```

```

67
68     Private Sub mnuSortOriginal_Click(sender As Object, e As EventArgs) Handles mnuSortOriginal.Click
69         ' Adds the array values in the original order to the list box.
70         FillArrayAndListBox()
71     End Sub
72 End Class

```

### CH9\_A3 - Accumulate the Values Stored in a File example: 05.Harkins Solution

- Harkins Company stores its annual sales information in a sequential access file named **sales.txt**:

```

sales.txt  X  Main Form.vb [Design]
1 Beverage           <--- product category &
2 340500            <--- its sales amount
3 Food
4 540200
5 Packaged coffee
6 437000
7 Packaged tea
8 124000
9 Other
10 235700
11

```

- the file contains the current year's sales for each of **5** different product categories:
  - Beverage, Food, Packed coffee, Packed tea, and Other
- <- notice that each category and its corresponding sales amount appear on separate lines
- **The Harkins Company application**, which you code in the next set of steps, accumulates the annual sales amounts stored in the file and then displays the total annual sales in a label

- finish coding **The Harkins Company application**:

1). open the: ...VB2017\Chap09\Exercise\05.Harkins Solution\Harkins Solution.sln

2). open the Code Editor window and locate the **btnCalc\_Click** procedure

- the procedure declares **4** variables:

```

14     Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
15         ' Calculate and display the annual sales amount.
16
17         ' Declare variables:
18         Dim inFile As IO.StreamReader
19         Dim strCategory As String
20         Dim intSales As Integer
21         Dim intAnnualSales As Integer
22

```

<- will store the **StreamReader** object that represents the **sales.txt** file  
 <- will store the **product categories** read from the **inFile**  
 <- will store the **sales amounts** read from the **inFile**  
 <- will be used by the procedure to **accumulate** the sales **amounts**

3). first, the procedure needs to determine whether the **sales.txt** file exists in the project's **bin\Debug** folder, so:

-> modify the **If** clause as follows:

```
23     If IO.File.Exists("sales.txt") Then
```

4). if the file exists, the selection structure's **True** path should open the file for **input**, so:

-> click the blank line below the **If** clause and type the following assignment statement:

```
24 inFile = IO.File.OpenText("sales.txt")
```

5). the procedure needs to **read** the file **line by line**, stopping only when the **end** of the file is reached, so:

-> type the following **Do** clause:

```
25 Do Until inFile.Peek = -1
```

6). the **first** instruction in the **loop** will use the **ReadLine** function to **read the product category** from the **inFile** and then store it in the **strCategory** variable, so:

-> enter the following comment and assignment statement:

```
26 ' Read product category:
```

```
27 strCategory = inFile.ReadLine
```

<- every **odd** line in **sales.txt** = product name

7). the **second** instruction in the **loop** will use the **ReadLine** function/method to **read the sales amount** from the **inFile**,

and then use the **Integer.TryParse** method to convert the amount to the **Integer** data type

<- recall that the **ReadLine** method returns a **String**

-> enter the following comment and statement:

```
28 ' Read sales amount and convert to Integer:
```

```
29 Integer.TryParse(inFile.ReadLine, intSales)
```

<- every **even** line in **sales.txt** = number, sales amount

8). the **third** instruction in the **loop** will add the sales amount stored in **intSales** to the accumulator variable **intAnnualSales**, so:

-> enter the following comment and statement:

```
30 ' Add sales amount to accumulator:
```

```
31 intAnnualSales += intSales
```

```
32 Loop
```

9). after the **loop** has finished processing, the procedure needs to close the **inFile** and then display the annual sales in the **lblAnnualSales** control, so:

-> below the **Loop** clause enter the additional statements indicated:

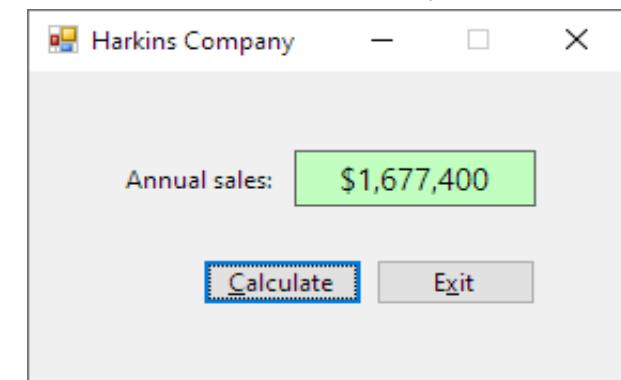
```
33 inFile.Close()
```

```
34 lblAnnualSales.Text = intAnnualSales.ToString("C0")
```

10). save the solution and then start and test the application:

-> click the **Calculate** button to see the company's annual sales amount **\$1,677.400**

-> click the **Exit** button, close the Code Editor window and then close the solution.



## 11). completed code: 05.Harkins Solution

```
1  ' Name:      Harkins Project
2  ' Purpose:    Display the company's annual sales amount.
3  ' Programmer: <your name> on <current date>
4
5  Option Explicit On
6  Option Strict On
7  Option Infer Off
8
9  Public Class frmMain
10     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
11         Me.Close()
12     End Sub
13
14     Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
15         ' Calculate and display the annual sales amount.
16
17         ' Declare variables:
18         Dim inFile As IO.StreamReader      ' will store the StreamReader object that represents the sales.txt file
19         Dim strCategory As String          ' will store the product categories read from the inFile
20         Dim intSales As Integer            ' will store the sales amounts read from the inFile
21         Dim intAnnualSales As Integer      ' will be used by the procedure to accumulate the sales amounts
22
23         If IO.File.Exists("sales.txt") Then
24             inFile = IO.File.OpenText("sales.txt")
25             Do Until inFile.Peek = -1
26                 ' Read product category:
27                 strCategory = inFile.ReadLine
28                 ' Read sales amount and convert to Integer:
29                 Integer.TryParse(inFile.ReadLine, intSales)
30                 ' Add sales amount to accumulator:
31                 intAnnualSales += intSales
32             Loop
33             inFile.Close()
34             lblAnnualSales.Text = intAnnualSales.ToString("C0")
35         Else
36             MessageBox.Show("Cannot find the file.", "Company Sales", MessageBoxButtons.OK, MessageBoxIcon.Information)
37             lblAnnualSales.Text = "N/A"
38         End If
39     End Sub
40 End Class
```

<- every odd line in sales.txt = product name

<- every even line in sales.txt = number, sales amount

## CH9\_A4 - easy way to Sort the Data contained in a File example: 06.States Solution

- in this section, you will learn an easy way to **Sort** the data contained in a **Sequential access file**

1. first you **read** the unsorted data from the file, storing it in a list box whose property **Sorted = True**
2. then you **write** the sorted contents of the list box either to the same file or to a different file

- sort the data contained in a sequential access file:

- 1). open the: ...VB2017\Chap09\Exercise06.States Solution\States Solution.sln

- the **(LstStates)** control in the interface has set property **Sorted = True**

- 2). open the **states.txt** file:

-> in a Visual Studio's menu bar click **File**, choose **Open**, click **File...**, open the States Project's **bin\Debug** folder and open the **states.txt** file

- the file contains the names of the **50** US states

<- notice that the names are not in alphabetical order

-> close the **states.txt** window

- 3). open the Code Editor window and locate the **frmMain\_Load** procedure

<- notice that the procedure reads the names from the **states.txt** file and adds each name to the **(LstStates)** control

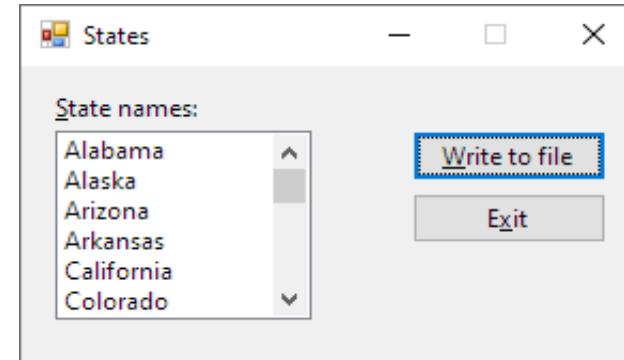
```
14      Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
15          ' Add file's contents to a list box.
16
17          Dim inFile As IO.StreamReader
18
19          If IO.File.Exists("states.txt") Then
20              inFile = IO.File.OpenText("states.txt")
21              Do Until inFile.Peek = -1
22                  LstStates.Items.Add(inFile.ReadLine)
23              Loop
24              inFile.Close()
25          Else
26              MessageBox.Show("Cannot find the file.", "States", MessageBoxButtons.OK, MessageBoxIcon.Information)
27          End If
28      End Sub
```

- 4). start the application

- the **(LstStates)** control in the interface has set property **Sorted = True**,

so the state names appear in ascending order

-> click the **Exit** button



5). locate the **btnWrite\_Click** procedure on line 30

- the procedure will write the sorted contents of a list box to the **states.txt** file, so:

-> enter the following code:

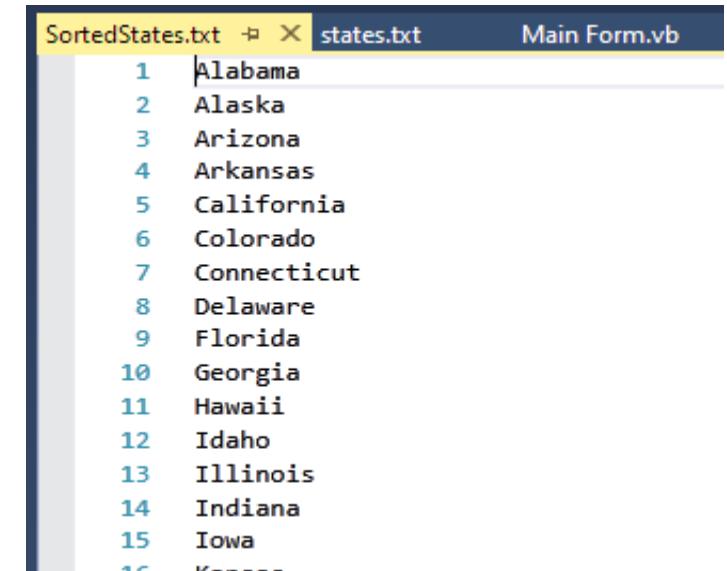
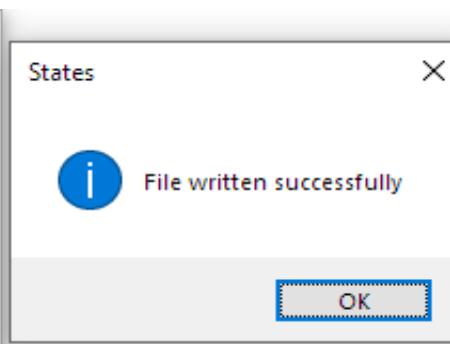
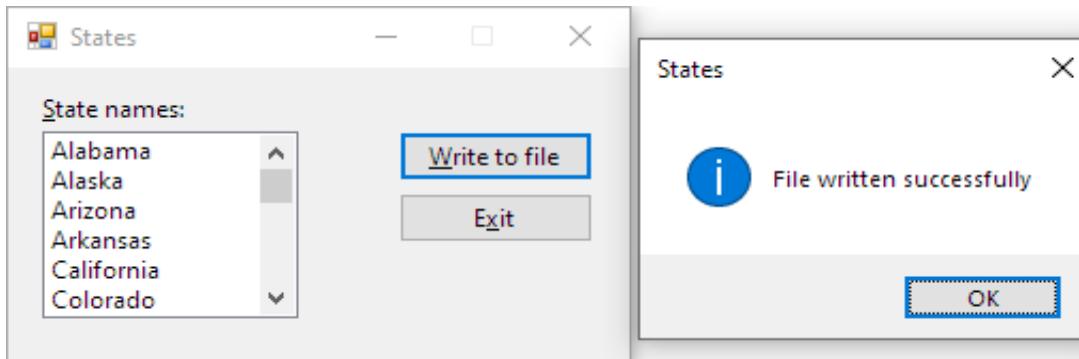
```
30  Private Sub btnWrite_Click(sender As Object, e As EventArgs) Handles btnWrite.Click
31      ' Write the sorted contents of the list box to a file.
32
33      Dim outFile As IO.StreamWriter
34
35      outfile = IO.File.CreateText("SortedStates.txt")
36
37      For intIndex As Integer = 0 To lstStates.Items.Count - 1
38          outFile.WriteLine(lstStates.Items(intIndex))
39      Next intIndex
40
41      outFile.Close()
42
43      MessageBox.Show("File written successfully.", "States", MessageBoxButtons.OK, MessageBoxIcon.Information)
44
45  End Sub
46 End Class
```

6). save the solution and then start and test the application

-> click the button **Write to file**

- the **btnWrite\_Click** procedure writes the contents of the list box to the **SortedStates.txt** file and displays the "File written successfully." message

-> click the **OK** button to close the message box and then click the **Exit** button



7). open the **SortedStates.txt** file to verify that 50 names are in alphabetical order

-> close the **SortedStates.txt** window, close the Code Editor window and close the solution.

## 8). completed code: 06.States Solution

```
1  ' Name:      States Project
2  ' Purpose:    Sort the contents of a file and then save to the file.
3  ' Programmer: <your name> on <current date>
4
5  Option Explicit On
6  Option Strict On
7  Option Infer Off
8
9  Public Class frmMain
10     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
11         Me.Close()
12     End Sub
13
14     Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
15         ' Add file's contents to a list box.
16
17         Dim inFile As IO.StreamReader
18
19         If IO.File.Exists("states.txt") Then
20             inFile = IO.File.OpenText("states.txt")
21             Do Until inFile.Peek = -1
22                 lstStates.Items.Add(inFile.ReadLine)
23             Loop
24             inFile.Close()
25         Else
26             MessageBox.Show("Cannot find the file.", "States", MessageBoxButtons.OK, MessageBoxIcon.Information)
27         End If
28     End Sub
29
30     Private Sub btnWrite_Click(sender As Object, e As EventArgs) Handles btnWrite.Click
31         ' Write the sorted contents of the list box to a file.
32
33         Dim outFile As IO.StreamWriter
34
35         outFile = IO.File.CreateText("SortedStates.txt")
36
37         For intIndex As Integer = 0 To lstStates.Items.Count - 1
38             outFile.WriteLine(lstStates.Items(intIndex))
39         Next intIndex
40
41         outFile.Close()
42
43         MessageBox.Show("File written successfully", "States", MessageBoxButtons.OK, MessageBoxIcon.Information)
```

```
44  
45      End Sub  
46  End Class
```

### CH9\_A5 - Professionalize Your Application's Interface: confirmation message when the file is written (07.Game Show Solution-Professionalize)

- in this section, you will professionalize the GUI for one of the **Game Show** applications from this chapter's **Focus** lesson
- more specifically, you will have the application display a **message** each time the user clicks the **Write to file** button
- the message will confirm that the contestant's name was written to the **contestants.txt** file
- you can display the message in either:
  - a). a message box `MessageBox.Show(.....)`
  - b). a label on the form `(lbl...)`
- in this example you will use a **label** so that the user will not need to close the message box each time a record is written to the file

- professionalize the application's GUI by adding a confirmation message when the file is written:

1). open the: ...VB2017\Chap09\Exercise\07.Game Show Solution-Professionalize\Game Show Solution.sln

2). open the Designer window

-> add a **label** to the form and position it in the lower-left corner  
-> change its **Properties** as follows:

- **(Name)** = `lblMessage`
- **Text** = `Name written to file.`
- **Visible** = `False`

3). open the Code Editor window and locate the **btnWrite\_Click** procedure on line **10**

-> insert a blank line above the **End Sub** clause, and insert the following lines of code:

```
29      ' Make the confirmation label visible:  
30      lblMessage.Visible = True  
31  End Sub
```

4). when user enters a name in a text box, the confirmation message should not be visible, so:

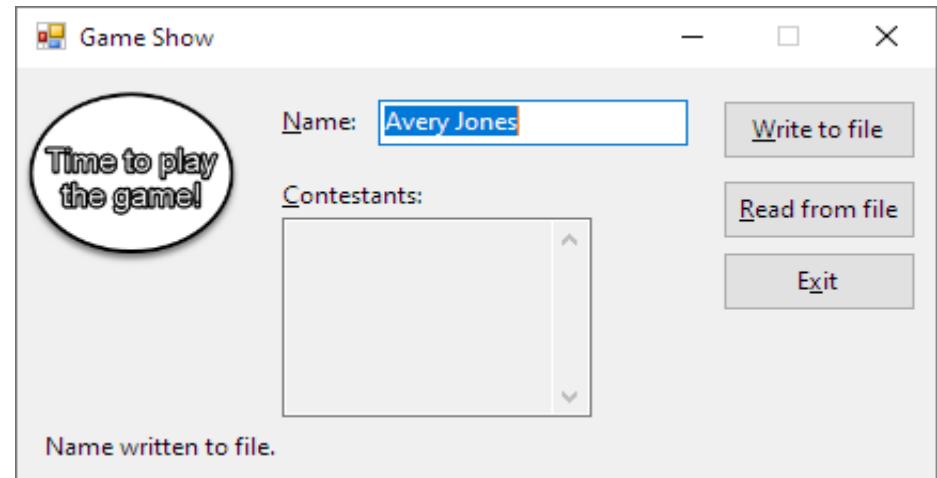
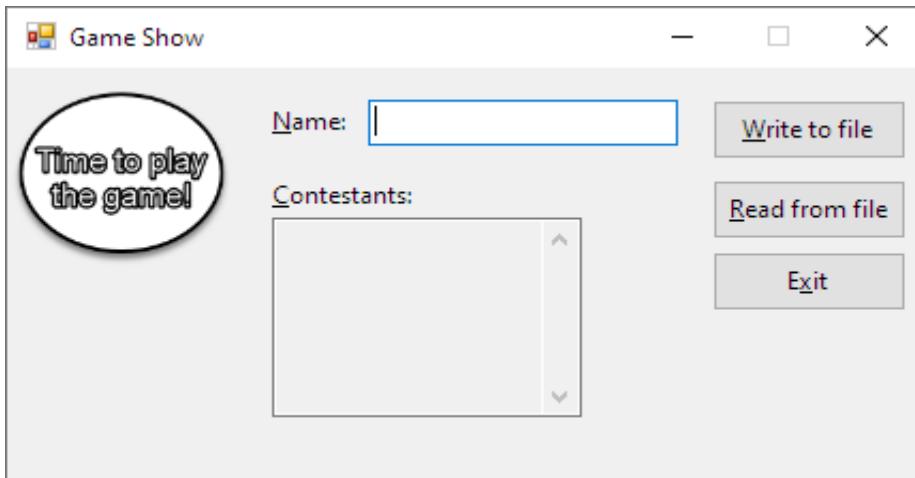
-> open the code template for the **txtName\_TextChanged** procedure and enter the following lines of code:

```
59  Private Sub txtName_TextChanged(sender As Object, e As EventArgs) Handles txtName.TextChanged  
60      ' when user enters a name, the confirmation label is not visible:  
61      lblMessage.Visible = False  
62  End Sub  
63 End Class
```

5). save the solution and then start and test the application:

-> in the **Name** box type: **Avery Jones** and then click the button **Write to file**

<- the **btnWrite\_Click** procedure changes the (**lblMessage**) control's property **Visible = True** and as a result, the "Name written to file." message appears in the lower-left corner of the GUI

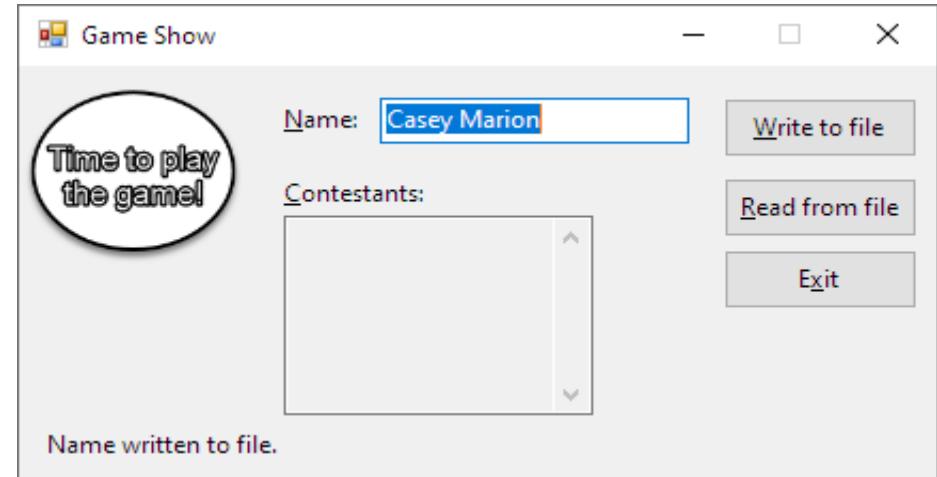
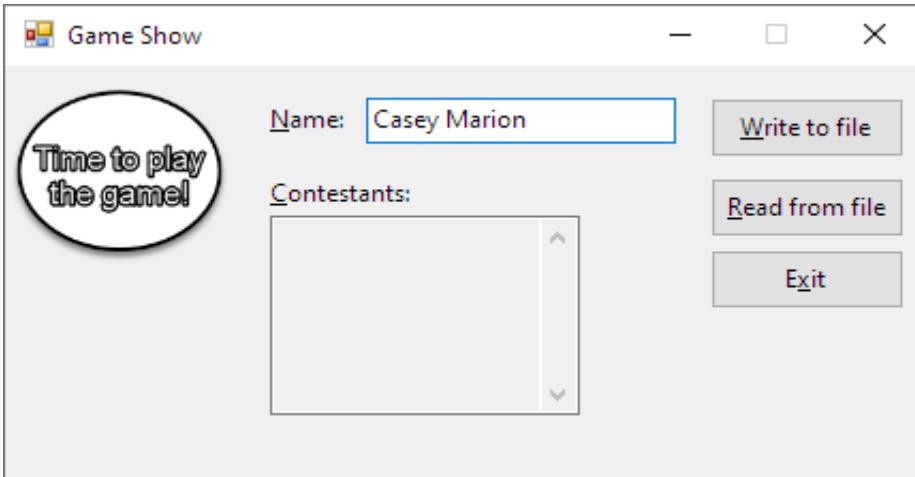


-> in the **Name** box type: **Casey Marion**

<- the confirmation message disappears because the **txtName\_TextChanged** procedure changes the (**lblMessage**) control's property **Visible = False**

-> click the button **Write to file**

<- the confirmation message appears once again



-> click the button **Exit**, close the Code Editor window and then close the solution

## 6). completed code: 07.Game Show Solution-Professionalize

```
1  ' Name:      Game Show Project
2  ' Purpose:    Writes names to and read names from a sequential access file.
3  ' Programmer: <your name> on <current date>
4
5  Option Explicit On
6  Option Strict On
7  Option Infer Off
8
9  Public Class frmMain
10     Private Sub btnWrite_Click(sender As Object, e As EventArgs) Handles btnWrite.Click
11         ' Writes a name to a sequential access file.
12
13         ' Declare a StreamWriter variable:
14         Dim outFile As IO.StreamWriter
15
16         ' Open the file for append.
17         outFile = IO.File.AppendText("contestants.txt")
18
19         ' Write the name on a separate line in the file:
20         outFile.WriteLine(txtName.Text.Trim)
21
22         ' Close the file:
23         outFile.Close()
24
25         ' Clear the Contestants box and then set the focus:
26         txtContestants.Text = String.Empty
27         txtName.Focus()
28
29         ' Make the confirmation label visible:
30         lblMessage.Visible = True
31     End Sub
32
33     Private Sub btnRead_Click(sender As Object, e As EventArgs) Handles btnRead.Click
34         ' Reads names from a sequential access file and displays them in the interface.
35
36         ' Declare variable:
37         Dim inFile As IO.StreamReader
38
39         ' Determine whether the file exists:
40         If IO.File.Exists("contestants.txt") Then
41             ' Open the file for input:
42             inFile = IO.File.OpenText("contestants.txt")
```

```

43     ' Read the file and assign to Contestants box:
44     txtContestants.Text = inFile.ReadToEnd
45     inFile.Close()
46 
47     Else
48         MessageBox.Show("Cannot find the file.", "Game Show", MessageBoxButtons.OK, MessageBoxIcon.Information)
49     End If
50 
51     End Sub
52 
53     Private Sub txtName_Enter(sender As Object, e As EventArgs) Handles txtName.Enter
54         txtName.SelectAll()
55     End Sub
56 
57     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
58         Me.Close()
59     End Sub
60 
61     Private Sub txtName_TextChanged(sender As Object, e As EventArgs) Handles txtName.TextChanged
62         ' when user enters a name, the confirmation label is not visible:
63         lblMessage.Visible = False
64     End Sub
65 
66 End Class

```

### CH9\_Summary: Sequential access files/Text files, **MenuStrip** tool, easy sorting of a Text file, confirmation message

1. Sequential access files:
  - also called **text files** with suffix **.txt**
  - can be used for an application's **input** or for its **output**
  - are composed of **streams of characters** that are both **read** and **written** in consecutive order
  
2. to create and use an **output** sequential access file, use the steps and syntaxes shown in:
 

[CH9\\_F2 - Sequential Access Output Files: Class StreamWriter](#) - how to create and use them step by step
  
- 3a. to create and use an **input** sequential access file, use the steps and syntaxes shown in:
 

[CH9\\_F3 - Sequential Access Input Files: Class StreamReader](#) - how to create and use them step by step
  
- 3b. the **ReadLine** function and the **ReadToEnd** function return strings
  
- 4a. to add a **menu strip** control to a form, use: **Designer** window's **Toolbox / Menus & Toolbars** section and choose: **MenuStrip** tool / **Class**
- 4b. you can create a **menu** by replacing the words "**Type Here**" with the menu element's caption
  - you should assign a meaningful name and a unique access key to each menu element, with the exception of separator bars
- 4c. to include a **separator bar** on a menu, place your mouse pointer on a "**Type Here**" box, click the list arrow that appears inside the box, and then click **Separator**
- 4d. you can use a menu item's **Shortcut keys** property to assign shortcut keys to the item
- 4e. you can use a menu item's **access key** to select the item when the menu is open and you can use its **shortcut** keys to select it when the menu is closed

[CH9\\_A1 - Add a Menu\(mnu\) to a Form: Toolbox / Menus & Toolbars / MenuStrip control](#)

5. an easy way to **sort** the contents of a file is to add the contents to a list box whose property **Sorted = True** and then write the contents of the list box to either same file or a different file

**CH9\_A4 - easy way to Sort the Data contained in a File example: 06.States Solution**

6. you can display a confirmation message in either a message box or a label on the form

**CH9\_A5 - Professionalize Your Application's Interface: confirmation message when the file is written (07.Game Show Solution-Professionalize)**

### CH9\_Key Terms

- **AppendText function/method** - used with a **StreamWriter** variable to open a sequential access file for append [přidat]
- **Character stream / Stream of characters** - a sequence of characters
- **Close method** - used with either a **StreamWriter** variable or a **StreamReader** variable to close a sequential access file
- **CreateText function/method** - used with a **StreamWriter** variable to open a sequential access file for output
- **Exists function/method** - used to determine whether a file exists
- **Input files** - files from which an application reads data
- **Line** - a sequence/stream of characters followed by the newline character
- **MenuStrip control** - used to include one or more menus on a form: instantiated using the **MenuStrip** tool located in the **Menus & Toolbars** section of the **Toolbox**
- **OpenText function/method** - used with a **StreamReader** variable to open a sequential access file for input
- **Output files** - files to which an application writes data
- **Peek function/method** - used with a **StreamReader** variable to determine whether a file contains another character to read
  - if the function returns a value of **-1**, then there is no other character to read
- **ReadLine function/method** - used with a **StreamReader** variable to read a line of text from a sequential access file
- **ReadToEnd function/method** - used with a **StreamReader** variable to read a sequential access file, all at once
- **Sequential access files / Text files** - files composed of characters that are both read and written sequentially
- **Sequential access input files** - a **StreamReader** object -> used to refer to a sequential access input file in code
- **Sequential access output files** - a **StreamWriter** object -> used to refer to a sequential access output file in code
- **Shortcut keys** - appear to the right of a menu item and allow the user to select the item without opening the menu
  - e.g. **Ctrl+C** for **Copy**, **Ctrl+P** for **Print**, etc...
- **Stream of characters / Character stream** - a sequence of characters
- **StreamReader class** - the Visual Basic class used to create **StreamReader** objects
- **StreamReader object** - used to read a sequence/stream of characters from a sequential access file
- **StreamReader variable** - a variable that stores a **StreamReader** object -> used to refer to a sequential access input file in code
- **StreamWriter class** - the Visual Basic class used to create **StreamWriter** objects
- **StreamWriter object** - used to write a sequence/stream of characters to a sequential access file
- **StreamWriter variable** - a variable that stores a **StreamWriter** object -> used to refer to a sequential access output file in code
- **Text files / Sequential access files** - files composed of characters that are both read and written sequentially
- **Write function/method** - used with a **StreamWriter** variable to write data to a sequential access file
  - differs from the **WriteLine** function/method in that it does not write a newline character after the data
- **WriteLine function/method** - used with a **StreamWriter** variable to write data to a sequential access file
  - differs from the **Write** function/method in that it writes a newline character after the data

## **CH9\_Exercises**

**Chap09\Exercise1**

**08.Electricity Solution\_EXERCISE 1\_introductory**

**Chap09\Exercise1**

**09.Workers Solution\_EXERCISE 2\_introductory**

**Chap09\Exercise1**

**10.Customer Solution\_EXERCISE 3\_introductory**

**Chap09\Exercise1**

**11.Potter Solution\_EXERCISE 4\_intermediate**

**Chap09\Exercise1**

**12.Cookies Solution\_EXERCISE 5\_intermediate**

**Chap09\Exercise1**

**13.Warren Solution\_EXERCISE 6\_intermediate**

**Chap09\Exercise1**

**14.Vacation Solution\_EXERCISE 7\_advanced**

**Chap09\Exercise1**

**15.Workers Solution-Filename\_EXERCISE 8\_advanced**

**Chap09\Exercise1**

**16.Wedding Solution\_EXERCISE 9\_advanced**

**Chap09\Exercise1**

**17.OnYourOwn Solution\_EXERCISE 10\_**

**Chap09\Exercise1**

**18.FixIt Solution\_EXERCISE 11**

**Chap09\Exercise1**

**\_Appendix E - Case Projects\_09.Shopping Cart-CH01-09 - using .NET Framework 4.8**

**Chap09\Exercise1**

**\_Appendix E - Case Projects\_10.Airplane Seats-CH01-09 - using .NET Framework 4.8**

08.Electricity Solution\_EXERCISE 1\_introductory

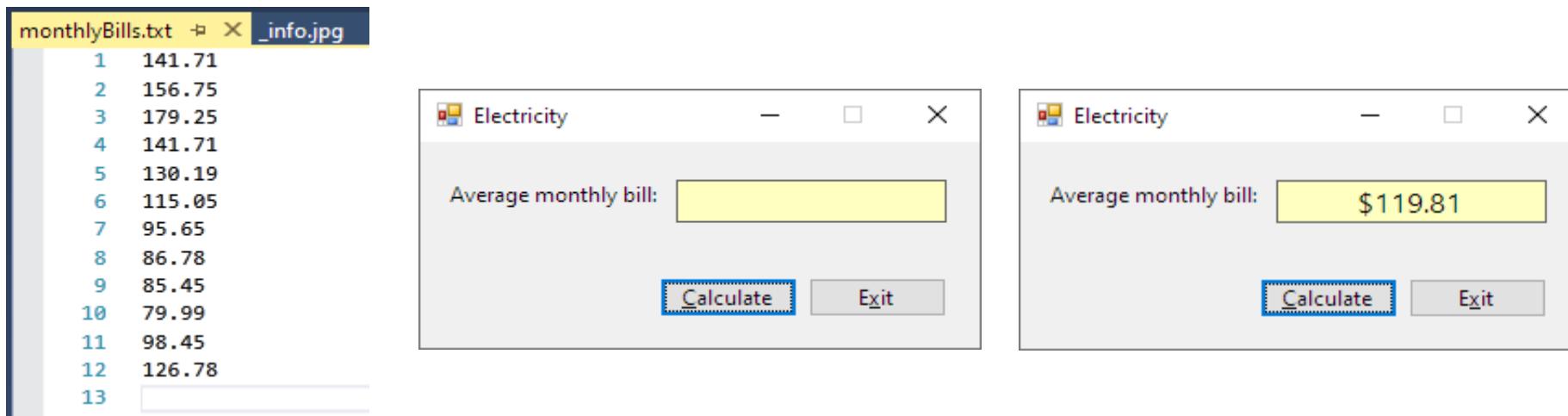
1. Open the Electricity Solution.sln file contained in the VB2017\Chap09\Electricity Solution folder. Open the monthlyBills.txt file. Each of the 12 numbers in the file represents the cost of electricity for a month. Close the monthlyBills.txt window. The btnCalc\_Click procedure should display the average monthly cost for electricity. Display the average with a dollar sign and two decimal places. Code the procedure. Save the solution and then start and test the application. (The average is \$119.81. If you need to recreate the file, the project's bin\Debug folder contains a copy of the original file.)

```
1  ' Name:          Electricity Project
2  ' Purpose:       Calculate the average monthly electric bill from Sequential Access File.
3  ' Programmer:    <your name> on <current date>
4  '_monthlyBills.txt = each of the 12 numbers represents the cost of electricity for a month
5  '_btnCalc_Click procedure should display the AVERAGE monthly cost for electricity in lblAvg
6  '_lblAvg = dollar sign and 2 decimals ("C2")
7  '_the result should be = $119.81
8  '__EXTRA: counter for the number of amounts
9
10 Option Explicit On
11 Option Strict On
12 Option Infer Off
13
14 Public Class frmMain
15     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
16         Me.Close()
17     End Sub
18
19     Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
20         Dim inFile As IO.StreamReader
21         Dim strPrice As String
22         Dim dblPrice As Double
23         Dim dblSuma As Double
24         Dim intCounter As Integer
25
26         If IO.File.Exists("monthlyBills.txt") Then
27             inFile = IO.File.OpenText("monthlyBills.txt")
28             Do Until inFile.Peek = -1
29                 strPrice = inFile.ReadLine
30                 Double.TryParse(strPrice, dblPrice)
31                 dblSuma += dblPrice
32                 intCounter += 1
33             Loop
34             lblAvg.Text = dblSuma / intCounter
35         Else
36             MessageBox.Show("File does not exist")
37         End If
38     End Sub
39
40 End Class
```

```

34         inFile.Close()
35         lblAvg.Text = (dblSuma / intCounter).ToString("C2")
36     Else
37         MessageBox.Show("The Input file does not exist.", "Electricity", MessageBoxButtons.OK, MessageBoxIcon.Information)
38     End If
39 End Sub
40 End Class

```



## Chap09\Exercise1

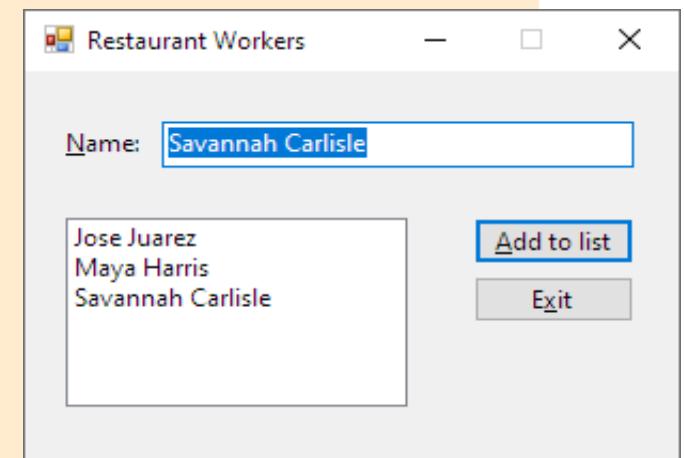
### 09.Workers Solution\_EXERCISE 2\_introductory

- Open the Workers Solution.sln file contained in the VB2017\Chap09\Workers Solution folder. The user will enter a name in the txtName control and then click the Add to list button, which should add the name to the lstWorkers control. When the user is finished entering names, the frmMain\_FormClosing procedure should write the contents of the list box to a new sequential access file named workers.txt. Code the procedure. Save the solution and then start the application. Test the application by entering the following two names: Henry Kaplan and Mario Brown. Stop the application and verify that the workers.txt file contains both names. Now, start the application and enter the following three names: Jose Juarez, Maya Harris, and Savannah Carlisle. Stop the application and verify that the workers.txt file contains only three names.

```

1  ' Name:      Workers Project
2  ' Purpose:    Saves worker names to a sequential access files.
3  ' Programmer: <your name> on <current date>
4  ' _user will: - enter a name in the (txtName) control &
5  '                 - click the button (btnAdd) which should add the name to the (lstWorkers) control
6  ' _when finished entering names:
7  ' - the frmMain_FormClosing procedure should write the contents Of the (lstWorkers) to a new sequential access file: "workers.txt"
8  ' _test by entering 2 names: "Henry Kaplan" and "Mario Brown", stop the app and verify the contents of the "workers.txt"
9  ' <- it should countain 2 names
10 ' _start the app again and:
11 '   _test by entering 3 names: "Jose Juarez", "Maya Harris", "Savannah Carlisle", stop the app and verify
12 '   <- it should contain only 3 names
13
14 Option Explicit On
15 Option Strict On
16 Option Infer Off
17
18 Public Class frmMain
19     Private Sub btnAdd_Click(sender As Object, e As EventArgs) Handles btnAdd.Click
20         lstWorkers.Items.Add(txtName.Text)
21         txtName.Focus()
22         txtName.SelectAll()
23     End Sub
24
25     Private Sub frmMain_FormClosing(sender As Object, e As FormClosingEventArgs) Handles Me.FormClosing
26         Dim outFile As IO.StreamWriter
27         outFile = IO.File.CreateText("workers.txt")
28
29         For intCounter As Integer = 0 To lstWorkers.Items.Count - 1
30             outFile.WriteLine(lstWorkers.Items(intCounter))
31         Next intCounter
32
33         outFile.Close()
34     End Sub
35
36     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
37         Me.Close()
38     End Sub
39
40     Private Sub txtName_Enter(sender As Object, e As EventArgs) Handles txtName.Enter
41         txtName.SelectAll()
42     End Sub
43
44 End Class

```



## 10.Customer Solution\_EXERCISE 3\_introductory

3. Create a Windows Forms application. Use the following names for the project and solution, respectively: Customer Project and Customer Solution. Save the application in the VB2017\Chap09 folder.
- Create the interface shown in Figure 9-36. The File menu should have an Exit option. Change the `lblMessage` control's `Visible` property to `False`.
  - Code the File menu's Exit option.
  - The `lblMessage` control should disappear when a change is made to any of the text boxes. Create one event-handling Sub procedure to handle this task.
  - Code each text box's Enter event procedure.
  - The Save button should save the customer information to a sequential access file named `customers.txt`. Before saving the information, be sure to verify that all of the text boxes contain data; display an appropriate message if at least one text box is empty. Use the following format when saving the information: Save the customer's first name and last name, separated by a space character, on the same line. Then, on the next line, save the customer's address, followed by a comma, a space character, the city name, a comma, a space character, and the ZIP code. A blank line should separate each customer's information from the next customer's information. Figure 9-36 shows a sample of the sequential access file.
  - Save the solution and then start and test the application. Stop the application and verify the contents of the `customers.txt` file.

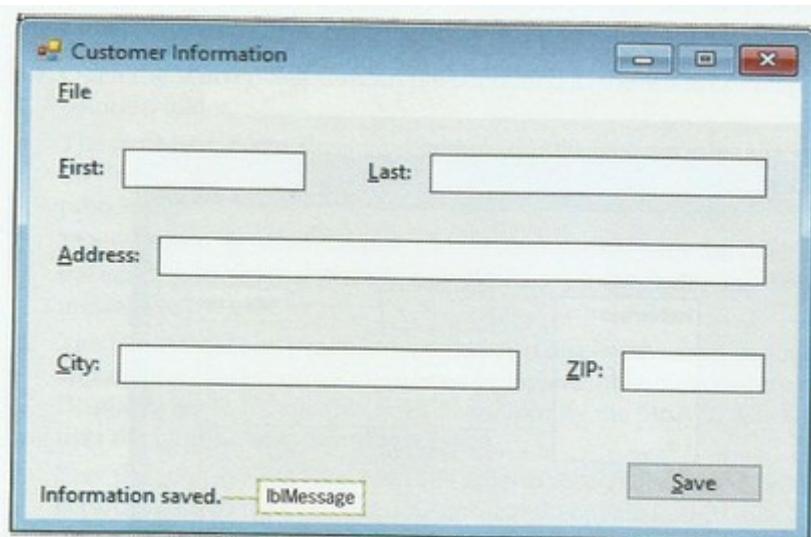


Figure 9-36 Interface and sample file for Exercise 3 (continues)

(continued)

| customers.txt | Main Form.vb                          | Main Form.vb [Design] |
|---------------|---------------------------------------|-----------------------|
| 1             | Thomas Young                          |                       |
| 2             | 1155 W. Main Street, Nashville, 37011 |                       |
| 3             |                                       |                       |
| 4             | Susan Rogers                          |                       |
| 5             | 56 Main Street, Bowling Green, 42101  |                       |
| 6             |                                       |                       |
| 7             |                                       |                       |

Figure 9-36 Interface and sample file for Exercise 3

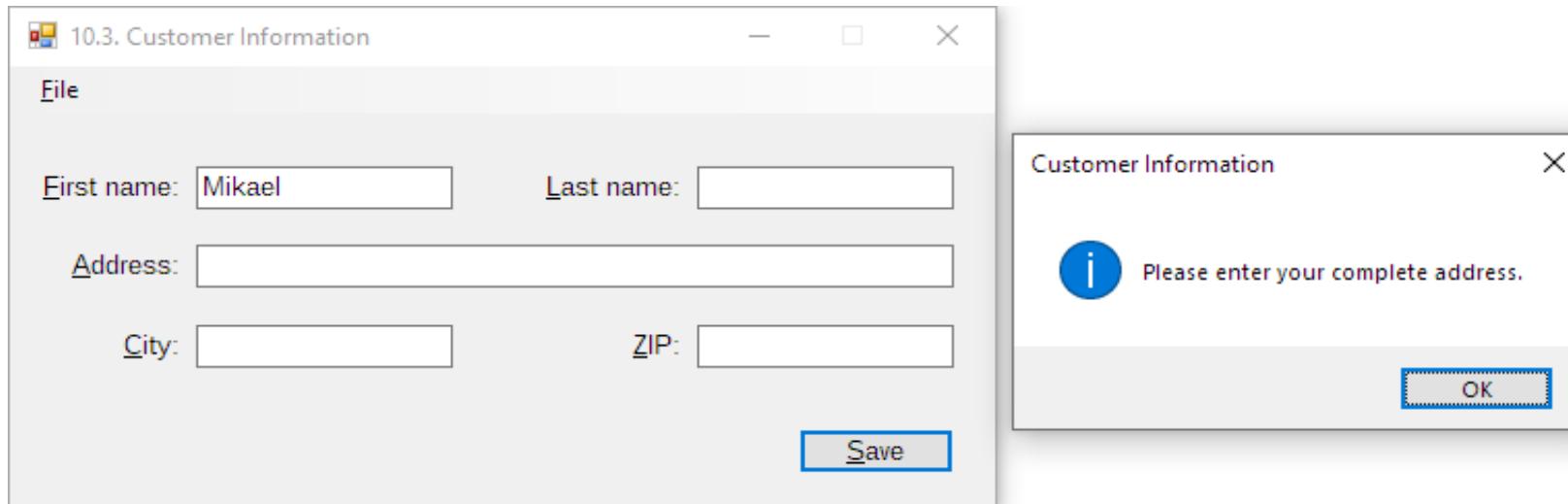
```

1  '-1. create GUI; (lblMessage) control's property Visible = False
2  '-2. code the File menu's Exit option
3  '-3. create event-handling Sub procedure: when a change is made to any (txt), the (lblMessage) should disappear
4  '-4. code each (txt)'s Enter event procedure (MY EXTRA: MouseClick)
5  '-5. btnSave_Click event procedure should:
6    '-5.1. verify that ALL of the (txt) contain data -> display an appropriate msg if NOT
7    '-5.2. save the customer information to: "customers.txt" using the following format:
8      line1 = (txt0FirstName) & " " & (txt1LastName)
9      line2 = (txt2Address) & ", " & (txt3City) & ", " & (txt4Zip)
10     line3 = blank line
11
12 Option Explicit On
13 Option Strict On
14 Option Infer Off

```

```
15
16  Public Class frmMain
17      Private Sub btnSave_Click(sender As Object, e As EventArgs) Handles btnSave.Click
18
19          If txt0FirstName.Text = Nothing OrElse txt1LastName.Text = Nothing OrElse txt2Address.Text = Nothing OrElse txt3City.Text = Nothing OrElse
20              txt4Zip.Text = Nothing Then
21                  MessageBox.Show("Please enter your complete address.", "Customer Information", MessageBoxButtons.OK, MessageBoxIcon.Information)
22          Else
23              Dim outFile As IO.StreamWriter
24              outFile = IO.File.AppendText("customers.txt")
25
26              outFile.WriteLine(txt0FirstName.Text & " " & txt1LastName.Text)
27              outFile.WriteLine(txt2Address.Text & ", " & txt3City.Text & ", " & txt4Zip.Text)
28              outFile.WriteLine()
29              outFile.Close()
30              lblMessage.Visible = True
31          End If
32      End Sub
33
34      Private Sub txtChanged(sender As Object, e As EventArgs) Handles txt0FirstName.TextChanged, txt1LastName.TextChanged,
35                      txt2Address.TextChanged, txt3City.TextChanged, txt4Zip.TextChanged
36          lblMessage.Visible = False
37      End Sub
38
39      Private Sub txt0FirstName_Enter(sender As Object, e As EventArgs) Handles txt0FirstName.Enter, txt0FirstName.MouseClick
40          txt0FirstName.SelectAll()
41      End Sub
42
43      Private Sub txt1LastName_Enter(sender As Object, e As EventArgs) Handles txt1LastName.Enter, txt1LastName.MouseClick
44          txt1LastName.SelectAll()
45      End Sub
46
47      Private Sub txt2Address_Enter(sender As Object, e As EventArgs) Handles txt2Address.Enter, txt2Address.MouseClick
48          txt2Address.SelectAll()
49      End Sub
50
51      Private Sub txt3City_Enter(sender As Object, e As EventArgs) Handles txt3City.Enter, txt3City.MouseClick
52          txt3City.SelectAll()
53      End Sub
54
55      Private Sub txt4Zip_Enter(sender As Object, e As EventArgs) Handles txt4Zip.Enter, txt4Zip.MouseClick
56          txt4Zip.SelectAll()
57      End Sub
58
```

```
59      Private Sub mnuExit_Click(sender As Object, e As EventArgs) Handles mnuExit.Click
60          Me.Close()
61      End Sub
62
63  End Class
```



The screenshot shows a code editor with two tabs: "customers.txt" and "Main Form.vb [Design]". The "customers.txt" tab displays the following text:

```
1 Thomas Young
2 1155 W. Main Street, Nashville, 37011
3
4 Susan Rogers
5 56 Main Street, Bowling Green, 42101
6
7
```

## Chap09\Exercise

### 11.Potter Solution\_EXERCISE 4\_intermediate

4. Create a Windows Forms application. Use the following names for the project and solution, respectively: Potter Project and Potter Solution. Save the application in the VB2017\Chap09 folder.
- The application will display the names of students who have earned the grade selected in a list box. The student names and grades are stored in the NamesAndGrades.txt file. Copy the file from the VB2017\Chap09 folder to the Potter Project's bin\Debug folder. Now, open the NamesAndGrades.txt file. The file contains 15 names and 15 grades. Close the NamesAndGrades.txt window.
  - Create the interface shown in Figure 9-37. Use the String Collection Editor to enter the five grades in the lstGrades control. Change the lstNames control's SelectionMode and Sorted properties to None and True, respectively.
  - Code the Exit button's Click event procedure.
  - The first item in the lstGrades control should be selected when the interface appears. Code the appropriate procedure.
  - The contents of the lstNames and lblNumber controls should be cleared when a different grade is selected in the lstGrades control. Code the appropriate procedure.
  - The Display button should display the names of students who have earned the grade selected in the lstGrades control. It should also display the number of students who have earned that grade. Code the appropriate procedure.
  - Save the solution and then start and test the application.

```
1  ' The application will display the names of students who have earned the grade selected in a (lstGrades) _0=A, 1=B, 2=C, 3=D, 4=F.
2  ' the student names and grades are stored in the: NamesAndGrades.txt file (15 names & grades)
3  '-1.create GUI
4  '-1.1. use the String Collection Editor to enter the 5 grades in the (lstGrades) control
5  '-1.2. change (lstNames) SelectionMode = None; Sorted = True
6  '-2. code the Exit button's Click event procedure
7  '-3. when GUI appears, the 1st item in the (lstGrades) should be selected _SelectedIndex = 0
8  '-4. when a different grade in (lstGrades) is selected, the (lstNames) & (lblNumber) should be cleared
9  '-5. (btnDisplay) should:
10   '- 5.1. in (lstNames) display the names of students who have earned the grade selected in the (lstGrades) control
11   '- 5.2. in (lblNumber) display the number of students who have earned that grade
12
13 'MY CHANGE in input file "NamesAndGrades.txt":
14 'originally: line1 = name & line2 = grade -> but how to choose a previous line for (lst) ???, therefore:
15 'modified: line1 = grade & line2 = name
16
17 Option Explicit On
18 Option Strict On
19 Option Infer Off
20
21 Public Class frmMain
```

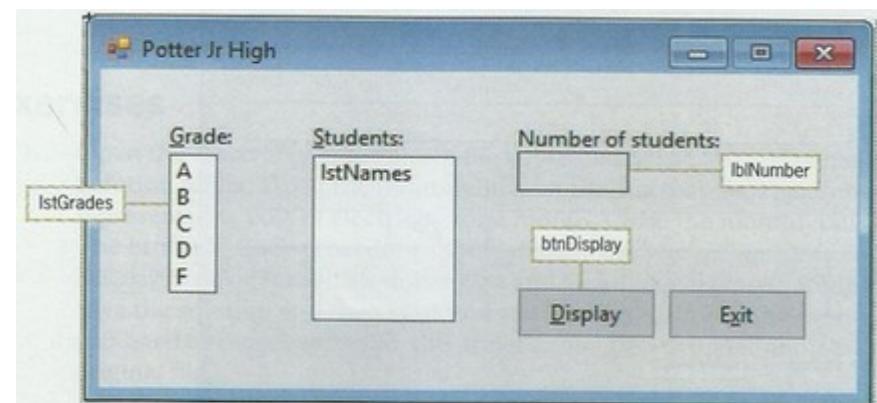
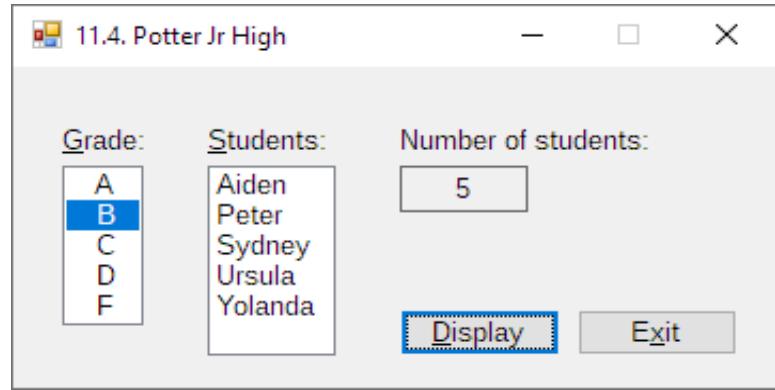
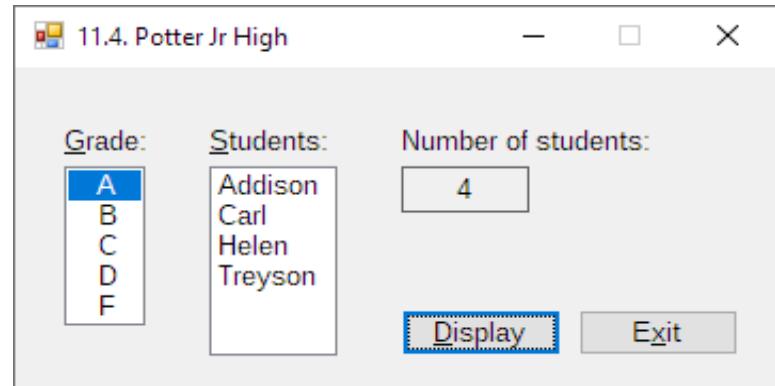


Figure 9-37 Interface for Exercise 4

```
22
23     Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
24
25         Dim inFile As IO.StreamReader
26
27         If IO.File.Exists("NamesAndGrades.txt") Then
28             inFile = IO.File.OpenText("NamesAndGrades.txt")
29
30             Dim strLineOfText As String
31             Dim intCounter As Integer
32
33             Do Until inFile.Peek = -1      ' cannot read CLOSED textreader
34                 strLineOfText = inFile.ReadLine
35                 'MessageBox.Show(strLineOfText)      ' shows line by line_ok
36
37             If strLineOfText Like lstGrades.SelectedItem.ToString.Trim Then
38                 lstNames.Items.Add(inFile.ReadLine)    ' MY CHANGE: had to modify input file
39                 intCounter += 1
40             End If
41
42             lblNumber.Text = intCounter.ToString
43         Loop
44
45         inFile.Close()
46     Else
47         MessageBox.Show("Can't find the file: NamesAndGrades.txt.", "Potter Jr High", MessageBoxButtons.OK, MessageBoxIcon.Information)
48     End If
49
50 End Sub
51
52 Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
53     lstGrades.SelectedIndex = 0
54 End Sub
55
56 Private Sub lstGrades_SelectedIndexChanged(sender As Object, e As EventArgs) Handles lstGrades.SelectedIndexChanged
57     lstNames.Items.Clear()
58     lblNumber.Text = Nothing
59 End Sub
60
61 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
62     Me.Close()
63 End Sub
64
65 End Class
```

```
NamesAndGrades.txt _original_NamesAndGrades.txt ↗ X Ma
1 Helen
2 A
3 Peter
4 B
5 Yolanda
6 B
7 Carl
8 A
9 Jennifer
10 D
11 Charles
12 F
13 Addison
14 A
15 Aiden
16 B
17 Treyson
18 A
19 Sydney
20 B
21 Jacob
22 F
23 Nancy
24 C
25 George
26 C
27 Ursula
28 B
29 Jack
30 D
31
```

```
NamesAndGrades.txt ↗ X _original_N
1 A
2 Helen
3 B
4 Peter
5 B
6 Yolanda
7 A
8 Carl
9 D
10 Jennifer
11 F
12 Charles
13 A
14 Addison
15 B
16 Aiden
17 A
18 Treyson
19 B
20 Sydney
21 F
22 Jacob
23 C
24 Nancy
25 C
26 George
27 B
28 Ursula
29 D
30 Jack
```



5. Create a Windows Forms application. Use the following names for the project and solution, respectively: Cookies Project and Cookies Solution. Save the application in the VB2017\Chap09 folder.
- Copy the cookieSales.txt file from the VB2017\Chap09 folder to the Cookies Project's bin\Debug folder. Open the cookieSales.txt file. The file contains the numbers of boxes sold for four different cookie types. Close the cookieSales.txt window.
  - Create the interface shown in Figure 9-38. Code the Exit button's Click event procedure.
  - The Display totals button should calculate and display the total number of boxes of each cookie type sold. Use a one-dimensional array to accumulate the numbers of boxes sold by cookie type. Then, display the array values in the label controls in the interface.
  - Save the solution and then start and test the application. (The total number of boxes of chocolate chip cookies sold is 30.)

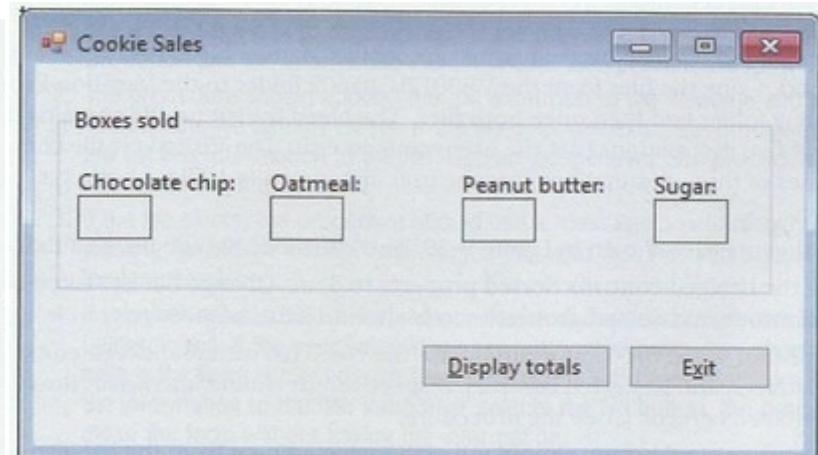


Figure 9-38 Interface for Exercise 5

```

1  'The application will display the total number of sold cookies
2
3  ' the file "cookieSales.txt" contains the number of boxes sold for 4 different cookie types:
4  '   lbl0_Chocolate = Chocolate chip = int1DTotals(0) = 30
5  '   lbl1_Oatmeal    = Oatmeal      = int1DTotals(1) = 4
6  '   lbl2_Peanut     = Peanut butter = int1DTotals(2) = 19
7  '   lbl3_Sugar      = Sugar        = int1DTotals(3) = 25
8
9  '-1. create GUI
10 '-2. code the "Exit" button's Click event procedure
11 '-3. button "Display totals" should calculate and display the total number of boxes of each cookie type sold
12 '-   3.1. use a 1D Array to accumulate the numbers of boxes sold by cookie type,
13 '-   3.2. then, display the Array values in the label controls in the GUI
14 '-4. test: Chocolate chip ttl = 30
15
16 Option Explicit On
17 Option Strict On
18 Option Infer Off
19
20 Public Class frmMain
21     Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
22         Dim int1DTotals(3) As Integer
23         Dim inFile As IO.StreamReader
24         Dim intNumber As Integer
25

```

```
26     If IO.File.Exists("cookieSales.txt") Then
27         inFile = IO.File.OpenText("cookieSales.txt")
28
29         Dim strLineOfText As String
30
31         Do Until inFile.Peek = -1
32             strLineOfText = inFile.ReadLine
33
34             ' instructions:
35             Select Case True
36                 Case strLineOfText Like "Chocolate chip".Trim
37                     Integer.TryParse(inFile.ReadLine, intNumber)
38                     int1DTotals(0) += intNumber
39                 Case strLineOfText Like "Oatmeal".Trim
39                     Integer.TryParse(inFile.ReadLine, intNumber)
40                     int1DTotals(1) += intNumber
41                 Case strLineOfText Like "Peanut butter".Trim
42                     Integer.TryParse(inFile.ReadLine, intNumber)
43                     int1DTotals(2) += intNumber
44                 Case strLineOfText Like "Sugar".Trim
45                     Integer.TryParse(inFile.ReadLine, intNumber)
46                     int1DTotals(3) += intNumber
47
48             End Select
49
50             Loop
51
52             lbl0_Chocolate.Text = int1DTotals(0).ToString
53             lbl1_Oatmeal.Text = int1DTotals(1).ToString
54             lbl2_Peanut.Text = int1DTotals(2).ToString
55             lbl3_Sugar.Text = int1DTotals(3).ToString
56
57             Else
58                 MessageBox.Show("Can't locate the cookieSales.txt file.", "Cookie Sales", MessageBoxButtons.OK, MessageBoxIcon.Information)
59             End If
60
61         End Sub
62
63         Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
64             Me.Close()
65         End Sub
66     End Class
```

```

cookieSales.txt  Main Form.
1 Chocolate chip|  

2 20  

3 Sugar  

4 10  

5 Peanut butter  

6 5  

7 Chocolate chip  

8 2  

9 Oatmeal  

10 4  

11 Peanut butter  

12 5  

13 Sugar  

14 13  

15 Sugar  

16 2  

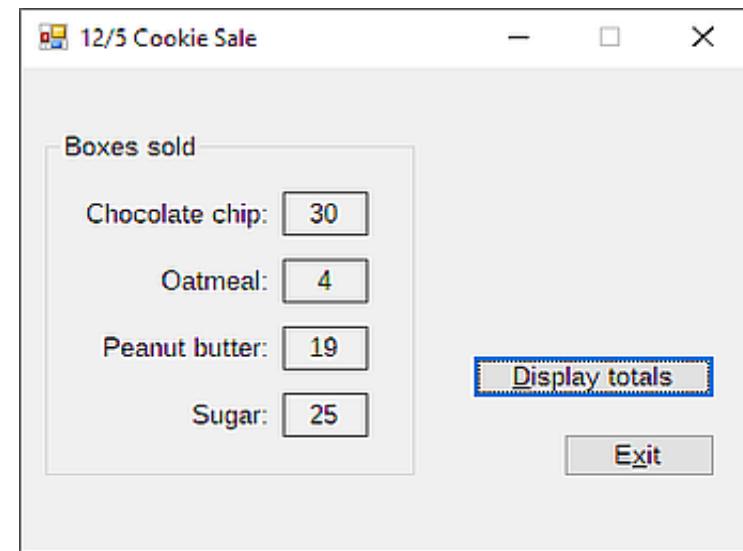
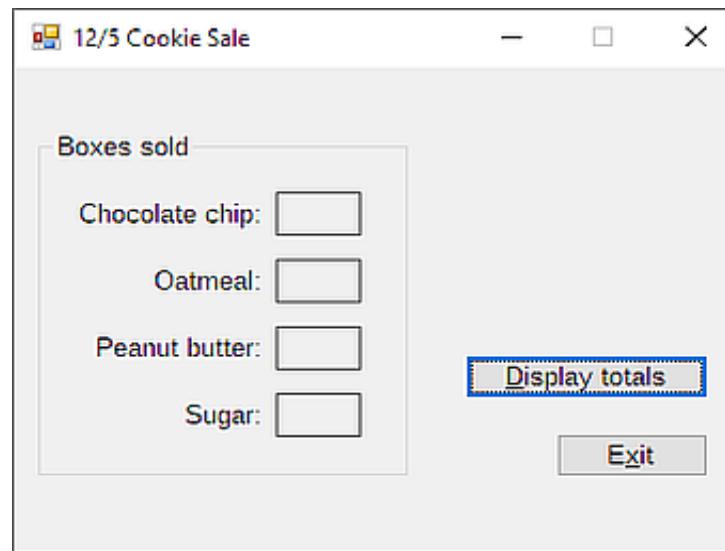
17 Peanut butter  

18 9  

19 Chocolate chip  

20 8

```



## Chap09\Exercises

### 13.Warren Solution\_EXERCISE 6\_intermediate

In this exercise, you will modify the Warren School application from Chapter 8's Apply the Concepts lesson.

- Open the Warren Solution.sln file contained in the VB2017\Chap09\Warren Solution folder.
- The frmMain\_FormClosing procedure should save the sales amounts stored in the intCandy array to a sequential access file named candySales.txt. Code the procedure.
- The frmMain\_Load procedure should fill the array with the sales amounts from the candySales.txt file. (*Hint:* If the file does not exist, you do not need to display a message to the user.)
- The frmMain\_Load procedure should also display the array amounts in the appropriate label controls. Create an independent Sub procedure named DisplayArray to handle this task. Then, modify the btnAdd\_Click procedure so it uses the DisplayArray procedure.

- Save the solution and then start the application. Type 7 in the Sold box and then click the Add to total button. The number 7 appears in the Choco Bar box. Now, click Kit Kat in the Candy list box, type 10 in the Sold box, and then click the Add to total button. The number 10 appears in the Kit Kat box. Click the Exit button.
- Open the candySales.txt file. The file contains the numbers 7, 0, 10, 0, and 0. Close the candySales.txt file.
- Start the application again. The candy boxes on the form reflect the amounts stored in the file: 7, 0, 10, 0, and 0.
- Click Kit Kat in the Candy box, type 20 in the Sold box, and then click the Add to total button. The number 30 appears in the Kit Kat box. Click the Exit button.
- Open the candySales.txt file. The file now contains the numbers 7, 0, 30, 0, and 0. Close the candySales.txt file.
- Start the application again. The candy boxes on the form reflect the amounts stored in the file: 7, 0, 30, 0, and 0. On your own, continue testing the application. When you are finished testing, click the Exit button.

```

1  ' Name:      Warren Project
2  ' Purpose:    Displays the amount sold for each candy type.
3  ' Programmer: <your name> on <current date>
4  'You will modify the Warren School app from CH8:
5
6  '-1. the "frmMain_FormClosing" procedure should:
7  '- - save the sales amounts stored in 1D Array "intCandy(0-4)" to Sequential Access File: "candySales.txt"
8
9  '-2. the "frmMain_Load" procedure should:
10 '- 2.1: fill 1D Array "intCandy(0-4)" with the sales amounts from the "candySales.txt" file
11 '-       - hint: if the does not exist, you do not need to display a message to the user
12 '- 2.2: display 1D Array "intCandy(0-4)" amounts in the appropriate label controls
13 '-       - create an Independent Sub procedure named "DisplayArray" to handle this task
14
15 '-3. then, modify the "btnAdd_Click" procedure so it uses the "DisplayArray()" procedure
16 '_4 moved to the end
17
18 Option Explicit On
19 Option Strict On
20 Option Infer Off
21
22 Public Class frmMain
23     Private intCandy(4) As Integer
24     Private inFile As IO.StreamReader
25
26     Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
27         ' Fills the list box with values and then selects the first value.
28         lstCandy.Items.Add("Choco Bar")           ' = intCandy(0)
29         lstCandy.Items.Add("Choco Bar-Peanuts")   ' = intCandy(1)
30         lstCandy.Items.Add("Kit Kat")             ' = intCandy(2)
31         lstCandy.Items.Add("Peanut Butter Cups") ' = intCandy(3)
32         lstCandy.Items.Add("Take 5 Bar")          ' = intCandy(4)
33         lstCandy.SelectedIndex = 0
34
35         ' 2. the "frmMain_Load" procedure should:
36         ' 2.1: fill 1D Array "intCandy(0-4)" with the sales amounts from the "candySales.txt" file
37         '       - hint: if the does not exist, you do not need to display a message to the user
38         ' 2.2: display 1D Array "intCandy(0-4)" amounts in the appropriate label controls
39         '       - create an Independent Sub procedure named "DisplayArray" to handle this task
40
41         If IO.File.Exists("candySales.txt") Then
42             inFile = IO.File.OpenText("candySales.txt")
43

```

```
44     Do Until inFile.Peek = -1
45         ' instructions:
46         For intCounter As Integer = 0 To 4
47             Integer.TryParse(inFile.ReadLine, intCandy(intCounter))
48             'MessageBox.Show(intCandy(intCounter).ToString)
49             Next intCounter
50
51             DisplayArray()
52         Loop
53
54         inFile.Close()
55
56     End If
57
58 End Sub
59
60 Private Sub DisplayArray()
61     lblChocoBar.Text = intCandy(0).ToString
62     lblChocoBarPeanuts.Text = intCandy(1).ToString
63     lblKitKat.Text = intCandy(2).ToString
64     lblPeanutButCups.Text = intCandy(3).ToString
65     lblTake5Bar.Text = intCandy(4).ToString
66
67 End Sub
68
69 Private Sub frmMain_FormClosing(sender As Object, e As FormClosingEventArgs) Handles Me.FormClosing
70     ' 1. the "frmMain_FormClosing" procedure should:
71     '      - save the sales amounts stored in 1D Array "intCandy(0-4)" to Sequential Access File: "candySales.txt"
72     Dim outFile As IO.StreamWriter
73     outFile = IO.File.CreateText("candySales.txt")
74
75     ' intCandy(0-4)
76     For intCounter As Integer = 0 To intCandy.GetUpperBound(0)
77         outFile.WriteLine(intCandy(intCounter))
78     Next intCounter
79
80     outFile.Close()
81 End Sub
82
83 Private Sub btnAdd_Click(sender As Object, e As EventArgs) Handles btnAdd.Click
84     ' Adds the amount sold to the appropriate total.
85
86     ' Declare array and variable.
87     'MOD from Static to Private: Static intCandy(4) As Integer
```

```
88     Dim intSold As Integer
89
90     Integer.TryParse(txtSold.Text, intSold)
91
92     ' Update array value.
93     intCandy(lstCandy.SelectedIndex) += intSold
94
95     ' 3. then, modify the "btnAdd_Click" procedure so it uses the "DisplayArray()" procedure
96     DisplayArray()
97
98     ' Display array values.
99     'lblChocoBar.Text = intCandy(0).ToString
100    'lblChocoBarPeanuts.Text = intCandy(1).ToString
101    'lblKitKat.Text = intCandy(2).ToString
102    'lblPeanutButCups.Text = intCandy(3).ToString
103    'lblTake5Bar.Text = intCandy(4).ToString
104
105    txtSold.Focus()
106 End Sub
107
108 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
109     Me.Close()
110 End Sub
111
112 Private Sub txtSold_SelectAll(sender As Object, e As EventArgs) Handles txtSold.Enter, txtSold.MouseClick
113     ' MY MOD: + MouseClick
114     txtSold.SelectAll()
115 End Sub
116
117 Private Sub txtSold_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtSold.KeyPress
118     ' Accept only numbers, the hyphen, and the Backspace.
119
120     If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> "-" AndAlso e.KeyChar <> ControlChars.Back Then
121         e.Handled = True
122     End If
123 End Sub
124
125 Private Sub ClearOutput(sender As Object, e As EventArgs) Handles txtSold.TextChanged ', lstCandy.SelectedIndexChanged
126     ' MY MOD:
127     Select Case True
128         Case lstCandy.SelectedIndex = 0
129             lblChocoBar.Text = Nothing
130         Case lstCandy.SelectedIndex = 1
131             lblChocoBarPeanuts.Text = Nothing
```

```
132     Case lstCandy.SelectedIndex = 2
133         lblKitKat.Text = Nothing
134     Case lstCandy.SelectedIndex = 3
135         lblPeanutButCups.Text = Nothing
136     Case lstCandy.SelectedIndex = 4
137         lblTake5Bar.Text = Nothing
138     End Select
139
140     ' ORIGINAL:
141     'lblChocoBar.Text = String.Empty
142     'lblChocoBarPeanuts.Text = String.Empty
143     'lblKitKat.Text = String.Empty
144     'lblPeanutButCups.Text = String.Empty
145     'lblTake5Bar.Text = String.Empty
146 End Sub
147
148 End Class
149
150
151 ' 4. test: save the solution and then start the application:
152 ' 4.1. -> type "7" in the "Sold" box and then click the button "Add to total"
153 '      <- the number "7" appears in the "Choco Bar" box
154 ' 4.2. -> click the "Kit Kat" in the "Candy" list box, type "10" in the "Sold" box and then click the button "Add to total"
155 '      <- the number "10" appears in the "Kit Kat" box
156 ' 4.3. -> click the button "Exit"
157 ' 4.4. -> open the "candySales.txt" file
158 '      <- the file contains the numbers "7, 0, 10, 0, 0"
159 '      -> close the "candySales.txt" file
160 ' 4.5. -> start the app again
161 '      <- the "Candy" boxes on the form reflect the amounts stored in the "candySales.txt" file: "7, 0, 10, 0, 0"
162 ' 4.6. -> click "Kit Kat" in the "Candy" box, type "20" in the "Sold" box and then click the button "Add to total"
163 '      <- the number "30" appears in the "Kit Kat" box
164 ' 4.7. -> click the button "Exit"
165 ' 4.8. -> open the file "candySales.txt"
166 '      <- the file now contains the numbers: "7, 0, 30, 0, 0"
167 '      -> close the file
168 ' 4.9. -> start the application again
169 '      <- the "Candy" boxes on the form reflect the amounts stored in the "candySales.txt" file: "7, 0, 30, 0, 0"
170 'that's all
```

## Chap09\Exercise1

### 14.Vacation Solution\_EXERCISE 7\_advanced

Create a Windows Forms application. Use the following names for the project and solution, respectively: Vacation Project and Vacation Solution. Save the application in the VB2017\Chap09 folder.

- The application will use two sequential access files named NeedToVisit.txt and Visited.txt. Copy the files from the VB2017\Chap09 folder to the Vacation Project's bin\Debug folder and then open both files. The NeedToVisit.txt file contains the names of five destinations that the user wants to visit. The Visited.txt file contains the names of three destinations that the user already visited. Close both .txt windows.
- Create the interface shown in Figure 9-39. The File menu should have an Exit option.
- Change the lstNeed control's Sorted property to True. Change the lstVisited control's SelectionMode and Sorted properties to None and True, respectively.
- The frmMain\_Load procedure should use the NeedToVisit.txt and Visited.txt files to fill the lstNeed and lstVisited controls, respectively. It should also select the first item in the lstNeed control. Code the procedure.
- The Move to visited button should move the selected item from the lstNeed control to the lstVisited control. Code the appropriate procedure.

- If at least one change was made to the list boxes, the frmMain\_FormClosing procedure should use a message box to ask the user whether he or she wants to save the change(s). If the user clicks the Yes button, the procedure should save the items in both list boxes to the appropriate files before the form closes.
- Save the solution and then start and test the application.

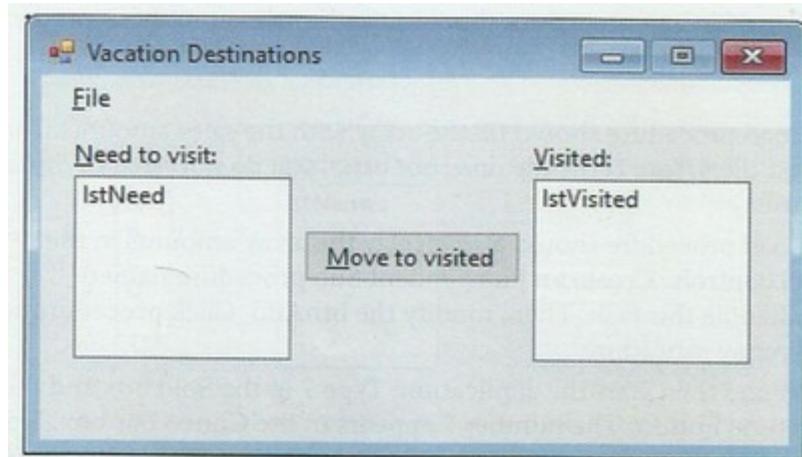


Figure 9-39 Interface for Exercise 7

```

1  'the app will use 2x Sequential Access Files:
2  '1. "NeedToVisit.txt" = contains the names of 5 destinations that the user wants to visit
3  '2. "Visited.txt"   = contains the names of 3 destinations that the user already visited
4
5  '-1. create GUI, where:
6  '- 1.1. the File menu should have an Exit option
7  '- 1.2. "lstNeed" control's property: Sorted = True
8  '- 1.3. "lstVisited" control's properties: Sorted = True; SelectionMode = None
9
10 '-2. the "frmMain_Load" procedure should:
11 '- 2.1. use the "NeedToVisit.txt" file to fill the "lstNeed" control
12 '- 2.2. select the 1st item in the "lstNeed" control
13 '- 2.3. use the "Visited.txt" file to fill the "lstVisited" control
14
15 '-3. the "btnMove" control should move the selected item from the "lstNeed" to the "lstVisited"
16
17 '-4. if at least 1 change was made to the list boxes, the "frmMain_FormClosing" procedure should:
18 '- use a MessageBox to ask the user whether wants to save the changes
19 '- -> if the user clicks the "Yes" button, the procedure should save the items in both list boxes to appropriate files before the form closes
20 'that's all
21
22 Option Explicit On
23 Option Strict On
24 Option Infer Off
25 Public Class frmMain
26
27     Private intCount As Integer
28
29     Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
30         ' 2. the "frmMain_Load" procedure should:
31         '    2.1. use the "NeedToVisit.txt" file to fill the "lstNeed" control
32         '    2.2. select the 1st item in the "lstNeed" control
33         '    2.3. use the "Visited.txt" file to fill the "lstVisited" control
34
35         Dim inNeed As IO.StreamReader
36         Dim inVisited As IO.StreamReader
37
38         If IO.File.Exists("NeedToVisit.txt") AndAlso IO.File.Exists("Visited.txt") Then
39             inNeed = IO.File.OpenText("NeedToVisit.txt")
40             inVisited = IO.File.OpenText("Visited.txt")
41
42             Do Until inNeed.Peek = -1
43                 lstNeed.Items.Add(inNeed.ReadLine)
44             Loop

```

```
45
46     If lstNeed.Items.Count > 0 Then
47         lstNeed.SelectedIndex = 0
48     Else
49         btnMove.Enabled = False
50     End If
51
52     Do Until inVisited.Peek = -1
53         lstVisited.Items.Add(inVisited.ReadLine)
54     Loop
55
56     ' count the input number items:
57     intCount = lstNeed.Items.Count
58     'MessageBox.Show(intCount.ToString)
59
60     inNeed.Close()
61     inVisited.Close()
62
63     Else
64         MessageBox.Show("Can't locate the files.", "Vacations Destinations", MessageBoxButtons.OK, MessageBoxIcon.Information)
65     End If
66 End Sub
67
68 Private Sub btnMove_Click(sender As Object, e As EventArgs) Handles btnMove.Click
69     ' 3. the "btnMove" control should move the selected item from the "lstNeed" to the "lstVisited"
70
71     ' add:
72     If lstNeed.Items.Count > 0 Then
73         lstVisited.Items.Add(lstNeed.SelectedItem)
74     Else
75         btnMove.Enabled = False
76     End If
77     ' remove:
78     lstNeed.Items.Remove(lstNeed.SelectedItem)
79
80     ' always select the 1st item in the (lstNeed):
81     If lstNeed.Items.Count > 0 Then
82         lstNeed.SelectedIndex = 0
83     Else
84         ' if there are no items:
85         btnMove.Enabled = False
86     End If
87 End Sub
88
```

```
89  Private Sub frmMain_FormClosing(sender As Object, e As FormClosingEventArgs) Handles Me.FormClosing
90      ' 4. if at least 1 change was made to the list boxes, the "frmMain_FormClosing" procedure should:
91      ' use a MessageBox to ask the user whether wants to save the changes
92      ' -> if the user clicks the "Yes" button, the procedure should save the items in both list boxes to appropriate files before the form closes
93
94      If lstNeed.Items.Count <> intCount Then
95          'MessageBox.Show("changed")
96
97          Dim dlgButton As DialogResult
98          dlgButton = MessageBox.Show("Save the changes?", "Vacation Destinations", MessageBoxButtons.YesNoCancel, MessageBoxIcon.Information)
99          If dlgButton = DialogResult.Yes Then
100
101              ' #1 step:
102              Dim outNeed As IO.StreamWriter      ' (lstNeed), "NeedToVisit.txt"
103              Dim outVisited As IO.StreamWriter    ' (lstVisited), "Visited.txt"
104
105              ' #2 step:
106              outNeed = IO.File.CreateText("NeedToVisit.txt")
107              outVisited = IO.File.CreateText("Visited.txt")
108
109              ' #3 step:
110
111              For intNeed As Integer = 0 To lstNeed.Items.Count - 1
112                  outNeed.WriteLine(lstNeed.Items(intNeed))
113              Next intNeed
114
115              For intVisited As Integer = 0 To lstVisited.Items.Count - 1
116                  outVisited.WriteLine(lstVisited.Items(intVisited))
117              Next intVisited
118
119              ' #4 step:
120              outNeed.Close()
121              outVisited.Close()
122
123              ElseIf dlgButton = DialogResult.Cancel Then
124                  e.Cancel = True
125              End If
126
127              'Else
128              'MessageBox.Show("no change")
129          End If
130      End Sub
131
132  Private Sub mnuFileExit_Click(sender As Object, e As EventArgs) Handles mnuFileExit.Click
```

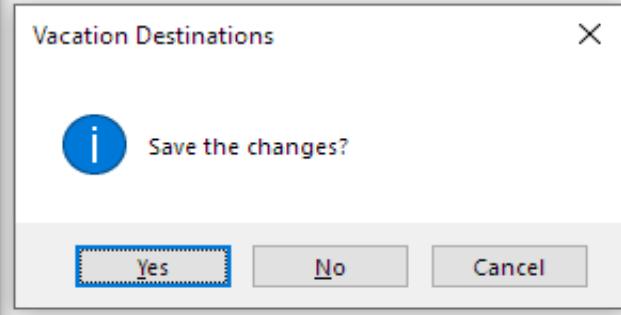
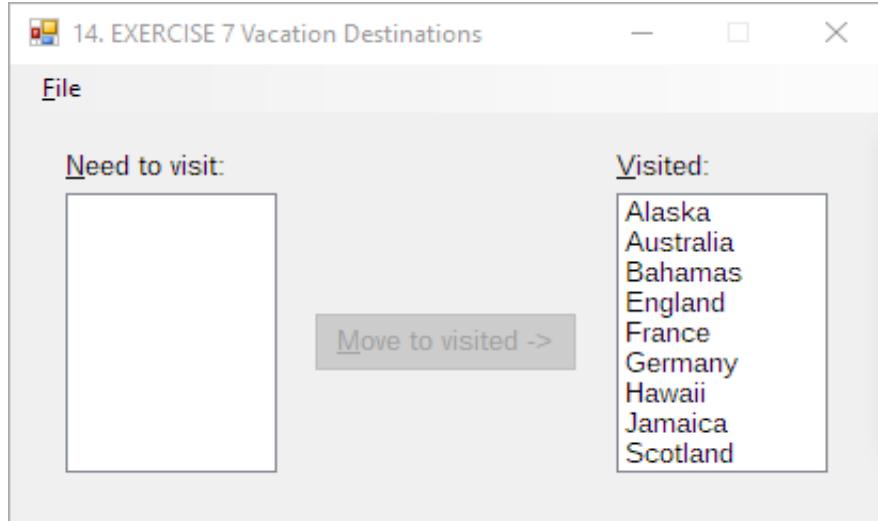
```
133     Me.Close()  
134 End Sub  
135  
136 End Class
```

\_original\_NeedToVisit.txt X

```
1 France  
2 England  
3 Jamaica  
4 Bahamas  
5 Hawaii  
6 Alaska  
7
```

\_original\_Visited.txt X

```
1 Germany  
2 Scotland  
3 Australia  
4
```



## Chap09\Exercise

### 15.Workers Solution-Filename\_EXERCISE 8\_advanced

8. In this exercise, you modify the application from Exercise 2. Use Windows to make a copy of the Workers Solution folder. Rename the copy Workers Solution-Filename.
  - a. Open the Workers Solution.sln file contained in the Workers Solution-Filename folder. Add a label and a text box to the form. Position both above the existing controls. (You will need to unlock the controls and then reposition the existing ones.) Change the new label's Text property to &Filename: Change the new text box's name to txtFilename. Lock the controls and then reset the tab order.
  - b. Open the Code Editor window. Code the txtFilename\_KeyPress procedure to accept only numbers, letters (uppercase and lowercase), and the Backspace key.
  - c. Modify the frmMain\_FormClosing procedure using the rules shown in Figure 9-40.
  - d. Save the solution and then start and test the application.

1. If the user does not provide a filename in the txtFilename control, the procedure should use a message box to display the "Please provide a filename." message along with an OK button and the Information icon; it should then prevent the form from being closed.
2. The procedure should append the .txt extension to the filename and then determine whether the file already exists. If the file does not exist, the procedure should write the list box information to the file. Use an independent Sub procedure to write the information to the file.
3. If the file exists, the procedure should use a message box to display the "Replace existing file before exiting?" message along with Yes, No, and Cancel buttons and the Exclamation icon.
4. If the user selects the Cancel button, the procedure should prevent the form from being closed. If the user selects the Yes button, the procedure should replace the file before the form is closed; use the independent Sub procedure from Step 2 to write the information to the file. If the user selects the No button, the procedure should close the form without saving the information.

Figure 9-40 Rules for Exercise 8

### 09.Workers Solution\_EXERCISE 2\_introductory

2. Open the Workers Solution.sln file contained in the VB2017\Chap09\Workers Solution folder. The user will enter a name in the txtName control and then click the Add to list button, which should add the name to the lstWorkers control. When the user is finished entering names, the frmMain\_FormClosing procedure should write the contents of the list box to a new sequential access file named workers.txt. Code the procedure. Save the solution and then start the application. Test the application by entering the following two names: Henry Kaplan and Mario Brown. Stop the application and verify that the workers.txt file contains both names. Now, start the application and enter the following three names: Jose Juarez, Maya Harris, and Savannah Carlisle. Stop the application and verify that the workers.txt file contains only three names.

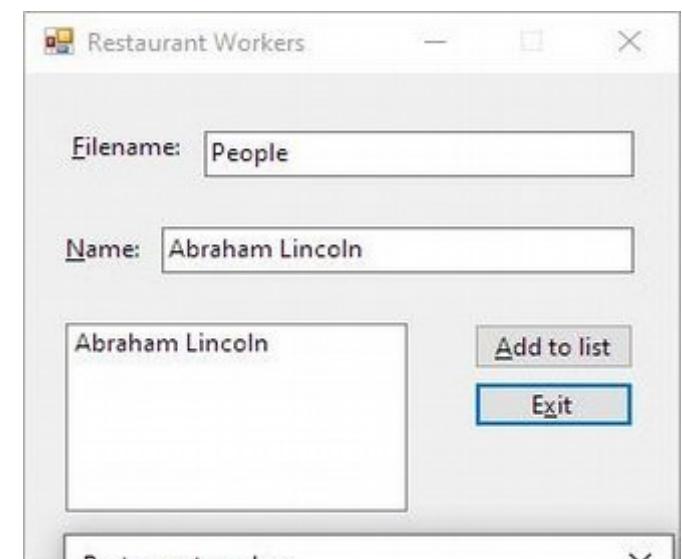
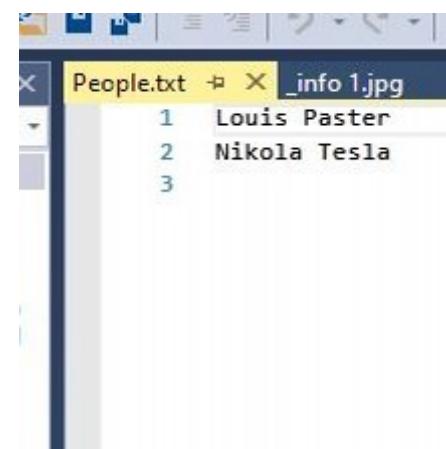
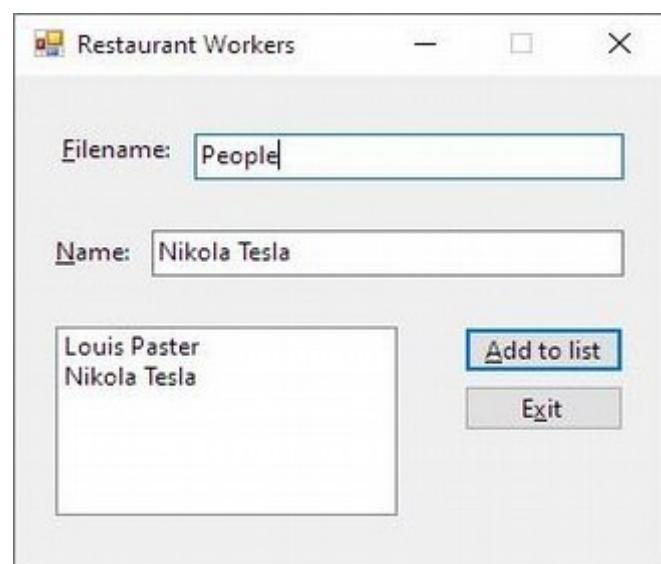
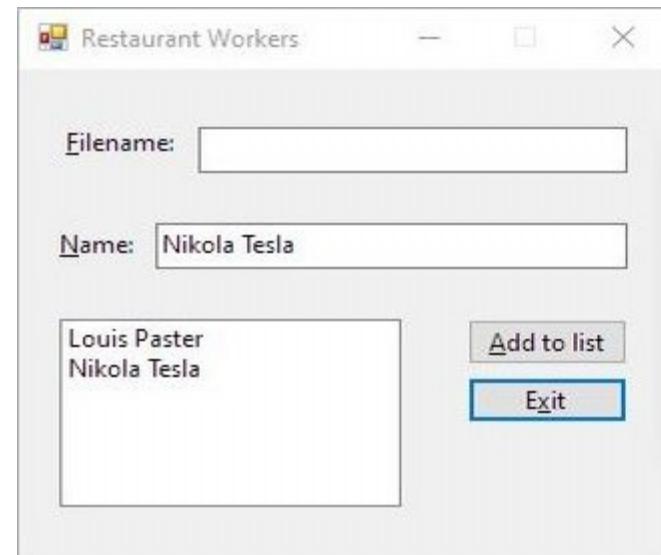
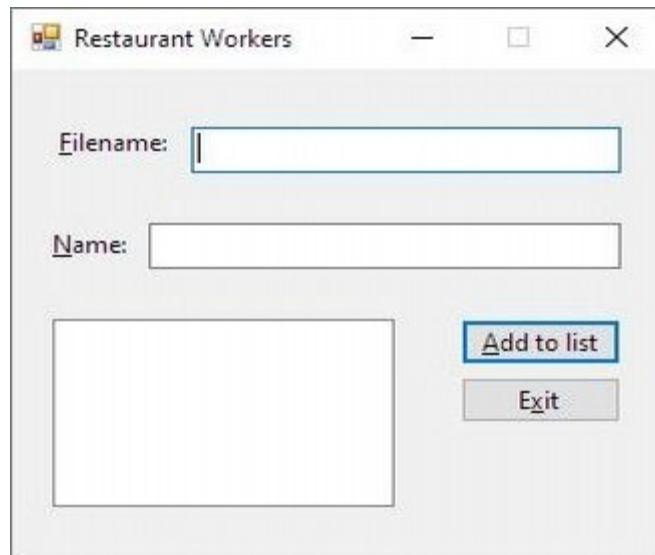
```
1  'ORIGINAL: 09. EXERCISE 2
2  ' Name:      Workers Project
3  ' Purpose:    Saves worker names to a sequential access files.
4  ' Programmer: <your name> on <current date>
5  '_user will: - enter a name in the (txtName) control &
6  '              - click the button (btnAdd) which should add the name to the (lstWorkers) control
7  '_when finished entering names:
8  ' - the frmMain_FormClosing procedure should write the contents Of the (lstWorkers) to a new sequential access file: "workers.txt"
9  '_test by entering 2 names: "Henry Kaplan" and "Mario Brown", stop the app and verify the contents of the "workers.txt"
10 ' <- it should countain 2 names
11 ' start the app again and:
12 ' _test by entering 3 names: "Jose Juarez", "Maya Harris", "Savannah Carlisle", stop the app and verify
13 ' <- it should contain only 3 names
14
15 'MODIFICATION: 15. EXERCISE 8
```



```
60         Select Case True
61             Case dlgButton = DialogResult.Yes
62                 WriteToFile()
63             Case dlgButton = DialogResult.No
64                 e.Cancel = False
65             Case dlgButton = DialogResult.Cancel
66                 e.Cancel = True
67         End Select
68
69     Else
70         WriteToFile()
71     End If
72 End If
73 End Sub
74
75 Private Sub WriteToFile()
76     outFile = IO.File.CreateText(strFilename)
77     For intCounter As Integer = 0 To lstWorkers.Items.Count - 1
78         outFile.WriteLine(lstWorkers.Items(intCounter))
79     Next intCounter
80     outFile.Close()
81 End Sub
82
83 Private Sub txtFilename_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtFilename.KeyPress
84     ' 2: code the "txtFilename_KeyPress" procedure to accept only numbers, uppercase & lowercase letters & the Backspace key
85     If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back AndAlso
86     (e.KeyChar < "a" OrElse e.KeyChar > "z") AndAlso (e.KeyChar < "A" OrElse e.KeyChar > "Z") Then
87         e.Handled = True
88     End If
89 End Sub
90
91 Private Sub btnAdd_Click(sender As Object, e As EventArgs) Handles btnAdd.Click
92     lstWorkers.Items.Add(txtName.Text)
93     txtName.Focus()
94     txtName.SelectAll()
95 End Sub
96
97 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
98     Me.Close()
99 End Sub
100
101 Private Sub txtName_Enter(sender As Object, e As EventArgs) Handles txtName.Enter
102     txtName.SelectAll()
103 End Sub
```

104

105 End Class



## Chap09\Exercise

### 16.Wedding Solution EXERCISE 9\_advanced

Create a Windows Forms application. Use the following names for the project and solution, respectively: Wedding Project and Wedding Solution. Save the application in the VB2017\Chap09 folder.

- a. Create a sequential access file named invited.txt and save it in the Wedding Project's bin\Debug folder. Enter the names of 20 guests in the file. Each guest's name should be entered on the same line in this format: last, first (the last name followed by a comma, a space character, and the first name). Close the invited.txt window.
- b. Create a second sequential access file named accepted.txt and save it in the Wedding Project's bin\Debug folder. Enter the following two names in the file: Nitzki, Akanna and Jefferson, Josephine. Close the accepted.txt file.
- c. Create a third sequential access file named rejected.txt and save it in the Wedding Project's bin\Debug folder. Enter the following name in the file: Kellog, Zelda. Close the rejected.txt file.

- d. The application's interface should contain three list boxes named lstInvited, lstAccepted, and lstRejected. Each list box's Sorted property should be set to True. The lstAccepted and lstRejected controls should have their SelectionMode property set to None. When the interface appears, the contents of the invited.txt, accepted.txt, and rejected.txt files should appear in the appropriate list boxes.
- e. The interface should also contain three buttons with the following captions: Accepted, Rejected, and Exit. The Accepted button should move the name selected in the lstInvited control to the lstAccepted control. The Rejected button should move the name selected in the lstInvited control to the lstRejected control.
- f. Before the form is closed, its FormClosing procedure should save the contents of each list box in the appropriate file.
- g. Save the solution and then start and test the application.

```
1  Option Explicit On
2  Option Strict On
3  Option Infer Off
4
5  'The application should:
6  'create GUI:
7  '-1. 3x (lst): (lstInvited); Sorted = True;SelectionMode = One.
8  '-      (lstAccepted); Sorted = True;SelectionMode = None.
9  '-      (lstRejected); Sorted = True;SelectionMode = None.
10 '-2. 3x (btn): "Accepted" = (btnAccepted)
11 '-          "Rejected" = (btnRejected)
12 '-          "Exit"     = (btnExit)
13
14 'Other:
15 '-1. create a Sequential access file: "invited.txt" and save it in the Wedding Project's bin\Debug folder
16 '- - enter the names of 20 guests in the file, each name in separate file, in the format: "LastName, FirstName"
17 '-2. create a Sequential access file: "accepted.txt" and save it in the Wedding Project's bin\Debug folder
18 '- - enter the following 2 names in the file: "Nitzki, Akanna", and "Jefferson, Josephine"
19 '-3. create a Sequential access file: "rejected.txt" and save it in the Wedding Project's bin\Debug folder
20 '- - enter the following name in the file: "Kellog, Zelda"
21
22 'Code:
23 '-1. when the GUI appears, each (lst) should be filled with a names from the appropriate Sequential Access Files
24
25 ' 2. (btnAccepted) should: move the name selected in (lstInvited) to the (lstAccepted)
```

= invited.txt = inFiles(0)  
= accepted.txt = inFiles(1)  
= rejected.txt = inFiles(2)

```
26
27  ' 3. (btnRejected) should: move the name selected in (lstInvited) to the (lstRejected)
28
29  ' 4. before the form is closed, its "FormClosing" procedure should: save the contents of each (lst) in the appropriate file
30  'that's all
31
32  Public Class frmMain
33      Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
34          ' 1. when the GUI appears, each (lst) should be filled with a names from the appropriate Sequential Access Files
35
36          'Invited()
37
38          Dim inFile(2) As IO.StreamReader
39
40          If IO.File.Exists("invited.txt") AndAlso IO.File.Exists("accepted.txt") AndAlso IO.File.Exists("rejected.txt") Then
41              'MessageBox.Show("OK")
42
43              inFile(0) = IO.File.OpenText("invited.txt")
44              inFile(1) = IO.File.OpenText("accepted.txt")
45              inFile(2) = IO.File.OpenText("rejected.txt")
46
47              Do Until inFile(0).Peek = -1
48                  lstInvited.Items.Add(inFile(0).ReadLine)
49              Loop
50              'lstInvited.SelectedIndex = 0
51              inFile(0).Close()
52
53              Invited()
54
55              Do Until inFile(1).Peek = -1
56                  lstAccepted.Items.Add(inFile(1).ReadLine)
57              Loop
58              inFile(1).Close()
59
60              Do Until inFile(2).Peek = -1
61                  lstRejected.Items.Add(inFile(2).ReadLine)
62              Loop
63              inFile(2).Close()
64
65          Else
66              MessageBox.Show("Can't locate the files.", "Wedding list", MessageBoxButtons.OK, MessageBoxIcon.Information)
67          End If
68
69      End Sub
```

```
70
71     Private Sub Invited()
72         If lstInvited.Items.Count = 0 Then
73             btnAccepted.Enabled = False
74             btnRejected.Enabled = False
75             'lstInvited.SelectedIndex = Nothing
76             lstInvited.SelectionMode = Nothing
77         End If
78     End Sub
79
80     Private Sub btnAccepted_Click(sender As Object, e As EventArgs) Handles btnAccepted.Click
81         ' 2. (btnAccepted) should: move the name selected in (lstInvited) to the (lstAccepted)
82
83         If lstInvited.SelectedIndex >= 0 Then
84             lstAccepted.Items.Add(lstInvited.SelectedItem)
85             lstInvited.Items.Remove(lstInvited.SelectedItem)
86         End If
87
88         Invited()
89
90     End Sub
91
92     Private Sub btnRejected_Click(sender As Object, e As EventArgs) Handles btnRejected.Click
93         ' 3. (btnRejected) should: move the name selected in (lstInvited) to the (lstRejected)
94
95         If lstInvited.SelectedIndex >= 0 Then
96             lstRejected.Items.Add(lstInvited.SelectedItem)
97             lstInvited.Items.Remove(lstInvited.SelectedItem)
98         End If
99
100        Invited()
101
102    End Sub
103
104    Private Sub frmMain_FormClosing(sender As Object, e As FormClosingEventArgs) Handles Me.FormClosing
105        ' 4. before the form is closed, its "FormClosing" procedure should: save the contents of each (lst) in the appropriate file
106        Dim outFiles(2) As IO.StreamWriter
107        outFiles(0) = IO.File.CreateText("invited.txt")
108        outFiles(1) = IO.File.CreateText("accepted.txt")
109        outFiles(2) = IO.File.CreateText("rejected.txt")
110
111        For intCounter As Integer = 0 To lstInvited.Items.Count - 1
112            outFiles(0).WriteLine(lstInvited.Items(intCounter))
113        Next
```

```

114
115     For intCounter As Integer = 0 To lstAccepted.Items.Count - 1
116         outFiles(1).WriteLine(lstAccepted.Items(intCounter))
117     Next
118
119     For intCounter As Integer = 0 To lstRejected.Items.Count - 1
120         outFiles(2).WriteLine(lstRejected.Items(intCounter))
121     Next
122
123     outFiles(0).Close()
124     outFiles(1).Close()
125     outFiles(2).Close()
126
127 End Sub
128
129 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
130     Me.Close()
131 End Sub
132
133 End Class

```

invited.txt - Notepad

File Edit Format View

Doe, John  
Brix, Steven  
Duerincx, Bill  
Yo, Mikel  
Panama, Elisabeth  
Potter, William  
Turek, Osvald  
Simala, Unfufu  
Peter, Peter  
File, Peter  
Zykel, Manfred  
Ricky, Raoul  
Beans, Muhamad  
Buddha, Gautama  
Mahaal, Tajin  
Kopec, Martin  
Baal, Sindy  
Cooper, Laszlo  
Stark, Timothy  
Brush, Anthony

accepted.txt - Notepad

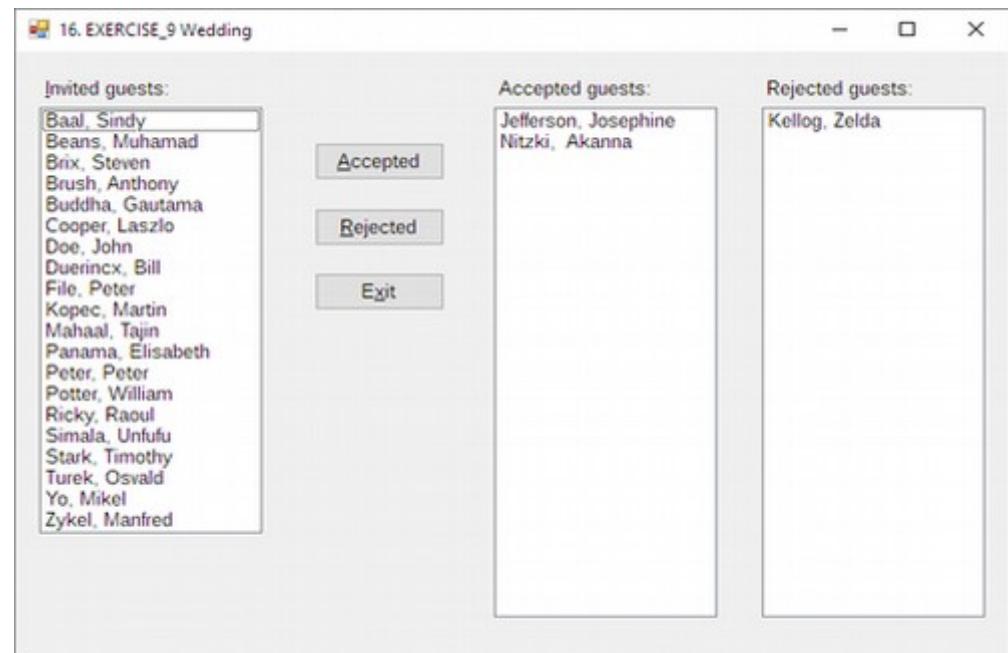
File Edit Format View

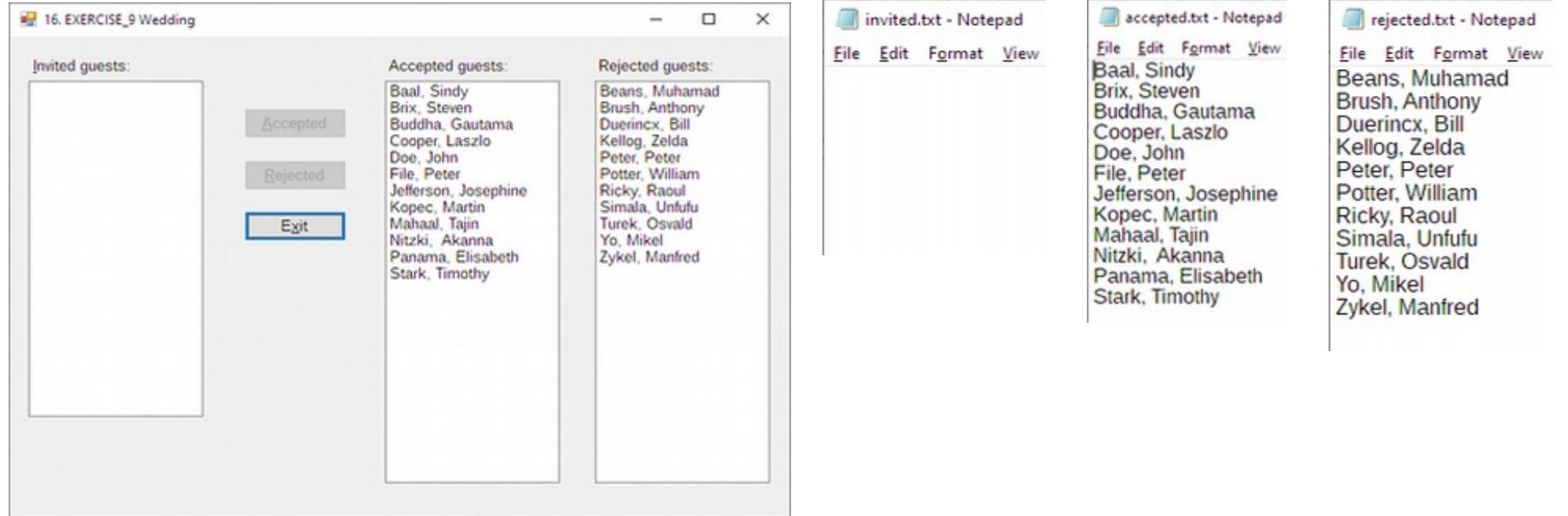
Nitzki, Akanna  
Jefferson, Josephine

rejected.txt - Notepad

File Edit Format View

Kellog, Zelda





invited.txt - Notepad

File Edit Format View

accepted.txt - Notepad

File Edit Format View

Baal, Sindy

Brix, Steven

Buddha, Gautama

Cooper, Laszlo

Doe, John

File, Peter

Jefferson, Josephine

Kopec, Martin

Mahaal, Tajin

Nitzki, Akanna

Panama, Elisabeth

Stark, Timothy

rejected.txt - Notepad

File Edit Format View

Beans, Muhamad

Brush, Anthony

Duerincx, Bill

Kellog, Zelda

Peter, Peter

Potter, William

Ricky, Raoul

Simala, Unfufo

Turek, Osvald

Yo, Mikel

Zykel, Manfred

## Chap09\Exercise1

### 17.OnYourOwn Solution\_EXERCISE 10

Create a Windows Forms application. Use the following names for the project and solution, respectively: OnYourOwn Project and OnYourOwn Solution. Save the application in the VB2017\Chap09 folder. Plan and design an application of your choice. The only requirement is that you must follow the minimum guidelines listed in Figure 9-41. Before starting the application, be sure to verify the name of the startup form. Save the solution and then start and test the application.

1. The user interface must contain a minimum of three labels and one button. It must also contain a File menu with an Exit option. You can include other menus and/or options as well.
2. The interface can include a picture box, but this is not a requirement.
3. The interface must follow the GUI design guidelines summarized for Chapters 2 through 9 in Appendix A.
4. Objects that are either coded or referred to in code should be named appropriately.
5. The application must use at least one input sequential access file and one output sequential access file.
6. The Code Editor window must contain comments, the three Option statements, at least two variables, at least two assignment statements, and the Me.Close() statement.

Figure 9-41 Guidelines for Exercise 10

not done, lack of idea

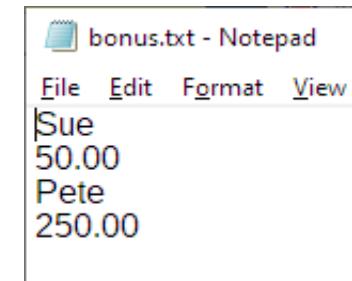
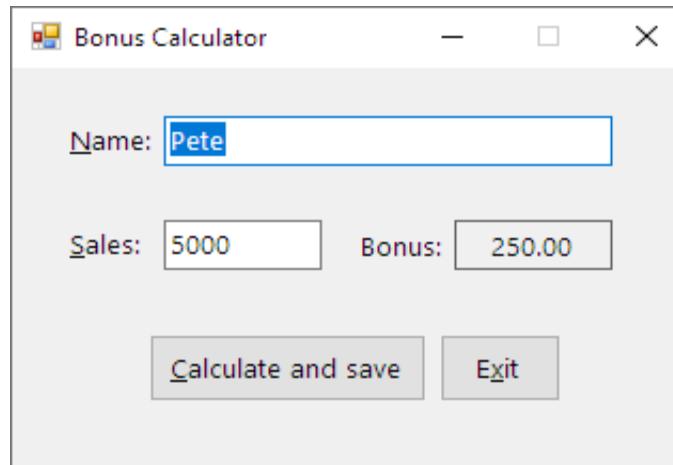
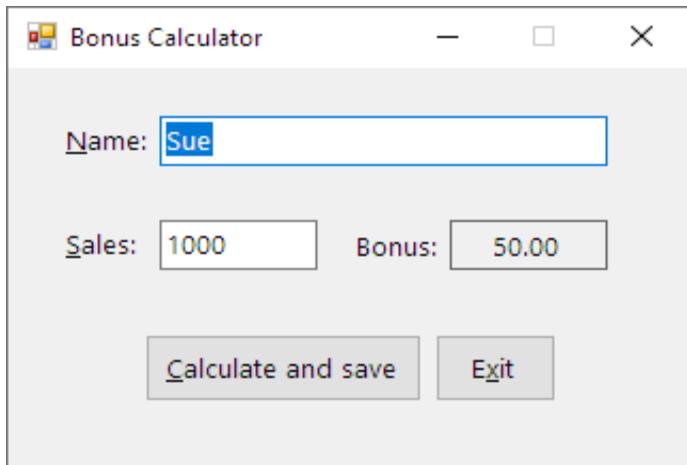
11. Open the VB2017\Chap09\FixIt Solution\FixIt Solution.sln file. Open the Code Editor window and study the existing code. Start the application. Enter Sue and 1000 and then click the Calculate and save button. Now, enter Pete and 5000 and then click the Calculate and save button. A run time error occurs. Read the error message. Click Debug on the menu bar and then click Stop Debugging. Open the bonus.txt file contained in the project's bin\Debug folder. Notice that the file is empty. Close the bonus.txt window. Locate and correct the error(s) in the code. Save the solution and then start and test the application again. Verify that the bonus.txt file contains the two names and bonus amounts.

```
1  ' Name:      FixIt Project
2  ' Purpose:    Calculates the bonus and then saves the name and bonus amount to a sequential access file.
3  ' Programmer: <your name> on <current date>
4
5  ' 1. study the existing code
6  ' 2. start the application and:
7  '    2.1. enter "Sue" and "1000" and then click the button: "Calculate and save"
8  '    2.2. enter "Pete" and "5000" and then click the button "Calculate and save"
9  '    <- a Runtime ERROR occurs, so read the error message, then click "Debug" on the menu bar and then click: "Stop Debugging"
10 '   3. open the file "bonus.txt" <- notice that the file is empty
11 '   4. locate and correct the error(errors) in the code
12 '   5. verify, that the file "bonus.txt" contains the 2 names and bonus amounts
13 'that's all
14
15 Option Explicit On
16 Option Strict On
17 Option Infer Off
18
19 Public Class frmMain
20     Private Sub btnCalcAndSave_Click(sender As Object, e As EventArgs) Handles btnCalcAndSave.Click
21         ' Writes the name and 5% bonus amount to a sequential access file.
22
23         Const dblBONUS_RATE As Double = 0.05
24         Dim outFile As IO.StreamWriter
25         Dim dblSales As Double
26         Dim dblBonus As Double
27
28         ' Assign sales amount to a variable.
29         Double.TryParse(txtSales.Text, dblSales)
30         ' Calculate and display bonus.
31         dblBonus = dblSales * dblBONUS_RATE
32         lblBonus.Text = dblBonus.ToString("N2")
```

```

33
34     ' Save name and bonus to the file.
35     outFile = IO.File.AppendText("bonus.txt")
36     outFile.WriteLine(txtName.Text)
37     outFile.WriteLine(lblBonus.Text)
38
39     'my fix:
40     outFile.Close()
41
42     txtName.Focus()
43 End Sub
44
45 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
46     Me.Close()
47 End Sub
48
49 Private Sub txtName_Enter(sender As Object, e As EventArgs) Handles txtName.Enter
50     txtName.SelectAll()
51 End Sub
52
53 Private Sub txtName_TextChanged(sender As Object, e As EventArgs) Handles txtName.TextChanged, txtSales.TextChanged
54     lblBonus.Text = String.Empty
55 End Sub
56
57 Private Sub txtSales_Enter(sender As Object, e As EventArgs) Handles txtSales.Enter
58     txtSales.SelectAll()
59 End Sub
60 End Class

```



## Shopping Cart (Chapters 1–9)

The shopping cart application should list the names of 10 different DVDs in a list box and store the associated prices in a one-dimensional array. (Use at least four different prices for the DVDs.) To purchase a DVD, the user needs to click its name in the list box and then click an Add to cart button. The button's Click event procedure should display the DVD's name and price in another list box, which will represent the shopping cart. A DVD can appear more than once in the shopping cart. The interface should also provide a Remove from cart button. (Research the Items collection's Remove and RemoveAt methods for a list box. Or, complete Exercise 17 in Chapter 5.) It should also provide a button that clears the shopping cart and prepares the application for the next customer's order. The application should display the subtotal (which is the cost of the items in the shopping cart), the sales tax, the shipping charge, and the total cost. The sales tax rate is 4%. The shipping charge is \$1 per DVD, up to a maximum shipping charge of \$5. (It may help to complete the Chapter 7: Aligning Columns section in Appendix B.)

```
1  ' Name:      Shopping cart.
2  ' Purpose:    Calculate and display the selected DVD's price.
3  ' Programmer: me on just now.
4  Option Explicit On
5  Option Strict On
6  Option Infer Off
7
8  Public Class frmMain
9      ' global variables:
10     Private decSubtotal As Decimal : Private intCounter As Integer : Private decPrices() As Decimal = {1.99D, 2.56D, 3.65D, 4.35D, 5.12D}
11
12     Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
13         ' used a monospace font "Courier New" to align easily:
14         lst1Shop.Items.Add(" 1000 nights" & "$".PadLeft(12) & decPrices(0))          ' index = 0, decPrices(0), $ = 1.99
15         lst1Shop.Items.Add(" Alien" & "$".PadLeft(18) & decPrices(1))           ' index = 1, decPrices(1), $ = 2.56
16         lst1Shop.Items.Add(" Bambi" & "$".PadLeft(18) & decPrices(2))           ' index = 2, decPrices(2), $ = 3.65
17         lst1Shop.Items.Add(" Conan the Barbarian" & "$".PadLeft(4) & decPrices(3))   ' index = 3, decPrices(3), $ = 4.35
18         lst1Shop.Items.Add(" Good, Bad, Ugly" & "$".PadLeft(8) & decPrices(1))       ' index = 4, decPrices(1), $ = 2.56
19         lst1Shop.Items.Add(" Lolita" & "$".PadLeft(17) & decPrices(0))           ' index = 5, decPrices(0), $ = 1.99
20         lst1Shop.Items.Add(" LOTR" & "$".PadLeft(19) & decPrices(3))           ' index = 6, decPrices(3), $ = 4.35
```

```

21     lst1Shop.Items.Add(" Platoon" & "$".PadLeft(16) & decPrices(1))           ' index = 7, decPrices(1), $ = 2.56
22     lst1Shop.Items.Add(" Rambo 19" & "$".PadLeft(15) & decPrices(4))           ' index = 8, decPrices(4), $ = 5.12
23     lst1Shop.Items.Add(" Shining" & "$".PadLeft(16) & decPrices(2))           ' index = 9, decPrices(2), $ = 3.65
24     lst1Shop.SelectedIndex = 4 : btnRemove.Enabled = False
25 End Sub
26
27 Private Sub btnAddToCart_Click(sender As Object, e As EventArgs) Handles btnAddToCart.Click
28     lst2Cart.Items.Add(lst1Shop.SelectedItem) : lst2Cart.Sorted = True : btnRemove.Enabled = True : lst2Cart.SelectedIndex = 0
29
30     ' Add the selected item's price to Subtotal:
31     Select Case True
32         Case lst1Shop.SelectedIndex = 0 Or lst1Shop.SelectedIndex = 5 : decSubtotal += decPrices(0)
33         Case lst1Shop.SelectedIndex = 1 Or lst1Shop.SelectedIndex = 4 Or lst1Shop.SelectedIndex = 7 : decSubtotal += decPrices(1)
34         Case lst1Shop.SelectedIndex = 2 Or lst1Shop.SelectedIndex = 9 : decSubtotal += decPrices(2)
35         Case lst1Shop.SelectedIndex = 3 Or lst1Shop.SelectedIndex = 6 : decSubtotal += decPrices(3)
36         Case lst1Shop.SelectedIndex = 8 : decSubtotal += decPrices(4)
37     End Select
38
39     ' Independent Sub - do the math and display:
40     Math()
41 End Sub
42
43 Private Sub btnRemove_Click(sender As Object, e As EventArgs) Handles btnRemove.Click
44     ' Subtract the selected item's price from Subtotal:
45     Select Case True
46         Case lst2Cart.SelectedItem.ToString Like "?1*" Or lst2Cart.SelectedItem.ToString Like "?Lol*": decSubtotal -= decPrices(0) ' $1.99
47         Case lst2Cart.SelectedItem.ToString Like "?A*" Or lst2Cart.SelectedItem.ToString Like "?G*" Or
48             lst2Cart.SelectedItem.ToString Like "?P*": decSubtotal -= decPrices(1) ' $2.56
49         Case lst2Cart.SelectedItem.ToString Like "?B*" Or lst2Cart.SelectedItem.ToString Like "?S*": decSubtotal -= decPrices(2) ' $3.65
50         Case lst2Cart.SelectedItem.ToString Like "?C*" Or lst2Cart.SelectedItem.ToString Like "?LOT*": decSubtotal -= decPrices(3) ' $4.35
51         Case lst2Cart.SelectedItem.ToString Like "?R*": decSubtotal -= decPrices(4) ' $5.12
52     End Select
53
54     ' Remove the selected item by its index:
55     lst2Cart.Items.RemoveAt(lst2Cart.SelectedIndex)
56
57     ' disable the button "Remove..." if there is no item, otherwise select the 1st item:
58     If lst2Cart.Items.Count = 0 Then : btnRemove.Enabled = False
59     Else : lst2Cart.SelectedIndex = 0
60     End If
61
62     ' Independent Sub - do the math and display:
63     Math()
64 End Sub

```

```
65
66  Private Sub Math()    ' Independent Sub - do the math and display:
67
68      ' sales Tax = 4% = 0.04
69      Dim decSalesTax As Decimal : decSalesTax = (decSubtotal * 0.04D) ' + decSubtotal
70
71      ' counter:
72      Dim intShipping As Integer : intCounter = lst2Cart.Items.Count
73
74      Select Case True
75          Case intCounter = 0 Or intCounter > 5 : intShipping = 0
76          Case Else : intShipping = intCounter
77      End Select
78
79      ' add all values together & display all:
80      lbl1Subtotal.Text = decSubtotal.ToString("C2")
81      lbl2SalesTax.Text = decSalesTax.ToString("C2")
82      lbl3Shipping.Text = intShipping.ToString("C2")
83      lbl4Total.Text = (decSubtotal + decSalesTax + intShipping).ToString("C2")
84  End Sub
85
86  Private Sub btnClear_Click(sender As Object, e As EventArgs) Handles btnClear.Click
87      lst2Cart.Items.Clear() : btnRemove.Enabled = False : decSubtotal = Nothing : intCounter = 0
88      lbl1Subtotal.Text = Nothing : lbl2SalesTax.Text = Nothing
89      lbl3Shipping.Text = Nothing : lbl4Total.Text = Nothing
90  End Sub
91
92  Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
93      Me.Close()
94  End Sub
95 End Class
```

09.Shopping Cart-CH01-09

Select a DVD:

|                     |        |
|---------------------|--------|
| 1000 nights         | \$1.99 |
| Alien               | \$2.56 |
| Bambi               | \$3.65 |
| Conan the Barbarian | \$4.35 |
| Good, Bad, Ugly     | \$2.56 |
| Lolita              | \$1.99 |
| LOTR                | \$4.35 |
| Platoon             | \$2.56 |
| Rambo 19            | \$5.12 |
| Shining             | \$3.65 |

Add to cart ->

Your cart:

|  |
|--|
|  |
|--|

Subtotal:

Sales tax:

Shipping:

Total cost:

Remove selected item   Clear   Exit

09.Shopping Cart-CH01-09

Select a DVD:

|                     |        |
|---------------------|--------|
| 1000 nights         | \$1.99 |
| Alien               | \$2.56 |
| Bambi               | \$3.65 |
| Conan the Barbarian | \$4.35 |
| Good, Bad, Ugly     | \$2.56 |
| Lolita              | \$1.99 |
| LOTR                | \$4.35 |
| Platoon             | \$2.56 |
| Rambo 19            | \$5.12 |
| Shining             | \$3.65 |

Add to cart ->

Your cart:

|                 |        |
|-----------------|--------|
| 1000 nights     | \$1.99 |
| 1000 nights     | \$1.99 |
| 1000 nights     | \$1.99 |
| Good, Bad, Ugly | \$2.56 |
| Platoon         | \$2.56 |
| Rambo 19        | \$5.12 |

Subtotal:  \$ 16.21

Sales tax:  \$ 0.65

Shipping:  \$ 0.00

Total cost:  \$ 16.86

Remove selected item   Clear   Exit

09.Shopping Cart-CH01-09

Select a DVD:

|                     |               |
|---------------------|---------------|
| 1000 nights         | \$1.99        |
| Alien               | \$2.56        |
| Bambi               | \$3.65        |
| Conan the Barbarian | \$4.35        |
| Good, Bad, Ugly     | \$2.56        |
| Lolita              | \$1.99        |
| LOTR                | \$4.35        |
| Platoon             | \$2.56        |
| Rambo 19            | <b>\$5.12</b> |
| Shining             | \$3.65        |

Add to cart ->

Your cart:

|             |               |
|-------------|---------------|
| 1000 nights | \$1.99        |
| 1000 nights | \$1.99        |
| Platoon     | \$2.56        |
| Rambo 19    | <b>\$5.12</b> |

Subtotal: **\$ 11.66**

Sales tax: **\$ 0.47**

Shipping: **\$ 4.00**

Total cost: **\$ 16.13**

Remove selected item      Clear      Exit

09.Shopping Cart-CH01-09

Select a DVD:

|                     |               |
|---------------------|---------------|
| 1000 nights         | \$1.99        |
| Alien               | \$2.56        |
| Bambi               | \$3.65        |
| Conan the Barbarian | \$4.35        |
| Good, Bad, Ugly     | \$2.56        |
| Lolita              | \$1.99        |
| LOTR                | \$4.35        |
| Platoon             | \$2.56        |
| Rambo 19            | <b>\$5.12</b> |
| Shining             | \$3.65        |

Add to cart ->

Your cart:

|          |               |
|----------|---------------|
| Rambo 19 | <b>\$5.12</b> |
|----------|---------------|

Subtotal: **\$ 5.12**

Sales tax: **\$ 0.20**

Shipping: **\$ 1.00**

Total cost: **\$ 6.32**

Remove selected item      Clear      Exit

## Airplane Seats (Chapters 1–9)

Create an interface that contains a list box with the following 18 items: 1A, 1B, 1C, 2A, 2B, 2C, 3A, 3B, 3C, 4A, 4B, 4C, 5A, 5B, 5C, 6A, 6B, and 6C. Each item represents a seat designation on an airplane. When the user clicks a list box item, the application should display the seat designation, passenger's name, and ticket price. The application should use both a sequential access file and a two-dimensional array for the passenger information. You can create the sequential access file by using the New File option on the File menu. The file should contain 18 seat designations, passenger names, and ticket prices.

```
1  ' Name:      Airplane Seats.
2  ' Purpose:    a
3  ' Programmer: Me on just now.
4  Option Explicit On
5  Option Strict On
6  Option Infer Off
7
8  Public Class frmMain
9
10   ' 17,2
11  Private str2DInfo() As String =
12  {{"1A", "Federico Ocampo", "$25.60"}, {"1B", "Joel Maharashi", "$25.60"}, {"1C", "Peter Black", "$25.60"},
13  {"2A", "Jaralala Muhato", "$27.90"}, {"2B", "Josephine Mutti", "$27.90"}, {"2C", "Alex Submin", "$27.90"},
14  {"3A", "Anna Pemba", "$29.99"}, {"3B", "Ricardo Blemba", "$29.99"}, {"3C", "Louis Blierott", "$29.99"},
15  {"4A", "Leon Burda", "$31.45"}, {"4B", "Albert Cvaistain", "$31.45"}, {"4C", "Michelle Pavarotti", "$31.45"},
16  {"5A", "Jack Kerouack", "$33.68"}, {"5B", "Martin Falang", "$33.68"}, {"5C", "Sophia Lornon", "$33.68"},
17  {"6A", "Victor Hugo", "$36.90"}, {"6B", "Emanuel Butterfly", "$36.90"}, {"6C", "Helenne Salamander", "$36.90"}}
18
19  Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
20      ' lstSeat.SelectedIndex = 0 -> 17 = OK
21
22      ' fill the lstSeat by a hand:
23      'For intCounter As Integer = 1 To 6
24      'lstSeat.Items.Add(intCounter & "A") : lstSeat.Items.Add(intCounter & "B") : lstSeat.Items.Add(intCounter & "C")
25      'Next intCounter
26      'lstSeat.SelectedIndex = 0
```

```

27
28     ' fill the lstSeat with a 2D array:
29     For intIndex As Integer = 0 To 17
30         lstSeat.Items.Add(str2DInfo(intIndex, 0))
31     Next intIndex
32     lstSeat.SelectedIndex = 0
33 End Sub
34
35 Private Sub lstSeat_SelectedIndexChanged(sender As Object, e As EventArgs) Handles lstSeat.SelectedIndexChanged
36     lbl2Display.Text = Nothing
37     Dim inFile As IO.StreamReader
38
39     If IO.File.Exists("SeatsInfo.txt") Then
40         'MessageBox.Show("Yees") ' file location: ...\\bin\\Debug
41
42         ' 1. use a Sequential Access File to fill the 1st lbl:
43         inFile = IO.File.OpenText("SeatsInfo.txt")
44         Dim strFirstLine As String : Dim strSecondLine As String
45
46         Do Until inFile.Peek = -1
47             strFirstLine = inFile.ReadLine
48             strSecondLine = inFile.ReadLine
49
50             ' instructions here:
51             If strFirstLine Like lstSeat.SelectedItem.ToString Then
52                 'MessageBox.Show(strFirstLine & strSecondLine & inFile.ReadLine) ' showing: seat name price =OK juchuuu
53                 lbl1Display.Text = strFirstLine & " " & strSecondLine & " " & inFile.ReadLine ' =OK juchuuu
54             End If
55         Loop
56
57         ' close the file:
58         inFile.Close()
59
60         ' 2. use a 2D array to fill the 2nd lbl:
61         For intIndex As Integer = 0 To 2
62             lbl2Display.Text &= str2DInfo(lstSeat.SelectedIndex, intIndex) & " "
63         Next intIndex
64
65         Else ' if the SeatsInfo.txt doesn't exists:
66             MessageBox.Show("Can't locate the SeatsInfo.txt file.", "Airplane Seats", MessageBoxButtons.OK, MessageBoxIcon.Error)
67         End If
68     End Sub
69

```

```

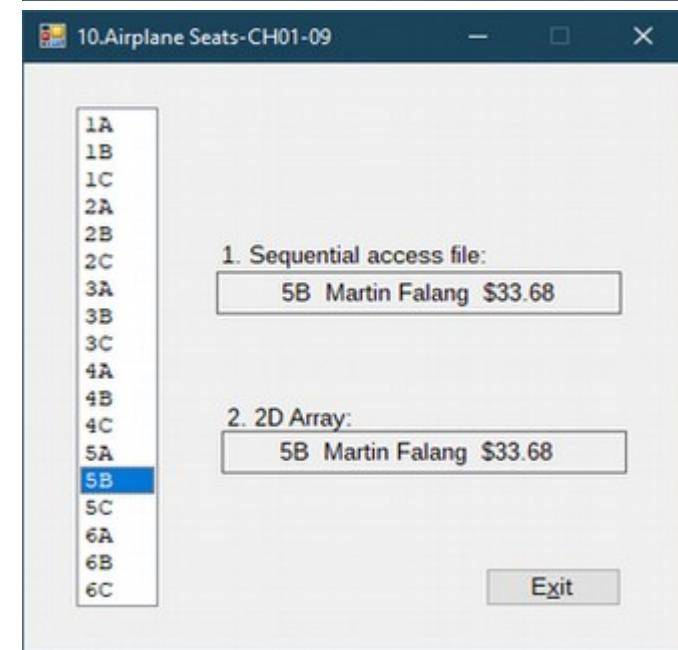
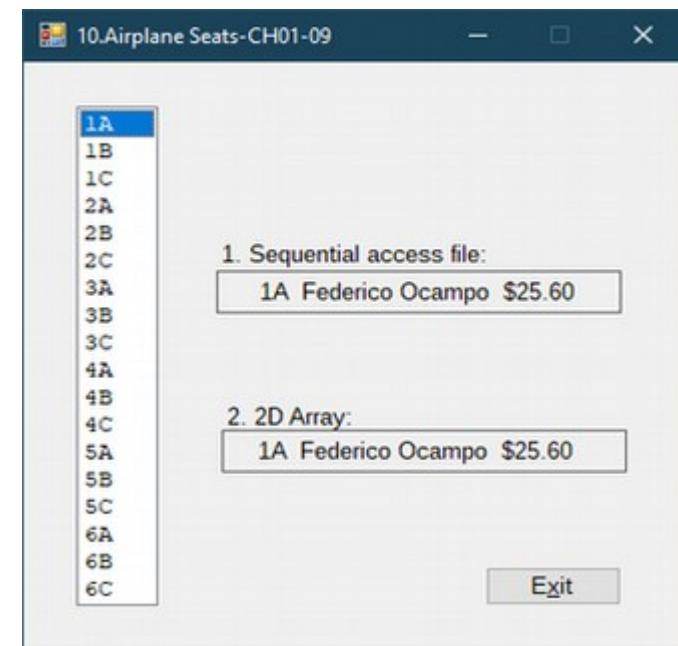
70      Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
71          Me.Close()
72      End Sub
73  End Class

```

...\\bin\\Debug\\SeatsInfo.txt

SeatsInfo.txt

|    |                    |
|----|--------------------|
| 1  | 1A                 |
| 2  | Federico Ocampo    |
| 3  | \$25.60            |
| 4  |                    |
| 5  | 1B                 |
| 6  | Joel Maharashi     |
| 7  | \$25.60            |
| 8  |                    |
| 9  | 1C                 |
| 10 | Peter Black        |
| 11 | \$25.60            |
| 12 |                    |
| 13 | 2A                 |
| 14 | Jaralala Muhato    |
| 15 | \$27.90            |
| 16 |                    |
| 17 | 2B                 |
| 18 | Josephine Mutti    |
| 19 | \$27.90            |
| 20 |                    |
| 21 | 2C                 |
| 22 | Alex Submin        |
| 23 | \$27.90            |
| 24 |                    |
| 25 | 3A                 |
| 26 | Anna Pemba         |
| 27 | \$29.99            |
| 28 |                    |
| 29 | 3B                 |
| 30 | Ricardo Blemba     |
| 31 | \$29.99            |
| 32 |                    |
| 33 | 3C                 |
| 34 | Louis Bleriot      |
| 35 | \$29.99            |
| 36 |                    |
| 37 | 4A                 |
| 38 | Leon Burda         |
| 39 | \$31.45            |
| 40 |                    |
| 41 | 4B                 |
| 42 | Albert Cvaistain   |
| 43 | \$31.45            |
| 44 |                    |
| 45 | 4C                 |
| 46 | Michelle Pavarotti |
| 47 | \$31.45            |
| 48 |                    |
| 49 | 5A                 |
| 50 | Jack Kerouack      |
| 51 | \$33.68            |
| 52 |                    |
| 53 | 5B                 |
| 54 | Martin Falang      |
| 55 | \$33.68            |
| 56 |                    |
| 57 | 5C                 |
| 58 | Sophia Lornon      |
| 59 | \$33.68            |
| 60 |                    |
| 61 | 6A                 |
| 62 | Victor Hugo        |
| 63 | \$36.90            |
| 64 |                    |
| 65 | 6B                 |
| 66 | Emanuel Butterfly  |
| 67 | \$36.90            |
| 68 |                    |
| 69 | 6C                 |
| 70 | Helenne Salamander |
| 71 | \$36.90            |



- you already are familiar with **Classes** and **Objects** because you have been using them since CH1 ->
  - > e.g. you used VB's **Class Button** to create a **button Object** and used its **Class String** to create a **String** variable
- in this chapter's Focus on the Concepts lesson, you will learn how to define your own **Classes** and then use them to instantiate **Objects** in an application
- the Apply the Concepts lesson expands on the topics covered in the Focus lesson

## CH10\_FOCUS ON THE CONCEPTS LESSON

- CH10\_F1** - Object-Oriented Programming / OOP basic info and e.g.: **Class** and instance of the **Class** called **Object**
- CH10\_F2** - creating a **Class** - basic info
- CH10\_F2.1** - creating and adding a **Class file** to an open project: **Class's member** variables & **Property** section & **Behavior** section basic info
- CH10\_F3** - **Class's member** variables: info & syntax ; **Property** section of a **Class**: **Property** procedures: info & syntax
- CH10\_F3.1** - **Property** section of a **Class** example: creating the **Rectangle Class** for the **Franklin Decks app** (01.Franklin Solution) **1/3**
- CH10\_F4** - **Behavior** section of a **Class** info: usually **methods**: **Sub** procedures or **Functions**
- CH10\_F4.1** - **Behavior** section of a **Class**: **Sub** procedure method **Constructor** (default & parameterized) - info & syntax & e.g.
- CH10\_F4.2** - **Behavior** section of a **Class**: other **methods** than **Constructor** - **Function** procedure, **Sub** procedure - infos & syntaxes & examples
- CH10\_F4.3** - **Behavior** section of a **Class** example: complete the **Rectangle Class** for the: **Franklin Decks app** (01.Franklin Solution) **2/3**
- CH10\_F5** - instantiating an **Object** and invoking the **constructor** - info, syntax and example
- CH10\_F5.1** - using the **Rectangle Class** in the application example: pseudocode & GUI for the: **Franklin Decks app** (01.Franklin Solution) incl. entire code **3/3**
- CH10\_F6** - **You Do It 1:** create and use **Class** exercise: **02.You Do It 1 Solution** -> **Public Property**, default **constructor**, **function**
- CH10\_F7** - adding a **parameterized Constructor** example: a **Rectangle Class** of **03.Franklin Solution-Parameterized** **1/5**
- CH10\_F7.1** - comparing **default** vs **parameterized Constructor** example: **Rectangle Class** of the **03.Franklin Solution-Parameterized** **2/5**
- CH10\_F7.2** - using/invoking a **parameterized Constructor** example: a **btnCalc\_Click** procedure of **03.Franklin Solution-Parameterized** **3/5**
- CH10\_F7.3** - analyse the process of using/invoking a **parameterized Constructor**: a **btnCalc\_Click** procedure of **03.Franklin Solution-Parameterized** **4/5**
- CH10\_F7.4** - the entire code for a **parameterized Constructor** example and testing: **03.Franklin Solution-Parameterized** **5/5**
- CH10\_F8** - **You Do It 2:** create and use **Class** exercise: **04.You Do It 2 Solution** -> **Public Property**, parameterized **constructor**, **function**
- CH10\_F9** - reusing a **Class** example: previously created **Class Rectangle** used in: **05.Pizzeria Solution** - GUI & pseudocode & entire code & test

## CH10\_APPLY THE CONCEPTS LESSON

- CH10\_A1** - use a **ReadOnly Property** procedure in a **Class** example: complete the **CourseGrade Class** used in: **06.Grade Solution** **1/2**
- CH10\_A1.1** - use a **ReadOnly Property** procedure in a **Class** example: complete the **frmMain Class** used in: **06.Grade Solution** **2/2**
- CH10\_A2** - create **auto-implemented Public Property**: it automatically creates hidden **Private** member variable: **\_variable** & hidden **Get** & **Set** block of codes
- CH10\_A2.1** - use **auto-implemented Public Property**: example with **CourseGrade Class** used in **07.Grade Solution-autoimplemented**
- CH10\_A3** - **You Do It 3:** create and use **auto-implemented Public Property** exercise: **08.You Do It 3 Solution**
- CH10\_A4** - use an **overload method** / a method with several signatures in a **Class** - info and example with **Employee Class**
- CH10\_A4.1** - use an **overload method** / a method with several signatures in a **Class** - example with **Employee Class** used in **09.Woods Solution**

**CH10\_Summary:** **Class** statement, instantiating an **Object** from a **Class**, **Property Procedure**, default and parameterized **Constructor**, invoking the **Constructor**, other methods than **Constructor** - **Function** procedure & **Sub** procedure, **auto-implemented Public Property**, **overload** the methods.

**CH10\_Key Terms**

**CH10\_Exercises**

## CH10\_FOCUS ON THE CONCEPTS LESSON

### CH10\_F1 - Object-Oriented Programming / OOP basic info and e.g.: Class and instance of the Class called Object

- as you learned in CH1, Visual Basic is an **object-oriented programming language / OOP** allows the coder to use **Objects** in app's GUI and also in its code
- in **OOP**, an **Object** is anything that can be seen, touched, or used -> an **Object** is nearly any thing
- the **Objects** used in **OOP** can take on many different forms:
  - TextBox (**txt**), ListBox (**lst**), Label (**lbl**), Button (**btn**)
  - variables & named constants declared in an application's code
  - Sequential Access **I/O** files
- every **Object** in **OOP** is created from a **Class** = pattern that the computer uses to create the **Object**
  - contains the instructions that tell the computer how the **Object** should look and behave
- an **Object** created from a **Class** is called an **Instance of the Class** and is said to be **instantiated from the Class**
  - e.g. Button control (**btn**) is an instance of the **Button Class** and is instantiated when you drag the Button tool from the **Toolbox** to the form
  - e.g. a String variable is an instance of the **String Class** and is instantiated the first time you refer to the variable in code
- keep in mind that **Class** itself is not an **Object** - only an instance of a **Class** is an **Object**
- a **Class** contains - or in OOP terms - it **encapsulates** all of the **attributes** and **behaviours** of the **Object** it instantiates
- the term **encapsulates** means to enclose in a capsule
- in the context of OOP, the "**capsule**" is a **Class**
- every **Object** has:
  - set of **attributes/properties** = characteristics that describe the **Object**
    - e.g. **(Name)** and **Text** properties of the **(btn)**
    - e.g. **(Name)** and **Text** properties of the **(txt)**
    - e.g. **Length** property of the String variables
  - set of **behaviors** which include:
    - **methods/functions** = operations/actions that the **Object** is capable of performing
      - e.g. **(btn)** can use its method **Focus** to send the focus to itself
      - e.g. String variable can use its **ToUpper** method to temporarily convert its contents to uppercase
    - **events** = actions to which an **Object** can respond
      - e.g. button's **Click** event allows the button to respond to a mouse click

#### Mini-Quiz 10-1:

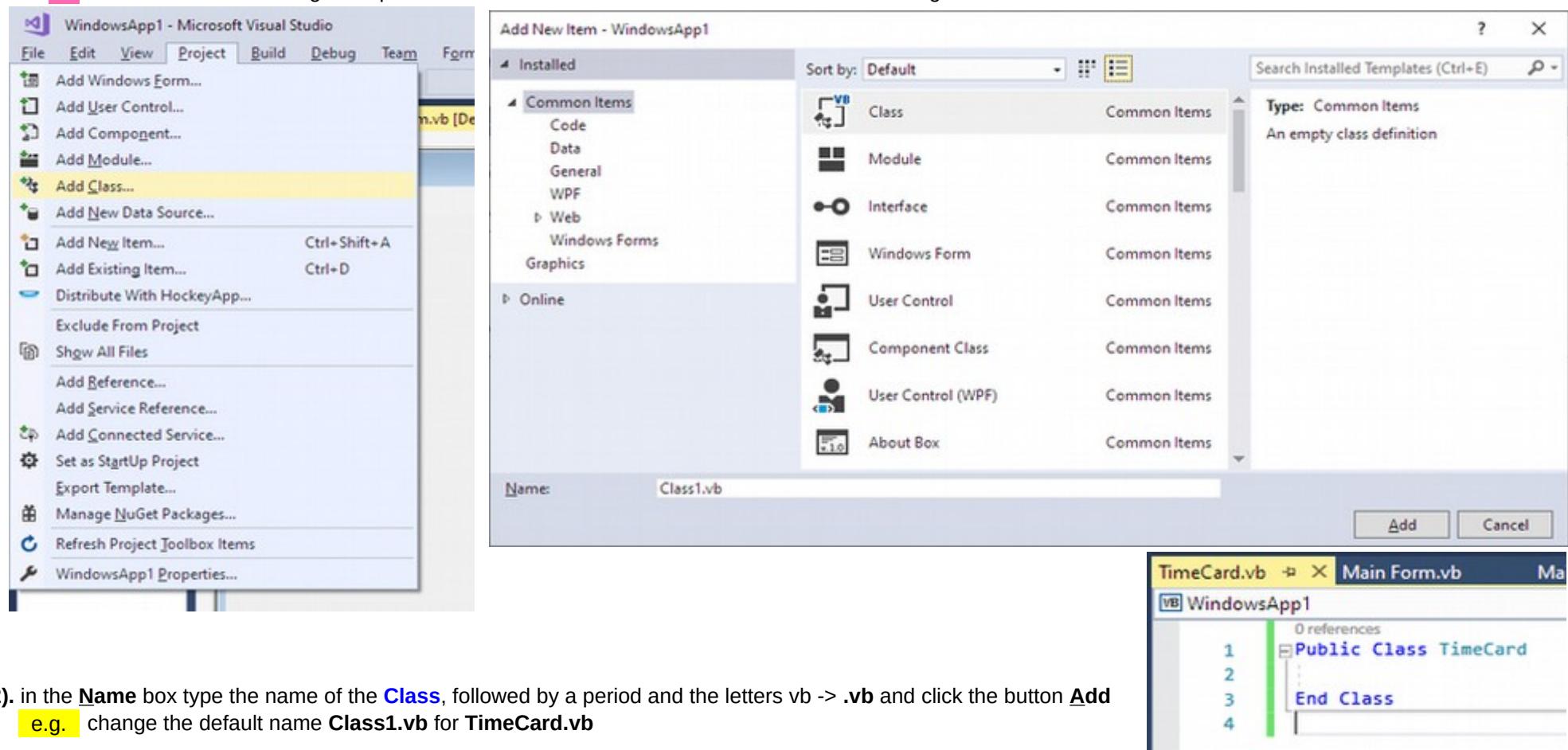
1. A class is an object. True or False?
2. An object is an instance of a class. True or False?
3. An object's attributes indicate the tasks that the object can perform. True or False?
4. What is a class?

## CH10\_F2 - creating a **Class** - basic info

- in previous Chapters, you instantiated **Objects** using **Classes** that are built into Visual Basic, such as **TextBox Class**, **Label Class**, etc...
  - you used the instantiated **Objects** in a variety of ways in many different applications
    - e.g. in some apps you used a **TextBox** to enter a name, while in others you used it to enter a sales tax rate
    - e.g. you used **Label** controls to identify **TextBoxes** and also to display the result of calculations
  - the ability to use an **Object** for more than one purpose saves programming time and money - an advantage that contributes to the **OOP**
- 
- you can also define your own **Classes** in Visual Basic and then create instances/**Objects** from those **Classes**
  - creating a **Class** whose **Objects** can be used in a variety of ways by many different apps requires a lot of planning
  - however, the time and effort spent planning the **Class** will pay off in the long run

## CH10\_F2.1 - creating and adding a **Class** file to an open project: **Class's member variables & Property section & Behavior section** basic info

- 1). -> on a Visual Studio's menu bar click **Project** and select **Add Class...**  
<- the **Add New Item** dialog box opens with **Class** selected in the middle column of the dialog box



3). you define a **Class** using the **Class statement**, which you enter in a **Class file**

**Class** statement syntax:

```
Public Class YourClassName
    1. Class's member variable
    2. property section
    3. behavior section
End Class
```

**CH10\_F3 - Class's member variables: info & syntax ;**

**Property** section of a **Class**: **Property** procedures: info & syntax

**CH10\_F4 - Behavior** section of a **Class** info: usually **methods**: **Sub** procedures or **Functions**

- Class** - a keyword  
= declares the name of a **Class** and introduces the definitions of the variables, properties, and methods that make up the **Class**  
**YourClassName**  
- **Class YourSolutionName.YourClassName**  
    e.g. **Class WindowsApp1.TimeCard**  
- not a requirement, but the convention is to use **Pascal** case for the **Class** name, like: **TextBox**, **String**, etc...  
= associated with the **Property** procedure  
- each **member variable** represents a **Property** of the **Object** that the **Class** will create  
- most of the **member variables** are declared like **Private**  
- declared before **Public Property** procedure

**CH10\_F3 - Class's member variables: info & syntax ; Property section of a Class: Property procedures: info & syntax**

- = within the **Class** statement, you define the attributes/properties of the Objects the **Class** will create  
- in most cases are represented by **Private** variables and **Public Property** procedures

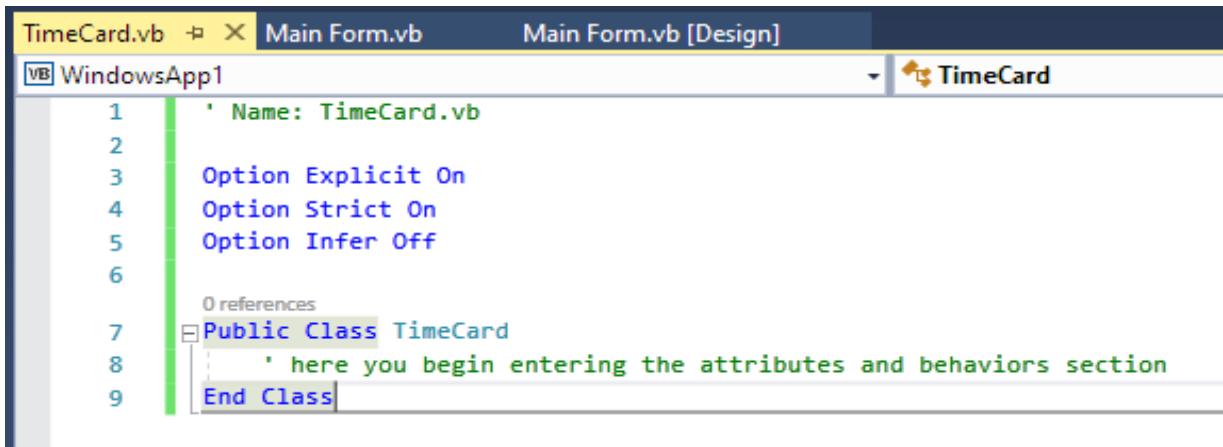
**CH10\_F3 - Class's member variables: info & syntax ; Property section of a Class: Property procedures: info & syntax**

- = within the **Class** statement, you define the behaviors of the Objects the **Class** will create  
- represented by **methods**, which are usually **Sub procedures** or **Functions**  
- you can also include **Event procedures** - however, that topic is beyond the scope of this book

**CH10\_F4 - Behavior section of a Class info: usually methods: Sub procedures or Functions**

e.g. **Class** file named **TimeCard.vb**

- the 3 **Option** statements in the **TimeCard.vb** file have the same meaning as they have in a **Main Form** file



```
TimeCard.vb  ✎ X  Main Form.vb  Main Form.vb [Design]
VB WindowsApp1
1 ' Name: TimeCard.vb
2
3 Option Explicit On
4 Option Strict On
5 Option Infer Off
6
7 0 references
8 □ Public Class TimeCard
9     ' here you begin entering the attributes and behaviors section
10 End Class
```

## CH10\_F3 - Class's member variables: info & syntax ; Property section of a Class: Property procedures: info & syntax

- when the **Class** is used to instantiate an object, each object will have its own copy of the **member variables** and **Public Properties** defined in the **Class**'s property section

## CH10\_F5 - instantiating an Object and invoking the constructor - info, syntax and example

- within the **Class** statement, you define the attributes/properties of the Objects the **Class** will create

- in most cases are represented by **Private** variables and **Public Property** procedures

- typically contains the declaration statements for one or more variables, referred to as **member variables**

- each **member variable** represents a **Property** of the **Object** that the **Class** will create

- most of the **member variables** are declared using the keyword **Private**

- when an application uses the **Class** to instantiate an Object, the **Private member variables** will not be visible to the application -> they will be hidden from it

- for an application to assign data to or retrieve data from a **Private member variable**, it must use a **Public Property**

- this is because only items declared using the keyword **Public** will be visible to the application -> they will be exposed to it

- i.e. an application cannot directly refer to a **Private member** in a **Class** -> instead, it must refer to the member, indirectly, through the use of a **Public member**

- **Public Property** procedure allows an application to refer to a **Private member variable** in a **Class**

Class's member variable syntax:

```
Private/... yourMemberVariable As dataType
```

<- associated with the:  
- property section and  
- behavior section

Public Property procedures syntax:

```
Public ReadOnly/WriteOnly Property YourPropertyName(parameterList) As dataType

    Get
        optional instructions
        Return yourMemberVariable
    End Get

    Set(value As dataType)
        optional instructions
        yourMemberVariable = value/defaultValue
    End Set

End Property
```

<- procedure's header

<- nothing or **ReadOnly**

<- nothing or **WriteOnly**

<- an instruction

<- procedure's footer

*yourMemberVariable* = associated with the **Property** procedure

- each **member variable** represents a **Property** of the **Object** that the **Class** will create

- most of the **member variables** are declared like **Private**

- declared before **Public Property** procedure

procedure's header:

- Public Property** = a keyword, in most cases procedure header beginning, but can contain optional keywords **ReadOnly** or **WriteOnly**
- you include both:
    - **Get** block of code to retrieve/read and
    - **Set** block of code to set/write
  - if the header doesn't contain the **ReadOnly** or **WriteOnly** keywords, you include both **Get & Set blocks of codes** in the procedure
- e.g. 1: **Public Property** - an application can both retrieve/read and set/write the **Side Property's value**

```
Private intSide As Integer  
  
Public Property Side As Integer  
    Get  
        Return intSide  
    End Get  
    Set(value As Integer)  
        If value > 0 Then  
            intSide = value  
        Else  
            intSide = 0  
        End If  
    End Set  
End Property
```

<- member variable associated with the **Property**

#### ReadOnly

- **optional**
  - = indicates, that the **Property's value** can be retrieved/read by an application -> but the application cannot set/write to the **Property**
  - the **Property** would get its **value** from the **Class** itself rather than from the application
  - you include only a **Get block of code** to retrieve/read
- e.g. 2: **Public ReadOnly Property** - an app can only retrieve/read the **Bonus Property's value**

- it cannot set/write the **Property's value** -> can be set/written only by a procedure within the **Class**

```
Private dblBonus As Double  
  
Public ReadOnly Property Bonus As Double  
    Get  
        Return dblBonus  
    End Get  
End Property
```

#### WriteOnly

- **optional**
  - = indicates, that the application can set/write the **Property's value** -> but it cannot retrieve/read the **value**
  - the **Property's value** would be set/written by the application for use within the **Class**
  - you include only a **Set block of code** to set/write
- e.g. 3: **Public WriteOnly Property** - an app can only set/write the **AnnualSales Property's value**

- it cannot retrieve/read the **Property's value** -> only a procedure within the **Class** can

```
Private decAnnualSales As Decimal  
  
Public WriteOnly Property AnnualSales As Decimal  
    Set(value As Decimal)  
        decAnnualSales = value
```

|                                                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |                                       |
|---------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------|
|                                                         |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 | <b>End Set</b><br><b>End Property</b> |
| <i>YourPropertyName</i><br><br>( <i>parameterList</i> ) | = name of your <b>Property</b><br>- you should use nouns/substantiv and adjective using Pascal case - e.g.: Side, Bonus, AnnualSales...<br>- <b>optional</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |                                       |
| <b>As dataType</b>                                      | - <b>Property</b> 's data type must match the data type of <i>yourMemberVariable</i> associated with the <b>Property</b> procedure -><br>-> <i>yourMemberVariable</i> declared before <b>Public Property</b> procedure                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |                                       |
| procedure's body:<br><b>Get</b> block of code           | <pre style="background-color: #d9ffd9; padding: 10px;"> <b>Get</b>   optional instructions   <b>Return</b> <i>yourMemberVariable</i> <b>End Get</b> </pre> <p>e.g. 1: <b>Public Property</b>    e.g. 2: <b>Public ReadOnly Property</b></p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                     |                                       |
|                                                         | - can be used only alongside with an <b>optional Readonly</b> keyword, or without any <b>optional</b> keywords in the procedure header -><br>-> <b>Public Property</b> , or <b>Public Readonly Property</b><br>- contains the <b>Get</b> statement, which begins with the <b>Get</b> clause and ends with the <b>End Get</b> clause<br>- most of the times, you enter only the <b>Return</b> <i>yourMemberVariable</i> instruction within the <b>Get</b> statement                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                              |                                       |
| <b>Return</b>                                           | - contained code returns the <b>value</b> of <i>yourMemberVariable</i> associated with the <b>Property</b>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                      |                                       |
| <b>Set</b> block of code                                | <pre style="background-color: #d9ffd9; padding: 10px;"> <b>Set</b>(<i>value As dataType</i>)   optional instructions   <i>yourMemberVariable</i> = <i>value/defaultValue</i> <b>End Set</b> </pre> <p>e.g. 1: <b>Public Property</b>    e.g. 3: <b>Public WriteOnly Property</b></p>                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |                                       |
|                                                         | - can be used only alongside <b>optional WriteOnly</b> keyword, or without any <b>optional</b> keywords in the procedure header -><br>-> <b>Public Property</b> , or <b>Public WriteOnly Property</b><br>- contains the <b>Set</b> statement, which begins with the <b>Set</b> clause and ends with the <b>End Set</b> clause<br>- contained code assigns the <b>value</b> received from an app to <i>yourMemberVariable</i> associated with the <b>Property</b><br>- you can enter one or more instructions between the <b>Set</b> and <b>End Set</b> clauses<br>- you often include instructions to validate the <b>value</b> received from the app before assigning it to <i>yourMemberVariable</i><br>- in: e.g. 1: <b>Public Property</b> lines: <pre style="background-color: #f2e0aa; padding: 10px; margin-left: 20px;"> <b>If</b> <i>value</i> &gt; 0 <b>Then</b>   <i>intSide</i> = <i>value</i> <b>Else</b>   <i>intSide</i> = 0 <b>End If</b> </pre> =><br>=> selection structure determines whether the side measurement <b>intSide</b> received from the app is greater than 0<br>- if it is, the instruction <b>intSide</b> = <i>value</i> assigns the integer stored in the <b>value</b> parameter to the <b>Private intSide</b> variable<br>- otherwise, the instruction <b>intSide</b> = 0 assigns a <b>default value</b> - in this case 0 - to the variable<br>- one of the instructions should assign the contents of the <b>value</b> parameter to <i>yourMemberVariable</i> associated with the <b>Property</b><br>- in: e.g. 3: <b>Public WriteOnly Property</b> line: <b>decAnnualSales</b> = <i>value</i> =><br>=> the statement assigns the contents of the <b>AnnualSales</b> <b>Property</b> 's <b>value</b> parameter to the member variable <b>decAnnualSales</b> |                                       |
| <b>value</b>                                            | - parameter temporarily stores the <b>value</b> that is passed to the <b>Property</b> by the application<br>- the <b>value</b> parameter's <b>dataType</b> must match the <b>dataType</b> of <i>yourMemberVariable</i> associated with the <b>Property</b> procedure                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |                                       |

## CH10 F3.1 - Property section of a Class example: creating the Rectangle Class for the Franklin Decks app (01.Franklin Solution) 1/3

- when the **Class** is used to instantiate an object, each object will have its own copy of the **member variables** and **Public Properties** defined in the **Class**'s property section
- you will begin defining the **Rectangle Class**, which the **Franklin Decks application** will use to instantiate a **Rectangle** object that represents a rectangular deck
- the app will display the number of square feet of building material required for the deck as well as the total cost of the deck

- to begin creating the **Rectangle Class**:

1). open the: ...VB2017\Chap10\Exercise\01.Franklin Solution\Franklin Solution.sln

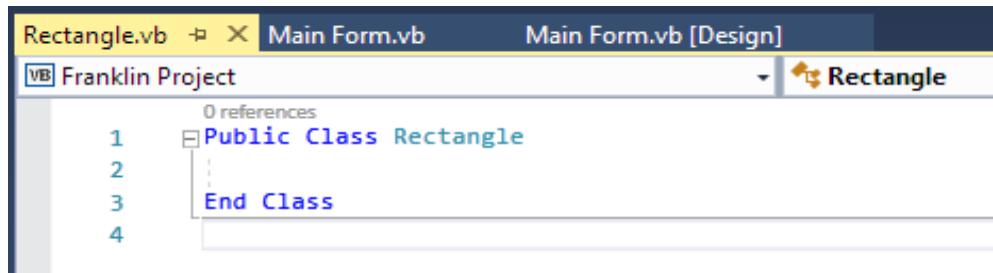
2). you gonna add a new **Class** named **Rectangle.vb** to your project

> on the VS menu bar click **Project**, click **Add Class...** , choose **Class Common Item** ,name it **Rectangle.vb**, and click the button **Add**

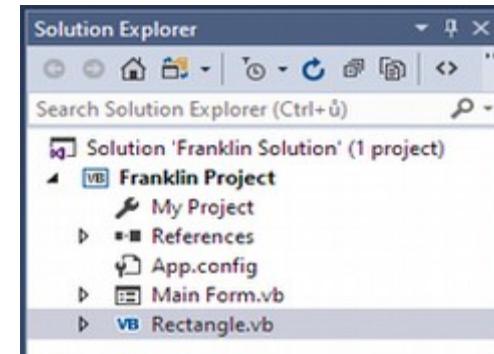
< the computer adds the **Rectangle.vb** file to the project

< it also opens the file, which contains the **Class** statement

> in the VS window **Solution Explorer** verify that the Class file's name appears in the list of project items



```
Rectangle.vb Main Form.vb Main Form.vb [Design]
VB Franklin Project
0 references
1  Public Class Rectangle
2
3  End Class
4
```



3). you gonna add some lines of code to a new **Class** named **Rectangle.vb**

> enter the lines of code as shown below:

```
1  ' Name:      Rectangle.vb
2  ' Programmer: vincenttheclown on <current date>
3
4  Option Explicit On
5  Option Strict On
6  Option Infer Off
7
8  Public Class Rectangle
9    ' insertion point:
10 End Class
11
```

4). a rectangle has **2** attributes: **length** and **width**, therefore you will use **2 member variables** to represent these attributes

> within the **Class** statement enter the following **2 Private** statements:

```
9  ' insertion point:
10 Private intLength As Integer
11 Private intWidth As Integer
12
```

5). next, you need to create a **Public Property** for each of **2 member** variables

- as you learned earlier, an app needs to use a **Public** procedure to access a **Private** variable in a **Class**

-> enter the following **Property** procedure header and **Get** clause:

```
13     Public Property Length As Integer
14         Get
15
16             End Get
17             Set(value As Integer)
18
19         End Set
20     End Property
21 End Class
```

<- when you press **Enter** after typing the **Get** clause, the Code Editor automatically enters the **End Get** clause, the **Set** statement, and the **End Property** clause

6). recall that in most cases, the **Get** statement simply returns the contents of the **member variable** associated with the **Property** procedure, so:

-> within the **Get block of code**, type on the line **15** the following statement, but don't press Enter:

```
15     Return intLength
```

7). the **Set** statement should assign either the contents of its **value** parameter or a **default value** to the **member variable** associated with the **Property** procedure

- in this case, you will assign the integer stored in the **value** parameter only when it is **greater** than **0**, otherwise you will assign the number **0**

-> within the **Set block of code**, enter on the line **18** the selection structure shown below:

```
17     Set(value As Integer)
18         If value > 0 Then
19             intLength = value
20         Else
21             intLength = 0
22         End If
23     End Set
```

8). on your own, enter a similar **Property** procedure for the **intWidth** member variable, using **Width** as the **Property**'s name

-> when done, save the solution

9). the entire code for **Property** section of the **Rectangle Class**:

```
1  ' Name:          Rectangle.vb
2  ' Programmer:    vincenttheclown on <current date>
3
4  Option Explicit On
5  Option Strict On
6  Option Infer Off
7
8  Public Class Rectangle
9      ' insertion point:
10     Private intLength As Integer
11     Private intWidth As Integer
12
```

```

13     Public Property Length As Integer      ' associated with intLength member variable
14         Get
15             Return intLength
16         End Get
17         Set(value As Integer)
18             If value > 0 Then
19                 intLength = value
20             Else
21                 intLength = 0
22             End If
23         End Set
24     End Property
25
26     Public Property Width As Integer       ' associated with intWidth member variable
27         Get
28             Return intWidth
29         End Get
30         Set(value As Integer)
31             If value > 0 Then
32                 intWidth = value
33             Else
34                 intWidth = 0
35             End If
36         End Set
37     End Property
38
39 End Class

```

### Mini-Quiz 10-3:

1. An application must use a **Public** member of a **Class** to access the **Class's Private** members. True or False?
2. Which keyword indicates that a **Property**'s **value** can be displayed but not changed by an application?
3. In a **Property** procedure, validation code is typically entered in which statement: **Get** or **Set**?

1... true

2... ReadOnly

3... Set

## CH10\_F4 - Behavior section of a Class info: usually methods: Sub procedures or Functions

- now, that you completed the **property/attribute** section of a **Class**, you can begin coding the **behavior** section
- within the **Class** statement, you define the behaviors of the Objects the **Class** will create
- represented by **methods**, usually:
  - either **Sub** procedures
  - like **Constructor** method
  - or **Function** procedures
- you can also include **Event** procedures - however, that topic is beyond the scope of this book
- the methods allow the **Class**'s objects to perform tasks
- here too, any object instantiated from the **Class** will have its own copy of the methods defined in the behavior section of the **Class**

### CH10\_F4.1 - Behavior section of a Class: Sub procedure method Constructor (default & parameterized) - info & syntax & e.g.

- **Constructor:** = **Class** method that is always named **New**
  - most **Classes** contain at least one constructor
  - the sole purpose is to initialize the **member variable** declared in the **Class**:  
**Private/... yourMemberVariable As dataType**
- **CH10\_F3 - Class's member variables:** info & syntax ; **Property** section of a **Class**: **Property** procedures: info & syntax
- never return a **value**, so they are always **Sub** procedures rather than **Function** procedure
  - for more info about **Sub** procedures and **Function** procedures see: **CH6 - Sub & Function procedures**

- **default constructor** = without parameters **e.g.1:**

- a **Class** can have only **1**
  - allowed to initialize an object's **member** variables directly

- **parameterized constructor** = with 1 or more parameters

- a **Class** can have as many as needed, but **unique** within the **Class** **e.g.2:**
    - the method name - in this case **New** - combined with its optional **parameterList** is called: the **method's signature**
    - should use the **Class's Public Property** to access the **member** variables indirectly -> this is because the **value** passed to it come from the application rather than from the **Class** itself
    - using a **Public Property** to access a **Private member** variable ensures that the **Property** procedure's **Set block of code**, which typically contains **validation** code, is processed

**e.g.2:** uses the **Class's Public Properties** to initialize its **Private member** variables, thereby invoking each **Property's validation** code

```
Public Sub New(parameterList)
```

instructions to initialize already declared **Class's member variables**

```
End Sub
```

**Class's method constructor syntax:**

**Sub New()** - a constructor is a **Class** method - **Sub** procedure that is always named **New**

**parameterList** - **optional**

- without parameters = **default constructor** -> a **Class** can have only **1** **e.g.1:**
- with parameter/s = **parameterized constructor** -> a **Class** can have as many as needed, but **unique** within the **Class**
  - the method name - in this case **New** - combined with its optional **parameterList** is called: the **method's signature**

**e.g.2:**

e.g.1: default constructor for previously created Rectangle Class

```
Public Sub New()
    intLength = 0
    intWidth = 0
End Sub
```

<- no parameters  
<- initializes the Private member variable directly  
<- initializes the Private member variable directly

<- a default constructor is allowed to initialize an object's member variables directly

e.g.2: parameterized constructor for previously created Rectangle Class

```
Public Sub New(ByVal intL As Integer, ByVal intW As Integer)
    Length = intL
    Width = intW
End Sub
```

<- 2 parameters  
<- uses the Public Property to initialize the Private member variable indirectly  
<- uses the Public Property to initialize the Private member variable indirectly

## CH10\_F4.2 - Behavior section of a Class: other methods than Constructor - Function procedure, Sub procedure - infos & syntaxes & examples

- except for constructors, which must be Sub procedures, the other methods in a Class can be either:

- Sub procedures or
- Functions

- as you learned in: **CH6 - Sub & Function procedures**, the difference between a Sub procedure and a Function is that:

- a Function returns a value after performing its assigned task
- a Sub procedure does not return a value

Class's other method: Function syntax:

```
Public Function YourMethodName() As dataType
    instructions
End Function
```

<- returns a value

Class's other method: Sub procedure syntax:

```
Public Sub YourMethodName(parameterList)
    instructions
End Sub
```

<- doesn't return a value

```
Public Function
Public Sub
YourMethodName
    (parameterList)
```

- = a method which returns a value after performing its assigned task
- = a method which does not return a value
- = name of your method
- should be entered using Pascal case with a verb as a 1st word; e.g. **SelectAll** method, **TryParse** method, etc...
- **optional**
- can pass by value - **ByVal** keyword = a copy of the variable's value; no change to the content
- can pass by reference - **ByRef** keyword = the variable's address in RAM

**CH6\_F3 - Passing Information to a Procedure & passing a variable by value & passing a variable by reference**

e.g.1 **Function** procedure used for previously created **Class Rectangle**, allowing a **Rectangle** object to calculate its area

```
Public Function GetArea() As Integer  
    Return intLength * intWidth  
End Function
```

<- notice that you can write the method as: either a **Function** or a **Sub** procedure

e.g.2 **Sub** procedure used for previously created **Class Rectangle**, allowing a **Rectangle** object to calculate its area

```
Public Sub GetArea(ByRef intA As Integer)  
    intA = intLength * intWidth  
End Sub
```

<- notice that you can write the method as: either a **Function** or a **Sub** procedure

<- you will create and use the **GetArea** method in the **Franklin Decks application** to calculate the deck's area

- calculating the area will give you the number of square feet of material required for the deck

### CH10\_F4.3 - Behavior section of a Class example: complete the **Rectangle Class** for the: **Franklin Decks app (01.Franklin Solution)** 2/3

- in this section, you will complete the **Rectangle Class** by adding a constructor and the **GetArea** method to its **behavior** section

- to complete the **Rectangle Class** by adding the behavior section:

1). open the: ...VB2017\Chap10\Exercise01.Franklin Solution\Franklin Solution.sln

2). you gonna continue defining your new **Class** named **Rectangle**, i.e. continue editing the tab named **Rectangle.vb** in your project

>> below the **Width** property's **End Property** clause enter:

- the **default constructor** method and
- the **GetArea** method

```
37     End Property
```

```
38
```

```
39     Public Sub New()
```

<- the sole purpose of the **constructor** is to initialize the **member variable** declared in the **Class**

**CH10\_F4.1 - Behavior section of a Class: Sub procedure method Constructor (default & parameterized) - info & syntax & e.g.**

```
40         intLength= 0
```

```
41         intWidth = 0
```

```
42     End Sub
```

```
43
```

```
44     Public Function GetArea() As Integer
```

<- a method, the **Function** procedure which returns a value after performing its assigned task

**CH10\_F4.2 - Behavior section of a Class: other methods than Constructor - Function procedure,**

**Sub** procedure - infos & syntaxes & examples

```
45         Return intLength * intWidth
```

```
46     End Function
```

```
47
```

```
48 End Class
```

3). save the solution

4). the entire code of the **Rectangle Class**:

```
1  ' Name:          Rectangle.vb  
2  ' Programmer:   vincenttheclown on <current date>  
3  
4  Option Explicit On  
5  Option Strict On
```

```
6 Option Infer Off
7
8 Public Class Rectangle
9     ' insertion point:
10    Private intLength As Integer
11    Private intWidth As Integer
12
13    Public Property Length As Integer      ' associated with intLength member variable
14        Get
15            Return intLength
16        End Get
17        Set(value As Integer)
18            If value > 0 Then
19                intLength = value
20            Else
21                intLength = 0
22            End If
23        End Set
24    End Property
25
26    Public Property Width As Integer      ' associated with intWidth member variable
27        Get
28            Return intWidth
29        End Get
30        Set(value As Integer)
31            If value > 0 Then
32                intWidth = value
33            Else
34                intWidth = 0
35            End If
36        End Set
37    End Property
38
39    Public Sub New()
40        intLength= 0
41        intWidth = 0
42    End Sub
43
44    Public Function GetArea() As Integer
45        Return intLength * intWidth
46    End Function
47
48 End Class
```

## CH10\_F5 - instantiating an Object and invoking the constructor - info, syntax and example

e.g.: `Dim randGen As New Random`

statement from CH7 instantiates a `Random` object and invokes the object's default **constructor**

### CH7\_A3 - Generate Random Integers - object `Random`, method `Next`

- after you define a **Class**:

```
Public Class YourClassName  
    1. Class's member variable  
    2. property section  
    3. behavior section  
End Class
```

**CH10\_F3 - Class's member** variables: info & syntax ;

**Property** section of a **Class**: **Property** procedures: info & syntax

**CH10\_F4 - Behavior** section of a **Class** info: usually **methods**: **Sub** procedures or **Functions**

- you can use either of the following syntax **v.1** or **v.2** to instantiate 1 or more **Objects**

- when an Object is instantiated, the computer uses one of the **Class's constructors** to initialize the Object's **member** variables

- if a **Class** contains more than 1 constructor, the computer determines the appropriate **constructor** by  
matching the: number, data type, and position of the arguments in the statement that instantiates the Object  
with the: number, data type, and position of the parameters listed in each **constructor's parameterList**

syntax v.1:

```
Dim/Private yourVariable As YourClassName  
yourVariable = New YourClassName
```

**As** `YourClassName`

## instantiating an Object

- the difference between the versions relates to when the **Object** is actually created

syntax v.2:

```
Dim/Private yourVariable As New YourClassName
```

**As** `YourClassName`

**Dim/Private** - variable declaration rules learned in previous lessons apply, so:  
- **Dim** = procedure-level variable, declared in procedure, procedure scope  
- **Private** = class-level variable, declared in the form class's declarations section, class-level scope  
**yourVariable** = name of a variable that will represent the **Object**  
**YourClassName** - **Class** `YourSolutionName.YourClassName`  
e.g. `Class WindowsApp1.TimeCard`  
**New** - not a requirement, but the convention is to use **Pascal** case for the **Class** name, like: **TextBox**, **String**, etc...  
= a keyword  
= creates a new **Object** instance  
- the computer creates the **Object** only when it processes the statement containing the **New** keyword,

e.g. v.1:

```
Private hoursInfo As TimeCard  
hoursInfo = New TimeCard
```

<- the **Private** instruction creates a `TimeCard` variable named `hoursInfo`

<- then the assignment statement instantiates a `TimeCard` **Object** and assigns it to the `hoursInfo` variable

e.g. v.2:

```
Dim hoursInfo As New TimeCard
```

<- the **Dim** instruction creates a `TimeCard` variable named `hoursInfo` and also instantiates a `TimeCard` **Object**,  
which it assigns to the `hoursInfo` variable

e.g.1 invoking the default **constructor** = no arguments, from previously created **Class Rectangle** - syntax v.1

```
Dim deck As New Rectangle
```

e.g.2 invoking the default **constructor** = no arguments, from previously created **Class Rectangle** - syntax v.2

```
Dim deck As Rectangle  
...  
deck = New Rectangle
```

e.g.3 invoking the parameterized **constructor** = 2 integer arguments, from previously created **Class Rectangle** - syntax v.1

```
Dim deck As New Rectangle(16, 14)
```

e.g.4 invoking the parameterized **constructor** = 2 integer arguments, from previously created **Class Rectangle** - syntax v.2

```
Dim deck As Rectangle  
...  
deck = New Rectangle(intDeckLen, intDeckWid)
```

### Mini-Quiz 10-2:

1. Write a statement that declares a procedure-level **Employee** variable named **manager**.
2. Write a statement that instantiates an **Employee** object and assigns it to the **manager** variable from Question 1.
3. Write a statement that declares a procedure-level **Employee** variable named **hourly**. The statement should also instantiate an **Employee** object, storing it in the **hourly** variable.

```
1...Dim manager As Employee  
2...manager = New Employee  
3...Dim hourly As New Employee
```

**CH10\_F5.1** - using the **Rectangle Class** in the application example: pseudocode & GUI for the: **Franklin Decks app** (01.Franklin Solution) incl. entire code **3/3**

- now that you defined the **Rectangle Class**, you can use it to instantiate a **Rectangle Object** in the **Franklin Decks application**

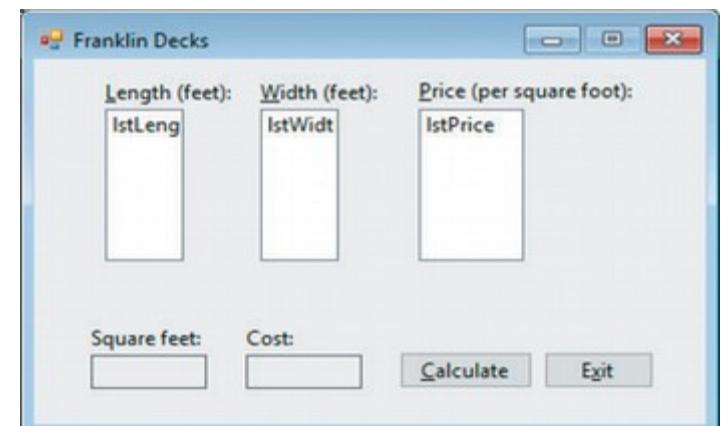
#### btnCalc Click pseudocode:

**step 1:** - declare a **Rectangle** variable named **deck** = instantiate a **Rectangle** object and store it in the **deck** variable

```
Dim deck As New Rectangle
```

**step 2:** - declare variables:    - **dblSqFtPrice** to store the price per square foot of material  
                               - **intSqFt** to store the required number of material's square feet  
                               - **dblCost** to store the cost of the deck

**step 3:** - the item selected in (**1stLength**) control convert to Integer and then store the result in the **deck** Object's **Length Property**  
- the item selected in (**1stWidth**) control convert to Integer and then store the result in the **deck** Object's **Width Property**



**step 4:** - the item selected in (lstPrice) control convert to Double and then store it in the dblSqFtPrice variable

**step 5:** - calculate the required number of square feet of material by finding the deck's area, using the deck Object's GetArea method and assign the method's return value to the intSqFt variable

**step 6:** - calculate the cost of the deck by multiplying the required number of square feet of material stored in intSqFt by the price per square foot of material stored in dblSqFtPrice; assign the result to the dblCost variable

**step 7:** - display the required number of square feet of material stored in intSqFt and the cost of the deck stored in dblCost in (lblSqFt) and (lblCost)

- to use the **Rectangle Class** in the **Franklin Decks** application:

1). open the: ...VB2017\Chap10\Exercise01.Franklin Solution\Franklin Solution.sln

-> open the Code Editor window and locate the btnCalc\_Click procedure

**step 1:** - the first step in the pseudocode declares a **Rectangle** variable named **deck**

- it also instantiates a **Rectangle** Object and then stores the Object (which represents the **deck**) in the **Rectangle** variable

-> below the comment line '**Step 1: Instantiate a Rectangle object:**' type the following **Dim** statement:

```
9  Public Class frmMain  
10     Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click  
11         ' Display square feet and deck cost.  
12  
13         ' Step 1: Instantiate a Rectangle Object:  
14         Dim deck As New Rectangle  
15     
```

**step 2:** - next, the procedure will declare variables: - **dblSqFtPrice** to store the price per square foot of material  
- **intSqFt** to store the required number of square feet of material  
- **dblCost** to store the cost of the deck

-> below the comment line '**Step 2: Declare variables:**' enter the following 3 **Dim** statements:

```
16         ' Step 2: Declare variables:  
17         Dim dblSqFtPrice As Double      ' store the price per square foot of material  
18         Dim intSqFt As Integer        ' store the required number of square feet of material  
19         Dim dblCost As Double        ' store the cost of the deck  
20     
```

**step 3:** - the item selected in (lstLength) control convert to Integer and then store the result in the **deck** Object's **Length Property**  
- the item selected in (lstWidth) control convert to Integer and then store the result in the **deck** Object's **Width Property**

-> below the comment line '**Step 3: Assign Length and Width to the Object's Public Properties:**' enter the **TryParse** methods:

<- notice that when you press the period after typing **deck**, the **deck** Object's **Length** and **Width Properties** appear in the **IntelliSense** list

```
21         ' Step 3: Assign Length and Width to the Object's Public Properties:  
22         Integer.TryParse(lstLength.SelectedItem.ToString, deck.Length)  
23         Integer.TryParse(lstWidth.SelectedItem.ToString, deck.Width)  
24     
```

**step 4:** - the item selected in (lstPrice) control convert to Double and store it in the dblSqFtPrice variable

-> below the comment line ' Step 3: Assign price to variable: enter the TryParse method:

```
25      ' Step 4: Assign price to variable:  
26      Double.TryParse(lstPrice.SelectedItem.ToString(), dblSqFtPrice)  
27
```

**step 5:** - the fifth step in the pseudocode uses the deck Object's GetArea method to calculate the required number of square feet of material

- it assigns the method's return value to the intSqFt variable

-> below the comment line ' Step 5: Calculate area in square feet: enter the following assignment statement:

<- here again notice that when you press the period after typing deck, the deck Object's GetArea method appears in the IntelliSense list

```
28      ' Step 5: Calculate area in square feet:  
29      intSqFt = deck.GetArea  
30
```

**step 6:** - calculate the cost of the deck by multiplying the required number of square feet of material stored in intSqFt by the

price per square foot of material stored in dblSqFtPrice; assign the result to the dblCost variable

-> below the comment line ' Step 6: Calculate cost of deck: enter the following assignment statement:

```
31      ' Step 6: Calculate cost of deck:  
32      dblCost = intSqFt * dblSqFtPrice  
33
```

**step 7:** - in (lblSqFt) display the required number of square feet of material stored in intSqFt

- in (lblCost) display the cost of the deck stored in dblCost

<- the code has already been entered for you, but anyway:

```
34      ' Step 7: Display output:  
35      lblSqFt.Text = intSqFt.ToString  
36      lblCost.Text = dblCost.ToString("C2")  
37      End Sub
```

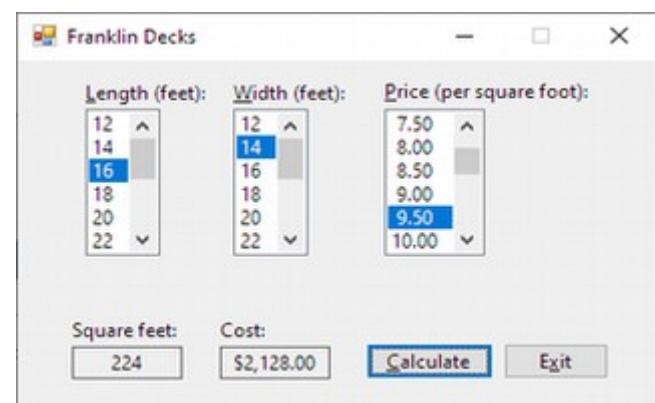
2). save the solution

3). test the Franklin Decks application:

-> start the application, choose: Length = 16, Width = 14, Price = 9.50

and then click the Calculate button

<- the results should be: Square feet = 224, Cost = \$2,128.00



4). entire code for **Rectangle.vb** - Class **Rectangle** & **Main Form.vb** - Class **frmMain**

```
1  ' Name:          Rectangle.vb
2  ' Programmer:    vincenttheclown on <current date>
3
4  Option Explicit On
5  Option Strict On
6  Option Infer Off
7
8  Public Class Rectangle
9      ' insertion point:
10     Private intLength As Integer
11     Private intWidth As Integer
12
13     Public Property Length As Integer      ' associated with intLength member variable
14         Get
15             Return intLength
16         End Get
17         Set(value As Integer)
18             If value > 0 Then
19                 intLength = value
20             Else
21                 intLength = 0
22             End If
23         End Set
24     End Property
25
26     Public Property Width As Integer      ' associated with intWidth member variable
27         Get
28             Return intWidth
29         End Get
30         Set(value As Integer)
31             If value > 0 Then
32                 intWidth = value
33             Else
34                 intWidth = 0
35             End If
36         End Set
37     End Property
38
39     Public Sub New()
40         intLength= 0
41         intWidth = 0
42     End Sub
43
```

```
44     Public Function GetArea() As Integer
45         Return intLength * intWidth
46     End Function
47
48 End Class
```

```
1  ' Name:      Franklin Project
2  ' Purpose:    Displays the number of square feet of material and the cost of a deck.
3  ' Programmer: vincenttheclown on <current date>
4
5 Option Explicit On
6 Option Strict On
7 Option Infer Off
8
9 Public Class frmMain
10    Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
11        ' Displays square feet and deck cost.
12
13        ' Step 1: Instantiate a Rectangle Object:
14        Dim deck As New Rectangle
15
16        ' Step 2: Declare variables:
17        Dim dblSqFtPrice As Double      ' store the price per square foot of material
18        Dim intSqFt As Integer          ' store the required number of square feet of material
19        Dim dblCost As Double           ' store the cost of the deck
20
21        ' Step 3: Assign Length and Width to the Object's Public Properties:
22        Integer.TryParse(lstLength.SelectedItem.ToString, deck.Length)
23        Integer.TryParse(lstWidth.SelectedItem.ToString, deck.Width)
24
25        ' Step 4: Assign price to variable:
26        Double.TryParse(lstPrice.SelectedItem.ToString, dblSqFtPrice)
27
28        ' Step 5: Calculate area in square feet:
29        intSqFt = deck.GetArea
30
31        ' Step 6: Calculate cost of deck:
32        dblCost = intSqFt * dblSqFtPrice
33
34        ' Step 7: Display output:
35        lblSqFt.Text = intSqFt.ToString
36        lblCost.Text = dblCost.ToString("C2")
37    End Sub
38
```

<- instantiates a **Rectangle** Object and initializes it using the default **constructor**

<- assigns value to Object's **Public Property**

<- assigns value to Object's **Public Property**

<- invokes the Object's **GetArea** method

```

39      Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
40          Me.Close()
41      End Sub
42
43      Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
44          ' Fills the list boxes and then selects a default item.
45
46          For intNums As Integer = 12 To 36 Step 2
47              lstLength.Items.Add(intNums.ToString())
48              lstWidth.Items.Add(intNums.ToString())
49          Next intNums
50
51          For dblPrices As Double = 5 To 25 Step 0.5
52              lstPrice.Items.Add(dblPrices.ToString("N2").PadLeft(5))
53          Next dblPrices
54
55          lstLength.SelectedIndex = 0
56          lstWidth.SelectedIndex = 0
57          lstPrice.SelectedItem = "10.00"
58      End Sub
59
60      Private Sub ClearOutput(sender As Object, e As EventArgs) Handles lstLength.SelectedIndexChanged, lstWidth.SelectedIndexChanged,
61   lstPrice.SelectedIndexChanged
62          lblSqFt.Text = String.Empty
63          lblCost.Text = String.Empty
64      End Sub
65  End Class

```

**CH10\_F6 - You Do It 1:** create and use **Class** exercise: **02.You Do It 1 Solution -> Public Property, default constructor, function**

1. create an application named **You Do It 1** and save it in the ...VB2017\Chap10\Exercise02\You Do It 1 Solution
2. add: **text box, label, and a button** to the form
3. add a **Class** file named **Circle.vb** to the project and define a **Class** named **Circle** that contains:
  - attribute/**property**: the circle's radius
  - a default constructor
  - a method that calculates and returns the circle's area using the following formula:  $3.141592 * \text{radius}^2 = (\text{radius} * \text{radius}) * \pi$
4. open the form's Code Editor window and code the button's Click event procedure that:
  - it instantiates a Circle Object
  - then it uses the radius entered by the user to calculate the Object's area
  - display the area in the label control
5. save the solution and then start and test the application

code for Circle.vb, frmMain.vb & my GUI:

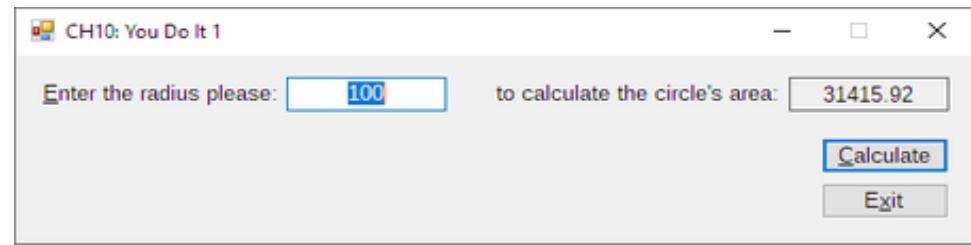
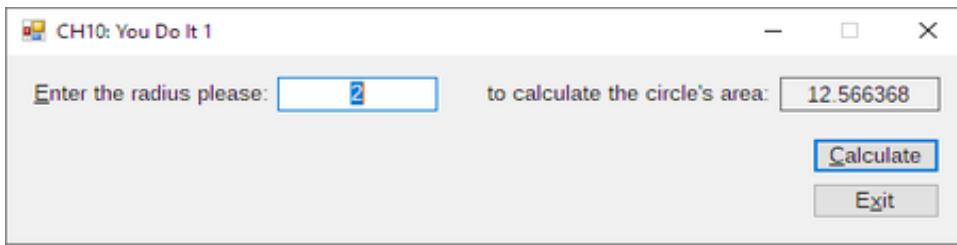
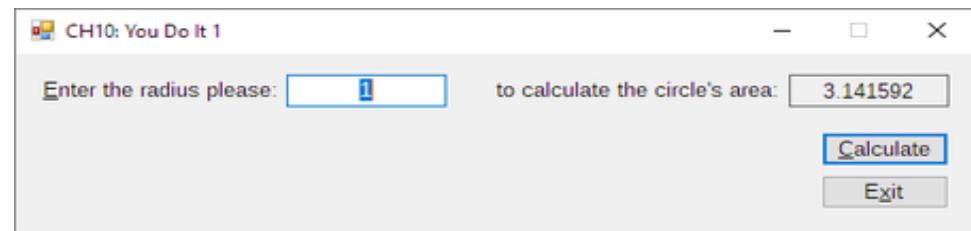
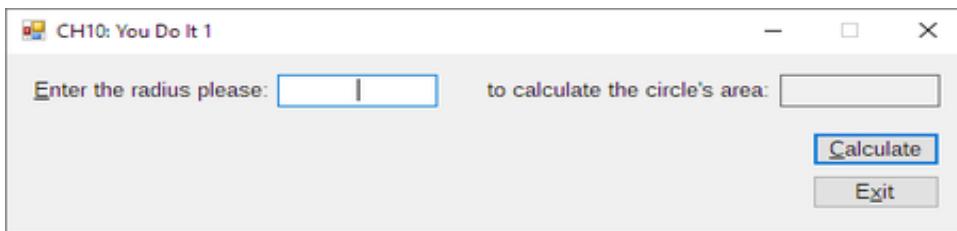
```
1  ' Name:          Circle.vb
2  ' Programmer:    vincenttheclown
3
4  Option Explicit On
5  Option Strict On
6  Option Infer Off
7  '3. add a Class file named Circle.vb to the project and define a Class named Circle that contains:
8  ' - attribute/property: the circle 's radius
9  ' - a default constructor
10 ' - a method that calculates and returns the circle's area using the following formula: 3.141592 * radius^2
11
12 Public Class Circle
13     Private dblNumber As Double
14
15     Public Property CircleRadius As Double
16         Get
17             ' something here?
18             Return dblNumber
19         End Get
20         Set(value As Double)
21             If value > 0 Then
22                 dblNumber = value
23             Else
24                 dblNumber = 0
25             End If
26         End Set
27     End Property
28
29     ' default constructor:
30     Public Sub New() ' without parameterList = default constructor
31         ' instructions to initialize already declared member variable:
32         dblNumber = 0
33     End Sub
34
35     '- a method that calculates and returns the circle's area using the following formula: 3.141592 * radius^2
36     Public Function GetCircleArea() As Double
37         Return (dblNumber * dblNumber) * 3.141592
38     End Function
39
40 End Class
```

```
1  '-2. add: text box, label, and a button To the form
2  '-3. add a Class file named Circle.vb to the project and define a Class named Circle that contains:
3  ' - attribute/property: the circle's radius
4  ' - a default constructor
5  ' - a method that calculates and returns the circle's area using the following formula: 3.141592 * radius^2
6  '4. open the form's Code Editor window and code the button's Click event procedure that:
7  ' - it instantiates a Circle Object
8  ' - than it uses the radius (= polomér; diameter = průměr) entered by the user to calculate the Object's area
9  ' - display the area in the label control
10 '5. save the solution And then start And test the application
11
12 Option Explicit On
13 Option Strict On
14 Option Infer Off
15
16 Public Class frmMain
17
18     Private Sub btnCalculate_Click(sender As Object, e As EventArgs) Handles btnCalculate.Click
19         txtInput.Focus()
20
21         ' Step 1: instantiate a Circle Object:
22         Dim Area As New Circle
23
24         ' Step 2: declare variables:
25         Dim dblInput As Double      ' store the input number of the radius
26         Dim dblOutput As Double     ' store the calculated area
27
28         ' assign user input to the variable:
29         Double.TryParse(txtInput.Text, dblInput)
30
31         ' assign user input (dblInput) to the Object's (named Area) Public Property:
32         Double.TryParse(dblInput.ToString, Area.CircleRadius)
33
34         ' by invoking the Object's method: GetCircleArea, calculate the area; assign the result of the calculation to the variable:
35         dblOutput = Area.GetCircleArea
36
37         ' display the result:
38         lblOutput.Text = dblOutput.ToString
39
40         txtInput.SelectAll()
41     End Sub
42
43     Private Sub txtInput_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtInput.KeyPress
44         If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> "." AndAlso e.KeyChar <> ControlChars.Back Then
```

```

45         e.Handled = True
46     End If
47 End Sub
48
49 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
50     Me.Close()
51 End Sub
52
53 Private Sub txtInput_Enter(sender As Object, e As EventArgs) Handles txtInput.Enter, txtInput.MouseClick, txtInput.TextChanged
54     'txtInput.SelectAll()
55     lblOutput.Text = Nothing
56 End Sub
57 End Class

```



## CH10\_F7 - adding a parameterized Constructor example: a Rectangle Class of 03.Franklin Solution-Parameterized 1/5

- in this section, you will: - add a parameterized Constructor to the Rectangle Class created in: 01.Franklin Solution

- to modify the Rectangle Class in the Franklin Decks application: -> adding the parameterized constructor to the Class:

1). open the: ...VB2017\Chap10\Exercise03.Franklin Solution-Parameterized\Franklin Solution.sln

2). open the Rectangle.vb tab to view the code

-> below the default Constructor's End Sub clause on line 42 enter the parameterized Constructor shown below:

```
39     Public Sub New()      ' default constructor
40         intLength= 0
41         intWidth = 0
42     End Sub
43
44     Public Sub New(ByVal intL As Integer, ByVal intW As Integer)    ' parameterized constructor
45         Length = intL
46         Width = intW
47     End Sub
48
```

3). save the solution and then close the Rectangle.vb tab

## CH10\_F7.1 - comparing default vs parameterized Constructor example: Rectangle Class of the 03.Franklin Solution-Parameterized 2/5

- more info about Constructor: CH10\_F4.1 - Behavior section of a Class: Sub procedure method Constructor (default & parameterized) - info & syntax & e.g.

- default constructor of the Rectangle Class: (= without parameters)

- a Class can have only 1
- allowed to initialize an object's member variables directly
- automatically initializes the Private member variables to 0 when a Rectangle Object is created

```
39     Public Sub New()
40         intLength= 0           <- accesses the Private member variable directly
41         intWidth = 0          <- accesses the Private member variable directly
42     End Sub
```

- parameterized constructor of the Rectangle Class: (= with 1 or more parameters)

- a Class can have as many as needed, but unique within the Class
- the method name - in this case New - combined with its optional parameterList is called: the method's signature
- unlike the default constructor, which automatically initializes the Private member variables to 0 when a Rectangle Object is created, the parameterized constructor allows an application to specify the Object's initial values ->

-> in this case, the initial values must have the Integer data type because the constructor's parameterList contains 2 Integer variables

```
44     Public Sub New(ByVal intL As Integer, ByVal intW As Integer)    ' parameterized constructor
45         Length = intL          <- uses the Public Property to access the Private member variable
46         Width = intW          <- uses the Public Property to access the Private member variable
47     End Sub
```

- intL & intW -> these values come from the application that instantiates a Rectangle Object

- you include the initial values, enclosed in a set of parentheses, in the statement that instantiates the Object
- in other words, you include them in the statement that contains the **New** keyword, such as the:

```
Dim deck As Rectangle
...
deck = New Rectangle(intDeckLen, intDeckWid)
```

-more info about invoking the **Constructor**: [CH10\\_F5 - instantiating an Object and invoking the constructor - info, syntax and example](#)

## CH10\_F7.2 - using/invoking a parameterized Constructor example: a btnCalc\_Click procedure of 03.Franklin Solution-Parameterized 3/5

- in this section, you will:
  - modify the **Franklin Deck application** to use the **parameterized Constructor** rather than the **default Constructor**

- to use the parameterized constructor in the **btnCalc\_Click** procedure, therefore to modify the procedure:

1). open the: ...VB2017\Chap10\Exercise\03.Franklin Solution-Parameterized\Franklin Solution.sln

2). open the form's Code Editor window and locate the **btnCalc\_Click** procedure

-> on the line **13**, replace the word: **Instantiate** with the words: **Declare a variable for**

-> on the line **14**, delete the **New** keyword from the **Dim** statement, so it say now: **Dim deck As Rectangle**

*original:*

```
13      ' Step 1: Instantiate a Rectangle Object:
14      Dim deck As New Rectangle
```

*modification:*

```
13      ' Step 1: Declare a variable for a Rectangle Object:
14      Dim deck As Rectangle
```

<- declares a variable that can store a **Rectangle** Object

-> under the comment: **' Step 2: Declare variables:** on line **16** and below the last **Dim** statement enter the following **2** declaration statements:

```
16      ' Step 2: Declare variables:
17      Dim dblSqFtPrice As Double      ' store the price per square foot of material
18      Dim intSqFt As Integer        ' store the required number of square feet of material
19      Dim dblCost As Double        ' store the cost of the deck
20      Dim intDeckLen As Integer    ' Object's initial value, linked with a Public Property Length
21      Dim intDeckWid As Integer    ' Object's initial value, linked with a Public Property Width
22
```

-> on line **23** in the comment line, replace the words: **the Object's Public Properties** with the word: **variables**

-> on line **24** in the first **TryParse** method, replace: **deck.Length** with: **intDeckLen**

-> on line **25** in the second **TryParse** method, replace: **deck.Width** With: **intDeckWid**

*original:*

```
23      ' Step 3: Assign Length and Width to the Object's Public Properties:
24      Integer.TryParse(lstLength.SelectedItem.ToString, deck.Length)
25      Integer.TryParse(lstWidth.SelectedItem.ToString, deck.Width)
```

*modification:*

```
23      ' Step 3: Assign Length and Width to the variables:
24      Integer.TryParse(lstLength.SelectedItem.ToString, intDeckLen)
25      Integer.TryParse(lstWidth.SelectedItem.ToString, intDeckWid)
26
```

-> below the line 29 enter the following comment and assignment statement:

```
27      ' Step 4: Assign price to variable:  
28      Double.TryParse(lstPrice.SelectedItem.ToString, dblSqFtPrice)  
29  
30      ' Instantiate and initialize a Rectangle Object:  
31      deck = New Rectangle(intDeckLen, intDeckWid)  
32
```

<- instantiates a **Rectangle** Object and initializes it using the **parameterized constructor**

-> save the solution

### CH10\_F7.3 - analyse the process of using/invoking a parameterized Constructor: a btnCalc\_Click procedure of 03.Franklin Solution-Parameterized 4/5

- when the user clicks the button **Calculate**, the:

```
9   Public Class frmMain  
13      ' Step 1: Instantiate a Rectangle Object:  
14      Dim deck As Rectangle
```

instruction creates a variable that can store a **Rectangle** Object, but it doesn't create the Object -> that is done on line number 31

- the remaining **Dim** statements create and initialize **5** variables:

```
9   Public Class frmMain  
16      ' Step 2: Declare variables:  
17      Dim dblSqFtPrice As Double          ' store the price per square foot of material  
18      Dim intSqFt As Integer            ' store the required number of square feet of material  
19      Dim dblCost As Double             ' store the cost of the deck  
20      Dim intDeckLen As Integer        ' Object's initial value, linked with a Public Property Length  
21      Dim intDeckWid As Integer         ' Object's initial value, linked with a Public Property Width
```

- the **TryParse** methods assign the input values to **intDeckLen**, **intDeckWid**, **dblSqFtPrice** variables:

```
9   Public Class frmMain  
23      ' Step 3: Assign Length and Width to the variables:  
24      Integer.TryParse(lstLength.SelectedItem.ToString, intDeckLen)  
25      Integer.TryParse(lstWidth.SelectedItem.ToString, intDeckWid)  
26  
27      ' Step 4: Assign price to variable:  
28      Double.TryParse(lstPrice.SelectedItem.ToString, dblSqFtPrice)
```

- the next statement in the procedure instantiates a **Rectangle** Object named **deck**

- the **2** Integer arguments in the statement **intDeckLen** and **intDeckWid** tell the computer to use the **parameterized constructor** to initialize the Object's **Private member variables**

```
9   Public Class frmMain  
30      ' Instantiate and initialize a Rectangle Object:  
31      deck = New Rectangle(intDeckLen, intDeckWid)
```

- the computer passes the 2 arguments by value -> **ByVal** to the constructor, which stores them in its **intL** and **intW** parameters
- the assignment statements in the constructor then assign the parameter values to the Object's **Public Property Length** and **Public Property Width**
- notice that a **parameterized constructor** uses the **Class's Public Properties** to access the **Private member variables** indirectly
  - this is because the **values** passed to a **parameterized constructor** come from the application rather than from the **Class** itself
  - as mentioned earlier, **values** that originate outside of the **Class** should always be assigned to the **Private member variables** indirectly through the **Public Properties** to ensure that the **Property** procedure's **Set** block, which typically contains validation code, is processed

```

8  Public Class Rectangle
44     Public Sub New(ByVal intL As Integer, ByVal intW As Integer)      ' parameterized constructor
45         Length = intL
46         Width = intW
47     End Sub

```

- when you assign a value to a **Property**, the computer passes the **value** to the **Property's Set** statement, where it is stored in the **Set** statement's **value** parameter
  - in this case, the selection structure in the **Public Property Length's Set** statement compares the value stored in the **value** parameter with the number **0**:
    - if the **value** is greater than **0**, the selection structure's **True** path assigns the **value** to the **Private member variable** named **intLength**
    - otherwise its **False** path assigns the number **0** to the **Private member variable** named **intLength**
  - the s.s. in the **Public Property Width's Set** statement works the same way, except it assigns the number to the **Private member variable** named **intWidth**

```

8  Public Class Rectangle
17      Set(value As Integer)
18          If value > 0 Then
19              intLength = value
20          Else
21              intLength = 0
22          End If
23      End Set
...
30      Set(value As Integer)
31          If value > 0 Then
32              intWidth = value
33          Else
34              intWidth = 0
35          End If
36      End Set

```

- after the **Rectangle** Object is instantiated and its **Private member variables** are initialized, the **btnCalc\_Click** procedure uses the Object's **GetArea** method to calculate and return the area of the deck, which represents the required number of square feet of building material

```

9  Public Class frmMain
33      ' Step 5: Calculate area in square feet:
34      intSqFt = deck.GetArea

```

```

8  Public Class Rectangle
49      Public Function GetArea() As Integer
50          Return intLength * intWidth
51      End Function

```

- finally, the procedure calculates the cost of the deck and then displays both the required number of square feet and the cost

```
9  Public Class frmMain
36      ' Step 6: Calculate cost of deck:
37      dblCost = intSqFt * dblSqFtPrice
38
39      ' Step 7: Display output:
40      lblSqFt.Text = intSqFt.ToString
41      lblCost.Text = dblCost.ToString("C2")
```

## CH10\_F7.4 - the entire code for a parameterized Constructor example and testing: 03.Franklin Solution-Parameterized 5/5

### 1). entire code for Rectangle.vb - Class Rectangle

```
1  ' Name:          Rectangle.vb
2  ' Programmer:    vincenttheclown on <current date>
3
4  Option Explicit On
5  Option Strict On
6  Option Infer Off
7
8  Public Class Rectangle
9      ' insertion point:
10     Private intLength As Integer
11     Private intWidth As Integer
12
13     Public Property Length As Integer      ' associated with intLength member variable
14         Get
15             Return intLength
16         End Get
17         Set(value As Integer)
18             If value > 0 Then
19                 intLength = value
20             Else
21                 intLength = 0
22             End If
23         End Set
24     End Property
25
26     Public Property Width As Integer      ' associated with intWidth member variable
27         Get
28             Return intWidth
29         End Get
```

```

30     Set(value As Integer)
31         If value > 0 Then
32             intWidth = value
33         Else
34             intWidth = 0
35         End If
36     End Set
37 End Property
38
39 Public Sub New()      ' default constructor
40     intLength= 0
41     intWidth = 0
42 End Sub
43
44 Public Sub New(ByVal intL As Integer, ByVal intW As Integer)      ' parameterized constructor
45     Length = intL
46     Width = intW
47 End Sub
48
49 Public Function GetArea() As Integer
50     Return intLength * intWidth
51 End Function
52
53 End Class

```

## 2). entire code for Main Form.vb - Class frmMain

```

1  ' Name:      Franklin Project
2  ' Purpose:    Displays the number of square feet of material and the cost of a deck.
3  ' Programmer: vincenttheclown on <current date>
4
5  Option Explicit On
6  Option Strict On
7  Option Infer Off
8
9  Public Class frmMain
10     Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
11         ' Displays square feet and deck cost.
12
13         ' Step 1: Declare a variable for a Rectangle Object:
14         Dim deck As Rectangle
15
16         ' Step 2: Declare variables:
17         Dim dblSqFtPrice As Double      ' store the price per square foot of material
18         Dim intSqFt As Integer          ' store the required number of square feet of material

```

```
19      Dim dblCost As Double          ' store the cost of the deck
20      Dim intDeckLen As Integer      ' Object's initial value, linked with a Public Property Length
21      Dim intDeckWid As Integer      ' Object's initial value, linked with a Public Property Width
22
23      ' Step 3: Assign Length and Width to the variables:
24      Integer.TryParse(lstLength.SelectedItem.ToString, intDeckLen)
25      Integer.TryParse(lstWidth.SelectedItem.ToString, intDeckWid)
26
27      ' Step 4: Assign price to variable:
28      Double.TryParse(lstPrice.SelectedItem.ToString, dblSqFtPrice)
29
30      ' Instantiate and initialize a Rectangle Object:
31      deck = New Rectangle(intDeckLen, intDeckWid)
32
33      ' Step 5: Calculate area in square feet:
34      intSqFt = deck.GetArea
35
36      ' Step 6: Calculate cost of deck:
37      dblCost = intSqFt * dblSqFtPrice
38
39      ' Step 7: Display output:
40      lblSqFt.Text = intSqFt.ToString
41      lblCost.Text = dblCost.ToString("C2")
42 End Sub
43
44 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
45     Me.Close()
46 End Sub
47
48 Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
49     ' Fills the list boxes and then selects a default item.
50
51     For intNums As Integer = 12 To 36 Step 2
52         lstLength.Items.Add(intNums.ToString)
53         lstWidth.Items.Add(intNums.ToString)
54     Next intNums
55
56     For dblPrices As Double = 5 To 25 Step 0.5
57         lstPrice.Items.Add(dblPrices.ToString("N2").PadLeft(5))
58     Next dblPrices
59
60     lstLength.SelectedIndex = 0
61     lstWidth.SelectedIndex = 0
62     lstPrice.SelectedItem = "10.00"
```

```

63      End Sub
64
65      Private Sub ClearOutput(sender As Object, e As EventArgs) Handles lstLength.SelectedIndexChanged, lstWidth.SelectedIndexChanged,
66   lstPrice.SelectedIndexChanged
67          lblSqFt.Text = String.Empty
68          lblCost.Text = String.Empty
69      End Sub
70  End Class

```

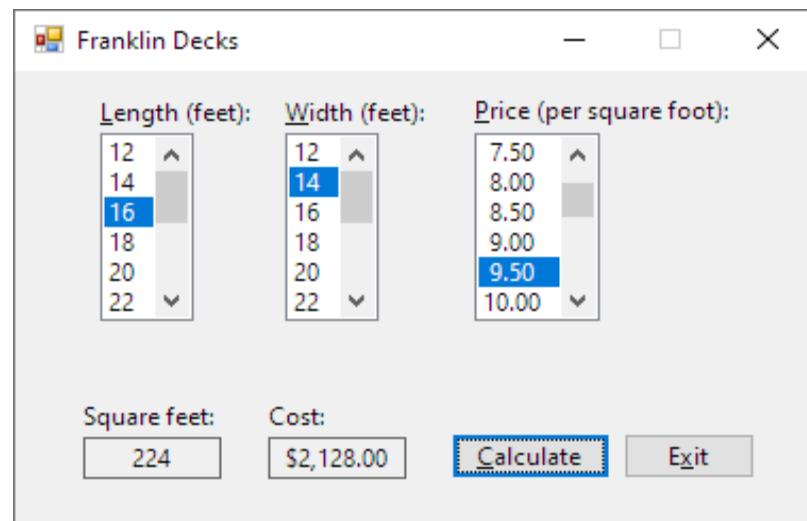
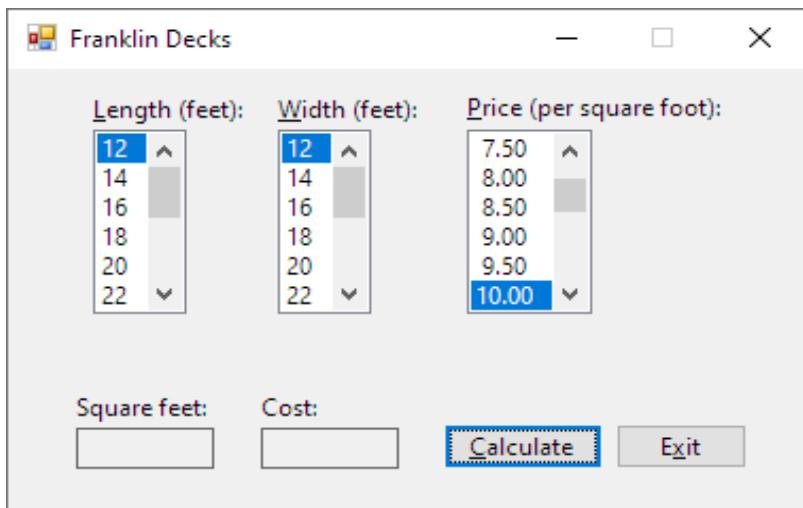
3). to test the modified application:

-> start the modified application and choose:  
 - Length = **16**  
 - Width = **14**  
 - Price = **9.50**

-> click the button **Calculate**

<- the **btnCalc\_Click** procedure displays:  
 - the required number of square feet = **224**  
 - the cost of the deck = **\$2,128.00**

-> when done, close the application and close the solution



## CH10\_F8 - You Do It 2: create and use Class exercise: 04.You Do It 2 Solution -> Public Property, parameterized constructor, Function

1. create an application named **You Do It 2** and save it in the ...VB2017\Chap10\Exercise\04.You Do It 2 Solution
2. add: **text box, label**, and a **button** to the form
3. add a **Class** file named **Circle.vb** to the project and define a **Class** named **Circle** that contains:
  - one attribute/**property**: the circle's radius
  - a default constructor
  - a parameterized constructor
  - a method that calculates and returns the circle's area using the following formula:  $3.141592 * \text{radius}^2 = (\text{radius} * \text{radius}) * \pi$
4. open the form's Code Editor window and code the button's Click event procedure that:
  - it instantiates a Circle Object using the radius entered by the user to initialize the Object's radius
  - the procedure should calculate the Object's area and then display the area in the label control
5. save the solution and then start and test the application

code for Circle.vb, frmMain.vb & my GUI:

```
1  '3. add a Class file named Circle.vb to the project and define a Class named Circle that contains:
2  ' - one attribute/property: the circle 's radius
3  ' - a default constructor
4  ' -a parameterized constructor
5  ' - a method that calculates and returns the circle's area using the following formula: 3.141592 * radius^2
6  ' txtRadius = dblRadius = dblRadiusCircleVb
7
8  Option Explicit On
9  Option Strict On
10 Option Infer Off
11
12 Public Class Circle
13     Private dblRadiusCircleVb As Double
14
15     Public Property Radius As Double
16         Get
17             Return dblRadiusCircleVb
18         End Get
19         Set(value As Double)
20             If value > 0 Then
21                 dblRadiusCircleVb = value
22             Else
23                 dblRadiusCircleVb = 0
24             End If
25         End Set
26     End Property
27
28     Public Sub New()          ' default constructor
29         dblRadiusCircleVb = 0
30     End Sub
```

```

31
32     Public Sub New(ByRef dblRad As Double)      ' parameterized constructor
33         Radius = dblRad
34     End Sub
35
36     Public Function GetCircleArea() As Double
37         Return (dblRadiusCircleVb ^ 2) * 3.141592      ' actual calculation
38     End Function
39
40 End Class

```

```

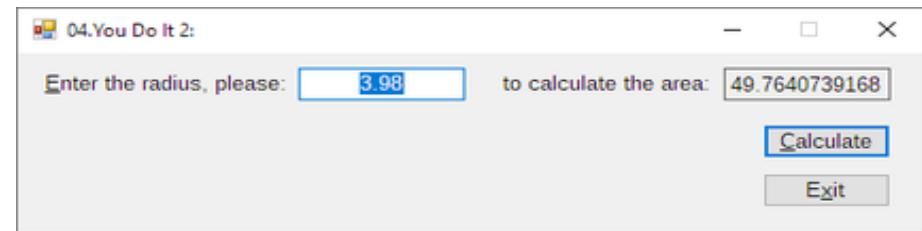
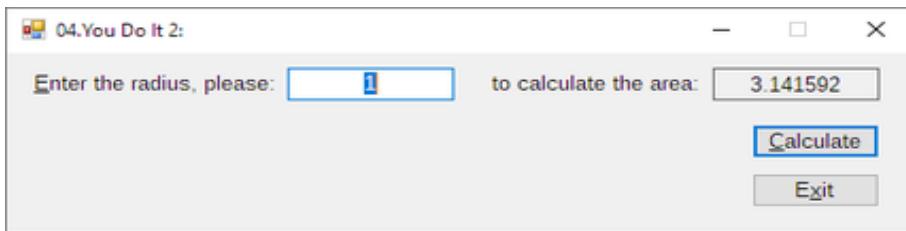
1  '2. add: text box, label, and a button to the form
2  '3. add a Class file named Circle.vb to the project and define a Class named Circle that contains:
3  ' - one attribute/property: the circle's radius
4  ' - a default constructor
5  ' -a parameterized constructor
6  ' - a method that calculates and returns the circle's area using the following formula: 3.141592 * radius^2
7  '4. open the form's Code Editor window and code the button's Click event procedure that:
8  ' - it instantiates a Circle Object using the radius entered by the user to initialize the Object's radius
9  ' - the procedure should calculate the Object's area and then display the area in the label control
10 '5. save the solution and then start and test the application
11 ' txtRadius = dblRadius = dblRadiusCircleVb
12 ' lblResult = dblResult
13
14 Option Explicit On
15 Option Strict On
16 Option Infer Off
17
18 Public Class frmMain
19     Private Sub btnCalculate_Click(sender As Object, e As EventArgs) Handles btnCalculate.Click
20         ' display the circle area
21
22         ' declare a variable for Circle Object:
23         Dim circle As Circle
24
25         Dim dblRadius As Double
26         Dim dblResult As Double
27
28         Double.TryParse(txtRadius.Text, dblRadius)      ' Object's initial value, linked with a Public Property named Radius
29
30         ' instantiate and initialize the circle Object:
31         circle = New Circle(dblRadius)
32

```

```

33     ' assign the calculation result from the GetCircleArea method in Circle.vb Class:
34     dblResult = circle.GetCircleArea
35
36     lblResult.Text = dblResult.ToString()
37     txtRadius.Focus()
38     txtRadius.SelectAll()
39 End Sub
40
41 Private Sub txtRadius_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtRadius.KeyPress
42     If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> "." AndAlso e.KeyChar <> ControlChars.Back Then
43         e.Handled = True
44     End If
45 End Sub
46
47 Private Sub txtRadius_TextChanged(sender As Object, e As EventArgs) Handles txtRadius.TextChanged
48     lblResult.Text = Nothing
49 End Sub
50
51 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
52     Me.Close()
53 End Sub
54 End Class

```



## CH10\_F9 - reusing a Class example: previously created Class Rectangle used in: 05.Pizzeria Solution - GUI & pseudocode & entire code & test

- a good Class is one that can be used in a variety of ways in many different applications
- in this section, you will use the Rectangle Class that you created in the 03.Franklin Solution-Parameterized in the Pete's Pizzeria application
- in this case, the Rectangle Object will represent a square pizza rather than a rectangular deck (a square is simply a rectangle that has 4 equal sides)
- the application will calculate the number of square pizza slices that can be cut from the entire pizza

- to add the Rectangle.vb file created in the 03.Franklin Solution-Parameterized to the 05.Pizzeria Solution application:

1). first you gonna copy already created Rectangle.vb file, so it can be reused:

- > open the folder: ...VB2017\Chap10\\_Exercise\03.Franklin Solution-Parameterized\Franklin Project\
- > locate the file named Rectangle.vb and:
- > copy it into the folder: ...VB2017\Chap10\\_Exercise\05.Pizzeria Solution\Pizzeria Project\

2). open the: ...VB2017\Chap10\\_Exercise\05.Pizzeria Solution\Pizzeria Solution.sln

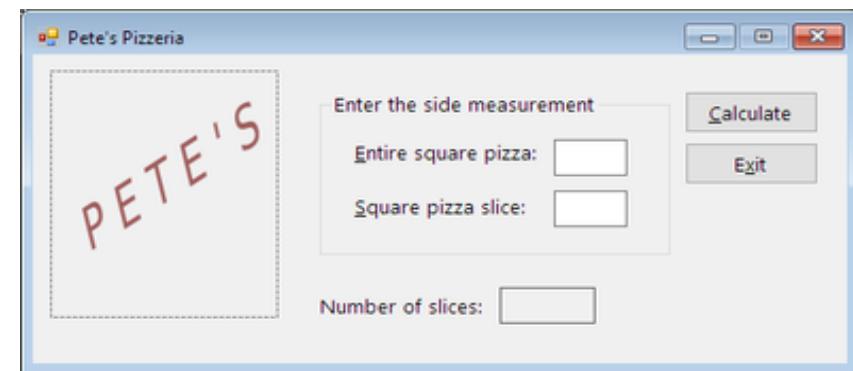
- > open the Designer window and notice that the GUI provides:
  - text box for entering the side measurement of the entire pizza
  - text box for entering the side measurement of the pizza slice
- <- the application will use these measurements to calculate the number of pizza slices

3). add the Rectangle.vb file to the project:

- > on the menu bar click: Project and choose: Add Existing Item... Ctrl+D
- > in the list of filenames choose the Rectangle.vb and click the Add button
- > in the Solution Explorer window verify that the file was added to the project

4). entire code for Rectangle.vb - Class Rectangle

```
1  ' Name:          Rectangle.vb
2  ' Programmer:    vincenttheclown on <current date>
3
4  Option Explicit On
5  Option Strict On
6  Option Infer Off
7
8  Public Class Rectangle
9    ' insertion point:
10   Private intLength As Integer
11   Private intWidth As Integer
12
13   Public Property Length As Integer      ' associated with intLength member variable
14     Get
15       Return intLength
16     End Get
17     Set(value As Integer)
18       If value > 0 Then
19         intLength = value
```



```

20         Else
21             intLength = 0
22         End If
23     End Set
24 End Property
25
26 Public Property Width As Integer      ' associated with intWidth member variable
27     Get
28         Return intWidth
29     End Get
30     Set(value As Integer)
31         If value > 0 Then
32             intWidth = value
33         Else
34             intWidth = 0
35         End If
36     End Set
37 End Property
38
39 Public Sub New()      ' default constructor
40     intLength= 0
41     intWidth = 0
42 End Sub
43
44 Public Sub New(ByVal intL As Integer, ByVal intW As Integer)      ' parameterized constructor
45     Length = intL
46     Width = intW
47 End Sub
48
49 Public Function GetArea() As Integer
50     Return intLength * intWidth
51 End Function
52
53 End Class

```

#### btnCalc Click event procedure pseudocode:

- step 1:** - declare a **Rectangle** variable named **entirePizza** to store the square Object that represents the entire pizza  
 - declare a **Rectangle** variable named **pizzaSlice** to store the square Object that represents a pizza slice

- step 2:** - declare variables:    - **intEntireSide** to store the side measurement of the entire pizza entered by the user in: (**txtEntireSide**)  
                           - **intSliceSide** to store the side measurement of a pizza slice entered by the user in: (**txtSliceSide**)  
                           - **intEntireArea** to store the area of the entire pizza  
                           - **intSliceArea** to store the area of the pizza slice  
                           - **dblSlices** to store the number of slices

**step 3:** - convert the side measurement of entire pizza entered by the user in (**txtEntireSide**) to **Integer** and store it in the **intEntireSide** variable

**step 4:** - convert the side measurement of a pizza slice entered by the user in (**txtSliceSide**) to **Integer** and store it in the **intSliceSide** variable

**step 5:** - selection structure that determines whether the side measurements entered by the user are greater than **0**

**step 5.1:** - instantiate a **Rectangle** Object and store it in the **entirePizza** variable, and  
- use the value in the **intEntireSide** variable as the Object's **length** and **width** parameters

**step 5.2:** - instantiate a **Rectangle** Object and store it in the **pizzaSlice** variable, and  
- use the value in the **intSliceSide** variable as the Object's **length** and **width** parameters

**step 5.3:** - use the **entirePizza** Object's **GetArea** method to calculate the area of the entire pizza, and  
- assign the method's return value to the **intEntireArea** variable

**step 5.4:** - use the **pizzaSlice** Object's **GetArea** method to calculate the area of a pizza slice, and  
- assign the method's return value to the **intSliceArea** variable

**step 5.5:** - calculate the number of pizza slices by dividing the area of the entire pizza by the area of a pizza slice, and  
- assign the result to the **dblSlices** variable  
- end if / end selection structure

**step 6:** - in the (**lblSlices**) display the number of pizza slices stored in **dblSlices**

- to code the **btnCalc\_Click** procedure:

1). open the form's Code Editor window and locate the **btnCalc\_Click** procedure

**step 1:** - declare a **Rectangle** variable named **entirePizza** to store the square Object that represents the entire pizza  
- declare a **Rectangle** variable named **pizzaSlice** to store the square Object that represents a pizza slice

-> below the comment line '**' Display the number of square pizza slices.**' enter the following lines of code:

```
14      'step 1: Declare a Rectangle variables:  
15      Dim entirePizza As Rectangle      ' to store the square Object that represents the entire pizza  
16      Dim pizzaSlice As Rectangle       ' to store the square Object that represents a pizza slide  
17
```

**step 2:** - declare variables

-> enter the following **comment** and **5 Dim** statements:

```
18      'step 2: Declare variables:  
19      Dim intEntireSide As Integer      ' to store the side measurement of the entire pizza entered by the user in: (txtEntireSide)  
20      Dim intSliceSide As Integer       ' to store the side measurement of a pizza slice entered by the user in: (txtSliceSide)  
21      Dim intEntireArea As Integer     ' to store the area of the entire pizza  
22      Dim intSliceArea As Integer      ' to store the area of the pizza slice  
23      Dim dblSlices As Double         ' to store the number of slices  
24
```

**step 3:** - convert the side measurement of entire pizza entered by the user in (`txtEntireSide`) to `Integer` and store it in the `intEntireSide` variable

-> enter the following `comment` and `TryParse` method:

```
25     'step 3: Convert the side measurement of entire pizza entered by the user in (txtEntireSide) to Integer and store it in the intEntireSide:  
26         Integer.TryParse(txtEntireSide.Text, intEntireSide)  
27
```

**step 4:** - convert the side measurement of a pizza slice entered by the user in (`txtSliceSide`) to `Integer` and store it in the `intSliceSide` variable

-> enter the following `comment` and `TryParse` method:

```
28     'step 4: Convert the side measurement of a pizza slice entered by the user in (txtSliceSide) to Integer and store it in the intSliceSide:  
29         Integer.TryParse(txtSliceSide.Text, intSliceSide)  
30
```

**step 5:** - selection structure that determines whether the side measurements entered by the user are greater than 0

-> enter the following `comment` and `If` clause:

```
31     'step 5: Selection structure that determines whether the side measurements entered by the user are greater than 0:  
32     If intEntireSide > 0 AndAlso intSliceSide > 0 Then  
33
```

- if both side measurements are greater than 0, the selection structure's `True` path performs 5 tasks: [step 5.1:](#) [step 5.2:](#) [step 5.3:](#) [step 5.4:](#) [step 5.5:](#)

**step 5.1:** - instantiate a `Rectangle` Object and store it in the `entirePizza` variable, and

- use the value in the `intEntireSide` variable as the Object's `length` and `width` parameters

-> enter the following `comments` and the assignment statement:

```
34     'step 5.1: Instantiate a Rectangle Object and store it in the entirePizza variable, and  
35         '           use the value in the intEntireSide variable as the Object's length and width parameters:  
36         entirePizza = New Rectangle(intEntireSide, intEntireSide)  
37
```

**step 5.2:** - instantiate a `Rectangle` Object and store it in the `pizzaSlice` variable, and

- use the value in the `intSliceSide` variable as the Object's `length` and `width` parameters

-> enter the following `comments` and the assignment statement:

```
38     'step 5.2: Instantiate a Rectangle Object and store it in the pizzaSlice variable, and  
39         '           use the value in the intSliceSide variable as the Object's length and width parameters:  
40         pizzaSlice = New Rectangle(intSliceSide, intSliceSide)  
41
```

**step 5.3:** - use the `entirePizza` Object's `GetArea` method to calculate the area of the entire pizza, and

- assign the method's return value to the `intEntireArea` variable

-> enter the following `comments` and the assignment statement:

```
42     'step 5.3: use the entirePizza Object's GetArea method to calculate the area of the entire pizza, and  
43         '           assign the method's return value to the intEntireArea variable:  
44         intEntireArea = entirePizza.GetArea  
45
```

**step 5.4:**

- use the **pizzaSlice** Object's **GetArea** method to calculate the area of a pizza slice, and
- assign the method's return value to the **intSliceArea** variable

-> enter the following **comments** and the assignment statement:

```
46      'step 5.4: use the pizzaSlice Object's GetArea method to calculate the area of a pizza slice, and  
47          '           assign the method's return value to the intSliceArea variable:  
48          intSliceArea = PizzaSlice.GetArea  
49
```

**step 5.5:**

- calculate the number of pizza slices by dividing the area of the entire pizza by the area of a pizza slice, and
- assign the result to the **dblSlices** variable

-> enter the following **comments** and the assignment statement:

```
50      'step 5.5: calculate the number of pizza slices by dividing the area of the entire pizza by the area of a pizza slice, and  
51          '           assign the result to the dblSlices variable:  
52          dblSlices = intEntireArea / intSliceArea  
53  
54      End If  
55
```

**step 6:** - in the **(lblSlices)** display the number of pizza slices stored in **dblSlices**

-> enter the following **comment** and the assignment statement:

```
56      'step 6: in the (lblSlices) display the number of pizza slices stored in dblSlices:  
57      lblSlices.Text = dblSlices.ToString("N1")
```

2). entire code for the Main Form.vb (**frmMain**)

```
1      ' Name:      Pizzeria Project  
2      ' Purpose:    Display the number of square pizza slices that can be cut from a square pizza.  
3      ' Programmer: vincenttheclown on <current date>  
4  
5      Option Explicit On  
6      Option Strict On  
7      Option Infer Off  
8  
9      Public Class frmMain  
10  
11      Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click  
12          ' Displays the number of square pizza slices.  
13  
14          'step 1: Declare a Rectangle variables:  
15          Dim entirePizza As Rectangle      ' to store the square Object that represents the entire pizza  
16          Dim pizzaSlice As Rectangle      ' to store the square Object that represents a pizza slide  
17
```

```

18     'step 2: Declare variables:
19     Dim intEntireSide As Integer      ' to store the side measurement of the entire pizza entered by the user in: (txtEntireSide)
20     Dim intSliceSide As Integer       ' to store the side measurement of a pizza slice entered by the user in: (txtSliceSide)
21     Dim intEntireArea As Integer      ' to store the area of the entire pizza
22     Dim intSliceArea As Integer       ' to store the area of the pizza slice
23     Dim dblSlices As Double          ' to store the number of slices
24
25     'step 3: Convert the side measurement of entire pizza entered by the user in (txtEntireSide) to Integer and store it in the intEntireSide:
26     Integer.TryParse(txtEntireSide.Text, intEntireSide)
27
28     'step 4: Convert the side measurement of a pizza slice entered by the user in (txtSliceSide) to Integer and store it in the intSliceSide:
29     Integer.TryParse(txtSliceSide.Text, intSliceSide)
30
31     'step 5: Selection structure that determines whether the side measurements entered by the user are greater than 0:
32     If intEntireSide > 0 AndAlso intSliceSide > 0 Then
33
34         'step 5.1: Instantiate a Rectangle Object and store it in the entirePizza variable, and
35         '           use the value in the intEntireSide variable as the Object's length and width parameters:
36         entirePizza = New Rectangle(intEntireSide, intEntireSide)      <- instantiates a Rectangle Object and uses the parameterized Constructor to
37   initialize the Object's Length Property and Width Property
38
39         'step 5.2: Instantiate a Rectangle Object and store it in the pizzaSlice variable, and
40         '           use the value in the intSliceSide variable as the Object's length and width parameters:
41         pizzaSlice = New Rectangle(intSliceSide, intSliceSide)        <- instantiates a Rectangle Object and uses the parameterized Constructor to
42   initialize the Object's Length Property and Width Property
43
44         'step 5.3: use the entirePizza Object's GetArea method to calculate the area of the entire pizza, and
45         '           assign the method's return value to the intEntireArea variable:
46         intEntireArea = entirePizza.GetArea                         <- invokes Object's GetArea method
47
48         'step 5.4: use the pizzaSlice Object's GetArea method to calculate the area of a pizza slice, and
49         '           assign the method's return value to the intSliceArea variable:
50         intSliceArea = pizzaSlice.GetArea                          <- invokes Object's GetArea method
51
52         'step 5.5: calculate the number of pizza slices by dividing the area of the entire pizza by the area of a pizza slice, and
53         '           assign the result to the dblSlices variable:
54         dblSlices = intEntireArea / intSliceArea
55
56     End If
57
58     'step 6: in the (lblSlices) display the number of pizza slices stored in dblSlices:
59     lblSlices.Text = dblSlices.ToString("N1")
60
61     End Sub
62
63     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click

```

```

62         Me.Close()
63     End Sub
64
65     Private Sub txtEntireSide_Enter(sender As Object, e As EventArgs) Handles txtEntireSide.Enter
66         txtEntireSide.SelectAll()
67     End Sub
68
69     Private Sub txtSliceSide_Enter(sender As Object, e As EventArgs) Handles txtSliceSide.Enter
70         txtSliceSide.SelectAll()
71     End Sub
72
73     Private Sub CancelKeys(sender As Object, e As KeyPressEventArgs) Handles txtEntireSide.KeyPress, txtSliceSide.KeyPress
74         ' Accept only numbers and the Backspace key.
75
76         If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
77             e.Handled = True
78         End If
79     End Sub
80
81     Private Sub ClearSlices(sender As Object, e As EventArgs) Handles txtEntireSide.TextChanged, txtSliceSide.TextChanged
82         lblSlices.Text = String.Empty
83     End Sub
84 End Class

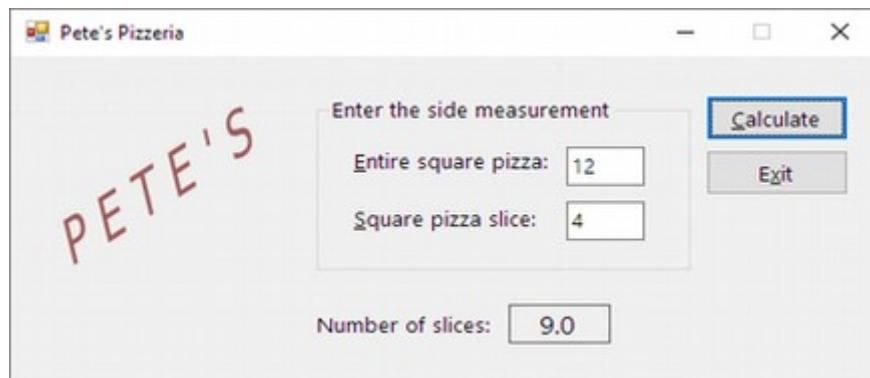
```

3). start and test the application:

- determine the number of 4-inch square slices that can be cut from a 12-inch square pizza:

- > in the **Entire square pizza** side measurement text box type: **12**
- > in the **Square pizza slice** side measurement text box type: **4**
- > click the **Calculate** button

<- the result should be: the pizza can be cut into **9** pieces.



<- easy math:  $(12*12) / (4*4) = 144 / 16 = 9$

## CH10\_APPLY THE CONCEPTS LESSON

### CH10\_A1 - use a **ReadOnly Property** procedure in a **Class** example: complete the **CourseGrade Class** used in: **06.Grade Solution** 1/2

- you gonna modify/complete the **CourseGrade Class** (**CourseGrade.vb**) in the **Grade Calculator application**, which displays a grade based on 2 test scores entered by the user

- you already learned that the **ReadOnly** keyword in a **Property** procedure's header indicates that: ->
  - > the **Property**'s value can only be retrieved (read) by an application
  - > the application cannot **Set** (write to) the **Property**
  - > a **ReadOnly Property** gets its value from the **Class** itself rather than from the application
- more info in:

#### CH10\_F3 - **Class's member variables: info & syntax ; Property** section of a **Class: Property** procedures: info & syntax

- in the next set of steps you will:

- add a **ReadOnly Property** named **Grade** to a **Class** named **CourseGrade**
- add the **Default Constructor** named **New** <- all D.C.s are named **New**, more info in:

#### CH10\_F4.1 - **Behavior** section of a **Class: Sub** procedure method **Constructor** (default & parameterized) - info & syntax & e.g.

- add a method / **Sub** procedure that will assign the appropriate grade to the member variables, associated with created **ReadOnly Property** named **Grade**

- to modify/complete the **CourseGrade Class / CourseGrade.vb**

1). open the: ...VB2017\Chap10\Exercise06.Grade Solution\Grade Solution.sln

2). open the **Designer** window

<- notice that the GUI provides **2** list boxes for entering **2** test scores that can range from **0** to **100** points each

3). in the **Solution Explorer** window locate and open the **CourseGrade.vb** file

- the **CourseGrade Class** should contain **3 Properties**:
  - test score 1 = **Public Property Score1**
  - test score 2 = **Public Property Score2**
  - letter grade = **Public Property Grade** <- missing
- the **Private** member variable **strGrade** for the letter grade is also missing from the code

-> click the blank line below the **Private intScore2 As Integer** statement and then enter the following **Private** statement:

```
8  Public Class CourseGrade  
9      Private intScore1 As Integer  
10     Private intScore2 As Integer  
11     Private strGrade As String      ' member variable for the letter grade  
12
```

4). next, you will create a **Public Property** for the **Private strGrade** member variable

- you will make the **Property ReadOnly** so that the **Class** determines the appropriate grade, rather than the **Grade Calculator application**

- i.e. you gonna have to include a code to determine the grade using some method for instance

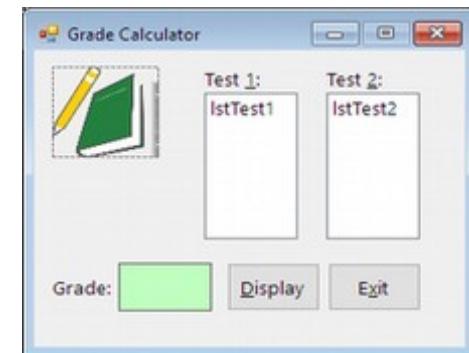
- by making the **Property ReadOnly**, the application will only be able to retrieve the grade, it will not be able to change the grade

- i.e. you gonna have to include a code to determine the grade using some method for instance

-> click the blank line immediately above the **End Class** clause and then enter the following **Property Procedure** header and **Get** clause:

<- notice: when you press Enter after typing the **Get** clause, the **Code Editor** automatically includes the **End Get** and **End Property** clauses in the procedure

<- notice: it does not enter the **Set** block of code because the header contains the **ReadOnly** keyword



```
8  Public Class CourseGrade  
...  
31     Public ReadOnly Property Grade As String      ' Public Property for the strGrade member variable  
32         Get  
33             ' ReadOnly = the app can only retrieve, can't change the grade  
34         End Get  
35     End Property  
36
```

5). type the following **Return** statement in the blank line below **Get** clause:

```
8  Public Class CourseGrade  
...  
32     Get  
33         Return strGrade      ' ReadOnly = the app can only retrieve, can't change the grade  
34     End Get
```

6). next, you will enter the **default constructor** in the **Class**

- the **default constructor** will initialize the **Private** member variables when a **CourseGrade** Object is instantiated

-> above the **End Class** clause enter the following **default constructor**:

```
8  Public Class CourseGrade  
...  
37     Public Sub New()      ' Default Constructor will initialize member variables when a CourseGrade Object is instantiated  
38         intScore1 = 0  
39         intScore2 = 0  
40         strGrade = String.Empty  
41     End Sub  
42
```

7). finally, you will enter the **DetermineGrade** method, which will assign the appropriate letter grade to the **strGrade** member variable

- the method will be a **Sub** procedure because it will not need to return a value to the application that invokes it

-> above the **End Class** clause then enter the following selection structure:

```
8  Public Class CourseGrade  
...  
43     Public Sub DetermineGrade()      ' your method will assign the appropriate letter grade to the strGrade member variable  
44         Select Case intScore1+ intScore2  
45             Case Is >= 180  
46                 strGrade = "A"  
47             Case Is >= 160  
48                 strGrade = "B"  
49             Case Is >= 140  
50                 strGrade = "C"  
51             Case Is >= 120  
52                 strGrade = "D"  
53             Case Else
```

```
54         strGrade = "F"
55     End Select
56 End Sub
57
58 End Class
```

8). entire code for the [CourseGrade Class / CourseGrade.vb](#):

```
1  ' Name:          CourseGrade.vb
2  ' Programmer:    vincenttheclown on <current date>
3
4  Option Explicit On
5  Option Strict On
6  Option Infer Off
7
8  Public Class CourseGrade
9      Private intScore1 As Integer
10     Private intScore2 As Integer
11     Private strGrade As String      ' member variable for the letter grade
12
13     Public Property Score1 As Integer
14         Get
15             Return intScore1
16         End Get
17         Set(value As Integer)
18             intScore1 = value
19         End Set
20     End Property
21
22     Public Property Score2 As Integer
23         Get
24             Return intScore2
25         End Get
26         Set(value As Integer)
27             intScore2 = value
28         End Set
29     End Property
30
31     Public ReadOnly Property Grade As String      ' Public Property for the strGrade member variable
32         Get
33             Return strGrade
34         End Get
35     End Property
36
```

```

37     Public Sub New()      ' Default Constructor will initialize member variables when a CourseGrade Object is instantiated
38         intScore1 = 0
39         intScore2 = 0
40         strGrade = String.Empty
41     End Sub
42
43     Public Sub DetermineGrade()    ' your method will assign the appropriate letter grade to the strGrade member variable
44         Select Case intScore1 + intScore2
45             Case Is >= 180
46                 strGrade = "A"
47             Case Is >= 160
48                 strGrade = "B"
49             Case Is >= 140
50                 strGrade = "C"
51             Case Is >= 120
52                 strGrade = "D"
53             Case Else
54                 strGrade = "F"
55         End Select
56     End Sub
57
58 End Class

```

- 9). now that you have finished defining the **Class**, you can use the **Class** to instantiate a **CourseGrade** Object in the **Grade Calculator application**, which displays a grade based on two test scores entered by the user

#### CH10\_A1.1 - use a **ReadOnly Property** procedure in a **Class** example: complete the **frmMain Class** used in: **06.Grade Solution 2/2**

- now, when the **CourseGrade Class** (**CourseGrade.vb**) is finished, you will complete the **Main Form Class** (**frmMain.vb**) in the **Grade Calculator application**, which displays a grade based on 2 test scores entered by the user

- to complete the **frmMain Class / Main Form.vb** = to complete the **Grade Calculator application**:

- 1). open the: ...VB2017\Chap10\Exercise06.Grade Solution\Grade Solution.sln
- 2). open the form's **Code Editor** window and locate the **btnDisplay\_Click** procedure

- first, the procedure will instantiate a **CourseGrade** Object, so:

-> under the comment line enter the following **Dim** statement:

```

9     Public Class frmMain
10    Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
11        ' Calculates and displays a letter grade.
12
13        Dim studentGrade As New CourseGrade      ' instantiate a CourseGrade Object and name it as studentGrade.
14

```

3). now the procedure will assign the test scores selected in the list boxes to the Object's **Properties**

-> below the **Dim** statement enter the **comment** and **2 TryParse** methods:

```
15      ' Assign 2 test scores to Object's 2 Properties:  
16      Integer.TryParse(lstTest1.SelectedItem.ToString, studentGrade.Score1)      ' Property Score1  
17      Integer.TryParse(lstTest2.SelectedItem.ToString, studentGrade.Score2)      ' Property Score2  
18
```

4). next, the procedure will use the Object's **DetermineGrade** method / **Sub** procedure to determine the appropriate grade

-> enter the **comment** and the statement:

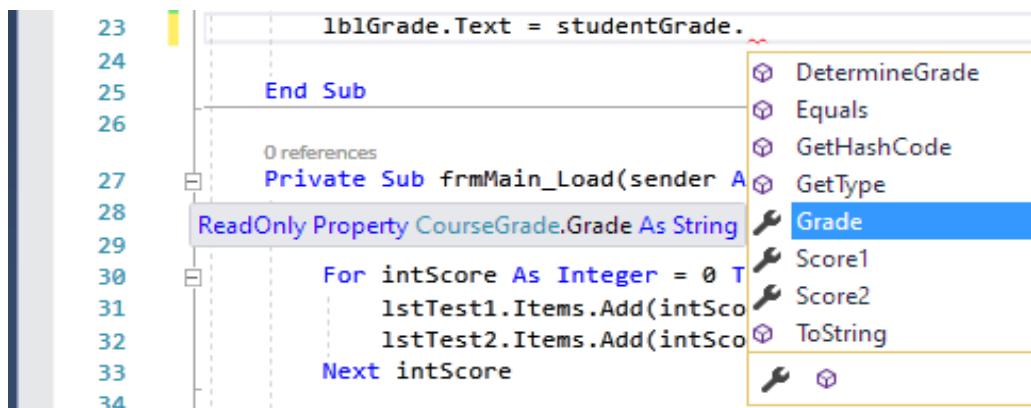
```
19      ' Calculate grade invoking/using/calling Object's DetermineGrade method/Sub procedure:  
20      studentGrade.DetermineGrade()  
21
```

5). finally, the procedure will display the grade stored in the Object's **ReadOnly Grade Property**

-> below the **comment** on line **22** type the following code - including the period at the end - but don't press Enter

```
22      ' Display grade stored in Object's ReadOnly Grade Property:  
23      lblGrade.Text = studentGrade.
```

-> in the **IntelliSense** list click Grade to view info about selected item:



<- the message that appears next to the **IntelliSense** list indicates that the **Grade Property**, belonging to the **CourseGrade Class**, is **ReadOnly**

-> press **Tab** key to include the **Grade Property** in the assignment statement and then save the solution

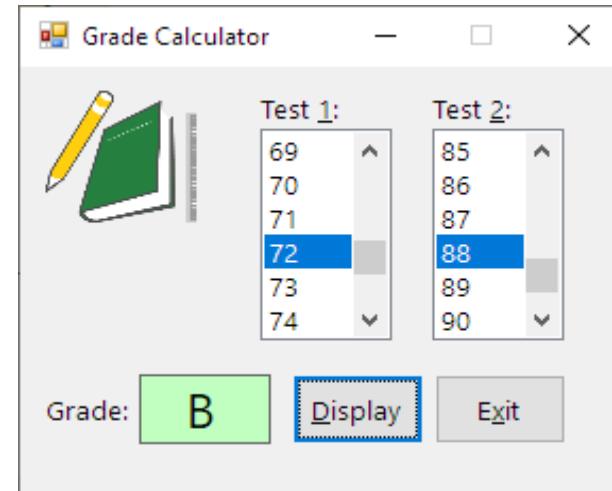
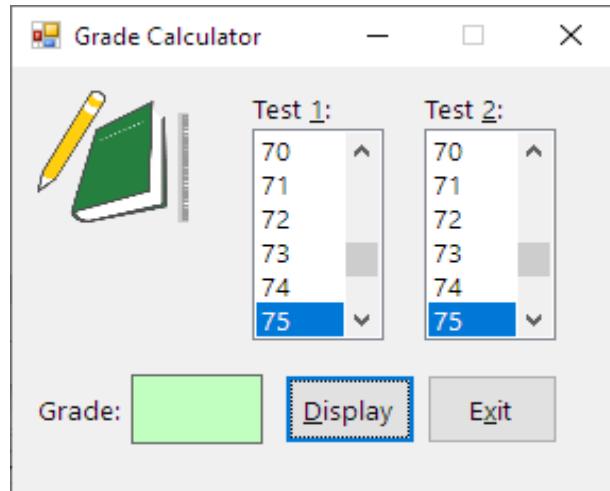
```
22      ' Display grade stored in Object's ReadOnly Grade Property:  
23      lblGrade.Text = studentGrade.Grade      ' refers to the Object's ReadOnly Grade Property
```

6). start and test the application:

-> choose: **Test 1 = 72**, **Test 2 = 88** and click the **Display** button

<- the letter **B** appears in the Grade box

- test by your own and when done, close the solution.



7). entire code for the **frmMain Class / Main Form.vb**:

```
1  ' Name:      Grade Project
2  ' Purpose:    Displays a grade based on two test scores.
3  ' Programmer: vincenttheclown on <current date>
4
5  Option Explicit On
6  Option Strict On
7  Option Infer On
8
9  Public Class frmMain
10     Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
11         ' Calculates and displays a letter grade.
12
13         Dim studentGrade As New CourseGrade      ' instantiate a CourseGrade Object and name it as studentGrade.
14
15         ' Assign 2 test scores to Object's 2 Properties:
16         Integer.TryParse(lstTest1.SelectedItem.ToString, studentGrade.Score1)      ' Property Score1
17         Integer.TryParse(lstTest2.SelectedItem.ToString, studentGrade.Score2)      ' Property Score2
18
19         ' Calculate grade invoking/using/calling Object's DetermineGrade method/Sub procedure:
20         studentGrade.DetermineGrade()
21
```

```
22     ' Display grade stored in Object's ReadOnly Grade Property:  
23     lblGrade.Text = studentGrade.Grade      ' refers to the Object's ReadOnly Grade Property  
24  
25 End Sub  
26  
27 Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load  
28     ' Fills the list boxes with values and selects default values.  
29  
30     For intScore As Integer = 0 To 100  
31         lstTest1.Items.Add(intScore.ToString)  
32         lstTest2.Items.Add(intScore.ToString)  
33     Next intScore  
34  
35     lstTest1.SelectedItem = "75"  
36     lstTest2.SelectedItem = "75"  
37 End Sub  
38  
39 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click  
40     Me.Close()  
41 End Sub  
42  
43 Private Sub ClearGrade(sender As Object, e As EventArgs) Handles lstTest1.SelectedValueChanged, lstTest2.SelectedValueChanged  
44     lblGrade.Text = String.Empty  
45 End Sub  
46  
47 End Class
```

## CH10\_A2 - create auto-implemented Public Property: it automatically creates hidden Private member variable: \_variable & hidden Get & Set block of codes

- the auto-implemented Property feature enables you to specify the Property of a Class in just 1 line of code

auto-implemented Public Property procedures syntax:

```
Public Property YourPropertyName As dataType
```

- automatically creates: a hidden Private member variable: \_variable, a hidden Get block of code, a hidden Set block of code
- can be used when: - basically when Public Property has no optional keywords/instructions
- can not be used when:
  - Public Property is either ReadOnly or WriteOnly
  - Set block of code contains validation code
- although the auto-implemented Property feature provides a shorter syntax for you to use when creating a Class, keep in mind that you will need to use the standard syntax if you want to add validation code to the Set block, or if you want the Property to be either ReadOnly or WriteOnly

e.g. `Public Property Score1 As Integer`

- it automatically creates:
  1. a hidden Private member variable associated with the Property
    - the name will be same as the Property's name, but it will be preceded by an underscore\_

e.g.1 `Private _Score1 As Integer`

2. a hidden Get block of code

e.g.2 `Get`  
    `Return _Score1`  
    `End Get`

3. a hidden Set block of code

e.g.3 `Set(value As Integer)`  
        `_Score1 = value`  
        `End Set`

## CH10\_A2.1 - use auto-implemented Public Property: example with CourseGrade Class used in 07.Grade Solution-autoimplemented

- to use the auto-implemented Property feature in previously created CourseGrade Class / CourseGrade.vb

1). open the: ...VB2017\Chap10\_Exercises\07.Grade Solution-autoimplemented\Grade Solution.sln

2). in the Solution Explorer window open the CourseGrade.vb file

-> replace the first 2 Private member variable declaration statements with the following auto-implemented Property statements:

```
8  Public Class CourseGrade  
9    Private intScore1 As Integer  
10   Private intScore2 As Integer  
11   Private strGrade As String      ' member variable for the letter grade
```

```
9    Public Property Score1 As Integer      ' auto-implemented Property Score1  
10   Public Property Score2 As Integer      ' auto-implemented Property Score2  
11   Private strGrade As String            ' member variable for the letter grade
```

3). you gonna delete the original standard **Property Procedures**, because you already replaced them

-> delete the **Score1 Property** procedure and **Score2 Property** procedure, but do not delete the **ReadOnly Grade Property** procedure

4). recall that the name of the **Private** member variable associated with an **auto-implemented Property** is the **Property's name preceded by an underscore**\_

- you gonna have to change the original **Private** member variable names from **intScore1** to **\_Score1** and **intScore2** to **\_Score2**

-> so, change the names in the **default Constructor** method and **DetermineGrade** method:

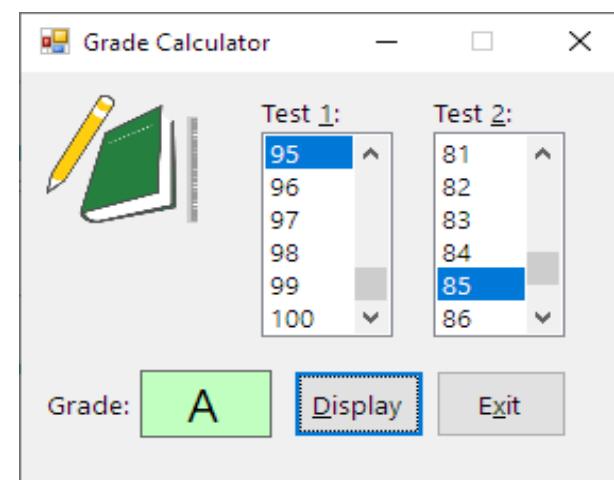
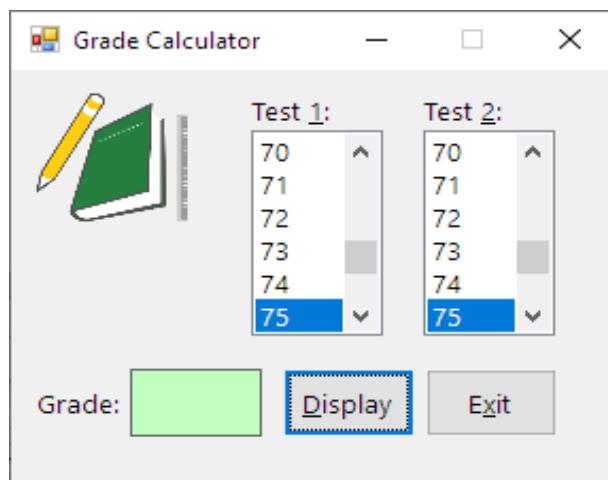
```
8  Public Class CourseGrade  
...  
37     Public Sub New()      ' Default Constructor will initialize member variables when a CourseGrade Object is instantiated  
38         intScore1 = 0  
39         intScore2 = 0  
40         strGrade = String.Empty  
41     End Sub  
42  
43     Public Sub DetermineGrade()      ' your method will assign the appropriate letter grade to the strGrade member variable  
44         Select Case intScore1 + intScore2  
  
38             _Score1 = 0  
39             _Score2 = 0  
  
44         Select Case _Score1 + _Score2
```

5). start and test the modified **Grade Calculator** application:

-> choose: **Test 1 = 95, Test 2 = 85** and click the **Display** button

<- the letter **A** appears in the Grade box

- test by your own and when done, close the solution.



6). entire code for the **CourseGrade Class / CourseGrade.vb**:

```
1  ' Name:          CourseGrade.vb
2  ' Programmer:    vincenttheclown on <current date>
3
4  Option Explicit On
5  Option Strict On
6  Option Infer Off
7
8  Public Class CourseGrade
9      Public Property Score1 As Integer      ' auto-implemented Property Score1
10     Public Property Score2 As Integer      ' auto-implemented Property Score2
11     Private strGrade As String           ' member variable for the letter grade
12
13    Public ReadOnly Property Grade As String      ' Public Property for the strGrade member variable
14        Get
15            Return strGrade
16        End Get
17    End Property
18
19    Public Sub New()      ' Default Constructor will initialize member variables when a CourseGrade Object is instantiated
20        _Score1 = 0
21        _Score2 = 0
22        strGrade = String.Empty
23    End Sub
24
25    Public Sub DetermineGrade()      ' your method will assign the appropriate letter grade to the strGrade member variable
26        Select Case _Score1 + _Score2
27            Case Is >= 180
28                strGrade = "A"
29            Case Is >= 160
30                strGrade = "B"
31            Case Is >= 140
32                strGrade = "C"
33            Case Is >= 120
34                strGrade = "D"
35            Case Else
36                strGrade = "F"
37        End Select
38    End Sub
39
40 End Class
```

### CH10\_A3 - You Do It 3: create and use auto-implemented Public Property exercise: 08.You Do It 3 Solution

1. create an application named **You Do It 3** and save it in the ...VB2017\Chap10\Exercise\08.You Do It 3 Solution
2. add: **text box**, **label**, and a **button** to the form
3. add a **Class** file named **Square.vb** to the project and define a **Class** named **Square** that contains:
  - an auto-implemented **Property** that will store the side measurement of a square
  - a default constructor
  - method that calculates and returns the square's perimeter [obvod] using following formula: **4 \* side**
4. open the form's Code Editor window and code the buttons Click event procedure so it:
  - instantiates a **Square** Object
  - calculate the square's perimeter [obvod] using the side measurement entered by the user
  - display the perimeter in the label control
5. save the solution and then start and test the application

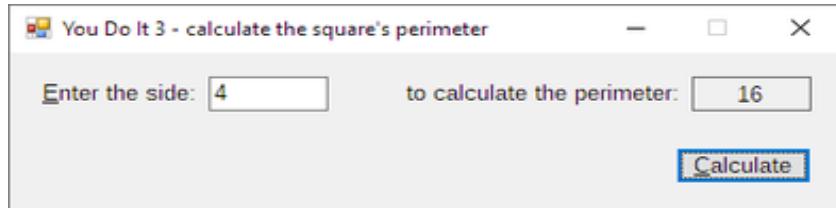
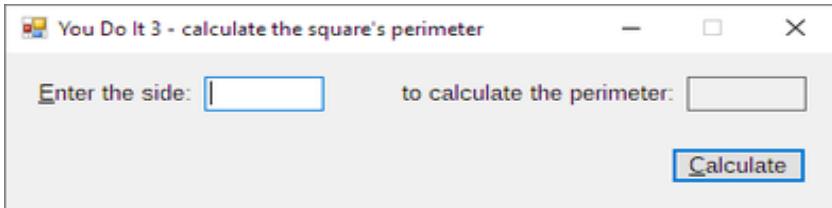
my GUI & code for **Square.vb** and **frmMain.vb**:

```
1  ' 3. add a Class file named Square.vb to the project and define a Class named Square that contains:
2  ' - an auto-implemented Property that will store the side measurement of a square
3  ' - a default constructor
4  ' - method that calculates and returns the square's perimeter [obvod] using following formula: 4 * side
5
6 Option Explicit On
7 Option Strict On
8 Option Infer Off
9
10 Public Class Square
11     Private dblPerimeter As Double      ' used as ReadOnly result from calculation
12     Public Property Side As Double      ' auto-implemented Property
13
14     Public ReadOnly Property Perimeter As Double
15         Get
16             Return dblPerimeter
17         End Get
18     End Property
19
20     Public Sub New()      ' default constructor
21         _Side = 0
22         dblPerimeter = 0
23     End Sub
24
25     Public Sub DeterminePerimeter()      ' method that calculates
26         dblPerimeter = 4 * _Side
27     End Sub
28
29 End Class
```

```

1  ' 4. open the form's Code Editor window and code the buttons Click event procedure so it:
2  ' - instantiates a Square Object
3  ' - calculate the square's perimeter [obvod] using the side measurement entered by the user
4  ' - display the perimeter in the label control
5
6
7  Option Explicit On
8  Option Strict On
9  Option Infer Off
10
11 Public Class frmMain
12     Private Sub btnCalculate_Click(sender As Object, e As EventArgs) Handles btnCalculate.Click
13         Dim XSquare As New Square
14
15         Double.TryParse(txtSide.Text, XSquare.Side)
16
17         XSquare.DeterminePerimeter()
18
19         lblPerimeter.Text = XSquare.Perimeter.ToString
20
21     End Sub
22
23     Private Sub txtSide_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtSide.KeyPress
24         If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> "." AndAlso e.KeyChar <> ControlChars.Back Then
25             e.Handled = True
26         End If
27     End Sub
28 End Class

```



## CH10\_A4 - use an **overload method** / a method with several signatures in a **Class** - info and example with **Employee Class**

- **overloaded methods:** = when 2 or more methods have the same name but different parameters
  - you can **overload** any of the methods contained in a **Class**
  - **overloading** is useful when **2** or more methods require different parameters (*parameterList*) to perform essentially the same task

e.g.1 the **2** constructors in **Figure A4-1** are considered **overloaded methods** because each is named **New** and each has a different parameters (*parameterList*)

```
8  Public Class Employee  
...  
12  Public Sub New()  
13      _Number = String.Empty  
14      _EmpName = String.Empty  
15  End Sub  
16  
17  Public Sub New(ByVal strNum As String, ByVal strName As String)  
18      Number = strNum  
19      EmpName = strName  
20  End Sub
```

<- initializes the hidden **Private** member variables directly

<- uses the **Public Properties** to initialize the **Private** member variables indirectly

default constructor

parameterized constructor

overloaded constructors

<- both **overloaded constructors** in the **Employee Class** initialize the **Class's Private** member variables, however:

- the **default constructor** does not need to be passed any information to perform the task, whereas
- the **parameterized constructor** requires **2 items** of information: the employee number and the employee name

e.g.2 the **2 GetGross** methods in **Figure A4-1** are also **overloaded methods** because they have the same name but a different parameters (*parameterList*)

```
8  Public Class Employee  
...  
22  Public Function GetGross(ByVal dblSalary As Double) As Double  
23      ' Calculates the gross pay for salaried employees,  
24      ' who are paid twice per month.  
25  
26      Return dblSalary / 24  
27  End Function  
28  
29  Public Function GetGross(ByVal dblHours As Double, ByVal dblRate As Double) As Double  
30      ' Calculates the weekly gross pay for hourly employees.  
31  
32      Return dblHours * dblRate  
33  End Function
```

<- overloaded **GetGross** methods

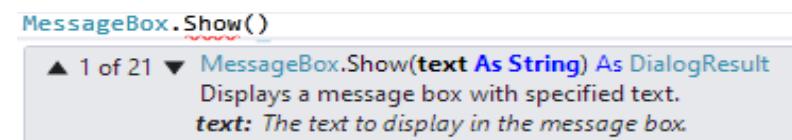
<- both **GetGross** methods in the **Employee Class** calculate and return a gross pay amount, however:

- the **1st** method performs its task for salaried employees and requires an application to pass it only **1 item** of information: - **1**). the employee's annual salary
- the **2nd** method performs its task for hourly employees and requires **2 items** of information: - **1**). the number of hours the employee worked and - **2**). his or her rate of pay
- rather than using **2 overloaded GetGross** methods, you could have used **2** methods having different names
  - e.g. **GetSalariedGross** and **GetHourlyGross**
- an advantage of **overloading** the **GetGross** method is that you need to remember the name of only **1** method

- in previous chapters, you used several of the **overloaded methods** built into Visual Basic, such as the **ToString**, **TryParse**, and **MessageBox.Show** methods
- when you enter the name of an **overloaded method** in the Code Editor window, the Code Editor's IntelliSense feature displays a box that allows you to view the method's **signatures**, one signature at a time -> recall that a method's signature includes its name and an optional parameters (*parameterList*)

e.g. the first of the **MessageBox.Show** method's 21 signatures:

- you use the up and down arrows in the box to display the other signatures
- if a **Class** you create contains **overloaded methods**, the signatures of those methods will also be displayed in the IntelliSense box



**Figure A4-1 Employee Class / Employee.vb definition:**

```

1  ' Name:      Employee.vb
2  ' Programmer: <your name> on <current date>
3
4  Option Explicit On
5  Option Strict On
6  Option Infer Off
7
8  Public Class Employee
9    Public Property Number As String
10   Public Property EmpName As String
11
12  Public Sub New()
13    _Number = String.Empty
14    _EmpName = String.Empty
15  End Sub
16
17  Public Sub New(ByVal strNum As String, ByVal strName As String)
18    Number = strNum
19    EmpName = strName
20  End Sub
21
22  Public Function GetGross(ByVal dblSalary As Double) As Double
23    ' Calculates the gross pay for salaried employees,
24    ' who are paid twice per month.
25
26    Return dblSalary / 24
27  End Function
28
29  Public Function GetGross(ByVal dblHours As Double, ByVal dblRate As Double) As Double
30    ' Calculates the weekly gross pay for hourly employees.
31
32    Return dblHours * dblRate
33  End Function
34 End Class

```

Annotations from Figure A4-1:

- Line 9: <- auto-implemented Property
- Line 10: <- auto-implemented Property
- Line 12: <- initializes the hidden **Private** member variables directly
- Line 17: <- uses the **Public Properties** to initialize the **Private** member variables indirectly
- Line 12: <- default constructor
- Line 17: <- parameterized constructor
- Line 17: <- overloaded constructors
- Line 22: <- overloaded **GetGross** methods

#### Figure A4-1 - shows the Employee Class defined in the Employee.vb file

- the Class contains:
  - 2 auto-implemented Properties on lines 9 and 10
  - 4 methods:
    - a default constructor on line 12
    - a parameterized constructor on line 17
    - 2 Functions GetGross on lines 22 and 29

<- notice that:

- the **default constructor** on line 12 initializes the Class's **Private** member variables directly, while
- the **parameterized constructor** on line 17 uses the Class's **Public Properties** to initialize the **Private** member variables indirectly
- as you already learned, using a **Public Property** in this manner ensures that the computer processes any validation code associated with the **Property**
- even though the **Property Number** and the **Property EmpName** do not have any validation code, you should use the **Properties** in the parameterized constructor in case validation code is added to the **Class** in the future

- now you can use a **Class** named **Employee** to instantiate an Object

- **Employee** Objects have the attributes and behaviors listed below:

##### Attributes of an Employee Object:

1. employee number
2. employee name

##### Behaviors of an Employee Object:

1. An **Employee** object can initialize its attributes using values provided by the **Class** <- the default constructor on line 12
2. An **Employee** object can initialize its attributes using values provided by the application in which it is instantiated <- the parameterized constructor on line 17
3. An **Employee** object can calculate and return the gross pay for salaried employees, who are paid twice per month <- the **GetGross** function on line 22
  - the gross pay is calculated by dividing the salaried employee's annual salary by number 24
4. An **Employee** object can calculate and return the gross pay for hourly employees, who are paid weekly <- the **GetGross** function on line 29
  - the gross pay is calculated by multiplying the number of hours the employee worked during the week by his or her pay rate

#### CH10\_A4.1 - use an **overload method** / a method with several signatures in a **Class** - example with **Employee Class** used in **09.Woods Solution**

- you will use the **Employee Class** when coding the **Woods Manufacturing application**, which displays the **gross pay** for salaried and hourly employees
- salaried employees:
  - paid twice per month
    - the gross pay is calculated by dividing annual salary by number 24
- hourly employees:
  - paid weekly
    - the gross pay is calculated by multiplying the number of hours the employee worked during the week by hourly pay rate
- the application also displays a report showing each employee's number, name, and gross pay

##### btnCalc Click pseudocode:

- step 1:** - declare an **Employee** variable named **ourEmployee** / instantiate an **Employee** object and store it in the **ourEmployee** variable

```
Dim ourEmployee As Employee
```

- declare variables:
  - **dblAnnualSalary** to store the annual salary
  - **dblHours** to store the hours worked
  - **dblHourRate** to store hourly pay rate
  - **dblGross** to store the gross pay

- step 2:** - instantiate an **Employee** object to represent an employee and initialize the object's variables using the user's input ->
  - > the **number** entered in the (**txtNum**) and the **name** entered in the (**txtName**)

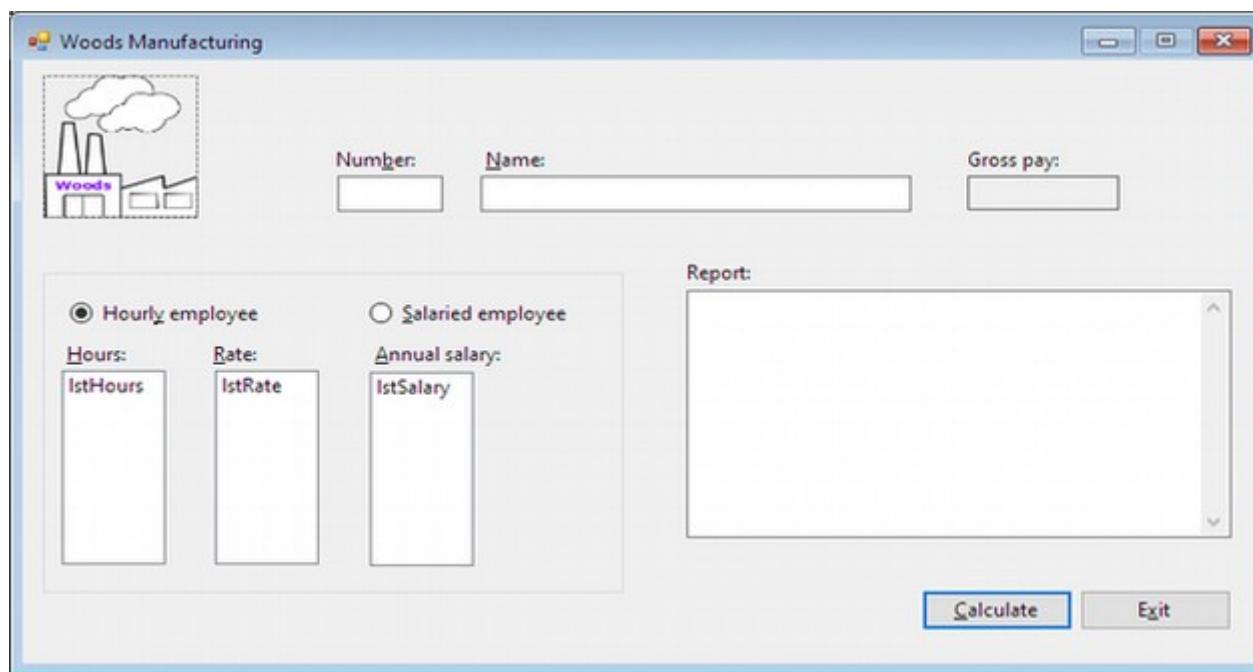
- the selection structure determines the selected radio button for:
  - a). hourly employee (radHourly)
  - b). salaried employee (radSalaried)

- and takes the appropriate action:

- if the Hourly employee radio button is selected: (radHourly)
  1. assign the hours worked and the hourly pay rate to the **dblHours** and **dblHourRate** variables
  2. use the **Employee** Object's **GetGross** function to calculate the gross pay
- else: (radSalaried)
  1. assign the annual salary to the **dblAnnualSalary** variable
  2. use the **Employee** Object's **GetGross** function to calculate the gross pay

step 3:

step 4: - display the gross pay and the report and then send the focus to the (**txtNum**) control



1). open the: ...VB2017\Chap10\Exercise09.Woods Solution\Woods Solution.sln

2). in the **Solution Explorer** window open the **Employee.vb** file and view the code same like in: **Figure A4-1 Employee Class / Employee.vb definition**

3). open the form's **Code Editor** window and locate the **btnCalc\_Click** procedure on line **10**

- step 1:**
- declare an **Employee** variable named **ourEmployee** / instantiate an **Employee** object and store it in the **ourEmployee** variable
  - declare **4** necessary variables

-> below the comment on line **13**: '**step 1\_Declare variables:**' enter the following lines of code:

```
9  Public Class frmMain  
...  
13      'step 1_Declare variables:  
14      Dim ourEmployee As Employee      ' instantiate an Employee object and store it in the variable  
15      Dim dblAnnualSalary As Double    ' will store the annual salary  
16      Dim dblHours As Double          ' will store the hours worked  
17      Dim dblHourRate As Double        ' will store hourly pay rate  
18      Dim dblGross As Double          ' will store the gross pay  
19
```

- step 2:**
- instantiate an **Employee** object to represent an employee and initialize the object's variables using the user's input ->
  - > the **number** entered in the (**txtNum**) and the **name** entered in the (**txtName**)

-> below the comment on line **20**: '**step 2\_Instantiate and initialize an Employee object:**' enter the following assignment statement:

```
9  Public Class frmMain  
...  
20      'step 2_Instantiate and initialize an Employee object:  
21      ourEmployee = New Employee(txtNum.Text.Trim, txtName.Text.Trim)  
22
```

- step 3:**
- the selection structure determines the selected radio button for: **a).** hourly employee      (**radHourly**)  
**b).** salaried employee      (**radSalaried**)
  - and takes the appropriate action:
    - if the Hourly employee radio button is selected: (**radHourly**)
      1. assign the hours worked and the hourly pay rate to the **dblHours** and **dblHourRate** variables
      2. use the **Employee** Object's **GetGross** function to calculate the gross pay
    - else: (**radSalaried**)
      1. assign the annual salary to the **dblAnnualSalary** variable
      2. use the **Employee** Object's **GetGross** function to calculate the gross pay

-> below the comment on line **23**: '**step 3\_Determine the selected radio button:**' enter the following lines:

```
9  Public Class frmMain  
...  
23      'step 3_Determine the selected radio button:  
24      If radHourly.Checked Then  
          'a). calculate the gross pay for an hourly employee:  
          Double.TryParse(1stHours.SelectedItem.ToString, dblHours)  
          Double.TryParse(1stRate.SelectedItem.ToString, dblHourRate)  
          dblGross = ourEmployee.GetGross(dblHours, dblHourRate)
```

<- overloaded **GetGross Function**, signature 2/2

<- overloaded **GetGross Function**, signature 2/2

<- overloaded **GetGross Function**, signature 1/2

```

29     Else
30         'b). calculate the gross pay for a salaried employee:
31         Double.TryParse(1stSalary.SelectedItem.ToString, dblAnnualSalary)
32         dblGross = ourEmployee.GetGross(dblAnnualSalary)
33     End If
34

```

<- overloaded **GetGross** Function, signature 1/2

```

28     dblGross = ourEmployee.GetGross()
29     Else
30         'b). calculate the gross pay for a salaried employee:
31         Double.TryParse(1stSalary.SelectedItem.ToString, dblAnnualSalary)
32         dblGross = ourEmployee.GetGross()
33     End If
34

```

<- overloaded **GetGross** Function, signature 2/2

**step 4:** - display the gross pay and the report and then send the focus to the (txtNum) control

-> below the comment on line 35: 'step 4\_Display the gross pay and report and then send the focus to the (txtNum) control: enter the following lines:

```

9  Public Class frmMain
...
35      'step 4_Display the gross pay and report and then send the focus to the (txtNum) control:
36      lblGross.Text = dblGross.ToString("C2")
37      txtReport.Text = txtReport.Text & ourEmployee.Number.PadRight(6) &
38          ourEmployee.EmpName.PadRight(25) &
39          dblGross.ToString("N2").PadLeft(9) & ControlChars.NewLine
40
41      txtNum.Focus()
42
End Sub

```

<- overloaded **GetGross** Function, signature 1/2

4). save the solution and then start and test the application:

-> use the following inputs and then click the **Calculate** button:

- 1). - Number = **0487**
- Name = **Shaun Jones**
- selected radio button = **Hourly employee**
- Hours = **40.0**
- Rate = **14.50**

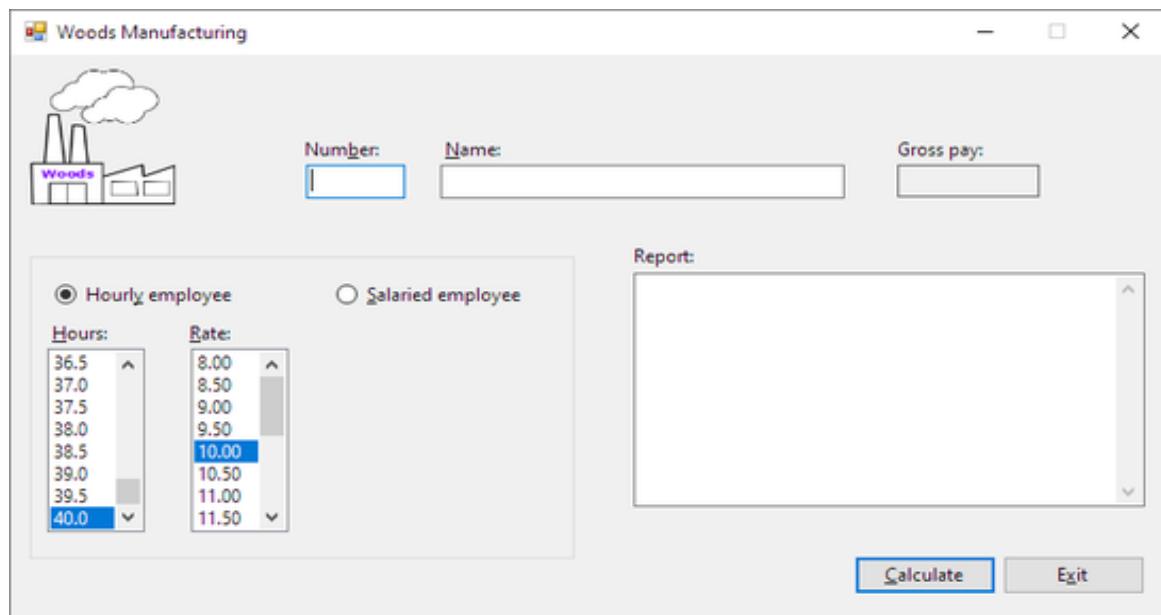
<- the Gross pay should be: **\$580.00**

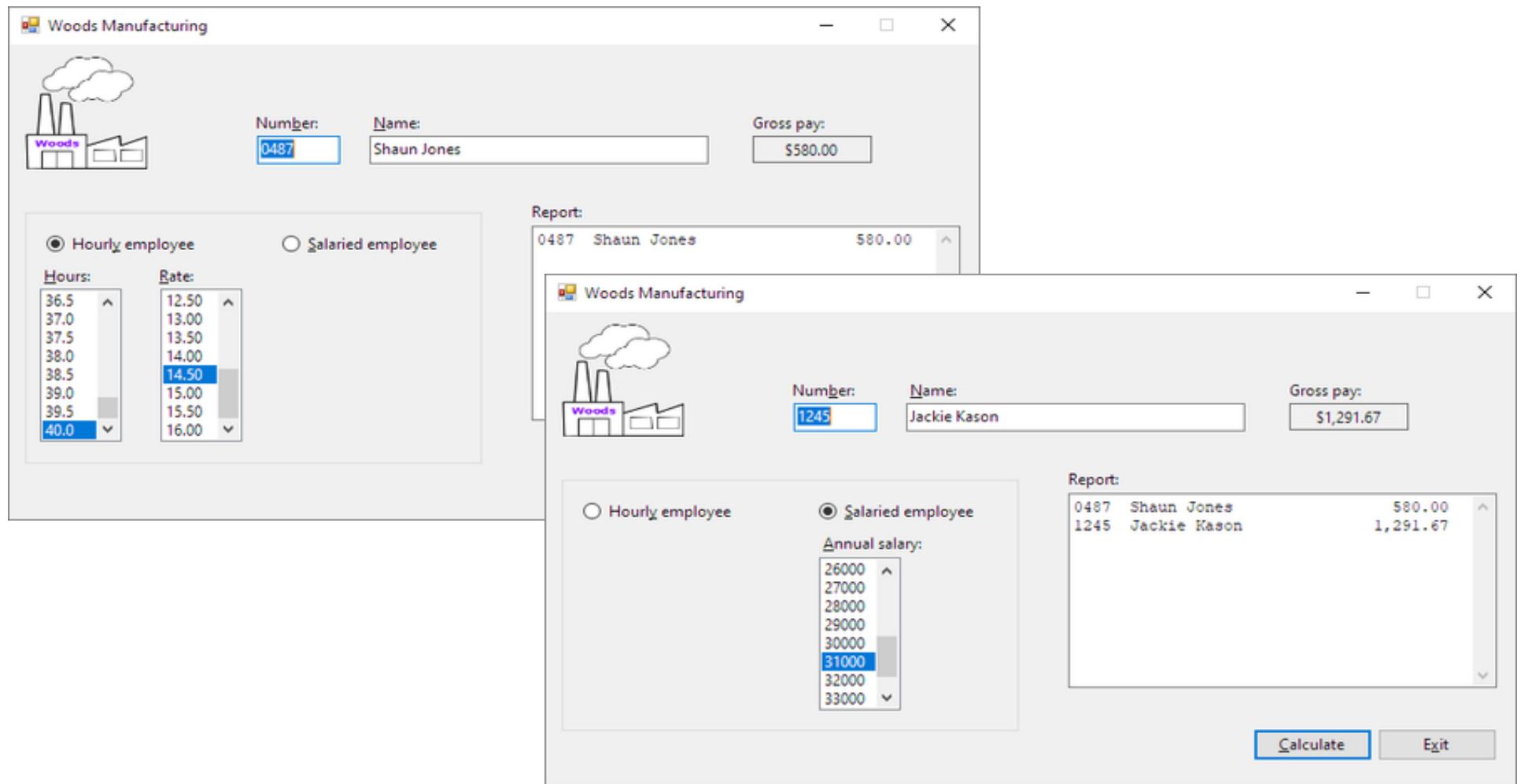
<- the Report should show: **0487 Shaun Jones 580.00**

- 2). - Number = **1245**
- Name = **Jackie Kason**
- selected radio button = **Salaried employee**
- Annual salary = **31000**

<- the Gross pay should be: **\$1,291.67**

<- the Report should show: **0487 Shaun Jones 580.00  
1245 Jackie Kason 1,291.67**





5). entire code for **frmMain / Main Form.vb** -> add there info from the book p.463 and 464

```

1  ' Name:      Woods Project
2  ' Purpose:    Calculates the gross pay for salaried and hourly employees and displays a report.
3  ' Programmer: <your name> on <current date>
4
5  Option Explicit On
6  Option Strict On
7  Option Infer Off
8
9  Public Class frmMain
10     Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
11         ' Displays the gross pay and a report.
12

```

```

13     'step 1_Declare variables:
14     Dim ourEmployee As Employee      ' instantiate an Employee object and store it in the variable
15     Dim dblAnnualSalary As Double    ' will store the annual salary
16     Dim dblHours As Double          ' will store the hours worked
17     Dim dblHourRate As Double       ' will store hourly pay rate
18     Dim dblGross As Double          ' will store the gross pay
19
20     'step 2_Instantiate and initialize an Employee object:
21     ourEmployee = New Employee(txtNum.Text.Trim, txtName.Text.Trim)
22
23     'step 3_Determine the selected radio button:
24     If radHourly.Checked Then
25         'a). calculate the gross pay for an hourly employee:
26         Double.TryParse(lstHours.SelectedItem.ToString, dblHours)
27         Double.TryParse(lstRate.SelectedItem.ToString, dblHourRate)
28         dblGross = ourEmployee.GetGross(dblHours, dblHourRate)           <- calculates the gross pay for an hourly employee
29     Else
30         'b). calculate the gross pay for a salaried employee:
31         Double.TryParse(lstSalary.SelectedItem.ToString, dblAnnualSalary)
32         dblGross = ourEmployee.GetGross(dblAnnualSalary)                  <- calculates the gross pay for a salaried employee
33     End If
34
35     'step 4_Display the gross pay and report and then send the focus to the (txtNum) control:
36     lblGross.Text = dblGross.ToString("C2")
37     txtReport.Text = txtReport.Text & ourEmployee.Number.PadRight(6) &
38             ourEmployee.EmpName.PadRight(25) &
39             dblGross.ToString("N2").PadLeft(9) & ControlChars.NewLine
40     txtNum.Focus()
41
42 End Sub
43
44 Private Sub ClearGross(sender As Object, e As EventArgs) Handles lstHours.SelectedValueChanged,
45                         lstRate.SelectedValueChanged, txtNum.TextChanged, txtName.TextChanged
46     lblGross.Text = String.Empty
47 End Sub
48
49 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
50     Me.Close()
51 End Sub
52
53 Private Sub txtNum_Enter(sender As Object, e As EventArgs) Handles txtNum.Enter
54     txtNum.SelectAll()
55 End Sub
56

```

```
23  Private Sub txtName_Enter(sender As Object, e As EventArgs) Handles txtName.Enter
24      txtName.SelectAll()
25  End Sub
26
27  Private Sub radHourly_CheckedChanged(sender As Object, e As EventArgs) Handles radHourly.CheckedChanged
28      ' Displays the labels and list boxes used to enter the number of hours worked and
29      ' the rate of pay. Hides the label and list box used to enter the salary.
30
31      lblGross.Text = String.Empty
32      lblHoursId.Visible = True
33      lstHours.Visible = True
34      lblRateId.Visible = True
35      lstRate.Visible = True
36      lblSalaryId.Visible = False
37      lstSalary.Visible = False
38  End Sub
39
40
41  Private Sub radSalaried_CheckedChanged(sender As Object, e As EventArgs) Handles radSalaried.CheckedChanged
42      ' Displays the label and list box used to enter the salary. Hides the labels and
43      ' list boxes used to enter the number of hours worked and the rate of pay.
44
45
46      lblGross.Text = String.Empty
47      lblSalaryId.Visible = True
48      lstSalary.Visible = True
49      lblHoursId.Visible = False
50      lstHours.Visible = False
51      lblRateId.Visible = False
52      lstRate.Visible = False
53  End Sub
54
55
56  Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
57      ' Fills the list boxes with values and then selects a default item in each list box.
58
59
60      For intSalary As Integer = 15000 To 35000 Step 1000
61          lstSalary.Items.Add(intSalary.ToString())
62      Next intSalary
63
64
65      For dblHour As Double = 0.5 To 40 Step 0.5
66          lstHours.Items.Add(dblHour.ToString("N1"))
67      Next dblHour
68
69
70      For dblRate As Double = 8 To 16 Step 0.5
71          lstRate.Items.Add(dblRate.ToString("N2"))
72      Next dblRate
```

```

33
34     lstHours.SelectedItem = "40.0"
31     lstRate.SelectedItem = "10.00"
32     lstSalary.SelectedItem = "25000"
33 End Sub
106 End Class

```

6). entire code for Employee Class / Employee.vb      same like in: **Figure A4-1** Employee Class / Employee.vb definition:

```

1  ' Name:          Employee.vb
2  ' Programmer:    <your name> on <current date>
3
4  Option Explicit On
5  Option Strict On
6  Option Infer Off
7
8  Public Class Employee
9      Public Property Number As String
10     Public Property EmpName As String
11
12     Public Sub New()
13         _Number = String.Empty
14         _EmpName = String.Empty
15     End Sub
16
17     Public Sub New(ByVal strNum As String, ByVal strName As String)
18         Number = strNum
19         EmpName = strName
20     End Sub
21
22     Public Function GetGross(ByVal dblSalary As Double) As Double
23         ' Calculates the gross pay for salaried employees,
24         ' who are paid twice per month.
25
26         Return dblSalary / 24
27     End Function
28
29     Public Function GetGross(ByVal dblHours As Double, ByVal dblRate As Double) As Double
30         ' Calculates the weekly gross pay for hourly employees.
31
32         Return dblHours * dblRate
33     End Function
34 End Class

```

-> auto-implemented Property

-> auto-implemented Property

-> initializes the hidden **Private** member variables directly

-> default constructor

-> overloaded constructors

-> uses the **Public Properties** to initialize the **Private** member variables indirectly

-> parameterized constructor

-> overloaded **GetGross** methods

**CH10\_Summary:** Class statement, instantiating an **Object** from a **Class**, **Property Procedure**, default and parameterized **Constructor**, invoking the **Constructor**, other methods than **Constructor** - **Function** procedure & **Sub** procedure, **auto-implemented Public Property**, **overload** the methods.

**1a.** you use the **Class** statement to create your own **Classes** in Visual Basic:

**CH10\_F2.1** - creating and adding a **Class** file to an open project: **Class's member variables & Property** section & **Behavior** section basic info

```
Public Class YourClassName
    1. Class's member variable
    2. property section
    3. behavior section
End Class
```

**CH10\_F3** - **Class's member variables: info & syntax ;**  
**Property** section of a **Class: Property** procedures: info & syntax

**CH10\_F4** - **Behavior** section of a **Class** info: usually **methods: Sub** procedures or **Functions**

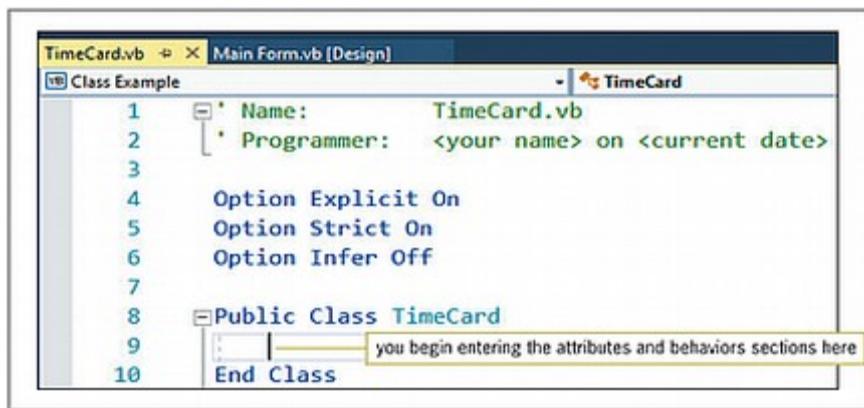


Figure 10-1 Class statement syntax, steps, and example

**1b.** to instantiate (create) an **Object** from a **Class**, use either of the syntax versions:

**CH10\_F5** - instantiating an **Object** and invoking the **constructor** - info, syntax and example

syntax v.1:

```
Dim/Private yourVariable As YourClassName
yourVariable = New YourClassName
```

As YourClassName

instantiating an Object

- the difference between the versions relates to when the **Object** is actually created

syntax v.2:

```
Dim/Private yourVariable As New YourClassName
```

As YourClassName

2. to create a **Public Property** procedure, use the syntax shown below ( **Class** member variable & **Property** section of a **Class** )
- the **Get** block allows an application to retrieve the contents of the **Private** member variable associated with the **Public Property** procedure
  - the **Set** block allows an application to assign a value to the **Private** member variable associated with the **Public Property** procedure
  - to create a **Public Property** whose value an application can only retrieve, include the **ReadOnly** keyword in the **Property** procedure's header
  - to create a **Public Property** whose value an application can only set, include the **WriteOnly** keyword in the **Property** procedure's header

**CH10\_F3 - Class's member variables: info & syntax ; Property section of a Class: Property procedures: info & syntax**

Class's member variable syntax:

```
Private/... yourMemberVariable As dataType
```

Public Property procedures syntax:

```
Public ReadOnly/WriteOnly Property YourPropertyName(parameterList) As dataType

    Get
        optional instructions
        Return yourMemberVariable
    End Get

    Set(value As dataType)
        optional instructions
        yourMemberVariable = value/defaultValue
    End Set

End Property
```

- 3a. to create a **constructor**, use the syntax shown below ( **Behavior** section of a **Class** )

- a constructor that has no parameters is called the **default constructor** -> a **Class** can have only one **default constructor**
- a constructor that has one or more parameters is called a **parameterized constructor** -> a **Class** can have as many **parameterized constructors** as needed, but each **parameterList** must be unique
- all constructors are **Sub** procedures that are named **New** -> each constructor must have a unique **parameterList** (if any) within the **Class**

**CH10\_F4 - Behavior section of a Class** info: usually **methods: Sub** procedures or **Functions**

**CH10\_F4.1 - Behavior section of a Class: Sub** procedure method **Constructor** (default & parameterized) - info & syntax & e.g.

Class's method constructor syntax:

```
Public Sub New(parameterList)
    instructions to initialize already declared Class's member variables
End Sub
```

3b. you invoke the **constructor** by using one of the syntaxes shown in 1b.

**CH10\_F5** - instantiating an **Object** and invoking the **constructor** - info, syntax and example

e.g.1 invoking the default **constructor** = no arguments, from previously created **Class Rectangle** - syntax v.1

```
Dim deck As New Rectangle
```

e.g.2 invoking the default **constructor** = no arguments, from previously created **Class Rectangle** - syntax v.2

```
Dim deck As Rectangle  
...  
deck = New Rectangle
```

e.g.3 invoking the parameterized **constructor** = 2 integer arguments, from previously created **Class Rectangle** - syntax v.1

```
Dim deck As New Rectangle(16, 14)
```

e.g.4 invoking the parameterized **constructor** = 2 integer arguments, from previously created **Class Rectangle** - syntax v.2

```
Dim deck As Rectangle  
...  
deck = New Rectangle(intDeckLen, intDeckWid)
```

4. to create a method other than a constructor, use the syntax show below:

**CH10\_F4.2 - Behavior** section of a **Class**: other **methods** than **Constructor** - **Function** procedure, **Sub** procedure - infos & syntaxes & examples

Class's other method: **Function** syntax:

```
Public Function YourMethodName() As dataType  
    instructions  
End Function
```

<- returns a value

Class's other method: **Sub** procedure syntax:

```
Public Sub YourMethodName(parameterList)  
    instructions  
End Sub
```

<- doesn't return a value

5. to specify the **Public Property** of a **Class** in one line, create an **auto-implemented Public Property**

- it automatically includes: hidden **Private member variables**, hidden **Get** block, hidden **Set** block

**CH10\_A2** - create **auto-implemented Public Property** info, syntax, example with **CourseGrade Class** used in **07.Grade Solution-autoimplemented**

**CH10\_A3 - You Do It 3**: create and use **auto-implemented Public Property** exercise: **08.You Do It 3 Solution**

6. to create two or more **methods** that perform the same task, but require different parameters,

**overload** the methods by giving them the same **name** but different **(parameterLists)**

**CH10\_F3 - Class's member variables**: info & syntax ; **Property** section of a **Class**: **Property** procedures: info & syntax

**CH10\_A4 - use an overload method** / a method with several signatures in a **Class** - info and example with **Employee Class**

**CH10\_A4.1 - use an overload method** / a method with several signatures in a **Class** - example with **Employee Class** used in **09.Woods Solution**

## CH10\_Key Terms

- **Attributes** - the characteristics that describe an **Object**
- **Auto-implemented Properties** - the Visual Basic feature that enables you to specify the **Property** of a **Class** in one line
- **Behaviors** - an **Object's** methods and events
- **Class** - a pattern that the computer follows when instantiating / creating an **Object**
- **Class statement** - the statement used to define a **Class** in Visual Basic
- **Constructor** - a method whose instructions are automatically processed each time the **Class** is used to instantiate an **Object**
  - used to initialize the **Class's Private** member variables
  - always a **Sub** procedure named **New**
- **Default constructor** - a constructor that has no parameters
  - a **Class** can have only **one** default constructor
- **Encapsulates** - an **OOP** term that means "contains" in a **Class**
- **Events** - the actions to which an **Object** can respond
- **Exposed** - in **OOP**, this term refers to the fact that an **Object's Public** members are visible to an application
- **Get block** - the section of a **Property** procedure that contains the **Get** statement
- **Get statement** - appears in a **Get** block in a **Property** procedure
  - contains the code that allows an application to retrieve the contents of the **Private** member variable associated with the **Property**
- **Hidden** - in **OOP**, this term refers to the fact that an **Object's Private** members are not visible to an application
- **Instance** - an **Object** created from a **Class**
- **Instantiated** - the process of creating an **Object** from a **Class**
- **Member variables** - the variables declared in the attributes section of a **Class**
- **Methods** - the actions that an **Object** is capable of performing
- **Object** - anything that can be seen, touched, or used
- **Object-oriented programming language** - a programming language that allows the use of **Objects** in an application's GUI and code
- **OOP** - the acronym for object-oriented programming
- **Overloaded methods** - two or more **Class** methods that have the same name but different *parameterList*
- **Parameterized constructor** - a constructor that contains one or more parameters
- **Public Property procedure** - creates a **Public Property** that an application can use to access a **Private** member variable in a **Class**
- **ReadOnly keyword** - used when defining a **Property** procedure
  - indicates that the **Property**'s value can only be retrieved / read by an application
- **Set block** - the section of a **Property** procedure that contains the **Set** statement
- **Set statement** - appears in a **Set** block in a **Property** procedure
  - contains the code that allows an application to assign a value to the **Private** member variable associated with the **Property**
  - may also contain validation code
- **Signature** - a method's name combined with its optional *parameterList*
- **WriteOnly keyword** - used when defining a **Property** procedure
  - indicates that an application can only set / write to the **Property**'s value

**CH10\_Exercises****Chap10\Exercise1****10.Pizzeria Solution-Default\_EXERCISE 1\_introductory**

- using **Default constructor** instead of **Parameterized constructor** from my **Class Rectangle**  
- `Dim x As New Rectangle`, `txt_Enter`, `txt.TextChanged`, `txt.KeyPress`, `e.KeyChar`

**11.Palace Solution\_EXERCISE 2\_introductory**

- VS: **Project / Add Existing Item...Ctrl+D** to add existing **Class**  
- instantiate an **Object** from **Class Rectangle** using a **Default constructor**: `Dim x As New Rectangle`,  
`txt_Enter`, `txt.TextChanged`, `txt.KeyPress`, `e.KeyChar`, `ControlChars.Back`

**12.Sod Solution\_EXERCISE 3\_intermediate**

- VS: **Project / Add Existing Item...Ctrl+D** to add existing **Class**  
- instantiate an **Object** from **Class Rectangle** using a **Default constructor**: `Dim x As New Rectangle`,  
`txt_Enter`, `txt.TextChanged`, `txt.KeyPress`, `e.KeyChar`, `ControlChars.Back`  
- VS: **Project / Add Existing Item...Ctrl+D** to add existing **Class**  
- instantiate an **Object** from **Class Rectangle** using a **Default constructor**: `Dim x As New Rectangle`,  
`txt_Enter`, `txt_MouseClick`, `txt.SelectAll()`, `txt.KeyPress`, `e.KeyChar`, `ControlChars.Back`,  
`frmMain_FormClosing`, `Dim dlgButton As DialogResult`, `dlgButton = MessageBox.Show(...)`,  
`If dlgButton = DialogResult.No Then...`, `txt.TextChanged`

**14.Grade Solution-Intermediate EXERCISE 5 intermediate**

- instantiate an **Object** from **Class CourseGrade** using a **Default constructor**:  
`Dim x As New CourseGrade`, `lst.Items.Add`, `lst.SelectedItem`, `txt_Enter`,  
`lst.SelectedValueChanged`, `txt.TextChanged`, `txt_MouseClick`, `txt.SelectAll()`,  
`txt.KeyPress`, `e.KeyChar`, `ControlChars.Back`, `MessageBox.Show`, `frmMain_Load`

**15.Playground Solution\_EXERCISE 6\_advanced**

- instantiate an **Object** from **Class Tringle** using a **Default constructor**: `Dim x As New Triangle`,  
`For ... To ... Step ... Next`, `lst.Items.Add`, `lst.SelectedIndex`, `lst.SelectedIndexChanged`,  
`lbl.BackColor = SystemColors.Control`, `lbl.BackColor = Color.Yellow`, `frmMain_Load`

**16.Fire Solution\_EXERCISE 7\_advanced**

- instantiate an **Object** from **Class WaterTank**: `Dim MyWaterTank As WaterTank`, ...  
- invoke a **Parameterized constructor**: `MyWaterTank = New WaterTank(inputs/parametres)`  
- `lbl.BackColor = Color.Yellow`, `lbl.BackColor = SystemColors.Control`, `txt_KeyPress`, `e.KeyChar`,  
`ControlChars.Back`, `txt_Enter`, `txt_MouseClick`, `txt_TextChanged`, `txt.SelectAll()`

**17.Parking Solution\_EXERCISE 8\_advanced**

- instantiate an **Object** from **Class ParkingLot**: `Dim MyParkingLot As ParkingLot`, ...  
- invoke a **Parameterized constructor**: `MyParkingLot = New ParkingLot(inputs/parametres)`  
- `lbl.BackColor = Color.Yellow`, `lbl.BackColor = SystemColors.Control`, `txt_KeyPress`, `e.KeyChar`,  
`ControlChars.Back`, `txt_Enter`, `txt_MouseClick`, `txt_TextChanged`, `txt.SelectAll()`

**18.Glasgow Solution\_EXERCISE 9\_advanced**

- instantiate an **Object** from **Class Dues**: `Dim MyDues As Dues`, ...  
- **Private**...class-level variable used as an accumulator, `Select Case False`, `OrElse`, `MessageBox.Show`,  
`chk.Checked`, `rad.Checked`, `lbl.BackColor = Color.Yellow`, `= SystemColors.Control`  
- invoke a **Parameterized constructor**: `MyDues = New Dues(inputs/parametres)`

**19.Serenity Solution\_EXERCISE 10\_advanced**

- 2D **Array**: `Dim int2DPrices(,) As Integer = {{50, 25, 30, 20}, {...2nd row column values}}`  
- instantiate an **Object** from **Class Check**: `Dim MyCheck As Check`, ...  
- invoke a **Parameterized constructor**: `MyCheck = New Check(inputs/parametres)`  
- `txt.Text.Trim`, `lbl.Visible =...`, `txt_KeyPress`, `e.KeyChar`, `ControlChars.Back`, `e.Handled`,  
`txt_TextChanged`, `txt_Enter`, `txt.SelectAll()`, `If value Like "##/#/#/###" Then`,  
`outFile As IO.StreamWriter`, `Public Function Save() As IO.StreamWriter`, `Return outFile`,  
`IO.File.AppendText`, `outFile.WriteLine`, `outFile.Close`, `MessageBox.Show`,  
`Public Function ShowLabel() As Boolean`

**21.FixIt Solution\_EXERCISE 12**

- instantiate an **Object** from **Class Employee** using a **Default constructor**: `Dim companyEmployee As New Employee`,  
- auto-implemented **Public Property** with hidden: **Private** member variable: `_variable & Get & Set` block of codes

- using **Default constructor** instead of **Parameterized constructor** from my [Class Rectangle](#)  
 - **Dim x As New Rectangle**, **txt\_Enter**, **txt.TextChanged**, **txt.KeyPress**,  
**e.KeyChar**, **ControlChars.Back**

1. In this exercise, you modify the Pete's Pizzeria application from this chapter's Focus lesson. Use Windows to make a copy of the Pizzeria Solution folder. Rename the copy Pizzeria Solution-Default. Open the Pizzeria Solution.sln file contained in the Pizzeria Solution-Default folder. Open the Code Editor window and locate the **btnCalc\_Click** procedure. Modify the procedure so it uses the **Rectangle** class's default constructor (rather than its parameterized constructor) when instantiating the **entirePizza** and **pizzaSlice** objects. Save the solution and then start and test the application.

```

1  ' Name:      Pizzeria Project-Default
2  ' Purpose:   Display the number of square pizza slices that can be cut from a square pizza.
3  ' Mod: - locate the btnCalc_Click procedure and:
4  '       - when instantiating the objects: entirePizza & pizzaSlice, modify the procedure so it uses
5  '           the Rectangle class's default constructor, rather than its parameterized constructor
6  ' test: 12,4 easy math: (12*12) / (4*4) = 144 / 16 = 9
7  Option Explicit On
8  Option Strict On
9  Option Infer Off
10
11 Public Class frmMain
12
13     Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
14         ' Displays the number of square pizza slices.
15
16         'step 1: Declare a Rectangle variables:
17         ' original:
18         'Dim entirePizza As Rectangle      ' to store the square Object that represents the entire pizza
19         'Dim pizzaSlice As Rectangle       ' to store the square Object that represents a pizza slide
20         ' mod:
21         Dim entirePizza As New Rectangle
22         Dim pizzaSlice As New Rectangle
23
24
25         'step 2: Declare variables:
26         ' original:
27         'Dim intEntireSide As Integer    ' to store the input side measurement of the entire pizza (txtEntireSide)
28         'Dim intSliceSide As Integer     ' to store the input side measurement of a pizza slice (txtSliceSide)
29         'Dim intEntireArea As Integer   ' to store the area of the entire pizza
30         'Dim intSliceArea As Integer    ' to store the area of the pizza slice
31         'Dim dblSlices As Double        ' to store the number of slices

```

```
32
33     ' original:
34     'Integer.TryParse(txtEntireSide.Text, intEntireSide)
35     ' mod:
36     Integer.TryParse(txtEntireSide.Text, entirePizza.Length)
37     Integer.TryParse(txtEntireSide.Text, entirePizza.Width)
38
39     ' original:
40     'Integer.TryParse(txtSliceSide.Text, intSliceSide)
41     ' mod:
42     Integer.TryParse(txtSliceSide.Text, pizzaSlice.Length)
43     Integer.TryParse(txtSliceSide.Text, pizzaSlice.Width)
44
45
46     ' original:
47     'If intEntireSide > 0 AndAlso intSliceSide > 0 Then
48     ' mod:
49     If entirePizza.Length > 0 AndAlso pizzaSlice.Length > 0 Then
50         ' original:
51         'entirePizza = New Rectangle(intEntireSide, intEntireSide)
52         'pizzaSlice = New Rectangle(intSliceSide, intSliceSide)
53
54         intEntireArea = entirePizza.GetArea
55         intSliceArea = pizzaSlice.GetArea
56         dblSlices = intEntireArea / intSliceArea
57     End If
58
59     lblSlices.Text = dblSlices.ToString("N1")
60 End Sub
61
62 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
63     Me.Close()
64 End Sub
65
66 Private Sub txtEntireSide_Enter(sender As Object, e As EventArgs) Handles txtEntireSide.Enter
67     txtEntireSide.SelectAll()
68 End Sub
69
70 Private Sub txtSliceSide_Enter(sender As Object, e As EventArgs) Handles txtSliceSide.Enter
71     txtSliceSide.SelectAll()
72 End Sub
73
```

```

74  Private Sub CancelKeys(sender As Object, e As KeyPressEventArgs) Handles txtEntireSide.KeyPress, txtSliceSide.KeyPress
75      ' Accept only numbers and the Backspace key.
76
77      If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
78          e.Handled = True
79      End If
80  End Sub
81
82  Private Sub ClearSlices(sender As Object, e As EventArgs) Handles txtEntireSide.TextChanged, txtSliceSide.TextChanged
83      lblSlices.Text = String.Empty
84  End Sub
85 End Class

```

```

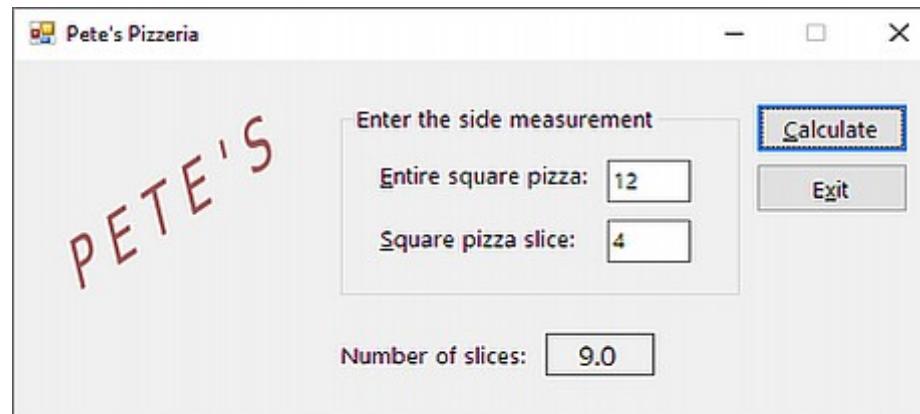
1  ' Name:          Rectangle.vb
2  ' Programmer:    vincenttheclown on <current date>
3
4  Option Explicit On
5  Option Strict On
6  Option Infer Off
7
8  Public Class Rectangle
9      ' insertion point:
10     Private intLength As Integer
11     Private intWidth As Integer
12
13     Public Property Length As Integer           ' associated with intLength member variable
14         Get
15             Return intLength
16         End Get
17         Set(value As Integer)
18             If value > 0 Then
19                 intLength = value
20             Else
21                 intLength = 0
22             End If
23         End Set
24     End Property
25

```

```

26     Public Property Width As Integer      ' associated with intWidth member variable
27         Get
28             Return intWidth
29         End Get
30         Set(value As Integer)
31             If value > 0 Then
32                 intWidth = value
33             Else
34                 intWidth = 0
35             End If
36         End Set
37     End Property
38
39     Public Sub New()      ' default constructor
40         intLength = 0
41         intWidth = 0
42     End Sub
43
44     Public Sub New(ByVal intL As Integer, ByVal intW As Integer)      ' parameterized constructor
45         Length = intL
46         Width = intW
47     End Sub
48
49     Public Function GetArea() As Integer
50         Return intLength * intWidth
51     End Function
52
53 End Class

```



## Chap10\Exercise

### 11.Palace Solution\_EXERCISE 2\_introductory

- VS: **Project / Add Existing Item...Ctrl+D** to add existing **Class**
- instantiate an **Object** from **Class Rectangle** using a **Default constructor**: **Dim x As New Rectangle**, **txt\_Enter**, **txt.TextChanged**, **txt.KeyPress**, **e.KeyChar**, **ControlChars.Back**

Open the Palace Solution.sln file contained in the VB2017\Chap10\Palace Solution folder.

- a. Use Windows to copy the Rectangle.vb file from the VB2017\Chap10 folder to the Palace Project folder. Then, use the Add Existing Item option on the Project menu to add the file to the project.
- b. Modify the Rectangle class to use Double variables rather than Integer variables.
- c. Change the name of the GetArea method to GetAreaSqFt.
- d. Add another method to the class. Use GetAreaSqYds as the method's name. The method should calculate and return the area of a rectangle in square yards.
- e. The application's Calculate button should calculate and display the number of square yards of carpeting needed to carpet a rectangular floor. Code the btnCalc\_Click procedure. Display the number of yards with one decimal place. Save the solution and then start and test the application.

```
1  ' Name:      Palace Project
2  ' Purpose:    Displays the number of square yards needed to carpet a rectangular floor.
3  ' Programmer: PoaKamaNdizi on just now
4  '_1. from VB2017\Chap10 folder copy Rectangle.vb to your Palace Project folder
5  '_2. in VS: use the Add Existing Item option on the Project menu to add the file to the project
6  '_3. modify the Rectangle class to use Double variables rather than Integer variables
7  '_4. change the name of the GetArea method to GetAreaSqFt
8  '_5. add another method to the class named GetAreaSqYds
9  '      - it should calculate and return the area of a rectangle in square yards
10 '      my info: 1 yard = 3 feet, 1 square yard = 9 square feet, therefore:
11 '          length in feet * width in feet = square footage
12 '          square footage / 9 = square yards
13 '_6. the application's Calculate button should:
14 '      - calculate the number of square yards of carpeting needed to carpet a rectangular floor
15 '      - display the number of yards with 1 decimal place
16 Option Explicit On
17 Option Strict On
18 Option Infer Off
19
20 Public Class frmMain
21     Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
22         ' Calculates the number of square yards needed to carpet a rectangular floor.
23         ' i choosed a Default Constructor:
24         Dim CarpetMeasures As New Rectangle : Dim Carpet As Double
25
26         ' i choosed a Default Constructor:
27         Double.TryParse(txtLength.Text, CarpetMeasures.Length)
28         Double.TryParse(txtWidth.Text, CarpetMeasures.Width)
29
30         Carpet = CarpetMeasures.GetAreaSqYds
31         lblSqYards.Text = Carpet.ToString("N1")
32     End Sub
```

```

33
34     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
35         Me.Close()
36     End Sub
37
38     Private Sub CancelKeys(sender As Object, e As KeyPressEventArgs) Handles txtLength.KeyPress, txtWidth.KeyPress
39         ' Accept only numbers and the Backspace key.
40         If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> "." AndAlso e.KeyChar <> ControlChars.Back Then
41             e.Handled = True
42         End If
43     End Sub
44
45     Private Sub txtLength_Enter(sender As Object, e As EventArgs) Handles txtLength.Enter
46         txtLength.SelectAll()
47     End Sub
48
49     Private Sub ClearOutput(sender As Object, e As EventArgs) Handles txtLength.TextChanged, txtWidth.TextChanged
50         lblSqYards.Text = String.Empty
51     End Sub
52
53     Private Sub txtWidth_Enter(sender As Object, e As EventArgs) Handles txtWidth.Enter
54         txtWidth.SelectAll()
55     End Sub
56 End Class

```

```

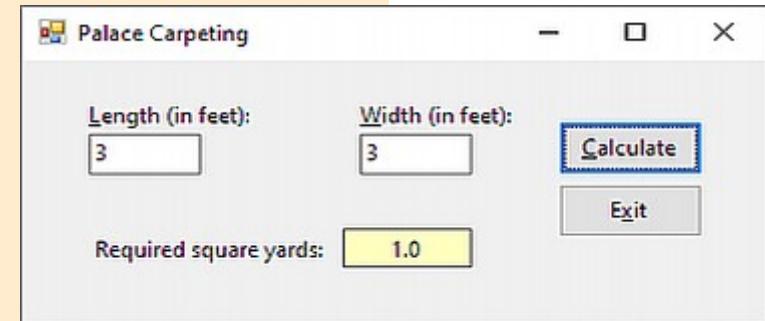
1  ' Name:      Rectangle.vb
2  ' Programmer: PoaKamaNdizi on just now
3  ' _3. modify the Rectangle class to use Double variables rather than Integer variables
4  ' _4. change the name of the GetArea method to GetAreaSqFt
5  ' _5. add another method to the class named GetAreaSqYds
6  '       - it should calculate and return the area of a rectangle in square yards
7  '       my info: 1 yard = 3 feet, 1 square yard = 9 square feet, therefore:
8  '       length in feet * width in feet = square footage
9  '       square footage / 9 = square yards
10 Option Explicit On
11 Option Strict On
12 Option Infer Off
13
14 Public Class Rectangle
15     Private dblLength As Double
16     Private dblWidth As Double
17
18     Public Property Length As Double

```

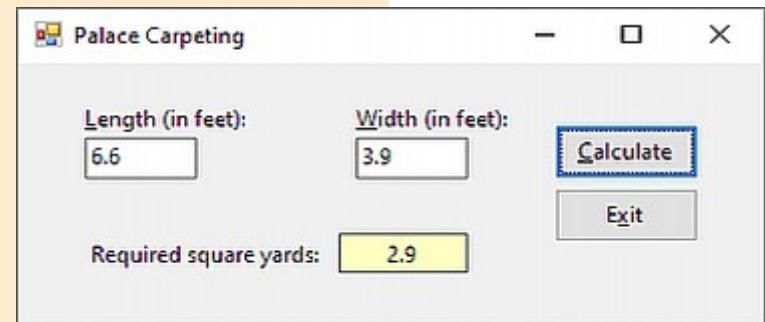
```

19     Get
20         Return dblLength
21     End Get
22     Set(value As Double)
23         If value > 0 Then
24             dblLength = value
25         Else
26             dblLength = 0
27         End If
28     End Set
29 End Property
30
31 Public Property Width As Double
32     Get
33         Return dblWidth
34     End Get
35     Set(value As Double)
36         If value > 0 Then
37             dblWidth = value
38         Else
39             dblWidth = 0
40         End If
41     End Set
42 End Property
43
44 Public Sub New() ' used for Default Constructor
45     dblLength = 0
46     dblWidth = 0
47 End Sub
48
49 Public Sub New(ByVal dblL As Double, ByVal dblW As Double) ' used for Parameterized Constructor
50     Length = dblL
51     Width = dblW
52 End Sub
53
54 Public Function GetAreaSqFt() As Double
55     Return dblLength * dblWidth
56 End Function
57
58 Public Function GetAreaSqYds() As Double
59     Return (GetAreaSqFt() / 9)
60 End Function
61
62 End Class

```



$3 * 3 = 9$  square feet  
 $9 / 9 = 1$  square yards



$6.6 * 3.9 = 25.74$  square feet  
 $25.74 / 9 = 2.86$  square yards

## Chap10\Exercise

### 12.Sod Solution\_EXERCISE 3\_intermediate

- VS: **Project / Add Existing Item...Ctrl+D** to add existing **Class**
- instantiate an **Object** from **Class Rectangle** using a **Default constructor: Dim x As New Rectangle**,  
**txt\_Enter, txt.TextChanged, txt.KeyPress, e.KeyChar, ControlChars.Back**

3. In this exercise, you create an application that can be used to estimate the cost of laying sod on a rectangular piece of property. Create a Windows Forms application. Use the following names for the project and solution, respectively: Sod Project and Sod Solution. Save the application in the VB2017\Chap10 folder. Use Windows to copy the Rectangle.vb file from the VB2017\Chap10 folder to the Sod Project folder. Then, use the Project menu to add the file to the project. Create the interface shown in Figure 10-40 and then code the application using the Integer data type for the length and width measurements. Display the total price with a dollar sign and two decimal places. Save the solution and then start and test the application.

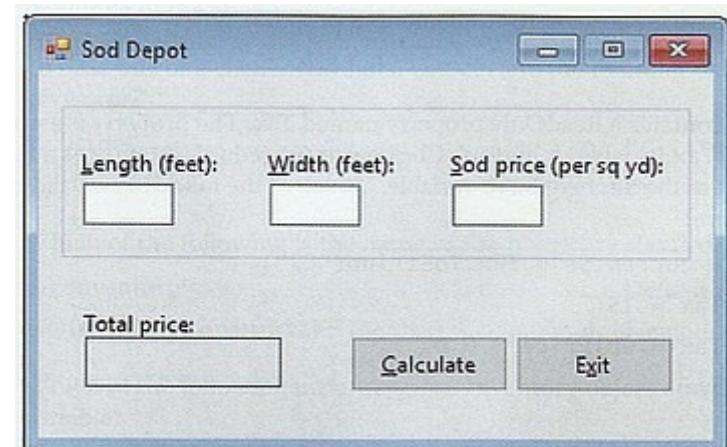


Figure 10-40 Interface for Exercise 3

```
1  ' Name:      Sod Project (grass carpet)
2  ' Purpose:    Estimate the cost of laying sod on a rectangular piece of property
3  ' Programmer: PoakamaNdizi on today
4  ' _1. from VB2017\Chap10 folder copy Rectangle.vb to your Sod Project folder
5  ' _2. in VS: use the Add Existing Item option on the Project menu to add the file to the project
6  ' _3. create an interface shown in Figure 10-40
7  ' _4. code the application, where:
8  '       - use Integer data type for Length & Width measurements
9  '       - display the total price with a dollar sign and 2 decimal places
10 ' test: 3 * 3 = 9 sq ft = 1 sq yd
11 Option Explicit On
12 Option Strict On
13 Option Infer Off
14
15 Public Class frmMain
16     Private Sub btnCalculate_Click(sender As Object, e As EventArgs) Handles btnCalculate.Click
17         Dim Carpet As New Rectangle
18         Dim dblSodPrice As Double
19         Dim dblTotalPrice As Double
20
21         Double.TryParse(txtSodPrice.Text, dblSodPrice)
22         Integer.TryParse(txtLength.Text, Carpet.Length)
23         Integer.TryParse(txtWidth.Text, Carpet.Width)
```

```

24      dblTotalPrice = (((Carpet.GetArea) / 9) * dblSodPrice)
25
26      lblTotalPrice.Text = dblTotalPrice.ToString("C2")
27  End Sub
28
29
30  Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
31      Me.Close()
32  End Sub
33
34  Private Sub txts_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtLength.KeyPress, txtWidth.KeyPress
35      ' input only Integers & Backspace key:
36      If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
37          e.Handled = True
38      End If
39  End Sub
40
41  Private Sub txtSodPrice_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtSodPrice.KeyPress
42      ' input only numbers & decimal point & Backspace key:
43      If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> "." AndAlso e.KeyChar <> ControlChars.Back Then
44          e.Handled = True
45      End If
46  End Sub
47
48  Private Sub txts_TextChanged(sender As Object, e As EventArgs) Handles txtLength.TextChanged, txtWidth.TextChanged,
49   txtSodPrice.TextChanged
50      ' input text changed will clear the label:
51      lblTotalPrice.Text = Nothing
52  End Sub
53 End Class

```

```

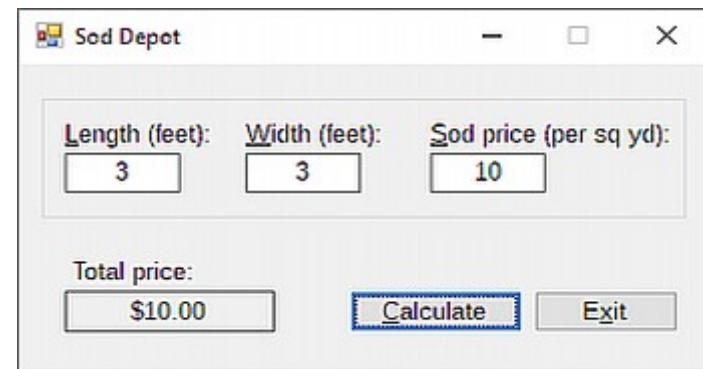
1  ' Name:          Rectangle.vb
2  ' Programmer:    <your name> on <current date>
3
4  Option Explicit On
5  Option Strict On
6  Option Infer Off
7

```

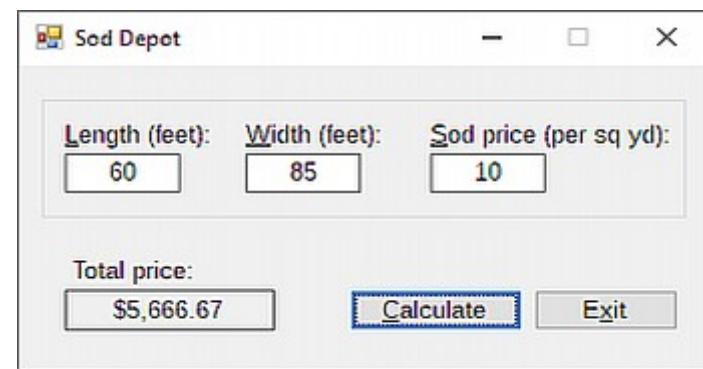
```

8  Public Class Rectangle
9      Private intLength As Integer
10     Private intWidth As Integer
11
12     Public Property Length As Integer
13         Get
14             Return intLength
15         End Get
16         Set(value As Integer)
17             If value > 0 Then
18                 intLength = value
19             Else
20                 intLength = 0
21             End If
22         End Set
23     End Property
24
25     Public Property Width As Integer
26         Get
27             Return intWidth
28         End Get
29         Set(value As Integer)
30             If value > 0 Then
31                 intWidth = value
32             Else
33                 intWidth = 0
34             End If
35         End Set
36     End Property
37
38     Public Sub New()
39         intLength = 0
40         intWidth = 0
41     End Sub
42
43     Public Sub New(ByVal intL As Integer, ByVal intW As Integer)
44         Length = intL
45         Width = intW
46     End Sub
47
48     Public Function GetArea() As Integer
49         Return intLength * intWidth
50     End Function
51 End Class

```



$3 * 3 = 9$  square feet  
 $9 / 9 = 1$  square yard



$60 * 85 = 5100$  square feet  
 $5100 / 9 = 566.6666666666$  square yards

## Chap10\Exercise

### 13.Fence Solution\_EXERCISE 4\_intermediate

- VS: **Project / Add Existing Item...Ctrl+D** to add existing **Class**  
- instantiate an **Object** from **Class Rectangle** using a **Default constructor**: **Dim x As New Rectangle**,  
**txt\_Enter**, **txt\_MouseClick**, **txt.SelectAll()**, **txt.KeyPress**, **e.KeyChar**, **ControlChars.Back**,  
**frmMain\_FormClosing**, **Dim dlgButton As DialogResult**, **dlgButton = MessageBox.Show(...)**,  
**If dlgButton = DialogResult.No Then...**, **txt.TextChanged**

4. In this exercise, you create an application that can be used to calculate the cost of installing a fence around a rectangular area. Create a Windows Forms application. Use the following names for the project and solution, respectively: Fence Project and Fence Solution. Save the application in the VB2017\Chap10 folder.
- Use Windows to copy the Rectangle.vb file from the VB2017\Chap10 folder to the Fence Project folder. Then, use the Project menu to add the file to the project.
  - Modify the Rectangle class to use Double (rather than Integer) variables.
  - Add a method named GetPerimeter to the Rectangle class. The method should calculate and return the perimeter of a rectangle. To calculate the perimeter, the method will need to add together the length and width measurements and then multiply the sum by 2.
  - Create the interface shown in Figure 10-41 and then code the application. Save the solution and then start and test the application.

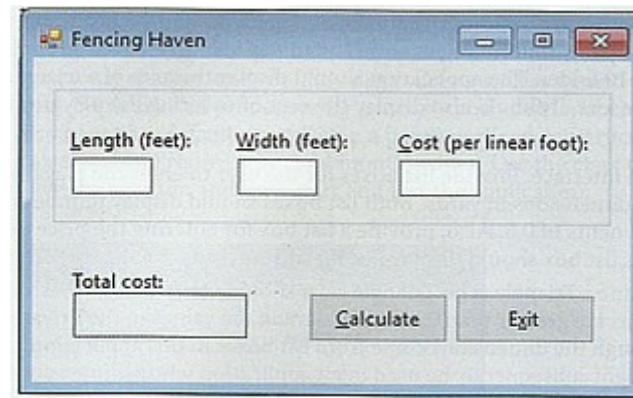


Figure 10-41 Interface for Exercise 4

```
1  ' Name:      Fence Project
2  ' Purpose:    Calculate the cost of installing a fence around a rectangular area
3  ' Programmer: PoaKamaNdizi on today
4  '_1. from VB2017\Chap10 folder copy Rectangle.vb to your Fence Project folder
5  '_2. in VS: use the Add Existing Item option on the Project menu to add the file to the project
6  '_3. create an interface shown in Figure 10-41
7  '_4. modify the Rectangle class to use Double (rather than Integer) variables
8  '_5. add a method named GetPerimeter to the Rectangle class. The method should calculate and return the perimeter of a rectangle
9  '     (to calculate the perimeter, the method will need to add together the length and width measurements
10 '       and then multiply the sum by 2)
11 '_6. code the application
12 Option Explicit On
13 Option Strict On
14 Option Infer Off
15
16 Public Class frmMain
17     Private Sub btnCalculate_Click(sender As Object, e As EventArgs) Handles btnCalculate.Click
18         Dim MyPerimeter As New Rectangle      ' using Default constructor
19         Dim dblCostPerLinearFoot As Double
20         Dim dblTotalCost As Double
21
22         Double.TryParse(txtLength.Text, MyPerimeter.Length)   ' using Default constructor
23         Double.TryParse(txtWidth.Text, MyPerimeter.Width)     ' using Default constructor
24         Double.TryParse(txtCostPerLinearFoot.Text, dblCostPerLinearFoot)
```

```
25
26     ' verify the next step only if there are values to calculate with:
27     If MyPerimeter.Length > 0 AndAlso MyPerimeter.Width > 0 Then
28         dblTotalCost = MyPerimeter.GetPerimeter * dblCostPerLinearFoot
29         lblTotalCost.Text = dblTotalCost.ToString("C2")
30     End If
31 End Sub
32
33 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
34     Me.Close()
35 End Sub
36
37 Private Sub txts_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtLength.KeyPress, txtWidth.KeyPress,
38   txtCostPerLinearFoot.KeyPress
39     ' accept input only: numbers & decimal point & Backspace key:
40     If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> "." AndAlso e.KeyChar <> ControlChars.Back Then
41         e.Handled = True
42     End If
43 End Sub
44
45 Private Sub Txts_TextChanged(sender As Object, e As EventArgs) Handles txtLength.TextChanged, txtWidth.TextChanged,
46   txtCostPerLinearFoot.TextChanged
47     lblTotalCost.Text = Nothing
48 End Sub
49
50 Private Sub txtLength_Enter(sender As Object, e As EventArgs) Handles txtLength.Enter
51     txtLength.SelectAll()
52 End Sub
53
54 Private Sub txtWidth_Enter(sender As Object, e As EventArgs) Handles txtWidth.Enter
55     txtWidth.SelectAll()
56 End Sub
57
58 Private Sub txtCostPerLinearFoot_Enter(sender As Object, e As EventArgs) Handles txtCostPerLinearFoot.Enter
59     txtCostPerLinearFoot.SelectAll()
60 End Sub
61
62 Private Sub txtLength_MouseClick(sender As Object, e As MouseEventArgs) Handles txtLength.MouseClick
63     txtLength.SelectAll()
64 End Sub
65
66 Private Sub txtWidth_MouseClick(sender As Object, e As MouseEventArgs) Handles txtWidth.MouseClick
67     txtWidth.SelectAll()
68 End Sub
```

```

69
70     Private Sub txtCostPerLinearFoot_MouseClick(sender As Object, e As MouseEventArgs) Handles txtCostPerLinearFoot.MouseClick
71         txtCostPerLinearFoot.SelectAll()
72     End Sub
73
74     Private Sub frmMain_FormClosing(sender As Object, e As FormClosingEventArgs) Handles Me.FormClosing
75         ' confirm killing the application:
76         Dim dlgButton As DialogResult
77         dlgButton = MessageBox.Show("Do you want to exit the application?", "Fencing haven",
78                                     MessageBoxButtons.YesNo, MessageBoxIcon.Question)
79         If dlgButton = DialogResult.No Then
80             e.Cancel = True
81         End If
82     End Sub
83 End Class

```

```

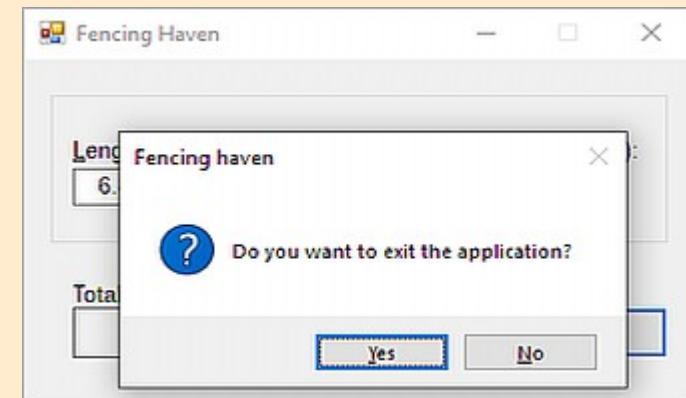
1  ' Name:          Rectangle.vb
2  ' Programmer:    PoaKamaNdizi on today
3  ' _4. modify the Rectangle class to use Double (rather than Integer) variables
4  ' _5. add a method named GetPerimeter to the Rectangle class. The method should calculate and return the perimeter of a rectangle
5  '       (to calculate the perimeter, the method will need to add together the length and width measurements
6  '           and then multiply the sum by 2)
7  Option Explicit On
8  Option Strict On
9  Option Infer Off
10
11 Public Class Rectangle
12     Private dblLength As Double
13     Private dblWidth As Double
14
15     Public Property Length As Double
16         Get
17             Return dblLength
18         End Get
19         Set(value As Double)
20             If value > 0 Then
21                 dblLength = value
22             Else
23                 dblLength = 0
24             End If
25         End Set
26     End Property
27

```

```

28     Public Property Width As Double
29         Get
30             Return dblWidth
31         End Get
32         Set(value As Double)
33             If value > 0 Then
34                 dblWidth = value
35             Else
36                 dblWidth = 0
37             End If
38         End Set
39     End Property
40
41     Public Sub New()
42         dblLength = 0
43         dblWidth = 0
44     End Sub
45
46     Public Sub New(ByVal dblL As Double, ByVal dblW As Double)
47         Length = dblL
48         Width = dblW
49     End Sub
50
51     Public Function GetArea() As Double
52         Return dblLength * dblWidth
53     End Function
54
55     Public Function GetPerimeter() As Double
56         Return (dblLength + dblWidth) * 2
57     End Function
58 End Class

```



**Fencing Haven**

|                |               |                         |
|----------------|---------------|-------------------------|
| Length (feet): | Width (feet): | Cost (per linear foot): |
| 10             | 10            | 10                      |

Total cost:

|          |
|----------|
| \$400.00 |
|----------|

**Calculate**   **Exit**

**Fencing Haven**

|                |               |                         |
|----------------|---------------|-------------------------|
| Length (feet): | Width (feet): | Cost (per linear foot): |
| 6.85           | 9.35          | 6                       |

Total cost:

|          |
|----------|
| \$194.40 |
|----------|

**Calculate**   **Exit**

## Chap10\\_Exercise

### 14.Grade Solution-Intermediate\_EXERCISE 5\_intermediate

- instantiate an Object from **Class CourseGrade** using a **Default constructor**:

```
Dim x As New CourseGrade, lst.Items.Add, lst.SelectedItem, txt_Enter,  
lst.SelectedValueChanged, txt.TextChanged, txt_MouseClick, txt.SelectAll(),  
txt.KeyPress, e.KeyChar, ControlChars.Back, MessageBox.Show, frmMain_Load
```

In this exercise, you modify the Grade Calculator application from this chapter's Apply lesson. Use Windows to make a copy of the Grade Solution folder. Rename the copy Grade Solution-Intermediate. Open the Grade Solution.sln file contained in the Grade Solution-Intermediate folder.

- a. Open the CourseGrade.vb file. The DetermineGrade method should accept an integer that represents the total number of points that can be earned in the course. (Currently, the total number of points is 200: 100 points per test.) For an A grade, the student must earn at least 90% of the total points. For a B, C, and D grade, the student must earn at least 80%, 70%, and 60%, respectively. If the student earns less than 60% of the total points, the grade is F. Make the appropriate modifications to the DetermineGrade method and then save the solution.
- b. Unlock the controls on the form. Add a label control and a text box to the form. Change the label control's Text property to "&Maximum points:" (without the quotation marks). Change the text box's name to txtMax. Lock the controls and then reset the tab order.

- c. Open the form's Code Editor window. The txtMax control should accept only numbers and the Backspace key. Code the appropriate procedure.
- d. The grade should be cleared when the user makes a change to the contents of the txtMax control. Code the appropriate procedure.
- e. Modify the frmMain\_Load procedure so that each list box displays numbers from 0 through 200.
- f. Locate the btnDisplay\_Click procedure. If the txtMax control does not contain a value, display an appropriate message. The maximum number allowed in the txtMax control should be 400; if the control contains a number that is more than 400, display an appropriate message. The statement that calculates the grade should pass the maximum number of points to the studentGrade object's DetermineGrade method. Make the necessary modifications to the procedure.
- g. Save the solution and then start and test the application.

```
1  ' Name:      Grade Project
2  ' Purpose:    Displays a grade based on two test scores.
3  ' Programmer: vincenttheclown on <current date>
4  ' 1. in CourseGrade.vb: modify the DetermineGrade method:
5  '   - it should accept an integer that represents the total number of points that can be earned in the course
6  '   - (currently, the ttl number of points is 200. That means 100 points per test)
7  '   - students must earn at least of the total points: 90% for an A grade, 80% for B, 70% for C, 60% for D, less than 60% F
8  ' 2. in Designer window: add to the form and reset the tab order:
9  '   - a label control with a Text property: &Maximum points:
10 '   - a text box named txtMax
11 ' in Code editor window:
12 ' 3. txtMax control should accept only numbers and the Backspace key
13 ' 4. the grade should be cleared when the user makes a change to the contents of the txtMax control
14 ' 5. modify the frmMain_Load procedure so that each list box displays numbers from 0 through 200
15 ' 6. in btnDisplay_Click procedure: make the necessary modifications:
16 '   6.1. if the txtMax control does not contain a value, display an appropriate message
17 '   6.2. the max number allowed in the txtMax control should be 400 -> if the control contains a number that is more than 400,
18 '       display an appropriate message
19 '   6.3. the statement that calculates the grade should pass the maximum number of points
20 '       to the studentGrade object's DetermineGrade method
21 Option Explicit On
22 Option Strict On
23 Option Infer On
```

```

24
25  Public Class frmMain
26      Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
27          ' Calculates and displays a letter grade.
28          Dim intMax As Integer
29          Dim studentGrade As New CourseGrade      ' instantiate a CourseGrade Object and name it as studentGrade.
30          Integer.TryParse(txtMax.Text, intMax)
31
32          If intMax > 0 AndAlso intMax < 401 Then    ' Maximum points can be only between 1 through 400
33
34              ' Assign 2 test scores to Object's 2 Properties:
35              Integer.TryParse(lstTest1.SelectedItem.ToString, studentGrade.Score1)      ' Property Score1
36              Integer.TryParse(lstTest2.SelectedItem.ToString, studentGrade.Score2)      ' Property Score2
37
38              ' my fix: 2 scores added together must be equal to or smaller then Max points, right?:
39              If studentGrade.Score1 + studentGrade.Score2 <= intMax Then
40                  ' Calculate grade invoking/using/calling Object's DetermineGrade method, passing it an input from the user:
41                  studentGrade.DetermineGrade(intMax)
42
43                  ' Display grade stored in Object's ReadOnly Grade Property:
44                  lblGrade.Text = studentGrade.Grade      ' refers to the Object's ReadOnly Grade Property
45
46              Else    ' if 2 scores added together are more then Maximum points:
47                  MessageBox.Show("Max points are less then your scores, please correct.", "Grade Calculator",
48                                  MessageBoxButtons.OK, MessageBoxIcon.Exclamation)
49              End If
50              Else    ' if Maximum points are 0 or more then 400:
51                  MessageBox.Show("Please enter a valid value between 1 and 400.", "Grade Calculator",
52                                  MessageBoxButtons.OK, MessageBoxIcon.Exclamation)
53          End If
54      End Sub
55
56      Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
57          ' Fills the list boxes with values and selects default values.
58          For intScore As Integer = 0 To 200
59              lstTest1.Items.Add(intScore.ToString)
60              lstTest2.Items.Add(intScore.ToString)
61          Next intScore
62
63          lstTest1.SelectedItem = "100"
64          lstTest2.SelectedItem = "100"
65      End Sub
66

```

```

67      Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
68          Me.Close()
69      End Sub
70
71      Private Sub ClearGrade(sender As Object, e As EventArgs) Handles lstTest1.SelectedValueChanged,
72                               lstTest2.SelectedValueChanged, txtMax.TextChanged
73          lblGrade.Text = String.Empty
74      End Sub
75
76      Private Sub txtMax_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtMax.KeyPress
77          If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
78              e.Handled = True
79          End If
80      End Sub
81
82      Private Sub txtMax_MouseClick(sender As Object, e As MouseEventArgs) Handles txtMax.MouseClick
83          txtMax.SelectAll()
84      End Sub
85
86      Private Sub txtMax_Enter(sender As Object, e As EventArgs) Handles txtMax.Enter
87          txtMax.SelectAll()
88      End Sub
89  End Class

```

```

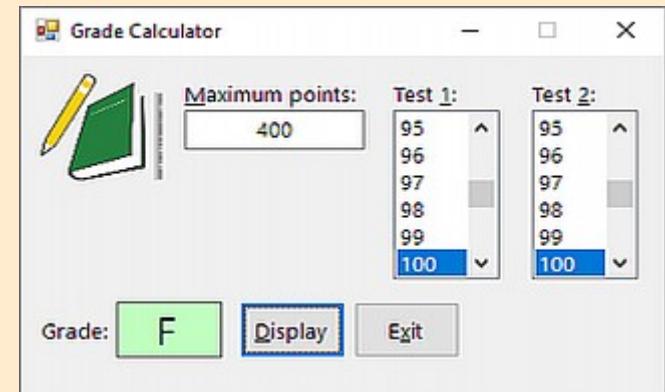
1  ' Name:          CourseGrade.vb
2  ' Programmer:    vincenttheclown on <current date>
3  Option Explicit On
4  Option Strict On
5  Option Infer Off
6
7  Public Class CourseGrade
8      Private intScore1 As Integer
9      Private intScore2 As Integer
10     Private strGrade As String      ' member variable for the letter grade
11
12    Public Property Score1 As Integer
13        Get
14            Return intScore1
15        End Get
16        Set(value As Integer)
17            intScore1 = value
18        End Set
19    End Property
20

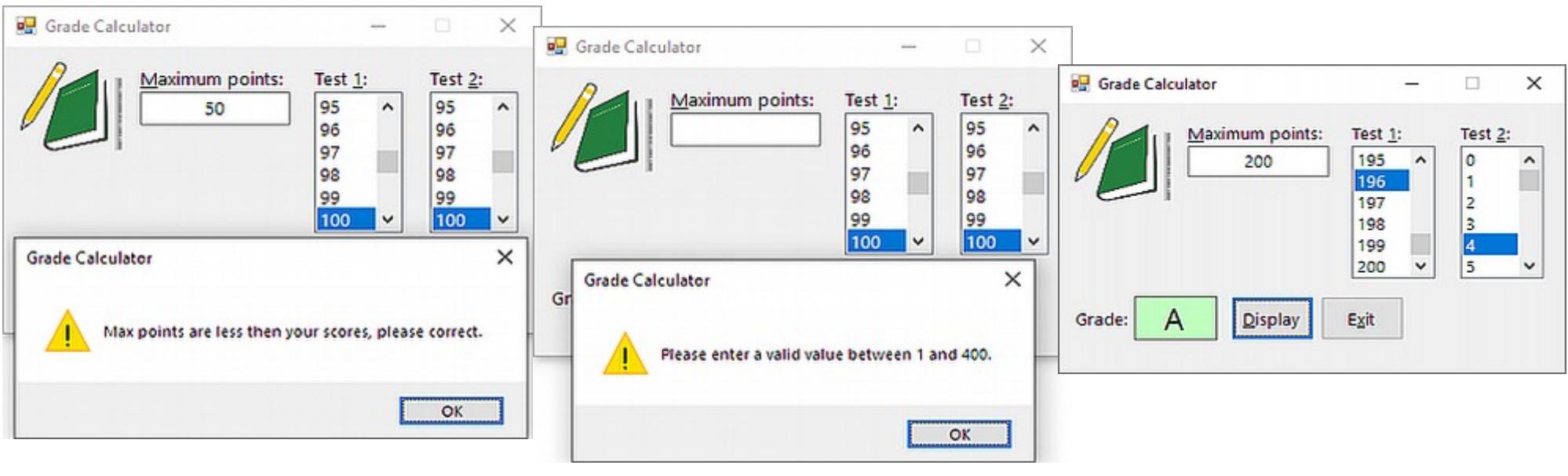
```

```

21     Public Property Score2 As Integer
22         Get
23             Return intScore2
24         End Get
25         Set(value As Integer)
26             intScore2 = value
27         End Set
28     End Property
29
30     Public ReadOnly Property Grade As String      ' Public Property for the strGrade member variable
31         Get
32             Return strGrade
33         End Get
34     End Property
35
36     Public Sub New()      ' Default Constructor will initialize member variables when a CourseGrade Object is instantiated
37         intScore1 = 0
38         intScore2 = 0
39         strGrade = String.Empty
40     End Sub
41
42     ' 1. in CourseGrade.vb: modify the DetermineGrade method:
43     ' - it should accept an integer that represents the total number of points that can be earned in the course
44     ' - (currently, the ttl number of points is 200. That means 100 points per test)
45     ' - students must earn at least of the total points: 90% for an A grade, 80% for B, 70% for C, 60% for D, less than 60% F
46     ' intMax = input from txtMax = maximum points
47     Public Sub DetermineGrade(ByVal intMax As Integer)    ' method will assign the appropriate letter grade
48         'MessageBox.Show(intMax.ToString)    '_ok
49         'MessageBox.Show((intScore1 + intScore2).ToString)    '_ok
50         'MessageBox.Show(((intScore1 + intScore2) / intMax) * 100).ToString)    '_ok
51         Select Case ((intScore1 + intScore2) / intMax) * 100
52             Case Is >= 90
53                 strGrade = "A"
54             Case Is >= 80
55                 strGrade = "B"
56             Case Is >= 70
57                 strGrade = "C"
58             Case Is >= 60
59                 strGrade = "D"
60             Case Else
61                 strGrade = "F"
62         End Select
63     End Sub
64 End Class

```





## Chap10\Exercise1

### 15.Playground Solution\_EXERCISE 6\_advanced

- instantiate an **Object** from **Class Triangle** using a **Default constructor**: **Dim x As New Triangle**,  
**For...To...Step...Next**, **lst.Items.Add**, **lst.SelectedIndex**, **lst.SelectedIndexChanged**,  
**lbl.BackColor = SystemColors.Control**, **lbl.BackColor= Color.Yellow**, **frmMain\_Load**

Create a Windows Forms application. Use the following names for the project and solution, respectively: Playground Project and Playground Solution. Save the application in the VB2017\Chap10 folder. The application should display the area of a triangular playground in square feet. It should also display the cost of covering the playground with artificial grass.

- Create a suitable interface. Provide list boxes for the user to enter the playground's base and height dimensions in yards. Both list boxes should display numbers from 20 to 50 in increments of 0.5. Also, provide a list box for entering the price per square foot. This list box should display numbers from 1 to 6 in increments of 0.5.

- Create a class named **Triangle**. The **Triangle** class should verify that the base and height dimensions are greater than 0 before assigning the values to the **Private** variables. (Although the dimensions come from list boxes in this application, the **Triangle** class might subsequently be used in an application whose dimensions come from text boxes. Therefore, it is a good idea to verify the user's input.) The class should also include a default constructor, a parameterized constructor, and a method to calculate and return the area of a triangle.
- Code the application. Save the solution and then start and test the application.

```

1  ' Name:      Playground Project.
2  ' Purpose:   Displays the area in square feet and cost of laying artificial grass for a triangular playground.
3  ' Programmer: PoaKamaNdizi on time stamp.
4
5  '_1. GUI: - 2x input lst filled with numbers 20-50 in increments of 0.5: - for "base" dimension in yards
6  '_
7  '_
8  '     - 1x input lst filled with numbers 1-6 in increments of 0.5: - for price per square foot
9  ' 1 yard = ??? feet, 1 square foot = ??? square yards
10
11 '_2. create a Class named "Triangle", where the class should:
12 '_2.1. verify that the base and height dimensions are greater than 0, before assigning the values to the Private variables
13 '_     (although the dimensions come from list boxes in this application, the Triangle class might subsequently be used in
14 '_         an application whose dimensions come from text boxes - therefore, it is a good idea to verify the user's input)
15 '_2.2. include a default constructor
16 '_2.3. include a parameterized constructor
17 '_2.4. include a method to calculate and return the area of a triangle
18 ' test
19 Option Explicit On
20 Option Strict On
21 Option Infer Off
22
23 Public Class frmMain
24     Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
25         'lst1Base & lst2Height: 20->50 step 0.5;  lst3Price -> 1-6 step 0.5
26         For dblCounter1 As Double = 20 To 50 Step 0.5
27             lst1Base.Items.Add(" " & dblCounter1)
28             lst2Height.Items.Add(" " & dblCounter1)
29             Next dblCounter1
30
31         For dblCounter2 As Double = 1 To 6 Step 0.5
32             lst3Price.Items.Add(" " & dblCounter2)
33             Next dblCounter2
34
35         lst1Base.SelectedIndex = 0 : lst2Height.SelectedIndex = 0 : lst3Price.SelectedIndex = 0
36     End Sub
37
38     Private Sub lst_MyCleaner(sender As Object, e As EventArgs) Handles lst1Base.SelectedIndexChanged,
39         lst2Height.SelectedIndexChanged, lst3Price.SelectedIndexChanged
40         lblArea.Text = Nothing
41         lblTotalPrice.Text = Nothing
42         lblArea.BackColor = SystemColors.Control
43         lblTotalPrice.BackColor = SystemColors.Control
44     End Sub

```

```

45      Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
46          Me.Close()
47      End Sub
48
49      Private Sub btnCalculate_Click(sender As Object, e As EventArgs) Handles btnCalculate.Click
50          lblArea.BackColor = Color.Yellow
51          lblTotalPrice.BackColor = Color.Yellow
52
53          Dim MyTriangleArea As New Triangle
54          Dim dblPrice As Double
55
56          Double.TryParse(lst1Base.SelectedItem.ToString, MyTriangleArea.Base)
57          Double.TryParse(lst2Height.SelectedItem.ToString, MyTriangleArea.Height)
58          Double.TryParse(lst3Price.SelectedItem.ToString, dblPrice)
59
60          ' input in square yards, output should be in square foot: 1 square yard = 9 square feet
61          lblArea.Text = (MyTriangleArea.GetTriangleArea * 9).ToString("N2")
62          lblTotalPrice.Text = ((MyTriangleArea.GetTriangleArea * 9) * dblPrice).ToString("C2")
63      End Sub
64  End Class

```

```

1  ' Name:      Triangle.vb.
2  ' Programmer: PoaKamaNdizi on time stamp.
3
4  '_2. create a Class named "Triangle", where the class should:
5  '_2.1. verify that the base and height dimensions are greater than 0, before assigning the values to the Private variables
6  '_       (although the dimensions come from list boxes in this application, the Triangle class might subsequently be used in
7  '_       an application whose dimensions come from text boxes - therefore, it is a good idea to verify the user's input)
8  '_2.2. include a default constructor
9  '_2.3. include a parameterized constructor
10 '_2.4. include a method to calculate and return the area of a triangle
11 Option Explicit On
12 Option Strict On
13 Option Infer Off
14
15 Public Class Triangle
16     Private dblBase As Double
17     Private dblHeight As Double
18

```

```
19     Public Property Base As Double
20         Get
21             Return dblBase
22         End Get
23         Set(value As Double)
24             If value > 0 Then
25                 dblBase = value
26             Else
27                 dblBase = 0
28             End If
29         End Set
30     End Property
31
32     Public Property Height As Double
33         Get
34             Return dblHeight
35         End Get
36         Set(value As Double)
37             If value > 0 Then
38                 dblHeight = value
39             Else
40                 dblHeight = 0
41             End If
42         End Set
43     End Property
44
45     Public Sub New()
46         ' default Constructor:
47         dblBase = 0
48         dblHeight = 0
49     End Sub
50
51     Public Sub New(ByVal dblB As Double, ByVal dblH As Double)
52         ' parameterized Constructor:
53         Base = dblB
54         Height = dblH
55     End Sub
56
57     Public Function GetTriangleArea() As Double
58         ' method/function to calculate and return the area of a triangle
59         Return (dblBase / 2) * dblHeight
60     End Function
61 End Class
```

**Triangular playground**

Please choose your values here:

|                                                      |                                                      |                                              |
|------------------------------------------------------|------------------------------------------------------|----------------------------------------------|
| Base dimension<br>in yards:                          | Height dimension<br>in yards:                        | Price per square foot<br>in dollars:         |
| 20<br>20.5<br>21<br>21.5<br>22<br>22.5<br>23<br>23.5 | 20<br>20.5<br>21<br>21.5<br>22<br>22.5<br>23<br>23.5 | 1<br>1.5<br>2<br>2.5<br>3<br>3.5<br>4<br>4.5 |

Area in square feet:  Calculate  
Price:  Exit

**Triangular playground**

Please choose your values here:

|                                                      |                                                      |                                              |
|------------------------------------------------------|------------------------------------------------------|----------------------------------------------|
| Base dimension<br>in yards:                          | Height dimension<br>in yards:                        | Price per square foot<br>in dollars:         |
| 20<br>20.5<br>21<br>21.5<br>22<br>22.5<br>23<br>23.5 | 20<br>20.5<br>21<br>21.5<br>22<br>22.5<br>23<br>23.5 | 1<br>1.5<br>2<br>2.5<br>3<br>3.5<br>4<br>4.5 |

Area in square feet: **1,800.00** Calculate  
Price: **\$1,800.00** Exit

**Triangular playground**

Please choose your values here:

|                                                             |                                                             |                                                     |
|-------------------------------------------------------------|-------------------------------------------------------------|-----------------------------------------------------|
| Base dimension<br>in yards:                                 | Height dimension<br>in yards:                               | Price per square foot<br>in dollars:                |
| 39<br>39.5<br>40<br>40.5<br>41<br><b>41.5</b><br>42<br>42.5 | 34.5<br>35<br>35.5<br>36<br><b>36.5</b><br>37<br>37.5<br>38 | 2.5<br>3<br>3.5<br>4<br><b>4.5</b><br>5<br>5.5<br>6 |

Area in square feet: **6,816.38** Calculate  
Price: **\$30,673.69** Exit

triangle area formula:  $(0.5 * b) * h = (b * h) * 0.5$

$$20 / 2 = 10$$

$10 * 20 = 200$  square yards

1 square yard = 9 square feet

$200 * 9 = 1800$  square feet

triangle area formula:  $(0.5 * b) * h = (b * h) * 0.5$

$$41.5 / 2 = 20.75$$

$20.75 * 36.5 = 757.375$  square yards

1 square yard = 9 square feet

$757.375 * 9 = 6816.38$  square feet

$6816.38 * 4.5 = 30673.69$

## Chap10\Exercise1

### 16.Fire Solution\_EXERCISE 7\_advanced

- instantiate an Object from Class WaterTank: `Dim MyWaterTank As WaterTank, ...`
- invoke a Parameterized constructor: `MyWaterTank = New WaterTank(inputs/parametres)`
- `lbl.BackColor = Color.Yellow, lbl.BackColor = SystemColors.Control, txt_KeyPress, e.KeyChar, ControlChars.Back, txt_Enter, txt_MouseClick, txt_TextChanged, txt.SelectAll()`

Create a Windows Forms application. Use the following names for the project and solution, respectively: Fire Project and Fire Solution. Save the application in the VB2017\Chap10 folder. The application should display the capacity (volume) of a water tank on a fire engine in both cubic feet and gallons, given the tank's length, width, and height measurements. Create a suitable interface. Code the application by using a class to instantiate a water tank object. Display the output with one decimal place. Save the solution and then start and test the application. (Hint: There are 7.48 gallons in one cubic foot.)

```

1  ' Name:      Water tank volume calculator.
2  ' Purpose:   Display capacity/volume of a water tank.
3  ' Programmer: PoaKamaNdizi on Time Stamp.
4  ' display the capacity (volume) of a water tank in - cubic feet & gallons
5  ' (1 cubic foot = 7.48 gallons)
6  ' input: tank's length, width, and height measurements
7  ' create a suitable interface
8  ' code the application by using a Class to instantiate a water tank Object
9  ' display the output with 1 decimal place
10 ' test
11 Option Explicit On
12 Option Strict On
13 Option Infer Off
14
15 Public Class frmMain
16     Private Sub btnCalculate_Click(sender As Object, e As EventArgs) Handles btnCalculate.Click
17         lbl1CubicFeet.BackColor = Color.Yellow : lbl2Gallons.BackColor = Color.Yellow
18         Dim dbl1Length As Double : Dim dbl2Width As Double : Dim dbl3Height As Double
19
20         ' im gonna use Parameterized constructor, who will verify the input for me:
21         Dim MyWaterTank As WaterTank
22
23         Double.TryParse(txt1Length.Text, dbl1Length)
24         Double.TryParse(txt2Width.Text, dbl2Width)
25         Double.TryParse(txt3Height.Text, dbl3Height)
26
27         ' invoke Parameterized Constructor and flush him inputs to verify:
28         MyWaterTank = New WaterTank(dbl1Length, dbl2Width, dbl3Height)
29
30         lbl1CubicFeet.Text = MyWaterTank.GetCapacity.ToString("N1")
31         lbl2Gallons.Text = MyWaterTank.GetVolume.ToString("N1")
32     End Sub
33
34     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
35         Me.Close()
36     End Sub
37
38     Private Sub txts_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txt1Length.KeyPress,
39                               txt2Width.KeyPress, txt3Height.KeyPress
40         ' accept only numbers, decimal point, and BackSpace key:
41         If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back AndAlso e.KeyChar <> "." Then
42             e.Handled = True
43         End If
44     End Sub

```

```

45
46     Private Sub txt1Length_TextChanged(sender As Object, e As EventArgs) Handles txt1Length.TextChanged,
47   txt2Width.TextChanged, txt3Height.TextChanged
48         lbl1CubicFeet.Text = Nothing : lbl1CubicFeet.BackColor = SystemColors.Control
49         lbl2Gallons.Text = Nothing : lbl2Gallons.BackColor = SystemColors.Control
50     End Sub
51
52     Private Sub txt1Length_Enter(sender As Object, e As EventArgs) Handles txt1Length.Enter
53         txt1Length.SelectAll()
54     End Sub
55
56     Private Sub txt2Width_Enter(sender As Object, e As EventArgs) Handles txt2Width.Enter
57         txt2Width.SelectAll()
58     End Sub
59
60     Private Sub txt3Height_Enter(sender As Object, e As EventArgs) Handles txt3Height.Enter
61         txt3Height.SelectAll()
62     End Sub
63
64     Private Sub txt1Length_MouseClick(sender As Object, e As MouseEventArgs) Handles txt1Length.MouseClick
65         txt1Length.SelectAll()
66     End Sub
67
68     Private Sub txt2Width_MouseClick(sender As Object, e As MouseEventArgs) Handles txt2Width.MouseClick
69         txt2Width.SelectAll()
70     End Sub
71
72     Private Sub txt3Height_MouseClick(sender As Object, e As MouseEventArgs) Handles txt3Height.MouseClick
73         txt3Height.SelectAll()
74     End Sub
75 End Class

```

```

1  ' Name:          WaterTank.vb
2  ' Programmer:    PoaKamaNdizi on Time Stamp.
3  Option Explicit On
4  Option Strict On
5  Option Infer Off
6
7  Public Class WaterTank
8      Private dbl1Length As Double
9      Private dbl2Width As Double
10     Private dbl3Height As Double

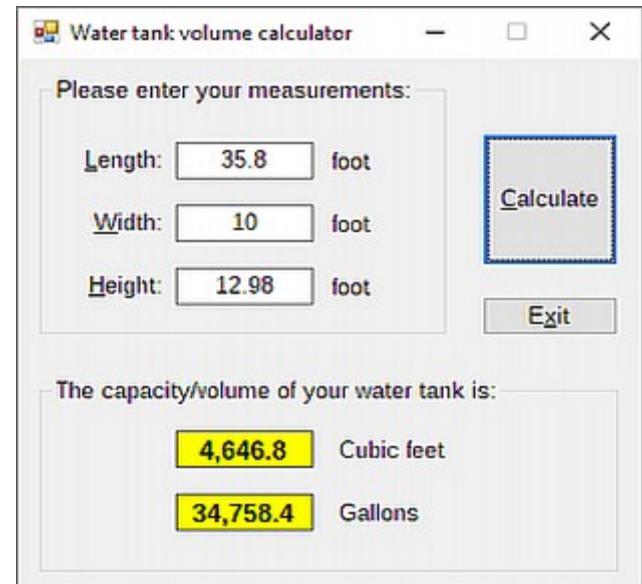
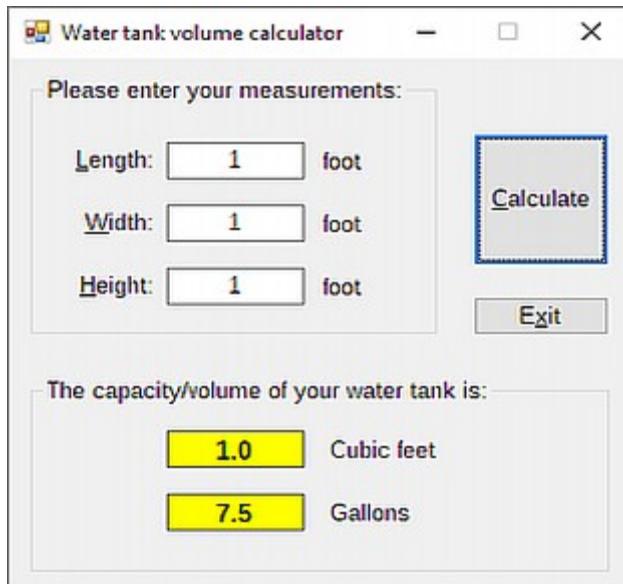
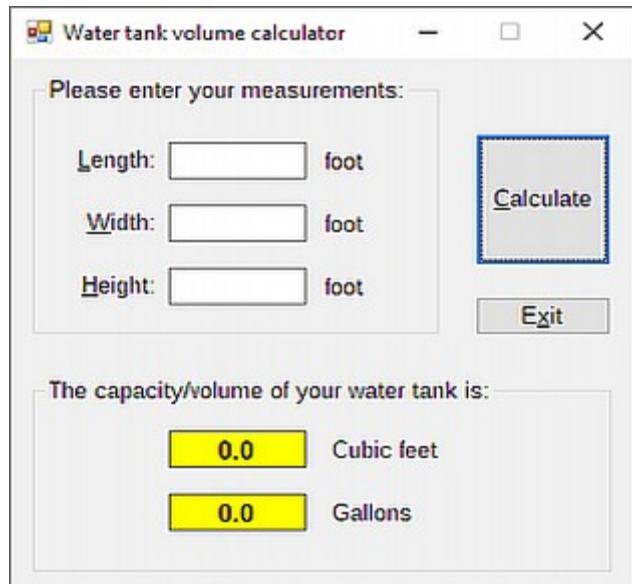
```

```
11
12     Public Property Length As Double
13         Get
14             Return dbl1Length
15         End Get
16         Set(value As Double)
17             If value > 0 Then
18                 dbl1Length = value
19             Else
20                 dbl1Length = 0
21             End If
22         End Set
23     End Property
24
25     Public Property Width As Double
26         Get
27             Return dbl2Width
28         End Get
29         Set(value As Double)
30             If value > 0 Then
31                 dbl2Width = value
32             Else
33                 dbl2Width = 0
34             End If
35         End Set
36     End Property
37
38     Public Property Height As Double
39         Get
40             Return dbl3Height
41         End Get
42         Set(value As Double)
43             If value > 0 Then
44                 dbl3Height = value
45             Else
46                 dbl3Height = 0
47             End If
48         End Set
49     End Property
50
51     Public Sub New()
52         ' default constructor:
53         dbl1Length = 0 : dbl2Width = 0 : dbl3Height = 0
54     End Sub
```

```

55
56     Public Sub New(ByVal dblL As Double, ByVal dblW As Double, ByVal dblH As Double)
57         ' parameterized constructor:
58         Length = dblL : Width = dblW : Height = dblH
59     End Sub
60
61     Public Function GetCapacity() As Double
62         ' function to calculate in Cubic feet:
63         Return dbl1Length * dbl2Width * dbl3Height
64     End Function
65
66     Public Function GetVolume() As Double
67         ' function to calculate in Gallons:
68         ' (1 cubic foot = 7.48 gallons)
69         Return GetCapacity() * 7.48
70     End Function
71 End Class

```



rectangular tank capacity:  $a * b * c$   
 $35.8 * 10 * 12.98 = 4646.84$  cubic feet  
 1 cubic foot = 7.48 gallons  
 $4646.84 * 7.48 = 34758.3632$

- instantiate an **Object** from **Class** `ParkingLot: Dim MyParkingLot As ParkingLot, ...`,
- invoke a **Parameterized constructor**: `MyParkingLot = New ParkingLot(inputs/parametres)`
- `lbl.BackColor = Color.Yellow, lbl.BackColor = SystemColors.Control, txt_KeyPress, e.KeyChar, ControlChars.Back, txt_Enter, txt_MouseClick, txt_TextChanged, txt.SelectAll()`

8. Create a Windows Forms application. Use the following names for the project and solution, respectively: Parking Project and Parking Solution. Save the application in the VB2017\Chap10 folder. The application should display the total cost of paving the parking lot illustrated in Figure 10-42. (The parking lot's shape is a parallelogram.) Create a suitable interface. The user will enter the dimensions in feet and the cost per square yard. Code the application by using a class to instantiate a parking lot object. Save the solution and then start and test the application.

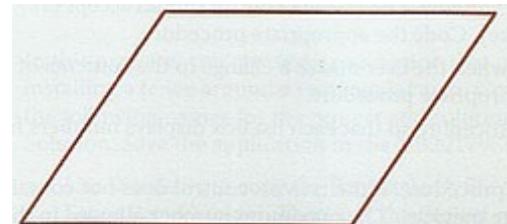


Figure 10-42 Parking lot illustration for Exercise 8

```

1  ' Name:      Parking lot (parallelogram shape) pavement price calculator.
2  ' Purpose:   Calculate the cost of paving the parking lot in a shape of parallelogram.
3  ' Programmer: PoaKamaNdizi on Time Stamp.
4  ' the application should display the total cost of paving the parking lot in a shape of parallelogram
5  ' create a suitable interface
6  ' the user will enter the: - dimensions in feet
7  '                           - cost per square yard
8  ' area of parallelogram (rovnoběžník) = base * height
9  ' 1 feet = 0.33333 yards; 1 yard = 3 feet
10 ' 1 square yard = 9 square feet; 1 square feet = 0.1111111 square yards (1/9)
11 ' code the application by using a Class to instantiate a parking lot Object
12 ' test
13 Option Explicit On
14 Option Strict On
15 Option Infer Off
16
17 Public Class frmMain
18     Private Sub btnCalculate_Click(sender As Object, e As EventArgs) Handles btnCalculate.Click
19         lbl1TotalArea_SquareFeet.BackColor = Color.Yellow
20         lbl2TotalCost.BackColor = Color.Yellow
21
22         Dim dblBase As Double : Dim dblHeight As Double : Dim dblCost As Double
23         Dim MyParkingLot As ParkingLot
24
25         Double.TryParse(txt1Base_Feet.Text, dblBase)
26         Double.TryParse(txt2Height_Feet.Text, dblHeight)
27         Double.TryParse(txt3Price_SquareYard.Text, dblCost)
28
29         ' invoke parameterized constructor and give him input values:
30         MyParkingLot = New ParkingLot(dblBase, dblHeight, dblCost)

```

```
31      lbl1TotalArea_SquareFeet.Text = MyParkingLot.GetArea.ToString("N2")
32      lbl2TotalCost.Text = MyParkingLot.GetCost.ToString("C2")
33  End Sub
34
35
36  Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
37      Me.Close()
38  End Sub
39
40  Private Sub txtInputs_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txt1Base_Feet.KeyPress,
41                               txt2Height_Feet.KeyPress, txt3Price_SquareYard.KeyPress
42      If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> "." AndAlso e.KeyChar <> ControlChars.Back Then
43          e.Handled = True
44      End If
45  End Sub
46
47  Private Sub txtInputs_TextChanged(sender As Object, e As EventArgs) Handles txt1Base_Feet.TextChanged,
48                               txt2Height_Feet.TextChanged, txt3Price_SquareYard.TextChanged
49      lbl1TotalArea_SquareFeet.Text = Nothing : lbl2TotalCost.Text = Nothing
50      lbl1TotalArea_SquareFeet.BackColor = SystemColors.Control
51      lbl2TotalCost.BackColor = SystemColors.Control
52  End Sub
53
54  Private Sub txt1Base_Feet_Enter(sender As Object, e As EventArgs) Handles txt1Base_Feet.Enter
55      txt1Base_Feet.SelectAll()
56  End Sub
57
58  Private Sub txt2Height_Feet_Enter(sender As Object, e As EventArgs) Handles txt2Height_Feet.Enter
59      txt2Height_Feet.SelectAll()
60  End Sub
61
62  Private Sub txt3Price_SquareYard_Enter(sender As Object, e As EventArgs) Handles txt3Price_SquareYard.Enter
63      txt3Price_SquareYard.SelectAll()
64  End Sub
65
66  Private Sub txt1Base_Feet_MouseClick(sender As Object, e As MouseEventArgs) Handles txt1Base_Feet.MouseClick
67      txt1Base_Feet.SelectAll()
68  End Sub
69
70  Private Sub txt2Height_Feet_MouseClick(sender As Object, e As MouseEventArgs) Handles txt2Height_Feet.MouseClick
71      txt2Height_Feet.SelectAll()
72  End Sub
73
```

```

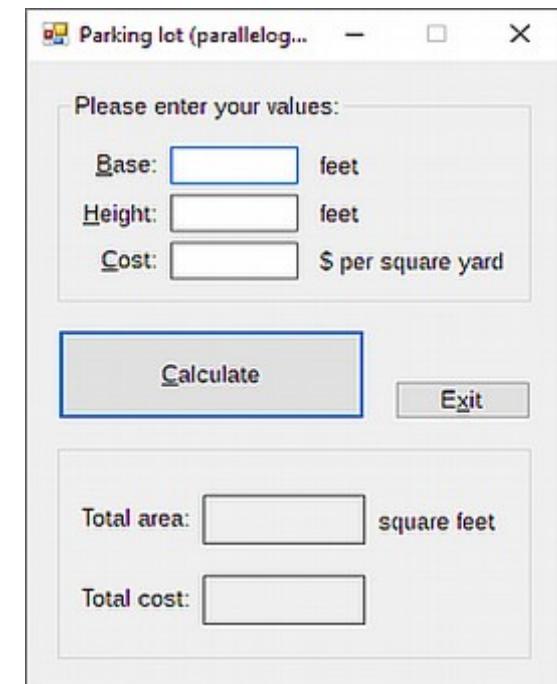
74      Private Sub txt3Price_SquareYard_MouseClick(sender As Object, e As MouseEventArgs) Handles txt3Price_SquareYard.MouseClick
75          txt3Price_SquareYard.SelectAll()
76      End Sub
77  End Class

```

```

1  ' Name:          ParkingLot.vb
2  ' Programmer:    PoakamaNdizi on Time Stamp.
3  Option Explicit On
4  Option Strict On
5  Option Infer Off
6
7  Public Class ParkingLot
8      Private dblBase As Double
9      Private dblHeight As Double
10     Private dblCost As Double
11
12     Public Property Base As Double
13         Get
14             Return dblBase
15         End Get
16         Set(value As Double)
17             If value > 0 Then
18                 dblBase = value
19             Else
20                 dblBase = 0
21             End If
22         End Set
23     End Property
24
25     Public Property Height As Double
26         Get
27             Return dblHeight
28         End Get
29         Set(value As Double)
30             If value > 0 Then
31                 dblHeight = value
32             Else
33                 dblHeight = 0
34             End If
35         End Set
36     End Property
37

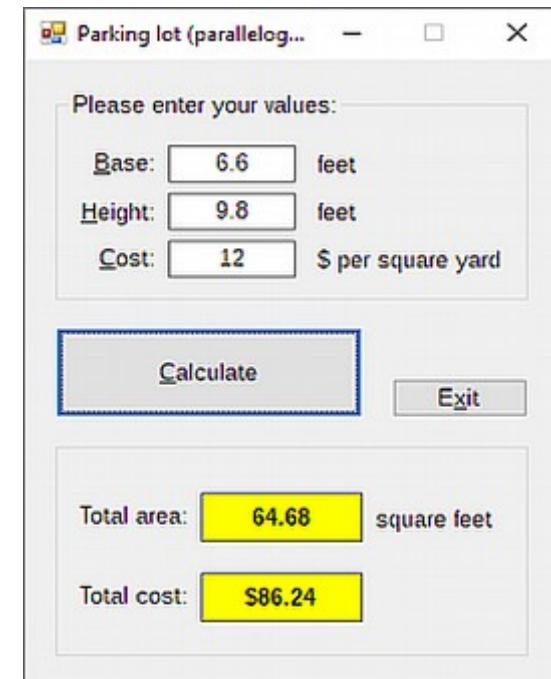
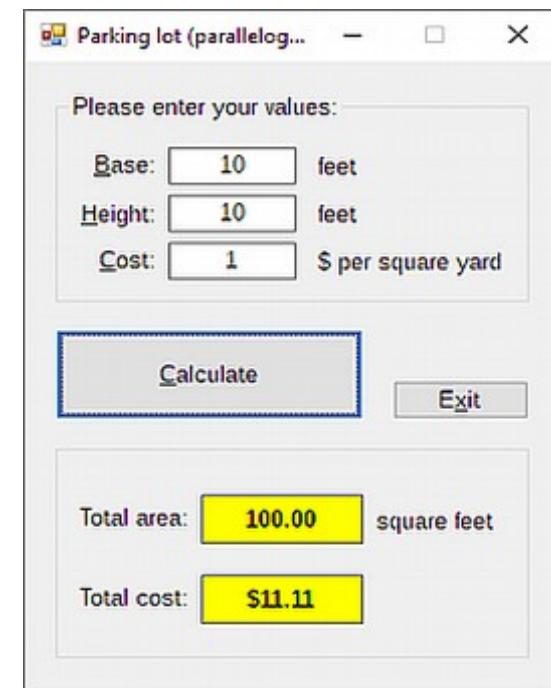
```



```

38     Public Property Cost As Double
39         Get
40             Return dblCost
41         End Get
42         Set(value As Double)
43             If value > 0 Then
44                 dblCost = value
45             Else
46                 dblCost = 0
47             End If
48         End Set
49     End Property
50
51     Public Sub New()
52         ' default constructor
53         dblBase = 0 : dblHeight = 0 : dblCost = 0
54     End Sub
55
56     Public Sub New(ByVal dblB As Double, ByVal dblH As Double, ByVal dblC As Double)
57         ' parameterized constructor
58         Base = dblB : Height = dblH : Cost = dblC
59     End Sub
60
61     Public Function GetArea() As Double
62         ' area in square feet
63         Return dblBase * dblHeight
64     End Function
65
66     Public Function GetCost() As Double
67         ' calculate price in square yards (1 square yard = 9 square feet)
68         Return (GetArea() / 9) * dblCost
69     End Function
70 End Class

```



- instantiate an **Object** from **Class Dues**: **Dim MyDues As Dues**, ...,
- **Private**...class-level variable used as an accumulator, **Select Case False, orElse, MessageBox.Show**,
- chk.Checked, rad.Checked, lbl.BackColor = Color.Yellow, = SystemColors.Control**
- invoke a **Parameterized constructor**: **MyDues = New Dues(inputs/parametres)**
- **2D Array**: **Dim int2DPrices(,) As Integer = {{50, 25, 30, 20}, {...2nd row column values}}**

9. Create a Windows Forms application. Use the following names for the project and solution, respectively: Glasgow Project and Glasgow Solution. Save the application in the VB2017\Chap10 folder. Create the interface shown in Figure 10-43. Each member of Glasgow Health Club must pay monthly dues that consist of a basic fee and one or more optional charges. The basic monthly fee for a single membership is \$50; for a family membership, it is \$90. If the member has a single membership, the additional monthly charges are \$25 for golf, \$30 for tennis, and \$20 for racquetball. If the member

has a family membership, the additional monthly charges are \$35 for golf, \$50 for tennis, and \$30 for racquetball. The application should display the member's basic fee, additional charges, and monthly dues. Create a class named **Dues** that contains two auto-implemented properties for the basic and additional charges. The class should also contain a default constructor, a parameterized constructor, and a method that calculates and returns the total monthly dues. Use the class to code the application. Save the solution and then start and test the application.

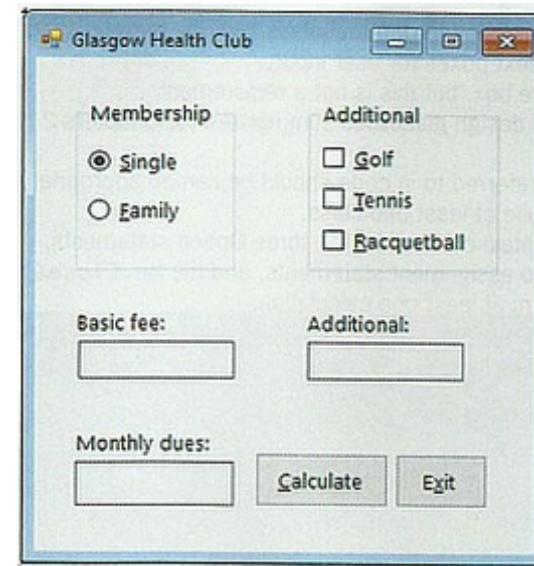


Figure 10-43 Interface for Exercise 9

```

1  ' Name:      Glasgow Project
2  ' Purpose:   Calculate member's monthly due
3  ' Programmer: PoaKamaNdizi on JustNow
4  ' _1. create GUI shown in Figure 10-43
5
6  '_2. frmMain: display: member's basic fee, additional charges, monthly dues
7  ' - monthly dues consists of a basic fee and 1 or more optional charges
8  '   -> single/family + Golf, Tennis, Racquetball
9  '   - if single membership ($50), then: Golf = $25, Tennis = $30, Racquetball = $20
10 '   - if family membership ($90), then: Golf = $35, Tennis = $50, Racquetball = $30
11
12 '_3. class.vb: - use the class to code the application
13 '   - create a Class named "Dues", that:
14 '   - contains 2 auto-implemented properties for: Basic charges and Additional charges
15 '   - default constructor, parameterized constructor
16 '   - method that calculates and returns the total monthly dues
17 ' test

```

```
18 Option Explicit On
19 Option Strict On
20 Option Infer Off
21
22 Public Class frmMain
23     ' accumulator for chks:
24     Public intAdditional As Integer
25
26     Private Sub btnCalculate_Click(sender As Object, e As EventArgs) Handles btnCalculate.Click
27
28         Select Case False
29             ' if non chk is selected:
30             Case chk1golf.Checked OrElse chk2tennis.Checked OrElse chk3racq.Checked
31                 MessageBox.Show("Please select at least 1 activity.", "Glasgow Health Club",
32                                 MessageBoxButtons.OK, MessageBoxIcon.Information)
33
34             ' if at least 1 chk is selected:
35             Case Else
36                 lbl1basic.BackColor = Color.Yellow : lbl2additional.BackColor = Color.Yellow
37                 lbl3dues.BackColor = Color.Yellow
38
39                 intAdditional = 0      ' erase previous values, otherwise they would accumulate
40
41                 Dim MyDues As Dues
42                 Dim intBasic As Integer
43                 Dim int2DPrices(,) As Integer = {{50, 25, 30, 20}, {90, 35, 50, 30}} ' 0,0 = 50; 0,1 = 25...
44                 'MessageBox.Show(int2DPrices(0, 1).ToString) '= 25, ok
45
46                 Select Case True
47                     Case rad1single.Checked
48                         intBasic = int2DPrices(0, 0)
49
50                     If chk1golf.Checked = True Then
51                         intAdditional += int2DPrices(0, 1)
52                     End If
53
54                     If chk2tennis.Checked = True Then
55                         intAdditional += int2DPrices(0, 2)
56                     End If
57
58                     If chk3racq.Checked = True Then
59                         intAdditional += int2DPrices(0, 3)
60                     End If
61
```

```

62             Case rad2family.Checked
63                 intBasic = int2DPrices(1, 0)
64
65                 If chk1golf.Checked = True Then
66                     intAdditional += int2DPrices(1, 1)
67                 End If
68
69                 If chk2tennis.Checked = True Then
70                     intAdditional += int2DPrices(1, 2)
71                 End If
72
73                 If chk3racq.Checked = True Then
74                     intAdditional += int2DPrices(1, 3)
75                 End If
76             End Select
77
78             'MessageBox.Show(intBasic.ToString) ' check the dimension_OK
79             'MessageBox.Show(intAdditional.ToString) ' check additional_OK
80
81             ' invoke parameterized constructor by flushing him input data:
82             MyDues = New Dues(intBasic, intAdditional)
83
84             lbl1basic.Text = intBasic.ToString("C2")
85             lbl2additional.Text = intAdditional.ToString("C2")
86             lbl3dues.Text = MyDues.GetTotal.ToString("C2")
87         End Select
88     End Sub
89
90     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
91         Me.Close()
92     End Sub
93
94     Private Sub radsANDchks_CheckedChanged(sender As Object, e As EventArgs) Handles rad1single.CheckedChanged,
95         rad2family.CheckedChanged, chk1golf.CheckedChanged, chk2tennis.CheckedChanged, chk3racq.CheckedChanged
96         lbl1basic.BackColor = SystemColors.Control : lbl1basic.Text = Nothing
97         lbl2additional.BackColor = SystemColors.Control : lbl2additional.Text = Nothing
98         lbl3dues.BackColor = SystemColors.Control : lbl3dues.Text = Nothing
99     End Sub
100    End Class

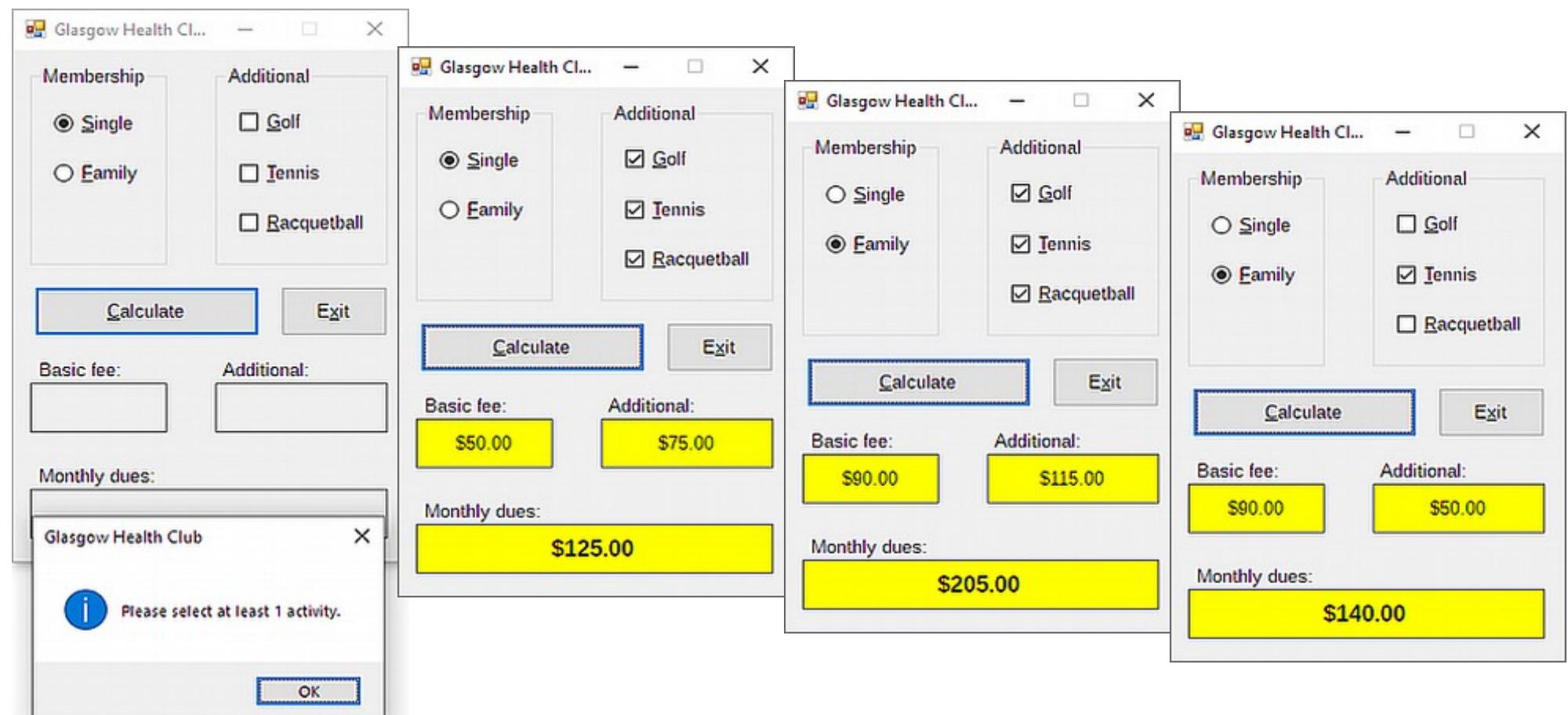
```

```
1  ' Name:      Dues.vb
2  ' Programmer: PoaKamaNdizi on JustNow
3  ' _3. class.vb: - use the class to code the application
4  '   - create a Class named "Dues", that:
5  '   - contains 2 auto-implemented properties for: Basic charges and Additional charges
6  '   - default constructor, parameterized constructor
7  '   - method that calculates and returns the total monthly dues
8 Option Explicit On
9 Option Strict On
10 Option Infer Off
11
12 Public Class Dues
13     Private intBasic As Integer
14     Private intAdditional As Integer
15
16     Public Property BasicCharges As Integer
17         Get
18             Return intBasic
19         End Get
20         Set(value As Integer)
21             If value > 0 Then
22                 intBasic = value
23             Else
24                 intBasic = 0
25             End If
26         End Set
27     End Property
28
29     Public Property AdditionalCharges As Integer
30         Get
31             Return intAdditional
32         End Get
33         Set(value As Integer)
34             If value > 0 Then
35                 intAdditional = value
36             Else
37                 intAdditional = 0
38             End If
39         End Set
40     End Property
41
42     Public Sub New()
43         intBasic = 0 : intAdditional = 0
44     End Sub
```

```

45
46     Public Sub New(ByVal intB As Integer, ByVal intA As Integer)
47         'intB = BasicCharges : intA = AdditionalCharges    ' this won't work
48         BasicCharges = intB : AdditionalCharges = intA      ' this is working
49     End Sub
50
51     Public Function GetTotal() As Integer
52         Return intBasic + intAdditional
53     End Function
54 End Class

```



- instantiate an **Object** from **Class Check**: `Dim MyCheck As Check`, ...,
- invoke a **Parameterized constructor**: `MyCheck = New Check(inputs/parametres)`
- `txt.Text.Trim`, `lbl.Visible =...`, `txt_KeyPress`, `e.KeyChar`, `ControlChars.Back`, `e.Handled`, `txt_TextChanged`, `txt_Enter`, `txt.SelectAll()`, `If value Like "##/#/#/#/#" Then`, `outFile As IO.StreamWriter`, `Public Function Save() As IO.StreamWriter`, `Return outFile`, `IO.File.AppendText`, `outFile.WriteLine`, `outFile.Close`, `MessageBox.Show`, `Public Function ShowLabel() As Boolean`

10. Create a Windows Forms application. Use the following names for the project and solution, respectively: Serenity Project and Serenity Solution. Save the application in the VB2017\Chap10 folder. Create the interface shown in Figure 10-44. The manager of the Accounts Payable department at Serenity Photos wants you to create an application that keeps track of the checks written by her department. More specifically, she wants to record (in a sequential access file) the check number, date, payee, and amount of each check. Create a class that can instantiate a check object. The class should contain a default constructor, a parameterized constructor, and a method that saves the check information to a sequential access file. Use the class to code the application. Save the solution and then start and test the application.

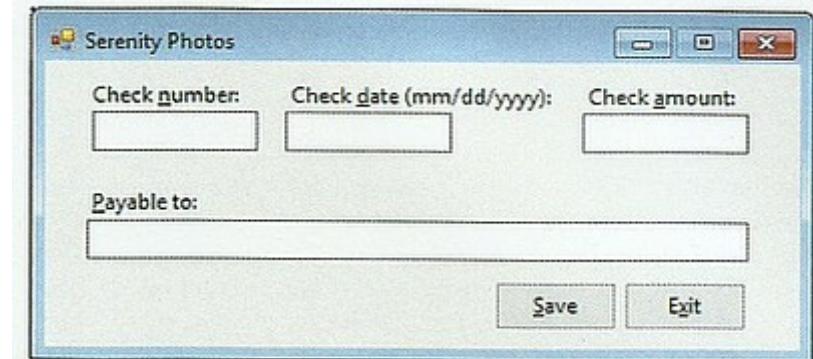


Figure 10-44 Interface for Exercise 10

```

1  ' Name:      Serenity project
2  ' Purpose:    Record the payments
3  ' Programmer: PoaKamaNdizi on TimeStamp
4  '_1. create GUI shown in Figure 10-44
5  '_2. The manager of the Accounts Payable department at Serenity Photos wants an app, that
6  '_   keeps track of the checks written by her department: she wants to record
7  '_   - in sequential access file: the check number, date, payee, amount of each check.
8  '_3. class.vb
9  '_   - create a class that can instantiate a "check" object. The Class should contain:
10 '_   - a default constructor, a parameterized constructor
11 '_   - a method that saves the check information to a sequential access file
12 '_   - use the Class to code the application. Test
13 Option Explicit On
14 Option Strict On
15 Option Infer Off
16
17 Public Class frmMain
18     Private Sub btnSave_Click(sender As Object, e As EventArgs) Handles btnSave.Click
19         Dim MyCheck As Check
20         Dim intNumber As Integer
21         Dim strDate As String = txt2Date.Text.Trim
22         Dim dblAmount As Double
23         Dim strName As String = txt4Name.Text.Trim

```

```
24
25     Integer.TryParse(txt1Number.Text.Trim, intNumber)
26     Double.TryParse(txt3Amount.Text.Trim, dblAmount)
27
28     ' invoke a parameterized constructor and flush him input data:
29     MyCheck = New Check(intNumber, strDate, dblAmount, strName)
30
31     MyCheck.Save()      ' call "Check" Class's function named "Save"
32     lblSaved.Visible = MyCheck.ShowLabel    ' call "Check" Class's function named "ShowLabel" and assign its return value
33 End Sub
34
35 Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
36     Me.Close()
37 End Sub
38
39 Private Sub txt1Number_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txt1Number.KeyPress
40     ' input: Check number - only numbers, BackSpace key:
41     If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
42         e.Handled = True
43     End If
44 End Sub
45
46 Private Sub txt2Date_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txt2Date.KeyPress
47     ' input: date - only numbers, "/", BackSpace key:
48     If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> "/" AndAlso e.KeyChar <> ControlChars.Back Then
49         e.Handled = True
50     End If
51 End Sub
52
53 Private Sub txt3Amount_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txt3Amount.KeyPress
54     ' input: Amount - only numbers, ".", BackSpace key:
55     If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> "." AndAlso e.KeyChar <> ControlChars.Back Then
56         e.Handled = True
57     End If
58 End Sub
59
60 Private Sub txts_TextChanged(sender As Object, e As EventArgs) Handles txt1Number.TextChanged, txt2Date.TextChanged,
61   txt3Amount.TextChanged, txt4Name.TextChanged
62     lblSaved.Visible = False
63 End Sub
64
65 Private Sub txt1Number_Enter(sender As Object, e As EventArgs) Handles txt1Number.Enter
66     txt1Number.SelectAll()
67 End Sub
```

```

68
69      Private Sub txt2Date_Enter(sender As Object, e As EventArgs) Handles txt2Date.Enter
70          txt2Date.SelectAll()
71      End Sub
72
73      Private Sub txt3Amount_Enter(sender As Object, e As EventArgs) Handles txt3Amount.Enter
74          txt3Amount.SelectAll()
75      End Sub
76
77      Private Sub txt4Name_Enter(sender As Object, e As EventArgs) Handles txt4Name.Enter
78          txt4Name.SelectAll()
79      End Sub
80  End Class

```

```

1  ' Name:      Check.vb
2  ' Programmer: PoaKamaNdizi on TimeStamp
3  ' _2. The manager of the Accounts Payable department at Serenity Photos wants an app, that
4  '   keeps track of the checks written by her department: she wants to record
5  '   - in sequential access file: the check number, date, payee, amount of each check.
6  ' _3. class.vb
7  '   - create a class that can instantiate a "check" object. The Class should contain:
8  '   - a default constructor, a parameterized constructor
9  '   - a method that saves the check information to a sequential access file
10 '   - use the Class to code the application. Test
11 Option Explicit On
12 Option Strict On
13 Option Infer Off
14
15 Public Class Check
16     Private intNumber As Integer    ' input: txt1Number
17     Private strDate As String      ' input: txt2Date
18     Private dblAmount As Double    ' input: txt3Amount
19     Private strName As String      ' input: txt4Name
20     Private outFile As IO.StreamWriter        ' CH9_F2
21     Private blnIsTrue As Boolean    ' will b used to show a confirmation label: "lblSaved"
22
23     Public Property CheckNumber As Integer
24         Get
25             Return intNumber
26         End Get

```

```
27      Set(value As Integer)
28          If value > 0 Then
29              intNumber = value
30          Else
31              intNumber = 0
32          End If
33      End Set
34  End Property
35
36  Public Property CheckDate As String
37      Get
38          Return strDate
39      End Get
40      Set(value As String)
41          If value Like "##/##/####" Then
42              strDate = value
43          Else
44              strDate = Nothing
45          End If
46      End Set
47  End Property
48
49  Public Property CheckAmount As Double
50      Get
51          Return dblAmount
52      End Get
53      Set(value As Double)
54          If value > 0 Then
55              dblAmount = value
56          Else
57              dblAmount = 0
58          End If
59      End Set
60  End Property
61
62  Public Property CheckName As String
63      Get
64          Return strName
65      End Get
```

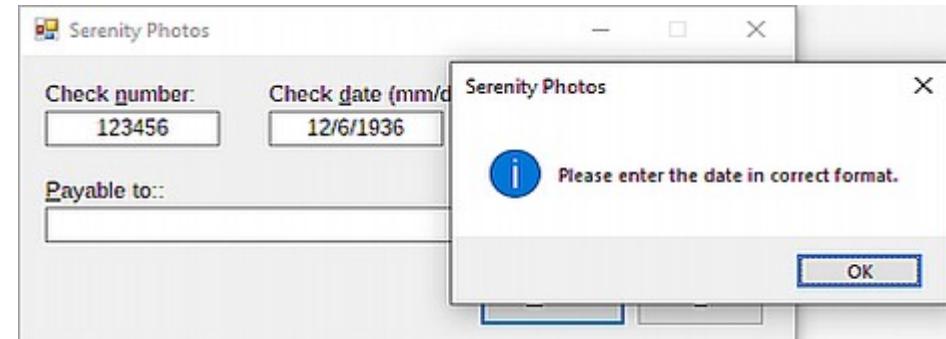
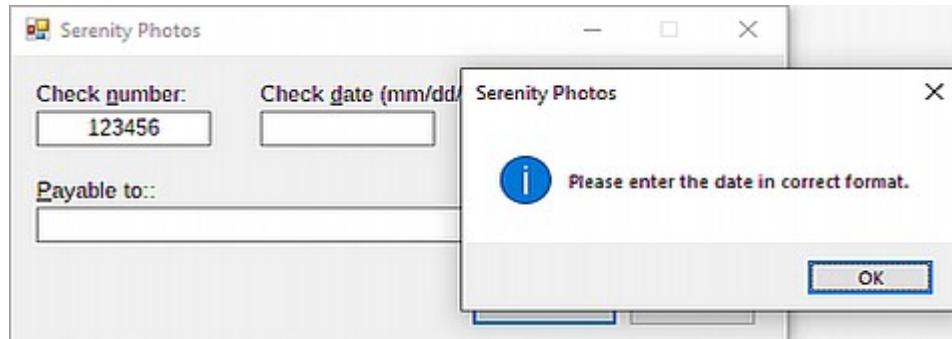
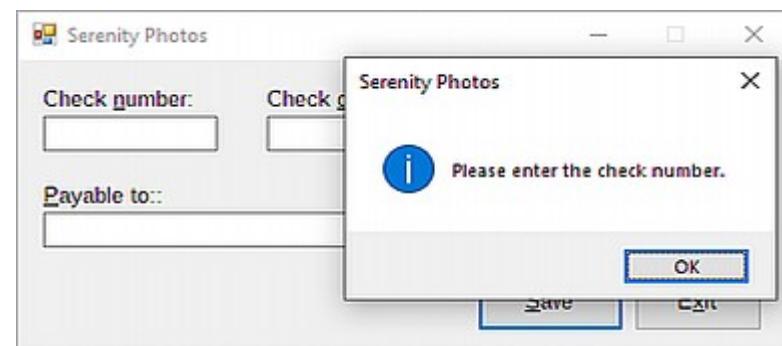
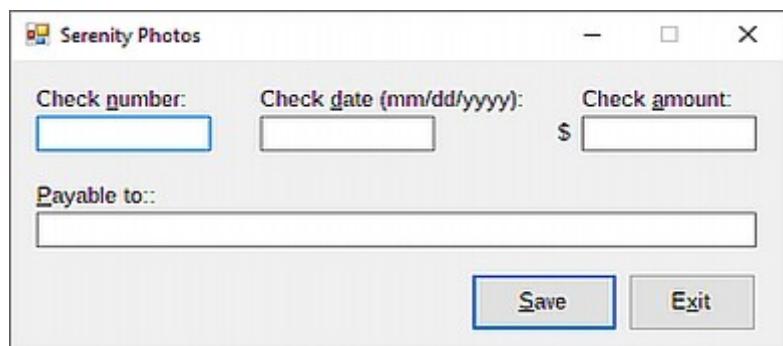
```
66     Set(value As String)
67         If value <> Nothing Then
68             strName = value
69         Else
70             strName = Nothing
71         End If
72     End Set
73 End Property
74
75 Public Sub New()
76     ' default constructor:
77     intNumber = 0 : strDate = Nothing : dblAmount = 0 : strName = Nothing
78 End Sub
79
80 Public Sub New(ByVal intN As Integer, ByVal strD As String, ByVal dblA As Double, ByVal strN As String)
81     ' parameterized constructor:
82     CheckNumber = intN : CheckDate = strD : CheckAmount = dblA : CheckName = strN
83 End Sub
84
85 Public Function Save() As IO.StreamWriter
86     ' conditions:
87     ' - txt1Number > 0; txt2Date must have a form: ##/##/####; txt3Amount > 0, txt4Name <> String.Empty
88
89     If intNumber > 0 Then          ' #1 condition: number > 0
90         If strDate <> Nothing Then      ' #2 condition: date in correct format
91             If dblAmount > 0 Then        ' #3 condition: amount > 0
92                 If strName <> Nothing Then    ' #4 condition: Name is filled
93                     ' if all of the conditions are met:
94                     outFile = IO.File.AppendText("Checks.txt")
95                     outFile.WriteLine("Check number: " & intNumber.ToString & ", Date: " & strDate &
96   ", Payee: " & strName & ", Amount: " & dblAmount.ToString("C2"))
97                     outFile.Close()
98                 Else
99                     MessageBox.Show("Please enter the name.", "Serenity Photos", MessageBoxButtons.OK,
100   MessageBoxIcon.Information)
101                 End If
102             Else
103                 MessageBox.Show("Please enter the amount.", "Serenity Photos", MessageBoxButtons.OK,
104   MessageBoxIcon.Information)
105             End If
106         Else
107             MessageBox.Show("Please enter the date in correct format.", "Serenity Photos", MessageBoxButtons.OK,
108   MessageBoxIcon.Information)
109         End If

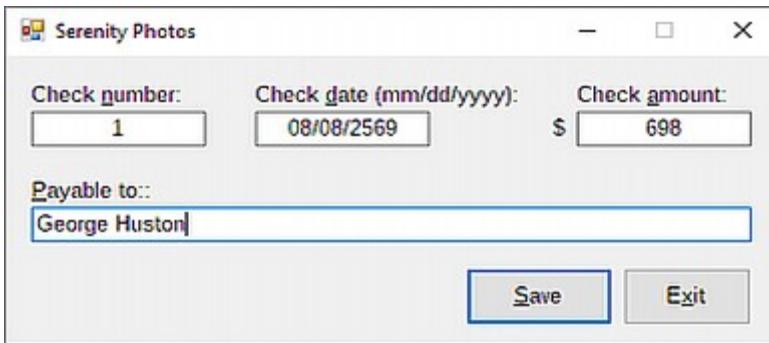
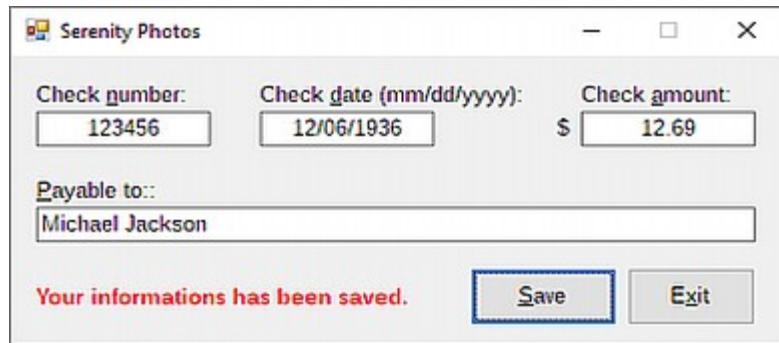
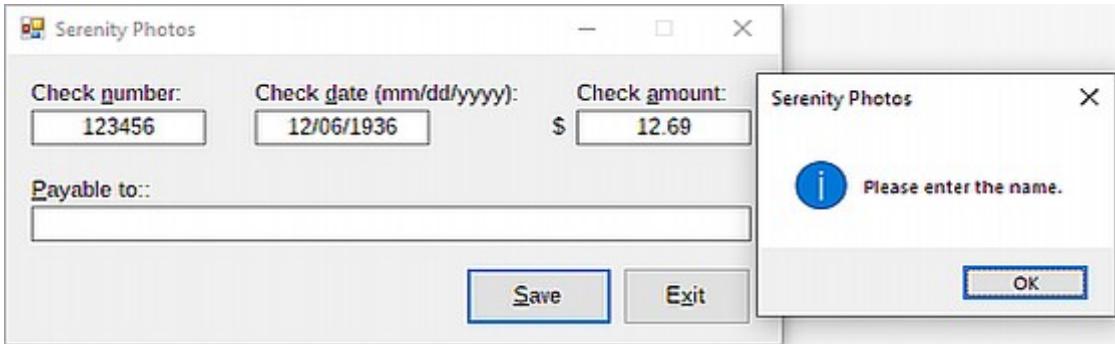
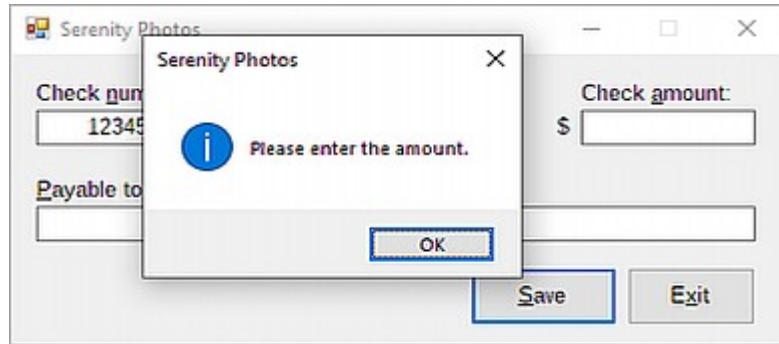
```

```

110     Else
111         MessageBox.Show("Please enter the check number.", "Serenity Photos", MessageBoxButtons.OK,
112   MessageBoxIcon.Information)
113     End If
114
115     Return outFile
116 End Function
117
118 Public Function ShowLabel() As Boolean
119     ' function for showing the confirmation label:
120     If intNumber <> 0 AndAlso strDate <> Nothing AndAlso dblAmount <> 0 AndAlso strName <> Nothing Then
121         blnIsTrue = True
122     End If
123     Return blnIsTrue
124 End Function
125 End Class

```





## Chap10\Exercise1

### 21.FixIt Solution\_EXERCISE 12

- instantiate an **Object** from **Class Employee** using a **Default constructor**: **Dim companyEmployee As New Employee**,
- auto-implemented **Public Property** with hidden: **Private** member variable: **\_variable** & **Get** & **Set** block of codes

12. Open the VB2017\Chap10\FixIt Solution\FixIt Solution.sln file. The application should calculate an employee's new salary, which is based on a 5% raise. It should display the new salary with a dollar sign and no decimal places. Open the Employee.vb file and review the existing code. Then, open the form's Code Editor window. Notice the squiggles in the btnCalc\_Click procedure. Correct the code to remove the squiggles. Save the solution and then start and test the application. (If the current salary is 12000, the new salary is \$12,600.)

```
1  ' Name:      FixIt Project
2  ' Purpose:    Display an employee's new salary.
3  ' Programmer: PoaKamaNdizi on TimeStamp
4  ' The application should calculate an employee's new salary, which is based on a 5% raise.
5  ' It should display the new salary with a dollar sign and no decimal places
6  ' 1. open the "Employee.vb" file and review the existing code
7  ' 2. open the form's Code Editor window: notice the squiggles in the btnCalc_Click procedure
8  '   - correct the code to remove the squiggles
9  ' 3. test: if the current salary is 12000, the new salary is $12,600.
10 Option Explicit On
11 Option Strict On
12 Option Infer Off
13
14 Public Class frmMain
15     Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
16         ' Calculates an employee's new salary, which is based on a 5% raise.
17
18         'Dim companyEmployee As Employee      ' original 1
19         Dim companyEmployee As New Employee   ' fix 1
20
21         Dim dblNewSalary As Double
22         Const dblRate As Double = 0.05           ' fix 2.0
23
24         'Double.TryParse(txtCurrentSalary.Text, companyEmployee._Salary)      ' original 3
25         Double.TryParse(txtCurrentSalary.Text, companyEmployee.Salary)        ' fix 3
26
27         'dblNewSalary = companyEmployee.GetNewSalary()          ' original 2.1
28         dblNewSalary = companyEmployee.GetNewSalary(dblRate)       ' fix 2.1
29
30         lblNewSalary.Text = dblNewSalary.ToString("C0")
31 End Sub
```

```

32
33  Private Sub txtEmpNum_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtEmpNum.KeyPress,
34  txtCurrentSalary.KeyPress
35      ' Accept only numbers and the Backspace key.
36      If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
37          e.Handled = True
38      End If
39  End Sub
40
41  Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
42      Me.Close()
43  End Sub
44
45  Private Sub txtCurrentSalary_Enter(sender As Object, e As EventArgs) Handles txtCurrentSalary.Enter
46      txtCurrentSalary.SelectAll()
47  End Sub
48
49  Private Sub txtEmpNum_Enter(sender As Object, e As EventArgs) Handles txtEmpNum.Enter
50      txtEmpNum.SelectAll()
51  End Sub
52
53  Private Sub ClearNewSalary(sender As Object, e As EventArgs) Handles txtEmpNum.TextChanged, txtCurrentSalary.TextChanged
54      lblNewSalary.Text = String.Empty
55  End Sub
56 End Class

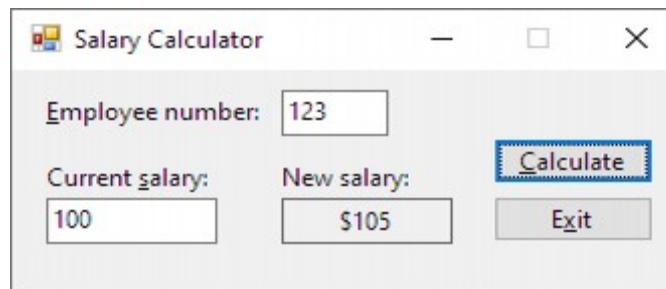
```

```

1  ' Name:      Employee.vb
2  ' Programmer: <your name> on <current date>
3  Option Explicit On
4  Option Strict On
5  Option Infer Off
6
7  Public Class Employee
8      Public Property EmployeeName As String      ' CH10_A2 - auto-implemented Public Property
9      Public Property Salary As Double            ' CH10_A2 - auto-implemented Public Property
10
11     Public Sub New()
12         _EmployeeName = String.Empty
13         _Salary = 0
14     End Sub
15

```

```
16     Public Function GetNewSalary(ByVal dblRate As Double) As Double
17         ' Calculates the new salary.
18
19         Return _Salary * (1 + dblRate)
20     End Function
21 End Class
```



## Theater Seats (Chapters 1–10)

Create an interface that contains a list box with the following 10 items: A1, B1, A2, B2, A3, B3, A4, B4, A5, and B5. Each item represents a seat number in a theater. When the user clicks a list box item, the application should display the seat number, theater patron's name, and ticket price. The application should use a sequential access file, a class, and a one-dimensional array. You can create the sequential access file by using the New File option on the File menu. The file should contain 10 seat numbers, patron names, and ticket prices.

### MainForm.vb

```
1  ' Name:      Theater seats.
2  ' Purpose:   Display the theater seat's patron and price.
3  ' Programmer: me on just now.
4  Option Explicit On
5  Option Strict On
6  Option Infer Off
7
8  Public Class frmMain
9      Private strSeats() As String = {"A1", "B1", "A2", "B2", "A3", "B3", "A4", "B4", "A5", "B5"}
10     Private strSelectedSeat As MyTheaterClass
11
12     Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles Me.Load
13         For intIndex As Integer = 0 To 9
14             lstSeats.Items.Add(strSeats(intIndex))
15         Next intIndex
16     End Sub
17
18     Private Sub lstSeats_SelectedIndexChanged(sender As Object, e As EventArgs) Handles lstSeats.SelectedIndexChanged
19         strSelectedSeat = New MyTheaterClass(lstSeats.SelectedItem.ToString)
20     End Sub
21
22     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
23         Me.Close()
24     End Sub
25 End Class
```

## MyTheaterClass.vb

```
1  ' Name: MyTheaterClass
2  Option Explicit On
3  Option Strict On
4  Option Infer Off
5
6  Public Class MyTheaterClass
7      ' 1. Class's member variables.
8      ' 2. Property section.
9      ' 3. Behavior section.
10
11     ' 1. Class's member variables.
12     ' i will send a selectet item from 1st to here and according to that this class will pick the right
13     '   line in Sequential Access File named TheaterSeats.txt.
14     Private strSeat As String
15
16     ' 2. Property section.
17     Public Property Seat As String
18         Get
19             Return strSeat
20         End Get
21
22         Set(value As String)
23             strSeat = value
24         End Set
25     End Property
26
27     ' 3. Behavior section.
28     Public Sub New(ByVal StrA As String)
29         Seat = StrA
30
31         Dim inFile As IO.StreamReader
32
33         If IO.File.Exists("TheaterSeats.txt") Then
34             'MessageBox.Show("Yees") ' file location: ...\\bin\\Debug
35
36             ' 1. use a Sequential Access File to fill the 1st lbl:
37             inFile = IO.File.OpenText("TheaterSeats.txt")
38             Dim strFirstLine As String : Dim strSecondLine As String
39
```

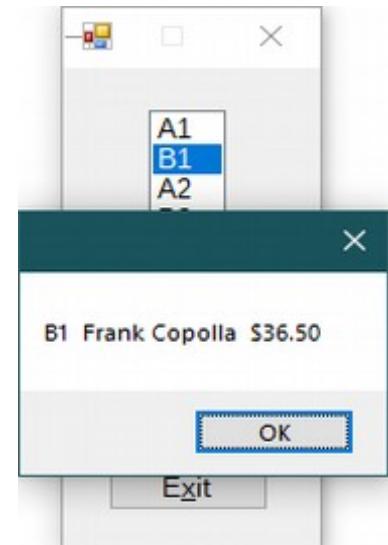
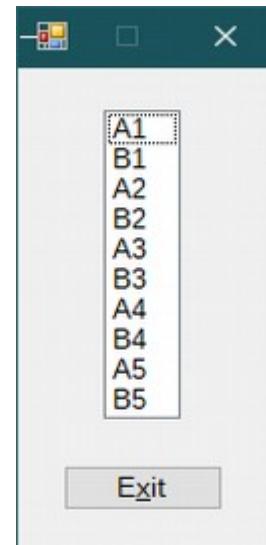
```

40      Do Until inFile.Peek = -1
41          strFirstLine = inFile.ReadLine
42          strSecondLine = inFile.ReadLine
43
44          ' instructions here:
45          If StrA Like strFirstLine Then
46              MessageBox.Show(strFirstLine & " " & strSecondLine & " " & inFile.ReadLine) ' =OK juchuuu
47          End If
48      Loop
49
50      ' close the file:
51      inFile.Close()
52  Else
53      MessageBox.Show("Can't locate the TheaterSeats.txt file.", "Theater Seats", MessageBoxButtons.OK, MessageBoxIcon.Error)
54  End If
55 End Sub
56 End Class

```

...|bin|Debug|TheaterSeats.txt

| 1  | A1                   |  |
|----|----------------------|--|
| 2  | Joseph Messiah       |  |
| 3  | \$25.50              |  |
| 4  |                      |  |
| 5  | B1                   |  |
| 6  | Frank Copolla        |  |
| 7  | \$36.50              |  |
| 8  |                      |  |
| 9  | A2                   |  |
| 10 | Martin Hulala        |  |
| 11 | \$31.20              |  |
| 12 |                      |  |
| 13 | B2                   |  |
| 14 | Carl von Bahnhof     |  |
| 15 | \$18.58              |  |
| 16 |                      |  |
| 17 | A3                   |  |
| 18 | Francois Miteran     |  |
| 19 | \$22.30              |  |
| 20 |                      |  |
| 21 | B3                   |  |
| 22 | Richard Lion         |  |
| 23 | \$16.36              |  |
| 24 |                      |  |
| 25 | A4                   |  |
| 26 | Anetta Lumit         |  |
| 27 | \$38.20              |  |
| 28 |                      |  |
| 29 | B4                   |  |
| 30 | Smil Flek z Nohavice |  |
| 31 | \$11.45              |  |
| 32 |                      |  |
| 33 | A5                   |  |
| 34 | Jane Austinne        |  |
| 35 | \$32.30              |  |
| 36 |                      |  |
| 37 | B5                   |  |
| 38 | Tomas Marny          |  |
| 39 | \$21.20              |  |



Try...Catch "safety net" statement = handling/catching app's exception/error during run time without terminating the app

- most businesses need to keep track of vast amounts of information pertaining to their customers, employees, and inventory
- although a business could store the information in sequential access files, which you learned about in **Chapter 9**, it is much easier to manipulate the information when it is stored in computer databases
- in this chapter's **Focus on the Concepts lesson**, you will learn how to create computer databases and then use them in applications
- the **Apply the Concepts lesson** expands on the topics covered in the Focus lesson

### MySummary:

|                |                                                                                                                                                                                                                                                                                                 |
|----------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>CH11_FA</b> | - Create db with <b>1 table</b> & create <b>dataset</b> & use <b>DataGridView</b> control & include <b>Try...Catch</b> "safety net" statement into <b>Binding Navigator</b> 's events:                                                                                                          |
|                | 1.create db & 2.create Table & 3.add Records & 4.create DataSet &<br>5.create bound control & 6.improve DataGridView control & 7.set db Copy... property &<br>8.include <b>Try...Catch</b> 'safety net' statement into Binding Navigator control                                                |
| <b>CH11_FB</b> | - import db with <b>2 tables</b> & create <b>dataset</b> & use <b>DataGridView</b> control:                                                                                                                                                                                                     |
|                | 1.add existing db containing 2 tables to dataset &<br>2.relate 2 tables by their common field & 3.create Query / custom dataset &<br>4.create bound control & improve DataGridView control                                                                                                      |
| <b>CH11_AA</b> | - import db & create <b>dataset</b> & create <b>Data Form</b> (separate control for each table's field) instead of using <b>DataGridView</b> control &<br>& add <b>Try...Catch</b> 'safety net' statement & <b>KeyPress</b> example: <b>03.Course Info Solution-Data Form</b>                   |
| <b>CH11_AB</b> | - already: imported db, created dataset & <b>Bind</b> table's field objects to <b>existing</b> Label controls by <b>dragging</b> them &<br>& add <b>BindingNavigator</b> control example: <b>04.Course Info Solution-Labels</b>                                                                 |
| <b>CH11_AC</b> | - already: imported db, created dataset, created DataGridView bound control & <b>Code</b> the <b>btnCalc_Click</b> procedure to:<br><b>Loop</b> all the rows/records and accumulate numbers if the field is not empty nor contains letter W example: <b>05.Course Info Solution-Total Hours</b> |

### CH11\_FOCUS ON THE CONCEPTS LESSON

|                  |                                                                                                                                                                                                                                            |                |
|------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------------|
| <b>CH11_F0</b>   | - <b>SQL Server Database</b> : basic info & terminology & e.g. single-table db, two-table db                                                                                                                                               |                |
| <b>CH11_F0.1</b> | - <b>SQL Server data types</b> : comparison with Visual Basic data types basic info: <b>bit</b> , <b>decimal(p, s)</b> , <b>float</b> , <b>int</b> , <b>char(n)</b> , <b>varchar(n)</b>                                                    |                |
| <b>CH11_F1</b>   | - step <b>1/8</b> : create a <b>SQL Server Database</b> with <b>.mdf</b> extension: step-by-step example <b>01.Course Info Solution</b>                                                                                                    | <b>CH11_FA</b> |
| <b>CH11_F2</b>   | - step <b>2/8</b> : add a <b>Table</b> to a <b>Database</b> and define it: step-by-step example <b>01.Course Info Solution</b>                                                                                                             |                |
| <b>CH11_F3</b>   | - step <b>3/8</b> : add <b>Records</b> to a <b>Table</b> : step-by-step example <b>01.Course Info Solution</b>                                                                                                                             |                |
| <b>CH11_F4</b>   | - step <b>4/8</b> : create <b>Dataset</b> = connection between <b>Database</b> and <b>Application</b> - using <b>Data Source Configuration Wizard</b> &<br>& view contents of your <b>dataset</b> : example <b>01.Course Info Solution</b> |                |
| <b>CH11_F5</b>   | - info for <b>step 5/8</b> : about <b>Binding</b> /connecting an object in <b>DataSet</b> to the new or existing <b>control</b> in GUI: basic info & example with a <b>new control</b>                                                     |                |
| <b>CH11_F5.1</b> | - step <b>5/8 Bind</b> an object <b>Courses</b> table to default <b>DataGridView</b> control example <b>01.Course Info Solution</b>                                                                                                        |                |
| <b>CH11_F6</b>   | - info for <b>step 6/8</b> : about <b>DataGridView control</b> for displaying table data - basic info & example                                                                                                                            |                |
| <b>CH11_F6.1</b> | - step <b>6/8</b> : improve the appearance of the <b>DataGridView control</b> - set the <b>columns</b> & its <b>cells</b> example <b>01.Course Info Solution</b>                                                                           |                |
| <b>CH11_F7</b>   | - info for <b>step 7/8</b> : set the local database file's <b>*.mdf</b> property: <b>Copy to Output Directory</b> = correlation between source <b>*.mdf</b> and output <b>*.mdf</b>                                                        |                |
| <b>CH11_F7.1</b> | - step <b>7/8</b> : set the local database file's <b>*.mdf</b> property: <b>Copy to Output Directory</b> - example and testing <b>01.Course Info Solution</b>                                                                              |                |
| <b>CH11_F8</b>   | - info for <b>step 8/8</b> : automatically generated code when <b>binding</b> objects - autopsy & example with <b>01.Course Info Solution</b>                                                                                              |                |
| <b>CH11_F8.1</b> | - info for <b>step 8/8</b> : <b>Try...Catch</b> "safety net" statement = handling/catching app's <b>exception/error</b> during run time without terminating the app & example                                                              |                |
| <b>CH11_F8.2</b> | - step <b>8/8</b> : include a <b>Try...Catch</b> "safety net" statement to handle/catch possible exceptions/errors example <b>01.Course Info Solution</b>                                                                                  |                |

CH11\_FB

CH11\_F9 - explore the **Two-table** database: basic info with example **02.Charleston Sales Solution**

CH11\_F9.1 - step 1/4: add an existing database file **Charleston.mdf** to an application & create a **dataset** &

& add it's **2 tables** to the **dataset** example **02.Charleston Sales Solution**

CH11\_F9.2 - step 2/4: relate 2 tables by their **common field: CountryCode** example **02.Charleston Sales Solution**

CH11\_F9.3 - step 3/4: create a **Database Query object** from **2 table objects** related by their common field, using **foreign & primary keys** =

= **single dataset** as a **combination** of fields from **2 tables** - info & example **02.Charleston Sales Solution**

CH11\_F9.4 - step 4/4: display the **Database Query/Query** information in the **DataGridView** control = create a bound control from your "custom" dataset &

& improve control's appearance - info & example **02.Charleston Sales Solution**

## CH11\_APPLY THE CONCEPTS LESSON

CH11\_A1 - Create a **Data Form** (instead of using **DataGridView** control) for displaying **dataset**, created from existing \*.mdf database file &

CH11\_AA

& code: add the **Try...Catch** 'safety net' statement & KeyPress procedures example: **03.Course Info Solution-Data Form**

CH11\_A2 - **Bind Field** objects to **existing** controls - basic info

CH11\_A2.1 - **Bind Field** objects to **existing** controls by dragging them to the form & add **BindingNavigator** control example: **04.Course Info Solution-Labels**

CH11\_AB

CH11\_A3 - perform **Calculations** on the **Fields** in a Dataset using loop: **For Each row As MyCoursesDataSet.CoursesRow In MyCoursesDataSet.Courses.Rows**

info & example: **05.Course Info Solution-Total Hours**

CH11\_AC

CH11\_Summary:

CH11\_Key Terms & terminology marked: ->

CH11\_Exercises

**CH11\_FA - Create db with 1 table & create dataset & use DataGridView control & include Try...Catch "safety net" statement into Binding Navigator's events:**

- 1.create db & 2.create Table & 3.add Records & 4.create DataSet &
- 5.create bound control & 6.improve DataGridView control & 7.set db Copy... property &
- 8.include **Try...Catch** 'safety net' statement into Binding Navigator control

- an example with: **01.Course Info Solution**

Your Project Name = **Course\_Info** YourDatabaseName = **MyCourses.mdf**

YourTableName = **Courses.dbo** PrimaryKeyHeaderField = **ID** OtherHeaderField = **Code, Title, Hours, Grade**

YourConnectionString = **MyCoursesConnectionString** YourDataSetName = **MyCoursesDataSet.xsd** YourBoundControl = **CoursesDataGridView**

YourSaveDataButtonProcedure = **CoursesBindingNavigatorSaveItem\_Click**

**Step 1: create a database: MyCourses.mdf**

**CH11\_F1 -**

- |                 |                                                       |                                                                                                                       |
|-----------------|-------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------|
| <b>step 1.1</b> | - Add new database:                                   | - VS / menu: <b>Project -&gt; Add New Item...</b> <b>Ctrl+Shift+A</b>                                                 |
| <b>step 1.2</b> | - Choose Data item:                                   | - window: <b>Add New Item</b> / left tab -> <b>Installed \ Common Items \ Data</b>                                    |
| <b>step 1.3</b> | - Choose Service-based Database:                      | - window: <b>Add New Item</b> / middle tab -> <b>Service-based Database</b> , with default <b>Name: Database1.mdf</b> |
| <b>step 1.4</b> | - Change db name:                                     | - window: <b>Add New Item</b> / middle tab -> change default Name to: <b>MyCourses</b>                                |
| <b>step 1.5</b> | - Create db:                                          | - window: <b>Add New Item</b> / middle tab -> click the button <b>Add</b>                                             |
| <b>step 1.6</b> | - Verify db name:                                     | - window: <b>Solution Explorer</b> / <b>Course_Info Project</b> -> <b>MyCourses.mdf</b>                               |
| <b>step 1.7</b> | - Open db for edit in window <b>Server Explorer</b> : | - window: <b>Solution Explorer</b> / <b>MyCourses.mdf</b> -> Rclick: <b>Open</b>                                      |
|                 | - and expand the node for next step:                  | - window: <b>Server Explorer</b> / node: <b>Data Connections</b> -> expand <b>MyCourses.mdf</b>                       |

**Step 2: create table: dbo.Courses & define it's header - primary field & other fields in window dbo.Courses [Design]**

**CH11\_F2 -**

- |                 |                                         |                                                                                                                                                                                                                                                                             |
|-----------------|-----------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>step 2.1</b> | - Add default table:                    | - window: <b>Server Explorer</b> / <b>Tables</b> -> Rclick: <b>Add New Table</b>                                                                                                                                                                                            |
| <b>step 2.2</b> | - Change table name:                    | - window: <b>dbo.Table [Design]</b> / pane <b>T-SQL</b> -> line of code: <b>CREATE TABLE [dbo].[Courses]</b>                                                                                                                                                                |
| <b>step 2.3</b> | - Change primary field's name:          | - window: <b>dbo.Courses [Design]</b> / column: -> <b>Name = ID</b>                                                                                                                                                                                                         |
| <b>step 2.4</b> | - Allow primary field's auto-counter:   | - window: <b>Properties</b> <b>ID Column</b> / <b>Identity Specification</b> / <b>(Is Identity) -&gt; = True</b><br><b>Identity Seed -&gt; = 1</b><br><b>Identity Increment -&gt; = 1</b>                                                                                   |
|                 | - and define counter's starting number: | <b>Data Type -&gt; = int</b> (it will be a counter, so Integer is fine)                                                                                                                                                                                                     |
|                 | - and define counter's increment by:    | <b>Allow Nulls -&gt; = False</b> (counter field should not be empty)                                                                                                                                                                                                        |
| <b>step 2.5</b> | - Define primary field's data type:     |                                                                                                                                                                                                                                                                             |
|                 | - and define primary field's emptiness: |                                                                                                                                                                                                                                                                             |
| <b>step 2.6</b> | - Add other fields to a table header:   | - window: <b>dbo.Courses [Design]</b> / column: -><br>-> <b>Name = OtherHeaderFieldName</b> , <b>Data Type = OtherHeaderFieldType</b> , <b>Allow Nulls = can be empty or not?</b><br>-> <b>Name = Code</b> , <b>Data Type = varchar(8)</b> , <b>Allow Nulls = unchecked</b> |
| <b>step 2.7</b> | - Save your table:                      | - window: <b>dbo.Courses [Design]</b> -> button: <b>Update</b> -> button <b>Update Database</b>                                                                                                                                                                             |
|                 | - and close:                            | - window: <b>Data Tools Operations</b> -> button: <b>Close</b>                                                                                                                                                                                                              |

**Step 3: add records to a table in window: dbo.Courses [Data]**

**CH11\_F3 -**

- |                 |                        |                                                                                                     |
|-----------------|------------------------|-----------------------------------------------------------------------------------------------------|
| <b>step 3.1</b> | - expand the nodes:    | - window: <b>Server Explorer</b> / <b>Tables</b> -> <b>Courses</b>                                  |
| <b>step 3.2</b> | - open table for edit: | - window: <b>Server Explorer</b> / <b>Tables</b> / <b>Courses</b> -> Rclick: <b>Show table Data</b> |
| <b>step 3.3</b> | - add records:         | - window: <b>dbo.YourTableName [Data]</b> / <b>otherHeaderFieldName -&gt; ItsData</b>               |
| <b>step 3.4</b> | - save your solution   | - window: <b>dbo.Courses [Data]</b> / <b>Code -&gt; ACCOU110</b>                                    |

**Step 4: create dataset = connect database with your application using Data Source Configuration Wizard**

**CH11\_F4 -**

- |                 |                                                                   |                                                                                              |
|-----------------|-------------------------------------------------------------------|----------------------------------------------------------------------------------------------|
| <b>step 4.1</b> | - Open window to manage <b>Data Sources</b> :                     | - VS / menu: <b>View / Other Windows -&gt; Data Sources</b> <b>Shift+Alt+D</b>               |
| <b>step 4.2</b> | - Start the <b>Data Source Configuration Wizard</b> :             | - window <b>Data Sources</b> -> click <b>Add New Data Sources...</b>                         |
|                 | - and choose a type of your data:                                 | - screen <b>Choose a Data Source Type</b> -> <b>Database</b> & click button <b>Next &gt;</b> |
| <b>step 4.3</b> | - Determine the types of data objects your application code uses: | - screen <b>Choose a Database Model</b> -> <b>Dataset</b> & click button <b>Next &gt;</b>    |

- step 4.4** - Choose your database file to connect:  
     / **Which data connection should your application use to connect to the database?** -> *MyCourses.mdf* & click button **Next >**
- step 4.5** - Save the connection to your application config file:  
     - and **verify or change** the config name:
- step 4.6** - Choose which objects to include (tables, fields) in dataset:  
     - and **verify or change** the dataset name  
     - and close the **Data Source Configuration Wizard**:
- screen **Choose Your Data Connection /**  
     - screen **Save the Connection String to the Application Configuration File /**  
         / check box: **Yes, save the connection as:** -> checked  
         -> *MyCoursesConnectionString* & click button **Next >**  
     - screen **Choose Your Database Objects / node Tables ->**  
         -> node *Courses*  
     - **DataSet name:** -> *MyCoursesDataSet.xsd*  
     -> click button **Finish**

#### Step 4: review content of your dataset      CH11\_F4 -

- step 4.7** - Expand dataset:  
**step 4.8** - Open new screen **Preview Data**:  
**step 4.9** - View your table  
     - and Close the dialog box
- window **Data Sources** / node *MyCoursesDataSet* -> node *Courses*  
     - window **Data Sources** / Rclick *MyCoursesDataSet* -> **Preview Data...**  
     - screen **Preview Data** / node *MyCoursesDataSet* / node *Courses* / node *Fill,GetData()* -> button **Preview**  
     - screen **Preview Data** -> button **Close**

#### Step 5: bind objects in **DataSet** to controls in GUI = create a **Bound control** using **DataGridView** control      CH11\_F5 - & CH11\_F5.1 -

- step 5.1** - choose desired type of control for your object -> **DataGridView** control:  
     - and then drag object from window **Data Source** to **Designer** window:  
**step 5.2** - save and start **Solution** just to see the output, then close the application
- window **Data Sources** / table *Courses* -> list arrow: **DataGridView**  
     - window **Data Sources** / table *Courses* -> drag to window **Designer**

#### Step 6: if you use the **DataGridView** control to display **table**, you can improve the appearance -> set **columns & its cells**      CH11\_F6 - & CH11\_F6.1 -

- step 6.1** - Improve the **DataGridView** control's appearance:  
     - Adjust the table, so no need for a scroll bars:  
**step 6.2** - Improve the **DataGridView** control's appearance:  
     - Anchor the table borders to container borders  
**step 6.3** - open **Edit Columns** dialog box to set **Bound Column Properties**:  
     - and select the header item/column you want to set:  
**step 6.4** - set the header item's **Bound Column Properties**:  
**step 6.5** - if needed, you can set the header item cell's property in the **CellStyle Builder** dialog box:  
**step 6.6** - just see how your application looks to the user:  
     - and learn as a user to navigate between table fields
- window **Designer** -> choose the *CoursesDataGridView* control &  
     & window **Properties** / change default **AutoSizeColumnsMode** = **None** to-> = **Fill**  
     - window **Designer** -> choose the *CoursesDataGridView* control &  
         &-> click the **task box arrow** & -> click the task: **Dock in Parent Container**  
     - window **Designer** -> choose the *CoursesDataGridView* control &  
         &-> click the **task box arrow** & -> click the task: **Edit Columns...**  
     - dialog box **Edit Columns** / Lbox -> choose a header field you want to set  
     - dialog box **Edit Columns** / Rbox -> set the properties you desire  
     - dialog box **Edit Columns** / Rbox **Bound Columns Properties** /  
         / **DefaultCellStyle** = **DataGridViewCellStyle { }** ... -> click the ellipsis  
     -> **F5** to start the application  
     -> use **BindingNavigator** control, a **Tab** key, arrow keys

#### Step 7: set db property: **Copy to Output Directory** = what to do with an output db file: **bin\Debug\\*.mdf** modified during a run time      CH11\_F7 - & CH11\_F7.1 -

- step 7.1** - set to use the most recent copy of the database file:  
     - window **Solution Explorer** / choose *MyCourses.mdf* &  
         & window **Properties** / **Copy to Output Directory** -> = **Copy if newer**

- step 7.2** - just test if the modifications made during a run time are saved for the next session

...

- step 7.5** - etc.

#### Step 8: include a **Try...Catch** 'safety net' statement into a code, automatically generated when binding an object in **step 5.1**      CH11\_F8.1 - & CH11\_F8.2 -

- step 8.1** - include a **Try...Catch** statement into procedure code for **Binding Navigator** control's button **Save Data**:  
     - **Try...Catch** syntax in:      CH11\_F8.1 -
- window **Code Editor** / procedure *CoursesBindingNavigatorSaveItem\_Click*

## CH11\_FB - import db with 2 tables & create dataset & use DataGridView control:

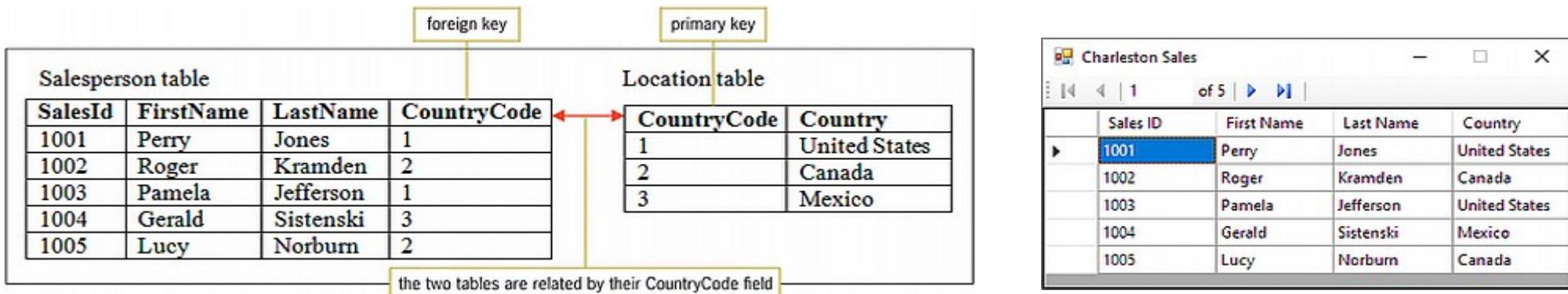
- 1.add existing db containing 2 tables to dataset &
- 2.relate 2 tables by their common field & 3.create Query / custom dataset &
- 4.create bound control & improve DataGridView control

- an example with: **02.Charleston Sales Solution**

Your Project Name = **Charleston Sales**

ImportedDatabaseName = **Charleston.mdf**

|                                                          |                                            |                                                |                                                  |
|----------------------------------------------------------|--------------------------------------------|------------------------------------------------|--------------------------------------------------|
| ChildTableName = <b>Courses.dbo</b>                      | PrimaryKeyHeaderField = <b>SalesId</b>     | OtherHeaderFields = <b>FirstName, LastName</b> | ForeignKeyHeaderField = <b>CountryCode</b>       |
| ParentTableName = <b>Location.dbo</b>                    | PrimaryKeyHeaderField = <b>CountryCode</b> | OtherHeaderFields = <b>Country</b>             |                                                  |
| YourConnectionString = <b>CharlestonConnectionString</b> |                                            | YourDataSetName = <b>CharlestonDataSet.xsd</b> | YourBoundControl = <b>CharlestonDataGridView</b> |



**Step 1:** add existing database: **Charleston.mdf** to an application & create a **dataset** & add it's 2 tables: **Location** & **Salesperson** to the **CharlestonDataSet**

**CH11\_F9.1 -**

**step 1.1** - Open window to manage Data Sources:

- and Start the **Data Source Configuration Wizard**:

**step 1.2**

- Choose a type of your data:

**step 1.3**

- Determine the types of data objects your application code uses:

**step 1.4**

- Choose your database file to connect:

**step 1.4.1**

- Choose source type:

**step 1.4.2**

- Verify previous step

**step 1.4.3**

- Add existing database file **Charleston.mdf** from HDD:

- and Test your connection:

- and close the **Add Connection** dialog box:

**step 1.5**

- Verify your database file name:

- and Confirm:

- and Copy the database file to project:

**step 1.6**

- Save the connection to your application config file:

- and Verify or Change the config name:

**step 1.7**

- Choose which objects to include (tables, fields) in dataset:

- and Verify or Change the dataset name

- and Close the **Data Source Configuration Wizard**:

**step 1.8**

- Verify your dataset:

- VS / menu: **View / Other Windows** -> **Data Sources** Shift+Alt+D

- window **Data Sources** -> click [Add New Data Sources...](#)

- screen **Choose a Data Source Type** -> **Database** & click button **Next >**

- screen **Choose a Database Model** -> **Dataset** & click button **Next >**

- screen **Choose Your Data Connection** -> click button [New Connection...](#)

- dialog box **Choose Data Source** -> click **Microsoft SQL Server Database File** & click button **Continue**

- dialog box **Add Connection** ->

-> verify **Microsoft SQL Server Database File (SqlClient)**

- dialog box **Add Connection** -> button **Browse...** and locate your file

- dialog box **Add Connection** -> button **Test Connection**

- dialog box **Add Connection** -> button **OK**

- screen **Choose Your Data Connection** / box **Which data connection should your application use to connect to the database?** -> **Charleston.mdf**

- screen **Choose Your Data Connection** -> click button **Next >**

- dialog box **Microsoft Visual Studio** -> click button **Yes**

- screen **Save the Connection String to the Application Configuration File /** / check box: **Yes, save the connection as:** -> checked &

-> **CharlestonConnectionString** & click button **Next >**

- screen **Choose Your Database Objects** / node **Tables** ->

-> select tables **Location** & **Salesperson**

- **DataSet name:** -> **CharlestonDataSet.xsd**

-> click button **Finish**

- window **Data Sources** / node **CharlestonDataSet** / nodes **Location** & **Salesperson**

**Step 2:** relate 2 tables by their common field: *CountryCode*

**CH11\_F9.2 -**

- step 2.1** - Open the **DataSet Designer** window:
- Open the **Relation** dialog box:
- Set the relation:

- window **Data Sources** / node *CharlestonDataSet* -> Rclick: **Edit DataSet with Designer**  
- window *CharlestonDataSet.xsd* -> Rclick empty area: **Add / Relation...**  
- dialog box **Relation** / box **Name**: -> = *Location\_Salesperson*  
box **Parent Table**: -> = *Location*  
box **Child Table**: -> = *Salesperson*  
box **Columns**: / **Key Columns** -> = *CountryCode*  
**Foreign Key Columns** -> = *CountryCode*

**Step 3:** create a **Database Query** object from **2 tables** = create a **single dataset** as a **combination** of fields from **2 tables**

**CH11\_F9.3 -**

- step 3.1** - Open screen **TableAdapter Configuration Wizard**:
- step 3.2** - Open dialog box **Query Builder**:
- step 3.3** - Open dialog box **Add Table**:
  - and **Add** a second table *Location*:
  - and **Close** the dialog box **Add Table**:
- step 3.4** - Verify both of the tables:
  - and **Notice** that primary keys appears boldfaced
- step 3.5** - Select the parent table's field to include in child table:
  - Deselect the foreign key field in child table:
- step 3.6** - Execute your changes/setting:
- step 3.7** - Close dialog box **Query Builder**:
  - and **Close** the **TableAdapter Configuration Wizard**:
- step 3.8** - Verify your new Query object:

- window *CharlestonDataSet.xsd* / *SalespersonTableAdapter* / line **Fill,GetData ()** ->  
-> Rclick: **Configure...**  
- screen **TableAdapter Configuration Wizard** -> click button **Query Builder...**  
- dialog box **Query Builder** / diagram pane -> Rclick empty area: **Add Table...**  
- dialog box **Add Table** / tab **Tables** -> select table *Location* & click button **Add** &  
& click button **Close**  
- dialog box **Query Builder** / diagram pane / tables *Salesperson* & *Location*  
- dialog box **Query Builder** / diagram pane / field *SalesId* & field *CountryCode*  
- dialog box **Query Builder** / diagram pane / table *Location* -> select field *Country*  
table *Salesperson* -> deselect field *CountryCode*  
- dialog box **Query Builder** -> click button **Execute Query**  
- dialog box **Query Builder** -> click button **OK**  
- screen **TableAdapter Configuration Wizard** -> click button **Finish**  
- window **Data Sources** / node *CharlestonDataSet* / table *Salesperson* ->  
-> fields: *SalesId*, *FirstName*, *LastName*, & original *CountryCode* changed to: *Country*

**Step 4:** display the **Database Query** information in the **DataGridView** control = create a bound control & improve control's appearance

**CH11\_F9.4 -**

- step 4.1** - Drag table from Data Source to Designer window to create the **DataGridView** control:
- step 4.2** - Improve the **DataGridView** control's appearance:
  - Adjust the table, so no need for a scroll bars:
- step 4.3** - Open the **DataGridView Tasks**:
  - Anchor **DataGridView** borders to container borders:
  - Disallow user to add, edit, or delete records
  - Open dialog box **Edit Columns**:
  - Change the header text appearance:
    - and **Close** the dialog box **Edit Columns**:
- step 4.4** - Change the **frmMain** form size:
- step 4.5** - because the user will be allowed to see the table, you will delete buttons to **Save, Add, Delete**
  - Delete some buttons from **Binding Navigator**:
- step 4.6** - start and test the application

- window **Data Sources** / node *CharlestonDataSet* / table *Salesperson* ->  
-> drag to window **Designer**  
- window **Designer** -> choose the *SalespersonDataGridView* control &  
& window **Properties** / change default **AutoSizeColumnsMode** = **None** to -> = **Fill**  
- window **Designer** -> choose the *SalespersonDataGridView* control &  
& -> click the **task box arrow**  
- **DataGridView Tasks** -> click the task: **Dock in Parent Container**  
- **DataGridView Tasks** -> uncheck 3 chks: **Enable Adding**, **Enable Editing**, **Enable Deleting**  
- **DataGridView Tasks** -> click the task: **Edit Columns ...**  
- dialog box **Edit Columns** / Lbox: *SalesId* & Rbox: -> **HeaderText** = *Sales ID*  
- dialog box **Edit Columns** / Lbox: *FirstName* & Rbox: -> **HeaderText** = *First Name*  
- dialog box **Edit Columns** / Lbox: *LastName* & Rbox: -> **HeaderText** = *Last Name*  
- dialog box **Edit Columns** -> click button **OK**  
- window **Designer** -> click the **frmMain** form & window **Properties** -> **Size** = *575, 235*  
- window **Designer** / *SalespersonBindingNavigator* / Rclick button **Save Data** -> click **Delete**  
Rclick button **Add new** -> click **Delete**  
Rclick button **Delete** -> click **Delete**

**CH11\_AA - import db & create dataset & create Data Form** (separate control for each table's field) instead of using **DataGridView** control &  
 & add **Try...Catch** 'safety net' statement & **KeyPress** example: **03.Course Info Solution-Data Form**

- an example with: **03.Course Info Solution-Data Form**

|                                                                            |                                                                                       |
|----------------------------------------------------------------------------|---------------------------------------------------------------------------------------|
| Your Project Name = <b>Course Info</b>                                     | ImportedDatabaseName = <b>MyCourses.mdf</b>                                           |
| YourTableName = <b>Courses dbo</b>                                         | PrimaryKeyHeaderField = <b>ID</b> OtherHeaderField = <b>Code, Title, Hours, Grade</b> |
| YourConnectionString = <b>MyCoursesConnectionString</b>                    | YourDataSetName = <b>MyCoursesDataSet.xsd</b>                                         |
| YourBoundControls = Labels & TextBoxes                                     |                                                                                       |
| YourSaveDataButtonProcedure = <b>CoursesBindingNavigatorSaveItem_Click</b> |                                                                                       |

**Step 1:** add existing database **MyCourses.mdf** to an application & create a dataset **MyCoursesDataSet** & create **Data Form** = separate control for each table's field

- |                   |                                                                                                                                                                                                                                                                                   |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |                    |
|-------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------|
| <b>step 1.1</b>   | <ul style="list-style-type: none"> <li>- Open window to manage Data Sources:</li> <li>- and Start the Data Source Configuration Wizard:</li> </ul>                                                                                                                                | <ul style="list-style-type: none"> <li>- VS / menu: <b>View / Other Windows -&gt; Data Sources Shift+Alt+D</b></li> <li>- window <b>Data Sources</b> -&gt; click <b>Add New Data Sources...</b></li> </ul>                                                                                                                                                                                                                                                                                                                                                | <b>CH11_A1.1 -</b> |
| <b>step 1.2</b>   | <ul style="list-style-type: none"> <li>- Choose a type of your data:</li> </ul>                                                                                                                                                                                                   | <ul style="list-style-type: none"> <li>- screen <b>Choose a Data Source Type</b> -&gt; <b>Database</b> &amp; click button <b>Next &gt;</b></li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                     |                    |
| <b>step 1.3</b>   | <ul style="list-style-type: none"> <li>- Determine the types of data objects your app uses:</li> </ul>                                                                                                                                                                            | <ul style="list-style-type: none"> <li>- screen <b>Choose a Database Model</b> -&gt; <b>Dataset</b> &amp; click button <b>Next &gt;</b></li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                        |                    |
| <b>step 1.4</b>   | <ul style="list-style-type: none"> <li>- Choose your database file to connect:</li> </ul>                                                                                                                                                                                         | <ul style="list-style-type: none"> <li>- screen <b>Choose Your Data Connection</b> -&gt; click button <b>New Connection...</b></li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                 |                    |
| <b>step 1.4.1</b> | <ul style="list-style-type: none"> <li>- Choose source type:</li> </ul>                                                                                                                                                                                                           | <ul style="list-style-type: none"> <li>- dialog box <b>Choose Data Source</b> -&gt; click <b>Microsoft SQL Server Database File</b> &amp; click button <b>Continue</b></li> </ul>                                                                                                                                                                                                                                                                                                                                                                         |                    |
| <b>step 1.4.2</b> | <ul style="list-style-type: none"> <li>- Verify previous step</li> </ul>                                                                                                                                                                                                          | <ul style="list-style-type: none"> <li>- dialog box <b>Add Connection</b> -&gt;</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                                                                |                    |
| <b>step 1.4.3</b> | <ul style="list-style-type: none"> <li>- Add existing database file <b>MyCourses.mdf</b> from HDD:</li> <li>- and Test your connection:</li> <li>- and Close the Add Connection dialog box:</li> <li>- Verify your database file name:</li> </ul>                                 | <ul style="list-style-type: none"> <li>-&gt; verify <b>Microsoft SQL Server Database File (SqlClient)</b></li> <li>- dialog box <b>Add Connection</b> -&gt; button <b>Browse...</b> and locate your file</li> <li>- dialog box <b>Add Connection</b> -&gt; button <b>Test Connection</b></li> <li>- dialog box <b>Add Connection</b> -&gt; button <b>OK</b></li> </ul>                                                                                                                                                                                    |                    |
| <b>step 1.5</b>   | <ul style="list-style-type: none"> <li>- and Confirm:</li> <li>- and Copy the database file to project:</li> <li>- Save the connection to your application config file:</li> </ul>                                                                                                | <ul style="list-style-type: none"> <li>- screen <b>Choose Your Data Connection / box Which data connection should your application use to connect to the database?</b> -&gt; <b>MyCourses.mdf</b></li> <li>- screen <b>Choose Your Data Connection</b> -&gt; click button <b>Next &gt;</b></li> <li>- dialog box <b>Microsoft Visual Studio</b> -&gt; click button <b>Yes</b></li> <li>- screen <b>Save the Connection String to the Application Configuration File /</b> / check box: <b>Yes, save the connection as:</b> -&gt; checked &amp;</li> </ul> |                    |
| <b>step 1.6</b>   | <ul style="list-style-type: none"> <li>- and Verify or Change the config name:</li> <li>- Choose which objects to include in dataset:<br/>(tables, fields)</li> <li>- and Verify or Change the dataset name</li> <li>- and Close the Data Source Configuration Wizard:</li> </ul> | <ul style="list-style-type: none"> <li>-&gt; <b>MyCoursesConnectionString</b> &amp; click button <b>Next &gt;</b></li> <li>- screen <b>Choose Your Database Objects / node Tables</b> -&gt;</li> </ul>                                                                                                                                                                                                                                                                                                                                                    |                    |
| <b>step 1.7</b>   | <ul style="list-style-type: none"> <li>- Verify your dataset:</li> </ul>                                                                                                                                                                                                          | <ul style="list-style-type: none"> <li>-&gt; select table <b>Courses</b></li> <li>- <b>DataSet name:</b> -&gt; <b>MyCoursesDataSet.xsd</b></li> <li>- click button <b>Finish</b></li> </ul>                                                                                                                                                                                                                                                                                                                                                               |                    |
| <b>step 1.8</b>   | <ul style="list-style-type: none"> <li>- Notice the icon before the table <b>Courses</b>'s name, for it indicates the type of control the computer will create:</li> </ul>                                                                                                        | <ul style="list-style-type: none"> <li>- window <b>Data Sources</b> / node <b>MyCoursesDataSet</b> / node <b>Courses</b> / fields: <b>ID, Code, Title, Hours, Grade</b></li> </ul>                                                                                                                                                                                                                                                                                                                                                                        |                    |
| <b>step 1.9.0</b> | <ul style="list-style-type: none"> <li>- Change default table's DataGridView to Details, for creating separate controls for each field:</li> </ul>                                                                                                                                | <ul style="list-style-type: none"> <li>- window <b>Data Sources</b> / node <b>MyCoursesDataSet</b> / node <b>Courses</b> /</li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                                     |                    |
| <b>step 1.9.1</b> | <ul style="list-style-type: none"> <li>- Set auto-numbered field <b>ID</b> to label control:</li> </ul>                                                                                                                                                                           | <ul style="list-style-type: none"> <li>- window <b>Data Sources</b> / node <b>MyCoursesDataSet</b> / node <b>Courses</b> -&gt; click list arrow: <b>Details</b></li> </ul>                                                                                                                                                                                                                                                                                                                                                                                |                    |
| <b>step 1.9.2</b> | <ul style="list-style-type: none"> <li>- Verify appropriate type of control for table:</li> </ul>                                                                                                                                                                                 | <ul style="list-style-type: none"> <li>- ... / node <b>MyCoursesDataSet</b> / node <b>Courses</b> / field <b>ID</b> -&gt; click list arrow: <b>Label</b></li> </ul>                                                                                                                                                                                                                                                                                                                                                                                       |                    |
| <b>step 1.9.3</b> | <ul style="list-style-type: none"> <li>- Verify appropriate type of control for <b>ID</b> field:</li> <li>- and Verify appropriate type of control for other fields:</li> </ul>                                                                                                   | <ul style="list-style-type: none"> <li>- ... / node <b>MyCoursesDataSet</b> / node <b>Courses</b> / field <b>ID</b> -&gt; icon: <b>Details</b></li> <li>- ... / node <b>MyCoursesDataSet</b> / node <b>Courses</b> / field <b>ID</b> -&gt; icon: <b>Label</b></li> <li>- ... / node <b>MyCoursesDataSet</b> / node <b>Courses</b> / field <b>Code, Title, Hours, Grade</b> -&gt; icon: <b>TextBox</b></li> </ul>                                                                                                                                          |                    |

**Step 2: create & modify GUI as desired & set to use the most recent copy of the database file: property **Copy to Output Directory****

**CH11\_A1.1 -**

**step 2.1** - Drag the **Courses** table to the form:

- ... / node **MyCoursesDataSet** / table **Courses** -> drag to window: **Designer**

**step 2.2** - and **Notice** controls added to the form:

- window **Designer** / **Course Information**

**step 2.3** - **Modify** GUI as desired:

- change appearance, add access keys, change the TextBoxes (**Name**)s, set Tab order, etc...

**step 2.4** - Set to use the most recent copy of the database file:

- window **Solution Explorer** / choose **MyCourses.mdf** &

& window **Properties** / **Copy to Output Directory** -> = **Copy if newer**

- just **Test** the application if the records are shown: - **F5** key

**Step 3: add the Try...Catch 'safety net' statement to BindingNavigator's button Save Data & add the KeyPress procedure & test your app**

**CH11\_A1.1 -**

**step 3.0** - See automatically generated code when binding:

- window **Code Editor**

**step 3.1** - Include a **Try...Catch** statement into procedure code for

Binding Navigator control's button Save Data &  
appropriate **MessageBoxes**:

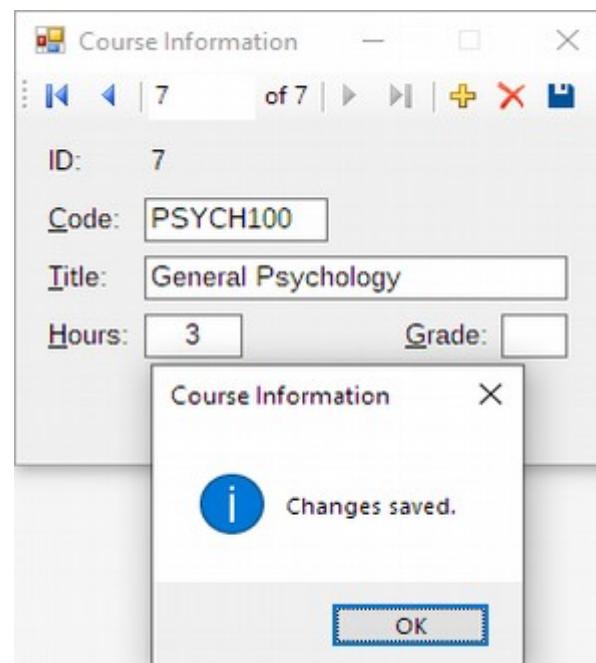
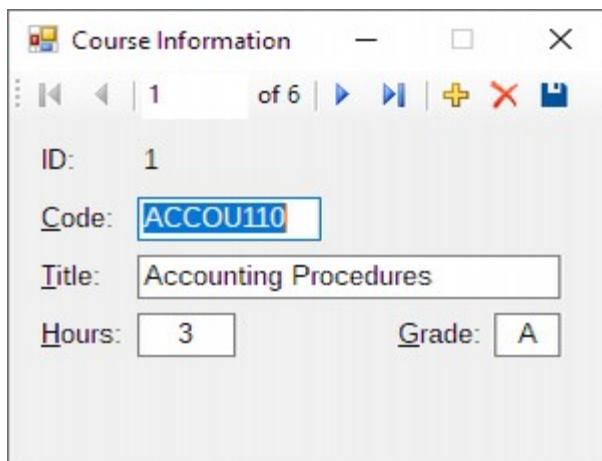
- window **Code Editor** / procedure **CoursesBindingNavigatorSaveItem\_Click**

**step 3.2** - **Try...Catch** syntax in: **CH11\_F8.1 -**

- window **Code Editor** / procedure **txtHours\_KeyPress**

**step 4.0** - Allow the TextBox **Hours** to accept only numbers:

- **F5** key



**CH11\_AB - already:** imported db, created dataset & **Bind** table's field objects to **existing** Label controls by **dragging** them &

& add **BindingNavigator** control example: **04.Course Info Solution-Labels**

- an example with: **04.Course Info Solution-Labels**

|                                                       |                                                                                                               |                                                         |
|-------------------------------------------------------|---------------------------------------------------------------------------------------------------------------|---------------------------------------------------------|
| Your Project Name = <i>Course Info</i>                | ImportedDatabaseName = <i>MyCourses.mdf</i>                                                                   | TableName = <i>Courses dbo</i>                          |
| TableFields = <i>ID, Code, Title, Hours, Grade</i>    | UsedTableFields = <i>Code, Grade</i>                                                                          | AlreadyCreatedDataSetName = <i>MyCoursesDataSet.xsd</i> |
| AlreadyCreatedControls = <i>(LblCode), (LblGrade)</i> | YourBoundControls = TableField <i>Code</i> -> <i>(LblCode)</i> , TableField <i>Grade</i> -> <i>(LblGrade)</i> |                                                         |

**step 1** - Verify already created dataset:

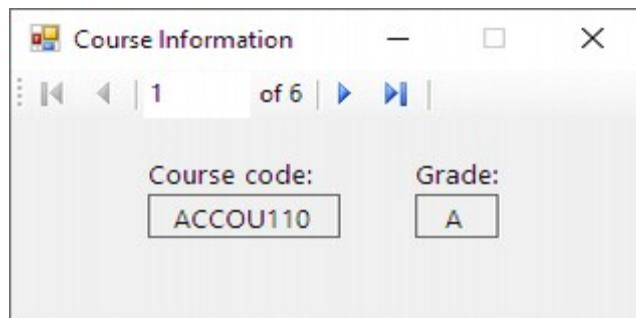
**step 2** - Drag the field object to the Label control:  
- and **Notice** there is no **BindingNavigator** control

**step 3.1** - Add BindingNavigator control to the form:  
- and **Notice** there is no button **Save Data**

**step 3.2** - Set the source for BindingNavigator control:  
- window **Designer** -> choose the **BindingNavigator1** control &  
& window **Properties** / **BindingSource** -> = *CoursesBindingSource*

**step 3.3** - Disallow user to add new & delete records,  
since app will only display info:  
- window **Designer** / **BindingNavigator1** / Rclick button **Delete** -> click **Delete**  
- window **Designer** / **BindingNavigator1** / Rclick button **Add new** -> click **Delete**

**step 4** - Save, start, test



**CH11\_AC - already:** imported db, created dataset, created DataGridView bound control & **Code** the **btnCalc\_Click** procedure to:

**Loop** all the rows/records and accumulate numbers if the field is not empty nor contains letter W example: **05.Course Info Solution-Total Hours**

- an example with: **05.Course Info Solution-Total Hours**

|                                                       |                                                                   |                                                  |
|-------------------------------------------------------|-------------------------------------------------------------------|--------------------------------------------------|
| Your Project Name = <i>Course Info</i>                | ImportedDatabaseName = <i>MyCourses.mdf</i>                       | TableName = <i>Courses dbo</i>                   |
| TableFields = <i>ID, Code, Title, Hours, Grade</i>    | AlreadyCreatedDataSetName = <i>MyCoursesDataSet.xsd</i>           | AlreadyCreatedBoundControl = <i>DataGridView</i> |
| UsedLoopInCode = <b>For Each...Next</b>               | NameOfMyLoop = <i>row</i>                                         | UsedClass = <i>CoursesRow</i>                    |
| MyLoopDeclaredAs = <i>MyCoursesDataSet.CoursesRow</i> | MyLoopDeclaredInCollection = <i>MyCoursesDataSet.Courses.Rows</i> |                                                  |
| UsedTableFieldsInCode = <i>row.Hours, row.Code</i>    | AlreadyCreatedAndUsedMethodInS.S. = <i>row.IsDBNull</i>           |                                                  |

VS: **Function** *MyCoursesDataSet.CoursesRow.IsDBNull()* **As Boolean**

- step 1** - Start the app to see wocogo  
**step 2** - Locate `btnCalc_Click` procedure:  
 - and Create pseudocode

- step 2.1** - Declare variable used as accumulator:

- step 2.2** - Access each row/record in dataset via loop:

- step 2.3** - Condition each row/record:

- and **Specify** if True path:

- and **Close** condition

- step 2.2** - and **Close** the loop:

- step 2.2** - Generic statement:

- step 2.4** - Display accumulator variable in Label:

- step 3** - Save, start, and test

- window **Code Editor** / `Private Sub btnCalc_Click...`

```
Dim intTotal As Integer
For Each row As MyCoursesDataSet.CoursesRow In MyCoursesDataSet.Courses.Rows
    If row.IsDBNull = False AndAlso row.Grade <> "W" Then
        intTotal += row.Hours
    End If
Next row
```

```
For Each NameOfMyLoop As dataset.class In dataset.table.RowsCollection
    ...some conditions
```

```
Next NameOfMyLoop
```

```
lblTotal.Text = intTotal.ToString
```

```
10 Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
11     ' Display the total number of credit hours completed.
12     Dim intTotal As Integer
13
14     For Each row As MyCoursesDataSet.CoursesRow In MyCoursesDataSet.Courses.Rows
15         If row.IsDBNull = False AndAlso row.Grade <> "W" Then
16             intTotal += row.Hours
17         End If
18     Next row
19     lblTotal.Text = intTotal.ToString
20 End Sub
```

**step 2.1**

**step 2.2**

**step 2.3**

**step 2.3**

**step 2.3**

**step 2.2**

**step 2.4**

| ID | Code     | Title                       | Hours | Grade |
|----|----------|-----------------------------|-------|-------|
| 1  | ACCOU110 | Accounting Procedures       | 3     | A     |
| 2  | ENG101   | English Composition         | 3     | W     |
| 3  | CIS110   | Introduction to Programming | 3     | A     |
| 4  | BIO111   | Environmental Biology       | 3     | C     |
| 5  | ENG101   | English Composition         | 3     | B     |
| 6  | CIS112   | The Internet                | 2     | A     |
| 7  | EARTH110 | Geology                     | 4     | C     |
| 8  | EARTH112 | Astronomy                   | 3     | B     |
| 9  | ENG113   | Literature                  | 3     | B     |

Total hours completed: **40**

**Calculate** **Exit**

## CH11\_FOCUS ON THE CONCEPTS LESSON

### CH11\_F0 - SQL Server Database: basic info & terminology & e.g. single-table db, two-table db

- in order to maintain accurate records, most businesses store their employee, customer, and inventory information in **computer databases**
- most computer databases are created and manipulated using **relational database / RDBMSs**, for many advantages:
  - the computer can retrieve data stored in a relational format both quickly and easily, and the data can be displayed in any order
  - Figure 11-2** - the information can be arranged by the salesperson's Id, first name, last name, or country
  - you also can control the amount of information you want to view
  - Figure 11-2** - you can view all of the information in the database, or only the information pertaining to a certain salesperson, etc...

-> a **computer database**: = an electronic file that contains an organized collection of related information

- most **db**s are created and manipulated using **RDBMSs** -> relational database management systems

-> **.mdf** extension: = filename extension is commonly used when naming **SQL Server databases**  
= stands for: **master database file**

-> **RDBMSs**: = Relational DataBase Management Systems

- the databases are called **relational databases** because the information in them can be related in different ways
- some of the most popular **RDBMSs**:
  - Microsoft **SQL Server** - used in this chapter
  - Oracle
  - IBM DB2
- can contain multiple tables

**Figure 11-1** - the information in a **relational database** stored in a **single-table**:

**Figure 11-2** - the information in a **relational database** stored in a **two-tables**:

-> a **table**: = composition of columns and rows, similar to the format used in a spreadsheet; = group of related **records**

- most tables have a **primary key** and a **foreign key**
- in a case of multiple tables, those can be divided into a **parent table** and a **child table**
- when defining the table, you will need to provide both:
  - 1). **name** of each field
  - 2). **data type** of each field, indicating the type of data the field will store

-> a **parent table**: = a table that contains a **primary key** matching the **foreign key** from another table

**Figure 11-2** Location table

-> a **child table**: = a table that contains the **foreign key**

**Figure 11-2** Salesperson table

-> a **one-to-many relationship**: = a relationship between **parent table** and **child table**

**Figure 11-2** each **CountryCode** can appear only once in the **parent table - Location table**, but can appear many times in the **child table - SalesPerson table**

-> a **record**: = group/combination of related fields creates a record

**Figure 11-1** the table contains 6 records, each composed of 5 fields

-> a **field**: = single item of information about a person, place, or thing

**Figure 11-1** the 5 fields contain information related to each college course

-> a **primary key**: = field that uniquely identifies each record

**Figure 11-1** Courses table, field: ID

**Figure 11-2** Location table, field: CountryCode

-> a **foreign key**: = field that typically refers to a **primary key** in another table, creating a relationship between the two tables

= can also refer to any other key that uniquely identifies each row in another table

**Figure 11-2** Salesperson table, field: CountryCode

- e.g.1
- an example of a **single-table** relational database: a **Courses table**: **Figure 11-1**
  - keeps track of a student's college courses
  - each column represents a **field**, and each row represents a **record**
  - the table contains 6 records, each composed of 5 fields
  - the 5 fields contain information related to each college course
  - the **primary key** = the **ID** field

e.g.1 single-table database: **Courses table**

- 5 fields, 6 records
- **primary key** = **ID** field (1 - 6)

| ID | Code     | Title                       | Hours | Grade |
|----|----------|-----------------------------|-------|-------|
| 1  | ACCOU110 | Accounting Procedures       | 3     | A     |
| 2  | ENG101   | English Composition         | 3     | W     |
| 3  | CIS110   | Introduction to Programming | 3     | A     |
| 4  | BIO111   | Environmental Biology       | 3     | C     |
| 5  | ENG101   | English Composition         | 3     | B     |
| 6  | CIS112   | The Internet                | 2     | A     |

Figure 11-1

e.g.2 two-table relational database: **Salesperson table** and **Location table**

- **Salesperson table** = child table, **foreign key** = **CountryCode** field
- **Location table** = parent table, **primary key** = **CountryCode** field

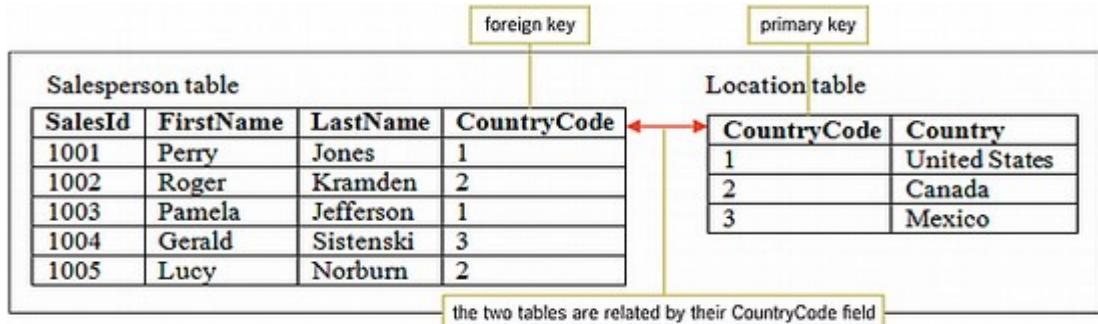


Figure 11-2

- e.g.2
- an example of a **two-table** relational database: **Figure 11-2**
  - a **Salesperson table**:
    - contains: ID, first name, last name, and a numeric code (1, 2, or 3) that represents the country in which the salesperson is located
    - the meaning of each code can be found in the **Location table**
    - the **primary key** = **SalesId** field
    - a **foreign key** = notice that the table also has a **CountryCode** field, referring to a primary key in the **Location table**, both keys create a relationship between the two tables
  - a **Location table**:
    - contains: Country code and Country: 1 for the United States, 2 for Canada, and 3 for Mexico
    - the **primary key** = **CountryCode** field
  - storing the country codes in a separate table has the following advantages:
    - it allows the input clerk to enter a number, rather than the country name, in each record in the Salesperson table, resulting in less chance of a typing error
    - also, the Salesperson table will require less of the computer's main memory to store a number in each record rather than storing the country name
  - more info in: **CH11\_F9 - explore the Two-table database: basic info with example 02.Charleston Sales Solution**

### Mini-Quiz 11-1

1. What is a field?
  2. What is a group of related fields called?
  3. What is a group of related records called?
  4. What is a primary key?
1. a single item of information about a person, place, or thing...
  2. a record
  3. a table
  4. a field that uniquely identifies each record in a table

## CH11 F0.1 - SQL Server data types: comparison with Visual Basic data types basic info: bit, decimal(p, s), float, int, char(n), varchar(n)

- when defining the table, you will need to provide both:

- 1). **name** of each field
- 2). **data type** of each field, indicating the type of data the field will store

e.g. CH11\_F2 - step 2/8: add a **Table** to a **Database** and define it: step-by-step example 01.Course Info Solution

step 2.6

- comparison of Visual Basic & SQL Server data types:

| Visual Basic: |     | SQL:          |
|---------------|-----|---------------|
| Boolean       | bln | bit           |
| Decimal       | dec | decimal(p, s) |
| Double        | dbl | float         |
| Integer       | int | int           |
| String        | str | char(n)       |
|               |     | varchar(n)    |

- used for financial purposes - same like with VB  
**p** = precision argument, an integer  
- represents the total number of digits that will be stored  
**s** = scale argument, an integer  
- represents the total number of digits that will be stored to the right of the decimal point  
e.g. decimal(5, 2) = data type will store a number with 5 digits:  
- 3 digits to the left of the decimal point, and  
- 2 digits to the right of the decimal point  
= 123. 45

- used to store fixed-length strings  
- used to store variable-length strings  
(n) = an integer that represents the max number of characters the field will store

## CH11\_F1 - step 1/8: create a SQL Server Database with .mdf extension: step-by-step example 01.Course Info Solution

- in this section, you will learn how to create a **SQL Server database** named **MyCourses.mdf**
- > **.mdf** extension:
  - = filename extension is commonly used when naming **SQL Server databases**
  - = stands for: **master database file**
- the database will contain the **Courses table** shown earlier

- to create a SQL Server database example: **01.Course Info Solution**

1). open the: ...VB2017\Chap11\Exercise01.Course Info Solution\Course Info Solution.sln

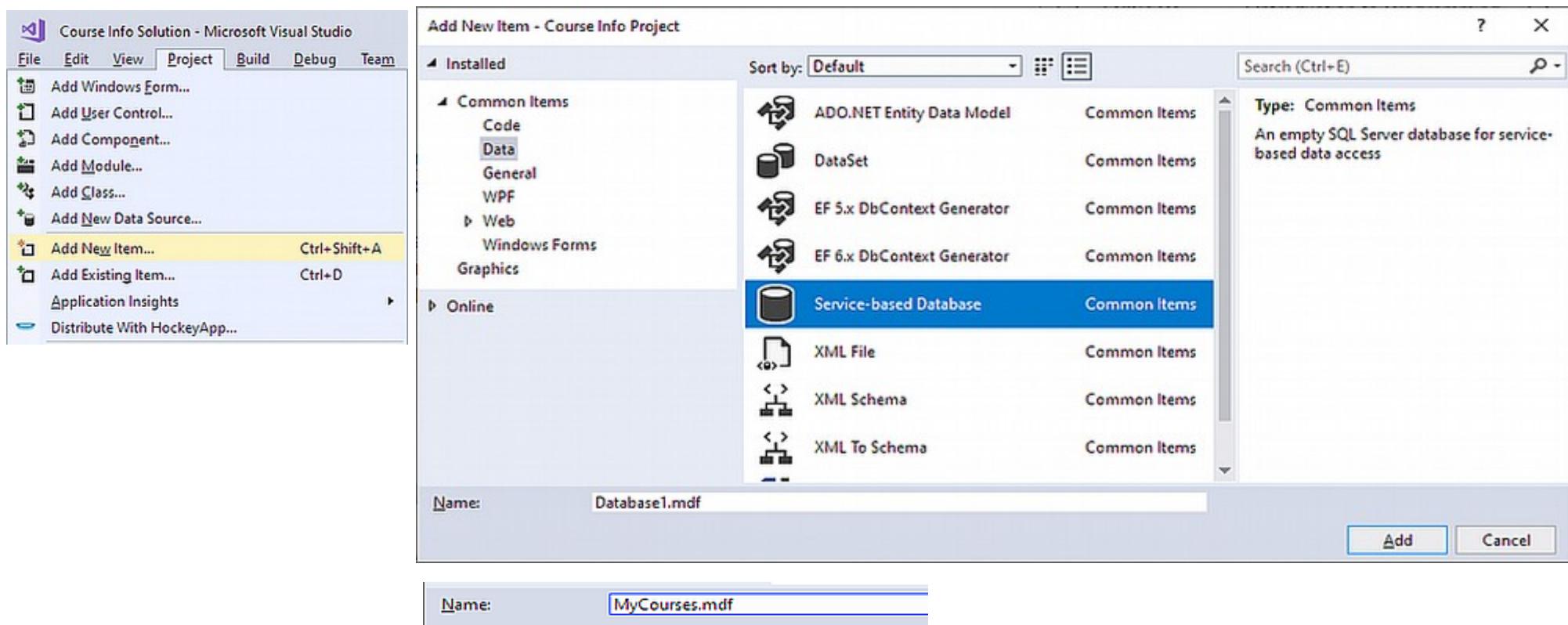
2). open the designer window and:

step 1.1 -> on the menu bar click **Project** and then click **Add New Item...** **Ctrl+Shift+A**

step 1.2 -> on the left click: **Installed \ Common Items \ Data**

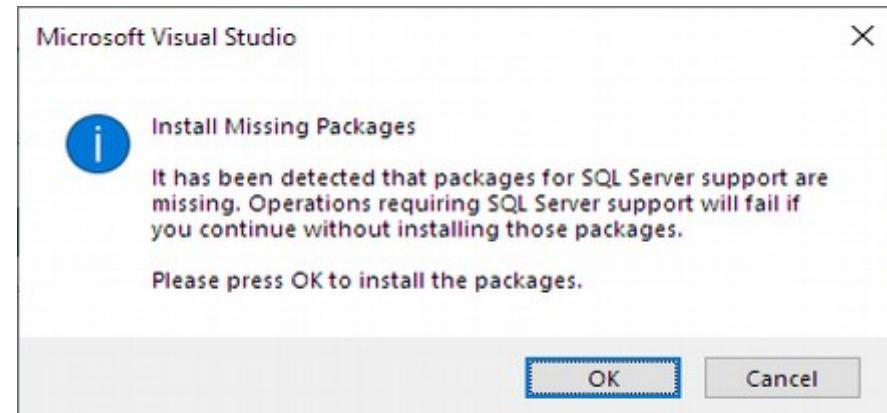
step 1.3 -> in the middle choose: **Service-based Database**, with original Name: **Database1.mdf**

step 1.4 -> change the name from **Database1.mdf** to **MyCourses.mdf**

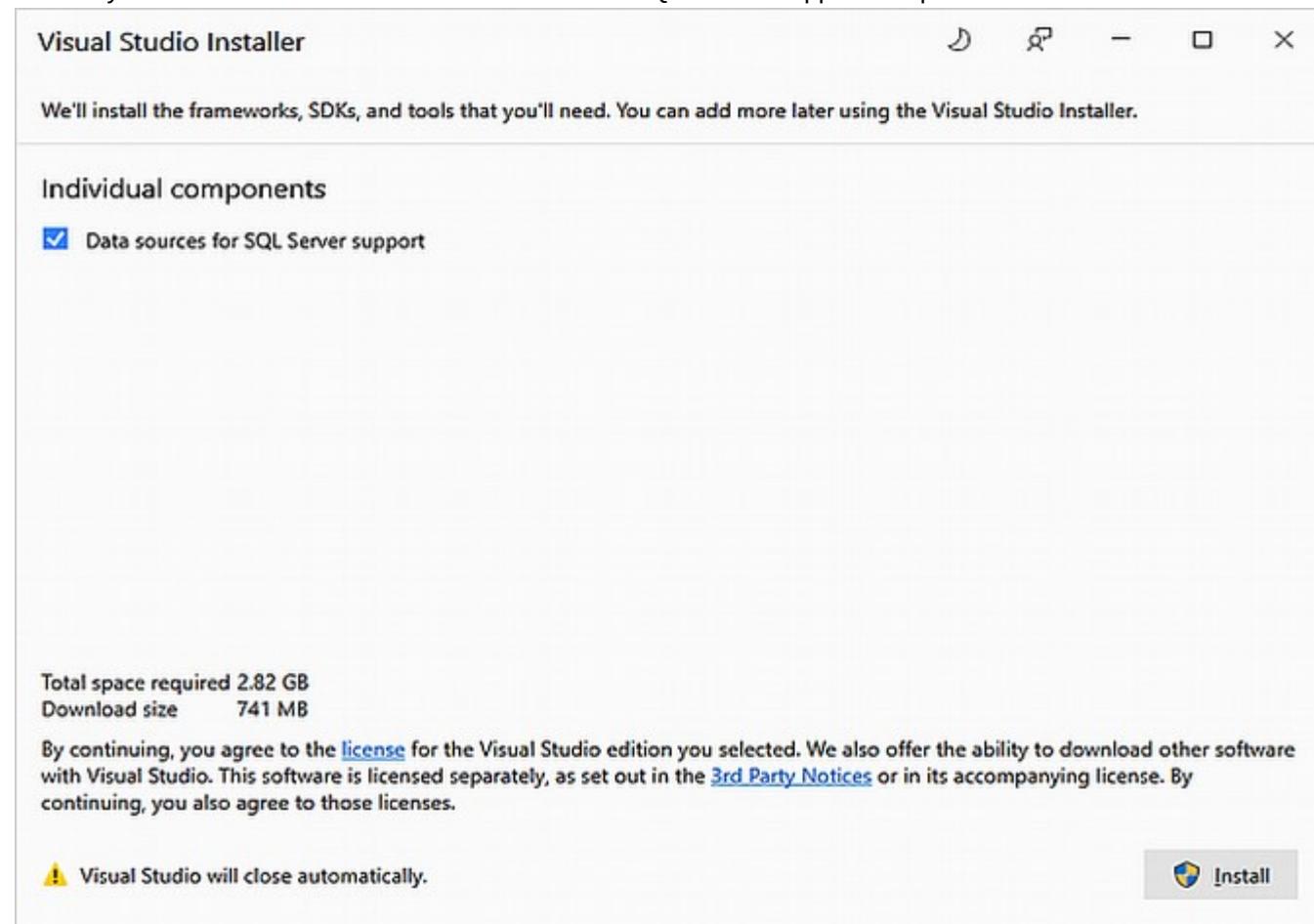


**step 1.5** -> click the button **Add** ->

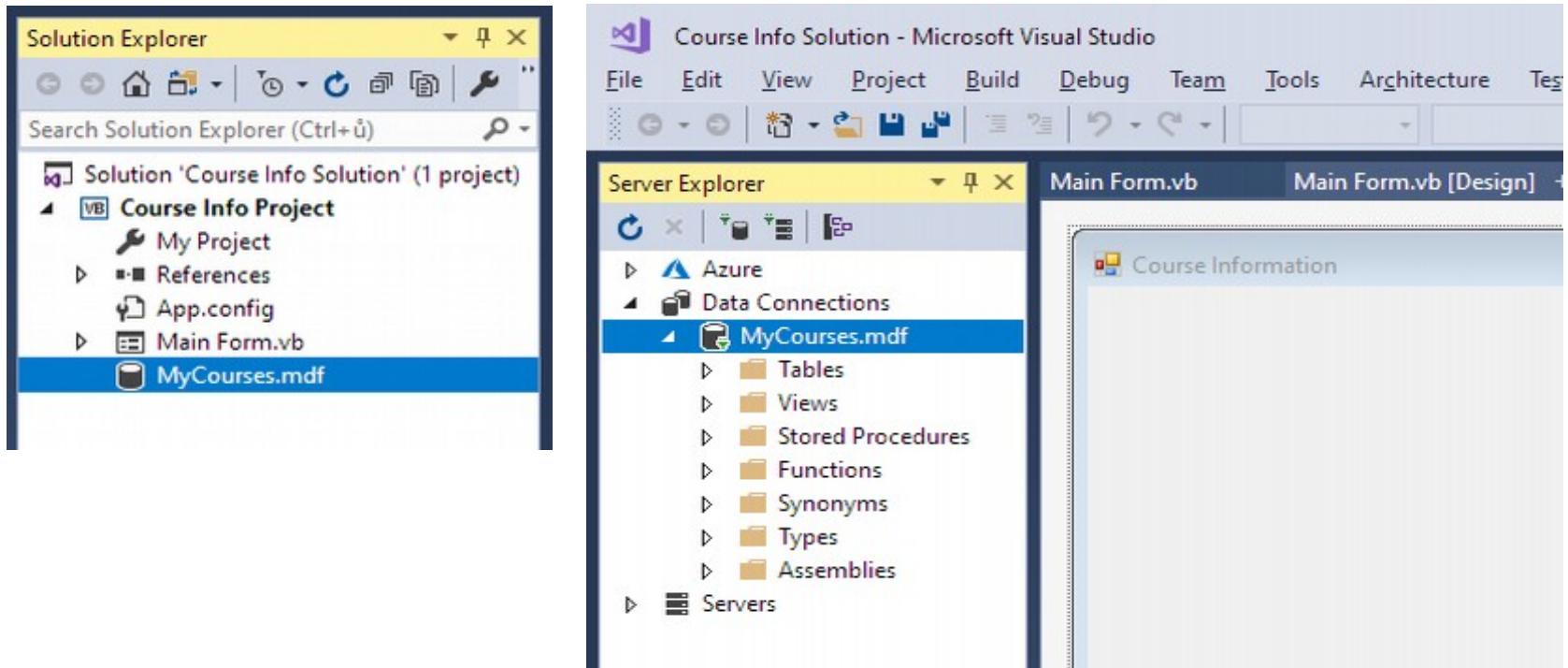
- if a message box appears alerting you that some packages are missing, click **OK** button and then **Yes** button to allow changes to your device



-> then, click the **Install** button in the Visual Studio Community 2017 window to install the Data sources for SQL Server support component



**step 1.6** - notice in **Solution Explorer** window, that the **MyCourses.mdf** database file is saved in the **Course Info Project** folder



**step 1.7** -> in the **Solution Explorer** window, right-click the **MyCourses.mdf** file and then click **Open** to permanently display the **Server Explorer** window

- > expand the nodes: **Data Connections \ MyCourses.mdf**
- do not be concerned if you are not connected to Azure

### Mini-Quiz 11-2

1. What SQL data type is equivalent to Visual Basic's **Double** data type?
2. Which of the following SQL data types can store the number **2316.26**?
  - a. decimal(2,4)
  - b. decimal(2,6)
  - c. decimal(4,2)
  - d. decimal(6,2)
3. What SQL data type is equivalent to the **Integer** data type in Visual Basic?

3. int

2. d - decimal(6,2)

1. float

## CH11\_F2 - step 2/8: add a Table to a Database and define it: step-by-step example 01.Course Info Solution

- in this section, you will add the **Courses** table to the **MyCourses.mdf** database

- to add a table to the database example: **01.Course Info Solution**

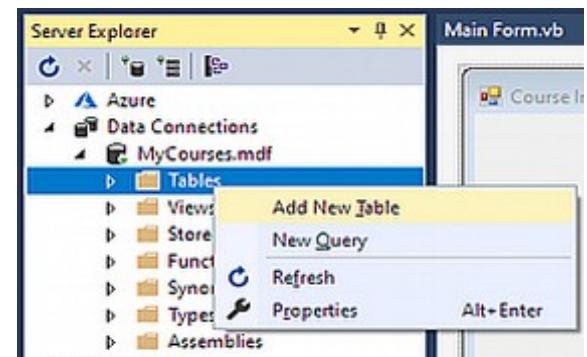
1). open the: ...VB2017\Chap11\Exercise\01.Course Info Solution\Course Info Solution.sln

2). open the designer window and notice the **Server Explorer** window:

**step 2.1** -> in **Server Explorer** window, right-click **Tables** and then click **Add New Table** **Figure 11-6**

- the database table designer window opens

- when the design surface has finished loading, you will see a window like in **Figure 11-7**



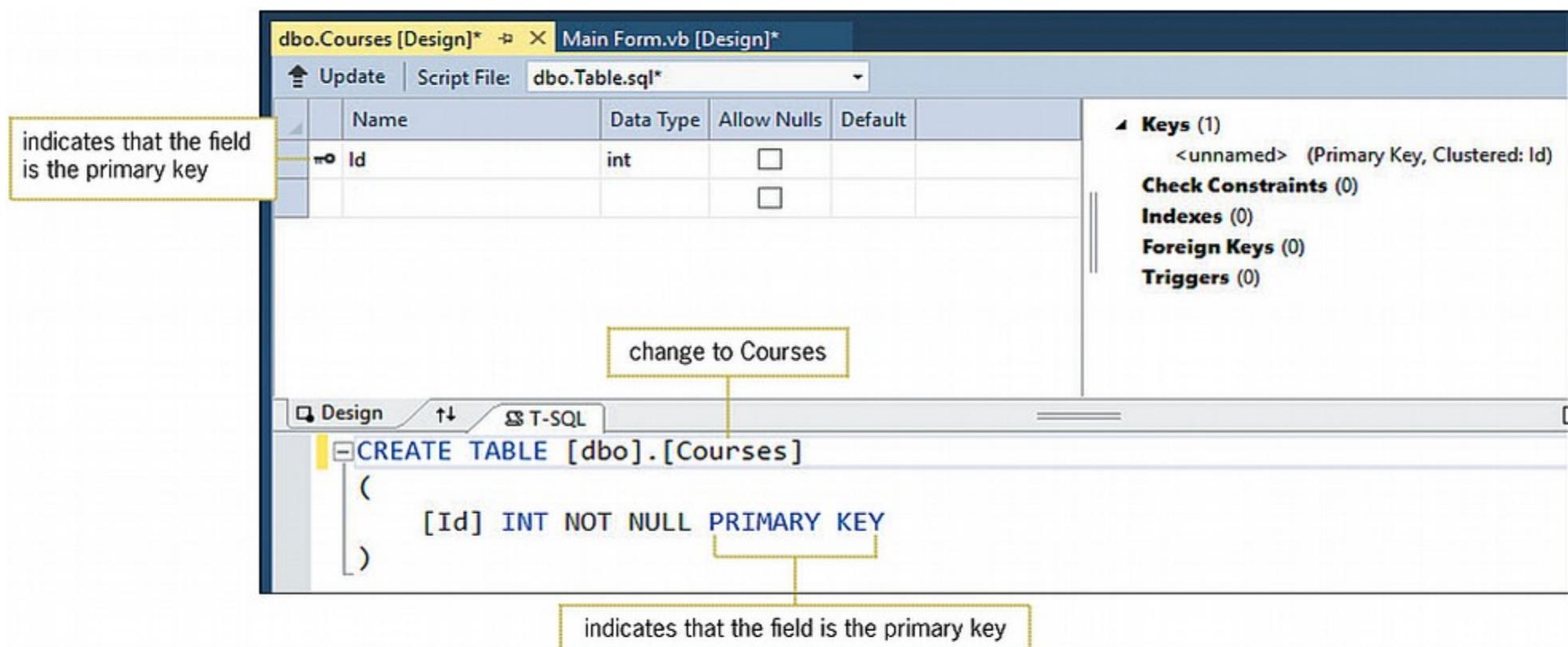
**step 2.2** -> in the **T-SQL** pane change the line: - from: `CREATE TABLE [dbo].[Table]` **Figure 11-7**

- to: `CREATE TABLE [dbo].[Courses]`

- the key that appears next to **Id** in the **Name** column indicates that the **Id** field is the **primary key**

- the keywords **PRIMARY KEY** also appear in the **T-SQL** pane

Figure 11-6



|    | Name | Data Type | Allow Nulls              | Default |  |
|----|------|-----------|--------------------------|---------|--|
| Id |      | int       | <input type="checkbox"/> |         |  |
|    |      |           | <input type="checkbox"/> |         |  |

Keys (1)  
<unnamed> (Primary Key, Clustered: Id)

Check Constraints (0)

Indexes (0)

Foreign Keys (0)

Triggers (0)

change to Courses

```
CREATE TABLE [dbo].[Courses]
(
    [Id] INT NOT NULL PRIMARY KEY
)
```

indicates that the field is the primary key

**Figure 11-7** Database table designer window

**step 2.3** - you will change the field name from **id** to **ID**

**Figure 11-7**

-> in the Name column, change **Id** to **ID**

- notice, that in **T-SQL** pane an automatic change from has been made from **[Id]** to **[ID]**

**step 2.4** - you will have a database automatically number the field whenever a new record is added to the table

**Figure 11-8**

-> select newly named **ID** column and look at its **Properties window**

-> expand the **Identity Specification** property

-> change **(Is Identity) = False**, to **True**

- the keyword **IDENTITY** also appear in the **T-SQL** pane: **[ID] INT NOT NULL PRIMARY KEY IDENTITY**

- the **Identity Seed** property indicates the starting number for the first record's ID

-> in this case, the starting number will be **1**

- the **Identity Increment** property indicates the number by which each subsequent **ID** will be increased

-> in this case, each subsequent **ID** will be **1** number more than the previous **ID**

**step 2.5** - the **Data Type = int** property indicates the data type of **ID** field

**Figure 11-8**

-> the **ID** field will contain integers, so its data type should be left at the default value - **int**

- a field that is a **primary key** should never be empty, therefore:

-> the property **Allow Nulls = False** <- you can change it also by a checkbox

The screenshot shows the SQL Server Management Studio interface. On the left, the 'dbo.Courses [Design]' window displays a table structure with one column 'ID'. The 'T-SQL' tab shows the CREATE TABLE statement. A callout box labeled 'IDENTITY keyword' points to the 'IDENTITY' keyword in the T-SQL code. On the right, the 'Properties' window is open for the 'ID' column. The 'Data Type' is set to 'int'. The 'Allow Nulls' property is checked (False). The 'Identity Specification' section is expanded, showing 'Is Identity' is checked (True), 'Identity Increment' is 1, and 'Identity Seed' is 1. A callout box points to these properties with the text: 'use these properties to change the increment and the starting number'.

**Figure 11-8** Properties and designer windows

**step 2.6**

- you will add fields to the table, set their Data Types, and choose if the field can be empty: **Code, Title, Hours, Grade**
- you will type another field **Name**, then press a Tab key and choose appropriate **Data Type** - change the number if needed, then manage chk **Allow Nulls**
- > **Name = Code, Data Type = varchar(8), Allow Nulls = unchecked**
  - the **Course Codes** will contain from 6 to 8 characters, so in the Data Type choose: **varchar(50)** and change **50** to **8**
  - each record should always have a course code, therefore the check box **Allow Nulls = unchecked**
- > **Name = Title, Data Type = varchar(40), Allow Nulls = unchecked**
  - the **Title** field will contain strings of variable lengths, so you will use **varchar(40)** as the datatype by choosing **varchar(50)** and changing **50** to **40**
  - this data type will allow the field to store up to 40 characters, which should be more than enough for a course title
  - the **Course Title** should always be completed in each record, therefore the check box **Allow Nulls = unchecked**
- > **Name = Hours, Data Type = int, Allow Nulls = unchecked**
  - the **Hours** field will contain an integer and should never contain a null value, therefore the check box **Allow Nulls = unchecked**
- > **Name = Grade, Data Type = char(1), Allow Nulls = checked**
  - the Grade field will contain 1 character, so you will use **char(1)** as its data type, by choosing **char(10)** and changing the number **10** to **1**
  - you will allow nulls in this field in case the student wants to add the course to the table before receiving a grade, therefore **Allow Nulls = checked**

The screenshot shows the 'dbo.Courses [Design]' tab in SSMS. A table structure is being defined with columns: Name, Data Type, Allow Nulls, and Default. The 'Code' column is currently selected, showing 'nchar(10)' in the Data Type field and a checked checkbox for Allow Nulls. A dropdown menu is open over the Data Type field, with 'varchar(50)' highlighted. The 'Default' field for the 'Code' column is empty. The 'Name' column has 'ID' as its name and 'int' as its data type. The 'Title', 'Hours', and 'Grade' columns are not yet defined.

**Courses table:**

| ID | Code     | Title                       | Hours | Grade |
|----|----------|-----------------------------|-------|-------|
| 1  | ACCOU110 | Accounting Procedures       | 3     | A     |
| 2  | ENG101   | English Composition         | 3     | W     |
| 3  | CIS110   | Introduction to Programming | 3     | A     |
| 4  | BIO111   | Environmental Biology       | 3     | C     |
| 5  | ENG101   | English Composition         | 3     | B     |
| 6  | CIS112   | The Internet                | 2     | A     |

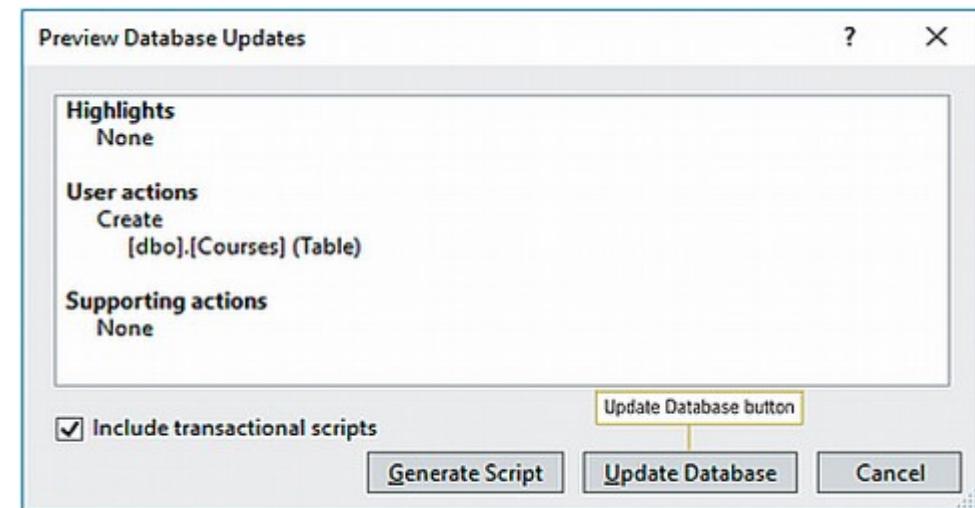
The screenshot shows the 'dbo.Courses [Design]' tab in SSMS. The table structure is now fully defined with all four columns: ID, Code, Title, Hours, and Grade. The 'Code' column has 'varchar(8)' as its data type and 'Allow Nulls' unchecked. The 'Title' column has 'varchar(40)' as its data type and 'Allow Nulls' unchecked. The 'Hours' column has 'int' as its data type and 'Allow Nulls' unchecked. The 'Grade' column has 'char(1)' as its data type and 'Allow Nulls' checked. The 'Name' column is still 'ID' with 'int' as its data type.

**step 2.7** -> click the **Update** button, which appears below the **dbo.Courses [Design]** tab

- dialog box opens: **Preview Database Update**

**Figure 11-10**

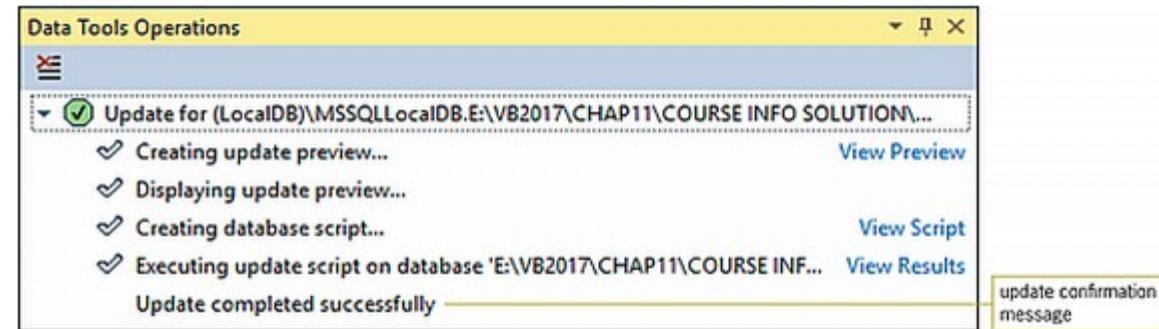
-> click the **Update Database** button



**Figure 11-10** Preview Database Updates dialog box

- the "Update completed successfully" message appears in the Data Tools Operations window

**Figure 11-11**



**Figure 11-11** Data Tools Operations window

-> close the **Data Tools Operations** window

## CH11\_F3 - step 3/8: add Records to a Table: step-by-step example 01.Course Info Solution

- now that you have finished defining the **Courses** table, you can add **records** to it

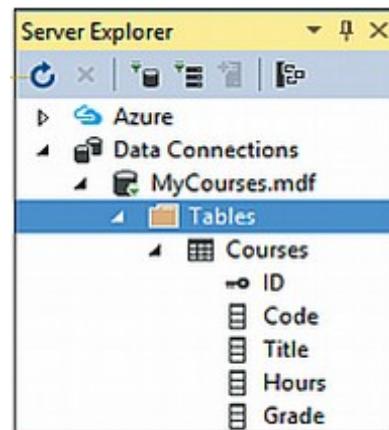
- to add **Records** to a table example: **01.Course Info Solution**

1). open the: ...VB2017\Chap11\Exercise\01.Course Info Solution\Course Info Solution.sln

2). open the designer window and notice the **Server Explorer** window:

**step 3.1** -> in **Server Explorer** window, expand the nodes: **Tables / Courses**

- if the **Courses** node does not appear below the **Tables** node, click the **Server Explorer** window's Refresh button



**Figure 11-12**

**Figure 11-12** Courses table in the

**step 3.2** -> right-click **Courses** and then click **Show Table Data**

- new tab opens: **dbo.Courses [Data]**, with a fields: **ID, Code, Title, Hours, Grade**

A screenshot of the Microsoft Visual Studio IDE showing multiple windows. On the left, the Server Explorer window shows a connection to 'Azure' and a local database 'MyCourses.mdf' with a 'Tables' node. A context menu is open over the 'Courses' table, with 'Show Table Data' selected. In the center, the 'dbo.Courses [Design]' window shows the table structure with columns: Name, ID, and Code. On the right, the 'dbo.Courses [Data]' window shows the table data with one row: ID is NULL, Code is NULL, Title is NULL, Hours is NULL, and Grade is NULL. Other tabs visible include 'Main Form.vb' and 'Main Form.vb [Design]'.

|   | ID   | Code | Title | Hours | Grade |
|---|------|------|-------|-------|-------|
| * | NULL | NULL | NULL  | NULL  | NULL  |

**step 3.3**

- the first record you will add is for the **Accounting Procedures** course
- you do not need to enter the course's **ID** because the database will automatically enter it each time a record is added to the table ->
- as you defined in: **step 2.4** Properties: **ID Column \ Identity Specifications \ (Is Identity) = True; Identity Seed = 1; Identity Increment = 1**
- > in a **Code** field click **NULL**, type: **ACCOU110**, and then press Tab key
- the circle with the exclamation point inside will disappear when you press Tab key after completing the record

| Code     | Title |                                                                                                                                                                                 |       |
|----------|-------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-------|
| ACCOU110 | NULL  |                                                                                                                                                                                 |       |
| Code     | Title | Hours                                                                                                                                                                           | Grade |
| ACCOU110 | NULL  | The data in this cell has changed.<br>The change has not been committed to the database.<br>The original value was NULL.<br>Press Esc to revert the cell to its original value. |       |

- > in a **Title** field type: **Accounting Procedures**, and press Tab key
- > in a **Hours** field type: **3**, and press Tab key
- > in a **Grade** field type: **A**, and press Tab key
- notice that the number **1** now appears in this record's **ID** field
- > enter additional 5 records as shown in: **Figure 11-13**

you can use the vertical line to adjust the Title column's width

| dbo.Courses [Data] |      | dbo.Courses [Design] |                             | Main Form.vb [Design] |       |  |
|--------------------|------|----------------------|-----------------------------|-----------------------|-------|--|
|                    | ID   | Code                 | Title                       | Hours                 | Grade |  |
|                    | 1    | ACCOU110             | Accounting Procedures       | 3                     | A     |  |
|                    | 2    | ENG101               | English Composition         | 3                     | W     |  |
|                    | 3    | CIS110               | Introduction to Programming | 3                     | A     |  |
|                    | 4    | BIO111               | Environmental Biology       | 3                     | C     |  |
|                    | 5    | ENG101               | English Composition         | 3                     | B     |  |
|                    | 6    | CIS112               | The Internet                | 2                     | A     |  |
| ▶*                 | NULL | NULL                 | NULL                        | NULL                  | NULL  |  |

**Figure 11-13** Records entered in the table

**step 3.4** -> save the solution and then close the windows: **dbo.Courses [Data]** and **dbo.Courses [Design]**

**CH11\_F4 - step 4/8: create Dataset = connection between Database and Application - using Data Source Configuration Wizard & & view contents of your dataset: example 01.Course Info Solution**

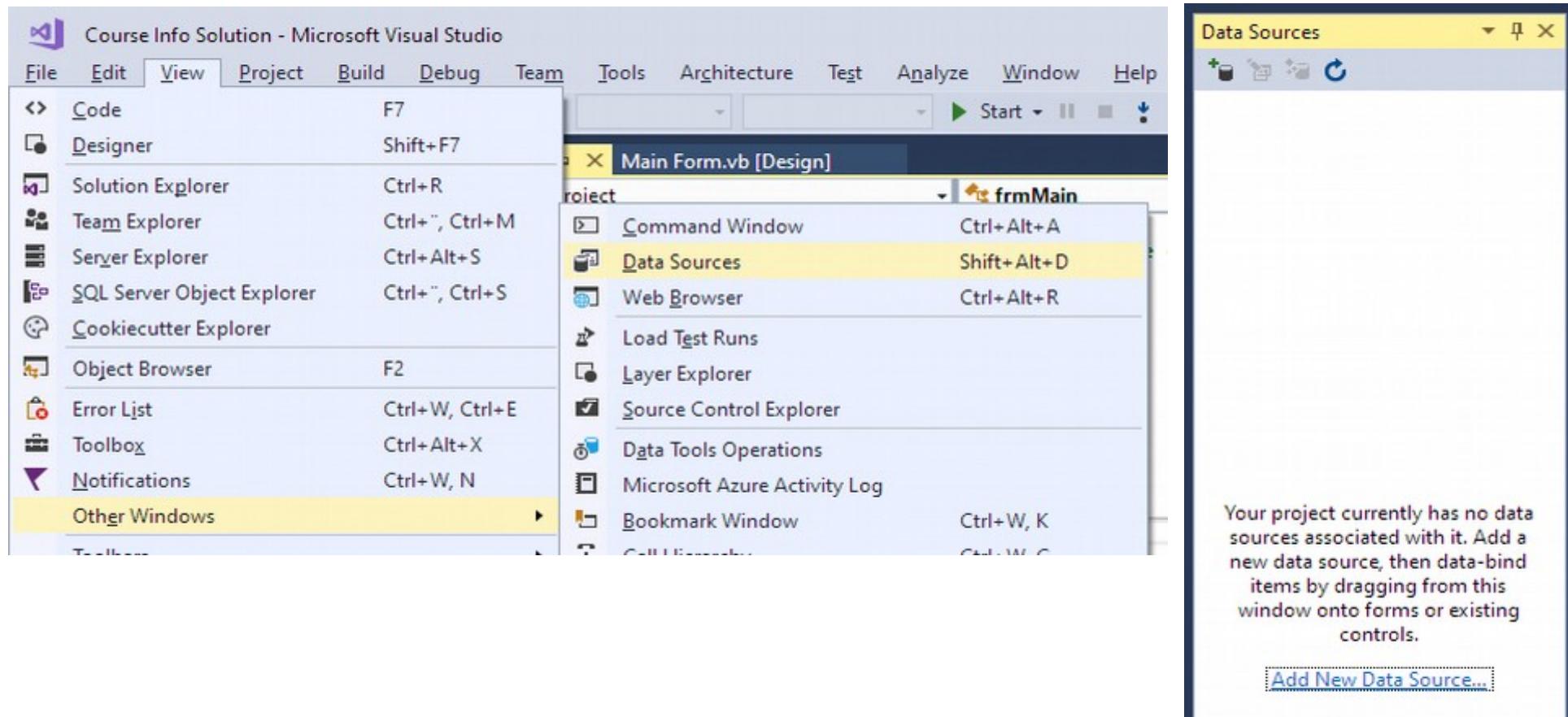
- although the database MyCourses.mdf file is contained in the **Course Info Project** folder, you still need to tell the application to use the file as a data source
- you can do this by using the **Data Source Configuration Wizard** to connect the application to the file
- the wizard allows you to specify the data you want to access from the file
- the computer stores in its main memory a copy of the specified data, called a **dataset**
- > a **dataset**: = a copy of the specified data stored in computer's main memory

- to create a **dataset** and then view its contents example: **01.Course Info Solution**

-> open the: ...VB2017\Chap11\Exercise\01.Course Info Solution\Course Info Solution.sln

**step 4.1** -> on the VS menu bar click **View / Other Windows** and choose: **Data Sources** Shift+Alt+D

- a new window named **Data Sources** will open



**step 4.2** -> in the **Data Sources** window, click [Add New Data Source...](#) to start the **Data Source Configuration Wizard**

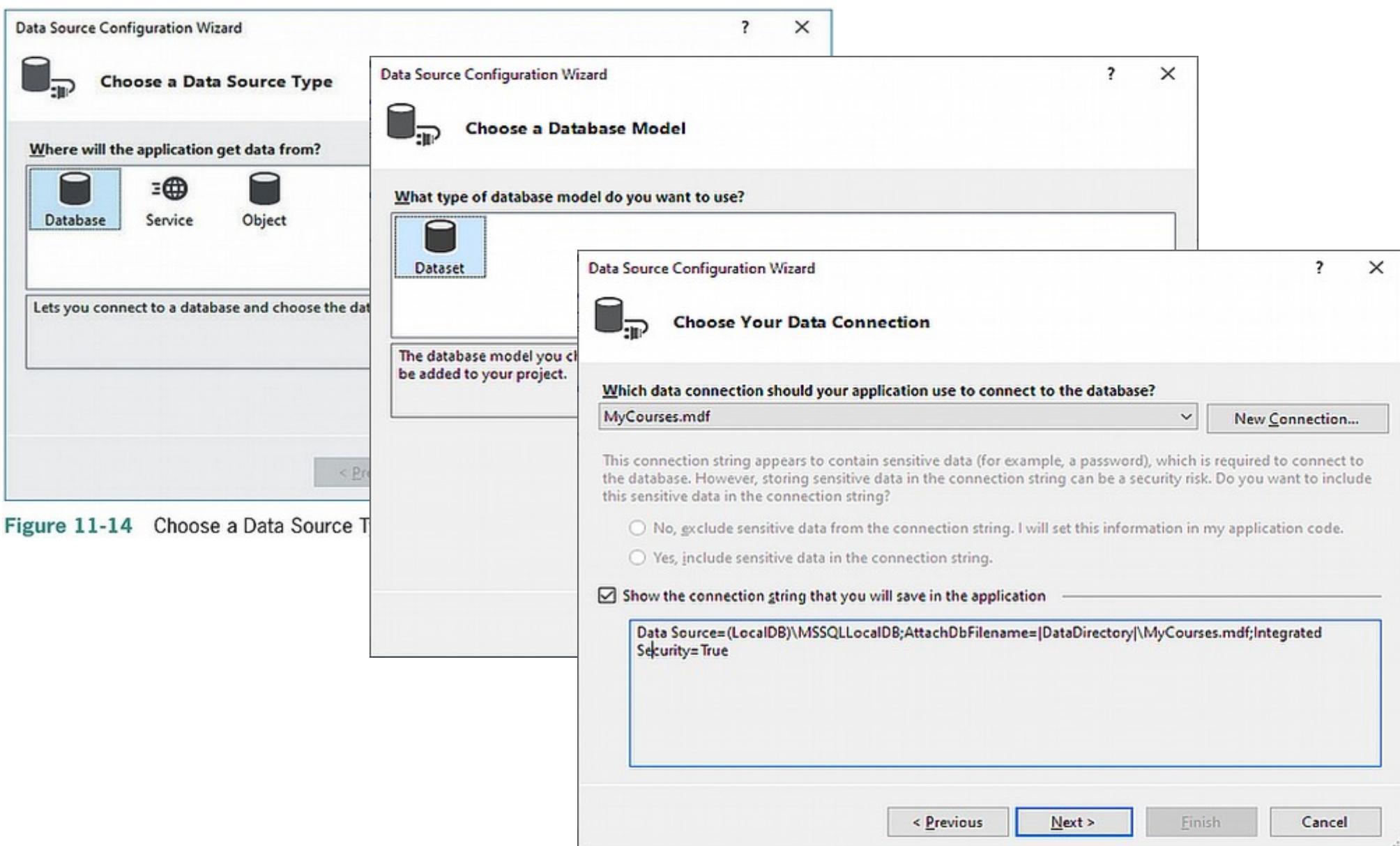
Figure 11-14

-> in screen **Choose a Data Source Type**, choose **Database** and click the button [Next >](#)

**step 4.3** -> in screen **Choose a Database Model**, choose **Dataset** and click the button [Next >](#)

- determines the types of **data objects** your application code uses

**step 4.4** -> in screen **Choose Your Data Connection**, verify that the **MyCourses.mdf** appears as the data connection, and click the button [Next >](#)



**step 4.5** - the screen **Save the Connection String to the Application Configuration File**, displays the name of the connection string: **MyCoursesConnectionString**  
-> verify that the check box is selected: **Yes, save the connection as:**, and click the button **Next >**

**step 4.6** - the screen **Choose Your Database Objects** appears

- you use this screen to select the database objects you want to include in your **dataset**

- the default name for the **dataset** is: **MyCoursesDataSet**

-> expand the nodes **Tables / Courses** and select the check box next to **Courses**

- doing this selects the **table object** and its **field objects**

-> click the button **Finish**

The screenshot shows the Data Source Configuration Wizard with two open windows.

**Left Window: Save the Connection String to the Application Configuration File**

- Icon: Database with a connection line.
- Title: **Save the Connection String to the Application Configuration File**
- Text: "Storing connection strings in your application configuration file. To store the connection string in the application configuration file, enter a name in the 'Connection string name:' box." **MyCoursesConnectionString**
- Checkboxes:
  - Yes, save the connection as: **MyCoursesConnectionString**
- Buttons: < Previous, Next >

**Right Window: Choose Your Database Objects**

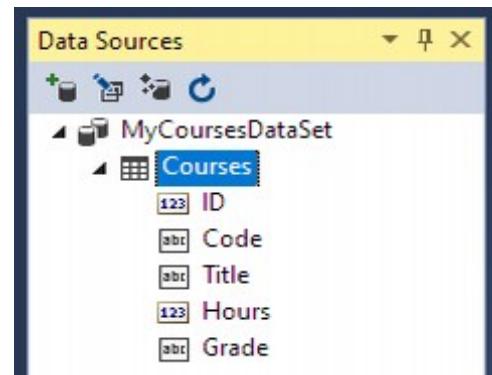
- Icon: Database with a connection line.
- Title: **Choose Your Database Objects**
- Text: "Which database objects do you want in your dataset?"
- Tree View:
  - Tables
    - Courses
      - ID
      - Code
      - Title
      - Hours
      - Grade
    - Views
    - Stored Procedures
    - Functions
- Text: **DataSet name:** **MyCoursesDataSet**
- Buttons: < Previous, Next >, **Finish** (highlighted in blue), Cancel

**step 4.7** - the computer adds the **MyCoursesDataSet** to the window **Data Sources**

-> in window **Data Sources**, expand nodes **MyCoursesDataSet / Courses**

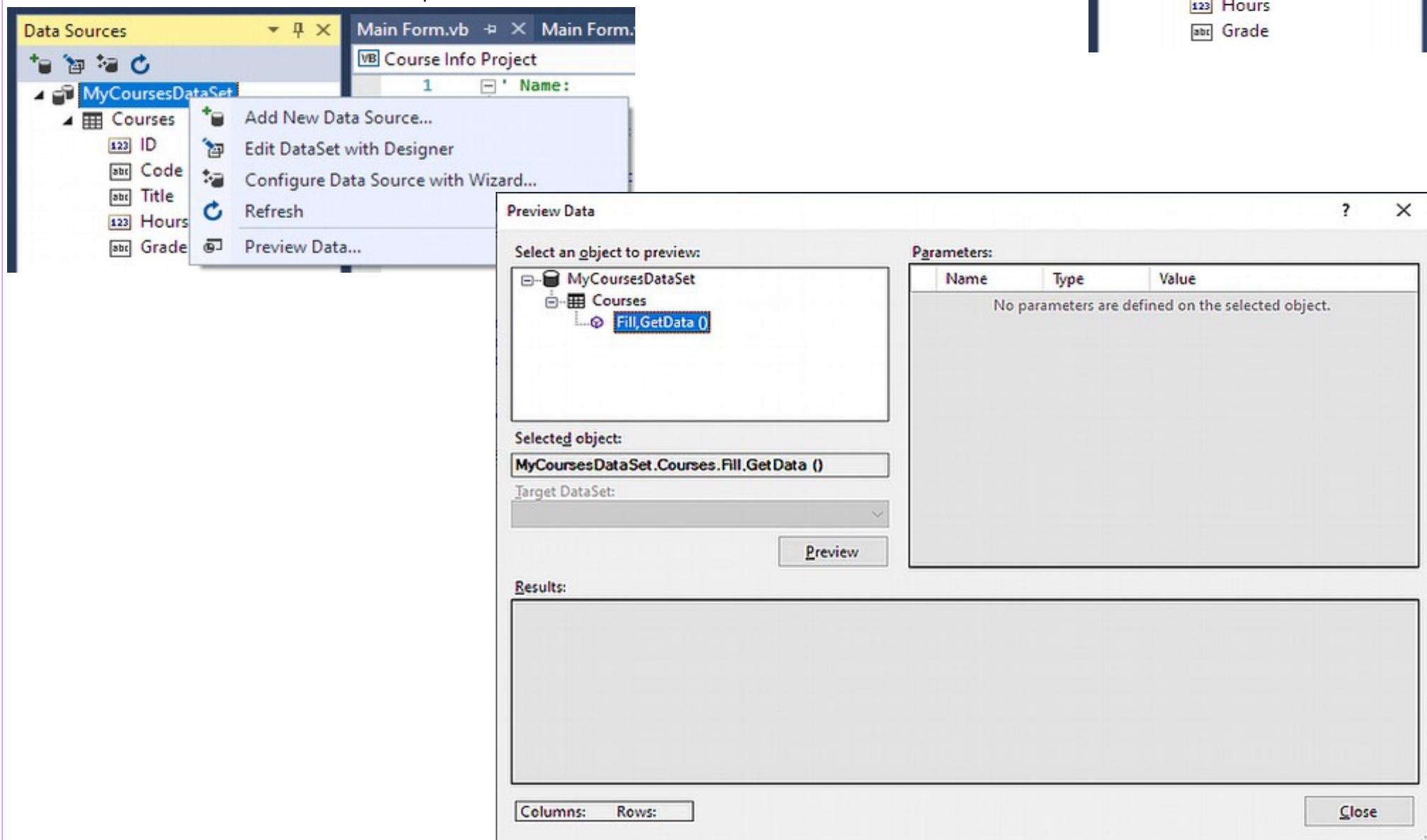
- the dataset contains: - **1 table** object named: **Courses**, and

- **5 field** objects named: **ID, Code, Title, Hours, Grade**



**step 4.8** -> in window **Data Sources**, right-click **MyCoursesDataSet** and click: **Preview Data...**

- a new screen named: **Preview Data** opens



**step 4.9** -> in screen **Preview Data**, expand nodes: **MyCoursesDataSet / Courses / Fill,GetData()** and click button **Preview** to view your dataset:

- the **dataset** contains: - 5 columns = **header items**, as you defined in **Step 2**

- 6 rows = **records**, as you defined in **Step 3**

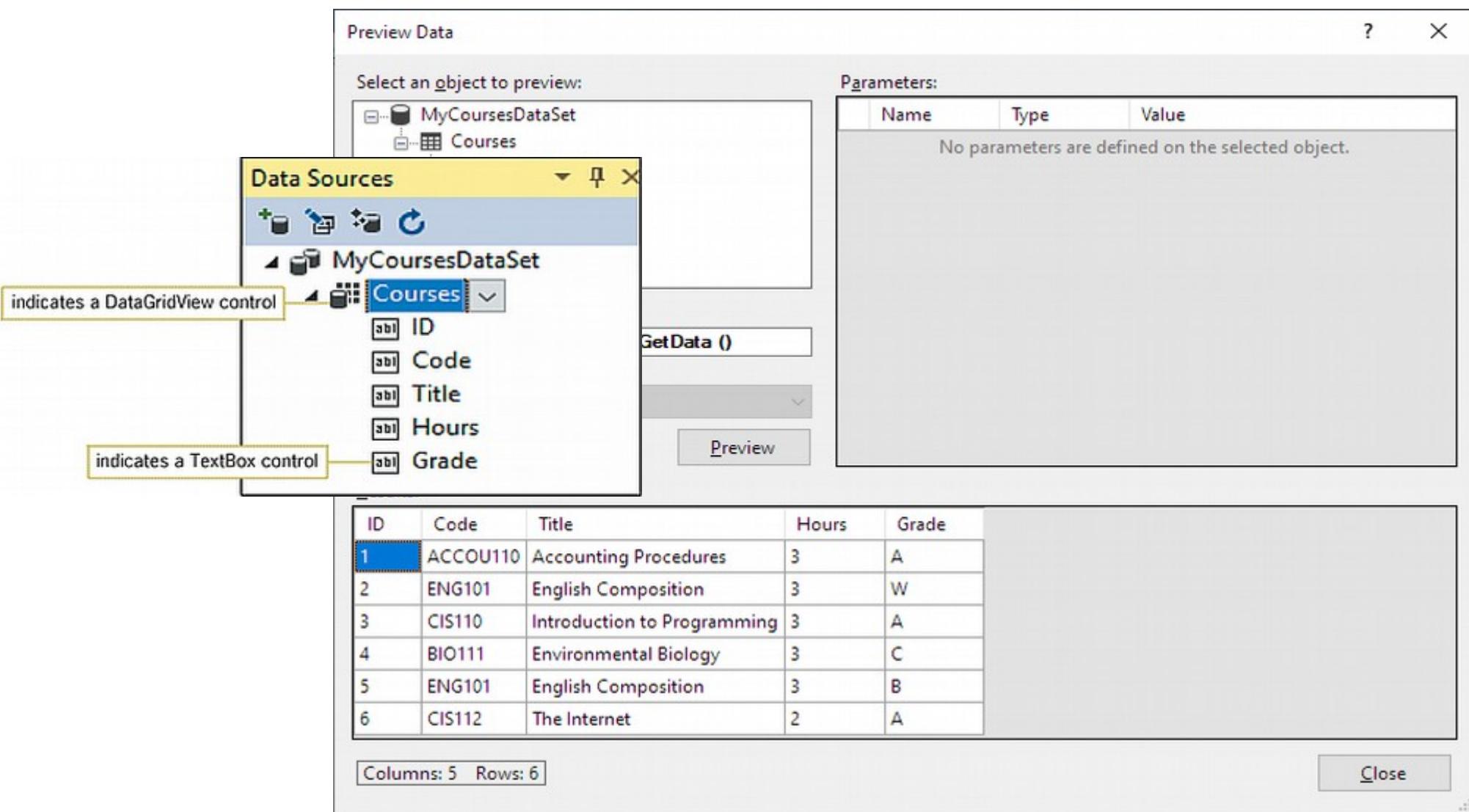
- notice box named: **Select an object to preview:**

- **MyCoursesDataSet** = name of the **dataset** in the application

- **Courses** = name of the **table** included in the **dataset**

- **Fill** = method, it populates an existing **table** with data

- **GetData()** = method, it creates a new **table** and populates it with data



-> click the **Close** button to close the **Preview Data** dialog box

## CH11\_F5 - info for step 5/8: about Binding/connecting an object in DataSet to the new or existing control in GUI: basic info & example with a new control

- for the user to view the contents of a **dataset** during an application run time, you need to: **connect/bind** at least **1 object** in the dataset, to at least **1 control** in GUI

-> **binding:** = connecting an object to a control

-> a **bound control:** = a control connected to an object in **DataSet**

- you can **bind/connect** an object in a **DataSet** to a **GUI control** in **2 ways**:

a). **bind an existing control** - more info and e.g. in: [CH11\\_A2 - Bind Field objects to existing controls - basic info](#)

a1). bind by dragging

- more info and e.g. in: [CH11\\_A2.1 - Bind Field objects to ...](#)

a2). bind by setting control's **Properties**

b). have the computer **create a Bound control** = let the computer to: **create** a control and then **bind** an object to it

b1). create **DataGridView** bound control from a table

b2). create **Data Form** bound control = separate control for each field in the table

[CH11\\_A1 - Create a Data Form \(instead of using DataGridView control\)...](#)

a). **bind an existing control** - more info and e.g. in: [CH11\\_A2 - Bind Field objects to existing controls - basic info](#)

a1). bind by dragging - the easiest way

- more info and e.g. in: [CH11\\_A2.1 - Bind Field objects to ...](#)

-> from window **Data Sources** drag the object to the control in **Designer** window

a2). bind by setting control's **Properties** - the appropriate property/s depends on the control type you are binding

-> in the **Designer** window click the control and then in **Properties** window set 1 or more **Properties**

-> to bind a **ListBox** control: 1). use property **DataSource** and select a table

2). use property **DisplayMember** and select a field

-> to bind a **DataGridView** control to a **table**, use the property: **DataSource** and select the table

-> to bind a **Label** or **TextBox** control to a table's field, use the property: **(DataBindings)** / **Text** and select the field

Figure 11-47

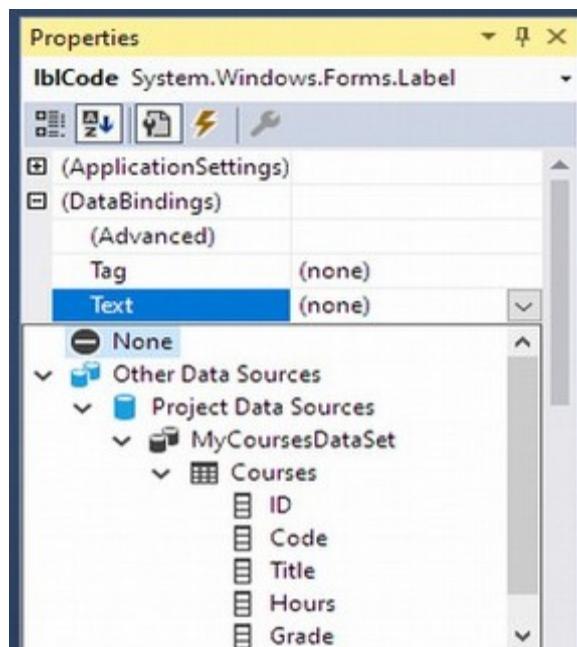


Figure 11-47 Label, TextBox

b). have the computer **create a Bound control** = let the computer to: **create** a control and then **bind** an object to it

b1). create **DataGridView** bound control from a table

- e.g. in: **CH11\_F5.1 - step 5/8 Bind an object Courses table to default DataGridView control...**

-> a **DataGridView control**: = displays the table data in a row and column format, similar to a spreadsheet  
- more info in: **CH11\_F6 -** & **CH11\_F6.1 -** & **step 6.x**

-> in the window **Data Sources**, click the table object you want to bind **Figure 11-20a**

- the icon that appears before the object's name in the **Data Sources** window indicates the **type of control** the computer will create

-> if necessary, use the field object's list arrow to change the field's **control type** **Figure 11-20b**

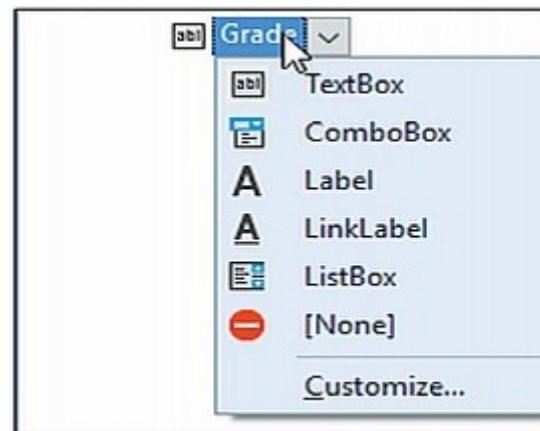
-> and drag the object to an **empty** area on the form

- when you drag an object from a **dataset** to an empty area on the form, the computer creates a **control** and automatically **binds** the object to it

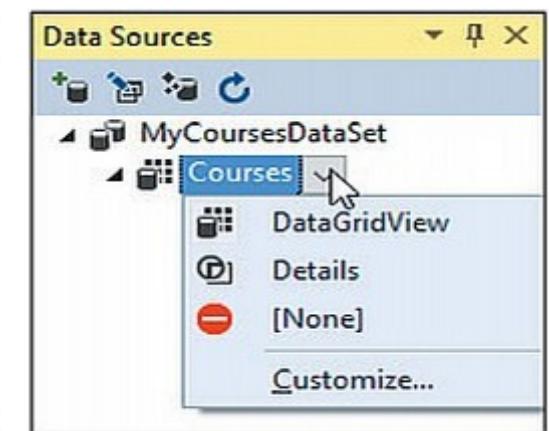
e.g. example with **01.Course Info Solution**

- the icon next to **Courses** indicates that a **DataGridView control** will be created when you drag the **Courses table** object to the form

**Figure 11-20a**



**Figure 11-20a**



**Figure 11-20c**

**Figure 11-20b**

b2). create **Data Form** bound control = separate control for each field in the table

**CH11\_A1 - Create a Data Form** (instead of using **DataGridView** control)...

e.g. example with **01.Course Info Solution**

- to display the **Courses table**'s data in separate **text boxes** rather than in a **DataGridView** control:

-> in window **Data Sources** / node **MyCoursesDataSet** / node **Courses** click the list arrow and choose: **Details** **Figure 11-20c**

- choosing **Details** in the list tells the computer to create a separate appropriate control for each field in the table (text boxes, labels, and so on...)

- it also adds the **BindingNavigator** control to the form and the **5** objects to the component tray

- the appropriate controls and objects are also automatically included when you drag a field object to an empty area on the form

-> if necessary, use the field object's list arrow to change the field's **control type** **Figure 11-20b**

- the icon next to each of the **5** field objects indicates that the computer creates a text box when a field object is dragged to the form

- to display the **Grade** field's data in a **label** control rather than in a **text box**:

-> in window **Data Sources** / **Courses** / **Grade** click the list arrow and choose: **Label** **Figure 11-20b**

**Figure 11-20**

-> and drag the object to an **empty** area on the form

- when you drag an object from a **dataset** to an empty area on the form, the computer creates a **control** and automatically **binds** the object to it

## CH11\_F5.1 - step 5/8 Bind an object Courses table to default DataGridView control example 01.Course Info Solution

- in the following set of steps, you will drag the **Courses** table object from the **Data Sources** window to the form, using the default control type for a table -

- **DataGridView** control

- to bind the **Courses** object to a **DataGridView** control example: **01.Course Info Solution**

-> open the: ...VB2017\Chap11\Exercise\01.Course Info Solution\Course Info Solution.sln

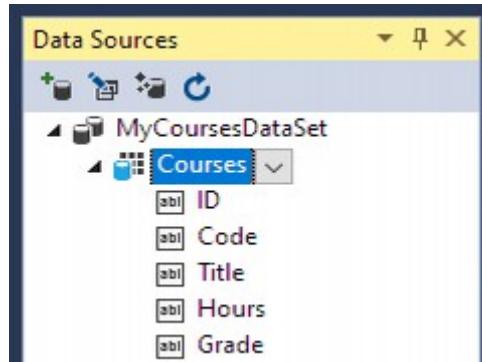
step 5.1 -> in the window **Data Sources** click **Courses** object and drag it to the center of the form's **Designer window** named **Main Form.vb [Design]**

**Figure 11-21**

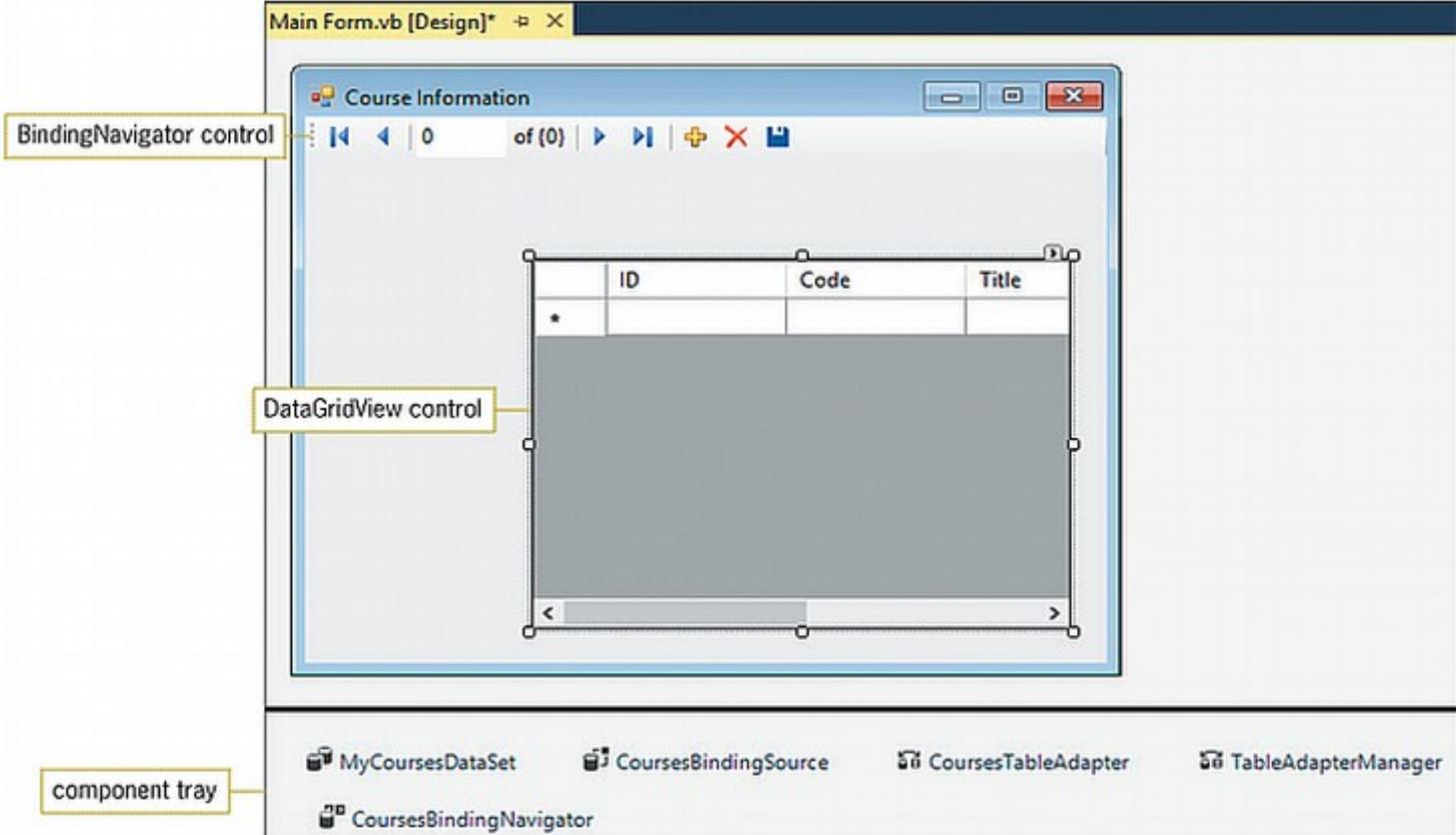
- the computer adds:

1). a **DataGridView** control to the form, and binds the **Courses** object to it

**Figure 11-22**



**Figure 11-21**



**Figure 11-22** Result of dragging the table object to the form

2). a **BindingNavigator** control to the form, used by the user during **run time**

**Figure 11-22**

-> a **BindingNavigator** control: = during **run time**, it allows the user to interact with a **dataset**, like:

- move from one record to the next
- add or delete a record
- save any changes

3). **five objects** to the component tray:

**Figure 11-22**

1). **MyCoursesDataSet** object

- object **DataSet** for database named **MyCourses** created in previous **Step 4**

2). **CoursesBindingNavigator** object

- object **BindingNavigator** for table named **Courses** created in previous **Step 2**

3). **CoursesTable Adapter** object

- object **TableAdapter** = connects the database to the **DataSet** object, which stores the information you want to access from the database

- responsible for: - retrieving the appropriate information from the database, and  
- storing it in the **DataSet**

- also can be used to: - insert records in the **DataSet**  
- delete records from the **DataSet**

4). **TableAdapterManager** object = saves to the database any changes made to the **DataSet** in the object **TableAdapter**

5). **CoursesBindingSource** object

**Figure 11-24**

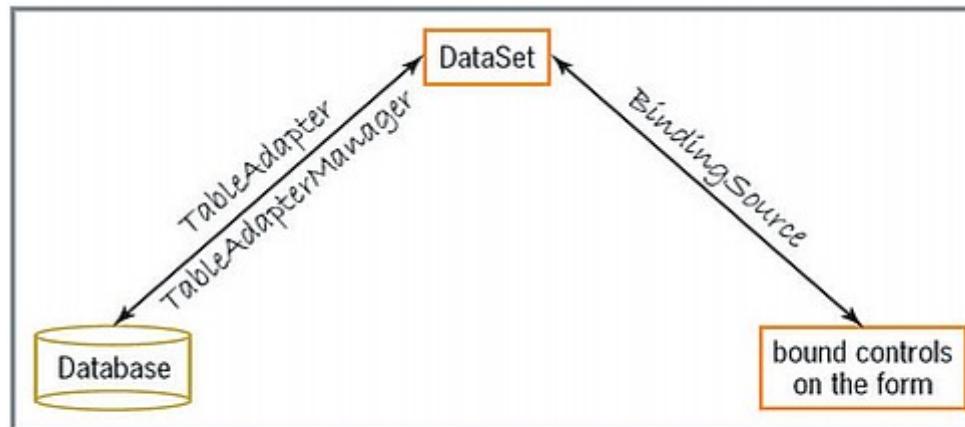
- object **BindingSource** = provides the connection between the **DataSet** and the **Bound controls** on the form

- **Bound controls** can be a **TextBox** control, **Label** control, **DataGridView** control, etc...

- in my example it connects the **MyCoursesDataSet** to 2 bound controls: - a **DataGridView** control, and  
- a **BindingNavigator** control

- so it allows the **DataGridView** control to display the data contained in the **MyCoursesDataSet**

- it also allows the **BindingNavigator** control to access the records stored in the **MyCoursesDataSet**



**Figure 11-24** Illustration of the relationships among the database, the objects in the component tray, and the bound controls

**step 5.2** -> save the solution and then start the application

- 6 records appear in the **DataGridView** control

- you will improve the appearance of the control in the next section:

**CH11\_F6 -** & **CH11\_F6.1 -** & **step 6.x**

|   | ID | Code     | Title       |
|---|----|----------|-------------|
| ▶ | 1  | ACCOU110 | Accountin   |
|   | 2  | ENG101   | English Co  |
|   | 3  | CIS110   | Introducti  |
|   | 4  | BIO111   | Environment |
|   | 5  | ENG101   | English Co  |
|   | 6  | CIS112   | The Intern  |
| * |    |          |             |

use the scroll bar to view the remaining fields

**step 5.3** - you can learn how to display the records in ascending or descending order by any field in table header

-> in the **Course Information** application window, click the header **Code** to display the records in ascending order by the **Code** field

-> then, click the header **Code** again to display the records in descending order by the **Code** field

|   | ID | Code     | ▲ Title     |
|---|----|----------|-------------|
| ▶ | 1  | ACCOU110 | Accountin   |
|   | 4  | BIO111   | Environment |
|   | 3  | CIS110   | Introducti  |
|   | 6  | CIS112   | The Intern  |
|   | 2  | ENG101   | English Co  |
|   | 5  | ENG101   | English Co  |
| * |    |          |             |

|   | ID | Code     | ▼ Title     |
|---|----|----------|-------------|
| ▶ | 2  | ENG101   | English Co  |
|   | 5  | ENG101   | English Co  |
|   | 6  | CIS112   | The Intern  |
|   | 3  | CIS110   | Introducti  |
|   | 4  | BIO111   | Environment |
|   | 1  | ACCOU110 | Accountin   |
| * |    |          |             |

-> close the application

## CH11\_F6 - info for step 6/8: about DataGridView control for displaying table data - basic info & example

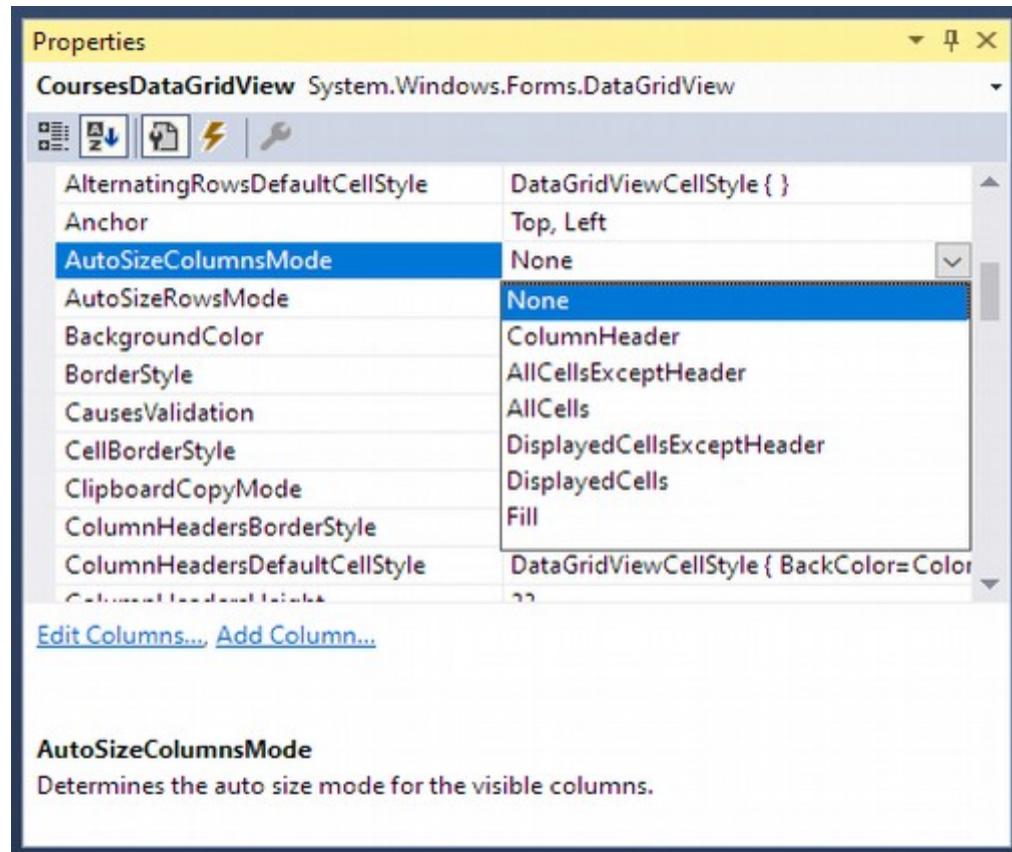
- one of the most popular controls for displaying table data, for it allows the user to view a great deal of information at the same time
- the control displays the data in a row and column format, similar to a spreadsheet:
  - each **column** represents a **field**
  - each **row** represents a **record**
- the intersection of a column and a row in a **DataGridView** control is called a **cell**

-> a **cell**: = intersection of a table's **field** and **record** in a **DataGridView** control

- to determine the way the column widths are sized in the control:

-> window **Designer** -> choose **control**

-> window **Properties** / property **AutoSizeColumnsMode** -> choose mode, that the column widths adjust:

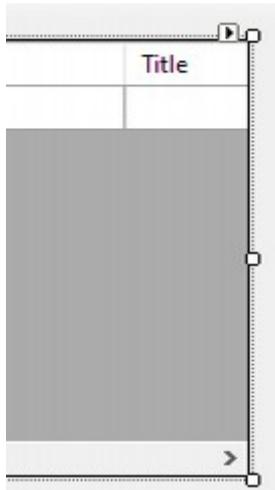


- **None**: default; do not automatically adjust
- **ColumnHeader**: to fit the contents of the column header cells
- **AllCellsExceptHeader**: to fit the contents of all cells in the columns, excluding header cells
- **AllCells**: to fit the contents of all cells in the columns, including header cells
- **DisplayedCellsExceptHeader**: to fit the contents of all cells in the columns that are in rows currently displayed onscreen, excluding header cells
- **DisplayedCells**: to fit the contents of all cells in the columns that are in rows currently displayed onscreen, including header cells
- **Fill**: so that the widths of all columns exactly fill the display area of the control

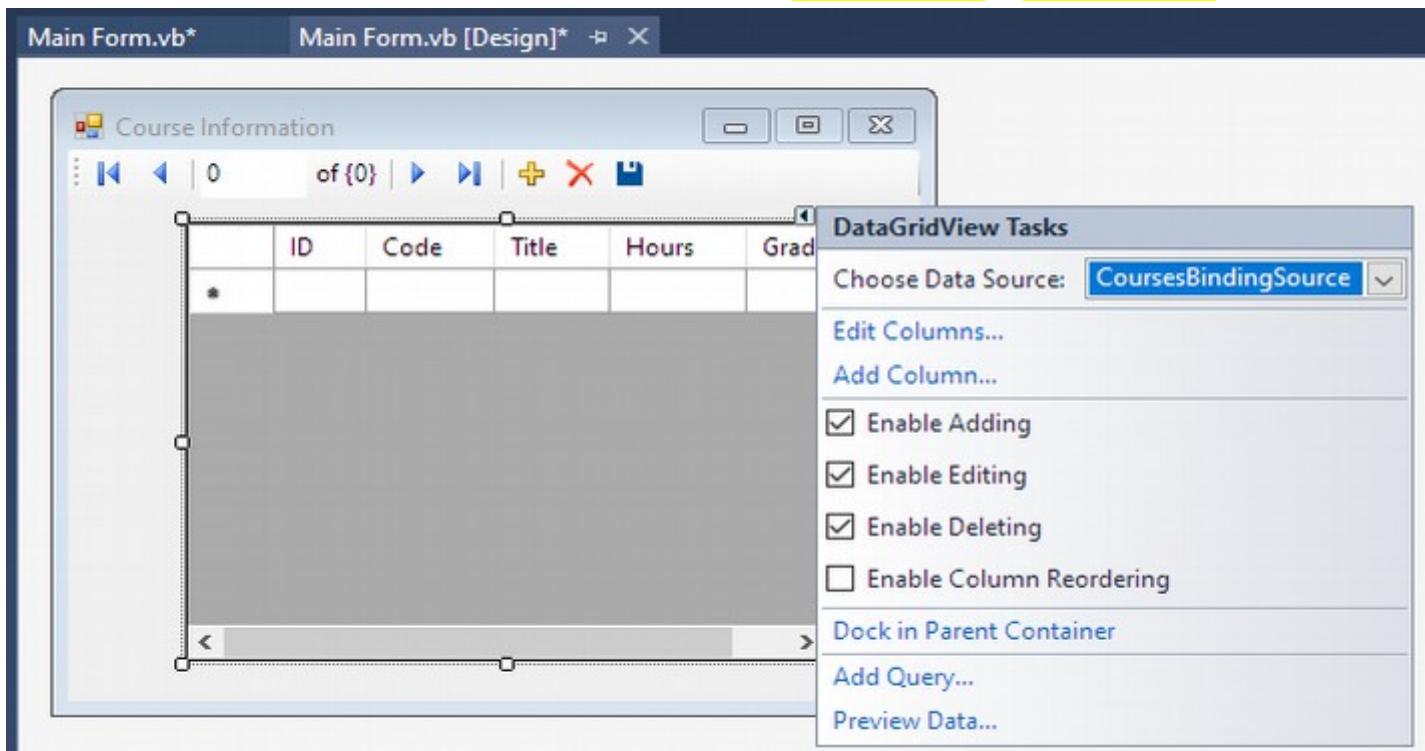
- like the **PictureBox** control, the **DataGridView** control has a **task list**, accessible via **Designer** window

> in **Designer** window, select the control and then click the small arrow in its upper right corner

**Figure 11-25a** & **Figure 11-25b**



**Figure 11-25a**



**Figure 11-25b**

- **DataGridView** control's **task list** items:

task:

**Choose Data Source**

**Edit Columns...**

**Add Columns...**

**Enable Adding**

**Enable Editing**

**Enable Deleting**

**Enable Column Reordering**

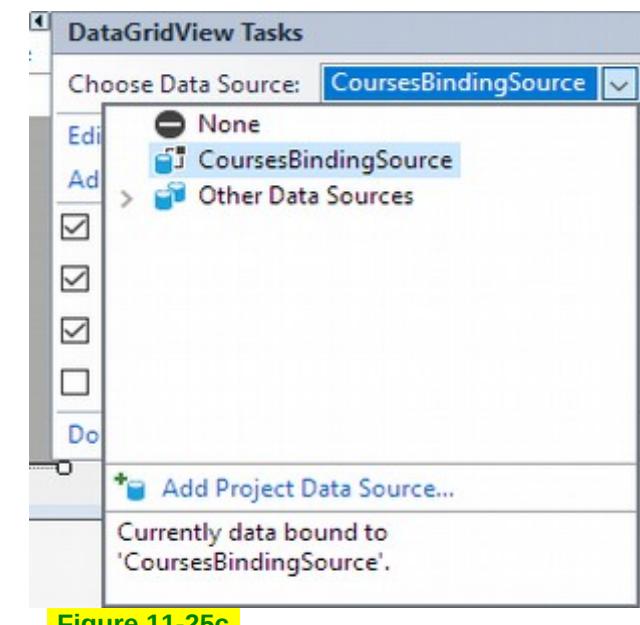
**Dock in Parent Container**

**Add Query...**

**Preview Data...**

**purpose:**

- select a data source **Figure 11-25c**
- open the **Edit Columns** dialog box **Figure 11-26**
- add a new column
- allow/disallow the user to add data
- allow/disallow the user to edit data
- allow/disallow the user to delete data
- allow/disallow the user to reorder the columns
- bind the borders of the control to its container
- filter data from a dataset
- view the data bound to the control



**Figure 11-25c**

- Figure 11-26 shows the **Edit Columns** dialog box that opens when you click **Edit Columns...** on the **DataGridView** control's **task list**

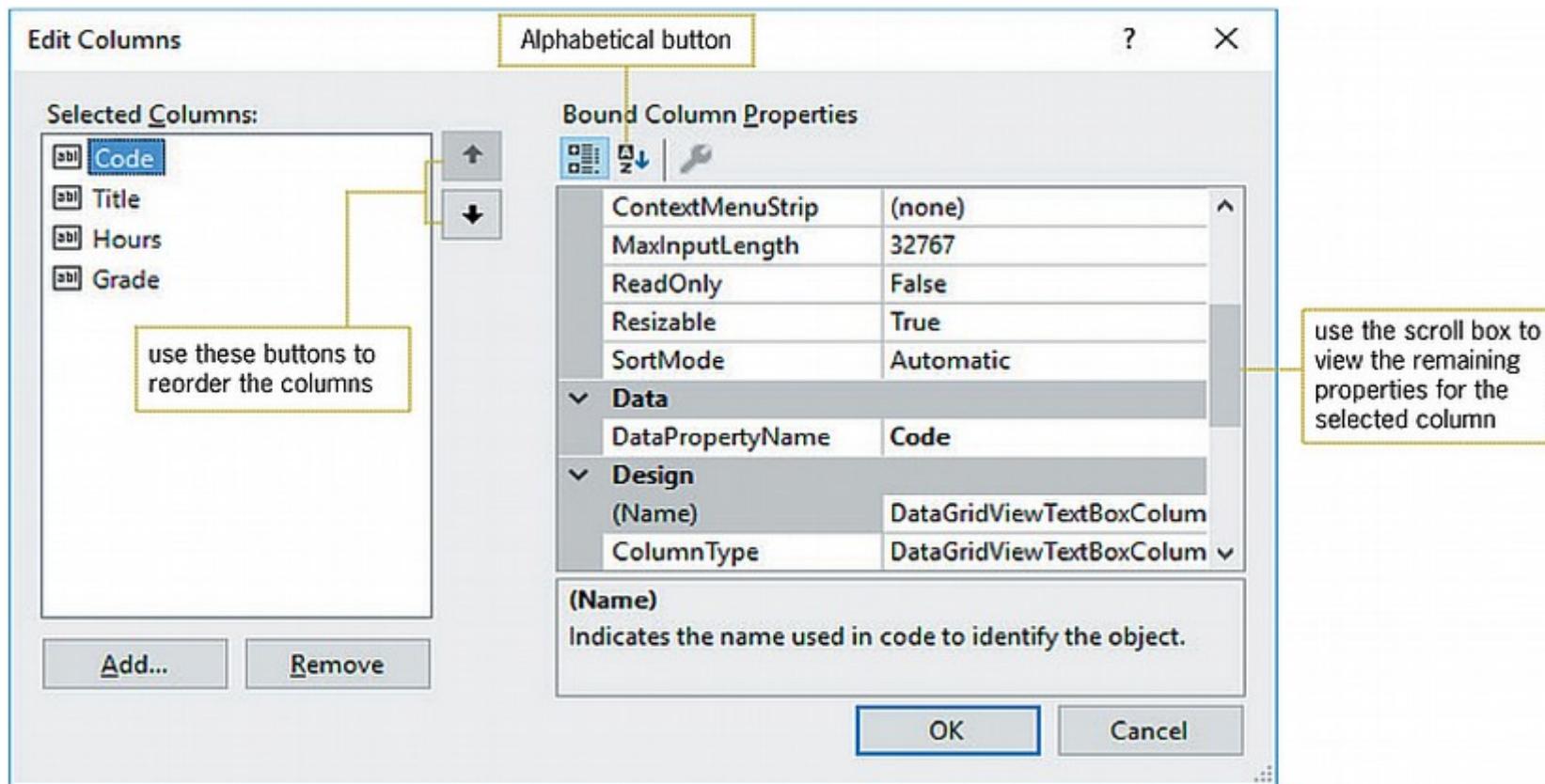
- mistake in a book: in Lbox **Selected Columns:** is missing an **ID** field

- you can use this dialog box during **design time** to:
  - add columns to the control
  - remove columns from the control
  - reorder the columns
  - set the properties of the bound columns

e.g. - property **DefaultCellStyle** to:

- format the column's data
- change the column's width and alignment

e.g. - property **HeaderText** to: - change a column's heading



**Figure 11-26** Edit Columns dialog box

- mistake in a book: in Lbox **Selected Columns:** is missing an **ID** field

## CH11\_F6.1 - step 6/8: improve the appearance of the DataGridView control - set the columns & its cells example 01.Course Info Solution

- in this step, you will play with a control to display your table

- to improve the appearance of the CoursesDataGridView control example: 01.Course Info Solution

-> open the: ...VB2017\Chap11\Exercise\01.Course Info Solution\Course Info Solution.sln

**step 6.1** - you will choose a mode, that the column widths adjust so that the widths of all columns exactly fill the display area of the control - no need for scroll bars

-> in **Designer** window, choose the CoursesDataGridView control to view its **Properties**

-> in the window **Properties**, scroll to the property **AutoSizeColumnsMode** and set it to: **Fill**

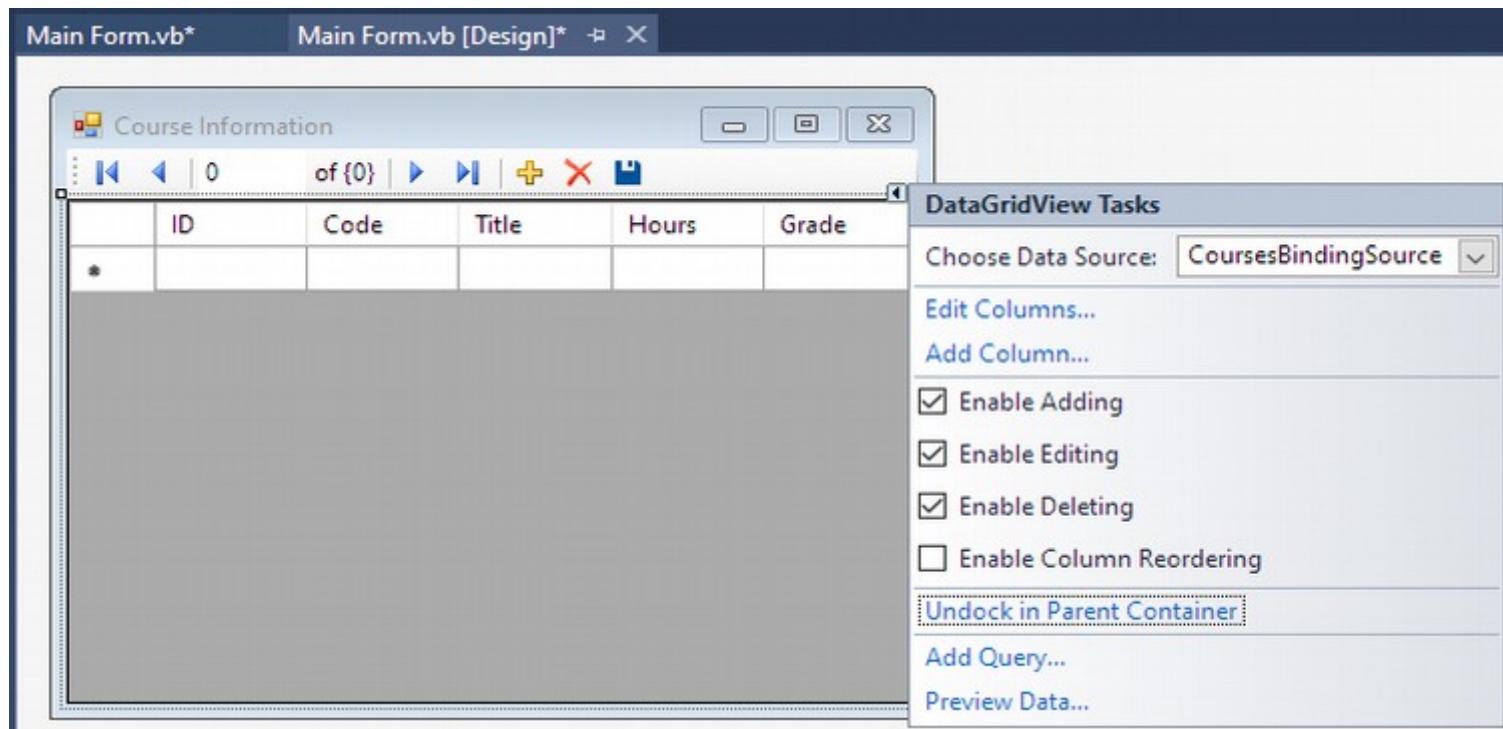
- the table is now adjusted, so that all of the fields are visible and there is no need for a scroll bars

**step 6.2** - you will anchor the control's borders to the borders of its container - which is the **form** in this case

-> in **Designer** window, choose the CoursesDataGridView control and click the small arrow in the upper right corner - the control's **task box**

-> in the task list, click the task: **Dock in Parent Container**

- see picture, where the table is spread using all of the free space of the **frmMain form** - compare to: **Figure 11-25b**

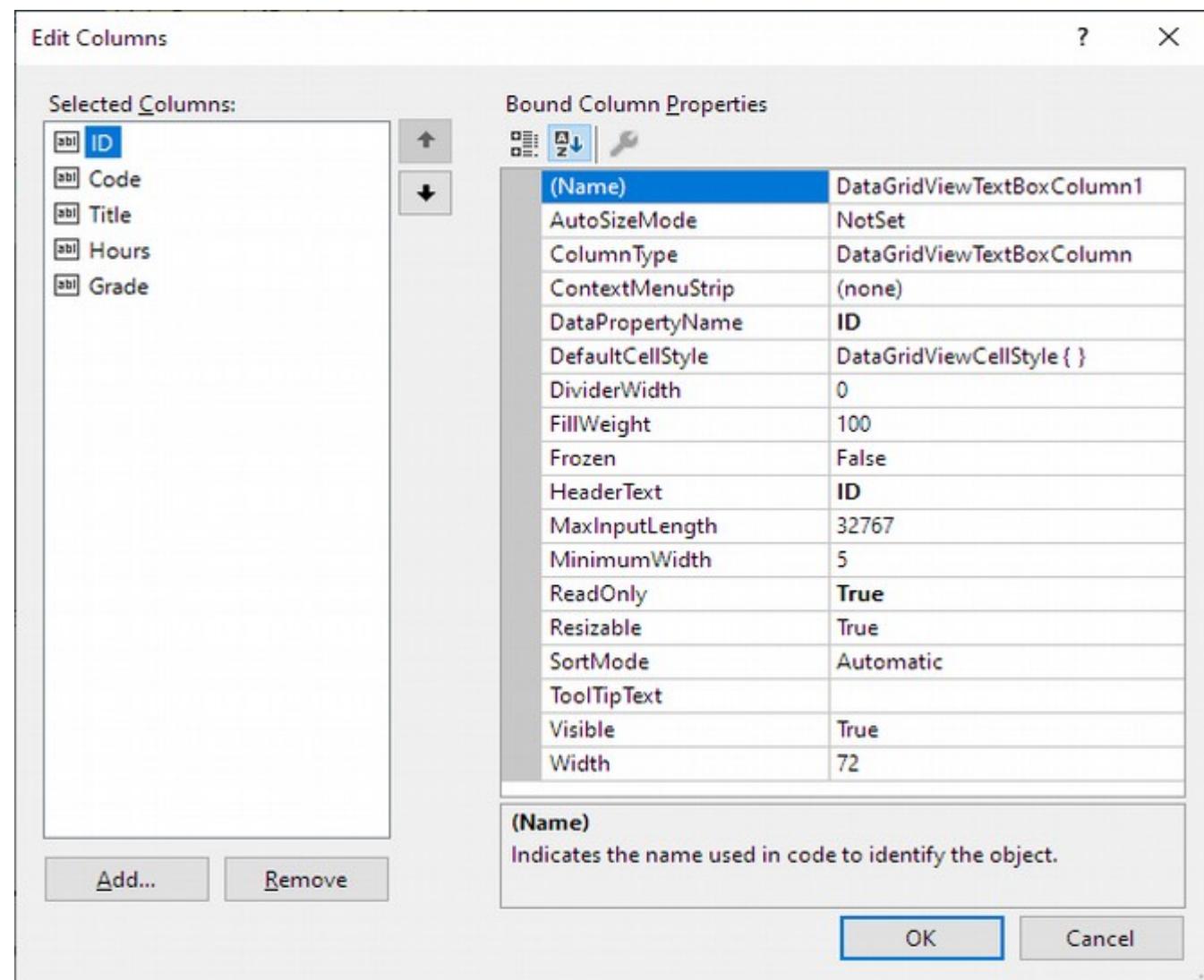


**step 6.3**

- you will open the **Edit Columns** dialog box and set the properties for each of your header column **Figure 11-26b**
  - > in **Designer** window, choose the **CoursesDataGridView** control and click the small arrow in the upper right corner - the control's **task box**
  - > in the task list, click the task: **Edit Columns...** to open **Edit Columns** dialog box

- you can see in Lbox: **Selected Columns**, the list of your fields you can edit: **ID, Code, Title, Hours, Grade**

-> in a Rbox: **Bound Column Properties**, change the properties display style from **Categorized** to **Alphabetical** - its more easy to navigate



**Figure 11-26b** Edit Columns dialog box

**step 6.4**

- in **Edit Columns** dialog box, you will select a **Column** a set some of the **Bound Column Properties**:

-> **ID** -> verify that the property **ReadOnly = True**

- recall that the **ID** field is an auto-numbered field, as you set in:

**step 2.4**

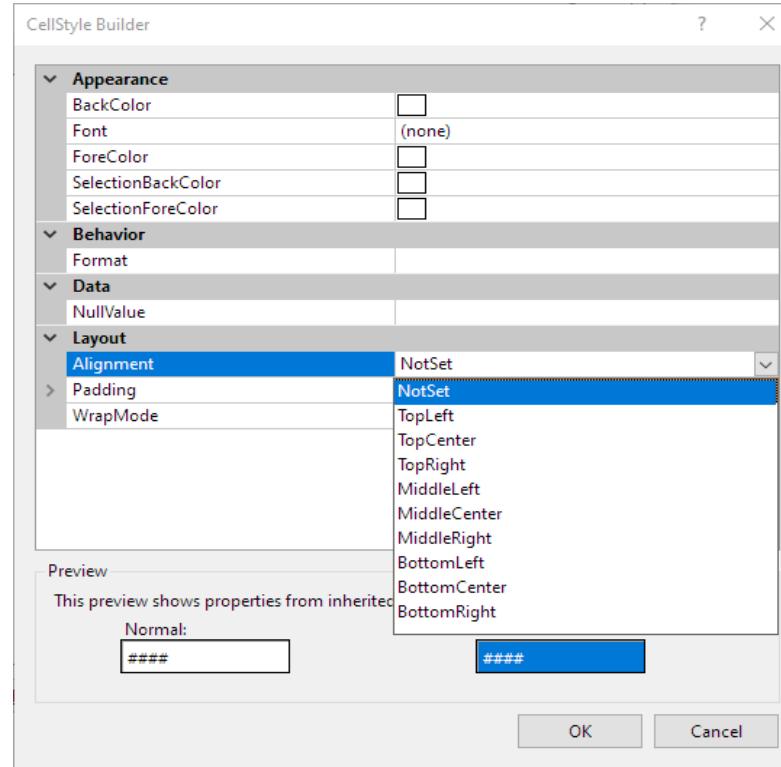
-> **Code** -> **AutoSizeMode = AllCells** (originally: **NotSet**)

-> **Title** -> **AutoSizeMode = AllCells** (originally: **NotSet**)

**Figure 11-26b**

- step 6.5**
- if needed, you can open the **CellStyle Builder** dialog box to specify the style of the **cell** - so you go more deep in setting
  - the **CellStyle Builder** dialog box can be used to:
    - set the appearance like **BackColor**, **Font**, **ForeColor**, etc...
    - format a column's numbers display type like **Numeric**, **Currency**, **Data Time**, etc...
    - specify the cell's value alignment like **TopLeft**, **MiddleCenter**, etc...
    - and so on...

**Figure 11-27**



**Figure 11-27**  
**CellStyle Builder** dialog box

-> Hours -> click **DefaultCellStyle = DataGridViewCellStyle { }**, and then click the ... (ellipsis) button to open the **CellStyle Builder** dialog box

-> click **Alignment**, click the **list arrow**, and choose = **MiddleCenter**, and then click the **OK** button

- see in **Edit Columns** dialog box, in Rbox **Bound Column Properties**, that a change has been made:



-> Grade -> click **DefaultCellStyle = DataGridViewCellStyle { }**, and then click the ... (ellipsis) button to open the **CellStyle Builder** dialog box

-> click **Alignment**, click the **list arrow**, and choose = **MiddleCenter**, and then click the **OK** button

- see in **Edit Columns** dialog box, in Rbox **Bound Column Properties**, that a change has been made:

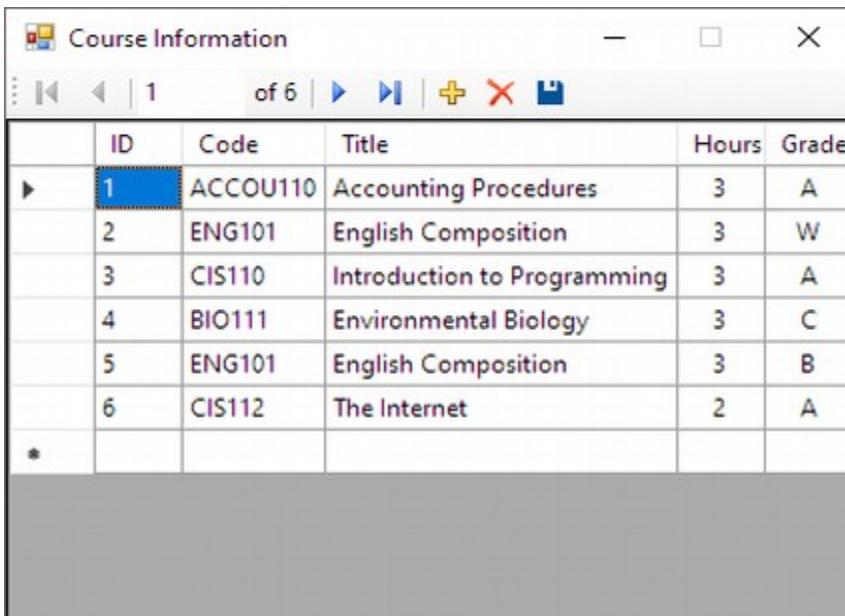


-> in **Edit Columns** dialog box click the button **OK** to close the **Edit Columns** dialog box

-> in **Designer** window, click the small arrow to close the **CoursesDataGridView** control's task list

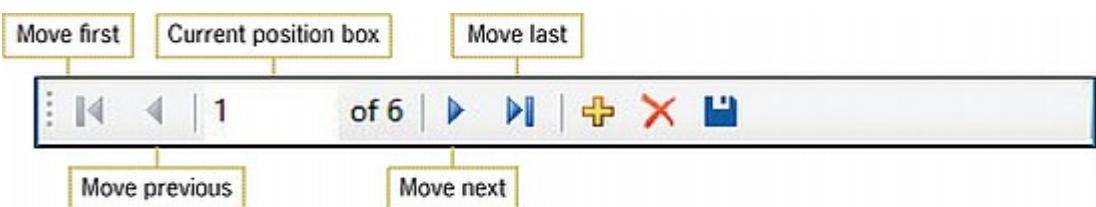
### step 6.6

- just start the application and learn various ways to navigate the highlighted area
- > save the solution and then start the application
- the **six records** in the **dataset** appear in the **CoursesDataGridView** control, with the first record's **ID** highlighted



- you can use various ways of moving highlight in a **DataGridView** control:

-> use the **BindingNavigator** control - arrows in its menu bar:



-> or use the **Tab** key on your keyboard  
-> or use the arrow keys on your keyboard

-> when done, close the application

### Mini-Quiz 11-3

1. Connecting a field object from a dataset to a control is called.....
2. Which object connects a dataset to a text box?
3. Which control contains buttons for adding, deleting, and saving records?
4. The intersection of a column and a row in a **DataGridView** control is called.....

## CH11\_F7 - info for step 7/8: set the local database file's \*.mdf property: **Copy to Output Directory** = correlation between source \*.mdf and output \*.mdf

- a database contained in a project is referred to as a **Local database**

→ a **local database file \*.mdf**: = a database file contained in a project

- in VS can be found in a **Solution Explorer** window

e.g. window **Solution Explorer / Course Info Project / MyCourses.mdf**

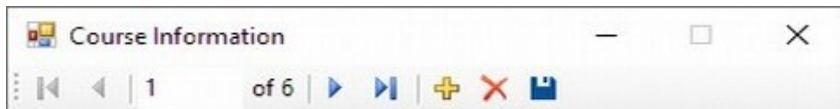
- physically located in: 1). in a source **Project folder**, and

e.g. 01.Course Info Solution\Course Info Project\MyCourses.mdf

2). in the output folder **bin\Debug**

e.g. 01.Course Info Solution\Course Info Project\bin\Debug\MyCourses.mdf

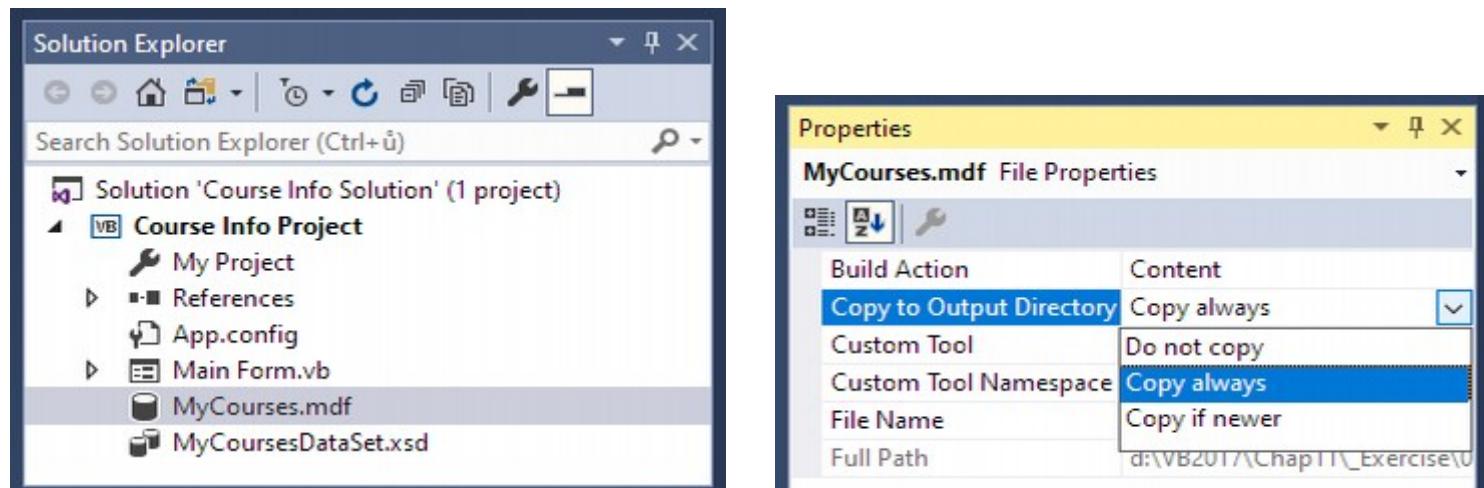
- when user makes some changes to the database during run time, and clicks the button **Save Data** on the **BindingNavigator** control menu bar:



- any changes made in the **DataGridView** control are recorded only in the file stored in the output folder **bin\Debug\\*.mdf**

- but the file stored in the **Project folder**\*.mdf is not changed

- you can modify correlation between **source** file and **output** file by setting a database property: **Copy to Output Directory**



→ database property **Copy to Output Directory**: = specifies the way, how the **source** file from **Project folder** will be copied to the **output** directory in **bin\Debug**

a). **Do not copy** = when the application is started, the file in the **Project folder** is not copied to the output directory **bin\Debug**

b). **Copy Always** - default setting

= when the application is started, the file in the **Project folder** is always copied to the output directory **bin\Debug**

- i.e. next time the app will start, any modification made during a previous run time will be lost

c). **Copy if newer** = when the application is started, the computer compares dates on the files:

1). \*.mdf file in the source **Project folder**

with

2). \*.mdf file in the output **bin\Debug** folder

- the file in the **Project folder** is copied only when its date is newer than the file's in **bin\Debug** folder

- this setting makes a sense, i think

### CH11\_F7.1 - step 7/8: set the local database file's \*.mdf property: **Copy to Output Directory** - example and testing 01.Course Info Solution

- in this step, you will set a correlation between a db source file located in a **Project folder**, and the db file modified during a **run time**, located in an output directory

- to change the **MyCourses.mdf** file's property & test: **Copy to Output Directory** example: **01.Course Info Solution**

-> open the: ...VB2017\Chap11\_Exercise\01.Course Info Solution\Course Info Solution.sln

**step 7.1** - you will change the **MyCourses.mdf** database's setting, how to behave when a changes are made during a run time

-> in **Solution Explorer** window, click the database file **MyCourses.mdf** and see the window **Properties**

-> in the window **Properties / Copy to Output Directory = default Copy Always**, change to: -> = **Copy if newer**

-> save the solution

**step 7.2** - you will start and test the application by modifying the database during a **run time**

-> start the application

-> below **Code** field **CIS112** click the empty cell and type: **PSYCH100**, and press **Tab** key **Figure 11-30**

- do not be concerned about the **-1** that appears in the **ID** column

-> type: **General Psychology**, and press **Tab** key

-> type: **3**

- the student has not completed the course yet, so you will leave the **Grade** field empty

-> in the **BindingNavigator** control, click the button **Save Data**, to save the new record to the dataset

- notice that the **-1** in the **ID** column changes to the number **7**

-> enter the next record: **ART100, Ceramics, 2, B**, and save the record to the dataset **Figure 11-31**

- notice that the **-2** in the **ID** column changes to the number **8**

-> close the application

|   | ID | Code     | Title                       | Hours | Grade |
|---|----|----------|-----------------------------|-------|-------|
|   | 1  | ACCOU110 | Accounting Procedures       | 3     | A     |
|   | 2  | ENG101   | English Composition         | 3     | W     |
|   | 3  | CIS110   | Introduction to Programming | 3     | A     |
|   | 4  | BIO111   | Environmental Biology       | 3     | C     |
|   | 5  | ENG101   | English Composition         | 3     | B     |
|   | 6  | CIS112   | The Internet                | 2     | A     |
| ▶ | -1 | PSYCH100 | The Internet                | 2     | A     |
| * |    |          |                             |       |       |

Figure 11-30

|   | ID | Code     | Title                       | Hours | Grade |
|---|----|----------|-----------------------------|-------|-------|
|   | 1  | ACCOU110 | Accounting Procedures       | 3     | A     |
|   | 2  | ENG101   | English Composition         | 3     | W     |
|   | 3  | CIS110   | Introduction to Programming | 3     | A     |
|   | 4  | BIO111   | Environmental Biology       | 3     | C     |
|   | 5  | ENG101   | English Composition         | 3     | B     |
|   | 6  | CIS112   | The Internet                | 2     | A     |
| ▶ | 7  | PSYCH100 | General Psychology          | 3     |       |
| * | 8  | ART100   | Ceramics                    | 2     | B     |
|   |    |          |                             |       |       |

Figure 11-31

**step 7.3** - you will verify the new records are saved in an output database file located in **bin\Debug** folder

-> now, start the application again

- the new records appear in the **DataGridView** control, because you set in

**step 7.1 Copy to Output Directory = Copy if newer**

The screenshot shows a Windows application window titled "Course Information". At the top, there is a toolbar with icons for back, forward, search, and other controls. Below the toolbar is a status bar showing "of 8". The main area is a **DataGridView** containing the following data:

|   | ID | Code     | Title                       | Hours | Grade |
|---|----|----------|-----------------------------|-------|-------|
| ▶ | 1  | ACCOU110 | Accounting Procedures       | 3     | A     |
|   | 2  | ENG101   | English Composition         | 3     | W     |
|   | 3  | CIS110   | Introduction to Programming | 3     | A     |
|   | 4  | BIO111   | Environmental Biology       | 3     | C     |
|   | 5  | ENG101   | English Composition         | 3     | B     |
|   | 6  | CIS112   | The Internet                | 2     | A     |
|   | 7  | PSYCH100 | General Psychology          | 3     |       |
| * | 8  | ART100   | Ceramics                    | 2     | B     |

**step 7.4** - we will continue testing by adding an info to existing record, and deleting a record

-> in a record number 7, click the empty **Grade** field and type: **C**

-> in the column **Code** click the cell **ART100** to choose it, and in the **BindingNavigator** control click the button **Delete** to delete the entire record

-> save the solution and close the application

**step 7.5** - we will start the application again to verify that the changes you made were saved

-> start the application again **Figure 11-32**

- notice that: - the **PSYCH100** grade: **C** appears in the course's **Grade** field  
- the **ART100** record was removed from the dataset

-> close the application

The screenshot shows the same "Course Information" application window after changes have been made. The **DataGridView** now contains 7 rows of data:

|   | ID | Code     | Title                       | Hours | Grade |
|---|----|----------|-----------------------------|-------|-------|
| ▶ | 1  | ACCOU110 | Accounting Procedures       | 3     | A     |
|   | 2  | ENG101   | English Composition         | 3     | W     |
|   | 3  | CIS110   | Introduction to Programming | 3     | A     |
|   | 4  | BIO111   | Environmental Biology       | 3     | C     |
|   | 5  | ENG101   | English Composition         | 3     | B     |
|   | 6  | CIS112   | The Internet                | 2     | A     |
| ▶ | 7  | PSYCH100 | General Psychology          | 3     | C     |

**Figure 11-32**

## CH11\_F8 - info for step 8/8: automatically generated code when binding objects - autopsy & example with 01.Course Info Solution

- in addition to adding the appropriate controls and objects to the application when a table or field object is **dragged** to the form, VB also automatically enters some code in the **Code Editor** window
- in "Step 5: bind objects in **DataSet** to controls in GUI -> create a **Bound control** using **DataGridView** control", and in **step 5.1** you dragged an object table named **Courses** from window **Data Source** to the **Designer** windows
- when you dragged the **Course** table to the form, the 2 procedures were automatically entered by Visual Basic:
  - 1). **CoursesBindingNavigatorSaveItem** with an event **Click**
  - 2). **frmMain** with an event **Load**
- recall that the *optional* keyword **Me** refers to the current form

**Figure 11-33** automatically generated code when binding/dragging an object to the form: example with **01.Course Info Solution**

```
9  Public Class frmMain
10     Private Sub CoursesBindingNavigatorSaveItem_Click(sender As Object, e As EventArgs) Handles CoursesBindingNavigatorSaveItem.Click
11         Me.Validate()
12         Me.CoursesBindingSource.EndEdit()
13         Me.TableAdapterManager.UpdateAll(Me.MyCoursesDataSet)
14
15     End Sub
16
17     Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles MyBase.Load
18         'TODO: This line of code loads data into the 'MyCoursesDataSet.Courses' table. You can move, or remove it, as needed.
19         Me.CoursesTableAdapter.Fill(Me.MyCoursesDataSet.Courses)
20
21     End Sub
22 End Class
```

### 1). the **CoursesBindingNavigatorSaveItem\_Click** procedure:

- is processed when the user clicks the button **Save Data** on the **BindingNavigator** control
- the procedure's code validates the changes made to the data before saving the data to the database
- the **2** methods are involved in the **save** operation:
  - **EndEdit** method - the **BindingSource** object's method: line **12**  
= applies any pending changes to the dataset (such as new records, deleted records, and changed records)
  - **UpdateAll** method - the **TableAdapterManager**'s method: line **13**  
= commits the changes to the database

```
11     Me.Validate()
12     Me.CoursesBindingSource.EndEdit()
13     Me.TableAdapterManager.UpdateAll(Me.MyCoursesDataSet)
```

### 2). the form's **Load** even procedure:

- uses the **TableAdapter** object's method **Fill** to retrieve the data from the database and store it in the **DataSet** object
- in most applications, the code to fill a **DataSet** belongs in this procedure, however you can move or delete the code

```
19     Me.CoursesTableAdapter.Fill(Me.MyCoursesDataSet.Courses)
```

## CH11 F8.1 - info for step 8/8: Try...Catch "safety net" statement = handling/catching app's exception/error during run time without terminating the app & example

- because it is possible for an error to occur when saving data to a database, you should add **error-handling code** to button's **Click** event procedure

-> an **exception**: = an error that occurs while an application is running/during a run time

- when an error occurs in a procedure's code during run time, programmers say: "**the procedure threw an exception**"

- if your code does **not contain** specific instructions for handling the **exceptions** that may occur, Visual Basic handles them for you in a bad way -

- typically, it displays an **error message** and then abruptly [náhle] **terminates** the application

**Figure 11-34a** & **Figure 11-34b**

- you can prevent your application from behaving in such as unfriendly manner by taking control of the **exception handling** in your code -

- you can do this by using the statement: **Try...Catch**

**Figure 11-35a** & **Figure 11-35b** & **Figure 11-35c**

### Try...Catch statement syntax:

```
Try
    statements that might generate an exception
    ...
Catch ex As Exception
    statements to execute when an exception occurs
    ...
Finally
    statements to process whether or not an exception is thrown
    ...
End Try
```

**Try** block of code

**Try** block of code

- automatically generated line with a local variable **ex**

**Catch** block of code

**Catch** block of code

*optional* **Finally** block of code

*optional* **Finally** block of code

**Try** block of code      - within this block, you place the code that could possibly generate an exception  
**Try**      = keyword, provides a way to handle possible errors, while still running the code

**Catch** block of code      - when an exception occurs in the **Try** block's code, the computer processes the code contained in this block and then skips to the code following the **End Try** clause

**ex As Exception**      - **As Exception class**

= automatically generated local variable, whose name can be changed

- an exception/error description is stored in **Catch** block's **ex** parameter's **Message** property

- you can access the description using the code: **ex.Message**

- or you can display your own message

**Figure 11-35b**

**Figure 11-35a**

**Finally** block of code      - *optional*  
= introduces a statement block to be run before exiting a **Try** structure  
- always processed, no matter if an exception is thrown or not

**Figure 11-35c**

- e.g. - button's **Click** event procedure wants to open a data stream file named **names.txt** - but the file does not exists  
 - btw there is missing s.s. & function: **If IO.File.Exists("names.txt") Then...** more info in: **CH9\_F3 -**

```

9  Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
10  Dim inFile As IO.StreamReader
11  Try
12      inFile = IO.File.OpenText("names.txt")
13      Do Until inFile.Peek = -1
14          lstNames.Items.Add(inFile.ReadLine)
15      Loop
16      inFile.Close()
17  Catch ex As Exception
18      MessageBox.Show("Sequential file error.", "JK's", MessageBoxButtons.OK, MessageBoxIcon.Information)
19      MessageBox.Show(ex.Message, "JK's", MessageBoxButtons.OK, MessageBoxIcon.Information)
20  Finally
21      MessageBox.Show("Finally block of code - always executed", "JK's", MessageBoxButtons.OK, MessageBoxIcon.Information)
22  End Try
23 End Sub

```

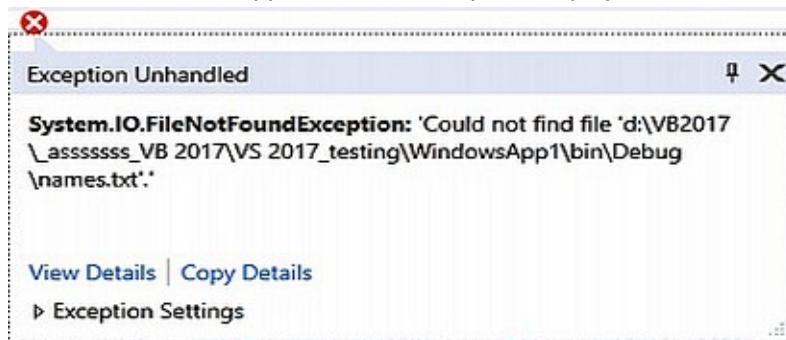
**Figure 11-35a**

**Figure 11-35b**

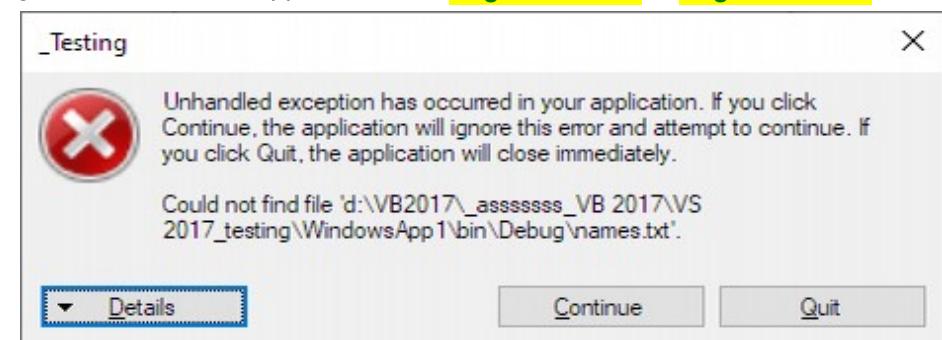
**Figure 11-35c**

- without a **Try...Catch** statement, the app throws an exception/displays an error message & terminates the application:

**Figure 11-34a** & **Figure 11-34b**



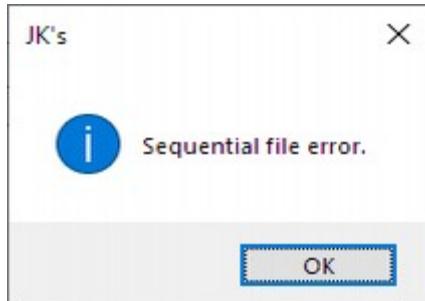
**Figure 11-34a** an exception displayed during testing



**Figure 11-34b** an exception displayed during run time

- in the **Try...Catch** statement, you can set how the app behaves, should an exception occurs:

**Figure 11-35a** & **Figure 11-35b** & **Figure 11-35c**



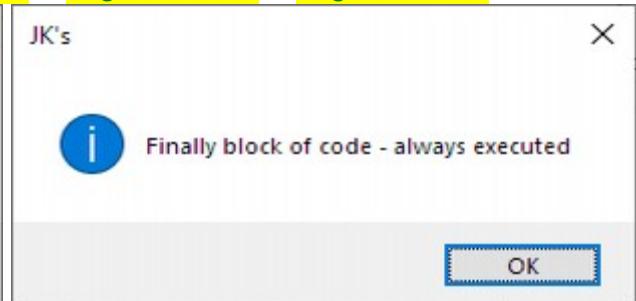
**Figure 11-35a** line 18

- result of block: **Catch**



**Figure 11-35b** line 19

- result of block: **Catch**, using **ex.Message** property



**Figure 11-35c** line 21

- result of block: **Finally**

## CH11 F8.2 - step 8/8: include a Try...Catch "safety net" statement to handle/catch possible exceptions/errors example 01.Course Info Solution

- in this step, you will modify the code automatically generated when you made a bound control in: **step 5.1**
- you gonna include a "safety net" statement **Try...Catch**, for a case there would be an error in your code

- to include a **Try...Catch** statement in the **BindingNavigator's Save Data** button's **Click** event procedure

-> open the: ...VB2017\Chap11\_Exercise\01.Course Info Solution\Course Info Solution.sln

**step 8.1** -> below line **10**, **Private Sub CoursesBindingNavigatorSaveItem\_Click.....** insert a blank line

-> on the new blank line **11** type: **Try** and press **Enter** key

- the **Code Editor** automatically enters the lines **12 - 15: Catch ex As Exception** and **End Try** clauses for you

```
10     Private Sub CoursesBindingNavigatorSaveItem_Click(sender As Object, e As EventArgs) Handles CoursesBindingNavigatorSaveItem.Click
11         Try
12
13             Catch ex As Exception
14
15         End Try
16         Me.Validate()
17         Me.CoursesBindingSource.EndEdit()
18         Me.TableAdapterManager.UpdateAll(Me.MyCoursesDataSet)
...
...
```

**step 8.2** -> move the **3** automatically generated statements - that might generate an exception - into the **Try block of code** - lines **12 - 14**

- if those **3** statements in the **Try block** do not throw an exception, the block should display the confirmation message - line **15**

- if there is an error, the **Catch block** should display a description of the exception - line **17**

-> enter the **MessageBox.Show** methods on the lines **15 & 17**

```
9     Public Class frmMain
10    Private Sub CoursesBindingNavigatorSaveItem_Click(sender As Object, e As EventArgs) Handles CoursesBindingNavigatorSaveItem.Click
11        Try
12            Me.Validate()
13            Me.CoursesBindingSource.EndEdit()
14            Me.TableAdapterManager.UpdateAll(Me.MyCoursesDataSet)
15            MessageBox.Show("Changes saved.", "Course Information", MessageBoxButtons.OK, MessageBoxIcon.Information)
16        Catch ex As Exception
17            MessageBox.Show(ex.Message, "Course Information", MessageBoxButtons.OK, MessageBoxIcon.Information)
18        End Try
19    End Sub
20
21    Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles MyBase.Load
22        'TODO: This line of code loads data into the 'MyCoursesDataSet.Courses' table. You can move, or remove it, as needed.
23        Me.CoursesTableAdapter.Fill(Me.MyCoursesDataSet.Courses)
24    End Sub
25 End Class
```

**step 8.3**

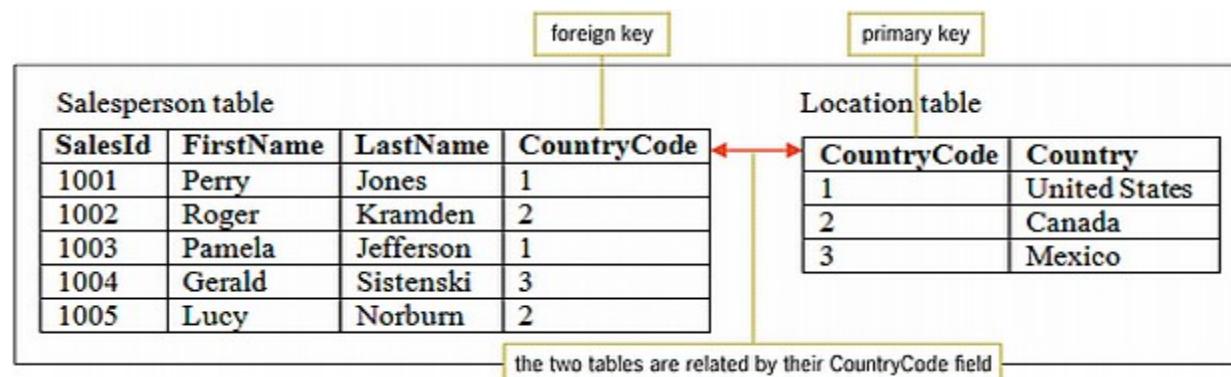
- the statement in the **frmMain\_Load** procedure retrieves the appropriate data from the **MyCourses.mdf** database file and and loads the data into the **MyCoursesDataSet** - line **23**
- > save the solution and then start and test the application
- > delete the record number 7, and then click the button **Save Data**
  - if you would just delete the record and close the application without saving, the next time you start the application, the record would be still there
- the "Changes saved." message appears in a message box
- > stop the application and then start it again to verify that the **PSYCH100** record is no longer in the dataset
- > stop the application & close the solution

|   | ID | Code     | Title                       | Hours | Grade |
|---|----|----------|-----------------------------|-------|-------|
| ▶ | 1  | ACCOU110 | Accounting Procedures       | 3     | A     |
|   | 2  | ENG101   | English Composition         | 3     | W     |
|   | 3  | CIS110   | Introduction to Programming | 3     | A     |
|   | 4  | BIO111   | Environmental Biology       | 3     | C     |
|   | 5  | ENG101   | English Composition         | 3     | B     |
|   | 6  | CIS112   | The Internet                | 2     | A     |
|   | 7  | PSYCH100 | General Psychology          | 3     | C     |
| * |    |          |                             |       |       |

|   | ID | Code     | Title                       | Hours | Grade |
|---|----|----------|-----------------------------|-------|-------|
| ▶ | 1  | ACCOU110 | Accounting Procedures       | 3     | A     |
|   | 2  | ENG101   | English Composition         | 3     | W     |
|   | 3  | CIS110   | Introduction to Programming | 3     | A     |
|   | 4  | BIO111   | Environmental Biology       | 3     | C     |
|   | 5  | ENG101   | English Composition         | 3     | B     |
|   | 6  | CIS112   | The Internet                | 2     | A     |
| * |    |          |                             |       |       |

## CH11\_F9 - explore the Two-table database: basic info with example 02.Charleston Sales Solution

- as you could see in: **CH11\_F0 - Relational database: ...** & **e.g.2** a database example, that contains 2 tables: - 1). **Salesperson** table, and - 2). **Location** table



- a **Salesperson table**:  
- a child table  
- primary key = **SalesId** field  
- foreign key = **CountryCode** field

- a **Location table**:  
- a parent table  
- primary key = **CountryCode** field

- foreign key from **Salesperson** table and primary key from **Location** table create a **relationship** between the **2** tables - the **common field** named **CountryCode**  
- storing the country codes in a separate table has the following advantages:

- it allows the input clerk to enter a number, rather than the country name, in each record in the Salesperson table, resulting in less chance of a typing error
- also, the Salesperson table will require less of the computer's main memory to store a number in each record rater than storing the country name

- you can see the result after: **step 4.x** in: **Figure 11-39j**

The screenshot shows a Windows application window titled "Charleston Sales". The window contains a **SalespersonDataGridView** control. The grid has the following data:

|   | Sales ID | First Name | Last Name | Country       |
|---|----------|------------|-----------|---------------|
| ▶ | 1001     | Perry      | Jones     | United States |
|   | 1002     | Roger      | Kramden   | Canada        |
|   | 1003     | Pamela     | Jefferson | United States |
|   | 1004     | Gerald     | Sistenski | Mexico        |
|   | 1005     | Lucy       | Norburn   | Canada        |

**Figure 11-39j** information from both tables displayed in the **SalespersonDataGridView** control

**CH11\_F9.1 - step 1/4:** add an existing database file **Charleston.mdf** to an application & create a **dataset** && add its **2 tables** to the **dataset** example **02.Charleston Sales Solution**

- first, you need to open the **Charleston Sales** application and connect it to the **Charleston.mdf** database file = create a **dataset**

- to connect your application with an existing \*.mdf database files located somewhere in HDD example **02.Charleston Sales Solution**

-> open the: ...VB2017\Chap11\_Exercise\02.Charleston Sales Solution\Charleston Sales Solution.sln

-> open the **Designer** window, and display the **Data Sources** window (VS menu bar View / Other Windows / Data Sources Shift+Alt+D)

**step 1.1** -> in **Data Sources** window, click [Add New Data Source...](#) to start the **Data Source Configuration Wizard**, described in: **CH11\_F4 - & step 4.x**

**step 1.2** -> **Data Source Configuration Wizard** / screen **Choose a Data Source Type** -> = **Database**

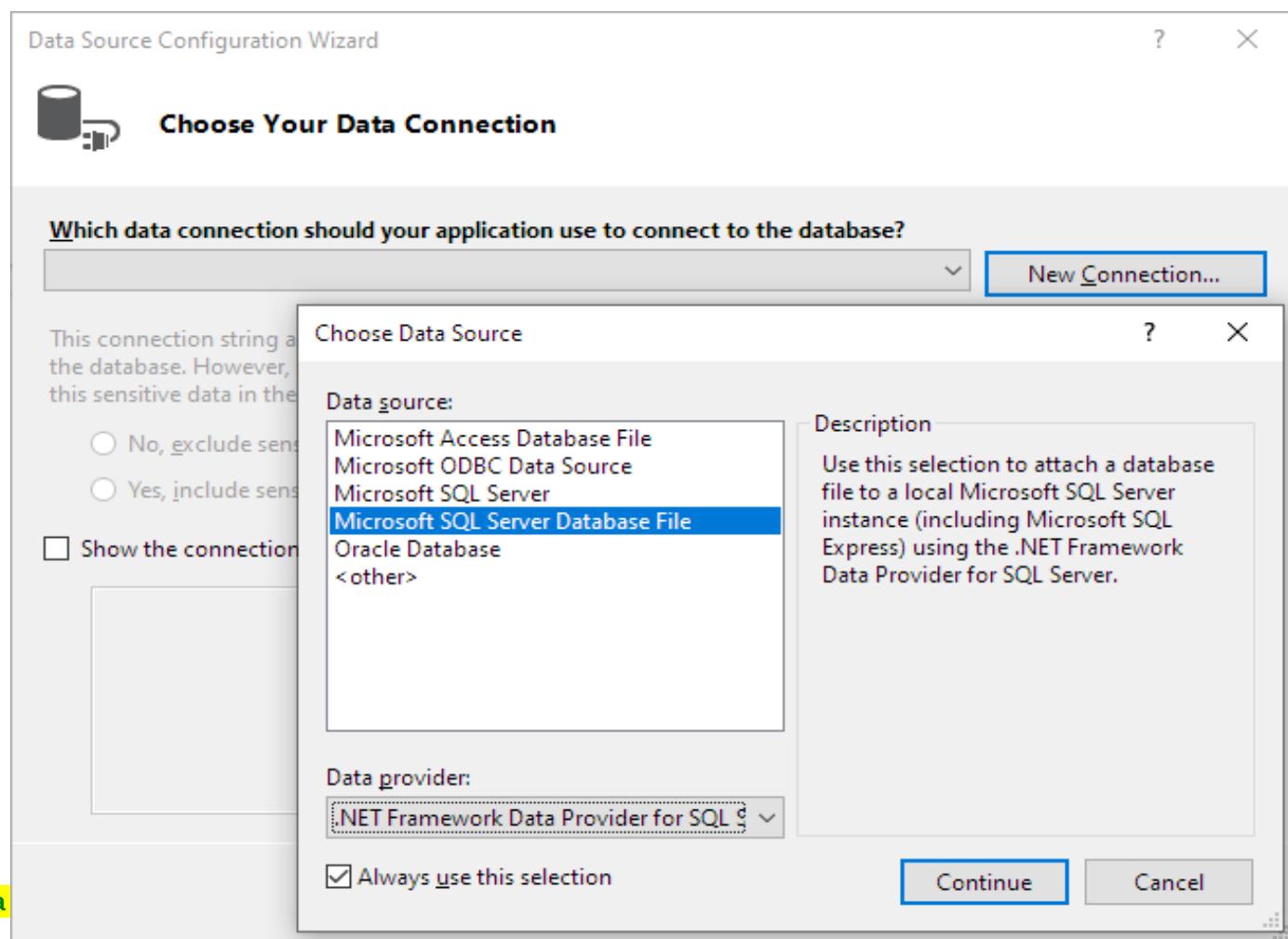
**step 1.3** -> **Data Source Configuration Wizard** / screen **Choose a Database Model** -> = **Dataset**

**step 1.4** -> **Data Source Configuration Wizard** / screen **Choose Your Data Connection** -> click the button [New Connection...](#) **Figure 11-36a**

**step 1.4.1** - one of the dialog boxes opens: either **Choose Data Source** dialog box, or **Add Connection** dialog box **Figure 11-36a** or **Figure 11-36b**

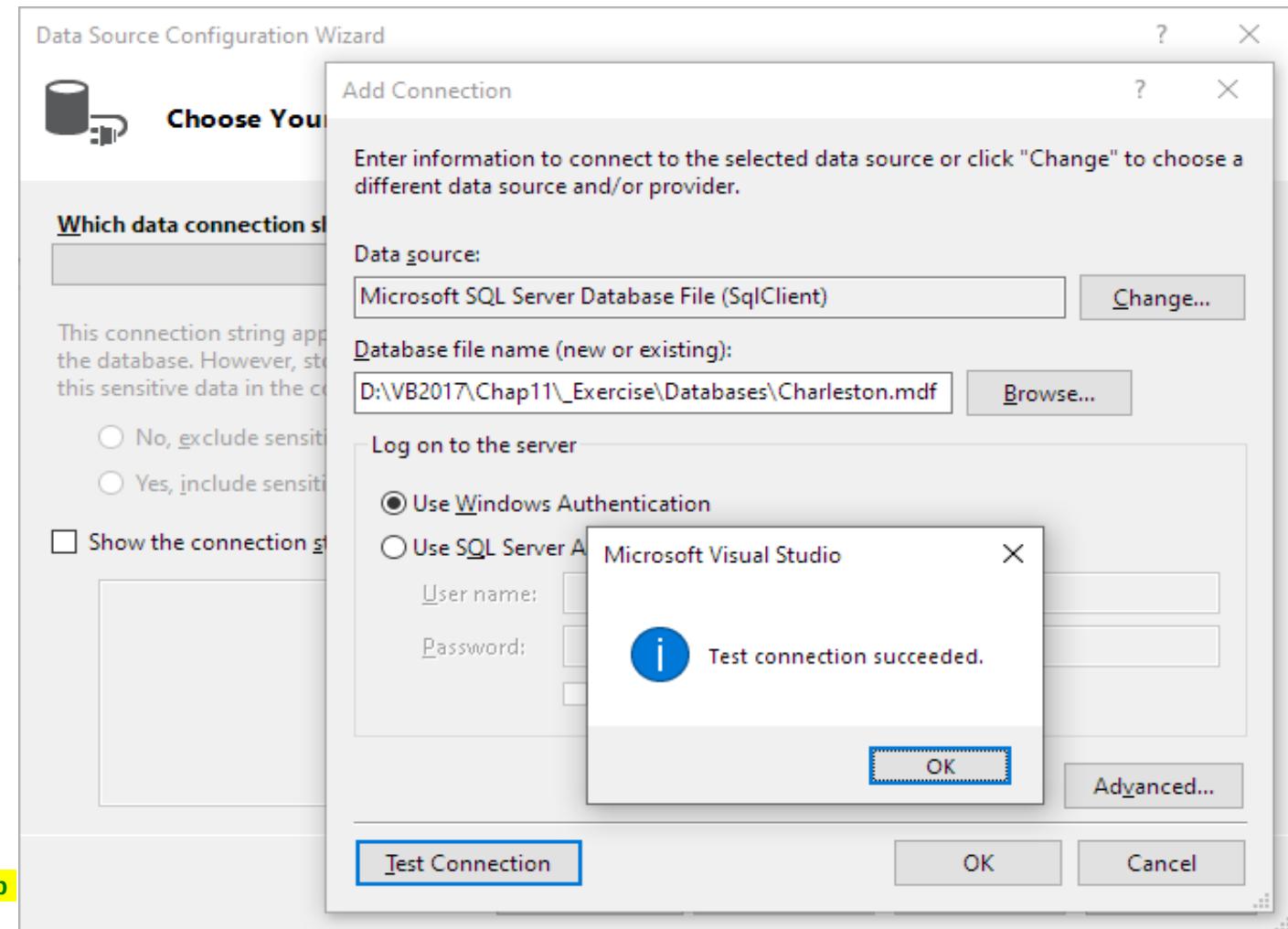
- if **Choose Data Source** dialog box opens:

-> choose **Microsoft SQL Server Database File**, and then click the button **Continue** **Figure 11-36a**

**Figure 11-36a**

- step 1.4.2** - a new dialog box **Add Connection** opens: **Figure 11-36b**  
-> verify that the box **Data source:** contains: **Microsoft SQL Server Database File (SqlClient)**  
-> if it does **not**, click the button **Change...** and choose the **Microsoft SQL Server Database File (SqlClient)**

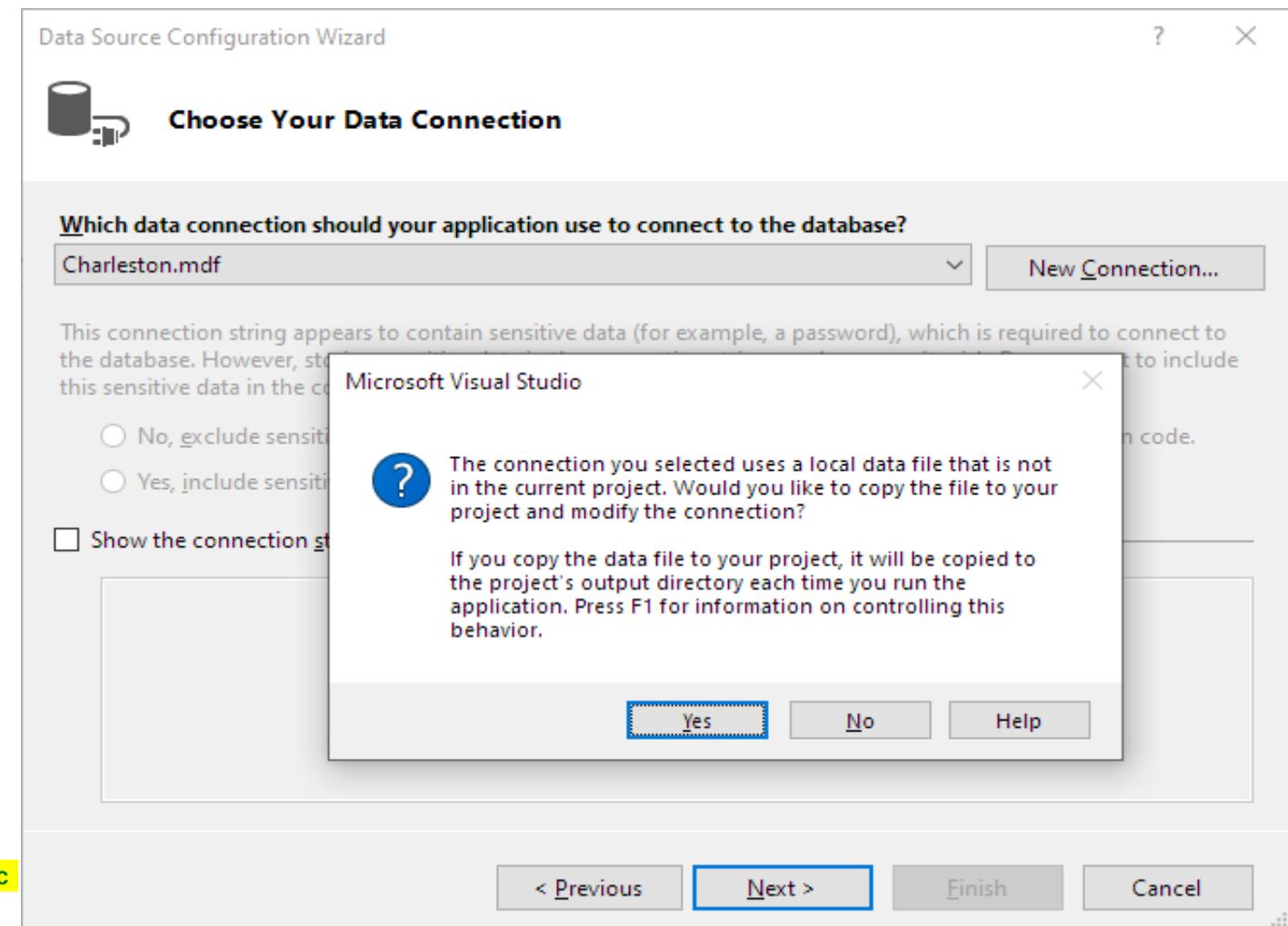
- step 1.4.3** -> click the button **Browse...** to add the **Charleston.mdf** database file from your location **Figure 11-36b**  
-> click the button **Test Connection**  
- the message "Test connection succeeded." appears in a message box **Figure 11-36b**  
-> you can close the message box  
-> in a dialog box **Add Connection**, click the button **OK**



**Figure 11-36b**

**step 1.5**

- in Data Source Configuration Wizard / screen **Choose Your Data Connection** -> will now appear **Charleston.mdf** database file
  - > click the button **Next >**
  - the message box named "Microsoft Visual Studio" opens and asks, whether you want to include the database file in the current project
    - by including the file in the current project, you can more easily copy the application and its database to another computer
  - > click the button **Yes**, to add the **Charleston.mdf** database file to the application's project folder

**Figure 11-36c**

**step 1.6**

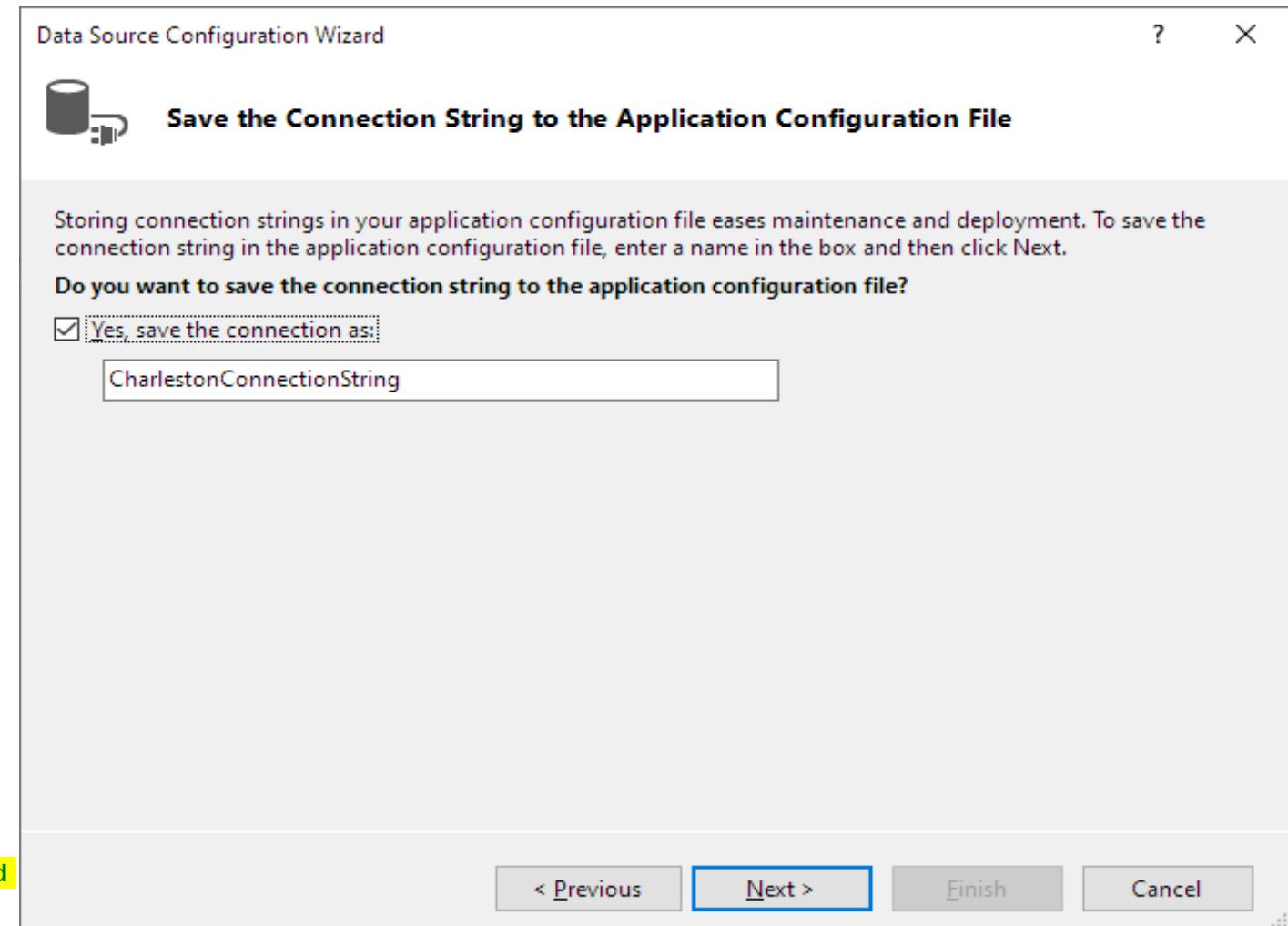
- the Data Source Configuration Wizard / screen Save the Connection String to the Application Configuration File opens

**Figure 11-36d**

- it displays the name of the connection string: **CharlestonConnectionString**

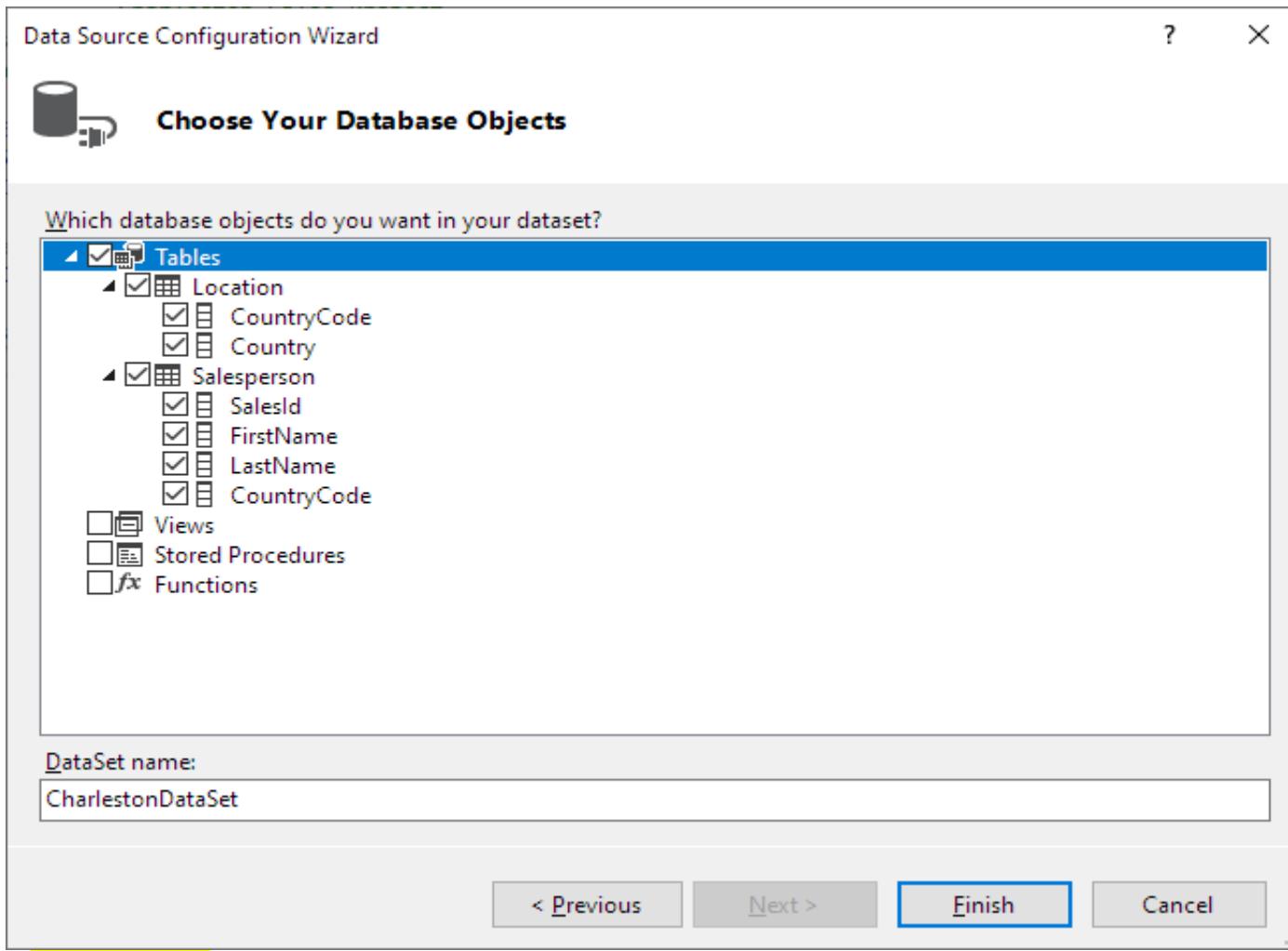
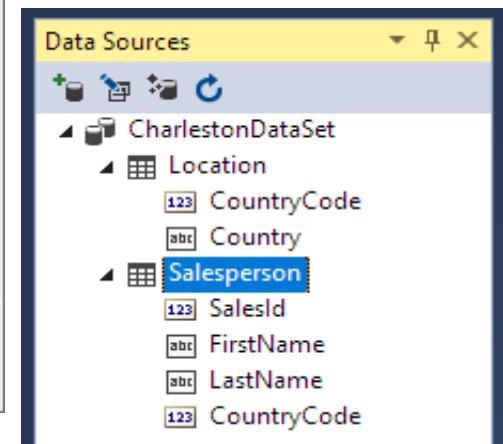
- similar to **CH11\_F4 - step 4.5**

-> verify that check box **Yes...** is selected and click the button **Next >**

**Figure 11-36d****Figure 11-36d**

**step 1.7**

- the Data Source Configuration Wizard / screen **Choose Your Database Objects** appears **Figure 11-36e**
- as you learned in **CH11\_F4 - step 4.6** you use this screen to select the database objects you want to include in your **dataset**
- the default name for this dataset is **CharlestonDataSet**, no need to change the name
- > expand the node **Tables** to see the **2 tables** and its **fields**, included within already created **Charleston.mdf** database
  - there is a **table** object named **Location**, including **field** objects: **CountryCode**, **Country**
  - there is a **table** object named **Salesperson**, including **field** objects: **SalesId**, **FirstName**, **LastName**, **CountryCode**
- > click the check box next to the node **Tables** to select both of them, including their fields **Figure 11-36e**
- > click the button **Finish** **Figure 11-36e**

**Figure 11-36e****Figure 11-36f****step 1.8**

- the computer adds to the **Data Sources** window a new dataset named **CharlestonDataSet**, containing **2 table** objects:
  - a **Location** table, and
  - a **Salesperson** table
- > expand the nodes **Location** and **Salesperson** just to view field objects **Figure 11-36f**

## CH11\_F9.2 - step 2/4: relate 2 tables by their **common field**: CountryCode example 02.Charleston Sales Solution

- now, that both tables are in the dataset, you can tell the computer to **relate** them by their **common field**: **CountryCode** field

- to **relate** both tables by their **common field**: **CountryCode** field, example 02.Charleston Sales Solution

-> open the: ...VB2017\Chap11\Exercise\02.Charleston Sales Solution\Charleston Sales Solution.sln

**step 2.1** -> in **Data Sources** window, Rclick **CharlestonDataSet** and then click: **Edit DataSet with Designer**

- the **DataSet Designer** window opens

Figure 11-37b

Figure 11-37a

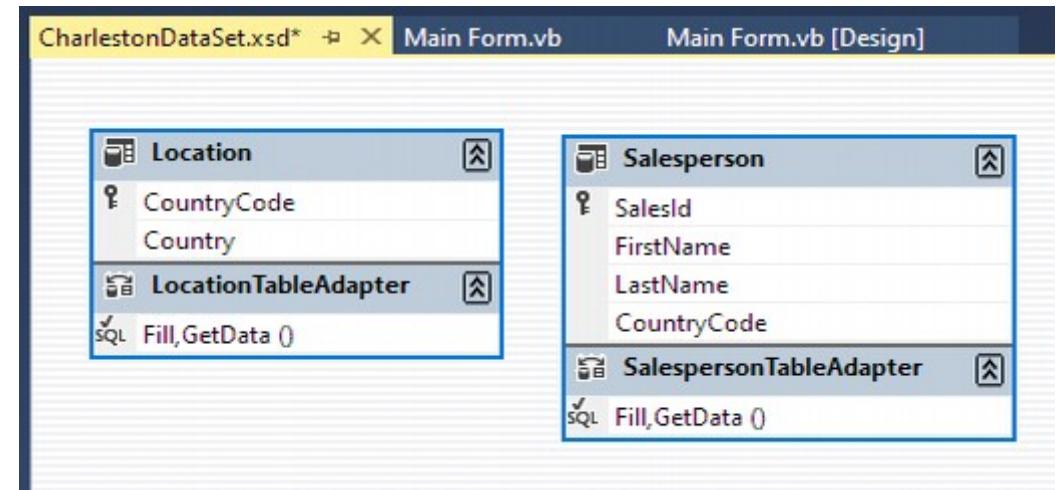
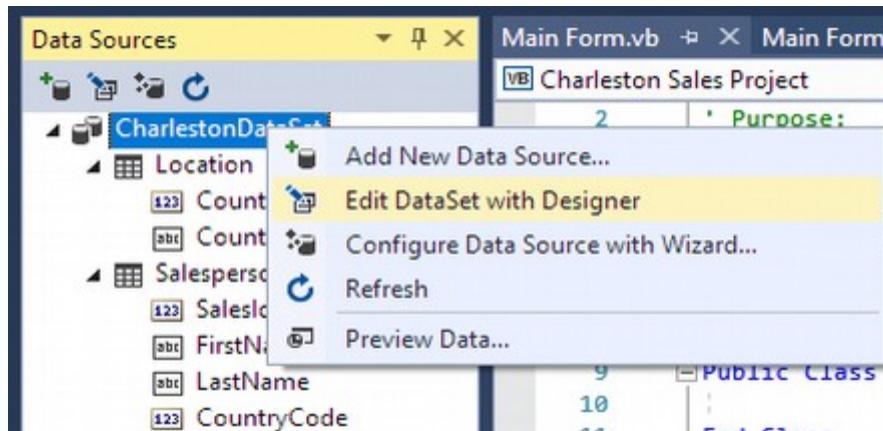


Figure 11-37b

**step 2.2** -> in the **DataSet Designer** window **CharlestonDataSet.xsd**, -> Rclick an empty area

-> click **Add / Relation...** to open the **Relation** dialog box

Figure 11-37d

Figure 11-37c

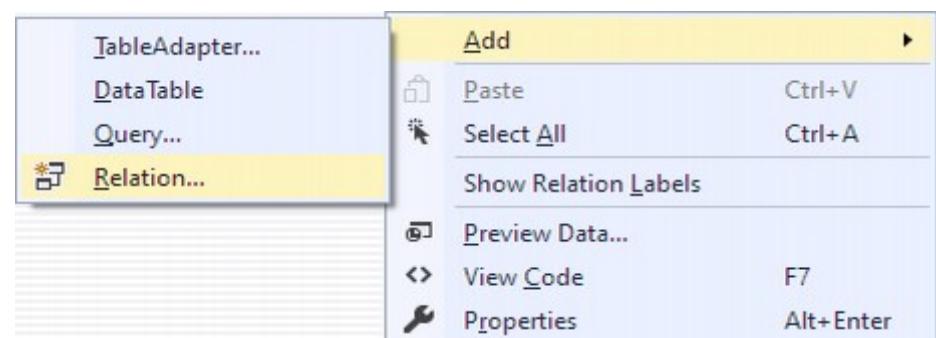


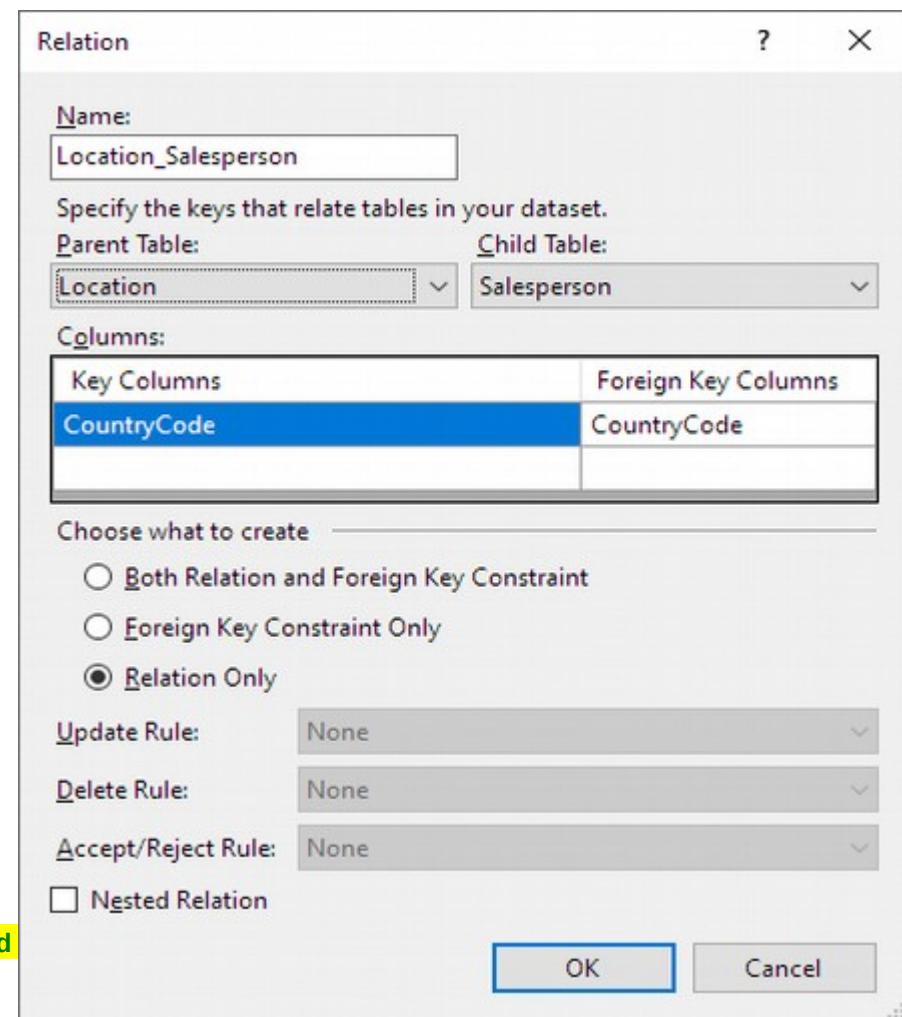
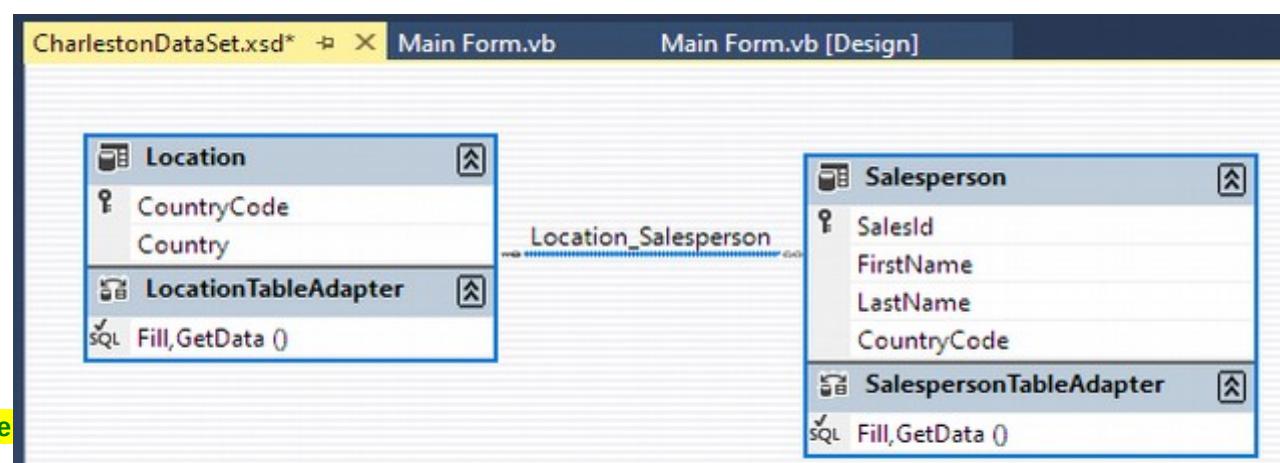
Figure 11-37c

**step 2.3**

- see **Figure 11-37d**
  - **Name** box: = shows the names of the **two tables** you want to **relate**
    - table **Location** and table **Salesperson**
  - **Parent Table** box: = the table with a **primary key**
    - the table named **Location**,
    - because the field **CountryCode** is the **primary key** in that table
  - **Child Table** box: = the table with a **foreign key**
    - the table named **Salesperson**,
    - because the field **CountryCode** is the **foreign key** in that table
- > click the **OK** button to confirm

**see **Figure 11-37e****

- the line between both tables in the **DataSet Designer** window indicates that a **one-to-many** relationship exists between the tables
- > save the solution

**Figure 11-37d****Figure 11-37e**

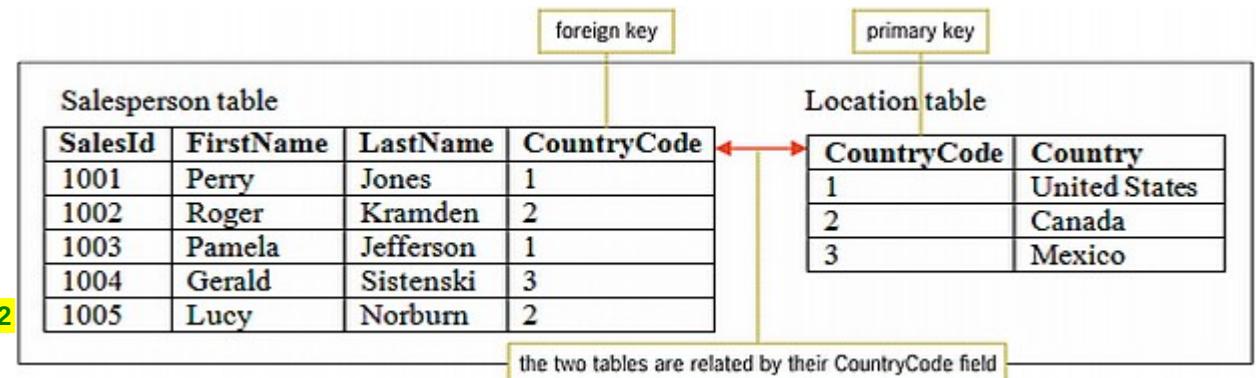


Figure 11-2

- in the:

**step 4.x**

- , you will use a **DataGridView** control to display: **Figure 11-39j**
- the **SalesId**, **FirstName**, and **LastName** information from the table **Salesperson**, along with **Figure 11-2**
  - the **Country** information from the table **Location** **Figure 11-2**
  - to accomplish this, you will need to open the **Query Builder** dialog box and create a **database query/query**

-> **database query/query:**

- = a statement/task that allows you to retrieve/filter out specific information from a database, using a special language called **SQL**  
 - you can use **query task** to specify the fields and records you want to display

-> **query object:** = result of **query task**

- enable you to retrieve records from one or more tables and then combine the data into rows and columns in a single dataset
- can also perform calculations on data
- there are 2 types: **1). normal** - can be used to display data in user interface  
**2. API** - used to generate web service endpoints and can not be displayed in user interface

e.g. a single dataset as a combination of fields from 2 tables, like in: **02.Charleston Sales Solution** **Figure 11-39j**

-> **SQL/Structured Query Language:**

- = a special language used to create the database queries  
 = contains statements that allow you to retrieve and manipulate the data stored in databases  
 - more about **SQL** in: **Chapter 12 Database Queries with SQL**

- to create a query example 02.Charleston Sales Solution

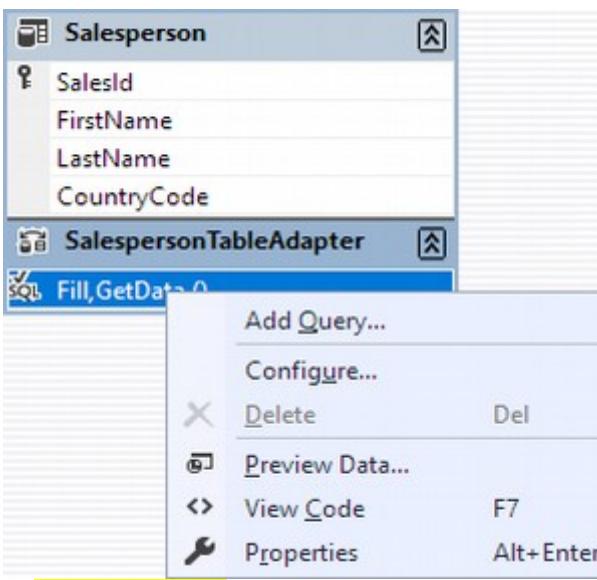
-> open the: ...VB2017\Chap11\Exercise\02.Charleston Sales Solution\Charleston Sales Solution.sln

step 3.1 -> in the **DataSet Designer** window named **CharlestonDataSet.xsd** click the **SalespersonTableAdapter** **Figure 11-38a**

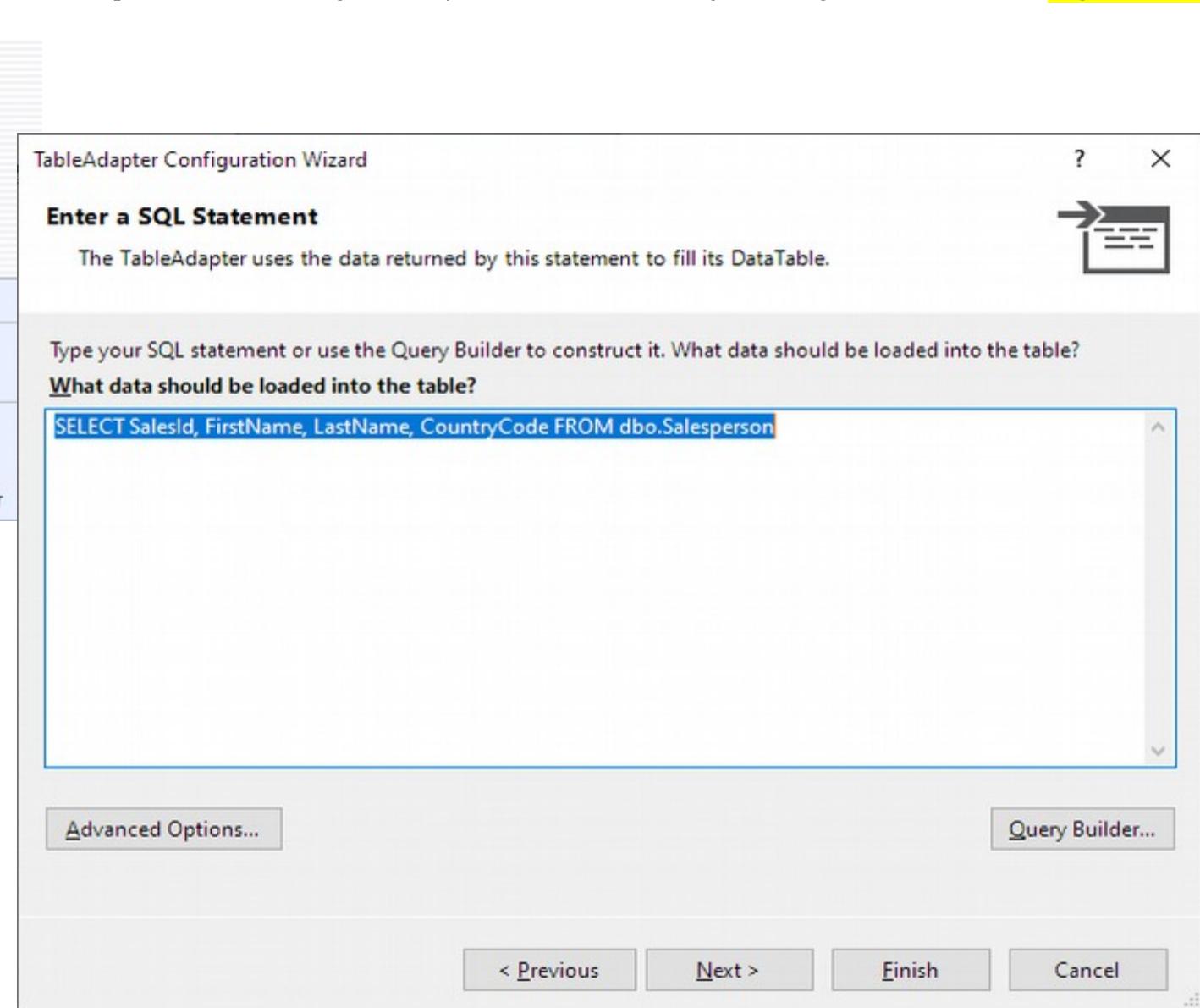
**Figure 11-38a**

**Figure 11-38b**

-> in the row under, Rclick the line: **Fill,GetData()** and then click **Configure...** to open the screen: **TableAdapter Configuration Wizard**



**Figure 11-38a**



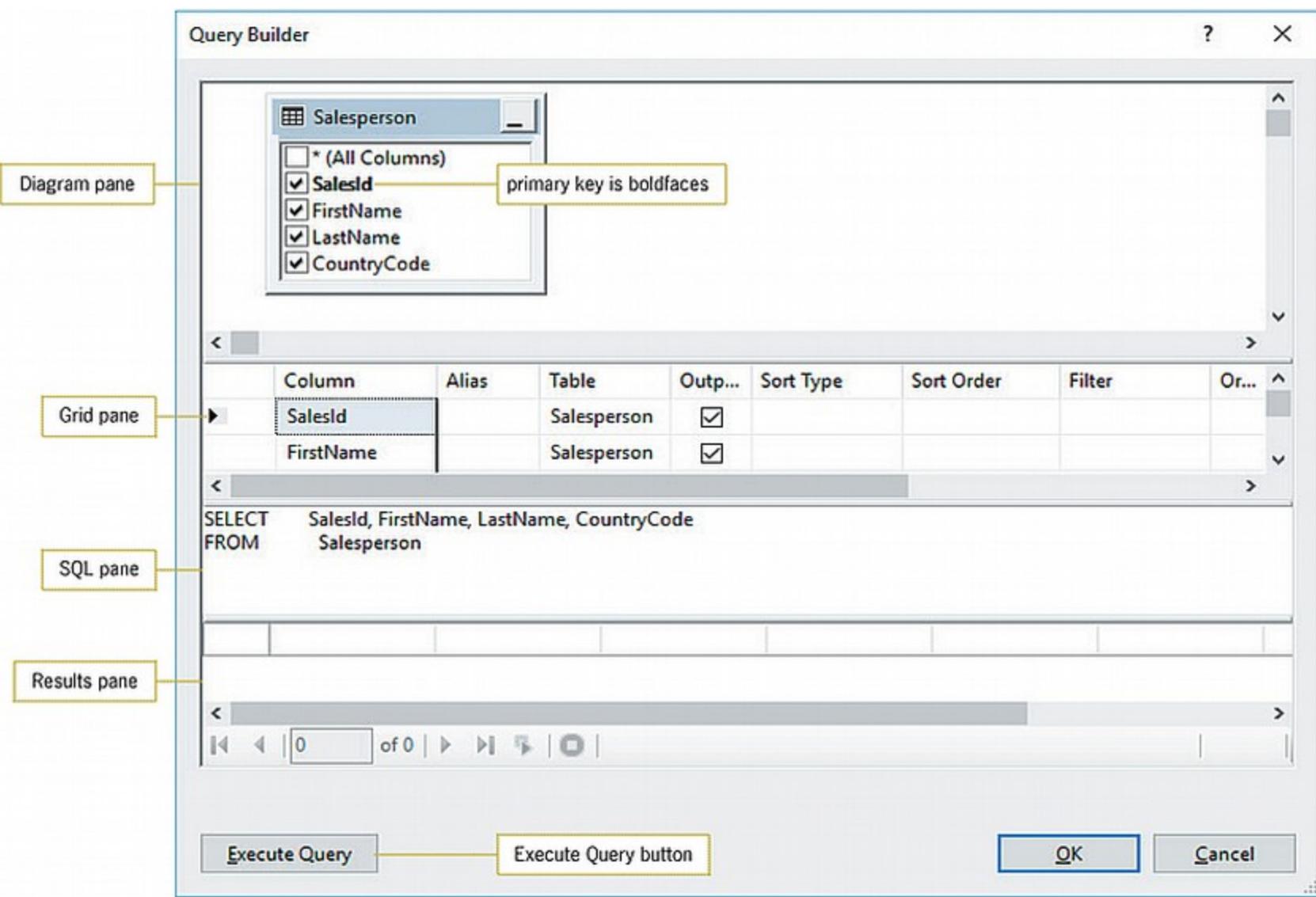
**Figure 11-38b**

step 3.2

-> in **TableAdapter Configuration Wizard** screen, click the button: **Query Builder...** to open the **Query Builder** dialog box

**Figure 11-38c**

- the Salesperson table's primary key (SalesId) appears boldfaced in the Diagram pane



**Figure 11-38c**

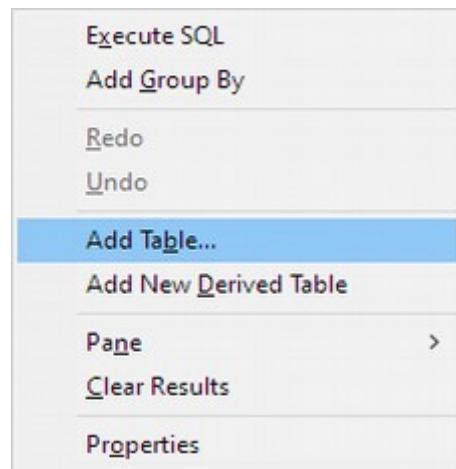
**step 3.3** -> in the **Diagram** pane, Rclick an empty area and then click **Add Table** to open the **Add Table** dialog box

**Figure 11-38d** & **Figure 11-38e**

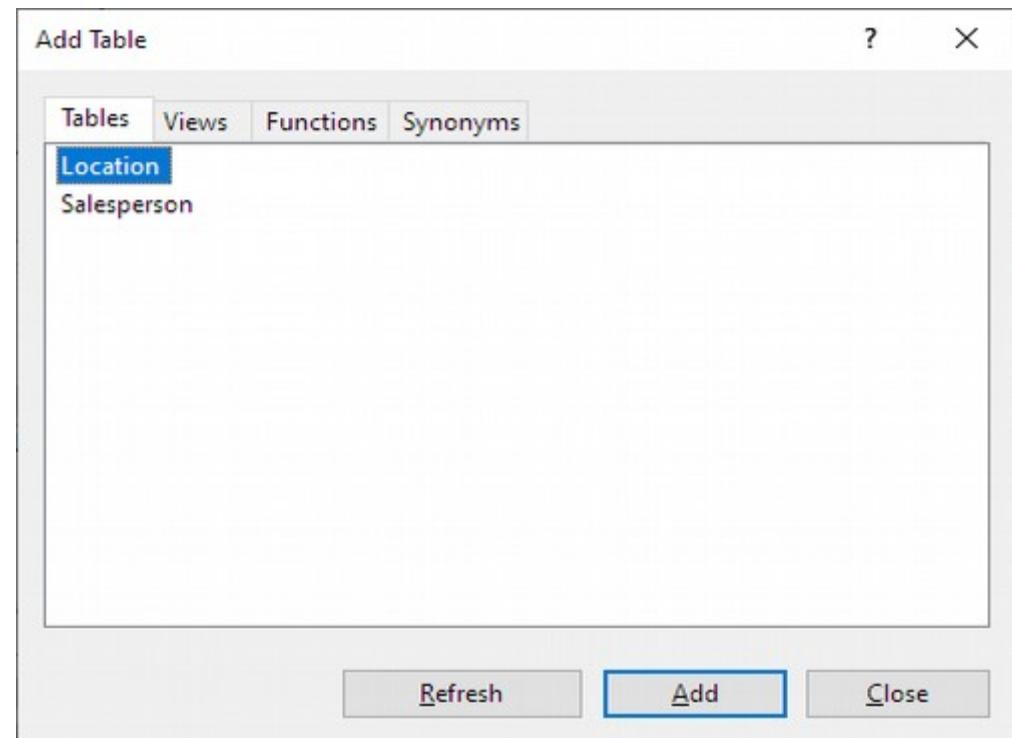
- on the **Tables** tab, the table **Location** is selected

**Figure 11-38e**

-> click the button **Add** and then click the **Close** button to close the **Add Table** dialog box



**Figure 11-38d**



**Figure 11-38e**

**step 3.4**

- in the **Diagram** pane, the table **Location** now appears next to the table **Salesperson**
- notice that its primary key **CountryCode** appears boldfaced

**Figure 11-38f**

Query Builder

**Salesperson**

- \* (All Columns)
- SalesId**
- FirstName**
- LastName**
- CountryCode**

**Location**

- \* (All Columns)
- CountryCode**
- Country

Column Alias Table Outp... Sort Type Sort Order Filter

|             |  |             |                                     |  |  |  |
|-------------|--|-------------|-------------------------------------|--|--|--|
| SalesId     |  | Salesperson | <input checked="" type="checkbox"/> |  |  |  |
| FirstName   |  | Salesperson | <input checked="" type="checkbox"/> |  |  |  |
| LastName    |  | Salesperson | <input checked="" type="checkbox"/> |  |  |  |
| CountryCode |  | Salesperson | <input checked="" type="checkbox"/> |  |  |  |
|             |  |             | <input type="checkbox"/>            |  |  |  |
|             |  |             | <input type="checkbox"/>            |  |  |  |
|             |  |             | <input type="checkbox"/>            |  |  |  |
|             |  |             | <input type="checkbox"/>            |  |  |  |

```
SELECT Salesperson.SalesId, Salesperson.FirstName, Salesperson.LastName, Salesperson.CountryCode  
FROM Salesperson INNER JOIN  
Location ON Salesperson.CountryCode = Location.CountryCode
```

Execute Query OK Cancel

Figure 11-38f

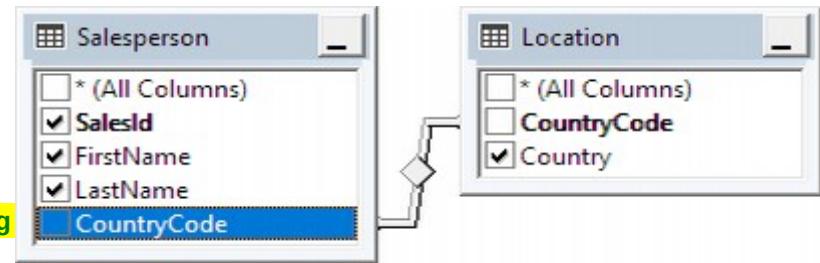
**step 3.5**

-> in the table **Location**, click the check box **Country** to select the **Country** field

**Figure 11-38g**

-> in the table **Salesperson**, click the check box **CountryCode** to deselect the **CountryCode** field

**Figure 11-38g**



**Figure 11-38g**

**step 3.6**

-> click the button **Execute Query**

**Figure 11-38h**

- the **Results** pane displays the **SalesId**, **FirstName**, and **LastName** fields for each record in the table **Salesperson**, as well as  
- each record's corresponding **Country** field from the table **Location**

**Figure 11-38h**

The screenshot shows the results of the executed query. The data is presented in a grid format with the following information:

|   | SalesId | FirstName | LastName  | Country       |
|---|---------|-----------|-----------|---------------|
| ▶ | 1001    | Perry     | Jones     | United States |
|   | 1002    | Roger     | Kramden   | Canada        |
|   | 1003    | Pamela    | Jefferson | United States |
|   | 1004    | Gerald    | Sistenski | Mexico        |
|   | 1005    | Lucy      | Norburn   | Canada        |

Below the grid, there is a message: "Cell is Read Only." At the bottom of the window, there are buttons for "Execute Query", "OK", and "Cancel".

**Figure 11-38h**

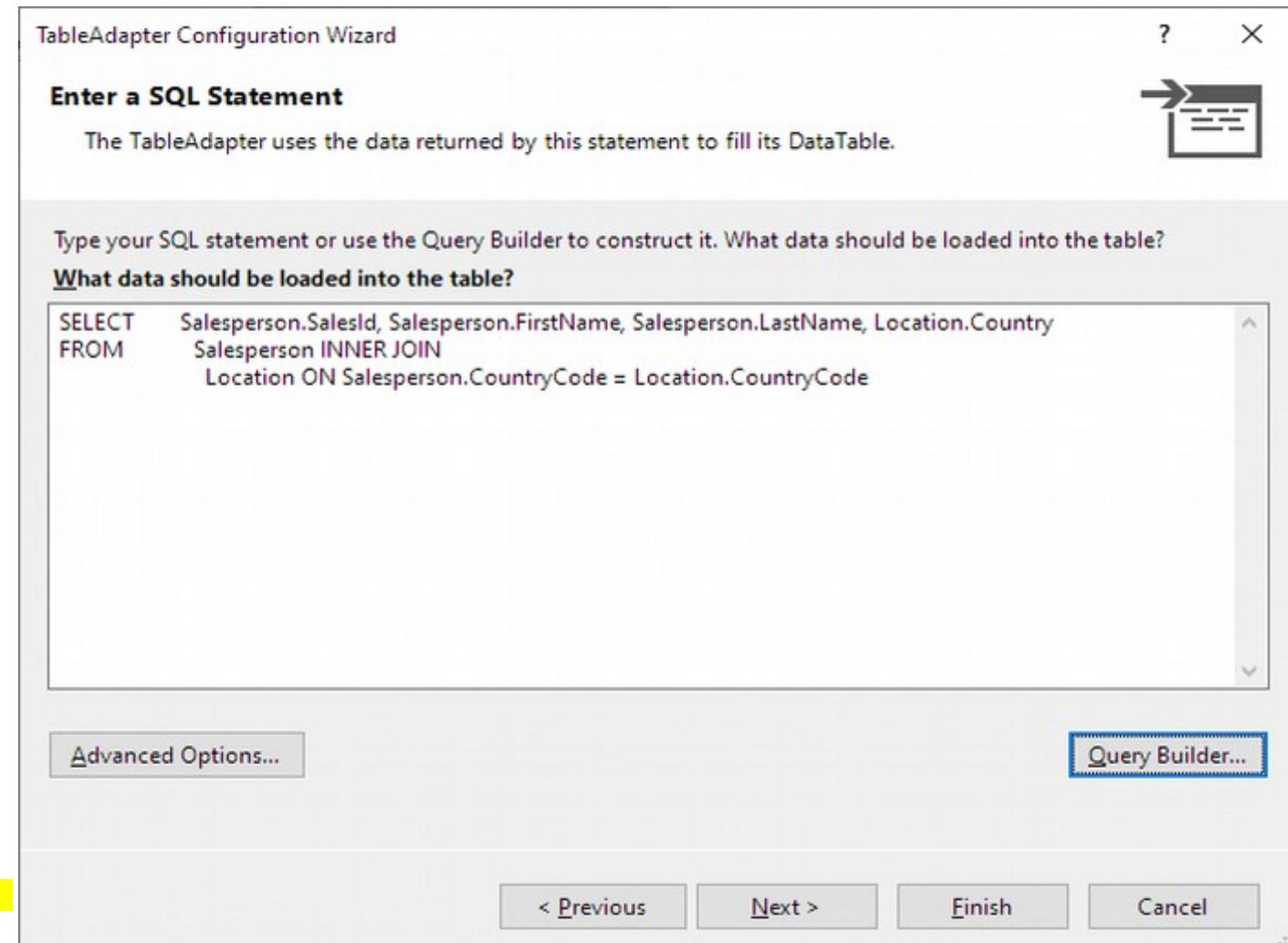
**step 3.7**

-> click the button **OK** to close the **Query Builder** dialog box

-> compare the screen **TableAdapter Configuration Wizard** in:

**Figure 11-38i**

with a previous screen: **Figure 11-38b** from: **step 3.1**



**Figure 11-38i**

**Figure 11-38b**

Type your SQL statement or use the Query Builder to construct it. What data should be loaded into the table?

What data should be loaded into the table?

`SELECT SalesId, FirstName, LastName, CountryCode FROM dbo.Salesperson`

-> click the button **Finish** to close the **TableAdapter Configuration Wizard**

-> save the solution, you can close the window **DataSet Designer**, named **CharlestonDataSet.xsd**

**step 3.8**

-> in the window **Data Sources**, expand the dataset named **CharlestonDataSet** and expand the nodes/tables **Location** and **Salesperson**

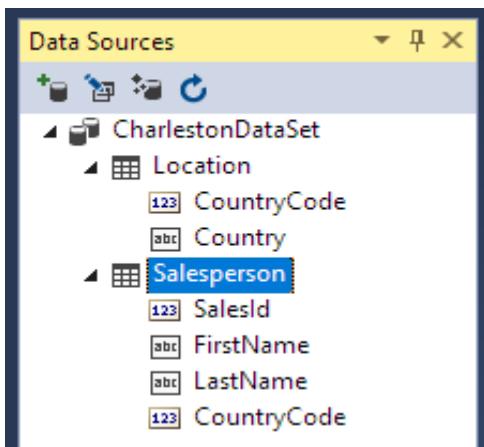
- notice that the **Salesperson** entry now contains: - **SalesId**, **FirstName**, and **LastName** fields from the table **Salesperson**, and

- **Country** field from the table **Location**

**Figure 11-38j**

- so you can see the result of the **query** in: **Figure 11-38j**

-> just compare with a previous entry from: **step 1.7** & **Figure 11-36f**



**Figure 11-36f** original 2 tables

original dataset: **CharlestonDataSet**

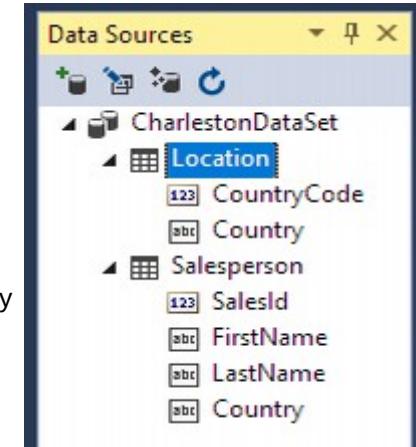
- table **Salesperson**:

- fields: - **SalesId** = primary key
  - **FirstName**
  - **LastName**
  - **CountryCode** = foreign key
- table **Location**:
- fields: - **CountryCode** = primary key
  - **Country**

"custom" dataset: **CharlestonDataSet**

- table **Salesperson**:

- fields: - **SalesId** = primary key
  - **FirstName**
  - **LastName**
  - **Country** = **Location's** field
- table **Location**:
- fields: - **CountryCode** = primary key
  - **Country**



**Figure 11-38j** after Query

**CH11\_F9.4 - step 4/4: display the Database Query/Query information in the DataGridView control = create a bound control from your "custom" dataset & & improve control's appearance - info & example 02.Charleston Sales Solution**

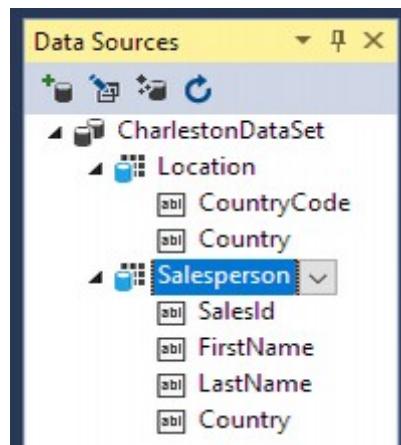
- you will use a **DataGridView** control to display the result of the **query**, created in: **CH11\_F9.3 - step 3/4: create a Database Query ...** & **Figure 11-38j**
- in other words: you will create a **DataGridView** bound control, connected to a table object named **Salesperson**, which has a field named **CountryCode**, "linked" through a foreign key to a primary key of a table **Location**
- more info about **binding objects / creating a bound control** in previous: **CH11\_F5 -** & **CH11\_F5.1 -** & **step 5.1**
- to display in the **DataGridView** control the **query** information created in: **CH11\_F9.3 - step 3/x:** / to bind your query object to the **DataGridView** control & & improve the appearance of the **DataGridView** control: example **02.Charleston Sales Solution**

-> open the: ...VB2017\Chap11\Exercise\02.Charleston Sales Solution\Charleston Sales Solution.sln

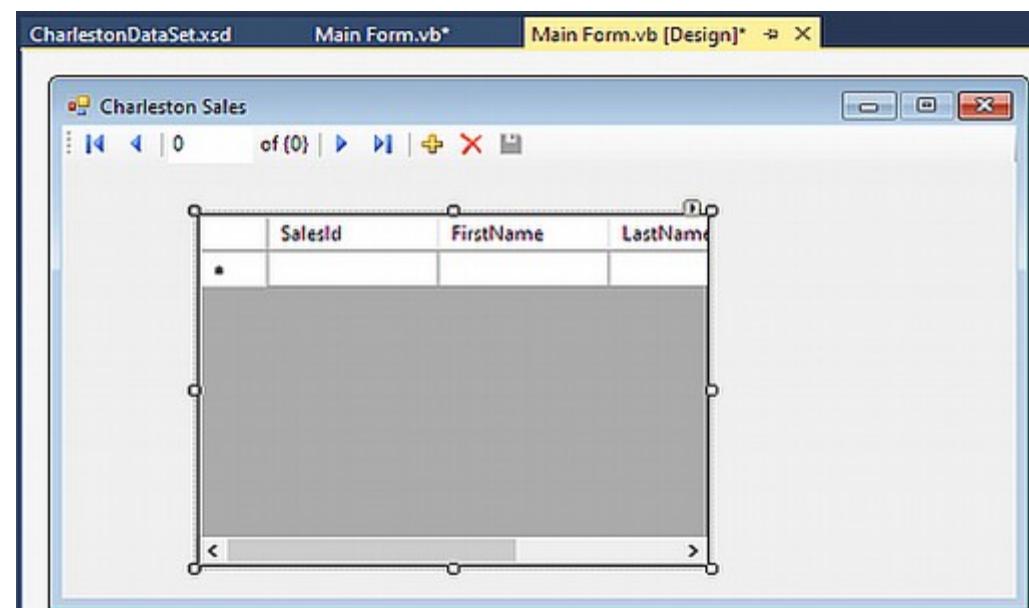
**step 4.1**

- you will create a **DataGridView bound** control named **SalespersonDataGridView**, by the similar way like in: **CH11\_F5.1 -** & **step 5.x**

-> in the window **Data Sources** click **Salesperson** table object and drag it to the center of the form's **Designer** window named **Main Form.vb [Design]**  
**Figure 11-39a** & **Figure 11-39b**



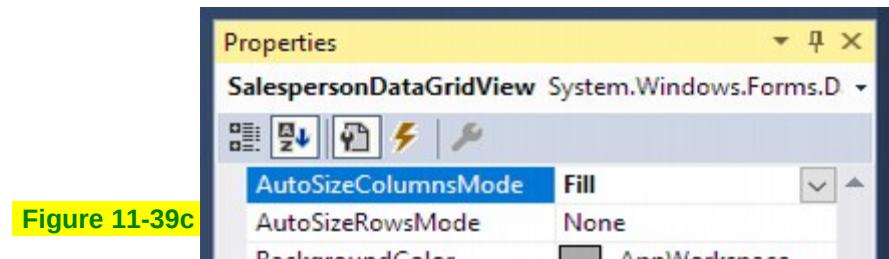
**Figure 11-39a**



**Figure 11-39b**

**step 4.2**

- you will improve the appearance of the control by the similar way like in previous: **CH11\_F6 -** & **CH11\_F6.1 -** & **step 6.1**
- > in the **Designer window**, click the **SalespersonDataGridView** control to select it, and view its Properties in the **Properties window**
- > in the **Properties window**, change default **AutoSizeColumnsMode = None** to -> = **Fill** **Figure 11-39c**

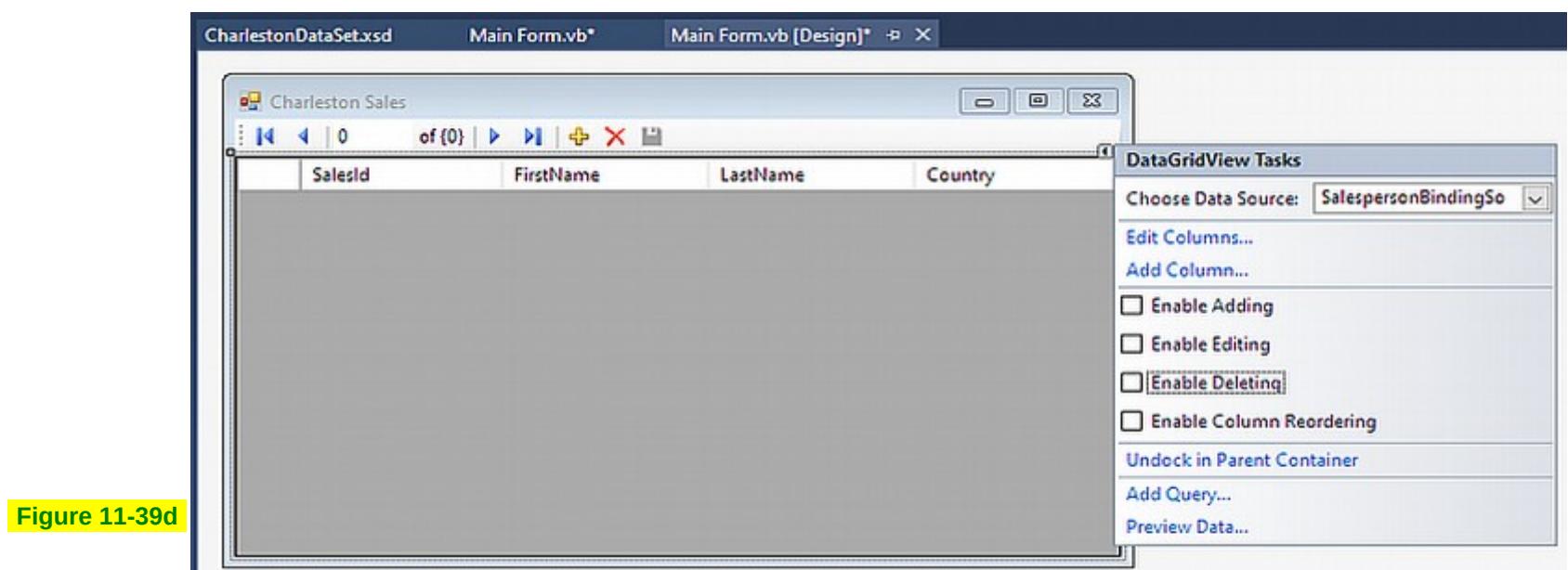
**Figure 11-39c****step 4.3**

- you will improve the appearance of the control by the similar way like in previous: **CH11\_F6 -** & **CH11\_F6.1 -** & **step 6.2**
- next, you will set some of the **SalespersonDataGridView** control's tasks:
- > in the **Designer window**, click the **SalespersonDataGridView** control's **task box** and there: **Figure 11-39d**

**step 4.3.1** -> click the **Dock in parent container**, to spread the table using all available space in your form

**Figure 11-39d**

**step 4.3.2** - in this application, the **DataGridView** control will only display the records - the user will not be allowed to **add**, **edit**, or **delete** records, so:  
-> uncheck the 3 check boxes: **Enable Adding**, **Enable Editing**, and **Enable Deleting** **Figure 11-39d**

**Figure 11-39d**

**step 4.3.3** -> click **Edit Columns...** to open the **Edit Columns** dialog box, and click the **Alphabetical** button to view the settings in that kind of order

**step 4.3.3.1** - you will change the caption text on the header cell of the first 3 columns: **SalesId**, **FirstName**, and **LastName**:

-> use the **HeaderText** property of each of the first 3 columns to change the headings:

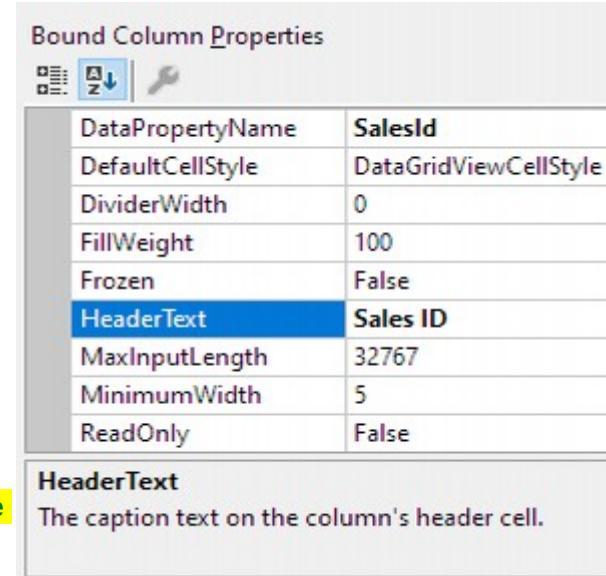
-> column **SalesId** / property **HeaderText** from default = **SalesId** change to -> = **Sales ID** **Figure 11-39e**

-> column **FirstName** / property **HeaderText** from default = **FirstName** change to -> = **First Name**

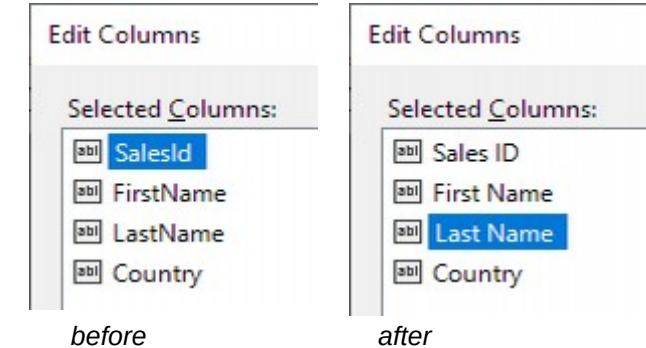
-> column **LastName** / property **HeaderText** from default = **LastName** change to -> = **Last Name**

- notice that the names in **Select Columns:** box also changed **Figure 11-39f**

-> click the button **OK** to close the **Edit Columns** dialog box



**Figure 11-39e**

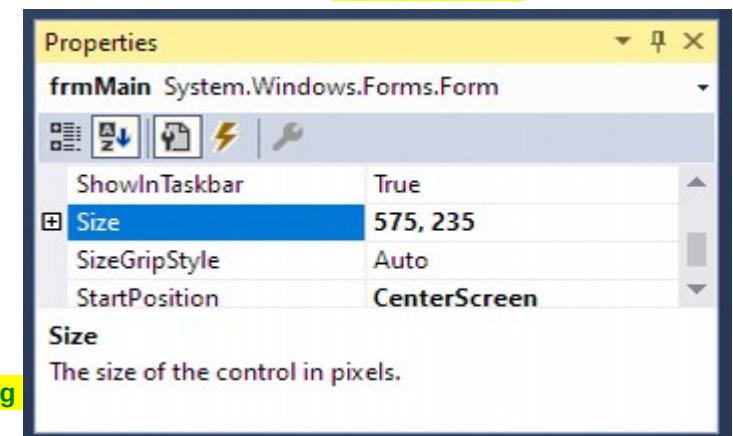


**Figure 11-39f**

**step 4.4** - next, you will change the size of your **form**, named **frmMain**

-> in the **Designer window**, click the **frmMain** form and change its property **Size** from = **575, 334**, to -> = **575, 235**

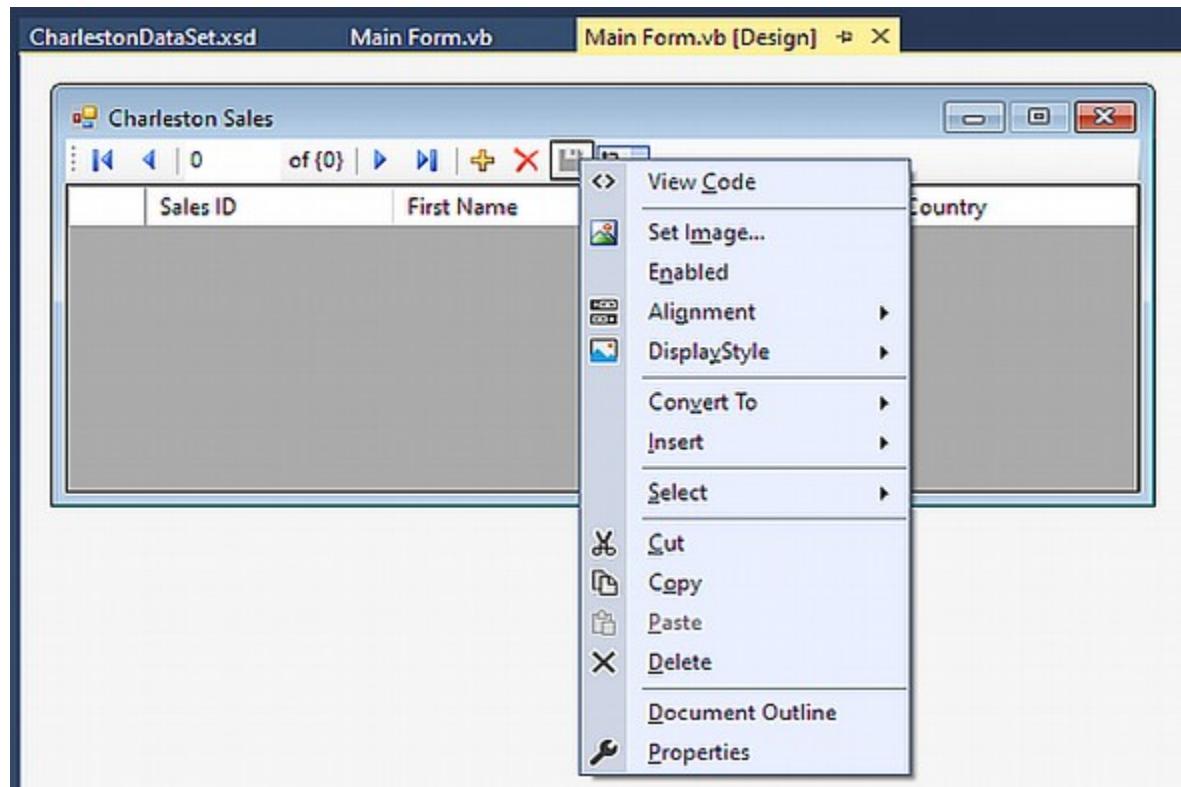
**Figure 11-39g**



**Figure 11-39g**

**step 4.5**

- because in this application the **DataGridView** control will only display the records and the user will not be allowed to modify them -
  - as you set in: **step 4.3.2** & **Figure 11-39d**, you will modify the **SalespersonBindingNavigator** control's buttons, so that during run time, the user can not modify the table
- > in the **Designer window / SalespersonBindingNavigator** control:
  - > Rclick the button **Save Data** and click **Delete** **Figure 11-39h**
  - > Rclick the button **Add new** and click **Delete**
  - > Rclick the button **Delete** and click **Delete**
- > compare the original and modified **SalespersonBindingNavigator** control: **Figure 11-39i**
- > save the solution

**Figure 11-39h****Figure 11-39i**

step 4.6

-> start and test the application

Figure 11-39j

|   | Sales ID | First Name | Last Name | Country       |
|---|----------|------------|-----------|---------------|
| ▶ | 1001     | Perry      | Jones     | United States |
|   | 1002     | Roger      | Kramden   | Canada        |
|   | 1003     | Pamela     | Jefferson | United States |
|   | 1004     | Gerald     | Sistenski | Mexico        |
|   | 1005     | Lucy       | Norburn   | Canada        |

Figure 11-39j information from both tables displayed in the **SalespersonDataGridView** control

-> you can stop the application and then close the solution

## CH11\_APPLY THE CONCEPTS LESSON

**CH11\_A1 - Create a Data Form** (instead of using **DataGridView** control) for displaying **dataset**, created from existing \*.mdf database file & & code: add the **Try...Catch** 'safety net' statement & **KeyPress** procedures example: **03.Course Info Solution-Data Form**

- although a **DataGridView** control is a good choice for displaying data, it is not always the best choice for inputting data -
  - especially if the number of fields/columns will require the user to scroll horizontally during data entry ->
  - > in those cases, you should create a **Data form**

-> a **Data form**: = bound control like **DataGridView** control, but allowing you to create a separate appropriate control for each field in the table

- typically provides text boxes for entering data, and labels for just displaying data

- to create a **Data form** for existing **MyCourses.mdf** database file example: **03.Course Info Solution-Data Form**

-> open the: ...VB2017\Chap11\_Exercise\03.Course Info Solution-Data Form\Course Info Solution.sln  
-> open the **Designer** window, and display the **Data Sources** window (VS menu bar **View / Other Windows / Data Sources** Shift+Alt+D)

**step 1.1** -> in **Data Sources** window, click **Add New Data Source...** to start the **Data Source Configuration Wizard**, described in:

**CH11\_F4 -** & **step 4.x**

**step 1.2** -> **Data Source Configuration Wizard** / screen **Choose a Data Source Type** -> = **Database**

**step 1.3** -> **Data Source Configuration Wizard** / screen **Choose a Database Model** -> = **Dataset**

**step 1.4** -> **Data Source Configuration Wizard** / screen **Choose Your Data Connection** -> click the button **New Connection...**

**step 1.4.1** - one of the dialog boxes opens: either **Choose Data Source** dialog box, or **Add Connection** dialog box

- if **Choose Data Source** dialog box opens:

-> choose **Microsoft SQL Server Database File**, and then click the button **Continue**

**step 1.4.2** - a new dialog box **Add Connection** opens:

-> verify that the box **Data source**: contains: **Microsoft SQL Server Database File (SqlClient)**

-> if it does **not**, click the button **Change...** and choose the **Microsoft SQL Server Database File (SqlClient)**

**step 1.4.3** -> click the button **Browse...** to add the **MyCourses.mdf** database file from your location

-> click the button **Test Connection**

- the message "**Test connection succeeded.**" appears in a message box

-> you can close the message box

-> in a dialog box **Add Connection**, click the button **OK**

**step 1.5** - in **Data Source Configuration Wizard** / screen **Choose Your Data Connection** -> will now appear **MyCourses.mdf** database file

-> click the button **Next >**

- the message box named "**Microsoft Visual Studio**" opens and asks, whether you want to include the database file in the current project

- by including the file in the current project, you can more easily copy the application and its database to another computer

-> click the button **Yes**, to add the **MyCourses.mdf** database file to the application's project folder

**step 1.6** - the **Data Source Configuration Wizard** / screen **Save the Connection String to the Application Configuration File** opens

- it displays the name of the connection string: **MyCoursesConnectionString**

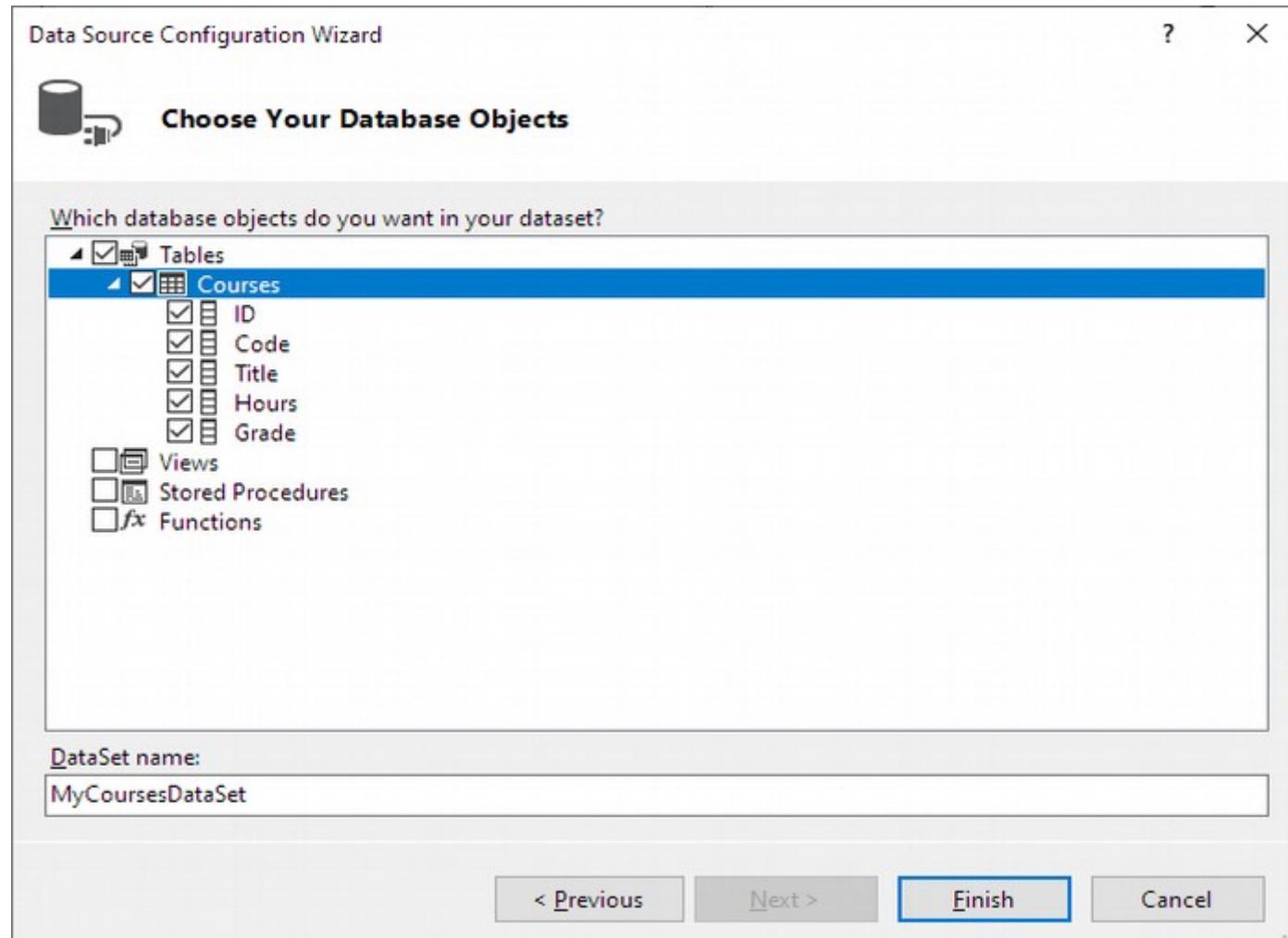
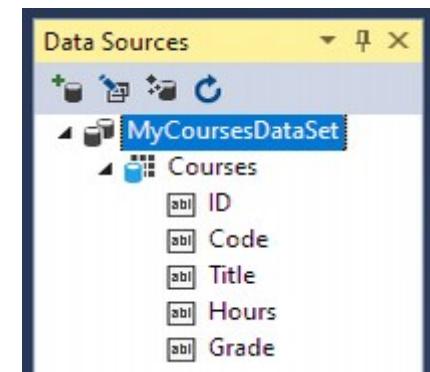
- similar to **CH11\_F4 - step 4.5** & same like: **CH11\_F9.1 - step 1/4:**

**step 1.6**

-> verify that check box **Yes...** is selected and click the button **Next >**

**step 1.7**

- the Data Source Configuration Wizard / screen **Choose Your Database Objects** appears
- as you learned in **CH11\_F4 - step 4.6** you use this screen to select the database objects you want to include in your **dataset**
- the default name for this dataset is **MyCoursesDataSet**, no need to change the name
- > expand the node **Tables** and click the check box next to the node **Courses** to add the entire table with all of the fields
- > click the button **Finish**

**Figure 11-40a****Figure 11-40b****step 1.8**

- the computer adds to the **Data Sources** window a new dataset named **MyCoursesDataSet**
- dataset contains:
  - **1** table object named **Courses**, and
  - **5** field objects named: **ID**, **Code**, **Title**, **Hours**, **Grade**

**Figure 11-40b****Figure 11-40b**

**step 1.9.0**

- for viewing data, you will change the default **DataGridView** control to other type of control - more info in: **CH11\_F5 - info for step 5/8: about Binding**
- the icon that appears before the object's name in the **Data Sources** window indicates the **type of control** the computer will create

**step 1.9.1**

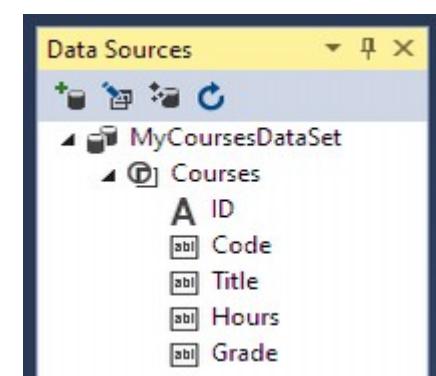
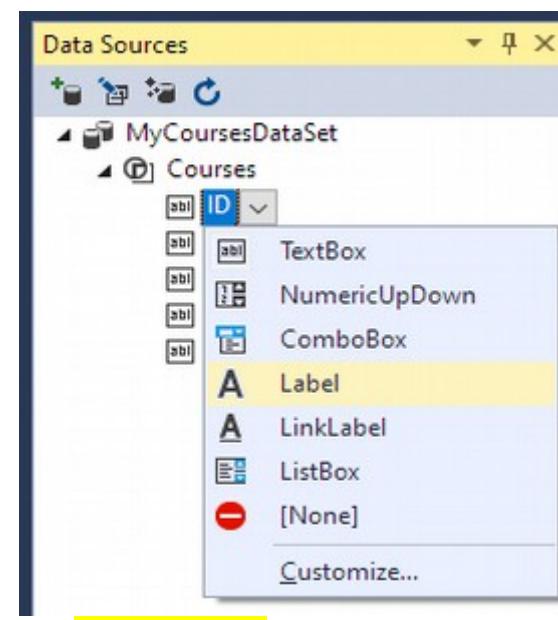
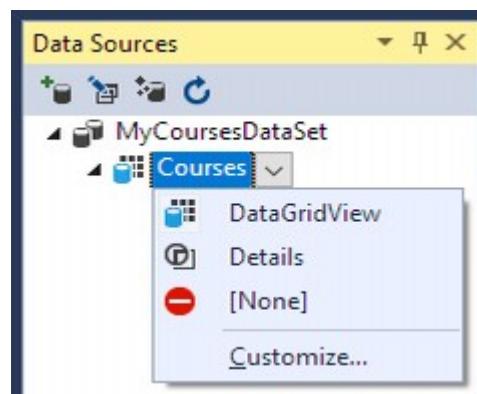
- > in the window **Data Sources**, click the **Courses** node's **list arrow** and choose **Details** **Figure 11-40c**

- this will display data in separate **text boxes** rather than in a **DataGridView** control  
- choosing **Details** in the list tells the computer to create a separate appropriate control for each field in the table (text boxes, labels, and so on...)

**step 1.9.2**

- recall that the **ID** field is an auto-numbered field, which means that the database will take care of completing the field for each new record -  
- therefore you will display the **ID** field in a **label control**

- > in the window **Data Sources**, click the **ID** field's **list arrow** and choose **Label** **Figure 11-40d**

**step 1.9.3**

- > notice the icons before the object's names in the **Data Sources** window **Figure 11-40e**
- **Courses** - **Details** = computer will create separate appropriate control for each field in the table
- **ID** - **Label** = the field will be displayed in **label** control
- **Code, Title, Hours, Grade** - **TextBox** = the fields will be displayed in separate **text boxes**

**step 2.1**

-> drag the **Courses** table to the form

- the computer adds to the form: 1). the **BindingNavigator** control

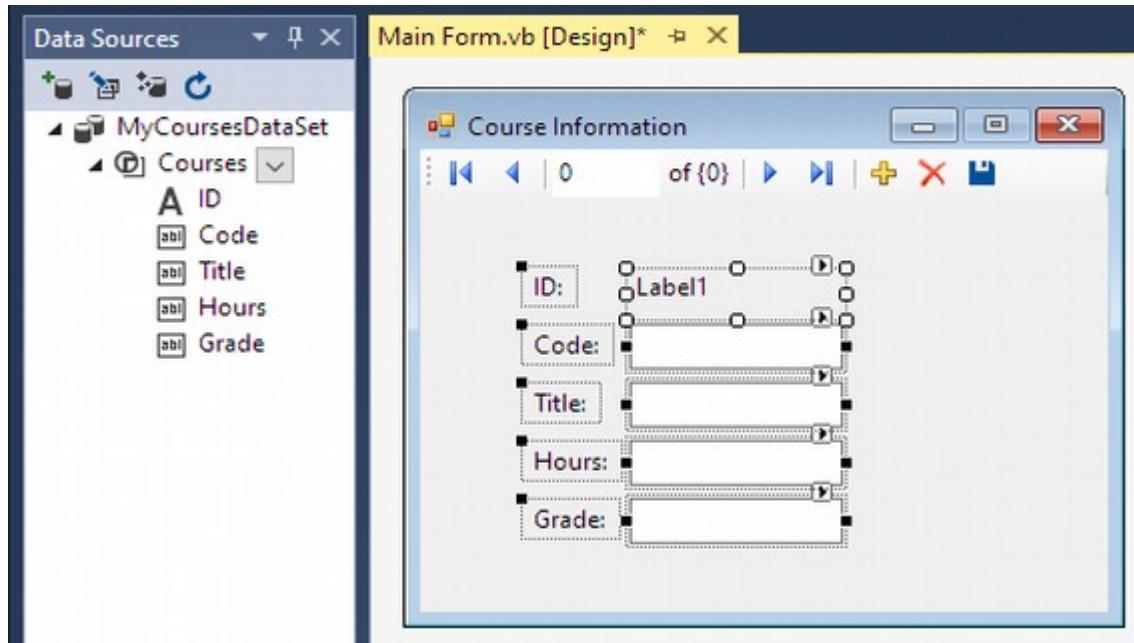
**Figure 11-41a**

2). **10 controls**: - 6 labels, and 4 text boxes

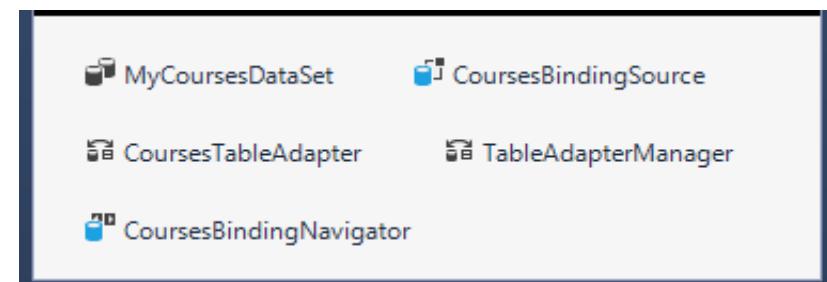
**Figure 11-41a**

3). 5 objects to the component tray: - **MyCoursesDataSet**, **CoursesBindingSource**, **CoursesTableAdapter**, **TableAdapterManager**, **CoursesBindingNavigator**

**Figure 11-41b**

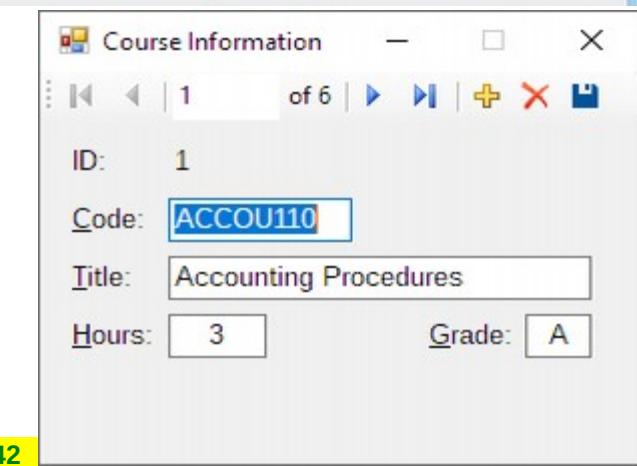
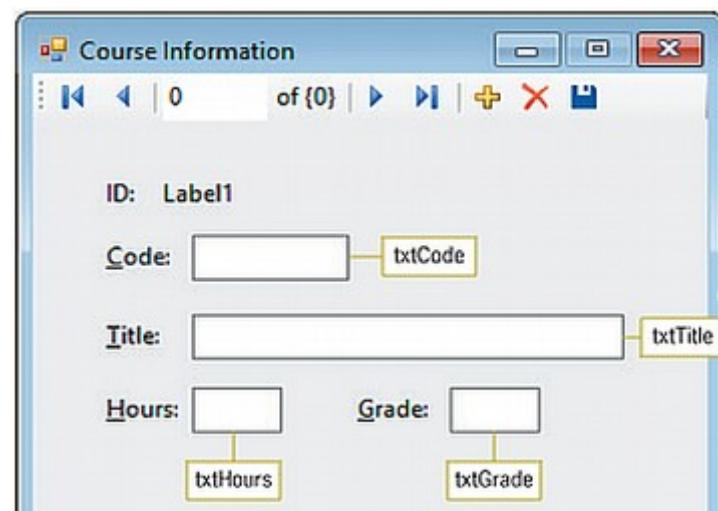


**Figure 11-41a**



**Figure 11-41b**

**Figure 11-41c**



**Figure 11-42**

**step 2.2**

- you will modify GUI, change the text boxes (Name)s, and add access keys

-> change the appearance, (Names), and access keys as shown in: **Figure 11-41c**

-> set the **txtCode** property: **MaxLength = 8**

-> set the **txtCode** property: **CharacterCasing = Upper**

-> set the **txtHours** property: **MaxLength = 1**

-> set the **txtGrade** property: **MaxLength = 1**

-> set the **txtGrade** property: **CharacterCasing = Upper**

**step 2.3**

- the application will allow the user to **add**, **edit**, and **delete** records -

- so you will need to set the database file's **Copy to Output Directory** property

-> set the **MyCourses.mdf** property -> **Copy to Output Directory = Copy if newer**

**step 2.4**

-> save the solution and start and test the application

- the first record appears in the **Data Form** **Figure 11-42**

-> play with the **Move next**, **Move last**, **Move previous**, and **Move first** buttons on the **BindingNavigator** control to view the remaining records

-> when done exploring, close the application

**step 3.0**

- in the next set of steps, you will add the **Try...Catch** 'safety net' statement to the **BindingNavigator's Save Data** button's Click event procedure
- you will also code the **txtHours\_KeyPress** procedure to allow the text box to accept only **numbers** and the **Backspace** key
- > open the **Code Editor** window and look the automatically generated code when binding/dragging an object to the form Figure 11-43a

**Figure 11-43a** automatically generated code when binding/dragging an object to the form: example with **03.Course Info Solution-Data Form**

```
9  Public Class frmMain
10     Private Sub CoursesBindingNavigatorSaveItem_Click(sender As Object, e As EventArgs) Handles CoursesBindingNavigatorSaveItem.Click
11         Me.Validate()
12         Me.CoursesBindingSource.EndEdit()
13         Me.TableAdapterManager.UpdateAll(Me.MyCoursesDataSet)
14
15     End Sub
16
17     Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles MyBase.Load
18         'TODO: This line of code loads data into the 'MyCoursesDataSet.Courses' table. You can move, or remove it, as needed.
19         Me.CoursesTableAdapter.Fill(Me.MyCoursesDataSet.Courses)
20
21     End Sub
22 End Class
```

**step 3.1**

- you will include a **Try...Catch** 'safety net' statement into **BindingNavigator's button Save Data** to avoid any potential runtime errors:
- > locate the procedure **CoursesBindingNavigatorSaveItem\_Click** and modify as shown below:

```
10    Private Sub CoursesBindingNavigatorSaveItem_Click(sender As Object, e As EventArgs) Handles CoursesBindingNavigatorSaveItem.Click
11        Try
12            Me.Validate()
13            Me.CoursesBindingSource.EndEdit()
14            Me.TableAdapterManager.UpdateAll(Me.MyCoursesDataSet)
15            MessageBox.Show("Changes saved.", "Course Information", MessageBoxButtons.OK, MessageBoxIcon.Information)
16        Catch ex As Exception
17            MessageBox.Show(ex.Message, "Course Information", MessageBoxButtons.OK, MessageBoxIcon.Information)
18        End Try
19    End Sub
20
21    Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles MyBase.Load
```

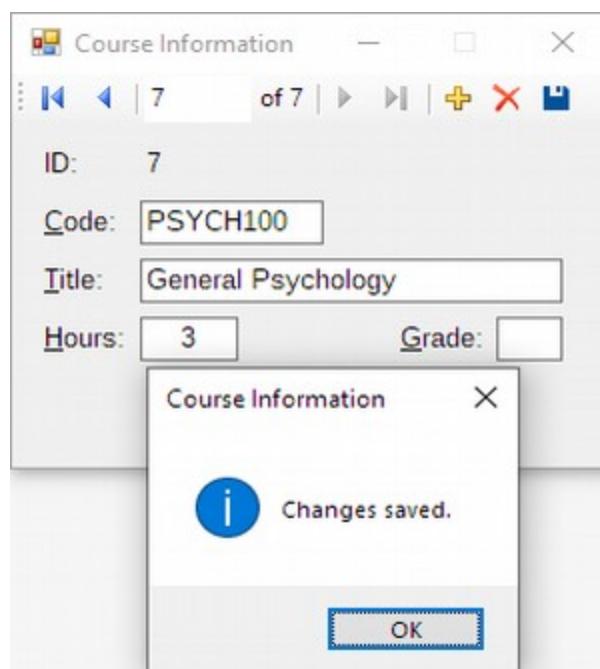
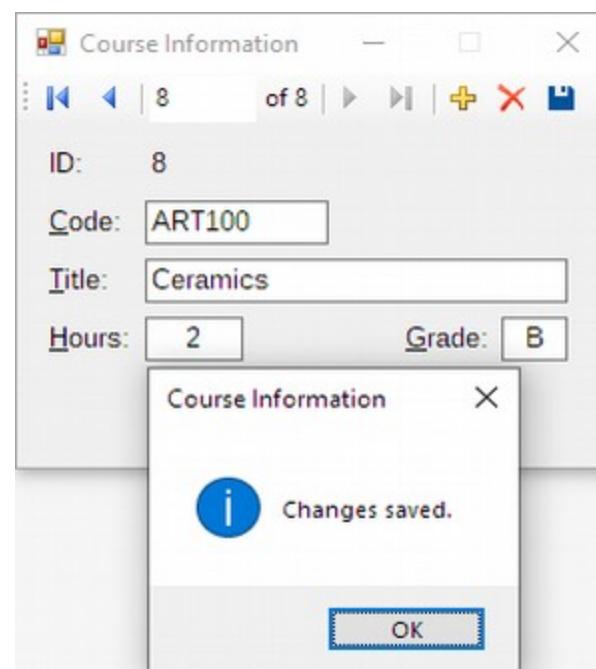
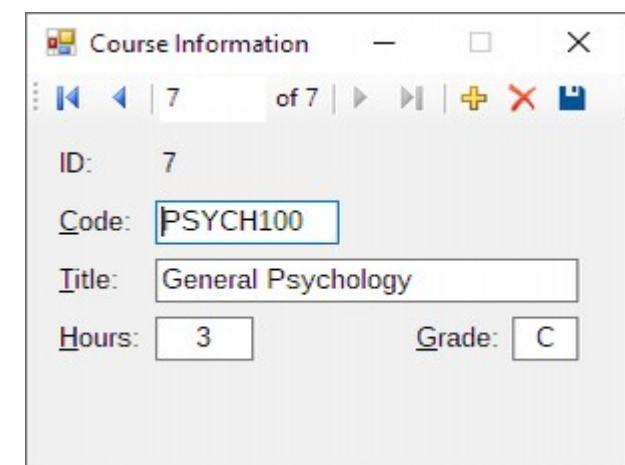
**step 3.2**

- you will allow the text box (**txtHours**) to accept only numbers and the Backspace key:
- > open/create the code template for the **txtHours\_KeyPress** procedure, and enter the code shown below:

```
26    Private Sub txtHours_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtHours.KeyPress
27        ' Accept only numbers and the Backspace key:
28        If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
29            e.Handled = True
30        End If
31    End Sub
32 End Class
```

**step 4.0**

- > save the solution and then start and test the application
- > on the **BindingNavigator** control click the button **Add new** (the plus sign) and type: **psych100**, **General Psychology**, **3**
  - the student has not completed the course yet, so you will leave the **Grade** field empty
  - > on the **BindingNavigator** control click the **Save Data** button and close the message box that confirms that the changes were saved
- > now, click the **Add new** button and type: **art100**, **Ceramics**, **2**, **b**, and click the **Save Data** button
- > stop the application and then start it again
- > use the **BindingNavigator** control to locate the **PSYCH100** record, which is record **7** in the dataset, and type **C** in the record's **Grade** field
- > locate the **ART100** record, which is the last record in the dataset, **Delete** the record, and **Save Data**
- > stop the application and start it again
- > verify that the **PSYCH100** grade was saved, and that the **ART100** record is no longer in the dataset
- > stop the application and close the solution

**Figure 11-44a****Figure 11-44a****Figure 11-44b****Figure 11-44c**

## CH11\_A2 - Bind Field objects to existing controls - basic info

- as indicated earlier, you can **bind** an object in a dataset to an **existing control** on a form in 2 ways:
  - more info about binding objects in: [CH11\\_F5 - info for step 5/8: about Binding/connecting an object](#)

a). by dragging - the easiest way

-> from window **Data Sources** drag the object to the control in **Designer** window

b). by setting control's **Properties** - the appropriate property/s depends on the control type you are binding

-> in the **Designer** window click the control and then in **Properties** window set 1 or more **Properties**

-> to bind a **ListBox** control: 1). use property **DataSource** and select a table

2). use property **DisplayMember** and select a field

-> to bind a **DataGridView** control to a **table**, use the property: **DataSource** and select the table

-> to bind a **Label** or **TextBox** control to a table's field, use the property: **(DataBindings) / Text** and select the field

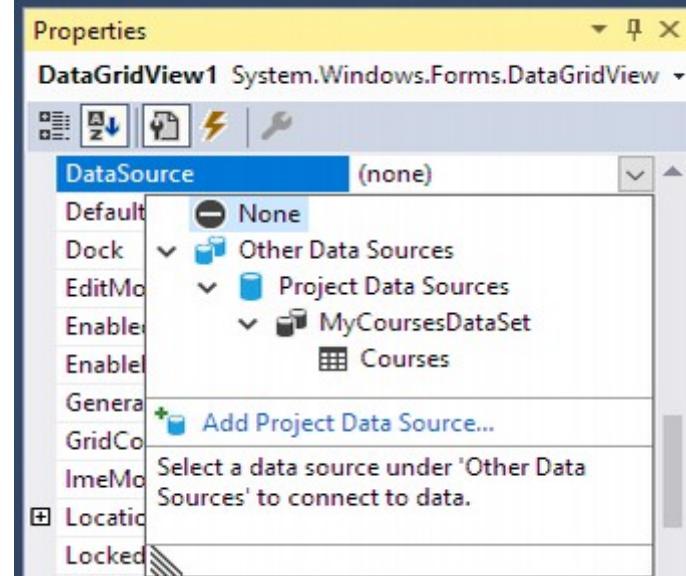
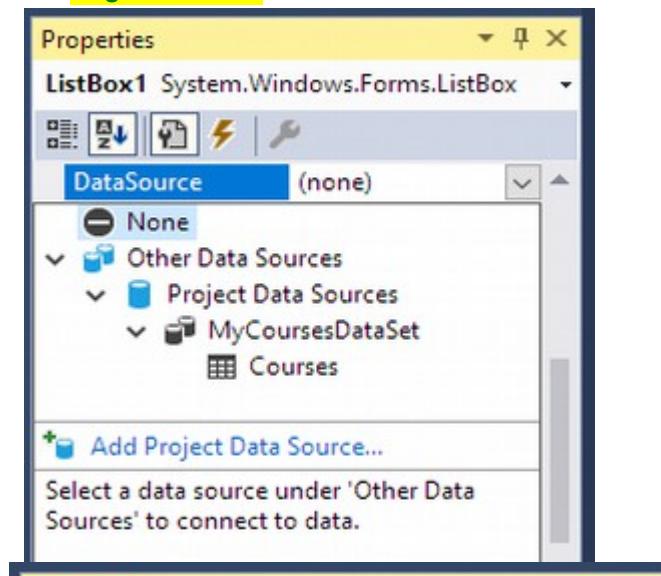
**Figure 11-45a**

**Figure 11-45b**

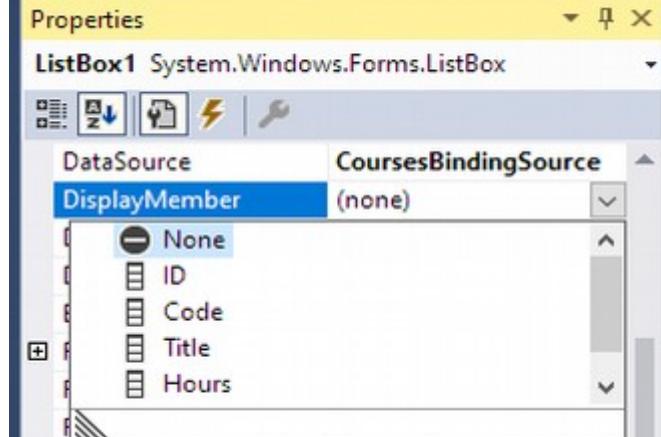
**Figure 11-46**

**Figure 11-47**

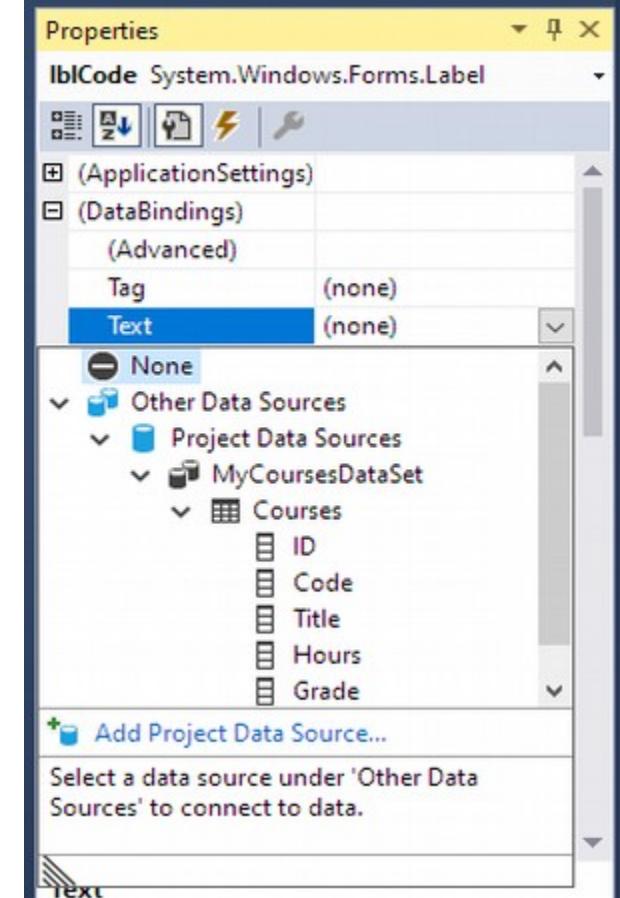
**Figure 11-45a** ListBox



**Figure 11-46** DataGridView



**Figure 11-45b** ListBox



**Figure 11-47** Label, TextBox

## CH11\_A2.1 - Bind Field objects to existing controls by dragging them to the form & add BindingNavigator control example: 04.Course Info Solution-Labels

- in the next set of steps, you will use the **MyCourses.mdf** database file in a different version of the **Course Information** application
- in this version, the information will be displayed in **Label** controls (therefore editing will be disabled), rather than in a **DataGridView** control
- to bind objects to existing controls by dragging them example: **04.Course Info Solution-Labels**

-> open the: ...VB2017\Chap11\Exercise\04.Course Info Solution-Labels\Course Info Solution.sln

-> open the **Designer** window, and display the **Solution Explorer** window **Figure 11-48**

- the application is already connected to the **MyCourses.mdf** database file, and

- the dataset **MyCoursesDataSet.xsd** has already been created

-> display the **Data Sources** window (VS menu bar **View / Other Windows / Data Sources** Shift+Alt+D)

**step 1** -> in **Data Sources** window, expand the node **MyCoursesDataSet / Courses**

- dataset contains:
  - 1 table object named **Courses**, and
  - 5 field objects named **ID, Code, Title, Hours, Grade**

**step 2** -> drag the field object **Code** to the **lblCode** control **Figure 11-49**

-> drag the field object **Grade** to the **lblGrade** control **Figure 11-49**

-> notice that you do not need to change the control type in the **Data Sources** window to match the existing control's type

- the computer binds the field objects to the controls **Figure 11-50**

- the computer adds to the component tray: **MyCoursesDataSet, CoursesBindingSource, CoursesTableAdapter, TableAdapterManager** objects

-> notice that when you drag an object from the **Data Sources** window to an existing control, the computer does **not add**:

- a. a **BindingNavigator** object to the component tray
- b. a **BindingNavigator** control to the form

- you will add the missing control and object in **step 3**

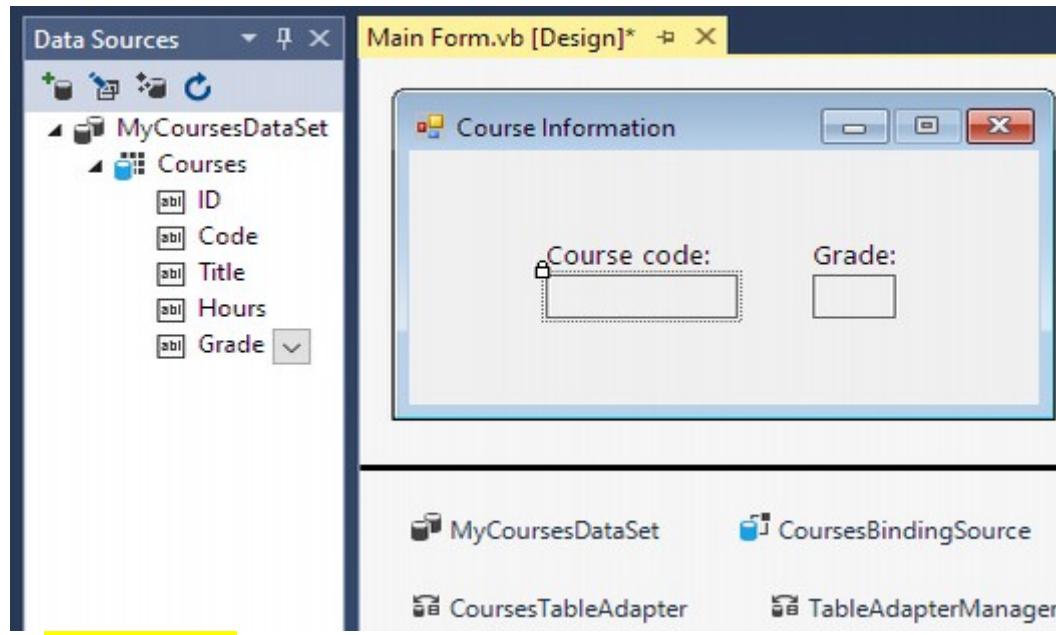


Figure 11-49

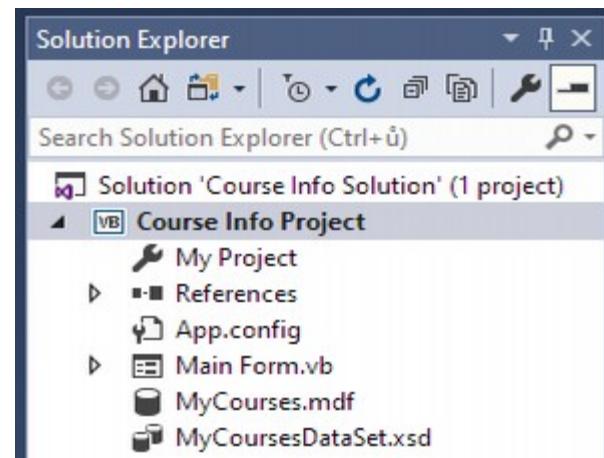
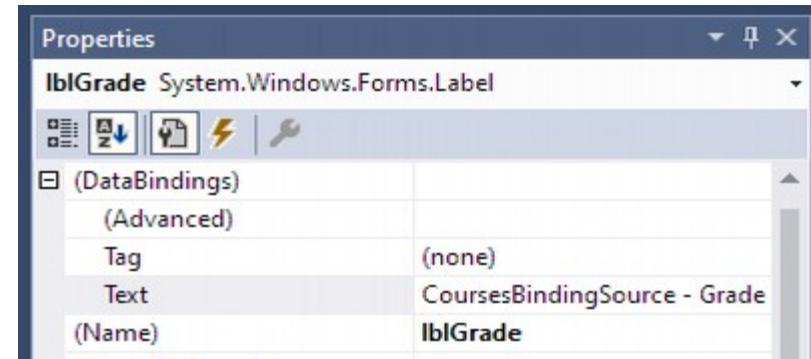


Figure 11-48



Figure 11-50 bound controls for **lblCode** and **lblGrade**

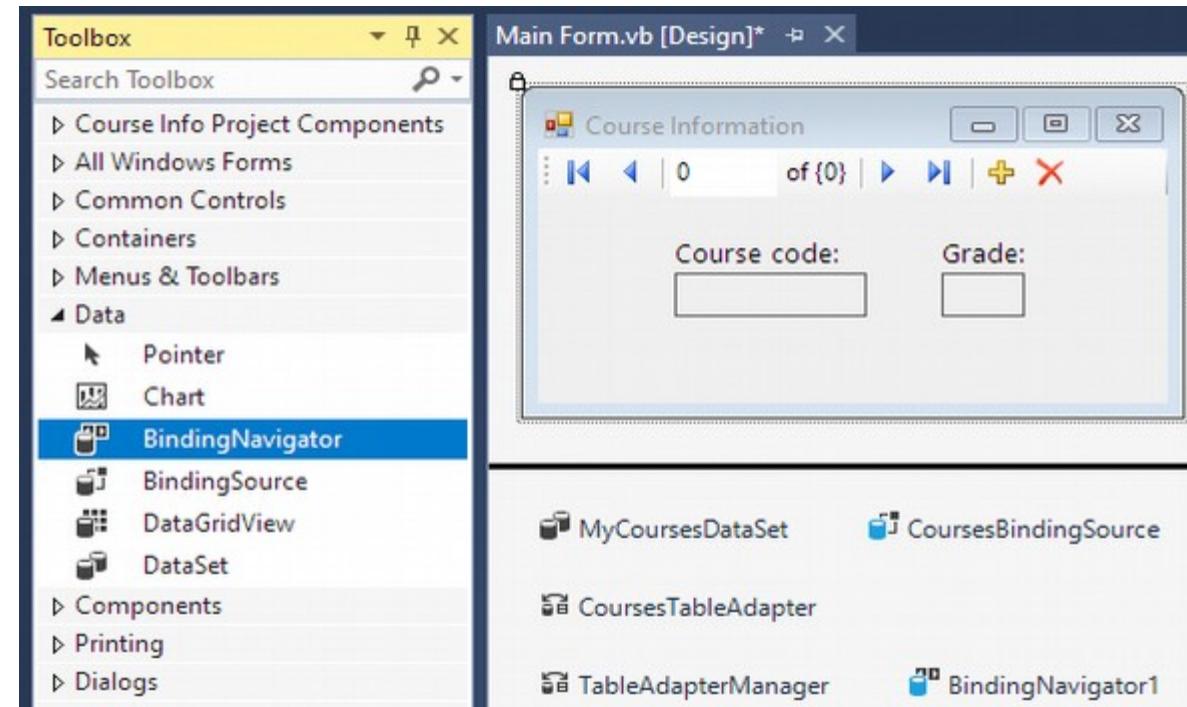
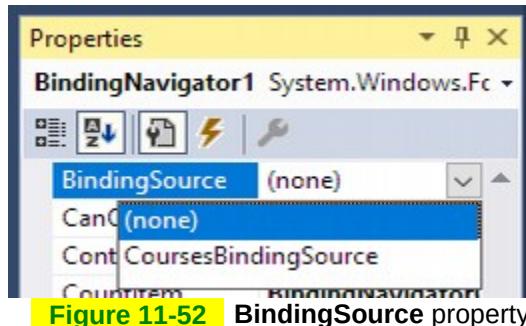


**step 3.1** - to add a **BindingNavigator** control and object to the application, you can use: **Toolbox / Data / BindingNavigator** tool

- > drag the **BindingNavigator** tool to the top of the form
- the computer adds:  
1). the **BindingNavigator1** object to the component tray  
2). the **BindingNavigator1** control to the form  
- without button **Save Data**

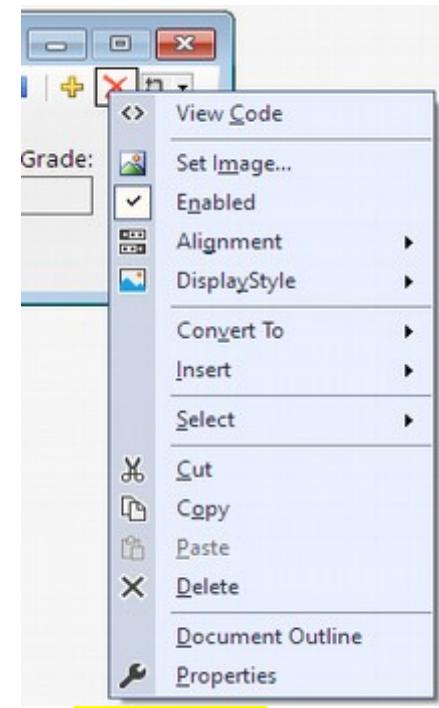
**step 3.2** - you have to set the source for **BindingNavigator**

- > set the property:  
**BindingSource** -> = **CoursesBindingSource**



**step 3.3** - because the application will only display the data, you will disallow the user to **add new** and **delete** the records

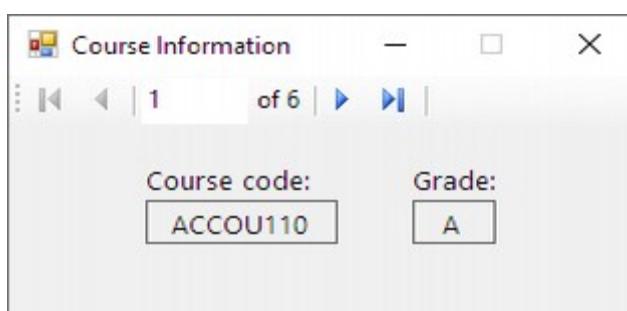
- > on **BindingNavigator1** control Rclick the button **Delete** and click **Delete** **Figure 11-53**
- > on **BindingNavigator1** control Rclick the button **Add new** and click **Delete** **Figure 11-53**



**step 4** -> save the solution and then start and test the application **Figure 11-54**

- > use the Move next, Move last, Move previous, and Move first buttons on the **BindingNavigator** control to view the remaining records

- > when done testing, close the application and then close the solution



**CH11\_A3 - perform Calculations on the Fields in a Dataset using loop:** `For Each row As MyCoursesDataSet.CoursesRow In MyCoursesDataSet.Courses.Rows`  
info & example: **05.Course Info Solution-Total Hours**

- in this section, you will use the **MyCourses.mdf** file in an application that totals the number of credit hours that the student has completed
- you can make this calculation by using a **loop** to **accumulate** the numbers stored in each record's **Hours** field

- to **sum** numeric values on selected cells example: **05.Course Info Solution-Total Hours**

- > open the: ...VB2017\Chap11\_Exercise\05.Course Info Solution-Total Hours\Course Info Solution.sln
- > open the **Designer** window, and display the **Solution Explorer** window
  - the application is already connected to the the **MyCourses.mdf** database file, and
  - the dataset **MyCoursesDataSet.xsd** has already been created

- step 1** -> start the application to view the dataset information that appears in the **DataGridView** control - it contains **14** records and **5** fields **Figure 11-55**
- > notice: **Grade** cell **W** for record no.**2**, and empty **Grade** cell for record no.**13** - both fields should not be included in the total number of credit hours

| ID | Code     | Title                       | Hours | Grade |
|----|----------|-----------------------------|-------|-------|
| 1  | ACCOU110 | Accounting Procedures       | 3     | A     |
| 2  | ENG101   | English Composition         | 3     | W     |
| 3  | CIS110   | Introduction to Programming | 3     | A     |
| 4  | BIO111   | Environmental Biology       | 3     | C     |
| 5  | ENG101   | English Composition         | 3     | B     |
| 6  | CIS112   | The Internet                | 2     | A     |
| 7  | EARTH110 | Geology                     | 4     | C     |
| 8  | EARTH112 | Astronomy                   | 3     | B     |
| 9  | ENG113   | Literature                  | 3     | B     |
| 10 | MATH110  | Business Mathematics        | 3     | A     |
| 11 | MATH115  | Technical Mathematics       | 5     | B     |
| 12 | HISTO110 | Western Civilization        | 4     | A     |
| 13 | SOCIO100 | Introduction to Sociology   | 3     |       |
| 14 | SPAN110  | Elementary Spanish I        | 4     | A     |

**Figure 11-55**

- > close the application

**step 2** - you will code the **btnCalc\_Click** procedure-> open the **Code Editor** window and locate the **btnCalc\_Click** procedure on line **10**

- pseudocode:
  1. declare **intTotal** variable for accumulating the number of credit hours completed
  2. repeat for each row/record in the dataset

if the current row's **Grade** field is **not empty** and it does **not** contain the letter **W**,  
     add the hours from the current row's **Hours** field to the **intTotal** variable  
     end if  
     end repeat

3. display the contents of the **intTotal** variable in the (**lblTotal**) control

<- recall that an empty Grade field contains: **NULL**

**step 2.1** - the procedure will use an Integer variable named **intTotal** to accumulate the values in the **Hours** field, excluding the values stored in the records no.**2 & 13**-> within the procedure, type the declaration statement on line **12****step 2.2** - the easiest way to access each record/row in a dataset is by using a **For Each...Next** statement-> under the declaration statement, enter the loop statement **Figure 11-56**

```
For Each NameOfMyLoop As dataset.class In dataset.table.RowsCollection
```

```
...
```

```
Next NameOfMyLoop
```

**step 2.3** - the loop will use a selection structure to determine whether the **Grade** field's value is not **NULL** and is not the letter **W**- you can make the **NULL** determination by using the method **IsGradeNull**, returning a Boolean value- if both of the conditions are met, the s.s.'s **True** path should add the number of hours stored in the record's **Hours** field to the **intTotal** accumulator variable-> within the loop statement, enter the selection structure statement on lines **15 - 17****step 2.4** - the last step in the procedure's pseudocode is to display the contents of the **intTotal** variable in the (**lblTotal**) control-> in the end of the **btnCalc\_Click** procedure, enter the assignment statement on line **19**

```

9  Public Class frmMain
10     Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
11         ' Display the total number of credit hours completed.
12         Dim intTotal As Integer
13
14         For Each row As MyCoursesDataSet.CoursesRow In MyCoursesDataSet.Courses.Rows
15             If row.IsGradeNull = False AndAlso row.Grade <> "W" Then
16                 intTotal += row.Hours
17             End If
18         Next row
19         lblTotal.Text = intTotal.ToString
20     End Sub

```

**step 2.1**

**step 2.2**

**step 2.3**

**step 2.3**

**step 2.3**

**step 2.2**

**step 2.4**

- step 3**
- > save the solution, and start and test the application
  - > when done, close the solution

Figure 11-57

The screenshot shows a Windows application window titled "Course Information". At the top, there are navigation buttons: back, forward, first, last, and search. The text "of 14" is displayed between the first and last buttons. Below the grid, there are buttons for "Calculate" and "Exit".

| ID | Code     | Title                       | Hours | Grade |
|----|----------|-----------------------------|-------|-------|
| 1  | ACCOU110 | Accounting Procedures       | 3     | A     |
| 2  | ENG101   | English Composition         | 3     | W     |
| 3  | CIS110   | Introduction to Programming | 3     | A     |
| 4  | BIO111   | Environmental Biology       | 3     | C     |
| 5  | ENG101   | English Composition         | 3     | B     |
| 6  | CIS112   | The Internet                | 2     | A     |
| 7  | EARTH110 | Geology                     | 4     | C     |
| 8  | EARTH112 | Astronomy                   | 3     | B     |
| 9  | ENG113   | Literature                  | 3     | B     |

Total hours completed:

Figure 11-57

- entire code:

```

1  ' Name:      Course Info Project
2  ' Purpose:   Add, edit, delete, display records and display total number of credit hours completed.
3  ' Name:      <your name> on <current date>
4
5  Option Explicit On
6  Option Strict On
7  Option Infer Off
8
9  Public Class frmMain
10    Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
11      ' Display the total number of credit hours completed.
12      Dim intTotal As Integer
13
14      For Each row As MyCoursesDataSet.CoursesRow In MyCoursesDataSet.Courses.Rows
15          If row.IsGradeNull = False AndAlso row.Grade <> "W" Then
16              intTotal += row.Hours
17          End If
18      Next row
19      lblTotal.Text = intTotal.ToString
20    End Sub
21
22    Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
23        Me.Close()
24    End Sub

```

```

25
26      Private Sub CoursesBindingNavigatorSaveItem_Click(sender As Object, e As EventArgs) Handles CoursesBindingNavigatorSaveItem.Click
27          Try
28              Me.Validate()
29              Me.CoursesBindingSource.EndEdit()
30              Me.TableAdapterManager.UpdateAll(Me.MyCoursesDataSet)
31              MessageBox.Show("Changes saved.", "Course Information", MessageBoxButtons.OK, MessageBoxIcon.Information)
32          Catch ex As Exception
33              MessageBox.Show(ex.Message, "Course Information", MessageBoxButtons.OK, MessageBoxIcon.Information)
34          End Try
35      End Sub
36
37      Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles MyBase.Load
38          'TODO: This line of code loads data into the 'MyCoursesDataSet.Courses' table. You can move, or remove it, as needed.
39          Me.CoursesTableAdapter.Fill(Me.MyCoursesDataSet.Courses)
40
41      End Sub
42  End Class

```

## CH11\_Summary

01. most businesses store information in computer databases
- 02a. the information in a relational database is stored in 1 or more tables
- 02b. each table is composed of fields (columns) and records (rows)
- 03a. most tables have a primary key that uniquely identifies each record in the table
- 03b. some tables also have a foreign key that us used to relate one table to another table
- 04a. you can use the Data Source Configuration Wizard to connect an application to a database file
- 04b. to start the wizard, open the Data Sources window by clicking View, pointing to Other Windows, and then clicking Data Sources
- 04c. then, click Add New Data Source in the Data Sources window
05. for the user to view the information contained in a dataset, you must bind 1 or more of the objects in the dataset to 1 or more controls
- 06a. a DataGridView control displays data in a row (record) and column (field) format, similar to a spreadsheet
- 06b. to have the columns fill the control's display area, set the control's AutoSizeColumnsMode property to Fill
- 06c. to anchor the control to the borders of its container (which is typically the form), click the Dock in Parent Container option on the control's task list
- 07a. you can use the Edit Columns option on the DataGridView control's task list to add columns, remove columns, and reorder columns
- 07b. you also can use it to change the properties of the columns
08. a database file's Copy to Output Directory property determines the way Visual Basic saves the changes made to a local database file
09. you can use the Try...Catch statement to handle exceptions (errors) that occur during run time
- 10a. a database query allows you to retrieve specific information from a database. such as the fields and records you want to display
- 10b. you can create the query using the Query Builder dialog box and Structured Query Language (SQL)
11. when inputting data, it is often easier to use a data form, which typically provides text boxes for entering data, instead of a DataGridView control
12. you can access each record (row) in a dataset using a loop along with the dataset table's Rows collection
13. you can use the **IsfieldNull** method to determine whether the *field* contains the NULL value = empty

## CH11\_Key Terms & terminology marked: ->

- **AutoSizeColumnsMode property** - determines the way the column widths are sized in a **DataGridView** control; default = **None**
  - **None, ColumnHeader, AllCellsExceptHeader, AllCells, DisplayedCellsExceptHeader, DisplayedCells, Fill**
- **Binding** - the process of connecting an object in a dataset to a control on a form
- **BindingNavigator control** - allows the user to interact with a dataset during run time
  - can be used to add, delete, and save records and also to move the record pointer from one record to another in a dataset
- **BindingSource object** - connects a **DataSet** object to the bound control on a form
- **Bound controls** - the controls connected to an object in a **DataSet**
- **Cell** - the intersection of a row and a column in a **DataGridView** control
  - the intersection of a record and a field in a **DataGridView** control
- **Child table** - a table that contains a **Foreign key** matching the **Primary key** from a **Parent table**
- **Computer database** - an electronic file that contains an organized collection of related information
  - most databases are created and manipulated using **RDBMSs** -> relational database management systems
- **Copy to Output Directory property** - a property of a database file; default = **Do not copy**
  - determines both when and if the file is copied from the project folder to the project output **bin\Debug** folder
    - **Do not copy, Copy Always, Copy if newer**
- **Data form** - bound control like **DataGridView** control, but allowing you to create a separate appropriate control for each field in the table
  - typically provides text boxes for entering data, and labels for just displaying data
- **Database query/query task** - a statement/task that allows you to retrieve/filter out specific information from a database, using a special language called **SQL**
  - you can use **query task** to specify the fields and records you want to display
  - the result is called **query object**
- **DataGridView control** - displays the table data in a row and column format, similar to a spreadsheet
- **Dataset** - a copy of specified data - database fields and records, that can be accessed by an application, and stored in computer's RAM
- **DataSet object** - stores the information you want to access from a database
- **Exception** - an error that occurs while an application is running/during a run time
  - when an error occurs in a procedure's code during run time, programmers say: "**the procedure threw an exception**"
- **Field** - a single item of information about a person, place, or thing
  - group/combination of related fields creates a **record**
- **Foreign key** - field that typically refers to a **primary key** in another table, creating a relationship between the two tables
  - can also refer to any other key that uniquely identifies each row in another table
- **Local database file \*.mdf** - a database file contained in a project
  - in Visual Studio can be found in a **Solution Explorer** window
  - physically located in:
    - 1). in a source **Project folder**, and
    - 2). in the output folder **bin\Debug**
- **mdf extension** - filename extension is commonly used when naming **SQL Server databases**
  - stands for: **master database file**
- **Message property** - in a **Try...Catch** statement, a property of the **Catch** block's **ex** parameter
  - contains a description of the error that occurred in the **Try** block's code
  - e.g. **Catch ex As Exception**

```
MessageBox.Show(ex.Message, "Course Information", MessageBoxButtons.OK, MessageBoxIcon.Information)
```
- **one-to-many relationship** - a relationship between **parent table** and **child table**
- **Parent table** - a table that contains a **primary key** matching the **foreign key** from a **Child table**
- **Primary key** - a field that uniquely identifies each record in a table

- **Query Builder dialog box** - used to create a **query**
- **Query/database query task** - a statement/task that allows you to retrieve/filter out specific information from a database, using a special language called **SQL**
  - you can use **query task** to specify the fields and records you want to display
  - the result is called **query object**
- **Query object** - result of **query task**
  - enable you to retrieve records from one or more tables and then combine the data into rows and columns in a single dataset
  - can also perform calculations on data
  - there are **2** types:
    - 1). normal** - can be used to display data in user interface
    - 2. API** - used to generate web service endpoints and can not be displayed in user interface
- **RDBMSs** - **Relational DataBase Management Systems**
  - the databases are called **relational databases** because the information in them can be related in different ways
  - can contain multiple tables
  - the most popular: **Microsoft SQL Server, Oracle, IBM DB2**
- **Record** - a group of related **fields** that contain all of the necessary data about a specific person, place, or thing
- **Relational databases** - databases that store information in tables composed of columns - **fields** and rows - **records**
  - the information in these databases can be related in different ways
- **SQL/Structured Query Language** - a special language used to create the **database queries**
  - contains statements that allow you to retrieve and manipulate the data stored in databases
  - more about **SQL** in: **Chapter 12 Database Queries with SQL**
- **Table** - composition of columns and rows, similar to the format used in a spreadsheet; = group of related **records**
  - most tables have a **primary key** and a **foreign key**
  - in a case of multiple tables, those can be divided into a **parent table** and a **child table**
  - when defining the table, you will need to provide both:
    - 1). name** of each field
    - 2). data type** of each field, indicating the type of data the field will store
- **TableAdapter object** - connects a database to a **DataSet object**
- **TableAdapterManager object** - handles saving data to the tables in a dataset
- **Try...Catch statement** - used for exception handling in a procedure

e.g. Try

```

Me.Validate()
Me.CoursesBindingSource.EndEdit()
Me.TableAdapterManager.UpdateAll(Me.MyCoursesDataSet)
MessageBox.Show("Changes saved.", "Course Information", MessageBoxButtons.OK, MessageBoxIcon.Information)
Catch ex As Exception
    MessageBox.Show(ex.Message, "Course Information", MessageBoxButtons.OK, MessageBoxIcon.Information)
End Try

```

**CH11\_Exercises****Chap11\Exercise1****06.Cookies Solution\_EXERCISE 1\_introductory**

- create **DataGridView** control

- proportional width of all columns

**07.MusicBox Solution\_EXERCISE 2\_introductory**

- create **DataGridView** control

- non proportional table's column widths when **BoxesDataGridView**'s property: **AutoSizeColumnsMode = Fill**

- **DataGridView Tasks / Edit Columns...** / dialog box **Edit Columns / Selected Columns** property **FillWeight**

- property **FillWeight** = the weight that is used when sizing this column in the Fill auto size mode

- **ID**'s property **FillWeight** = 44.82118, **Shape**'s property **FillWeight** = 78.79391

- **Source**'s property **FillWeight** = 78.79391, **Song**'s property **FillWeight** = 132.2225

**08.Sales Solution\_EXERCISE 3\_introductory**

- create a **data form** with a Label and TextBoxes

**09.MusicBox Solution-ListBox\_EXERCISE 4\_introductory**

- bind existing **Label** controls by dragging selected **Table fields**

- bind existing **ListBox** control by setting its **Properties**: **DataSource = BoxesBindingSource**  
**DisplayMember = ID**

**10.Total Cookie Sales Solution\_EXERCISE 5\_intermediate**

- right-align the numbers in columns: **DataGriedView1 / Properties / DefaultCellStyle** / oval button  
& **CellStyle Builder** dialog box -> **Alignment = MiddleRight**

- **DataGridView1**'s property: **AutoSizeColumnsMode = Fill**

- non proportional table's column widths:

- **DataGridView Tasks / Edit Columns...** & dialog box **Edit Columns**

- property **FillWeight** = the weight that is used when sizing this column in the Fill auto size mode

- column **Week**: **FillWeight** = 20, each other **FillWeight** = 100

- loop: **For Each row As CookieSalesDataSet.SalesRow In CookieSalesDataSet.Sales.Rows**

**11.Utilities Solution-DataGrid\_EXERCISE 6\_intermediate**

- create database step by step, create dataset, create **DataGridView** control, fill table

- include **Try...Catch** statement

**12.Utilities Solution-Totals\_EXERCISE 7\_intermediate**

- modified: 11.Utilities Solution-DataGrid\_EXERCISE 6\_intermediate

- extra: button to calculate totals & labels to display them, clear labels after save

**13.Clancy Solution\_EXERCISE 8\_advanced**

- bound existing **DataGridView** control by dragging

- loop: **For Each row As ClancyDataSet.StoresRow In ClancyDataSet.Stores.Rows**

**14.Course Info Solution-GPA\_EXERCISE 9\_advanced**

- modified: 05.Course Info Solution-Total Hours

- loop: **For Each row As MyCoursesDataSet.CoursesRow In MyCoursesDataSet.Courses.Rows**

- extra: display GPA points based on value in Grade cell

**15.Global Solution\_EXERCISE 10\_advanced**

- create database; create and fill 2 tables; create dataset from 2 tables related by their common field

- create **DataGridView** control only for view

**16.OnYourOwn Solution\_EXERCISE 11\_NET 4.5.2**

- create database; create and fill 1 table; create **DataGridView** control; perform % calculation

- use **Try...Catch** "safety net statement"

**17.FixIt Solution\_EXERCISE 12**

- error in loop code:

- originally: **For Each row As InventoryDataSet In InventoryDataSet.OnHand.Rows**

**intTotal = row**

- should be: **For Each row As InventoryDataSet.OnHandRow In InventoryDataSet.OnHand.Rows**

**intTotal += row.Quantity**

- create **DataGridView** control
- proportional width of all columns

In this exercise, you create an application that keeps track of cookie sales. Create a Windows Forms application. Use the following names for the project and solution, respectively: Cookies Project and Cookies Solution. Save the application in the VB2017\Chap11 folder.

- a. Figure 11-59 shows the Sales table contained in the VB2017\Chap11\Databases\ Cookies.mdf file. The table contains the numbers of boxes of cookies sold in each of three weeks. The Week field is an auto-numbered field. Open the Data Sources window and click Add New Data Source to start the Data Source Configuration Wizard. Connect the Cookies.mdf file to the application. Include the entire Sales table in the dataset.
- b. Set the Cookies.mdf file's Copy to Output Directory property to "Copy if newer".
- c. Display the dataset information in a DataGridView control and then make the necessary modifications to the control.
- d. Enter an appropriate Try...Catch statement in the SalesBindingNavigatorSaveItem\_Click procedure.

- e. Save the solution and then start the application. Change the Chocolate Chip cookie sales in Week 1 to 205. Then, enter the following record for Week 4: 150, 112, and 76. Save the changes.
- f. Stop the application and then start it again to verify that the changes you made were saved.

| Week | Chocolate Chip | Peanut Butter | Pecan Sandies |
|------|----------------|---------------|---------------|
| 1    | 200            | 150           | 75            |
| 2    | 185            | 170           | 100           |
| 3    | 165            | 160           | 120           |

Figure 11-59 Sales table for Exercise 1 (before any changes)

```

1  Option Explicit On
2  Option Strict On
3  Option Infer Off
4
5  Public Class frmMain
6      Private Sub SalesBindingNavigatorSaveItem_Click(sender As Object, e As EventArgs) Handles SalesBindingNavigatorSaveItem.Click
7          Try
8              Me.Validate()
9              Me.SalesBindingSource.EndEdit()
10             Me.TableAdapterManager.UpdateAll(Me.CookiesDataSet)
11             MessageBox.Show("Changes saved.", "Cookies Sale", MessageBoxButtons.OK, MessageBoxIcon.Information)
12         Catch ex As Exception
13             MessageBox.Show(ex.Message, "Cookies Sale", MessageBoxButtons.OK, MessageBoxIcon.Information)
14         End Try
15     End Sub
16
17     Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles MyBase.Load
18         'TODO: This line of code loads data into the 'CookiesDataSet.Sales' table. You can move, or remove it, as needed.
19         Me.SalesTableAdapter.Fill(Me.CookiesDataSet.Sales)
20
21     End Sub
22 End Class

```

Cookie Sales

1 of 3

|   | Week | Chocolate Chip | Peanut Butter | Pecan Sandies |
|---|------|----------------|---------------|---------------|
| ▶ | 1    | 200            | 150           | 75            |
|   | 2    | 185            | 170           | 100           |
|   | 3    | 165            | 160           | 120           |
| * |      |                |               |               |

Cookie Sales

4 of 4

|   | Week | Chocolate Chip | Peanut Butter | Pecan Sandies |
|---|------|----------------|---------------|---------------|
| ▶ | 1    | 200            | 150           | 75            |
|   | 2    | 185            | 170           | 100           |
|   | 3    | 165            | 160           | 120           |
| ▶ | 4    | 150            | 112           | 76            |
| * |      |                |               |               |

Cookies Sale

Changes saved.

OK

Cookie Sales

1 of 4

|   | Week | Chocolate Chip | Peanut Butter | Pecan Sandies |
|---|------|----------------|---------------|---------------|
| ▶ | 1    | 200            | 150           | 75            |
|   | 2    | 185            | 170           | 100           |
|   | 3    | 165            | 160           | 120           |
| ▶ | 4    | 150            | 112           | 76            |
| * |      |                |               |               |

## 07.MusicBox Solution\_EXERCISE 2\_introductory

- create **DataGridView** control
- non proportional table's column widths when **BoxesDataGridView**'s property: **AutoSizeColumnsMode = Fill**
- **DataGridView Tasks / Edit Columns...** / dialog box **Edit Columns / Selected Columns** property **FillWeight**
- property **FillWeight** = the weight that is used when sizing this column in the Fill auto size mode
- **ID**'s property **FillWeight** = 44.82118, **Shape**'s property **FillWeight** = 78.79391
- **Source**'s property **FillWeight** = 78.79391, **Song**'s property **FillWeight** = 132.2225

2. In this exercise, you create an application that keeps track of music boxes. Create a Windows Forms application. Use the following names for the project and solution, respectively: MusicBox Project and MusicBox Solution. Save the application in the VB2017\Chap11 folder.

- Figure 11-60 shows the Boxes table contained in the VB2017\Chap11\Databases\MusicBoxes.mdf file. The ID field is an auto-numbered field. Open the Data Sources window and click Add New Data Source to start the Data Source Configuration Wizard. Connect the MusicBoxes.mdf file to the application. Include the entire Boxes table in the dataset.
- Set the MusicBoxes.mdf file's Copy to Output Directory property to "Copy if newer".
- Display the dataset information in a DataGridView control and then make the necessary modifications to the control.
- Enter an appropriate Try...Catch statement in the BoxesBindingNavigatorSaveItem\_Click procedure.
- Save the solution and then start the application. Change record 9's Source field to Gift. Then, enter the following new record: Round, Purchase, and Music of the Night. Save the changes.

- Stop the application and then start it again to verify that the changes you made were saved.

| ID | Shape     | Source   | Song                 |
|----|-----------|----------|----------------------|
| 1  | Round     | Purchase | As Time Goes By      |
| 2  | Octagon   | Gift     | Nadia's Theme        |
| 3  | Round     | Purchase | My Way               |
| 4  | Rectangle | Purchase | Clair de Lune        |
| 5  | Octagon   | Purchase | Beauty and the Beast |
| 6  | Rectangle | Gift     | Endless Love         |
| 7  | Rectangle | Gift     | Yesterday            |
| 8  | Octagon   | Purchase | You Light Up My Life |
| 9  | Rectangle | Purchase | Edelweiss            |
| 10 | Rectangle | Gift     | Happy Birthday       |

Figure 11-60 Boxes table for Exercise 2 (before any changes)

```
1 Option Explicit On
2 Option Strict On
3 Option Infer Off
4
5 Public Class frmMain
6     Private Sub BoxesBindingNavigatorSaveItem_Click(sender As Object, e As EventArgs) Handles BoxesBindingNavigatorSaveItem.Click
7         Try
8             Me.Validate()
9             Me.BoxesBindingSource.EndEdit()
10            Me.TableAdapterManager.UpdateAll(Me.MusicBoxesDataSet)
11            MessageBox.Show("Changes saved.", "Music Boxes", MessageBoxButtons.OK, MessageBoxIcon.Information)
12        Catch ex As Exception
13            MessageBox.Show(ex.Message, "Music Boxes", MessageBoxButtons.OK, MessageBoxIcon.Information)
14        End Try
15    End Sub
16
17    Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles MyBase.Load
18        'TODO: This line of code loads data into the 'MusicBoxesDataSet.Boxes' table. You can move, or remove it, as needed.
19        Me.BoxesTableAdapter.Fill(Me.MusicBoxesDataSet.Boxes)
20    End Sub
21 End Class
```

Music Boxes

1 of 10 | + X

|   | ID | Shape     | Source   | Song                 |
|---|----|-----------|----------|----------------------|
| ▶ | 1  | Round     | Purchase | As Time Goes By      |
|   | 2  | Octagon   | Gift     | Nadia's Theme        |
|   | 3  | Round     | Purchase | My Way               |
|   | 4  | Rectangle | Purchase | Clair de Lune        |
|   | 5  | Octagon   | Purchase | Beauty and the Beast |
|   | 6  | Rectangle | Gift     | Endless Love         |
|   | 7  | Rectangle | Gift     | Yesterday            |
|   | 8  | Octagon   | Purchase | You Light Up My Life |
|   | 9  | Rectangle | Purchase | Edelweiss            |
|   | 10 | Rectangle | Gift     | Happy Birthday       |
| * |    |           |          |                      |

Music Boxes

11 of 11 | + X

|   | ID | Shape     | Source   | Song                 |
|---|----|-----------|----------|----------------------|
|   | 2  | Octagon   | Gift     | Nadia's Theme        |
|   | 3  | Round     | Purchase | My Way               |
|   | 4  | Rectangle | Purchase | Clair de Lune        |
|   | 5  | Octagon   | Purchase | Beauty and the Beast |
|   | 6  | Rectangle | Gift     | Endless Love         |
|   | 7  | Rectangle | Gift     | Yesterday            |
|   | 8  | Octagon   | Purchase | You Light Up My Life |
|   | 9  | Rectangle | Gift     | Edelweiss            |
|   | 10 | Rectangle | Gift     | Happy Birthday       |
| ▶ | 11 | Round     | Purchase | Music of the Night   |
| * |    |           |          |                      |

Music Boxes

Changes saved.

OK

Music Boxes

1 of 11 | + X

|   | ID | Shape     | Source   | Song                 |
|---|----|-----------|----------|----------------------|
| ▶ | 1  | Round     | Purchase | As Time Goes By      |
|   | 2  | Octagon   | Gift     | Nadia's Theme        |
|   | 3  | Round     | Purchase | My Way               |
|   | 4  | Rectangle | Purchase | Clair de Lune        |
|   | 5  | Octagon   | Purchase | Beauty and the Beast |
|   | 6  | Rectangle | Gift     | Endless Love         |
|   | 7  | Rectangle | Gift     | Yesterday            |
|   | 8  | Octagon   | Purchase | You Light Up My Life |
|   | 9  | Rectangle | Gift     | Edelweiss            |
|   | 10 | Rectangle | Gift     | Happy Birthday       |
|   | 11 | Round     | Purchase | Music of the Night   |
| * |    |           |          |                      |

## 08.Sales Solution\_EXERCISE 3\_introductory - create a data form with a Label and TextBoxes

In this exercise, you create an application that keeps track of cookie sales. Create a Windows Forms application. Use the following names for the project and solution, respectively: Sales Project and Sales Solution. Save the application in the VB2017\Chap11 folder.

- a. Figure 11-61 shows the Sales table contained in the VB2017\Chap11\Databases\ Cookies.mdf file. The table contains the numbers of boxes of cookies sold in each of three weeks. The Week field is an auto-numbered field. Open the Data Sources window and click Add New Data Source to start the Data Source Configuration Wizard. Connect the Cookies.mdf file to the application. Include the entire Sales table in the dataset.
- b. Set the Cookies.mdf file's Copy to Output Directory property to "Copy if newer".
- c. Create a data form for the user to enter the sales information. (The Week field's data should appear in a label control.) Lock the controls on the form and then set the tab order.
- d. Rename the text boxes as follows: txtChocolate, txtPeanut, and txtPecan.
- e. Each text box should accept only numbers and the Backspace key. Create an event-handling Sub procedure that will handle the three event procedures.

- f. Enter an appropriate Try...Catch statement in the SalesBindingNavigatorSaveItem\_Click procedure.
- g. Save the solution and then start the application. Change the Peanut Butter sales for Week 1 to 125. Then, enter the following record for Week 4: 100, 100, and 100. Save the changes.
- h. Stop the application and then start it again to verify that the changes you made were saved.

| Week | Chocolate Chip | Peanut Butter | Pecan Sandies |
|------|----------------|---------------|---------------|
| 1    | 200            | 150           | 75            |
| 2    | 185            | 170           | 100           |
| 3    | 165            | 160           | 120           |

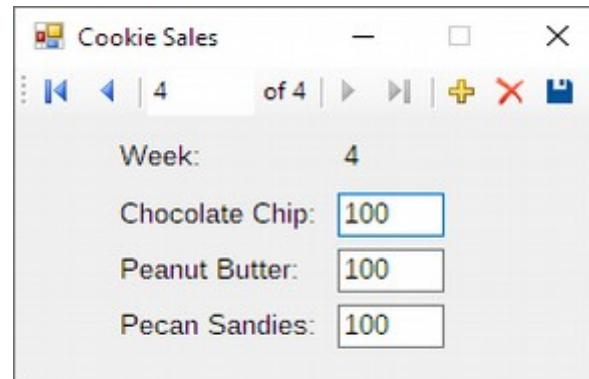
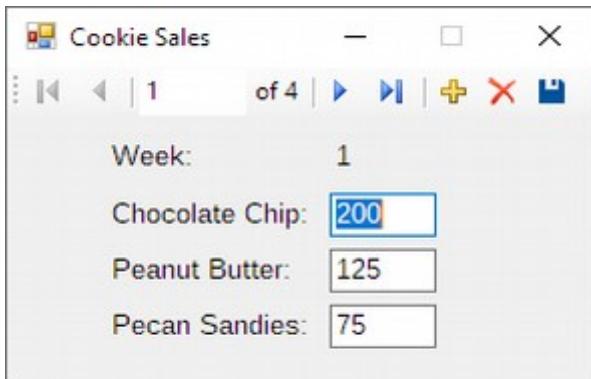
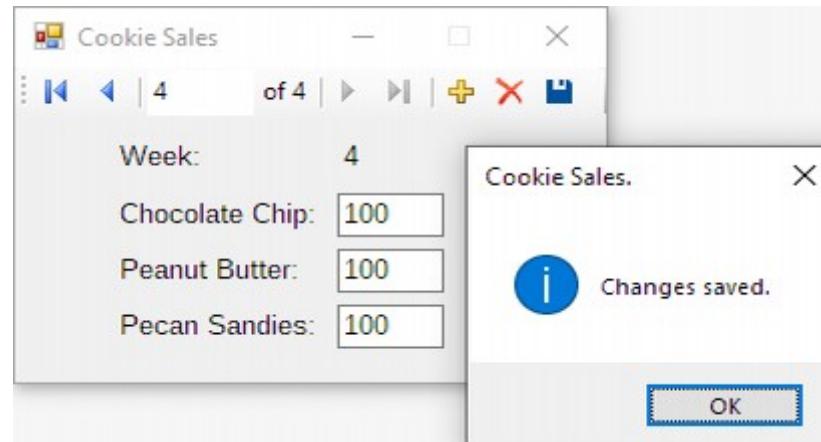
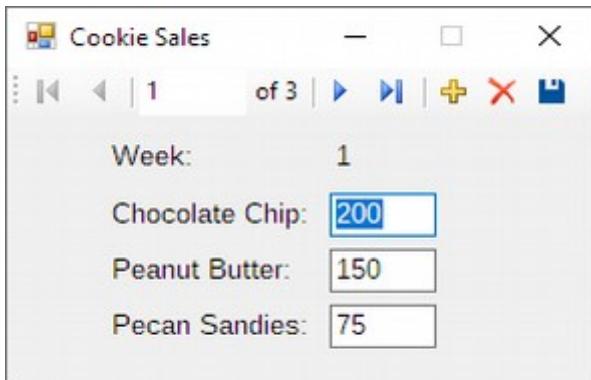
Figure 11-61 Sales table for Exercise 3 (before any changes)

```
1  Option Explicit On
2  Option Strict On
3  Option Infer Off
4
5  Public Class frmMain
6      Private Sub SalesBindingNavigatorSaveItem_Click(sender As Object, e As EventArgs) Handles SalesBindingNavigatorSaveItem.Click
7          Try
8              Me.Validate()
9              Me.SalesBindingSource.EndEdit()
10             Me.TableAdapterManager.UpdateAll(Me.CookiesDataSet)
11             MessageBox.Show("Changes saved.", "Cookie Sales.", MessageBoxButtons.OK, MessageBoxIcon.Information)
12         Catch ex As Exception
13             MessageBox.Show(ex.Message, "Cookie Sales.", MessageBoxButtons.OK, MessageBoxIcon.Information)
14         End Try
15     End Sub
16
17     Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles MyBase.Load
18         'TODO: This line of code loads data into the 'CookiesDataSet.Sales' table. You can move, or remove it, as needed.
19         Me.SalesTableAdapter.Fill(Me.CookiesDataSet.Sales)
20     End Sub
21 
```

```

22  Private Sub txts_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtChocolate.KeyPress, txtPeanut.KeyPress
23      ' input TextBoxes will accept only integers and the BackSpace key:
24      If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
25          e.Handled = True
26      End If
27  End Sub
28
29 End Class

```



## 09.MusicBox Solution-ListBox\_EXERCISE 4\_introductory

- bind existing **Label** controls by dragging selected **Table fields**
- bind existing **ListBox** control by setting it's **Properties**: **DataSource = BoxesBindingSource**  
**DisplayMember = ID**

4. Open the MusicBox Solution.sln file contained in the VB2017\Chap11\MusicBox Solution-ListBox folder.

- Figure 11-62 shows the Boxes table contained in the VB2017\Chap11\Databases\MusicBoxes.mdf file. Open the Data Sources window and click Add New Data Source to start the Data Source Configuration Wizard. Connect the MusicBoxes.mdf file to the application. Include the entire Boxes table in the dataset.
- Bind the Shape, Source, and Song field objects to the existing labels controls. Then, set the lstId control's DataSource and DisplayMember properties to BoxesBindingSource and ID, respectively. Save the solution and then start the application. Test the application by clicking each ID in the lstIds control.

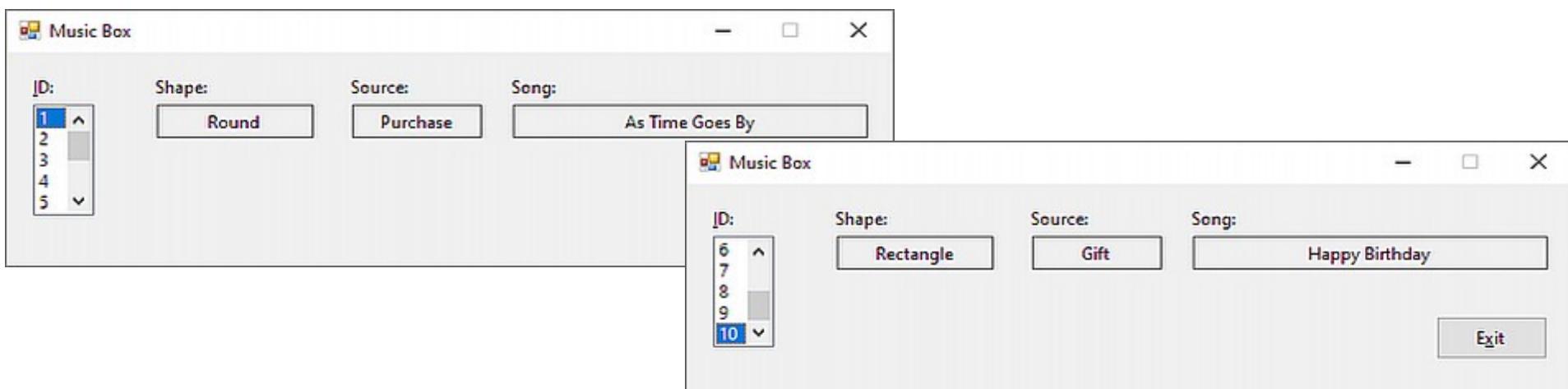
```

1 Option Explicit On
2 Option Strict On
3 Option Infer Off
4
5 Public Class frmMain
6     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
7         Me.Close()
8     End Sub
9
10    Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles MyBase.Load
11        'TODO: This line of code loads data into the 'MusicBoxesDataSet.Boxes' table. You can move, or remove it, as needed.
12        Me.BoxesTableAdapter.Fill(Me.MusicBoxesDataSet.Boxes)
13    End Sub
14 End Class

```

| ID | Shape     | Source   | Song                 |
|----|-----------|----------|----------------------|
| 1  | Round     | Purchase | As Time Goes By      |
| 2  | Octagon   | Gift     | Nadia's Theme        |
| 3  | Round     | Purchase | My Way               |
| 4  | Rectangle | Purchase | Clair de Lune        |
| 5  | Octagon   | Purchase | Beauty and the Beast |
| 6  | Rectangle | Gift     | Endless Love         |
| 7  | Rectangle | Gift     | Yesterday            |
| 8  | Octagon   | Purchase | You Light Up My Life |
| 9  | Rectangle | Purchase | Edelweiss            |
| 10 | Rectangle | Gift     | Happy Birthday       |

Figure 11-62 Boxes table for Exercise 4



## 10.Total Cookie Sales Solution\_EXERCISE 5\_intermediate

- right-align the numbers in columns: **DataGriedView1 / Properties / DefaultCellStyle** / oval button & **CellStyle Builder** dialog box -> **Alignment = MiddleRight**
- **DataGridView1**'s property: **AutoSizeColumnsMode = Fill**
- non proportional table's column widths:
  - **DataGridView Tasks / Edit Columns...** & dialog box **Edit Columns**
  - property **FillWeight** = the weight that is used when sizing this column in the Fill auto size mode
  - column **Week**: **FillWeighth = 20**, each other **FillWeight = 100**
- loop: **For Each row As CookieSalesDataSet.SalesRow In CookieSalesDataSet.Sales.Rows**

Open the Total Cookie Sales Solution.sln file contained in the VB2017\Chap11\Total Cookie Sales Solution folder.

- Figure 11-63 shows the Sales table contained in the CookieSales.mdf file. The table contains the numbers of boxes of cookies sold in each of six weeks. The database is already connected to the application and the CookieSalesDataSet has already been created.
- Open the Data Sources window and then drag the Sales table to the DataGridView control. Change the control's AutoSizeColumnsMode to Fill. Use the control's task list to disable adding, editing, and deleting records. Also, right-align the numbers in the cookie sales columns.
- Lock the controls on the form. Start the application to verify that the six records appear in the DataGridView control. Stop the application.

- The Calculate button should display the total sales for each cookie type. Code the **btnCalc\_Click** procedure. (The database does not allow NULLs in any of the fields, so you do not need to check if a field contains the NULL value.)
- Save the solution and then start and test the application.

| Week | Chocolate Chip | Peanut Butter | Pecan Sandies |
|------|----------------|---------------|---------------|
| 1    | 200            | 150           | 75            |
| 2    | 185            | 170           | 100           |
| 3    | 165            | 160           | 120           |
| 4    | 120            | 125           | 110           |
| 5    | 95             | 80            | 50            |
| 6    | 101            | 100           | 100           |

Figure 11-63 Sales table for Exercise 5

```
1  ' Name:      Total Cookie Sales Project
2  ' Purpose:    Display the total number of boxes of each cookie type sold.
3  ' Programmer: <your name> on <current date>
4
5  Option Explicit On
6  Option Strict On
7  Option Infer Off
8
9  Public Class frmMain
10  Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
11      Me.Close()
12  End Sub
13
14  Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles MyBase.Load
15      'TODO: This line of code loads data into the 'CookieSalesDataSet.Sales' table. You can move, or remove it, as needed.
16      Me.SalesTableAdapter.Fill(Me.CookieSalesDataSet.Sales)
17  End Sub
```

```

18
19  Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
20      ' Display the total sales for each cookie type in: lblChocolate, lblPeanut, lblPecan.
21      Dim intTTLChocolate As Integer : Dim intTTLPeanut As Integer : Dim intTTLPecan As Integer
22
23      For Each row As CookieSalesDataSet.SalesRow In CookieSalesDataSet.Sales.Rows
24          intTTLChocolate += row.Chocolate_Chip
25          intTTLPeanut += row.Peanut_Butter
26          intTTLPecan += row.Pecan_Sandies
27      Next row
28      lblChocolate.Text = intTTLChocolate.ToString
29      lblPeanut.Text = intTTLPeanut.ToString
30      lblPecan.Text = intTTLPecan.ToString
31  End Sub
32 End Class

```

The screenshot shows a Windows application window titled "Cookie Sales". The main area contains a data grid with the following data:

|   | Week | Chocolate Chip | Peanut Butter | Pecan Sandies |
|---|------|----------------|---------------|---------------|
| 1 | 200  | 150            | 75            |               |
| 2 | 185  | 170            | 100           |               |
| 3 | 165  | 160            | 120           |               |
| 4 | 120  | 125            | 110           |               |
| 5 | 95   | 80             | 50            |               |
| 6 | 101  | 100            | 100           |               |

Below the grid, there is a "Totals" section with three entries:

- Chocolate Chip:
- Peanut Butter:
- Pecan Sandies:

On the right side of the totals section are two buttons: "Calculate" and "Exit".

## 11.Utilities Solution-DataGrid\_EXERCISE 6\_intermediate

- create database step by step, create dataset, create **DataGridView** control, fill table  
- include Try...Catch statement

Open the Utilities Solution.sln file contained in the VB2017\Chap11\Utilities Solution-DataGrid folder.

- Create a SQL Server database named Utilities.mdf.
- Add the Bills table definition shown in Figure 11-64 to the database. The Month field's (Is Identity), Identity Increment, and Identity Seed properties are set to True, 1, and 1, respectively. (Recall that you need to expand the Identity Specification property to access these properties.)
- After defining the table, click the Update button and then click the Update Database button.
- Open the Data Sources window and start the Data Source Configuration Wizard. Connect the Utilities.mdf file to the application. Include the entire Bills table in the dataset.
- Set the Utilities.mdf file's Copy to Output Directory property to "Copy if newer".
- Drag the Bills table to the form. Set the DataGridView control's AutoSizeColumnsMode property to Fill.
- Open the DataGridView control's task list and click Dock in Parent Container. Click Edit Columns. Change the Month column's AutoSizeMode property to ColumnHeader.

- Click Electricity in the Edit Columns dialog box, click DefaultCellStyle, click the ... (ellipsis) button, click Format, click the ... (ellipsis) button, click Numeric, and then click the OK button. The Format box now shows N2. Change the Alignment property to MiddleRight and then click the OK button to close the CellStyle Builder dialog box.
- Now, format the Water and Gas columns using the Numeric setting with two decimal places. Also, align the values in both columns using the MiddleRight setting. When you are finished setting the properties, close the Edit Columns dialog box.
- Change the form's Size property to 330, 200.
- Open the Code Editor window and enter an appropriate Try...Catch statement.
- Save the solution and then start the application. Enter the three records shown in Figure 11-64. (Recall that the Month field is an auto-numbered field. The numbers 1, 2, and 3 will appear when you click the Save Data button.)
- Stop the application and then start it again to verify that the three records were saved.

The screenshot shows the SQL Server Object Explorer with the Bills table selected. The table has four columns: Month (int, primary key, identity), Electricity (decimal(5,2)), Water (decimal(5,2)), and Gas (decimal(5,2)). The T-SQL script pane shows the CREATE TABLE statement:

```
CREATE TABLE [dbo].[Bills] -- be sure to change the table name to Bills
(
    [Month] INT NOT NULL PRIMARY KEY IDENTITY, -- auto-numbered field
    [Electricity] DECIMAL(5, 2) NOT NULL,
    [Water] DECIMAL(5, 2) NOT NULL,
    [Gas] DECIMAL(5, 2) NOT NULL
)
```

|   | Month | Electricity | Water | Gas   |
|---|-------|-------------|-------|-------|
| ▶ | 1     | 56.78       | 19.89 | 34.58 |
|   | 2     | 45.00       | 24.96 | 39.00 |
| * | 3     | 42.43       | 18.12 | 42.55 |

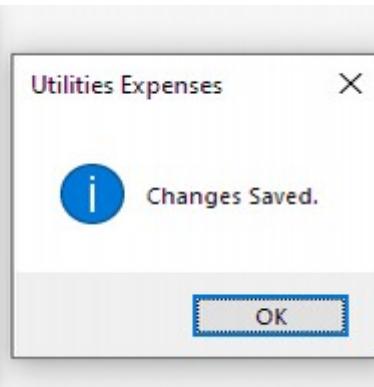
Figure 11-64 Table definition and records for Exercise 6

```

1  ' Name:      Utilities Project
2  ' Purpose:    Display utility bills.
3  ' Programmer: <your name> on <current date>
4
5  Option Explicit On
6  Option Strict On
7  Option Infer Off
8
9  Public Class frmMain
10     Private Sub BillsBindingNavigatorSaveItem_Click(sender As Object, e As EventArgs) Handles BillsBindingNavigatorSaveItem.Click
11         Try
12             Me.Validate()
13             Me.BillsBindingSource.EndEdit()
14             Me.TableAdapterManager.UpdateAll(Me.UtilitesDataSet)
15             MessageBox.Show("Changes Saved.", "Utilities Expenses", MessageBoxButtons.OK, MessageBoxIcon.Information)
16         Catch ex As Exception
17             MessageBox.Show(ex.Message, "Utilities Expenses", MessageBoxButtons.OK, MessageBoxIcon.Information)
18         End Try
19     End Sub
20
21     Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles MyBase.Load
22         'TODO: This line of code loads data into the 'UtilitesDataSet.Bills' table. You can move, or remove it, as needed.
23         Me.BillsTableAdapter.Fill(Me.UtilitesDataSet.Bills)
24     End Sub
25 End Class

```

|   | Month | Electricity | Water | Gas |
|---|-------|-------------|-------|-----|
| 1 | 56.78 | 19.89       | 34.58 |     |
| 2 | 45.00 | 24.96       | 39.00 |     |
| 3 | 42.43 | 18.12       | 42.55 |     |
| * |       |             |       |     |



|   | Month | Electricity | Water | Gas |
|---|-------|-------------|-------|-----|
| 1 | 56.78 | 19.89       | 34.58 |     |
| 2 | 45.00 | 24.96       | 39.00 |     |
| 3 | 42.43 | 18.12       | 42.55 |     |
| * |       |             |       |     |

## 12.Utilities Solution-Totals\_EXERCISE 7\_intermediate

- modified: 11.Utilities Solution-DataGrid\_EXERCISE 6\_intermediate  
- extra: button to calculate totals & labels to display them, clear labels after save

In this exercise, you modify the Utility Expenses application from Exercise 6. Use Windows to make a copy of the Utilities Solution-DataGrid folder. Rename the copy Utilities Solution-Totals.

- Open the Utilities Solution.sln file contained in the VB2017\Chap11\Utilities Solution-Totals folder. Undock the DataGridView control. The interface should now include a Calculate button that displays (in label controls) the following four values: the total cost for electricity, the total cost for water, the total cost for gas, and the total utility cost. Modify the interface and code. Display the totals with a dollar sign and two decimal places. Save the solution and then start and test the application.

- Stop the application. Each time the user clicks the Save Data button, the button's Click event procedure should clear the totals from the four labels. Modify the procedure's code.
- Save the solution and then start the application. Click the Calculate button. Now, enter the following values for month 4: 55, 20, and 50. Click the Save Data button and then close the message box. The button's Click event procedure clears the contents of the four labels.
- Click the Calculate button to verify that the values for month 4 are included in the totals.

```
1  ' Name:      Utilities Project
2  ' Purpose:    Display utility bills.
3  ' Programmer: <your name> on <current date>
4
5  Option Explicit On
6  Option Strict On
7  Option Infer Off
8
9  Public Class frmMain
10  Private Sub BillsBindingNavigatorSaveItem_Click(sender As Object, e As EventArgs) Handles BillsBindingNavigatorSaveItem.Click
11      Try
12          Me.Validate()
13          Me.BillsBindingSource.EndEdit()
14          Me.TableAdapterManager.UpdateAll(Me.UtilitesDataSet)
15          MessageBox.Show("Changes Saved.", "Utilities Expenses", MessageBoxButtons.OK, MessageBoxIcon.Information)
16
17          ' each time the user clicks button save, clear the ttls from 4 labels:
18          lbl1Electricity.Text = Nothing : lbl2Water.Text = Nothing : lbl3Gas.Text = Nothing : lbl4Utility.Text = Nothing
19
20      Catch ex As Exception
21          MessageBox.Show(ex.Message, "Utilities Expenses", MessageBoxButtons.OK, MessageBoxIcon.Information)
22      End Try
23  End Sub
24
25  Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles MyBase.Load
26      'TODO: This line of code loads data into the 'UtilitesDataSet.Bills' table. You can move, or remove it, as needed.
27      Me.BillsTableAdapter.Fill(Me.UtilitesDataSet.Bills)
28  End Sub
```

```

29
30  Private Sub btnCalculate_Click(sender As Object, e As EventArgs) Handles btnCalculate.Click
31      ' totals with $ and 2 decimals.
32      Dim dblElectricity As Double : Dim dblWater As Double : Dim dblGas As Double
33
34      For Each row As UtilitesDataSet.BillsRow In UtilitesDataSet.Bills.Rows
35          dblElectricity += row.Electricity
36          dblWater += row.Water
37          dblGas += row.Gas
38      Next row
39      lbl1Electricity.Text = dblElectricity.ToString("C2")
40      lbl2Water.Text = dblWater.ToString("C2")
41      lbl3Gas.Text = dblGas.ToString("C2")
42      lbl4Utility.Text = (dblElectricity + dblWater + dblGas).ToString("C2")
43  End Sub
44 End Class

```

**Utility Expenses**

|   | Month | Electricity | Water | Gas   |
|---|-------|-------------|-------|-------|
| ▶ | 1     | 56.78       | 19.89 | 34.58 |
|   | 2     | 45.00       | 24.96 | 39.00 |
|   | 3     | 42.43       | 18.12 | 42.55 |
| * |       |             |       |       |

**Calculate Totals**

Total cost for electricity:

Total cost for water:

Total cost for gas:

Total utility cost:

**Utility Expenses**

|   | Month | Electricity | Water | Gas   |
|---|-------|-------------|-------|-------|
|   | 1     | 56.78       | 19.89 | 34.58 |
|   | 2     | 45.00       | 24.96 | 39.00 |
|   | 3     | 42.43       | 18.12 | 42.55 |
| ▶ | 5     | 55.00       | 20.00 | 50.00 |
|   |       |             |       |       |

**Calculate Totals**

Total cost for electricity:

Total cost for water:

Total cost for gas:

Total utility cost:

### 13.Clancy Solution\_EXERCISE 8\_advanced

- bound existing DataGridView control by dragging

- loop: `For Each row As ClancyDataSet.StoresRow In ClancyDataSet.Stores.Rows`

8. Create a Windows Form application for Clancy Boutique. Use the following names for the project and solution, respectively: Clancy Project and Clancy Solution. Save the application in the VB2017\Chap11 folder.
- Create the interface shown in Figure 11-65. The interface contains a DataGridView control, a group box, six labels, and two buttons. (The DataGridView tool is located in the Data section of the toolbox.)
  - Figure 11-66 shows the table definition and records for the Stores table, which is contained in the VB2017\Chap11\Databases\Clancy.mdf file. The Ownership field indicates whether the store is company-owned (C) or a franchisee (F). Connect the database to the application. Include the entire table in the dataset.
  - Drag the Stores table to the DataGridView control. Use the control's task list to disable adding, editing, and deleting records. The numbers in the Sales column should be right-aligned and displayed with a comma and no decimal places. Center the letters in the Ownership column. Also, be sure that the entire city name appears in the City column. Make the necessary modifications to the control.
  - Lock the controls on the form and then set the tab order.
  - The Calculate button should display the total sales made by company-owned stores and the total sales made by franchisees. It should also display the total sales for all of the stores. Code the button's Click event procedure. Display the total sales amounts with a dollar sign and no decimal places.
  - Save the solution and then start and test the application.

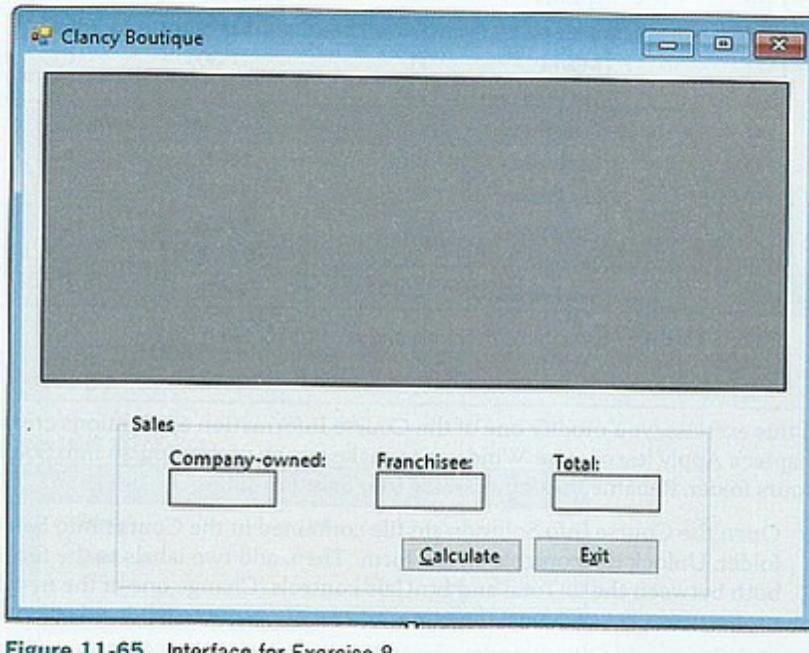


Figure 11-65 Interface for Exercise 8

|   | Name      | Data Type   | Allow Nulls              |
|---|-----------|-------------|--------------------------|
| 1 | Store     | int         | <input type="checkbox"/> |
| 2 | City      | varchar(40) | <input type="checkbox"/> |
| 3 | State     | char(2)     | <input type="checkbox"/> |
| 4 | Sales     | int         | <input type="checkbox"/> |
| 5 | Ownership | char(1)     | <input type="checkbox"/> |

| Store | City          | State | Sales  | Ownership |
|-------|---------------|-------|--------|-----------|
| 100   | San Francisco | CA    | 236700 | C         |
| 101   | San Diego     | CA    | 125900 | C         |
| 102   | Burbank       | CA    | 96575  | F         |
| 103   | Chicago       | IL    | 135400 | C         |
| 104   | Chicago       | IL    | 108000 | F         |
| 105   | Denver        | CO    | 212600 | C         |
| 106   | Atlanta       | GA    | 123500 | C         |
| 107   | Louisville    | KY    | 178500 | C         |
| 108   | Lexington     | KY    | 167450 | F         |
| 109   | Nashville     | TN    | 205625 | C         |
| 110   | Atlanta       | GA    | 198600 | F         |
| 111   | Denver        | CO    | 45900  | F         |
| 112   | Miami         | FL    | 175300 | C         |
| 113   | Las Vegas     | NV    | 245675 | C         |
| 114   | New Orleans   | LA    | 213400 | C         |
| 115   | Louisville    | KY    | 68900  | F         |
| 116   | Las Vegas     | NV    | 110340 | F         |
| 118   | Indianapolis  | IN    | 97500  | C         |
| 119   | Raleigh       | NC    | 86400  | C         |
| 120   | San Francisco | CA    | 65975  | F         |

Figure 11-66 Stores table definition and records for Exercise 8

```
1 Option Explicit On
2 Option Strict On
3 Option Infer Off
4
5 Public Class frmMain
6     Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles MyBase.Load
7         'TODO: This line of code loads data into the 'ClancyDataSet.Stores' table. You can move, or remove it, as needed.
8         Me.StoresTableAdapter.Fill(Me.ClancyDataSet.Stores)
9     End Sub
10
11    Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
12        Me.Close()
13    End Sub
14
15    Private Sub btnCalculate_Click(sender As Object, e As EventArgs) Handles btnCalculate.Click
16        ' field Ownership: "C" = Company-owned, "F" = Franchisee
17        ' lbls: $ and integers
18        Dim intC As Integer : Dim intF As Integer
19
20        For Each row As ClancyDataSet.StoresRow In ClancyDataSet.Stores.Rows
21            Select Case True
22                Case row.Ownership = "C"
23                    intC += row.Sales
24                Case Else
25                    intF += row.Sales
26            End Select
27        Next row
28        lbl1C.Text = intC.ToString("C0") : lbl2F.Text = intF.ToString("C0") : lbl3T.Text = (intC + intF).ToString("C0")
29    End Sub
30 End Class
```

Clancy Boutique

|   | Store | City          | State | Sales   | Ownership |
|---|-------|---------------|-------|---------|-----------|
| ▶ | 100   | San Francisco | CA    | 236,700 | C         |
|   | 101   | San Diego     | CA    | 125,900 | C         |
|   | 102   | Burbank       | CA    | 96,575  | F         |
|   | 103   | Chicago       | IL    | 135,400 | C         |
|   | 104   | Chicago       | IL    | 108,000 | F         |
|   | 105   | Denver        | CO    | 212,600 | C         |
|   | 106   | Atlanta       | GA    | 123,500 | C         |
|   | 107   | Louisville    | KY    | 178,500 | C         |

Sales

Company-owned: \$2,036,500

Franchisee: \$861,740

Total: \$2,898,240

#### 14.Course Info Solution-GPA\_EXERCISE 9\_advanced

- modified: 05.Course Info Solution-Total Hours
- loop: `For Each row As MyCoursesDataSet.CoursesRow In MyCoursesDataSet.Courses.Rows`
- extra: display GPA points based on value in Grade cell

In this exercise, you modify one of the Course Information applications created in this chapter's Apply lesson. Use Windows to make a copy of the Course Info Solution-Total Hours folder. Rename the copy Course Info Solution-GPA.

- a. Open the Course Info Solution.sln file contained in the Course Info Solution-GPA folder. Unlock the controls on the form. Then, add two labels to the form. Position both between the `lblTotal` and `btnCalc` controls. Change one of the new label's `Text` property to `GPA`. Change the other new label's name to `lblGpa`. Lock the controls.

- b. Open the Code Editor window. In addition to displaying the total number of hours completed, the `btnCalc_Click` procedure should also display the student's GPA. Grades of A, B, C, D, and F are worth 4 points, 3 points, 2 points, 1 point, and no points, respectively. Display the GPA with one decimal place. Modify the procedure's code.
- c. Save the solution and then start and test the application.

```
1  ' Name:    Course Info Project
2  ' Purpose: Add, edit, delete, display records and display total number of credit hours completed.
3  ' Name:    <your name> on <current date>
4
5  Option Explicit On
6  Option Strict On
7  Option Infer Off
8
9  Public Class frmMain
10  Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
11      ' Display the total number of credit hours completed.
12      ' and display student's GPA with 1 decimal in lblGpa.
13      Dim intTotal As Integer : Dim intGPA As Integer
14
15      For Each row As MyCoursesDataSet.CoursesRow In MyCoursesDataSet.Courses.Rows
16          If row.IsDBNull(Grade) = False AndAlso row.Grade <> "W" Then
17              intTotal += row.Hours
18
19              ' row.Grade: : A = 4, B = 3, C = 2, D = 1, F = 0 points.
20              Select Case True
21                  Case row.Grade = "A"
22                      intGPA += 4
23                  Case row.Grade = "B"
24                      intGPA += 3
25                  Case row.Grade = "C"
26                      intGPA += 2
27                  Case row.Grade = "D"
28                      intGPA += 1
29              End Select
30          End If
31      Next row
32      lblTotal.Text = intTotal.ToString
33      lblGpa.Text = intGPA.ToString("N1")
34  End Sub
```

```

35
36     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
37         Me.Close()
38     End Sub
39
40     Private Sub CoursesBindingNavigatorSaveItem_Click(sender As Object, e As EventArgs) Handles CoursesBindingNavigatorSaveItem.Click
41         Try
42             Me.Validate()
43             Me.CoursesBindingSource.EndEdit()
44             Me.TableAdapterManager.UpdateAll(Me.MyCoursesDataSet)
45             MessageBox.Show("Changes saved.", "Course Information", MessageBoxButtons.OK, MessageBoxIcon.Information)
46         Catch ex As Exception
47             MessageBox.Show(ex.Message, "Course Information", MessageBoxButtons.OK, MessageBoxIcon.Information)
48         End Try
49     End Sub
50
51     Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles MyBase.Load
52         'TODO: This line of code loads data into the 'MyCoursesDataSet.Courses' table. You can move, or remove it, as needed.
53         Me.CoursesTableAdapter.Fill(Me.MyCoursesDataSet.Courses)
54     End Sub
55 End Class

```

**Course Information**

|   | ID | Code     | Title                     | Hours | Grade |
|---|----|----------|---------------------------|-------|-------|
|   | 8  | EARTH112 | Astronomy                 | 3     | B     |
|   | 9  | ENG113   | Literature                | 3     | B     |
|   | 10 | MATH110  | Business Mathematics      | 3     | A     |
|   | 11 | MATH115  | Technical Mathematics     | 5     | B     |
|   | 12 | HISTO110 | Western Civilization      | 4     | A     |
|   | 13 | SOCIO100 | Introduction to Sociology | 3     |       |
|   | 14 | SPAN110  | Elementary Spanish I      | 4     | A     |
| * |    |          |                           |       |       |

Total hours completed: **40**    GPA: **40.0**    **Calculate**    **Exit**

## 15.Global Solution\_EXERCISE 10\_advanced

- create database; create and fill 2 tables; create dataset from 2 tables related by their common field
- create **DataGridView** control only for view

In this exercise, you create a two-table SQL Server database. You also create an application that displays the database information in a **DataGridView** control.

- Create a Windows Forms application. Use the following names for the project and solution, respectively: Global Project and Global Solution. Save the application in the VB2017\Chap11 folder.

- Create a SQL Server database named Global.mdf. Add the Salespeople table definition and records, which are shown in Figure 11-67, to the database. (Remember to click the Update button and then click the Update Database button after defining the table.) Then, add the Sales table definition and records, which are also shown in Figure 11-67, to the database.

Salespeople table and records

|             | Name    | Data Type   | Allow Nulls              | Default |
|-------------|---------|-------------|--------------------------|---------|
| primary key | SalesId | int         | <input type="checkbox"/> |         |
|             | Name    | varchar(50) | <input type="checkbox"/> |         |

CREATE TABLE [dbo].[Salespeople] (  
[SalesId] INT NOT NULL,  
[Name] VARCHAR (50) NOT NULL,  
);

| SalesId | Name            |
|---------|-----------------|
| 115     | Jack Parks      |
| 201     | Jose Guermo     |
| 363     | Sharon Williams |

Sales table and records

|  | Name    | Data Type | Allow Nulls              | Default |
|--|---------|-----------|--------------------------|---------|
|  | SalesId | int       | <input type="checkbox"/> |         |
|  | Sales   | int       | <input type="checkbox"/> |         |

CREATE TABLE [dbo].[Sales] (  
[SalesId] INT NOT NULL,  
[Sales] INT NOT NULL  
);

| SalesId | Sales |
|---------|-------|
| 115     | 75000 |
| 201     | 83000 |
| 363     | 56000 |

Figure 11-67 Table definitions and records for Exercise 10

- Open the Data Sources window and start the Data Source Configuration Wizard. Connect the Global.mdf file to the application. Include both tables in the dataset.
- Open the DataSet Designer window by right-clicking GlobalDataSet in the Data Sources window and then clicking Edit DataSet with Designer. Add a relation to the window. Relate both tables by the SalesId field. (The Salespeople table is the parent table, and the Sales table is the child table.)
- Right-click Fill,GetData() in the SalespeopleTableAdapter box and then click Configure. Open the Query Builder dialog box. Add the Sales table to the Diagram pane. The application will need to display the SalesId and Name fields from the Salespeople table along with the Sales field from the Sales table. Select the appropriate check box(es). Execute the query to verify that it retrieves the required information. Then, close the Query Builder dialog box and continue configuring the Fill and GetData methods.
- Save the solution and then close the GlobalDataSet.xsd window.

- Use the DataGridView tool, which is located in the Data section of the toolbox, to add a DataGridView control to the form. Drag the appropriate object from the Data Sources window to the control.
- The DataGridView control should not allow the user to add, edit, or delete records. When the application is started, its interface should appear similar to the one shown in Figure 11-68. Save the solution and then start and test the application.

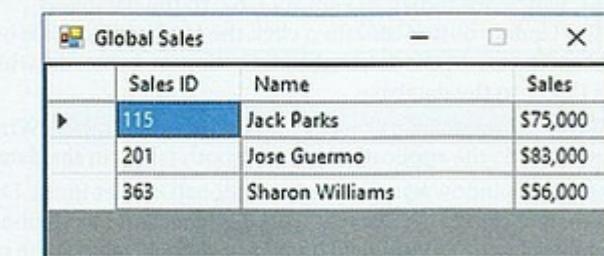


Figure 11-68 Global Sales application for Exercise 10

```
1  Public Class frmMain
2      Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles MyBase.Load
3          'TODO: This line of code loads data into the 'GlobalDataSet.Salespeople' table. You can move, or remove it, as needed.
4          Me.SalespeopleTableAdapter.Fill(Me.GlobalDataSet.Salespeople)
5      End Sub
6  End Class
```

|   | SalesId | Name            | Sales    |
|---|---------|-----------------|----------|
| ▶ | 115     | Jack Parks      | \$75,000 |
|   | 201     | Jose Guermo     | \$83,000 |
|   | 363     | Sharon Williams | \$56,000 |

#### 16.OnYourOwn Solution EXERCISE 11 .NET 4.5.2

- create database; create and fill 1 table; create **DataGridView** control; perform % calculation
- use **Try...Catch** "safety net statement"

Create a Windows Forms application. Use the following names for the project and solution, respectively: OnYourOwn Project and OnYourOwn Solution. Save the application in the VB2017\Chap11 folder. Plan and design an application of your choice. The only requirement is that you must follow the minimum guidelines listed in Figure 11-69. Before starting the application, be sure to verify the name of the startup form. Save the solution and then start and test the application.

1. You must create a SQL Server database that contains at least one table and at least four fields.
2. The application must allow the user to add, edit, save, and delete records.
3. The application must perform at least one calculation on a field in the database.
4. The interface must follow the GUI design guidelines summarized for Chapters 2 through 11 in Appendix A.
5. Objects that are either coded or referred to in code should be named appropriately.
6. The Code Editor window must contain comments and the three Option statements.

Figure 11-69 Guidelines for Exercise 11

```

1  Option Explicit On
2  Option Strict On
3  Option Infer Off
4
5  Public Class frmMain
6      Private Sub MyFactuurBindingNavigatorSaveItem_Click(sender As Object, e As EventArgs) Handles MyFactuurBindingNavigatorSaveItem.Click
7          Try
8              Me.Validate()
9              Me.MyFactuurBindingSource.EndEdit()
10             Me.TableAdapterManager.UpdateAll(Me.OnYourOwnDatabaseDataSet)
11             MessageBox.Show("Saved.", "% Calculator", MessageBoxButtons.OK, MessageBoxIcon.Information)
12             lblCounter.Text = Nothing : lblCZ.Text = Nothing : lblFR.Text = Nothing : lblResult.Text = Nothing
13         Catch ex As Exception
14             MessageBox.Show(ex.Message, "% Calculator", MessageBoxButtons.OK, MessageBoxIcon.Information)
15         End Try
16     End Sub

```

```

17
18  Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles MyBase.Load
19      'TODO: This line of code loads data into the 'OnYourOwnDatabaseDataSet.MyFactuur' table. You can move, or remove it, as needed.
20      Me.MyFactuurTableAdapter.Fill(Me.OnYourOwnDatabaseDataSet.MyFactuur)
21  End Sub
22
23  Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
24      Me.Close()
25  End Sub
26
27  Private Sub btnCalculate_Click(sender As Object, e As EventArgs) Handles btnCalculate.Click
28      Dim dblCZ As Double : Dim dblOther As Double : Dim dblALL As Double : Dim intCounter As Integer
29      For Each row As OnYourOwnDatabaseDataSet.MyFactuurRow In OnYourOwnDatabaseDataSet.MyFactuur.Rows
30          If row.Code = "CZ" Then
31              dblCZ += 1
32          Else
33              dblOther += 1
34          End If
35          intCounter += 1
36      Next row
37      'MessageBox.Show(intCZ.ToString) = 2 = 40%
38      'MessageBox.Show(intFR.ToString) = 3 = 60%
39
40      dblALL = dblCZ + dblOther ' intALL = 5 = 100%
41
42      dblCZ /= dblALL
43      dblOther = 1 - dblCZ
44
45      lblCZ.Text = dblCZ.ToString("P0")
46      lblFR.Text = dblOther.ToString("P0")
47      lblCounter.Text = intCounter.ToString
48
49      If dblCZ >= 0.25 Then
50          lblResult.ForeColor = Color.Green
51          lblResult.Text = "So far so good."
52      Else
53          lblResult.ForeColor = Color.Red
54          lblResult.Text = "Need some CZ invoices."
55      End If
56  End Sub
57 End Class

```

**A1 % calculator**      from 30/11/2020 till 29/11/2021

|   | ID | Factuur | Info         | Code |
|---|----|---------|--------------|------|
| ▶ | 1  | 2020007 | ICT Pannes   | FR   |
|   | 2  | 2020008 | Jiri Repík   | CZ   |
|   | 3  | 2021001 | ID Logistics | FR   |
|   | 4  | 2021002 | Máry         | CZ   |
|   | 5  | 2021003 | BIORAD       | FR   |
| • |    |         |              |      |

Currently:

|           |                                |                |
|-----------|--------------------------------|----------------|
| Calculate | CZ: <input type="text"/>       | Should be min: |
|           | Other: <input type="text"/>    | CZ: 25 %       |
|           | Invoices: <input type="text"/> | Other: 75 %    |

Exit

**A1 % calculator** from 30/11/2020 till 29/11/2021

| ID | Factuur | Info         | Code |
|----|---------|--------------|------|
| 1  | 2020007 | ICT Pannes   | FR   |
| 2  | 2020008 | Jiri Repik   | CZ   |
| 3  | 2021001 | ID Logistics | FR   |
| 4  | 2021002 | Marty        | CZ   |
| 5  | 2021003 | BIORAD       | FR   |
| *  |         |              |      |

Currently:  
 CZ:   
 Other:   
 Invoices:

Should be min:  
 CZ: 25 %  
 Other: 75 %

**Calculate** **Exit**

**A1 % calculator** from 30/11/2020 till 29/11/2021

| ID | Factuur | Info         | Code |
|----|---------|--------------|------|
| 1  | 2020007 | ICT Pannes   | FR   |
| 2  | 2020008 | Jiri Repik   | CZ   |
| 3  | 2021001 | ID Logistics | FR   |
| 4  | 2021002 | Marty        | CZ   |
| 5  | 2021003 | BIORAD       | FR   |
| *  |         |              |      |

Currently:  
 CZ: 40%  
 Other: 60%  
 Invoices: 5

Should be min:  
 CZ: 25 %  
 Other: 75 %

**Calculate** **Exit**

So far so good.

**A1 % calculator** from 30/11/2020 till 29/11/2021

| ID | Factuur | Info         | Code |
|----|---------|--------------|------|
| 3  | 2021001 | ID Logistics | FR   |
| 4  | 2021002 | Marty        | CZ   |
| 5  | 2021003 | BIORAD       | FR   |
| 16 | 123     | test         | EN   |
| 17 | 123     | test         | GY   |
| 18 | 123     | test         | EN   |
| 19 | 123     | test         | FR   |
| *  |         |              |      |

Currently:  
 CZ: 25 %  
 Other: 75 %  
 Invoices: 8

Should be min:  
 CZ: 25 %  
 Other: 75 %

**Calculate** **Exit**

So far so good.

**A1 % calculator** from 30/11/2020 till 29/11/2021

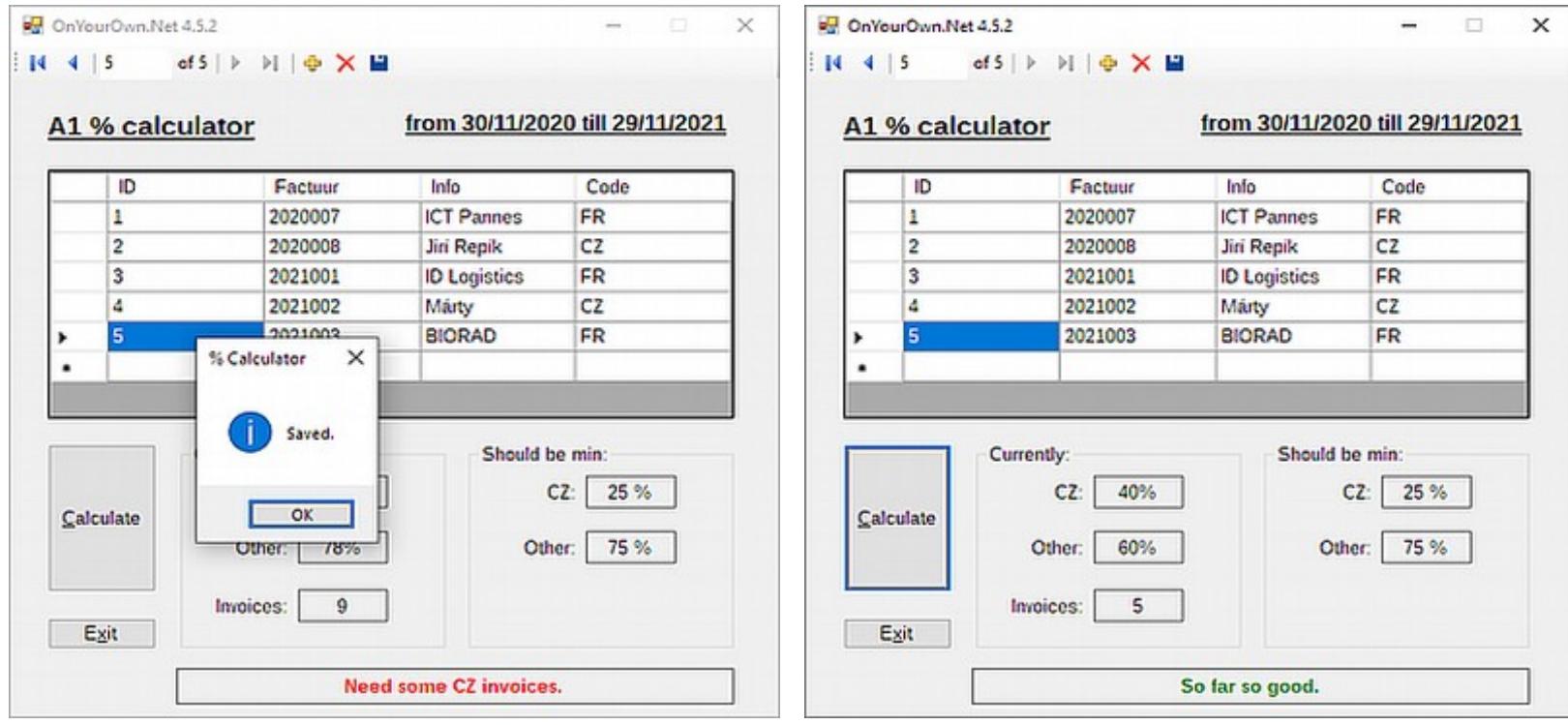
| ID | Factuur | Info         | Code |
|----|---------|--------------|------|
| 3  | 2021001 | ID Logistics | FR   |
| 4  | 2021002 | Marty        | CZ   |
| 5  | 2021003 | BIORAD       | FR   |
| 16 | 123     | test         | EN   |
| 17 | 123     | test         | GY   |
| 18 | 123     | test         | EN   |
| 19 | 123     | test         | FR   |
| *  |         |              |      |

Currently:  
 CZ: 22%  
 Other: 78%  
 Invoices: 9

Should be min:  
 CZ: 25 %  
 Other: 75 %

**Calculate** **Exit**

Need some CZ invoices.



### 17.FixIt Solution EXERCISE 12

- error in loop code:

- originally: `For Each row As InventoryDataSet In InventoryDataSet.OnHand.Rows  
 intTotal = row`

- should be: `For Each row As InventoryDataSet.OnHandRow In InventoryDataSet.OnHand.Rows  
 intTotal += row.Quantity`

- Open the VB2017\Chap11\FixIt Solution\FixIt Solution.sln file. Open the Code Editor window. Correct any errors in the code. Start the application. Click the Calculate button. The total on hand should be 125.

```

1  ' Name:      FixIt Project
2  ' Purpose:    Display the total amount on hand.
3  ' Programmer: <your name> on <current date>
4  Option Explicit On
5  Option Strict On
6  Option Infer Off
7
8  Public Class frmMain
9      Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
10         Me.Close()
11     End Sub
12
13     Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
14         ' Calculate the total inventory on hand.
15         Dim intTotal As Integer
16
17         'original error: Value of type 'InventoryDataSet' cannot be converted to 'Integer'
18         'For Each row As InventoryDataSet In InventoryDataSet.OnHand.Rows
19         'intTotal = row
20         'Next row
21
22         'MyFix:
23         For Each row As InventoryDataSet.OnHandRow In InventoryDataSet.OnHand.Rows
24             intTotal += row.Quantity
25         Next row
26
27         lblOnHand.Text = intTotal.ToString 'should be: 125
28     End Sub
29
30     Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles MyBase.Load
31         'TODO: This line of code loads data into the 'InventoryDataSet.OnHand' table. You can move, or remove it, as needed.
32         Me.OnHandTableAdapter.Fill(Me.InventoryDataSet.OnHand)
33     End Sub
34 End Class

```

| ID | Quantity |
|----|----------|
| 1  | 10       |
| 2  | 20       |
| 3  | 40       |
| 4  | 25       |
| 5  | 30       |

Total on hand: 125

# Database Queries with SQL

= retrieve a specific information from a database

Parameter Query & Parameter marker @; use SQL to add a Calculated field; use SQL Aggregate functions

- in CH11, you learned how to:
  - create a SQL Server database
  - perform common database tasks: - adding, editing, deleting, saving records, performing calculations on fields
- in CH12, you will learn how to:
  - perform another common database task - create the database **queries** using Structured Query Language - **SQL**
  - use **SQL** to add a calculated field to a dataset
  - use the **SQL aggregate** functions.

- for an application to use a **query** during run time, you need to:
  - 1). **create** the query - by using the **TableAdapter Query Configuration Wizard**
  - 2). **save** the query - by associating the query with one or more methods (Fill a DataTable, Return a DataTable - methods **Fill**, **Get**)
  - 3). **invoke** it from code - by entering in procedure the **Fill** method associated with the query

default Fill method's calling/invoking statement:

`YourTableAdapter.Fill(YourDataSet.YourTable)`

parameterized Fill method's calling/invoking statement:

`YourTableAdapter.YourParameterizedMethodFill(YourDataSet.YourTable, YourParameter)`

## CH12\_FOCUS ON THE CONCEPTS LESSON

- CH12\_F1 - **SELECT** statement syntax & its **WHERE** clause's condition operators & e.g.
- CH12\_F2 - how to create a **Query** matching specific criteria, using **Query Builder** dialog box - several examples with **01.Oscars Solution-SELECT**
- CH12\_F2.1 - **You Do It 1:** practice creating **Query** using **Query Builder** dialog box - 3 examples from Mini-Quiz 12-1 with **02.You Do It 1 Solution**
- CH12\_F3 - **Parameter Query & Parameter Marker @** - basic info & e.g.
- CH12\_F3.1 - how to use a **Parameter Query & parameter marker @** - using 3x e.g. from previous lesson, with solution **03.Oscars Solution-Parameter Queries**
- CH12\_F3.2 - **You Do It 2:** practice creating **Parameter Queries** using **Query Builder** dialog box - 3 examples from Mini-Quiz 12-2 with **04.You Do It 2 Solution**
- CH12\_F4 - use **Parameter Query** during run time - step 1/2: **Create** and then **Save** Query by associating it with a **method** - e.g. **05.Oscars Solution-Save Query**
- CH12\_F5 - use **Parameter Query** during run time - step 2/2: **Invoke/Call a method** associated with a **Query** - syntax, e.g. **05.Oscars Solution-Save Query**
- CH12\_F6 - **My Test:** - use **Parameter Query** with a **parameter marker @** during run time with a **LIKE** operator & allow users to use its wildcards %, \_
  - plus Q & A on several FB programmer's groups about **LIKE** operator
  - e.g. with: **05.Oscars Solution-Save Query\_My Test - Parameter Query and LIKE operator**

## CH12\_APPLY THE CONCEPTS LESSON

- CH12\_A1 - add a **Calculated Field** to a Dataset by modifying the default **SELECT** statement / using **Query Builder** e.g. with **06.Ellington Solution**
- CH12\_A2 - use the **SQL Aggregate Functions** like **AVG**, **COUNT**, **MAX**, **MIN**, **SUM** = return a single value from a group of values - basic info & e.g. using **SUM**:
  - create additional **TableAdapter** in the **DataSet**, whose modified default **SELECT** statement
  - will **create a new field** containing a **SUM** of the cells - e.g. with **07.Ellington Solution-Aggregate**

**CH12\_Summary:**

**CH12\_Key Terms & terminology marked:** ->

**CH12\_Exercises**

## CH12\_FOCUS ON THE CONCEPTS LESSON

### CH12\_F1 - SELECT statement syntax & its WHERE clause's condition operators & e.g.

- the most commonly used statement in **Structured Query Language**, or **SQL**, is the **SELECT** statement
- you use the **SELECT** statement to create database queries
- as you learned in CH11, a database query, often referred to more simply as a query, is a statement that allows you to retrieve specific information from a database
  - e.g. you can use a query to specify the fields and records you want either to display or to use in a calculation
- capitalizing the boldfaced keywords in a **SELECT** statement is optional; however, many programmers do so for clarity

SQL query SELECT statement basic syntax:

|                 |                       |
|-----------------|-----------------------|
| <b>SELECT</b>   | <i>fieldList</i>      |
| <b>FROM</b>     | <i>table</i>          |
| <b>WHERE</b>    | <i>condition</i>      |
| <b>ORDER BY</b> | <i>fieldName DESC</i> |

**SELECT fieldList FROM dbo.table WHERE condition ORDER BY fieldName DESC**

**fieldList** = one or more field names separated by commas

e.g. **SELECT** Year, Actor, Animated

**table** = name of the table containing the fields

e.g. **FROM** Winners

**WHERE** - optional clause

= contains some/compound condition to limit the records you want to retrieve

= similar to the condition in the **If...Then...Else** statement, it specifies a requirement that must be met for a record to be selected

- condition contains operators: **Figure 12-1**

- must use a single quotes '' when comparing text field with a literal constant

e.g.3 ...**WHERE** Picture = 'Argo'

- no need for single quotes '' when comparing number field with a lit. const.

e.g.2 ...**WHERE** Year >= 2014

- text comparisons are not case sensitive

e.g.3 ...**WHERE** Picture = 'argo'

**ORDER BY** - optional clause

= used to arrange the records by one or more fields

- default is ascending order

**DESC** = arrange the record in descending order

**Figure 12-1** operators for the WHERE clause's condition

|                |                                                                                                                                                     |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| =              | - equal to                                                                                                                                          |
| <>             | - not equal to                                                                                                                                      |
| >              | - greater than                                                                                                                                      |
| >=             | - greater than or equal to                                                                                                                          |
| <              | - less than                                                                                                                                         |
| <=             | - less than or equal to                                                                                                                             |
| <b>AND</b>     | - all subconditions must be <b>true</b> for the compound condition to evaluate to <b>True</b>                                                       |
| <b>OR</b>      | - only one of the subconditions needs to be <b>true</b> for the compound condition to evaluate to <b>True</b>                                       |
| <b>NOT</b>     | - reverses the <b>truth</b> -value of the condition                                                                                                 |
| <b>LIKE</b>    | - uses a wildcard character to compare text values<br>% = wildcard char for <b>0 or more</b> characters<br>_ = wildcard char for <b>1</b> character |
| <b>IS NULL</b> | - compares a value with a <b>NULL</b> value (empty)                                                                                                 |

**Figure 12-2** - a query matching specific criteria, with database: **Osars.mdf**, table: **dbo.Winners**

e.g.1 -> e.g.5

| Name     | Data Type   | Allow Nulls              | Default |
|----------|-------------|--------------------------|---------|
| Year     | int         | <input type="checkbox"/> |         |
| Actor    | varchar(50) | <input type="checkbox"/> |         |
| Actress  | varchar(50) | <input type="checkbox"/> |         |
| Picture  | varchar(50) | <input type="checkbox"/> |         |
| Animated | varchar(50) | <input type="checkbox"/> |         |

| Year | Actor               | Actress           | Picture                | Animated    |
|------|---------------------|-------------------|------------------------|-------------|
| 2008 | Daniel Day-Lewis    | Marion Cotillard  | No Country for Old Men | Ratatouille |
| 2009 | Sean Penn           | Kate Winslet      | Slumdog Millionaire    | WALL-E      |
| 2010 | Jeff Bridges        | Sandra Bullock    | The Hurt Locker        | Up          |
| 2011 | Colin Firth         | Natalie Portman   | The King's Speech      | Toy Story 3 |
| 2012 | Jean Dujardin       | Meryl Streep      | The Artist             | Rango       |
| 2013 | Daniel Day Lewis    | Jennifer Lawrence | Argo                   | Brave       |
| 2014 | Matthew McConaughey | Cate Blanchett    | 12 Years a Slave       | Frozen      |
| 2015 | Eddie Redmayne      | Julianne Moore    | Birdman                | Big Hero 6  |
| 2016 | Leonardo DiCaprio   | Brie Larson       | Spotlight              | Inside Out  |
| 2017 | Casey Affleck       | Emma Stone        | Moonlight              | Zootopia    |

e.g.1 SELECT Year, Actor, Actress, Picture, Animated FROM Winners

<- selects all of the fields and records from the table **Winners.dbo**

e.g.2 SELECT Year, Actor, Actress, Picture, Animated FROM Winners WHERE Year >= 2014

<- selects all of the fields from records for the Year 2014 and later

- no need for single quotes around 2014 when comparing number field Year (int) with a literal constant

e.g.3 SELECT Year FROM Winners WHERE Picture = 'Argo' / can use 'argo' - text comparisons are not case sensitive

<- selects the Year field for the record Argo

- necessary single quotes around 'Argo' when comparing text field Picture(varchar) with a literal constant 'Argo'

- text comparisons in SQL are not case sensitive

e.g.4 SELECT Year, Picture FROM Winners WHERE Picture LIKE 'The %'

<- selects the Year and Picture fields for all records whose Picture field begins with the word "The" followed by a space and 0 or more characters

- operator **LIKE** uses a wildcard character to compare text values

e.g.5 SELECT Year, Animated FROM Winners WHERE Year = 2010 OR Year = 2015 ORDER BY Year DESC

<- selects the Year and Animated fields for any record whose Year field contains either 2010 or 2015 and then arranges the records in descending order by the Year field

**Mini-Quiz 12-1** Using the **dbo.Winners** table from **Figure 12-2** , write a **SELECT** statement that:

Q1. selects only the **Actress** field for the year **2010**.

Q2. selects only the **Animated** field for records whose **Animated** field begins with the letter **R**. Sort the records in **ascending** order by the **Animated** field.

Q3. selects the **Actor** and **Actress** fields for the years **2008** through **2010**. Sort the records in descending order by the **Year** field.

A1. SELECT Actress FROM Winners WHERE Year = 2010

A2. SELECT Animated FROM Winners WHERE Animated LIKE 'R%' ORDER BY Animated

A3. SELECT Actor, Actress FROM Winners WHERE Year >= 2008 AND Year <= 2010 ORDER BY Year DESC

## CH12\_F2 - how to create a Query matching specific criteria, using **Query Builder** dialog box - several examples with 01.Oscars Solution-SELECT

- in this section, you will use the **Query Builder dialog box** to create queries for the **Figure 12-2** and examples: e.g.1 through e.g.5
- the queries will allow you to observe the way the **SELECT** statements in those examples retrieve the desired fields and records from the database
- you already learned a bit about **Query Builder dialog box** in previous: **CH11\_F9.3 - step 3/4: create a Database Query object ...**

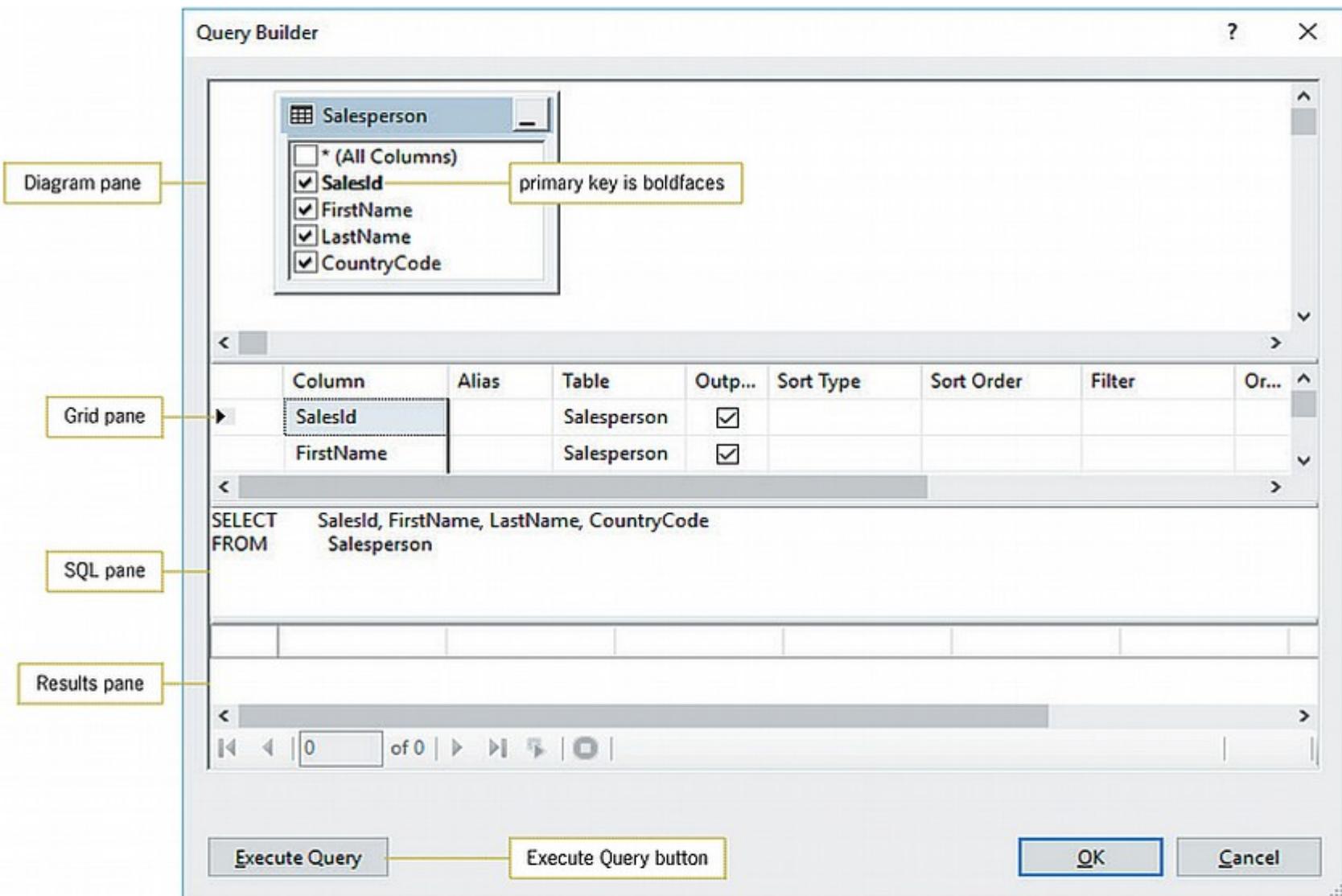


Figure 11-38c

- to create **queries** for the **Figure 12-2** & **e.g.1** -> **e.g.5** example with solution: **01.Oscars Solution-SELECT**

-> open the: ...VB2017\Chap12\Exercise01.Oscars Solution-SELECT\Oscars Solution.sln

-> open the **Designer** and **Solution Explorer** windows

- as you can see, The **Oscar Winners** application is already connected to the **Oscars.mdf** database, and the **OscarsDataSet.xsd** is already created

**step 1** -> start the application to view the **OscarsDataSet** containing 10 records: **Figure 12-3**

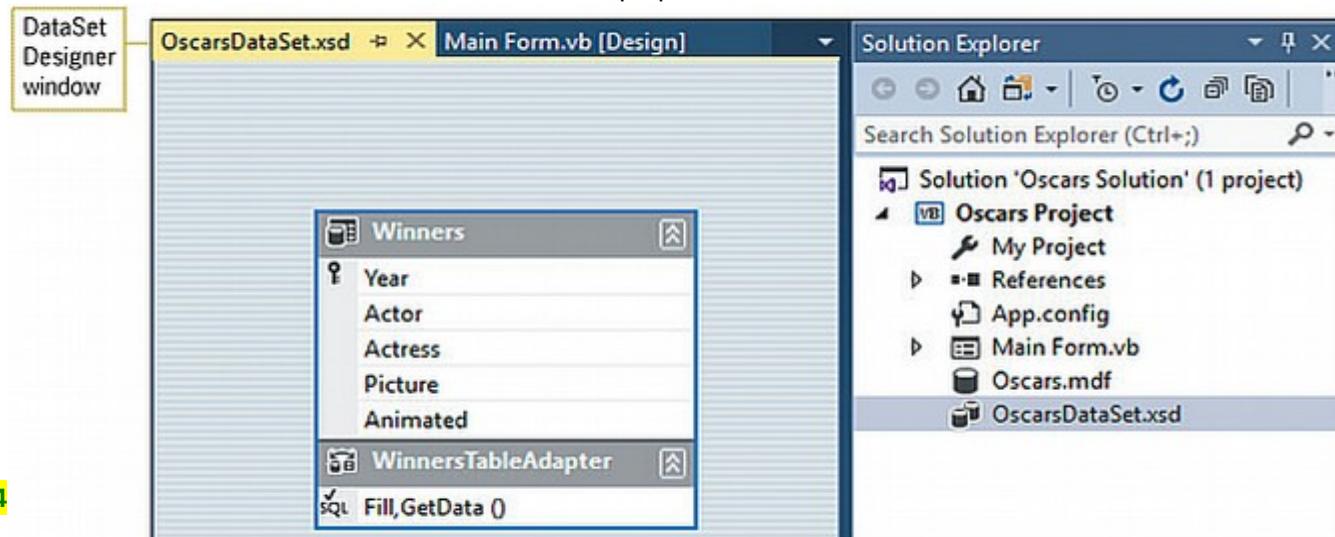
|   | Year | Actor               | Actress           | Picture                | Animated    |
|---|------|---------------------|-------------------|------------------------|-------------|
| ▶ | 2008 | Daniel Day-Lewis    | Marion Cotillard  | No Country for Old Men | Ratatouille |
|   | 2009 | Sean Penn           | Kate Winslet      | Slumdog Millionaire    | WALL-E      |
|   | 2010 | Jeff Bridges        | Sandra Bullock    | The Hurt Locker        | Up          |
|   | 2011 | Colin Firth         | Natalie Portman   | The King's Speech      | Toy Story 3 |
|   | 2012 | Jean Dujardin       | Meryl Streep      | The Artist             | Rango       |
|   | 2013 | Daniel-Day Lewis    | Jennifer Lawrence | Argo                   | Brave       |
|   | 2014 | Matthew McConaughey | Cate Blanchett    | 12 Years a Slave       | Frozen      |
|   | 2015 | Eddie Redmayne      | Julianne Moore    | Birdman                | Big Hero 6  |
|   | 2016 | Leonardo DiCaprio   | Brie Larson       | Spotlight              | Inside Out  |
|   | 2017 | Casey Affleck       | Emma Stone        | Moonlight              | Zootopia    |

Figure 12-3

Exit

-> click the **Exit** button to close the DataSet

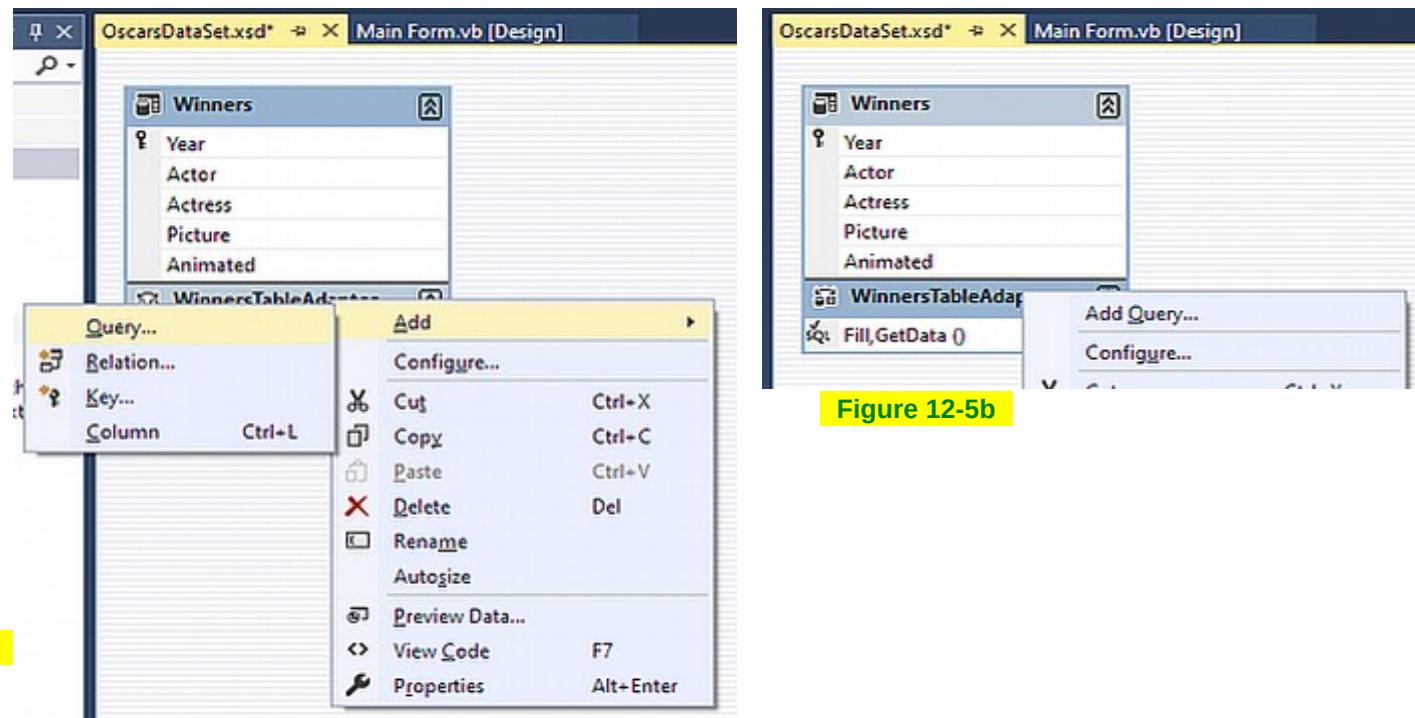
- step 2** -> in the **Solution Explorer** window, Rclick the **OscarsDataSet.xsd** and from the menu choose Open to open the **DataSet Designer** window: **Figure 12-4**
- or -> **Data Sources** window / **OscarsDataSet** / Rclick: **Edit DataSet with Designer**
- the **.xsd** file - called the dataset's schema file - contains info about the tables, fields, records, and properties included in the **OscarsDataSet.xsd**



**Figure 12-4**

- step 3** - in an order to create a query, you need to open the **TableAdapter Query Configuration Wizard**:

- > in the **DataSet Designer** - window named **OscarsDataSet.xsd** - Rclick the line: **WinnersTableAdapter** and choose: Add / Query... **Figure 12-5a**
- > if the line is already selected and you **Rclick** on it, choose the: **Add Query...** **Figure 12-5b**



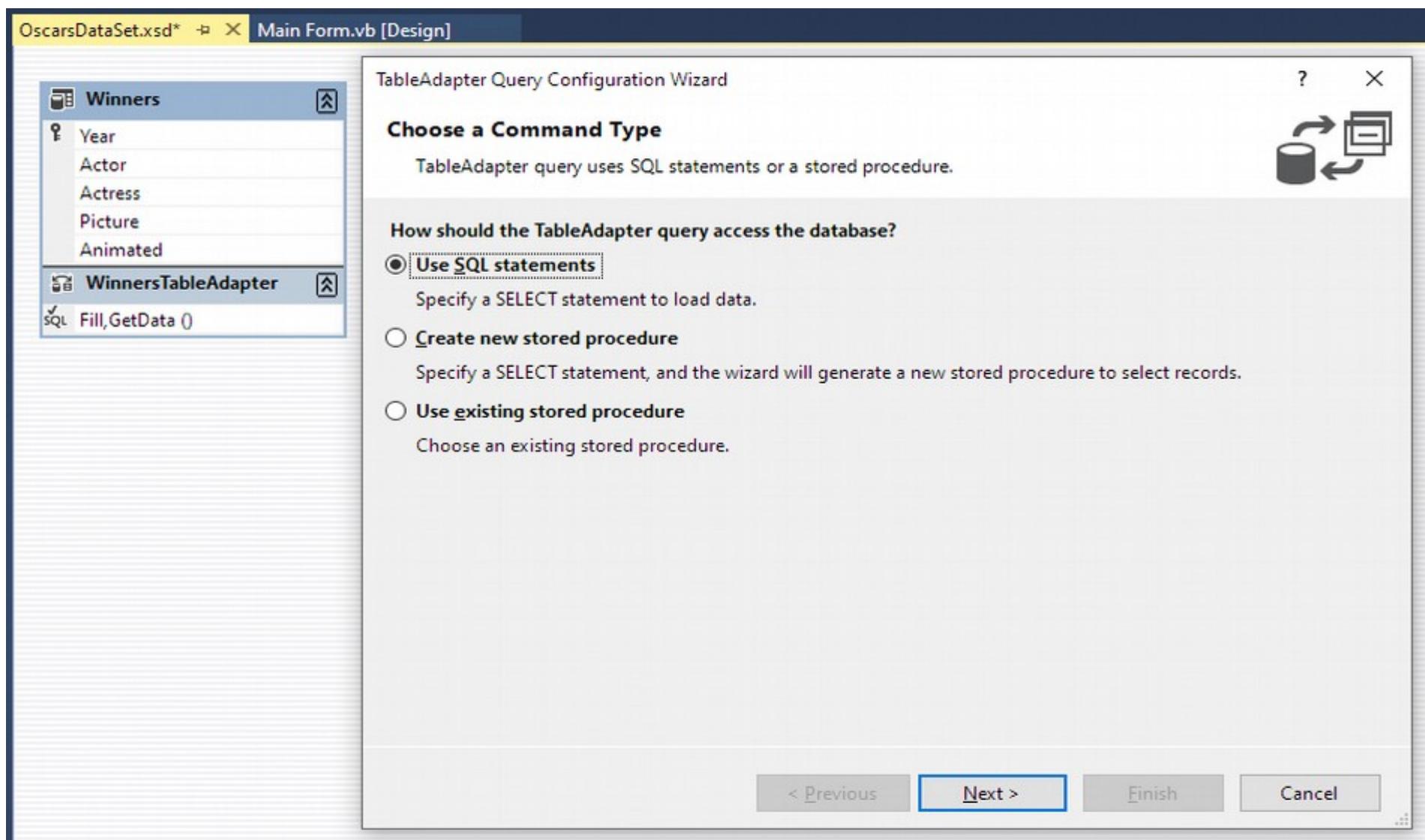
**Figure 12-5a**

**Figure 12-5b**

**step 4.1** - previous step started the **TableAdapter Query Configuration Wizard** and opened the 1st screen: **Choose a Command Type**

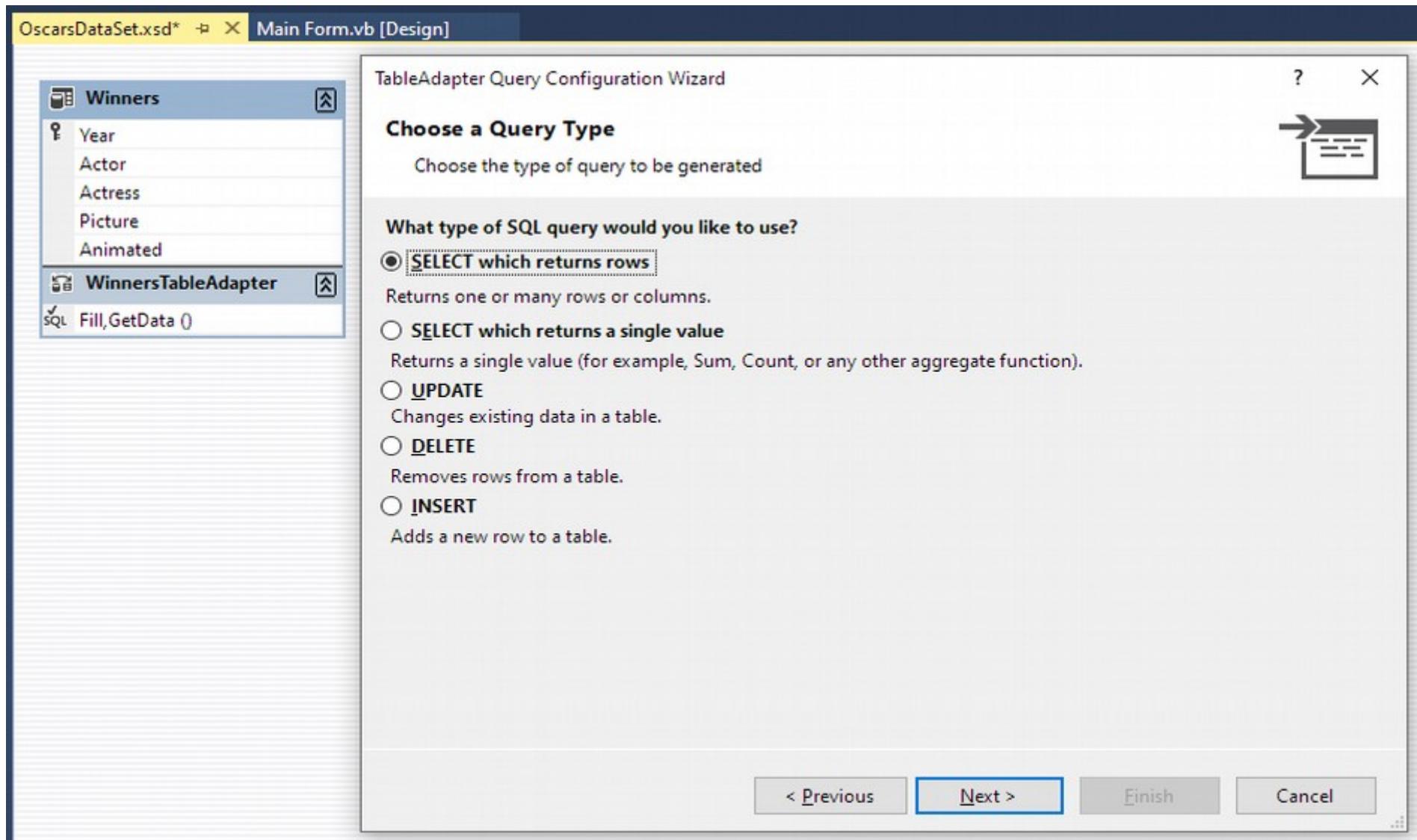
- > verify that the 1st screen: **Choose a Command Type** has selected radio button: **Use SQL statements**
- > click the button **Next >** to display the 2nd screen: **Choose a Query Type**

**Figure 12-6**



**Figure 12-6** 1st screen

- step 4.2** - previous step opened the 2nd screen: **Choose a Query Type:** **Figure 12-7**
- > verify that the 2nd screen: **Choose a Query Type** has selected radio button: **SELECT which returns rows**
  - > click the button **Next >** to display the 3rd screen: **Specify a SQL SELECT statement**



**Figure 12-7** 2nd screen

**step 4.3** - previous step opened the 3rd screen: **Specify a SQL SELECT statement**

**Figure 12-8**

<- notice the box: **What data should the table load?**

<- notice that the box contains the default query:    `SELECT Year, Actor, Actress, Picture, Animated FROM dbo.Winners`

- the default query: 1). selects all of the fields and records from the table, and

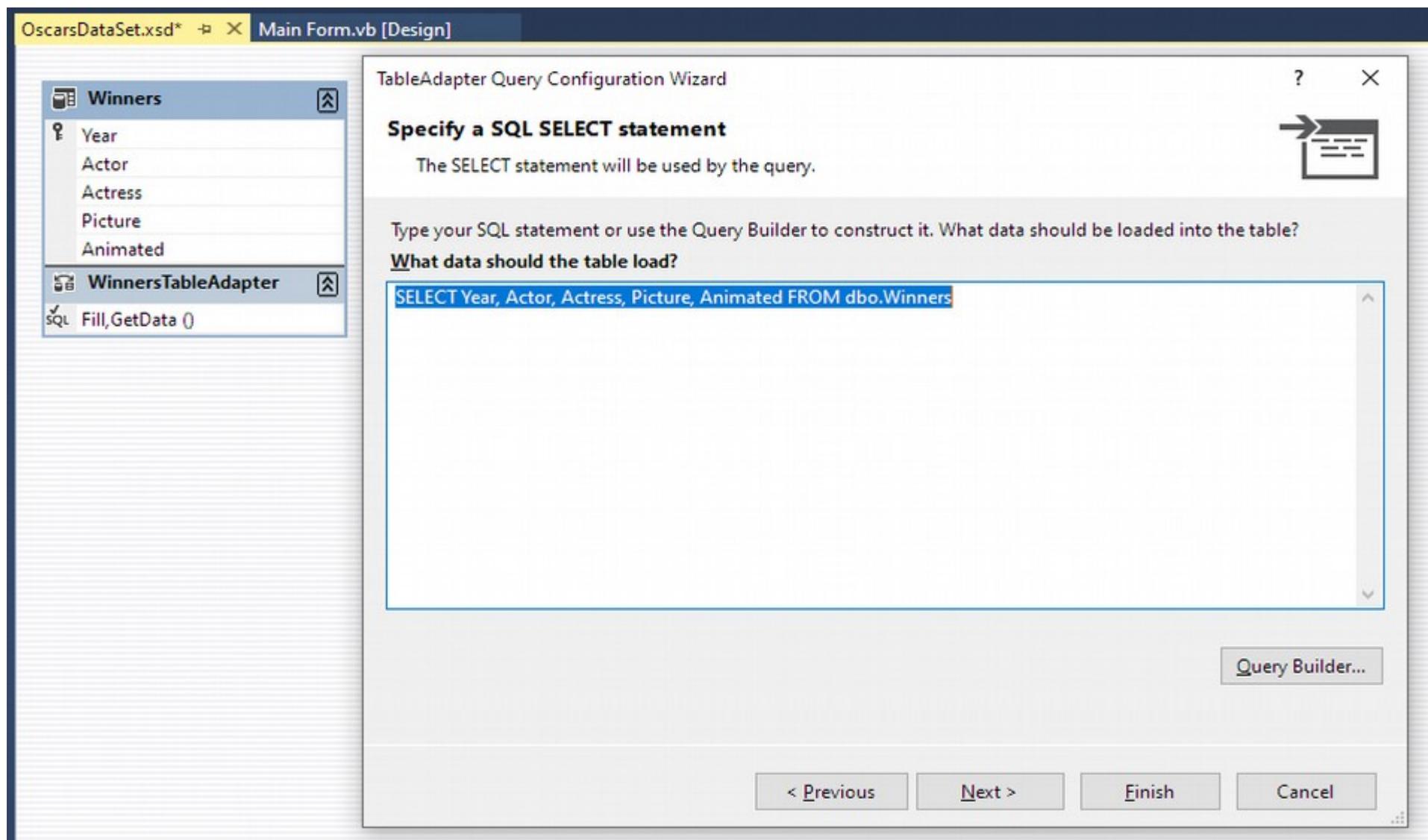
2). is automatically executed when the **frmMain\_Load** procedure invokes the **WinnersTableAdapter** object's **Fill** method

- in an order to create a query, you can:

a). type a desired **SELECT** statement straight

b). or, use the **Query Builder** dialog box to construct the statement for you

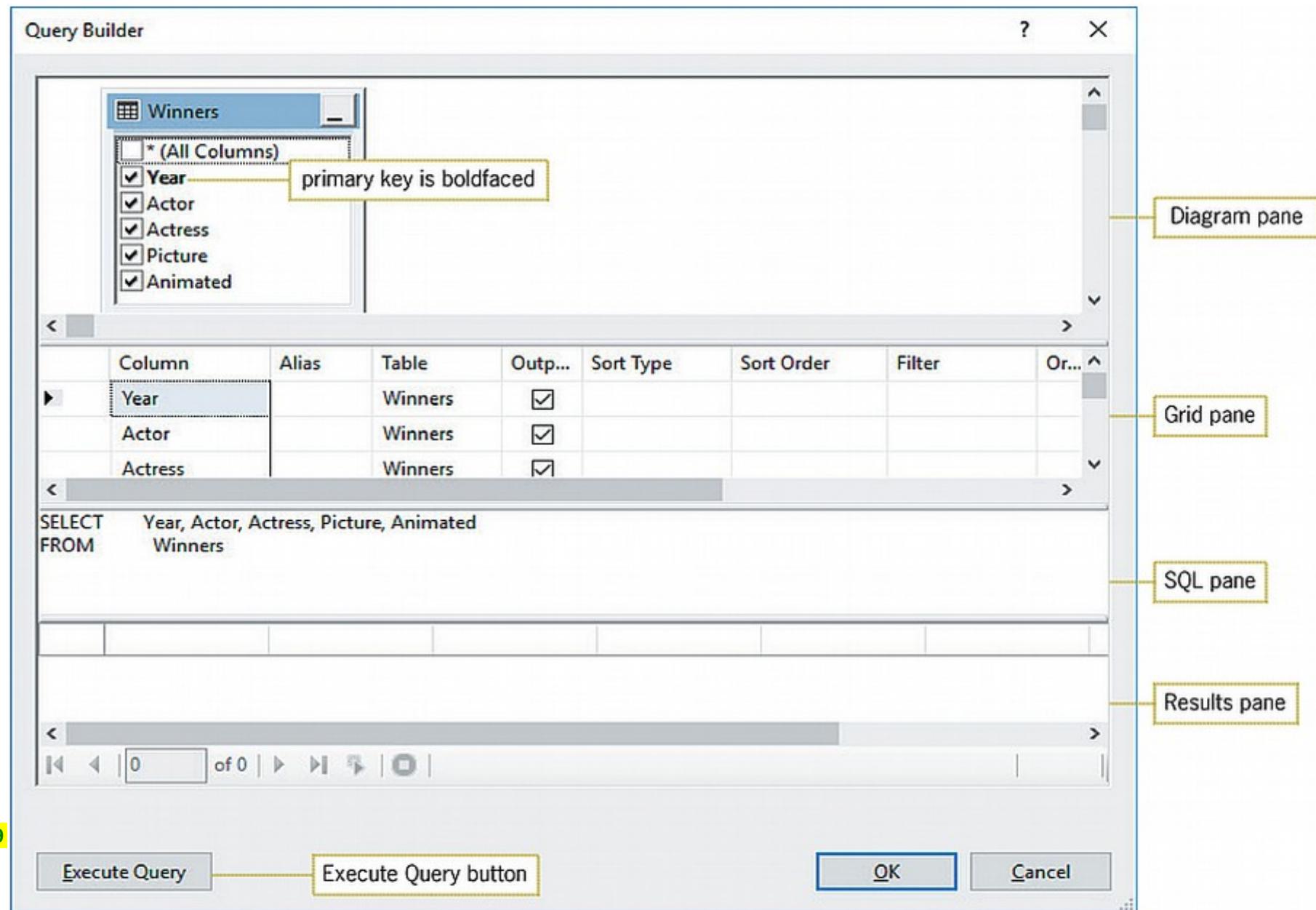
> click the button: **Query Builder...** to continue to the: **step 5**



**Figure 12-8** 3rd screen

**step 5**

- previous step started/opened the: **Query Builder** dialog box **Figure 12-9**
- in this step, you will use the **Query Builder** dialog box to create queries for the: **Figure 12-2** & **e.g.1** -> **e.g.5** from previous: **CH12\_F1** -
- the example queries will allow you to observe the way the **SELECT** statements in those examples retrieve the desired fields and records from the database
- the button **Execute Query** allows you to run/preview/test your creation without saving

**Figure 12-9**

**Figure 12-2** & e.g.1 **SELECT** Year, Actor, Actress, Picture, Animated **FROM** Winners

- the **SQL** pane in: **Figure 12-9** contains the same **SELECT** statement - the default query

- the statement selects all of the fields and records from the **Winners** table

-> click the button: **Execute Query** to run/preview/test the query **Figure 12-9.1**

The screenshot shows the Grid pane with the following data:

| SELECT Year, Actor, Actress, Picture, Animated<br>FROM Winners |      |                  |                    |                    |             |
|----------------------------------------------------------------|------|------------------|--------------------|--------------------|-------------|
|                                                                | Year | Actor            | Actress            | Picture            | Animated    |
| ▶                                                              | 2008 | Daniel Day-Lewis | Marion Cotillard   | No Country for ... | Ratatouille |
|                                                                | 2009 | Sean Penn        | Kate Winslet       | Slumdog Millio...  | WALL-E      |
|                                                                | 2010 | Jeff Bridges     | Sandra Bullock     | The Hurt Locker    | Up          |
|                                                                | 2011 | Colin Firth      | Natalie Portman    | The King's Spee... | Toy Story 3 |
|                                                                | 2012 | Jean Dujardin    | Meryl Streep       | The Artist         | Rango       |
|                                                                | 2013 | Daniel Day Lewis | Jennifer Lawren... | Argo               | Brave       |
|                                                                | 2014 | Matthew McCo...  | Cate Blanchett     | 12 Years a Slave   | Frozen      |
|                                                                | 2015 | Eddie Redmayne   | Julianne Moore     | Birdman            | Big Hero 6  |
|                                                                | 2016 | Leonardo DiCa... | Brie Larson        | Spotlight          | Inside Out  |
|                                                                | 2017 | Casey Affleck    | Emma Stone         | Moonlight          | Zootopia    |
| *                                                              | NULL | NULL             | NULL               | NULL               | NULL        |

At the bottom, there is a navigation bar with icons for back, forward, search, and other grid operations.

**Figure 12-9.1**

**Figure 12-2** & e.g.2 **SELECT** Year, Actor, Actress, Picture, Animated **FROM** Winners **WHERE** Year  $\geq$  2014

- the 2nd example selects all of the fields in the **Winners** table, but only for records for the **Year** 2014 and later

-> in the **Grid** pane / **Year** field / column **Filter**: click the blank cell and type: **>= 2014** and press Enter key

**Figure 12-10**

**Figure 12-10**

The screenshot shows the Microsoft Query Builder interface. In the top-left corner, there's a 'Winners' table with several columns: Year, Actor, Actress, Picture, and Animated. A yellow callout box points to the 'Year' column in the table, with the text 'indicates that the field is used to filter the records'. In the main pane, there's a table setup with columns: Column, Alias, Table, Output, Sort Type, Sort Order, Filter, and Or...^ (partial view). The 'Year' column is listed under 'Column' and 'Alias', with a checkmark in the 'Output' column. The 'Filter' column contains the condition ' $\geq 2014$ '. A yellow callout box points to this filter entry with the text 'used by the WHERE clause'. Below this, the SQL query is displayed: `SELECT Year, Actor, Actress, Picture, Animated FROM Winners WHERE (Year >= 2014)`. A yellow callout box points to the WHERE clause with the text 'WHERE clause'. At the bottom, there's a results pane showing a table of data for the years 2008, 2009, 2010, and 2011. A yellow callout box points to the 'Query Changed icon' (an exclamation mark icon) in the top-left corner of the results pane. Another yellow callout box points to a message box titled 'Query Changed' with the text 'Query Changed message'. At the bottom right of the results pane are 'OK' and 'Cancel' buttons.

**Figure 12-10**

<- the funnel symbol that appears in the **Diagram** pane indicates that the **Year** field is used to filter the records

<- the **Filter** column entry tells the **Query Builder** to include the **WHERE (Year  $\geq 2014$ )** clause in the **SELECT** statement

<- for clarity, the **Query Builder** places the **WHERE** clause's condition in parentheses - however, the parentheses are not required for the **SELECT** statement

<- notice the icon that appear in the **Results** pane

<- notice the **Query Changed** message and icon that appear  
<- the message and icon alert you that the information displayed in the **Results** pane is not from the current query

**Execute Query**

-> click the button: **Execute Query** to run/preview/test the current query

-> verify the **Results** pane contains only the records for the years 2014 through 2017

**Figure 12-10.1**

**Figure 12-10.1**

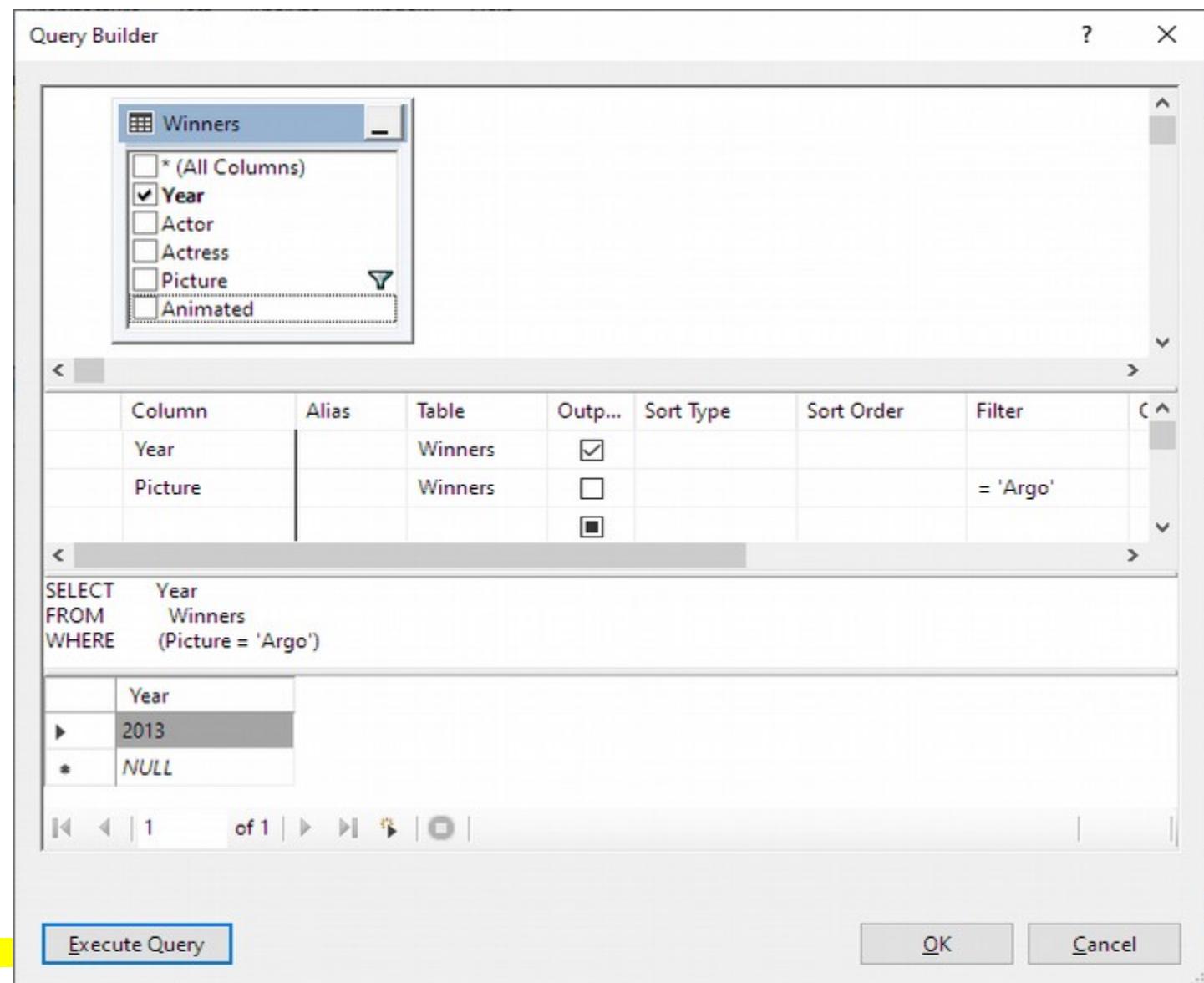
**Figure 12-10.1**

| SELECT Year, Actor, Actress, Picture, Animated<br>FROM Winners<br>WHERE (Year $\geq 2014$ ) |      |                     |                |                  |            |
|---------------------------------------------------------------------------------------------|------|---------------------|----------------|------------------|------------|
|                                                                                             | Year | Actor               | Actress        | Picture          | Animated   |
| ▶                                                                                           | 2014 | Matthew McConaughey | Cate Blanchett | 12 Years a Slave | Frozen     |
|                                                                                             | 2015 | Eddie Redmayne      | Julianne Moore | Birdman          | Big Hero 6 |
|                                                                                             | 2016 | Leonardo DiCaprio   | Brie Larson    | Spotlight        | Inside Out |
|                                                                                             | 2017 | Casey Affleck       | Emma Stone     | Moonlight        | Zootopia   |

**Figure 12-2** & **e.g.3** **SELECT Year FROM Winners WHERE Picture = 'Argo'** / can use 'argo' - text comparisons are not case sensitive

- the 3rd example selects only the **Year** field for the records whose **Picture** field includes the word Argo/argo
- > in the **Grid pane / Year field / column Filter**: delete the previous entry **>= 2014**
- > in the **Grid pane / Picture field / column Filter**: click the blank cell and type: **Argo** and press Enter key
  - the **Query Builder** changes the entry in the **Filter** column to 'Argo'
  - it also enters the **WHERE (Picture = 'Argo')** clause in the **SELECT** statement in the **SQL pane**
- > because we want only the **Year** field - in the **Diagram** pane deselect the check boxes: **Actor**, **Actress**, **Picture**, **Animated**
  - the **Query Builder** changes the first line in the **SELECT** statement to **SELECT Year** in the **SQL pane**
- > click the button: **Execute Query** to run/preview/test the current query

**Figure 12-11**

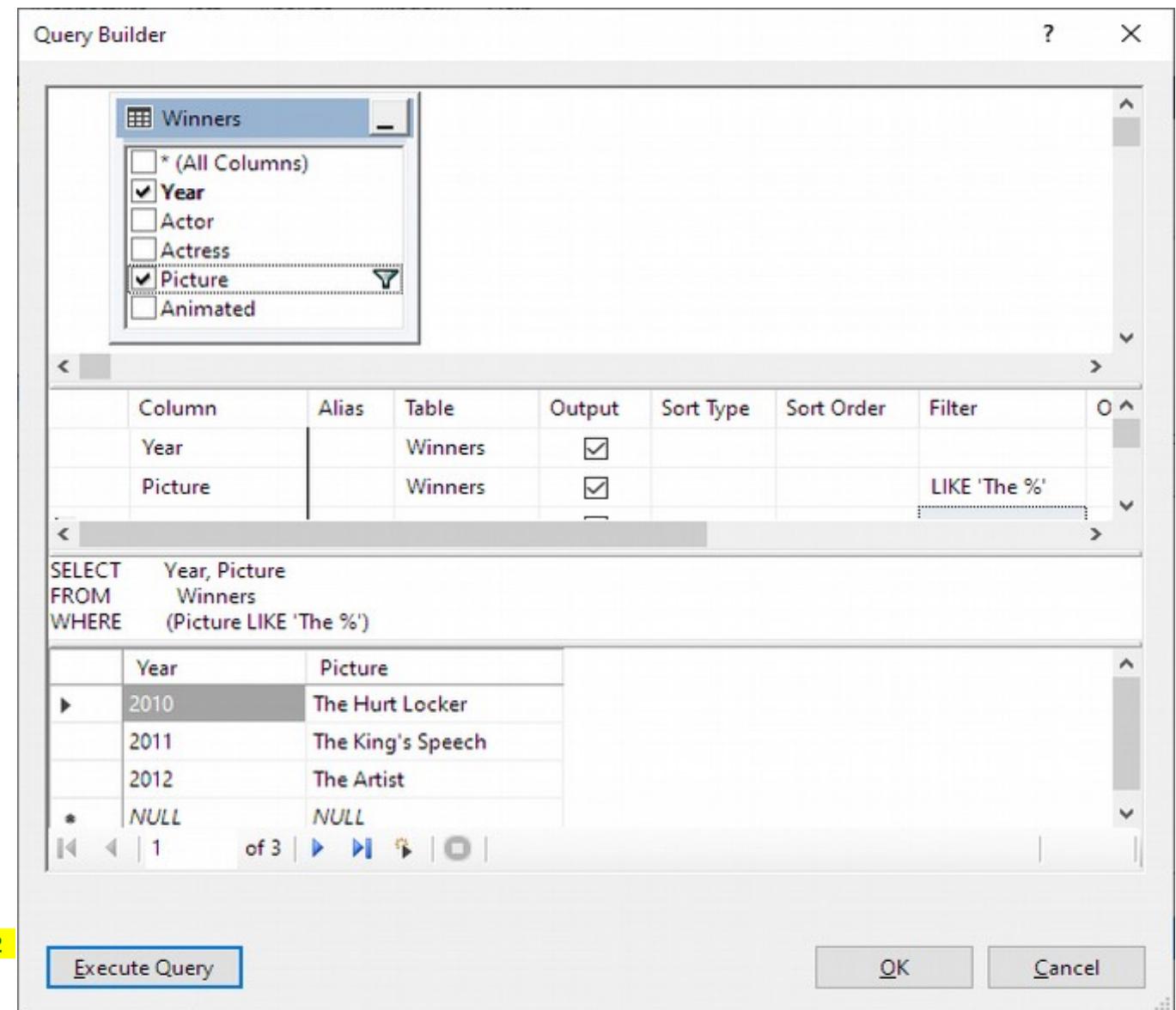


**Figure 12-11**

**Figure 12-2** & **e.g.4** **SELECT Year, Picture FROM Winners WHERE Picture LIKE 'The %'**

- the 4th example selects the **Year** and **Picture** fields for all records whose **Picture** field begins with the word "The" followed by a space and 0 or more characters
- > in the **Diagram** pane, select the check box **Picture**; the check box **Year** is already selected
- > in the **Grid** pane / **Picture** field / column **Filter**: replace previous entry: = 'Argo' with a: **LIKE 'The %'** and press Enter key
  - be sure to include a space character before the % wildcard
- > click the button: **Execute Query** to run/preview/test the current query

**Figure 12-12**



**Figure 12-12**

**Figure 12-2** & **e.g.5** **SELECT Year, Animated FROM Winners WHERE Year = 2010 OR Year = 2015 ORDER BY Year DESC**

- the 5th example selects the **Year** and **Animated** fields from the records for the **Years** 2010 and 2015 and then sort the results in descending order by the **Year** field

-> in the **Diagram** pane, deselect previously selected check box **Picture**, select **Animated**; the check box **Year** is already selected

-> in the **Grid** pane / **Picture** field / column **Filter**: delete previous entry: **LIKE 'The %'**

-> in the **Grid** pane / **Year** field / column **Sort Type**: click the blank cell and then click the list arrow and choose **Descending**

- the word **Descending** appears as the **Year** field's **Sort Type**, and the number **1** appears as its **Sort Order**

- the number **1** indicates that the **Year** field is the primary field in the sort

-> notice that the **Query Builder** adds the **ORDER BY Year DESC** clause to the **SELECT** statement in the **SQL** pane

-> in the **Grid** pane / **Year** field / column **Filter**: type: **2010 or 2015** and press Enter key

-> notice the **Query Builder** changes the entry to: **= 2010 OR = 2015**

-> notice that it also adds the: **WHERE (Year = 2010 OR Year = 2015)** clause to the **SELECT** statement in the **SQL** pane

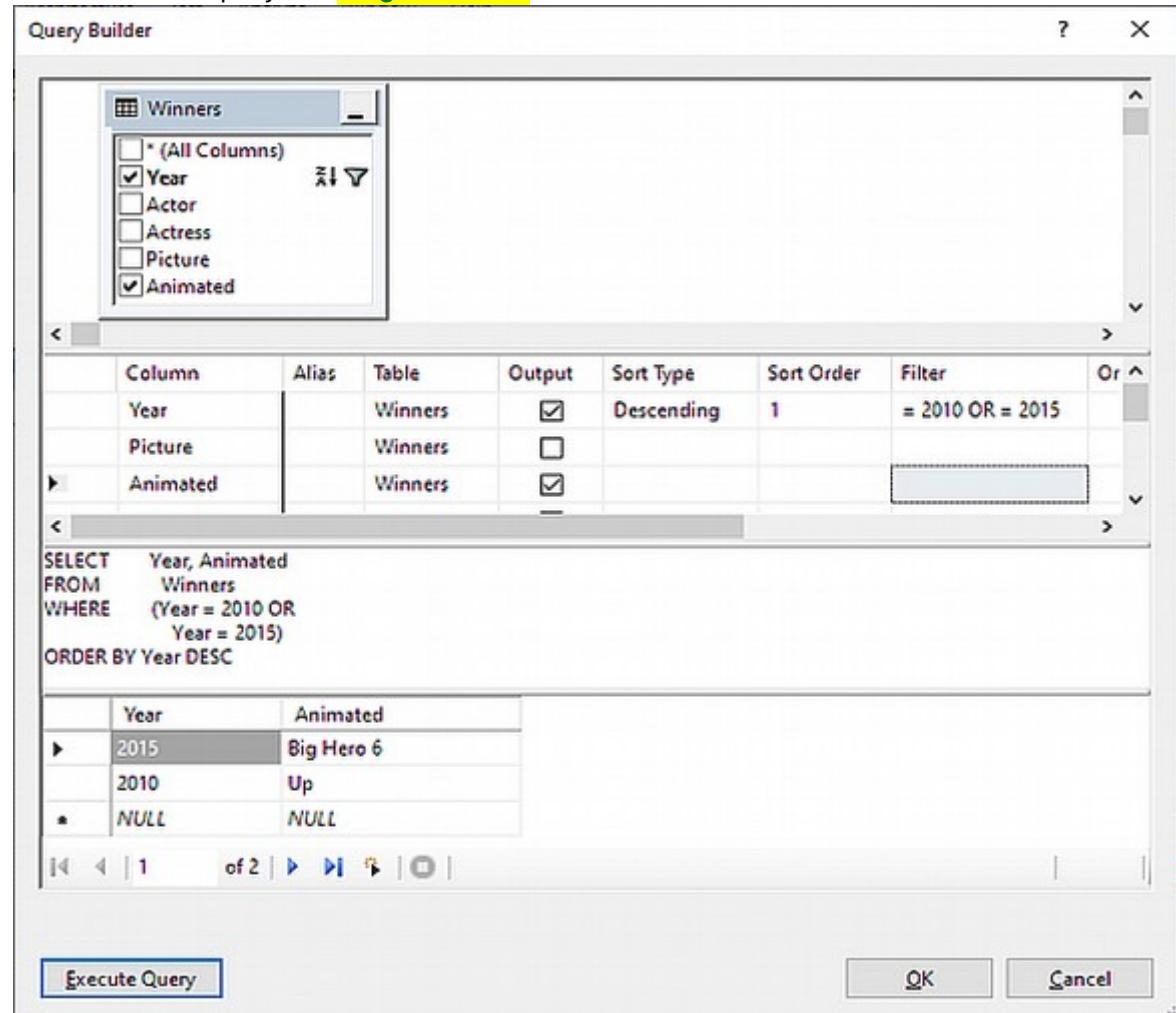
-> click the button: **Execute Query** to run/preview/test the current query

**Figure 12-13**

- that is all for now, you will cancel all of the changes, and close the solution

-> in the **Query Builder** dialog box click the button **Cancel** and then in the **TableAdapter Query Configuration Wizard** click the button **Cancel**

-> save and close the solution



**Figure 12-13**

## CH12\_F2.1 - You Do It 1: practice creating Query using Query Builder dialog box - 3 examples from Mini-Quiz 12-1 with 02.You Do It 1 Solution

1. open the: ...VB2017\Chap12\Exercise02.You Do It 1 Solution\You Do It 1 Solution.sln

2. use the application to test the 3 SELECT statements from **Mini-Quiz 12-1**:

**Mini-Quiz 12-1** Using the **dbo.Winners** table from **Figure 12-2**, write a **SELECT** statement that:

Q1. selects only the **Actress** field for the year **2010**.

Q2. selects only the **Animated** field for records whose **Animated** field begins with the letter **R**. Sort the records in **ascending** order by the **Animated** field.

Q3. selects the **Actor** and **Actress** fields for the years **2008** through **2010**. Sort the records in **descending** order by the **Year** field.

A1. `SELECT Actress FROM Winners WHERE Year = 2010`

A2. `SELECT Animated FROM Winners WHERE Animated LIKE 'R%' ORDER BY Animated`

A3. `SELECT Actor, Actress FROM Winners WHERE Year >= 2008 AND Year <= 2010 ORDER BY Year DESC`

**step 1** - open the **DataSet Designer** window:

-> **Solution Explorer** window / **OscarsDataSet.xsd** **DataSet** / Rclick: **Open**

or -> **Data Sources** window / **OscarsDataSet** / Rclick: **Edit DataSet with Designer**

**step 2** - start/open the **TableAdapter Query Configuration Wizard**:

-> **OscarsDataSet.xsd** **Designer** window / line **WinnersTableAdapter** / Rclick: **Add / Query...**

**step 3** - get to the **Query Builder** dialog box:

-> Use **SQL statements** -> **SELECT** which returns rows -> button **Query Builder...** **Figure 12-14**

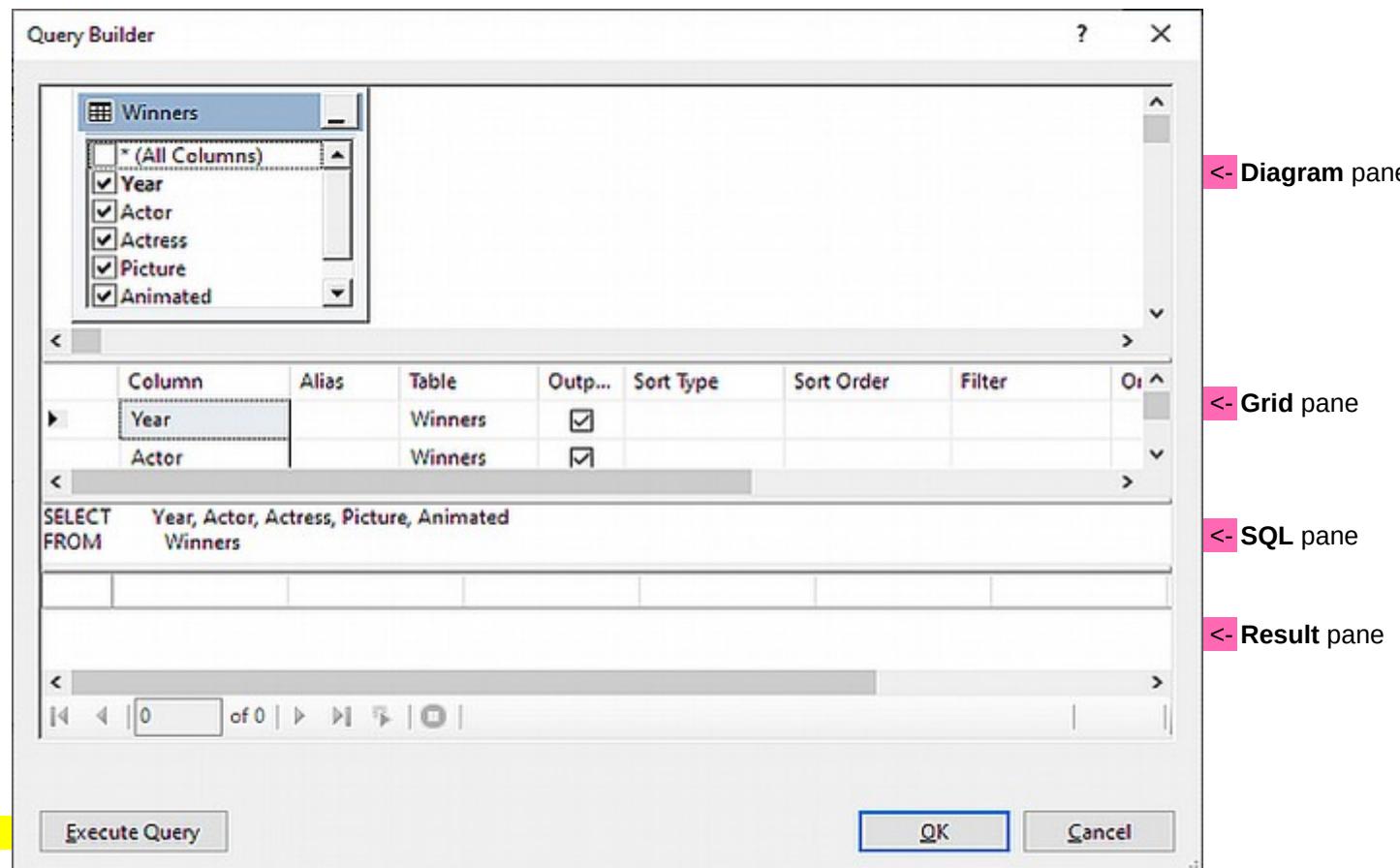


Figure 12-14

Q1. Write a **SELECT** statement that selects only the Actress field for the Year 2010:

A1. **SELECT** Actress **FROM** Winners **WHERE** Year = 2010

- > **Diagram** pane => select only check boxes for the columns: Year, Actress
- > **Grid** pane: column Year / Filter => **2010**
- > **Grid** pane: column Year / Output => **uncheck the check box**
- > button **Execute Query** to run/preview/test

| Column  | Alias | Table   | Output                              | Sort Type | Sort Order | Filter | Or... |
|---------|-------|---------|-------------------------------------|-----------|------------|--------|-------|
| Actress |       | Winners | <input checked="" type="checkbox"/> |           |            |        |       |
| Year    |       | Winners | <input type="checkbox"/>            |           |            | = 2010 |       |

|        |               |
|--------|---------------|
| SELECT | Actress       |
| FROM   | Winners       |
| WHERE  | (Year = 2010) |

|   |                |
|---|----------------|
|   | Actress        |
| ▶ | Sandra Bullock |
| * | NULL           |

Q2. Write a **SELECT** statement that selects only the Animated field for records whose Animated field begins with the letter **R**, and sort the records in **ascending** order by the Animated field:

A2. **SELECT** Animated **FROM** Winners **WHERE** Animated **LIKE** 'R%' **ORDER BY** Animated

- > **Diagram** pane => select only check box for the column: Animated
- > **Grid** pane: column Animated / Sort Type => **Ascending**
- > **Grid** pane: column Animated / Sort Order => **1**
- > **Grid** pane: column Animated / Filter => **LIKE 'R%'**
- > button **Execute Query** to run/preview/test

| Column   | Alias | Table   | Output                              | Sort Type | Sort Order | Filter    | Or... |
|----------|-------|---------|-------------------------------------|-----------|------------|-----------|-------|
| Animated |       | Winners | <input checked="" type="checkbox"/> | Ascending | 1          | LIKE 'R%' |       |

|        |                       |
|--------|-----------------------|
| SELECT | Animated              |
| FROM   | Winners               |
| WHERE  | (Animated LIKE 'R%)') |

|   |             |
|---|-------------|
|   | Animated    |
| ▶ | Rango       |
| * | Ratatouille |

Q3. Write a **SELECT** statement that selects the Actor and Actress fields for the Years 2008 through 2010, and sort the records in descending order by the Year field:

A3. **SELECT** Actor, Actress **FROM** Winners **WHERE** Year >= 2008 **AND** Year <= 2010 **ORDER BY** Year **DESC**

- > **Diagram** pane => select only check boxes for the columns: Year, Actor, Actress
- > **Grid** pane: column Year / Sort Type => **Descending**
- > **Grid** pane: column Year / Sort Order => **1**
- > **Grid** pane: column Year / Filter => **>= 2008 AND <= 2010**
- > **Grid** pane: column Year / Output => **uncheck the check box**
- > button **Execute Query** to run/preview/test

| Column  | Alias | Table   | Output                              | Sort Type  | Sort Order | Filter              | Or... |
|---------|-------|---------|-------------------------------------|------------|------------|---------------------|-------|
| Year    |       | Winners | <input type="checkbox"/>            | Descending | 1          | >= 2008 AND <= 2010 |       |
| Actor   |       | Winners | <input checked="" type="checkbox"/> |            |            |                     |       |
| Actress |       | Winners | <input checked="" type="checkbox"/> |            |            |                     |       |

|        |                                 |
|--------|---------------------------------|
| SELECT | Actor, Actress                  |
| FROM   | Winners                         |
| WHERE  | (Year >= 2008 AND Year <= 2010) |

|   |                  |                  |
|---|------------------|------------------|
|   | Actor            | Actress          |
| ▶ | Jeff Bridges     | Sandra Bullock   |
|   | Sean Penn        | Kate Winslet     |
|   | Daniel Day-Lewis | Marion Cotillard |

## CH12\_F3 - Parameter Query & Parameter Marker @ - basic info & e.g.

- the queries created in the previous section retrieve only records matching specific criteria, such as Picture = 'Argo', or Year >= 2014
- most times, however, you will not know ahead of time the value to include in the criteria
  - e.g. The user may want to retrieve the Argo record today, but retrieve the Slumdog Millionaire record tomorrow
- when you do not know the specific value to include in the criteria, you use a **parameter query**

-> a **parameter query**: = a query that contains a **parameter marker** in place of a criteria's value

-> a **parameter marker**: = typically used in SQL is the (at) symbol - @ - followed by the name of the field you are querying

- if the WHERE clause contains more than 1 parameter marker for the same field, you append a unique number after the field name
  - e.g. WHERE Year >= @Year1 AND <= @Year2

**Figure 12-15** - a parameter query with a parameter marker @, with database: **Oscars.mdf**, table: **dbo.Winners** e.g.1 -> e.g.3

|   | Name     | Data Type   | Allow Nulls              | Default |
|---|----------|-------------|--------------------------|---------|
| # | Year     | int         | <input type="checkbox"/> |         |
|   | Actor    | varchar(50) | <input type="checkbox"/> |         |
|   | Actress  | varchar(50) | <input type="checkbox"/> |         |
|   | Picture  | varchar(50) | <input type="checkbox"/> |         |
|   | Animated | varchar(50) | <input type="checkbox"/> |         |

| Year | Actor               | Actress           | Picture                | Animated    |
|------|---------------------|-------------------|------------------------|-------------|
| 2008 | Daniel Day-Lewis    | Marion Cotillard  | No Country for Old Men | Ratatouille |
| 2009 | Sean Penn           | Kate Winslet      | Slumdog Millionaire    | WALL-E      |
| 2010 | Jeff Bridges        | Sandra Bullock    | The Hurt Locker        | Up          |
| 2011 | Colin Firth         | Natalie Portman   | The King's Speech      | Toy Story 3 |
| 2012 | Jean Dujardin       | Meryl Streep      | The Artist             | Rango       |
| 2013 | Daniel Day Lewis    | Jennifer Lawrence | Argo                   | Brave       |
| 2014 | Matthew McConaughey | Cate Blanchett    | 12 Years a Slave       | Frozen      |
| 2015 | Eddie Redmayne      | Julianne Moore    | Birdman                | Big Hero 6  |
| 2016 | Leonardo DiCaprio   | Brie Larson       | Spotlight              | Inside Out  |
| 2017 | Casey Affleck       | Emma Stone        | Moonlight              | Zootopia    |

e.g.1 SELECT Year, Actor, Actress, Picture, Animated FROM Winners WHERE Picture = @Picture

<- selects all of the fields for the records whose Picture field contains the value represented by the parameter marker: @Picture

e.g.2 SELECT Year, Actor, Actress, Picture, Animated FROM Winners WHERE Year >= @Year

<- selects all of the fields for the records whose Year field contains a value that is greater than or equal to the value represented by the parameter marker: @Year

e.g.3 SELECT Year, Actor, Actress, Picture, Animated FROM Winners WHERE Year >= @Year1 AND Year <= @Year2

<- selects all of the fields for the records whose Year field contains a value that is:

- greater than or equal to the value represented by the 1st parameter marker: @Year1
- but less than or equal to the value represented by the 2nd parameter marker: @Year2

**Mini-Quiz 12-2** Using the **dbo.Winners** table from **Figure 12-15**, write a parameter query that:

- Q1. selects only an Actor's Picture field, where the name of the Actor is provided by the user.
- Q2. selects only an Animated picture's Year field, where the name of the Animated picture is provided by the user.
- Q3. selects the Actor and Actress fields for the Year provided by the user.

- |     |                                                       |
|-----|-------------------------------------------------------|
| A1. | SELECT Picture FROM Winners WHERE Actor = @Actor      |
| A2. | SELECT Year FROM Winners WHERE Animated = @Animated   |
| A3. | SELECT Actor, Actress FROM Winners WHERE Year = @Year |

## CH12\_F3.1 - how to use a Parameter Query & parameter marker @ - using 3x e.g. from previous lesson, with solution 03.Oscars Solution-Parameter Queries

- in the next set of steps, you will use the Oscar Winners application to test the **SELECT** statements from previous lesson

**Figure 12-15** & **e.g.1** -> **e.g.3**

- to create **parameter queries** for the **Figure 12-15** & **e.g.1** -> **e.g.3** example with solution: **03.Oscars Solution-Parameter Queries**

-> open the: ...VB2017\Chap12\Exercise\03.Oscars Solution-Parameter Queries\Oscars Solution.sln

-> open the **Designer** and **Solution Explorer** windows

- as you can see, The **Oscar Winners** application is already connected to the **Oscars.mdf** database, and the **OscarsDataSet.xsd** is already created

**step 1** - open the **DataSet Designer** window:

-> **Solution Explorer** window / **OscarsDataSet.xsd** **DataSet** / Rclick: **Open**

or -> **Data Sources** window / **OscarsDataSet** / Rclick: **Edit DataSet with Designer**

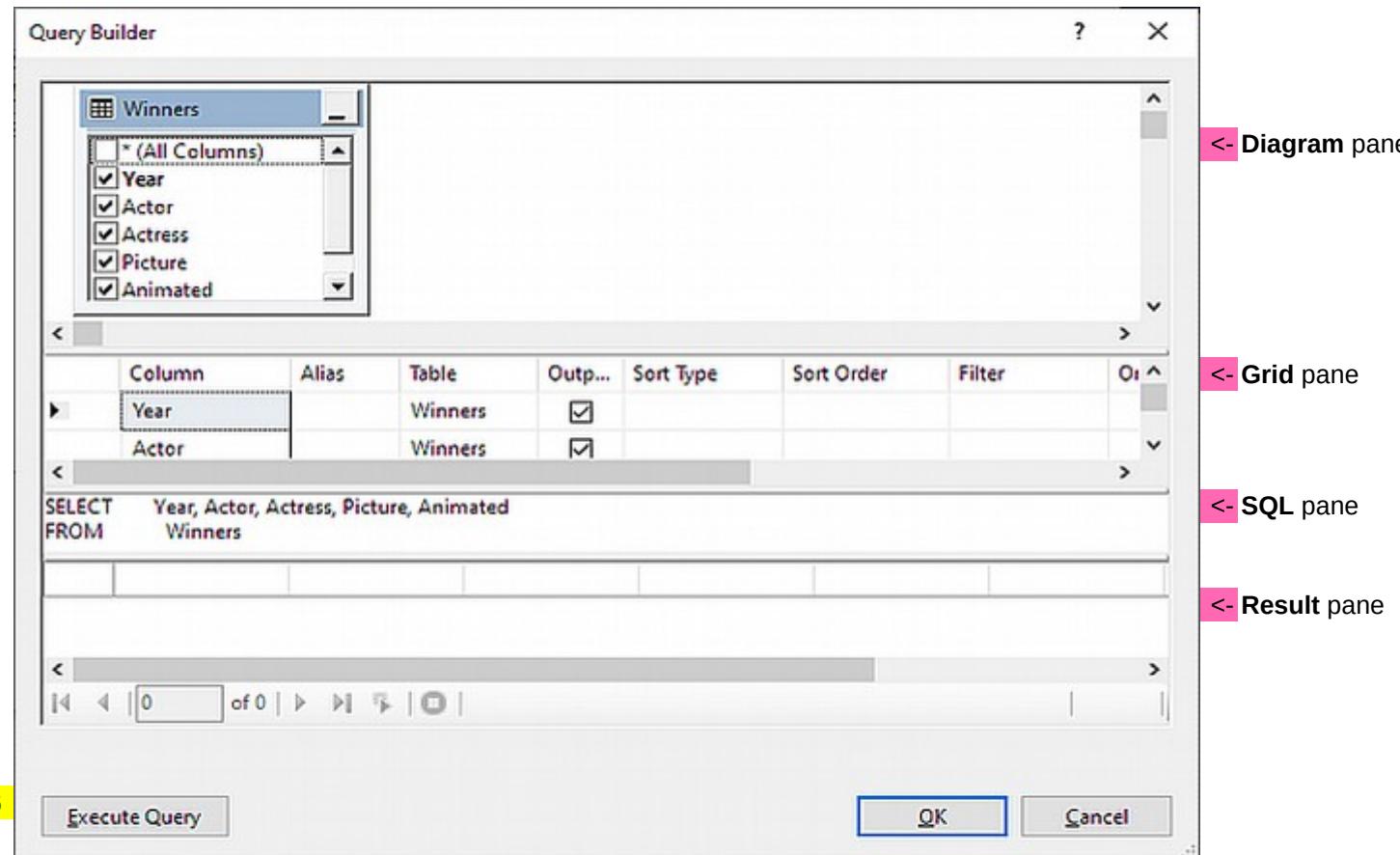
**step 2** - start/open the **TableAdapter Query Configuration Wizard**:

-> **OscarsDataSet.xsd** **Designer** window / line **WinnersTableAdapter** / Rclick: **Add / Query...**

**step 3** - get to the **Query Builder** dialog box:

-> **Use SQL statements** -> **SELECT** which returns rows -> button **Query Builder...**

**Figure 12-16**



**Figure 12-16**

**Figure 12-15** & e.g.1 | SELECT Year, Actor, Actress, Picture, Animated FROM Winners WHERE Picture = @Picture

- you will create a **parameter query** that selects a record based on the value in its Picture field

e.g.1.1 | you will use a Picture field's parameter: **Argo**

e.g.1.2 | you will use a Picture field's parameter: **Slumdog Millionaire**

-> in **Grid pane / column Picture / Filter** => type: **@Picture**

<- notice what **Query Builder** did: 1). in **Grid pane**: changed the entry in the **Filter** column to: **= @Picture** **Figure 12-17**  
2). in **SQL pane**: added the: **WHERE (Picture = @Picture)** clause to the **SELECT** statement

<- notice the **Result pane** includes an exclamation mark **Figure 12-17**

The screenshot shows the Microsoft Query Builder interface. On the left, there is a vertical toolbar with icons for New, Open, Save, and others. The main area has three panes: a Grid pane at the top, a SQL pane in the middle, and a Result pane at the bottom. In the Grid pane, there are two rows under the 'Column' header: 'Picture' and 'Animated'. The 'Table' column for both is 'Winners'. The 'Sort Type' column has checkboxes checked for both. The 'Filter' column contains the expression '= @Picture'. In the SQL pane, the query is displayed:

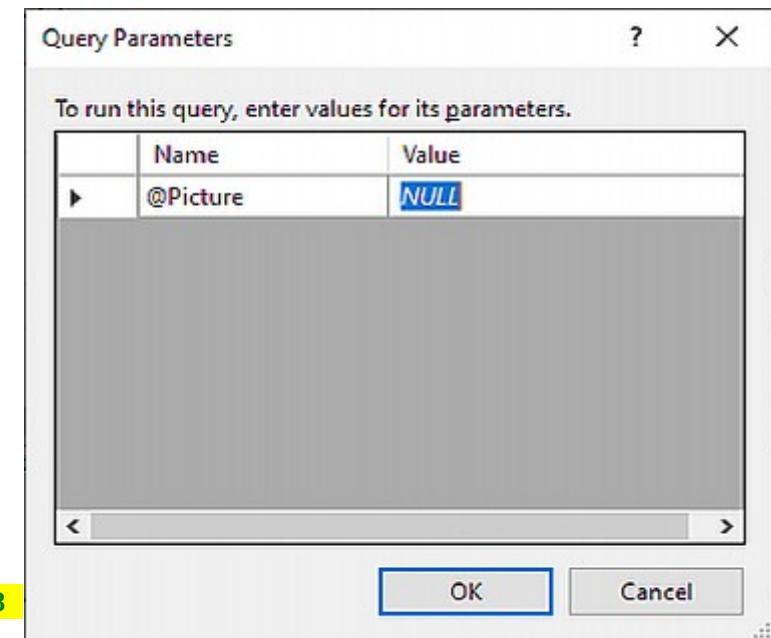
```
SELECT Year, Actor, Actress, Picture, Animated  
FROM Winners  
WHERE (Picture = @Picture)
```

In the Result pane, there is a yellow exclamation mark icon.

**Figure 12-17**

-> click the button **Execute Query** to run/preview/test

<- notice the **Query Parameters** dialog box opens: **Figure 12-18**



**Figure 12-18**

e.g.1.1 you will use a Picture field's parameter: Argo

-> Query Parameters window / column Value => Argo, and click the OK button to close the Query Parameters dialog box

<- notice the Results pane now includes the Argo record: **Figure 12-19**

**Figure 12-19**

|                                                                                              | Column   | Alias            | Table              | Outp...                             | Sort Type | Sort Order | Filter     | Or... | Or... | ^ |
|----------------------------------------------------------------------------------------------|----------|------------------|--------------------|-------------------------------------|-----------|------------|------------|-------|-------|---|
|                                                                                              | Picture  |                  | Winners            | <input checked="" type="checkbox"/> |           |            | = @Picture |       |       |   |
| ▶                                                                                            | Animated |                  | Winners            | <input checked="" type="checkbox"/> |           |            |            |       |       |   |
|                                                                                              |          |                  |                    |                                     |           |            |            |       |       |   |
| SELECT Year, Actor, Actress, Picture, Animated<br>FROM Winners<br>WHERE (Picture = @Picture) |          |                  |                    |                                     |           |            |            |       |       |   |
|                                                                                              | Year     | Actor            | Actress            | Picture                             | Animated  |            |            |       |       | ^ |
| ▶                                                                                            | 2013     | Daniel-Day Lewis | Jennifer Lawren... | Argo                                | Brave     |            |            |       |       |   |
| *                                                                                            | NULL     | NULL             | NULL               | NULL                                | NULL      |            |            |       |       |   |

e.g.1.2 you will use a Picture field's parameter: Slumdog Millionaire

-> click the button Execute Query to run/preview/test

<- notice the **Query Parameters** dialog box opens: **Figure 12-18**

-> Query Parameters window / column Value => Slumdog Millionaire, and click the OK button to close the dialog box

<- notice the Results pane now includes the Slumdog Millionaire record: **Figure 12-20**

**Figure 12-20**

|                                                                                              | Column   | Alias     | Table        | Outp...                             | Sort Type | Sort Order | Filter     | Or... | Or... | ^ |
|----------------------------------------------------------------------------------------------|----------|-----------|--------------|-------------------------------------|-----------|------------|------------|-------|-------|---|
|                                                                                              | Picture  |           | Winners      | <input checked="" type="checkbox"/> |           |            | = @Picture |       |       |   |
| ▶                                                                                            | Animated |           | Winners      | <input checked="" type="checkbox"/> |           |            |            |       |       |   |
|                                                                                              |          |           |              |                                     |           |            |            |       |       |   |
| SELECT Year, Actor, Actress, Picture, Animated<br>FROM Winners<br>WHERE (Picture = @Picture) |          |           |              |                                     |           |            |            |       |       |   |
|                                                                                              | Year     | Actor     | Actress      | Picture                             | Animated  |            |            |       |       | ^ |
| ▶                                                                                            | 2009     | Sean Penn | Kate Winslet | Slumdog Millionaire                 | WALL-E    |            |            |       |       |   |
| *                                                                                            | NULL     | NULL      | NULL         | NULL                                | NULL      |            |            |       |       |   |

**Figure 12-15** & **e.g.2** `SELECT Year, Actor, Actress, Picture, Animated FROM Winners WHERE Year >= @Year`

- you will create a **parameter query** that selects records whose Year field contains a value that is greater than or equal to the value provided by the user

-> in Grid pane / column Picture / Filter => delete the: = @Picture

-> in Grid pane / column Year / Filter => type the: >= @Year

<- notice what **Query Builder** did: - in **SQL** pane: added the: **WHERE (Year >= @Year)** clause to the **SELECT** statement

<- notice the **Result** pane still includes the previous result: **Slumdog Millionaire** - but also an exclamation mark and **Query Changed**

The screenshot shows the Microsoft Query Builder interface. In the top pane, there is a grid with columns: Column, Alias, Table, Outp..., Sort Type, Sort Order, Filter, and Or... (with a dropdown menu). Two rows are present: one for the 'Year' column with 'Winners' as the table and a checked 'Sort Type' box; another for the 'Actor' column with 'Winners' as the table and a checked 'Sort Type' box. A filter row below them has a dropdown menu set to '>= @Year'. In the bottom pane, the SQL query is:

```
SELECT Year, Actor, Actress, Picture, Animated
FROM Winners
WHERE (Year >= @Year)
```

The results pane shows a table with columns: Year, Actor, Actress, Picture, and Animated. It contains three rows:

| ! | Year | Actor     | Actress      | Picture             | Animated |
|---|------|-----------|--------------|---------------------|----------|
| ▶ | 2009 | Sean Penn | Kate Winslet | Slumdog Millionaire | WALL-E   |
| * | NULL | NULL      | NULL         | NULL                | NULL     |

At the bottom of the results pane, there are navigation buttons (first, previous, next, last) and a status bar with an exclamation mark icon and the text 'Query Changed'.

**Figure 12-21**

-> click the button **Execute Query** to run/preview/test

<- notice the **Query Parameters** dialog box opens

-> **Query Parameters** window / column **Value** => type: **2016**, and click the **OK** button to close the dialog box

<- notice the **Results** pane now includes the records for years 2016 and 2017 **Figure 12-22**

The screenshot shows the Microsoft Query Builder interface after executing the query. The grid in the top pane remains the same. In the bottom pane, the SQL query is the same as in Figure 12-21. The results pane now shows two additional rows for the years 2016 and 2017:

| ! | Year | Actor             | Actress      | Picture             | Animated   |
|---|------|-------------------|--------------|---------------------|------------|
| ▶ | 2009 | Sean Penn         | Kate Winslet | Slumdog Millionaire | WALL-E     |
| ▶ | 2016 | Leonardo DiCaprio | Brie Larson  | Spotlight           | Inside Out |
| ▶ | 2017 | Casey Affleck     | Emma Stone   | Moonlight           | Zootopia   |
| * | NULL | NULL              | NULL         | NULL                | NULL       |

**Figure 12-22**

**Figure 12-15** & e.g.3 SELECT Year, Actor, Actress, Picture, Animated FROM Winners WHERE Year >= @Year1 AND Year <= @Year2

- you will create a **parameter query** that selects records whose Year field contains a value that is:

- greater than or equal to the value provided by the user, represented by the **1st parameter marker**: @Year1
- but less than or equal to the value provided by the user, represented by the **2nd parameter marker**: @Year2

-> in **Grid pane / column Year / Filter** => change the previous value to: >= @Year1 and <= @Year2

<- notice what **Query Builder** did: 1). in **Grid pane**: changed the entry in the Filter column to: >= @Year1 AND <= @Year2  
2). in **SQL pane**: added the: WHERE (Year >= @Year1 AND Year <= @Year2) clause

<- notice the **Result pane** still includes the previous result - but also an exclamation mark and **Query Changed**

The screenshot shows the Microsoft Query Builder interface. On the left, there's a legend with icons for 'Column', 'Alias', 'Table', 'Output', 'Sort Type', 'Sort Order', and 'Filter'. Below this is a table structure with columns: Year, Actor, and Table (labeled 'Winners'). The 'Filter' column for the 'Year' column contains the expression '>= @Year1 AND <= @Year2'. In the center, the SQL pane displays the query:

```
SELECT Year, Actor, Actress, Picture, Animated
FROM Winners
WHERE (Year >= @Year1 AND Year <= @Year2)
```

Below the SQL pane is the Result pane, which shows two rows of data:

| ! | Year | Actor               | Actress           | Picture          | Animated |
|---|------|---------------------|-------------------|------------------|----------|
| ▶ | 2013 | Daniel Day Lewis    | Jennifer Lawrence | Argo             | Brave    |
| ◀ | 2014 | Matthew McConaughey | Cate Blanchett    | 12 Years a Slave | Frozen   |

At the bottom right of the Result pane, there's a yellow exclamation mark icon followed by the text 'Query Changed'.

Figure 12-23

-> click the button **Execute Query** to run/preview/test the query

<- notice the **Query Parameters** dialog box opens, with a 1st and the 2nd parameter markers: **Figure 12-24**

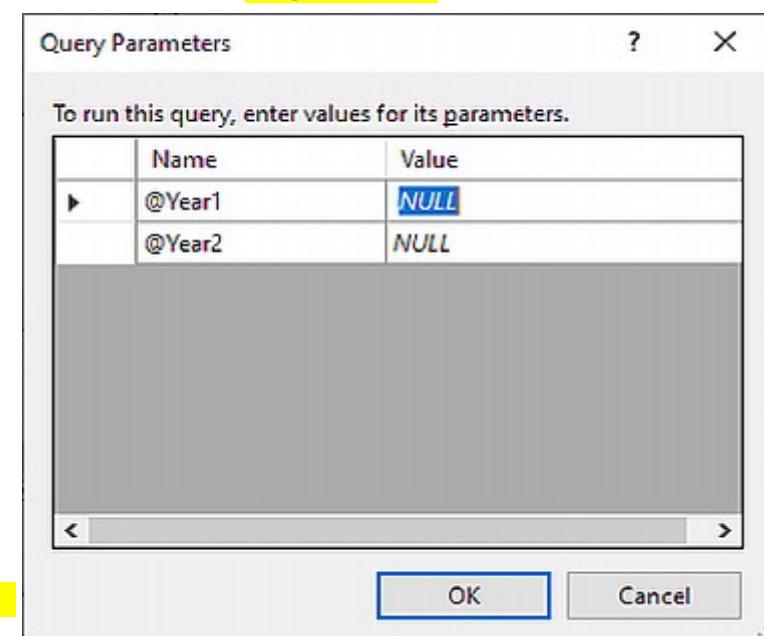


Figure 12-24

- > Query Parameters window / line @Year1 / column Value => type: 2010
- > Query Parameters window / line @Year2 / column Value => type: 2012, and click the OK button to close the dialog box
- <- notice the **Results** pane now includes the records for years 2010, 2011, and 2012 **Figure 12-25**

**Query Builder (Top):**

| Column | Alias | Table   | Output                              | Sort Type | Sort Order | Filter                                     | Or... |
|--------|-------|---------|-------------------------------------|-----------|------------|--------------------------------------------|-------|
| Year   |       | Winners | <input checked="" type="checkbox"/> |           |            | <code>&gt;= @Year1 AND &lt;= @Year2</code> |       |
| Actor  |       | Winners | <input type="checkbox"/>            |           |            |                                            |       |

**Results (Bottom):**

| Year | Actor         | Actress         | Picture           | Animated    |
|------|---------------|-----------------|-------------------|-------------|
| 2010 | Jeff Bridges  | Sandra Bullock  | The Hurt Locker   | Up          |
| 2011 | Colin Firth   | Natalie Portman | The King's Speech | Toy Story 3 |
| 2012 | Jean Dujardin | Meryl Streep    | The Artist        | Rango       |

**Figure 12-25**

- that is all for now, you will cancel all of the changes, and close the solution

- > in the **Query Builder** dialog box click the button **Cancel** and then in the **TableAdapter Query Configuration Wizard** click the button **Cancel**
- > save and close the solution

## CH12\_F3.2 - You Do It 2: practice creating Parameter Queries using Query Builder dialog box - 3 examples from Mini-Quiz 12-2 with 04.You Do It 2 Solution

1. open the: ...VB2017\Chap12\Exercise04.You Do It 2 Solution\You Do It 2 Solution.sln

2. use the application to test the 3 SELECT statements with a Parameter Queries from **Mini-Quiz 12-2**:

**Mini-Quiz 12-2** Using the **dbo.Winners** table from **Figure 12-15**, write a parameter query that:

Q1. selects only an Actor's Picture field, where the name of the Actor is provided by the user.

Q2. selects only an Animated picture's Year field, where the name of the Animated picture is provided by the user.

Q3. selects the Actor and Actress fields for the Year provided by the user.

A1. `SELECT Picture FROM Winners WHERE Actor = @Actor`

A2. `SELECT Year FROM Winners WHERE Animated = @Animated`

A3. `SELECT Actor, Actress FROM Winners WHERE Year = @Year`

**step 1** - open the DataSet Designer window:

-> **Solution Explorer** window / **OscarsDataSet.xsd** DataSet / Rclick: **Open**

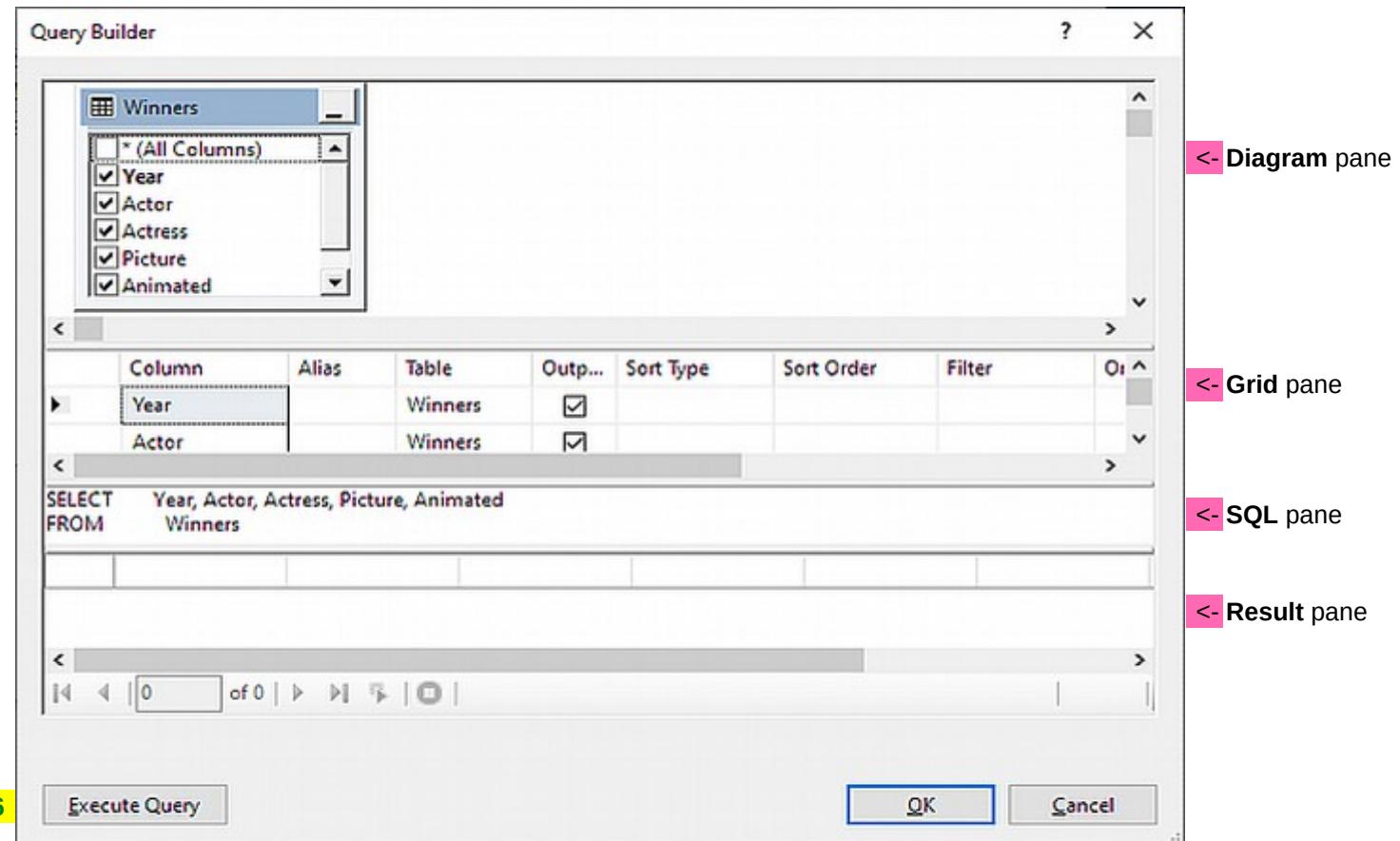
or -> **Data Sources** window / **OscarsDataSet** / Rclick: **Edit DataSet with Designer**

**step 2** - start/open the **TableAdapter Query Configuration Wizard**:

-> **OscarsDataSet.xsd** Designer window / line **WinnersTableAdapter** / Rclick: **Add / Query...**

**step 3** - get to the **Query Builder** dialog box:

-> Use SQL statements -> SELECT which returns rows -> button **Query Builder...** **Figure 12-26**



**Figure 12-26**

**Q1.** Write a **parameter query** that selects only an Actor's Picture field, where the name of the Actor is provided by the user.

**A1.** `SELECT Picture FROM Winners WHERE Actor = @Actor`

- > **Diagram pane** => select only check boxes for the columns: Actor, Picture
- > **Grid pane**: column Actor / Filter => **@Actor**
- > **Grid pane**: column Actor / Output => **uncheck the check box**
- > button Execute Query to run/preview/test
  - > in the **Query Parameters** dialog box, row: **@Actor** / column: **Value** => type any actor's name to display its corresponding Picture field
    - e.g. **Jeff Bridges** -> the **Result** pane will show the name of the Picture field: **The Hurt Locker**
    - <- notice that if you type a name; what is not included in the table **dbo.Winners**, the result pane will include a **NULL** value

**Q2.** Write a **parameter query** that selects only an Animated picture's Year field, where the name of the Animated picture is provided by the user.

**A2.** `SELECT Year FROM Winners WHERE Animated = @Animated`

- > **Grid pane** => as a first, delete any previous values
- > **Diagram pane** => select only check boxes for the columns: Year, Animated
- > **Grid pane**: column Animated / Filter => **@Animated**
- > **Grid pane**: column Animated / Output => **uncheck the check box**
- > button Execute Query to run/preview/test
  - > in the **Query Parameters** dialog box, row: **@Animated** / column: **Value** => type any animated picture's name to display its corresponding Year field
    - e.g. **Wall-e** -> the **Result** pane will show the value in its corresponding Year field: **2009**

**Q3.** Write a **parameter query** that selects the Actor and Actress fields for the Year provided by the user.

**A3.** `SELECT Actor, Actress FROM Winners WHERE Year = @Year`

- > **Grid pane** => as a first, delete any previous values
- > **Diagram pane** => select only check boxes for the columns: Year, Actor, Actress
- > **Grid pane**: column Year / Filter => **@Year**
- > **Grid pane**: column Year / Output => **uncheck the check box**
- > button Execute Query to run/preview/test
  - > in the **Query Parameters** dialog box, row: **@Year** / column: **Value** => type any year
    - e.g. **2010** -> the **Result** pane will show the corresponding values from fields: Actor, and Actress: Jeff Bridges, Sandra Bullock

- when done testing, cancel all of the changes, and close the solution

- > in the **Query Builder** dialog box click the button **Cancel** and then in the **TableAdapter Query Configuration Wizard** click the button **Cancel**
- > save and close the solution

## CH12\_F4 - use Parameter Query during run time - step 1/2: Create and then Save Query by associating it with a method - e.g. 05.Oscars Solution-Save Query

- for an application to use a **query** during run time, you need to:

- 1). **create** the query - by using the **TableAdapter Query Configuration Wizard**
- 2). **save** the query - by associating the query with one or more methods (Fill a DataTable, Return a DataTable - methods **Fill**, **Get**)
- 3). **invoke** it from code - by entering in procedure the **Fill** method associated with the query

- you save a **query** that contains the **SELECT** statement by associating the query with 1 or more **methods**

- the **TableAdapter Query Configuration Wizard** provides an easy way to perform this task

- to save a **Query** using the **TableAdapter Query Configuration Wizard** example with **05.Oscars Solution-Save Query**

-> open the: ...VB2017\Chap12\Exercise\05.Oscars Solution-Save Query\Oscars Solution.sln

-> open the **Designer** and **Solution Explorer** windows

- as you can see, The **Oscar Winners** application is already connected to the **Oscars.mdf** database, and the **OscarsDataSet.xsd** is already created

-> start the application to see the GUI: **Figure 12-30**

- the application allows the user to display:
  - a). either all of the records in the dataset
  - b). or only the record for the **year** entered in the (**txtYear**) control

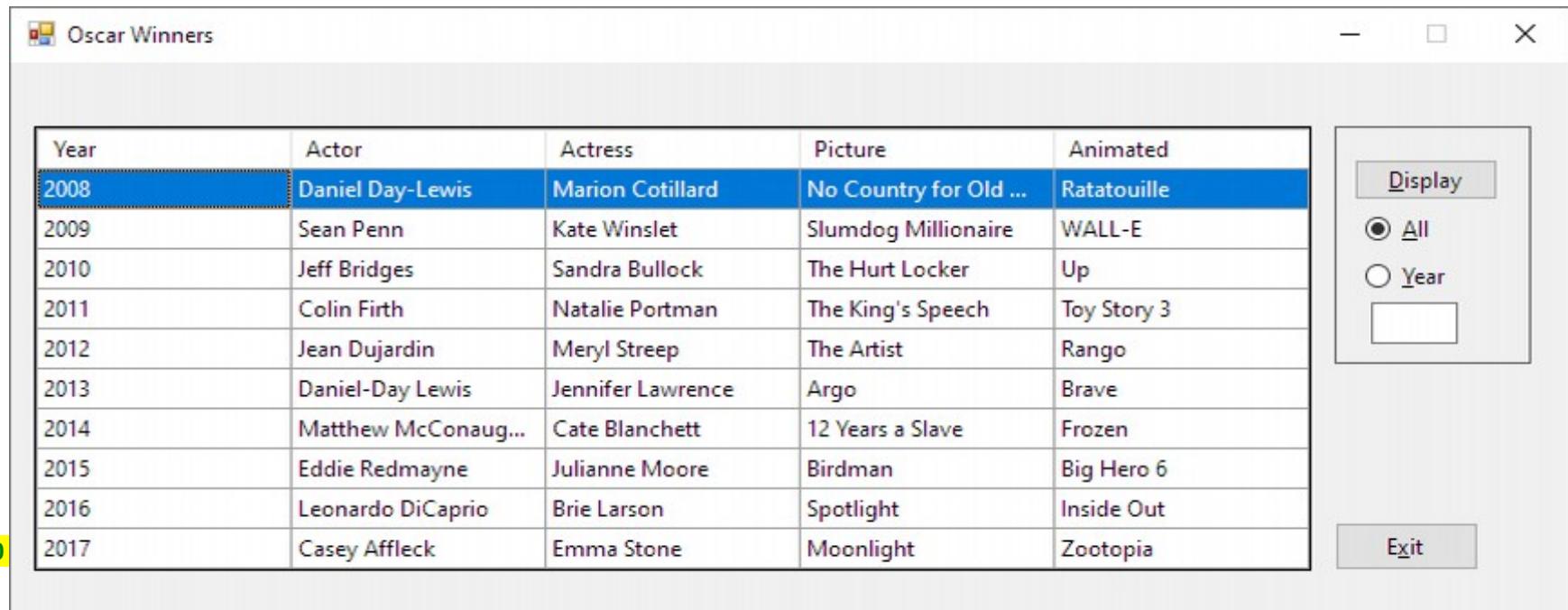


Figure 12-30

-> click the button **Exit** to close the GUI preview

**step 1.1** - open the DataSet Designer window:

-> Solution Explorer window / OscarsDataSet.xsd DataSet / Rclick: Open

or -> Data Sources window / OscarsDataSet / Rclick: Edit DataSet with Designer

**step 1.2** - start/open the TableAdapter Query Configuration Wizard:

-> OscarsDataSet.xsd Designer window / line WinnersTableAdapter / Rclick: Add / Query...

**step 1.3** - get to the Query Builder dialog box (the same way like in all of the previous instances):

-> screen: Choose a Command Type => Use SQL statements & button Next >

-> screen: Choose a Query Type => SELECT which returns rows & button Next >

-> screen: Specify a SQL SELECT statement opens

<- notice the box: What data should the table load?

<- notice that the box contains the default query: SELECT Year, Actor, Actress, Picture, Animated FROM dbo.Winners

- the default query: 1). selects all of the fields and records from the table, and

2). is automatically executed when the frmMain\_Load procedure invokes the WinnersTableAdapter object's Fill method

-> screen: Specify a SQL SELECT statement => button Query Builder... to open the Query Builder dialog box

**step 1.4** - you will create a Parameter Query that displays the Oscars winners for the year entered by the user during run time in the (txtYear) control

-> Grid pane: column Year / Filter => @Year

<- notice what Query Builder did: 1). in Grid pane: changed the entry in the Filter column to: = @Year

2). in SQL pane: added the: WHERE (Year = @Year) clause

-> button Execute Query to run/preview/test

<- notice the Query Parameters dialog box opens

-> in the row: @Year / column: Value => type: 2010 and press Enter key

<- in the Result pane now appears the 2020 record - that is correct

- now that you know that the query works correctly, you can save it and close the Query Builder dialog box

-> in the Query Builder dialog box => button OK

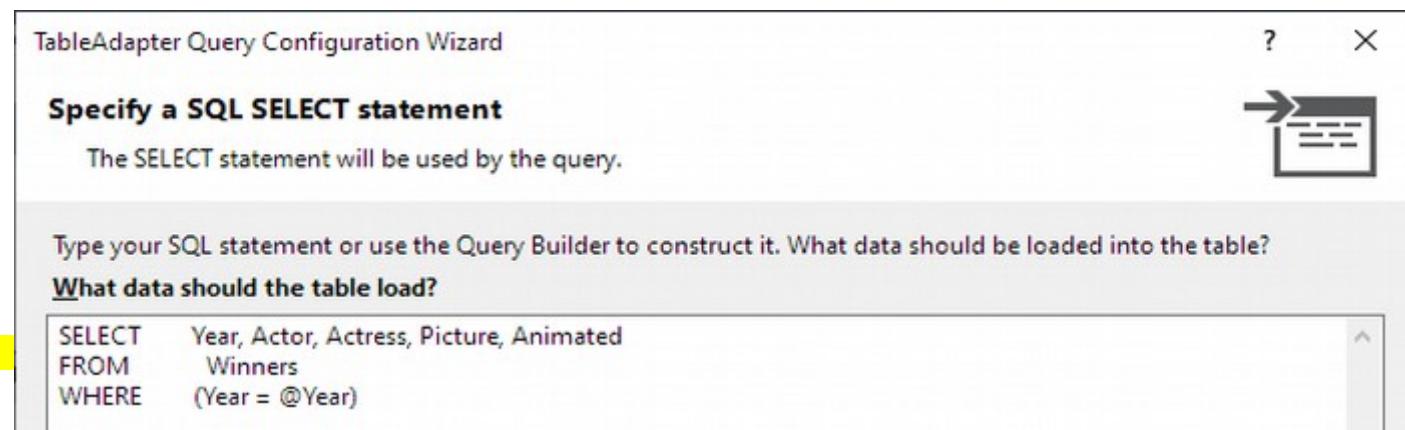
**step 1.5** - as the Query Builder dialog box closed, you are returned to the screen: Specify a SQL SELECT statements

<- notice the box: What data should the table load?

<- notice the default query: SELECT Year, Actor, Actress, Picture, Animated FROM dbo.Winners

from: step 3

is changed for the Parameter Query: **Figure 12-31**



-> click the button Next > to display the screen: Choose Methods to Generate

**step 1.6**

- the screen: **Choose Methods to Generate** opens **Figure 12-32**

<- notice the: **Which methods do you want to add to the TableAdapter?**

-> **Fill a DataTable** check box should be selected

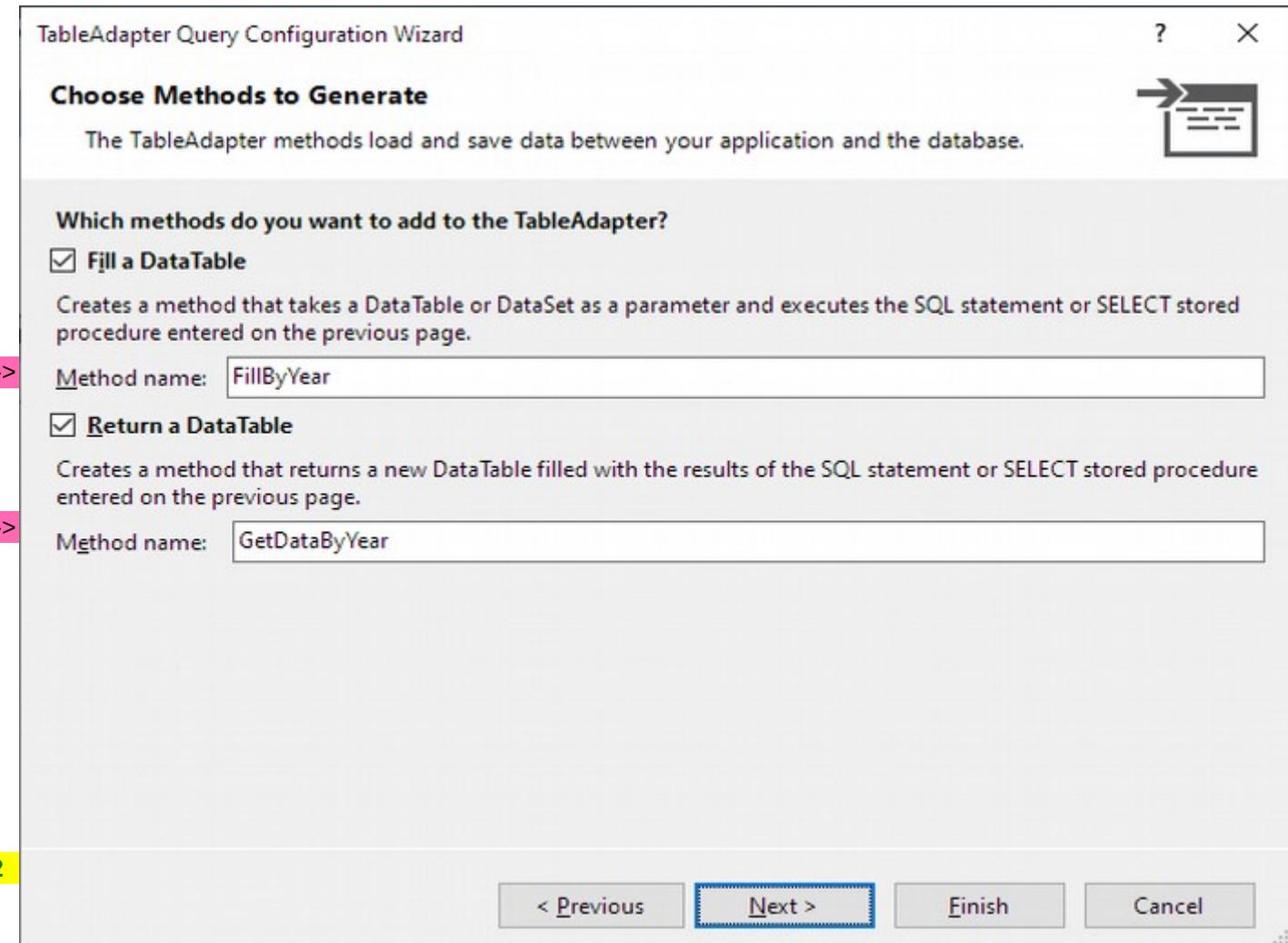
-> change the **Fill a DataTable** method's name from default: **FillBy** to a more meaningful method's name: **FillByYear**

-> **Return a DataTable** check box should be selected

-> change the **Return a DataTable** method's name from default: **GetDataBy** to a more meaningful method's name: **GetDataByYear**

- the methods: **FillByYear** and **GetDataByYear** are associated with the **Parameter Query** that you created in: **step 4** & **step 5**

<- therefore you can use them to invoke the query during run time



default method's name **FillBy**  
changed to **FillByYear** ->

default method's name **GetDataBy**  
changed to **GetDataByYear** ->

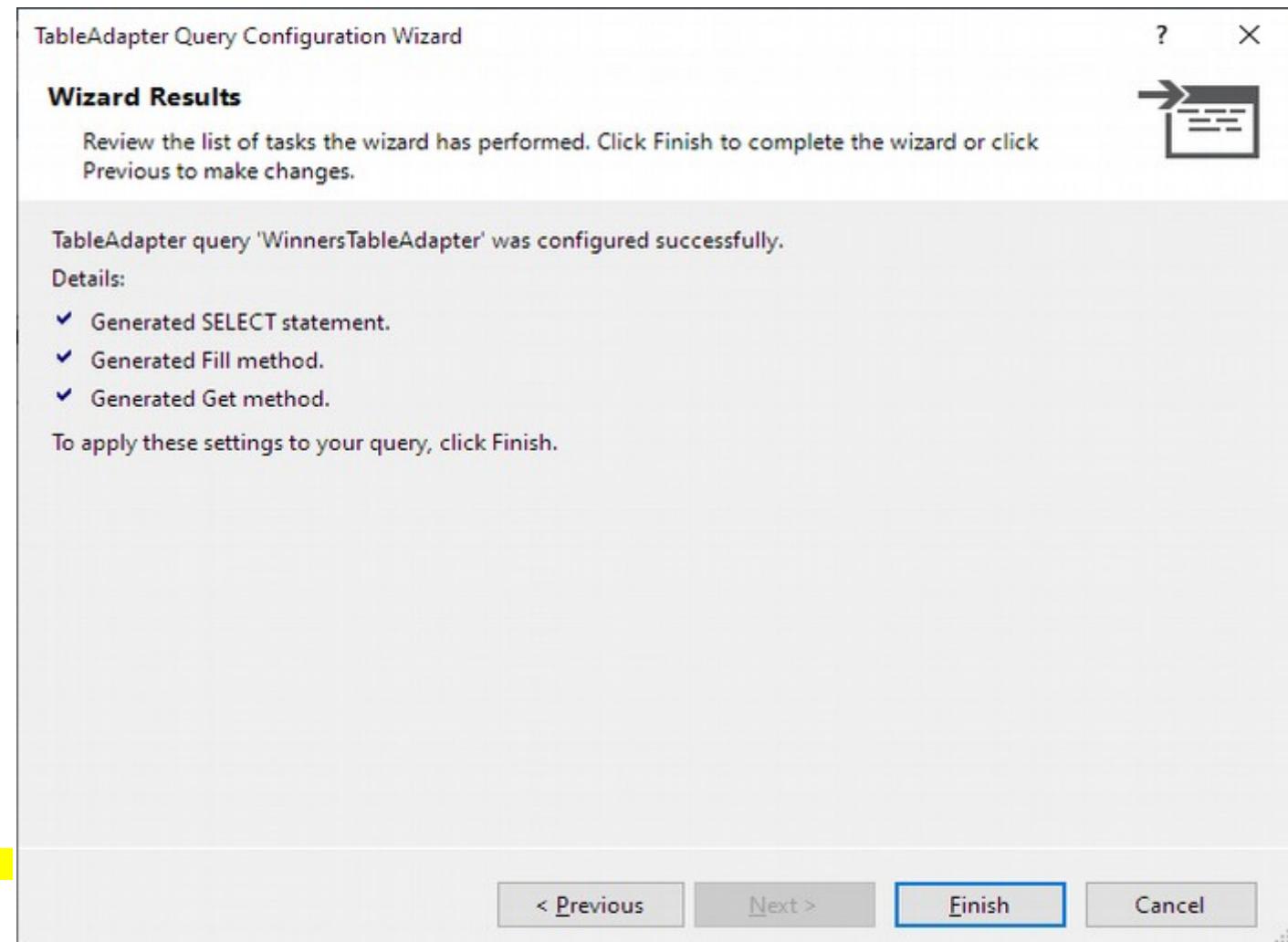
-> click the button **Next >** to display the last screen: **Wizard Results**

**step 1.7**

- the last screen: **Wizard Results** opens

**Figure 12-33**

<- here you can verify that all the steps were successful and you did not forget anything



**Figure 12-33**

-> click the button **Finish** to end the **TableAdapter Query Configuration Wizard** and return to **DataSet Designer** window named: **OscarsDataSet.xsd**

**step 1.8** <- the DataSet Designer window / **WinnersTableAdapter** now includes a new **Parameter Query** methods - **FillByYear**, **GetDataByYear(@Year)**

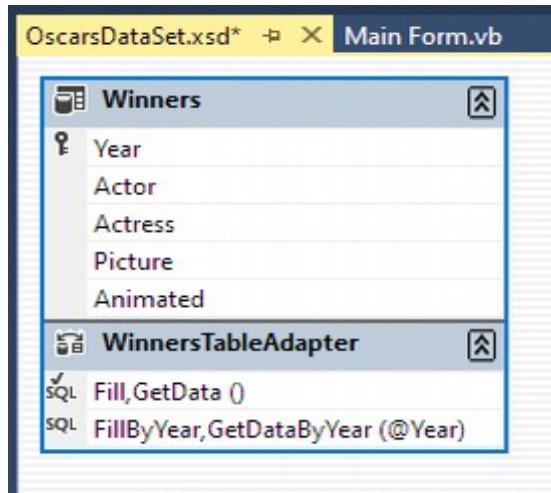


Figure 12-34

<- **default query**, whose method retrieves all of the records

<- **parameter query**, whose method retrieves only the record for the year entered by the user

-> save the solution and then close the **Dataset Designer** window named: **OscarsDataSet.xsd**

<- how to invoke a **default / parameter Query** from the code will be described in the next chapter:

#### CH12\_F5 - use Parameter Query during run time - step 2/2: Invoke/Call a method associated with a Query - syntax, e.g. 05.Oscars Solution-Save Query

- the **Display** button in the Oscar Winners application is responsible for invoking/calling the appropriate query:

Figure 12-30

- a. either the default **Fill** query - if the radio button All is selected
- b. or the parameterized **FillByYear** query - if the radio button Year is selected

default Fill method's calling/invoking statement:

`YourTableAdapter.Fill(YourDataSet.YourTable)`

**As YourPrimaryKey** data type

e.g. used in here:

`WinnersTableAdapter.Fill(OscarsDataSet.Winners)`

VS2017 syntax:

`Function OscarsDataSetTableAdapters.WinnersTableAdapter.Fill(dataTable As OscarsDataSet.WinnersDataTable) As Integer`

parameterized Fill method's calling/invoking statement:

`YourTableAdapter.YourParameterizedMethodFill(YourDataSet.YourTable, YourParameter)`

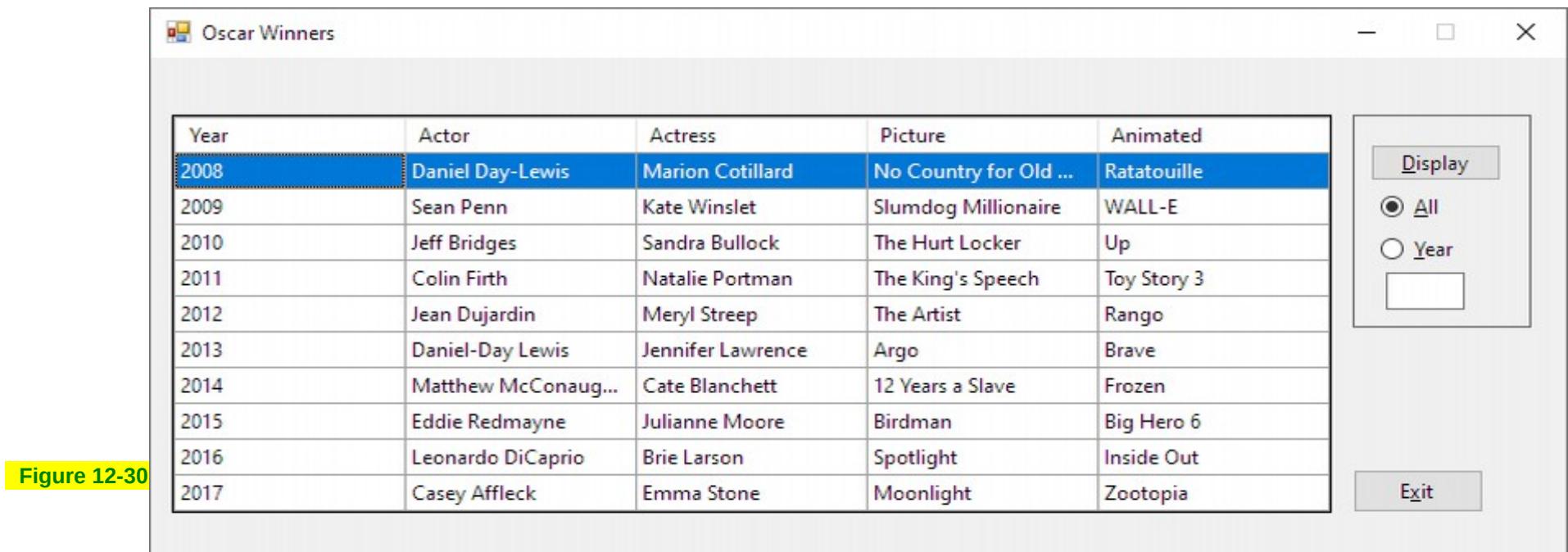
**As YourPrimaryKey** data type

e.g. used in here:

`WinnersTableAdapter.FillByYear(OscarsDataSet.Winners, intYear)`

VS2017 syntax:

`Function OscarsDataSetTableAdapters.WinnersTableAdapter.FillByYear(dataTable As OscarsDataSet.WinnersDataTable, Year As Integer) As Integer`



- you can invoke/call a query during run time by entering its associated **Fill** method in a procedure's code
- in this application, you will enter the methods in the **btnDisplay\_Click** procedure

- to code the **btnDisplay\_Click** procedure using default and parameterized methods **Fill**: example with **05.Oscars Solution-Save Query**

-> open the: ...VB2017\Chap12\Exercise\05.Oscars Solution-Save Query\Oscars Solution.sln

-> open the **Code Editor** window and locate the **btnDisplay\_Click** procedure

**step 2.1** - if the **(radAll)** is selected, the procedure will use the **WinnersTableAdapter** object's default **Fill** method to select all of the records:

```

<- YourTableAdapter.Fill(YourDataSet.YourTable) As YourPrimaryKey data type
<- WinnersTableAdapter.Fill(OscarsDataSet.Winners)
<- Function OscarsDataSetTableAdapters.WinnersTableAdapter.Fill(dataTable As OscarsDataSet.WinnersDataTable) As Integer
  
```

-> enter the selection structure below:

```

10  Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
11      ' Displays all records or a record for a specific year.
12
13      If radAll.Checked Then
14          WinnersTableAdapter.Fill(OscarsDataSet.Winners)
15      Else
16
17          End If
18      End Sub
  
```

**step 2.2** - if the (radYear) is selected, the procedure will use the **WinnersTableAdapter** object's parameterized **FillByYear** method to select the record whose Year field matches the year number entered during run time in the (txtYear) control

- first, the procedure will determine whether the control contains a value:

<- if it does not contain a value, the procedure will display an appropriate message

-> enter the selection structure and a message box as shown on lines: **16 - 20**

```
10  Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
11      ' Displays all records or a record for a specific year.
12
13      If radAll.Checked Then
14          WinnersTableAdapter.Fill(OscarsDataSet.Winners)
15      Else
16          If txtYear.Text.Trim = String.Empty Then
17              MessageBox.Show("Please enter the year.", "Oscar Winners", MessageBoxButtons.OK, MessageBoxIcon.Information)
18          Else
19
20          End If
21      End If
22  End Sub
```

**step 2.3** - the Year field in the database has the SQL **int** data type, so the procedure will need to convert the (txtYear) entry to an **integer**

-> enter the declaration statement and the **TryParse** method as shown below:

```
10  Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
11      ' Displays all records or a record for a specific year.
12
13      If radAll.Checked Then
14          WinnersTableAdapter.Fill(OscarsDataSet.Winners)
15      Else
16          If txtYear.Text.Trim = String.Empty Then
17              MessageBox.Show("Please enter the year.", "Oscar Winners", MessageBoxButtons.OK, MessageBoxIcon.Information)
18          Else
19              Dim intYear As Integer
20              Integer.TryParse(txtYear.Text.Trim, intYear)
21
22          End If
23      End If
24  End Sub
```

**step 2.4** - next, the procedure will invoke the **WinnersTableAdapter** object's **FillByYear** method

- the method is associated with a **parameter query**, so it will need to include the parameter information

<- the user's entry in **(txtYear) / intYear** will be the parameter sent to the method

<- **YourTableAdapter.YourParameterizedMethodFill(YourDataSet.YourTable, YourParameter)** **As YourPrimaryKey** data type

<- **WinnersTableAdapter.FillByYear(OscarsDataSet.Winners, intYear)**

<- **Function OscarsDataSetTableAdapters.WinnersTableAdapter.FillByYear(dataTable As OscarsDataSet.WinnersDataTable, Year As Integer) As Integer**

-> enter the parameterized method as shown below:

```
10     Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
11         ' Displays all records or a record for a specific year.
12
13         If radAll.Checked Then
14             WinnersTableAdapter.Fill(OscarsDataSet.Winners)
15         Else
16             If txtYear.Text.Trim = String.Empty Then
17                 MessageBox.Show("Please enter the year.", "Oscar Winners", MessageBoxButtons.OK, MessageBoxIcon.Information)
18             Else
19                 Dim intYear As Integer
20                 Integer.TryParse(txtYear.Text.Trim, intYear)
21                 WinnersTableAdapter.FillByYear(OscarsDataSet.Winners, intYear)
22             End If
23         End If
24     End Sub
```

**step 2.5** -> save the solution and then start and test the application **Figure 12-35a**

<- the default **Fill** method in the **frmMain\_Load** procedure fills the dataset with data appearing in the **DataGridView** control because the control is bound to the **Winners** table in the dataset

-> click the radio button **Year** and then click the button **Display**

<- the message box appears: **Please enter the year.** **Figure 12-35b**

-> close the message box

-> in the text box type: **2013** and then click the button **Display** **Figure 12-35c**

<- the **btnDisplay\_Click** procedure invokes the **FillByYear** method, which retrieves only the record for the year **2013** appearing in the **DataGridView**

-> click the radio button **All** and then click the button **Display** **Figure 12-35d**

<- the **btnDisplay\_Click** procedure invokes the default **Fill** method, which retrieves all of the records from the dataset appearing in the **DataGridView**

-> click the button **Exit**, close the **Code Editor** window and then close the solution

**Oscar Winners**

| Year | Actor               | Actress           |
|------|---------------------|-------------------|
| 2008 | Daniel Day-Lewis    | Marion Cotillard  |
| 2009 | Sean Penn           | Kate Winslet      |
| 2010 | Jeff Bridges        | Sandra Bullock    |
| 2011 | Colin Firth         | Natalie Portman   |
| 2012 | Jean Dujardin       | Meryl Streep      |
| 2013 | Daniel Day-Lewis    | Jennifer Lawrence |
| 2014 | Matthew McConaughey | Cate Blanchett    |
| 2015 | Eddie Redmayne      | Julianne Moore    |
| 2016 | Leonardo DiCaprio   | Brie Larson       |
| 2017 | Casey Affleck       | Emma Stone        |

Figure 12-35a

**Oscar Winners**

| Year | Actor               | Actress           | Picture                | Animated    |
|------|---------------------|-------------------|------------------------|-------------|
| 2008 | Daniel Day-Lewis    | Marion Cotillard  | No Country for Old Men | Ratatouille |
| 2009 | Sean Penn           |                   | Slumdog Millionaire    | WALL-E      |
| 2010 | Jeff Bridges        | Sandra Bullock    | The Hurt Locker        | Up          |
| 2011 | Colin Firth         | Natalie Portman   | The King's Speech      | Toy Story 3 |
| 2012 | Jean Dujardin       | Meryl Streep      | The Artist             | Rango       |
| 2013 | Daniel Day-Lewis    | Jennifer Lawrence | Argo                   | Brave       |
| 2014 | Matthew McConaughey | Cate Blanchett    | 12 Years a Slave       | Frozen      |
| 2015 | Eddie Redmayne      | Julianne Moore    | Birdman                | Big Hero 6  |
| 2016 | Leonardo DiCaprio   | Brie Larson       | Spotlight              | Inside Out  |
| 2017 | Casey Affleck       | Emma Stone        | Moonlight              | Zootopia    |

Please enter the year.

OK

Display

All

Year

Exit

Figure 12-35b

**Oscar Winners**

| Year | Actor            | Actress           | Picture | Animated |
|------|------------------|-------------------|---------|----------|
| 2013 | Daniel Day-Lewis | Jennifer Lawrence | Argo    | Brave    |

Figure 12-35c

**Oscar Winners**

| Year | Actor               | Actress           | Picture                | Animated    |
|------|---------------------|-------------------|------------------------|-------------|
| 2008 | Daniel Day-Lewis    | Marion Cotillard  | No Country for Old Men | Ratatouille |
| 2009 | Sean Penn           | Kate Winslet      | Slumdog Millionaire    | WALL-E      |
| 2010 | Jeff Bridges        | Sandra Bullock    | The Hurt Locker        | Up          |
| 2011 | Colin Firth         | Natalie Portman   | The King's Speech      | Toy Story 3 |
| 2012 | Jean Dujardin       | Meryl Streep      | The Artist             | Rango       |
| 2013 | Daniel Day-Lewis    | Jennifer Lawrence | Argo                   | Brave       |
| 2014 | Matthew McConaughey | Cate Blanchett    | 12 Years a Slave       | Frozen      |
| 2015 | Eddie Redmayne      | Julianne Moore    | Birdman                | Big Hero 6  |
| 2016 | Leonardo DiCaprio   | Brie Larson       | Spotlight              | Inside Out  |
| 2017 | Casey Affleck       | Emma Stone        | Moonlight              | Zootopia    |

Display

All

Year

2013

Exit

Figure 12-35d

- the entire code:

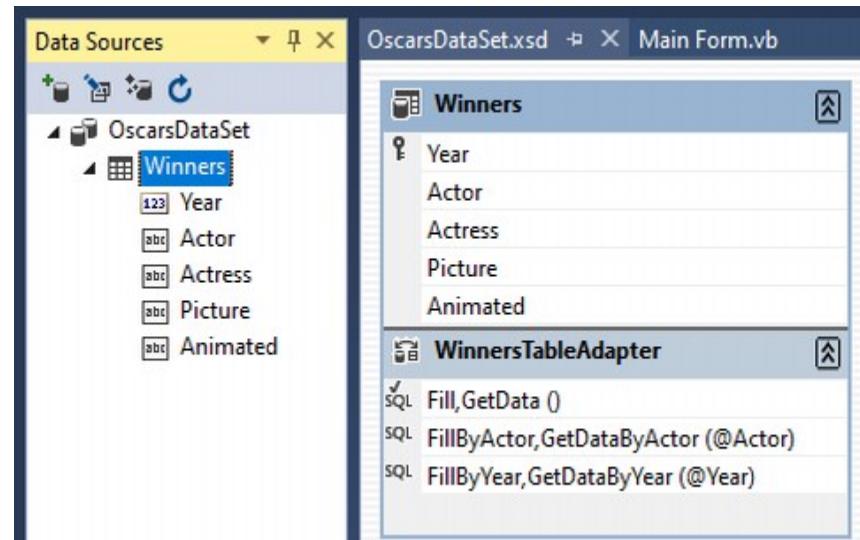
```
1  ' Name:      Oscars Project
2  ' Purpose:    Displays either all records or a record for a specific year.
3  ' Programmer: <your name> on <current date>
4
5  Option Explicit On
6  Option Strict On
7  Option Infer Off
8
9  Public Class frmMain
10     Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
11         ' Displays all records or a record for a specific year.
12
13         If radAll.Checked Then
14             WinnersTableAdapter.Fill(OscarsDataSet.Winners)
15         Else
16             If txtYear.Text.Trim = String.Empty Then
17                 MessageBox.Show("Please enter the year.", "Oscar Winners", MessageBoxButtons.OK, MessageBoxIcon.Information)
18             Else
19                 Dim intYear As Integer
20                 Integer.TryParse(txtYear.Text.Trim, intYear)
21                 WinnersTableAdapter.FillByYear(OscarsDataSet.Winners, intYear)
22             End If
23         End If
24     End Sub
25
26     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
27         Me.Close()
28     End Sub
29
30     Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles MyBase.Load
31         'TODO: This line of code loads data into the 'OscarsDataSet.Winners' table. You can move, or remove it, as needed.
32         Me.WinnersTableAdapter.Fill(Me.OscarsDataSet.Winners)
33     End Sub
34 End Class
```

**CH12\_F6 - My Test:** - use Parameter Query with a parameter marker @ during run time with a LIKE operator & allow users to use its wildcards %, \_  
 - plus Q & A on several FB programmer's groups about LIKE operator  
 - e.g. with: [05.Oscars Solution-Save Query\\_My Test - Parameter Query and LIKE operator](#)

- as a test, I added to:

- GUI: Radio box (**radActor**) and Text Box (**txtActor**)
- OscarsDataSet Designer / WinnersTableAdapter: **FillByActor,GetDataByActor(@Actor)**
- Code: - selection structure
- method:

```
WinnersTableAdapter.FillByActor(OscarsDataSet.Winners, txtActor.Text)
```



**WinnersTableAdapter** autopsy:

1. default SELECT statement: `SELECT Year, Actor, Actress, Picture, Animated FROM dbo.Winners`  
 associated with a methods: `Fill, GetData ()`  
 methods used in code: `WinnersTableAdapter.Fill(OscarsDataSet.Winners)` on lines: [16, 22, 34](#)

2. query SELECT statement: `SELECT Year, Actor, Actress, Picture, Animated  
FROM Winners  
WHERE (Actor LIKE@Actor)`  
 associated with a methods: `FillByActor, GetDataByActor (@Actor)`  
 methods used in code: `WinnersTableAdapter.FillByActor(OscarsDataSet.Winners, txtActor.Text)` line: [37](#)

3. query SELECT statement: `SELECT Year, Actor, Actress, Picture, Animated  
FROM Winners  
WHERE (Year = @Year)`  
 associated with a methods: `FillByYear, GetDataByYear (@Year)`  
 methods used in code: `WinnersTableAdapter.FillByYear(OscarsDataSet.Winners, intYear)` on line: [27](#)

```

1  ' Name:      05.Oscars Solution-Save Query_My Test - Parameter Query and LIKE operator
2  ' Purpose:    Displays either all records, a record for a specific year, or record by the name -
3  '             - can use a LIKE operator's wildcards: % = 0 or more characters, and _ = 1 character
4  ' Programmer: <Já> on <02/07/2021>
5  Option Explicit On
6  Option Strict On
7  Option Infer Off

```

```

8
9  Public Class frmMain
10     Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
11         ' Displays either all records, a record for a specific year, or a record for a specific actor.
12
13         ' a). display all records:
14         If radAll.Checked Then
15             'txtYear.Text = Nothing : txtActor.Text = Nothing
16             WinnersTableAdapter.Fill(OscarsDataSet.Winners)      ' fill whole Table
17
18         ' b). find record by the Year:
19         ElseIf radYear.Checked Then
20             If txtYear.Text.Trim = String.Empty Then
21                 'MessageBox.Show("Please enter the year.", "Oscar Winners", MessageBoxButtons.OK, MessageBoxIcon.Information)
22                 WinnersTableAdapter.Fill(OscarsDataSet.Winners)      ' fill whole Table
23             Else
24                 Dim intYear As Integer
25                 Integer.TryParse(txtYear.Text.Trim, intYear)
26                 ' Send the input Year and Fill the search result:
27                 WinnersTableAdapter.FillByYear(OscarsDataSet.Winners, intYear)
28             End If
29
30         ' c). find record by the Actor - can use LIKE's wildcards %, and _:
31         Else
32             If txtActor.Text.Trim = String.Empty Then
33                 'MessageBox.Show("Please enter the Actor's name.", "Oscar Winners", MessageBoxButtons.OK, MessageBoxIcon.Information)
34                 WinnersTableAdapter.Fill(OscarsDataSet.Winners)      ' fill whole Table
35             Else
36                 ' Send the Actor's name and Fill the search result:
37                 WinnersTableAdapter.FillByActor(OscarsDataSet.Winners, txtActor.Text)
38             End If
39         End If
40     End Sub
41
42     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
43         Me.Close()
44     End Sub
45
46     Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles MyBase.Load
47         'TODO: This line of code loads data into the 'OscarsDataSet.Winners' table. You can move, or remove it, as needed.
48         Me.WinnersTableAdapter.Fill(Me.OscarsDataSet.Winners)
49     End Sub
50 End Class

```

Oscar Winners

| Year | Actor               | Actress           | Picture                | Animated    |
|------|---------------------|-------------------|------------------------|-------------|
| 2008 | Daniel Day-Lewis    | Marion Cotillard  | No Country for Old Men | Ratatouille |
| 2009 | Sean Penn           | Kate Winslet      | Slumdog Millionaire    | WALL-E      |
| 2010 | Jeff Bridges        | Sandra Bullock    | The Hurt Locker        | Up          |
| 2011 | Colin Firth         | Natalie Portman   | The King's Speech      | Toy Story 3 |
| 2012 | Jean Dujardin       | Meryl Streep      | The Artist             | Rango       |
| 2013 | Daniel Day Lewis    | Jennifer Lawrence | Argo                   | Brave       |
| 2014 | Matthew McConaughey | Cate Blanchett    | 12 Years a Slave       | Frozen      |
| 2015 | Eddie Redmayne      | Julianne Moore    | Birdman                | Big Hero 6  |
| 2016 | Leonardo DiCaprio   | Brie Larson       | Spotlight              | Inside Out  |
| 2017 | Casey Affleck       | Emma Stone        | Moonlight              | Zootopia    |

All  
 Year   
 Actor

Info:  
 1. Leave the boxes empty to display all.  
 2. In the Actor's box can be used wildcards:  
     % ... for 0 or more characters  
     \_ ... for only 1 character

Oscar Winners

| Year | Actor         | Actress      | Picture    | Animated |
|------|---------------|--------------|------------|----------|
| 2012 | Jean Dujardin | Meryl Streep | The Artist | Rango    |

All  
 Year   
 Actor

Info:  
 1. Leave the boxes empty to display all.  
 2. In the Actor's box can be used wildcards:  
     % ... for 0 or more characters  
     \_ ... for only 1 character

| Year | Actor         | Actress        | Picture         | Animated |
|------|---------------|----------------|-----------------|----------|
| 2010 | Jeff Bridges  | Sandra Bullock | The Hurt Locker | Up       |
| 2012 | Jean Dujardin | Meryl Streep   | The Artist      | Rango    |

Oscar Winners

| Year | Actor       | Actress         | Picture           | Animated    |
|------|-------------|-----------------|-------------------|-------------|
| 2011 | Colin Firth | Natalie Portman | The King's Speech | Toy Story 3 |

 All Year

2012

 Actor

colin%

## Info:

1. Leave the boxes empty to display all.

Oscar Winners

| Year | Actor     | Actress      | Picture             | Animated |
|------|-----------|--------------|---------------------|----------|
| 2009 | Sean Penn | Kate Winslet | Slumdog Millionaire | WALL-E   |

 All Year

2012

 Actor

\_\_\_\_\_

## Info:

1. Leave the boxes empty to display all.
2. In the Actor's box can be used wildcards:  
% ... for 0 or more characters  
\_ ... for only 1 character

Oscar Winners

| Year | Actor         | Actress        | Picture         | Animated |
|------|---------------|----------------|-----------------|----------|
| 2010 | Jeff Bridges  | Sandra Bullock | The Hurt Locker | Up       |
| 2017 | Casey Affleck | Emma Stone     | Moonlight       | Zootopia |

 All Year

2012

 Actor

%ff%

## Info:

1. Leave the boxes empty to display all.
2. In the Actor's box can be used wildcards:  
% ... for 0 or more characters  
\_ ... for only 1 character

[Display](#)[Exit](#)

## Q & A on several FB programmer's groups about LIKE operator:

The screenshot shows the Microsoft Visual Studio IDE with the 'OscarsSolution' project open. In the 'Main Form.vb [Design]' window, there is a Windows Form titled 'Oscar Winners'. On the form, there is a DataGridView displaying data from a table named 'Winners'. The table has columns: Year, Actor, Actress, Picture, and Animated. The data in the DataGridView is as follows:

| Year | Actor               | Actress           | Picture                | Animated    |
|------|---------------------|-------------------|------------------------|-------------|
| 2008 | Daniel Day-Lewis    | Marion Cotillard  | No Country for Old Men | Ratatouille |
| 2009 | Sean Penn           | Kate Winslet      | Slumdog Millionaire    | WALL-E      |
| 2010 | Jeff Bridges        | Sandra Bullock    | The Hurt Locker        | Up          |
| 2011 | Colin Firth         | Natalie Portman   | The King's Speech      | Toy Story 3 |
| 2012 | Jean Dujardin       | Meryl Streep      | The Artist             | Rango       |
| 2013 | Daniel Day Lewis    | Jennifer Lawrence | Argo                   | Brave       |
| 2014 | Matthew McConaughey | Cate Blanchett    | 12 Years a Slave       | Frozen      |
| 2015 | Eddie Redmayne      | Julianne Moore    | Birdman                | Big Hero 6  |
| 2016 | Leonardo DiCaprio   | Brie Larson       | Spotlight              | Inside Out  |
| 2017 | Casey Affleck       | Emma Stone        | Moonlight              | Zootopia    |

To the right of the DataGridView, there is a search interface. It includes three radio buttons: 'All' (selected), 'Year', and 'Actor'. There are also two text input fields and a 'Display' button.

**Q CZ:** ...a přidal jsem vyhledávání podle jména herce a použil SELECT statement WHERE's operátor LIKE aby bylo možné vyhledávat s pomocí wildcards % a \_.  
Vše funguje skvěle a proto by mě zajímalo, jestli v praxi používáte operátor LIKE v podobných případech a umožníte uživatelům používat wildcards?

**Q EN:** ...and added another TextBox to search by the Actor's name using SELECT statement's LIKE operator and it's wildcards % and \_.  
Everything works great, so therefore I would like to ask if you are using the LIKE operator in db's allowing the user to use the wildcards?

### A EN / CZ:

Martin Widmer-Phutadon - Address SQL injection please.

Bartholomew ZQ Ho - Try linqtosql, it will work like an array or list.

### C#.Net Programming

Christopher O'Shell - When I use it, I have the user use the "\*" character instead of the '%' and do a character swap in the backend before submitting the query.  
The '%' is not something most users are familiar with, and it's pretty easy to do in your input validation stage  
(you should ALWAYS validate the user's input to prevent SQL injection attacks).

Barth Manobis Obiefuna - From your explanation, it shows you are more involved in ADO.Net.  
This will make you have more SELECT, INSERT, DELETE, UPDATE statements all over your project.  
I bet you this is cumbersome and annoying.  
- I would encourage you to resort to use of Entity Framework (EF).  
It takes care of all the worries and frustrations encountered with the former.  
With EF, you define your db tables in classes and can easily readjust the tables from the classes and update the db.  
Code1st approach is what I could recommend.

## .NET/C#/XAMARIN DEVELOPER

Henri Van Wesemael - Sure, just don't build the sql by concatenating the user entered value; use parameters to prevent sql injection attack.

### Programátoři začátečníci

Jakub Muller - Příkaz "LIKE" se v SQL používá zcela běžně. Pokud jde ale o nějaké "lepší" vyhledávání, používá se "FullText", protože se jedná mnohem silnější technologií a její možnosti jsou mnohem blíže potřebám "běžných smrtelníků (tm)".

Vlastimil Pospíchal - jen pozor, LIKE může dělat problémy při injekci.

- netvrďme vyhnout. Ošetřit vstupní data, aby se nedalo injektovat něco škodlivého

Milan Křepelka - Tohle je problém nižších jazyků který SQL šmudlají skládáním surového textu. Na platformě .net, v základní úrovni, stačí používat DB parametry které tě od toho v základu odstíní. Vyhnut se tomu všemu bastlení pomocí Linq technologií by tě mohlo posunout dále.

- for more info see downloaded file: [ADO.NET 4.5\\_docs.microsoft.com\\_Milan Křepelka suggestion to read\\_605p\\_2021.pdf](#)

[https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/configuring-parameters-and-parameter-data-types?fbclid=IwAR25YZ0b1U9d3lni2bD4bWjCuGYBeSVRYH10VLg7zagZqcYTc2RSp\\_0V2sM](https://docs.microsoft.com/en-us/dotnet/framework/data/adonet/configuring-parameters-and-parameter-data-types?fbclid=IwAR25YZ0b1U9d3lni2bD4bWjCuGYBeSVRYH10VLg7zagZqcYTc2RSp_0V2sM)

### Programátoři

1. Souris ET - Prakticky ano, akorát teda uživateli nedovoluješ používat wildcards, respektive z toho co ti zadá za kritéria uživatel jako vstup, vždycky nejdřív ošetříš a přizbůsobíš SQL dotaz tak aby se ti tam od uživatele nemohl dostat žádný bordel.

Spouštět sql dotaz přímo s uživatelským vstupem si přímo říká o SQL Injection.

- for more info see downloaded file: [SQL Injection\\_W3Schools\\_11p\\_2021.pdf](#)

2. Já 2 Souris ET - Děkuji za odpověď. Tedy vlastně nepoužíváte LIKE, ale user's input filujete přes variable kterou ošetříte podmínkama?

2.1 Souris ET - LIKE se používá, ale nepoužiješ ho tak že bys uživatele nechal zadat celý jeho parametr, místo toho uživateli nabídneš jiný způsob "filtrace" a ty si ověřil že ten vstup než ho umístid jako parametr toho LIKE operatoru neobsahuje škodlivý kód.

Tzn kdyby ti někdo do toho textového pole napsal "%;SELECT \* FROM users;" mohl by se dostat třeba k uživatelům, samozřejmě nemusí tam dát select ale drop all a tam už by to byl stoprocentně problém.

2.2 Josef Širůčka - Nene, SqlCommand to delas sam, když volas jakekoliv sql pres Proceduru a davas promenne jako parametry te procedury.

2.3. Já 2 Josef Širůčka - takže nefiltrujete user's input?

...

2.5. Josef Širůčka - Nemusíš filtrovat, pokud nepoužíváš string + string + string. čili když použiješ stored proceduru - pak se nemusí bát útoku přes injection. Nesmíš prostě nikde tvořit dotazy sčítáním stringů.

...

2.x Vít SerjožaŠutr Peprníček - Nemusí být stored procedure stačí použít Prepared Statement v běžném SQL.

## CH12\_APPLY THE CONCEPTS LESSON

### CH12\_A1 - add a Calculated Field to a Dataset by modifying the default SELECT statement / using Query Builder e.g. with 06.Ellington Solution

- > a **calculated field**: = a field whose values are the result of a calculation involving 1 or more of the dataset's existing fields
  - although included in the dataset, it is not stored in the database - instead, it is calculated each time the dataset is accessed
  - enclose the name in **square brackets [ ]** when it contains special characters like space char, ... etc - e.g. **[Total Sales]**
  - to add to a dataset:
    - modify the default SELECT statement, or
    - use the **Query Builder**: - in **Grid pane**:

-> and add new Column: **Jacksonville + Miami + Tampa**  
-> change default alias **Expr1** to **Total**

- in the next set of steps, you will create a **calculated field** for the **Ellington Company application**

- the company has 3 stores, which are located in the following cities in Florida: Jacksonville, Miami, and Tampa
- the company records each store's monthly sales amount in the **Sales2019** table, which is contained in the **Ellington.mdf** database
- the sales amounts for the first six months of 2019 are shown in: **Figure 12-40**

- in addition to displaying the table data shown below, the application will also display the total of each month's sales

<- to accomplish this, you will need to create a **calculated field** that adds together the monthly sales amounts for each of the 3 stores

|   | Name         | Data Type | Allow Nulls              | Default |
|---|--------------|-----------|--------------------------|---------|
| 1 | Month        | int       | <input type="checkbox"/> |         |
| 2 | Jacksonville | int       | <input type="checkbox"/> |         |
| 3 | Miami        | int       | <input type="checkbox"/> |         |
| 4 | Tampa        | int       | <input type="checkbox"/> |         |

Figure 12-40 Sales2019 table in the Ellington.mdf database

| Month | Jacksonville | Miami | Tampa |
|-------|--------------|-------|-------|
| 1     | 67000        | 45000 | 43500 |
| 2     | 69500        | 43600 | 47250 |
| 3     | 65900        | 46500 | 45000 |
| 4     | 64250        | 47000 | 46750 |
| 5     | 63000        | 45000 | 49500 |
| 6     | 62600        | 46700 | 50200 |

- to create a **calculated field** by modifying the default query **SELECT** statement: example with **06.Ellington Solution**

-> open the: ...VB2017\Chap12\Exercise\06.Ellington Solution\Ellington Solution.sln

-> open the windows: **Designer**, **Solution Explorer**, and **Data Sources**

<- as you can see, The **Ellington Company** application is already connected to the **Ellington.mdf** database, and the **EllingtonDataSet.xsd** is already created

**step 1** - open the **DataSet Designer** window:

**Figure 12-41**

-> **Solution Explorer** window / **EllingtonDataSet.xsd** => Rclick: **Open**

or -> **Data Sources** window / **EllingtonDataSet** => Rclick: **Edit DataSet with Designer**

<- notice the Month field is a **primary key field**

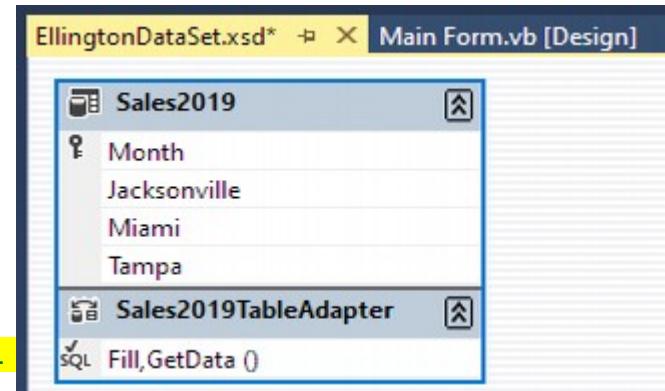


Figure 12-41

**step 2.1** - open/start the **TableAdapter Configuration Wizard**

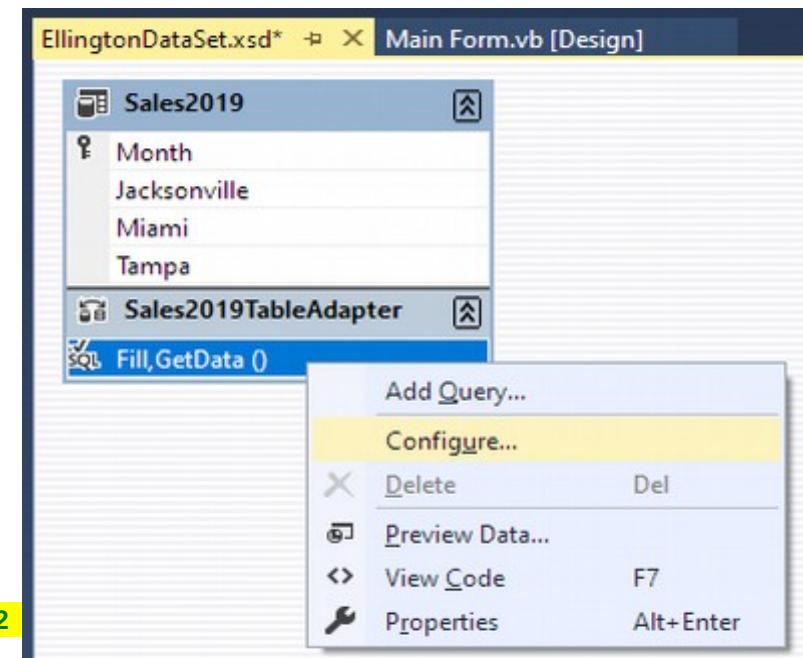
-> EllingtonDataSet.xsd Designer window / line: **Fill,GetData()** => Rclick: **Configure...**

**Figure 12-42**

<- important: do not confuse **TableAdapter Configuration Wizard** with a previous **TableAdapter Query Configuration Wizard** from:

**CH12\_F2** - how to create a **Query** matching specific criteria, using **Query Builder** dialog box - several examples with **01.Oscars Solution-SELECT**

which was started/opened by: -> **OscarsDataSet.xsd** Designer window / line **WinnersTableAdapter** / Rclick: **Add / Query...**



**Figure 12-42**

- the screen: **Enter a SQL Statement** appears

**Figure 12-43**

<- notice the box: **What data should be loaded into the table?**

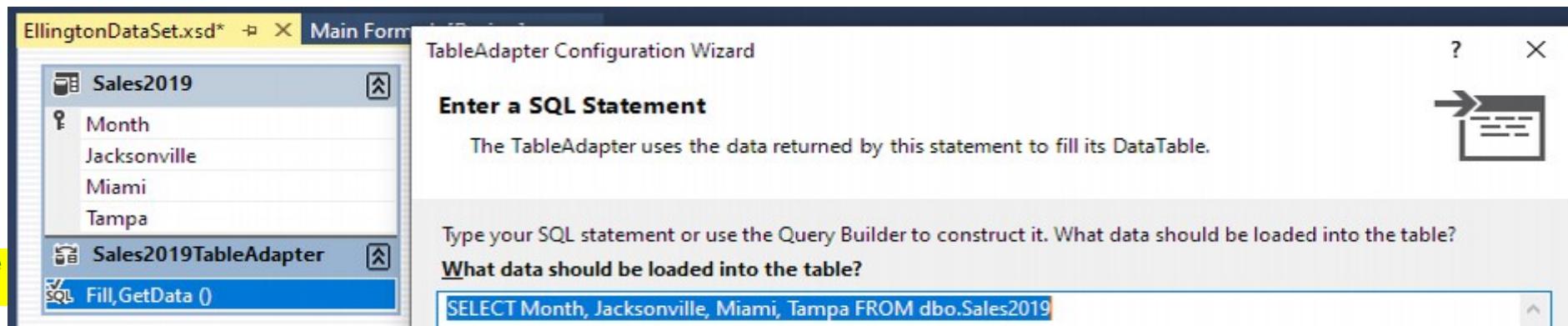
<- notice that the box contains the **default query**: `SELECT Month, Jacksonville, Miami, Tampa FROM dbo.Sales2019`

- the **default query**: 1). selects all of the fields and records from the table, and

2). is automatically executed when the **frmMain\_Load** procedure invokes/calls the **Sales2019TableAdapter** object's **Fill** method

`Me.Sales2019TableAdapter.Fill(Me.EllingtonDataSet.Sales2019)`

**Figure 12-43**



**Figure 12-43**

**step 2.2**

- you will need to modify this default statement to include a **calculated field**
- in this case, the **calculated field** in each record should contain the total sales made in that month - and will be named: Total

-> modify the default **SELECT** statement:

like:

```
SELECT Month, Jacksonville, Miami, Tampa FROM dbo.Sales2019
```

```
SELECT Month, Jacksonville, Miami, Tampa, Jacksonville + Miami + Tampa AS Total FROM dbo.Sales2019
```

Figure 12-43

Figure 12-44

<- the modification tells the computer to:

1. **add together the values** contained in a record's fields: Jacksonville, Miami, Tampa
2. and **create a field named Total** and **store** the sum in that field

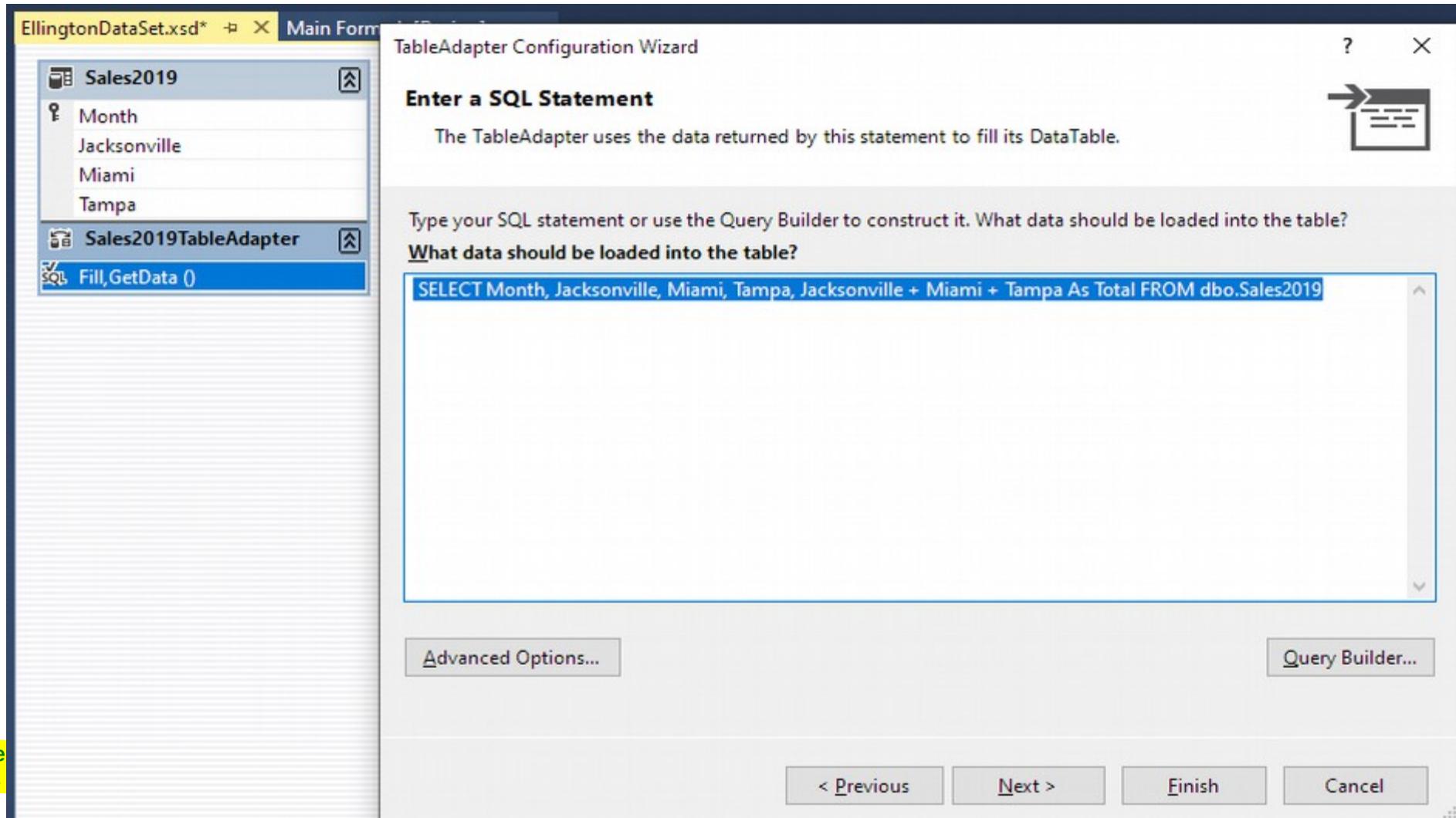


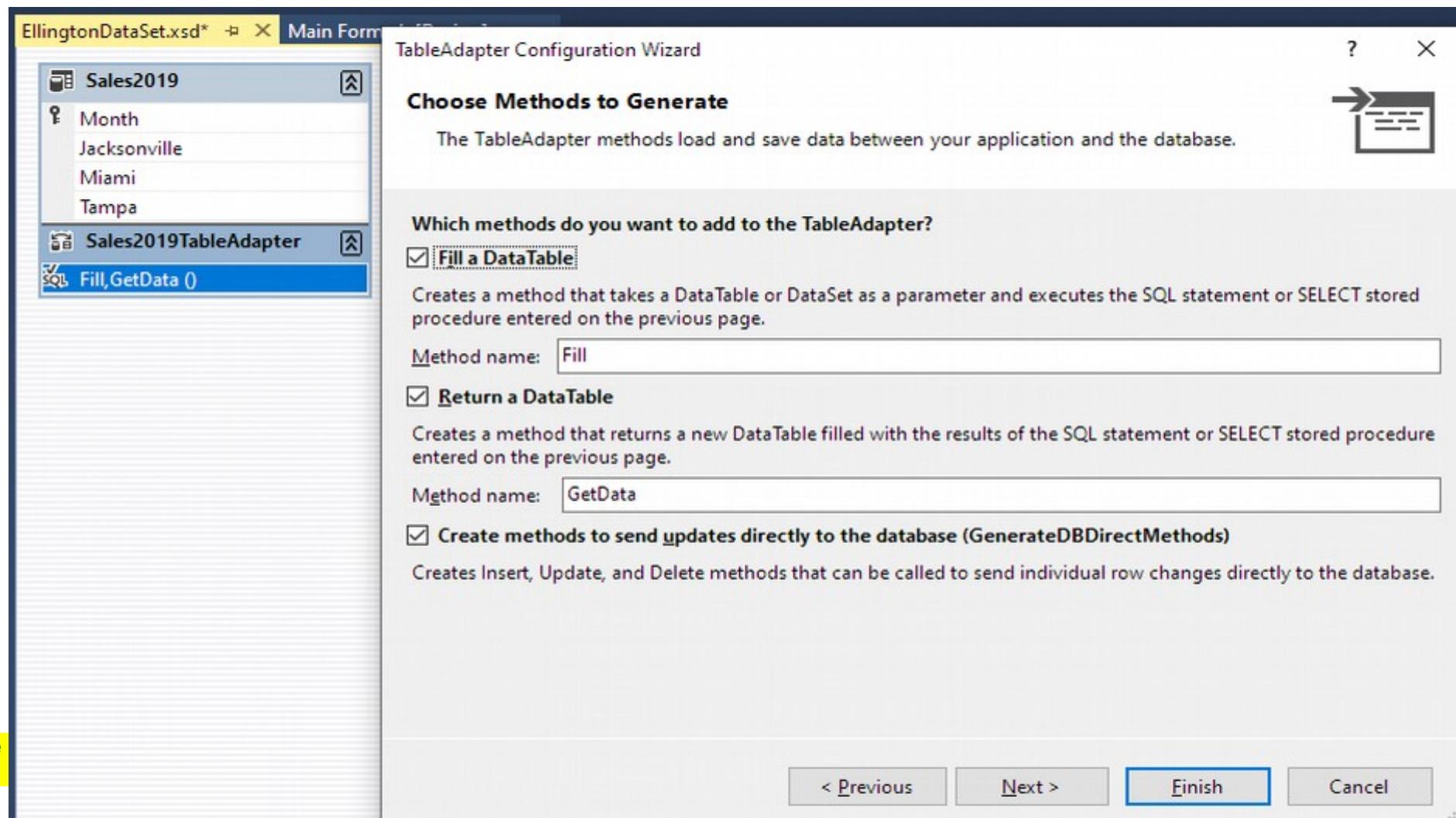
Figure  
12-44

-> click the button **Next >** to display the screen: **Choose Methods to Generate**

**step 2.3**

- the screen: **Choose Methods to Generate** opens **Figure 12-45**

- the statement: `SELECT Month, Jacksonville, Miami, Tampa, Jacksonville + Miami + Tampa As Total FROM dbo.Sales2019` should be the default query **SELECT** statement, so you will associate it with the default **Fill** and **GetData** methods
- > therefore, there is no need to change anything



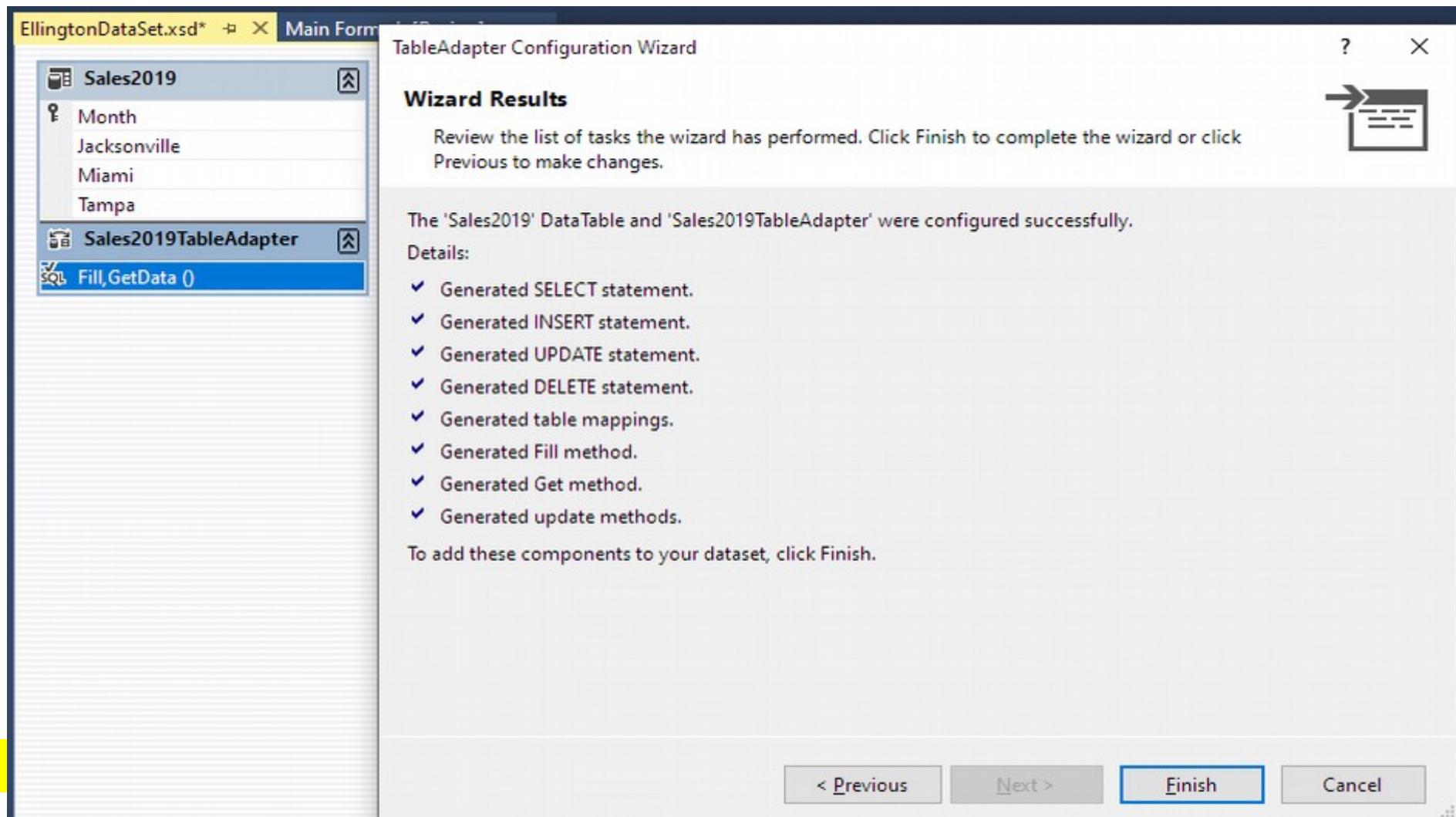
**Figure  
12-45**

-> click the button **Next >** to display the screen: **Wizard Results**

**step 2.4** - the screen: **Wizard Results** opens

**Figure 12-46**

<- here you can verify that all the steps were successful and you did not forget anything



**Figure  
12-46**

-> click the button **Finish** to end the **TableAdapter Configuration Wizard** and return to **DataSet Designer** window named: **EllingtonDataSet.xsd**

step 3 <- notice in window: **Data Sources** the new field named Total has been added to the table **Sales2019** in the **EllingtonDataSet** **Figure 12-47**

-> save the solution

-> you can close the **DataSet Designer** window

The screenshot shows the **Data Sources** window on the left with the **EllingtonDataSet** expanded, revealing the **Sales2019** table. Inside the table, the fields **Month**, **Jacksonville**, **Miami**, **Tampa**, and **Total** are listed. To the right, the **EllingtonDataSet.xsd\*** file is open, displaying the **Sales2019** table with the same five fields. Below the table, the **Sales2019TableAdapter** is shown with its **Fill** and **GetData** methods.

Figure 12-47

step 4 - you will create a bound control by dragging to an existing **DataGridView** control

-> window: **Data Sources** / **EllingtonDataSet** / click the table: **Sales2019** => drag it to the window **Designer** / already created **DataGridView** control

The screenshot shows the **Main Form.vb [Design]** window. A **DataGridView** control is present, displaying two columns: **Month** and **Jacksonville**. A single row is visible, starting with an asterisk (\*) in the first column and "Jacksonville" in the second column. On the left, the **Data Sources** window is open, showing the **EllingtonDataSet** and its **Sales2019** table.

Figure 12-48

**step 5.1** - now you will set some properties for the **DataGridView** control  
-> in **Designer** window => select the **DataGridView1** control & in window **Properties / AutoSizeColumnsMode** => change from default **None** to: = **Fill**  
<- notice that all of the table fields are visible now, including the new field named Total created in: **step 2.2**

**step 5.2** - you will set some properties using the **DataGridView Tasks** box  
-> in **Designer** window / **DataGridView1** control => click the arrow to open the: **DataGridView Tasks** box  
-> **DataGridView Tasks**: => click the: **Dock in Parent Container**  
<- notice that the control filled all of the available space in the **frmMain** form  
-> **DataGridView Tasks**: => deselect the check boxes: **Enable Adding**, **Enable Editing**, and **Enable Deleting**

-> **DataGridView Tasks**: => click: **Edit Columns...** to open the **Edit Columns** screen  
-> **Edit Columns**: => click the button **Alphabetical** to display the properties in that order, rather than by the **Categorized** order

- you will change/set the **AutoSize** property for the primary key field: Month:

-> **Edit Columns**: field: Month / property: **AutoSizeMode = NotSet** => change to: = **ColumnHeader**

- you will change/set the **Format** and **Alignment** properties for the cells: Jacksonville, Miami, Tampa, and Total:

-> **Edit Columns**: field Jacksonville / property: **DefaultCellStyle = DataGridViewCellStyle {}** => click the ellipsis button to open: **CellStyle Builder**  
-> **CellStyleBuilder**: Behavior / Format => click the ellipsis button to open: **Format String Dialog** box  
-> **Format String Dialog**: **Format type = No Formatting** => change to: = **Numeric**  
-> **Format String Dialog**: **Decimal places = 2** => change to: = **0**  
-> **Format String Dialog**: click the button **OK** to close  
-> **CellStyleBuilder**: Layout / Alignment = **NotSet** => change to: = **MiddleRight**  
-> **CellStyleBuilder**: click the button **OK** to close

<- notice the property value changed: **DefaultCellStyle = DataGridViewCellStyle {Format=N0, Alignment=MiddleRight}**

-> when you set the cells Jacksonville, Miami, Tampa, and Total: you can close the **Edit Columns** dialog box by clicking the button **OK**

**step 6** -> save the solution and then start and test the application

<- notice the column/field **Total** contains the total of the sales amounts in each row/record

|   | Month  | Jacksonville | Miami  | Tampa   | Total |
|---|--------|--------------|--------|---------|-------|
| 1 | 67,000 | 45,000       | 43,500 | 155,500 |       |
| 2 | 69,500 | 43,600       | 47,250 | 160,350 |       |
| 3 | 65,900 | 46,500       | 45,000 | 157,400 |       |
| 4 | 64,250 | 47,000       | 46,750 | 158,000 |       |
| 5 | 63,000 | 45,000       | 49,500 | 157,500 |       |
| 6 | 62,600 | 46,700       | 50,200 | 159,500 |       |

Figure 12-49

-> close the application and then close the solution

**CH12\_A2 - use the SQL Aggregate Functions like AVG, COUNT, MAX, MIN, SUM** = return a single value from a group of values - basic info & e.g. using **SUM**:  
 create additional **TableAdapter** in the **DataSet**, whose modified default **SELECT** statement  
 will **create a new field** containing a **SUM** of the cells - e.g. with **07.Ellington Solution-Aggregate**

-> **aggregate functions:**

- = several functions you can use when querying a dataset
- = from a group of values return a single value (aggregate = úhrnný)
- AVG** = calculates the average of a group of selected values
- COUNT** = counts how many rows are in a particular column
- MAX** = returns the highest value in a particular column
- MIN** = returns the lowest value in a particular column
- SUM** = adds together all the values in a particular column

source: web

- = returns the average of the values in a group
- = returns the number of values in a group
- = returns the maximum value in a group
- = returns the minimum value in a group
- = returns the sum of the values in a group

source: book

- in the next set of steps, you will use the **SUM aggregate function** in a slightly different version of the **Ellington Company application** from the previous section
- to use the **SUM** aggregate function: example with **07.Ellington Solution-Aggregate**

-> open the: ...VB2017\Chap12\Exercise\07.Ellington Solution-Aggregate\Ellington Solution.sln

-> open the windows: **Designer**, **Solution Explorer**, and **Data Sources**

<- as you can see, The **Ellington Company** application is already connected to the **Ellington.mdf** database, and the **EllingtonDataSet.xsd** is already created

<- notice in window **Designer**

- the application is the same as the one in the previous section: **CH12\_A1 - add a Calculated Field to a Dataset ...**
- the application also display the total of each month's sales in a **calculated field** named **Total**
- but with a **3** extra labels which will display the total sales made in each of the **3** stores during the first six months of 2019

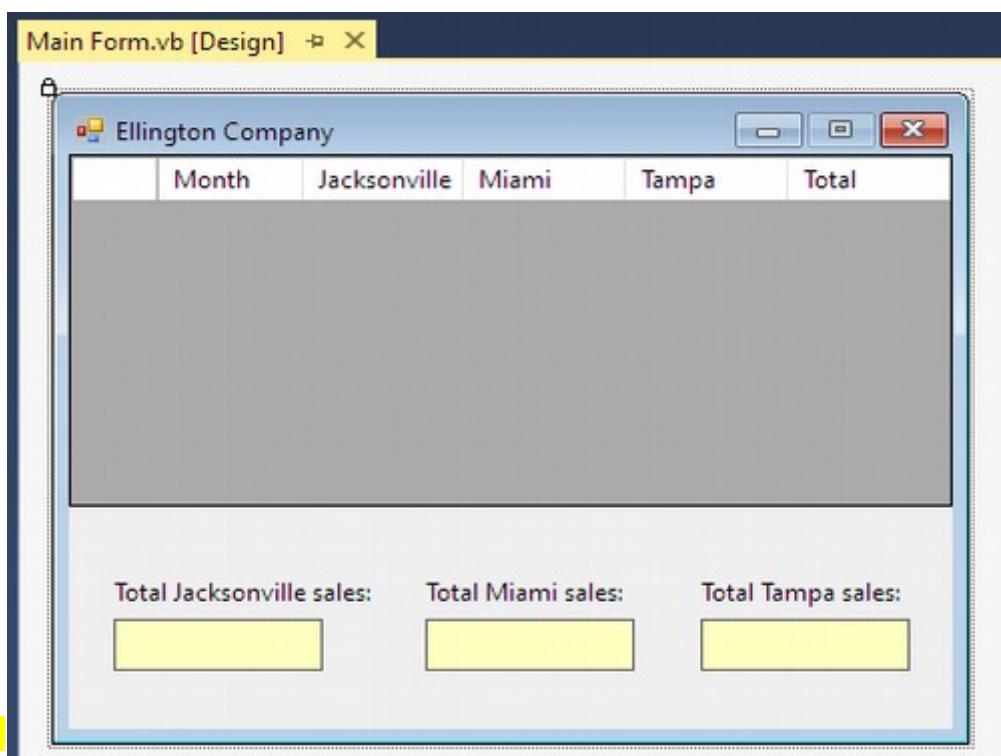


Figure 12-50

**step 1** - open the DataSet Designer window: **Figure 12-51**

-> Solution Explorer window / EllingtonDataSet.xsd => Rclick: Open

or -> Data Sources window / EllingtonDataSet => Rclick: Edit DataSet with Designer  
<- notice the Month field is a primary key field

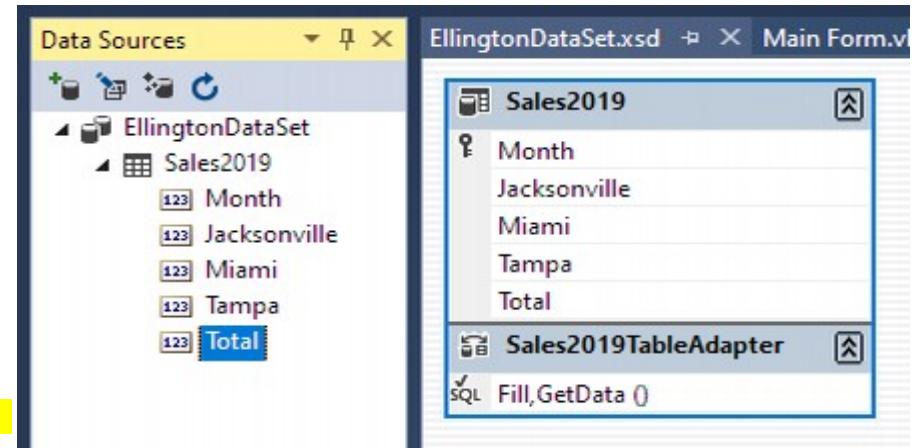
<- notice in window DataSet Designer named: EllingtonDataSet.xsd:

- the Sales2019TableAdapter is associated with the default SELECT statement, which displays the appropriate data in the DataGridView control

SELECT Month, Jacksonville, Miami, Tampa, Jacksonville + Miami + Tampa As Total FROM dbo.Sales2019

<- notice in window Data Sources: dataset EllingtonDataSet / table Sales2019:

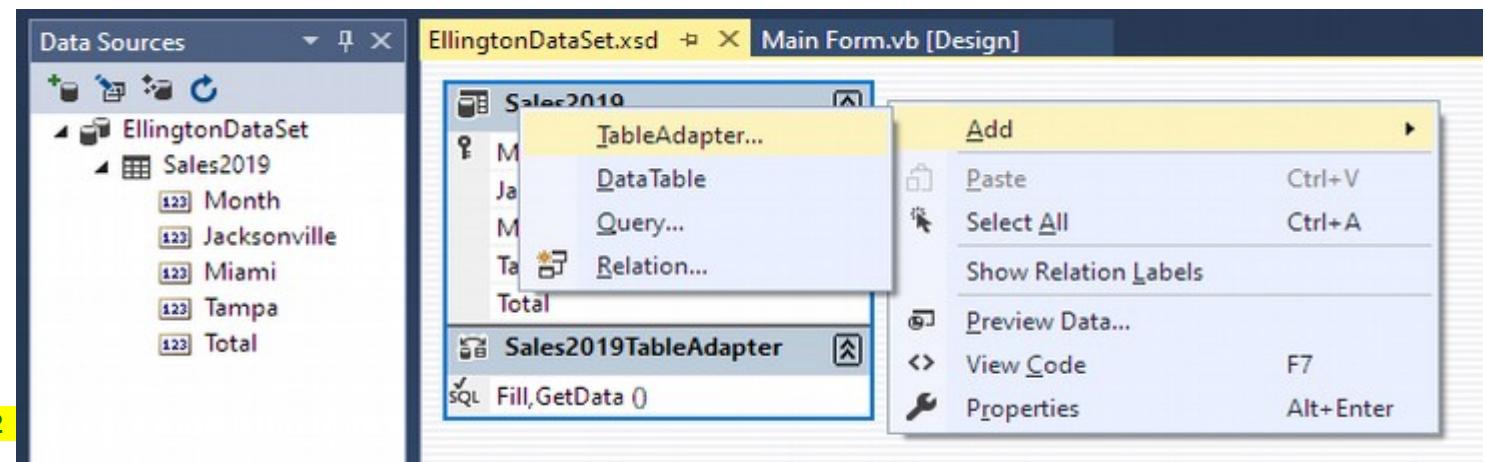
- calculated field Total created by the default SELECT statement associated with a methods: Fill,GetData ()



**step 2** - now, you will add a second TableAdapter to the project, which will be associated with a **SELECT** statement that uses the **SUM aggregate function** to total each store's sales amounts

-> window **Dataset Designer** named **EllingtonDataSet.xsd** => Rclick an empty area and select: Add / TableAdapter... **Figure 12-52**

- the **TableAdapter Configuration Wizard** opens



**step 3.1** - the **TableAdapter Configuration Wizard** with a 1st screen: **Choose Your Data Connection** opens

<- notice the box: **Which data connection...** contains the **EllingtonConnectionString** - the new **TableAdapter** can use the same connection

-> click the button **Next >**

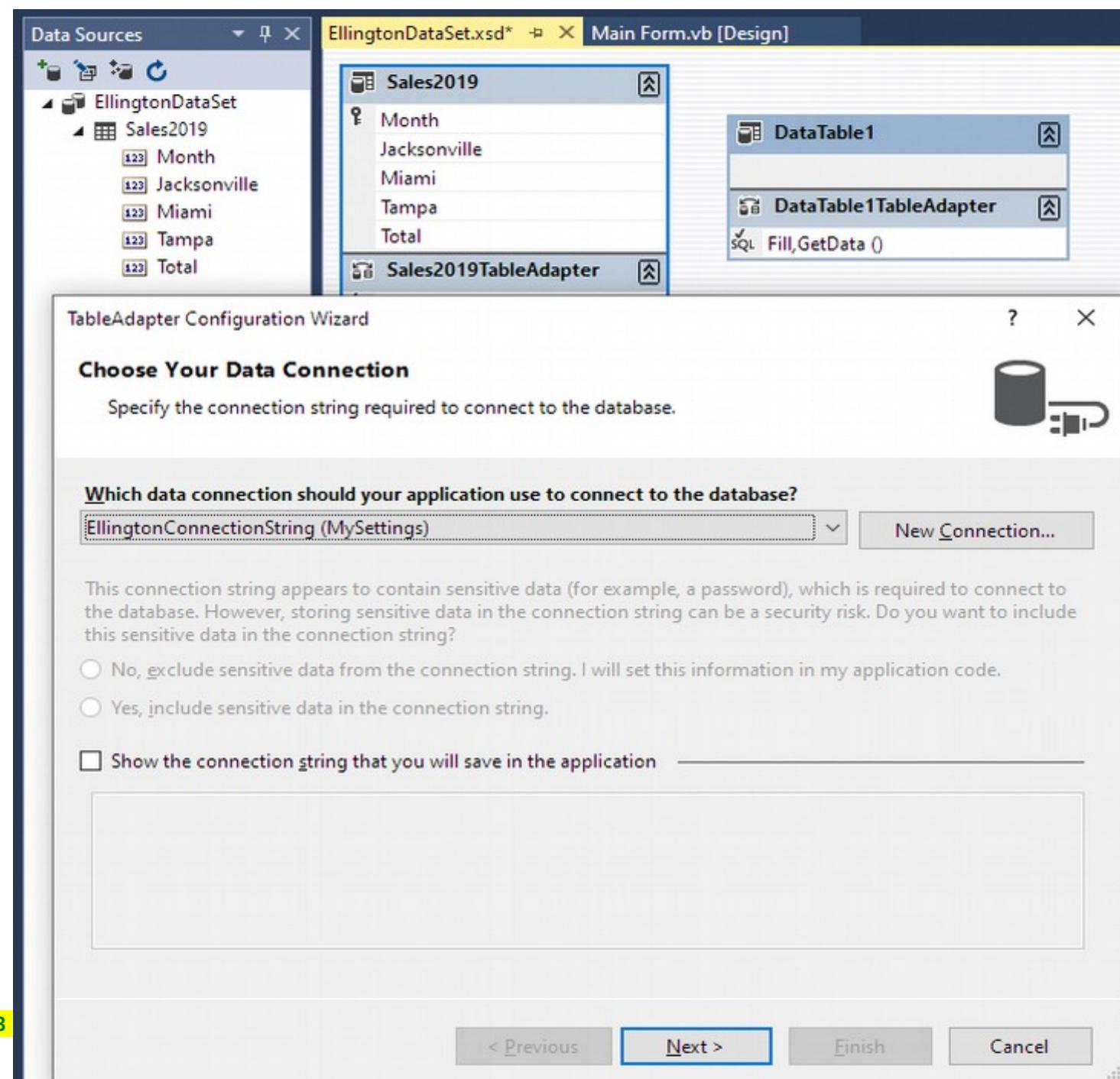


Figure 12-53

**step 3.2** -> 2nd screen **Choose a Command Type** => verify the radio button **Use SQL statements** is selected & click the button **Next >**

**step 3.3** - the 3rd screen **Enter a SQL Statement** opens

-> enter the **SELECT** statement containing the aggregate function **SUM** shown below: **Figure 12-54**

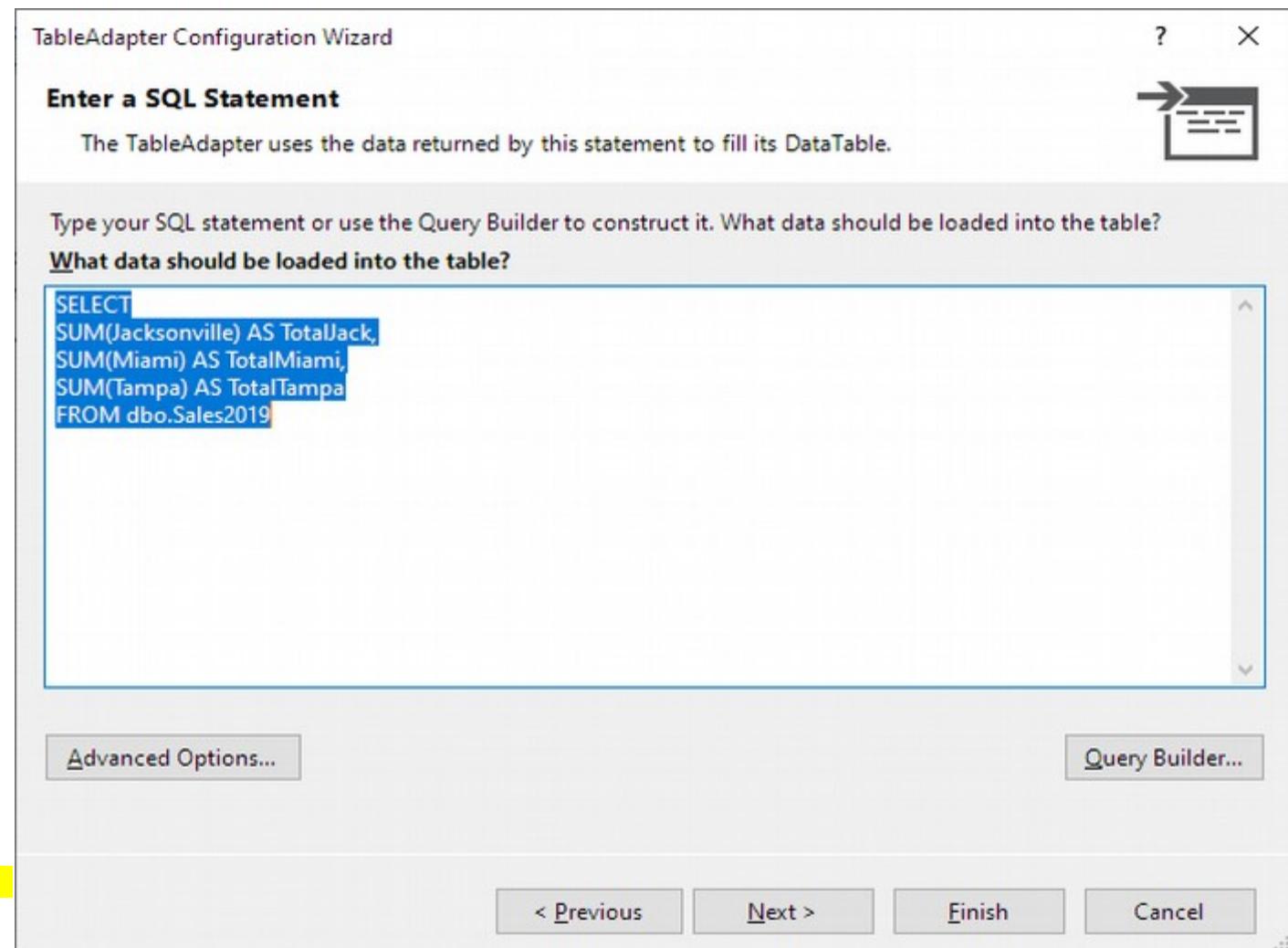
```
SELECT SUM(Jacksonville) AS TotalJack, SUM(Miami) AS TotalMiami, SUM(Tampa) AS TotalTampa FROM dbo.Sales2019
```

- the statement tells the computer to:

1. **SELECT**
2. **SUM(Jacksonville) AS TotalJack**,
3. **SUM(Miami) AS TotalMiami**,
4. **SUM(Tampa) AS TotalTampa**
5. **FROM dbo.Sales2019**

1. create a query
2. add together all of the values in the Jacksonville field (group) and store the sum in the TotalJack field, and
3. add together all of the values in the Miami field (group) and store the sum in the TotalMiami field, and
4. add together all of the values in the Tampa field (group) and store the sum in the TotalTampa field
5. use a table **dbo.Sales2019**

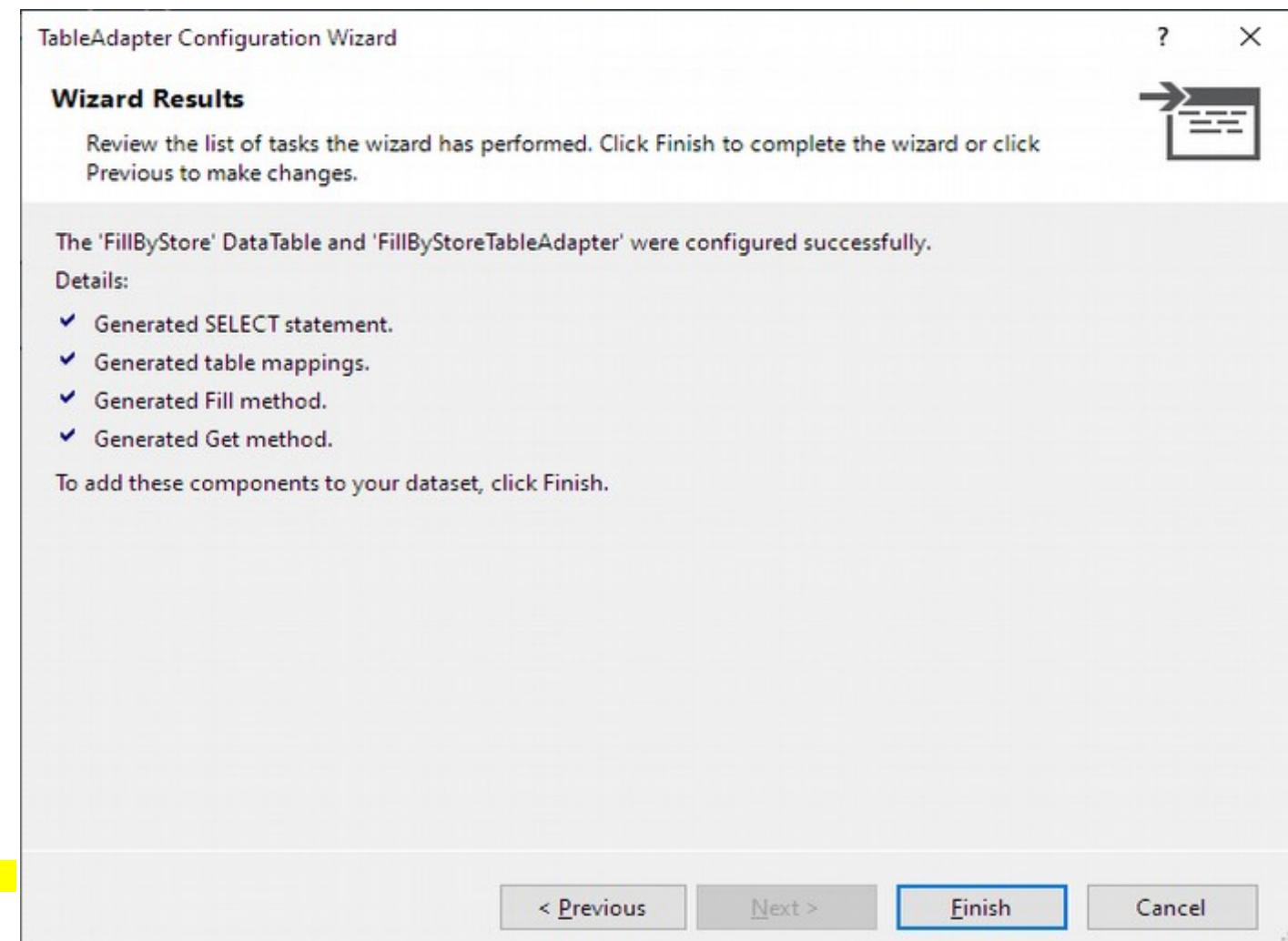
-> click the button **Next >**



**Figure 12-54**

**step 3.4**

- the 4th screen **Choose Methods to Generate** opens
- here you can choose to include and rename methods to load and save data between your application and the database
  - > **Fill a DataTable** method - verify the check box is selected & change the default method's name **Fill** to a more meaningful: **FillByStore**
  - > **Return a DataTable** method - verify the check box is selected & change the default method's name **GetData** to a more meaningful: **GetDataByStore**
- > click the button **Next >** to display the last screen: **Wizard Results** **Figure 12-55**
  - > if everything is ok, click the button **Finish** to close the **TableAdapter Configuration Wizard**



step 4 <- notice in **DataSet Designer** window appears a new **DataTable** with a default names: **DataTable1** & its **DataTable1TableAdapter** Figure 12-56

- we gonna change the default **DataTable1** name to more meaningful one:

-> window **DataSet Designer** named **EllingtonDataSet.xsd**:

-> line **DataTable1** => Rclick and choose **Rename** & type: **Store2019** & press Enter key Figure 12-57

-> line **DataTable1TableAdapter** => Rclick and choose **Rename** & type: **Store2019TableAdapter** & press Enter key Figure 12-57

<- notice in window **Data Sources** - the default name **DataTable1** changed to your **Store2019**

-> save the solution, you can close the window **DataSet Designer**

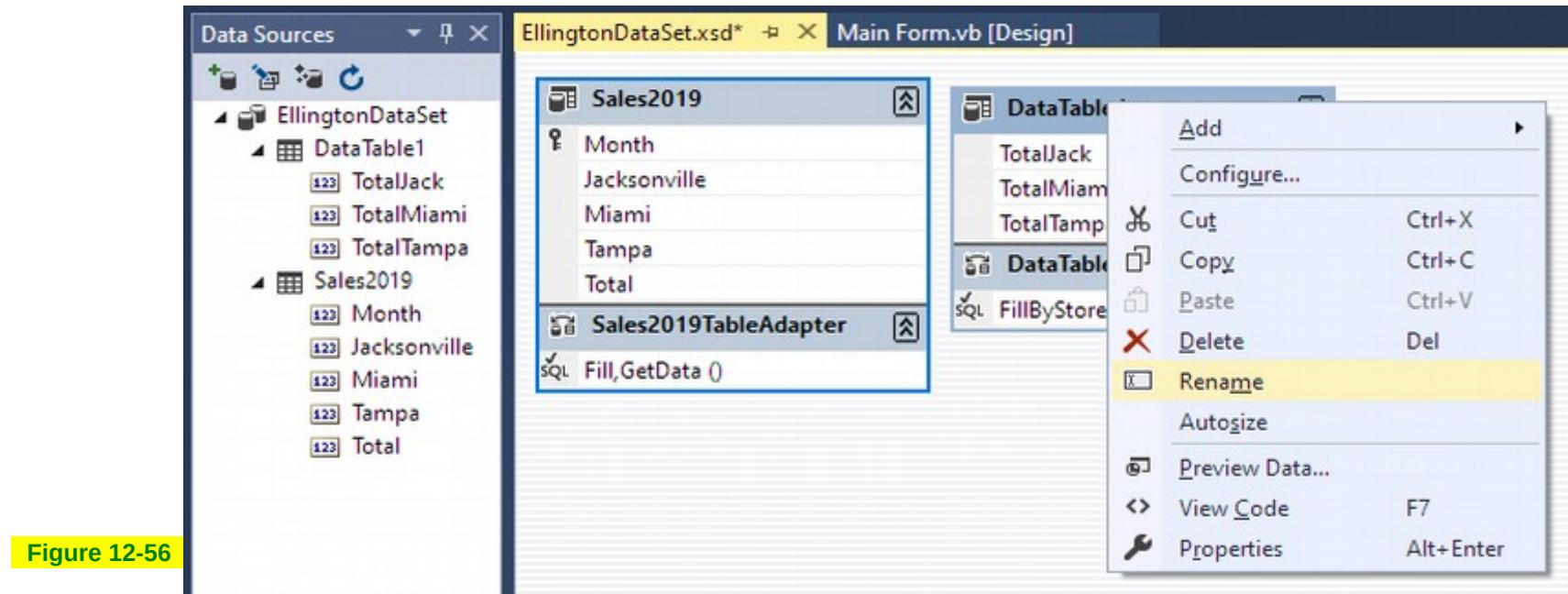


Figure 12-56

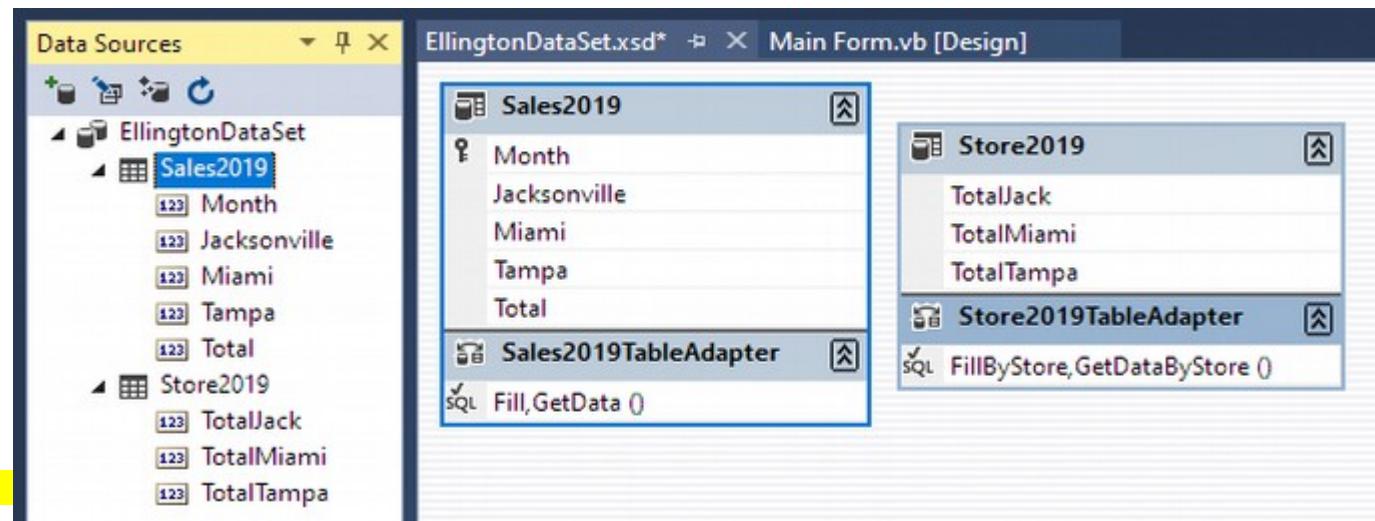
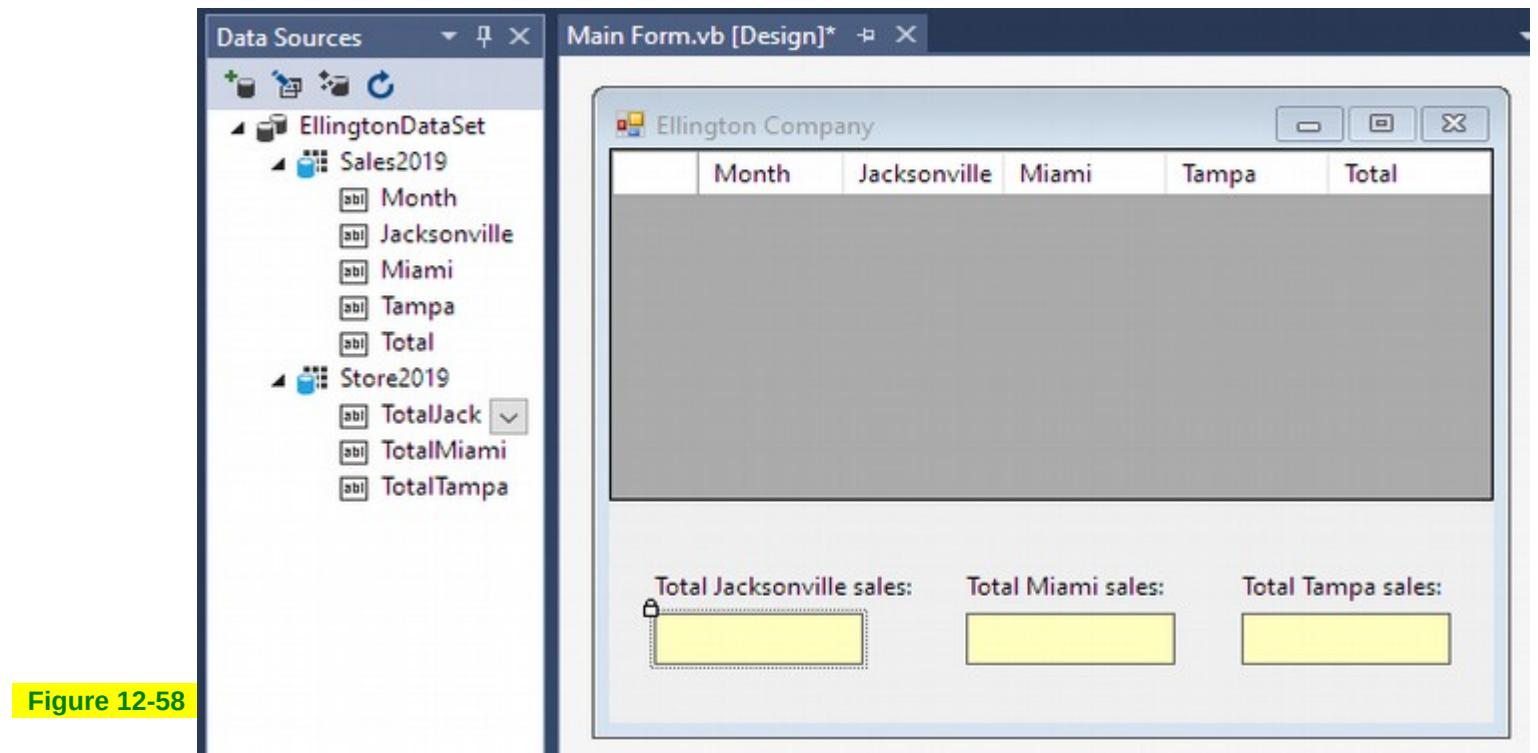
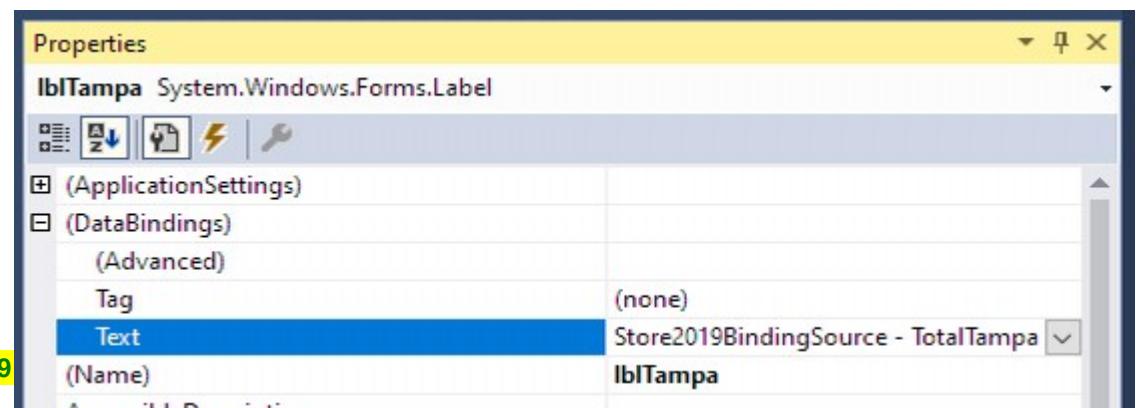


Figure 12-57

**step 5**

- you will associate the fields from a new Table named **Store2019**, with a label controls on GUI by dragging: **Figure 12-58**
- > window **Data Sources**: **EllingtonDataSet** / node **Store2019** -> drag field TotalJack to window **Designer**: label named (**lblJacksonville**)  
<- for verification, you can look at the **lblJacksonville**'s window **Properties** / node **(DataBindings)** / **Text** = **Store2019BindingSource - TotalJack** **Figure 12-59**
- > window **Data Sources**: **EllingtonDataSet** / node **Store2019** -> drag field TotalMiami to window **Designer**: label named (**lblMiami**)  
<- for verification, you can look at the **lblMiami**'s window **Properties** / node **(DataBindings)** / **Text** = **Store2019BindingSource - TotalMiami**
- > window **Data Sources**: **EllingtonDataSet** / node **Store2019** -> drag field TotalTampa to window **Designer**: label named (**lblTampa**)  
<- for verification, you can look at the **lblTampa**'s window **Properties** / node **(DataBindings)** / **Text** = **Store2019BindingSource - TotalTampa**

**Figure 12-58****Figure 12-59**

**step 6** -> start and test the application

**Figure 12-60**

- each store's total sales amount appears in its respective label control

<- notice that the labels displays the amounts only as a number -

- without any formatting -> this will be fixed in the next step: **step 7**

-> close the application

**Figure 12-60**

| Ellington Company |       |              |        |        |         |
|-------------------|-------|--------------|--------|--------|---------|
|                   | Month | Jacksonville | Miami  | Tampa  | Total   |
| ▶                 | 1     | 67,000       | 45,000 | 43,500 | 155,500 |
|                   | 2     | 69,500       | 43,600 | 47,250 | 160,350 |
|                   | 3     | 65,900       | 46,500 | 45,000 | 157,400 |
|                   | 4     | 64,250       | 47,000 | 46,750 | 158,000 |
|                   | 5     | 63,000       | 45,000 | 49,500 | 157,500 |
|                   | 6     | 62,600       | 46,700 | 50,200 | 159,500 |

|                           |                    |                    |
|---------------------------|--------------------|--------------------|
| Total Jacksonville sales: | Total Miami sales: | Total Tampa sales: |
| 392250                    | 273800             | 282200             |

- step 7** - professionalize your application's GUI by formatting the labels with a **dollar sign** & a **thousands separator** Figure 12-61 & Figure 12-62
- > **Designer** window => click the **lblJacksonville** label control & window **Properties** / node (**DataBindings**) / (**Advanced**) => click the ellipsis button & window **Properties** / node (**DataBindings**) / (**Advanced**) => click the ellipsis button to open the dialog box: **Formatting and Advanced Binding**
  - > dialog box **Formatting and Advanced Binding** / **Format type** => select: **Currency**
  - > dialog box **Formatting and Advanced Binding** / **Decimal places** => change to: **0**
  - > click the button **OK** to close the dialog box
  - > format the other label controls: **lblMiami**, and **lblTampa** the same way

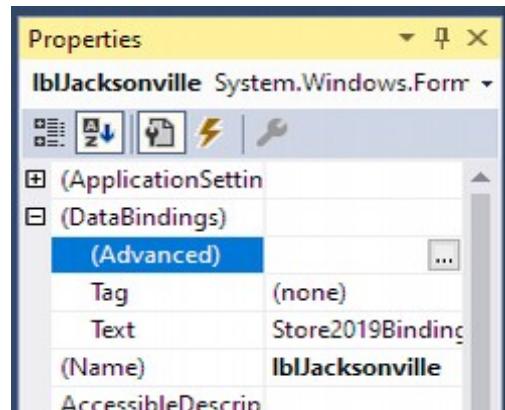


Figure 12-61

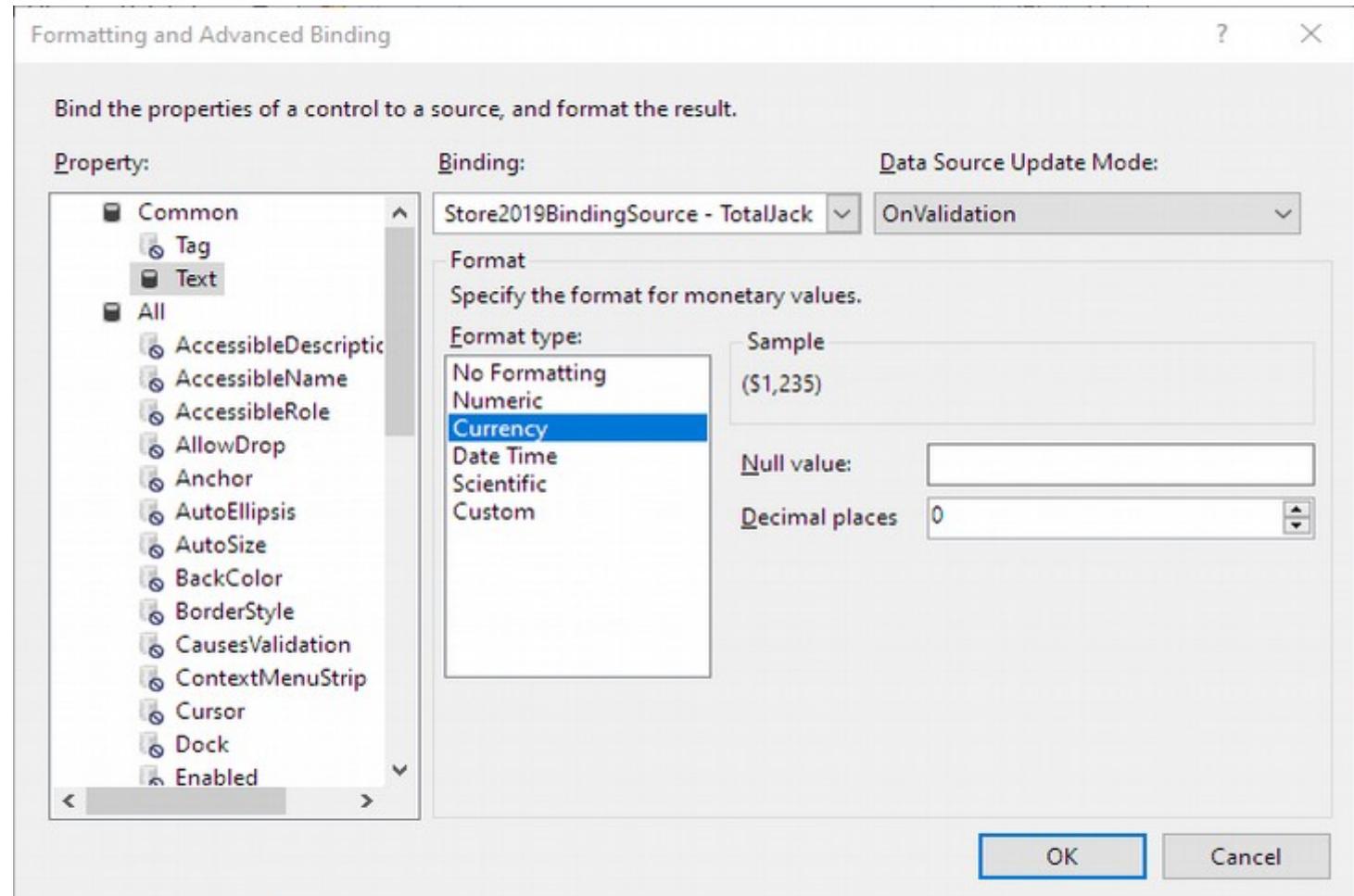


Figure 12-62

step 8 -> start and test the application

<- compare the unformatted result with a formatted one:

|   | Month | Jacksonville | Miami  | Tampa  | Total   |
|---|-------|--------------|--------|--------|---------|
| ► | 1     | 67,000       | 45,000 | 43,500 | 155,500 |
|   | 2     | 69,500       | 43,600 | 47,250 | 160,350 |
|   | 3     | 65,900       | 46,500 | 45,000 | 157,400 |
|   | 4     | 64,250       | 47,000 | 46,750 | 158,000 |
|   | 5     | 63,000       | 45,000 | 49,500 | 157,500 |
|   | 6     | 62,600       | 46,700 | 50,200 | 159,500 |

Figure 12-60 - unformatted

|   | Month | Jacksonville | Miami  | Tampa  | Total   |
|---|-------|--------------|--------|--------|---------|
| ► | 1     | 67,000       | 45,000 | 43,500 | 155,500 |
|   | 2     | 69,500       | 43,600 | 47,250 | 160,350 |
|   | 3     | 65,900       | 46,500 | 45,000 | 157,400 |
|   | 4     | 64,250       | 47,000 | 46,750 | 158,000 |
|   | 5     | 63,000       | 45,000 | 49,500 | 157,500 |
|   | 6     | 62,600       | 46,700 | 50,200 | 159,500 |

Figure 12-63 - formated to **Currency**, and 0 decimal places

```
1 Public Class frmMain
2     Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles MyBase.Load
3         'TODO: This line of code loads data into the 'EllingtonDataSet.Store2019' table. You can move, or remove it, as needed.
4         Me.Store2019TableAdapter.FillByStore(Me.EllingtonDataSet.Store2019)
5         'TODO: This line of code loads data into the 'EllingtonDataSet.Sales2019' table. You can move, or remove it, as needed.
6         Me.Sales2019TableAdapter.Fill(Me.EllingtonDataSet.Sales2019)
7     End Sub
8     End Class
```

## CH12\_Summary

01a. a database query - query - allows you to retrieve specific information from a database

01b. use the SQL **SELECT** statement to query a database

```
SELECT fieldList FROM dbo.table WHERE condition ORDER BY fieldName DESC
```

|          |                |
|----------|----------------|
| SELECT   | fieldList      |
| FROM     | table          |
| WHERE    | condition      |
| ORDER BY | fieldName DESC |

01c. use the **SELECT** statement's **WHERE** clause to limit the records you want to access

- must use a single quotes '' when comparing text field with a literal constant

e.g.3 ...**WHERE** Picture = 'Argo'

- no need for single quotes '' when comparing number field with a lit. const.

e.g.2 ...**WHERE** Year >= 2014

- text comparisons are not case sensitive

e.g.3 ...**WHERE** Picture = 'argo'

01d. use the SQL **LIKE** operator to compare text values in a **SELECT** statement's **WHERE** clause

- the operator can be combined with the following 2 wildcards: % = 0 or more characters

\_ = 1 character

01e. you use the **SELECT** statement's **ORDER BY** clause to sort the selected records

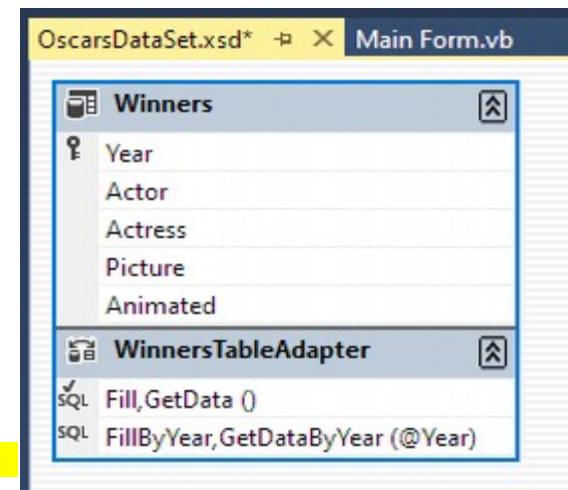
**Figure 12-1** operators for the **WHERE** clause's condition

|                |                                                                                                                                                     |
|----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------|
| =              | - equal to                                                                                                                                          |
| <>             | - not equal to                                                                                                                                      |
| >              | - greater than                                                                                                                                      |
| >=             | - greater than or equal to                                                                                                                          |
| <              | - less than                                                                                                                                         |
| <=             | - less than or equal to                                                                                                                             |
| <b>AND</b>     | - all subconditions must be <b>true</b> for the compound condition to evaluate to <b>True</b>                                                       |
| <b>OR</b>      | - only one of the subconditions needs to be <b>true</b> for the compound condition to evaluate to <b>True</b>                                       |
| <b>NOT</b>     | - reverses the <b>truth</b> -value of the condition                                                                                                 |
| <b>LIKE</b>    | - uses a wildcard character to compare text values<br>% = wildcard char for <b>0 or more</b> characters<br>_ = wildcard char for <b>1</b> character |
| <b>IS NULL</b> | - compares a value with a <b>NULL</b> value (empty)                                                                                                 |

2. to open the **DataSet Designer** window: **Figure 12-34**

a). in **Solution Explorer** window, **Rclick** the name of the dataset's schema file with suffix **.xsd** and click: **Open**

b). in **Data Sources** window, **Rclick** the dataset's name and click: **Edit DataSet with Designer**



**Figure 12-34**

3. to start the **TableAdapter Query Configuration Wizard**:

- 1). open the **DataSet Designer** window
- 2). Rclick the **TableAdapter**'s name and choose: **Add / Query...** or **Add Query...**

4. to open the **Query Builder** dialog box:

- 1). start the **TableAdapter Query Configuration Wizard**
- 2). button **Next >** to display the screen: **Choose a Query Type** & button **Next >** to display the screen: **Specify a SQL SELECT statement**
- 3). click the button: **Query Builder...**

5. to create a **Parameter Query** = when you do not know the specific value / the value for querying can be entered later by the user during runtime:

- use a parameter marker @ in place of the criteria's value followed by the name of the field you are querying

e.g. WHERE Year = @Year

- if the **WHERE** clause contains more than **1** parameter marker for the same field, you append a unique number after the field name

e.g. WHERE Year >= @Year1 AND <= @Year2

6. for an application to use any **Query** during runtime, you need to:

- 1). **create** the query - by using the **TableAdapter Query Configuration Wizard**
- 2). **save** the query - by associating the query with one or more methods (Fill a DataTable, Return a DataTable - methods **Fill**, **Get**)
- 3). **invoke** it from code - by entering in procedure the **Fill** method associated with the query

7. you can use the **SELECT** statement to add a **calculated field** to a dataset

e.g. SELECT Month, Jacksonville, Miami, Tampa, Jacksonville + Miami + Tampa As Total FROM dbo.Sales2019

- creates a field name Total, which is a sum of values from fields: Jacksonville, Miami, and Tampa

8. SQL provides the following aggregate functions: **AVG**, **COUNT**, **MAX**, **MIN**, **SUM** - these functions return a single value from a group of values

9. you can use the **DataSet Designer** window to add a new **TableAdapter** to a project

10. to format the data displayed in a bound label control:

- 1). in the control's **Properties** list / node (**DataBindings**) / (**Advanced**) => click the ellipsis button to open the dialog box: **Formatting and Advanced Binding**
- 2). dialog box named **Formatting and Advanced Binding** opens

## CH12\_Key Terms & terminology marked: ->

- % - the percent sign wildcard used along with the **LIKE** operator in a **SELECT** statement's **WHERE** clause
  - represents 0 or more characters
- \_ - the underscore sign wildcard used along with the **LIKE** operator in a **SELECT** statement's **WHERE** clause
  - represents 1 character
- **Aggregate functions** - functions that return a single value from a group of values
  - SQL functions: **AVG**, **COUNT**, **MAX**, **MIN**, **SUM**
- **Calculated field** - a field whose values are the result of a calculation involving 1 or more of the dataset's existing fields
  - although included in the dataset, it is not stored in the database - instead, it is calculated each time the dataset is accessed
- **Database query / query** - a statement used to retrieve specific information from a database
- **LIKE operator** - used with a wildcard character in a **SELECT** statement's **WHERE** clause
- **ORDER BY clause** - used in a **SELECT** statement to sort the selected records
- **Parameter marker** - used in a place of a criteria's value in a parameter query
  - typically used in SQL is the (at) symbol - @ - followed by the name of the field you are querying      e.g. **WHERE Year = @Year**
  - if the **WHERE** clause contains more than 1 parameter marker for the same field, you append a unique number after the field name
    - e.g. **WHERE Year >= @Year1 AND <= @Year2**   /   **WHERE Year >= @Year1 AND Year <= @Year2**
- **Parameter query** - a query that contains a parameter marker in place of a criteria's value
- **Query / database query** - a statement used to retrieve specific information from a database
- **Query Builder dialog box** - used to create a query
- **SELECT statement** - the SQL statement that allows you to specify the fields and records to select,
  - as well as the order in which the fields and records appear when displayed
- **SQL / Structured Query Language** - a set of statements that allows you to access and manipulate the data stored in a database
- **WHERE clause** - used in a **SELECT** statement to limit the records to be selected

- 08.Harken Solution\_EXERCISE 1\_introductory** - testing SELECT statements in **Query Builder** dialog box:  
- SELECT, ORDER BY ... DESC, WHERE, WHERE ...LIKE %, \_
- 09.Incor Solution\_EXERCISE 2\_introductory** - testing SELECT statements in **Query Builder** dialog box:  
- SELECT, ORDER BY ... DESC, WHERE, WHERE ...LIKE %, WHERE *field1* = 'condition1' AND *field2* = *condition2*
- 10.Adalene Solution\_EXERCISE 3\_intermediate** - create several **Queries** and associate them with a methods  
- use the methods in code to filter values displayed in **DataSet**
- 11.Opals Solution\_EXERCISE 4\_intermediate** - create a **calculated field** by modifying the default SELECT statement / using Query Builder,  
using **TableAdapter Configuration Wizard**  
- my discovery: enclose the name in **square brackets [ ]** when it contains special characters like space char, ... etc
- 12.Games Solution\_EXERCISE 5\_advanced** - create several **Queries** and **Parameter Queries**, and associate them with a methods  
- use the method in a code to filter values displayed in **DataSet**
- 13.Games Solution-Total\_EXERCISE 6\_advanced** - 1). add **TableAdapter**  
- 2). create Query using **Aggregate** function  
- 3). drag the **DataColumn** to the label control = create bound control by dragging
- 14.Adalene Solution-TotalSales\_EXERCISE 7\_advanced** - 1). add 3 **TableAdapters**  
- 2). for each, create Query using **Aggregate** function  
- 3). drag each **DataColumn** to a different label control = create bound controls by dragging
- 15.OnYourOwn Solution\_EXERCISE 8** - create **Database & DataSet & Table**  
- create **Calculated field** by modifying default SQL statement  
- use aggregate function **SUM**  
- step by step
- 16.FixIt Solution\_EXERCISE 9** - to the TableAdapter's modified default SQL statement containing **Aggregate** function - added **parameter @**  
**\_Appendix E - Case Projects\_12.Jefferson Realty-CH01-12** - using **.NET Framework 4.8**

## 08.Harken Solution\_EXERCISE 1\_introductory

- testing SELECT statements in **Query Builder** dialog box:  
- SELECT, ORDER BY ... DESC, WHERE, WHERE ...LIKE %, \_

Open the file: ...VB2017\Chap12\Exercise\08.Harken Solution\_EXERCISE 1\_introductory\Harken Solution.sln

The application is already connected to the **Harken.mdf** database, and the **HarkenDataSet** has already been created.

Start the application to view the records contained in the dataset and then stop the application.

Open the **DataSet Designer** window and then start the **TableAdapter Query Configuration Wizard**.

Open the **Query Builder** dialog box. Use the dialog box to test your **SELECT** statements from Steps **a** through **j**.

Note: The **Magazine** table contains 3 fields:

- field **Code** is a Text field
- field **Name** is a Text field
- field **Cost** is a Numeric field

| Code | Name           | Cost    |
|------|----------------|---------|
| EX33 | Fit and Fun    | \$ 3.00 |
| HE25 | Country MD     | \$ 8.00 |
| HE45 | Beautiful You  | \$ 4.00 |
| HE91 | Light Cooking  | \$ 2.50 |
| OU29 | Planting Roses | \$ 6.00 |
| OU36 | Gardening Day  | \$ 3.00 |
| PG10 | C++            | \$ 4.50 |
| PG15 | Java           | \$ 3.75 |
| PG24 | Visual Basic   | \$ 3.50 |
| PG45 | C#             | \$ 3.50 |

#. the default SELECT statement using all the fields.

SELECT Code, Name, Cost FROM Magazine

Write a SQL SELECT statement that selects:

- a). all the fields and arranges the records in descending order by the Cost field.
- b). only the Name and Cost fields from records having a code of PG10.

SELECT Code, Name, Cost FROM Magazine ORDER BY Cost DESC

SELECT Name, Cost FROM Magazine WHERE Code = 'PG10'

<- be carefull that you really type a single quotes - e.g. Libre Office by default changes the **Start quote** and **End quote** => error in SQL  
<- Libre Office fix: in menu bar: **Tools / Autocorrect Options... / tab Localized Options: Single Quotes** => uncheck **Replace** option

- c). only the Name and Cost fields from records having a cost of \$3 or more.

SELECT Name, Cost FROM Magazine WHERE Cost >= 3

- d). the Visual Basic record.

SELECT Code, Name, Cost FROM Magazine WHERE Name = 'Visual Basic'

- e). only the Name field from records whose magazine name begins with the letter C.

SELECT Name FROM Magazine WHERE Name LIKE 'C%'

- f). only the Name field from records whose magazine name contains 2 characters.

SELECT Name FROM Magazine WHERE Name LIKE '\_\_'

<- 2x underscore \_\_

- g). only the Name and Cost fields from records whose cost is from \$4 to \$6, inclusive.

SELECT Name, Cost FROM Magazine WHERE Cost >= 4 AND Cost <= 6

- h). only records for the Code provided by the user.

SELECT Code FROM Magazine WHERE Code = @Code

or

or SELECT Code FROM Magazine WHERE Code LIKE @Code

<-

<- i found out: using a LIKE operator with a String data type allows you using the wildcards during run time: % = wildcard char for **0 or more** characters

\_ = wildcard char for **1** character

<- more info about topic in:

**CH12\_F6 - My Test:** - use **Parameter Query** with a **parameter marker** @ during run time with a **LIKE** operator & allow users to use its wildcards %, \_  
- plus Q & A on several FB programmer's groups about **LIKE** operator  
- e.g. with: **05.Oscars Solution-Save Query\_My Test - Parameter Query and LIKE operator**

- i). only records that cost at least as much as the amount provided by the user.

SELECT Code, Name, Cost FROM Magazine WHERE Cost >= @Cost

- j). only records that cost more than the first amount but less than the second amount, both provided by the user

SELECT Code, Name, Cost FROM Magazine WHERE Cost > @Cost1 AND Cost < @Cost2

```

1  Public Class frmMain
2      Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles MyBase.Load
3          'TODO: This line of code loads data into the 'HarkenDataSet.Magazine' table. You can move, or remove it, as needed.
4          Me.MagazineTableAdapter.Fill(Me.HarkenDataSet.Magazine)
5      End Sub
6  End Class

```

QueryBuilder

| Column                                   | Alias | Table    | Output                              | Sort Type | Sort Order | Filter | O |
|------------------------------------------|-------|----------|-------------------------------------|-----------|------------|--------|---|
| <input checked="" type="checkbox"/> Code |       | Magazine | <input checked="" type="checkbox"/> |           |            |        | ^ |
| <input checked="" type="checkbox"/> Name |       | Magazine | <input checked="" type="checkbox"/> |           |            |        | ^ |
| <input checked="" type="checkbox"/> Cost |       | Magazine | <input checked="" type="checkbox"/> |           |            |        | ^ |

```

SELECT Code, Name, Cost
FROM Magazine

```

| Code   | Name           | Cost |
|--------|----------------|------|
| EX33   | Fit and Fun    | 3.00 |
| HE25   | Country MD     | 8.00 |
| HE45   | Beautiful You  | 4.00 |
| HE91   | Light Cooking  | 2.50 |
| OU29   | Planting Roses | 6.00 |
| OU36   | Gardening Day  | 3.00 |
| PG10   | C++            | 4.50 |
| PG15   | Java           | 3.75 |
| PG24   | Visual Basic   | 3.50 |
| PG45   | C#             | 3.50 |
| * NULL | NULL           | NULL |

[|< | 1 | > | 10 | < | > | ⌂ | ⌃ |]

Execute Query      OK      Cancel

#. the default SELECT statement using all the fields.

```
SELECT Code, Name, Cost FROM Magazine
```

QueryBuilder

| Column                                   | Alias | Table    | Output                              | Sort Type | Sort Order | Filter | O |
|------------------------------------------|-------|----------|-------------------------------------|-----------|------------|--------|---|
| <input checked="" type="checkbox"/> Code |       | Magazine | <input checked="" type="checkbox"/> |           |            |        | ^ |
| <input checked="" type="checkbox"/> Name |       | Magazine | <input checked="" type="checkbox"/> |           |            |        | ^ |
| <input checked="" type="checkbox"/> Cost |       | Magazine | <input checked="" type="checkbox"/> | Descen... | 1          |        | ^ |

```

SELECT Code, Name, Cost
FROM Magazine
ORDER BY Cost DESC

```

| Code   | Name           | Cost |
|--------|----------------|------|
| HE25   | Country MD     | 8.00 |
| OU29   | Planting Roses | 6.00 |
| PG10   | C++            | 4.50 |
| HE45   | Beautiful You  | 4.00 |
| PG15   | Java           | 3.75 |
| PG24   | Visual Basic   | 3.50 |
| PG45   | C#             | 3.50 |
| OU36   | Gardening Day  | 3.00 |
| EX33   | Fit and Fun    | 3.00 |
| HE91   | Light Cooking  | 2.50 |
| * NULL | NULL           | NULL |

[|< | 1 | > | 10 | < | > | ⌂ | ⌃ |]

Execute Query      OK      Cancel

a). all the fields and arranges the records in descending order by the Cost field.

```
SELECT Code, Name, Cost FROM Magazine ORDER BY Cost DESC
```

Query Builder

The screenshot shows the Query Builder interface with the following configuration:

- Selected Columns:** \* (All Columns), Code, Name, Cost
- Table:** Magazine
- Where Clause:** Code = 'PG10'
- Result Preview:** A table showing one record: C++ at cost 4.50.

Below the preview, the generated SQL query is:

```
SELECT Name, Cost
FROM Magazine
WHERE (Code = 'PG10')
```

Buttons at the bottom: Execute Query, OK, Cancel.

b). only the Name and Cost fields from records having a code of PG10.

SELECT Name, Cost FROM Magazine WHERE Code = 'PG10'

Query Builder

The screenshot shows the Query Builder interface with the following configuration:

- Selected Columns:** \* (All Columns), Code, Name, Cost
- Table:** Magazine
- Where Clause:** Cost >= 3
- Result Preview:** A table showing nine records where Cost is 3.00 or more.

Below the preview, the generated SQL query is:

```
SELECT Name, Cost
FROM Magazine
WHERE (Cost >= 3)
```

Buttons at the bottom: Execute Query, OK, Cancel.

c). only the Name and Cost fields from records having a cost of \$3 or more.

SELECT Name, Cost FROM Magazine WHERE Cost >= 3

Query Builder

The screenshot shows the Query Builder window with the following details:

- Selected Columns:** \* (All Columns), Code, Name, Cost
- Table:** Magazine
- Output:** Checkmarks for Code, Name, and Cost.
- Sort Type:** None
- Sort Order:** None
- Filter:** Name = 'Visual Basic'
- SQL Statement:**

```
SELECT Code, Name, Cost
FROM Magazine
WHERE (Name = 'Visual Basic')
```
- Result Preview:** A table showing one record:

|   | Code | Name         | Cost |
|---|------|--------------|------|
| ▶ | PG24 | Visual Basic | 3.50 |
| ● | NULL | NULL         | NULL |
- Buttons:** Execute Query, OK, Cancel

d). the Visual Basic record.

SELECT Code, Name, Cost FROM Magazine WHERE Name = 'Visual Basic'

Query Builder

The screenshot shows the Query Builder window with the following details:

- Selected Columns:** \* (All Columns), Name
- Table:** Magazine
- Output:** Checkmark for Name.
- Sort Type:** None
- Sort Order:** None
- Filter:** Name LIKE 'C%'
- SQL Statement:**

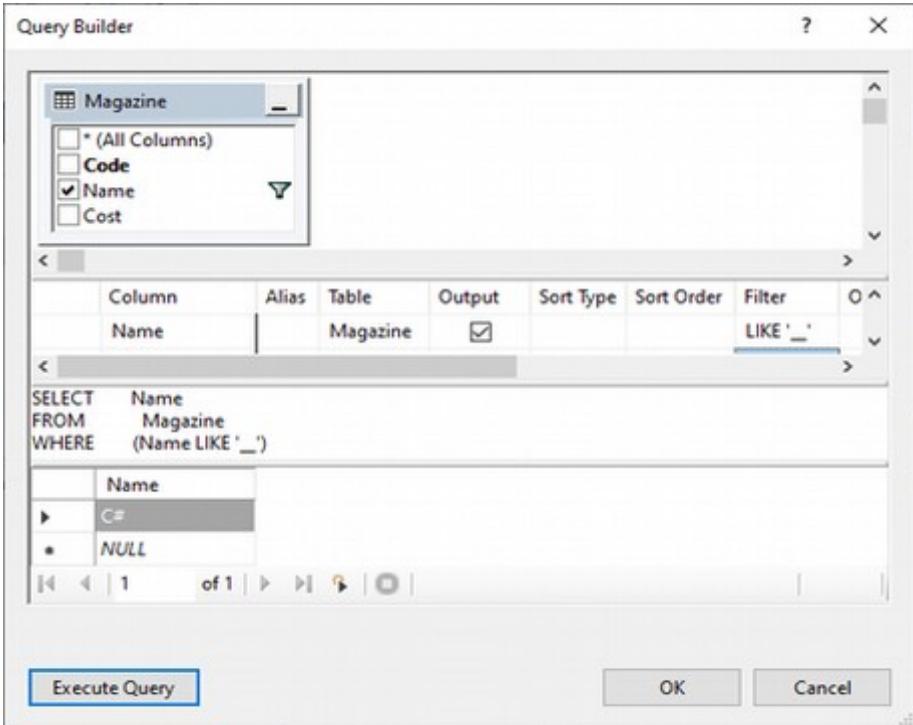
```
SELECT Name
FROM Magazine
WHERE (Name LIKE 'C%')
```
- Result Preview:** A table showing three records:

|   | Name       |
|---|------------|
| ▶ | Country MD |
| ● | C++        |
| ● | C#         |
| ● | NULL       |
- Buttons:** Execute Query, OK, Cancel

e). only the Name field from records whose magazine name begins with the letter C.

SELECT Name FROM Magazine WHERE Name LIKE 'C%'

Query Builder



The Query Builder interface shows a table named "Magazine" with columns "Code", "Name", and "Cost". The "Name" column is selected with a checkmark. The query pane displays:

```
SELECT Name
FROM Magazine
WHERE (Name LIKE '_')
```

The results pane shows the following table:

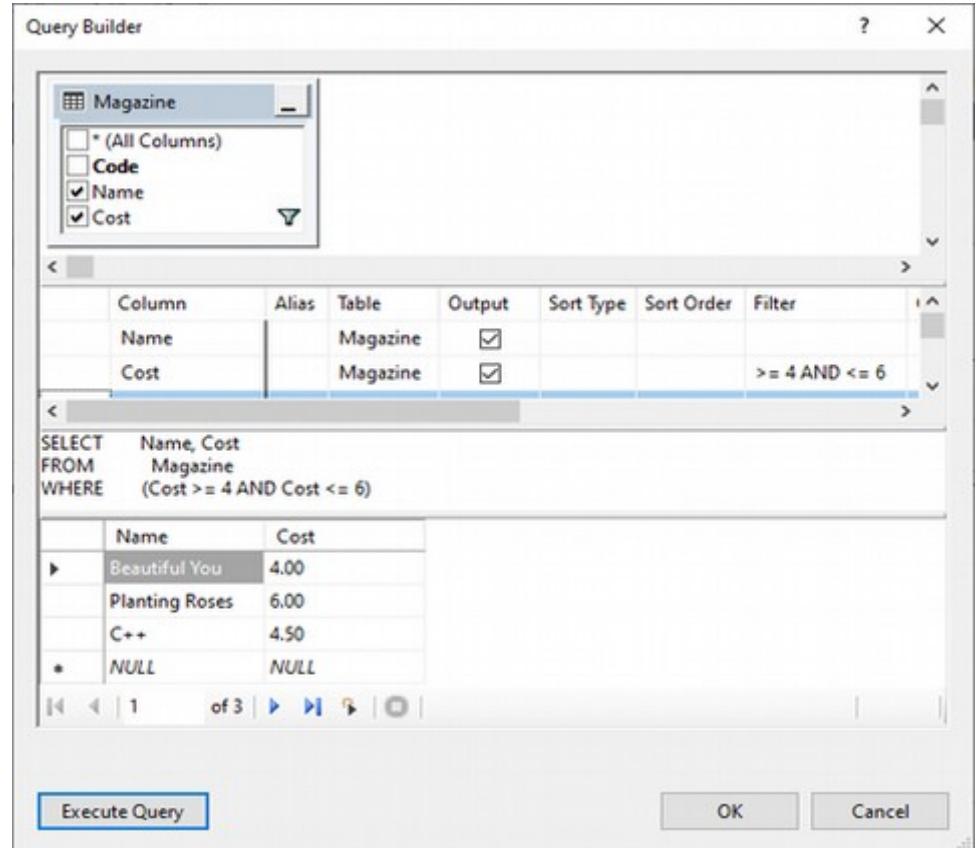
| Name | Code | Cost |
|------|------|------|
| C#   |      |      |
| NULL |      |      |

Buttons at the bottom include "Execute Query", "OK", and "Cancel".

f). only the Name field from records whose magazine name contains 2 characters.

`SELECT Name FROM Magazine WHERE Name LIKE '_'`

Query Builder



The Query Builder interface shows a table named "Magazine" with columns "Code", "Name", and "Cost". The "Name" and "Cost" columns are selected with checkmarks. The query pane displays:

```
SELECT Name, Cost
FROM Magazine
WHERE (Cost >= 4 AND Cost <= 6)
```

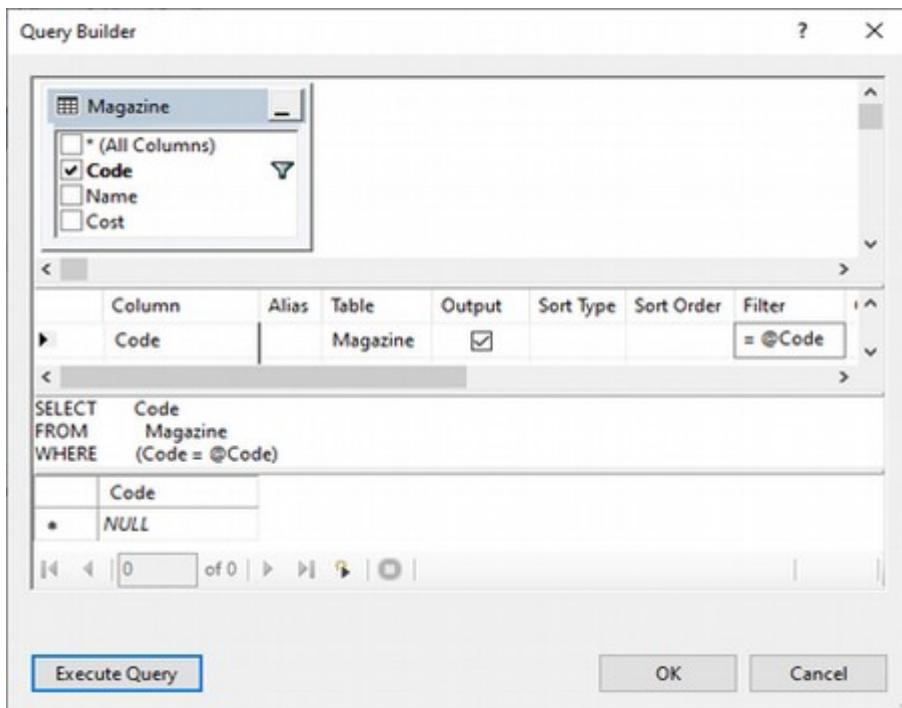
The results pane shows the following table:

| Name           | Cost |
|----------------|------|
| Beautiful You  | 4.00 |
| Planting Roses | 6.00 |
| C++            | 4.50 |
| NULL           | NULL |

Buttons at the bottom include "Execute Query", "OK", and "Cancel".

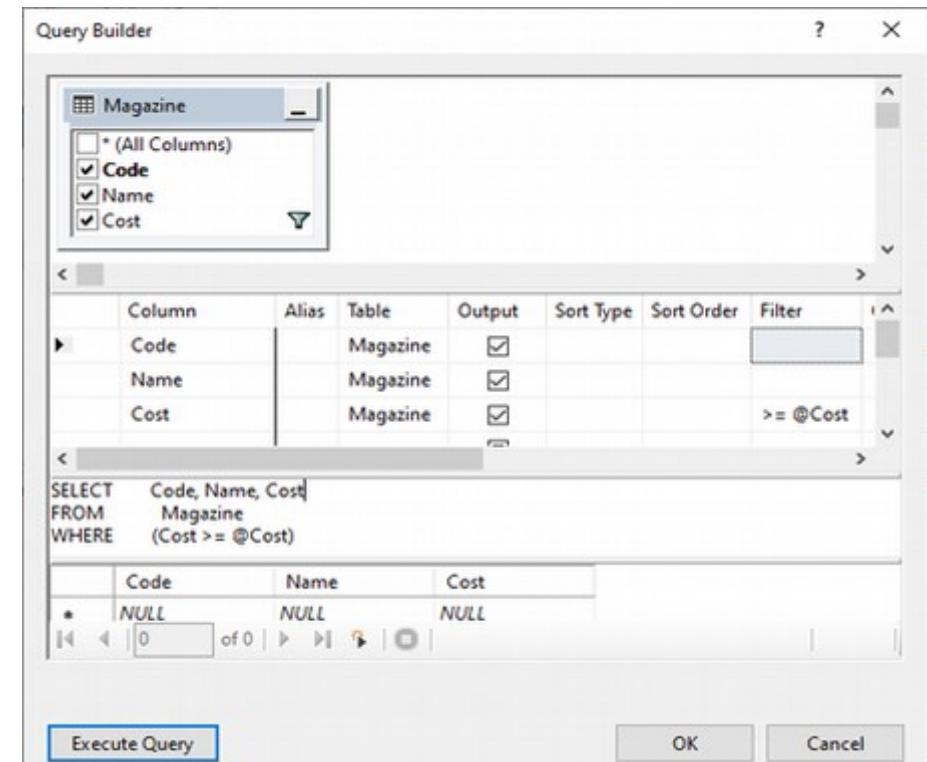
g). only the Name and Cost fields from records whose cost is from \$4 to \$6, inclusive.

`SELECT Name, Cost FROM Magazine WHERE Cost >= 4 AND Cost <= 6`



h). only records for the Code provided by the user.

`SELECT Code FROM Magazine WHERE Code = @Code`



i). only records that cost at least as much as the amount provided by the user.

`SELECT Code, Name, Cost FROM Magazine WHERE Cost >= @Cost`

Query Builder

Magazine

\* (All Columns)  
 Code  
 Name  
 Cost

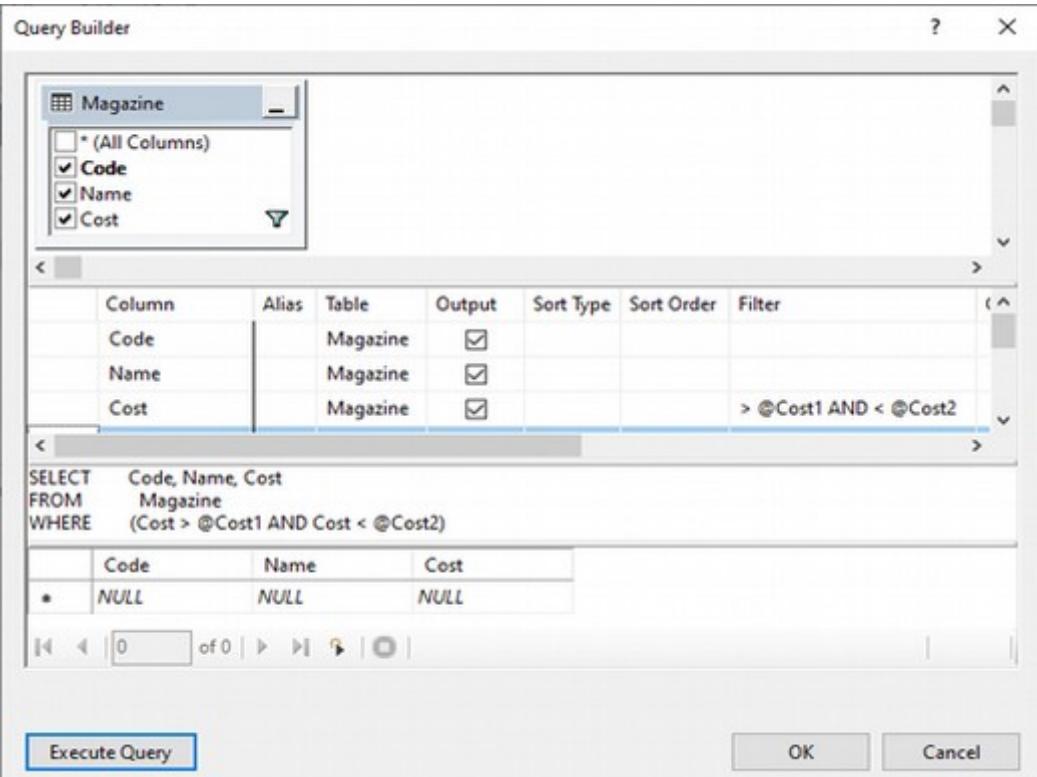
| Column | Alias | Table    | Output                              | Sort Type | Sort Order | Filter                |
|--------|-------|----------|-------------------------------------|-----------|------------|-----------------------|
| Code   |       | Magazine | <input checked="" type="checkbox"/> |           |            |                       |
| Name   |       | Magazine | <input checked="" type="checkbox"/> |           |            |                       |
| Cost   |       | Magazine | <input checked="" type="checkbox"/> |           |            | > @Cost1 AND < @Cost2 |

```
SELECT Code, Name, Cost
FROM Magazine
WHERE (Cost > @Cost1 AND Cost < @Cost2)
```

|   | Code | Name | Cost |
|---|------|------|------|
| * | NULL | NULL | NULL |

|< |< | 0 |> |>>| |

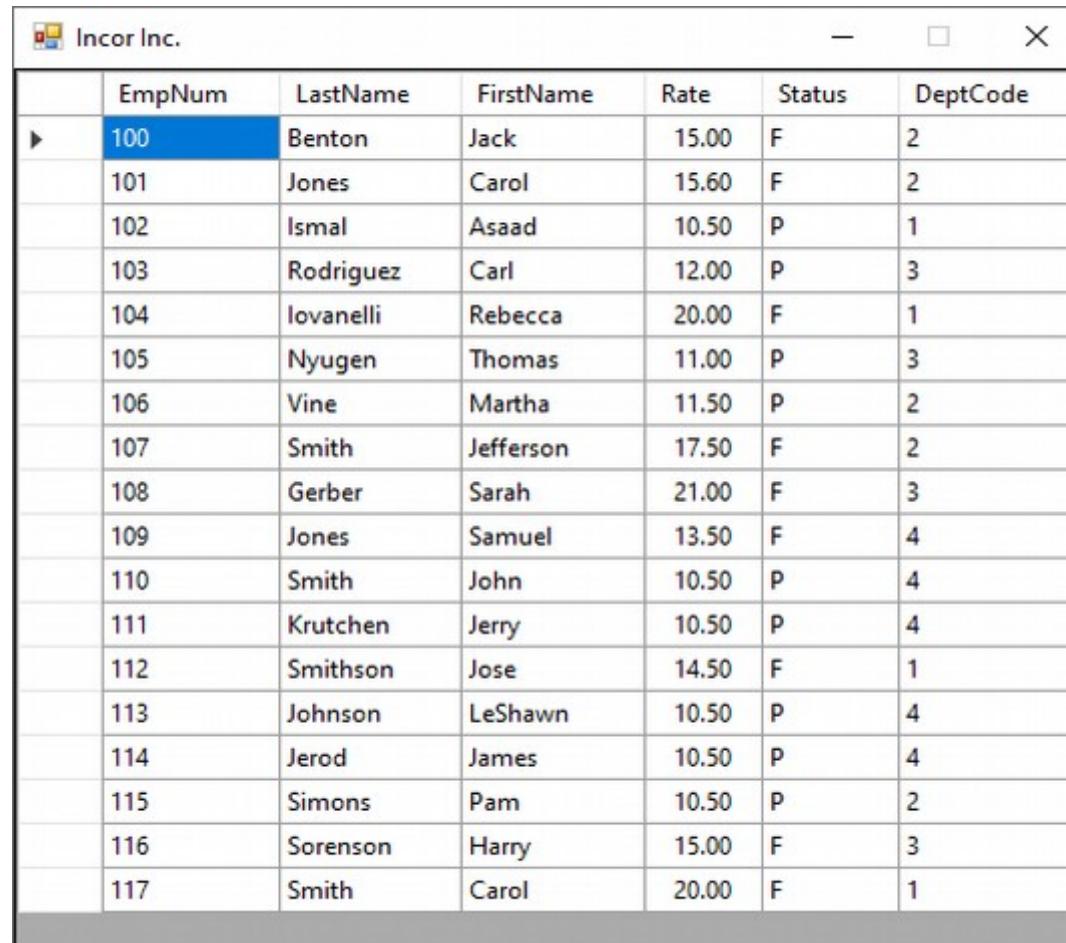
Execute Query      OK      Cancel



j). only records that cost more than the first amount but less than the second amount, both provided by the user

```
SELECT Code, Name, Cost FROM Magazine WHERE Cost > @Cost1 AND Cost < @Cost2
```

- 09.Incor Solution\_EXERCISE 2\_introductory**
- testing SELECT statements in **Query Builder** dialog box:
  - SELECT, ORDER BY ... DESC, WHERE, WHERE ...LIKE %, WHERE field1 = 'condition1' AND field2 = condition2



The screenshot shows a Windows application window titled "Incor Inc.". The main area is a data grid displaying 17 rows of employee information. The columns are labeled: EmpNum, LastName, FirstName, Rate, Status, and DeptCode. The data grid has a header row and 17 data rows. The first row (EmpNum 100) is highlighted with a blue background.

|   | EmpNum | LastName  | FirstName | Rate  | Status | DeptCode |
|---|--------|-----------|-----------|-------|--------|----------|
| ▶ | 100    | Benton    | Jack      | 15.00 | F      | 2        |
|   | 101    | Jones     | Carol     | 15.60 | F      | 2        |
|   | 102    | Ismal     | Asaad     | 10.50 | P      | 1        |
|   | 103    | Rodriguez | Carl      | 12.00 | P      | 3        |
|   | 104    | Iovanelli | Rebecca   | 20.00 | F      | 1        |
|   | 105    | Nyugen    | Thomas    | 11.00 | P      | 3        |
|   | 106    | Vine      | Martha    | 11.50 | P      | 2        |
|   | 107    | Smith     | Jefferson | 17.50 | F      | 2        |
|   | 108    | Gerber    | Sarah     | 21.00 | F      | 3        |
|   | 109    | Jones     | Samuel    | 13.50 | F      | 4        |
|   | 110    | Smith     | John      | 10.50 | P      | 4        |
|   | 111    | Krutchens | Jerry     | 10.50 | P      | 4        |
|   | 112    | Smithson  | Jose      | 14.50 | F      | 1        |
|   | 113    | Johnson   | LeShawn   | 10.50 | P      | 4        |
|   | 114    | Jerod     | James     | 10.50 | P      | 4        |
|   | 115    | Simons    | Pam       | 10.50 | P      | 2        |
|   | 116    | Sorenson  | Harry     | 15.00 | F      | 3        |
|   | 117    | Smith     | Carol     | 20.00 | F      | 1        |

Open the file: ...VB2017\Chap12\Exercise\09.Incor Solution\_EXERCISE 2\_introductory\Incor Solution.sln

The application is already connected to the **Incor.mdf** database, and the **IncorDataSet** has already been created.

Start the application to view the records contained in the dataset and then stop the application.

Open the **DataSet Designer** window and then start the **TableAdapter Query Configuration Wizard**.

Open the **Query Builder** dialog box.

|                                                            |                             |         |                                                                                     |
|------------------------------------------------------------|-----------------------------|---------|-------------------------------------------------------------------------------------|
| Note: The <b>Employees</b> table contains <b>6</b> fields: | - 1. field <u>EmpNum</u>    | numeric |                                                                                     |
|                                                            | - 2. field <u>LastName</u>  | text    |                                                                                     |
|                                                            | - 3. field <u>FirstName</u> | text    |                                                                                     |
|                                                            | - 4. field <u>Rate</u>      | numeric |                                                                                     |
|                                                            | - 5. field <u>Status</u>    | text    | = F for full time<br>= P for part time                                              |
|                                                            | - 6. field <u>DeptCode</u>  | numeric | = 1 for Accounting<br>= 2 for Advertising<br>= 3 for Personnel<br>= 4 for Inventory |
|                                                            |                             |         | (identifies the employee's department)                                              |

1). Write a **SQL SELECT** statement that selects (steps **a** through **m**) & use the **Query Builder** dialog box to test your **SELECT** statements.

#). default **SELECT** statement using all the fields.

```
SELECT EmpNum, LastName, FirstName, Rate, Status, DeptCode FROM Employees
```

a). all of the fields and records in the table and then **sorts** the records in **ascending** order by the DeptCode field.

```
SELECT EmpNum, LastName, FirstName, Rate, Status, DeptCode FROM Employees ORDER BY DeptCode
```

b). only the EmpNum, LastName and FirstName fields from **all** of the records.

```
SELECT EmpNum, LastName, FirstName FROM Employees
```

c). only the records for **full-time** employees.

```
SELECT EmpNum, LastName, FirstName, Rate, Status, DeptCode FROM Employees WHERE Status = 'F'
```

d). the EmpNum, Rate, and DeptCode fields for employees in the **Personnel** department.

```
SELECT EmpNum, Rate, DeptCode FROM Employees WHERE DeptCode = 3
```

e). the EmpNum, LastName, and FirstName fields for employees whose last name is **Smith**.

```
SELECT EmpNum, LastName, FirstName FROM Employees WHERE LastName = 'Smith'
```

f). the EmpNum, LastName, and FirstName fields for employees whose **last** name begins with the letter **S**.

```
SELECT EmpNum, LastName, FirstName FROM Employees WHERE LastName LIKE 'S%'
```

g). only the **first** and **last names** for **part-time** employees and then **sorts** the records in **descending** order by the LastName field.

```
SELECT FirstName, LastName FROM Employees WHERE Status = 'P' ORDER BY LastName DESC
```

h). the records for employees earning more than **\$15** per hour and sorts them in **ascending** order by the Rate field.

```
SELECT EmpNum, LastName, FirstName, Rate, Status, DeptCode FROM Employees WHERE Rate > 15 ORDER BY Rate
```

i). the records for employees whose Rate field contains a value that is at least **\$12** but not more than **\$15**.

```
SELECT EmpNum, LastName, FirstName, Rate, Status, DeptCode FROM Employees WHERE Rate >= 12 AND Rate <= 15
```

j). the records for employee numbers **103** and **109**.

```
SELECT EmpNum, LastName, FirstName, Rate, Status, DeptCode FROM Employees WHERE EmpNum = 103 OR EmpNum = 109
```

k). the records matching the Status provided by the **user**.

```
SELECT EmpNum, LastName, FirstName, Rate, Status, DeptCode FROM Employees WHERE Status = @Status
```

l). the records whose Rate field value is **greater** than the **first** amount provided by the **user** but **less** than the **second** amount provided by the **user**.

```
SELECT EmpNum, LastName, FirstName, Rate, Status, DeptCode FROM Employees WHERE Rate > @Rate1 AND Rate < @Rate2
```

m). records that match both the Status and the DeptCode provided by the user.

(e.g. if the user provides **P** and **1**, the statement should select only **part-time** employees in the **Accounting** department)

```
SELECT EmpNum, LastName, FirstName, Rate, Status, DeptCode FROM Employees WHERE Status = @Status AND DeptCode = @DeptCode
```

- 2). Use the **Query Builder** dialog box to create a statement that selects the records for all part-time employees earning more than \$11 per hour.  
(You can complete the Filter box for more than 1 field). What SQL statement appears in the SQL pane?

SELECT EmpNum, LastName, FirstName, Rate, Status, DeptCode FROM Employees WHERE Status = 'p' AND Rate > 11

Query Builder

Employees

| Column    | Alias | Table     | Output                              | Sort Type | Sort Order | Filter | Or... | Or... |
|-----------|-------|-----------|-------------------------------------|-----------|------------|--------|-------|-------|
| EmpNum    |       | Employees | <input checked="" type="checkbox"/> |           |            |        |       |       |
| LastName  |       | Employees | <input checked="" type="checkbox"/> |           |            |        |       |       |
| FirstName |       | Employees | <input checked="" type="checkbox"/> |           |            |        |       |       |
| Rate      |       | Employees | <input checked="" type="checkbox"/> |           |            | > 11   |       |       |
| Status    |       | Employees | <input checked="" type="checkbox"/> |           |            | = 'p'  |       |       |
| DeptCode  |       | Employees | <input checked="" type="checkbox"/> |           |            |        |       |       |

SELECT EmpNum, LastName, FirstName, Rate, Status, DeptCode  
FROM Employees  
WHERE (Status = 'p') AND (Rate > 11)

|   | EmpNum | LastName  | FirstName | Rate  | Status | DeptCode |
|---|--------|-----------|-----------|-------|--------|----------|
| ▶ | 103    | Rodriguez | Carl      | 12.00 | P      | 3        |
|   | 106    | Vine      | Martha    | 11.50 | P      | 2        |
| * | NULL   | NULL      | NULL      | NULL  | NULL   | NULL     |

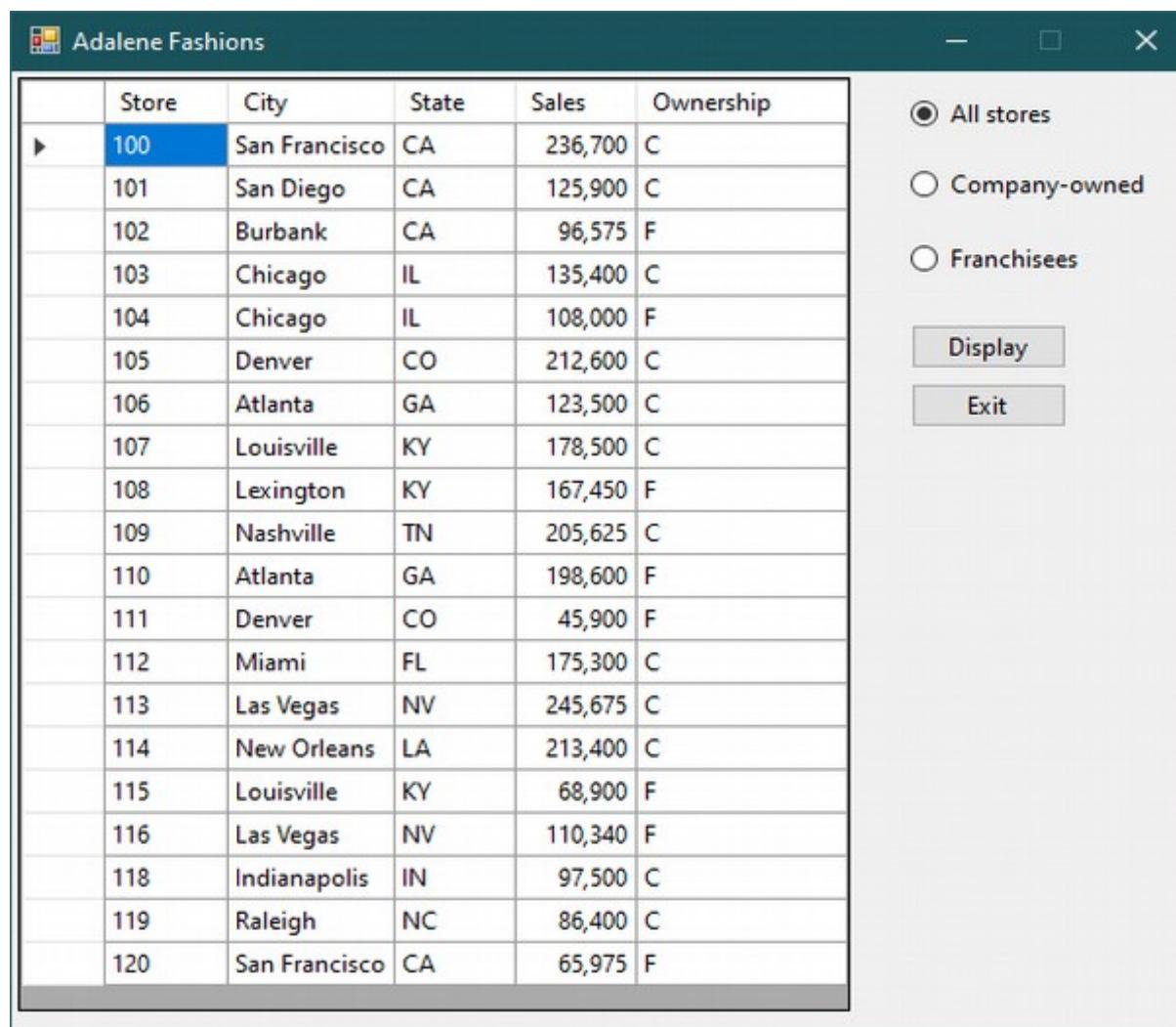
Cell is Read Only.

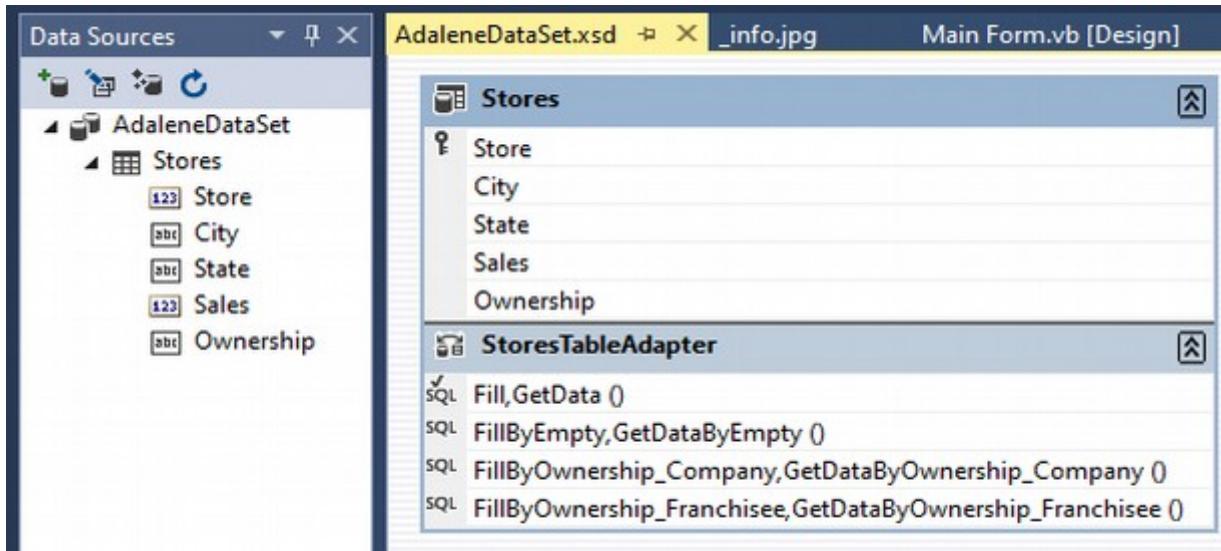
Execute Query      OK      Cancel

#### 10.Adalene Solution\_EXERCISE 3\_intermediate

- create several **Queries** and associate them with a methods
- use the methods in code to filter values displayed in **DataSet**

3. Open the Adalene Solution.sln file contained in the VB2017\Chap12\Adalene Solution folder. The application is already connected to the Adalene.mdf file, and the AdaleneDataSet has already been created. Start the application to view the records in the dataset and then stop the application. The Adalene Fashions application should allow the user to display all of the information in the dataset, only the information for company-owned stores, or only the information for franchisees. Create the appropriate queries and then use them to code the btnDisplay\_Click procedure. Save the solution and then start and test the application.





| StoresTableAdapter                                                                  |                                                                 |                                                                                                    |  |
|-------------------------------------------------------------------------------------|-----------------------------------------------------------------|----------------------------------------------------------------------------------------------------|--|
| SQL statement:                                                                      | associated with a methods:                                      | info:                                                                                              |  |
| SELECT Store, City, State, Sales, Ownership FROM dbo.Stores                         | Fill()<br>GetData()                                             | <b>TableAdapter default SQL statement</b><br>select all                                            |  |
| SELECT Store, City, State, Sales, Ownership<br>FROM Stores<br>WHERE Ownership = 'd' | FillByEmpty()<br>GetDataByEmpty()                               | <b>TableAdapter Query</b><br>my idea, using non-existing value for a method to display empty cells |  |
| SELECT Store, City, State, Sales, Ownership<br>FROM Stores<br>WHERE Ownership = 'C' | FillByOwnership_Company()<br>GetDataByOwnership_Company()       | <b>TableAdapter Query</b><br>selects only Company - C                                              |  |
| SELECT Store, City, State, Sales, Ownership<br>FROM Stores<br>WHERE Ownership = 'F' | FillByOwnership_Franchisee()<br>GetDataByOwnership_Franchisee() | <b>TableAdapter Query</b><br>select only Franchisees - F                                           |  |

```
1  ' Name:      Adalene Project
2  ' Purpose:    Select and view records.
3  ' Programmer: <your name> on <current date>
4  Option Explicit On
5  Option Strict On
6  Option Infer Off
7
8  Public Class frmMain
9      Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles MyBase.Load
10         'TODO: This line of code loads data into the 'AdaleneDataSet.Stores' table. You can move, or remove it, as needed.
11         Me.StoresTableAdapter.Fill(Me.AdaleneDataSet.Stores)
12     End Sub
13
14     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
15         Me.Close()
16     End Sub
17
18     Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
19         If radCompany.Checked Then ' display only Ownership = 'C'
20             StoresTableAdapter.FillByOwnership_Company(AdaleneDataSet.Stores)
21         ElseIf radFranchisee.Checked Then ' display only Ownership = 'F'
22             StoresTableAdapter.FillByOwnership_Franchisee(AdaleneDataSet.Stores)
23         Else ' display default all - C + F
24             StoresTableAdapter.Fill(AdaleneDataSet.Stores)
25         End If
26     End Sub
27
28     Private Sub Allrad_CheckedChanged(sender As Object, e As EventArgs) Handles radAll.CheckedChanged, radCompany.CheckedChanged,
29         radFranchisee.CheckedChanged
30         ' my idea for clearing values when (rad)s changed.
31         StoresTableAdapter.FillByEmpty(AdaleneDataSet.Stores)
32     End Sub
33 End Class
```

## Adalene Fashions

|  | Store | City | State | Sales | Ownership |
|--|-------|------|-------|-------|-----------|
|--|-------|------|-------|-------|-----------|

 All stores Company-owned Franchisees

Display

## Adalene Fashions

|   | Store | City          | State | Sales   | Ownership |
|---|-------|---------------|-------|---------|-----------|
| ▶ | 100   | San Francisco | CA    | 236,700 | C         |
|   | 101   | San Diego     | CA    | 125,900 | C         |
|   | 103   | Chicago       | IL    | 135,400 | C         |
|   | 105   | Denver        | CO    | 212,600 | C         |
|   | 106   | Atlanta       | GA    | 123,500 | C         |
|   | 107   | Louisville    | KY    | 178,500 | C         |
|   | 109   | Nashville     | TN    | 205,625 | C         |
|   | 112   | Miami         | FL    | 175,300 | C         |
|   | 113   | Las Vegas     | NV    | 245,675 | C         |
|   | 114   | New Orleans   | LA    | 213,400 | C         |
|   | 118   | Indianapolis  | IN    | 97,500  | C         |
|   | 119   | Raleigh       | NC    | 86,400  | C         |

 All stores Company-owned Fr

## Adalene Fashions

|   | Store | City          | State | Sales   | Ownership |
|---|-------|---------------|-------|---------|-----------|
| ▶ | 102   | Burbank       | CA    | 96,575  | F         |
|   | 104   | Chicago       | IL    | 108,000 | F         |
|   | 108   | Lexington     | KY    | 167,450 | F         |
|   | 110   | Atlanta       | GA    | 198,600 | F         |
|   | 111   | Denver        | CO    | 45,900  | F         |
|   | 115   | Louisville    | KY    | 68,900  | F         |
|   | 116   | Las Vegas     | NV    | 110,340 | F         |
|   | 120   | San Francisco | CA    | 65,975  | F         |

 All stores Company-owned Franchisees

Display

Exit

- 11.Opals Solution\_EXERCISE 4\_intermediate**
- create a **calculated field** by modifying the default SELECT statement / using Query Builder, using **TableAdapter Configuration Wizard**
  - my discovery: enclose the name in **square brackets [ ]** when it contains special characters like space char, ... etc

Open the Opals Solution.sln file contained in the VB2017\Chap12\Opals Solution folder. The application is already connected to the Opals.mdf file, and the OpalsDataSet has already been created. The Product table in the database is shown in Figure 12-40.

Create a query that calculates the total sales amount for each item. After creating the query, display the Product table (which will now contain the calculated field) in a DataGridView control, as shown in Figure 12-41. Save the solution and then start and test the application.

|       | Name | Data Type   | Allow Nulls              | Default |
|-------|------|-------------|--------------------------|---------|
| • ID  |      | varchar(5)  | <input type="checkbox"/> |         |
| Name  |      | varchar(50) | <input type="checkbox"/> |         |
| Price |      | int         | <input type="checkbox"/> |         |
| Sold  |      | int         | <input type="checkbox"/> |         |

| ID    | Name             | Price | Sold |
|-------|------------------|-------|------|
| F120  | Full Headboard   | 60    | 7    |
| F345  | Full Bed         | 110   | 10   |
| K120  | King Headboard   | 120   | 45   |
| K345  | King Bed         | 200   | 57   |
| K78   | King Pillows     | 45    | 35   |
| Q120  | Queen Headboard  | 75    | 38   |
| Q345  | Queen Bed        | 150   | 35   |
| S78   | Standard Pillows | 10    | 59   |
| TW120 | Twin Headboard   | 50    | 24   |
| TW345 | Twin Bed         | 75    | 24   |

**Figure 12-40**

database: **Opals.mdf**, table: **dbo.Product**

|   | ID    | Name             | Price | Sold | Total Sales |
|---|-------|------------------|-------|------|-------------|
| ► | F120  | Full Headboard   | 60    | 7    | \$420       |
|   | F345  | Full Bed         | 110   | 10   | \$1,100     |
|   | K120  | King Headboard   | 120   | 45   | \$5,400     |
|   | K345  | King Bed         | 200   | 57   | \$11,400    |
|   | K78   | King Pillows     | 45    | 35   | \$1,575     |
|   | Q120  | Queen Headboard  | 75    | 38   | \$2,850     |
|   | Q345  | Queen Bed        | 150   | 35   | \$5,250     |
|   | S78   | Standard Pillows | 10    | 59   | \$590       |
|   | TW120 | Twin Headboard   | 50    | 24   | \$1,200     |
|   | TW345 | Twin Bed         | 75    | 24   | \$1,800     |

**Figure 12-41** DataGridView control showing the total sales for each item

My important discovery about creating the **Calculated Field** by modifying the default query SELECT statement using **TableAdapter Configuration Wizard**:

- so, I follow the: **CH12\_A1 - add a Calculated Field to a Dataset by modifying the default SELECT statement / using Query Builder e.g. with 06.Ellington Solution**

- you can **modify** the default **SELECT** statement to add a **calculated field** to a dataset, using the **TableAdapter Configuration Wizard**

1). open the **DataSet Designer** window

2). open the **TableAdapter Configuration Wizard** (**ProductTableAdapter** / line: **Fill,GetData ()** => Rclick and choose: **Configure...**)

3). in **TableAdapter Configuration Wizard** / screen **Enter a SQL Statement:**

- see default: **SELECT ID, Name, Price, Sold FROM Product**

-> if you type: **SELECT ID, Name, Price, Sold, Price \* Sold As Total Sales FROM Product**

-> if you type: **SELECT ID, Name, Price, Sold, Price \* Sold As TotalSales FROM Product**

<- error

<- OK, but not what I want - so the name is the problem

4). but if you use **Query Builder**:

- in **Grid pane:**

- > and add new Column: **Price \* Sold**
- > change default alias **Expr1** to **Total Sales**
- <- the square brackets are added automatically

->

- see **SQL pane:**

```
SELECT ID, Name, Price, Sold, Price * Sold AS [Total Sales]
FROM Product
```

->

The screenshot shows the Microsoft Query Builder interface. In the top left, there's a 'Product' table editor showing columns: ID, Name, Price, and Sold. The Grid pane below it lists these columns with their respective tables (Product) and sort orders. A new row is being edited, showing 'Price \* Sold' as the column name, '[Total Sales]' as the alias, and checked boxes for Output, Sort Type, and Sort Order. The SQL pane at the bottom contains the generated SQL: 'SELECT ID, Name, Price, Sold, Price \* Sold AS [Total Sales] FROM Product'. The results grid shows ten rows of product data with the 'Total Sales' column populated.

| Column         | Alias         | Table   | Output                              | Sort Type | Sort Order |
|----------------|---------------|---------|-------------------------------------|-----------|------------|
| ID             |               | Product | <input checked="" type="checkbox"/> |           |            |
| Name           |               | Product | <input checked="" type="checkbox"/> |           |            |
| Price          |               | Product | <input checked="" type="checkbox"/> |           |            |
| Sold           |               | Product | <input checked="" type="checkbox"/> |           |            |
| ▶ Price * Sold | [Total Sales] |         | <input checked="" type="checkbox"/> |           |            |

|   | ID    | Name             | Price | Sold | Total Sales |
|---|-------|------------------|-------|------|-------------|
| ▶ | F120  | Full Headboard   | 60    | 7    | 420         |
|   | F345  | Full Bed         | 110   | 10   | 1100        |
|   | K120  | King Headboard   | 120   | 45   | 5400        |
|   | K345  | King Bed         | 200   | 57   | 11400       |
|   | K78   | King Pillows     | 45    | 35   | 1575        |
|   | Q120  | Queen Headbo...  | 75    | 38   | 2850        |
|   | Q345  | Queen Bed        | 150   | 35   | 5250        |
|   | S78   | Standard Pillows | 10    | 59   | 590         |
|   | TW120 | Twin Headboard   | 50    | 24   | 1200        |
|   | TW345 | Twin Bed         | 75    | 24   | 1800        |
| * | NULL  | NULL             | NULL  | NULL | NULL        |

Execute Query      OK      Cancel

5). back to **TableAdapter Configuration Wizard** /

/ screen **Enter a SQL Statement:**

- default SQL query statement:

```
SELECT ID, Name, Price, Sold FROM Product
```

- is modified to:

```
SELECT ID, Name, Price, Sold, Price * Sold AS [Total Sales]
FROM Product
```

6). after this, i tested again without **Query Builder**:

```
SELECT ID, Name, Price, Sold, Price * Sold AS [Total Sales] FROM Product
```

- and it works - supeeeeer

-> so the **[ ] square brackets** was the issue

-> I modified the **CH12\_A1** and added info about **[ ]**

Data Sources ▾ X

OpalsDataSet.xsd ▾ X Main Form.vb [Design] Main Form.vb

The screenshot shows the Visual Studio IDE's Data Sources window. Under 'OpalsDataSet', there is a 'Product' table. The table has columns: ID, Name, Price, Sold, and Total Sales. Below the table is a 'ProductTableAdapter' section with a 'Fill,GetData ()' method and its corresponding SQL query:

```
SQL Fill,GetData ()
SELECT ID, Name, Price, Sold, Price * Sold AS [Total Sales]
FROM Product
```

Opals Bedding

|   | ID    | Name             | Price | Sold | Total Sales |
|---|-------|------------------|-------|------|-------------|
| ▶ | F120  | Full Headboard   | 60    | 7    | \$ 420      |
|   | F345  | Full Bed         | 110   | 10   | \$ 1,100    |
|   | K120  | King Headboard   | 120   | 45   | \$ 5,400    |
|   | K345  | King Bed         | 200   | 57   | \$ 11,400   |
|   | K78   | King Pillows     | 45    | 35   | \$ 1,575    |
|   | Q120  | Queen Headboard  | 75    | 38   | \$ 2,850    |
|   | Q345  | Queen Bed        | 150   | 35   | \$ 5,250    |
|   | S78   | Standard Pillows | 10    | 59   | \$ 590      |
|   | TW120 | Twin Headboard   | 50    | 24   | \$ 1,200    |
|   | TW345 | Twin Bed         | 75    | 24   | \$ 1,800    |

```

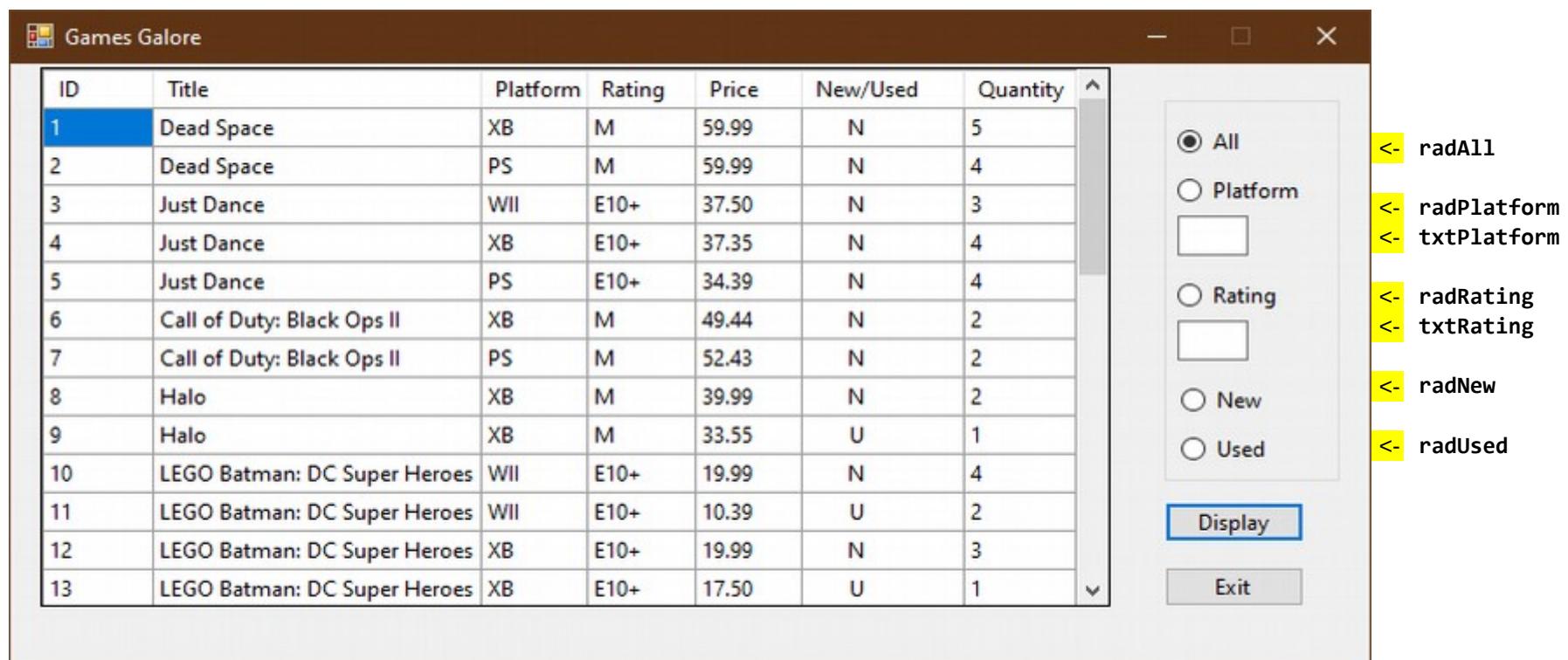
1  Public Class frmMain
2      Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles MyBase.Load
3          'TODO: This line of code loads data into the 'OpalsDataSet.Product' table. You can move, or remove it, as needed.
4          Me.ProductTableAdapter.Fill(Me.OpalsDataSet.Product)
5      End Sub
6  End Class

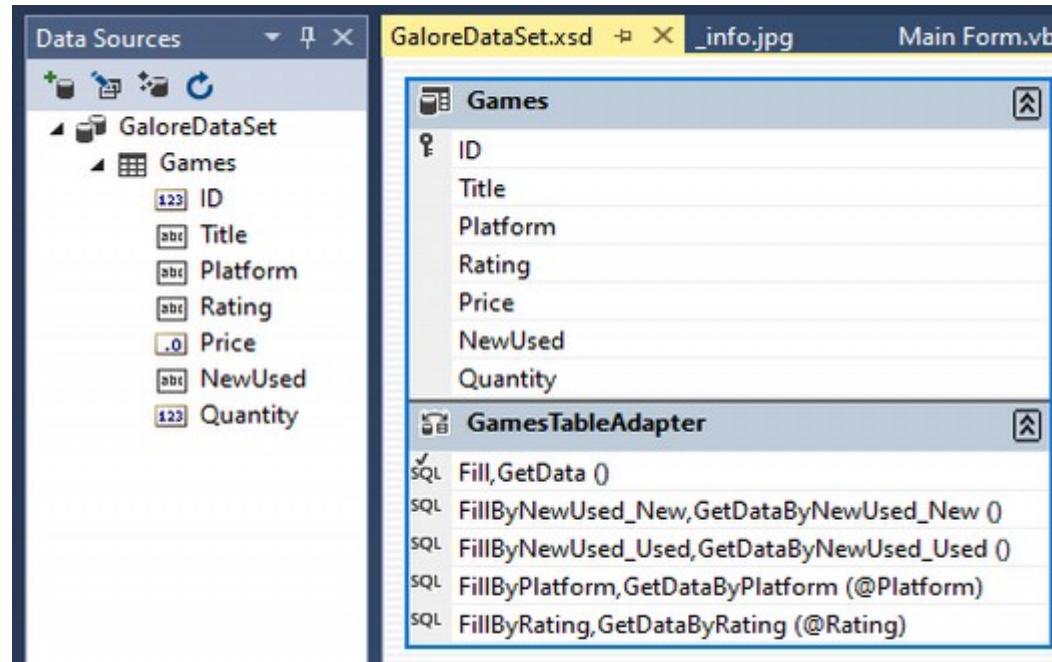
```

## 12.Games Solution\_EXERCISE 5\_advanced

- create several **Queries** and **Parameter Queries**, and associate them with a methods
- use the method in a code to filter values displayed in **DataSet**

5. Open the Games Solution.sln file contained in the VB2017\Chap12\Games Solution folder. Start the application to view the records, and then click the Exit button. In addition to displaying all of the records, the application should allow the user to display only the games for a specific platform, only the games with a specific rating, only the games that are marked as new, or only the games that are marked as used. Complete the application. Save the solution and then start and test the application.





| GamesTableAdapter                                                                                          |                                                             |                                                                                               |
|------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------|-----------------------------------------------------------------------------------------------|
| SQL statement:                                                                                             | associated with a methods:                                  | info:                                                                                         |
| SELECT ID, Title, Platform, Rating, Price, NewUsed, Quantity FROM dbo.Games                                | Fill()<br>GetData()                                         | TableAdapter default SQL statement<br>radAll.Checked, select all                              |
| SELECT ID, Title, Platform, Rating, Price, NewUsed, Quantity<br>FROM Games<br>WHERE (Platform = @Platform) | FillByPlatform (@Platform)<br>GetDataByPlatform (@Platform) | TableAdapter Query<br>radPlatform.Checked + txtPlatform<br>txtfield: Platform = XB / PS / WII |
| SELECT ID, Title, Platform, Rating, Price, NewUsed, Quantity<br>FROM Games<br>WHERE (Rating = @Rating)     | FillByRating (@Rating)<br>GetDataByRating (@Rating)         | TableAdapter Query<br>radRating.Checked + txtRating<br>txtfield: Rating = E10+ / E / M / T    |
| SELECT ID, Title, Platform, Rating, Price, NewUsed, Quantity<br>FROM Games<br>WHERE (NewUsed = 'N')        | FillByNewUsed_New()<br>GetDataByNewUsed_New()               | TableAdapter Query<br>radNew.Checked<br>txtfield: NewUsed = N                                 |
| SELECT ID, Title, Platform, Rating, Price, NewUsed, Quantity<br>FROM Games<br>WHERE (NewUsed = 'U')        | FillByNewUsed_Used()<br>GetDataByNewUsed_Used()             | TableAdapter Query<br>radUsed.Checked<br>txtfield: NewUsed = U                                |

```
1  ' Name:      Games Project
2  ' Purpose:    Display all records and records meeting specific criteria.
3  ' Programmer: <your name> on <current date>
4  Option Explicit On
5  Option Strict On
6  Option Infer Off
7
8  Public Class frmMain
9      Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles MyBase.Load
10         'TODO: This line of code loads data into the 'GaloreDataSet.Games' table. You can move, or remove it, as needed.
11         Me.GamesTableAdapter.Fill(Me.GaloreDataSet.Games)
12     End Sub
13
14     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
15         Me.Close()
16     End Sub
17
18     Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
19         Select Case True
20             Case radPlatform.Checked ' + txtPlatform
21                 GamesTableAdapter.FillByPlatform(GaloreDataSet.Games, txtPlatform.Text.Trim)
22             Case radRating.Checked ' + txtRating
23                 GamesTableAdapter.FillByRating(GaloreDataSet.Games, txtRating.Text.Trim)
24             Case radNew.Checked
25                 GamesTableAdapter.FillByNewUsed_New(GaloreDataSet.Games)
26             Case radUsed.Checked
27                 GamesTableAdapter.FillByNewUsed_Used(GaloreDataSet.Games)
28             Case Else ' radAll.Checked
29                 GamesTableAdapter.Fill(GaloreDataSet.Games)
30         End Select
31     End Sub
32 End Class
```

## Games Galore

| ID | Title | Platform | Rating | Price | New/Used | Quantity |
|----|-------|----------|--------|-------|----------|----------|
| a  |       |          |        |       |          |          |

All  
 Platform  
 Rating  
 a

## Games Galore

| ID | Title                        | Platform | Rating | Price | New/Used | Quantity |
|----|------------------------------|----------|--------|-------|----------|----------|
| 2  | Dead Space                   | PS       | M      | 59.99 | N        | 4        |
| 5  | Just Dance                   | PS       | E10+   | 34.39 | N        | 4        |
| 7  | Call of Duty: Black Ops II   | PS       | M      | 52.43 | N        | 2        |
| 14 | LEGO Batman: DC Super Heroes | PS       | E10+   | 19.99 | N        | 2        |
| 15 | LEGO Batman: DC Super Heroes | PS       | E10+   | 17.75 | U        | 1        |
| 18 | The Sims Pets                | PS       | T      | 17.65 | N        | 3        |

All  
 Platform  
 Rating  
 ps

## Games Galore

| ID | Title           |
|----|-----------------|
| 22 | Madden NFL 16   |
| 23 | Madden NFL 16   |
| 26 | Resident Evil 6 |
| 27 | Resident Evil 6 |
| 32 | NBA 2K16        |
| 33 | NBA 2K15        |

| ID | Title                        | Platform | Rating | Price | New/Used | Quantity |
|----|------------------------------|----------|--------|-------|----------|----------|
| 3  | Just Dance                   | WII      | E10+   | 37.50 | N        | 3        |
| 4  | Just Dance                   | XB       | E10+   | 37.35 | N        | 4        |
| 5  | Just Dance                   | PS       | E10+   | 34.39 | N        | 4        |
| 10 | LEGO Batman: DC Super Heroes | WII      | E10+   | 19.99 | N        | 4        |
| 11 | LEGO Batman: DC Super Heroes | WII      | E10+   | 10.39 | U        | 2        |
| 12 | LEGO Batman: DC Super Heroes | XB       | E10+   | 19.99 | N        | 3        |
| 13 | LEGO Batman: DC Super Heroes | XB       | E10+   | 17.50 | U        | 1        |
| 14 | LEGO Batman: DC Super Heroes | PS       | E10+   | 19.99 | N        | 2        |
| 15 | LEGO Batman: DC Super Heroes | PS       | E10+   | 17.75 | U        | 1        |

All  
 Platform  
 Rating  
 ps  
 Rating  
 e10+  
 New  
 Used

Display

Exit

## Games Galore

| ID | Title                        | Platform | Rating | Price | New/Used | Quantity |
|----|------------------------------|----------|--------|-------|----------|----------|
| 1  | Dead Space                   | XB       | M      | 59.99 | N        | 5        |
| 2  | Dead Space                   | PS       | M      | 59.99 | N        | 4        |
| 3  | Just Dance                   | WII      | E10+   | 37.50 | N        | 3        |
| 4  | Just Dance                   | XB       | E10+   | 37.35 | N        | 4        |
| 5  | Just Dance                   | PS       | E10+   | 34.39 | N        | 4        |
| 6  | Call of Duty: Black Ops II   | XB       | M      | 49.44 | N        | 2        |
| 7  | Call of Duty: Black Ops II   | PS       | M      | 52.43 | N        | 2        |
| 8  | Halo                         | XB       | M      | 39.99 | N        | 2        |
| 10 | LEGO Batman: DC Super Heroes | WII      | E10+   | 19.99 | N        | 4        |
| 12 | LEGO Batman: DC Super Heroes | XB       | E10+   | 19.99 | N        | 3        |
| 14 | LEGO Batman: DC Super Heroes | PS       | E10+   | 19.99 | N        | 2        |
| 16 | The Sims Pets                | XB       | T      | 19.99 | N        | 4        |
| 18 | The Sims Pets                | PS       | T      | 17.65 | N        | 3        |

All  
 Platform  
  
 Rating  
  
 New  
 Used

**Display****Exit**

## Games Galore

| ID | Title                        | Platform | Rating | Price | New/Used | Quantity |
|----|------------------------------|----------|--------|-------|----------|----------|
| 9  | Halo                         | XB       | M      | 33.55 | U        | 1        |
| 11 | LEGO Batman: DC Super Heroes | WII      | E10+   | 10.39 | U        | 2        |
| 13 | LEGO Batman: DC Super Heroes | XB       | E10+   | 17.50 | U        | 1        |
| 15 | LEGO Batman: DC Super Heroes | PS       | E10+   | 17.75 | U        | 1        |
| 17 | The Sims Pets                | XB       | T      | 7.99  | U        | 1        |
| 19 | The Sims Pets                | PS       | T      | 8.50  | U        | 2        |
| 21 | Madden NFL 16                | XB       | E      | 35.50 | U        | 2        |
| 23 | Madden NFL 16                | PS       | E      | 33.50 | U        | 2        |
| 24 | Madden NFL 15                | WII      | E      | 35.99 | U        | 3        |
| 25 | Madden NFL 14                | WII      | E      | 29.97 | U        | 1        |
| 27 | Resident Evil                | PS       | M      | 25.98 | U        | 1        |
| 29 | Resident Evil                | XB       | M      | 15.95 | U        | 1        |
| 31 | NBA 2K16                     | XB       | E      | 37.71 | U        | 1        |

All  
 Platform  
  
 Rating  
  
 New  
 Used

**Display****Exit**

### 13.Games Solution-Total\_EXERCISE 6\_advanced

- 1). add **TableAdapter**
- 2). create Query using **Aggregate function**
- 3). drag the **DataColumn** to the label control = create bound control by dragging

6. In this exercise, you modify the Games application from Exercise 5. Use Windows to make a copy of the Games Solution folder. Rename the copy Games Solution-Total. The frmMain\_Load procedure should display (in a label control) the total value of the games sold in the store. Display the value with a dollar sign and two decimal places. Complete the application. Save the solution and then start and test the application.

The screenshot shows the Visual Studio Data Sources window. On the left, the GaloreDataSet.xsd schema is displayed with two main tables: Games and Total. The Games table contains columns ID, Title, Platform, Rating, Price, NewUsed, and Quantity. The Total table contains a single column Total. Below each table is its corresponding TableAdapter, GamesTableAdapter and TotalTableAdapter respectively. The TotalTableAdapter has a SQL query defined:

```

SQL Fill,GetData()
SQL FillByNewUsed_New,GetDataByNewUsed_New()
SQL FillByNewUsed_Used,GetDataByNewUsed_Used()
SQL FillByPlatform,GetDataByPlatform (@Platform)
SQL FillByRating,GetDataByRating (@Rating)

```

Below the SQL queries, the generated SQL code for the FillByTotal, GetDataByTotal method is shown:

```

SELECT SUM(Quantity * Price) AS Total
FROM Games

```

|                      | price: | Q: | multiplied:  |
|----------------------|--------|----|--------------|
| 1                    | 59.99  | 5  | 299.95       |
| 2                    | 59.99  | 4  | 239.96       |
| 3                    | 37.5   | 3  | 112.5        |
| 4                    | 37.35  | 4  | 149.4        |
| 5                    | 34.39  | 4  | 137.56       |
| 6                    | 49.44  | 2  | 98.88        |
| 7                    | 52.43  | 2  | 104.86       |
| 8                    | 39.99  | 2  | 79.98        |
| 9                    | 33.55  | 1  | 33.55        |
| 10                   | 19.99  | 4  | 79.96        |
| 11                   | 10.39  | 2  | 20.78        |
| 12                   | 19.99  | 3  | 59.97        |
| 13                   | 17.5   | 1  | 17.5         |
| 14                   | 19.99  | 2  | 39.98        |
| 15                   | 17.75  | 1  | 17.75        |
| 16                   | 19.99  | 4  | 79.96        |
| 17                   | 7.99   | 1  | 7.99         |
| 18                   | 17.65  | 3  | 52.95        |
| 19                   | 8.5    | 2  | 17           |
| 20                   | 39.99  | 8  | 319.92       |
| 21                   | 35.5   | 2  | 71           |
| 22                   | 35.5   | 6  | 213          |
| 23                   | 33.5   | 2  | 67           |
| 24                   | 35.99  | 3  | 107.97       |
| 25                   | 29.97  | 1  | 29.97        |
| 26                   | 27.6   | 3  | 82.8         |
| 27                   | 25.98  | 1  | 25.98        |
| 28                   | 27.59  | 3  | 82.77        |
| 29                   | 15.95  | 1  | 15.95        |
| 30                   | 43.8   | 9  | 394.2        |
| 31                   | 37.71  | 1  | 37.71        |
| 32                   | 43.99  | 6  | 263.94       |
| 33                   | 42     | 1  | 42           |
| 34                   | 33.99  | 7  | 237.93       |
| 35                   | 29.99  | 3  | 89.97        |
| my test calculation: |        |    | 3732.59      |
| SQL calculation:     |        |    | \$ 3, 732.59 |

```
1  ' Name:      Games Project
2  ' Purpose:    Display all records and records meeting specific criteria.
3  ' Programmer: <your name> on <current date>
4  ' display in label control the total value of the games sold in the store. Display the value with a dollar sign and 2 decimal places:
5  ' lblTotal = Price * Quantity
6  Option Explicit On
7  Option Strict On
8  Option Infer Off
9
10 Public Class frmMain
11     Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles MyBase.Load
12         'TODO: This line of code loads data into the 'GaloreDataSet.Total' table. You can move, or remove it, as needed.
13         Me.TotalTableAdapter.FillByTotal(Me.GaloreDataSet.Total)
14         'TODO: This line of code loads data into the 'GaloreDataSet.Games' table. You can move, or remove it, as needed.
15         Me.GamesTableAdapter.Fill(Me.GaloreDataSet.Games)
16     End Sub
17
18     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
19         Me.Close()
20     End Sub
21
22     Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
23         Select Case True
24             Case radPlatform.Checked ' + txtPlatform
25                 GamesTableAdapter.FillByPlatform(GaloreDataSet.Games, txtPlatform.Text.Trim)
26             Case radRating.Checked ' + txtRating
27                 GamesTableAdapter.FillByRating(GaloreDataSet.Games, txtRating.Text.Trim)
28             Case radNew.Checked
29                 GamesTableAdapter.FillByNewUsed_New(GaloreDataSet.Games)
30             Case radUsed.Checked
31                 GamesTableAdapter.FillByNewUsed_Used(GaloreDataSet.Games)
32             Case Else ' radAll.Checked
33                 GamesTableAdapter.Fill(GaloreDataSet.Games)
34         End Select
35     End Sub
36 End Class
```

# Games Galore

| ID | Title                        | Platform | Rating | Price | New/Used | Quantity |
|----|------------------------------|----------|--------|-------|----------|----------|
| 1  | Dead Space                   | XB       | M      | 59.99 | N        | 5        |
| 2  | Dead Space                   | PS       | M      | 59.99 | N        | 4        |
| 3  | Just Dance                   | WII      | E10+   | 37.50 | N        | 3        |
| 4  | Just Dance                   | XB       | E10+   | 37.35 | N        | 4        |
| 5  | Just Dance                   | PS       | E10+   | 34.39 | N        | 4        |
| 6  | Call of Duty: Black Ops II   | XB       | M      | 49.44 | N        | 2        |
| 7  | Call of Duty: Black Ops II   | PS       | M      | 52.43 | N        | 2        |
| 8  | Halo                         | XB       | M      | 39.99 | N        | 2        |
| 9  | Halo                         | XB       | M      | 33.55 | U        | 1        |
| 10 | LEGO Batman: DC Super Heroes | WII      | E10+   | 19.99 | N        | 4        |
| 11 | LEGO Batman: DC Super Heroes | WII      | E10+   | 10.39 | U        | 2        |
| 12 | LEGO Batman: DC Super Heroes | XB       | E10+   | 19.99 | N        | 3        |
| 13 | LEGO Batman: DC Super Heroes | XB       | E10+   | 17.50 | U        | 1        |

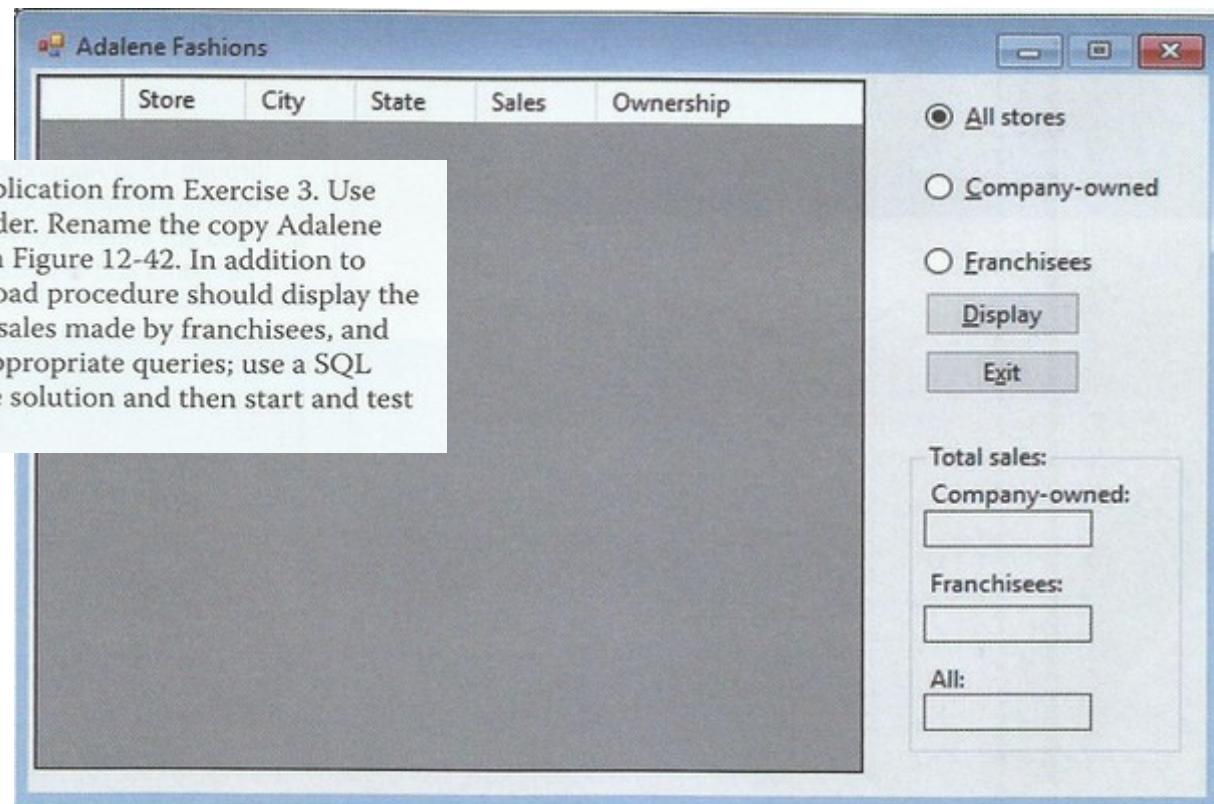
Total value of the games sold: **\$ 3,732.59**

All  
 Platform   
 Rating   
 New  
 Used

**Display** **Exit**

**14.Adalene Solution-TotalSales\_EXERCISE 7\_advanced**

- 1). add 3 **TableAdapters**
- 2). for each, create Query using **Aggregate** function
- 3). drag each **DataColumn** to a different label control = create bound controls by dragging



7. In this exercise, you modify the Adalene Fashions application from Exercise 3. Use Windows to make a copy of the Adalene Solution folder. Rename the copy Adalene Solution-TotalSales. Modify the interface as shown in Figure 12-42. In addition to displaying the records in the dataset, the frmMain\_Load procedure should display the total sales made by company-owned stores, the total sales made by franchisees, and the total sales made by both types of stores. Create appropriate queries; use a SQL aggregate function to make each calculation. Save the solution and then start and test the application.

Data Sources

AdaleneDataSet.xsd\* Main Form.vb Main Form.vb [Design] \_info0\_original EXERCISE 3.jpg

**Stores**

- Store
- City
- State
- Sales
- Ownership

**StoresTableAdapter**

- SQL: Fill, GetData()
- SQL: FillByEmpty, GetDataByEmpty()
- SQL: FillByOwnership\_Company, GetDataByOwnership\_Company()
- SQL: FillByOwnership\_Franchisee, GetDataByOwnership\_Franchisee()

**SalesAll**

- TotalAll

**SalesAllTableAdapter**

- SQL: FillByAll, GetDataByAll()

**SalesCompany**

- TotalCompany

**SalesCompanyTableAdapter**

- SQL: FillByCompany, GetDataByCompany()

**SalesFranchisees**

- TotalFranchisees

**SalesFranchiseesTableAdapter**

- SQL: FillByFranchisees, GetDataByFranchisees()

| StoresTableAdapter - already created in: 10.Adalene Solution_EXERCISE 3_intermediate |                                                                 |                                                                                                |  |
|--------------------------------------------------------------------------------------|-----------------------------------------------------------------|------------------------------------------------------------------------------------------------|--|
| SQL statement:                                                                       | associated with a methods:                                      | info:                                                                                          |  |
| SELECT Store, City, State, Sales, Ownership FROM dbo.Stores                          | Fill()<br>GetData()                                             | TableAdapter default SQL statement<br>select all                                               |  |
| SELECT Store, City, State, Sales, Ownership<br>FROM Stores<br>WHERE Ownership = 'd'  | FillByEmpty()<br>GetDataByEmpty()                               | TableAdapter Query<br>my idea, using non-existing value for<br>a method to display empty cells |  |
| SELECT Store, City, State, Sales, Ownership<br>FROM Stores<br>WHERE Ownership = 'C'  | FillByOwnership_Company()<br>GetDataByOwnership_Company()       | TableAdapter Query<br>selects only Company - C                                                 |  |
| SELECT Store, City, State, Sales, Ownership<br>FROM Stores<br>WHERE Ownership = 'F'  | FillByOwnership_Franchisee()<br>GetDataByOwnership_Franchisee() | TableAdapter Query<br>select only Franchisees - F                                              |  |

| TableAdapter:                | Query with a SQL aggregate function:                                            | associated with a methods:                      | DataTable:       |
|------------------------------|---------------------------------------------------------------------------------|-------------------------------------------------|------------------|
| SalesAllTableAdapter         | SELECT SUM(Sales) AS TotalAll<br>FROM Stores                                    | FillByAll ()<br>GetDataByAll ()                 | TotalAll         |
| SalesCompanyTableAdapter     | SELECT SUM(Sales) AS TotalCompany<br>FROM Stores<br>WHERE (Ownership = 'C')     | FillByCompany ()<br>GetDataByCompany ()         | TotalCompany     |
| SalesFranchiseesTableAdapter | SELECT SUM(Sales) AS TotalFranchisees<br>FROM Stores<br>WHERE (Ownership = 'F') | FillByFranchisees ()<br>GetDataByFranchisees () | TotalFranchisees |

```
1  ' Name:      Adalene Project
2  ' Purpose:    Select and view records.
3  ' Programmer: <your name> on <current date>
4  Option Explicit On
5  Option Strict On
6  Option Infer Off
7
8  Public Class frmMain
9      Private Sub FrmMain_Load(sender As Object, e As EventArgs) Handles MyBase.Load
10         'TODO: This line of code loads data into the 'AdaleneDataSet.SalesFranchisees' table. You can move, or remove it, as needed.
11         Me.SalesFranchiseesTableAdapter.FillByFranchisees(Me.AdaleneDataSet.SalesFranchisees)
12         'TODO: This line of code loads data into the 'AdaleneDataSet.SalesCompany' table. You can move, or remove it, as needed.
13         Me.SalesCompanyTableAdapter.FillByCompany(Me.AdaleneDataSet.SalesCompany)
14         'TODO: This line of code loads data into the 'AdaleneDataSet.SalesAll' table. You can move, or remove it, as needed.
15         Me.SalesAllTableAdapter.FillByAll(Me.AdaleneDataSet.SalesAll)
16         'TODO: This line of code loads data into the 'AdaleneDataSet.Stores' table. You can move, or remove it, as needed.
17         Me.StoresTableAdapter.Fill(Me.AdaleneDataSet.Stores)
18     End Sub
19
20     Private Sub BtnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
21         Me.Close()
22     End Sub
23
24     Private Sub BtnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
25         If radCompany.Checked Then ' display only Ownership = 'C'
26             StoresTableAdapter.FillByOwnership_Company(AdaleneDataSet.Stores)
27         ElseIf radFranchisee.Checked Then ' display only Ownership = 'F'
28             StoresTableAdapter.FillByOwnership_Franchisee(AdaleneDataSet.Stores)
29         Else ' display default all - C + F
30             StoresTableAdapter.Fill(AdaleneDataSet.Stores)
31         End If
32     End Sub
33
34     Private Sub Allrad_CheckedChanged(sender As Object, e As EventArgs) Handles radAll.CheckedChanged, radCompany.CheckedChanged,
35         radFranchisee.CheckedChanged
36         ' my idea for clearing values when (rad)s changed.
37         StoresTableAdapter.FillByEmpty(AdaleneDataSet.Stores)
38     End Sub
39 End Class
```

 Adalene Fashions

|   | Store | City          | State | Sales   | Ownership |
|---|-------|---------------|-------|---------|-----------|
| ▶ | 100   | San Francisco | CA    | 236,700 | C         |
|   | 101   | San Diego     | CA    | 125,900 | C         |
|   | 102   | Burbank       | CA    | 96,575  | F         |
|   | 103   | Chicago       | IL    | 135,400 | C         |
|   | 104   | Chicago       | IL    | 108,000 | F         |
|   | 105   | Denver        | CO    | 212,600 | C         |
|   | 106   | Atlanta       | GA    | 123,500 | C         |
|   | 107   | Louisville    | KY    | 178,500 | C         |
|   | 108   | Lexington     | KY    | 167,450 | F         |
|   | 109   | Nashville     | TN    | 205,625 | C         |
|   | 110   | Atlanta       | GA    | 198,600 | F         |
|   | 111   | Denver        | CO    | 45,900  | F         |
|   | 112   | Miami         | FL    | 175,300 | C         |
|   | 113   | Las Vegas     | NV    | 245,675 | C         |
|   | 114   | New Orleans   | LA    | 213,400 | C         |
|   | 115   | Louisville    | KY    | 68,900  | F         |
|   | 116   | Las Vegas     | NV    | 110,340 | F         |
|   | 118   | Indianapolis  | IN    | 97,500  | C         |
|   | 119   | Raleigh       | NC    | 86,400  | C         |
|   | 120   | San Francisco | CA    | 65,975  | F         |

 All stores Company-owned Franchisees

Display

Exit

Total sales:

Company-owned:

\$ 2,036,500

Franchisees:

\$ 861,740

All:

\$ 2,898,240

## 15.OnYourOwn Solution\_EXERCISE 8

- create **Database & DataSet & Table**
- create **Calculated field** by modifying default SQL statement
- use aggregate function **SUM**
- step by step

8. Create a Windows Forms application. Use the following names for the project and solution, respectively: OnYourOwn Project and OnYourOwn Solution. Save the application in the VB2017\Chap12 folder. Plan and design an application of your choice. The only requirement is that you must follow the minimum guidelines listed in Figure 12-43. Before starting the application, be sure to verify the name of the startup form. Save the solution and then start and test the application.

1. You must create a SQL Server database that contains at least one table and at least four fields.
2. The application must use at least one SELECT statement in addition to the default SELECT statement.
3. The interface must follow the GUI design guidelines summarized for Chapters 2 through 12 in Appendix A.
4. Objects that are either coded or referred to in code should be named appropriately.
5. If you are entering any code in the Code Editor window, then the window must contain comments and the three Option statements.

Figure 12-43 Guidelines for Exercise 8

1. create database **\*.mdf**
2. add new table **dbo.\***
3. define header columns
4. fill the rows-records with data
5. create dataset **\*DataSet.xsd**
6. open dataset Designer
7. add Calculated fields by modifying default SQL statement
8. create SQL Queries & methods association
9. create GUI
10. drag table to GUI
11. code the GUI using SQL Queries methods

- menu: **Project / Add New Item... Ctrl+Shift+A / Data / Service-based Database**  
window: **Server Explorer / Tables** -> Rclick: **Add New Table** = window **dbo.\* [Design]**  
window: **dbo.\* [Design]**  
window: **Server Explorer / Tables / \*** -> Rclick: **Show table Data** = window **dbo.\* [Data]**  
window: **Data Sources / Add New Data Sources...**  
window: **Data Sources / \*DataSet** -> Rclick: **Edit DataSet with Designer** = window **\*DataSet.xsd**  
window **\*DataSet.xsd / \*TableAdapter**  
window **\*DataSet.xsd / \*TableAdapter**  
window: **Main Form.vb [Design]**

MyDatabaseDataSet.xsd\* Main Form.vb Main Form.vb [Design] \_info.jpg

**Data Sources**

- MyDatabaseDataSet
  - MyTable
    - ID
    - FirstName
    - LastName
    - PerHour
    - HoursWorked
    - Total Salary
  - TotalSums
    - TotalHours
    - TotalSalaries

**MyTable**

- ID
- FirstName
- LastName
- PerHour
- HoursWorked
- Total Salary

**MyTableTableAdapter**

- Fill,GetData ()
- FillBy\_Empty,GetDataBy\_Empty ()
- FillBy\_rad2PerHourAsc,GetDataBy\_rad2PerHourAsc ()
- FillBy\_rad3PerHourDes,GetDataBy\_rad3PerHourDes ()
- FillBy\_rad4HoursWorkedAsc,GetDataBy\_rad4HoursWorkedAsc ()
- FillBy\_rad5HoursWorkedDes,GetDataBy\_rad5HoursWorkedDes ()
- FillBy\_rad6TotalAsc,GetDataBy\_rad6TotalAsc ()
- FillBy\_rad7TotalDes,GetDataBy\_rad7TotalDes ()

**TotalSums**

- TotalHours
- TotalSalaries

**TotalSumsTableAdapter**

- Fill,GetData ()

## MyTableTableAdapter

### SQL statement:

```
SELECT ID, FirstName, LastName, PerHour, HoursWorked, PerHour * HoursWorked AS [Total Salary]
FROM MyTable
```

### associated with a methods:

```
Fill ()
GetData ()
```

**TableAdapter default SQL statement modified - contains a calculated field rad1IdAsc.Checked**

```
SELECT ID, FirstName, LastName, PerHour, HoursWorked, PerHour * HoursWorked AS [Total Salary]
FROM MyTable
WHERE (FirstName = 'X')
```

```
FillBy_Empty ()
GetDataBy_Empty ()
```

**TableAdapter Query my idea to clear DataSet**

```
SELECT ID, FirstName, LastName, PerHour, HoursWorked, PerHour * HoursWorked AS [Total Salary]
FROM MyTable
ORDER BY PerHour
```

```
FillBy_rad2PerHourAsc ()
GetDataBy_rad2PerHourAsc ()
```

**TableAdapter Query rad2PerHourAsc.Checked**

```
SELECT ID, FirstName, LastName, PerHour, HoursWorked, PerHour * HoursWorked AS [Total Salary]
FROM MyTable
ORDER BY PerHour DESC
```

```
FillBy_rad3PerHourDes ()
GetDataBy_rad3PerHourDes ()
```

**TableAdapter Query rad3PerHourDes.Checked**

```
SELECT ID, FirstName, LastName, PerHour, HoursWorked, PerHour * HoursWorked AS [Total Salary]
FROM MyTable
ORDER BY HoursWorked
```

```
FillBy_rad4HoursWorkedAsc ()
GetDataBy_rad4HoursWorkedAsc ()
```

**TableAdapter Query rad4HoursWorkedAsc.Checked**

```
SELECT ID, FirstName, LastName, PerHour, HoursWorked, PerHour * HoursWorked AS [Total Salary]
FROM MyTable
ORDER BY HoursWorked DESC
```

```
FillBy_rad5HoursWorkedDes ()
GetDataBy_rad5HoursWorkedDes ()
```

**TableAdapter Query rad5HoursWorkedDes.Checked**

|                                                                                                                                               |                                                     |                                                          |
|-----------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------|----------------------------------------------------------|
| SELECT ID, FirstName, LastName, PerHour, HoursWorked, PerHour * HoursWorked AS [Total Salary]<br>FROM MyTable<br>ORDER BY [Total Salary]      | FillBy_rad6TotalAsc ()<br>GetDataBy_rad6TotalAsc () | <b>TableAdapter Query</b><br><b>rad6TotalAsc.Checked</b> |
| SELECT ID, FirstName, LastName, PerHour, HoursWorked, PerHour * HoursWorked AS [Total Salary]<br>FROM MyTable<br>ORDER BY [Total Salary] DESC | FillBy_rad7TotalDes ()<br>GetDataBy_rad7TotalDes () | <b>TableAdapter Query</b><br><b>rad7TotalDes.Checked</b> |

### TotalSumsTableAdapter

| <b>SQL statement:</b>                                                                              | <b>associated with a methods:</b> | <b>info:</b>                                                                   |
|----------------------------------------------------------------------------------------------------|-----------------------------------|--------------------------------------------------------------------------------|
| SELECT SUM(HoursWorked) AS TotalHours, SUM(PerHour * HoursWorked) AS TotalSalaries<br>FROM MyTable | Fill ()<br>GetData ()             | <b>TableAdapter default SQL statement</b><br><b>SQL aggregate function SUM</b> |

```
1  Public Class frmMain
2      Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles MyBase.Load
3          'TODO: This line of code loads data into the 'MyDatabaseDataSet.TotalSums' table. You can move, or remove it, as needed.
4          ' contains SQL aggregate function SUM:
5          ' SUM(HoursWorked) AS TotalHours -> creating a new field TotalHours
6          ' SUM(PerHour * HoursWorked) AS TotalSalaries -> creating a new field TotalSalaries
7          Me.TotalSumsTableAdapter.Fill(Me.MyDatabaseDataSet.TotalSums)
8
9          'TODO: This line of code loads data into the 'MyDatabaseDataSet.MyTable' table. You can move, or remove it, as needed.
10         ' contains a Calculated field created by modifying the default SQL statement:
11         '...PerHour * HoursWorked AS [Total Salary]
12         Me.MyTableTableAdapter.Fill(Me.MyDatabaseDataSet.MyTable)
13     End Sub
14
15     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
16         Me.Close()
17     End Sub
18
19     Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
20         Select Case True
21             Case rad1IdAsc.Checked
22                 MyTableTableAdapter.Fill(MyDatabaseDataSet.MyTable)
23             Case rad2PerHourAsc.Checked
24                 MyTableTableAdapter.FillBy_rad2PerHourAsc(MyDatabaseDataSet.MyTable)
25             Case rad3PerHourDes.Checked
26                 MyTableTableAdapter.FillBy_rad3PerHourDes(MyDatabaseDataSet.MyTable)
27             Case rad4HoursWorkedAsc.Checked
28                 MyTableTableAdapter.FillBy_rad4HoursWorkedAsc(MyDatabaseDataSet.MyTable)
29             Case rad5HoursWorkedDes.Checked
30                 MyTableTableAdapter.FillBy_rad5HoursWorkedDes(MyDatabaseDataSet.MyTable)
31             Case rad6TotalAsc.Checked
32                 MyTableTableAdapter.FillBy_rad6TotalAsc(MyDatabaseDataSet.MyTable)
33             Case rad7TotalDes.Checked
34                 MyTableTableAdapter.FillBy_rad7TotalDes(MyDatabaseDataSet.MyTable)
35         End Select
36     End Sub
37
38     Private Sub radAll_CheckedChanged(sender As Object, e As EventArgs) Handles rad1IdAsc.CheckedChanged, rad2PerHourAsc.CheckedChanged,
39   rad3PerHourDes.CheckedChanged, rad4HoursWorkedAsc.CheckedChanged, rad5HoursWorkedDes.CheckedChanged,
40   rad6TotalAsc.CheckedChanged, rad6TotalDes.CheckedChanged
41         ' my idea for a čistírna: I created a Query whose condition can't be fulfilled, therefore it shows nothing:
42         MyTableTableAdapter.FillBy_Empty(MyDatabaseDataSet.MyTable)
43     End Sub
44 End Class
```

15.OnYourOwn Solution\_EXERCISE 8

|   | ID | FirstName   | LastName     | PerHour   | HoursWorked | Total Salary |
|---|----|-------------|--------------|-----------|-------------|--------------|
| ▶ | 1  | Michael     | Jonkens      | \$ 16.00  | 160.00      | \$ 2,560.00  |
|   | 2  | Jean-Claude | Baptiste     | \$ 14.50  | 163.25      | \$ 2,367.13  |
|   | 3  | Oscar       | Neubauer     | \$ 13.25  | 80.50       | \$ 1,066.63  |
|   | 4  | Josephine   | Mutzenbacher | \$ 120.00 | 4.00        | \$ 480.00    |
|   | 5  | Laszlo      | Orban        | \$ 16.25  | 160.00      | \$ 2,600.00  |
|   | 6  | Ramses      | Onseng       | \$ 18.50  | 168.00      | \$ 3,108.00  |
|   | 7  | Che         | Guebada      | \$ 16.00  | 168.00      | \$ 2,688.00  |

Total Hours: 903.75      Total Salaries: \$ 14,869.75

Display by:

ID number ascending  
 PerHour ascending  
 PerHour descending  
 HoursWorked ascending  
 HoursWorked descending  
 Total ascending  
 Total descending

Display
Exit

15.OnYourOwn Solution\_EXERCISE 8

|   | ID | FirstName   | LastName     | PerHour   | HoursWorked | Total Salary |
|---|----|-------------|--------------|-----------|-------------|--------------|
| ▶ | 4  | Josephine   | Mutzenbacher | \$ 120.00 | 4.00        | \$ 480.00    |
|   | 3  | Oscar       | Neubauer     | \$ 13.25  | 80.50       | \$ 1,066.63  |
|   | 2  | Jean-Claude | Baptiste     | \$ 14.50  | 163.25      | \$ 2,367.13  |
|   | 1  | Michael     | Jonkens      | \$ 16.00  | 160.00      | \$ 2,560.00  |
|   | 5  | Laszlo      | Orban        | \$ 16.25  | 160.00      | \$ 2,600.00  |
|   | 7  | Che         | Guebada      | \$ 16.00  | 168.00      | \$ 2,688.00  |
|   | 6  | Ramses      | Onseng       | \$ 18.50  | 168.00      | \$ 3,108.00  |

Total Hours: 903.75      Total Salaries: \$ 14,869.75

Display by:

ID number ascending  
 PerHour ascending  
 PerHour descending  
 HoursWorked ascending  
 HoursWorked descending  
 Total ascending  
 Total descending

Display
Exit

15.OnYourOwn Solution\_EXERCISE 8

| ID | FirstName | LastName | PerHour | HoursWorked | Total Salary |
|----|-----------|----------|---------|-------------|--------------|
|    |           |          |         |             |              |

Display by:

- ID number ascending
- PerHour ascending
- PerHour descending
- HoursWorked ascending
- HoursWorked descending
- Total ascending
- Total descending

Total Hours:

Total Salaries:

15.OnYourOwn Solution\_EXERCISE 8

|   | ID | FirstName   | LastName     | PerHour   | HoursWorked | Total Salary |
|---|----|-------------|--------------|-----------|-------------|--------------|
| ▶ | 3  | Oscar       | Neubauer     | \$ 13.25  | 80.50       | \$ 1,066.63  |
|   | 2  | Jean-Claude | Baptiste     | \$ 14.50  | 163.25      | \$ 2,367.13  |
|   | 1  | Michael     | Jonkens      | \$ 16.00  | 160.00      | \$ 2,560.00  |
|   | 7  | Che         | Guebada      | \$ 16.00  | 168.00      | \$ 2,688.00  |
|   | 5  | Laszlo      | Orban        | \$ 16.25  | 160.00      | \$ 2,600.00  |
|   | 6  | Ramses      | Onseng       | \$ 18.50  | 168.00      | \$ 3,108.00  |
|   | 4  | Josephine   | Mutzenbacher | \$ 120.00 | 4.00        | \$ 480.00    |

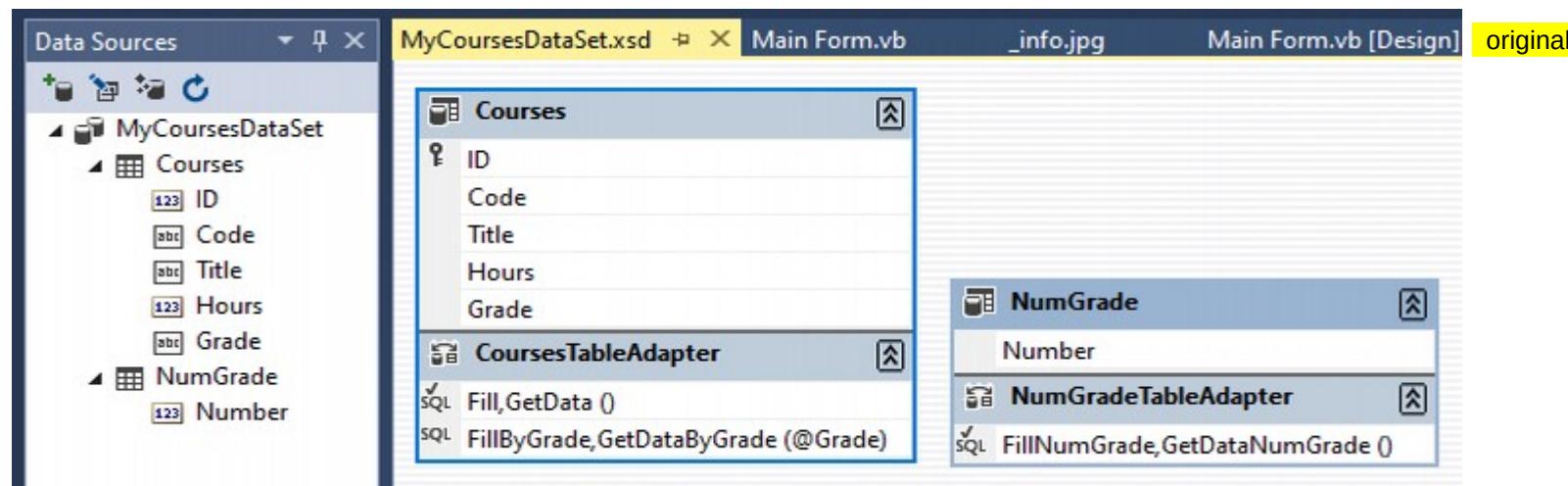
Total Hours:  Total Salaries:

Display by:

- ID number ascending
- PerHour ascending
- PerHour descending
- HoursWorked ascending
- HoursWorked descending
- Total ascending
- Total descending

**16.FixIt Solution\_EXERCISE 9** - to the TableAdapter's modified default SQL statement containing **Aggregate** function - added **parameter @**

9. Open the VB2017\Chap12\FixIt Solution\FixIt Solution.sln file. Start the application. Click the Grade radio button, type the letter a in the Grade box, and then click the Display button. The application should display three records in the DataGridView control. It should also display the number 3 in the **lblCount** control. Notice that the application is not working properly. Fix the application.



original - OK

#### CoursesTableAdapter

| SQL statement:                                                                 | assoc. with a methods & used in code:                                                                                             | info:                                                       |
|--------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------|
| SELECT ID, Code, Title, Hours, Grade FROM dbo.Courses                          | Fill, GetData()<br>Me.CoursesTableAdapter.Fill(Me.MyCoursesDataSet.Courses)<br>CoursesTableAdapter.Fill(MyCoursesDataSet.Courses) | TableAdapter default SQL statement<br>- select all          |
| SELECT ID, Code, Title, Hours, Grade<br>FROM Courses<br>WHERE (Grade = @Grade) | FillByGrade, GetDataByGrade (@Grade)<br>CoursesTableAdapter.FillByGrade(MyCoursesDataSet.Courses, txtGrade.Text.Trim)             | TableAdapter<br>Parameter Query<br>- @Grade = a / b / c / w |

original: **lblCount** shows always the total number of rows = 6, but not the number of rows selected by the user.

#### NumGradeTableAdapter

| SQL statement:                                 | assoc. with a methods & used in code:                                                            | info:                                                                                                                                                                                                                                                               |
|------------------------------------------------|--------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| SELECT Count(Grade) AS Number from dbo.Courses | FillNumGrade, GetDataNumGrade ()<br>NumGradeTableAdapter.FillNumGrade(MyCoursesDataSet.NumGrade) | TableAdapter modified default SQL statement<br>Aggregate function<br>- field <u>Number</u> bound to <b>lblCount</b> by dragging<br>- counts all <u>Grade</u> fields and saves integer to field <u>Number</u> = 6<br>- but it should count only the displayed fields |

My Fix: **1b1Count** shows the number of rows selected by the user - e.g. if the user types "a", the **1b1Count** shows number 3.

## NumGradeTableAdapter

| SQL statement:                                                                          | assoc. with a methods & used in code:                                                              | info:                                                  |
|-----------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------|--------------------------------------------------------|
| 1). original:<br>SELECT Count(Grade) AS Number from dbo.Courses                         | 2). original:<br>NumGradeTableAdapter.FillNumGrade(MyCoursesDataSet.NumGrade)                      | to the SQL statement I just added a parameter          |
| changed for:<br>SELECT Count(Grade) AS Number<br>FROM Courses<br>WHERE (Grade = @Grade) | changed for:<br>NumGradeTableAdapter.FillNumGrade(MyCoursesDataSet.NumGrade, txtGrade.Text.Trim()) | and, of course, changed the method to give it an input |

Query Builder

Courses

- \* (All Columns)
- ID
- Code
- Title
- Hours
- Grade

| Column | Alias  | Table   | Output                              | Sort Type | Sort Order | Group By | Filter   |
|--------|--------|---------|-------------------------------------|-----------|------------|----------|----------|
| Grade  | Number | Courses | <input checked="" type="checkbox"/> |           |            | Count    |          |
| Grade  |        | Courses | <input type="checkbox"/>            |           |            | Where    | = @Grade |

```
SELECT COUNT(Grade) AS Number
FROM Courses
WHERE (Grade = @Grade)
```

| Number |
|--------|
| 3      |

Cell is Read Only.

Execute Query      OK      Cancel

NumGrade

Number

NumGradeTableAdapter

FillNumGrade, GetDataNumGrade (@Grade)

My Fix

my thoughts:

1. use a parameter?
  - tried, but didn't work
2. use a Get method to get somehow an info from the previous method?
  - hmm, seems too complicated
3. do not try to use anything, until u analyse the current state and then u will see.
  - juchuuuu just when i made the tabs above, the solution came just like a swish

**original**

| ID | Code     | Title                       | Hours | Grade |
|----|----------|-----------------------------|-------|-------|
| 1  | ACCOU110 | Accounting Procedures       | 3     | A     |
| 3  | CIS110   | Introduction to Programming | 3     | A     |
| 6  | CIS112   | The Internet                | 2     | A     |

All  
 Grade  
 a       6

**Display**      **Exit**

| ID | Code     | Title                       | Hours | Grade |
|----|----------|-----------------------------|-------|-------|
| 1  | ACCOU110 | Accounting Procedures       | 3     | A     |
| 2  | ENG101   | English Composition         | 3     | W     |
| 3  | CIS110   | Introduction to Programming | 3     | A     |
| 4  | BIO111   | Environmental Biology       | 3     | C     |
| 5  | ENG101   | English Composition         | 3     | B     |
| 6  | CIS112   | The Internet                | 2     | A     |

All  
 Grade  
     

**Display**      **Exit**

**My Fix**

**My Fix**

| ID | Code     | Title                       | Hours | Grade |
|----|----------|-----------------------------|-------|-------|
| 1  | ACCOU110 | Accounting Procedures       | 3     | A     |
| 3  | CIS110   | Introduction to Programming | 3     | A     |
| 6  | CIS112   | The Internet                | 2     | A     |

All  
 Grade  
 a       3

**Display**      **Exit**

All  
 Grade  
 q       0

**Display**      **Exit**

```
1  ' Name:      Course Info Project
2  ' Purpose:    Display all records or only those for a specific grade.
3  ' Programmer: <your name> on <current date>
4  Option Explicit On
5  Option Strict On
6  Option Infer Off
7
8  Public Class frmMain
9      Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles MyBase.Load
10         'TODO: This line of code loads data into the 'MyCoursesDataSet.Courses' table. You can move, or remove it, as needed.
11         Me.CoursesTableAdapter.Fill(Me.MyCoursesDataSet.Courses)
12     End Sub
13
14     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
15         Me.Close()
16     End Sub
17
18     Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
19         ' Display all records or only those for a specific grade.
20
21         If radAll.Checked Then
22             CoursesTableAdapter.Fill(MyCoursesDataSet.Courses)
23             txtGrade.Text = String.Empty
24             lblCount.Text = String.Empty
25         Else
26             If txtGrade.Text.Trim = String.Empty Then
27                 MessageBox.Show("Please enter the grade.", "Course Information", MessageBoxButtons.OK, MessageBoxIcon.Information)
28             Else
29                 CoursesTableAdapter.FillByGrade(MyCoursesDataSet.Courses, txtGrade.Text.Trim)
30
31                 ' original: in (lblCount) always shows number 6 = total number of rows.
32                 ' SQL SELECT Aggregate statement:
33                 '   SELECT Count(Grade) AS Number from dbo.Courses
34                 'NumGradeTableAdapter.FillNumGrade(MyCoursesDataSet.NumGrade)
35
36                 ' My Fix: in (lblCount) shows the number of rows, depending of a input - I used a parameter query
37                 ' SQL SELECT Aggregate & Parameter Query statement:
38                 '   SELECT Count(Grade) AS Number
39                 '   FROM Courses
40                 '   WHERE (Grade = @Grade)
41                 NumGradeTableAdapter.FillNumGrade(MyCoursesDataSet.NumGrade, txtGrade.Text.Trim)
42             End If
43         End If
44     End Sub
```

```
45
46  Private Sub radAll_Click(sender As Object, e As EventArgs) Handles radAll.Click
47      txtGrade.Text = String.Empty
48      lblCount.Text = String.Empty
49  End Sub
50 End Class
```

## Jefferson Realty (Chapters 1–12)

Create a SQL Server database that contains one table named Homes. The table should contain 10 records, each having five fields. The ID field should be an auto-numbered field. The ZIP code field should contain text. Be sure to use at least three different ZIP codes. The number of bedrooms, number of bathrooms, and price fields should be numeric. Create an application that displays the contents of the database in a DataGridView control. The user should not be allowed to add, edit, delete, or save records. The application should allow the user to display the records for a specific number of bedrooms, a specific number of bathrooms, or a specific ZIP code. When the interface appears, it should display the average home price for the entire database.

creation flow:

1. dbo.Homes [Design]
2. dbo.Homes [Data]
3. JeffersonDBDataSet.xsd
4. Main Form.vb [Design]
5. Main Form.vb

## 1. dbo.Homes [Design]

The screenshot shows the 'dbo.Homes [Design]' tab selected in the top bar. Below it is a table structure with columns: Name, Data Type, Allow Nulls, and Default. The table has five rows: ID (int, not null), ZIP (varchar(10), not null), Bedrooms (int, not null), Bathrooms (int, not null), and Price (float, not null). At the bottom, there are tabs for 'Design' and 'T-SQL'. The 'T-SQL' tab is active, displaying the CREATE TABLE script:

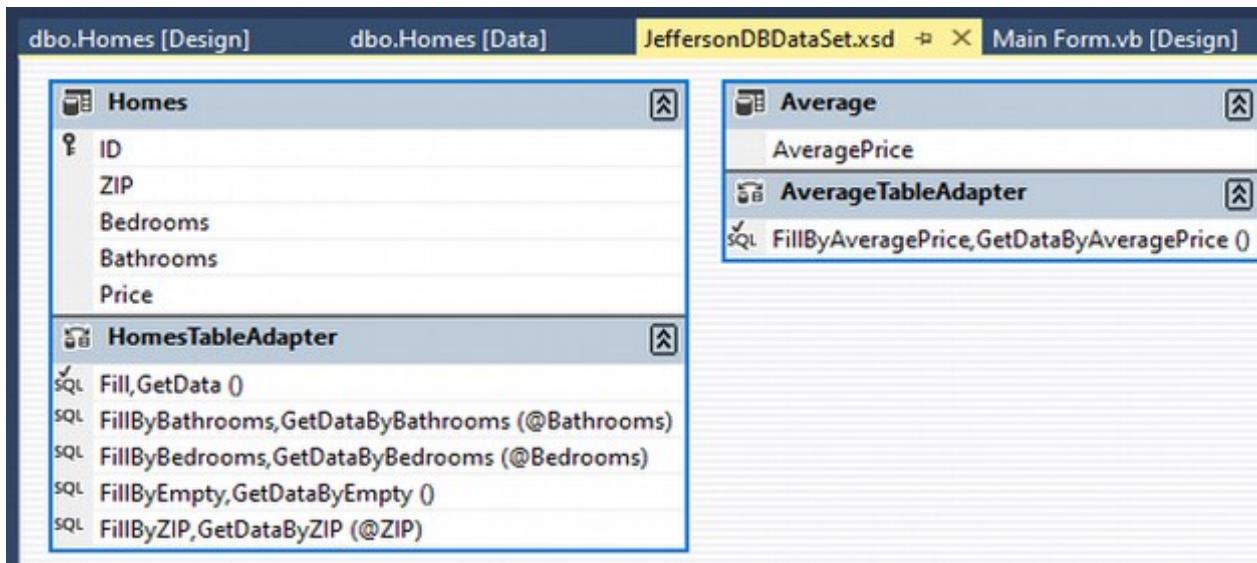
```
1  CREATE TABLE [dbo].[Homes] (
2      [ID]      INT      IDENTITY (1, 1) NOT NULL,
3      [ZIP]      VARCHAR (10) NOT NULL,
4      [Bedrooms] INT      NOT NULL,
5      [Bathrooms] INT      NOT NULL,
6      [Price]    FLOAT (53) NOT NULL,
7      PRIMARY KEY CLUSTERED ([ID] ASC)
8  );
9
10 |
```

## 2. dbo.Homes [Data]

The screenshot shows the 'dbo.Homes [Data]' tab selected in the top bar. It displays a grid of data with columns: ID, ZIP, Bedrooms, Bathrooms, and Price. The data consists of 10 rows, each with unique values for ID and ZIP, and varying values for Bedrooms, Bathrooms, and Price.

|   | ID   | ZIP   | Bedrooms | Bathrooms | Price |
|---|------|-------|----------|-----------|-------|
| ▶ | 1    | 12305 | 4        | 1         | 10000 |
|   | 2    | 12305 | 6        | 2         | 15000 |
|   | 3    | 20649 | 3        | 1         | 8000  |
|   | 4    | 20649 | 6        | 2         | 17000 |
|   | 5    | 30102 | 1        | 1         | 3000  |
|   | 6    | 30102 | 3        | 1         | 8500  |
|   | 7    | 30102 | 10       | 4         | 30000 |
|   | 8    | 40502 | 3        | 1         | 8500  |
|   | 9    | 60501 | 4        | 2         | 12000 |
|   | 10   | 90210 | 8        | 3         | 20000 |
| * | NULL | NULL  | NULL     | NULL      | NULL  |

### 3. JeffersonDBDataSet.xsd



```
1  ' Name:      Jefferson Realty.
2  ' Purpose:    Display houses.
3  ' Programmer: Me on just now.
4  Option Explicit On
5  Option Strict On
6  Option Infer Off
7
8  Public Class frmMain
9
10   ' automatically generated code:
11   Private Sub frmMain_Load(sender As Object, e As EventArgs) Handles MyBase.Load
12     'TODO: This line of code loads data into the 'JeffersonDBDataSet.Average' table. You can move, or remove it, as needed.
13     ' Aggregate Function AVG used for a cells Price
14     Me.AverageTableAdapter.FillByAveragePrice(Me.JeffersonDBDataSet.Average)
15
16     'TODO: This line of code loads data into the 'JeffersonDBDataSet.Homes' table. You can move, or remove it, as needed.
17     Me.HomesTableAdapter.Fill(Me.JeffersonDBDataSet.Homes)
18   End Sub
19
20   Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
21     Me.Close()
22   End Sub
23
```

```
24  Private Sub btnSubmit_Click(sender As Object, e As EventArgs) Handles btnSubmit.Click
25      Dim intTextBox As Integer : Integer.TryParse(txtSearch.Text, intTextBox)
26
27      ' Independent Sub - if the input search string empty:
28      SearchEmpty()
29
30      Select Case True
31          Case rad2Bedroom.Checked : HomesTableAdapter.FillByBedrooms(JeffersonDBDataSet.Homes, intTextBox)
32          Case rad3Bathroom.Checked : HomesTableAdapter.FillByBathrooms(JeffersonDBDataSet.Homes, intTextBox)
33          Case rad4Zip.Checked : HomesTableAdapter.FillByZIP(JeffersonDBDataSet.Homes, intTextBox.ToString())
34      End Select
35  End Sub
36
37  Private Sub rad1All_CheckedChanged(sender As Object, e As EventArgs) Handles rad1All.CheckedChanged
38
39      ' fill the DataGridView with all you've got:
40      Me.HomesTableAdapter.Fill(Me.JeffersonDBDataSet.Homes)
41
42      ' disable the txtSearch control:
43      txtSearch.Enabled = False
44
45      ' disable the btnSubmit control:
46      btnSubmit.Enabled = False
47  End Sub
48
49  Private Sub radsFilter_CheckedChanged(sender As Object, e As EventArgs) Handles rad2Bedroom.CheckedChanged, rad3Bathroom.CheckedChanged,
50   rad4Zip.CheckedChanged
51
52      ' when the rad controls with requirements checked changed:
53
54      ' enable the btnSubmit control:
55      btnSubmit.Enabled = True
56
57      ' enable the txtSearch:
58      txtSearch.Enabled = True
59
60      ' clear the input control & give him the focus:
61      txtSearch.Text = Nothing : txtSearch.Focus()
62
63      ' clear the table:
64      HomesTableAdapter.FillByEmpty(JeffersonDBDataSet.Homes)
65  End Sub
```

```

66      ' Independent Sub - if the search input string empty (true, no need - could just write it in the procedure - but i want :D )
67      Private Sub SearchEmpty()
68          If txtSearch.Text.Trim = Nothing Then
69              MessageBox.Show("Please enter some search value.", "Jefferson Realty", MessageBoxButtons.OK, MessageBoxIcon.Information)
70          End If
71
72          ' txtSearch control - select all the values & give him the focus:
73          txtSearch.SelectAll() : txtSearch.Focus()
74      End Sub
75
76      Private Sub txtSearch_KeyPress(sender As Object, e As KeyPressEventArgs) Handles txtSearch.KeyPress
77          ' allow only integers & the Backspace key in txtSearch control:
78          If (e.KeyChar < "0" OrElse e.KeyChar > "9") AndAlso e.KeyChar <> ControlChars.Back Then
79              e.Handled = True
80          End If
81      End Sub
82  End Class

```

12.Jefferson Realty-CH01-12

|   | ID | ZIP   | Bedrooms | Bathrooms | Price     |
|---|----|-------|----------|-----------|-----------|
| ▶ | 1  | 12305 | 4        | 1         | \$ 10,000 |
|   | 2  | 12305 | 6        | 2         | \$ 15,000 |
|   | 3  | 20649 | 3        | 1         | \$ 8,000  |
|   | 4  | 20649 | 6        | 2         | \$ 17,000 |
|   | 5  | 30102 | 1        | 1         | \$ 3,000  |
|   | 6  | 30102 | 3        | 1         | \$ 8,500  |
|   | 7  | 30102 | 10       | 4         | \$ 30,000 |
|   | 8  | 40502 | 3        | 1         | \$ 8,500  |
|   | 9  | 60501 | 4        | 2         | \$ 12,000 |
|   | 10 | 90210 | 8        | 3         | \$ 20,000 |

Average home price. \$ 13,200

Display all  
or filter by:

Number of bedrooms  
 Number of bathrooms  
 Zip code

Enter your search:

Submit

Exit

Jefferson Realty

Please enter some search value.

OK

## 12.Jefferson Realty-CH01-12

|   | ID | ZIP   | Bedrooms | Bathrooms | Price     |
|---|----|-------|----------|-----------|-----------|
| ▶ | 1  | 12305 | 4        | 1         | \$ 10,000 |
|   | 9  | 60501 | 4        | 2         | \$ 12,000 |

Average home price. 

- Display all  
or filter by:  
 Number of bedrooms  
 Number of bathrooms  
 Zip code

Enter your search:

## 12.Jefferson Realty-CH01-12

|   | ID | ZIP   | Bedrooms | Bathrooms | Price     |
|---|----|-------|----------|-----------|-----------|
| ▶ | 5  | 30102 | 1        | 1         | \$ 3,000  |
|   | 6  | 30102 | 3        | 1         | \$ 8,500  |
|   | 7  | 30102 | 10       | 4         | \$ 30,000 |

Average home price. 

- Display all  
or filter by:  
 Number of bedrooms  
 Number of bathrooms  
 Zip code

Enter your search:


The end, Ende, Fine, Konec, Basta.

Site.master; 2x Static Web pages: Default.aspx & About.aspx; Dynamic Web page: Calculator.aspx recycled from unused Contact.aspx  
 Start Without Debugging Ctrl+F5; user input Validation Tools

**Figure 13-7** 4 of the files included in the ASP.NET Web Forms Site template:

| filename     | purpose                                                                                                                  |
|--------------|--------------------------------------------------------------------------------------------------------------------------|
| About.aspx   | contains information about the website, such as its history or purpose                                                   |
| Contact.aspx | provides contact information, such as a phone number or e-mail address                                                   |
| Default.aspx | serves as the home page                                                                                                  |
| Site.master  | creates a consistent layout for the pages in your application (for example, the same color, header, footer, and buttons) |

**Figure 13-53** Toolbox / Validation controls:

CH13\_A4 - ...

|                            |                                                                                 |
|----------------------------|---------------------------------------------------------------------------------|
| CompareValidator           | compare an entry with a constant value or with a control's property             |
| CustomValidator            | verify that an entry passes the specified validation logic                      |
| RangeValidator             | verify that an entry is within the specified minimum and maximum values         |
| RegularExpressionValidator | verify that an entry matches a specific pattern                                 |
| RequiredFieldValidator     | verify that a control contains data                                             |
| ValidationSummary          | display all of the validation error messages in a single location on a Web page |

#### CH13\_FOCUS ON THE CONCEPTS LESSON

- CH13\_F1 - basic web terminology & ASP.NET basic info
- CH13\_F2 - template **ASP.NET Web Forms Site** to create a Web Site Application (using files: **About.aspx**, **Contact.aspx**, **Default.aspx**, **Site.master**)  
 (VS 2017 v 15.9.36 & .NET Framework 4.5.2) e.g. with **01.Fishbowl 1/9**
- CH13\_F3 - how to add **Start Without Debugging** option to the **Debug** menu in **VS 2017** step by step instructions &  
 & starting/testing a **Web Application** in VS 2017 v 15.9.36 **without debugging** option (**Ctrl + F5**) e.g. with **01.Fishbowl 2/9**
- CH13\_F4 - modify the **0th** page - master page **Site.master** (the page creates a consistent layout for the pages in your app = affects all of the **.aspx** pages)  
 & brief info about tags: **<p>**, **<ul>**, **<li>**, **<% %>**, e.g. with **01.Fishbowl 3/9**
- CH13\_F5 - create/personalize the **1st** page - **Static** Web page **Default.aspx** (Home page), e.g. with **01.Fishbowl 4/9**
- CH13\_F6 - create/personalize the **2nd** page - **Static** Web page **About.aspx**, e.g. with **01.Fishbowl 5/9**
- CH13\_F7 - starting/testing **Web Application** in VS 2017 v 15.9.36 with different browsers
- CH13\_F8 - closing and opening a Web Site Application - VS 2017 book version vs updated VS 2017 v 15.9.36

#### CH13\_APPLY THE CONCEPTS LESSON

- CH13\_A1 - create **3rd** page - **Dynamic** Web page **Calculator.aspx**: 1/4 Repurpose an existing unused **Contact.aspx**, e.g. with **01.Fishbowl 6/9**
- CH13\_A2 - create **3rd** page - **Dynamic** Web page **Calculator.aspx**: 2/4 add a **Table** and **Controls**, e.g. with **01.Fishbowl 7/9**
- CH13\_A3 - create **3rd** page - **Dynamic** Web page **Calculator.aspx.vb**: 3/4 code a **Button Control**, e.g. with **01.Fishbowl 8/9**
- CH13\_A4 - create **3rd** page - **Dynamic** Web page **Calculator.aspx**: 4/4 use a **Validation Control: RequiredFieldValidator** - basic info, e.g. with **01.Fishbowl 9/9**
- CH13\_X1 - **01.Fishbowl** application code & GUI for the pages:  
**Site.master** & **Site.master.vb** & **Default.aspx** & **Default.aspx.vb** & **About.aspx** & **About.aspx.vb** & **Calculator.aspx** & **Calculator.aspx.vb**
- CH13\_X2 - **ASP.NET Web Forms Site** template - automatically generated code for the pages:  
**Site.master** & **Default.aspx** & **About.aspx** & **Contact.aspx** & **Contact.aspx.vb**

CH13\_Summary

CH13\_Key Terms

CH13\_Exercises

## CH13\_FOCUS ON THE CONCEPTS LESSON

### CH13\_F1 - basic web terminology & ASP.NET basic info

- the Internet is the world's largest computer network, connecting millions of computers located all around the world
  - one of the most popular features of the Internet is the World Wide Web - **Web**
  - the **Web** consists of documents called **Web pages** that are stored on **Web servers**
  - a **Web server** is a computer that contains special software that "serves up" Web pages in response to requests from **client computers**
  - a **client computer** is a computer that requests information from a **Web server**
  - the information is requested and subsequently [následně] viewed through the use of a program called a **Web browser** - **browser**
  - examples of popular browsers include Microsoft Edge, Google Chrome, Mozilla Firefox, and Safari
- 
- **a Web pages** can be static or dynamic
  - a **static Web page** - is a document whose purpose is merely to display information to the viewer  
**Figure 13-1** an example of a static Web page that displays a store's name, address, and telephone number
  - a **dynamic Web page** - unlike a static Web page, is interactive in that it can accept information from the user and also retrieve information for the user
    - examples of dynamic Web pages include forms for purchasing merchandise online and for submitting online résumés

**Figure 13-2** a dynamic Web page that calculates the number of gallons of water a rectangular aquarium holds. To use the Web page, you enter the length, width, and height of the aquarium and then click the Submit button.

The button's Click event procedure displays the corresponding number of gallons on the Web page.

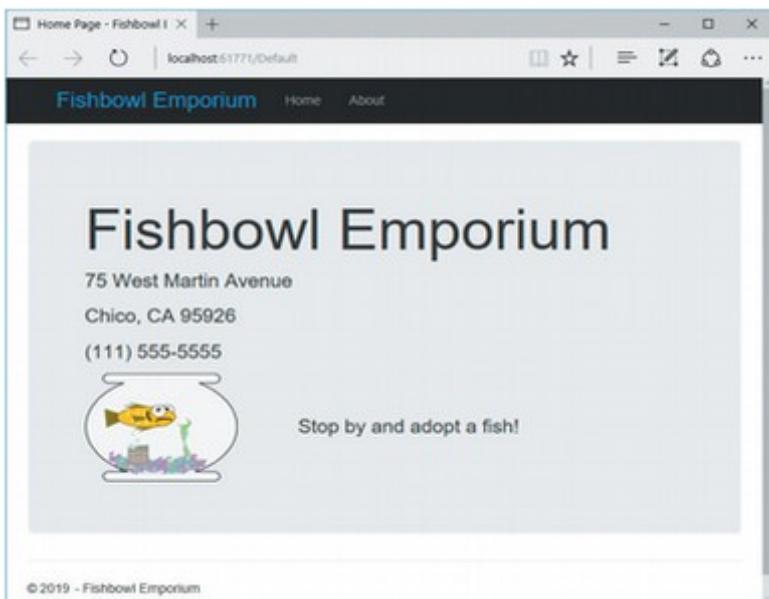
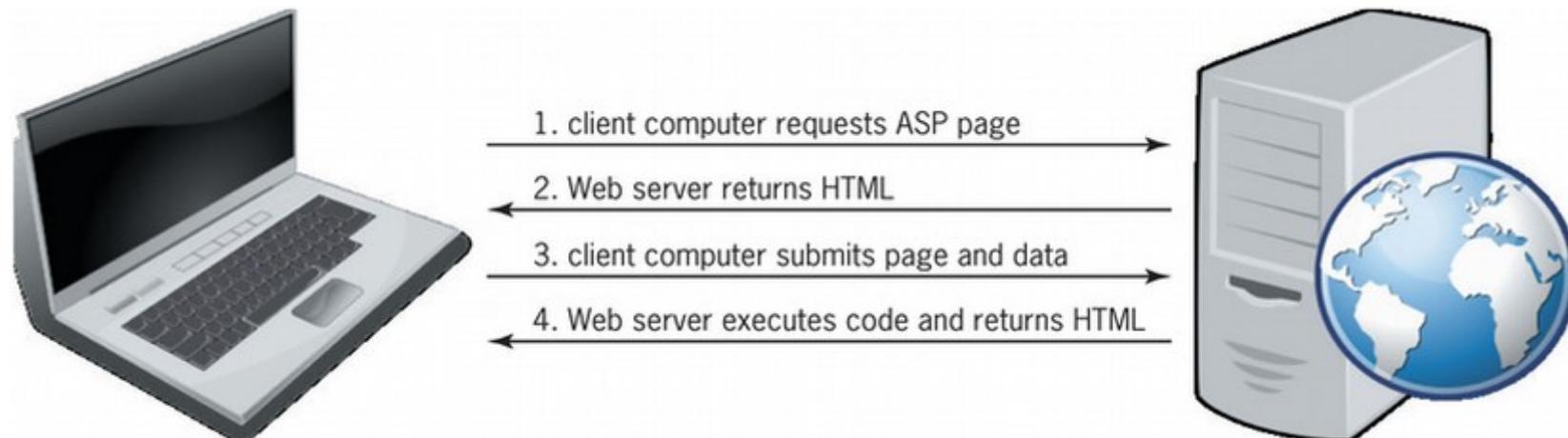


Figure 13-1

A screenshot of a dynamic web page titled "Calculator" under the "Fishbowl Emporium" header. The page has a navigation bar with links for Home, About, and Calculator. The main content is a form titled "Calculator: Gallons of water to fill my aquarium." It contains four input fields labeled "Length (inches)", "Width (inches)", "Height (inches)", and "Water (gallons)". Each input field has a red asterisk next to it indicating it is required. To the right of the form is an illustration of a rectangular aquarium containing an orange fish and some green plants. Below the form is a "Submit" button and a copyright notice "© 2019 - Fishbowl Emporium".

Figure 13-2

- the Web applications created in this chapter use a technology called **ASP.NET**
- **ASP** = Active Server Page - refers to the type of Web page created by the ASP technology
  - all **ASP** pages contain **HTML** (Hypertext Markup Language) **tags** that tell the client's browser how to render the page on the computer screen  
e.g. the instruction `<h1>Hello</h1>` uses the opening `<h1>` tag and its closing `</h1>` tag to display the word "Hello" as a heading on the page
  - many **ASP** pages also contain ASP tags that specify the controls to include on the page
  - in addition to the **HTML** and **ASP tags**, **dynamic ASP** pages contain code that tells the objects on the page how to respond to the user's actions
  - in this chapter, you will write the appropriate code using the **Visual Basic** programming language
- when a client computer's browser sends a request for an ASP page, the Web server locates the page and then sends the approp. HTML instructions to the client
  - the client's browser uses the instructions to render the Web page on the computer screen
  - if the Web page is a dynamic one, like the **Figure 13-2**, the user can interact with the page by entering data
  - in most cases, the user then clicks a button on the Web page to submit the page and its data to the server for processing ->  
<- using Web terminology, the information is "posted back" to the server - this event is referred to as a **postback**
- when the server receives the information, it executes the Visual Basic code associated with the Web page
  - it then sends back the appropriate HTML to the client for rendering in the browser window
  - the returned information includes the result of processing the code and data
  - notice that the Web page's **HTML** is interpreted and executed by the **client computer**, whereas the **program code** is executed by the **Web server**



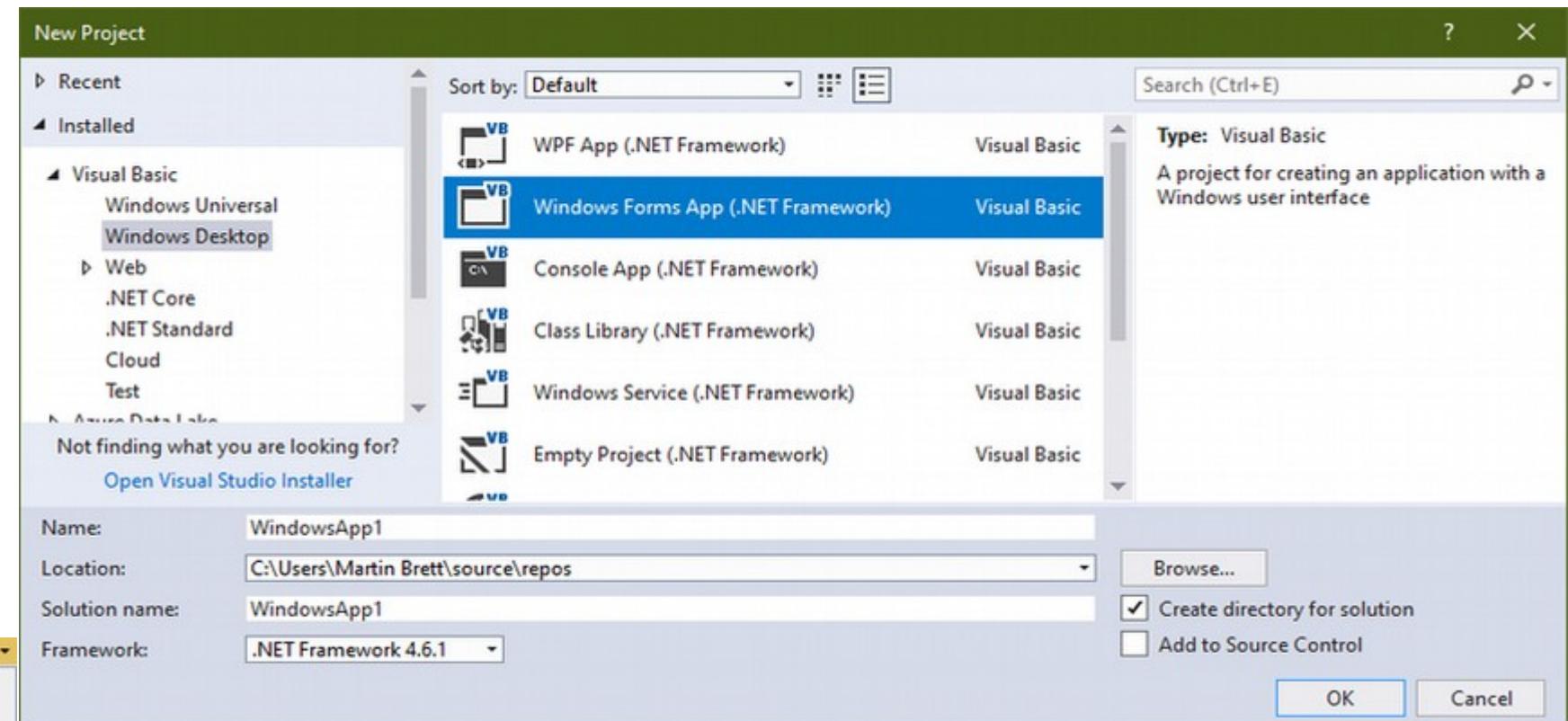
**Figure 13-3** Illustration of the relationship between a client computer and a Web server

#### Mini-Quiz 13-1

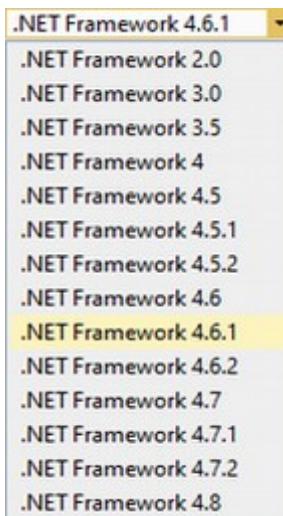
- Q1). What is Microsoft Edge? a). a browser; b). a requester; c). a server; or d). a transmitter.
- Q2). What is the difference between a static Web page and a dynamic Web page?
- Q3). Which of the following interprets and executes a Web page's HTML? a). the client computer, b). the Web server
- A1). a - a browser.
- A2). both types of Web pages display information - however, a dynamic Web page can interact with the viewer.
- A3). a - the client computer.

- in previous chapters, you created **Windows Forms applications** using the template: **Windows Forms App (.NET Framework)**  
-> menu: **File / New Project...** **Ctrl+N** / Installed / Visual Basic / Windows Desktop / **Windows Forms App (.NET Framework)**

<- max .NET Framework 4.8  
**Figure 13-4**



**Figure 13-4** Windows Forms App (.NET Framework) supporting .NET Framework up to version 4.8



- Visual Basic also provides templates for creating **Web applications** using the template: **ASP.NET Web Forms Site**

<- max .NET Framework 4.8

<- in older version of **Visual Studio Community 2017** used by the book:

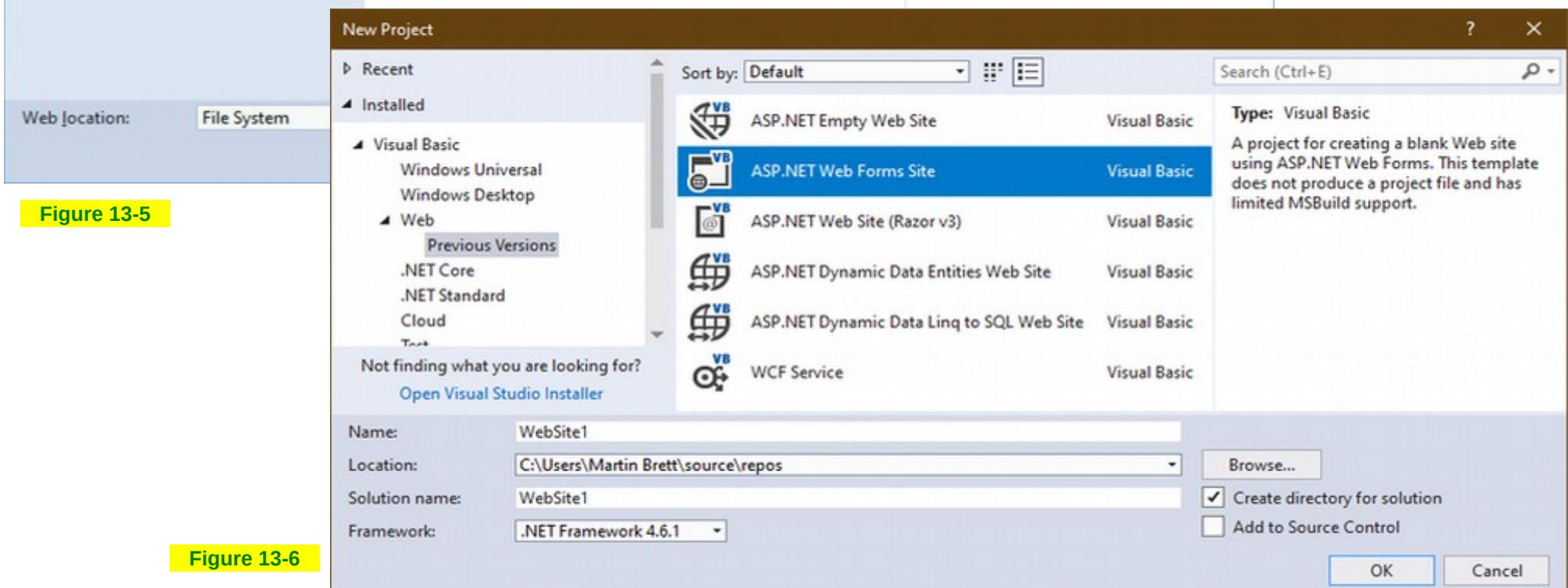
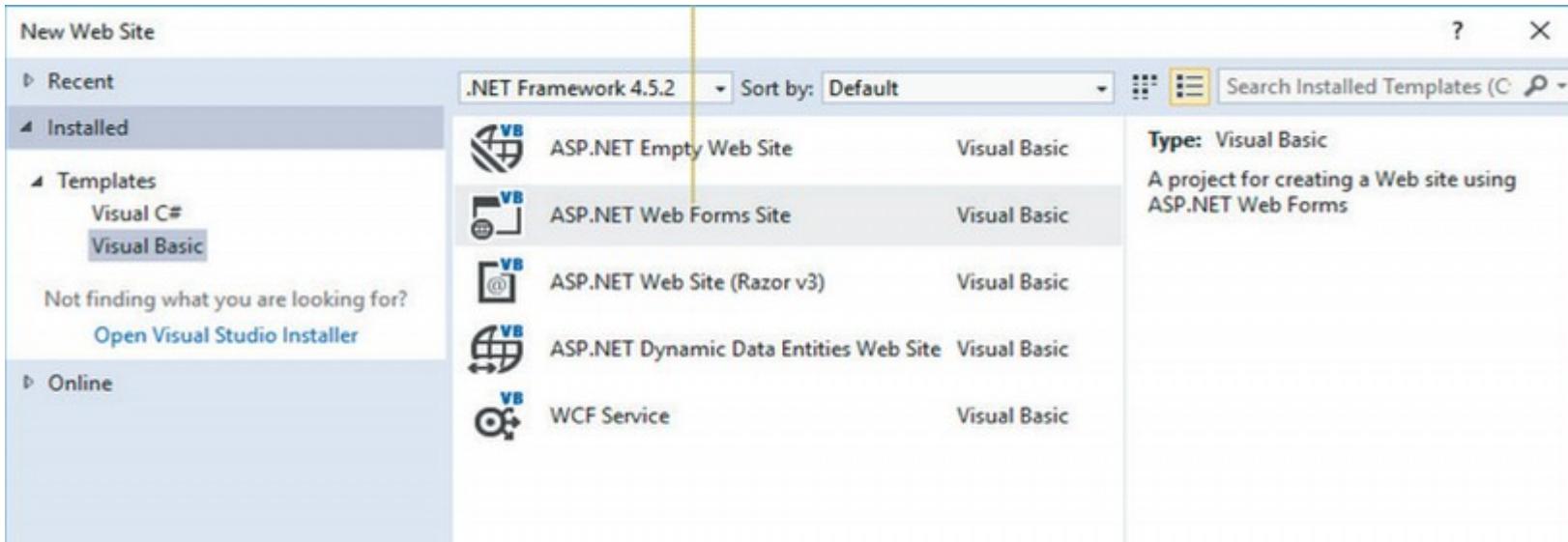
-> menu: **File / New Web Site / Installed / Visual Basic / ASP.NET Web Forms Site**

**Figure 13-5**

<- in my **Visual Studio Community 2017** updated to version: **15.9.36**:

-> menu: **File / New Project... Ctrl+N / Installed / Visual Basic / Web / Previous Versions / ASP.NET Web Forms Site**

**Figure 13-6**



- the template **ASP.NET Web Forms Site** contains:
  - files that are designed to make the creation of a **Web application** a fairly simple matter
  - among others are the **4 files**, which you will use in this lesson:

**Figure 13-7** 4 of the files included in the **ASP.NET Web Forms Site** template:

| filename            | purpose                                                                                                                  |
|---------------------|--------------------------------------------------------------------------------------------------------------------------|
| <b>About.aspx</b>   | contains information about the website, such as its history or purpose                                                   |
| <b>Contact.aspx</b> | provides contact information, such as a phone number or e-mail address                                                   |
| <b>Default.aspx</b> | serves as the home page                                                                                                  |
| <b>Site.master</b>  | creates a consistent layout for the pages in your application (for example, the same color, header, footer, and buttons) |

- you will create a **Web site** application for the **Fishbowl Emporium store**

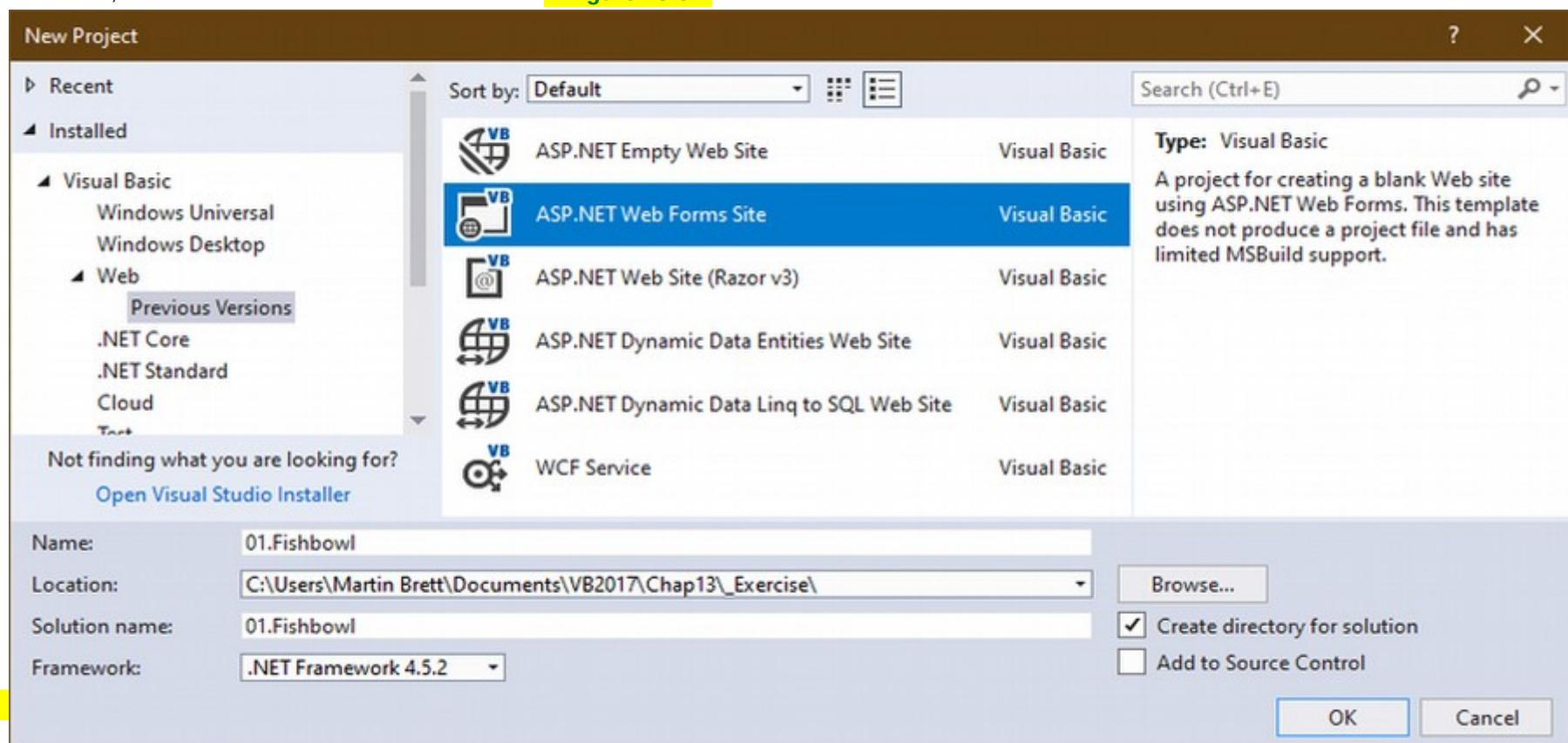
- the application will contain **3 Web pages**:
  - 2 static: - **Default.aspx** (Home page)
  - **About.aspx**
  - 1 dynamic: - **Calculator.aspx & Calculator.aspx.vb**

**CH13\_F5**  
**CH13\_F6**  
**CH13\_A1** -> **CH13\_A4**

- to create the **Web site application** for the **Fishbowl Emporium store**: e.g. with **01.Fishbowl 1/9**

- > start **Visual Studio 2017**, and display the **Solution Explorer** window
- step 1** - open the template for **ASP.NET Web Forms Site** in Visual Studio 2017 v 15.9.36 <- in each version may vary
- > menu: **File / New Project...** **Ctrl+N** / Installed / Visual Basic / Web / Previous Versions / **ASP.NET Web Forms Site**
- > fill the Name, Location and Framework as shown in: **Figure 13-8**

**Figure 13-8**



**Figure 13-8**

- the computer uses the selected template to create a Web site application named Fishbowl
- the template includes the files listed in the **Solution Explorer** window **Figure 13-9**
- the window **Default.aspx** shows the contents of the file in **Source** view
- this view reveals the **HTML** and **ASP** tags that tell a browser how to render the Web page
- the tags were automatically generated when you created the application

The screenshot shows the Visual Studio interface with two main windows:

- Solution Explorer** (right): Shows the project structure for "01.Fishbowl" with files like Account, App\_Code, App\_Data, Bin, Content, fonts, Scripts, About.aspx, Bundle.config, Contact.aspx, Default.aspx (selected), favicon.ico, Global.asax, packages.config, Site.master, Site.Mobile.master, ViewSwitcher.ascx, and Web.config.
- Default.aspx** (left): Shows the source code for the Default.aspx page. The code includes ASP.NET controls like <asp:Content> and <asp:Panel>, along with HTML and CSS classes. The code is as follows:

```

1  <%@ Page Title="Home Page" Language="VB" MasterPageFile="~/Site.Master" %>
2
3  <asp:Content ID="BodyContent" ContentPlaceHolderID="MainContent" %>
4
5      <div class="jumbotron">
6          <h1>ASP.NET</h1>
7          <p class="lead">ASP.NET is a free web framework for building dynamic websites. A design surface and hundreds of controls and components let you build dynamic websites quickly, easily, and affordably.</p>
8          <p><a href="http://www.asp.net" class="btn btn-primary btn-lg">Get started</a></p>
9      </div>
10
11     <div class="row">
12         <div class="col-md-4">
13             <h2>Getting started</h2>
14             <p>ASP.NET Web Forms lets you build dynamic websites. A design surface and hundreds of controls and components let you build dynamic websites quickly, easily, and affordably.</p>
15             <p><a class="btn btn-default" href="https://go.microsoft.com/fwlink/?LinkId=301518">Get started</a></p>
16         </div>
17         <div class="col-md-4">
18             <h2>Get more libraries</h2>
19             <p>NuGet is a free Visual Studio extension that makes it easy to add, remove, and update dependencies. NuGet packages contain code, documentation, and assets for your projects.</p>
20         </div>
21     </div>
22 
```

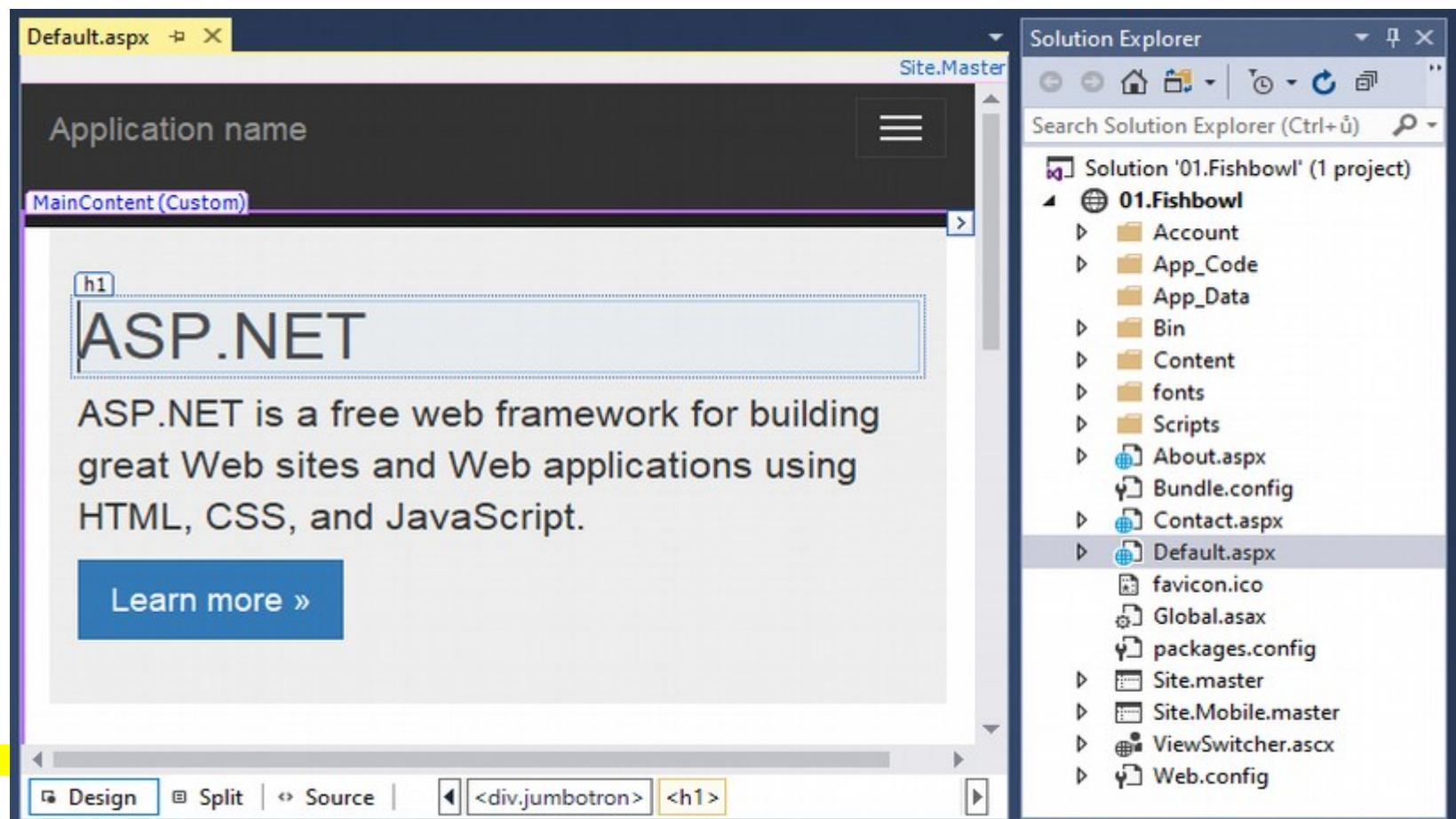
At the bottom of the Source view, there are tabs for Design, Split, and Source, with Source currently selected.

Figure 13-9

**step 2** - open the tab: **Design**

- > click the **Design** tab that appears at the bottom of the IDE
- when the **Design** tab is selected, the Web page appears in **Design** view
- you can use **Design** view to add text and controls to the **Web page**

**Figure 13-10**



**Figure 13-10**

**step 3** - to see both tabs - **Source** code and **Design** - in one window, split the **Default.aspx** window

-> click the **Split** tab at the bottom of the IDE to split the **Default.aspx** window into 2 parts

- the upper half displays the Web page in **Source** view

- the lower half displays the Web page in **Design** view

Figure 13-11

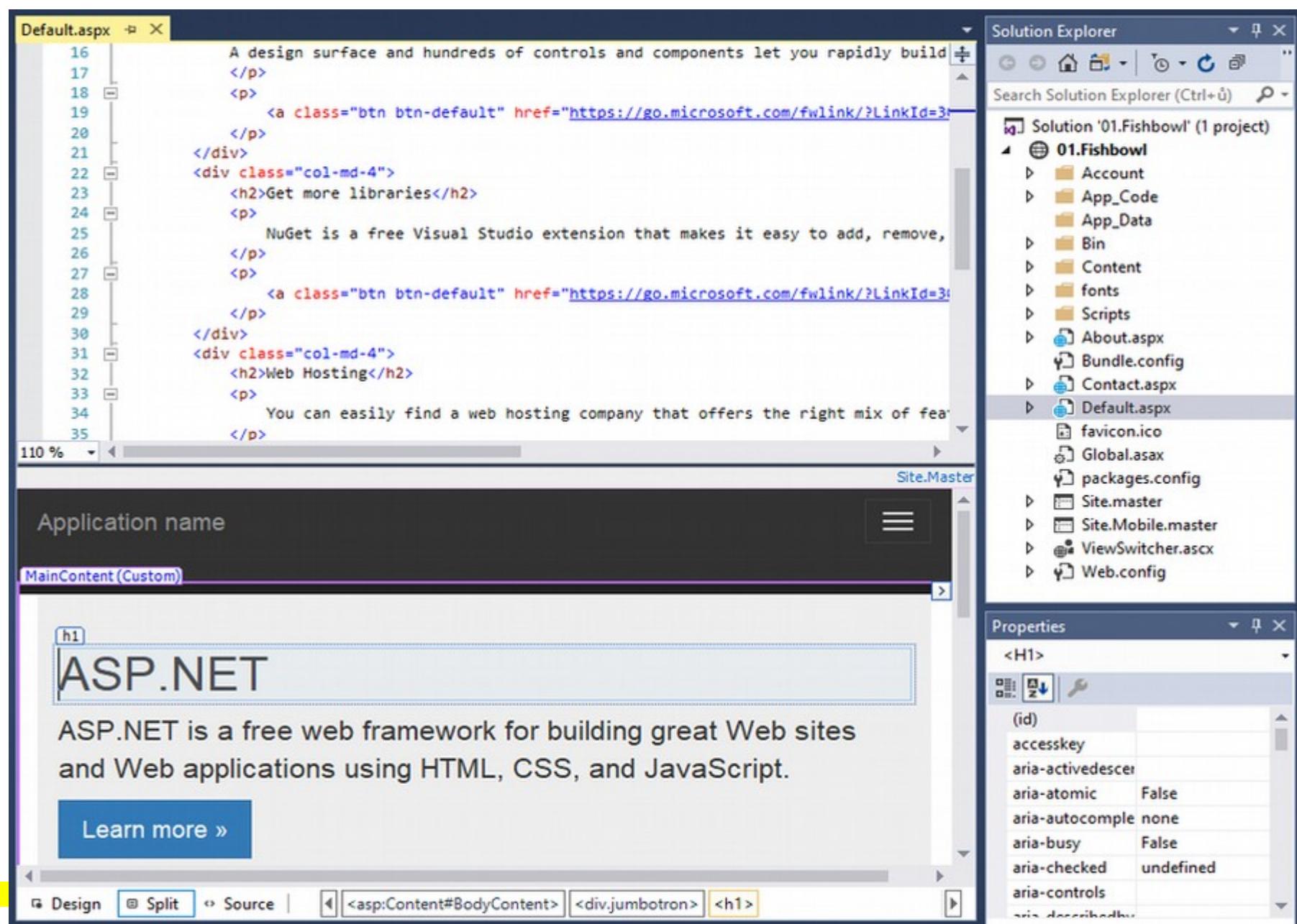
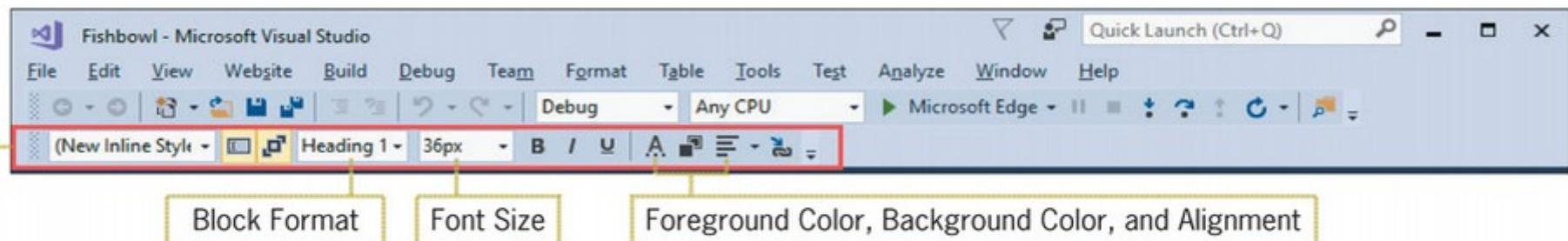


Figure 13-11

- step 4** - open/view the **Formatting toolbar** for the **Design tab**  
 -> click the **Design** tab at the bottom of the IDE to return to **Design view**  
 -> menu **View / Toolbars** / check the option: **Formatting**

**Figure 13-12**



**Figure 13-12**

**CH13\_F3 - how to add Start Without Debugging option to the Debug menu in VS 2017 step by step instructions &**  
**& starting/testing a Web Application in VS 2017 v 15.9.36 without debugging option (Ctrl + F5) e.g. with 01.Fishbowl 2/9**

- you can start/test/(debug) a **Web application** either:
  - a). with the debugging option      -> press the key: **F5**  
 -> menu: **Debug / Start Debugging F5**
  - b). without the debugging option      -> **press the key combo: Ctrl + F5**  
 -> menu: **Debug / Start Without Debugging Ctrl + F5**  
 <- not automatically included on the menu, you must add it
- in this chapter, you will start your Web applications without the option
  - the advantage of doing this is that you will not need to remember to stop the application each time you close the browser window after starting the application
  - my info: not an issue any more in **VS 2017 v 15.9.36**
- when you start/test/debug a **Web application**, the computer creates a temporary **Web server** that allows you to view your **Web page** in default browser
- keep in mind, that your **Web page** will need to be placed on an actual **Web server** for others to view it

- if you prefer to use a menu option, you might need to add the **Start Without Debugging** option to the **Debug** menu because the option is not automatically included on the menu

- to add the **Start Without Debugging** option to the **Debug** menu in VS 2017 & start/test/debug the Web site application **01.Fishbowl 2/9**

**step 1** - first, you will determine whether your **Debug** menu already contains the option

-> on the menu bar click: **Debug** **Figure 13-13a**

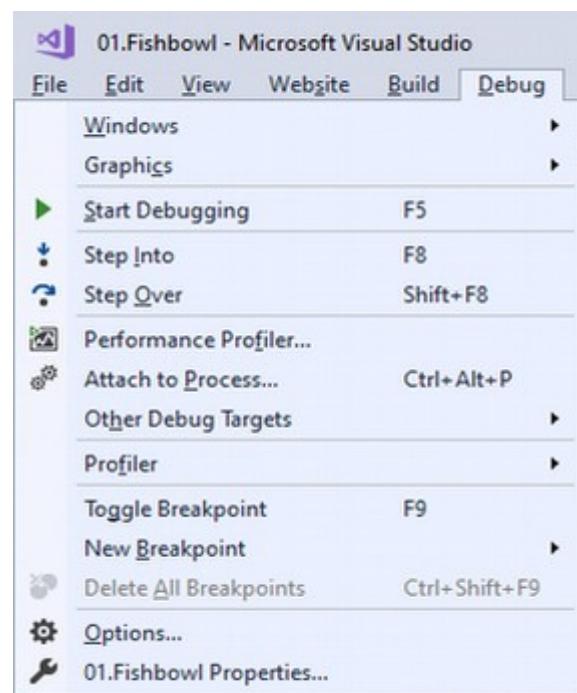
<- if the menu contains the **Start Without Debugging** option, skip the remaining steps

**step 2** - open the dialog box **Customize**

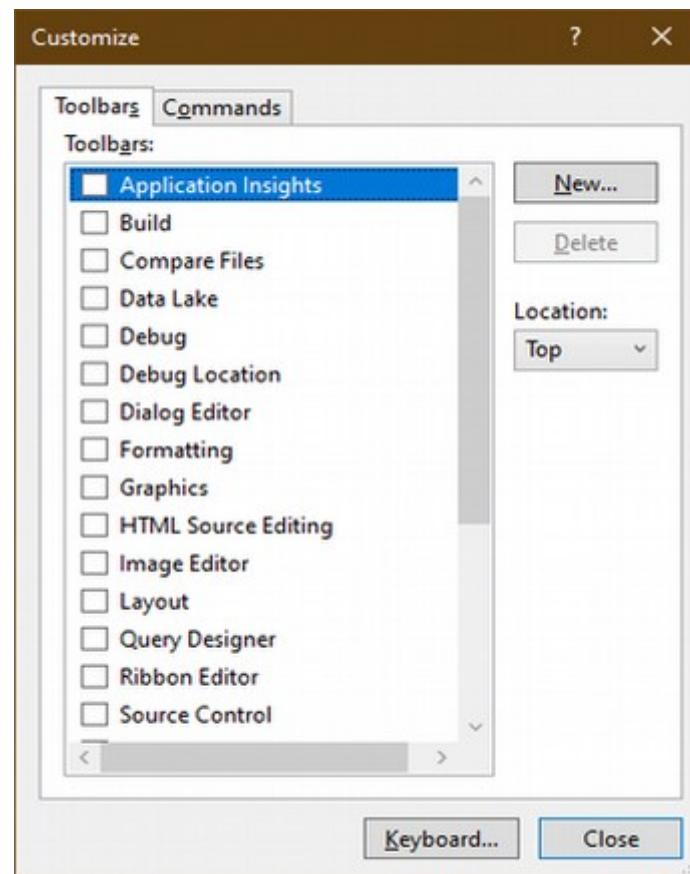
-> on the menu bar click: **Tools / Customize...** **Figure 13-13b**

**step 3** -> on the dialog box **Customize** click the tab **Commands** **Figure 13-13c**

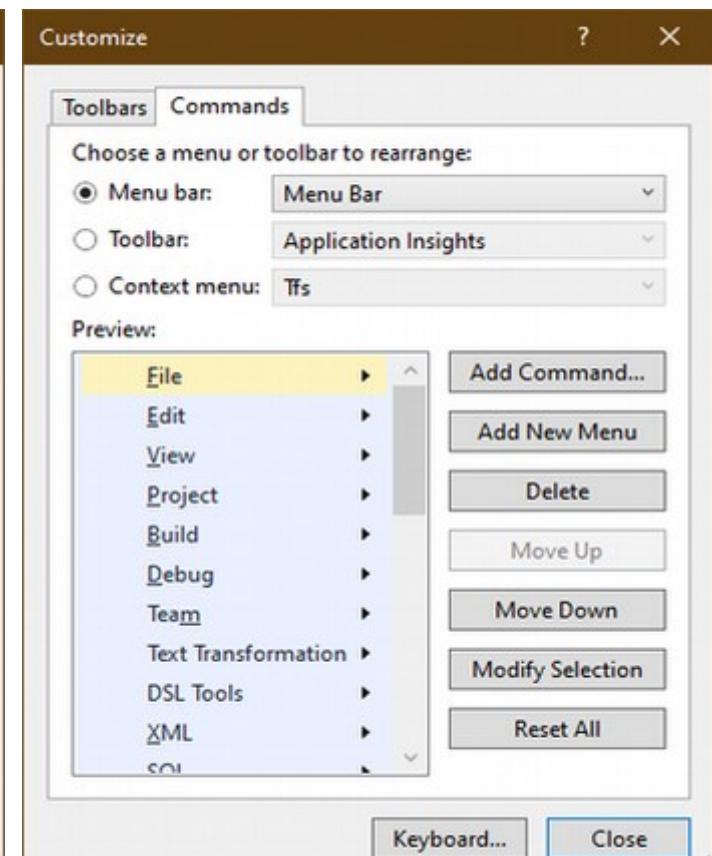
- the radio button **Menu bar:** should be selected



**Figure 13-13a**



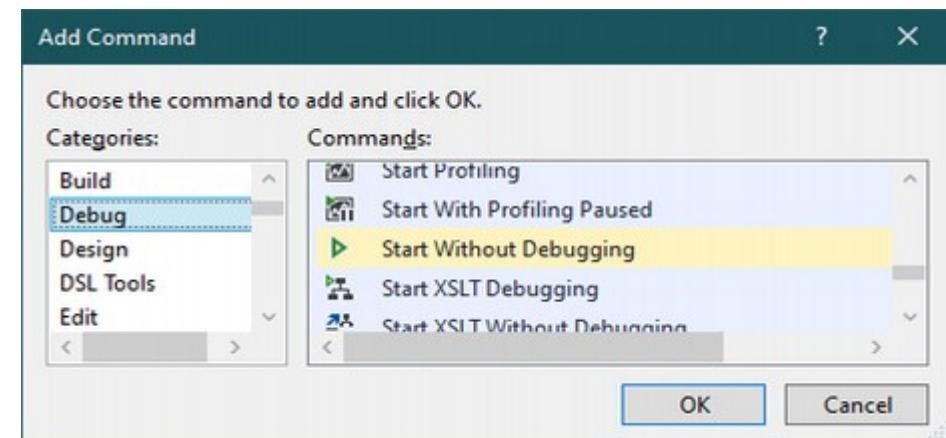
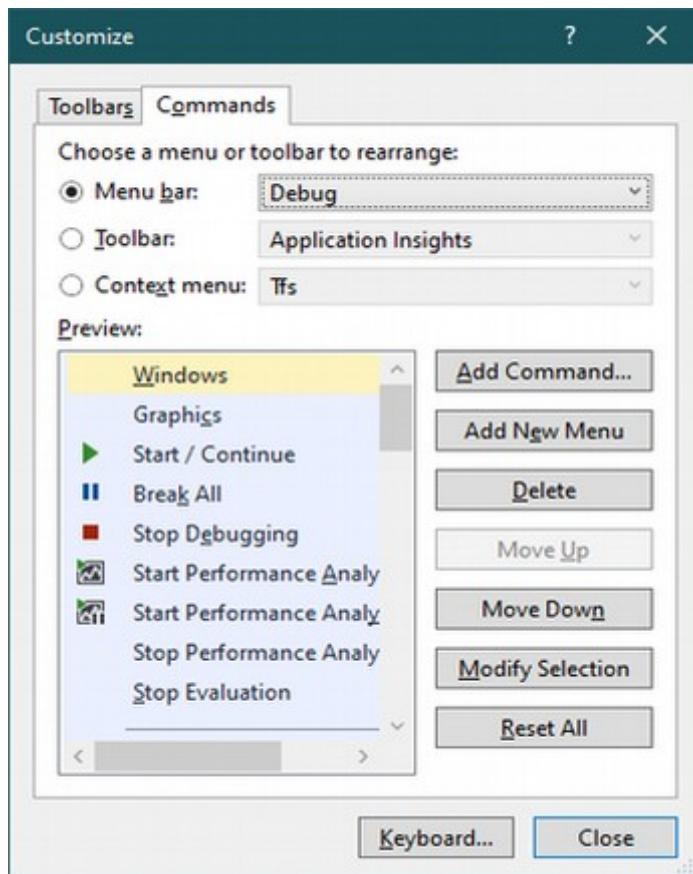
**Figure 13-13b**



**Figure 13-13c**

step 4 -> in the **Menu bar**: list box click the down arrow and scroll down the list until you see **Debug** and then click on it

Figure 13-13d



step 5 - open the dialog box **Add Command**

Figure 13-13e

-> click the button **Add Command...** to open the dialog box named **Add Command**

-> in the **Categories** list click **Debug**

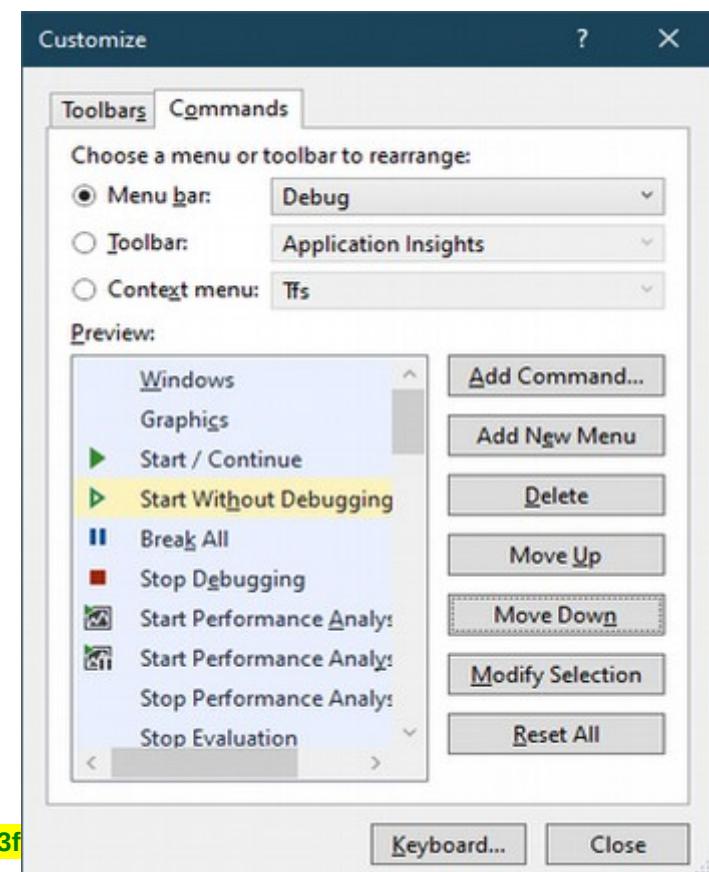
-> scroll down the **Commands** list until you see: **Start Without Debugging**, select it and then click the button **OK** to close the dialog box **Add Command**

step 6 -> click the button **Move Down** until the **Start Without Debugging** option appears below the **Start / Continue** option

Figure 13-13f

-> click the button **Close** to close the dialog box **Customize**

Figure 13-13f



**step 7** - now you can start/test/debug your application and see what the template automatically created

-> either press **Ctrl + F5**, or click the recently created option in menu: **Debug / Start Without Debugging Ctrl + F5**

**Figure 13-14**

1). your browser requests the **Default.aspx** page from the **Web server**

2). the server locates the page and then sends the appropriate **HTML** instructions to your default browser for rendering on the screen

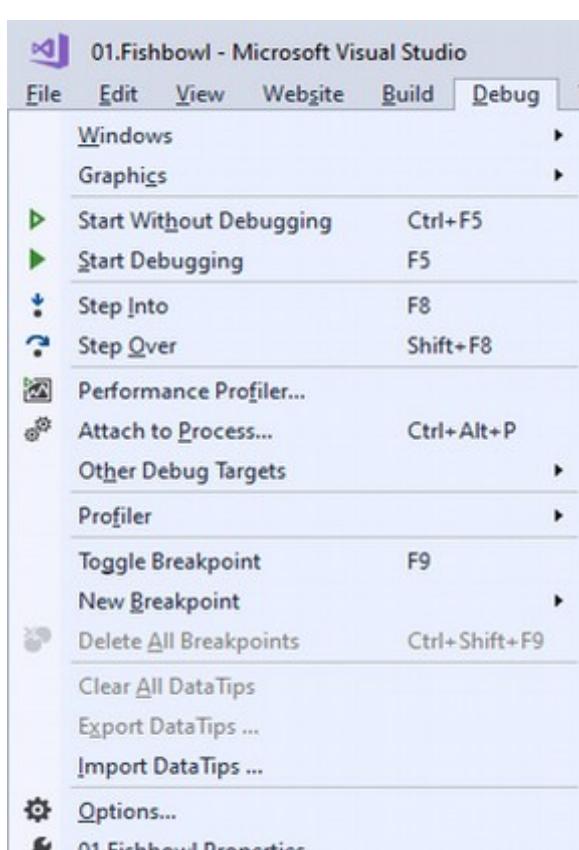
**Figure 13-15** - shows the **Default.aspx** page created by the template you are using: **ASP.NET Web Forms Site**

- the page is displayed in Chrome and might appear slightly different in a different browser <- MyNote: looks the same in MS Edge

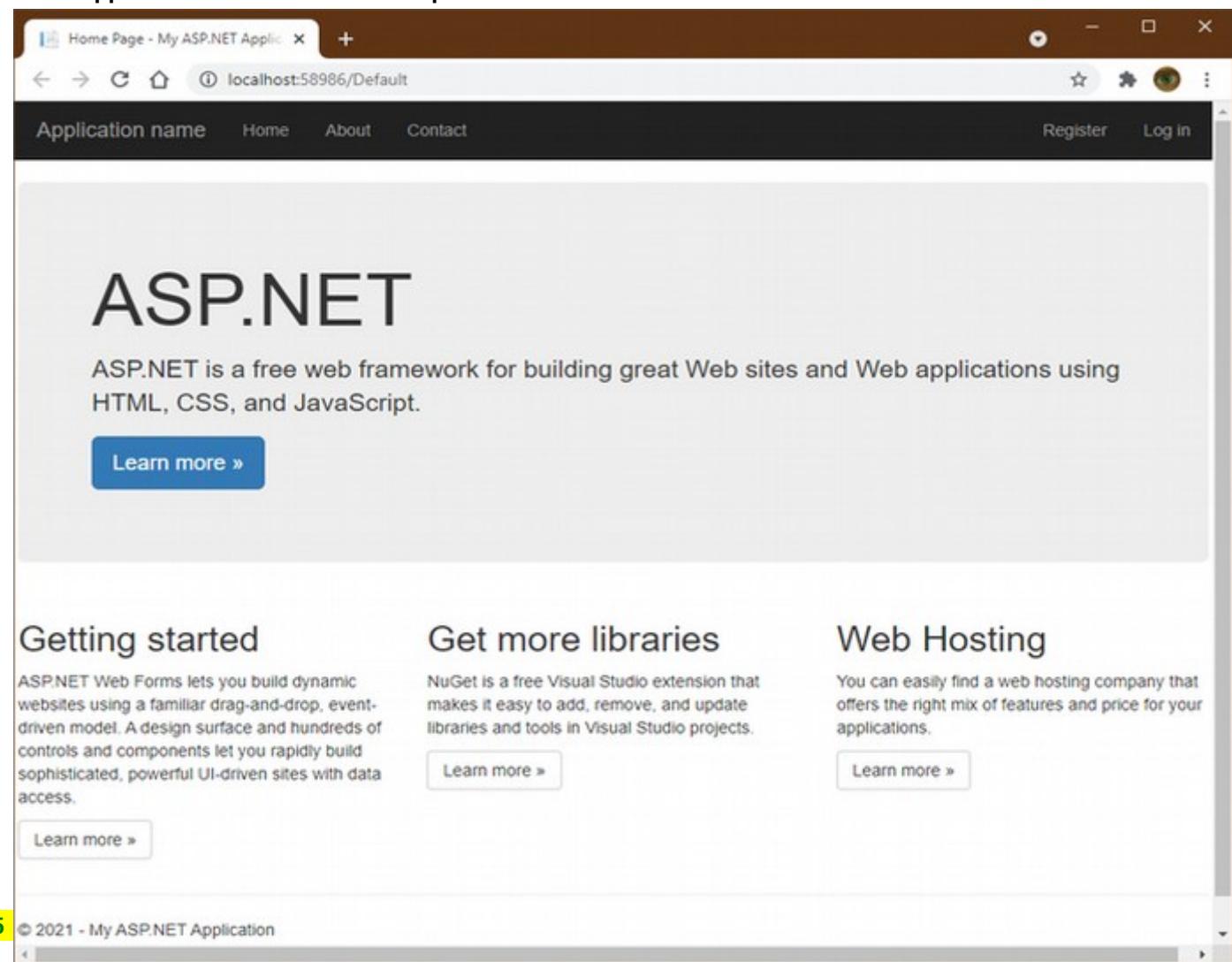
-> click **About** to display the About page that comes with the template, click **Contact**, then click **Register**, and then click **Log in** to display those pages

-> when done exploring, you can close the **browser** window

- in the next section, you will personalize the **Web application** for the **Fishbowl Emporium store**



**Figure 13-14**

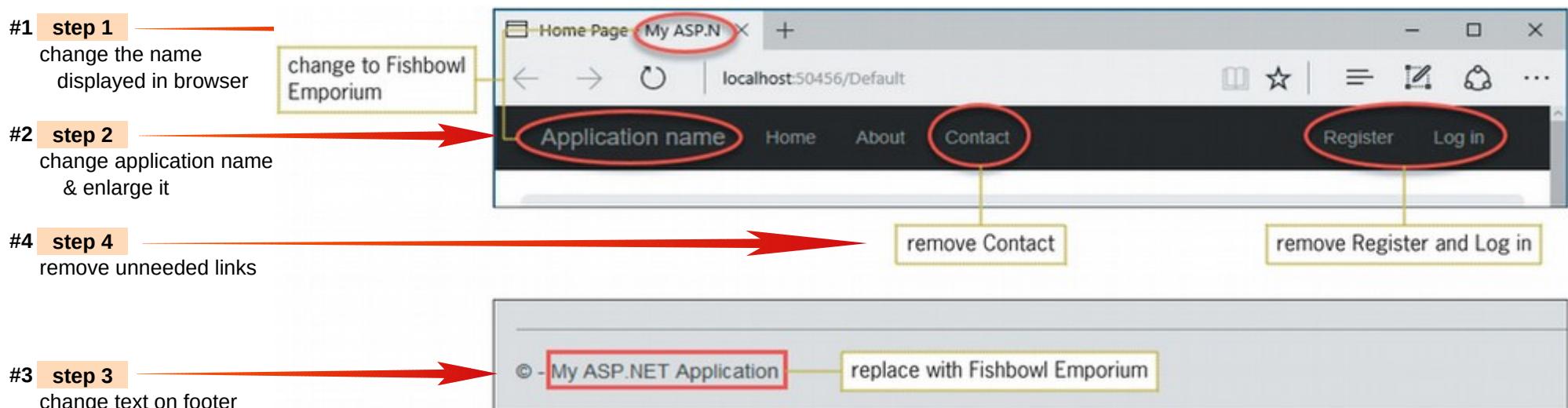


**Figure 13-15**

**Figure 13-7** 4 of the files included in the ASP.NET Web Forms Site template:

| filename     | purpose                                                                                                                  |
|--------------|--------------------------------------------------------------------------------------------------------------------------|
| About.aspx   | contains information about the website, such as its history or purpose                                                   |
| Contact.aspx | provides contact information, such as a phone number or e-mail address                                                   |
| Default.aspx | serves as the home page                                                                                                  |
| Site.master  | creates a consistent layout for the pages in your application (for example, the same color, header, footer, and buttons) |

- before adding any text or controls to the .aspx pages, you will make the changes to the **Site.master** page as indicated in: **Figure 13-16**
- any changes made to that page will affect all of the .aspx pages in the **Web application**
- in the **Site.master** page's footer you will also replace the original text: **My ASP.NET Application** with: **Fishbowl Emporium**



**Figure 13-16** Site.master page changes for e.g. 01.Fishbowl

- to make changes to the **Site.master** page: e.g. with **01.Fishbowl 3/9**

**step 0** - open the **Source** code for the **Site.master** file  
-> close the window: **Default.aspx**  
-> in the **Solution Explorer** window, double-click the **Site.master** to open the page and if necessary, click the tab: **Source** to view the code

**step 1** - #1 you will change the application name that appears on each **Web** page's tab in the browser window: **My ASP.NET Application** with **Fishbowl Emporium**  
-> in the **Site.master** file's **Source** code locate the tag: <**title**> on line 9

9       <**title**><%: Page.Title %> - My ASP.NET Application</**title**>

-> replace the original text: **My ASP.NET Application** with: **Fish Emporium**

9       <**title**><%: Page.Title %> - Fish Emporium</**title**>

**step 2.1** - #2 you will change the application name that appears on each Web page: **Application name** with **Fishbowl Emporium** and then enlarge it  
-> in the **Site.master** file's **Source** code search for the text: **application name**

48                   [Application name](#)

-> replace the text: **Application name** with: **Fishbowl Emporium**

48                   [Fishbowl Emporium](#)

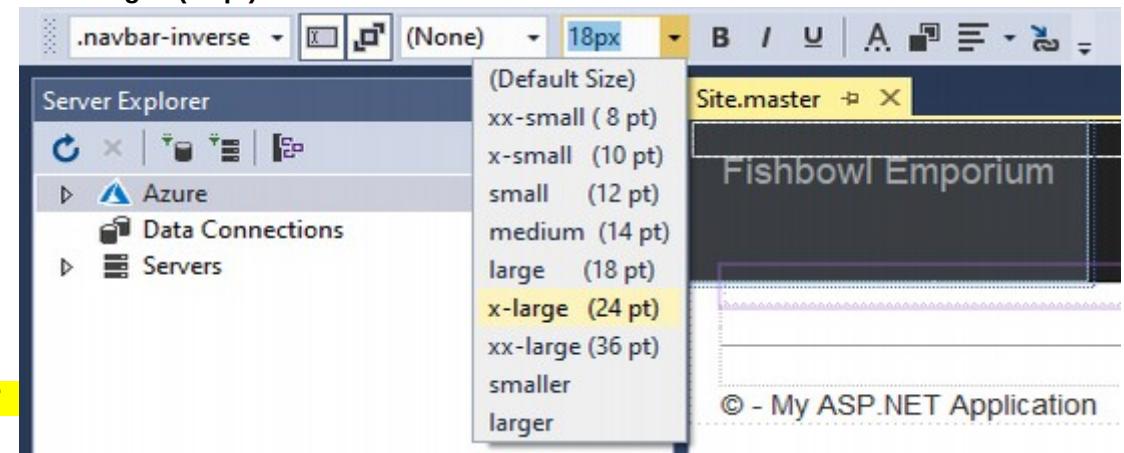
**step 2.2** - enlarge the **Fishbowl Emporium** text

**Figure 13-17**

-> at the bottom of the IDE click the **Design** tab to edit a design of your text

-> in the **Fishbowl Emporium** text that appears below the **Site.master** tab, click immediately before the letter **E**

-> on the **Formatting toolbar** click the **Font Size** arrow and then click: **x-large (24 pt)**



**Figure 13-17**

**step 3** - #3 you will change the text that appears in each page's footer: © 2021 - My ASP.NET Application with: © 2021 - Fishbowl Emporium

-> at the bottom of the IDE click the **Source** tab to edit a source code of the **Site.master** file

-> search for the text: **footer**

79                   <footer>  
80                   

&copy; <%: DateTime.Now.Year %> - My ASP.NET Application

  
81                   </footer>

- the **<p>** tag specifies the text that will appear at the bottom of each Web page

- the **DateTime.Now.Year** entry will display the current year

-> replace the text: **My ASP.NET Application** with: **Fishbowl Emporium**

79                   <footer>  
80                   

&copy; <%: DateTime.Now.Year %> - Fishbowl Emporium

  
81                   </footer>

**step 4.1** - #4 you will change into comments unneeded links from a navigation bar: **Contact**, **Register**, and **Log in**

-> search for the text: **Contact**

```
51          <ul class="nav navbar-nav">
52              <li><a runat="server" href="/">Home</a></li>
53              <li><a runat="server" href("~/About")>About</a></li>
54              <li><a runat="server" href "~/Contact">Contact</a></li>
55          </ul>
```

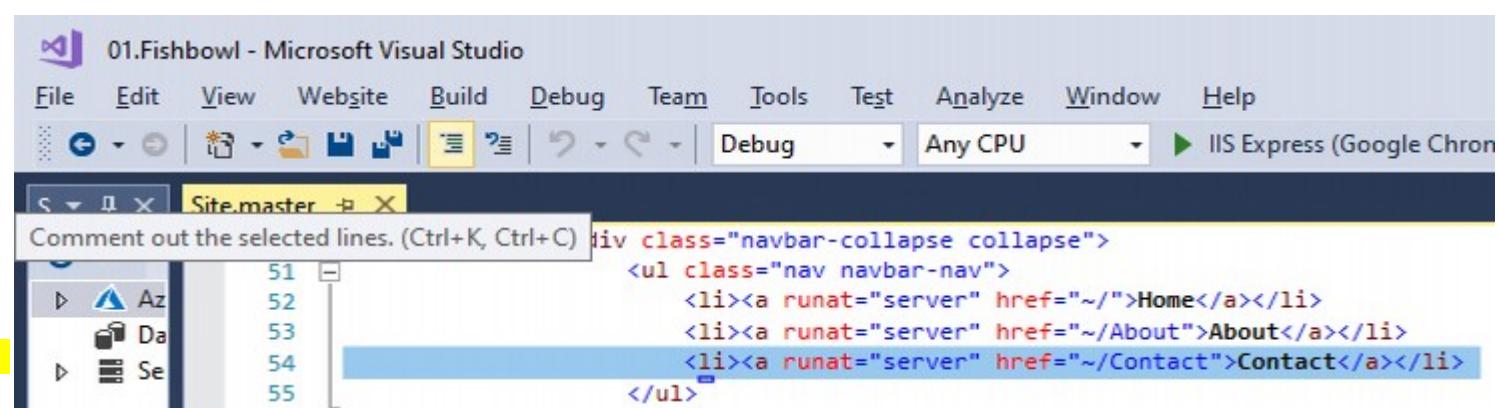
- the **<ul>** tag creates a list

- the **<li>** tag creates a list item

- this application does not need the **Contact** link, so you will change the line containing its **<li>** tag into a comment using tags: **<%** and **%>**

-> select/highlight the line you want to comment and on the toolbar **Standard** click the button named: **Comment out the selected lines. (Ctrl + K, Ctrl + C)**

Figure 13-18



- the **<% --** and **-- %>** tags indicate the beginning and the end of a comment

```
51          <ul class="nav navbar-nav">
52              <li><a runat="server" href="/">Home</a></li>
53              <li><a runat="server" href "~/About">About</a></li>
54      <%--          <li><a runat="server" href "~/Contact">Contact</a></li>--%>
55          </ul>
```

step 4.2 -> search for the text: **Register**

```
58             <ul class="nav navbar-nav navbar-right">
59                 <li><a runat="server" href("~/Account/Register">Register</a></li>
60                 <li><a runat="server" href "~/Account/Login">Log in</a></li>
61             </ul>
```

- here too, the application will not need these links, so you will change them to comments using tags: **<%** and **%>**

-> select/highlight all lines you want to comment and on the toolbar **Standard** click the button named: **Comment out the selected lines. (Ctrl + K, Ctrl + C)**

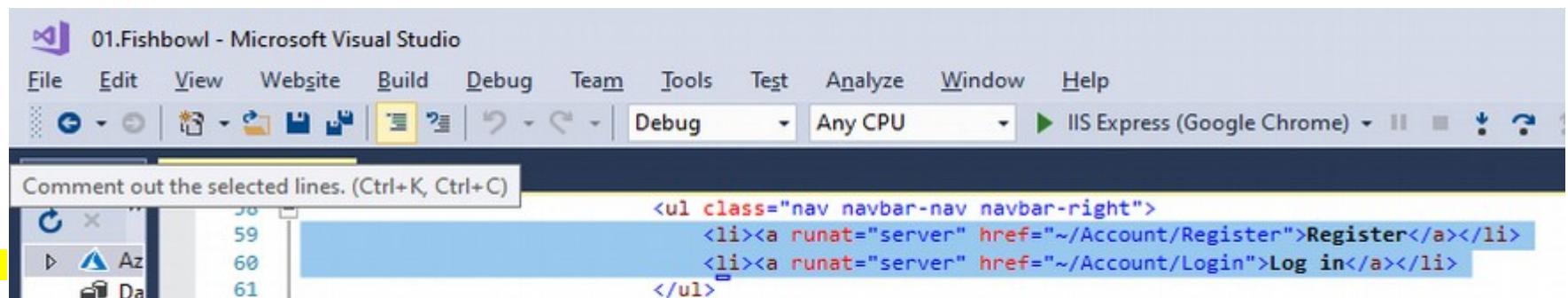


Figure 13-19

- the **<% --** and **-- %>** tags indicate the beginning and the end of a comment

```
58             <ul class="nav navbar-nav navbar-right">
59             <%-->
60                 <li><a runat="server" href "~/Account/Register">Register</a></li>
61                 <li><a runat="server" href "~/Account/Login">Log in</a></li>--%>
62             </ul>
```

**step 5** - view the changes made

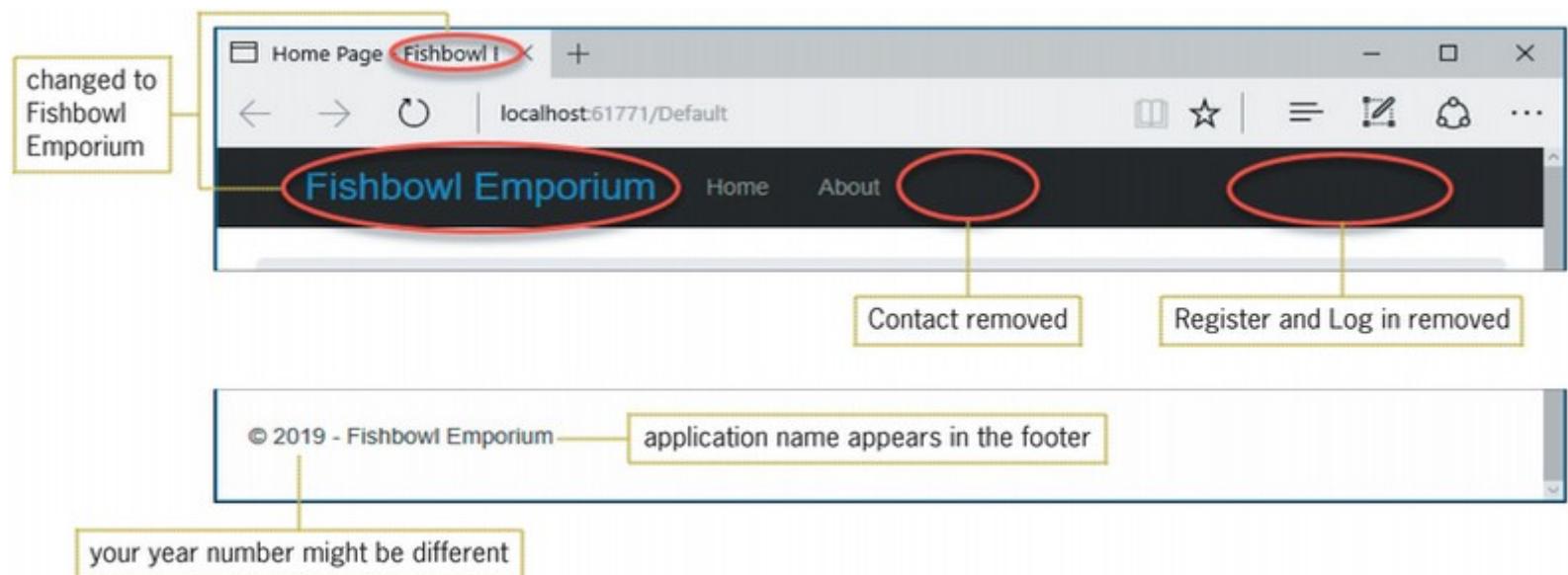
**Figure 13-20**

-> save the solution and then start/test the application without debugging (**Ctrl + F5**)

- **Fishbowl Emporium** appears as the application's name on the Web page and on the application's tab in the browser window (Home Page)

- the Web page no longer contains the links: **Contact**, **Register**, **Log in**

- the Web page's footer now includes the application's name



**Figure 13-20** Home page - **Default.aspx** page showing the changes

-> close the browser window, you can close the **Site.master** page

## CH13\_F5 - create/personalize the 1st page - Static Web page Default.aspx (Home page), e.g. with 01.Fishbowl 4/9

- the **Default.aspx** - Home page - created by the template **ASP.NET Web Forms Site** will be the 1st static Web page included in the application for the Fishbowl Emporium store, as shown earlier in: **Figure 13-1**

- to personalize the **Default.aspx** - Home page - the 1st **Static** Web page: e.g. with **01.Fishbowl 4/9**

**step 0** - open the: **Solution Explorer** window / **Default.aspx** page and click the **Design** tab

<- notice that the page inherits the application's name and footer from the **Site.master** page edited in previous lesson:

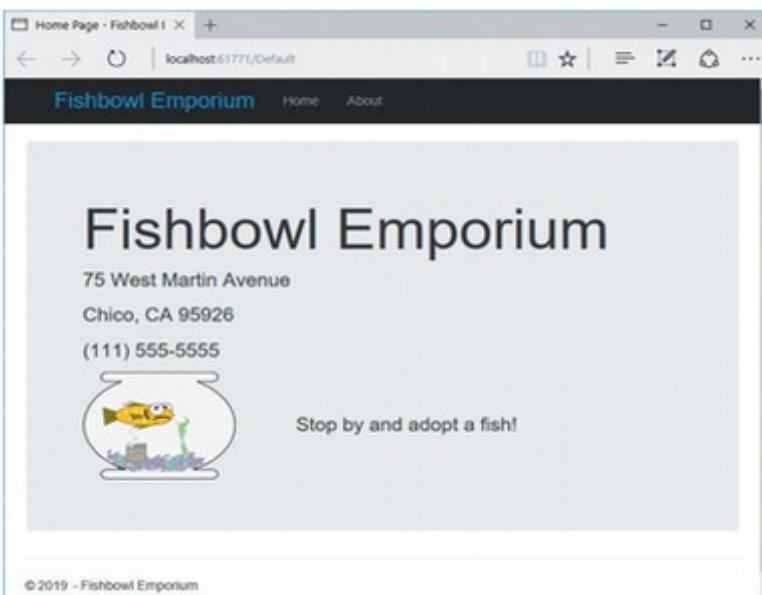
**CH13\_F4** - modify the **0th** page - master page **Site.master** (the page creates a consistent layout for the pages in your app = affects all of the **.aspx** pages)

**step 1** - add a company name and the address

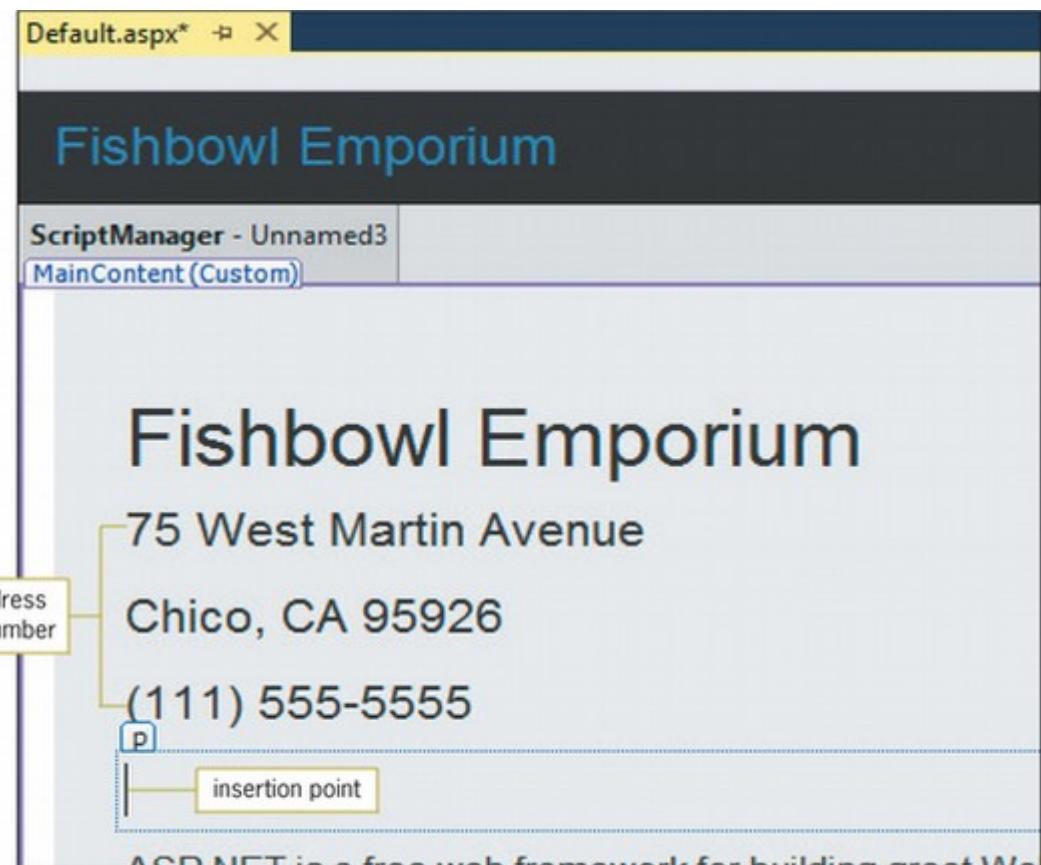
**Figure 13-21**

-> select the **ASP.NET** text that appears in the **MainContent(Custom)** area of the page, type: **Fishbowl Emporium**, and press Enter key

-> now, enter the address and phone number shown in: **Figure 13-21** and then press enter key = position the insertion point as shown in the figure



**Figure 13-1**



**Figure 13-21**

**step 2.0** - add an image file to your application and then to your Web page

**step 2.1** - before you can add the fishbowl image to the Web page, you need to add the image file to the application, so:  
-> on the menu bar click: **Website / Add Existing Item... Ctrl+D** Figure 13-22

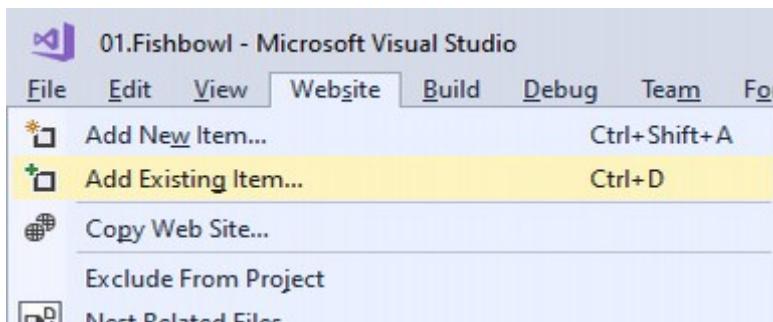


Figure 13-22

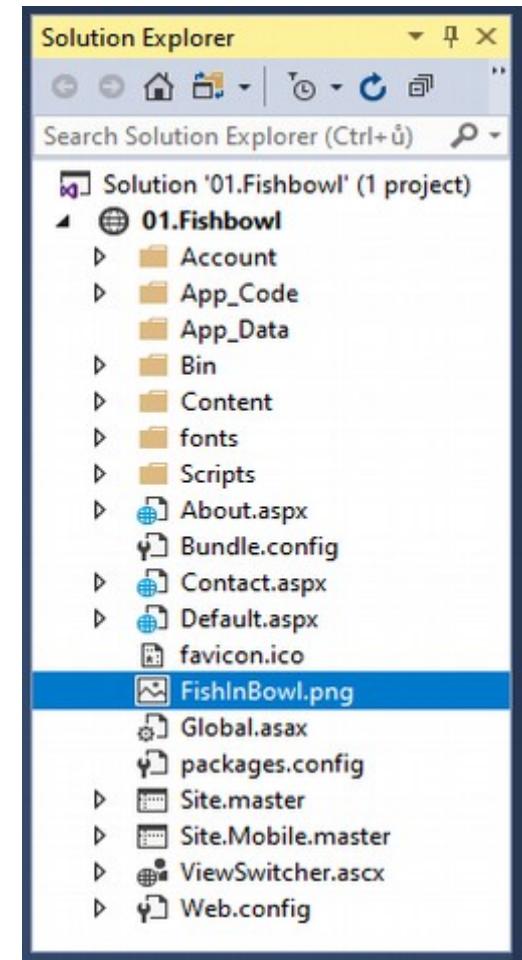


Figure 13-23

**step 2.2** - now the image is part of your application, you can use it on your Web page:

-> verify that your insertion point is located in the box as shown in: Figure 13-21

-> in the **Toolbox** window, double click the **Image** to add  
an image control at the location of the insertion point on the page  
-> display the image control's **Properties** window and change: Figure 13-24

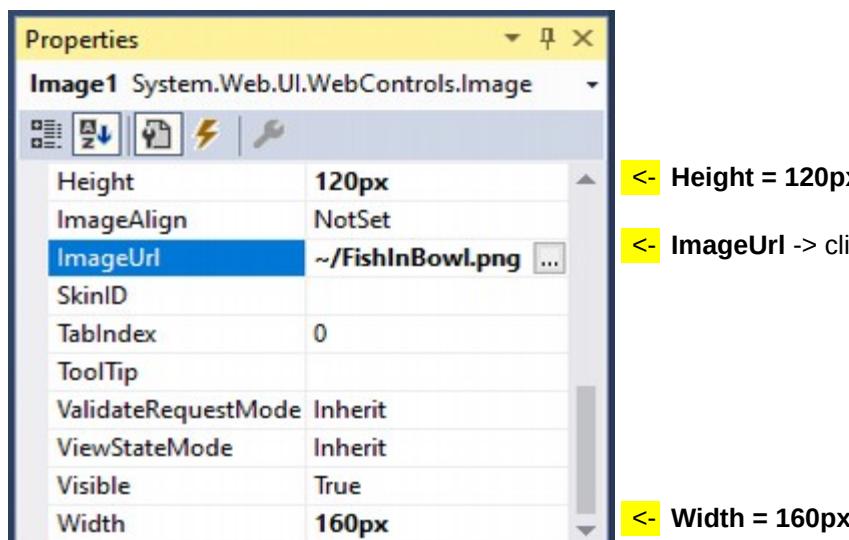
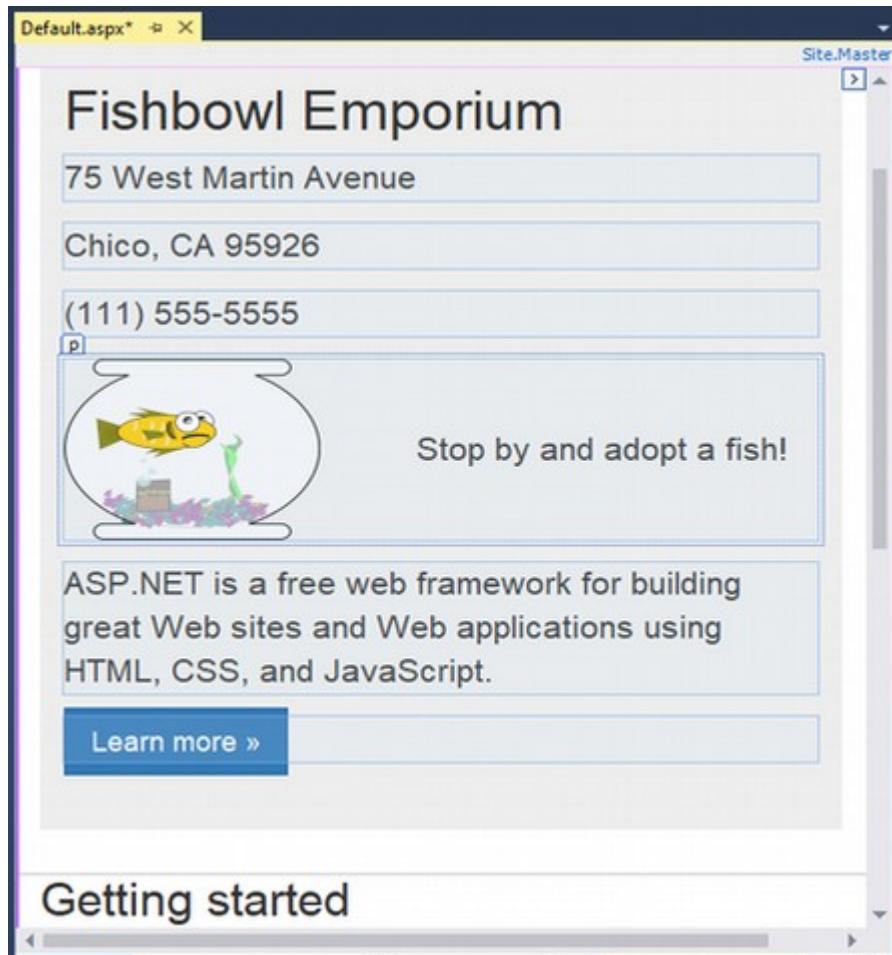


Figure 13-24

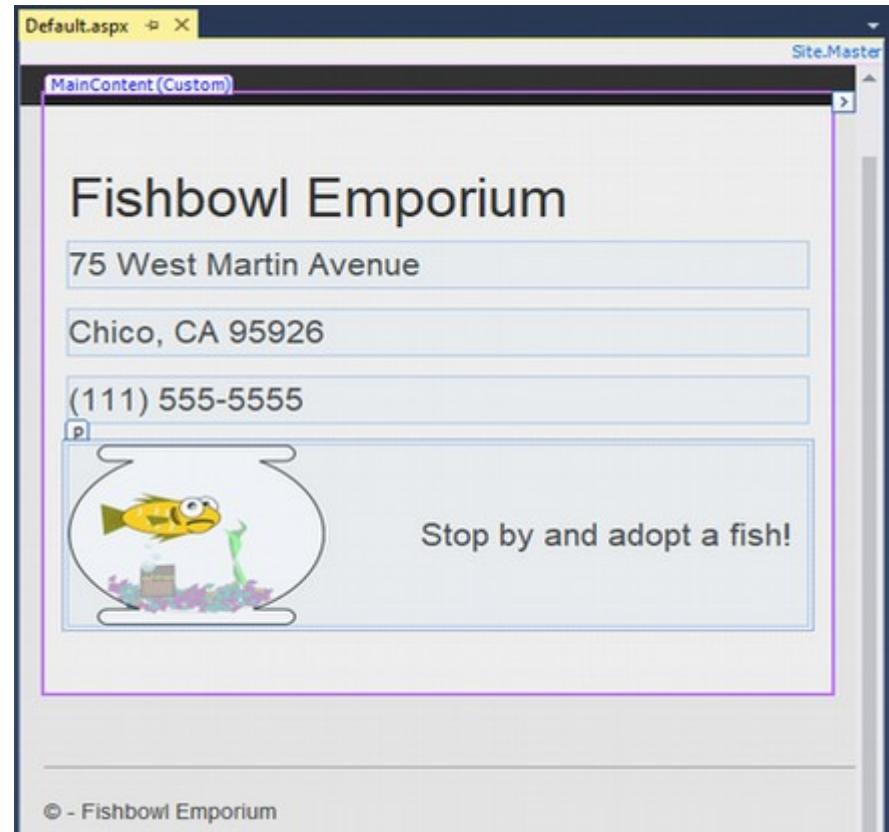
**step 3**

- add a text next by the image: **Figure 13-25**

-> click to the right of the image control and then press the Spacebar 10 times, and type: **Stop by and adopt a fish!** - but do not press Enter key



**Figure 13-25**



**Figure 13-26**

**step 4**

- delete all unnecessary info

**Figure 13-26**

-> select all of the information that appears below the image on the Web page, beginning with the text that starts with "ASP.NET is a free web framework" and ending with the text that appears above the footer

-> press the **Delete** key

- step 5** - start/test your **Default.aspx** - Home page without the Debugging option to see the changes made:  
-> save and then start the application

Figure 13-27

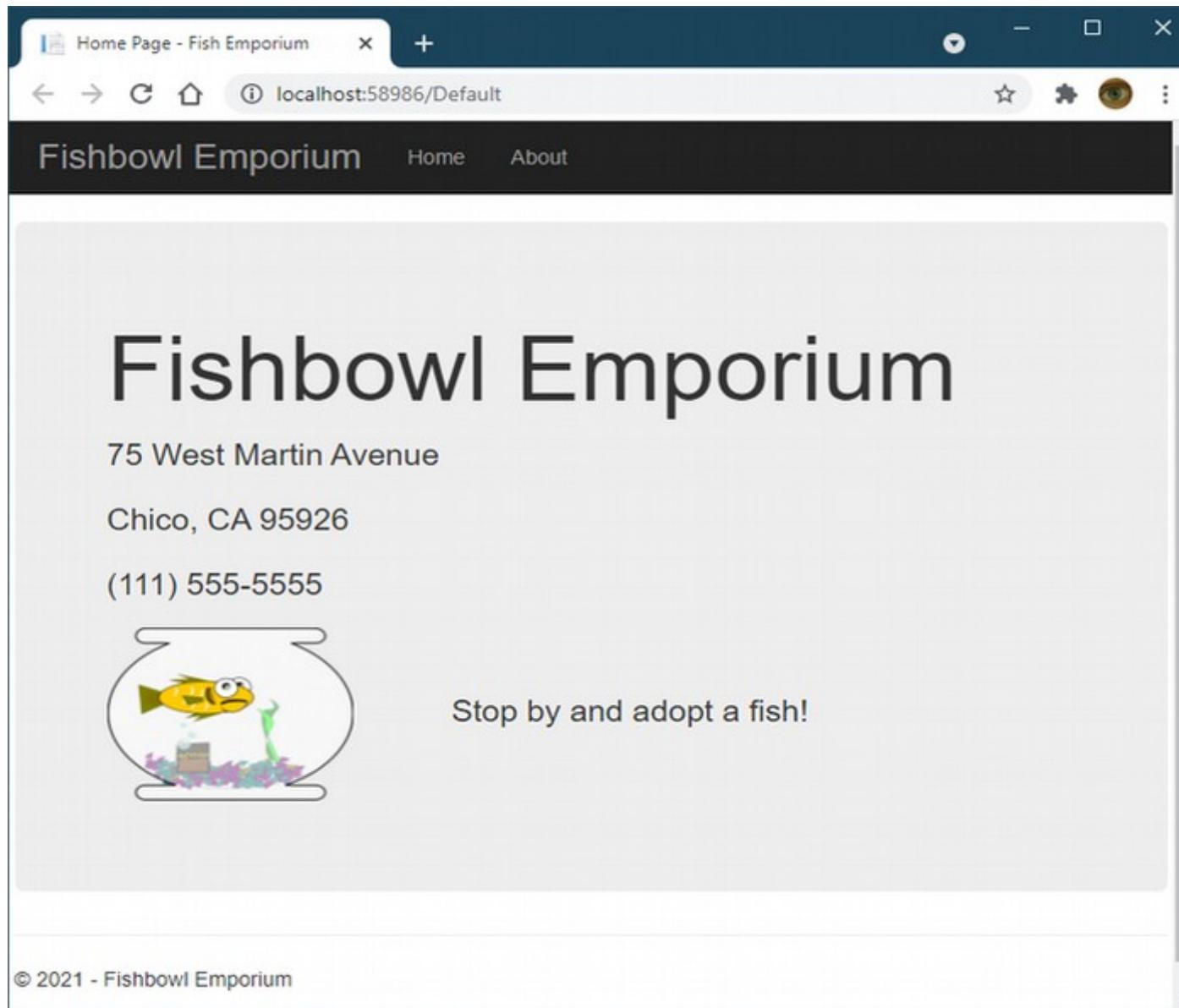


Figure 13-27

**CH13\_F6 - create/personalize the 2nd page - Static Web page About.aspx, e.g. with 01.Fishbowl 5/9**

- the **About.aspx** page created by the template **ASP.NET Web Forms Site** will be the 2nd static Web page included in the application
- you will personalize the page in the next set of steps
- the page will contain a brief description of the store's owners and operators as well as the store's hours of operation

- to personalize the **About.aspx** page - the **2nd Static Web page**: e.g. with **01.Fishbowl 5/9**

**step 0** - open the: **Solution Explorer** window / **About.aspx** page and click the **Design** tab

<- notice that the page inherits the application's name and footer from the **Site.master** page edited in previous lesson:

**CH13\_F4 - modify the 0th page - master page Site.master** (the page creates a consistent layout for the pages in your app = affects all of the **.aspx** pages)

**step 1** - in the **Source** tab change the page title from: **About** to: **About Us**

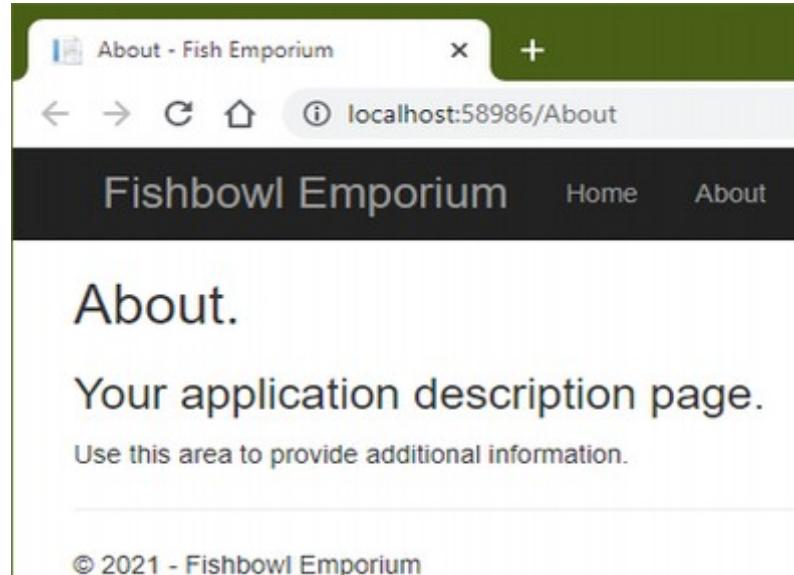
-> click the **Source** tab to view the code

```
1  <%@ Page Title="About" Language="VB" MasterPageFile="~/Site.Master" AutoEventWireup="true" CodeFile="About.aspx.vb" Inherits="About" %>
2
3  <asp:Content ID="BodyContent" ContentPlaceHolderID="MainContent" runat="server">
4      <h2><%: Title %>.</h2>
5      <h3>Your application description page.</h3>
6      <p>Use this area to provide additional information.</p>
7  </asp:Content>
8
```

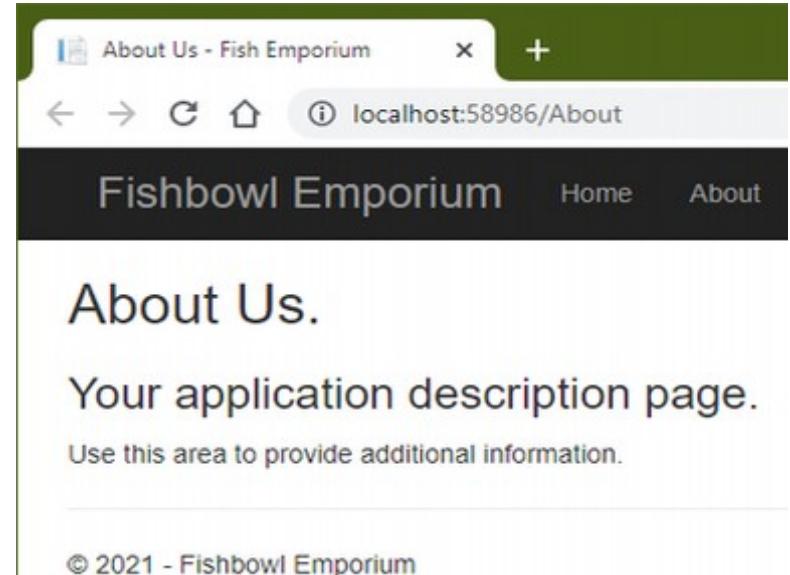
**Figure 13-28a** & **Figure 13-28b**

-> in the 1st line, change the page title from: **About** to: **About Us**

```
1  <%@ Page Title="About Us" Language="VB" MasterPageFile="~/Site.Master" AutoEventWireup="true" CodeFile="About.aspx.vb" Inherits="About" %>
```



**Figure 13-28a** before



**Figure 13-28b** after

**step 2.0** - personalize the text displayed  
-> click the **Design** tab

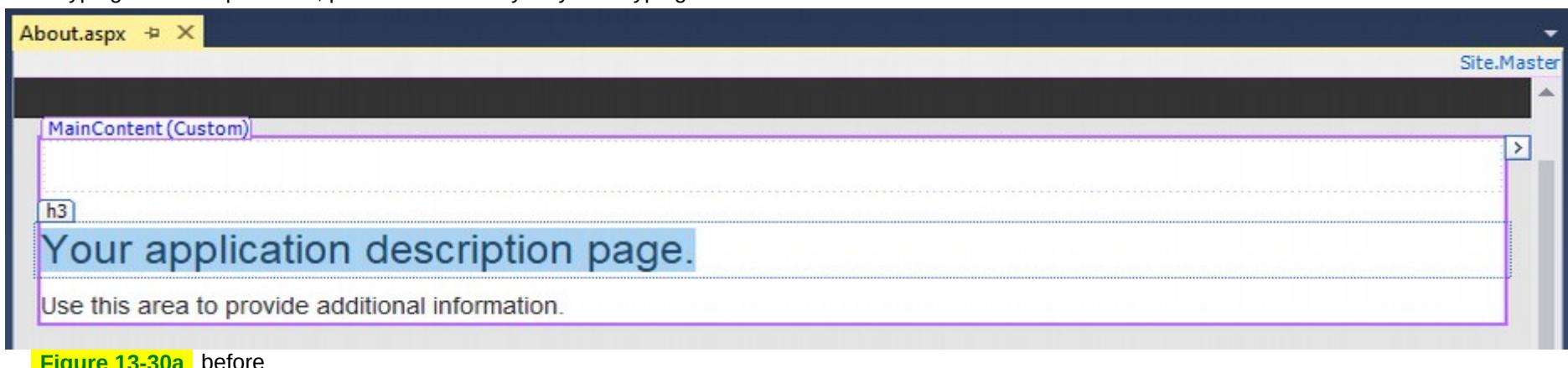
**step 2.1** -> in the 1st line of the **MainContent (Custom)** section, delete the: . (period) **Figure 13-29**



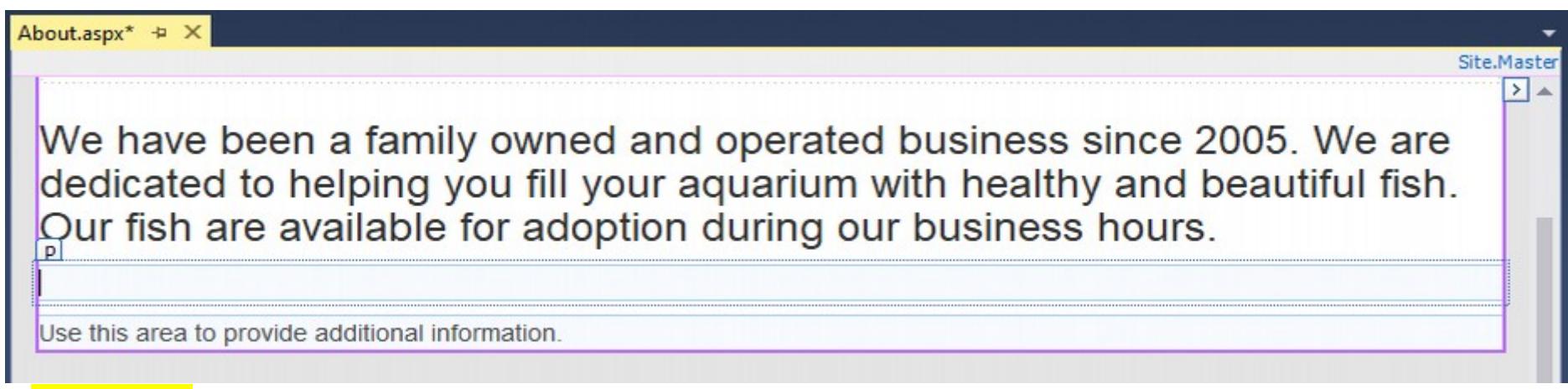
**Figure 13-29**

**Figure 13-29b**

**step 2.2** -> replace the original text: **Your application description page.** with a description shown in:  
<- when typing the description text, press the Enter key only after typing the last sentence



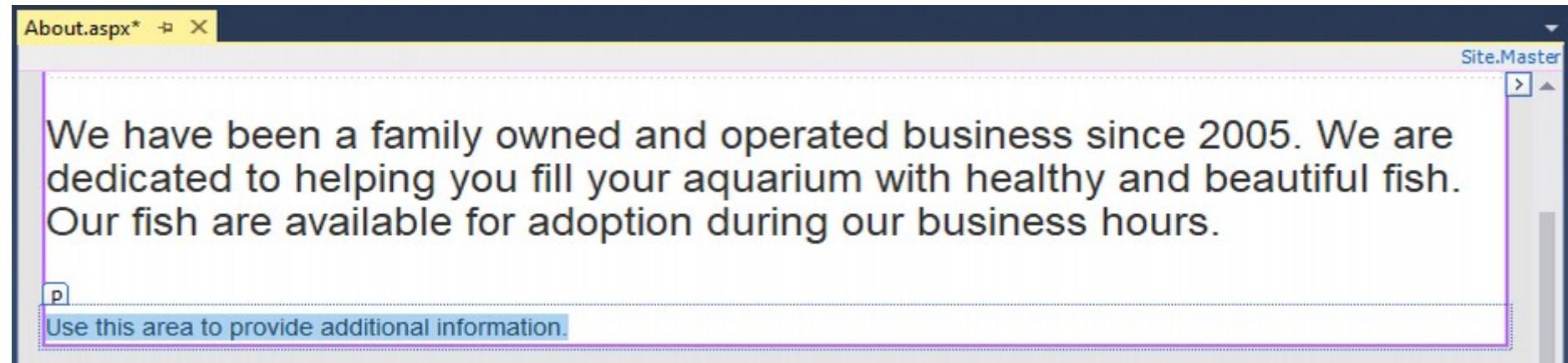
**Figure 13-30a** before



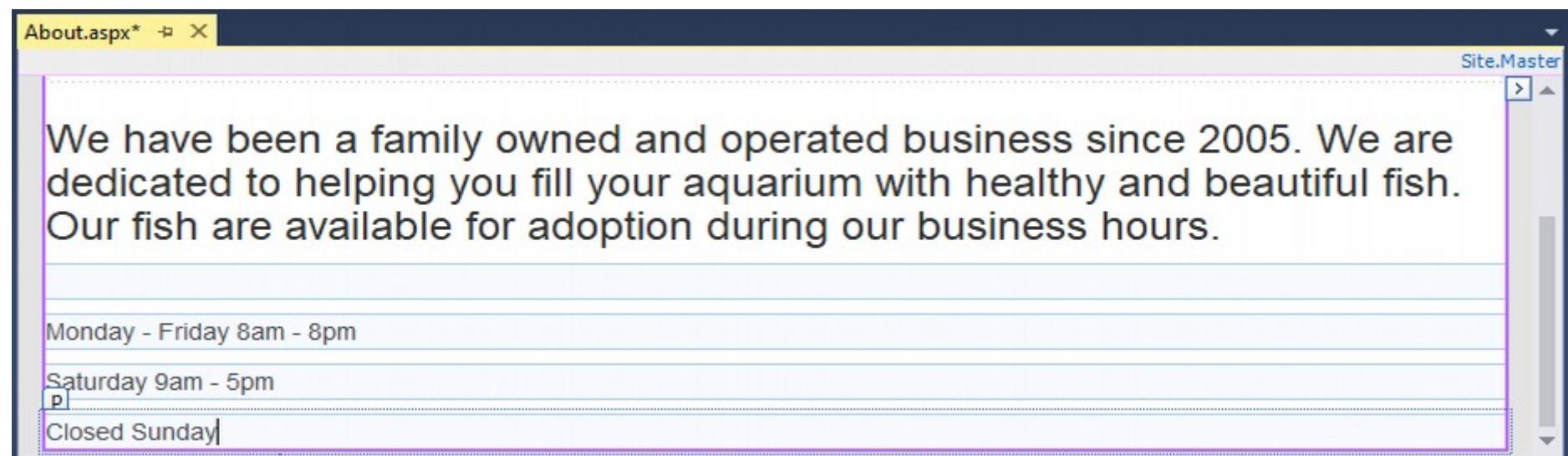
**Figure 13-30b** after

**step 2.3** -> replace the original text: **Use this area to provide additional information.** with a hours information shown in: **Figure 13-31b**

<- when typing the hours information, press the Enter key after every line



**Figure 13-31a** before



**Figure 13-31b** after

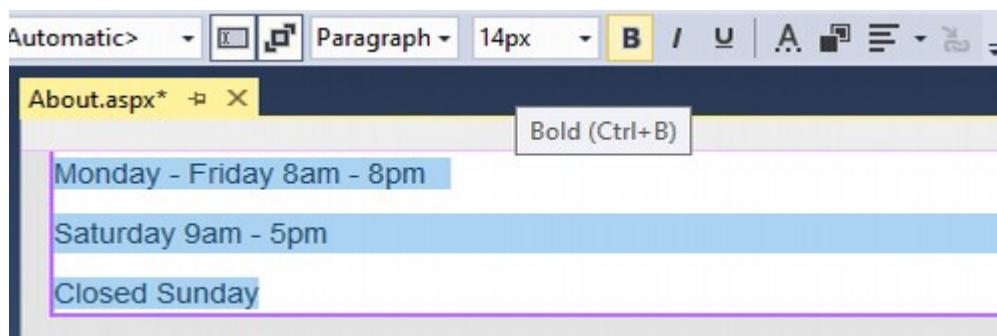
**step 2.4** - format the hours information entered in previous step - bold and align to the center

-> select/highlight the 3 lines that contain the hours information

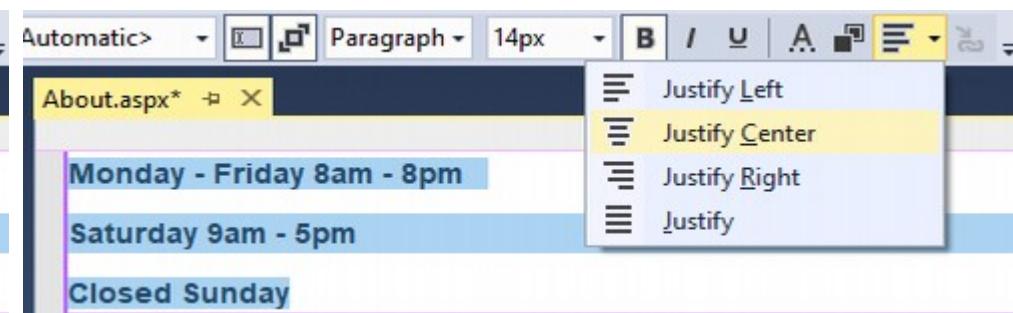
-> on the **Formatting toolbar** click the button **B - Bold (Ctrl+B)**

-> on the **Formatting toolbar** click the **Alignment** list arrow and then click: **Justify Center** in the list

**Figure 13-32a** & **Figure 13-32b**



**Figure 13-32a**



**Figure 13-32b**

**step 3** - start/test your **Default.aspx** - Home page without the Debugging option to see the changes made:

-> save and then start the application

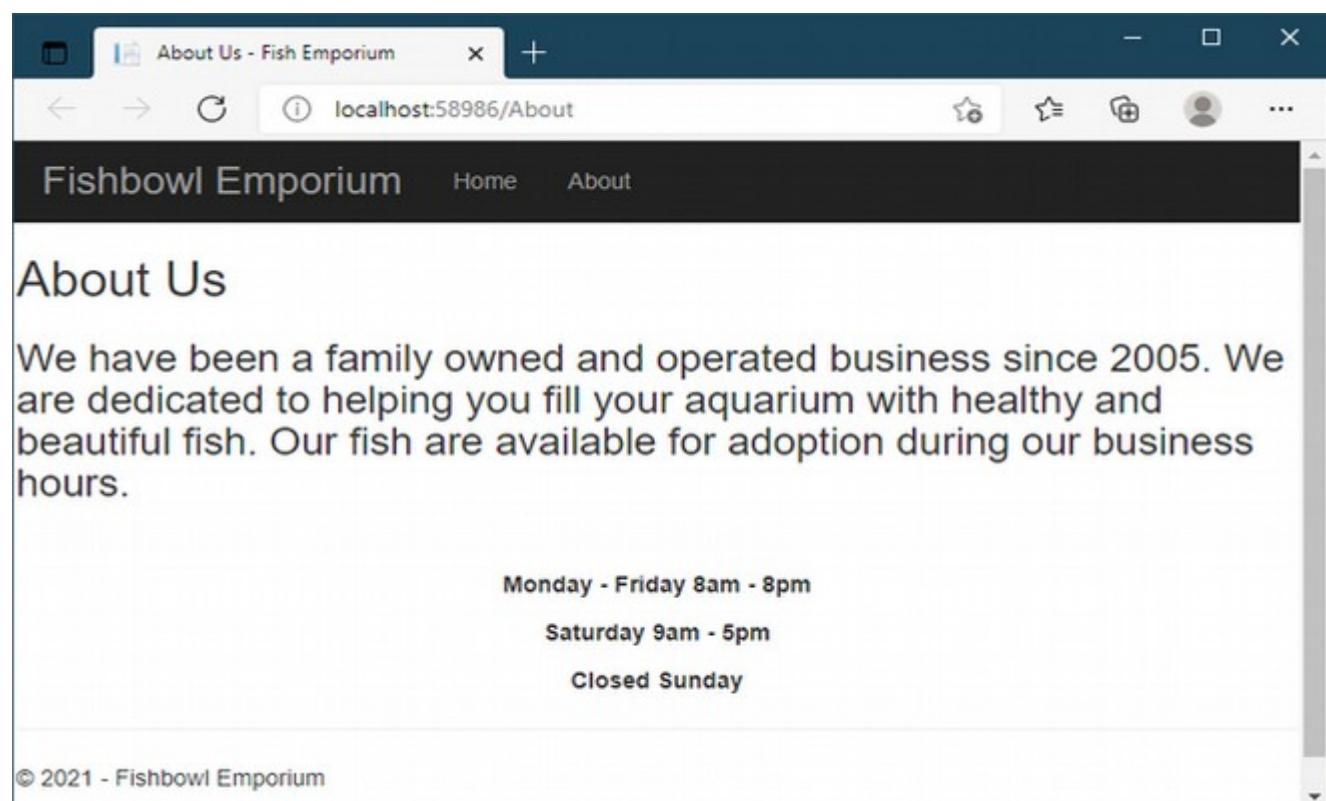
<- notice that when you resize the window, the description text follow it -> because you did not press the Enter key after each line, so the text is formatted to follow the window size

-> click **Home** to display the **Default.aspx** - Home page

-> click **About** to display the **About.aspx** page

-> click **Fishbowl Emporium** to also display the **Default.aspx** - Home page

-> close the browser window



**Figure 13-33**

### CH13\_F7 - starting/testing Web Application in VS 2017 v 15.9.36 with different browsers

- while you are creating a Web site application, it is helpful to see how the Web pages look in different browsers - because some pages might look slightly different
- it is very easy to change browsers from within the Visual Basic IDE:
  - > on the **Standard toolbar's Start** button click the list arrow and select the desired browser available on your computer

Figure 13-34

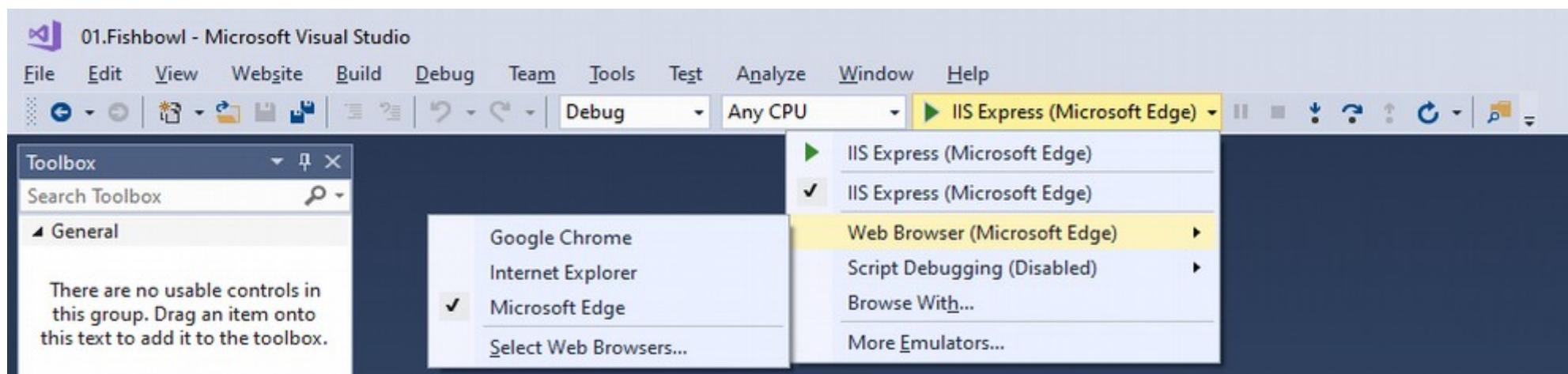


Figure 13-34

## CH13\_F8 - closing and opening a Web Site Application - VS 2017 book version vs updated VS 2017 v 15.9.36

- closing a Web Site Application: **->** on the **Menu** bar click: **File / Close Solution** **Figure 13-35**  
**->** for verification display the **Solution Explorer** window to verify the **Fishbowl** application is closed

- opening a Web Site Application:

- a). by the book with an old version of VS 2017:
  - > 1).** on the **Menu** bar click: **File / Open Web Site...** **Shift+Alt+O** **Figure 13-36a**
  - > 2).** open the ...**VB2017\Chap13\Exercise\01.Fishbowl** folder and then click **Open**
- <-** when starting/testing the page, an error occurs: **Figure 13-37**  
- so a different way:
  - > 2).** open the ...**VB2017\Chap13\Exercise\01.Fishbowl\01.Fishbowl** folder and then click **Open**

- b). in VS 2017 v 15.9.36:
  - > 1).** on the **Menu** bar click: **File / Open Project...** **Ctrl+O** **Figure 13-36b**
  - > 2).** open the ...**VB2017\Chap13\01.Fishbowl** folder and then click **Open**

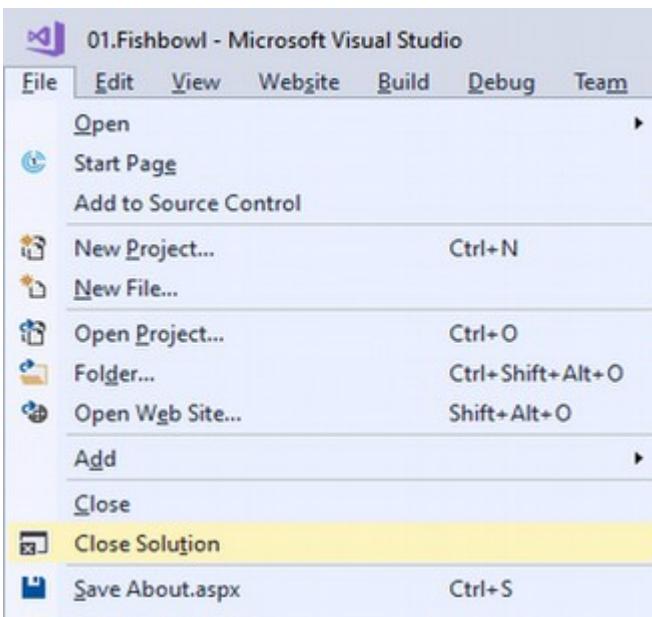


Figure 13-35

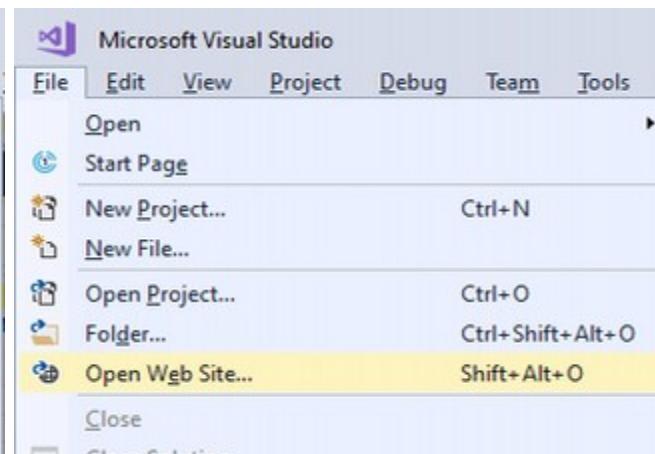


Figure 13-36a

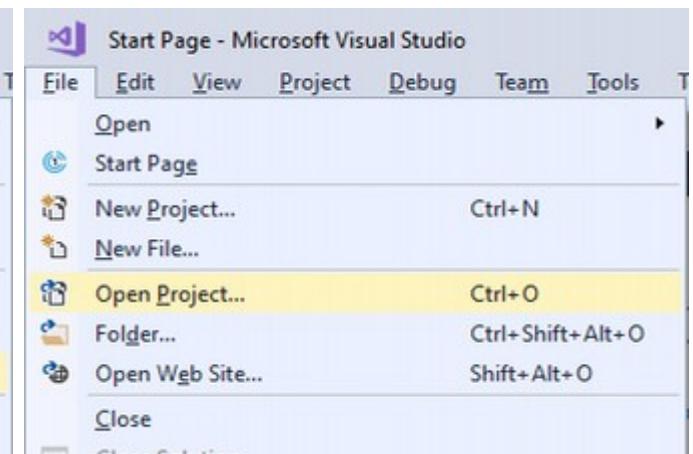


Figure 13-36b

IIS 10.0 Detailed Error - 403.14 - X +

localhost:61850

## HTTP Error 403.14 - Forbidden

The Web server is configured to not list the contents of this directory.

**Most likely causes:**

- A default document is not configured for the requested URL, and directory browsing is not enabled on the server.

**Things you can try:**

- If you do not want to enable directory browsing, ensure that a default document is configured and that the file exists.
- Enable directory browsing.
  - Go to the IIS Express install directory.
  - Run `appcmd set config /section:system.webServer/directoryBrowse /enabled:true` to enable directory browsing at the server level.
  - Run `appcmd set config ["SITE_NAME"] /section:system.webServer/directoryBrowse /enabled:true` to enable directory browsing at the site level.
- Verify that the configuration/system.webServer/directoryBrowse@enabled attribute is set to true in the site or application configuration file.

**Detailed Error Information:**

|                     |                        |                      |                                                                     |
|---------------------|------------------------|----------------------|---------------------------------------------------------------------|
| <b>Module</b>       | DirectoryListingModule | <b>Requested URL</b> | http://localhost:61850/                                             |
| <b>Notification</b> | ExecuteRequestHandler  | <b>Physical Path</b> | C:\Users\Martin Brett\Documents\VB2017\Chap13\_Exercise\01.Fishbowl |
| <b>Handler</b>      | StaticFile             | <b>Logon Method</b>  | Anonymous                                                           |
| <b>Error Code</b>   | 0x00000000             | <b>Logon User</b>    | Anonymous                                                           |

**More Information:**

This error occurs when a document is not specified in the URL, no default document is specified for the Web site or application, and directory listing is not enabled for the Web site or application. This setting may be disabled on purpose to secure the contents of the server.

[View more information >](#)

Figure 13-37

## CH13\_APPLY THE CONCEPTS LESSON

### CH13\_A1 - create 3rd page - Dynamic Web page Calculator.aspx: 1/4 Repurpose an existing unused Contact.aspx, e.g. with 01.Fishbowl 6/9

- in addition to:

- a master page created/modified in: CH13\_F4 - modify the 0th page - master page Site.master (the page creates a consistent layout for the pages ...)
- 2 static Web pages, created/personalized in: CH13\_F5 - create/personalize the 1st page - Static Web page Default.aspx (Home page)...  
CH13\_F6 - create/personalize the 2nd page - Static Web page About.aspx...

the Fishbowl Emporium application will contain 1 dynamic Web page named **Calculator.aspx**, as already seen on:

Figure 13-2

- the page provides areas for the user to enter the length, width, and height of an aquarium
- when the user clicks the Submit button, the page calculates and displays the number of gallons of water needed to fill the aquarium

- in addition to the Home and About links, the page contains:

- a Calcuator link
- a table
- 4 labels
- 3 text boxes
- an image
- 3 validation controls
- a button

- rather than adding a new Web page to the application, you can use one of the unused pages created by the **ASP.NET Web Forms Site** template

- in this case, you will repurpose the **Contact.aspx** page



- to repurpose/recycle an existing unused Contact.aspx for Calculator.aspx page: e.g. with 01.Fishbowl 6/9

-> open the: ...VB2017\Chap13\Exercise\01.Fishbowl\01.Fishbowl.sln

**step 1.0** - rename existing unused page and open it for editing

- > in the **Solution Explorer** window, Rclick the **Contact.aspx** page and rename it to: **Calculator.aspx**
- > when the page opens, click the **Source** tab to view the code

```
1  <%@ Page Title="Contact" Language="VB" MasterPageFile("~/Site.Master" AutoEventWireup="true" CodeFile="Calculator.aspx.vb" Inherits="Contact" %>
2
3  <asp:Content ID="BodyContent" ContentPlaceHolderID="MainContent" runat="server">
4      <h2><%: Title %>.</h2>
5      <h3>Your contact page.</h3>
6      <address>
7          One Microsoft Way<br />
8          Redmond, WA 98052-6399<br />
```

```

9      <abbr title="Phone">P:</abbr>
10     425.555.0100
11   </address>
12
13   <address>
14     <strong>Support:</strong> <a href="mailto:Support@example.com">Support@example.com</a><br />
15     <strong>Marketing:</strong> <a href="mailto:Marketing@example.com">Marketing@example.com</a>
16   </address>
17 </asp:Content>
18

```

**step 1.1** - you will change the page's title and close the page

-> in the first line of text, change the page's title from: **Contact** to: **Calculator**

```

1  <%@ Page Title="Contact" Language="VB" MasterPageFile="~/Site.Master" AutoEventWireup="true" CodeFile="Calculator.aspx.vb" Inherits="Contact" %>
1  <%@ Page Title="Calculator" Language="VB" MasterPageFile="~/Site.Master" AutoEventWireup="true" CodeFile="Calculator.aspx.vb" Inherits="Contact" %>

```

-> close the **Calculator.aspx** page

**step 2.0** - you will modify the **Site.master** page to use the recycled Web page

- before making any further modifications to the **Calculator.aspx** page, you will tell the **Site.master** page to display the **Calculator** link and also to open the **Calculator.aspx** file when the link is clicked.

**step 2.1** -> in the **Solution Explorer** window, open the **Site.master** page and click the **Source** tab to view the code

-> locate the **Contact** link, which you changed into comment in: **CH13\_F4 - modify the 0th page - master page Site.master ...** & **step 4.1**

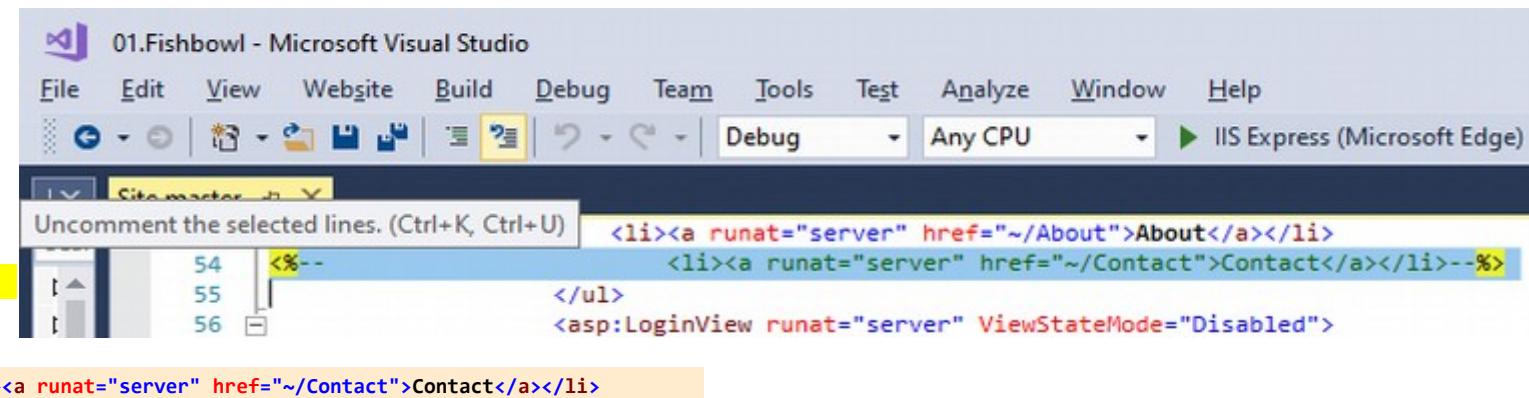
```

54  <%-->
      <li><a runat="server" href "~/Contact">Contact</a></li>--%>

```

-> select/highlight the line 54 and on the toolbar **Standard** click the button named: **Uncomment the selected lines. (Ctrl+K, Ctrl+U)**

**Figure 13-38**



**Figure 13-38**

**step 2.2** -> change both occurrences of the word: **Contact** to: **Calculator**

```

54  <li><a runat="server" href "~/Contact">Contact</a></li>

```

**step 2.3** -> save and then start the application to view the changes made (Ctrl + F5)

<- notice the **Calculator** link appears on the Home (Default) page

-> click the **Calculator** link to display the **Calculator.aspx** page

<- do not be concerned about the . (period) that follows  
the word **Calculator**

-> close the browser and **Site.master** window

Figure 13-39

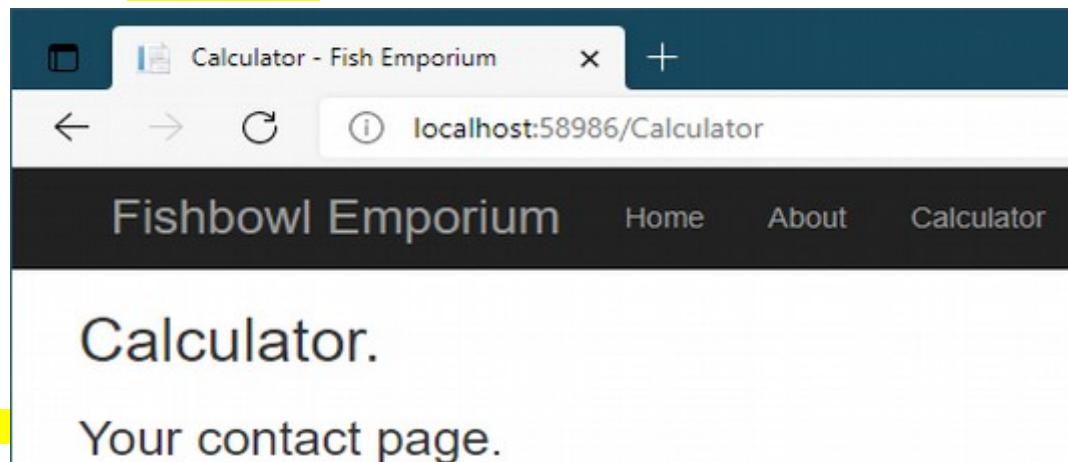


Figure 13-39

**step 3** - now, you will change the page's title from: **Calculator**. to: **Calculator: Gallons of water to fill my aquarium**.

-> in the **Solution Explorer** window, open the **Calculator.aspx** page and click the **Source** tab to view the code

-> locate the **<h2><%: Title %>.</h2>**

4      **<h2><%: Title %>.</h2>**

-> click immediately before the . (period) and type: : **Gallons of water to fill my aquarium**

<- notice the space after the : **(colon)**

<- be sure to keep . **(period)** after the word: ... **aquarium**.

4      **<h2><%: Title %>: Gallons of water to fill my aquarium.</h2>**

-> save and then start the application to view the changes made (Ctrl + F5)

Figure 13-40

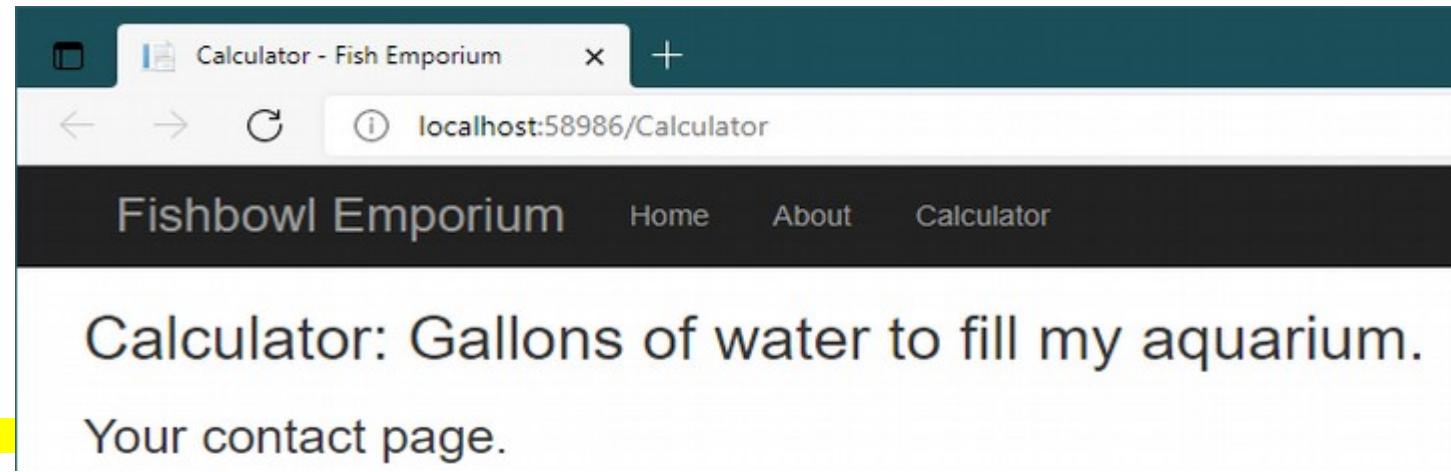


Figure 13-40

**CH13\_A2 - create 3rd page - Dynamic Web page Calculator.aspx: 2/4 add a Table and Controls, e.g. with 01.Fishbowl 7/9**

- in this section, you will add a **table** to the **Calculator.aspx** page
- **tables** are often used on Web pages to make it easier to align the controls on the page

- to add a **table** to the **Calculator.aspx** page: e.g. with **01.Fishbowl 7/9**

-> open the: **Solution Explorer** window / **Calculator.aspx** page and click the **Design** tab

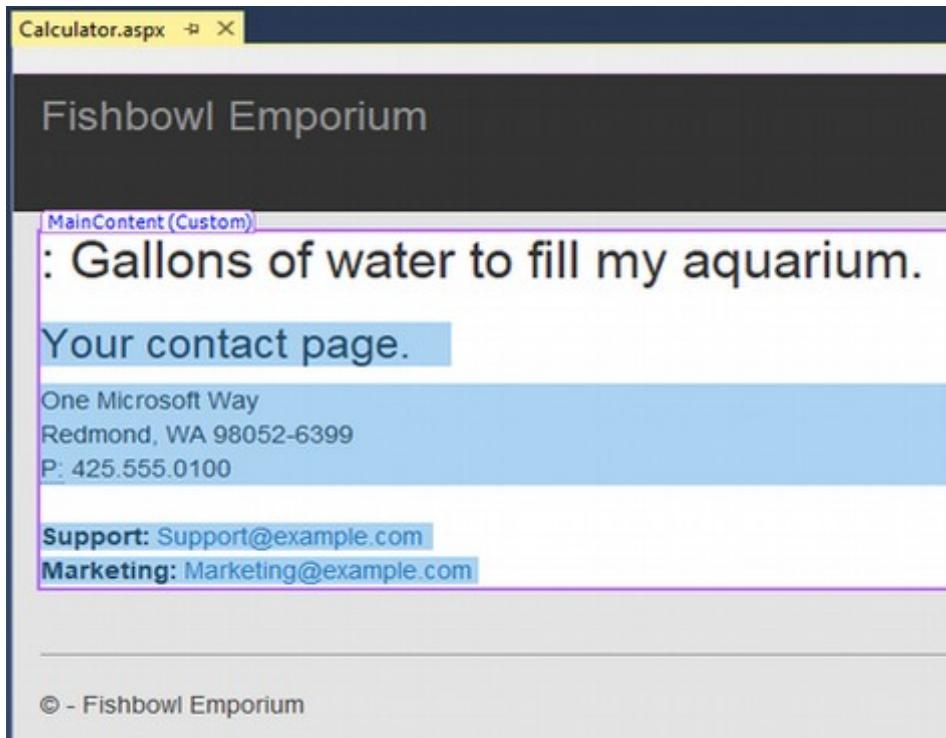
**step 1** - delete all the recycled page's text and make a space for the next step

-> select/highlight all of the text that appears below the page's title and press **Delete** (Your contact page.....Marketing: Marketing@example.com)

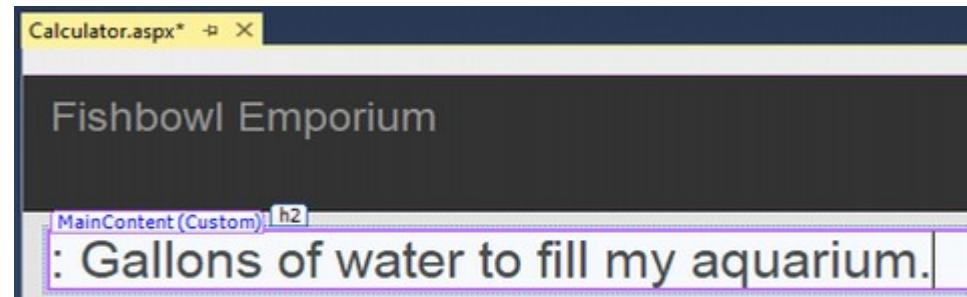
-> click immediately after the **.** (period) in the page's title and press **Enter**

**Figure 13-41b** & **Figure 13-41c**

**Figure 13-41a**

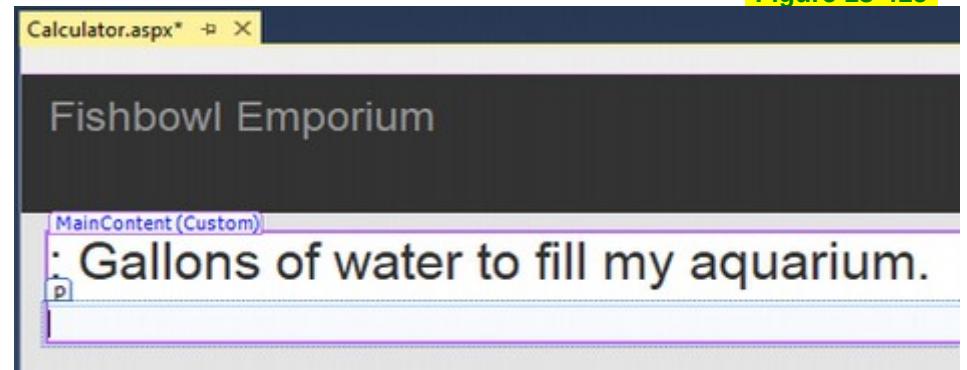


**Figure 13-41a**



**Figure 13-41b**

**Figure 13-41c**



**step 2.0** - you will add a **table** using dialog box **Insert Table** & resize the column widths & fill the cells with text and controls

**step 2.1** - you will open and use the dialog box: **Insert Table** to create your table

-> on the menu bar click **Table / Insert Table** to open the **Insert Table** dialog box

-> set the **Size** of your table: - **Rows = 6**

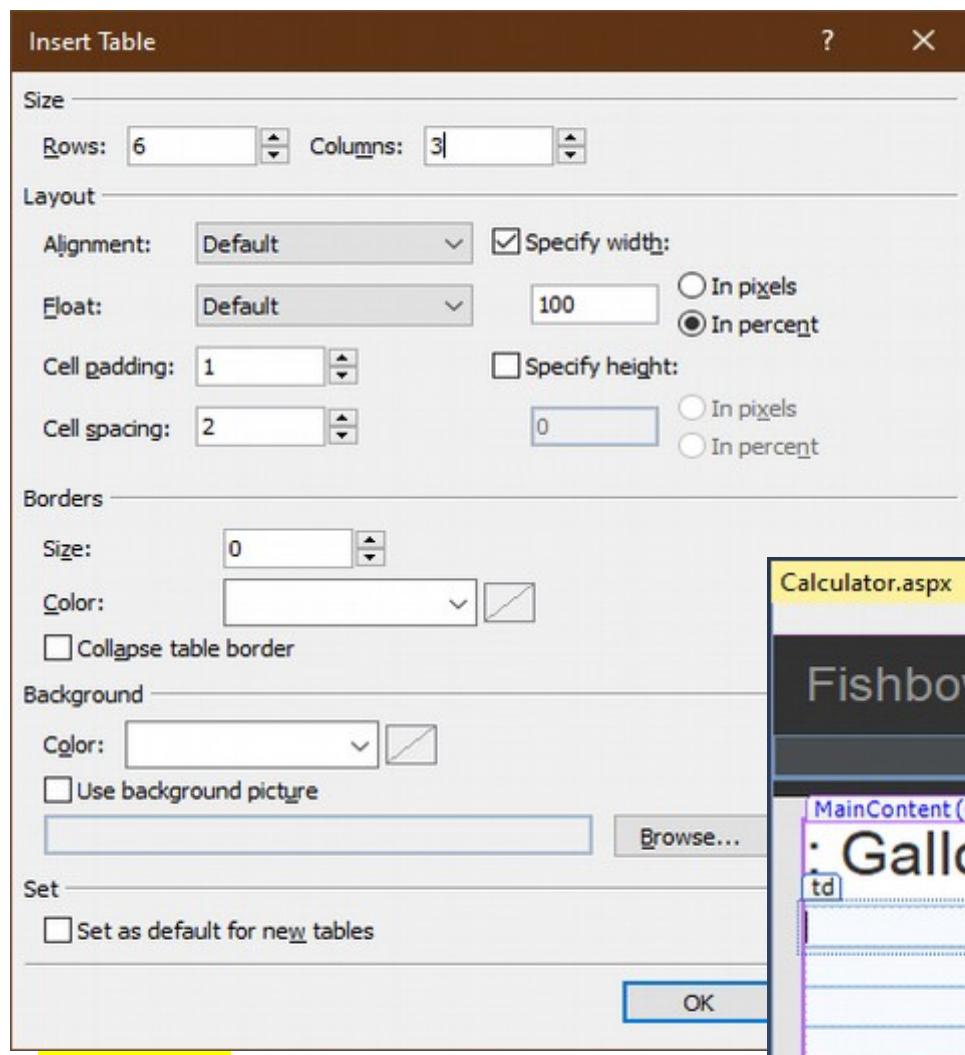
- **Columns = 3**

-> click the **OK** button to close the **Insert Table** dialog box

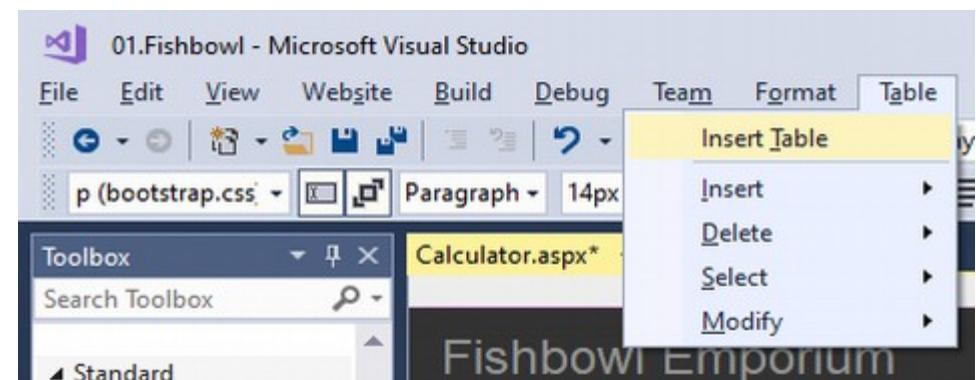
<- observe the table you just created (by default the cell size is proportional):

**Figure 13-42a** & **Figure 13-42b**

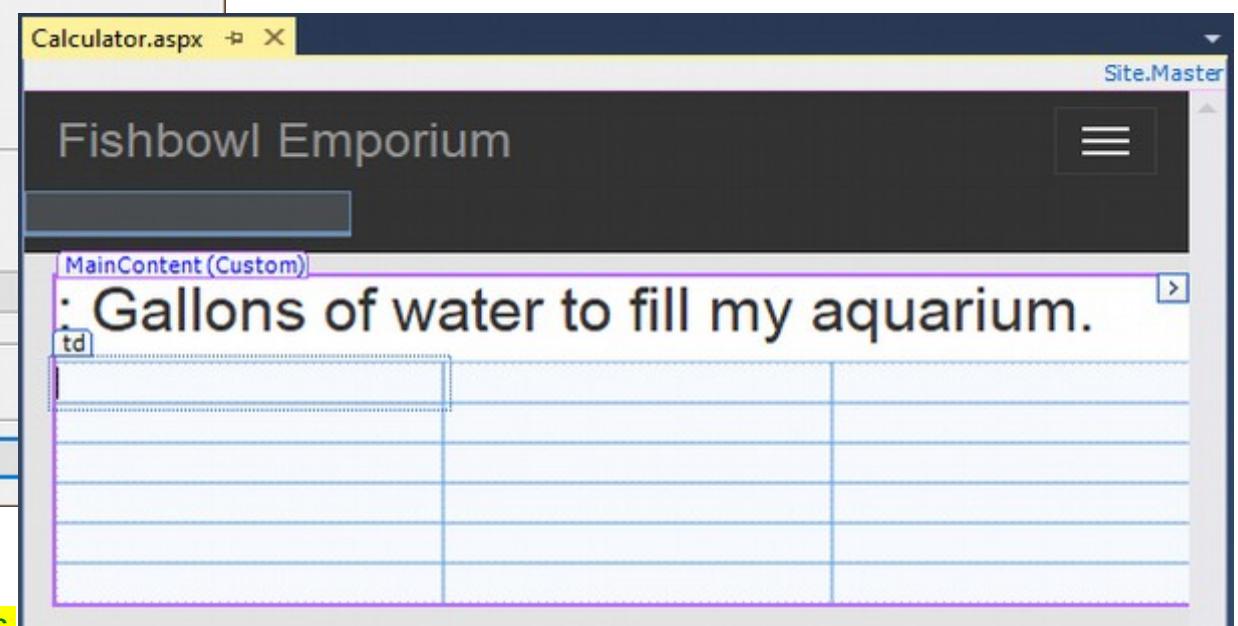
**Figure 13-42c**



**Figure 13-42b**



**Figure 13-42a**



**Figure 13-42c**

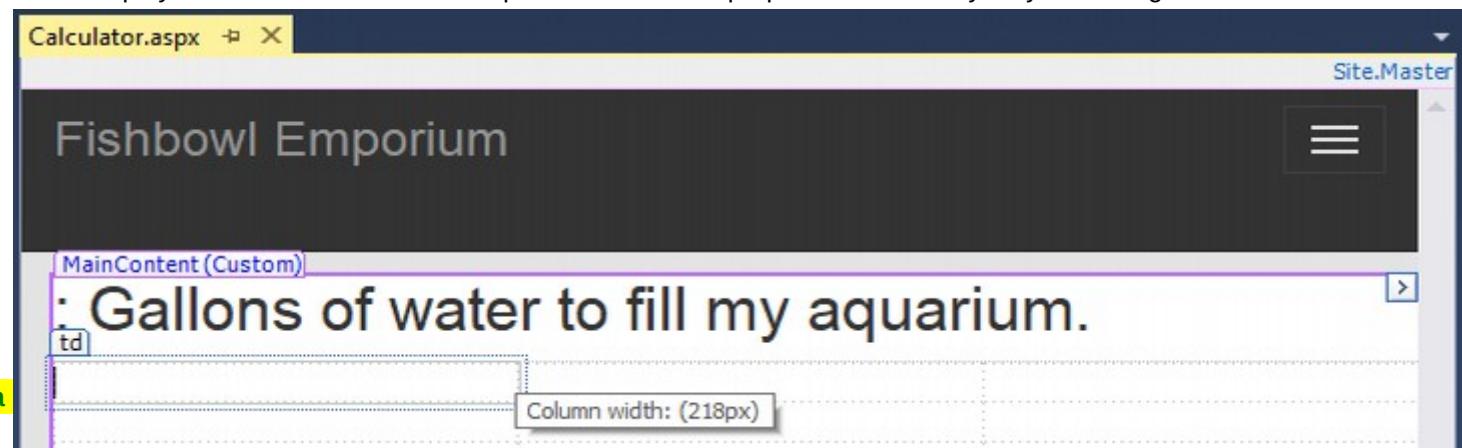
**step 2.2**

- you will resize the default proportional column widths

-> position your mouse pointer on the line that devides the **1st** and **2nd** columns in the table

**Figure 13-43a**

<- notice the screen tip box appears and displays the width of the column in pixels - the default proportional size may vary according the window size



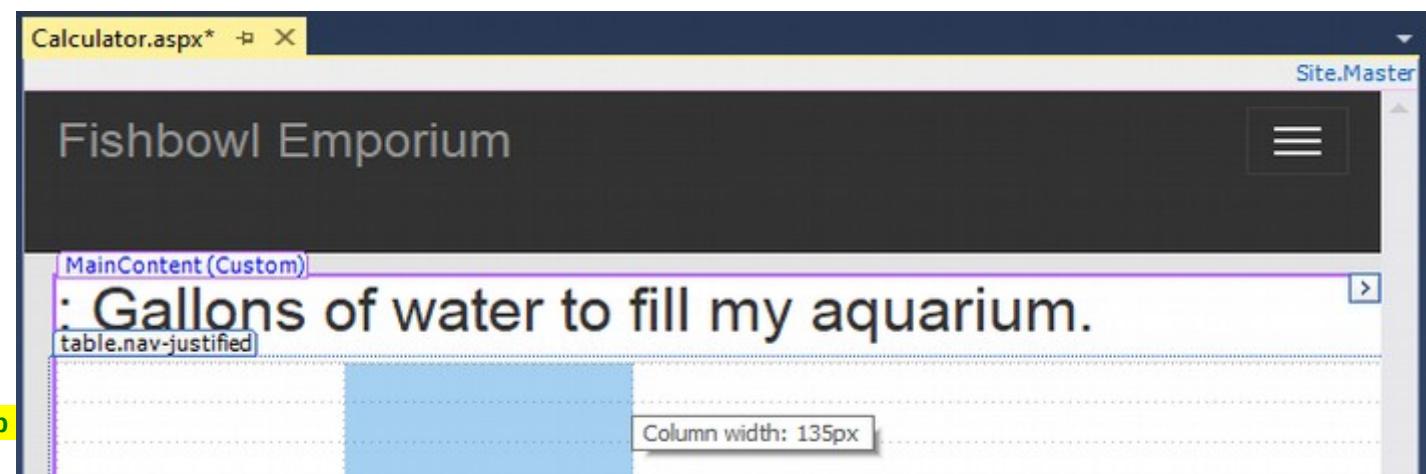
**Figure 13-43a**

-> use your mouse pointer to drag the line between the **1st** and **2nd** column to the left until the **1st column width = 135px** wide

-> use your mouse pointer to drag the line between the **2nd** and **3rd** column to the left until the **2nd column width = 135px** wide

**Figure 13-43b**

**Figure 13-43b**



**Figure 13-43b**

**step 2.3**

- you will fill the cells with a description texts & TextBoxes

-> into **1st row & 1st column** cell: type the text: **Length (inches):**

**Figure 13-44a**

-> press **Tab** key to get into **1st row & 2nd column** cell

-> into **1st row & 2nd column** cell: from **Toolbox** window, drag **TextBox** tool

**Figure 13-44b**

-> open window **Properties** for the **TextBox1**

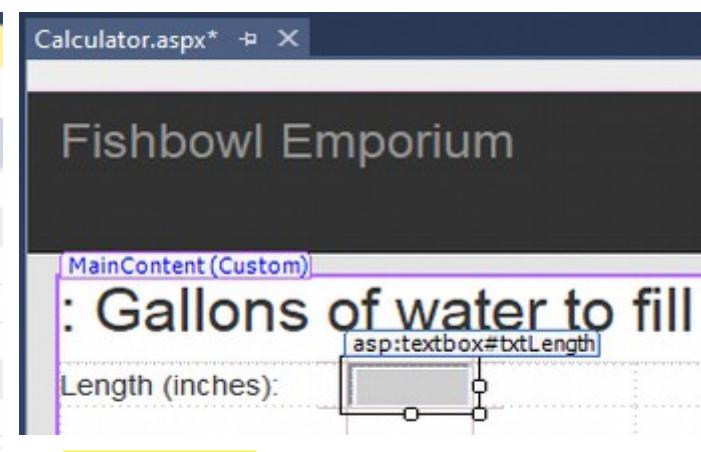
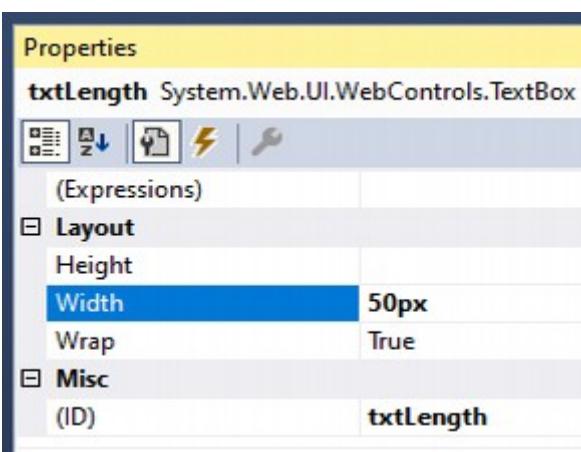
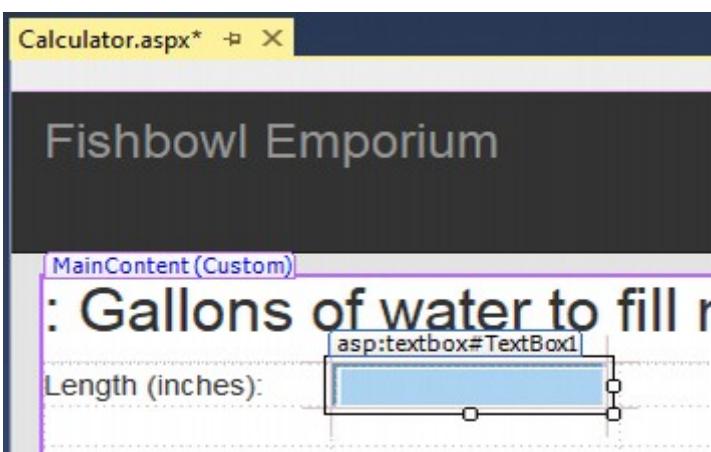
<- notice that unlike the controls on a Windows Form, the controls on a Web page have an **(ID)** property rather than a **(Name)** property

-> change original: **(ID) = TextBox1** to: **(ID) = txtLength**

-> set: **Width = 50px**

<- notice the changes you have just made

**Figure 13-44c**



-> fill the table cells as shown in: **Figure 13-44d**

|                  |                                         |
|------------------|-----------------------------------------|
| Length (inches): | <input type="text"/>                    |
| Width (inches):  | <input type="text"/>                    |
| Height (inches): | <input type="text"/>                    |
| Water (gallons): | <input type="text" value="lblGallons"/> |

- <- Toolbox window / TextBox control (ID) = txtLength, Width = 50px
- <- Toolbox window / TextBox control (ID) = txtWidth, Width = 50px
- <- Toolbox window / TextBox control (ID) = txtHeight, Width = 50px
- <- Toolbox window / Label control (ID) = lblGallons, Text = nothing
- <- Toolbox window / Button control (ID) = btnSubmit, Text = Submit

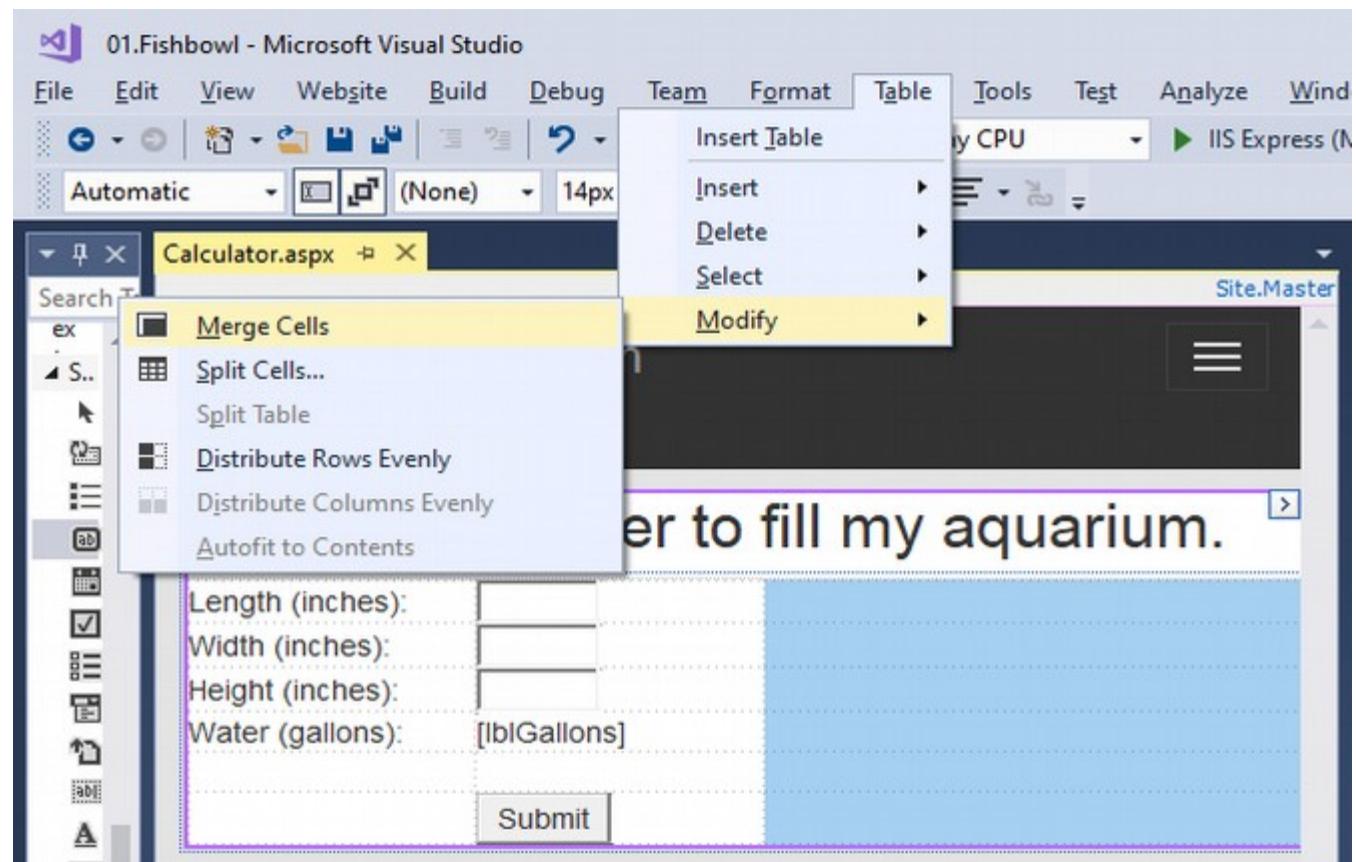
**Figure 13-44d**

**step 2.4** - the **3rd** column cells will be merged and will contain an **Image** control, containing **Aquarium.png**, but first the picture must be added to your application

- so, you can merge the cells in the table's **3rd** column:

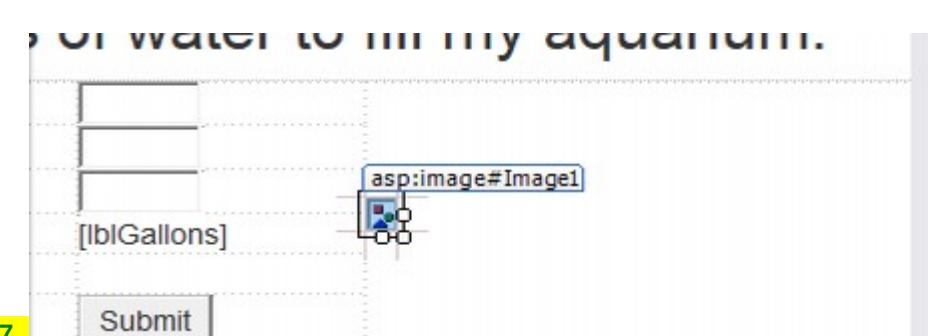
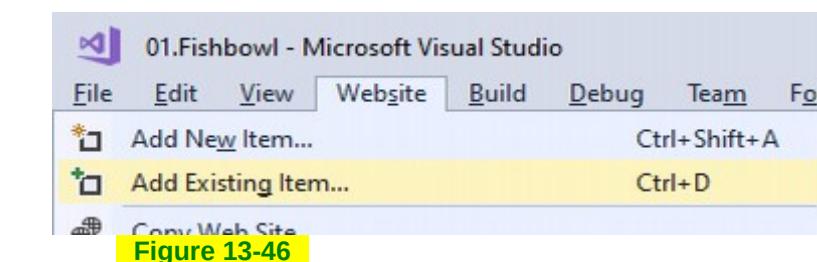
-> click the **1st** cell in the table's **3rd** column and select/highlight all **6 rows** under

-> click on menu bar: **Table / Modify / Merge Cells** **Figure 13-45**



- before you can add an image to your Web site, you must first add the image to the application

-> click on menu bar: **Website / Add Existing Item...** **Ctrl+D**, locate the **Aquarium.png** image in ...VB2017\Chap13 folder and click the **Add** button



- now you can add an **Image** control to the merged cells by dragging:

-> into the merged cell: from **Toolbox** window, drag **Image** tool **Figure 13-47**

- finally, you can add a picture **Aquarium.png** to your Web page:

-> in the **Image1** control's **Properties**, locate the property: **ImageUrl** and click the ... (ellipsis) button to open: **Select Image** dialog box

**Figure 13-48a**

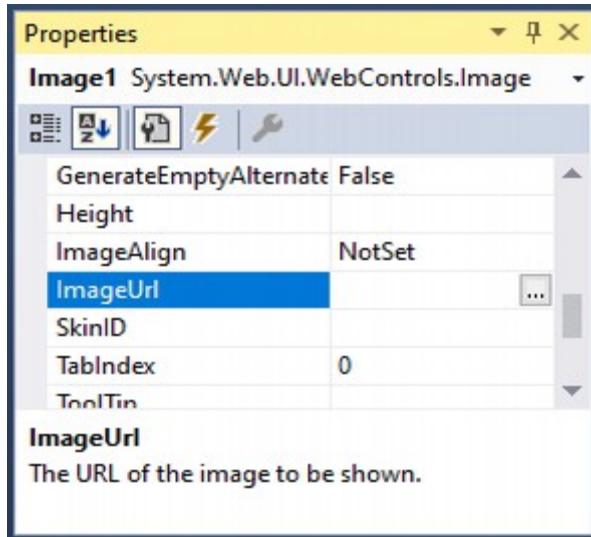
- the dialog box contains an images added to your application, which can be used in your Web pages

-> in the dialog box: **Select Image**, click the **Aquarium.png** file and then click the **OK** button to add file to your **Image** control

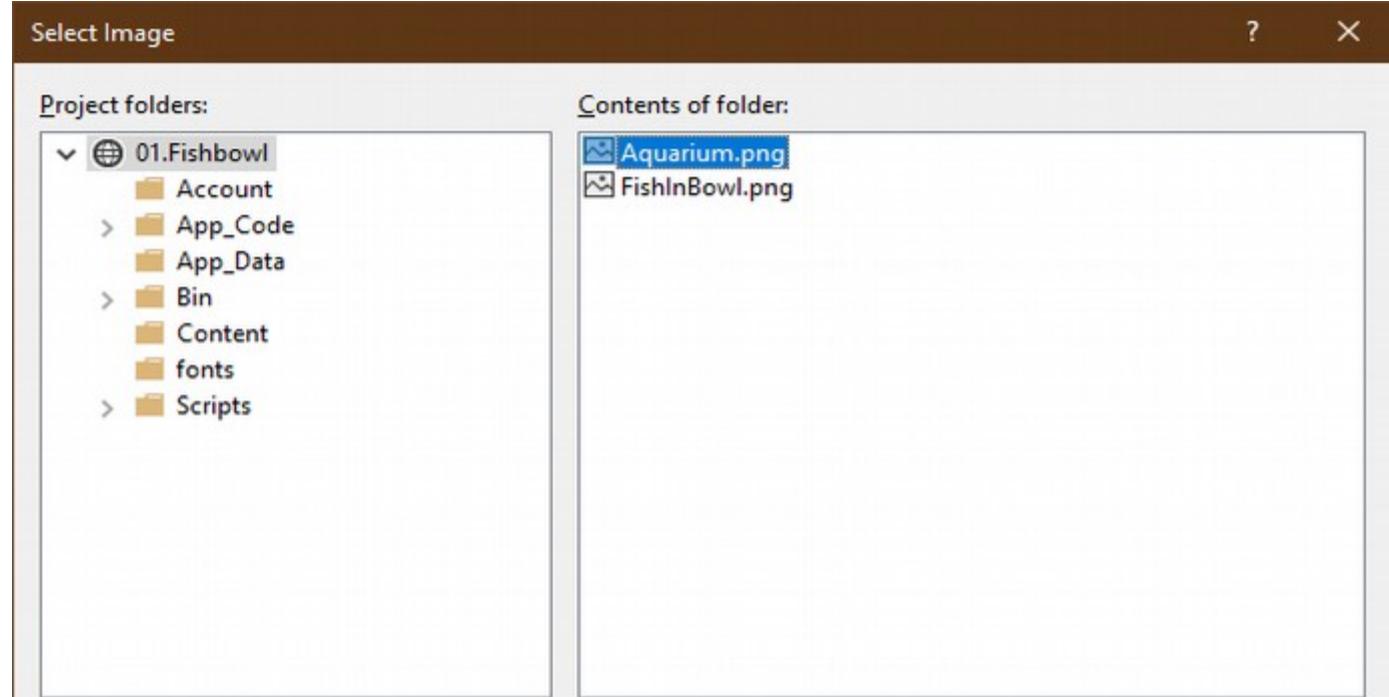
**Figure 13-48b**

-> see the **Properties** window with added picture **Aquarium.png**

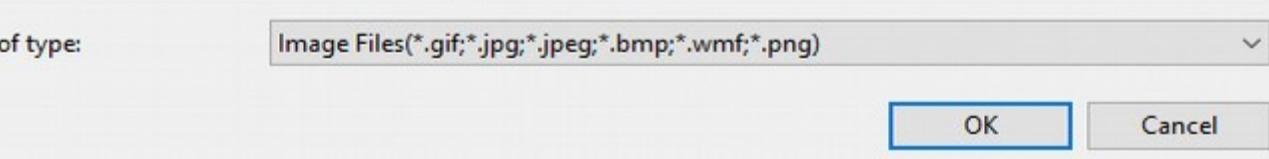
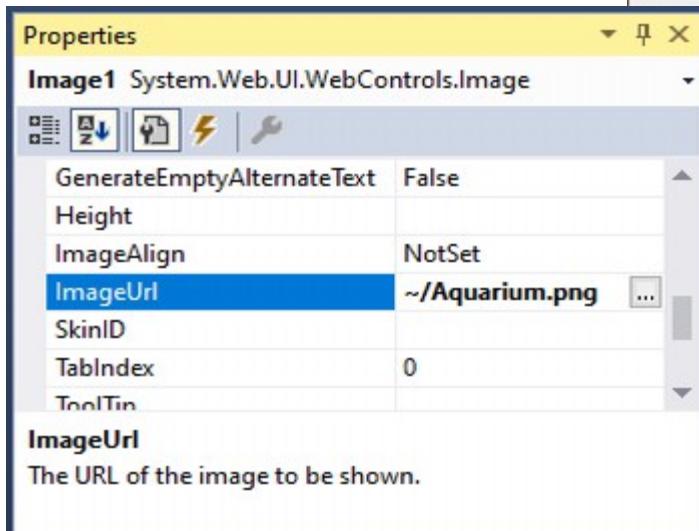
**Figure 13-48c**



**Figure 13-48a**



**Figure 13-48a**



**Figure 13-48b**

**Figure 13-48c**

-> look at the design, you just created:

Figure 13-49

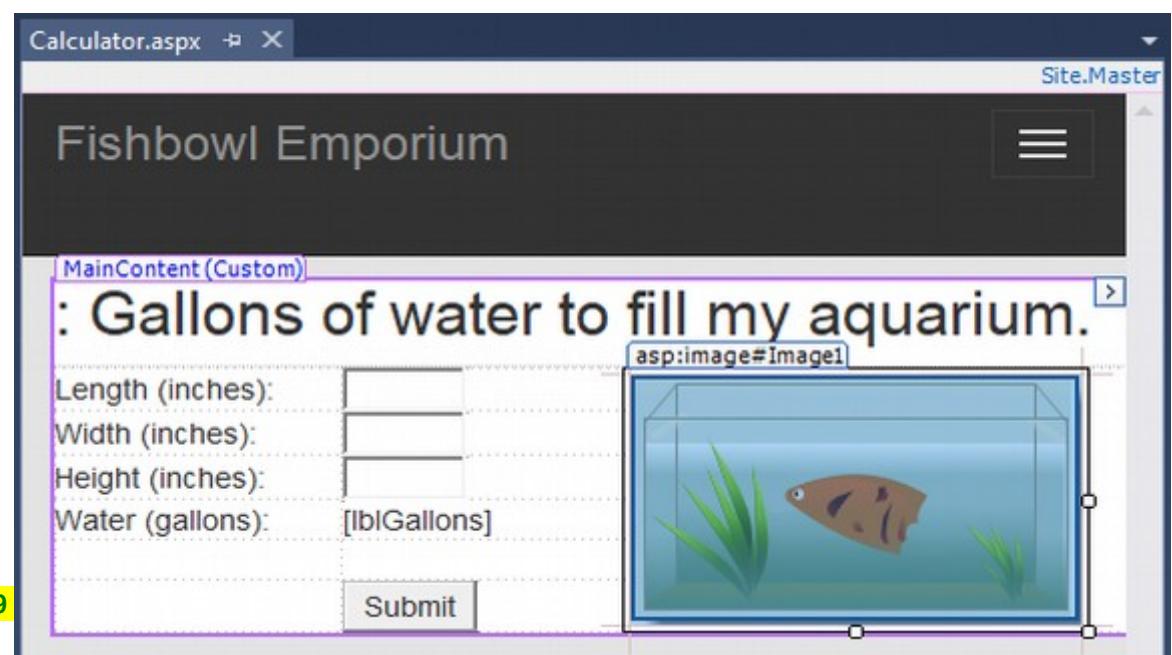


Figure 13-49

step 3 -> save and then start (Ctrl+F5)  
the application to see  
the changes made

Figure 13-50

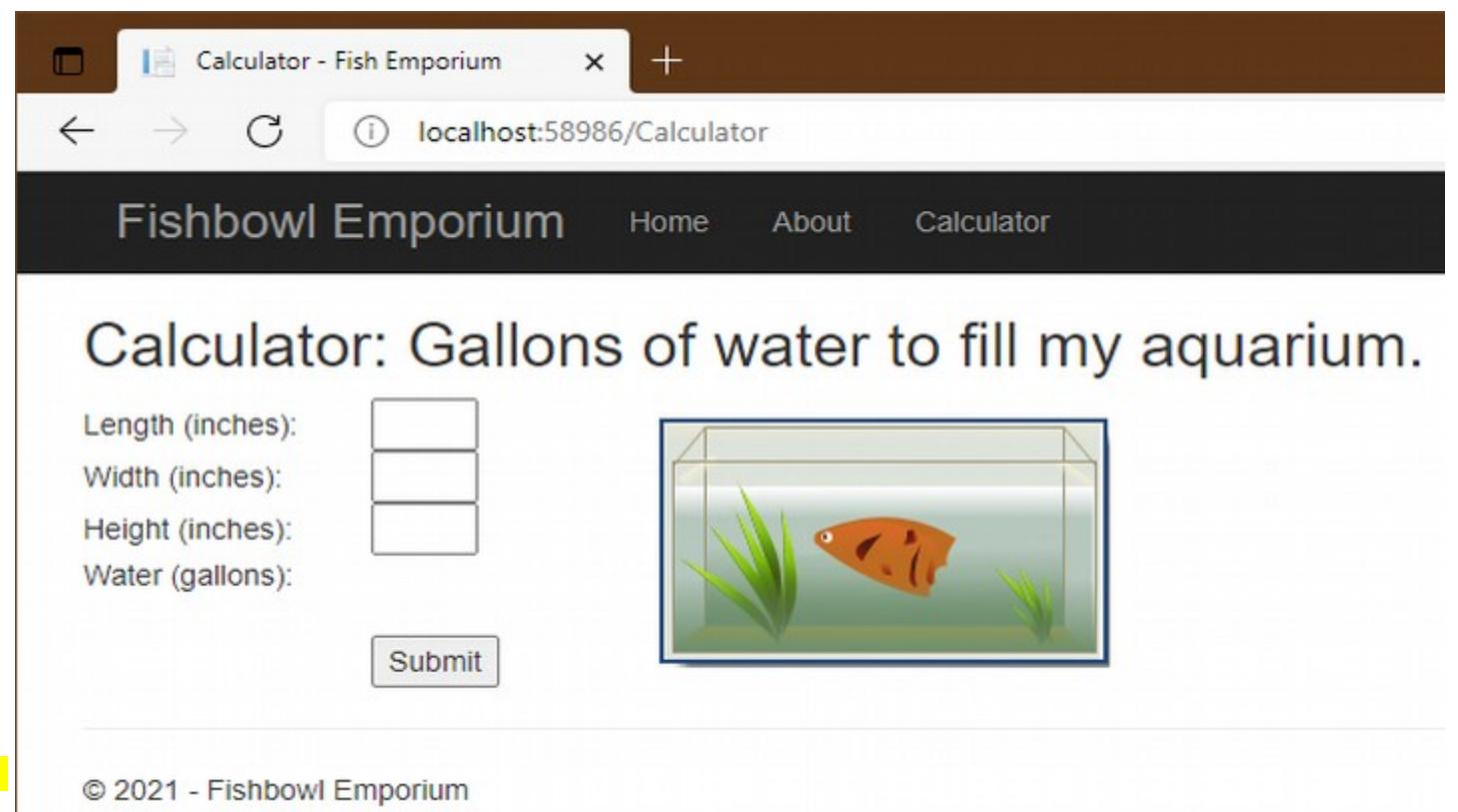


Figure 13-50

### CH13\_A3 - create 3rd page - Dynamic Web page **Calculator.aspx.vb**: 3/4 code a Button Control, e.g. with 01.Fishbowl 8/9

- when the user clicks a button on a Web page, a **postback** occurs and the button's Click event procedure is automatically sent to the server for processing
- in the following set of steps, you will code the **btnSubmit\_Click** procedure to calculate and display the number of gallons of water needed to fill an aquarium
- see the procedure's pseudocode:

**btnSubmit\_Click** pseudocode:

1. declare **dblLength**, **dblWidth**, and **dblHeight** to store aquarium's dimension in inches
2. declare **dblVolume** to store the aquarium's volume in cubic inches
3. declare **dblGallons** to store the number of gallons of water
4. store aquarium's **length**, **width**, and **height** in **dblLength**, **dblWidth**, and **dblHeight**
5. calculate the rectangular aquarium's volume in cubic inches and store: **dblVolume = dblLength \* dblWidth \* dblHeight**
6. calculate the number of gallons of water and store: **dblGallons = dblVolume / 231** -> (1 gallon = 231 cubic inches)
7. display the number of gallons of water - **dblGallons** in **lblGallons**

- to code and then test the **btnSubmit\_Click** procedure: e.g. with 01.Fishbowl 8/9

**step 1** - you will open for coding the **Calculator.aspx.vb**

-> recall that the **.vb** extension on a filename indicates the file contains Visual Basic code ->

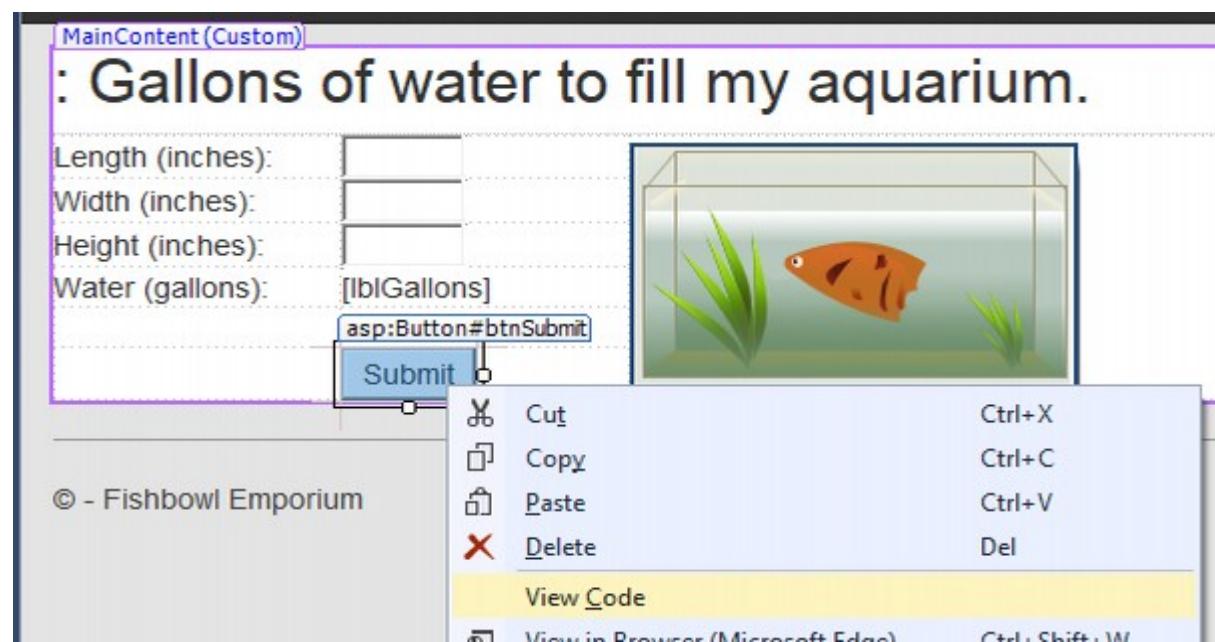
-> in this case, the file is referred to as the **code-behind file** because it contains code that supports the Web page

-> open the window: **Calculator.aspx.vb** by either way:

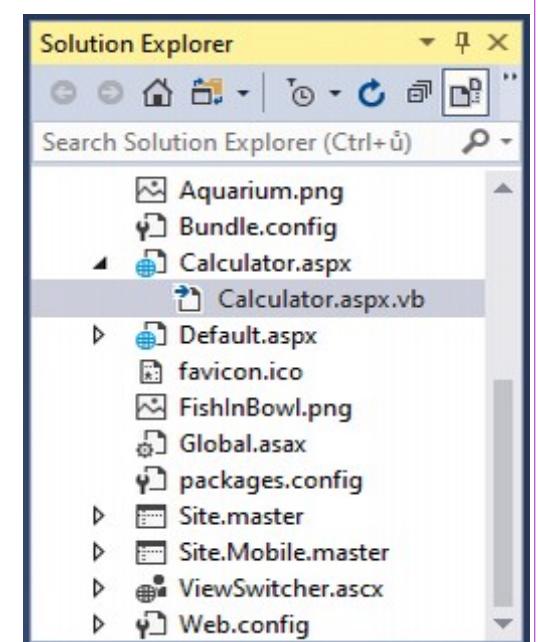
- a). on the **Design** tab, Rclick the **Submit** button and then click: **View Code**
- b). in the **Solution Explorer** window, expand the **Calculator.aspx** and doubleclick the line below: **Calculator.aspx.vb**

**Figure 13-51a**

**Figure 13-51b**



**Figure 13-51a**



**Figure 13-51b**

**step 2.0** <- in my version of VS 2017 v 15.9.36 the page **Calculator.aspx.vb** looks like this:

```
1  
2  Partial Class Contact  
3      Inherits Page  
4  
5  End Class
```

**step 2.1** -> type the comments and **Option** statements as shown below:

```
1  ' Name:      Aquarium  
2  ' Purpose:   Display the number of gallons of water.  
3  ' Programmer: <your name> on <current date>  
4  
5 Option Explicit On  
6 Option Strict On  
7 Option Infer Off  
8
```

**step 2.2** -> open the **btnSubmit\_Click** procedure and type the following comment and press the Enter key twice

-> then, enter the code indicated

```
9  Partial Class Contact  
10     Inherits Page  
11  
12     Private Sub btnSubmit_Click(sender As Object, e As EventArgs) Handles btnSubmit.Click  
13         ' Display the number of gallons of water.  
14  
15         Dim dblLength As Double  
16         Dim dblWidth As Double  
17         Dim dblHeight As Double  
18         Dim dblVolume As Double  
19         Dim dblGallons As Double  
20  
21         Double.TryParse(txtLength.Text, dblLength)  
22         Double.TryParse(txtWidth.Text, dblWidth)  
23         Double.TryParse(txtHeight.Text, dblHeight)  
24  
25         dblVolume = dblLength * dblWidth * dblHeight  
26         dblGallons = dblVolume / 231  
27         lblGallons.Text = dblGallons.ToString("N1")  
28     End Sub  
29 End Class
```

<- the first 3 steps in the pseudocode declare the procedure's variables

<- 4th step in the pseudocode stores the 3 input items in variables

<- 5th step calculates the volume of the rectangular aquarium in cubic inches  
<- 6th step calculates the number of gallons of water needed to fill the rectangular aquarium  
<- the last step displays the number of gallons of water in the **lblGallons** control

**step 3** -> save and then start (Ctrl + F5) and test the application

- wocogo now is that:

- 1). your browser requests the **Calculator.aspx** page from the server
- 2). the server locates the page and then sends the appropriate HTML instructions to your browser for rendering on the screen
- 3). when you hit the **Submit** button, it submits the Web page and data to the server along with a request for additional services  
-< using Web terminology, the information is "posted back" to the server - this event is referred to as a **postback**
- 4). the server processes the code contained in the **btnSubmit\_Click** procedure and then  
sends the appropriate HTML to your browser for rendering on the screen

-> test your Web page:

-> enter the following values: **Length = 20.5**

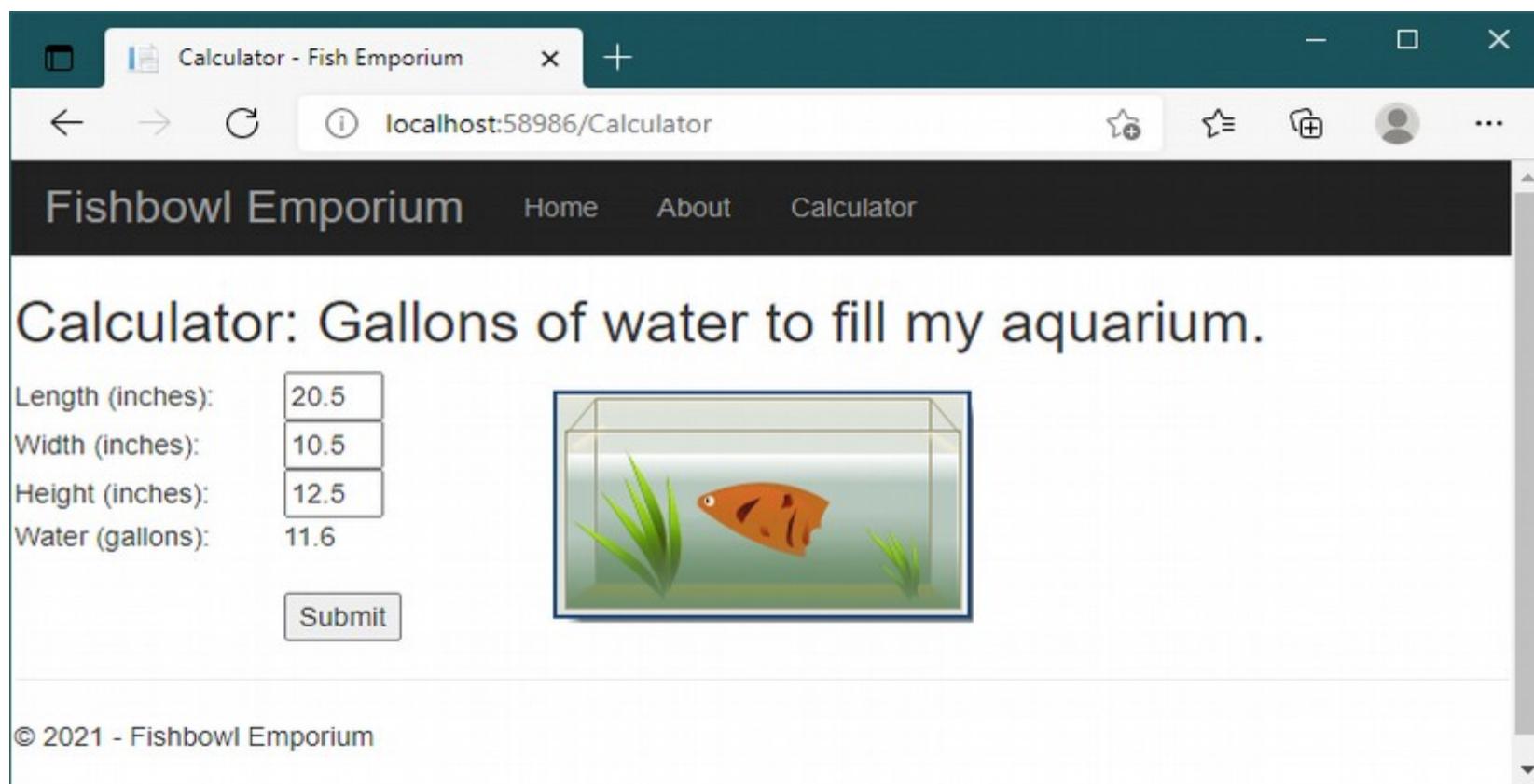
**Width = 10.5**

**Height = 12.5**

-> click the **Submit** button

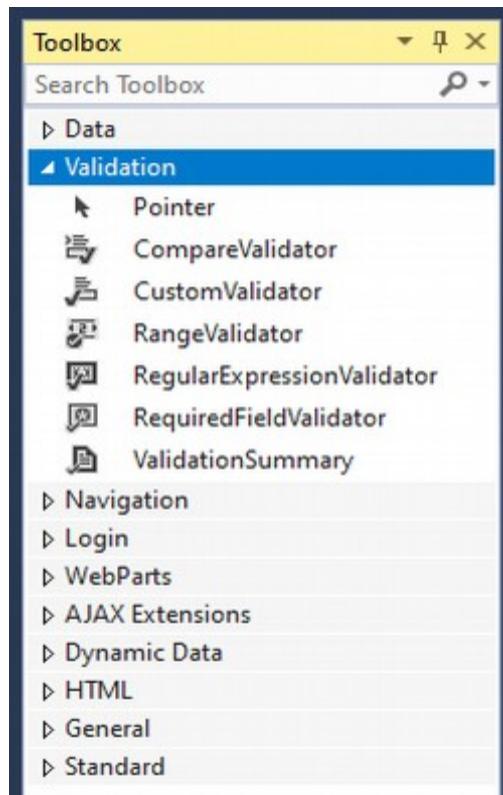
<- the result should be: **11.6** gallons of water, as indicated on: **Figure 13-52**

-> close the browser window and then close the **Calculator.aspx.vb** window



**Figure 13-52**

- the **Validation** section of the **Toolbox** provides several validation tools for **validating user input** **Figure 13-53**
- in the **Fishbowl Emporium** application, you will use **RequiredFieldValidator** controls to verify that the user entered the 3 input items



| Name                       | Purpose                                                                         | Important Properties                                                                        |
|----------------------------|---------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------|
| CompareValidator           | compare an entry with a constant value or with a control's property             | ControlToCompare<br>ControlToValidate<br>ErrorMessage<br>Operator<br>Type<br>ValueToCompare |
| CustomValidator            | verify that an entry passes the specified validation logic                      | ClientValidationFunction<br>ControlToValidate<br>ErrorMessage                               |
| RangeValidator             | verify that an entry is within the specified minimum and maximum values         | ControlToValidate<br>ErrorMessage<br>MaximumValue<br>MinimumValue<br>Type                   |
| RegularExpressionValidator | verify that an entry matches a specific pattern                                 | ControlToValidate<br>ErrorMessage<br>ValidationExpression                                   |
| RequiredFieldValidator     | verify that a control contains data                                             | ControlToValidate<br>ErrorMessage                                                           |
| ValidationSummary          | display all of the validation error messages in a single location on a Web page | DisplayMode<br>HeaderText                                                                   |

**Figure 13-53**

- to verify that the user entered the 3 input items using validation control: **RequiredFieldValidator** e.g. with **01.Fishbowl 9/9**

-> open the: **Solution Explorer** window / **Calculator.aspx** page and click the **Design** tab **Figure 13-54**

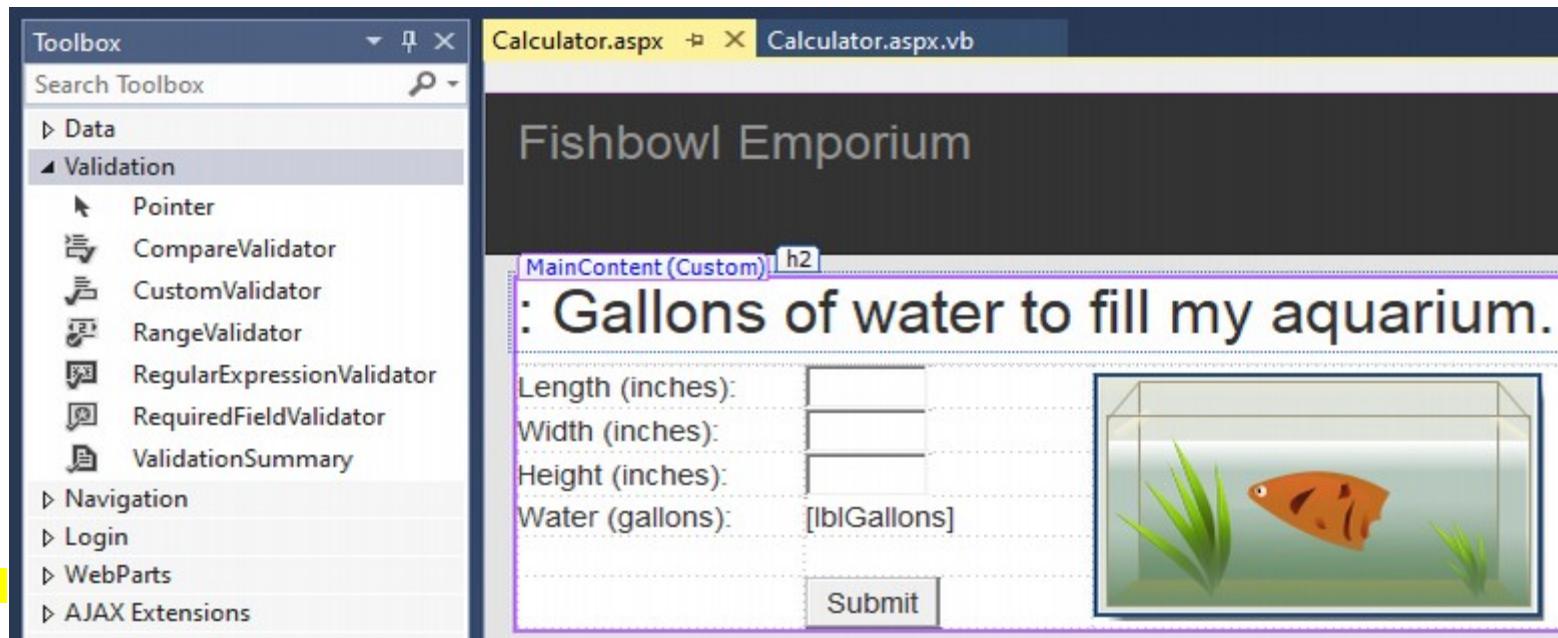


Figure 13-54

**step 1** - first, you will need to make the 2nd column wider that is now, to fit the validation control's text what you will set in the property: **ErrorMessage**

-> position your mouse pointer on the line that divides the 2nd and 3rd column in the table and drag the line to the right until **Column width = 200px**

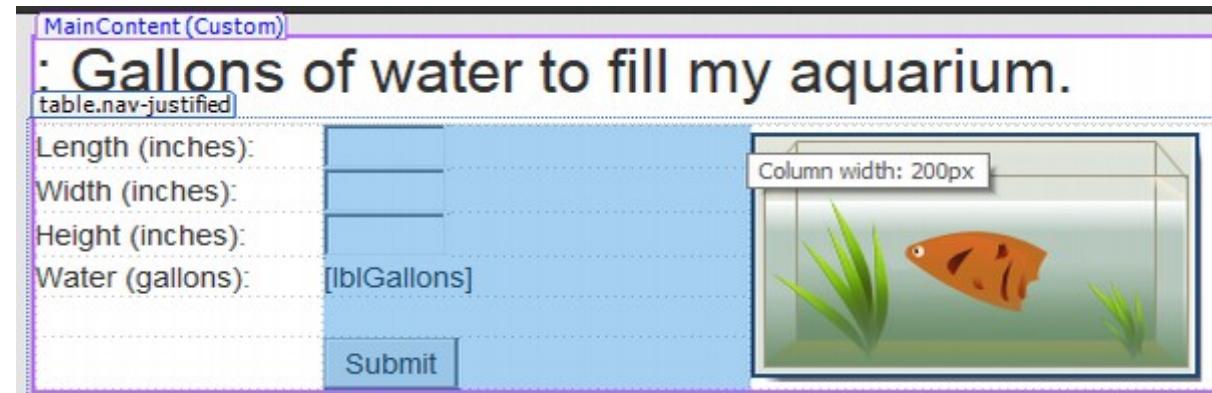


Figure 13-55

**step 2.0** - you will add **3** Validation controls named **RequiredFieldValidator** - one for each input TextBox - and set some of the **Properties**

**step 2.1** -> click to the immediate right of the **txtLength** control and press the **Spacebar** key **5** times **Figure 13-56a**

MainContent (Custom)

: Gallons of water to fill my aquarium.

|                  |                               |
|------------------|-------------------------------|
| Length (inches): | <input type="text" value=""/> |
| Width (inches):  | <input type="text" value=""/> |
| Height (inches): | <input type="text" value=""/> |
| Water (gallons): | [lblGallons]                  |

Submit



Figure 13-56a

**step 2.2** -> from window **Toolbox / Validation** drag the **RequiredFieldValidator** tool to the location of the insertion point

<- a **RequiredFieldValidator** control appears on the Web page

**Figure 13-56b**

Calculator.aspx\* ✎ ✎ Calculator.aspx.vb

Fishbowl Emporium

MainContent (Custom)

: Gallons of water to fill my aquarium.

|                  |                                             |
|------------------|---------------------------------------------|
| Length (inches): | asp:requiredfield...#RequiredFieldValidator |
| Width (inches):  | <input type="text" value=""/>               |
| Height (inches): | <input type="text" value=""/>               |
| Water (gallons): | [lblGallons]                                |

Submit

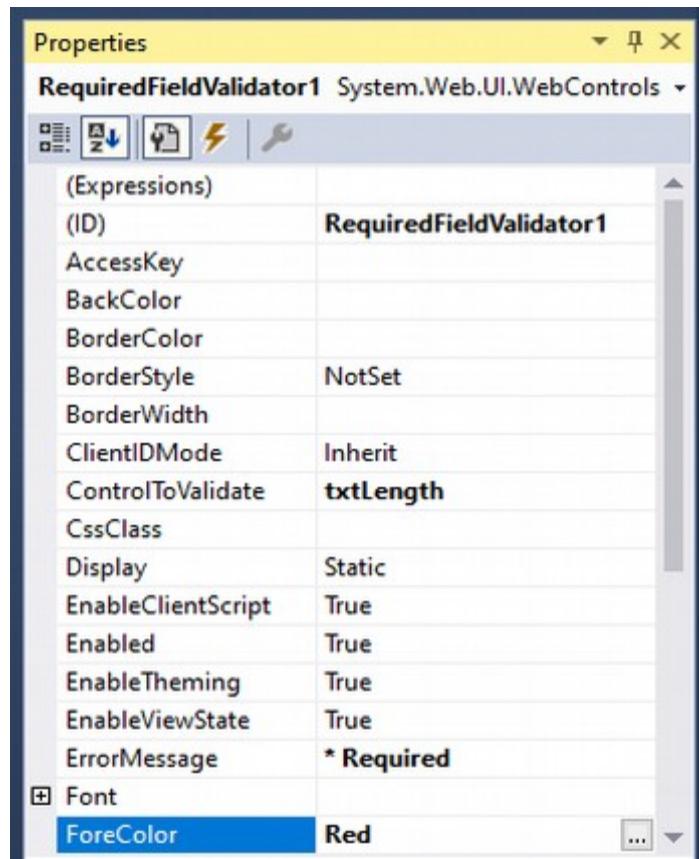


Figure 13-56b

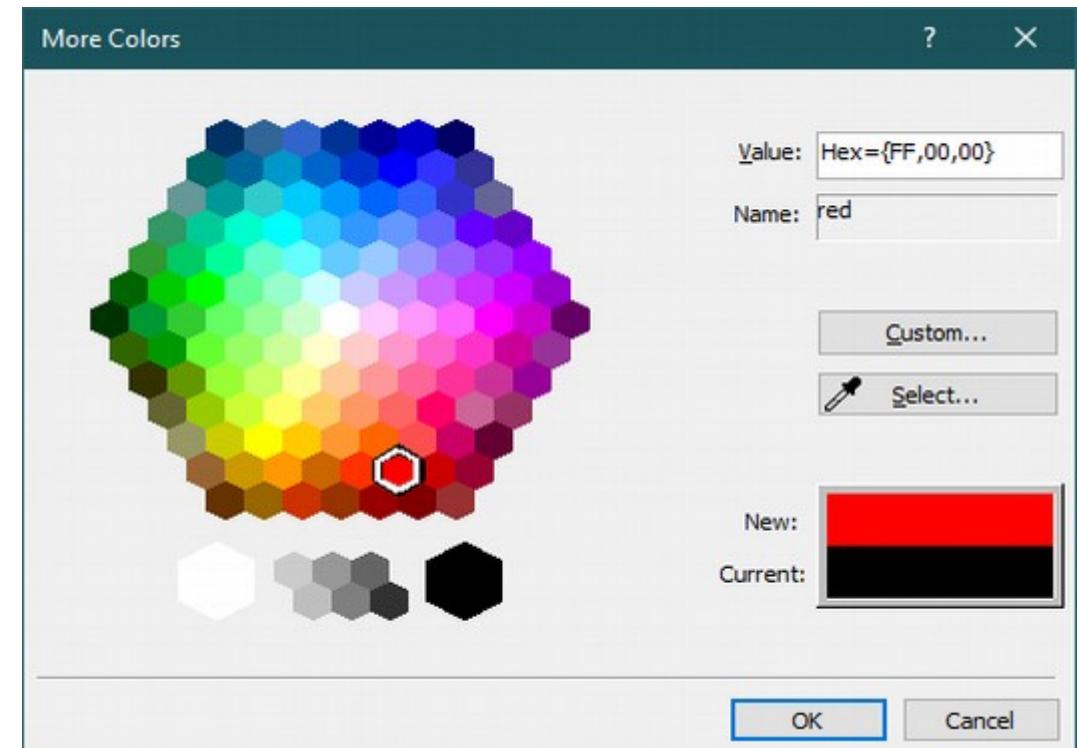
step 2.3 -> set the RequiredFieldValidator1's Properties: **Figure 13-56c**

- ControlToValidate = txtLength
- ErrorMessage = \* Required (notice the asterix and the space character before the word Required)
- ForeColor = Red
  - or
  - ForeColor -> click the ... (ellipsis) button to open the **More Colors** dialog box, click a red hexagon, and then click the **OK** button to close the dialog box

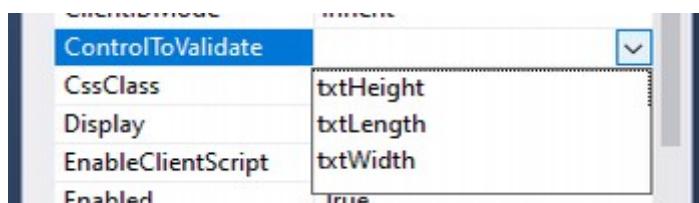
**Figure 13-56d**



**Figure 13-56c**



**Figure 13-56d**



<- list of the available controls

**step 2.4** - you will repeat the **step 2.1** through **step 2.3** for the remaining **txtWidth** and **txtHeight** controls in the 2nd column:

**Figure 13-56e**

MainContent (Custom) h2

## Gallons of water to fill

Length (inches):  \* Required

Width (inches):  \* Required

Height (inches):  \* Required

Water (gallons):  [lblGallons]

**Figure 13-56e**

<- **txtLength** + 5 Spaces + **RequiredFieldValidator1** control + property **ControlToValidate** = **txtLength**

<- **txtWidth** + 5 Spaces + **RequiredFieldValidator2** control + property **ControlToValidate** = **txtWidth**

<- **txtHeight** + 5 Spaces + **RequiredFieldValidator3** control + property **ControlToValidate** = **txtHeight**

common properties:

- **ErrorMessage** = \* Required (notice the asterix and the space character before the word Required)
- **ForeColor** = Red

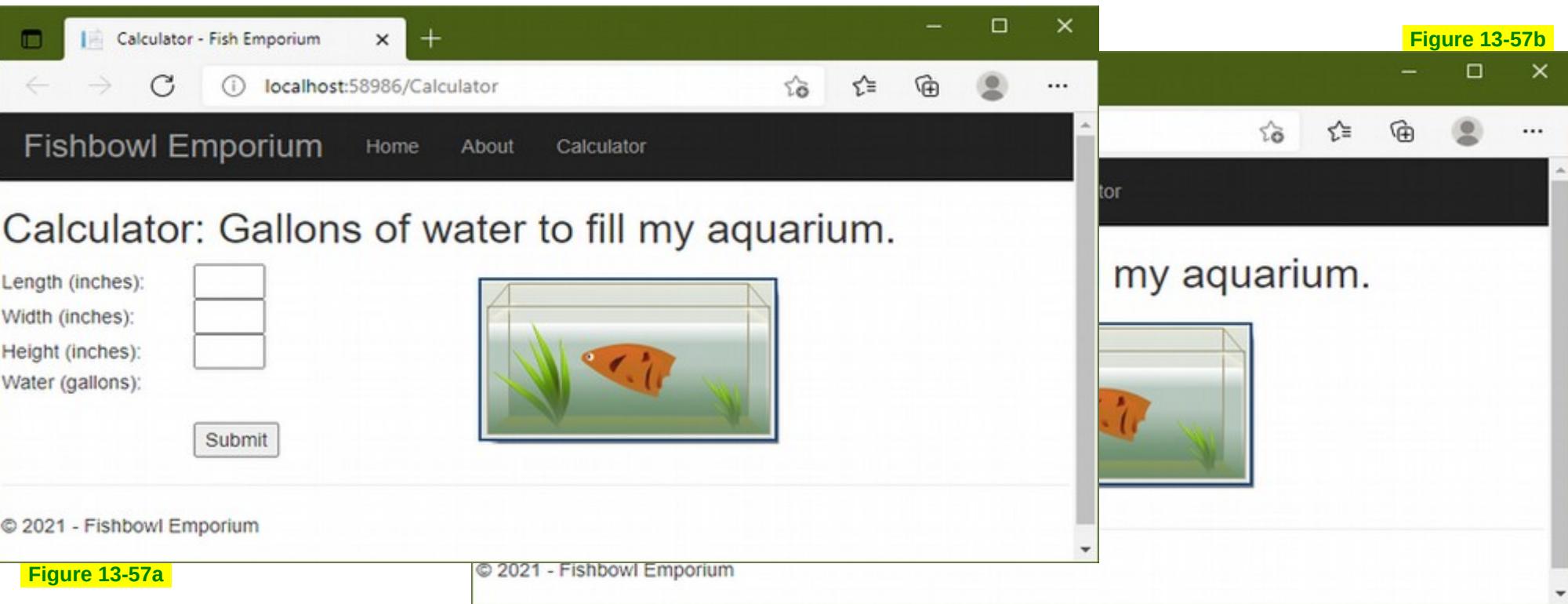
**step 3** -> save, start (Ctrl + F5) and then test the application

-> click the **Submit** button without entering any values

<- notice that each **RequiredFieldValidator** control displays the: \* Required message

**Figure 13-57a**

**Figure 13-57b**



-> in the **Length** box type **16** and then press a **Tab** key to move the insertion point into the next box  
-> notice that the **\* Required** message next to the **Length** box disappears **Figure 13-57c**

-> type **8** in the **Width** box and **10** in the **Height** box  
-> the Web page indicates that **5.5** gallons of water are needed to fill the aquarium **Figure 13-57d**

-> close the browser window, and close the solution

Calculator - Fish Emporium

localhost:58986/Calculator

Fishbowl Emporium Home About Calculator

Calculator: Gallons of water to fill my aquarium.

Length (inches):  \* Required

Width (inches):

Height (inches):  \* Required

Water (gallons):

Submit



© 2021 - Fishbowl Emporium

Figure 13-57c

### Calculator: Gallons of water to fill my aquarium.

Length (inches):   
Width (inches):   
Height (inches):   
Water (gallons):   
Submit



© 2021 - Fishbowl Emporium

Figure 13-57d

## CH13\_X1 - 01.Fishbowl application code & GUI for the pages:

**Site.master & Site.master.vb & Default.aspx & Default.aspx.vb & About.aspx & About.aspx.vb & Calculator.aspx & Calculator.aspx.vb**

- original unused page **Contact.aspx** had been repurposed/recycled to **Calculator.aspx**

### 01.Fishbowl application - Site.master page (template ASP.NET Web Forms Site)

```
1  <%@ Master Language="VB" AutoEventWireup="true" CodeFile="Site.master.vb" Inherits="SiteMaster" %>
2
3  <!DOCTYPE html>
4
5  <html lang="en">
6  <head runat="server">
7      <meta charset="utf-8" />
8      <meta name="viewport" content="width=device-width, initial-scale=1.0" />
9      <title><%: Page.Title %> - Fish Emporium</title>
10
11     <asp:PlaceHolder runat="server">
12         <%: Scripts.Render("~/bundles/modernizr") %>
13     </asp:PlaceHolder>
14     <webopt:bundlereference runat="server" path "~/Content/css" />
15     <link href "~/favicon.ico" rel="shortcut icon" type="image/x-icon" />
16
17 </head>
18 <body>
19     <form runat="server">
20         <asp:ScriptManager runat="server">
21             <Scripts>
22                 <%--To learn more about bundling scripts in ScriptManager see https://go.microsoft.com/fwlink/?LinkID=301884 --%>
23                 <%--Framework Scripts--%>
24                 <asp:ScriptReference Name="MsAjaxBundle" />
25                 <asp:ScriptReference Name="jquery" />
26                 <asp:ScriptReference Name="bootstrap" />
27                 <asp:ScriptReference Name="WebForms.js" Assembly="System.Web" Path="~/Scripts/WebForms/WebForms.js" />
28                 <asp:ScriptReference Name="WebUIValidation.js" Assembly="System.Web" Path="~/Scripts/WebForms/WebUIValidation.js" />
29                 <asp:ScriptReference Name="MenuStandards.js" Assembly="System.Web" Path="~/Scripts/WebForms/MenuStandards.js" />
30                 <asp:ScriptReference Name="GridView.js" Assembly="System.Web" Path="~/Scripts/WebForms/Gridview.js" />
31                 <asp:ScriptReference Name="DetailsView.js" Assembly="System.Web" Path="~/Scripts/WebForms/DetailsView.js" />
32                 <asp:ScriptReference Name="TreeView.js" Assembly="System.Web" Path="~/Scripts/WebForms/TreeView.js" />
33                 <asp:ScriptReference Name="WebParts.js" Assembly="System.Web" Path="~/Scripts/WebForms/WebParts.js" />
34                 <asp:ScriptReference Name="Focus.js" Assembly="System.Web" Path="~/Scripts/WebForms/Focus.js" />
35                 <asp:ScriptReference Name="WebFormsBundle" />
```

```
36         <%--Site Scripts--%>
37     </Scripts>
38 </asp:ScriptManager>
39
40 <div class="navbar navbar-inverse navbar-fixed-top">
41     <div class="container">
42         <div class="navbar-header">
43             <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">
44                 <span class="icon-bar"></span>
45                 <span class="icon-bar"></span>
46                 <span class="icon-bar"></span>
47             </button>
48             <a class="navbar-brand" runat="server" href="~/">Fishbowl Emporium</a>
49         </div>
50         <div class="navbar-collapse collapse">
51             <ul class="nav navbar-nav">
52                 <li><a runat="server" href="~/">Home</a></li>
53                 <li><a runat="server" href="~/About">About</a></li>
54                 <li><a runat="server" href="~/Calculator">Calculator</a></li>
55             </ul>
56             <asp:LoginView runat="server" ViewStateMode="Disabled">
57                 <AnonymousTemplate>
58                     <ul class="nav navbar-nav navbar-right">
59                         <%-->
60                         <li><a runat="server" href="~/Account/Register">Register</a></li>
61                         <li><a runat="server" href="~/Account/Login">Log in</a></li>--%>
62                     </ul>
63                 </AnonymousTemplate>
64                 <LoggedInTemplate>
65                     <ul class="nav navbar-nav navbar-right">
66                         <li><a runat="server" href="~/Account/Manage" title="Manage your account">Hello, <%: Context.User.Identity.GetUserName() %>!</a></li>
67                         <li>
68                             <asp:LoginStatus runat="server" LogoutAction="Redirect" LogoutText="Log off" LogoutPageUrl="~/" OnLoggingOut="Unnamed_LoggingOut" />
69                         </li>
70                     </ul>
71                 </LoggedInTemplate>
72             </asp:LoginView>
73         </div>
74     </div>
75 <div class="container body-content">
76     <asp:ContentPlaceHolder ID="MainContent" runat="server">
77     </asp:ContentPlaceHolder>
78     <hr />
79     <footer>
```

```

80             <p>&copy; <%: DateTime.Now.Year %> - Fishbowl Emporium</p>
81         </footer>
82     </div>
83 </form>
84 </body>
85 </html>
86

```

## 01.Fishbowl application - Site.master.vb page (template ASP.NET Web Forms Site)

```

1  Imports System.Collections.Generic
2  Imports System.Security.Claims
3  Imports System.Security.Principal
4  Imports System.Web
5  Imports System.Web.Security
6  Imports System.Web.UI
7  Imports System.Web.UI.WebControls
8
9  Public Partial Class SiteMaster
10    Inherits MasterPage
11    Private Const AntiXsrfTokenKey As String = "__AntiXsrfToken"
12    Private Const AntiXsrfUserNameKey As String = "__AntiXsrfUserName"
13    Private _antiXsrfTokenValue As String
14
15    Protected Sub Page_Init(sender As Object, e As EventArgs)
16        ' The code below helps to protect against XSS attacks
17        Dim requestCookie = Request.Cookies(AntiXsrfTokenKey)
18        Dim requestCookieGuidValue As Guid
19        If requestCookie IsNot Nothing AndAlso Guid.TryParse(requestCookie.Value, requestCookieGuidValue) Then
20            ' Use the Anti-XSRF token from the cookie
21            _antiXsrfTokenValue = requestCookie.Value
22            Page.ViewStateUserKey = _antiXsrfTokenValue
23        Else
24            ' Generate a new Anti-XSRF token and save to the cookie
25            _antiXsrfTokenValue = Guid.NewGuid().ToString("N")
26            Page.ViewStateUserKey = _antiXsrfTokenValue
27
28            Dim responseCookie = New HttpCookie(AntiXsrfTokenKey) With {
29                .HttpOnly = True,
30                .Value = _antiXsrfTokenValue
31            }
32            If FormsAuthentication.RequireSSL AndAlso Request.IsSecureConnection Then
33                responseCookie.Secure = True
34            End If

```

```
35     Response.Cookies.[Set](responseCookie)
36 End If
37
38     AddHandler Page.PreLoad, AddressOf master_Page_PreLoad
39 End Sub
40
41 Protected Sub master_Page_PreLoad(sender As Object, e As EventArgs)
42     If Not IsPostBack Then
43         ' Set Anti-XSRF token
44         ViewState(AntiXsrfTokenKey) = Page.ViewStateUserKey
45         ViewState(AntiXsrfUserNameKey) = If(Context.User.Identity.Name, [String].Empty)
46     Else
47         ' Validate the Anti-XSRF token
48         If DirectCast(ViewState(AntiXsrfTokenKey), String) <> _antiXsrfTokenValue OrElse DirectCast(ViewState(AntiXsrfUserNameKey),
49             String) <> (If(Context.User.Identity.Name, [String].Empty)) Then
50             Throw New InvalidOperationException("Validation of Anti-XSRF token failed.")
51         End If
52     End If
53 End Sub
54
55 Protected Sub Page_Load(sender As Object, e As EventArgs)
56
57 End Sub
58
59 Protected Sub UnnamedLoggingOut(sender As Object, e As LoginCancelEventArgs)
60     Context.GetOwinContext().Authentication.SignOut()
61 End Sub
62 End Class
63
```

## 01.Fishbowl application - Default.aspx page (template ASP.NET Web Forms Site)

**Figure 13-x1**

```
13     </div>
14
15 </asp:Content>
16
```

### 01.Fishbowl application - Default.aspx.vb page (template ASP.NET Web Forms Site)

```
1
2 Partial Class _Default
3     Inherits Page
4
5 End Class
```

### 01.Fishbowl application - About.aspx page (template ASP.NET Web Forms Site)

Figure 13-x2

```
1 <%@ Page Title="About Us" Language="VB" MasterPageFile="~/Site.Master" AutoEventWireup="true" CodeFile="About.aspx.vb" Inherits="About" %>
2
3 <asp:Content ID="BodyContent" ContentPlaceHolderID="MainContent" runat="server">
4     <h2><%: Title %></h2>
5     <h3>We have been a family owned and operated business since 2005. We are dedicated to helping you fill
6         your aquarium with healthy and beautiful fish. Our fish are available for adoption during our business hours.</h3>
7     <p>&nbsp;</p>
8     <p class="text-center"><strong>Monday - Friday 8am - 8pm</strong></p>
9     <p class="text-center"><strong>Saturday 9am - 5pm</strong></p>
10    <p class="text-center"><strong>Closed Sunday</strong></p>
11 </asp:Content>
12
```

### 01.Fishbowl application - About.aspx.vb page (template ASP.NET Web Forms Site)

```
1
2 Partial Class About
3     Inherits Page
4
5 End Class
```

### 01.Fishbowl application - Calculator.aspx page (template ASP.NET Web Forms Site)

Figure 13-x3

```
1 <%@ Page Title="Calculator" Language="VB" MasterPageFile="~/Site.Master" AutoEventWireup="true" CodeFile="Calculator.aspx.vb" Inherits="Contact" %>
2
3 <asp:Content ID="BodyContent" ContentPlaceHolderID="MainContent" runat="server">
4     <h2><%: Title %>: Gallons of water to fill my aquarium.</h2>
5     <table class="nav-justified">
6         <tr>
```

```
7     <td style="width: 135px">Length (inches):</td>
8     <td style="width: 200px">
9         <asp:TextBox ID="txtLength" runat="server" Width="50px"></asp:TextBox>
10        &nbsp;&nbsp;&nbsp;&nbsp;
11        <asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat="server" ControlToValidate="txtLength"
12            ErrorMessage="* Required" ForeColor="Red"></asp:RequiredFieldValidator>
13    </td>
14    <td rowspan="6">
15        <asp:Image ID="Image1" runat="server" ImageUrl="~/Aquarium.png" />
16    </td>
17 </tr>
18 <tr>
19     <td style="width: 135px">Width (inches):</td>
20     <td style="width: 200px">
21         <asp:TextBox ID="txtWidth" runat="server" Width="50px"></asp:TextBox>
22        &nbsp;&nbsp;&nbsp;&nbsp;
23        <asp:RequiredFieldValidator ID="RequiredFieldValidator2" runat="server" ControlToValidate="txtWidth"
24            ErrorMessage="* Required" ForeColor="Red"></asp:RequiredFieldValidator>
25    </td>
26 </tr>
27 <tr>
28     <td style="width: 135px">Height (inches):</td>
29     <td style="width: 200px">
30         <asp:TextBox ID="txtHeight" runat="server" Width="50px"></asp:TextBox>
31        &nbsp;&nbsp;&nbsp;&nbsp;
32        <asp:RequiredFieldValidator ID="RequiredFieldValidator3" runat="server" ControlToValidate="txtHeight"
33            ErrorMessage="* Required" ForeColor="Red"></asp:RequiredFieldValidator>
34    </td>
35 </tr>
36 <tr>
37     <td style="width: 135px">Water (gallons):</td>
38     <td style="width: 200px">
39         <asp:Label ID="lblGallons" runat="server"></asp:Label>
40    </td>
41 </tr>
42 <tr>
43     <td style="width: 135px">&nbsp;</td>
44     <td style="width: 200px">&nbsp;</td>
45 </tr>
46 <tr>
47     <td style="width: 135px">&nbsp;</td>
48     <td style="width: 200px">
49         <asp:Button ID="btnSubmit" runat="server" Text="Submit" />
50    </td>
```

```
51      </tr>
52    </table>
53  </asp:Content>
```

## 01.Fishbowl application - Calculator.aspx.vb page (template ASP.NET Web Forms Site)

```
1  ' Name:          Aquarium
2  ' Purpose:       Display the number of gallons of water.
3  ' Programmer:    <your name> on <current date>
4
5  Option Explicit On
6  Option Strict On
7  Option Infer Off
8
9  Partial Class Contact
10   Inherits Page
11
12  Private Sub btnSubmit_Click(sender As Object, e As EventArgs) Handles btnSubmit.Click
13    ' Display the number of gallons of water.
14
15    Dim dblLength As Double
16    Dim dblWidth As Double
17    Dim dblHeight As Double
18    Dim dblVolume As Double
19    Dim dblGallons As Double
20
21    Double.TryParse(txtLength.Text, dblLength)
22    Double.TryParse(txtWidth.Text, dblWidth)
23    Double.TryParse(txtHeight.Text, dblHeight)
24
25    dblVolume = dblLength * dblWidth * dblHeight
26    dblGallons = dblVolume / 231
27    lblGallons.Text = dblGallons.ToString("N1")
28  End Sub
29 End Class
```

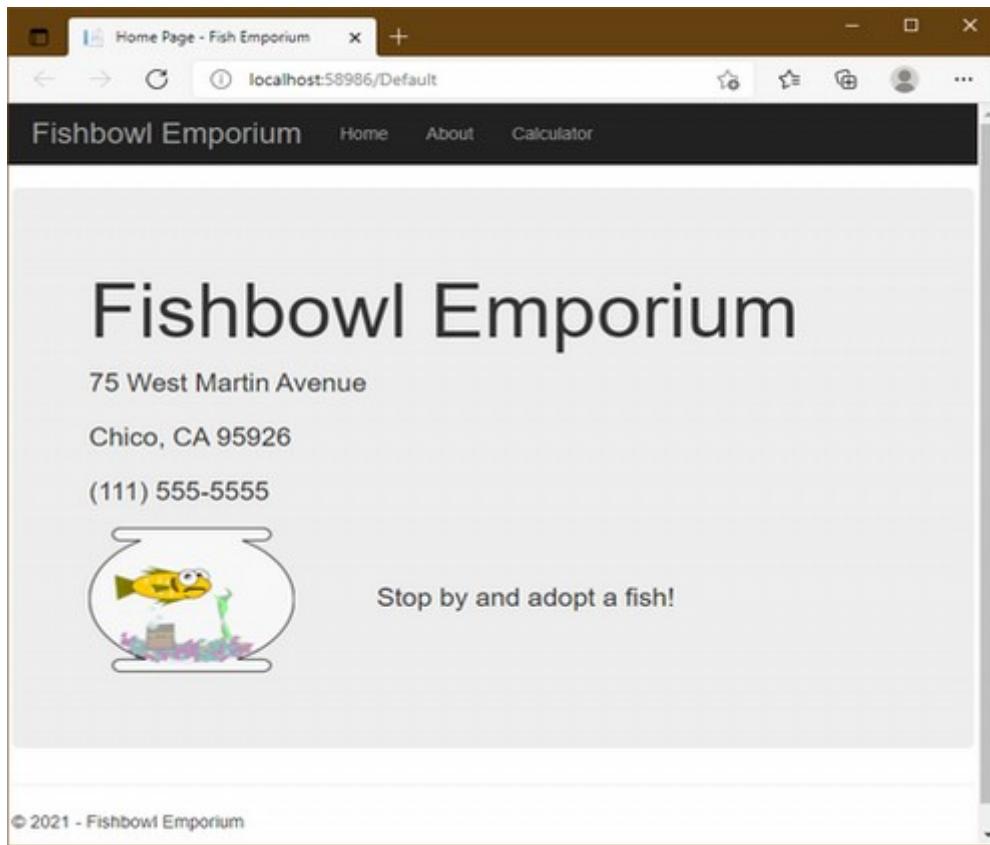


Figure 13-x1 Default.aspx page

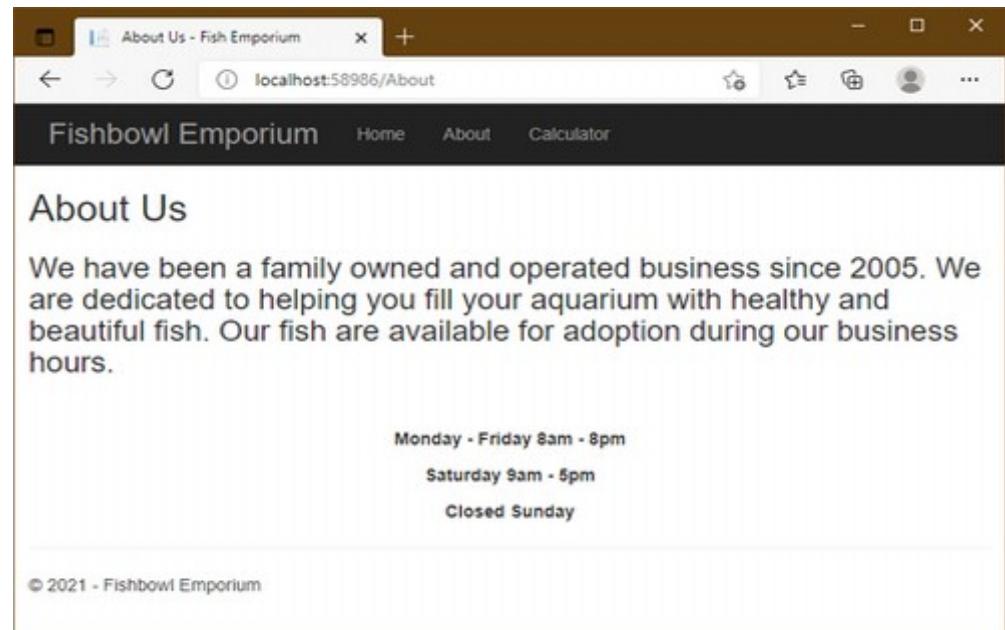


Figure 13-x2 About.aspx page



Figure 13-x3 Calculator.aspx page

**Site.master** - original automatically generated code with template **ASP.NET Web Forms Site**

```
1  <%@ Master Language="VB" AutoEventWireup="true" CodeFile="Site.master.vb" Inherits="SiteMaster" %>
2
3  <!DOCTYPE html>
4
5  <html lang="en">
6  <head runat="server">
7      <meta charset="utf-8" />
8      <meta name="viewport" content="width=device-width, initial-scale=1.0" />
9      <title><%: Page.Title %> - My ASP.NET Application</title>
10
11     <asp:PlaceHolder runat="server">
12         <%: Scripts.Render("~/bundles/modernizr") %>
13     </asp:PlaceHolder>
14     <webopt:bundlereference runat="server" path "~/Content/css" />
15     <link href "~/favicon.ico" rel="shortcut icon" type="image/x-icon" />
16
17 </head>
18 <body>
19     <form runat="server">
20         <asp:ScriptManager runat="server">
21             <Scripts>
22                 <%--To learn more about bundling scripts in ScriptManager see https://go.microsoft.com/fwlink/?LinkID=301884 --%>
23                 <%--Framework Scripts--%>
24                 <asp:ScriptReference Name="MsAjaxBundle" />
25                 <asp:ScriptReference Name="jquery" />
26                 <asp:ScriptReference Name="bootstrap" />
27                 <asp:ScriptReference Name="WebForms.js" Assembly="System.Web" Path "~/Scripts/WebForms/WebForms.js" />
28                 <asp:ScriptReference Name="WebUIValidation.js" Assembly="System.Web" Path "~/Scripts/WebForms/WebUIValidation.js" />
29                 <asp:ScriptReference Name="MenuStandards.js" Assembly="System.Web" Path "~/Scripts/WebForms/MenuStandards.js" />
30                 <asp:ScriptReference Name="GridView.js" Assembly="System.Web" Path "~/Scripts/WebForms/Gridview.js" />
31                 <asp:ScriptReference Name="DetailsView.js" Assembly="System.Web" Path "~/Scripts/WebForms/DetailsView.js" />
32                 <asp:ScriptReference Name="TreeView.js" Assembly="System.Web" Path "~/Scripts/WebForms/TreeView.js" />
33                 <asp:ScriptReference Name="WebParts.js" Assembly="System.Web" Path "~/Scripts/WebForms/WebParts.js" />
34                 <asp:ScriptReference Name="Focus.js" Assembly="System.Web" Path "~/Scripts/WebForms/Focus.js" />
35                 <asp:ScriptReference Name="WebFormsBundle" />
36                 <%--Site Scripts--%>
37             </Scripts>
38         </asp:ScriptManager>
39
```

```
40     <div class="navbar navbar-inverse navbar-fixed-top">
41         <div class="container">
42             <div class="navbar-header">
43                 <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">
44                     <span class="icon-bar"></span>
45                     <span class="icon-bar"></span>
46                     <span class="icon-bar"></span>
47                 </button>
48                 <a class="navbar-brand" runat="server" href="/">Application name</a>
49             </div>
50             <div class="navbar-collapse collapse">
51                 <ul class="nav navbar-nav">
52                     <li><a runat="server" href="/">Home</a></li>
53                     <li><a runat="server" href="/About">About</a></li>
54                     <li><a runat="server" href="/Contact">Contact</a></li>
55                 </ul>
56                 <asp:LoginView runat="server" ViewStateMode="Disabled">
57                     <AnonymousTemplate>
58                         <ul class="nav navbar-nav navbar-right">
59                             <li><a runat="server" href="/Account/Register">Register</a></li>
60                             <li><a runat="server" href="/Account/Login">Log in</a></li>
61                         </ul>
62                     </AnonymousTemplate>
63                     <LoggedInTemplate>
64                         <ul class="nav navbar-nav navbar-right">
65                             <li><a runat="server" href="/Account/Manage" title="Manage your account">Hello, <%: Context.User.Identity.GetUserName() %>!</a></li>
66                             <li>
67                                 <asp:LoginStatus runat="server" LogoutAction="Redirect" LogoutText="Log off" LogoutPageUrl="/" OnLoggingOut="Unnamed_LoggingOut" />
68                             </li>
69                         </ul>
70                     </LoggedInTemplate>
71                 </asp:LoginView>
72             </div>
73         </div>
74     </div>
75     <div class="container body-content">
76         <asp:ContentPlaceHolder ID="MainContent" runat="server">
77             </asp:ContentPlaceHolder>
78             <hr />
79             <footer>
80                 <p>&copy; <%: DateTime.Now.Year %> - My ASP.NET Application</p>
81             </footer>
82         </div>
83     </form>
```

```
84    </body>
85  </html>
86
```

## Default.aspx - original automatically generated code with template **ASP.NET Web Forms Site**

```
1  <%@ Page Title="Home Page" Language="VB" MasterPageFile="~/Site.Master" AutoEventWireup="true" CodeFile="Default.aspx.vb" Inherits="_Default" %>
2
3  <asp:Content ID="BodyContent" ContentPlaceHolderID="MainContent" runat="server">
4
5      <div class="jumbotron">
6          <h1>ASP.NET</h1>
7          <p class="lead">ASP.NET is a free web framework for building great Web sites and Web applications using HTML, CSS, and JavaScript.</p>
8          <p><a href="http://www.asp.net" class="btn btn-primary btn-lg">Learn more &raquo;</a></p>
9      </div>
10
11     <div class="row">
12         <div class="col-md-4">
13             <h2>Getting started</h2>
14             <p>
15                 ASP.NET Web Forms lets you build dynamic websites using a familiar drag-and-drop, event-driven model.
16                 A design surface and hundreds of controls and components let you rapidly build sophisticated, powerful UI-driven sites with data access.
17             </p>
18             <p>
19                 <a class="btn btn-default" href="https://go.microsoft.com/fwlink/?LinkId=301948">Learn more &raquo;</a>
20             </p>
21         </div>
22         <div class="col-md-4">
23             <h2>Get more libraries</h2>
24             <p>
25                 NuGet is a free Visual Studio extension that makes it easy to add, remove, and update libraries and tools in Visual Studio projects.
26             </p>
27             <p>
28                 <a class="btn btn-default" href="https://go.microsoft.com/fwlink/?LinkId=301949">Learn more &raquo;</a>
29             </p>
30         </div>
31         <div class="col-md-4">
32             <h2>Web Hosting</h2>
33             <p>
34                 You can easily find a web hosting company that offers the right mix of features and price for your applications.
35             </p>
36             <p>
37                 <a class="btn btn-default" href="https://go.microsoft.com/fwlink/?LinkId=301950">Learn more &raquo;</a>
38             </p>
```

```
39      </div>
40  </div>
41
42  </asp:Content>
43
```

## About.aspx - original automatically generated code with template ASP.NET Web Forms Site

```
1  <%@ Page Title="About" Language="VB" MasterPageFile="~/Site.Master" AutoEventWireup="true" CodeFile="About.aspx.vb" Inherits="About" %>
2
3  <asp:Content ID="BodyContent" ContentPlaceHolderID="MainContent" runat="server">
4      <h2><%: Title %>.</h2>
5      <h3>Your application description page.</h3>
6      <p>Use this area to provide additional information.</p>
7  </asp:Content>
8
```

## Contact.aspx - original automatically generated code with template ASP.NET Web Forms Site

```
1  <%@ Page Title="Contact" Language="VB" MasterPageFile="~/Site.Master" AutoEventWireup="true" CodeFile="Calculator.aspx.vb" Inherits="Contact" %>
2
3  <asp:Content ID="BodyContent" ContentPlaceHolderID="MainContent" runat="server">
4      <h2><%: Title %>.</h2>
5      <h3>Your contact page.</h3>
6      <address>
7          One Microsoft Way<br />
8          Redmond, WA 98052-6399<br />
9          <abbr title="Phone">P:</abbr>
10         425.555.0100
11     </address>
12
13     <address>
14         <strong>Support:</strong> <a href="mailto:Support@example.com">Support@example.com</a><br />
15         <strong>Marketing:</strong> <a href="mailto:Marketing@example.com">Marketing@example.com</a>
16     </address>
17  </asp:Content>
18
```

## Contact.aspx.vb - original automatically generated code with template ASP.NET Web Forms Site

```
1
2  Partial Class Contact
3      Inherits Page
4
5  End Class
```

## CH13\_Summary

- 01a. the Web is one of the most popular features of the Internet; it consists of Web pages that are stored on Web servers
- 01b. a client computer's browser requests information from a Web server; the information returned by the Web server is then displayed in the browser
- 01c. a Web page's HTML is interpreted and executed by a client computer; a Web page's code is executed by a Web server
- 02a. to create a Web Site application in VS 2017 v 15.9.36 click:

-> menu: **File / New Project... Ctrl+N / Installed / Visual Basic / Web / Previous Versions / ASP.NET Web Forms Site**

- 02b. following pages are automatically created:

**Figure 13-7 4 of the files included in the ASP.NET Web Forms Site template:**

| <u>filename</u>     | <u>purpose</u>                                                                                                           |
|---------------------|--------------------------------------------------------------------------------------------------------------------------|
| <b>About.aspx</b>   | contains information about the website, such as its history or purpose                                                   |
| <b>Contact.aspx</b> | provides contact information, such as a phone number or e-mail address                                                   |
| <b>Default.aspx</b> | serves as the home page                                                                                                  |
| <b>Site.master</b>  | creates a consistent layout for the pages in your application (for example, the same color, header, footer, and buttons) |

- 03. you can use the tabs at the bottom of a **.aspx** window to view a Web page in:
  - Design view
  - Split view
  - Source view
- 04. to start/view a Web Site application, either press **Ctrl+F5** or click: -> menu: **Debug / Start Without Debugging Ctrl+F5**
- 05. any changes made to an application's **Site.master** page will be reflected in/inherited by all of its Web pages
- 06a. you can change a line of text into a comment by selecting the line and then clicking the button: **Comment out the selected lines. (Ctrl+K, Ctrl+C)**
- 06b. button: **Uncomment the selected lines. (Ctrl+K, Ctrl+U)** removes the comment markers from the selected line
- 07a. before you can display an image on a Web page, you need to add the image file to the application by clicking:  
-> menu: **Website / Add Existing Item... Ctrl+D** and locating the folder and image
- 07b. to display an image in an image control, set the control's property: **ImageUrl** to the name of the image file
- 08. you can use the **Formatting toolbar** to format the static text on a Web page
- 09a. you should always view the Web pages in your applications using different browsers
- 09b. you can switch to a different browser by using the: **Standard toolbar** / selecting installed browser in drop down menu for the button **Start**
- 10a. Web controls have an ID property rather than a Name property
- 10b. you use the ID property to refer to the control in code
- 10c. to code a control on a Web page, enter the code in the page's Code Editor window
- 11. the Toolbox provides Validation Tool that you can use to validate the user input on a Web page

## CH13\_Key\_Terms

- **ASP** - stands for: active server page
- **Browser** - a program that allows a client computer to request and view Web pages
- **Client computer** - a computer that requests information from a Web server
- **Dynamic Web page** - an interactive document that can accept information from the user and also retrieve information for the user
- **Postback** - occurs when the information on a Dynamic Web page is sent (posted) back to a server for processing
- **Static Web page** - a noninteractive document whose purpose is merely to display information to the viewer
- **Validation tools** - the tools contained in the Validation section of the Toolbox
  - used to validate user input on a Web page
- **Web pages** - the documents stored on a Web servers
- **Web server** - a computer that contains special software that "serves up" Web pages in response to requests from client computers

**02.Spa\_EXERCISE 1\_introductory**

- in VS 2017 v 15.9.36 you create a Web Application:
  - > menu: **File / New Project...** **Ctrl+N** / Installed / Visual Basic / Web / Previous Versions / **ASP.NET Web Forms Site**
  - 2x Static Web pages
  - repurpose/recycle existing unused page

**03.Market\_EXERCISE 2\_introductory**

- in VS 2017 v 15.9.36 you create a Web Application:
  - > menu: **File / New Project...** **Ctrl+N** / Installed / Visual Basic / Web / Previous Versions / **ASP.NET Web Forms Site**
  - 3x Static Web pages
  - repurpose/recycle existing unused page
  - in Windows copy & rename pages: **\*.aspx** & **\*.aspx.vb** & add it to the project via:
    - > on the menu bar click: **Website / Add Existing Item...** **Ctrl+D**
    - <- dont forget to add both of them

**04.Circle\_EXERCISE 3\_intermediate**

- in VS 2017 v 15.9.36 you create a Web Application:
  - > menu: **File / New Project...** **Ctrl+N** / Installed / Visual Basic / Web / Previous Versions / **ASP.NET Web Forms Site**
  - dynamic Web page
  - table

**05.Meyer\_EXERCISE 4\_advanced**

- in VS 2017 v 15.9.36 you create a Web Application:
  - > menu: **File / New Project...** **Ctrl+N** / Installed / Visual Basic / Web / Previous Versions / **ASP.NET Web Forms Site**
  - 1 Static & 1 Dynamic Web page + calculation
  - repurpose/recycle unused page

**06.Market\_EXERCISE 5\_advanced**

- modified **03.Market\_EXERCISE 2\_introductory**
- in Windows copy & rename pages: **\*.aspx** & **\*.aspx.vb** & add it to the project via:
  - > on the menu bar click: **Website / Add Existing Item...** **Ctrl+D**
  - <- dont forget to add both of them

**07.OnYourOwn**

- skipped

**FixIt**

- 26 errors due to different versions, so fuck off MS and the compatibility

**\_Appendix E - Case Projects\_13.Rosette Catering-CH01-13**

- using **.NET Framework 4.8**

## 02.Spa\_EXERCISE 1\_introductory

- in VS 2017 v 15.9.36 you create a Web Application:
  - > menu: **File / New Project... Ctrl+N** / Installed / Visual Basic / Web / Previous Versions / **ASP.NET Web Forms Site**
  - 2x Static Web pages
  - repurpose/recycle existing unused page

In this exercise, you create two static Web pages for Spa Monique. Figures 13-35 and 13-36 show the completed pages displayed in Microsoft Edge. Keep in mind that the year displayed on your pages might be different than the one shown in the figures.

- a. Use the New Web Site option on the File menu to create a Web Site application named Spa. (Be sure to select the Visual Basic ASP.NET Web Forms Site template.) Save the application in the VB2017\Chap13 folder.
- b. Open the Site.master page in Source view. Replace **My ASP.NET Application** in the **<title>** tag with Spa Monique. Also, replace **Application name** in the **<a>** tag with Spa Monique. In addition, replace **My ASP.NET Application** in the page's footer with Spa Monique.
- c. Locate the About **<li>** tag. Replace both occurrences of **About** with **Massages**. In the Solution Explorer window, change the **About.aspx** filename to **Massages.aspx**. Press **Enter** after typing the name.
- d. Click the Site.master tab. Save the application and then (if necessary) close the **Massages.aspx** window.
- e. In the Site.master window, locate the Contact **<li>** tag and change it to a comment. Also, locate the Register and Login **<li>** tags and change both tags to comments.
- f. Click the Design tab. Click immediately before the letter M in the Spa Monique text that appears below the Site.master tab. Change the text's font size to xx-large (36pt). Save the application and then close the Site.master window.
- g. Now, you will customize the Default.aspx page. If necessary, click the Design tab to show the Default.aspx page in Design view. Use the Add Existing Item option on the Website menu to add the **Spa.png** file to the application. The file is contained in the VB2017\Chap13 folder. [Be sure to change the file type box to All Files (\*.\*)].
- h. Select the ASP.NET text that appears in the MainContent (Custom) area of the page. Type **Spa Monique of Glen Springs** but do not press Enter. Use the Alignment button to center the text.

- i. Delete the Learn more button and all of the information (except the footer) below it.
- j. Replace the sentence that appears below the heading with the two sentences shown in Figure 13-35. Center the sentences horizontally on the page. Then, select the sentences and change their font size to x-large (24pt). Also, change their font color to purple. (Select any purple hexagon in the More Colors dialog box.)
- k. Click immediately after the ! (exclamation point) and then press Enter. Add an image control to the page. Use the control's **ImageUrl** property to display the image stored in the **Spa.png** file. Save and then start the application to view the **Default.aspx** page in a browser window. (Be sure to use either **Ctrl+F5** or the **Start Without Debugging** option on the **Debug** menu.) Close the browser window and then close the **Default.aspx** window.
- l. Now, you will customize the **Massages.aspx** page. Double-click **Massages.aspx** in the Solution Explorer window. If necessary, click the **Source** tab. In the first line, change the page title from "About" to "Massages". Delete the . (period) in the **<h2>** tag, and then delete the entire line that contains the **<h3>** tag.
- m. Click the **Design** tab. Select the "Use this area to provide additional information." sentence and then press **Delete**.
- n. Click **Table** on the menu bar and then click **Insert Table**. The table will need two columns and six rows. The first column should be 140px wide.
- o. In the first cell in the first row, type **Types**. Select **Types** and then click the **B** (Bold) and **U** (Underline) buttons on the **Formatting toolbar**.
- p. In the cell below **Types**, type **Swedish** and press **Tab**. Then, type **50 minutes \$100**. Use the information shown in Figure 13-36 to complete the table.
- q. Save and then start the application. Click **Home** and then click **Massages**. Close the browser window and then close the solution.

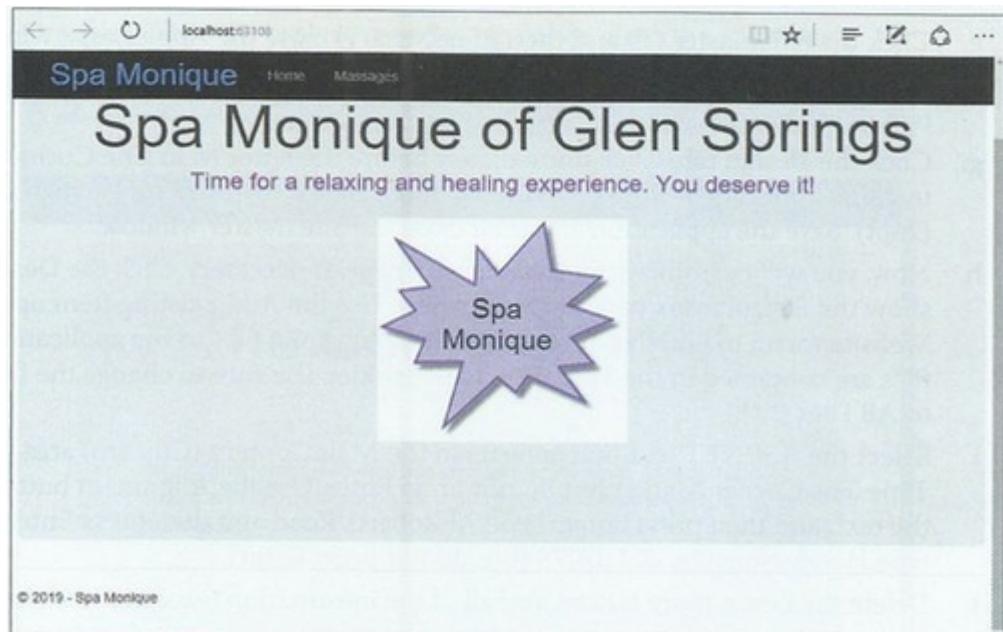


Figure 13-35 Default.aspx page for Spa Monique displayed in Microsoft Edge

### Site.master

```
1  <%@ Master Language="VB" AutoEventWireup="true" CodeFile="Site.master.vb" Inherits="SiteMaster" %>
2
3  <!DOCTYPE html>
4
5  <html lang="en">
6    <head runat="server">
7      <meta charset="utf-8" />
8      <meta name="viewport" content="width=device-width, initial-scale=1.0" />
9      <title><%: Page.Title %> - Spa Monique</title>
10
11    <asp:PlaceHolder runat="server">
12      <%: Scripts.Render("~/bundles/modernizr") %>
13    </asp:PlaceHolder>
14    <webopt:bundlereference runat="server" path "~/Content/css" />
15    <link href "~/favicon.ico" rel="shortcut icon" type="image/x-icon" />
16
17    <style type="text/css">
18      .auto-style1 {
19        float: left;
20        height: 50px;
21        padding: 15px 15px;
```

A screenshot of the Massages.aspx page for Spa Monique. The header is identical to the Default.aspx page. The main content area has a light blue background with the heading "Massages". Below this is a section titled "Types" with a table of massage types and their details. At the bottom is a copyright notice "© 2019 - Spa Monique".

| Type            | Duration   | Cost  |
|-----------------|------------|-------|
| Swedish         | 50 minutes | \$100 |
| Stress reliever | 30 minutes | \$50  |
| Herbal          | 50 minutes | \$115 |
| Stone           | 40 minutes | \$90  |
| Mom-to-be       | 50 minutes | \$100 |

Figure 13-36 Massages.aspx page for Spa Monique

```
22         font-size: xx-large;
23         line-height: 20px;
24     }
25 
```

```
</style>
```

```
26 
```

```
</head>
```

```
28 <body>
```

```
29     <form runat="server">
30         <asp:ScriptManager runat="server">
31             <Scripts>
32                 <%--To learn more about bundling scripts in ScriptManager see https://go.microsoft.com/fwlink/?LinkID=301884 --%>
33                 <%--Framework Scripts--%>
34                 <asp:ScriptReference Name="MsAjaxBundle" />
35                 <asp:ScriptReference Name="jquery" />
36                 <asp:ScriptReference Name="bootstrap" />
37                 <asp:ScriptReference Name="WebForms.js" Assembly="System.Web" Path "~/Scripts/WebForms/WebForms.js" />
38                 <asp:ScriptReference Name="WebUIValidation.js" Assembly="System.Web" Path "~/Scripts/WebForms/WebUIValidation.js" />
39                 <asp:ScriptReference Name="MenuStandards.js" Assembly="System.Web" Path "~/Scripts/WebForms/MenuStandards.js" />
40                 <asp:ScriptReference Name="GridView.js" Assembly="System.Web" Path "~/Scripts/WebForms/Gridview.js" />
41                 <asp:ScriptReference Name="DetailsView.js" Assembly="System.Web" Path "~/Scripts/WebForms/DetailsView.js" />
42                 <asp:ScriptReference Name="TreeView.js" Assembly="System.Web" Path "~/Scripts/WebForms/TreeView.js" />
43                 <asp:ScriptReference Name="WebParts.js" Assembly="System.Web" Path "~/Scripts/WebForms/WebParts.js" />
44                 <asp:ScriptReference Name="Focus.js" Assembly="System.Web" Path "~/Scripts/WebForms/Focus.js" />
45                 <asp:ScriptReference Name="WebFormsBundle" />
46                 <%--Site Scripts--%>
47             </Scripts>
48         </asp:ScriptManager>
49 
```

```
50     <div class="navbar navbar-inverse navbar-fixed-top">
51         <div class="container">
52             <div class="navbar-header">
53                 <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">
54                     <span class="icon-bar"></span>
55                     <span class="icon-bar"></span>
56                     <span class="icon-bar"></span>
57                 </button>
58                 <a class="auto-style1" runat="server" href="~/>Spa Monique</a>
59             </div>
60             <div class="navbar-collapse collapse">
61                 <ul class="nav navbar-nav">
62                     <li><a runat="server" href="~/>Home</a></li>
63                     <li><a runat="server" href="~/Massages">Massages</a></li>
64             <%--          <li><a runat="server" href="~/Contact">Contact</a></li>--%>
65             </ul>
```

```

66         <asp:LoginView runat="server" ViewStateMode="Disabled">
67             <AnonymousTemplate>
68                 <ul class="nav navbar-nav navbar-right">
69                     <%-->
70                         <li><a runat="server" href "~/Account/Register">Register</a></li>
71                         <li><a runat="server" href "~/Account/Login">Log in</a></li>--%>
72                     </ul>
73             </AnonymousTemplate>
74             <LoggedInTemplate>
75                 <ul class="nav navbar-nav navbar-right">
76                     <li><a runat="server" href "~/Account/Manage" title="Manage your account">Hello, <%: Context.User.Identity.GetUserName() %>!</a></li>
77                     <li>
78                         <asp:LoginStatus runat="server" LogoutAction="Redirect" LogoutText="Log off" LogoutPageUrl="~/" OnLoggingOut="Unnamed_LoggingOut" />
79                     </li>
80                 </ul>
81             </LoggedInTemplate>
82         </asp:LoginView>
83     </div>
84 </div>
85 <div class="container body-content">
86     <asp:ContentPlaceHolder ID="MainContent" runat="server">
87     </asp:ContentPlaceHolder>
88     <hr />
89     <footer>
90         <p>&copy; <%: DateTime.Now.Year %> - Spa Monique</p>
91     </footer>
92 </div>
93 </form>
94 </body>
95 </html>

```

## Default.aspx

```

1  <%@ Page Title="Home Page" Language="VB" MasterPageFile="~/Site.Master" AutoEventWireup="true" CodeFile="Default.aspx.vb" Inherits="_Default" %>
2
3  <asp:Content ID="BodyContent" ContentPlaceHolderID="MainContent" runat="server">
4
5      <div class="jumbotron">
6          <h1 class="text-center">Spa Monique of Glen Springs</h1>
7          <p class="lead" style="text-align: center; font-size: x-large; color: #6666FF">Time for a relaxing and healing experience. You deserve it!</p>
8          <p class="lead" style="text-align: center; font-size: x-large; color: #6666FF">
9              <asp:Image ID="Image1" runat="server" ImageUrl="~/Spa.png" />
10         </p>
11         <p>&nbsp;</p>

```

```
12      </div>
13
14  </asp:Content>
```

## Massages.aspx - repurposed/recycled unused page About.aspx

```
1  <%@ Page Title="Massages" Language="VB" MasterPageFile="~/Site.Master" AutoEventWireup="true" CodeFile="Massages.aspx.vb" Inherits="About" %>
2
3  <asp:Content ID="BodyContent" ContentPlaceHolderID="MainContent" runat="server">
4      <h2><%: Title %></h2>
5      <table class="nav-justified">
6          <tr>
7              <td style="width: 140px; text-decoration: underline"><strong>Types</strong></td>
8              <td>&nbsp;</td>
9          </tr>
10         <tr>
11             <td style="width: 140px">Swedish</td>
12             <td>50 minutes $100</td>
13         </tr>
14         <tr>
15             <td style="width: 140px">Stress reliever</td>
16             <td>30 minutes $50</td>
17         </tr>
18         <tr>
19             <td style="width: 140px">Herbal</td>
20             <td>50 minutes $115</td>
21         </tr>
22         <tr>
23             <td style="width: 140px">Stone</td>
24             <td>40 minutes $90</td>
25         </tr>
26         <tr>
27             <td style="width: 140px">Mom-to-be</td>
28             <td>50 minutes $100</td>
29         </tr>
30     </table>
31  </asp:Content>
```

### 03.Market\_EXERCISE 2\_introductory

- in VS 2017 v 15.9.36 you create a Web Application:
  - > menu: **File / New Project... Ctrl+N** / Installed / Visual Basic / Web / Previous Versions / **ASP.NET Web Forms Site**
- 3x Static Web pages
- repurpose/recycle existing unused page
- in Windows copy & rename pages: **\*.aspx & \*.aspx.vb** & add it to the project via:
  - > on the menu bar click: **Website / Add Existing Item... Ctrl+D**
  - <- dont forget to add both of them

In this exercise, you create three static Web pages for The Corner Market. Figures 13-37, 13-38, and 13-39 show the completed pages displayed in Microsoft Edge. Keep in mind that the year displayed on your pages might be different than the one shown in the figures.

- a. Use the New Web Site option on the File menu to create a Web Site application named Market. (Be sure to select the Visual Basic ASP.NET Web Forms Site template.) Save the application in the VB2017\Chap13 folder.
- b. Open the Site.master page in Source view. Replace **My ASP.NET Application** in the **<title>** tag with The Corner Market. Also, replace **Application name** in the **<a>** tag with The Corner Market. In addition, replace **My ASP.NET Application** in the page's footer with The Corner Market.
- c. Locate the About and Contact **<li>** tags. Replace both occurrences of **About** with **Apples**, and replace both occurrences of **Contact** with **Oranges**.
- d. In the Solution Explorer window, change the **About.aspx** filename to **Apples.aspx**. Save the application.
- e. Click the Site.master tab and then (if necessary) close the **Apples.aspx** window.
- f. In the Site.master window, locate the Register and Login **<li>** tags. Change both tags to comments.
- g. Click the Design tab. Click immediately before the letter M in The Corner Market text that appears below the Site.master tab. Change the text's font size to x-large (24pt). Save the application and then close the Site.master window.
- h. Now, you will customize the Default.aspx page. If necessary, click the Design tab to show the Default.aspx page in Design view. Use the Add Existing Item option on the Website menu to add the **Apple.png** and **Orange.png** files to the application. The files are contained in the VB2017\Chap13 folder. [Be sure to change the file type box to All Files (\*.\*)].
- i. Select the ASP.NET text that appears in the MainContent (Custom) area of the page. Type The Corner Market but do not press Enter. Use the Alignment button to center the text, and then press Enter. Type 75 Roberts Road and then press Enter. Then, type Hendersonville, TN 37075 (but do not press Enter).

- j. Delete the Learn more button and all of the information (except the footer) below it.
- k. Replace the sentence that appears below the address with the two sentences shown in Figure 13-37. Save and then start the application to view the Default.aspx page in a browser window. Close the browser window and then close the Default.aspx window.
- l. Now, you will customize the Apples.aspx page. Double-click Apples.aspx in the Solution Explorer window. In the first line, change the page title to Apples. Replace **<%: Title %>** in the **<h2>** tag with Varieties of apples in stock today!.
- m. Delete the entire line that contains the **<h3>** tag. Also, delete the entire line that contains the **<p>** tag.
- n. Click the Design tab. Click immediately after the ! (exclamation point). Use the Alignment button to center the text. Then, press Enter.
- o. Use the Font Size box to change the font size to large (18pt). Then, enter the five apple varieties shown in Figure 13-38. (Be sure to also press Enter after typing the last variety.)
- p. Drag an image control to the line below Red Delicious. The control should display the contents of the **Apple.png** file.
- q. Save and then start the application to view the Apples.aspx page in a browser window. Close the browser window and then close the Apples.aspx window.
- r. Use Windows to open the VB2017\Chap13\Market folder. Make a copy of the Apples.aspx file. Change the name of the copied file to Oranges.aspx. Also, make a copy of the Apples.aspx.vb file. Change the name of the copied file to Oranges.aspx.vb.
- s. Click Website on the Visual Basic menu bar and then click Add Existing Item. Change the file type box to All Files (\*.\*). Click Oranges.aspx and then Ctrl+click Oranges.aspx.vb. Click the Add button.
- t. Double-click Oranges.aspx in the Solution Explorer window. In the first line, change the page title to "Oranges". Also, change "**Apples.aspx.vb**" in the first line to "**Oranges.aspx.vb**".
- u. Next, modify the Oranges.aspx page so that it looks like the one shown in Figure 13-39.
- v. Save and then start the application. Click Home, click Apples, and then click Oranges. Close the browser window and then close the solution.

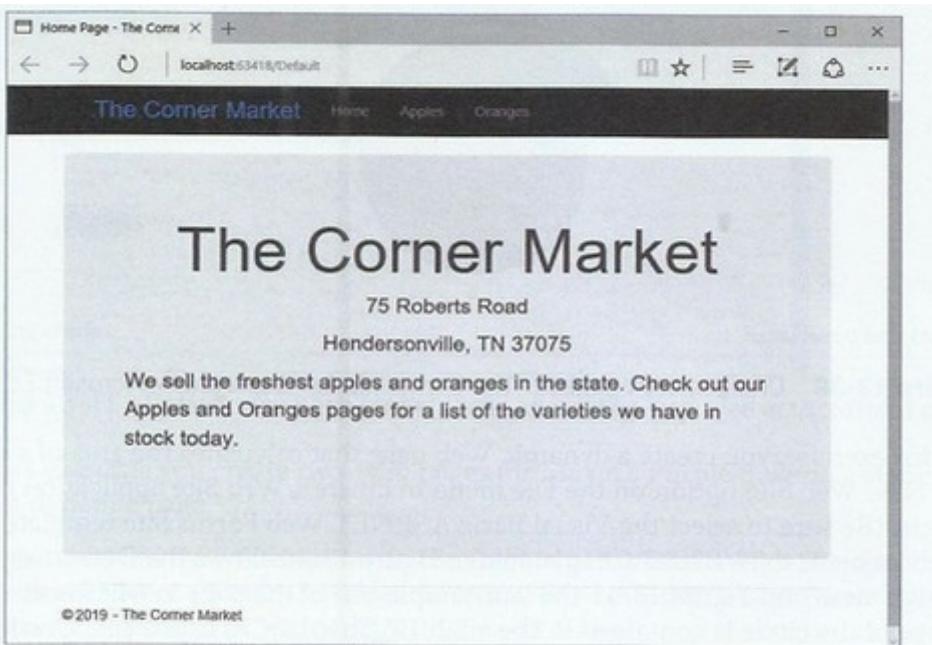


Figure 13-37 Default.aspx page for The Corner Market displayed in Microsoft Edge

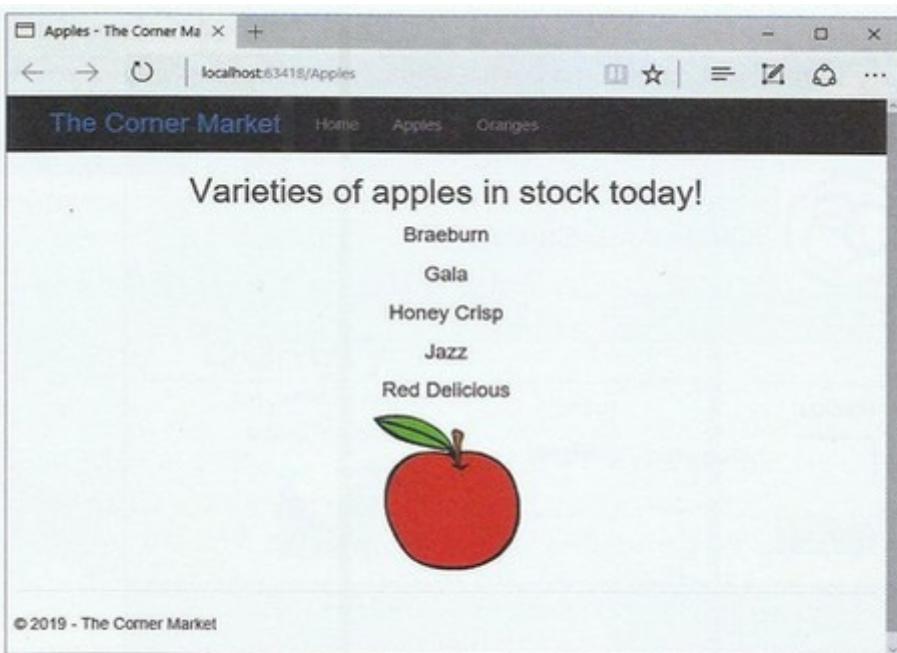


Figure 13-38 Apples.aspx page for The Corner Market displayed in Microsoft Edge

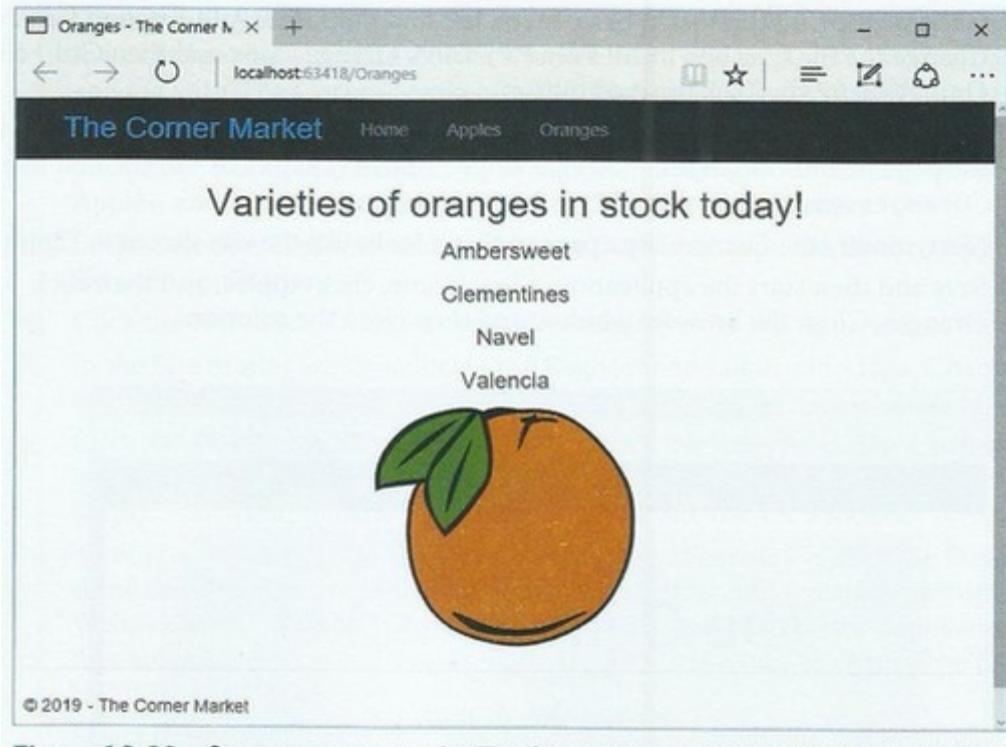


Figure 13-39 Oranges.aspx page for The Corner Market displayed in Microsoft Edge

## Default.aspx

```
1  <%@ Page Title="Home Page" Language="VB" MasterPageFile="~/Site.Master" AutoEventWireup="true" CodeFile="Default.aspx.vb" Inherits="_Default" %>
2
3  <asp:Content ID="BodyContent" ContentPlaceHolderID="MainContent" runat="server">
4
5      <div class="jumbotron">
6          <h1 class="text-center">The Corner Market</h1>
7          <p class="text-center">75 Roberts Road</p>
8          <p class="text-center">Hendersonville, TN 37075</p>
9          <p class="lead">We sell the freshest apples and oranges in the state. _same like any other "company" claiming the same, therefore
10             collective lie_ Check out our Apples and Oranges pages for a list of the varieties we have in stock today.</p>
11      </div>
12
13  </asp:Content>
```

## Apples.aspx - repurposed/recycled About.aspx

```
1  <%@ Page Title="Apples" Language="VB" MasterPageFile="~/Site.Master" AutoEventWireup="true" CodeFile="Apples.aspx.vb" Inherits="About" %>
2
3  <asp:Content ID="BodyContent" ContentPlaceHolderID="MainContent" runat="server">
4      <h2 class="text-center">Varieties of apples in stock today!</h2>
5      <p class="text-center"><span style="font-size: large">Braeburn</span></p>
6      <p class="text-center"><span style="font-size: large">Gala</span></p>
7      <p class="text-center"><span style="font-size: large">Honey Crisp</span></p>
8      <p class="text-center"><span style="font-size: large">Jazz</span></p>
9      <p class="text-center"><span style="font-size: large">Red Delicious</span></p>
10     <p class="text-center">
11         <asp:Image ID="Image1" runat="server" ImageUrl="~/Apple.png" />
12     </p>
13  </asp:Content>
```

## Oranges.aspx - copied and modified Apples.aspx (repurposed/recycled About.aspx)

```
1  <%@ Page Title="Oranges" Language="VB" MasterPageFile="~/Site.Master" AutoEventWireup="true" CodeFile="Oranges.aspx.vb" Inherits="About" %>
2
3  <asp:Content ID="BodyContent" ContentPlaceHolderID="MainContent" runat="server">
4      <h2 class="text-center">Varieties of oranges in stock today!</h2>
5      <p class="text-center" style="font-size: large">Ambersweet</p>
6      <p class="text-center"><span style="font-size: large">Clementines</span></p>
7      <p class="text-center"><span style="font-size: large">Navel</span></p>
8      <p class="text-center"><span style="font-size: large">Valencia</span></p>
9      <p class="text-center">
10         <asp:Image ID="Image1" runat="server" ImageUrl="~/Orange.png" />
11     </p>
12  </asp:Content>
```

Home Page - The Corner Market +

localhost:58713

The Corner Market Home Apples Oranges

# The Corner Market

75 Roberts Road  
Hendersonville, TN 37075

We sell the freshest apples and oranges in the state. \_same like any other "company" claiming the same, therefore collective lie\_ Check out our Apples and Oranges pages for a list of the varieties we have in stock today.

© 2021 - The Corner Market

Apples - The Corner Market +

localhost:58713/Apples

The Corner Market Home Apples Oranges

## Varieties of apples in stock today!

- Braeburn
- Gala
- Honey Crisp
- Jazz
- Red Delicious



© 2021 - The Corner Market

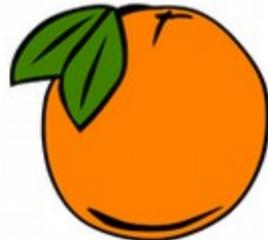
Oranges - The Corner Market +

localhost:58713/Oranges

The Corner Market Home Apples Oranges

## Varieties of oranges in stock today!

- Ambersweet
- Clementines
- Navel
- Valencia



© 2021 - The Corner Market

#### 04.Circle\_EXERCISE 3\_intermediate

- in VS 2017 v 15.9.36 you create a Web Application:  
-> menu: **File / New Project... Ctrl+N** / Installed / Visual Basic / Web / Previous Versions / **ASP.NET Web Forms Site**
- dynamic Web page
- table

3. In this exercise, you create a dynamic Web page that calculates the area of a circle. Use the New Web Site option on the File menu to create a Web Site application named Circle. (Be sure to select the Visual Basic ASP.NET Web Forms Site template.) Save the application in the VB2017\Chap13 folder. Figure 13-40 shows the Default.aspx page in Design view, and Figure 13-41 shows a sample run of the page in Microsoft Edge. The image of the circle is contained in the VB2017\Chap13\Circle.png file. Create the page and then code the Submit button. Use 3.14 as the value for Pi. Display the area with one decimal place. Save and then start and test the application. Close the browser window and then close the solution.

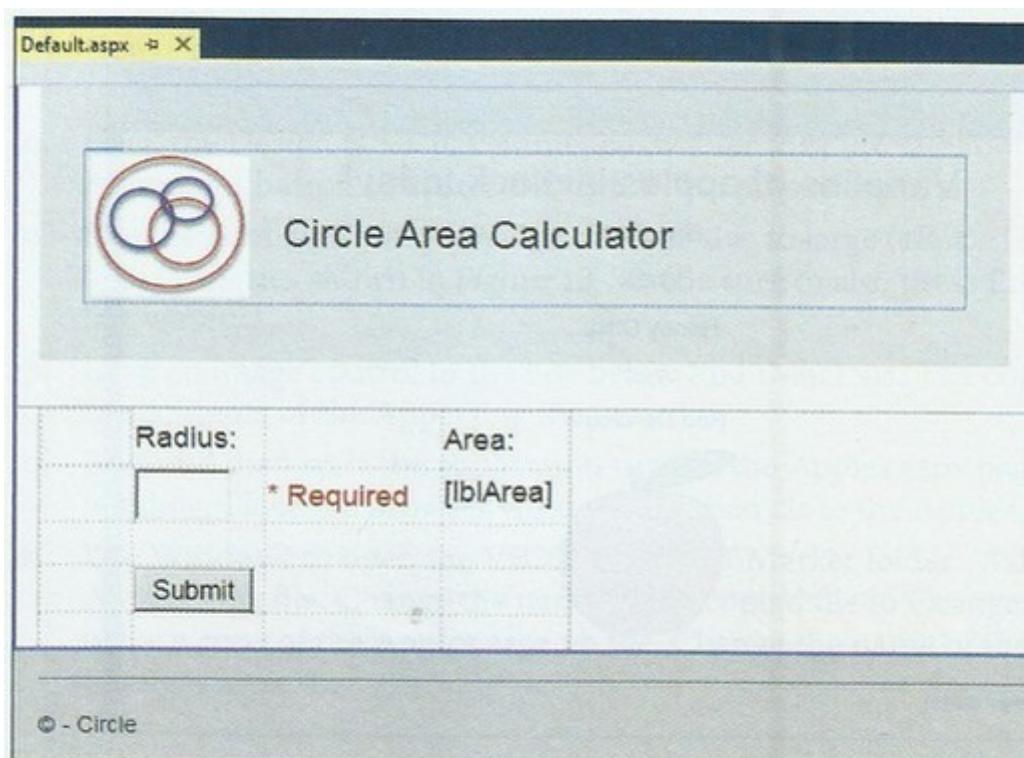


Figure 13-40 Default.aspx page for the Circle application displayed in De

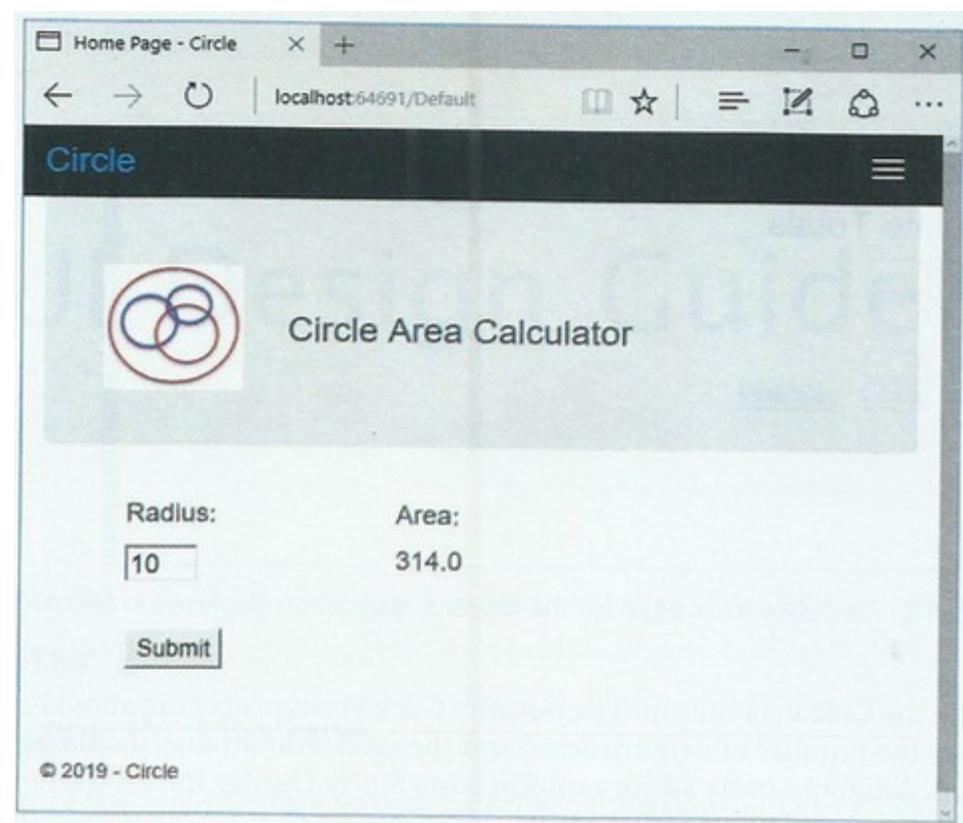


Figure 13-41 Default.aspx page for the Circle application displayed in Mi

# Default.aspx

```
1 <%@ Page Title="Home Page" Language="VB" MasterPageFile="~/Site.Master" AutoEventWireup="true" CodeFile="Default.aspx.vb" Inherits="_Default" %>
2
3 <asp:Content ID="BodyContent" ContentPlaceHolderID="MainContent" runat="server">
4
5     <div class="jumbotron">
6         <h1>
7             <asp:Image ID="Image1" runat="server" ImageUrl "~/Circle.png" />
8             &nbsp;&nbsp; <span style="font-size: xx-large">Circle Area Calculator</span></h1>
9         </div>
10
11     <div class="row">
12         <div class="col-md-4">
13             <table class="nav-justified" style="width: 40%">
14                 <tr style="font-size: large">
15                     <td style="width: 94px">&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp; &nbsp;</td>
16                     <td style="width: 134px">Radius:</td>
17                     <td class="modal-sm" style="width: 280px">&nbsp;</td>
18                     <td class="modal-sm" style="width: 280px">&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;</td>
19                     <td style="width: 196px">Area:</td>
20                 </tr>
21                 <tr style="font-size: large">
22                     <td style="width: 94px">&nbsp;</td>
23                     <td style="width: 134px">
24                         <asp:TextBox ID="txtRadius" runat="server" Width="65px"></asp:TextBox>
25                     </td>
26                     <td class="modal-sm" style="width: 280px">
27                         <asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat="server" ControlToValidate="txtRadius" ErrorMessage="*Required" ForeColor="Red"></asp:RequiredFieldValidator>
28                     </td>
29                     <td class="modal-sm" style="width: 280px">&nbsp;</td>
30                     <td style="width: 196px">
31                         <asp:Label ID="lblArea" runat="server"></asp:Label>
32                     </td>
33                 </tr>
34                 <tr style="font-size: large">
35                     <td style="width: 94px">&nbsp;</td>
36                     <td style="width: 134px">&nbsp;</td>
37                     <td class="modal-sm" style="width: 280px">&nbsp;</td>
38                     <td class="modal-sm" style="width: 280px">&nbsp;</td>
39                     <td style="width: 196px">&nbsp;</td>
40                 </tr>
41                 <tr style="font-size: large">
42                     <td style="width: 94px">&nbsp;</td>
```

```

44         <td style="width: 134px">
45             <asp:Button ID="btnSubmit" runat="server" Text="Submit" />
46         </td>
47         <td class="modal-sm" style="width: 280px">&nbsp;</td>
48         <td class="modal-sm" style="width: 280px">&nbsp;</td>
49         <td style="width: 196px">&nbsp;</td>
50     </tr>
51     <tr style="font-size: large">
52         <td style="width: 94px">&nbsp;</td>
53         <td style="width: 134px">&nbsp;</td>
54         <td class="modal-sm" style="width: 280px">&nbsp;</td>
55         <td class="modal-sm" style="width: 280px">&nbsp;</td>
56         <td style="width: 196px">&nbsp;</td>
57     </tr>
58   </table>
59 </div>
60 </div>
61
62 </asp:Content>

```

## Default.aspx.vb

```

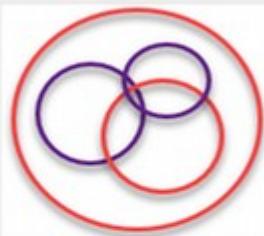
1  ' Circle Area Calculator
2  Option Explicit On
3  Option Strict On
4  Option Infer Off
5
6  Partial Class _Default
7      Inherits Page
8
9  Private Sub btnSubmit_Click(sender As Object, e As EventArgs) Handles btnSubmit.Click
10     ' input = txtRadius; use pi = 3.14; output = lblArea with 1 decimal; formula: A = pi * r^2
11     Dim dblRadius As Double
12     Dim dblArea As Double
13     Double.TryParse(txtRadius.Text, dblRadius)
14
15     dblArea = 3.14 * (dblRadius ^ 2)
16
17     lblArea.Text = dblArea.ToString("N1")
18 End Sub
19 End Class

```

Home Page - Circle

localhost:63815/Default

## Circle



### Circle Area Calculator

Radius:

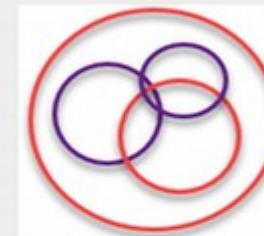
Area: \*Required

© 2021 - Circle

Home Page - Circle

localhost:63815/Default

## Circle



### Circle Area Calculator

Radius:

Area: 314.0

© 2021 - Circle

## 05.Meyer\_EXERCISE 4\_advanced

- in VS 2017 v 15.9.36 you create a Web Application:
  - > menu: **File / New Project... Ctrl+N** / Installed / Visual Basic / Web / Previous Versions / **ASP.NET Web Forms Site**
- 1 Static & 1 Dynamic Web page + calculation
- repurpose/recycle unused page

In this exercise, you create two Web pages for Meyer's Purple Bakery: a static page and a dynamic page.

- a. Use the New Web Site option on the File menu to create a Web Site application named Meyer. (Be sure to select the Visual Basic ASP.NET Web Forms Site template.) Save the application in the VB2017\Chap13 folder.
- b. Figures 13-42 and 13-43 show the Default.aspx and Calculate.aspx pages, respectively, in Microsoft Edge. The image of the chef's hat is contained in the VB2017\Chap13\Chef.png file. Create both pages.

- c. Next, code the Calculate button. The button's Click event procedure should calculate and display the number of items ordered and the total amount due, including a 5% sales tax. A doughnut costs \$0.50; a muffin costs \$0.75. Display the total amount due with a dollar sign and two decimal places.
- d. Save and then start and test the application. Close the browser window and then close the solution.

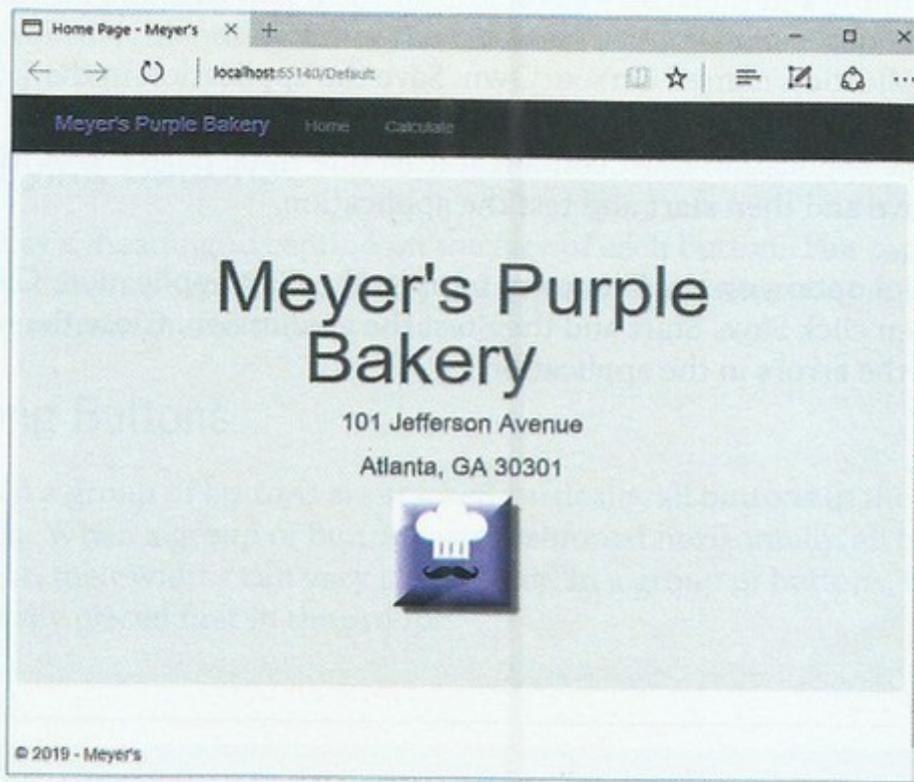


Figure 13-42 Default.aspx page for the Meyer's application displayed in Mi

A screenshot of a web browser showing the Calculate.aspx page for Meyer's Purple Bakery. The title bar says "Calculate - Meyer's". The URL in the address bar is "localhost:65140/Calculate". The page title is "Calculate Totals". It has input fields for "Doughnuts" and "Muffins", and output fields for "Total Items" (showing 0) and "Total due" (\$0.00). There is a "Calculate" button next to the total due field. At the bottom, it says "© 2019 - Meyer's".

Figure 13-43 Calculate.aspx page for the Meyer's application displayed in Mi

### My Test:

1 Doughnut = 0.5 \$	$100 * 0.5 =$	50
1 Muffin = 0.75 \$	$100 * 0.75 =$	75
Total price:		125
+ 5% sales tax	$0.05 * 125 =$	6.25
Total Due:	$125 + 6.25 =$	131.25

## Default.aspx

```
1  <%@ Page Title="Home Page" Language="VB" MasterPageFile="~/Site.Master" AutoEventWireup="true" CodeFile="Default.aspx.vb" Inherits="_Default" %>
2  <asp:Content ID="BodyContent" ContentPlaceHolderID="MainContent" runat="server">
3
4      <div class="jumbotron">
5          <h1 class="text-center">Meyer's Purple Bakery</h1>
6          <p class="text-center">101 Jefferson Avenue</p>
7          <p class="text-center">Atlanta, GA 30301</p>
8          <p class="text-center">
9              <asp:Image ID="Image1" runat="server" ImageUrl="~/Chef.png" />
10         </p>
11     </div>
12
13     <div class="row">
14         <div class="col-md-4">
15             </div>
16     </div>
17
18 </asp:Content>
```

## Calculate.aspx - repurposed/recycled About.aspx

```
1  <%@ Page Title="Calculate" Language="VB" MasterPageFile="~/Site.Master" AutoEventWireup="true" CodeFile="Calculate.aspx.vb" Inherits="About" %>
2
3  <asp:Content ID="BodyContent" ContentPlaceHolderID="MainContent" runat="server">
4      <h2><%: Title %>&nbsp;Totals</h2>
5      <table class="nav-justified">
6          <tr>
7              <td style="width: 54px">&nbsp;</td>
8              <td style="width: 87px">Doughnuts:</td>
9              <td style="width: 37px">
10                 <asp:TextBox ID="txt1Doughnuts" runat="server" Width="63px"></asp:TextBox>
11             </td>
12             <td style="width: 21px">&nbsp;</td>
13             <td>&nbsp;</td>
14         </tr>
15         <tr>
16             <td style="width: 54px">&nbsp;</td>
17             <td style="width: 87px">Muffins:</td>
18             <td style="width: 37px">
19                 <asp:TextBox ID="txt2Muffins" runat="server" Width="63px"></asp:TextBox>
20             </td>
21             <td style="width: 21px">&nbsp;</td>
22             <td>&nbsp;</td>
23         </tr>
```

```

24      <tr>
25          <td style="width: 54px; height: 20px"></td>
26          <td style="width: 87px; height: 20px"></td>
27          <td style="width: 37px; height: 20px"></td>
28          <td style="width: 21px; height: 20px"></td>
29          <td style="height: 20px"></td>
30      </tr>
31      <tr>
32          <td style="width: 54px; height: 20px"></td>
33          <td style="width: 87px; height: 20px">Total items:</td>
34          <td style="width: 37px; height: 20px">
35              <asp:Label ID="lbl1TotalItems" runat="server" BorderStyle="Solid" BorderWidth="2px" Text="0" Width="63px"></asp:Label>
36          </td>
37          <td style="width: 21px; height: 20px"></td>
38          <td style="height: 20px"></td>
39      </tr>
40      <tr>
41          <td style="width: 54px">&ampnbsp</td>
42          <td style="width: 87px">Total due:</td>
43          <td style="width: 37px">
44              <asp:Label ID="lbl2TotalDue" runat="server" BorderStyle="Solid" BorderWidth="2px" Text="$0.00" Width="63px"></asp:Label>
45          </td>
46          <td style="width: 21px">&ampnbsp</td>
47          <td>
48              <asp:Button ID="btnCalculate" runat="server" Text="Calculate" />
49          </td>
50      </tr>
51  </table>
52 </asp:Content>

```

## Calculate.aspx.vb - repurposed/recycled About.aspx.vb

```

1  ' Totals Calculator
2  Option Explicit On
3  Option Strict On
4  Option Infer Off
5
6  Partial Class About
7      Inherits Page
8
9      Private Sub btnCalculate_Click(sender As Object, e As EventArgs) Handles btnCalculate.Click
10         ' input: txt1Doughnuts = $0.50, txt2Muffins = $0.75, ConstSalesTax = 5% sales tax
11         ' lbl1TotalItems, lbl2TotalDue -> $ + 2 decimals
12         Dim int1Doughnuts As Integer : Dim int2Muffins As Integer : Const ConstSalesTax As Double = 0.05
13         Dim dbl2TotalDue As Double

```

```

14
15     Integer.TryParse(txt1Doughnuts.Text, int1Doughnuts)
16     Integer.TryParse(txt2Muffins.Text, int2Muffins)
17
18     ' calculation:
19     dbl12TotalDue = ((int1Doughnuts * 0.5) + (int2Muffins * 0.75)) +
20         ((int1Doughnuts * 0.5 + int2Muffins * 0.75) * ConstSalesTax)
21     ' or :
22     'Dim dblSemi As Double
23     'dblSemi = int1Doughnuts * 0.5 + int2Muffins * 0.75
24     'dbl12TotalDue = dblSemi + (dblSemi * ConstSalesTax)
25
26     lbl1TotalItems.Text = (int1Doughnuts + int2Muffins).ToString
27     lbl12TotalDue.Text = dbl12TotalDue.ToString("C2")
28 End Sub
29 End Class

```

Home Page - Meyer's Purple Bakery

localhost:50727

Meyer's Purple Bakery Home Calculate

# Meyer's Purple Bakery

101 Jefferson Avenue  
Atlanta, GA 30301



© 2021 - Meyer's

Calculate - Meyer's Purple Bakery

localhost:50727/Calculate

Meyer's Purple Bakery Home Calculate

### Calculate Totals

Doughnuts:

Muffins:

Total items:  0

Total due:  \$0.00

© 2021 - Meyer's

Calculate - Meyer's Purple Bakery

localhost:50727/Calculate

Meyer's Purple Bakery Home Calculate

### Calculate Totals

Doughnuts:  100

Muffins:  100

Total items:  200

Total due:  \$131.25

© 2021 - Meyer's

**06.Market\_EXERCISE 5\_advanced****03.Market\_EXERCISE 2\_introductory**

- in Windows copy & rename pages: \*.aspx & \*.aspx.vb & add it to the project via:

-> on the menu bar click: **Website / Add Existing Item...** **Ctrl+D**

<- dont forget to add both of them

5. In this exercise, you modify the application from Exercise 2. Use Windows to make a copy of the Market folder. Rename the copy MarketAdvanced. Use the Open Web Site option on the File menu to open the MarketAdvanced application. The Corner Market now offers four different varieties of peaches: Blushing Star, Crest Haven, Madison, and Summer Pearl. Create a Web page named Peaches.aspx. Include the image stored in the VB2017\Chap13\Peach.png file on the page. Then, make the appropriate modifications to the Site.master and Default.aspx pages. Save and then start and test the application. Close the browser window and then close the solution.

**Peaches.aspx**

```
1  <%@ Page Title="Apples" Language="VB" MasterPageFile="~/Site.Master" AutoEventWireup="true" CodeFile="Peaches.aspx.vb" Inherits="About" %>
2
3  <asp:Content ID="BodyContent" ContentPlaceHolderID="MainContent" runat="server">
4      <h2 class="text-center">Varieties of peaches in stock today!</h2>
5      <p class="text-center"><span style="font-size: large">Blushing Star</span></p>
6      <p class="text-center"><span style="font-size: large">Crest Haven</span></p>
7      <p class="text-center"><span style="font-size: large">Madison</span></p>
8      <p class="text-center"><span style="font-size: large">Summer Pearl</span></p>
9      <p class="text-center">
10         <asp:Image ID="Image1" runat="server" ImageUrl="~/Peach.png" />
11     </p>
12 </asp:Content>
```

Home Page - The Corner Market +  
localhost:58713

The Corner Market Home Apples Oranges Peaches

# The Corner Market

75 Roberts Road  
Hendersonville, TN 37075

We sell the freshest apples, oranges and peaches in the state.  
Same like any other "company" claiming the same, therefore  
collective lie\_ Check out our Apples and Oranges pages for a list of  
the varieties we have in stock today.

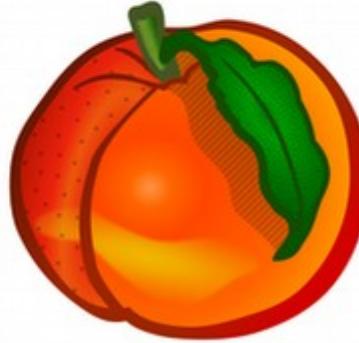
© 2021 - The Corner Market

Apples - The Corner Market +  
localhost:58713/Peaches

The Corner Market Home Apples Oranges Peaches

## Varieties of peaches in stock today!

Blushing Star  
Crest Haven  
Madison  
Summer Pearl



© 2021 - The Corner Market

**FixIt** - 26 errors due to different versions, so fuck off MS and the compatibility

7. Use the Open Web Site option on the File menu to open the FixIt application. Click File, click Save All, and then click Save. Start and then test the application. Close the browser window and then fix the errors in the application.

## Rosette Catering (Chapters 1–13)

Create a Web Site application for Rosette Catering. The interface should allow the user to enter the customer ID, the bride's name, the groom's name, and the date of the wedding reception. It should also allow the user to enter the number of chicken dinners, the number of pasta dinners, and the number of vegetarian dinners ordered for the reception. The interface should display the total number of dinners ordered, the total price of the order without sales tax, the sales tax, and the total price of the order with sales tax. Each dinner costs \$21, and the sales tax rate is 3%. Include an image in the interface. (You can find many different images on the Open Clip Art Library Web site at [openclipart.org](http://openclipart.org).)

1	2	3	4	5	6
1	Customer ID:		input TextBox		* Required
2	Bride's name:		input TextBox		* Required
3	Groom's name:		input TextBox		* Required
4	Date of wedding reception:		input TextBox	(DD/MM/YYYY)	* Required
5					
6	Number of dinners (á \$21.00):				
7	Chicken:		input TextBox		
8	Pasta:		input TextBox		
9	Vegetarian:		input TextBox		
10					
11			btn Calculate		
12					
13	Total dinners ordered:		output Label		
14	Subtotal:		output Label		
15	Sales tax (3%)		output Label		
16	Total price:		output Label		

table: 16 rows, 6 columns

1. Site.master
2. Default.aspx
3. Calculator.aspx
4. Calculator.aspx.vb

## 1. Site.master

```
1  <%@ Master Language="VB" AutoEventWireup="true" CodeFile="Site.master.vb" Inherits="SiteMaster" %>
2
3  <!DOCTYPE html>
4
5  <html lang="en">
6  <head runat="server">
7      <meta charset="utf-8" />
8      <meta name="viewport" content="width=device-width, initial-scale=1.0" />
9      <title><%: Page.Title %> - Rosette Catering</title>
10
11     <asp:PlaceHolder runat="server">
12         <%: Scripts.Render("~/bundles/modernizr") %>
13     </asp:PlaceHolder>
14     <webopt:bundlereference runat="server" path "~/Content/css" />
15     <link href "~/favicon.ico" rel="shortcut icon" type="image/x-icon" />
16
17     <style type="text/css">
18         .auto-style1 {
19             float: left;
20             height: 50px;
21             padding: 15px 15px;
22             font-size: x-large;
23             line-height: 20px;
24         }
25     </style>
26
27 </head>
28 <body>
29     <form runat="server">
30         <asp:ScriptManager runat="server">
31             <Scripts>
32                 <%--To learn more about bundling scripts in ScriptManager see https://go.microsoft.com/fwlink/?LinkID=301884 --%>
33                 <%--Framework Scripts--%>
34                 <asp:ScriptReference Name="MsAjaxBundle" />
35                 <asp:ScriptReference Name="jquery" />
36                 <asp:ScriptReference Name="bootstrap" />
37                 <asp:ScriptReference Name="WebForms.js" Assembly="System.Web" Path "~/Scripts/WebForms/WebForms.js" />
38                 <asp:ScriptReference Name="WebUIValidation.js" Assembly="System.Web" Path "~/Scripts/WebForms/WebUIValidation.js" />
39                 <asp:ScriptReference Name="MenuStandards.js" Assembly="System.Web" Path "~/Scripts/WebForms/MenuStandards.js" />
40                 <asp:ScriptReference Name="GridView.js" Assembly="System.Web" Path "~/Scripts/WebForms/Gridview.js" />
41                 <asp:ScriptReference Name="DetailsView.js" Assembly="System.Web" Path "~/Scripts/WebForms/DetailsView.js" />
```

```
42         <asp:ScriptReference Name="TreeView.js" Assembly="System.Web" Path "~/Scripts/WebForms/TreeView.js" />
43         <asp:ScriptReference Name="WebParts.js" Assembly="System.Web" Path "~/Scripts/WebForms/WebParts.js" />
44         <asp:ScriptReference Name="Focus.js" Assembly="System.Web" Path "~/Scripts/WebForms/Focus.js" />
45         <asp:ScriptReference Name="WebFormsBundle" />
46         <%--Site Scripts--%>
47     </Scripts>
48 </asp:ScriptManager>
49
50 <div class="navbar navbar-inverse navbar-fixed-top">
51     <div class="container">
52         <div class="navbar-header">
53             <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">
54                 <span class="icon-bar"></span>
55                 <span class="icon-bar"></span>
56                 <span class="icon-bar"></span>
57             </button>
58             <a class="auto-style1" runat="server" href("~/")>Rosette Catering</a>
59         </div>
60         <div class="navbar-collapse collapse">
61             <ul class="nav navbar-nav">
62                 <li><a runat="server" href "~/Home">Home</a></li>
63             <%-->
64                 <li><a runat="server" href "~/About">About us</a></li>--%
65                 <li><a runat="server" href "~/Calculator">Order form</a></li>
66             </ul>
67             <asp:LoginView runat="server" ViewStateMode="Disabled">
68                 <AnonymousTemplate>
69                     <ul class="nav navbar-nav navbar-right">
70                         <li><a runat="server" href "~/Account/Register">Register</a></li>
71                         <li><a runat="server" href "~/Account/Login">Log in</a></li>--%
72                     </ul>
73                     </AnonymousTemplate>
74                     <LoggedInTemplate>
75                         <ul class="nav navbar-nav navbar-right">
76                             <li><a runat="server" href "~/Account/Manage" title="Manage your account">Hello, <%: Context.User.Identity.GetUserName() %>!</a></li>
77                             <li>
78                                 <asp:LoginStatus runat="server" LogoutAction="Redirect" LogoutText="Log off" LogoutPageUrl "~/OnLoggingOut=Unnamed_LoggingOut" />
79                             </li>
80                         </ul>
81                     </LoggedInTemplate>
82                 </asp:LoginView>
83             </div>
84         </div>
85     <div class="container body-content">
```

```

86         <asp:ContentPlaceHolder ID="MainContent" runat="server">
87             </asp:ContentPlaceHolder>
88             <hr />
89             <footer>
90                 <p>&copy; <%: DateTime.Now.Year %> - Rosette Catering</p>
91             </footer>
92         </div>
93     </form>
94 </body>
95 </html>

```

## 2. Default.aspx

```

1  <%@ Page Title="Home Page" Language="VB" MasterPageFile="~/Site.Master" AutoEventWireup="true" CodeFile="Default.aspx.vb" Inherits="_Default" %>
2
3  <asp:Content ID="BodyContent" ContentPlaceholderID="MainContent" runat="server">
4
5      <div class="jumbotron">
6          <h1>Rosette Catering</h1>
7          <p>75 West Martin Avenue</p>
8          <p>Chico, CA95926</p>
9          <p>(111) 666-1234</p>
10         <p>rosettecatering@yahoo.com</p>
11         <p>
12             <asp:Image ID="Image1" runat="server" ImageUrl "~/wedding.jpg" />
13             &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp; Your special day catering.</p>
14     </div>
15
16 </asp:Content>

```

## 3. Calculator.aspx

```

1  <%@ Page Title="Order form" Language="VB" MasterPageFile="~/Site.Master" AutoEventWireup="true" CodeFile="Calculator.aspx.vb" Inherits="Contact" %>
2
3  <asp:Content ID="BodyContent" ContentPlaceholderID="MainContent" runat="server">
4      <h2><%: Title %>: plan and calculate your special day catering.</h2>
5      <table class="nav-justified">
6          <tr>
7              <td style="width: 25px">&nbsp;</td>
8              <td class="modal-sm" style="width: 207px">Customer ID:</td>

```

```
9      <td style="width: 17px">&nbsp;</td>
10     <td style="width: 245px">
11       <asp:TextBox ID="txt1ID" runat="server" Width="75px"></asp:TextBox>
12     </td>
13     <td style="width: 15px">&nbsp;</td>
14     <td>
15       <asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat="server" ControlToValidate="txt1ID" ErrorMessage="* Required" ForeColor="Red">
16         </asp:RequiredFieldValidator>
17     </td>
18   </tr>
19   <tr>
20     <td style="width: 25px">&nbsp;</td>
21     <td class="modal-sm" style="width: 207px">Bride's name:</td>
22     <td style="width: 17px">&nbsp;</td>
23     <td colspan="2">
24       <asp:TextBox ID="txt2Bride" runat="server" Width="245px"></asp:TextBox>
25     </td>
26     <td>
27       <asp:RequiredFieldValidator ID="RequiredFieldValidator2" runat="server" ControlToValidate="txt2Bride" ErrorMessage="* Required" ForeColor="Red">
28         </asp:RequiredFieldValidator>
29     </td>
30   </tr>
31   <tr>
32     <td style="width: 25px; height: 20px"></td>
33     <td class="modal-sm" style="height: 20px; width: 207px">Groom's name:</td>
34     <td style="height: 20px; width: 17px"></td>
35     <td style="height: 20px; width: 245px">
36       <asp:TextBox ID="txt3Groom" runat="server" Width="245px"></asp:TextBox>
37     </td>
38     <td style="height: 20px; width: 15px"></td>
39     <td style="height: 20px">
40       <asp:RequiredFieldValidator ID="RequiredFieldValidator3" runat="server" ControlToValidate="txt3Groom" ErrorMessage="* Required" ForeColor="Red">
41         </asp:RequiredFieldValidator>
42     </td>
43   </tr>
44   <tr>
45     <td style="width: 25px; height: 25px"></td>
46     <td class="modal-sm" style="width: 207px; height: 25px">Date of wedding reception:</td>
47     <td style="width: 17px; height: 25px"></td>
48     <td style="width: 245px; height: 25px">
49       <asp:TextBox ID="txt4Date" runat="server" Width="90px"></asp:TextBox>
50       (DD/MM/YYYY)</td>
51     <td style="width: 15px; height: 25px">&nbsp;</td>
52     <td style="height: 25px">
```

```
53         <asp:RequiredFieldValidator ID="RequiredFieldValidator4" runat="server" ControlToValidate="txt4Date" ErrorMessage="* Required" ForeColor="Red">
54             </asp:RequiredFieldValidator>
55         </td>
56     </tr>
57     <tr>
58         <td style="width: 25px">&nbsp;</td>
59         <td class="modal-sm" style="width: 207px">&nbsp;</td>
60         <td style="width: 17px">&nbsp;</td>
61         <td style="width: 245px">&nbsp;</td>
62         <td style="width: 15px">&nbsp;</td>
63         <td>&nbsp;</td>
64     </tr>
65     <tr>
66         <td style="width: 25px">&nbsp;</td>
67         <td class="modal-sm" style="width: 207px">Number of dinners (á $21.00):</td>
68         <td style="width: 17px">&nbsp;</td>
69         <td style="width: 245px">&nbsp;</td>
70         <td style="width: 15px">&nbsp;</td>
71         <td>&nbsp;</td>
72     </tr>
73     <tr>
74         <td style="width: 25px">&nbsp;</td>
75         <td class="modal-sm" style="width: 207px">Chicken:</td>
76         <td style="width: 17px">&nbsp;</td>
77         <td style="width: 245px">
78             <asp:TextBox ID="txt5Chicken" runat="server" Width="50px"></asp:TextBox>
79         </td>
80         <td style="width: 15px">&nbsp;</td>
81         <td>&nbsp;</td>
82     </tr>
83     <tr>
84         <td style="width: 25px">&nbsp;</td>
85         <td class="modal-sm" style="width: 207px">Pasta:</td>
86         <td style="width: 17px">&nbsp;</td>
87         <td style="width: 245px">
88             <asp:TextBox ID="txt6Pasta" runat="server" Width="50px"></asp:TextBox>
89         </td>
90         <td style="width: 15px">&nbsp;</td>
91         <td>&nbsp;</td>
92     </tr>
93     <tr>
```

```
94      <td style="width: 25px">&ampnbsp</td>
95      <td class="modal-sm" style="width: 207px">Vegetarian:</td>
96      <td style="width: 17px">&ampnbsp</td>
97      <td style="width: 245px">
98          <asp:TextBox ID="txt7Vege" runat="server" Width="50px"></asp:TextBox>
99      </td>
100     <td style="width: 15px">&ampnbsp</td>
101     <td>&ampnbsp</td>
102     </tr>
103     <tr>
104         <td style="width: 25px">&ampnbsp</td>
105         <td class="modal-sm" style="width: 207px">&ampnbsp</td>
106         <td style="width: 17px">&ampnbsp</td>
107         <td style="width: 245px">&ampnbsp</td>
108         <td style="width: 15px">&ampnbsp</td>
109         <td>&ampnbsp</td>
110     </tr>
111     <tr>
112         <td style="width: 25px">&ampnbsp</td>
113         <td class="modal-sm" style="width: 207px">&ampnbsp</td>
114         <td style="width: 17px">&ampnbsp</td>
115         <td style="width: 245px">
116             <asp:Button ID="btnSubmit" runat="server" Text="Submit" Width="150px" />
117         </td>
118         <td style="width: 15px">&ampnbsp</td>
119         <td>&ampnbsp</td>
120     </tr>
121     <tr>
122         <td style="width: 25px">&ampnbsp</td>
123         <td class="modal-sm" style="width: 207px">&ampnbsp</td>
124         <td style="width: 17px">&ampnbsp</td>
125         <td style="width: 245px">&ampnbsp</td>
126         <td style="width: 15px">&ampnbsp</td>
127         <td>&ampnbsp</td>
128     </tr>
```

```
129     <tr>
```

```
130      <td style="width: 25px">&ampnbsp</td>
131      <td class="modal-sm" style="width: 207px">Total dinners ordered:</td>
132      <td style="width: 17px">&ampnbsp</td>
133      <td style="width: 245px">
134          <asp:Label ID="lbl1Dinners" runat="server" BorderStyle="None" Width="89px"></asp:Label>
135      </td>
136      <td style="width: 15px">&ampnbsp</td>
137      <td>&ampnbsp</td>
138  </tr>
139  <tr>
140      <td style="width: 25px">&ampnbsp</td>
141      <td class="modal-sm" style="width: 207px">Subtotal:</td>
142      <td style="width: 17px">&ampnbsp</td>
143      <td style="width: 245px">
144          <asp:Label ID="lbl2Subtotal" runat="server" BorderStyle="None" Width="89px"></asp:Label>
145      </td>
146      <td style="width: 15px">&ampnbsp</td>
147      <td>&ampnbsp</td>
148  </tr>
149  <tr>
150      <td style="width: 25px">&ampnbsp</td>
151      <td class="modal-sm" style="width: 207px">Sales tax (3%):</td>
152      <td style="width: 17px">&ampnbsp</td>
153      <td style="width: 245px">
154          <asp:Label ID="lbl3SalesTax" runat="server" BorderStyle="None" Width="89px"></asp:Label>
155      </td>
156      <td style="width: 15px">&ampnbsp</td>
157      <td>&ampnbsp</td>
158  </tr>
159  <tr>
160      <td style="width: 25px">&ampnbsp</td>
161      <td class="modal-sm" style="width: 207px">Total price:</td>
162      <td style="width: 17px">&ampnbsp</td>
163      <td style="width: 245px">
164          <asp:Label ID="lbl4Total" runat="server" BorderStyle="None" Font-Bold="True" Width="89px"></asp:Label>
165      </td>
166      <td style="width: 15px">&ampnbsp</td>
167      <td>&ampnbsp</td>
168  </tr>
169  </table>
170 </asp:Content>
```

#### 4. Calculator.aspx.vb

```
1  ' Name:      Rosette Catering
2  ' Purpose:    Calculate the wedding catering.
3  ' Programmer: Me on just now.
4  Option Explicit On
5  Option Strict On
6  Option Infer Off
7
8  Partial Class Contact
9    Inherits Page
10
11  Private Sub btnSubmit_Click(sender As Object, e As EventArgs) Handles btnSubmit.Click
12    ' txt5Chicken, txt6Pasta, txt7Vege
13    Dim int1Chicken As Integer : Dim int2Pasta As Integer : Dim int3Vege As Integer
14    Dim intDinners As Integer : Dim intSubtotal As Integer : Dim decSales As Decimal
15
16    Integer.TryParse(txt5Chicken.Text, int1Chicken) : Integer.TryParse(txt6Pasta.Text, int2Pasta)
17    Integer.TryParse(txt7Vege.Text, int3Vege)
18
19    intDinners = int1Chicken + int2Pasta + int3Vege
20    intSubtotal = intDinners * 21
21    decSales = intSubtotal * 0.03D
22
23    lbl1Dinners.Text = intDinners.ToString
24    lbl2Subtotal.Text = intSubtotal.ToString("C2")
25    lbl3SalesTax.Text = decSales.ToString("C2")
26    lbl4Total.Text = (intSubtotal + decSales).ToString("C2")
27  End Sub
28 End Class
```

Home Page - Rosette Catering × +

localhost:50101

Rosette Catering Home Order form

# Rosette Catering

75 West Martin Avenue  
Chico, CA95926  
(111) 666-1234  
rosettecatering@yahoo.com



Your special day catering.

© 2021 - Rosette Catering

Order form - Rosette Catering

localhost:50101/Calculator

## Rosette Catering

### Order form: plan and calculate your special day catering.

Customer ID:

Bride's name:

Groom's name:

Date of wedding reception:

Number of dinners (á \$21.00):

Chicken:

Pasta:

Vegetarian:

Total dinners ordered: 100  
Subtotal: \$ 2,100.00  
Sales tax (3%): \$ 63.00  
Total price: \$ 2,163.00

© 2021 - Rosette Catering

## Rosette Catering (Chapters 1–13)

Create a Web Site application for Rosette Catering. The interface should allow the user to enter the customer ID, the bride's name, the groom's name, and the date of the wedding reception. It should also allow the user to enter the number of chicken dinners, the number of pasta dinners, and the number of vegetarian dinners ordered for the reception. The interface should display the total number of dinners ordered, the total price of the order without sales tax, the sales tax, and the total price of the order with sales tax. Each dinner costs \$21, and the sales tax rate is 3%. Include an image in the interface. (You can find many different images on the Open Clip Art Library Web site at [openclipart.org](http://openclipart.org).)

1	2	3	4	5	6
1	Customer ID:		input TextBox		* Required
2	Bride's name:		input TextBox		* Required
3	Groom's name:		input TextBox		* Required
4	Date of wedding reception:		input TextBox	(DD/MM/YYYY)	* Required
5					
6	Number of dinners (á \$21.00):				
7	Chicken:		input TextBox		
8	Pasta:		input TextBox		
9	Vegetarian:		input TextBox		
10					
11			btn Calculate		
12					
13	Total dinners ordered:		output Label		
14	Subtotal:		output Label		
15	Sales tax (3%)		output Label		
16	Total price:		output Label		

table: 16 rows, 6 columns

1. Site.master
2. Default.aspx
3. Calculator.aspx
4. Calculator.aspx.vb

## 1. Site.master

```
1  <%@ Master Language="VB" AutoEventWireup="true" CodeFile="Site.master.vb" Inherits="SiteMaster" %>
2
3  <!DOCTYPE html>
4
5  <html lang="en">
6  <head runat="server">
7      <meta charset="utf-8" />
8      <meta name="viewport" content="width=device-width, initial-scale=1.0" />
9      <title><%: Page.Title %> - Rosette Catering</title>
10
11     <asp:PlaceHolder runat="server">
12         <%: Scripts.Render("~/bundles/modernizr") %>
13     </asp:PlaceHolder>
14     <webopt:bundlereference runat="server" path "~/Content/css" />
15     <link href "~/favicon.ico" rel="shortcut icon" type="image/x-icon" />
16
17     <style type="text/css">
18         .auto-style1 {
19             float: left;
20             height: 50px;
21             padding: 15px 15px;
22             font-size: x-large;
23             line-height: 20px;
24         }
25     </style>
26
27 </head>
28 <body>
29     <form runat="server">
30         <asp:ScriptManager runat="server">
31             <Scripts>
32                 <%--To learn more about bundling scripts in ScriptManager see https://go.microsoft.com/fwlink/?LinkID=301884 --%>
33                 <%--Framework Scripts--%>
34                 <asp:ScriptReference Name="MsAjaxBundle" />
35                 <asp:ScriptReference Name="jquery" />
36                 <asp:ScriptReference Name="bootstrap" />
37                 <asp:ScriptReference Name="WebForms.js" Assembly="System.Web" Path "~/Scripts/WebForms/WebForms.js" />
38                 <asp:ScriptReference Name="WebUIValidation.js" Assembly="System.Web" Path "~/Scripts/WebForms/WebUIValidation.js" />
39                 <asp:ScriptReference Name="MenuStandards.js" Assembly="System.Web" Path "~/Scripts/WebForms/MenuStandards.js" />
40                 <asp:ScriptReference Name="GridView.js" Assembly="System.Web" Path "~/Scripts/WebForms/Gridview.js" />
41                 <asp:ScriptReference Name="DetailsView.js" Assembly="System.Web" Path "~/Scripts/WebForms/DetailsView.js" />
```

```
42         <asp:ScriptReference Name="TreeView.js" Assembly="System.Web" Path "~/Scripts/WebForms/TreeView.js" />
43         <asp:ScriptReference Name="WebParts.js" Assembly="System.Web" Path "~/Scripts/WebForms/WebParts.js" />
44         <asp:ScriptReference Name="Focus.js" Assembly="System.Web" Path "~/Scripts/WebForms/Focus.js" />
45         <asp:ScriptReference Name="WebFormsBundle" />
46         <%--Site Scripts--%>
47     </Scripts>
48 </asp:ScriptManager>
49
50 <div class="navbar navbar-inverse navbar-fixed-top">
51     <div class="container">
52         <div class="navbar-header">
53             <button type="button" class="navbar-toggle" data-toggle="collapse" data-target=".navbar-collapse">
54                 <span class="icon-bar"></span>
55                 <span class="icon-bar"></span>
56                 <span class="icon-bar"></span>
57             </button>
58             <a class="auto-style1" runat="server" href("~/")>Rosette Catering</a>
59         </div>
60         <div class="navbar-collapse collapse">
61             <ul class="nav navbar-nav">
62                 <li><a runat="server" href "~/Home">Home</a></li>
63             <%-->
64                 <li><a runat="server" href "~/About">About us</a></li>--%
65                 <li><a runat="server" href "~/Calculator">Order form</a></li>
66             </ul>
67             <asp:LoginView runat="server" ViewStateMode="Disabled">
68                 <AnonymousTemplate>
69                     <ul class="nav navbar-nav navbar-right">
70                         <li><a runat="server" href "~/Account/Register">Register</a></li>
71                         <li><a runat="server" href "~/Account/Login">Log in</a></li>--%
72                     </ul>
73                     </AnonymousTemplate>
74                     <LoggedInTemplate>
75                         <ul class="nav navbar-nav navbar-right">
76                             <li><a runat="server" href "~/Account/Manage" title="Manage your account">Hello, <%: Context.User.Identity.GetUserName() %>!</a></li>
77                             <li>
78                                 <asp:LoginStatus runat="server" LogoutAction="Redirect" LogoutText="Log off" LogoutPageUrl "~/OnLoggingOut=Unnamed_LoggingOut" />
79                             </li>
80                         </ul>
81                     </LoggedInTemplate>
82                 </asp:LoginView>
83             </div>
84         </div>
85     <div class="container body-content">
```

```

86         <asp:ContentPlaceHolder ID="MainContent" runat="server">
87             </asp:ContentPlaceHolder>
88             <hr />
89             <footer>
90                 <p>&copy; <%: DateTime.Now.Year %> - Rosette Catering</p>
91             </footer>
92         </div>
93     </form>
94 </body>
95 </html>

```

## 2. Default.aspx

```

1  <%@ Page Title="Home Page" Language="VB" MasterPageFile="~/Site.Master" AutoEventWireup="true" CodeFile="Default.aspx.vb" Inherits="_Default" %>
2
3  <asp:Content ID="BodyContent" ContentPlaceholderID="MainContent" runat="server">
4
5      <div class="jumbotron">
6          <h1>Rosette Catering</h1>
7          <p>75 West Martin Avenue</p>
8          <p>Chico, CA95926</p>
9          <p>(111) 666-1234</p>
10         <p>rosettecatering@yahoo.com</p>
11         <p>
12             <asp:Image ID="Image1" runat="server" ImageUrl "~/wedding.jpg" />
13             &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp; Your special day catering.</p>
14     </div>
15
16 </asp:Content>

```

## 3. Calculator.aspx

```

1  <%@ Page Title="Order form" Language="VB" MasterPageFile="~/Site.Master" AutoEventWireup="true" CodeFile="Calculator.aspx.vb" Inherits="Contact" %>
2
3  <asp:Content ID="BodyContent" ContentPlaceholderID="MainContent" runat="server">
4      <h2><%: Title %>: plan and calculate your special day catering.</h2>
5      <table class="nav-justified">
6          <tr>
7              <td style="width: 25px">&nbsp;</td>
8              <td class="modal-sm" style="width: 207px">Customer ID:</td>

```

```
9      <td style="width: 17px">&nbsp;</td>
10     <td style="width: 245px">
11       <asp:TextBox ID="txt1ID" runat="server" Width="75px"></asp:TextBox>
12     </td>
13     <td style="width: 15px">&nbsp;</td>
14     <td>
15       <asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat="server" ControlToValidate="txt1ID" ErrorMessage="* Required" ForeColor="Red">
16         </asp:RequiredFieldValidator>
17     </td>
18   </tr>
19   <tr>
20     <td style="width: 25px">&nbsp;</td>
21     <td class="modal-sm" style="width: 207px">Bride's name:</td>
22     <td style="width: 17px">&nbsp;</td>
23     <td colspan="2">
24       <asp:TextBox ID="txt2Bride" runat="server" Width="245px"></asp:TextBox>
25     </td>
26     <td>
27       <asp:RequiredFieldValidator ID="RequiredFieldValidator2" runat="server" ControlToValidate="txt2Bride" ErrorMessage="* Required" ForeColor="Red">
28         </asp:RequiredFieldValidator>
29     </td>
30   </tr>
31   <tr>
32     <td style="width: 25px; height: 20px"></td>
33     <td class="modal-sm" style="height: 20px; width: 207px">Groom's name:</td>
34     <td style="height: 20px; width: 17px"></td>
35     <td style="height: 20px; width: 245px">
36       <asp:TextBox ID="txt3Groom" runat="server" Width="245px"></asp:TextBox>
37     </td>
38     <td style="height: 20px; width: 15px"></td>
39     <td style="height: 20px">
40       <asp:RequiredFieldValidator ID="RequiredFieldValidator3" runat="server" ControlToValidate="txt3Groom" ErrorMessage="* Required" ForeColor="Red">
41         </asp:RequiredFieldValidator>
42     </td>
43   </tr>
44   <tr>
45     <td style="width: 25px; height: 25px"></td>
46     <td class="modal-sm" style="width: 207px; height: 25px">Date of wedding reception:</td>
47     <td style="width: 17px; height: 25px"></td>
48     <td style="width: 245px; height: 25px">
49       <asp:TextBox ID="txt4Date" runat="server" Width="90px"></asp:TextBox>
50       (DD/MM/YYYY)</td>
51     <td style="width: 15px; height: 25px">&nbsp;</td>
52     <td style="height: 25px">
```

```
53         <asp:RequiredFieldValidator ID="RequiredFieldValidator4" runat="server" ControlToValidate="txt4Date" ErrorMessage="* Required" ForeColor="Red">
54             </asp:RequiredFieldValidator>
55         </td>
56     </tr>
57     <tr>
58         <td style="width: 25px">&nbsp;</td>
59         <td class="modal-sm" style="width: 207px">&nbsp;</td>
60         <td style="width: 17px">&nbsp;</td>
61         <td style="width: 245px">&nbsp;</td>
62         <td style="width: 15px">&nbsp;</td>
63         <td>&nbsp;</td>
64     </tr>
65     <tr>
66         <td style="width: 25px">&nbsp;</td>
67         <td class="modal-sm" style="width: 207px">Number of dinners (á $21.00):</td>
68         <td style="width: 17px">&nbsp;</td>
69         <td style="width: 245px">&nbsp;</td>
70         <td style="width: 15px">&nbsp;</td>
71         <td>&nbsp;</td>
72     </tr>
73     <tr>
74         <td style="width: 25px">&nbsp;</td>
75         <td class="modal-sm" style="width: 207px">Chicken:</td>
76         <td style="width: 17px">&nbsp;</td>
77         <td style="width: 245px">
78             <asp:TextBox ID="txt5Chicken" runat="server" Width="50px"></asp:TextBox>
79         </td>
80         <td style="width: 15px">&nbsp;</td>
81         <td>&nbsp;</td>
82     </tr>
83     <tr>
84         <td style="width: 25px">&nbsp;</td>
85         <td class="modal-sm" style="width: 207px">Pasta:</td>
86         <td style="width: 17px">&nbsp;</td>
87         <td style="width: 245px">
88             <asp:TextBox ID="txt6Pasta" runat="server" Width="50px"></asp:TextBox>
89         </td>
90         <td style="width: 15px">&nbsp;</td>
91         <td>&nbsp;</td>
92     </tr>
```

93 <tr>

```
94      <td style="width: 25px">&ampnbsp</td>
95      <td class="modal-sm" style="width: 207px">Vegetarian:</td>
96      <td style="width: 17px">&ampnbsp</td>
97      <td style="width: 245px">
98          <asp:TextBox ID="txt7Vege" runat="server" Width="50px"></asp:TextBox>
99      </td>
100     <td style="width: 15px">&ampnbsp</td>
101     <td>&ampnbsp</td>
102     </tr>
103     <tr>
104         <td style="width: 25px">&ampnbsp</td>
105         <td class="modal-sm" style="width: 207px">&ampnbsp</td>
106         <td style="width: 17px">&ampnbsp</td>
107         <td style="width: 245px">&ampnbsp</td>
108         <td style="width: 15px">&ampnbsp</td>
109         <td>&ampnbsp</td>
110     </tr>
111     <tr>
112         <td style="width: 25px">&ampnbsp</td>
113         <td class="modal-sm" style="width: 207px">&ampnbsp</td>
114         <td style="width: 17px">&ampnbsp</td>
115         <td style="width: 245px">
116             <asp:Button ID="btnSubmit" runat="server" Text="Submit" Width="150px" />
117         </td>
118         <td style="width: 15px">&ampnbsp</td>
119         <td>&ampnbsp</td>
120     </tr>
121     <tr>
122         <td style="width: 25px">&ampnbsp</td>
123         <td class="modal-sm" style="width: 207px">&ampnbsp</td>
124         <td style="width: 17px">&ampnbsp</td>
125         <td style="width: 245px">&ampnbsp</td>
126         <td style="width: 15px">&ampnbsp</td>
127         <td>&ampnbsp</td>
128     </tr>
```

```
129     <tr>
```

```
130      <td style="width: 25px">&ampnbsp</td>
131      <td class="modal-sm" style="width: 207px">Total dinners ordered:</td>
132      <td style="width: 17px">&ampnbsp</td>
133      <td style="width: 245px">
134          <asp:Label ID="lbl1Dinners" runat="server" BorderStyle="None" Width="89px"></asp:Label>
135      </td>
136      <td style="width: 15px">&ampnbsp</td>
137      <td>&ampnbsp</td>
138  </tr>
139  <tr>
140      <td style="width: 25px">&ampnbsp</td>
141      <td class="modal-sm" style="width: 207px">Subtotal:</td>
142      <td style="width: 17px">&ampnbsp</td>
143      <td style="width: 245px">
144          <asp:Label ID="lbl2Subtotal" runat="server" BorderStyle="None" Width="89px"></asp:Label>
145      </td>
146      <td style="width: 15px">&ampnbsp</td>
147      <td>&ampnbsp</td>
148  </tr>
149  <tr>
150      <td style="width: 25px">&ampnbsp</td>
151      <td class="modal-sm" style="width: 207px">Sales tax (3%):</td>
152      <td style="width: 17px">&ampnbsp</td>
153      <td style="width: 245px">
154          <asp:Label ID="lbl3SalesTax" runat="server" BorderStyle="None" Width="89px"></asp:Label>
155      </td>
156      <td style="width: 15px">&ampnbsp</td>
157      <td>&ampnbsp</td>
158  </tr>
159  <tr>
160      <td style="width: 25px">&ampnbsp</td>
161      <td class="modal-sm" style="width: 207px">Total price:</td>
162      <td style="width: 17px">&ampnbsp</td>
163      <td style="width: 245px">
164          <asp:Label ID="lbl4Total" runat="server" BorderStyle="None" Font-Bold="True" Width="89px"></asp:Label>
165      </td>
166      <td style="width: 15px">&ampnbsp</td>
167      <td>&ampnbsp</td>
168  </tr>
169  </table>
170 </asp:Content>
```

#### 4. Calculator.aspx.vb

```
1  ' Name:      Rosette Catering
2  ' Purpose:    Calculate the wedding catering.
3  ' Programmer: Me on just now.
4  Option Explicit On
5  Option Strict On
6  Option Infer Off
7
8  Partial Class Contact
9    Inherits Page
10
11  Private Sub btnSubmit_Click(sender As Object, e As EventArgs) Handles btnSubmit.Click
12    ' txt5Chicken, txt6Pasta, txt7Vege
13    Dim int1Chicken As Integer : Dim int2Pasta As Integer : Dim int3Vege As Integer
14    Dim intDinners As Integer : Dim intSubtotal As Integer : Dim decSales As Decimal
15
16    Integer.TryParse(txt5Chicken.Text, int1Chicken) : Integer.TryParse(txt6Pasta.Text, int2Pasta)
17    Integer.TryParse(txt7Vege.Text, int3Vege)
18
19    intDinners = int1Chicken + int2Pasta + int3Vege
20    intSubtotal = intDinners * 21
21    decSales = intSubtotal * 0.03D
22
23    lbl1Dinners.Text = intDinners.ToString
24    lbl2Subtotal.Text = intSubtotal.ToString("C2")
25    lbl3SalesTax.Text = decSales.ToString("C2")
26    lbl4Total.Text = (intSubtotal + decSales).ToString("C2")
27  End Sub
28 End Class
```

Home Page - Rosette Catering × +

localhost:50101

Rosette Catering Home Order form

# Rosette Catering

75 West Martin Avenue  
Chico, CA95926  
(111) 666-1234  
rosettecatering@yahoo.com



Your special day catering.

© 2021 - Rosette Catering

Order form - Rosette Catering

localhost:50101/Calculator

## Rosette Catering

### Order form: plan and calculate your special day catering.

Customer ID:

Bride's name:

Groom's name:

Date of wedding reception:

Number of dinners (á \$21.00):

Chicken:

Pasta:

Vegetarian:

Total dinners ordered: 100  
Subtotal: \$ 2,100.00  
Sales tax (3%): \$ 63.00  
Total price: \$ 2,163.00

© 2021 - Rosette Catering

## Syntax errors &amp; Error List window

**red squiggle** = syntax error - alerts you of syntax error  
**green squiggle** = warning - warns you of a potential problem



## Logic errors

- 1a). **F8** to run **Step Into Debugger**
- 1b). **F9** set a Breakpoint  + **F5** run the application
- 2). compare variable's and control's expected values with an actual values

when debugging your code, always correct the **syntax errors** first  
because doing so will often remove any **warnings**

Appendix C\_01 - **Syntax Errors** & VS's Error List window - errors caught by Code Editor - **red** & **green** squiggle, e.g. with **01.Total Sales Solution**

Appendix C\_02 - **Logic Errors** & VS's **Debugger**: menu's **Debug / Step Into F8** option to step through code, e.g. with **02.Discount Solution**

Appendix C\_03 - **Logic Errors** & VS's **Debugger**: menu's **Debug / Toggle Breakpoint F9** option to set **Breakpoint**, e.g. with **03.Hours Worked Solution**

Appendix C\_04 - **Run Time Errors** - errors occurring while an application is running, e.g. with **04.Quotient Solution**

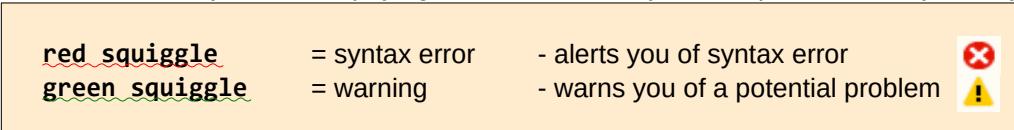
Appendix C\_Summary

Appendix C\_Terminology marked: 

Appendix C\_Exercises

## Appendix C\_01 - Syntax Errors & VS's Error List window - errors caught by Code Editor - red & green squiggle, e.g. with 01.Total Sales Solution

- > a **syntax error**:  
- occurs when you break one of a programming language's rules  
- mostly a result of typing errors that occur when entering instructions, such as typing Intger instead of Integer  
- the Code Editor detects as you enter the instructions ->  
-> however, if you are not paying close attention to your computer screen, you may not notice the error



- when debugging your code, always correct the **syntax errors** first because doing so will often remove any warnings

- > the **Error List window**:  
- part of VS debugging  
- indicates if the code contains:  
- **syntax errors**  
- **warnings**  
- and provides description <- by double-clicking the description, the blinking insertion point is positioned in the beginning of a problem  
- and provides localization of each in the code

- in the next set of steps, you will observe what happens when you start an application that contains a **syntax error** and learn to fix it

- to start debugging the **Total Sales Calculator** application - e.g. with **01.Total Sales Solution**

-> open the: ...VB2017\Appendix\_C1\_Exercise\01.Total Sales Solution\Total Sales Solution.sln

**step 1** -> open the **Designer** window and observe GUI

**Figure C-01**

- the application calculates and displays the total of the sales amounts entered by the user



**Figure C-01**

```
14 Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
15     ' Calculates and displays the total sales.
16     Dim intJack As Integer
17     Dim intMary As Integer
18     Dim intKhalid As Integer
19     Dim intSharon As Integer
20     Dim intTotal As Intger
21
22     Integer.TryParse(txtJack.Text, intJack)
23     Integer.TryParse(txtMary.Text, intMary)
24     Integer.TryParse(txtKhalid.Text, intKhalid)
25     Integer.TryParse(txtSharon.Text, intSharon)
26
27     inTotal = intJack + intMary + intKhalid + intSharon
28     lblTotal.Text = intTotal.ToString("C0")
29 End Sub
```

**Figure C-02**

**step 2** -> open the **Code Editor** window and observe the red and green jagged lines, called squiggles

- **red squiggle** = error - alerts you of syntax error  
- **green squiggle** = warning - warns you of a potential problem

step 3 -> press **F5** key to start/test/debug the application  
 <- notice the dialog box appears **Figure C-03**  
 -> click the button **No**



**Figure C-03**

<- the **Error List** window opens at the bottom of the IDE **Figure C-04**  
 - the **Error List** window:  
     - indicates that the code contains:  
         - **3 syntax errors**  
         - **1 warning**  
     - and provides description  
     - and provides localization of each in the code

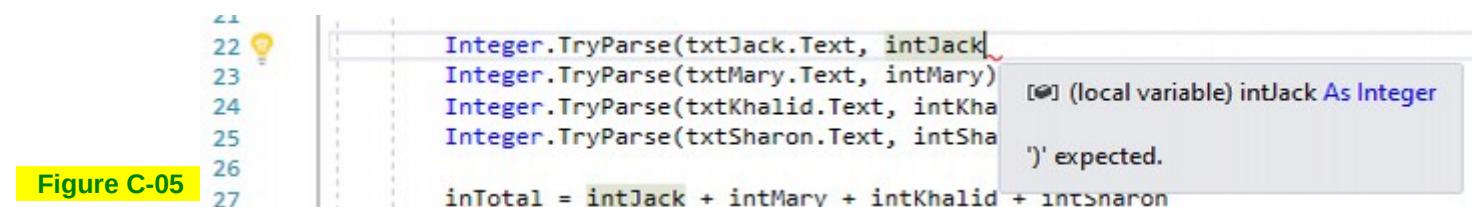
Error List				
Entire Solution		3 Errors	1 Warning	0 Messages
Code	Description	Project	File	Line
BC30198	')' expected.	Total Sales Project	Main Form.vb	22
BC30002	Type 'Intger' is not defined.	Total Sales Project	Main Form.vb	20
BC30451	'inTotal' is not declared. It may be inaccessible due to its protection level.	Total Sales Project	Main Form.vb	27
BC42104	Variable 'intTotal' is used before it has been assigned a value. A null reference exception could result at runtime.	Total Sales Project	Main Form.vb	28

**Figure C-04**

<- when debugging your code, always correct the syntax errors first because doing so will often remove any warnings

step 4.1 - the **1st** syntax error: **')' expected.** on the line **22**

- > double-click the 1st syntax error's description  
 <- notice the Code Editor positions the blinking insertion point just at the beginning of a red squiggle - at the beginning of an error
- > hover your mouse pointer over the red squiggle that appears next to the blinking insertion point  
 <- notice the error message box alerts you that the Code Editor is expecting an ending parenthesis on this line **Figure C-05**



- > type: )  
 <- notice the Code Editor removes the error from the **Error List** window

**step 4.2** - the 2nd syntax error: **Type 'Intger' is not defined.** on the line 20

-> double-click the 2nd syntax error's description

<- notice the Code Editor positions the blinking insertion point just at the beginning of a red squiggle - at the beginning of an error

<- notice a **LightBulb** indicator appears in the margin

-> hover your mouse pointer over the light bulb until a list arrow appears, and then click the list arrow

<- a list of suggestions for fixing the error appears

Figure C-06

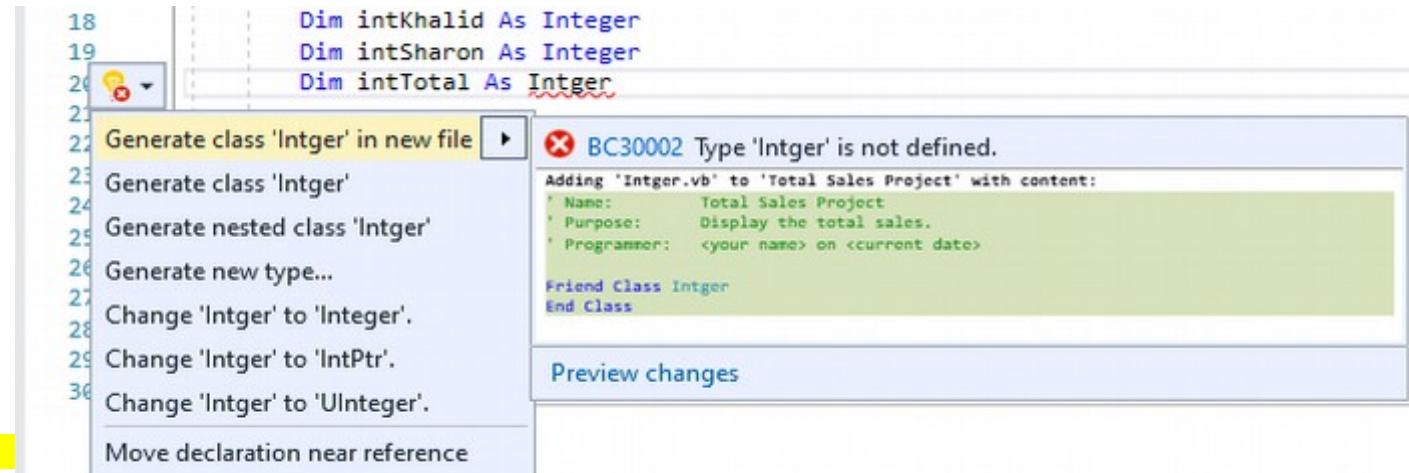


Figure C-06

- the error is obviously a typing error, the programmer meant to type **Integer** rather than **Intger**

-> you can either type the missing letter **e** yourself, or click the appropriate suggestion in the list

<- notice the Code Editor makes the changes in the **Dim** statement and also removes both the error and the warning from the **Error List** window

**step 4.3** - the remaining error: **'inTotal' is not declared. It may be inaccessible due to its protection level.** on the line 27

- the description of the remaining error indicates that the Code Editor does not recognize the name **inTotal**

-> double-click the syntax error's description

<- notice the Code Editor positions the blinking insertion point just at the beginning of a red squiggle - at the beginning of an error

<- notice a **LightBulb** indicator appears in the margin

-> hover your mouse pointer over the light bulb until a list arrow appears, and then click the list arrow

<- a list of suggestions for fixing the error appears

- this error is another typing error - the variable's name is **intTotal**, not **inTotal**

-> click: **Change 'inTotal' to 'intTotal'.** in the list

-> you can either type the missing letter **e** yourself, or click the appropriate suggestion in the list

<- notice the Code Editor removes the error from the **Error List** window

**step 5** -> save the solution and then start and test the application: type: **125600, 98700, 165000, 250400** and click the button **Calculate**. Label box shows: **\$ 639,700**

- > a **logic error**:
  - unlike syntax error, much more difficult to find because it does not trigger an error message from the Code Editor
  - can occur for a variety of reasons:
    - forgetting to enter an instruction
    - entering the instructions in the wrong order
    - calculation statements are correct syntactically but incorrect mathematically
- > the VS menu's **Debug / Step Into F8** option:
  - will start your application and allows you to step through your code by:
    - 1). pausing immediately before each statement is executed
    - 2). highlighting the statement
    - 3). viewing the content of the controls and variables just by hovering your mouse cursor over them
      - 1). controls & variables that appear in the highlighted line, and also
      - 2). controls & variables in already executed statements that precede the highlighted line
    - <- before you view the contents of a control or variable, you should consider the value you expect to find
    - 4). pressing **F8** key will execute the highlighted line and pauses just before the next line and highlight it

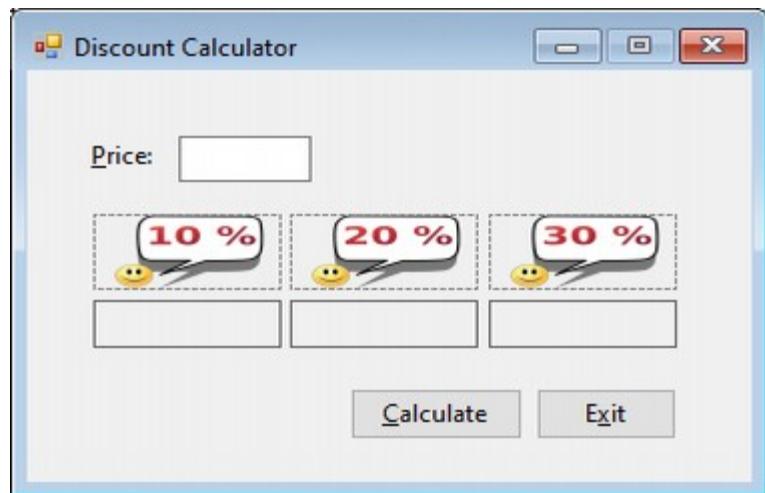
- in this section, you will debug an application that contains a logic error, using **Debug / Step Into F8** option to step through your code

- to debug the **Discount Calculator** application - e.g. with **02.Discount Solution**

-> open the: ...VB2017\Appendix\_C\Exercise\02.Discount Solution\Discount Solution.sln

**step 1** -> open the **Designer** window and observe GUI    **Figure C-07**

- the application calculates and displays **3 discount amounts**, which are based on the price entered by the user



**Figure C-07**

```
14      Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
15          ' Calculates and displays discounts.
16
17          Dim decPrice As Decimal
18          Dim decDiscount10 As Decimal
19          Dim decDiscount20 As Decimal
20          Dim decDiscount30 As Decimal
21
22          decDiscount10 = decPrice * 0.1D
23          decDiscount20 = decPrice * 0.2D
24          decDiscount30 = decPrice * 0.3D
25
26          lbl10.Text = decDiscount10.ToString("N2")
27          lbl20.Text = decDiscount20.ToString("N2")
28          lbl30.Text = decDiscount30.ToString("N2")
29      End Sub
```

**Figure C-08**

**step 2** -> open the **Code Editor** window and observe the **btnCalc\_Click** procedure

**Figure C-08**

- step 3** -> start the application - **F5** key - & type: **100** in the **Price:** box, followed by the **Calculate** button click  
 <- notice the interface shows that each discount is **0.00** <- which is incorrect  
 -> click the **Exit** button to close the application

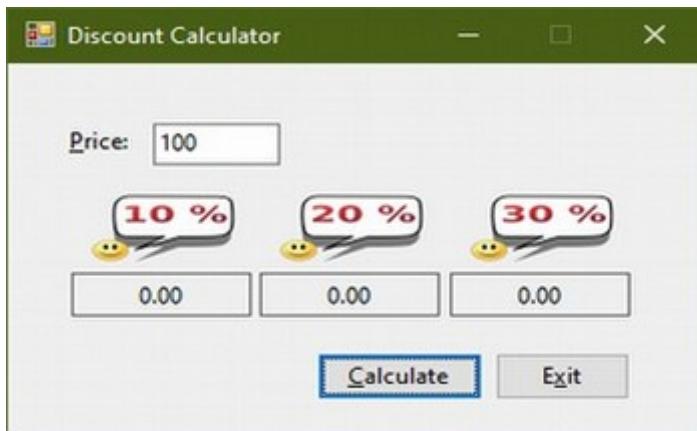


Figure C-09

Figure C-09

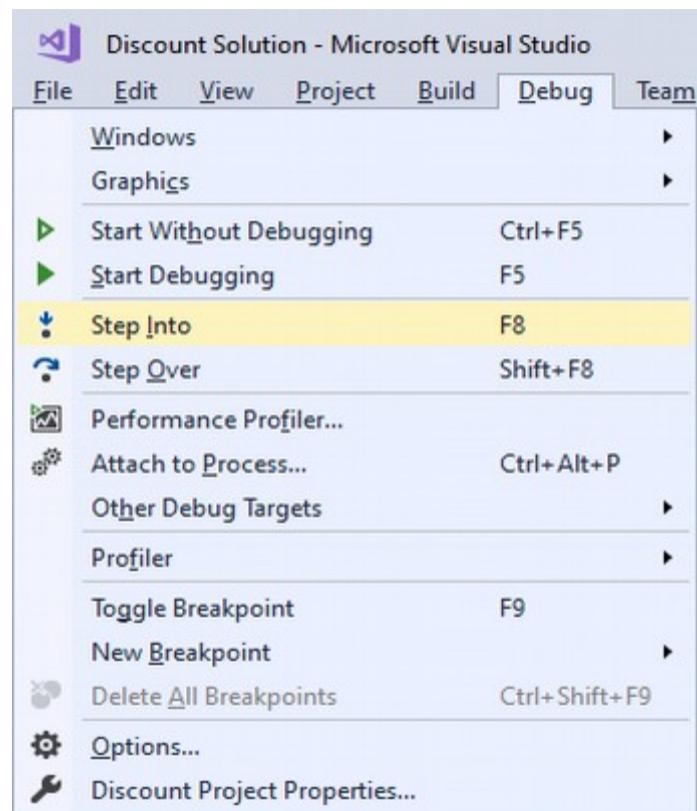


Figure C-10

- step 4.0** - you will use the **Debug** menu to run the **Visual Basic debugger**, which is a tool that helps you locate the logic errors in your code  
 - the menu's **Debug / Step Into F8** option: **Figure C-10**  
   - will start your application and allows you to step through your code  
   - it does this by executing the code 1 statement at a time,  
     pausing immediately before each statement is executed

- step 4.1** -> click the: menu bar: **Debug / Step Into F8** **Figure C-10**

- step 4.2** -> in the **Price:** box type: **100** and then click the **Calculate** button  
 <- notice: - the **Debugger** highlights the 1st instruction to be executed, which is **btnCalc\_Click** header  
   - it also pauses the code's execution  
   - in addition, an **arrow** points to the procedure header and the **LightBulb** indicator appears **Figure C-11**

- the **Debugger** is waiting - you can now change the highlighted code, or just hit the **F8** key to tell the computer to execute the highlighted instruction  
 -> to continue, either press the **F8** key, or click the menu's **Debug / Step Into F8** option

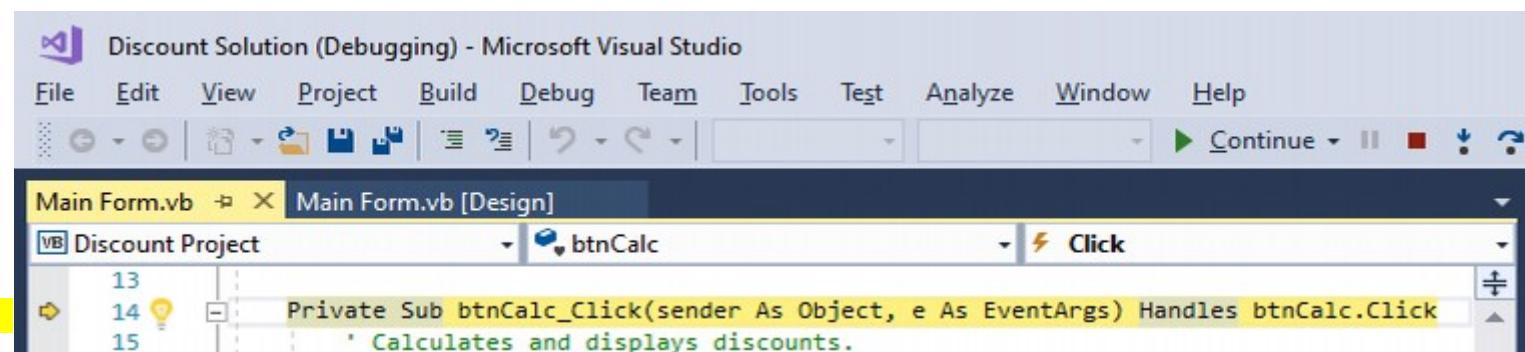


Figure C-11

**step 4.3** - after the computer processes the procedure header, the **Debugger** highlights the next statement to be processed: `decDiscount10 = decPrice * 0.1D`  
- it then pauses execution of the code  
<- notice the **Dim** statements are skipped over because they are not considered executable by the **Debugger**

Figure C-12

- while the execution of a procedure's code is paused, you can view the contents of controls and variables that appear in the highlighted statement and also in the statements that precede it in the procedure
- before you view the contents of a control or variable, you should consider the value you expect to find
- before the highlighted statement is processed, the `decDiscount10` variable should contain its initial value: `0`
  - recall that the **Dim** statement initializes numeric variables to `0`
- > place your mouse pointer on `decDiscount10` in the highlighted statement
- <- notice the variable's name and current value appear in a small box
- at this point, the `decDiscount10` variable's value is correct

Figure C-12

Figure C-12

```
Discount Solution (Debugging) - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
Main Form.vb [Design]
VB Discount Project btnCalc Click
13
14     Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
15         ' Calculates and displays discounts.
16
17         Dim decPrice As Decimal
18         Dim decDiscount10 As Decimal
19         Dim decDiscount20 As Decimal
20         Dim decDiscount30 As Decimal
21
22         decDiscount10 = decPrice * 0.1D
23         decDiscount20 = decPrice * 0.2D
24         decDiscount30 = decPrice * 0.3D
```

Figure C-12

**step 4.4** -> now, consider the value you expect to find in the `decPrice` variable

- before the highlighted statement is processed, the variable should contain the number `100`, which is the value you entered in the **Price** box

-> place your mouse pointer on `decPrice` in the highlighted statement

<- notice the variable contains `0`, which is its initial value

- the value is incorrect because no statement above the highlighted statement assigns the **Price** box's value to the `decPrice` variable

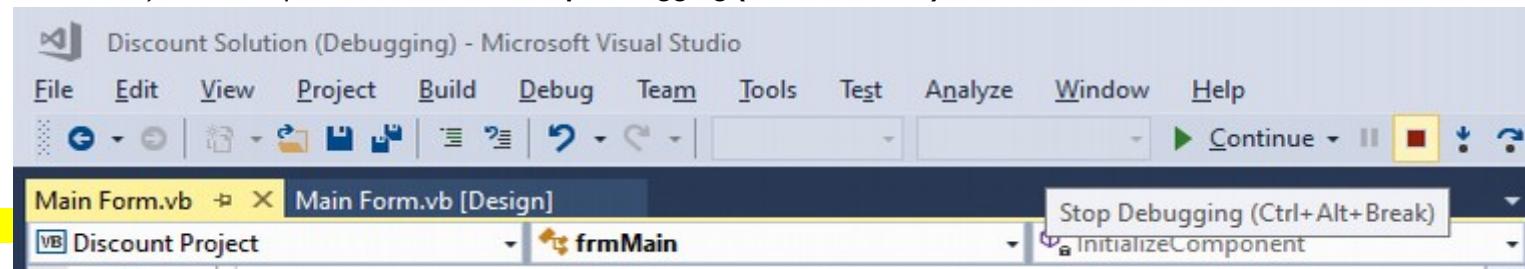
- in other words, a statement is missing from the procedure

```
22     decDiscount10 = decPrice * 0.1D
23     decDiscount20 = decPrice * 0.2D
24     decDiscount30 = decPrice * 0.3D
```

Figure C-13

**step 4.5** - you will stop the **Debugger** and add the missing statement

- > stop the **Debugger** by clicking either:
  - a). menu's **Debug / Stop Debugging Ctrl+Alt+Break** option, or
  - b). the red square button named **Stop Debugging (Ctrl+Alt+Break)** on the Standard toolbar



-> below the last **Dim** statement, type the following **TryParse** method:

```
20      Dim decDiscount30 As Decimal  
21  
22      Decimal.TryParse(txtPrice.Text, decPrice)  
23  
24      decDiscount10 = decPrice * 0.1D
```

**step 5.1** -> save the solution and repeat the **step 4.1** to start **Debugging** your application

-> in the **Price:** box type: **100** and then click the **Calculate** button

-> press **F8** to process the procedure header **Figure C-11**

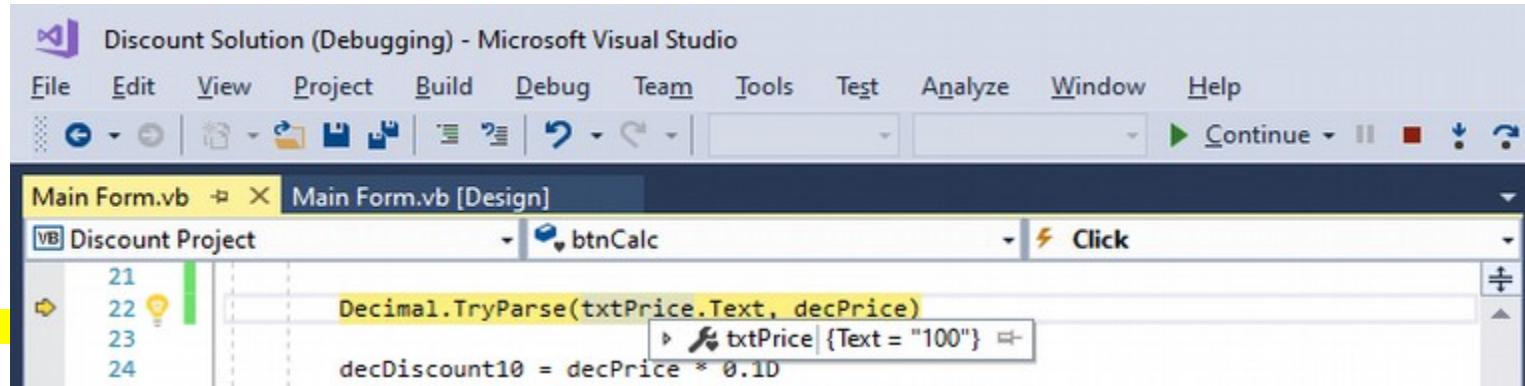
<- notice the **Debugger** highlights the **TryParse** method and then pauses execution of the code

- before the **TryParse** method is processed, the **txtPrice.Text** property should contain **100**, which is the value you entered in the **Price** box

-> place your mouse pointer on **txtPrice.Text** in the **TryParse** method **Figure C-15**

<- notice: - the box shows that the **Text** property contains the expected value **100** **Figure C-15**

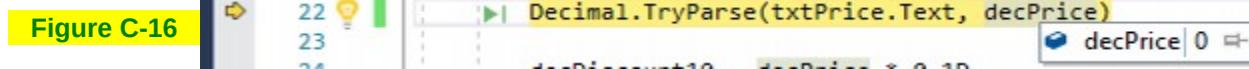
- the "**100**" is enclosed in quotation marks because Visual Basic treats the contents of a **TextBox** as a **string**



-> place your mouse pointer on **decPrice** in the **TryParse** method **Figure C-16**

<- notice the box shows that the variable contains the expected value - the **decPrice** variable should contain its initial value: **0**

**Figure C-16**



-> press **F8** to process the **TryParse** method

**step 5.2** - the Debugger highlights the: `decDiscount10 = decPrice * 0.1D` statement before pausing execution of the code

**Figure C-17**

-> place your mouse pointer on `decPrice` in the `TryParse` method above the highlighted line

**Figure C-17**

<- notice that after the method is processed by the computer, the `decPrice` variable contains the number **100**, which is correct

The screenshot shows the Microsoft Visual Studio interface with the title bar "Discount Solution (Debugging) - Microsoft Visual Studio". A tooltip labeled "Figure C-17" is visible on the left side. The main window displays the code for "Main Form.vb [Design]". The code editor shows the following lines of code:

```
22 |    Decimal.TryParse(txtPrice.Text, decPrice)
23 |
24 |    decDiscount10 = decPrice * 0.1D
25 |    decDiscount20 = decPrice * 0.2D
26 |    decDiscount30 = decPrice * 0.3D
27 |
28 |    lbl10.Text = decDiscount10.ToString("N2")
```

The line `decDiscount10 = decPrice * 0.1D` is highlighted in yellow, indicating it is the current statement being executed. A tooltip box appears over the variable `decPrice` with the value **100** and a small icon.

- before the highlighted statement is processed: 1). the `decDiscount10` variable should contain its initial value: **0**, and

2). the `decPrice` variable should contain the value assigned to it by the `TryParse` method

-> place your mouse pointer on `decDiscount10` in the highlighted statement

<- notice the box shows that the variable contains **0**, which is correct

**Figure C-18a**

-> place your mouse pointer on `decPrice` in the highlighted statement

<- notice the box shows that the variable contains **100**, which is also correct

**Figure C-18b**

The image contains two side-by-side screenshots of the Microsoft Visual Studio code editor. Both screenshots show the same code block:

```
23 |
24 |    decDiscount10 = decPrice * 0.1D
25 |    decDiscount20 = decPrice * 0.2D
```

In the top screenshot (labeled "Figure C-18a"), the variable `decDiscount10` is highlighted in yellow, and a tooltip box shows its value as **0**.

In the bottom screenshot (labeled "Figure C-18b"), the variable `decPrice` is highlighted in yellow, and a tooltip box shows its value as **100**.

**step 5.3** - after the highlighted statement is processed, the `decPrice` variable should still contain **100**

- however, the `decDiscount10` variable should contain **10**, which is 10% of 100

-> press **F8** to execute the highlighted statement

-> place your mouse pointer on `decDiscount10` in the statement

<- notice the box shows that the variable in the statement contains the appropriate value: **10.0**

**Figure C-19**

The screenshot shows the Microsoft Visual Studio code editor with the following code:

```
24 |
25 |    decDiscount10 = decPrice * 0.1D
26 |    decDiscount20 = decPrice * 0.2D
```

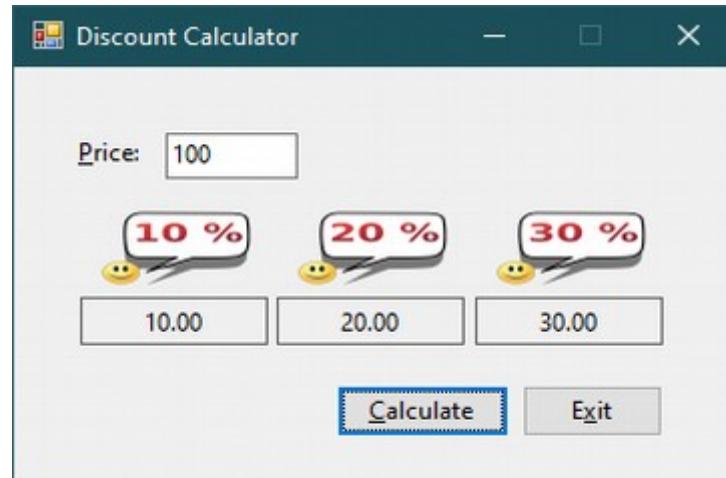
The variable `decDiscount10` is highlighted in yellow, and a tooltip box shows its value as **10.0**.

-> verify that the `decPrice` variable in the statement contains the appropriate value: **100**

**step 6** - now, that you fixed the logic error, you will continue program execution without using the **Debugger**

- > to continue the program without **Step Into Debugger**, either:
- click the menu bar's **Debug / Continue F5** option
  - click the green arrow button named **Continue** on the Standard toolbar
  - press **F5** key on your keyboard

<- notice the correct discount amounts appear in the interface



**Figure C-19**

-> close the window and close the solution

- > a **logic error**: - unlike syntax error, much more difficult to find because it does not trigger an error message from the Code Editor  
 - can occur for a variety of reasons: - forgetting to enter an instruction  
 - entering the instructions in the wrong order  
 - calculation statements are correct syntactically but incorrect mathematically

- stepping through code - **Debug / Step Into** F8 - one line at a time is not the only way to search for logic errors

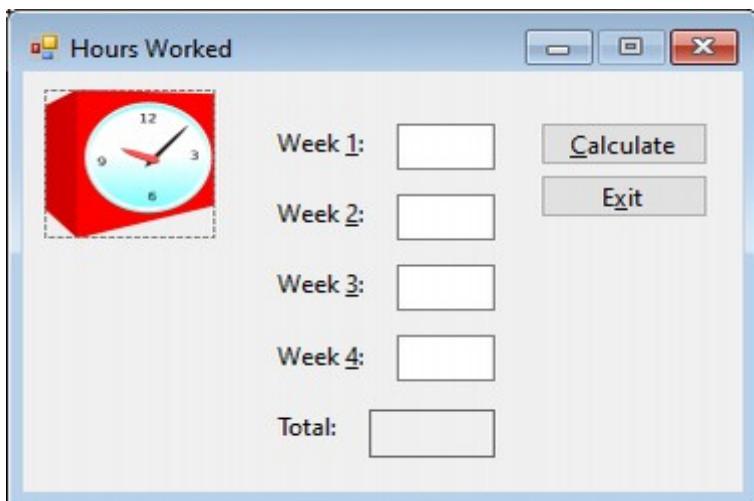
- > the VS menu's **Debug / Toggle Breakpoint** F9 option: - useful when you have at least a slight idea where to look, since this option places a **Breakpoint** to pause execution at a specific line in the code  
 - works only with: menu's **Debug / Start Debugging** F5 option  
 - won't work with: menu's **Debug / Start Without Debugging** Ctrl+F5 option  
 - same like when using **Step Into** F8 Debugger:  
 - just by mouse cursor hovering over, allowing you to view the contents of controls and variables:  
 1). that appear in the highlighted line, and  
 2). also in already executed statements that precede the highlighted line  
 <- before you view the contents of a control or variable, you should consider the value you expect to find

- to debug the **Hours Worked** application by toggling a **Breakpoint** - e.g. with **03.Hours Worked Solution**

-> open the: ...VB2017\Appendix\_C1\_Exercise\03.Hours Worked Solution\Hours Worked Solution.sln

**step 1** -> open the **Designer** window and observe GUI **Figure C-20**

- the application calculates and displays the total number of hours worked in **4 weeks**



**Figure C-20**

```

14      Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
15          ' Calculates and displays the total number of hours worked during 4 weeks.
16
17          Dim dblWeek1 As Double
18          Dim dblWeek2 As Double
19          Dim dblWeek3 As Double
20          Dim dblWeek4 As Double
21          Dim dblTotal As Double
22
23          Double.TryParse(txtWeek1.Text, dblWeek1)
24          Double.TryParse(txtWeek2.Text, dblWeek2)
25          Double.TryParse(txtWeek3.Text, dblWeek2)
26          Double.TryParse(txtWeek4.Text, dblWeek4)
27
28          dblTotal = dblWeek1 + dblWeek2 + dblWeek3 + dblWeek4
29          lblTotal.Text = dblTotal.ToString("N1")
30      End Sub

```

**Figure C-21**

**step 2** -> open the **Code Editor** window and observe the **btnCalc\_Click** procedure

**Figure C-21**

- step 3** -> start the application - **F5** key - & type in the TextBoxes: **10.5, 25, 33, and 40** & click the **Calculate** button  
 <- notice the interface shows that the total number of hours is **83.5** <- which is incorrect, it should be **108.5**  
 -> click the **Exit** button to close the application

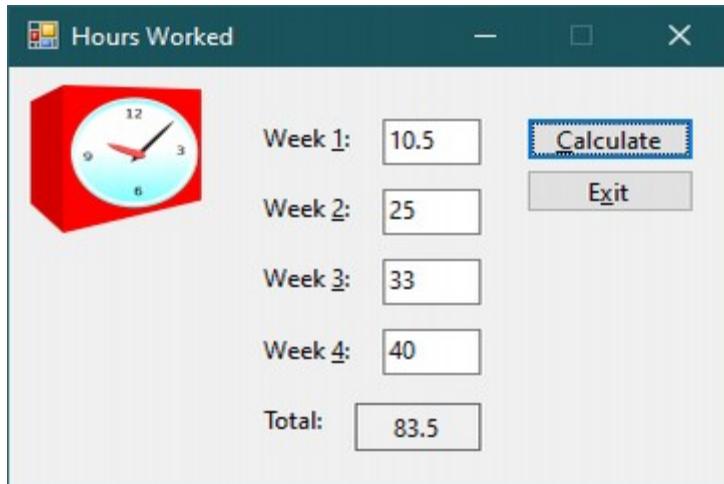


Figure C-22

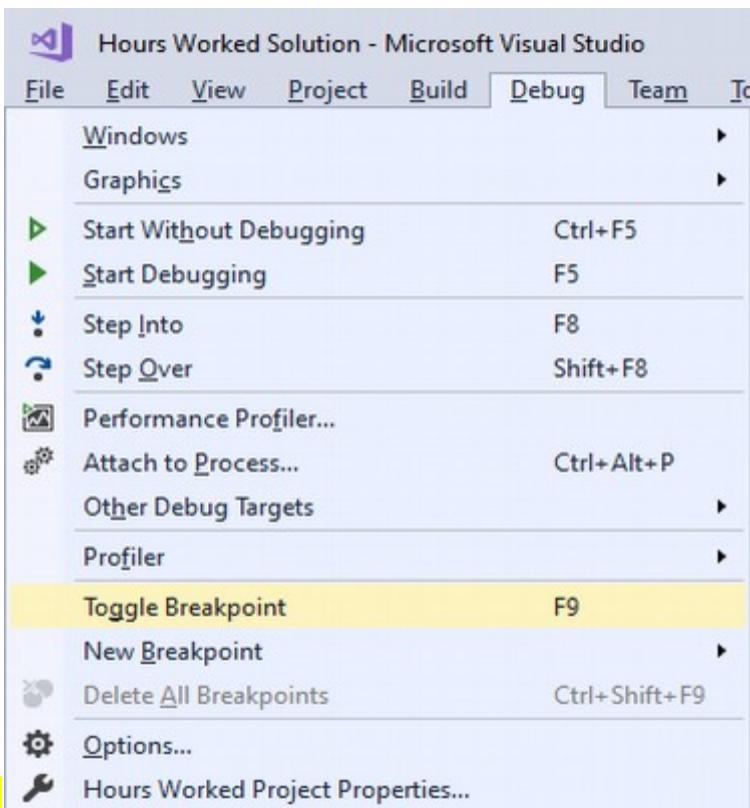


Figure C-23

- step 4.0**
- the statement that calculates the total number of hours worked is not giving the correct result
  - rather than having the computer pause before processing each line of code in the procedure, you will have it pause only before processing the calculation statement
  - you can do this by setting a **Breakpoint** on the statement by either:
    - a. clicking anywhere in the line & then clicking the menu's **Debug / Toggle Breakpoint F9**
    - b. clicking in the gray margin that appears to the left of the statement
 - the debugger then highlights the statement and places a red circle next to it

- step 4.1** - to set a **Breakpoint**, you can either:

- > click anywhere within the calculation statement on line **28** & click the menu's **Debug / Toggle Breakpoint F9** option, or
- > click in the gray margin that appears to the left of the statement

<- notice that in either way, the **Debugger** highlights the statement and places a red circle next to it

Figure C-23

Figure C-24

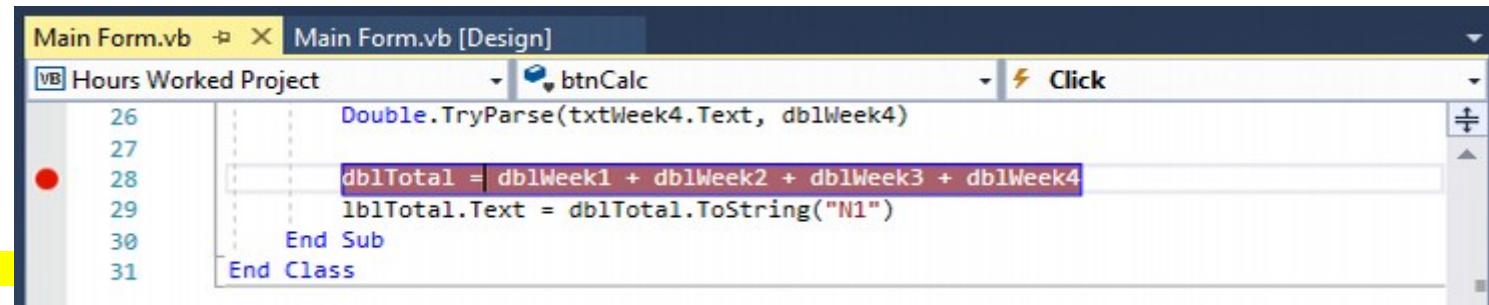
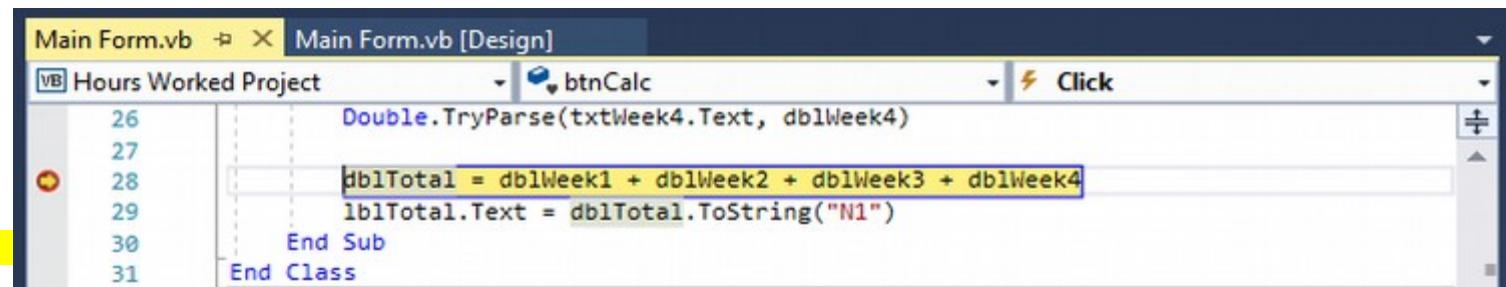


Figure C-24

- step 5.1** -> start the application - **F5** key - & type in the TextBoxes: **10.5**, **25**, **33**, and **40** & click the **Calculate** button  
 - the computer begins processing the code contained in the **btnCalc\_Click** procedure  
 - it stops processing when it reaches the **Breakpoint** statement - which it highlights  
 - the highlighting indicates that the statement is the next one to be processed  
 <- notice that a yellow arrow now appears in the red dot next to the **Breakpoint**

Figure C-25



```
Main Form.vb [Design]
VB Hours Worked Project btnCalc Click
26 Double.TryParse(txtWeek4.Text, dblWeek4)
27
28 dblTotal = dblWeek1 + dblWeek2 + dblWeek3 + dblWeek4
29 lblTotal.Text = dblTotal.ToString("N1")
30 End Sub
31 End Class
```

Figure C-25

- step 5.2** - before viewing the values contained in each variable in the highlighted statement, consider the values you expect to find  
 1). before the calculation statement is processed, the **dblTotal** variable should contain its initial value: **0**  
 <- you can verify the variable's intial value by placing your mouse pointer on **dblTotal** in its **Dim** declaration statement  
 2). the **dblWeek1** variable should contain the number **10.5**, which is the value you entered in the **txtWeek1** TextBox  
 3). the **dblWeek2** variable should contain the number **25**, which is the value you entered in the **txtWeek2** TextBox  
 4). the **dblWeek3** variable should contain the number **33**, which is the value you entered in the **txtWeek3** TextBox  
 5). the **dblWeek4** variable should contain the number **40**, which is the value you entered in the **txtWeek4** TextBox

-> place your mouse pointer on each of the variables in the highlighted statement and compare the actual values with the expected values

- <- notice the variable **dblTotal** = **0** <- which is correct
- <- notice the variable **dblWeek1** = **10.5** <- which is correct
- <- notice the variable **dblWeek2** = **33** <- which is incorrect, the expected/correct value should be **25**
- <- notice the variable **dblWeek3** = **0** <- which is incorrect, the expected/correct value should be **33**
- <- notice the variable **dblWeek4** = **40** <- which is correct

Figure C-26a

Figure C-26b

Figure C-26a

Figure C-26b

- since the **TryParse** methods are responsible for assigning the TextBox values to the variables -  
 - there is obviously something wrong in 2 cases: **dblWeek2** and **dblWeek3** <- juchuu you just located the errors

**step 6.0** - you will stop **Debugger** & look closely to the **TryParse** methods for the variables **dblWeek2** and **dblWeek3** & remove unneeded **Breakpoint**

**step 6.1** -> stop the **Debugger** by clicking either: a). menu's **Debug / Stop Debugging Ctrl+Alt+Break** option, or  
b). the red square button named **Stop Debugging (Ctrl+Alt+Break)** on the Standard toolbar

Figure C-27

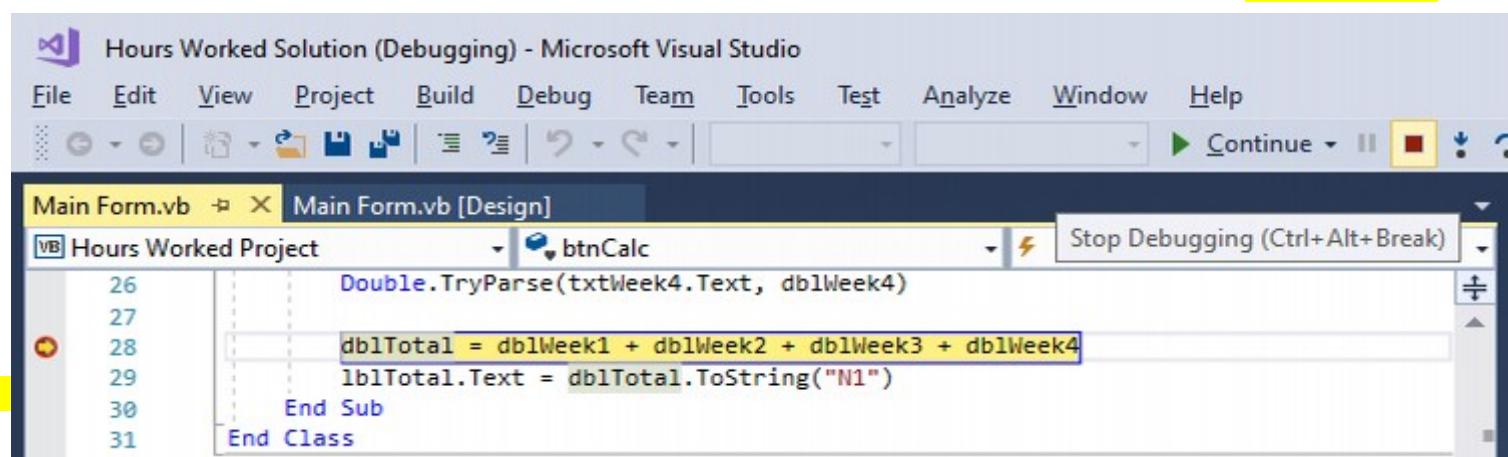


Figure C-27

**step 6.2** -> observe the **TryParse** methods and notice the line **25**

-> change the **dblWeek2** to **dblWeek3** on the line **25**

original:

```
23 Double.TryParse(txtWeek1.Text, dblWeek1)
24 Double.TryParse(txtWeek2.Text, dblWeek2)
25 Double.TryParse(txtWeek3.Text, dblWeek2)
26 Double.TryParse(txtWeek4.Text, dblWeek4)
```

fixed:

```
23 Double.TryParse(txtWeek1.Text, dblWeek1)
24 Double.TryParse(txtWeek2.Text, dblWeek2)
25 Double.TryParse(txtWeek3.Text, dblWeek3)
26 Double.TryParse(txtWeek4.Text, dblWeek4)
```

**step 6.3** - to remove the **Breakpoint**, follow the same way like to set it - you can either:

- > click anywhere within the calculation statement on line **28** & click the menu's **Debug / Toggle Breakpoint F9** option, or
- > click in the gray margin that appears to the left of the statement

<- notice that in either way, the **Debugger** removes the statement highlight and removes a red circle next to it

Figure C-23

Figure C-28

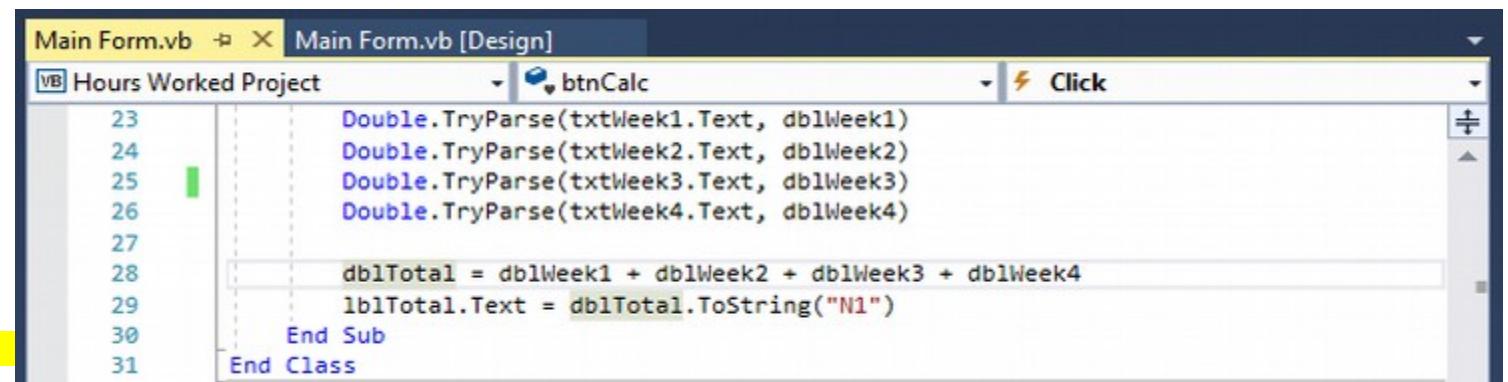


Figure C-28

**step 7** -> same like in **step 3**, start the application - **F5** key - & type in the TextBoxes: **10.5, 25, 33, and 40** & click the **Calculate** button  
<- notice the interface shows that the total number of hours is **108.5** <- which is correct  
-> click the **Exit** button to close the application and then close the solution

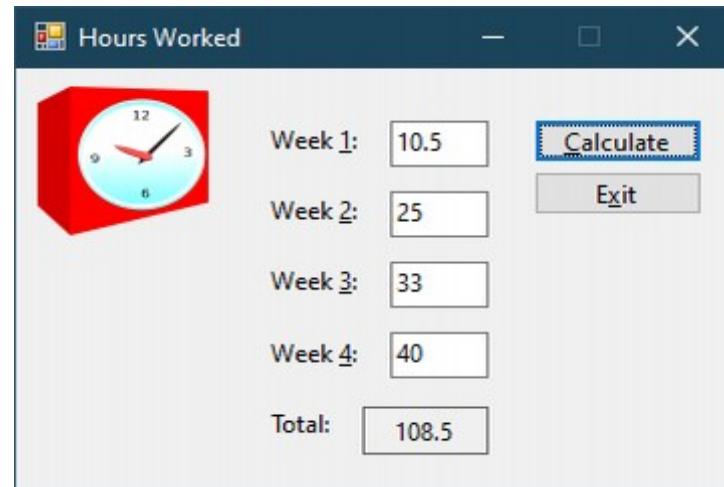


Figure C-29

#### Appendix C\_04 - Run Time Errors - errors occurring while an application is running, e.g. with **04.Quotient Solution**

- in addition to syntax errors and logic errors, programs also can have run time errors

-> a run time error: = an error that occurs while an application is running

- e.g.1 infinite loop causing memory overflow
- e.g.2 opening a text file that is already open
- e.g.3 opening a text file the computer cannot locate
- e.g.4 dividing by 0

**CH5\_A1 - use a Loop, a Counter, and an Accumulator:**

**CH9\_F2 - Sequential Access Output Files: Class StreamWriter - ...**

**CH9\_F3 - Sequential Access Input Files: Class StreamReader - ...**

e.g.1 as seen in: **CH5\_A1 - use a Loop, a Counter, and an Accumulator:**

- an infinite loop causing memory overflow
- when you test the application and in the **txtCurrentSales** TextBox write **nothing / 0** as a initial value for **dblSales** variable
  - the condition: **21      Do While dblSales < 150000** always evaluates to **TRUE**, **infinitely** increasing the **intYears**
  - once the **intYears** variable contains the largest value that can be stored in an **Integer**: **2147483647**, the statement causes an **overflow error**
- a **run time error** - an **overflow error** occur when the value assigned to a memory location is too large for the location's data type
- solution: use a compound condition to avoid an infinite loop

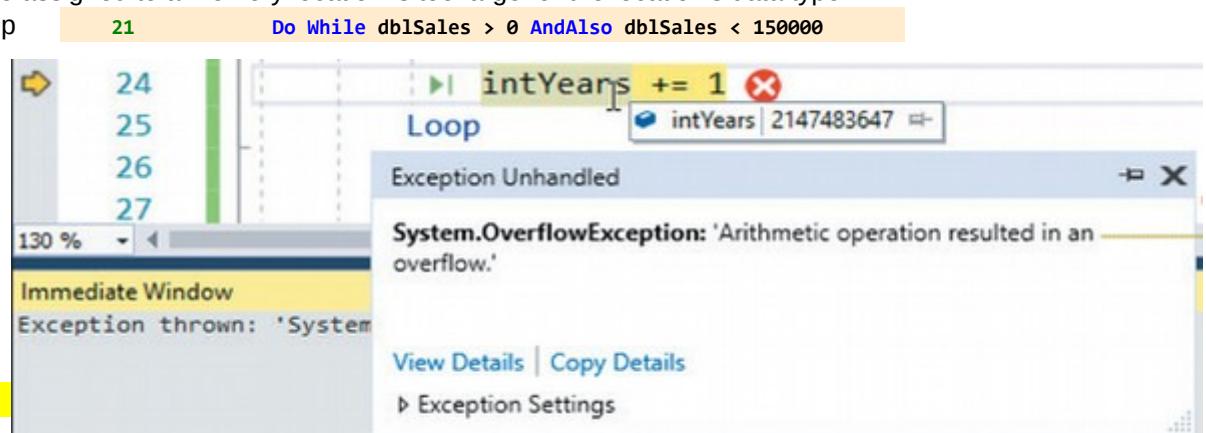


Figure 5-18

→ an **overflow error**:

- occurs when the value assigned to a memory location is too large for the location's data type
- similar to trying to fill a 0.2L glass with 1L of water
- in this case, the **intYears** variable already contains the largest value that can be stored in an Integer variable (**2,147,483,647** -**CH3**)
  - > therefore, when the **intYears += 1** statement attempts to increase the variable's value by **1**, an overflow error occurs

original: if using **dblSales = 0**, causing an infinite loop until the counter's data type overflows, causing a **run time error** - an **overflow error**

```
15      Const dblGROWTH_RATE As Double = 0.03
16      Dim dblSales As Double ' Used as an accumulator.
17      Dim dblIncrease As Double
18      Dim intYears As Integer ' Used as a counter.
19
20      Double.TryParse(txtCurrentSales.Text, dblSales)
21      Do While dblSales < 150000
22          dblIncrease = dblSales * dblGROWTH_RATE
23          dblSales += dblIncrease
24          intYears += 1
25      Loop
```

modified: avoiding the infinite loop

```
15      Const dblGROWTH_RATE As Double = 0.03
16      Dim dblSales As Double ' Used as an accumulator.
17      Dim dblIncrease As Double
18      Dim intYears As Integer ' Used as a counter.
19
20      Double.TryParse(txtCurrentSales.Text, dblSales)
21      Do While dblSales > 0 AndAlso dblSales < 150000
22          dblIncrease = dblSales * dblGROWTH_RATE
23          dblSales += dblIncrease
24          intYears += 1
25      Loop
```

e.g.2 as seen in: **CH9\_F2 - Sequential Access Output Files: Class StreamWriter** - how to create and use them step by step

- a **run time error - IO error** will occur if a program statement attempts to open a file that is **already** open
- you should always close an output file as soon as you are finished using it - this ensures that the data is saved, and it makes the file available for use elsewhere in the application
- solution: close the output file using **IO.StreamWriter Class**'s method **Close()**

e.g.3 as seen in: **CH9 F3 - Sequential Access Input Files: Class StreamReader** - how to create and use them step by step

- a **run time error - IO.file not found error** occur if a program statement attempts to open an input file that the computer cannot locate
- it is **important** to make this determination because a **run time error** will occur if your code attempts to open an input file that the computer cannot locate
- solution: use the **File Class**'s function/method **Exists** to determine whether the input file **exists**

```
If IO.File.Exists("employee.txt") Then  
    inFile = IO.File.OpenText("employee.txt")  
  
...  
inFile.Close()
```

e.g.4 - as you will observe in the following set of steps, a **run time error - a divide by zero** occur when an expression attempts to divide a value by the number 0

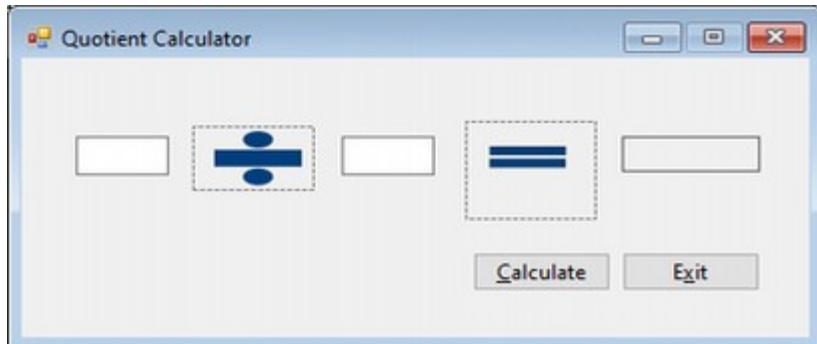
- recall the formula for division: **numerator / denominator = quotient**      **čitatel / jmenovatel = podíl** (výsledek dělení)

- to use the **Quotient Calculator** application to observe a **run time error** - e.g. with **04.Quotient Solution**

-> open the: ...VB2017\Appendix\_C\Exercise\04.Quotient Solution\Quotient Solution.sln

**step 1** -> open the **Designer** window and observe GUI **Figure C-30**

- the application divides **2** numbers entered by the user and displays the result - the **quotient** in a Label



**Figure C-30**

```
14      Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click  
15          ' Display the result of dividing two numbers.  
16  
17          Dim decNumerator As Decimal  
18          Dim decDenominator As Decimal  
19          Dim decQuotient As Decimal  
20  
21          Decimal.TryParse(txtNumerator.Text, decNumerator)  
22          Decimal.TryParse(txtDenominator.Text, decDenominator)  
23  
24          decQuotient = decNumerator / decDenominator  
25          lblQuotient.Text = decQuotient.ToString("N2")  
26      End Sub
```

**Figure C-31**

**step 2** -> open the **Code Editor** window and observe the **btnCalc\_Click** procedure

**Figure C-31**

**step 3.1** -> start the application - F5 key  
-> and type in: - txtNumerator = 100  
- txtDenominator = 5

-> then click the **Calculate** button  
<- notice the interface shows in **lblQuotient** the result is **20.00** <- which is correct

Figure C-32

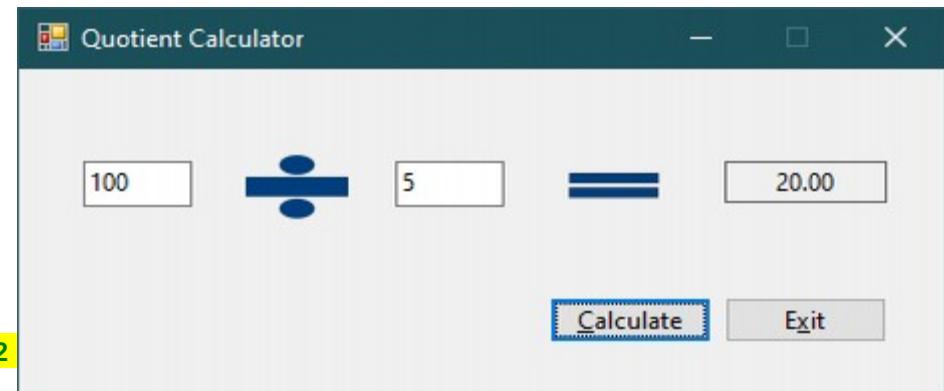


Figure C-32

**step 3.2** -> now, delete the number **5** from the **txtDenominator** control and then click the **Calculate** button

<- notice a run time error - divide by zero error occurs, indicating that the highlighted statement is attempting to divide by 0

Figure C-33

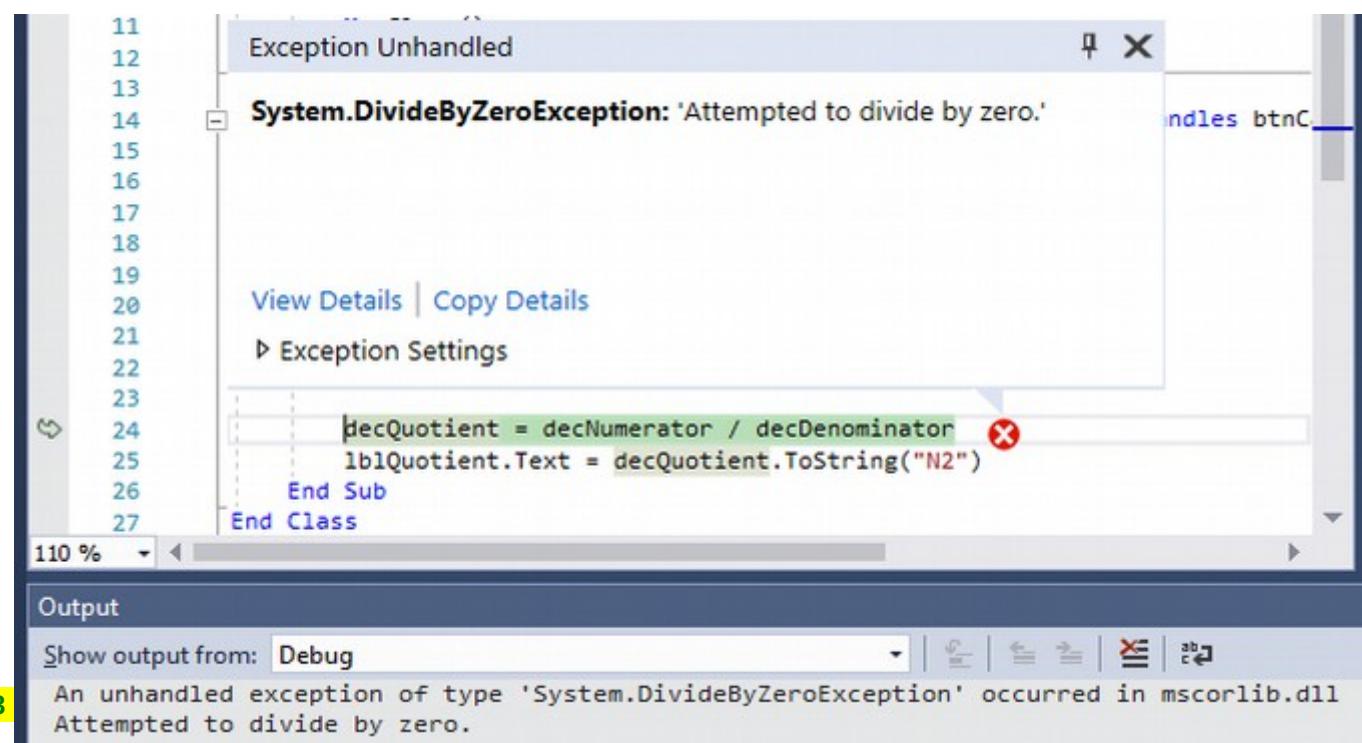


Figure C-33

**step 4.0** - to prevent this error from occurring, you will need to tell the computer to calculate and display the quotient only when the **decDenominator** variable does not contain the number **0** - otherwise, it should display the **N/A** message = **Not Available, No Answer, Not Applicable**  
- you can do this using a selection structure

**step 4.1** -> stop the **Debugger** by clicking either: a). menu's **Debug / Stop Debugging Ctrl+Alt+Break** option, or  
b). the red square button named **Stop Debugging (Ctrl+Alt+Break)** on the Standard toolbar

**step 4.2** -> enter the selection structure shown below  
<- be sure to move the statements that calculate and display the quotient into the selection structure's **TRUE** path

```
14      Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
15          ' Display the result of dividing two numbers.
16
17          Dim decNumerator As Decimal
18          Dim decDenominator As Decimal
19          Dim decQuotient As Decimal
20
21          Decimal.TryParse(txtNumerator.Text, decNumerator)
22          Decimal.TryParse(txtDenominator.Text, decDenominator)
23
24          If decDenominator <> 0 Then
25              decQuotient = decNumerator / decDenominator
26              lblQuotient.Text = decQuotient.ToString("N2")
27          Else
28              lblQuotient.Text = "N/A"
29          End If
30      End Sub
```

**step 4.3** -> start the application - **F5** key - type: **100**, and **5** & click the **Calculate** button  
<- notice the interface shows in **lblQuotient** the result is **20.00** <- which is correct  
-> now, delete the number **5** from the **txtDenominator** control and then click the **Calculate** button  
<- notice instead of a run time error, a **N/A** message appears in the **lblQuotient** indicating, that the calculation can't be processed

-> close GUI and close the solution

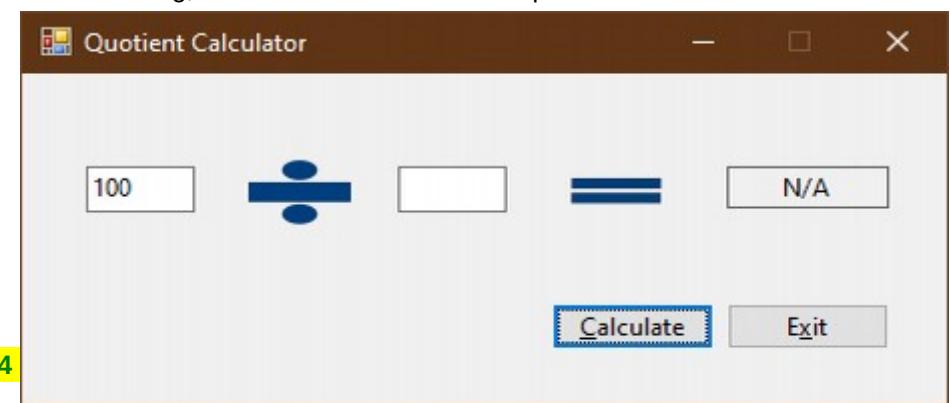


Figure C-34

## Appendix C\_Summary

1. to find the **syntax errors** :
  - look for **red** & **green** squiggles/jagged lines in the **Code Editor** window
  - or just start the application - **F5** key - and then look in the VS's **Error List** window
- 2a. to find the **logic errors** :
  - use the VS menu's **Debug / Step Into F8** option
  - to step through the code - line by line, and compare variable's and control's expected values with an actual values
- 2b. to find the **logic errors** :
  - use the VS menu's **Debug / Toggle Breakpoint F9** option
  - to set a **Breakpoint** - to pause execution at a specific line in the code and
  - and compare variable's and control's expected values with an actual values
- 3a. to avoid the **run time errors** - the **overflow errors** during program execution:
  - avoid infinite loop causing memory overflow by assigning an infinite value to a counter variable

e.g. use a compound condition to avoid an infinite loop      21      **Do While dblSales > 0 AndAlso dblSales < 150000**
- 3b. to avoid the **run time error** - the **IO error** during program execution:
  - will occur if a program statement attempts to open a text file that is **already** open
  - you should always close an output file as soon as you are finished using it to make it available for use elsewhere in the application

e.g.      **outFile.Close()**
- 3c. to avoid the **run time errors** - the **IO. file not found errors** during program execution:
  - will occur if a program statement attempts to open a text file the computer cannot locate
  - when opening the text file, you should always determine whether the text file exists

e.g.      **If IO.File.Exists("employee.txt") Then**  
              **inFile = IO.File.OpenText("employee.txt")**  
  
              ...  
              **inFile.Close()**
- 3d. to avoid the **run time errors** - the **divide by zero errors** during program execution:
  - you can use a selection structure to determine whether a variable in your division formula contains the number **0**

e.g.      **If decDenominator <> 0 Then**

## Appendix C\_Terminology marked:

->

### -> a syntax error:

- occurs when you break one of a programming language's rules
- mostly a result of typing errors that occur when entering instructions, such as typing Intger instead of Integer
- the Code Editor detects as you enter the instructions ->
  - > however, if you are not paying close attention to your computer screen, you may not notice the error

red squiggle  
green squiggle

= syntax error  
= warning

- alerts you of syntax error  
- warns you of a potential problem



- when debugging your code, always correct the **syntax errors** first because doing so will often remove any warnings

### -> the Error List window:

- part of VS debugging
- indicates if the code contains:
  - **syntax errors**
  - **warnings**
- and provides description <- by double-clicking the description, the blinking insertion point is positioned in the beginning of a problem
- and provides localization of each in the code

### -> a logic error:

- unlike syntax error, much more difficult to find because it does not trigger an error message from the Code Editor
- can occur for a variety of reasons:
  - forgetting to enter an instruction
  - entering the instructions in the wrong order
  - calculation statements are correct syntactically but incorrect mathematically

### -> the VS menu's Debug / Step Into F8 option:

- will start your application and allows you to step through your code by:
  - 1). pausing immediately before each statement is executed
  - 2). highlighting the statement
  - 3). viewing the content of the controls and variables just by hovering your mouse cursor over them
    - 1). controls & variables that appear in the highlighted line, and also
    - 2). controls & variables in already executed statements that precede the highlighted line<- before you view the contents of a control or variable, you should consider the value you expect to find
  - 4). pressing **F8** key will execute the highlighted line and pauses just before the next line and highlight it

### -> the VS menu's Debug / Toggle Breakpoint F9 option:

- useful when you have at least a slight idea where to look, since this option places a **Breakpoint** ●
  - to pause execution at a specific line in the code
  - works only with: menu's Debug / Start Debugging F5 option
  - won't work with: menu's Debug / Start Without Debugging Ctrl+F5 option
  - same like when using Step Into F8 Debugger:
  - just by mouse cursor hovering over, allowing you to view the contents of controls and variables:
    - 1). that appear in the highlighted line, and
    - 2). also in already executed statements that precede the highlighted line

<- before you view the contents of a control or variable, you should consider the value you expect to find

-> a run time error: = an error that occurs while an application is running

- e.g.1 infinite loop causing memory overflow
- e.g.2 opening a text file that is already open
- e.g.3 opening a text file the computer cannot locate
- e.g.4 dividing by 0

**CH5\_A1 - use a Loop, a Counter, and an Accumulator:**

**CH9\_F2 - Sequential Access Output Files:** [Class StreamWriter](#) - ...

**CH9\_F3 - Sequential Access Input Files:** [Class StreamReader](#) - ...

-> an overflow error:

- occurs when the value assigned to a memory location is too large for the location's data type
- similar to trying to fill a 0.2L glass with 1L of water
- in this case, the **intYears** variable already contains the largest value that can be stored in an Integer variable (2,147,483,647 - [CH3](#))
  - > therefore, when the **intYears += 1** statement attempts to increase the variable's value by 1, an overflow error occurs

**Appendix C\_Exercises****Appendix\_C\Exercise1**

**05.Commission Calculator Solution\_EXERCISE 1\_introductory** - 4x syntax error on lines **16, 22, 23, 24**

**06.New Pay Solution\_EXERCISE 2\_introductory** - 2x syntax error on lines **17**, and **18**

- 2x logic error on lines **16**, and **21**

- line **16**: 3% is not 0.3 but  $0.03 - 0.3$  is 30%

- line **21**: can't use a result in a calculation before it is counted

- logic error: `dblNewPay = dblCurrentPay + dblRATE * dblNewPay`

- fix: `dblNewPay = dblCurrentPay + dblRATE * dblCurrentPay`

**07.Hawkins Solution\_EXERCISE 3\_introductory** - 1x syntax error, missing property `.Text` on line **26**: `lblEnding.Text = decEnding.ToString("C2")`

- 1x logic error, missing statement on line **24**: `Decimal.TryParse(txtEarned.Text, decEarned)`

**08.Allenton Solution EXERCISE 4\_introductory**

- 2x logic error:

- missing statement on line **22**: `Double.TryParse(txtEarth.Text, dblEarth)`

- wrong statement on line **33**: - original: `lblMoon.Text = dblMars.ToString("N2")`

- fix: `lblMoon.Text = dblMoon.ToString("N2")`

**09.Martins Solution\_EXERCISE 5\_intermediate**

- 2x logic error:

- wrong statement on line **22**: - original: `Decimal.TryParse(txtOpening.Text, decClosePrice)`

- fix: `Decimal.TryParse(txtOpening.Text, decOpenPrice)`

- wrong operator precedence on line **25**:

- original: `decGainLoss = decClosePrice - decOpenPrice * intShares`

- fix: `decGainLoss = (decClosePrice - decOpenPrice) * intShares`

**10.Average Score Solution\_EXERCISE 6\_intermediate**

- 1x syntax error, missing method `.ToString` on line **23**:

`lblAvg.Text = dblAverage.ToString("N2")`

- 1x logic error, wrong operator precedence on line **22**:

- original: `dblAverage = dblScore1 + dblScore2 / 2`

- fix: `dblAverage = (dblScore1 + dblScore2) / 2`

**11.Beachwood Solution\_EXERCISE 7\_advanced**

- 5x syntax error on lines **14, 15, 16, 26, 32** + 1x warning on line **16**

- logic error - calculation missing on line **27**

`intTotalPounds = intRegular + intDecaf`

- logic error - wrong calculation on line **30**

- original: `decTotalPrice = decSubTotal + decSalesTax`

- fix: `decTotalPrice = decSubTotal + decSalesTax + decSHIPPING_CHARGE`

- logic error - statement missing on line **31**

`lblShipping.Text = decSHIPPING_CHARGE.ToString("C2")`

**12.Framington Solution\_EXERCISE 8\_advanced**

- 3x syntax error - method `.ToString` missing on lines **26, 27, 28**

- 2x syntax / logic error - wrong math operator: integer division operator \ without decimal should be used

- 1x logic error - statement incomplete on line **31**:

- original: `Private Sub btnExit_Click(sender As Object, e As EventArgs)`

- fix: `Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click`

## Appendix\_C\Exercise1

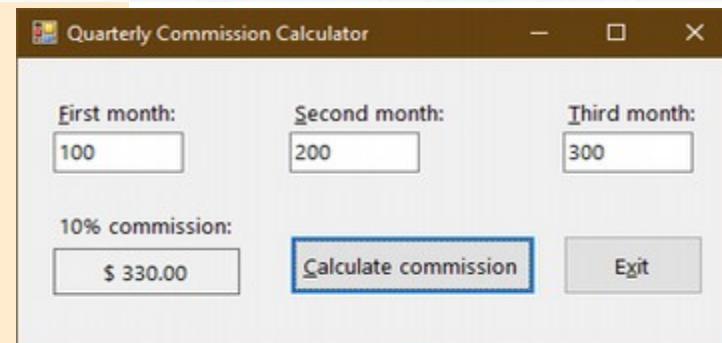
### 5.Commission Calculator Solution\_EXERCISE 1\_introductory

- 4x syntax error on lines 16, 22, 23, 24

original:

```
1  ' Name:      Commission Calculator Project
2  ' Purpose:    Display the quarterly commission.
3  ' Programmer: <your name> on <current date>
4  Option Explicit On
5  Option Strict On
6  Option Infer Off
7
8  Public Class frmMain
9      Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
10         Me.Close()
11     End Sub
12
13     Private Sub btnCalculate_Click(sender As Object, e As EventArgs) Handles btnCalculate.Click
14         ' Calculates and displays the quarterly commission.
15
16         Const decRATE As Decimal = 0.1
17         Dim decSales1 As Decimal
18         Dim decSales2 As Decimal
19         Dim decSales3 As Decimal
20         Dim decCommission As Decimal
21
22         Decimal.TryParse(txtSales1, decSales1)
23         Decimal.TryParse(txtSales2, decSales2)
24         Decimal.TryParse(txtSales3, decSales3)
25
26         decCommission = decSales1 + decSales2 + decSales3 * decRATE
27         lblCommission.Text = decCommission.ToString("C2")
28     End Sub
29 End Class
```

1. Open the Commission Calculator Solution.sln file contained in the VB2017\AppC\Commission Calculator Solution folder. Use what you learned in the appendix to debug the application.



fix:

Const decRATE As Decimal = 0.1D

<- Syntax error

Decimal.TryParse(txtSales1.Text, decSales1)  
Decimal.TryParse(txtSales2.Text, decSales2)  
Decimal.TryParse(txtSales3.Text, decSales3)

<- Syntax errors

Code		Description	Project	File	Line	Suppression State
✖ BC30512	Option Strict On	disallows implicit conversions from 'Double' to 'Decimal'.	Commission Calculator Project	Main Form.vb	16	Active
✖ BC30311	Value of type 'TextBox'	cannot be converted to 'String'.	Commission Calculator Project	Main Form.vb	22	Active
✖ BC30311	Value of type 'TextBox'	cannot be converted to 'String'.	Commission Calculator Project	Main Form.vb	23	Active
✖ BC30311	Value of type 'TextBox'	cannot be converted to 'String'.	Commission Calculator Project	Main Form.vb	24	Active

## Appendix\_C\Exercise1

### 06.New Pay Solution\_EXERCISE 2\_introductory

- 2x syntax error on lines **17**, and **18**
- 2x logic error on lines **16**, and **21**
  - line **16**: 3% is not 0.3 but  $0.03 - 0.3$  is 30%
  - line **21**: can't use a result in a calculation before it is counted
    - logic error: `dblNewPay = dblCurrentPay + dblRATE * dblNewPay`
    - fix: `dblNewPay = dblCurrentPay + dblRATE * dblCurrentPay`

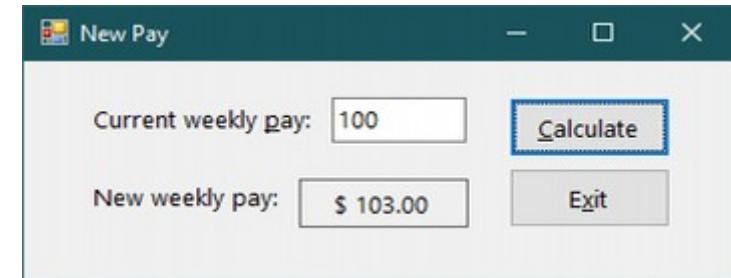
#### INTRODUCTORY

2. Open the New Pay Solution.sln file contained in the VB2017\AppC\New Pay Solution folder. Use what you learned in the appendix to debug the application.

original:

```
1  ' Name:      New Pay Project
2  ' Purpose:    Display the new weekly pay.
3  ' Programmer: <your name> on <current date>
4  Option Explicit On
5  Option Strict On
6  Option Infer Off
7
8  Public Class frmMain
9      Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
10         Me.Close()
11     End Sub
12
13     Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
14         ' Calculates and displays the new weekly pay using a raise rate of 3%.
15
16         Const dblRATE As Double = 0.3
17         Dim dblNewPay As Decimal
18         Dim dblCurrentPay As Decimal
19
20         Double.TryParse(txtCurrentPay.Text, dblCurrentPay)
21         dblNewPay = dblCurrentPay + dblRATE * dblNewPay
22         lblNewPay.Text = dblNewPay.ToString("C2")
23     End Sub
24 End Class
```

final test:



fix:

```
Const dblRATE As Double = 0.03           <- logic error
Dim dblNewPay As Double                  <- syntax error
Dim dblCurrentPay As Double              <- syntax error
```

```
dblNewPay = dblCurrentPay + dblRATE * dblCurrentPay           <- logic error
```

Error List			
Code	Description	Project	File
BC3029	Option Strict On disallows narrowing from type 'Double' to type 'Decimal' in copying the value of 'ByRef' parameter 'result' back to the matching argument.	New Pay Project	Main Form.vb
BC30512	Option Strict On disallows implicit conversions from 'Double' to 'Decimal'.	New Pay Project	Main Form.vb

## Appendix\_C\Exercise

### 07.Hawkins Solution\_EXERCISE 3\_introductory

- 1x syntax error, missing property .Text on line 26: `lblEnding.Text = decEnding.ToString("C2")`
- 1x logic error, missing statement on line 24: `Decimal.TryParse(txtEarned.Text, decEarned)`

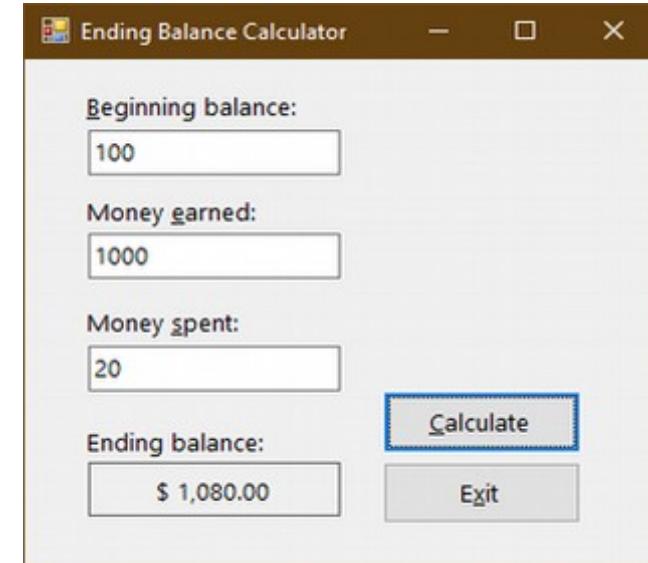
#### INTRODUCTORY

3. Open the Hawkins Solution.sln file contained in the VB2017\AppC\Hawkins Solution folder. Use what you learned in the appendix to debug the application.

original:

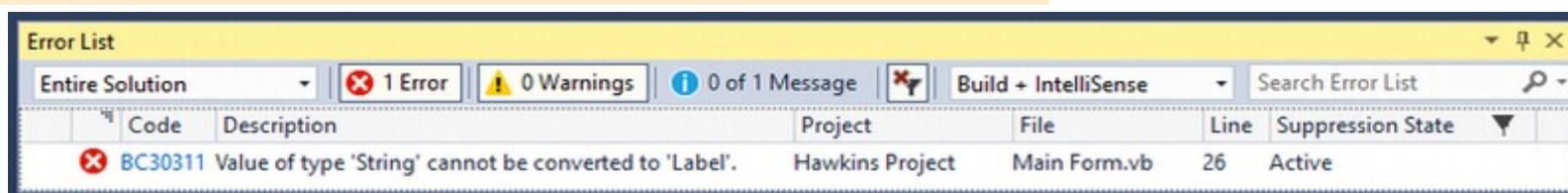
```
1  ' Name:      Hawkins Project
2  ' Purpose:    Display the ending balance.
3  ' Programmer: <your name> on <current date>
4  Option Explicit On
5  Option Strict On
6  Option Infer Off
7
8  Public Class frmMain
9      Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
10         Me.Close()
11     End Sub
12
13     Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
14         ' Calculates the ending balance based on the beginning balance,
15         ' amount earned, and amount spent.
16
17         Dim decBegin As Decimal
18         Dim decEarned As Decimal
19         Dim decSpent As Decimal
20         Dim decEnding As Decimal
21
22         Decimal.TryParse(txtBegin.Text, decBegin)
23         Decimal.TryParse(txtSpent.Text, decSpent)
24
25         decEnding = decBegin + decEarned - decSpent
26         lblEnding = decEnding.ToString("C2")
27     End Sub
28 End Class
```

final test:



fix:

```
Decimal.TryParse(txtEarned.Text, decEarned)           <- logic error - statement missing
lblEnding.Text = decEnding.ToString("C2")             <- syntax error - property .Text missing
```



## Appendix\_C\Exercise1

### 8.Allenton Solution\_EXERCISE 4\_introductory

- 2x logic error:

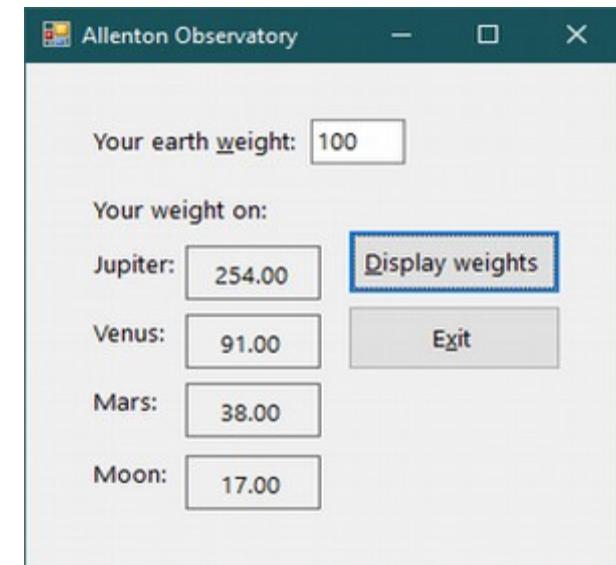
- missing statement on line 22: `Double.TryParse(txtEarth.Text, dblEarth)`
- wrong statement on line 33:
  - original: `lblMoon.Text = dblMars.ToString("N2")`
  - fix: `lblMoon.Text = dblMoon.ToString("N2")`

original:

```
1  ' Name:      Allenton Project
2  ' Purpose:    Display weight on planets and the moon.
3  ' Programmer: <your name> on <current date>
4  Option Explicit On
5  Option Strict On
6  Option Infer Off
7
8  Public Class frmMain
9      Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
10         Me.Close()
11     End Sub
12
13     Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
14         ' Calculates and displays your weight on other planets and the moon
15         ' Jupiter is 2.54 times earth weight, Venus is .91, Mars is .38, and the moon is .17.
16
17         Dim dblEarth As Double
18         Dim dblJupiter As Double
19         Dim dblVenus As Double
20         Dim dblMars As Double
21         Dim dblMoon As Double
22
23         ' Calculate weights.
24         dblJupiter = dblEarth * 2.54
25         dblVenus = dblEarth * 0.91
26         dblMars = dblEarth * 0.38
27         dblMoon = dblEarth * 0.17
28
29         ' Display weights.
30         lblJupiter.Text = dblJupiter.ToString("N2")
31         lblVenus.Text = dblVenus.ToString("N2")
32         lblMars.Text = dblMars.ToString("N2")
33         lblMoon.Text = dblMars.ToString("N2")
34     End Sub
35 End Class
```

- INTRODUCTORY
4. Open the Allenton Solution.sln file contained in the VB2017\AppC\Allenton Solution folder. Use what you learned in the appendix to debug the application.

final test:



fix:

`Double.TryParse(txtEarth.Text, dblEarth)`

<- logic error - statement missing

`lblMoon.Text = dblMoon.ToString("N2")`

<- logic error - wrong statement

## Appendix\_C\Exercise1

### 09.Martins Solution\_EXERCISE 5\_intermediate

- 2x logic error:

- wrong statement on line 22:
  - original: `Decimal.TryParse(txtOpening.Text, decClosePrice)`
  - fix: `Decimal.TryParse(txtOpening.Text, decOpenPrice)`
- wrong operator precedence on line 25:
  - original: `decGainLoss = decClosePrice - decOpenPrice * intShares`
  - fix: `decGainLoss = (decClosePrice - decOpenPrice) * intShares`

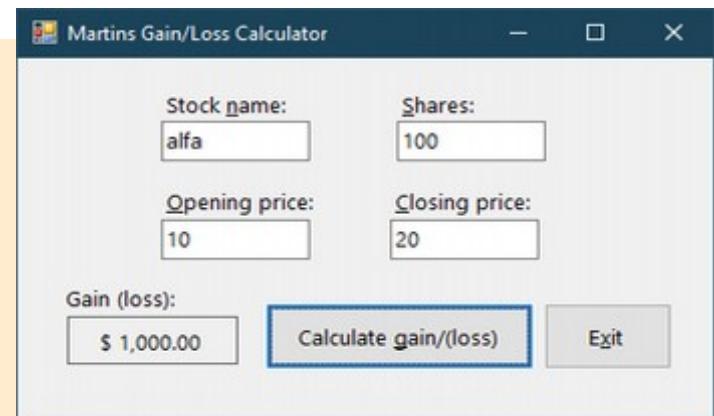
#### INTERMEDIATE

5. Open the Martins Solution.sln file contained in the VB2017\AppC\Martins Solution folder. Use what you learned in the appendix to debug the application.

original:

```
1  ' Name:      Martins Project
2  ' Purpose:    Display the gain or loss on a stock.
3  ' Programmer: <your name> on <current date>
4  Option Explicit On
5  Option Strict On
6  Option Infer Off
7
8  Public Class frmMain
9      Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
10         Me.Close()
11     End Sub
12
13     Private Sub btnGainLoss_Click(sender As Object, e As EventArgs) Handles btnGainLoss.Click
14         ' Calculates and displays the gain or loss on a stock.
15
16         Dim intShares As Integer
17         Dim decOpenPrice As Decimal
18         Dim decClosePrice As Decimal
19         Dim decGainLoss As Decimal
20
21         Integer.TryParse(txtShares.Text, intShares)
22         Decimal.TryParse(txtOpening.Text, decClosePrice)
23         Decimal.TryParse(txtClosing.Text, decClosePrice)
24
25         decGainLoss = decClosePrice - decOpenPrice * intShares
26         lblGainLoss.Text = decGainLoss.ToString("C2")
27     End Sub
28 End Class
```

final test:



fix:

`Decimal.TryParse(txtOpening.Text, decOpenPrice)`

<- logic error - wrong statement

`decGainLoss = (decClosePrice - decOpenPrice) * intShares`

<- logic error - wrong operator precedence

## Appendix\_C\Exercise1

### 10.Average Score Solution\_EXERCISE 6\_intermediate

- 1x syntax error, missing method `.ToString` on line 23:

`lblAvg.Text = dblAverage.ToString("N2")`

- 1x logic error, wrong operator precedence on line 22:

- original: `dblAverage = dblScore1 + dblScore2 / 2`

- fix: `dblAverage = (dblScore1 + dblScore2) / 2`

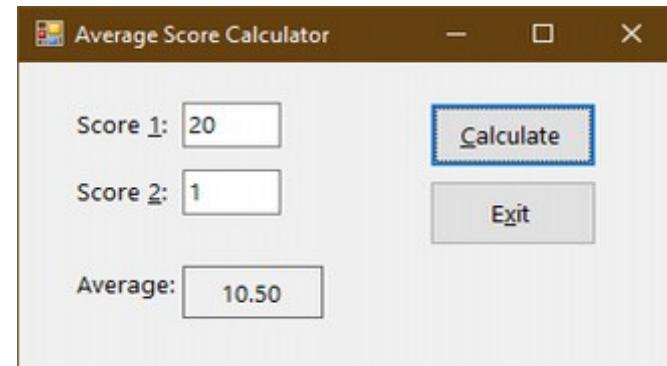
#### INTERMEDIATE

6. Open the Average Score Solution.sln file contained in the VB2017\AppC\Average Score Solution folder. Use what you learned in the appendix to debug the application.

original:

```
1  ' Name:      Average Score Project
2  ' Purpose:    Display the average score.
3  ' Programmer: <your name> on <current date>
4  Option Explicit On
5  Option Strict On
6  Option Infer Off
7
8  Public Class frmMain
9      Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
10         Me.Close()
11     End Sub
12
13     Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
14         ' Calculates and displays the average of two test scores.
15
16         Dim dblScore1 As Double
17         Dim dblScore2 As Double
18         Dim dblAverage As Double
19
20         Double.TryParse(txtScore1.Text, dblScore1)
21         Double.TryParse(txtScore2.Text, dblScore2)
22         dblAverage = dblScore1 + dblScore2 / 2
23         lblAvg.Text = dblAverage
24     End Sub
25 End Class
```

final test:



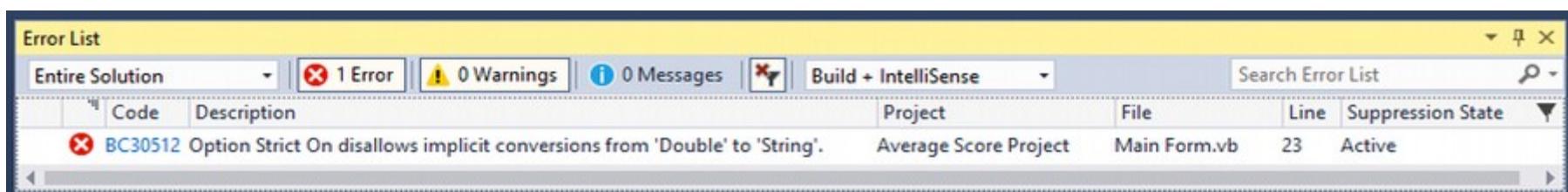
fix:

<- logic error - wrong operator precedence

`dblAverage = (dblScore1 + dblScore2) / 2`

`lblAvg.Text = dblAverage.ToString("N2")`

<- syntax error - method `.ToString` missing



## Appendix\_C\Exercise1

### 11.Beachwood Solution\_EXERCISE 7\_advanced

- 5x syntax error on lines 14, 15, 16, 26, 32 + 1x warning on lne 16

- logic error - calculation missing on line 27

```
intTotalPounds = intRegular + intDecaf
```

- logic error - wrong calculation on line 30

- original:

```
decTotalPrice = decSubTotal + decSalesTax
```

- fix:

```
decTotalPrice = decSubTotal + decSalesTax + decSHIPPING_CHARGE
```

- logic error - statement missing on line 31

```
lblShipping.Text = decSHIPPING_CHARGE.ToString("C2")
```

ADVANCED

7. Open the Beachwood Solution.sln file contained in the VB2017\AppC\Beachwood Solution folder. Use what you learned in the appendix to debug the application.

original:

```
1  ' Name:      Beachwood Project
2  ' Purpose:    Display the total pounds of coffee ordered and the total price of the order,
3  '               including sales tax and shipping.
4  ' Programmer: <your name> on <current date>
5  Option Explicit On
6  Option Strict On
7  Option Infer Off
8
9  Public Class frmMain
10 Private Sub btnCalc_Click(sender As Object, e As EventArgs) Handles btnCalc.Click
11     ' Calculates and displays the total pounds of coffee ordered
12     ' and the total price of the order, including sales tax and shipping.
13
14     Const decPRICE_PER_POUND As Decimal = 11.15
15     Const decSALES_TAX_RATE As Decimal = 0.02
16     Const decSHIPPING_CHARGE As Decimal = 5.0
17
18     Dim intRegular As Integer
19     Dim intDecaf As Integer
20     Dim intTotalPounds As Integer
21     Dim decSubTotal As Decimal
22     Dim decSalesTax As Decimal
23     Dim decTotalPrice As Decimal
24
25     Integer.TryParse(txtRegular.Text, intRegular)
26     Integer.TryParse(txtDecaf.Text, intDecaf)
27
28     decSubTotal = intTotalPounds * decPRICE_PER_POUND
29     decSalesTax = decSubTotal * decSALES_TAX_RATE
30     decTotalPrice = decSubTotal + decSalesTax
31
32     lblPounds.Text = intTotalPounds
33     lblTotalPrice.Text = decTotalPrice.ToString("C2")
34 End Sub
```

```
Const decPRICE_PER_POUND As Decimal = 11.15D
Const decSALES_TAX_RATE As Decimal = 0.02D
Const decSHIPPING_CHARGE As Decimal = 5.0D
```

<- 3x syntax error -  
- D for **Decimal** missing

```
Integer.TryParse(txtDecaf.Text, intDecaf)
intTotalPounds = intRegular + intDecaf
```

<- syntax error - end parenthesis ) missing  
<- logic error - calculation missing

```
decTotalPrice = decSubTotal + decSalesTax + decSHIPPING_CHARGE
lblShipping.Text = decSHIPPING_CHARGE.ToString("C2")
```

<- logic error -  
- statement missing

```
lblPounds.Text = intTotalPounds.ToString("N2")
```

<- syntax error - method **.ToString** missing

```

35
36     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
37         Me.Close()
38     End Sub
39 End Class

```

Code		Description	Project	File	..	▲	Suppression State
✖	BC30512	Option Strict On disallows implicit conversions from 'Double' to 'Decimal'.	Beachwood Project	Main Form.vb	14	Active	
✖	BC30512	Option Strict On disallows implicit conversions from 'Double' to 'Decimal'.	Beachwood Project	Main Form.vb	15	Active	
✖	BC30512	Option Strict On disallows implicit conversions from 'Double' to 'Decimal'.	Beachwood Project	Main Form.vb	16	Active	
⚠	BC42099	Unused local constant: 'decSHIPPING_CHARGE'.	Beachwood Project	Main Form.vb	16	Active	
✖	BC30198	')' expected.	Beachwood Project	Main Form.vb	26	Active	
✖	BC30512	Option Strict On disallows implicit conversions from 'Integer' to 'String'.	Beachwood Project	Main Form.vb	32	Active	

**my test calculation - expected values:**

intTotalPounds =	intRegular + intDecaf
decPRICE_PER_POUND	price per pound constant
decSubTotal =	intTotalPounds * decPRICE_PER_POUND
decSALES_TAX_RATE	sales tax constant - 2%
decSalesTax =	decSubTotal * decSALES_TAX_RATE
decTotalPrice =	decSubTotal + decSalesTax + decSHIPPING_CHARGE

**test 1**

$$\begin{aligned}
 & 2 + 5 = 7 \\
 & = 11.15 \\
 & 7 * 11.15 = 78.05 \\
 & = 0.02 \\
 & 78.05 * 0.02 = 1.561 \\
 & 78.05 + 1.561 + 5 = \mathbf{84.611}
 \end{aligned}$$

**test 2**

$$\begin{aligned}
 & 1 + 1 = 2 \\
 & = 11.15 \\
 & 2 * 11.15 = 22.3 \\
 & = 0.02 \\
 & 22.3 * 0.02 = 0.446 \\
 & 22.3 + 0.446 + 5 = \mathbf{27.746}
 \end{aligned}$$

## Appendix\_C\_Exercise1

### 12.Framington Solution\_EXERCISE 8\_advanced

- 3x syntax error - method `.ToString` missing on lines **26, 27, 28**
- 2x syntax / logic error - wrong math operator: integer division operator \ without decimal should be used
- 1x logic error - statement incomplete on line **31**:

- original:

```
Private Sub btnExit_Click(sender As Object, e As EventArgs)
```

- fix:

```
Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
```

ADVANCED

8. Open the Framington Solution.sln file contained in the VB2017\AppC\Framington Solution folder. Use what you learned in the appendix to debug the application.

```
1  ' Name:      Framington Project
2  ' Purpose:    Display the number of tables needed to seat guests at a party.
3  ' Programmer: <your name> on <current date>
4  Option Explicit On
5  Option Strict On
6  Option Infer Off
7
8  Public Class frmMain
9      Private Sub btnDisplay_Click(sender As Object, e As EventArgs) Handles btnDisplay.Click
10         ' Calculates the number of tables that seat 10, the number of tables that seat 5,
11         ' and the number of guests remaining.
12         ' Example: 67 guests need 6 tables of 10 and 1 table of 5, with 2 guests remaining.
13
14         Dim intGuests As Integer
15         Dim intTable10 As Integer
16         Dim intTable5 As Integer
17         Dim intRemain As Integer
18
19         Integer.TryParse(txtGuests.Text, intGuests)
20
21         intTable10 = intGuests / 10
22         intRemain = intGuests - intTable10 * 10
23         intTable5 = intRemain / 5
24         intRemain = intRemain - intTable5 * 5
25
26         lblTable10.Text = intTable10
27         lblTable5.Text = intTable5
28         lblRemain.Text = intRemain
29     End Sub
30
31     Private Sub btnExit_Click(sender As Object, e As EventArgs)
32         Me.Close()
33     End Sub
34 End Class
```

intTable10 = intGuests \ 10      <- syntax / logic error - wrong math operator  
intRemain = intGuests Mod 10      <- no error, just suggestion  
intTable5 = intRemain \ 5      <- syntax / logic error - wrong math operator  
intRemain = intRemain Mod 5      <- no error, just suggestion

lblTable10.Text = intTable10.ToString  
lblTable5.Text = intTable5.ToString  
lblRemain.Text = intRemain.ToString      <- 3x syntax error - method `.ToString` missing

Private Sub btnExit\_Click(sender As Object, e As EventArgs) Handles btnExit.Click      <- logic error - statement incomplete

Error List					Build + IntelliSense	Search Error List
	Code	Description	Project	File	Line ▲	Suppression State
✖	BC30512	Option Strict On disallows implicit conversions from 'Double' to 'Integer'.	Framington Project	Main Fom.vb	21	Active
✖	BC30512	Option Strict On disallows implicit conversions from 'Double' to 'Integer'.	Framington Project	Main Fom.vb	23	Active
✖	BC30512	Option Strict On disallows implicit conversions from 'Integer' to 'String'.	Framington Project	Main Fom.vb	26	Active
✖	BC30512	Option Strict On disallows implicit conversions from 'Integer' to 'String'.	Framington Project	Main Fom.vb	27	Active
✖	BC30512	Option Strict On disallows implicit conversions from 'Integer' to 'String'.	Framington Project	Main Fom.vb	28	Active

tests after fix:

The application window titled "Framington Parties" contains the following controls and data:

- Input field: "Number of guests:" with value "67".
- Buttons: "Display" (highlighted in blue) and "Exit".
- Output fields:
  - "Tables for 10:" with value "6"
  - "Tables for 5:" with value "1"
  - "Guests remaining:" with value "2"

The application window titled "Framington Parties" contains the following controls and data:

- Input field: "Number of guests:" with value "4".
- Buttons: "Display" (highlighted in blue) and "Exit".
- Output fields:
  - "Tables for 10:" with value "0"
  - "Tables for 5:" with value "0"
  - "Guests remaining:" with value "4"

THE END, KONEC, FINE, FINITO, BASTA.

## Appendix F

# SDI, MDI, TDI - multiple forms & dialog boxes

SDI app - main form & splash screen, main form & dialog box, 3x main form

MDI app - 2x primary form = Parent & Child; quasi TDI app - main form & TabControl tool

Appendix F\_00 - SDI, MDI, and TDI applications - basic info & e.g.

Appendix F\_01 - SDI application - main form & splash screen form - e.g. with **01.Country Solution**

Appendix F\_02 - SDI application - main form & dialog box - info about **Tool dialog box & Custom dialog box** - e.g. with **Tool dialog boxes: 02.Font Color Solution**

Appendix F\_03 - SDI application - 3x main/primary form - how to: create them & step between them & share data between them & close entire solution  
e.g. with **03.Zappet Solution**

Appendix F\_04 - MDI application - 2x main/primary forms separated into: **Parent** Form/window/GUI & its **Child** Forms/windows/GUIs  
e.g. with **04.JotPad Solution** (VS2022 & .NET 6.0)

Appendix F\_05 - quasi TDI application - main form & **TabControl** tool step by step, e.g. with **05.Currency Converter Solution** (VS2022 & .NET 6.0)

### Appendix F\_Key Terms

how to create:	1). open <b>Add New Item</b> dialog box: menu <b>Project / Add New Item...</b> <b>Ctrl+Shift+A</b>
Primary form:	2). on the left pane: <b>Installed / Common Items / Windows Forms</b>
Custom Dialog Box:	3a). in the middle: <b>Windows Form</b> with a default name: <b>Form2.vb</b>
Splash screen:	3b). in the middle: <b>Dialog</b> with a default name: <b>Dialog1.vb</b>
	3c). in the middle: <b>Splash Screen</b> with a default name: <b>SplashScreen1.vb</b>

### share data between forms:

1). declare Friend variable in form:

```
Friend FriendVariableName As dataType = expression
```

2). refer to it from another form:

```
reference = OriginFormName.FriendVariableName
```

form's property Title bar text

```
formName.Text = TitleBarText
```

form's method Close

```
formName/Me.Close()
```

form's method Show

```
formName/Me.Show()
```

form's method Hide

```
formName/Me.Hide()
```

## Appendix F\_00 - SDI, MDI, and TDI applications - basic info & e.g.

- applications can be classified as: **SDI**, **MDI**, **TDI**

**1). SDI** = single-document interface

- each window of the application holds a single document
- if you need to open another document, you must open a new window

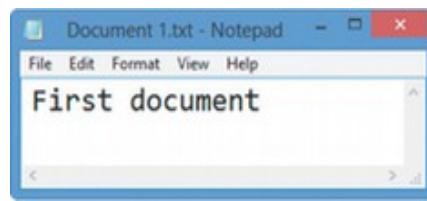
e.g.0 all applications you created in the **CH01-CH13** contain 1 form **frmMain**

e.g.1 MS Notepad **Figure F-00a**

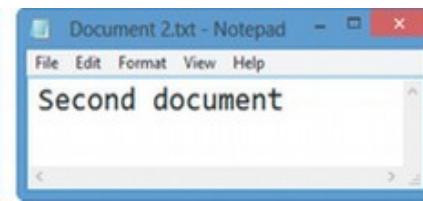
e.g.2 **01.Country Solution** - main form & splash screen form - **Appendix F\_01** **Figure F-06a** & **Figure F-06b**

e.g.3 **02.Font Color Solution** - main form & dialog box tool - **Appendix F\_02** **Figure F-07a** & **Figure F-07b**

e.g.4 **03.Zappet Solution** - 3x main/primary form - **Appendix F\_03** **Figure F-15a** & **Figure F-15b** & **Figure F-15c**



**Figure F-00a**



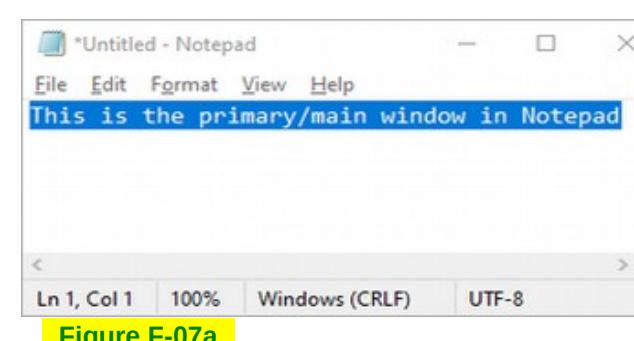
**Second document**



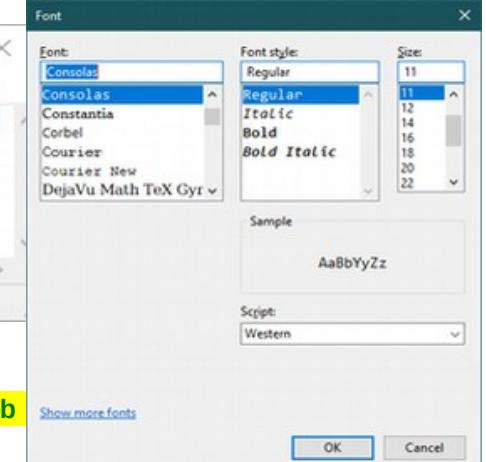
**Figure F-06a**



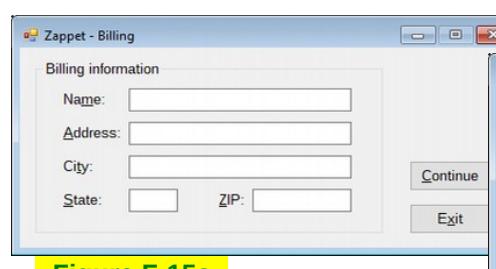
**Figure F-06b**



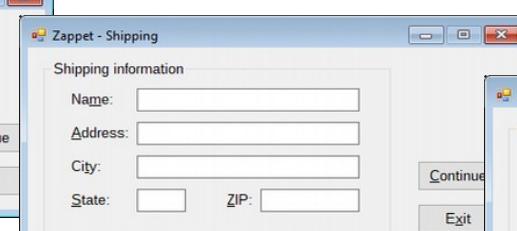
**Figure F-07a**



**Figure F-07b**



**Figure F-15a**



**Figure F-15b**



**Figure F-15c**

## 2). MDI = multiple-document interface

- consists of a parent window and 1 or more child windows that are contained within the parent window
- each document appears in a separate child window within the parent window, and each has access to the parent window's menu bar and toolbar
- not as popular due to confusing management of child windows
- most provides a menu named Window or View, that lists the open documents

e.g.1 some earlier versions of MS Excel, such as Excel 2007

Figure F-00b

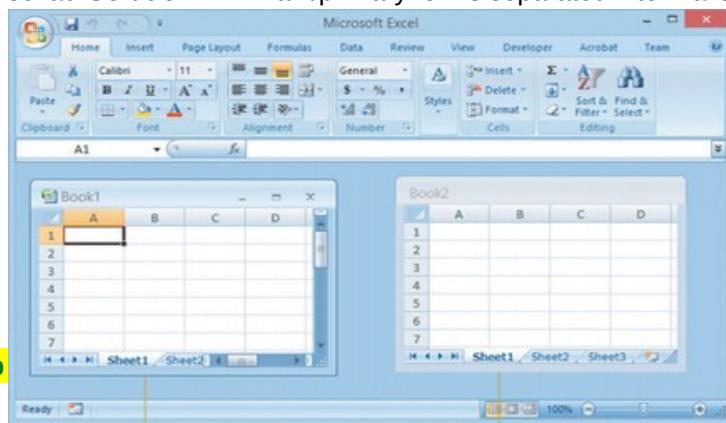


Figure F-00b

e.g.2 04.JotPad Solution - 2x main/primary forms separated into: Parent Form/window/GUI & its Child Forms/windows/GUIs - Appendix F\_04

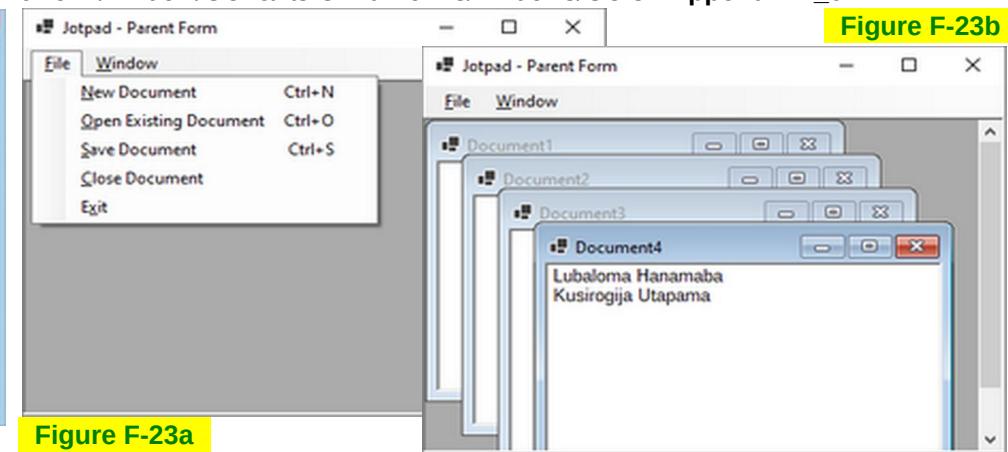


Figure F-23b

## 3). TDI = tabbed-document interface

- offshoot (odnož) of MDI
- like an MDI, allows more than 1 child window to be open in a parent window
- unlike an MDI, uses tabs to show the windows that are open in an application

e.g.1 Visual Studio [Figure F-00c](#)

- the creation of a TDI application is beyond the scope of this book

e.g.2 05.Currency Converter Solution - quasi TDI app - main form & TabControl tool - Appendix F\_05

Figure F-29a & Figure F-29a

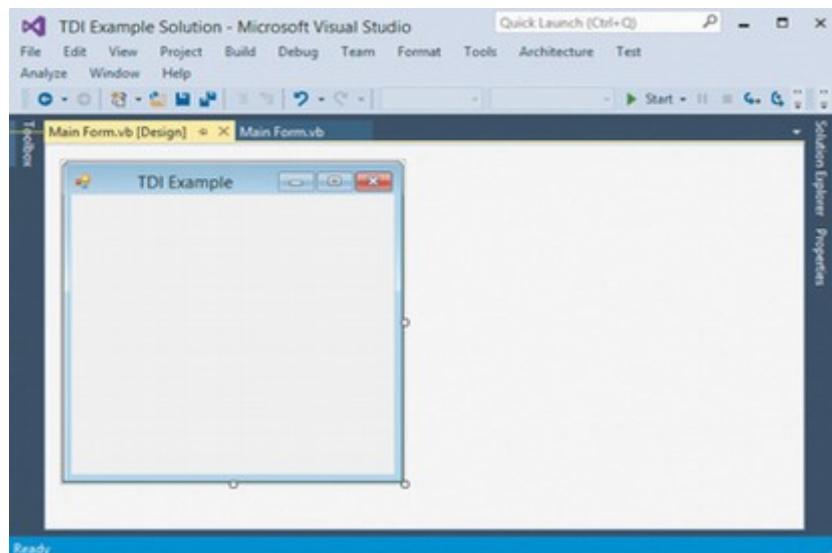


Figure F-29a

Figure F-00c



Figure F-29b



## Appendix F\_01 - SDI application - main form & splash screen form - e.g. with 01.Country Solution

- the simplest **multiple-form application** is one composed of a **main form** and a **splash screen**
- a **splash screen** - as you learned in **CH1\_A15 - Splash Screen** - additional topic from **Appendix B** - template & code a splash screen is the 1st image that appears when an application is started
- is used to introduce the application and to hold the user's attention while the application is being read into the computer's internal memory

- to add a **splash screen** to the **Country Charm Inn** application - e.g. with **01.Country Solution**

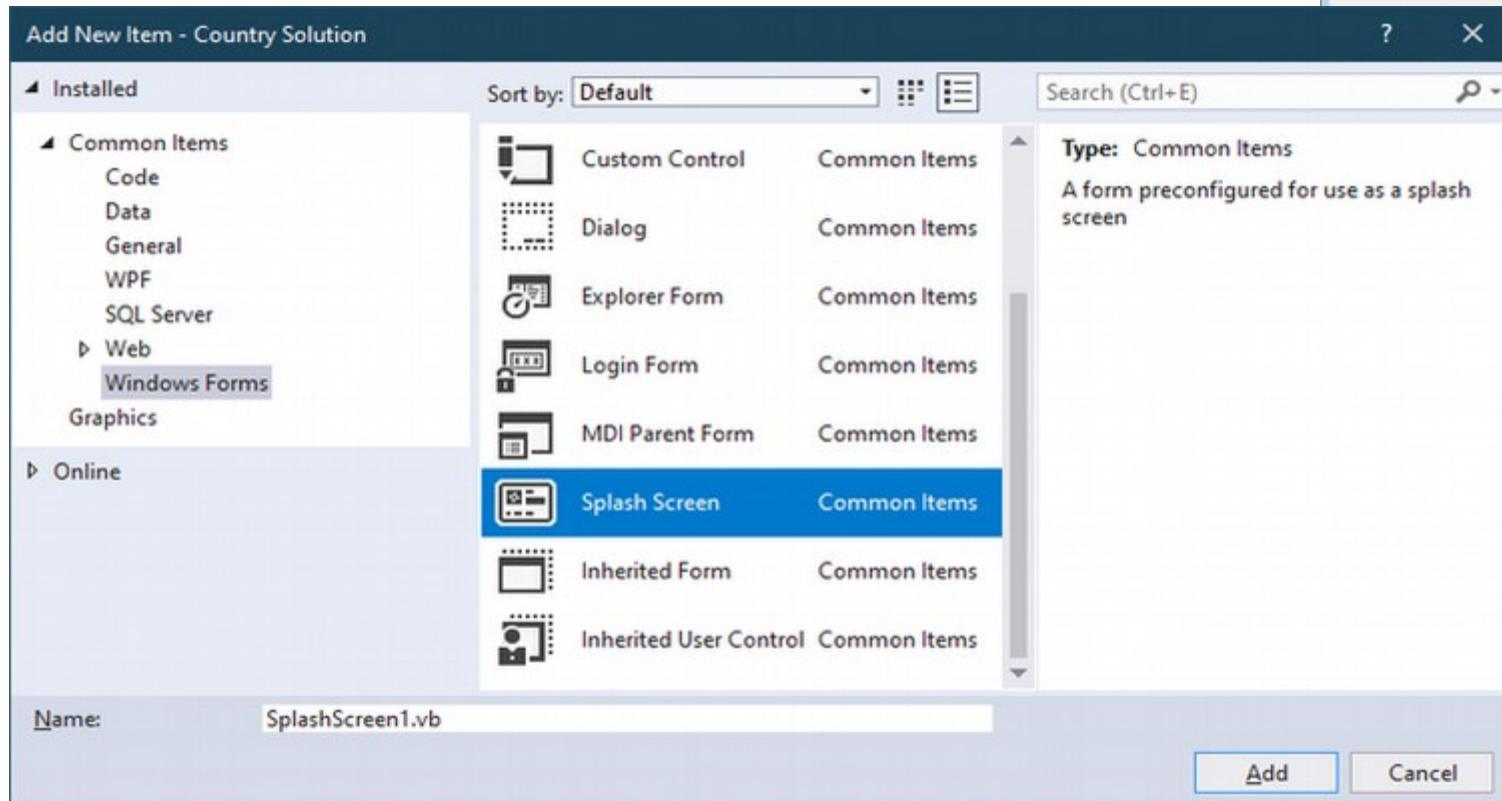
-> create the: ...VB2017\Appendix\_F\Exercise\01.Country Solution\Country Solution.sln  
-> create GUI as shown in **Figure F-01**

**step 1** - you will add a splash screen to the application

- > on menu bar click: **Project / Add Windows Form...** to open **Add New Item dialog box**
- > on the left pane choose: **Installed / Common Items / Windows Forms**
- > in the middle locate: **Splash Screen** named as: **SplashScreen1.vb**



**Figure F-01**



**Figure F-02**

- > rename the default name: **SplashScreen1.vb** into: **Splash Form.vb**
- > click the **Add** button to add the new item into your application

step 2 <- notice the new form appears in the VS

Figure F-03

- the form contains **5** controls:
  - **MainLayoutPanel** filling the interior of the form
  - **DetailsLayoutPanel**
- containing label: **Application Title**
- containing 2 labels:
  - **Version...**
  - **Copyright**

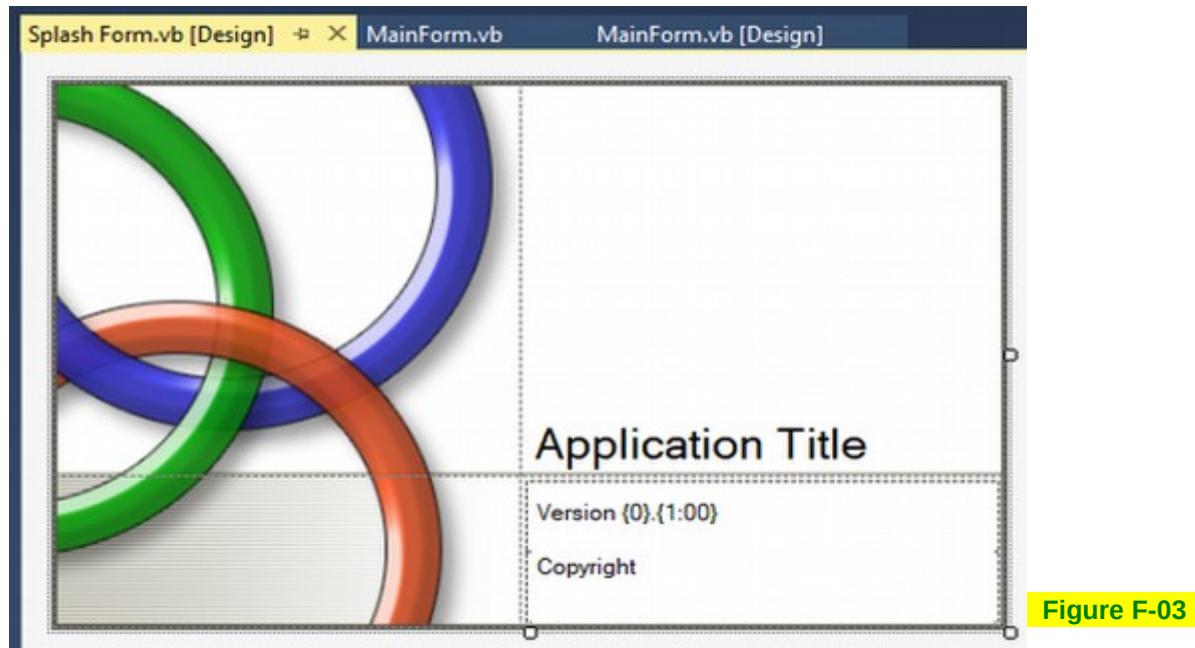


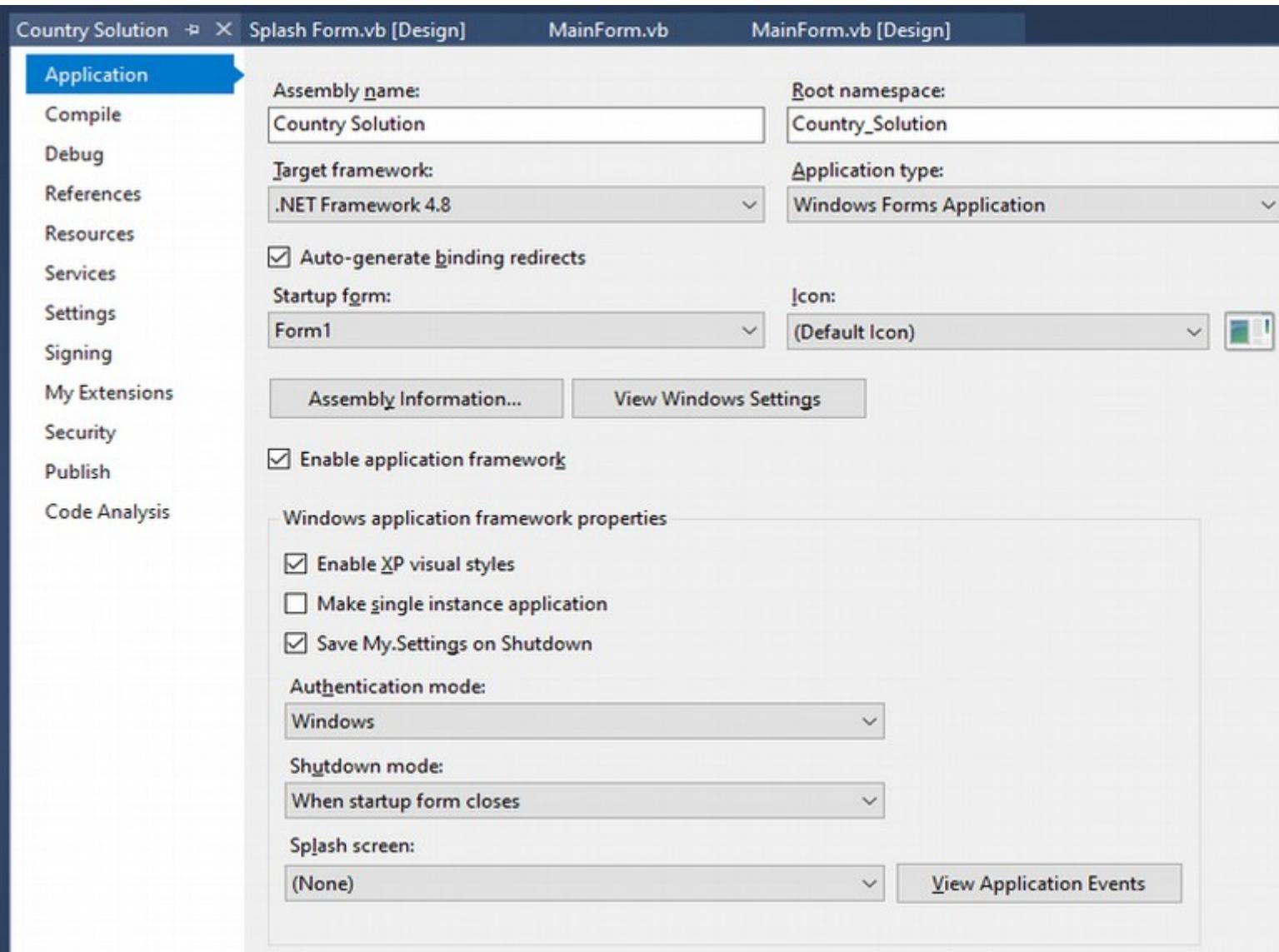
Figure F-03

- you can modify the form by adding controls to it or deleting controls from it
- you can also use **Properties** window to change the properties of the form and its controls

-> in the **Properties** window, change the default (Name): **Splash\_Form** to: **frmSplash**

**step 3.1**

- in **CH1** you learned that the computer automatically displays an application's startup form each time the application is started
  - it will also automatically display an application's splash screen as long as you specify the splash screen's name in the **Project Designer window**
  - when the application is started, the splash screen will appear first and after a few seconds will automatically disappear exchanged by the startup form
- you will open the **Project Designer window** and specify the splash screen name so it will start automatically  
-> in the **Solution Explorer** window, open the **My Project** (either by doubleclicking or Rclick/Open)



-> 1. change: **Startup form**  
- from default: **Form1**  
- to: **frmMain**

-> 4. click button  
**Assembly Information...**  
**step 3.2**

-> 2. change: **Shutdown mode**  
- from: **When startup form closes**  
- to: **When last form closes**

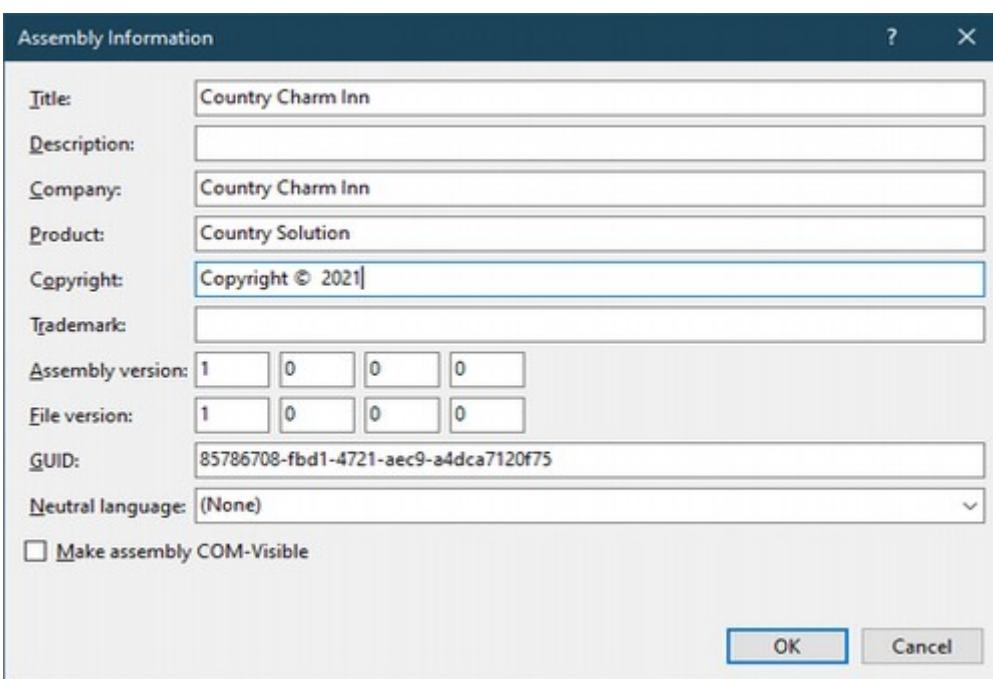
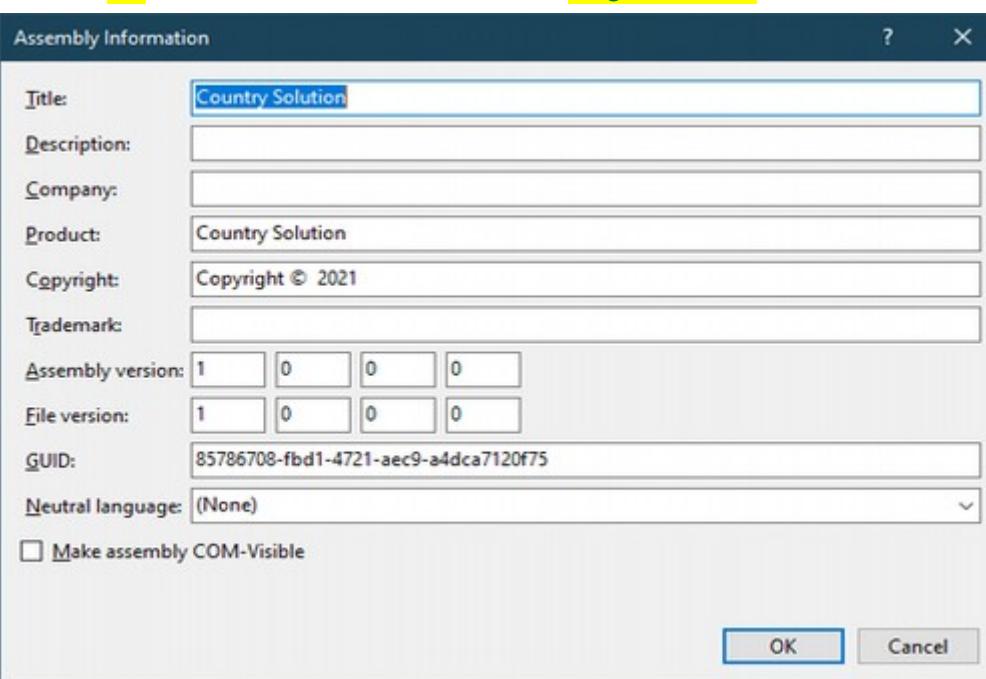
-> 3. change: **Splash screen**  
- from default: **(None)**  
- to: **frmSplash**

Figure F-04

**step 3.2** -> click button **Assembly Information...** to open the **Assembly Information dialog box**

**Figure F-05a**

-> fill the information as shown in: **Figure F-05b**



**Figure F-05a**

**Figure F-05b**

-> click the button **OK** to close the **Assembly Information dialog box** and save the solution

**step 4** -> start and test the application

<- notice the splash screen appears first

<- notice after a few seconds have elapsed, the splash screen disappears and the startup form fromMain appears

-> close the application and the solution



**Figure F-06a**



**Figure F-06b**

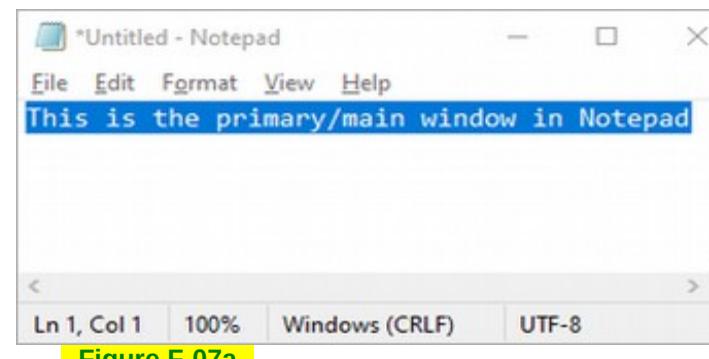


Figure F-07a

- most Windows applications consist of 1 or more:

**1). main window** = primary window

- used for primary viewing and editing of your application's data

e.g. **Figure F-07a** Notepad application's **main window** allows you to view and edit documents

**2). dialog box** = secondary window

- support and supplement a user's activities in a main window

e.g. **Figure F-07b** **Font dialog box** can be used to specify the font of the text selected in the main window

- dialog boxes are modal = they remain on the screen until the user closes them

- while a modal dialog box is on the screen, no input from the keyboard or mouse can occur in the application's primary window

- however, you can access other applications

- you close a modal dialog box by selecting either OK button or its Cancel button, or by clicking the Close button on its title bar

- can be created by either using: **2a). Tool**, or **2b). Template**

**2a). Tool dialog box** - using predefined tools located in: **Toolbox / Dialogs / ColorDialog, FolderBrowserDialog, FontDialog, OpenFileDialog, SaveFileDialog**

- VB & VS provides several tools for creating commonly used **dialog boxes**

- located in VS: **Toolbox / Dialogs /**

**ColorDialog**

- displays available colors along with controls that enable the user to define custom colors.

**FolderBrowserDialog**

- displays a dialog box that prompts the user to select a folder.

**FontDialog**

- displays a dialog box that prompts the user to choose a font installed on the local machine.

**OpenFileDialog**

- displays a dialog box that prompts the user to open a file.

**SaveFileDialog**

- displays a dialog box that prompts the user to select a location for saving a file.

- when dragged to a form, its instantiated control is placed in the component tray

e.g.

Figure F-10

- to show/open the dialog box during run time method syntax:

`dialogControlName.ShowDialog()`

- to use in a code: **1).** set the dialog box's initial value

**2).** open the dialog box using method

**3).** assign the dialog box choices to some control's property

e.g.

```
dlgColor.Color = lblMessage.ForeColor  
dlgColor.ShowDialog()  
lblMessage.ForeColor = dlgColor.Color
```

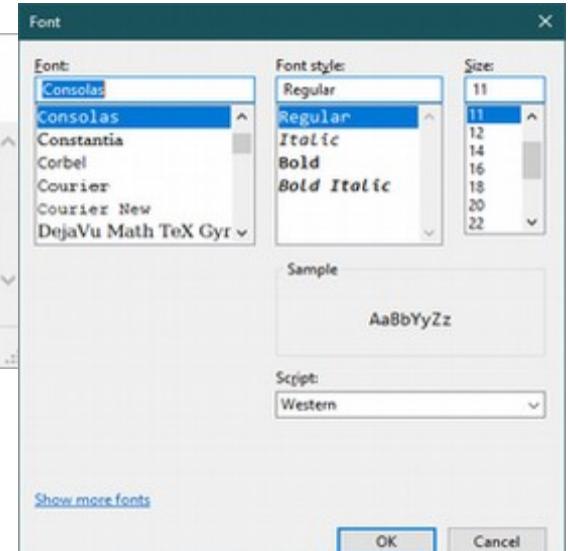


Figure F-07b

**2b). Custom dialog box** - using predefined template

- located in VS: 1). open **Add New Item** dialog box by either:
  - 1a). menu bar: **Project / Add Windows Form...**
  - 1b). menu bar: **Project / Add New Item... Ctrl+Shift+A**

2). on the left pane: **Installed / Common Items / Windows Forms** **Figure F-08a**

3). in the middle choose either: 3a). **Windows Form** template with a default name: **Form2.vb**

- creates blank Windows Form, what need to be configured as a dialog box:

1). window **Properties: FormBorderStyle = FixedDialog**

- it draws a fixed border around the form
- it removes the Control menu box from the form's title bar

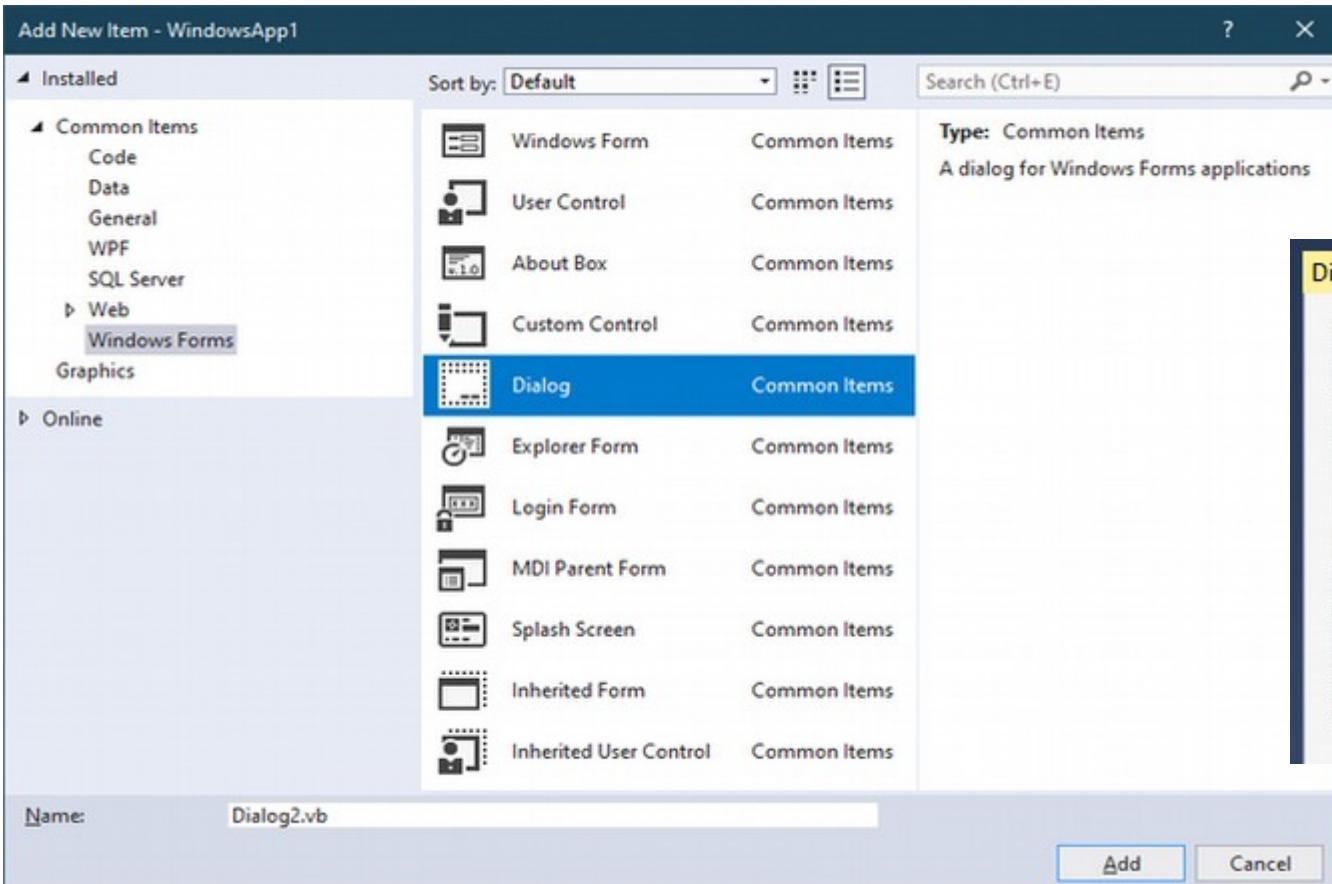
2). window **Properties: MaximizeBox = False**

3). window **Properties: MinimizeBox = False**

- the Windows standard is that dialog boxes contain only a Close button
- and, in some cases, a Help button

3b). **Dialog** template with a default name: **Dialog1.vb** **Figure F-08b**

- creates a Windows Form that is already configured for use as a dialog box



**Figure F-08a**

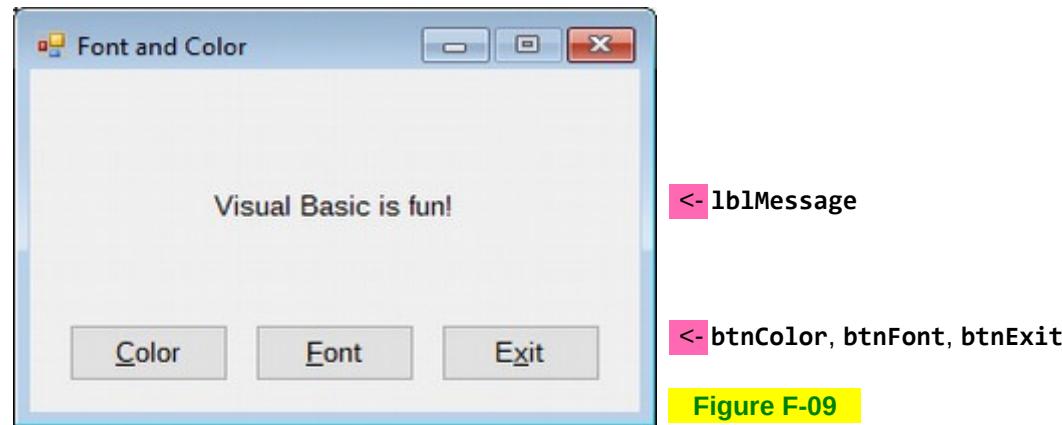
**Figure F-08b**

- to add **Font** and **Color** dialog boxes tools to the **Font and Color** application - e.g. with **02.Font Color Solution**

-> create the: ...VB2017\Appendix\_F\Exercise\02.Font Color Solution\Font Color Solution.sln

-> create GUI as shown in

**Figure F-09**



**Figure F-09**

**step 1.0** - you will add/drag **FontDialog** tool and **ColorDialog** tool to the form and give them a more meaningful name starting: **dlg...**

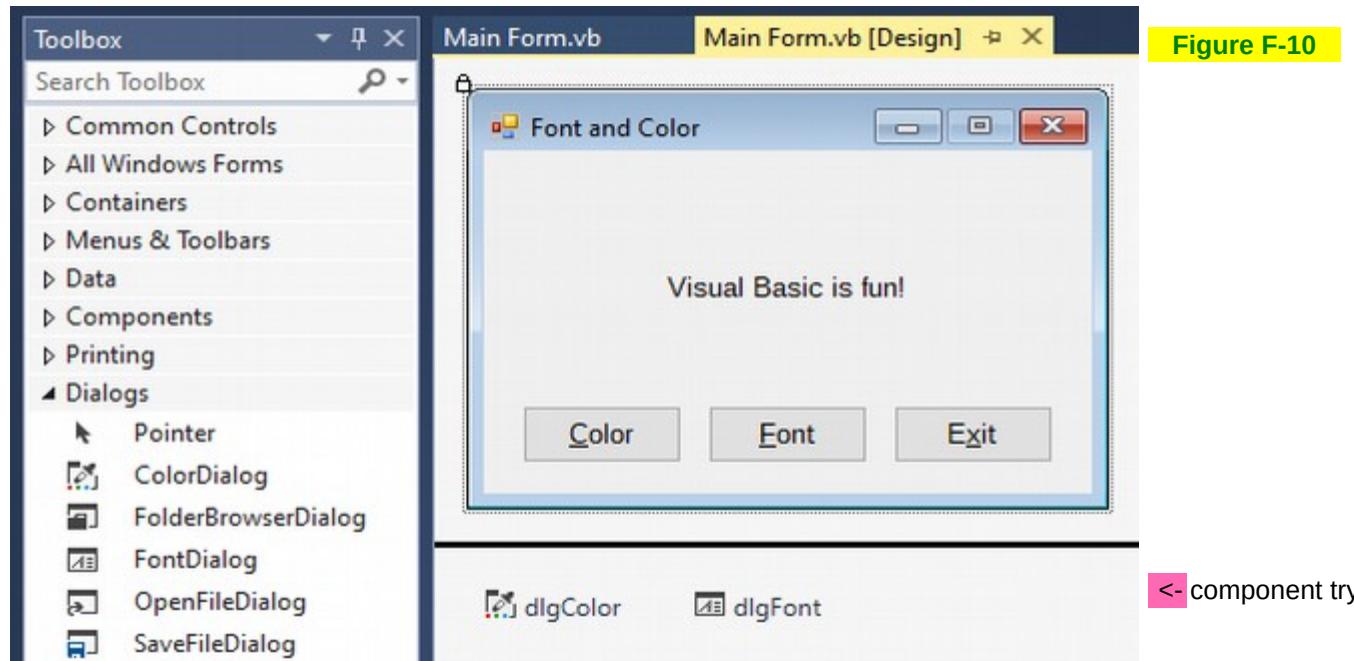
**Figure F-10**

**step 1.1** -> window **Toolbox / Dialogs** / -> drag **FontDialog** tool anywhere to the form or to the component try

-> in **Properties** window rename default: **FontDialog1** to: **dlgFont**

**step 1.2** -> window **Toolbox / Dialogs** / -> drag **ColorDialog** tool anywhere to the form or to the component try

-> in **Properties** window rename default: **ColorDialog1** to: **dlgColor**



**step 2.0** - the button **Font** in GUI should allow the user to change the **Font** property values for the **lblMessage** control  
 -> open/create the code template for the **btnFont\_Click** procedure

**step 2.1** -> as #1, to the **dlgFont** control's **Font** property, assign an initial value - **lblMessage** control's **Font** property

**step 2.2** -> as #2, use a dialog box's method to open the dialog box

**step 2.3** -> as #3, to the **lblMessage** control's **Font** property, assign **dlgFont** control's **Font** property selected in a dialog box tool originally named **FontDialog**

```

13     Private Sub btnFont_Click(sender As Object, e As EventArgs) Handles btnFont.Click
14         dlgFont.Font = lblMessage.Font
15         dlgFont.ShowDialog()
16         lblMessage.Font = dlgFont.Font
    
```

<- sets the dialog box's initial value  
 <- opens the dialog box using method **ShowDialog()**  
 <- assigns the dialog box choices to the label's **Font** property

**step 3.0** - the button **Color** in GUI should allow the user to change the **ForeColor** property value for the **lblMessage** control

-> open/create the code template for the **btnColor\_Click** procedure

**step 3.1** -> as #1, to the **dlgColor** control's **Color** property, assign an initial value - **lblMessage** control's **ForeColor** property

**step 3.2** -> as #2, use a dialog box's method to open the dialog box

**step 3.3** -> as #3, to **lblMessage** control's **ForeColor** property, assign **dlgColor** control's **Color** property selected in a dialog box tool originally named **ColorDialog**

```

19     Private Sub btnColor_Click(sender As Object, e As EventArgs) Handles btnColor.Click
20         dlgColor.Color = lblMessage.ForeColor
21         dlgColor.ShowDialog()
22         lblMessage.ForeColor = dlgColor.Color
    
```

<- sets the dialog box's initial value  
 <- opens the dialog box using method **ShowDialog()**  
 <- assigns the dialog box choice to the label's **ForeColor** property

```

1  ' Name:      Font and Color app.
2  ' Purpose:   Learn how the Dialog boxes works.
3  ' Programmer: me on just now.
4  Option Explicit On
5  Option Strict On
6  Option Infer Off
7
8  Public Class frmMain
9      Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
10         Me.Close()
11     End Sub
12
13     Private Sub btnFont_Click(sender As Object, e As EventArgs) Handles btnFont.Click
14         dlgFont.Font = lblMessage.Font
15         dlgFont.ShowDialog()
16         lblMessage.Font = dlgFont.Font
    
```

<- sets the dialog box's initial value  
 <- opens the dialog box using method **ShowDialog()**  
 <- assigns the dialog box choices to the label's **Font** property

```

17     End Sub
18
19     Private Sub btnColor_Click(sender As Object, e As EventArgs) Handles btnColor.Click
20         dlgColor.Color = lblMessage.ForeColor
21         dlgColor.ShowDialog()
22         lblMessage.ForeColor = dlgColor.Color
    
```

<- sets the dialog box's initial value  
 <- opens the dialog box using method **ShowDialog()**  
 <- assigns the dialog box choice to the label's **ForeColor** property

```

23     End Sub
24 End Class
    
```

step 4 -> save and start & test your application

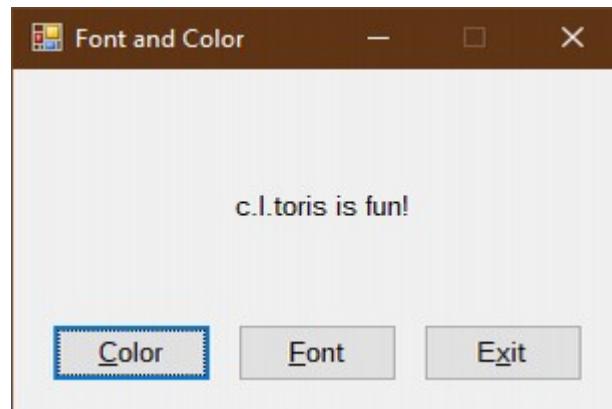


Figure F-11a

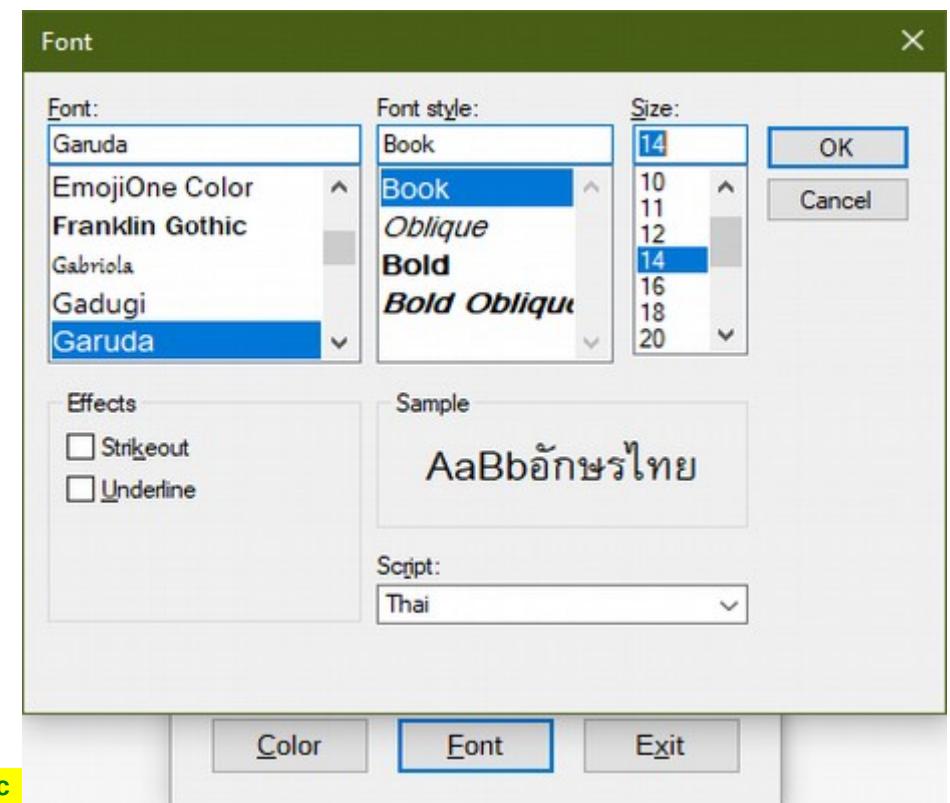


Figure F-11b

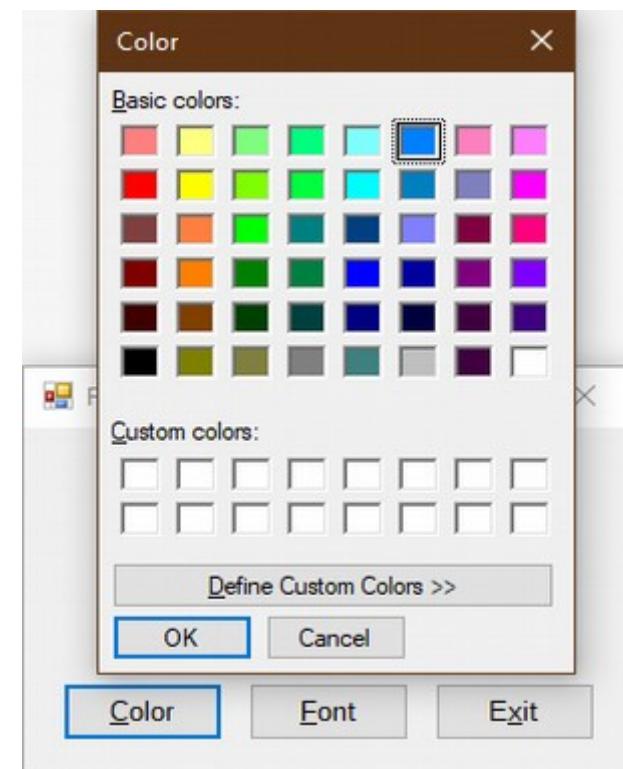


Figure F-11c

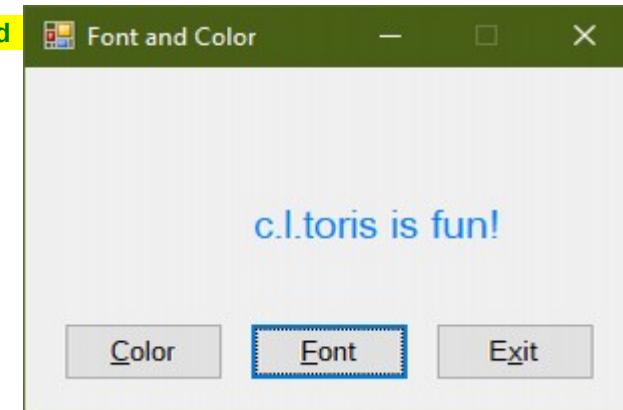


Figure F-11d

- some Windows applications contain more than one primary form

e.g. Zappet application has **3 primary forms**: `frmBilling`

`frmShipping`

`frmReview`

- the #1 form gets the customer's billing information

- the #2 form gets the customer's shipping information

- the #3 form displays the billing and shipping info for the customer to review

index:

**step 0**

**step 1**

- create Windows Forms app, which will automatically create the first main/primary **Windows Form** named **Form1.vb**

- add another **Windows Form templates** and personalize them

> 1). open **Add New Item** dialog box by either: 1a). menu bar: **Project / Add Windows Form...**

1b). menu bar: **Project / Add New Item... Ctrl+Shift+A**

> 2). on the left pane:

> 3). in the middle choose:

**Installed / Common Items / Windows Forms**

**Figure F-08a**

**Windows Form** template with a default name: **Form2.vb**

- define the startup form

> open **My Project** and set: **Startup form** to: **frmBilling** **Figure F-14**

- create GUI - controls are used only within the form, so in each form can be named the same

- code the forms using new syntaxes

### 1). sharing data between forms within current solution:

1.1). in one form, save data into a variable declared in form's declaration section, using the keyword: **Friend**

- variables declared using the **Friend keyword** are recognized by any form within the current solution

declaration syntax: `Friend FriendVariableName As dataType = expression`

1.2). in another form, refer to a **Friend** variable using the name of the form in which it had been declared, and variable's name

assignment syntax: `reference = OriginFormName.FriendVariableName`

### 2). stepping between forms within current solution:

<- notice when you **Hide** the current form, it still resides in RAM, just not visible

2.1). first you hide the current form, using the form's method **Hide()**

form's method: Hide syntax: `Me.Hide()`

**Me** = keyword

- provides a way to refer to the current instance of a class or structure = the instance in which the code is running

2.2). then you open-show the next form, using the form's method **Show()**

form's method: Show syntax: `formName.Show()`

### 3). when one of the forms is closing, close all the forms within current solution:

<- notice: as mentioned above - when you **Hide** any current form, it still resides in RAM, just not visible ->

-> therefore to close the entire solution, you must specify each **previously hidden forms** in current form's **FormClosing** event procedure

- form's **btnExit\_Click** event procedure with a method: `Me.Close()` will close only the current form

- to close all previously hidden forms:

3.1). open/create the current form's **FormClosing** event procedure

- as you learned in **CH7**, a form's **FormClosing** event occurs when a form is about to be closed

3.2). use a form's method **Close()**

form's method: Close syntax: `formName.Close()`

- to add another primary forms to the application & **create GUI & code** the forms - e.g. with **03.Zappet Solution**

**step 0** - you will create Windows Forms app, which will automatically create the first main/primary **Windows Form** named **Form1.vb**  
-> create the: ...VB2017\Appendix\_F\Exercise\03.Zappet Solution\Zappet Solution.sln

**step 1** - you will add another **2** main forms to the application & rename them & set Startup form & personalize all **3** main forms

**1.0** - add second main form

-> open **Add New Item** dialog box by either: a). menu bar: **Project / Add Windows Form...**

b). menu bar: **Project / Add New Item... Ctrl+Shift+A**

**Installed / Common Items / Windows Forms**

**Figure F-08a**

**1.1** -> on the left pane choose:

**Windows Form** template with a default name: **Form2.vb**

**1.2** -> in the middle choose:

**2.0** - add third main form

-> repeat steps **1.1** -> **1.3**, only the default form template's name will be: **Form3.vb**

<- notice in **Solution Explorer** window all **3 main forms** with default names

**Figure F-13a**

**3.0** -> rename: **Form1.vb** to: **Billing Form.vb** & change **(Name)** Property to: **frmBilling**  
**Form2.vb** to: **Shipping Form.vb** & change **(Name)** Property to: **frmShipping**  
**Form3.vb** to: **Review Form.vb** & change **(Name)** Property to: **frmReview**

**Figure F-13b**

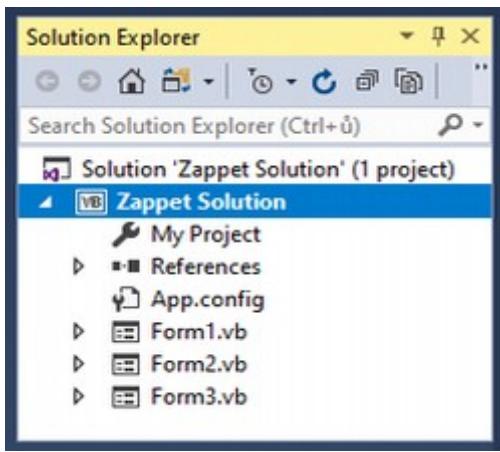
**Figure F-13b**

**Figure F-13b**

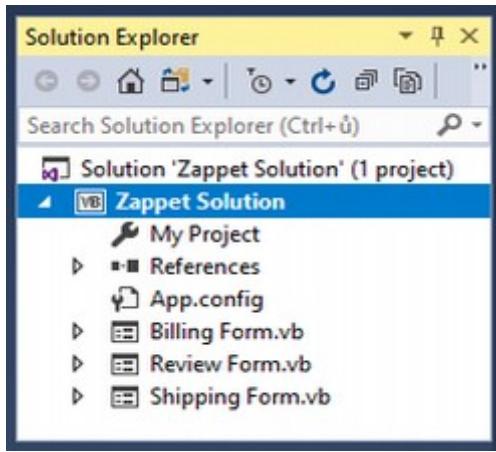
**4.0** - set the starting form

-> open **My Project** and set: **Startup form** to: **frmBilling**

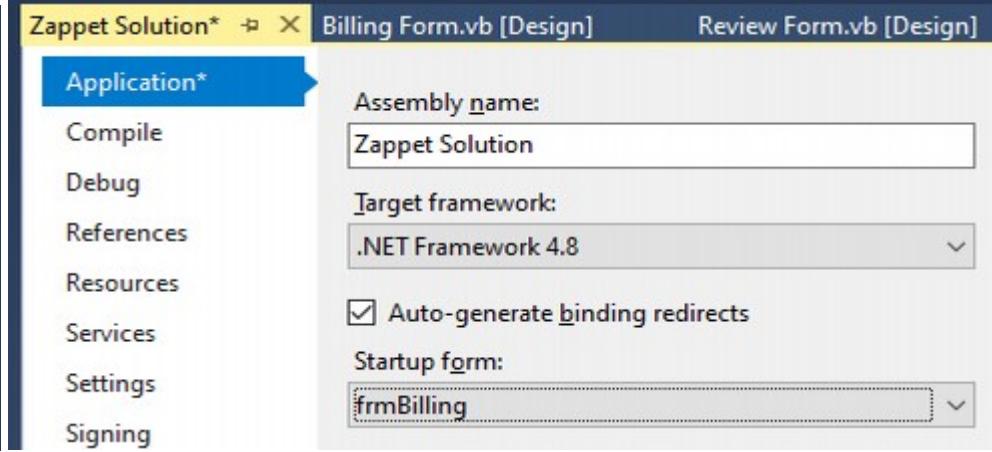
**Figure F-14**



**Figure F-13a**



**Figure F-13b**



**Figure F-14**

**5.0** -> like always, set each of the forms Properties:

**Font = Liberation Sans, 9.75px**

**FormBorderStyle = FixedSingle**

**MaximizeBox = False**

**StartPosition = CenterScreen**

**Text = for: frmBilling = Zappet - Billing**

**frmShipping = Zappet - Shipping**

**frmReview = Zappet - Billing/Shipping Review**

**step 2.0** - you will create GUI for each of the main forms

**Figure F-15a** & **Figure F-15b** & **Figure F-15c**

**step 2.1** -> frmBilling

**Figure F-15a**

Zappet - Billing

Billing information

Name:

Address:

City:

State:  ZIP:

Continue

Exit

<- txtName  
<- txtAddress  
<- txtCity  
<- txtState  
<- txtZip

<- btnContinue  
<- btnExit

**step 2.2** -> frmShipping

**Figure F-15b**

Zappet - Shipping

Shipping information

Name:

Address:

City:

State:  ZIP:

Continue

Exit

<- txtName  
<- txtAddress  
<- txtCity  
<- txtState  
<- txtZip

<- btnContinue  
<- btnExit

**step 2.3** -> frmReview

**Figure F-15c**

Zappet - Billing/Shipping Review

Billing information

Name:

Address:

City:

State:  ZIP:

Shipping information

Name:

Address:

City:

State:  ZIP:

Exit

<- lblBillName  
<- lblBillAddress  
<- lblBillCity  
<- lblBillState

<- lblBillZip

<- lblShipName  
<- lblShipAddress  
<- lblShipCity  
<- lblShipState

<- lblShipZip

<- btnExit

**step 3**

- you will code all **3** forms: **frmBilling**, **frmShipping**, **frmReview**

<- notice below: **new & important syntaxes**:

### 1). sharing data between forms within current solution:

1.1). in one form, save data into a variable declared in form's declaration section, using the keyword: **Friend**

- variables declared using the **Friend keyword** are recognized by any form within the current solution

declaration syntax:

**Friend FriendVariableName As dataType = expression**

e.g. in **frmBilling**

10      **Friend strBillName As String**

...

16      **strBillName = txtName.Text**

1.2). in another form, refer to a **Friend** variable using the name of the form in which it had been declared, and variable's name

assignment syntax: **reference = OriginFormName.FriendVariableName**

e.g. in **frmShipping**

16      **txtName.Text = frmBilling.strBillName**

<- fills the TextBox with a data saved in **frmBilling**'s variable **strBillName**

### 2). stepping between forms within current solution:

<- notice when you **Hide** the current form, it still resides in RAM, just not visible

2.1). first you hide the current form, using the form's method **Hide()**

form's method: Hide syntax: **Me.Hide()**

**Me** = keyword

- provides a way to refer to the current instance of a class or structure = the instance in which the code is running

2.2). then you open-show the next form, using the form's method **Show()**

form's method: Show syntax: **formName.Show()**

e.g. **frmBilling**

21      **Me.Hide()**

22      **frmShipping.Show()**

### 3). when one of the forms is closing, close all the forms within current solution:

<- notice: as mentioned above - when you **Hide** any current form, it still resides in RAM, just not visible ->

-> therefore to close the entire solution, you must specify each **previously hidden forms** in current form's **FormClosing** event procedure

- form's **btnExit\_Click** event procedure with a method: **Me.Close()** will close only the current form

- to close all previously hidden forms:

3.1). open/create the current form's **FormClosing** event procedure

- as you learned in **CH7**, a form's **FormClosing** event occurs when a form is about to be closed

3.2). use a form's method **Close()**

form's method: Close syntax: **formName.Close()**

e.g. in **frmReview**

```
22      Private Sub frmReview_FormClosing(sender As Object, e As FormClosingEventArgs) Handles Me.FormClosing
23          ' before closing the current form frmReview, close previous - Hidden - forms frmBilling & frmShipping:
24          frmBilling.Close()
25          frmShipping.Close()
26      End Sub
```

## step 3.1 - code the Billing Form - frmBilling

1.0 - create variables, which can store an input data from the current form & can be accessed by the other forms within the current solution

1.1 -> in form's declaration section, declare variables using a keyword **Friend**

2.0 - define how to continue to the next form, when the user clicks the button **Continue**

2.1 -> open/create **btnContinue\_Click** control's event procedure

2.2 - save input data into **Friend** variables, declared in a form's declaration section

2.3 -> hide the current form **frmBilling** and show the next form **frmShipping**, using the form's methods: **Me.Hide()** and **frmShipping.Show()**

3.0 - define how to close the current form, when user clicks the button **Exit**

3.1 -> open/create **btnExit\_Click** control's event procedure

3.2 -> use a method **Me.Close()** to close the current form -> since there are no previous forms hidden, this will do the work

```
1  ' Name:      Zappet Project.
2  ' Purpose:    Gets the billing input data.
3  ' Programmer: me on just now
4 Option Explicit
5 Option Strict On
6 Option Infer Off
7
8 Public Class frmBilling
9     ' declare Friend variables, visible to other forms within current solution:
10    Friend strBillName As String
11    Friend strBillAddress As String : Friend strBillCity As String : Friend strBillState As String : Friend strBillZip As String
12
13    ' continue to the frmShipping form:
14    Private Sub btnContinue_Click(sender As Object, e As EventArgs) Handles btnContinue.Click
15        ' 1. save input data to a Friend variables:
16        strBillName = txtName.Text
17        strBillAddress = txtAddress.Text : strBillCity = txtCity.Text
18        strBillState = txtState.Text : strBillZip = txtZip.Text
19
20        ' 2. hide the current form frmBilling & then show the next form frmShipping:
21        Me.Hide()
22        frmShipping.Show()
23    End Sub
24
25    ' close the current form:
26    Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
27        ' Me = keyword - refer to the current instance of a class or structure = the instance in which the code is running only:
28        Me.Close()
29    End Sub
30 End Class
```

**step 3.2**

- code the **Shipping Form** - `frmShipping`

- 1.0** - create variables, which can store an input data from the current form & can be accessed by the other forms within the current solution  
**1.1** -> in form's declaration section, declare variables using a keyword **Friend**

- 2.0** - for many customer orders, the shipping information is the same as the billing information - therefore when the `frmShipping` form is loaded into memory, its **Load** event procedure will display the billing info in the TextBoxes on the form  
- in cases where the shipping information is different from the billing information, the user can make the appropriate changes in the text boxes  
**2.1** -> open/create `frmShipping_Load` form's event procedure  
**2.2** -> refer to a **Friend** variables from previous form `frmBilling` using assignment statement: `reference = OriginFormName.FriendVariableName`

- 3.0** - define how to continue to the next form, when the user clicks the button **Continue**  
**3.1** -> open/create `btnContinue_Click` control's event procedure  
**3.2** -> save input data into **Friend** variables, declared in a form's declaration section  
**3.3** -> hide the current form `frmShipping` and show the next form `frmReview`, using the form's methods: `Me.Hide()` and `frmReview.Show()`

- 4.0** - define how to close the entire solution, when user clicks the button **Exit**  
- when you **hid** a previous form, it still resides in RAM, just not visible -> therefore to close the entire solution, you must **Close** each previously hidden forms in current form's **FormClosing** event procedure which occurs when a form is about to be closed  
**4.1** -> open/create `frmShipping_FormClosing` form's event procedure  
**4.2** -> use a method `frmBilling.Close()` to close previous and hidden form

- 5.0** - define how to close the current form, when user clicks the button **Exit**  
**5.1** -> open/create `btnExit_Click` control's event procedure  
**5.2** -> use a method `Me.Close()` to close the current form

```
1  ' Name:      Zappet Project.
2  ' Purpose:    Gets the shipping input data.
3  ' Programmer: me on just now
4  Option Explicit On
5  Option Strict On
6  Option Infer Off
7
8  Public Class frmShipping
9    ' declare Friend variables, visible to other forms within current solution:
10   Friend strShipName As String
11   Friend strShipAddress As String : Friend strShipCity As String : Friend strShipState As String : Friend strShipZip As String
12
13   ' when the form is started, load data:
14   Private Sub frmShipping_Load(sender As Object, e As EventArgs) Handles Me.Load
15     ' fill the TextBoxes with a data saved in frmBilling's Friend variables:
16     txtName.Text = frmBilling.strBillName
17     txtAddress.Text = frmBilling.strBillAddress : txtCity.Text = frmBilling.strBillCity
18     txtState.Text = frmBilling.strBillState : txtZip.Text = frmBilling.strBillZip
19   End Sub
```

```

20
3.0    21      ' continue to the frmReview form:
3.1    22      Private Sub btnContinue_Click(sender As Object, e As EventArgs) Handles btnContinue.Click
3.2    23          ' 1. save input data to a Friend variables:
24          strShipName = txtName.Text
25          strShipAddress = txtAddress.Text : strShipCity = txtCity.Text
26          strShipState = txtState.Text : strShipZip = txtZip.Text
27
28          ' 2. hide the current form frmShipping & then show the next form frmReview:
29          Me.Hide()
30          frmReview.Show()
End Sub
31
32
4.0    33      ' when the form is closing:
4.1    34      Private Sub frmShipping_FormClosing(sender As Object, e As FormClosingEventArgs) Handles Me.FormClosing
4.2    35          ' before closing the current form frmShipping, close previous - Hidden - form frmBilling:
36          frmBilling.Close()
End Sub
37
38
5.0    39      ' close the current form:
5.1    40      Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
41          ' Me = keyword - refer to the current instance of a class or structure = the instance in which the code is running only:
42          Me.Close()
43      End Sub
44  End Class

```

### step 3.3 - code the Review Form - `frmReview`

- 1.0 - the last form will only display the inputs from `frmBilling` and `frmShipping` - therefore when the `frmReview` form is loaded into memory, its `Load` event procedure will display the billing and shipping info in the Labels on the form

- in cases where the shipping information is different from the billing information, the user can make the appropriate changes in the text boxes

1.1 -> open/create `frmReview_Load` form's event procedure

1.2 -> refer to a `Friend` variables from previous forms `frmBilling` and `frmShipping` using assignment statement:

```
reference = OriginFormName.FriendVariableName
```

- 2.0 - define how to close the entire solution, when user clicks the button `Exit`

- when you `hid` a previous forms, they still reside in RAM, just not visible -> therefore to close the entire solution, you must `Close` each previously hidden forms in current form's `FormClosing` event procedure which occurs when a form is about to be closed

2.1 -> open/create `frmReview_FormClosing` form's event procedure

2.2 -> use a methods `frmBilling.Close()` and `frmShipping.Close()` to close each previous and hidden forms

- 3.0 - define how to close the current form, when user clicks the button `Exit`

3.1 -> open/create `btnExit_Click` control's event procedure

3.2 -> use a method `Me.Close()` to close the current form

```

1  ' Name:      Zappet Project.
2  ' Purpose:   Displays the billing and shipping input data.
3  ' Programmer: me on just now
4  Option Explicit On
5  Option Strict On
6  Option Infer Off
7
8  Public Class frmReview
9      ' load data when the form is started:
10     Private Sub frmReview_Load(sender As Object, e As EventArgs) Handles Me.Load
11         ' fill the Labels with a data from frmBilling's & frmShipping's Friend variables:
12         lblBillName.Text = frmBilling.strBillName
13         lblBillAddress.Text = frmBilling.strBillAddress : lblBillCity.Text = frmBilling.strBillCity
14         lblBillState.Text = frmBilling.strBillState : lblBillZip.Text = frmBilling.strBillZip
15
16         lblShipName.Text = frmShipping.strShipName
17         lblShipAddress.Text = frmShipping.strShipAddress : lblShipCity.Text = frmShipping.strShipCity
18         lblShipState.Text = frmShipping.strShipState : lblShipZip.Text = frmShipping.strShipZip
19
20     End Sub
21
22     ' when the form is closing:
23     Private Sub frmReview_FormClosing(sender As Object, e As FormClosingEventArgs) Handles Me.FormClosing
24         ' before closing the current form frmReview, close previous - Hidden - forms frmBilling & frmShipping:
25         frmBilling.Close()
26         frmShipping.Close()
27
28     ' close the current form:
29     Private Sub btnExit_Click(sender As Object, e As EventArgs) Handles btnExit.Click
30         ' Me = keyword - refer to the current instance of a class or structure = the instance in which the code is running only:
31         Me.Close()
32
33     End Sub
34 End Class

```

**step 4**

- you will start and test your solution

<- notice: when you start your application, the first appears the form **frmBilling**

**Figure F-16a**

as set in: **step 1** **4.0** -> open **My Project** and set: **Startup form to: frmBilling**

**Figure F-14**

-> fill the **Billing Form** and click the button **Continue**

**Figure F-16b**

<- notice the **frmShipping** form is already filled by the info entered in previous form **frmBilling**

**Figure F-16c**

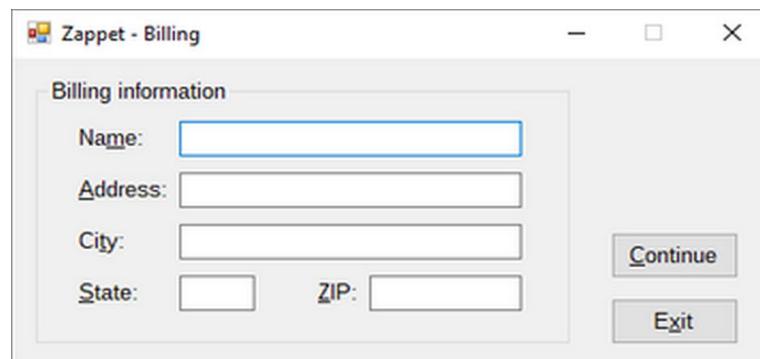
-> fill the **Shipping Form** and click the button **Continue**

**Figure F-16d**

-> review the **Review Form** and click the button **Exit**

**Figure F-16e**

<- notice the entire solution closes, not only the **frmReview**

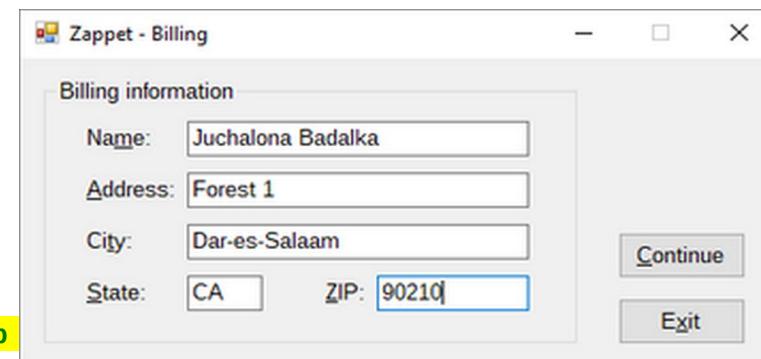


Zappet - Billing

Billing information

Name:	<input type="text"/>
Address:	<input type="text"/>
City:	<input type="text"/>
State:	<input type="text"/> ZIP: <input type="text"/>

**Continue** **Exit**

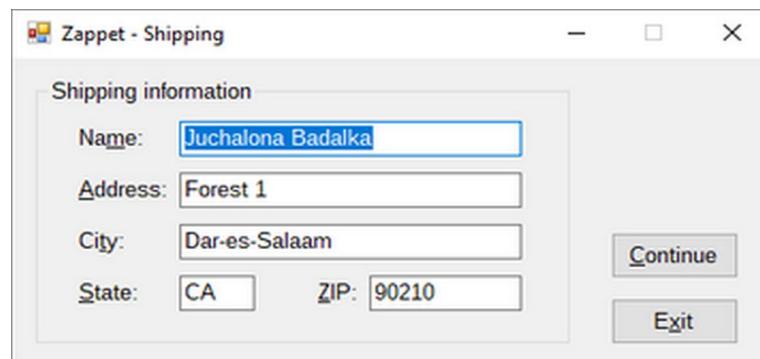
**Figure F-16a**

Zappet - Billing

Billing information

Name:	Juchalona Badalka	
Address:	Forest 1	
City:	Dar-es-Salaam	
State:	CA	ZIP: <input type="text"/> 90210

**Continue** **Exit**

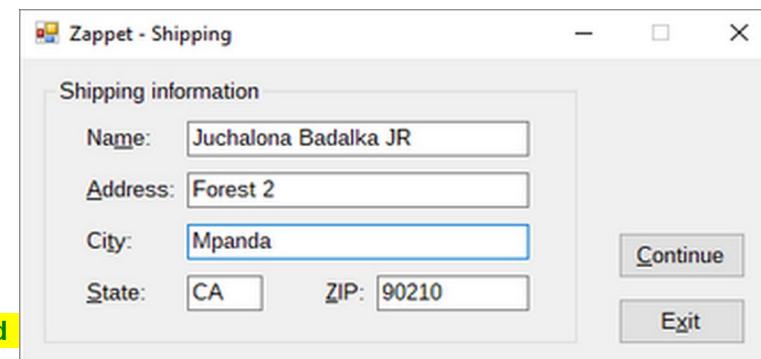
**Figure F-16b**

Zappet - Shipping

Shipping information

Name:	Juchalona Badalka	
Address:	Forest 1	
City:	Dar-es-Salaam	
State:	CA	ZIP: <input type="text"/> 90210

**Continue** **Exit**

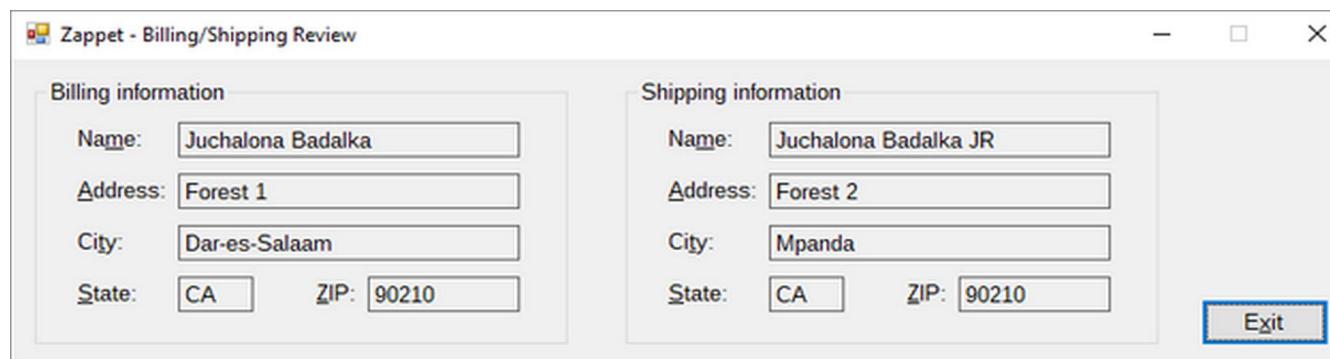
**Figure F-16c**

Zappet - Shipping

Shipping information

Name:	Juchalona Badalka JR	
Address:	Forest 2	
City:	Mpanda	
State:	CA	ZIP: <input type="text"/> 90210

**Continue** **Exit**

**Figure F-16d**

Zappet - Billing/Shipping Review

Billing information

Name:	Juchalona Badalka	
Address:	Forest 1	
City:	Dar-es-Salaam	
State:	CA	ZIP: <input type="text"/> 90210

Shipping information

Name:	Juchalona Badalka JR	
Address:	Forest 2	
City:	Mpanda	
State:	CA	ZIP: <input type="text"/> 90210

**Exit**

**Figure F-16e**

## Appendix F\_04 - MDI application - 2x main/primary forms separated into: Parent Form/window/GUI & its Child Forms/windows/GUIs e.g. with 04.JotPad Solution (VS2022 & .NET 6.0)

- MDI = multiple-document interface

- consists of a parent window and 1 or more child windows that are contained within the parent window
- each document appears in a separate child window within the parent window, and each has access to the parent window's menu bar and toolbar
- not as popular due to confusing management of child windows
- most provides a menu named Window or View, that lists the open documents

e.g. some earlier versions of MS Excel, such as Excel 2007

Figure F-00b

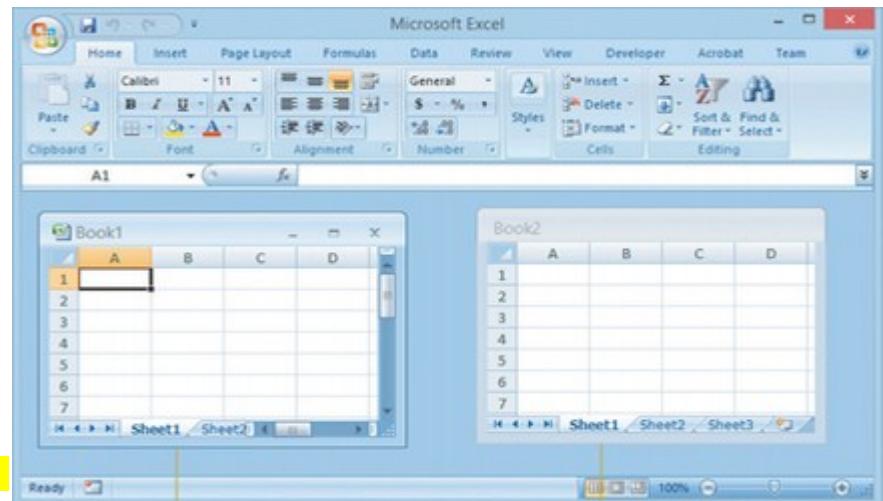


Figure F-00b

- you will create an MDI application named **JotPad**, which allow the user to save information to & read information from a sequential access file - **.txt** file
- the app will contain: 1). a **Parent Form** - **FrmParent**

- containing a **MenuStrip** control with a several elements (VS: **Toolbox / Menus & Toolbars / ToolStrip**)
- more about control in: [CH9\\_A1 - Add a Menu\(mnu\) to a Form: Toolbox / Menus & Toolbars / ToolStrip control](#)

### 2). a **Child Form** - **FrmChild**

- containing a **TextBox** filling entire interior because setting the property **Dock = Fill**

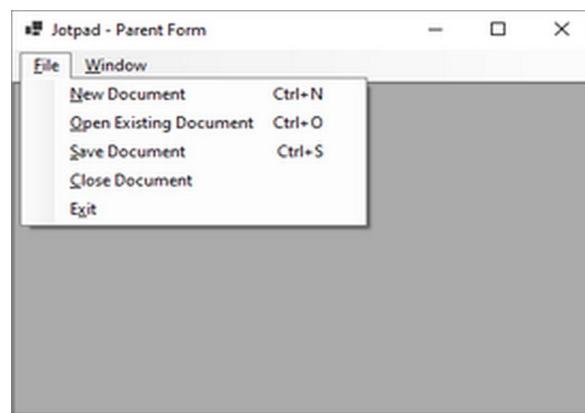


Figure F-23a

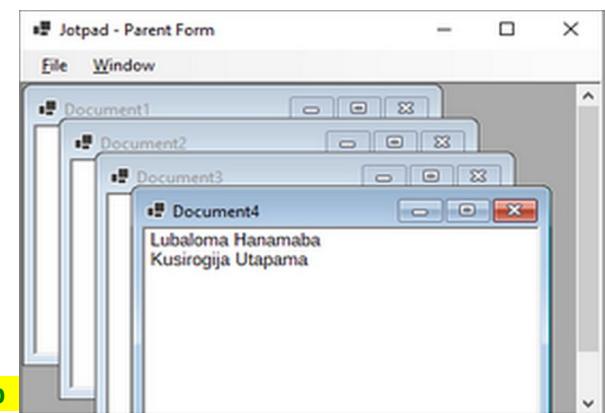


Figure F-23b

index:

- |        |                                                                                                                              |
|--------|------------------------------------------------------------------------------------------------------------------------------|
| step 0 | - create Windows Forms app, which will automatically create the first main/primary <b>Windows Form</b> named <b>Form1.vb</b> |
| step 1 | - add another main form to the application & rename both the forms & set Starting form & personalize all <b>2</b> main forms |
| step 2 | - create GUI for each of the main forms                                                                                      |
| step 3 | - specify the parent form by setting its property <b>IsMdiContainer = True</b>                                               |
| step 4 | - set the <b>FrmParent</b> form's <b>Window</b> menu title to list open child forms <b>FrmChild</b>                          |
| step 5 | - code the <b>FrmParent</b> - Parent Form's menu items                                                                       |
| step 6 | - test your solution                                                                                                         |

- to add another primary form to the application & **create GUI** & set the Parent form & **code** the forms - e.g. with **04.JotPad Solution**

<- note: using **VS2022 & .Net 6.0 Framework** (15-11-2021)

**step 0** - you will create Windows Forms app, which will automatically create the first main/primary **Windows Form** named **Form1.vb**  
-> create the: ...VB2017\Appendix\_F\Exercise\04.JotPad Solution\JotPad Solution.sln

**step 1** - you will add another main form to the application & rename both the forms & set Starting form & personalize all **2** main forms

- |            |                                                                                                                                                                                                                                                                                                                                                                                           |
|------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>1.0</b> | - add second main form                                                                                                                                                                                                                                                                                                                                                                    |
| <b>1.1</b> | -> open <b>Add New Item</b> dialog box by either:<br>a). menu bar: <b>Project / Add Form (Windows Forms)...</b><br>b). menu bar: <b>Project / Add New Item... Ctrl+Shift+A</b>                                                                                                                                                                                                            |
| <b>1.2</b> | -> on the left pane choose: <b>Installed / Common Items / Windows Forms</b> <b>Figure F-08a</b>                                                                                                                                                                                                                                                                                           |
| <b>1.3</b> | -> in the middle choose: <b>Form (Windows Forms)</b> template with a default name: <b>Form2.vb</b><br>note: <b>VS2022</b> offers a template: <b>MDI Parent Form (Windows Forms)</b> with a default name: <b>MDIParent1.vb</b><br><-<br>- with a description: Skeleton form for a Windows Forms (WinForms) Multiple Document Interface application<br>- but I will stick to info in a book |

<- notice the **Solution Explorer** window contains now all **2 main forms** with default names

**2.0** -> rename: **Form1.vb** to: **Parent Form.vb** & change **(Name)** Property to: **FrmParent**  
**Form2.vb** to: **Child Form.vb** & change **(Name)** Property to: **FrmChild**

**3.0** - set the starting form  
-> open **My Project** and set: **Startup object** to: **frmParent** **Figure F-18**

**4.0** -> set each of the forms **Properties**:

**Font** = Liberation Sans, 9.75px

**FormBorderStyle** = Sizable

**MaximizeBox** = True

**FrmParent:**

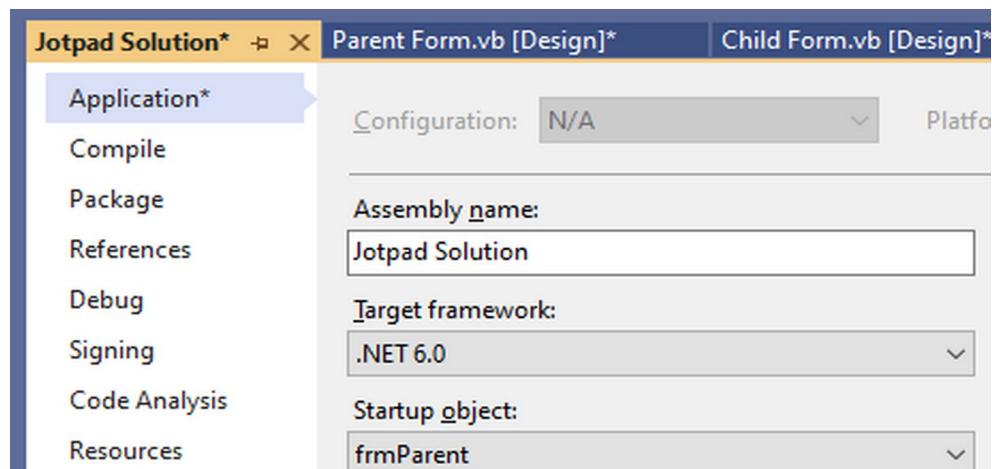
**StartPosition** = CenterScreen

**Text** = Jotpad - Parent Form

**FrmChild:**

**StartPosition** = WindowsDefaultLocation

**Text** = Jotpad - Child Form



**Figure F-18**

## step 2

- you will create GUI for each of the main forms

**Figure F-19** & **Figure F-20**

- you will add 2 menu titles & 5 menu items, change default Text and Name, and add a shortcut keys to some of the menu items

- **FrmParent** will contain only a **MenuStrip** control with several elements

-> drag the **MenuStrip** tool to the form: **Toolbox / Menus & Toolbars / MenuStrip**

<- notice the **MenuStrip1** control appears in GUI and in the component tray

**Figure F-19**

-> in Designer window, click the arrow down to open a drop down menu

**Figure F-19a**

<- notice the first menu title with a default Text **ToolStripMenuItem1**

**Figure F-19c**

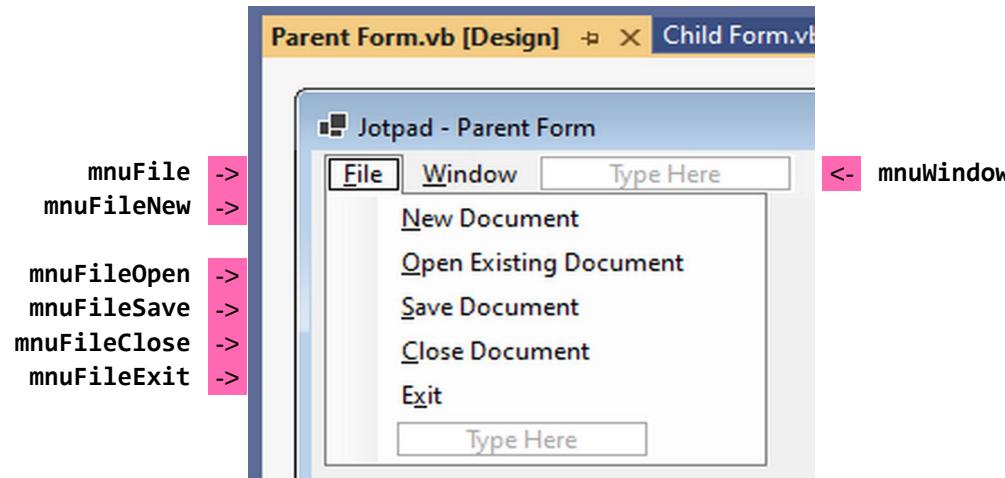
**Figure F-19b**

The figure consists of three side-by-side screenshots of the Microsoft Visual Studio IDE. The leftmost screenshot, labeled 'Figure F-19a', shows a Windows Form titled 'Jotpad - Parent Form' with a single text box containing the placeholder 'Type Here'. The middle screenshot, labeled 'Figure F-19b', shows the same form with a context menu open over the text box. The menu items listed are 'MenuItem', 'ComboBox', and 'TextBox'. The rightmost screenshot, labeled 'Figure F-19c', shows the form again, but now with a 'MenuStrip' control added at the top. The first item in the menu strip, 'ToolStripMenuItem1', is currently selected.

Figure F-19a

1.3 -> add several menu items to the first menu title and another menu title

-> change defaults: **Text** and **(Name)** according the: **Figure F-19d**



1.4

- in addition to access / Alt keys, some menu items will also have a shortcut keys
- > click the **New Document** menu item named **mnuFileNew** and in the Properties window set the property **ShortcutKeys**
- > the same way assign the shortcut key to other menu items

Figure F-19f

Figure F-19e

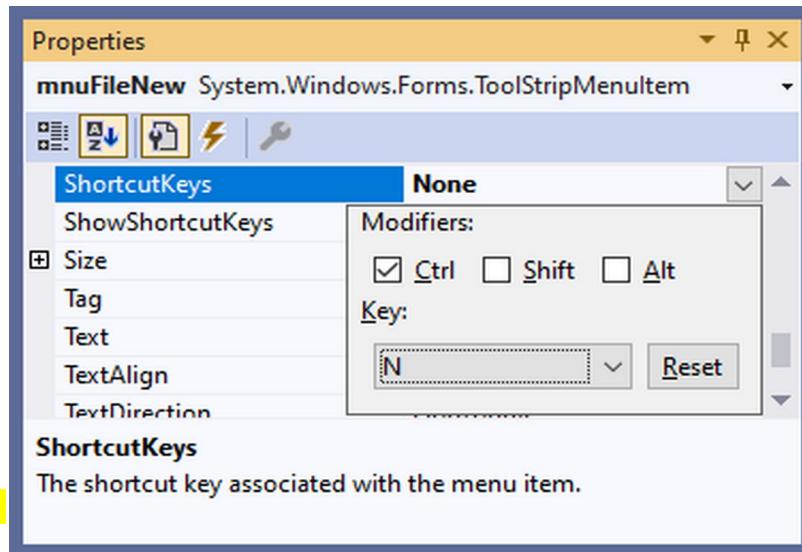


Figure F-19e

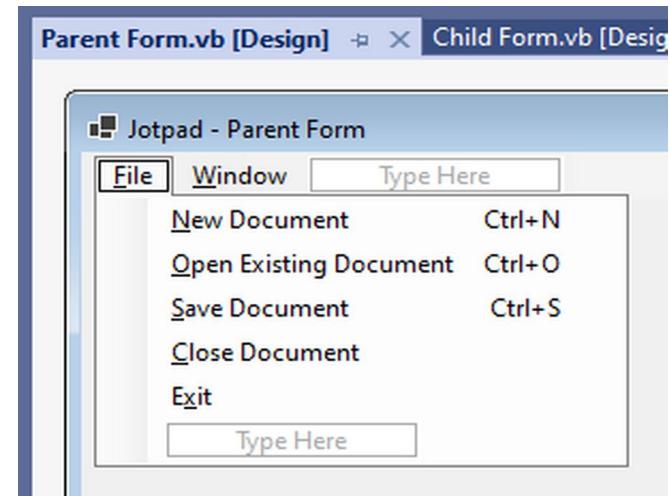


Figure F-19f

**step 2.2**

- you will add a TextBox, set it to multiline & fill the entire form
- **FrmChild** will contain only a TextBox filling the whole interior
- > drag the **TextBox** tool to the form
- > change the default (**Name**) to **txtNote**
- > set the properties: **Multiline = True**  
**Dock = Fill**      <- in VS2022 looks different

&lt;- notice the txtNote control is docked in the form

Figure F-20a

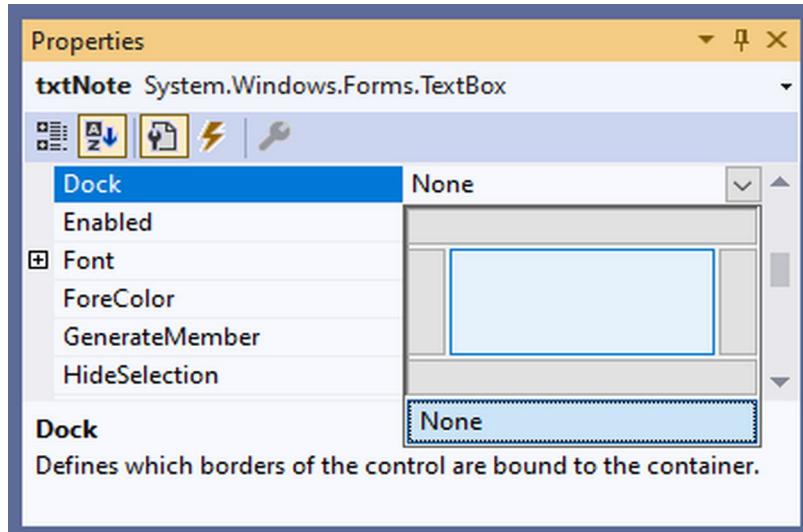


Figure F-20a

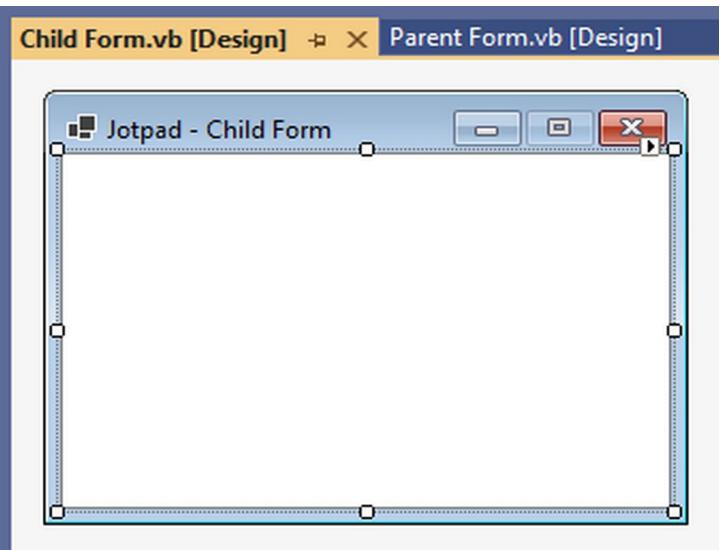


Figure F-20b

**step 3**

- recall that **MDI** applications contain a **parent form** and one or more **child forms** - currently, neither form in the application is the parent or the child

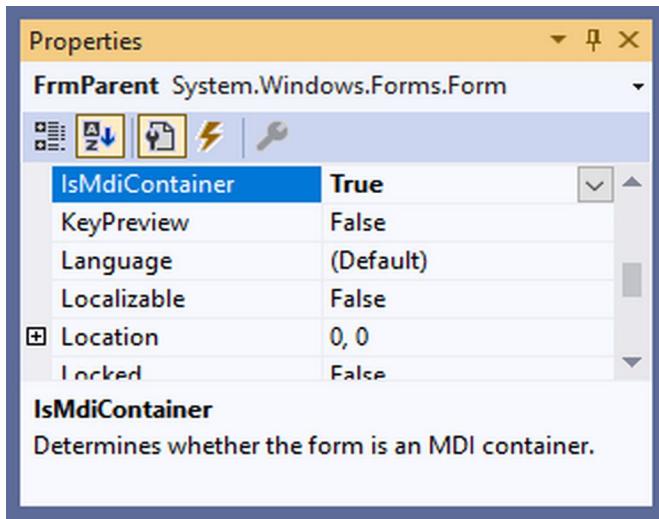
- you will specify the parent form by setting its property **IsMdiContainer = True**

**1.0** -> set the **FrmParent** form's property **IsMdiContainer = True**

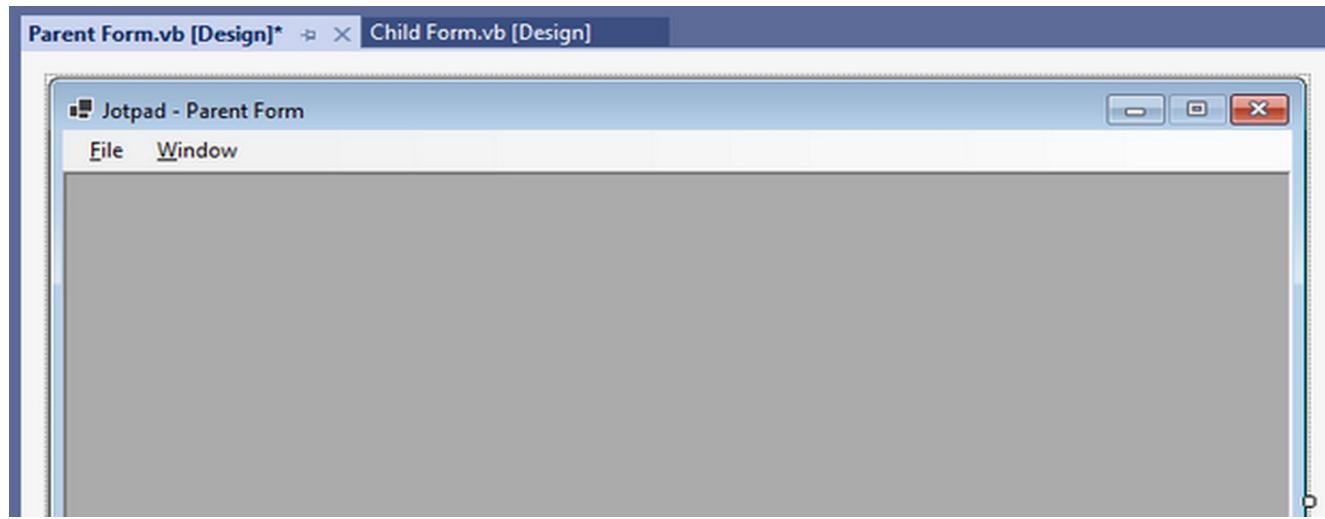
**Figure F-21a**

<- notice the color of the form's interior changes to dark gray, and the interior appears indented [vroubkovaný] within the form's borders

**Figure F-21b**



**Figure F-21a**



**Figure F-21b**

**step 4**

- you will set the **FrmParent** form's **Window** menu title to list open child forms **FrmChild**

**1.0** -> in the component tray, click the **MenuStrip1** control to open its **Properties**

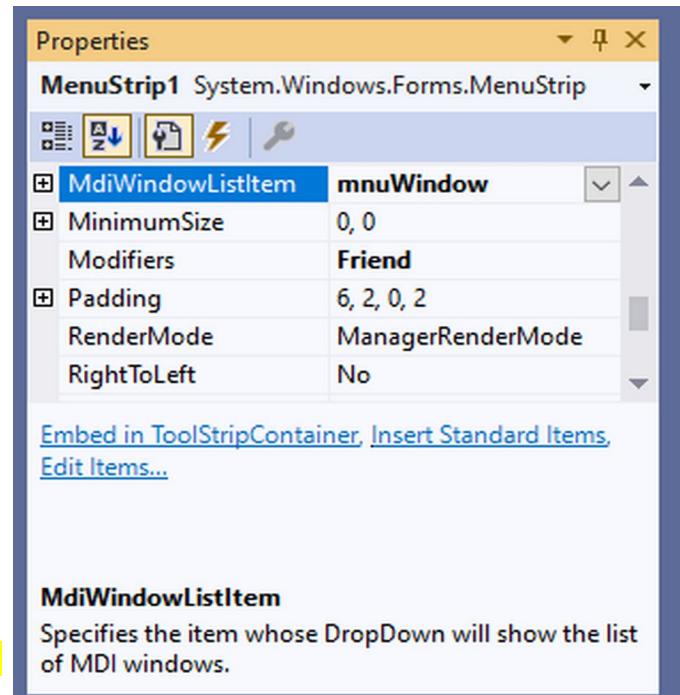
**Figure F-22a**

**1.1** -> set the property **MdiWindowListItem = mnuWindow**

**Figure F-22b**



**Figure F-22a**



**Figure F-22b**

**step 5.0**

- now that the interface is complete, you will code the application - all the code is entered in **Parent form FrmParent**

<- notice below: **new & important syntaxes:**

**step 5.1**

- code the **File** menu's option **New Document** - **mnuFileNew\_Click** event procedure

1). create / declare an instance of a form:

declaration syntax: `Dim newFormName As New formName`

e.g. 15 `Dim frmNewChild As New FrmChild`

2). make the instantiated form a child form of a current (**Me**) form:

form's property syntax: `newFormName.MdiParent = Me`

e.g. 16 `frmNewChild.MdiParent = Me`

3). use a form's **Text** property to name the form in it's title bar:

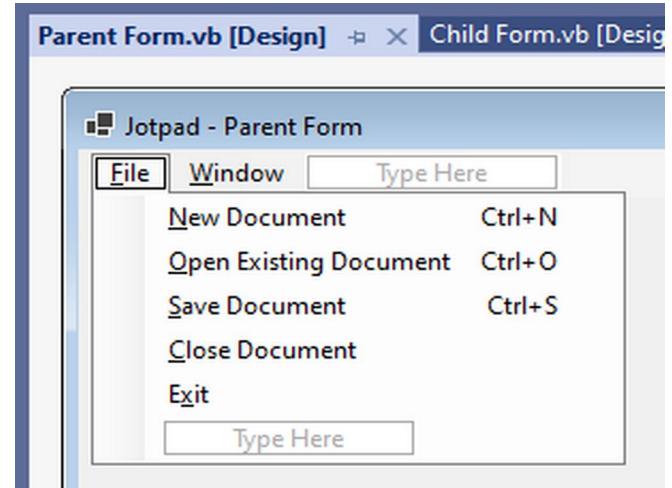
form's title bar **Text** property syntax: `newFormName.Text = form'sTitleBarText`

e.g. 20 `frmNewChild.Text = "Document" & intNum.ToString`

4). display the instantiated child form:

form's method to display it syntax: `newFormName.Show()`

e.g. 22 `frmNewChild.Show()`



**Figure F-19f**

**step 5.2**

- code the **File** menu's option **Open Existing Document** - **mnuFileOpen\_Click** event procedure

5). use an **InputBox** function to prompt the user during a run time, for a name of a document to open:

**InputBox** function syntax: `InputBox(Prompt As String, Title As String, DefaultResponse As String, XPos As Integer, YPos As Integer)`

e.g. 30 `strFileName = InputBox("Filename", "Jotpad")`

more info: **CH3\_A8 - InputBox function** - additional topic from **Appendix B** - interact with user while app is running - syntax & example

6). determine if there is any active child window, using a selection structure:

syntax: `ActiveMdiChild() Is Nothing / ActiveMdiChild() IsNot Nothing`

e.g. 36 `If ActiveMdiChild() Is Nothing Then`

7). use a **ToolStrip control's** method to perform a **Click** event procedure to invoke an existing menu item:

syntax: `toolStripMenuItem.PerformClick()`

e.g. 37 `mnuFileNew.PerformClick()`

8). to the active child window's **txtNote** control's **Text** property assign the sequential access file's contents, line by line:

refer to a control on a child form syntax: `ActiveMdiChild.Controls("controlName")`

refer to a property of the control on a child form syntax: `ActiveMdiChild.Controls("controlName").property`

e.g. 40 `ActiveMdiChild.Controls("txtNote").Text = inFile.ReadLine`

**step 5.3**

- code the **File** menu's option **Save Document** - **mnuFileSave\_Click** event procedure

9). to the sequential access file, on a new line, write the contents of a active child window's **txtNote** control's **Text** property:

e.g. 63 `outFile.WriteLine(ActiveMdiChild.Controls("txtNote").Text)`

**step 5.4**

- code the **File** menu's option **Close Document** - **mnuFileClose\_Click** event procedure

10). use a form's method to close the active child form:

form's method **Close** syntax: `formName.Close()`

e.g. 76 `ActiveMdiChild.Close()`

**step 5.5**

- code the **File** menu's option **Exit** - **mnuFileExit\_Click** event procedure

nothing new

**step 5.1**

- code the **File** menu's option **New Document** - **mnuFileNew\_Click** event procedure

**0.0** -> open / create the code template for the **mnuFileNew\_Click** procedure

**1.0** - create an instance of a Child form, assign it as a child of current form, and display it

**1.1** -> create / declare a new child form using statement: **Dim frmNewChild As New frmChild**

**1.2** -> make the new child form a child of the current form, which is the **FrmParent**, using form's property with a syntax:

**newFormName.MdiParent = Me**

**1.3** -> display the new child form using a form's method with a syntax: **newFormName.Show()**

**2.0** - the user can create a several new documents, therefore you will distinguish between them by appending a unique number in each child window's title bar

**2.1** -> declare a procedure-level **Integer** variable with a global scope using a keyword **Static**, which will be used as a counter

**2.2** -> each new form will have a number 1 digit higher than the previous one - if there is any

**2.3** -> use a form's **Text** property to name the form in its title bar, using a syntax: **newFormName.Text = form'sTitleBarText**

```

11   ' 5.1. File menu's option: New Document - mnuFileNew
12   Private Sub mnuFileNew_Click(sender As Object, e As EventArgs) Handles mnuFileNew.Click
13     ' displays a new child window:
14
15     Dim frmNewChild As New FrmChild      ' 1.1 create / declare a new frmChild form.
16     frmNewChild.MdiParent = Me          ' 1.2 make the new form a child of the current form, which is the frmParent form.
17
18     Static intNum As Integer           ' 2.1 declare variable used as new form's unique identifier.
19     intNum += 1                      ' 2.2 each new form +1.
20     frmNewChild.Text = "Document" & intNum.ToString ' 2.3 display in form's Title bar, using a form's Text property.
21
22     frmNewChild.Show()                '1.3 display the new child form.
23
24   End Sub

```

**step 5.2**

- code the **File** menu's option **Open Existing Document** - **mnuFileOpen\_Click** event procedure

**0.0** -> open / create the code template for the **mnuFileOpen\_Click** procedure

- you will prompt the user to enter a filename, using an **InputBox** function, which allows an application to interact with a user during run time

**1.0** - syntax: **InputBox(Prompt As String, Title As String, DefaultResponse As String, XPos As Integer, YPos As Integer)**

- more about **InputBox**: **CH3\_A8 - InputBox** function - additional topic from **Appendix B** - interact with user while app is running - syntax & example

**1.1** -> declare a procedure-level **String** variable, used as an user input from the **InputBox**

**1.2** -> get the filename from the user and save it to variable for use by a **StreamReader**, using assignment: **strFileName = InputBox("Filename", "Jotpad")**

- the file must be located in a project's default folder: **...bin\Debug**

**2.0** - you will use a **StreamReader** class to open the document with a name given by the user via **InputBox**

- more info: **CH9\_F3 - Sequential Access Input Files: Class StreamReader** - how to create and use them step by step

**2.1** -> declare a **StreamReader** variable, using the **StreamReader** class

**2.2** -> determine whether the input file exists, using the **File Class**'s function/method **Exists**

**2.3** -> to the **StreamReader** variable, assign the object created by **File class**'s function **OpenText**: **inFile = IO.File.OpenText(strFileName)**

**2.4** -> close the file, otherwise it can't be used again **inFile.Close()**

**2.2.1** -> if the input file does not exist, display a message box

- before you can read the contents of a sequential access file line by line and display it in a child form's **TextBox** control, you must determine, if there is any active child form

- if not, use a **ToolStrip** control's method to perform a **Click** event procedure of an existing menu item, which can create a new child window  
syntax: `toolStripMenuItem.PerformClick()`

- if yes, you can assign the sequential access file's contents - read line by line, to the **Text** property of the child form's **TextBox** control

- refer to a control on a child form syntax:

`ActiveMdiChild.Controls("controlName")`

- refer to a property of the control on a child form syntax:

`ActiveMdiChild.Controls("controlName").property`

**3.1** -> determine, if there is any active child window `If ActiveMdiChild() Is Nothing Then`

**3.2** -> if there is no child form, invoke the **mnuFileNew\_Click** event procedure using the **ToolStrip** control's method: `mnuFileNew.PerformClick()`

**3.3** -> if there is, then to the active child window's **txtNote** control's **Text** property assign the sequential access file's contents, line by line:

`ActiveMdiChild.Controls("txtNote").Text = inFile.ReadLine`

```
25      ' 5.2. File menu's option: Open Existing Document - mnuFileOpen
0.0    26  Private Sub mnuFileOpen_Click(sender As Object, e As EventArgs) Handles mnuFileOpen.Click
        27      ' reads information from a sequential access file - txt file:
        28
1.1    29      Dim strFileName As String                      ' 1.1 declare a variable used as a filename.
1.2    30      strFileName = InputBox("Filename", "Jotpad")   ' 1.2 get the filename from the user.
        31
2.1    32      Dim inFile As IO.StreamReader                 ' 2.1 delcare a StreamReader variable, using the StreamReader class.
2.2    33      If IO.File.Exists(strFileName) Then           ' 2.2 determine whether the input file exists.
2.3    34          inFile = IO.File.OpenText(strFileName)     ' 2.3 open the file for input.
        35
3.1    36      If ActiveMdiChild() Is Nothing Then ' 3.1 determine if there is any active child window.
3.2    37          mnuFileNew.PerformClick()           ' 3.2 if not, invoke an existing procedure which creates a new child window.
        38
3.3    39      '   3.3 if yes, to TextBox.Text property on the child form assign the file's contents:
40
        41      ActiveMdiChild.Controls("txtNote").Text = inFile.ReadLine
        42
2.4    42      inFile.Close()                                ' 2.4 close the file
2.2.1  43
        44      Else
                MessageBox.Show("Can't locate " & strFileName, "Jotpad", MessageBoxButtons.OK, MessageBoxIcon.Information)
        45
        46
End Sub
        47
```



```

4.1      60          Dim outFile As IO.StreamWriter           ' 4.1 declare a StreamWriter variable, using the StreamWriter class.
4.2      61          outFile = IO.File.CreateText(strFileName)    ' 4.2 open the file for output.
4.3      62          '   ' 4.3 to the file write contents of a txtNote.Text property:
4.4      63          outFile.WriteLine(ActiveMdiChild.Controls("txtNote").Text)
4.5      64          '   ' 4.4 display a confirmation:
4.6      65          MessageBox.Show("Saved to " & strFileName, "Jotpad", MessageBoxButtons.OK, MessageBoxIcon.Information)
4.7      66          outFile.Close()                           ' 4.5 close the file.

67          End If
68      End If
69  End Sub
70

```

**step 5.4** - code the **File** menu's option **Close Document** - **mnuFileClose\_Click** event procedure

**0.0** -> open / create the code template for the **mnuFileClose\_Click** procedure

**1.0** -> before the active child form can be closed, you should determine, whether there is any - closing the non existing form causes an **Exception Thrown**

76

ActiveMdiChild.Close() ' 1.2 if yes, Close it.



**1.1** -> determine, if there is any active child window

If ActiveMdiChild() IsNot Nothing Then

**1.2** -> if there is, use a form's method **Close()** to close only the active child window

ActiveMdiChild.Close()

```

71      ' 5.4. File menu's option: Close Document - mnuFileClose
72  Private Sub mnuFileClose_Click(sender As Object, e As EventArgs) Handles mnuFileClose.Click
73      ' closes the current (active) child window:
74
75      If ActiveMdiChild IsNot Nothing Then      ' 1.1 determine if there is any active child window.
76          ActiveMdiChild.Close()                ' 1.2 if yes, Close it.
77      End If
78  End Sub
79

```

**step 5.5** - code the **File** menu's option **Exit** - **mnuFileExit\_Click** event procedure

**0.0** -> open / create the code template for the **mnuFileExit\_Click** procedure

**1.1** -> use the form's method **Close()** to close the current Parent / Main form

Me.Close()

```

80      ' 5.5. File menu's option: Exit - mnuFileExit
81  Private Sub mnuFileExit_Click(sender As Object, e As EventArgs) Handles mnuFileExit.Click
82      ' close the current form:
83
84      Me.Close()                           ' 1.1 close the current Parent / Main form.
85  End Sub

```

entire code:

```
1  ' Name:      Jotpad Project
2  ' Purpose:    MDI app - Parent form & its Child form.
3  '           Saves text to and reads text from a sequential access file - txt file.
4  ' Programmer: me on just now
5  Option Explicit On
6  Option Strict On
7  Option Infer Off
8
9  Public Class FrmParent
10
11 ' 5.1. File menu's option: New Document - mnuFileNew
12 Private Sub mnuFileNew_Click(sender As Object, e As EventArgs) Handles mnuFileNew.Click
13     ' displays a new child window:
14
15     Dim frmNewChild As New FrmChild      ' 1.1 create / declare a new frmChild form.
16     frmNewChild.MdiParent = Me          ' 1.2 make the new form a child of the current form, which is the frmParent form.
17
18     Static intNum As Integer           ' 2.1 declare variable used as new form's unique identifier.
19     intNum += 1                        ' 2.2 each new form +1.
20     frmNewChild.Text = "Document" & intNum.ToString ' 2.3 display in form's Title bar, using a form's Text property.
21
22     frmNewChild.Show()                '1.3 display the new child form.
23
24 End Sub
25
26 ' 5.2. File menu's option: Open Existing Document - mnuFileOpen
27 Private Sub mnuFileOpen_Click(sender As Object, e As EventArgs) Handles mnuFileOpen.Click
28     ' reads information from a sequential access file - txt file:
29
30     Dim strFileName As String          ' 1.1 declare a variable used as a filename.
31     strFileName = InputBox("Filename", "Jotpad") ' 1.2 get the filename from the user.
32
33     Dim inFile As IO.StreamReader       ' 2.1 delcare a StreamReader variable, using the StreamReader class.
34     If IO.File.Exists(strFileName) Then   ' 2.2 determine whether the input file exists.
35         inFile = IO.File.OpenText(strFileName) ' 2.3 open the file for input.
36
37     If ActiveMdiChild() Is Nothing Then ' 3.1 determine if there is any active child window.
38         mnuFileNew.PerformClick()        ' 3.2 if not, invoke an existing procedure which creates a new child window.
39
40     ActiveMdiChild.Controls("txtNote").Text = inFile.ReadLine
41
42     inFile.Close()                   ' 2.4 close the file
43
44     Else                            ' 2.2.1 if the file does not exists.
        MessageBox.Show("Can't locate " & strFileName, "Jotpad", MessageBoxButtons.OK, MessageBoxIcon.Information)
```

```

45         End If
46     End Sub
47
48 ' 5.3. File menu's option: Save Document - mnuFileSave
49 Private Sub mnuFileSave_Click(sender As Object, e As EventArgs) Handles mnuFileSave.Click
50     ' saves information to a sequential access file - txt file:
51
52     If ActiveMdiChild() Is Nothing Then      ' 1.1 determine if there is any active child window.
53         '
54         MessageBox.Show("Nothing to be saved.", "Jotpad", MessageBoxButtons.OK, MessageBoxIcon.Information)
55     Else
56         Dim strFileName As String           ' 2.1 declare a variable used as a filename.
57         strFileName = InputBox("Filename", "Jotpad") ' 2.2 get the filename from the user.
58
59         If strFileName <> String.Empty Then      ' 3.1 determine if the user entered some name of a saving document.
60             Dim outFile As IO.StreamWriter        ' 4.1 declare a StreamWriter variable, using the StreamWriter class.
61             outFile = IO.File.CreateText(strFileName)   ' 4.2 open the file for output.
62             '
63             outFile.WriteLine(ActiveMdiChild.Controls("txtNote").Text)
64             '
65             MessageBox.Show("Saved to " & strFileName, "Jotpad", MessageBoxButtons.OK, MessageBoxIcon.Information)
66             outFile.Close()                      ' 4.5 close the file.
67         End If
68     End If
69 End Sub
70
71 ' 5.4. File menu's option: Close Document - mnuFileClose
72 Private Sub mnuFileClose_Click(sender As Object, e As EventArgs) Handles mnuFileClose.Click
73     ' closes the current (active) child window:
74
75     If ActiveMdiChild IsNot Nothing Then      ' 1.1 determine if there is any active child window.
76         ActiveMdiChild.Close()                 ' 1.2 if yes, Close it.
77     End If
78 End Sub
79
80 ' 5.5. File menu's option: Exit - mnuFileExit
81 Private Sub mnuFileExit_Click(sender As Object, e As EventArgs) Handles mnuFileExit.Click
82     ' close the current form:
83
84     Me.Close()                           ' 1.1 close the current Parent / Main form.
85 End Sub
86 End Class

```

step 6

- you will test the application

Figure F-23

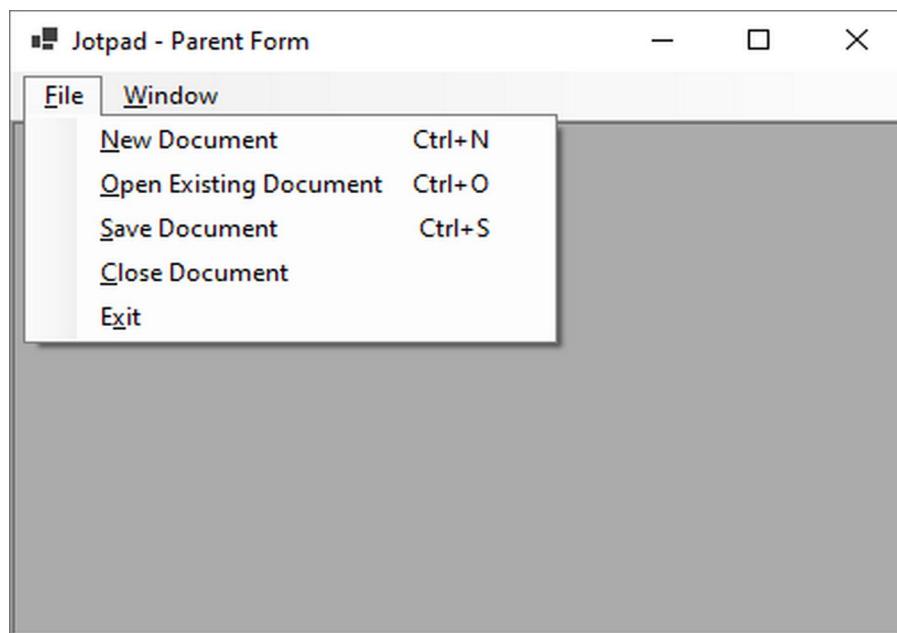


Figure F-23a

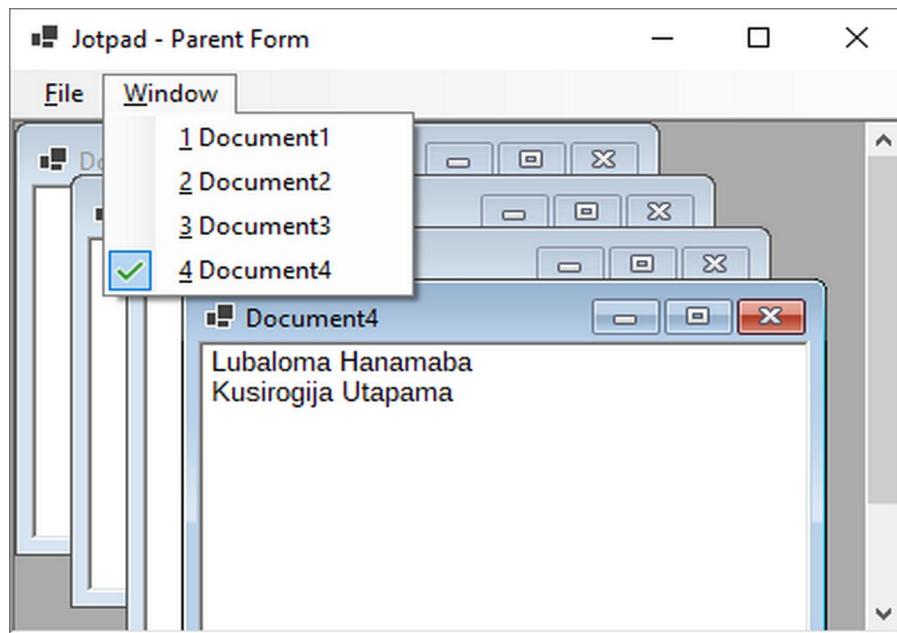


Figure F-23c

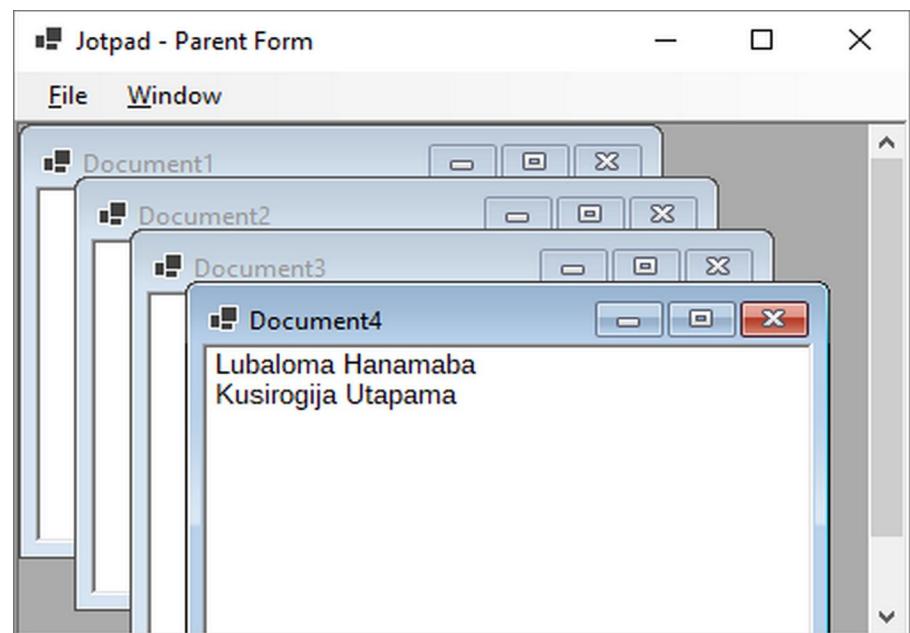


Figure F-23b

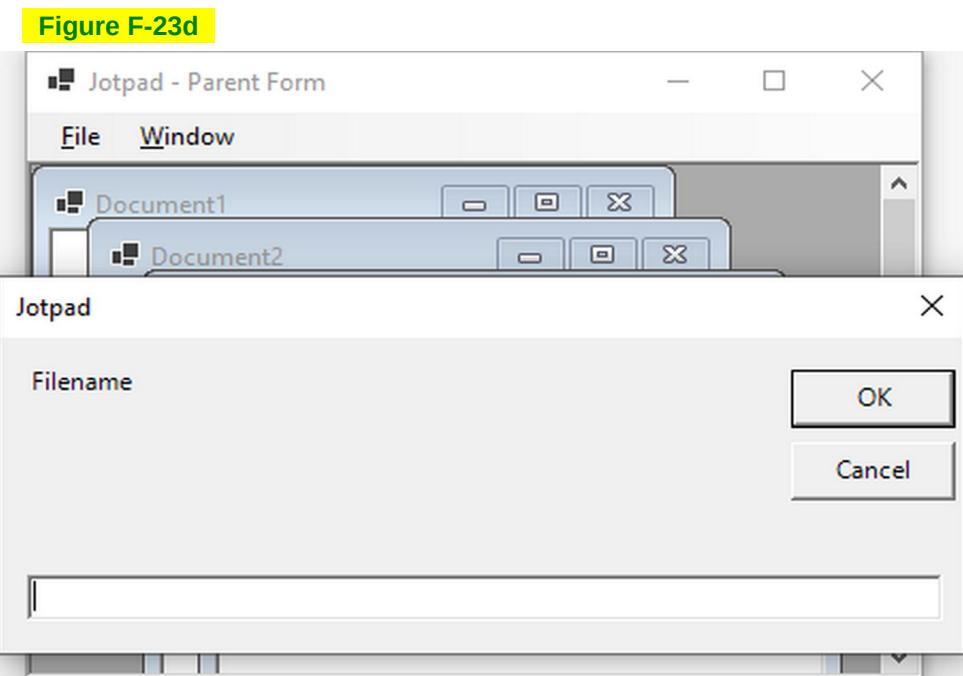


Figure F-23d

- **TDI** = tabbed-document interface

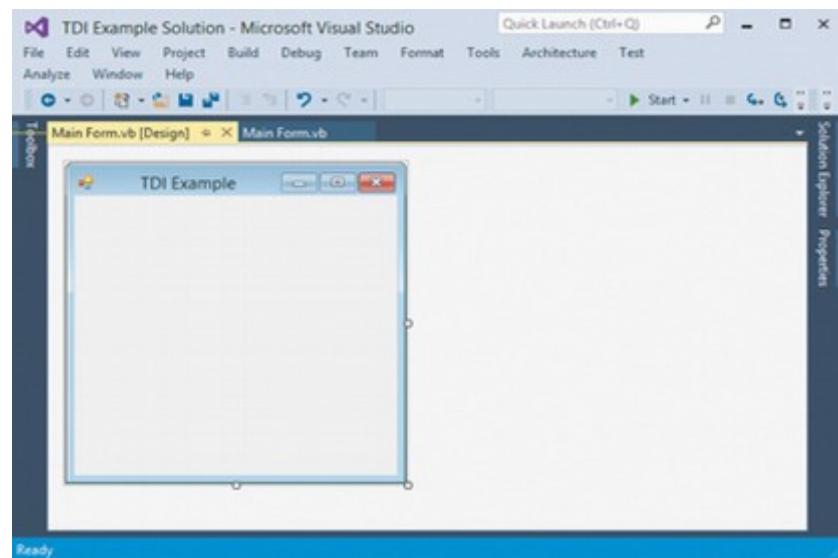
- offshoot (odnož) of **MDI**

- like an **MDI**, allows more than 1 child window to be open in a parent window

- unlike an **MDI**, uses tabs to show the windows that are open in an application

e.g. Visual Studio

**Figure F-00c**



- in this section, you will create a **quasi TDI** application using the **TabControl** tool

located in the: **Toolbox / Containers / TabControl**

- the application will allow the user to convert American dollars to either British pounds or

Mexican pesos, using a tab for each

**Figure F-29a**

& **Figure F-29b**

**Figure F-00c**

**Figure F-29a**

**Figure F-29b**

**step 0**

- create Windows Forms App and configure it

**step 1.1**

- create GUI - add a menu title containing only 1 menu item - **Exit**, and configure it

**step 1.2**

- create GUI - add a **Toolbox / Containers / TabControl** tool and configure it

**step 1.3**

- create GUI - add Labels & TextBoxes & Buttons to both of the **tab** controls, and configure them

<- note: the **(Name)** of the controls on the tab pages must be unique

**step 2.0**

- code the application

**step 3.0**

- test

- to create **quasi TDI** application - an app with **2 tabs** - e.g. with **05.Currency Converter Solution**

<- note: using **VS2022 & .Net 6.0 Framework** (29-11-2021)

**step 0**

- you will create Windows Forms app, which will automatically create the first main/primary **Windows Form** named **Form1.vb**, and then configure it

-> create the: ...VB2017\Appendix\_F\Exercise\05.Currency Converter Solution\Currency Converter Solution.sln

-> rename: **Form1.vb** to: **Main Form.vb** & change **(Name)** Property to: **FrmMain**

-> set the starting form: open **My Project** and set: **Startup object** to: **frmMain**

-> set the form's **Properties**:

Font = Liberation Sans, 9.75px

FormBorderStyle = FixedSingle

MaximizeBox = False

StartPosition = CenterScreen

Text = Currency Converter

**step 1.1**

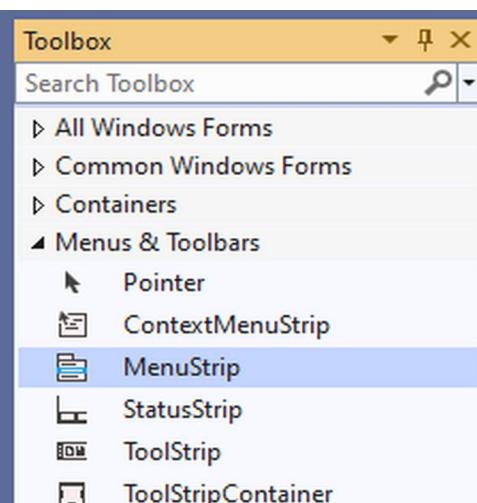
- create GUI - you will add a **menu title** named **File** containing only **1 menu item** - **Exit**, and then configure it

1. -> drag the **MenuStrip** tool to the form: **Toolbox / Menus & Toolbars / ToolStrip**

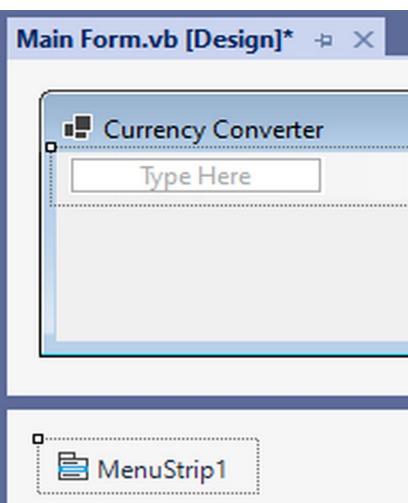
<- notice the **MenuStrip1** control appears in GUI and in the component tray **Figure F-24a**

2. -> in Designer window, click the arrow down to open a drop down menu and select a **MenuItem** **Figure F-24b**

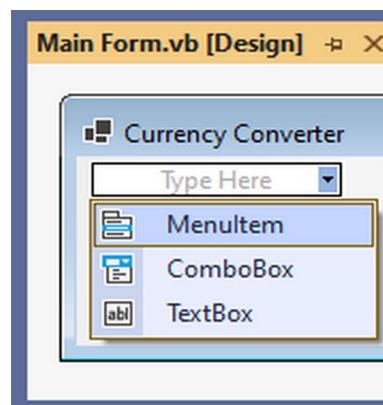
<- notice the first **menu title** with a default Text **ToolStripMenuItem1** **Figure F-24c**



**Figure F-24a**



**Figure F-24b**



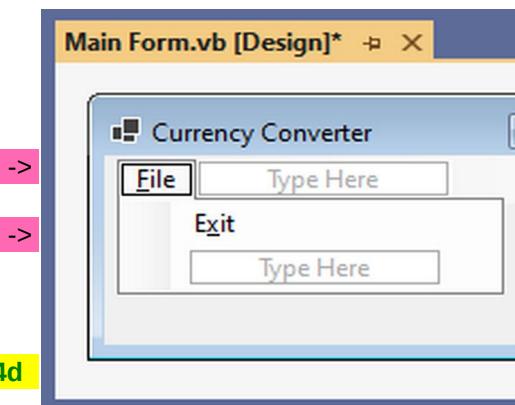
**Figure F-24c**

3. -> to the **menu title** add **1 menu item** and  
change defaults **Text** and **(Name)** according the: **Figure F-24d**

mnuFile ->

mnuFileExit ->

**Figure F-24d**



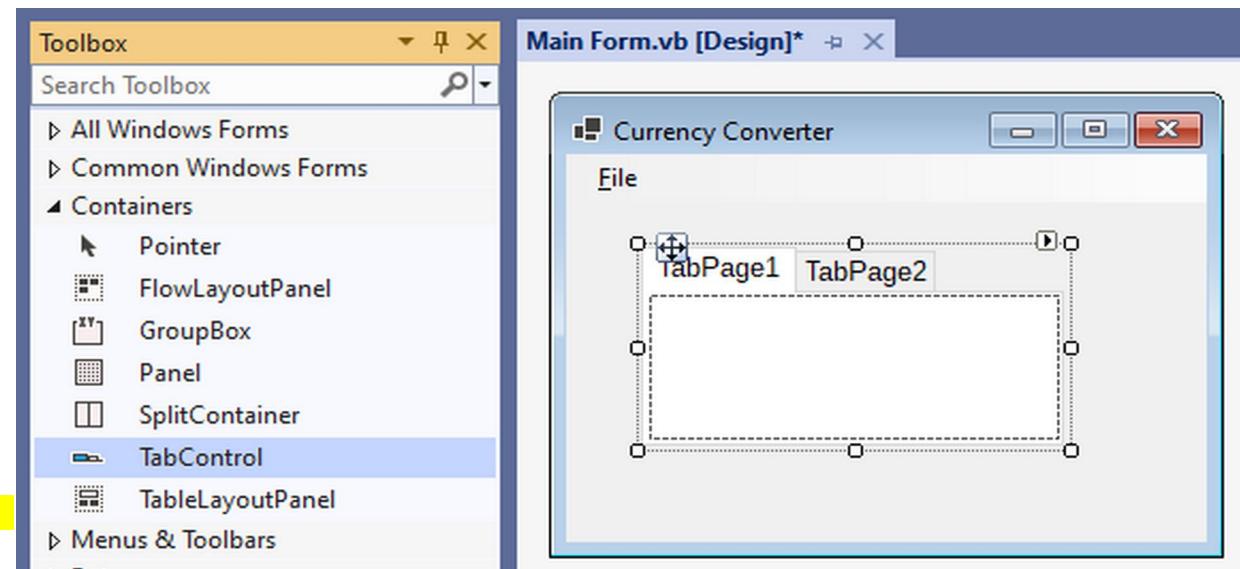
**step 1.2**

- create GUI - you will add a **Toolbox / Containers / TabControl** tool and configure it

1. -> drag the **TabControl** tool to the form: **Toolbox / Containers / TabControl**

<- notice a **tab control** that contains **2 tab pages** appears on the form

**Figure F-25a**

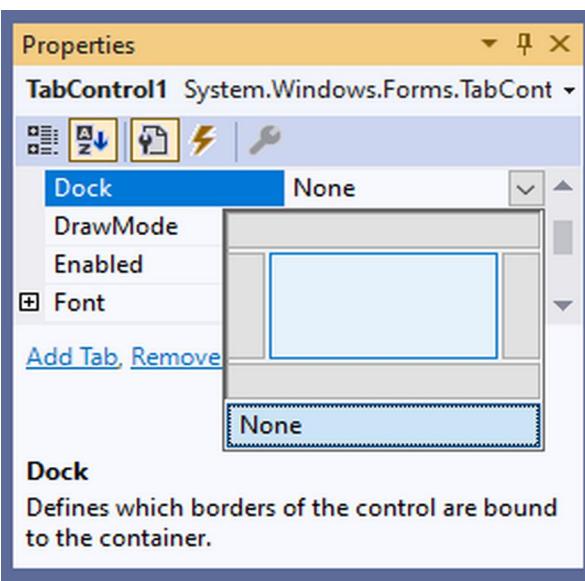


**Figure F-25a**

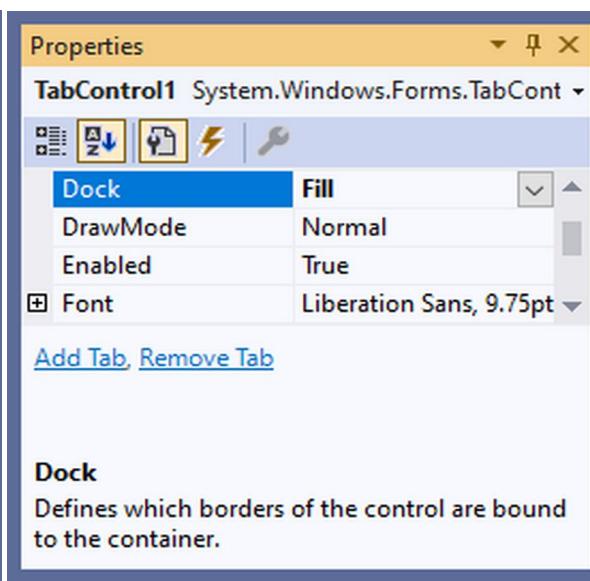
2. -> in the **TabControl1** control's **Properties** window set: **Dock = Fill** by clicking the **middle button**

<- notice the control filled the whole space of the form

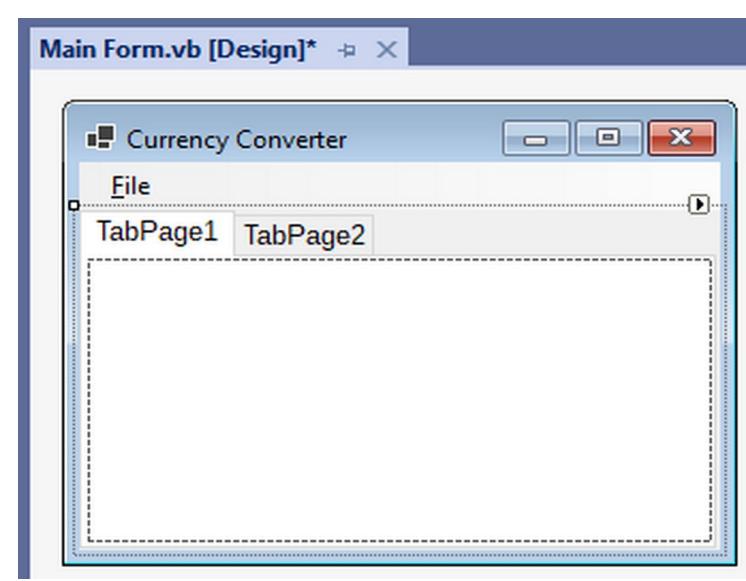
**Figure F-25b** & **Figure F-25c**



**Figure F-25b**



**Figure F-25c**



**Figure F-25d**

<- note: if you would like to **add** or **remove** tab pages, you can use the tab control's **task box**:

Figure F-26

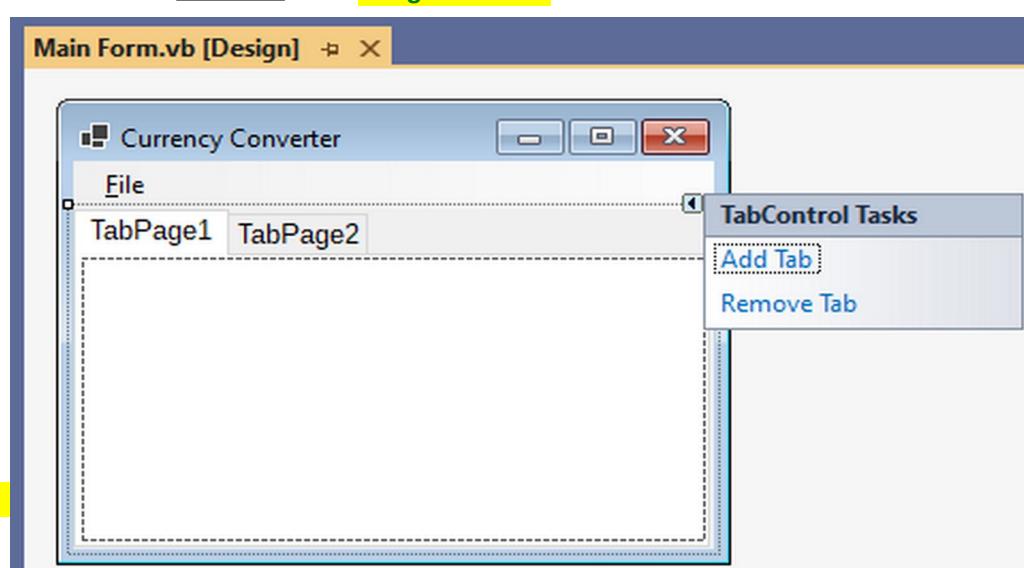


Figure F-26

3. > select the **TabControl1** control and in it's **Properties** window, click the ... (ellipsis) button in setting: **TabPages = (Collection)** ...

Figure F-27a

<- notice the **TabPage Collection Editor** dialog box opens

Figure F-27b

<- note: you can use dialog box to:

- add or remove tab pages
- set the properties of existing tab pages

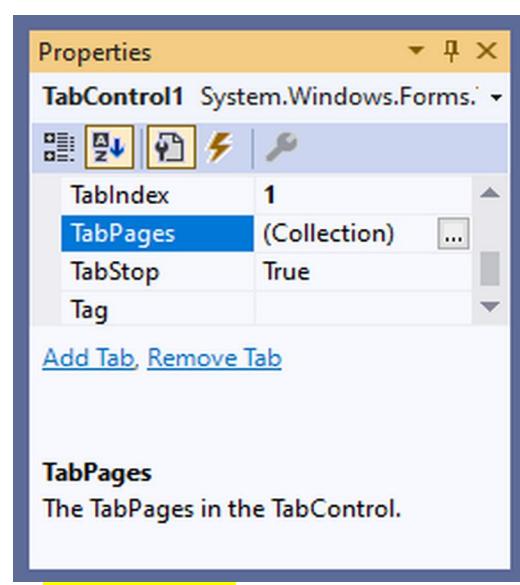


Figure F-27a

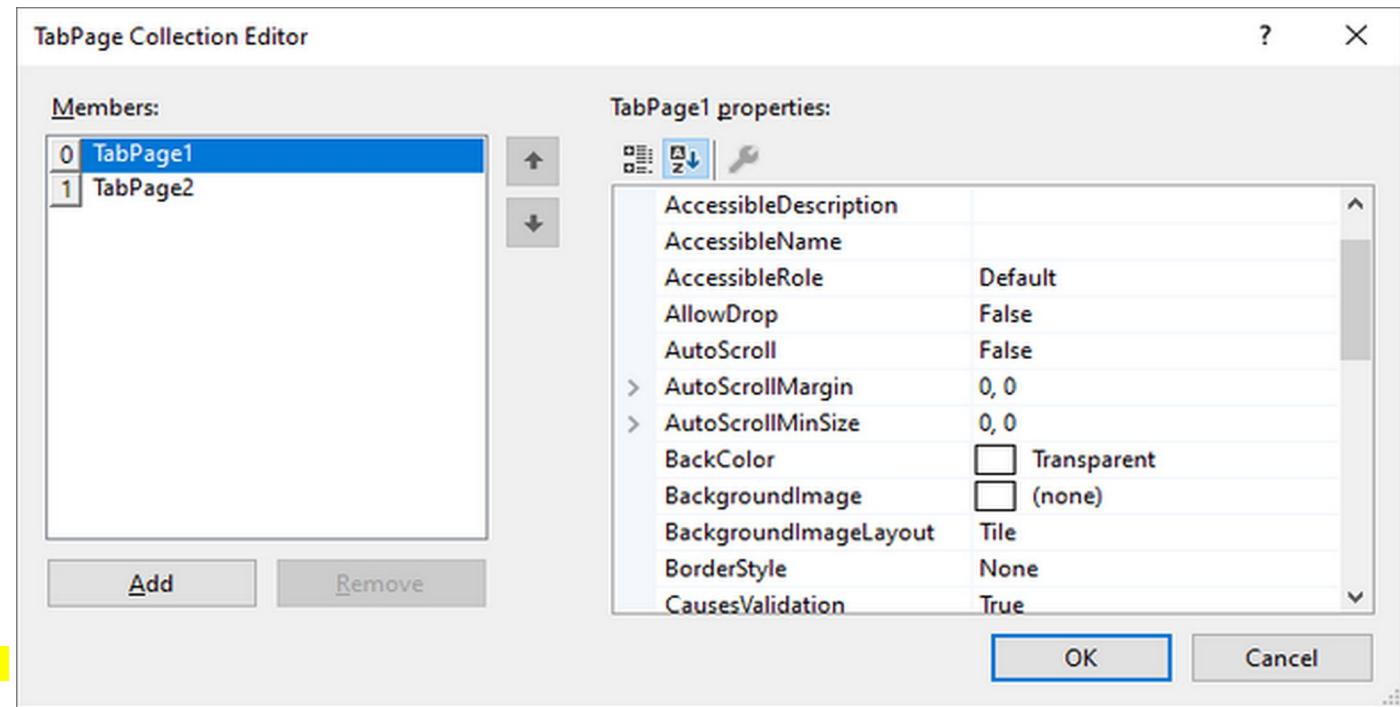
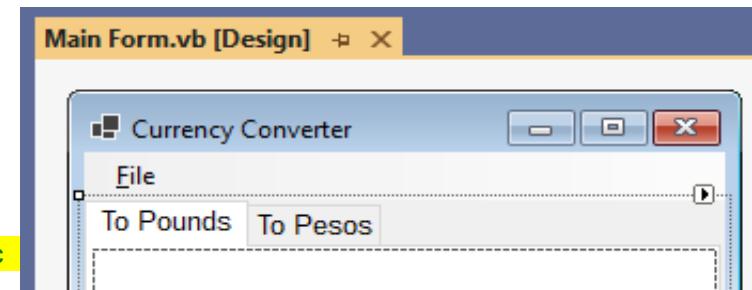


Figure F-27b

4. -> change the properties:
- Member 0: (Name) = tabpgPounds  
Text = To Pounds
  - Member 1: (Name) = tabpgPesos  
Text = To Pesos

-> you can close the **TabPage Collection Editor** dialog box

<- notice the tabs in GUI changed the Texts: **Figure F-27c**



**Figure F-27c**

### step 1.3

- create GUI - you will add **Labels & TextBoxes & Buttons** to both of the **tab** controls, and configure them

<- note: the **(Name)** of the controls on the tab pages must be unique

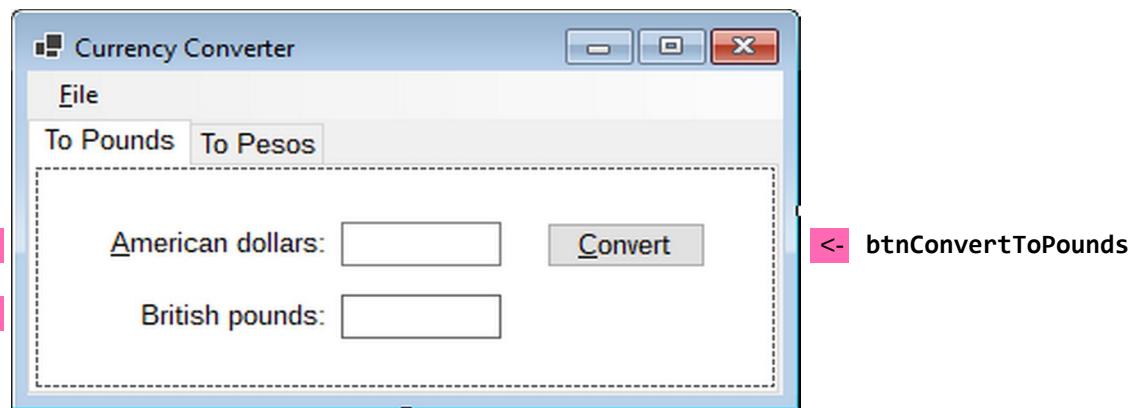
-> in Designer window, click the tab **To Pounds** and then click the window under to activate the **tabpgPounds** control's GUI

-> create controls and change their setting accordin the: **Figure F-28a**

```

txtDollars   ->
 TextAlign    = MiddleCenter
 lblPounds   ->
AutoSize     = False
BorderStyle  = FixedSingle
 TextAlign    = MiddleCenter

```



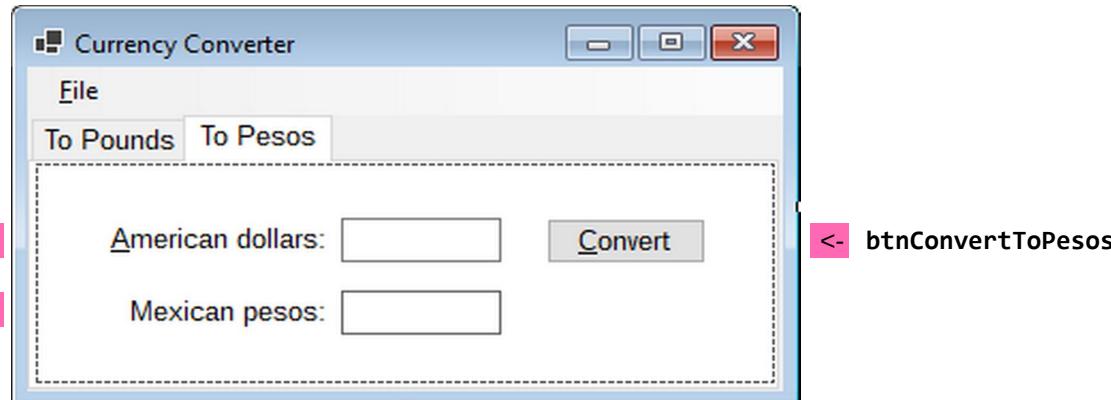
-> in Designer window, click the tab **To Pesos** and then click the window under to activate the **tabpgPesos** control's GUI

-> create controls and change their setting accordin the: **Figure F-28b**

```

txtAmerican ->
 TextAlign    = MiddleCenter
 lblPesos    ->
AutoSize     = False
BorderStyle  = FixedSingle
 TextAlign    = MiddleCenter

```



## step 2.0

- you will code the application

<- note: since every control **must** have a **unique name**, coding isn't a rocket science - therefore i just copy the code without any additional info

```
1  ' Name:      Currency Converter.
2  ' Purpose:    learn how to use a TabControl tool.
3  ' Programmer: me on just now.
4  Option Explicit On
5  Option Strict On
6  Option Infer Off
7
8  Public Class FrmMain
9      Private Sub btnConvertToPounds_Click(sender As Object, e As EventArgs) Handles btnConvertToPounds.Click
10         Const dblPOUND_RATE As Double = 0.67 : Dim dblDollars As Double : Dim dblPounds As Double
11         Double.TryParse(txtDollars.Text, dblDollars) : dblPounds = dblDollars * dblPOUND_RATE : lblPounds.Text = dblPounds.ToString("n2")
12     End Sub
13
14     Private Sub btnConvertToPesos_Click(sender As Object, e As EventArgs) Handles btnConvertToPesos.Click
15         Const dblPESO_RATE As Double = 15.39 : Dim dblDollars As Double : Dim dblPesos As Double
16         Double.TryParse(txtAmerican.Text, dblDollars) : dblPesos = dblDollars * dblPESO_RATE : lblPesos.Text = dblPesos.ToString("n2")
17     End Sub
18
19     Private Sub mnuFileExit_Click(sender As Object, e As EventArgs) Handles mnuFileExit.Click
20         Me.Close()
21     End Sub
22 End Class
```

## step 3.0

- you wil test the application

Figure F-29

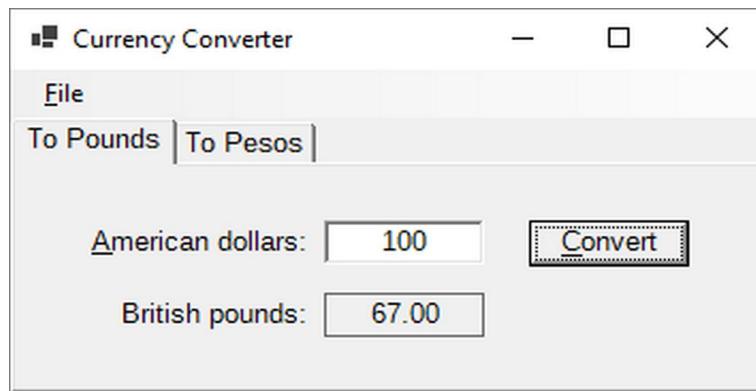


Figure F-29a

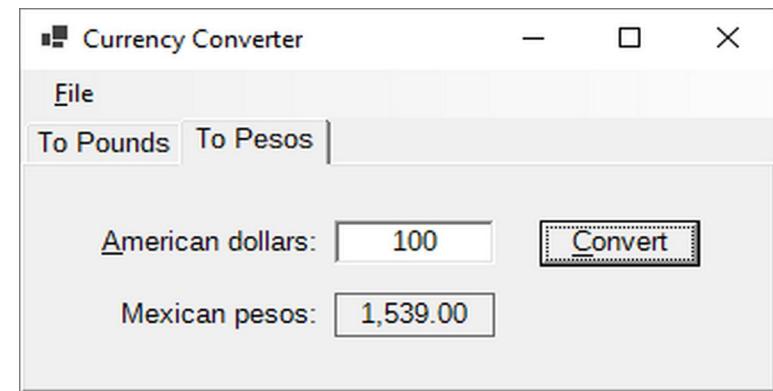


Figure F-29b

## Appendix F\_Key Terms

- **ActiveMdiChild** property - a property of a parent form; used to determine the active child window (if any)
- **Color** dialog box - one of the standard dialog boxes available in Visual Basic
  - created by using the **ColorDialog** tool, which is contained in the **Dialogs** section of the toolbox
- **Font** dialog box - one of the standard dialog boxes available in Visual Basic
  - created by using the **FontDialog** tool, which is contained in the **Dialogs** section of the toolbox
- **Friend** keyword - can be used to declare variables that are recognized by any form within the current solution
- **Hide** method - used to hide a form
- **IsMdiContainer** property - a property of a form; used to specify that the form is a parent form
- **MDI** - the acronym for multiple-document interface
- **MDI application** - an application that consists of a parent window and one or more child windows that are contained within the parent window
- **MdiWindowListItem** property - a property of a menu strip control
  - specifies the menu that lists the open child windows in an MDI application
- **Multiple-form application** - an application that contains more than one form
- **SDI** - the acronym for single-document interface
- **SDI application** - an application in which each window holds a single document
- **Show** method - used to show a form
- **TabControl** tool - located in the **Containers** section of the toolbox
  - used to display a tab control that contains tab pages
- **TDI** - the acronym for tabbed-document interface
- **TDI application** - an offshoot of **MDI**
  - uses tabs to show the names of the opened child windows

THE END, KONEC, BASTA, FINITO