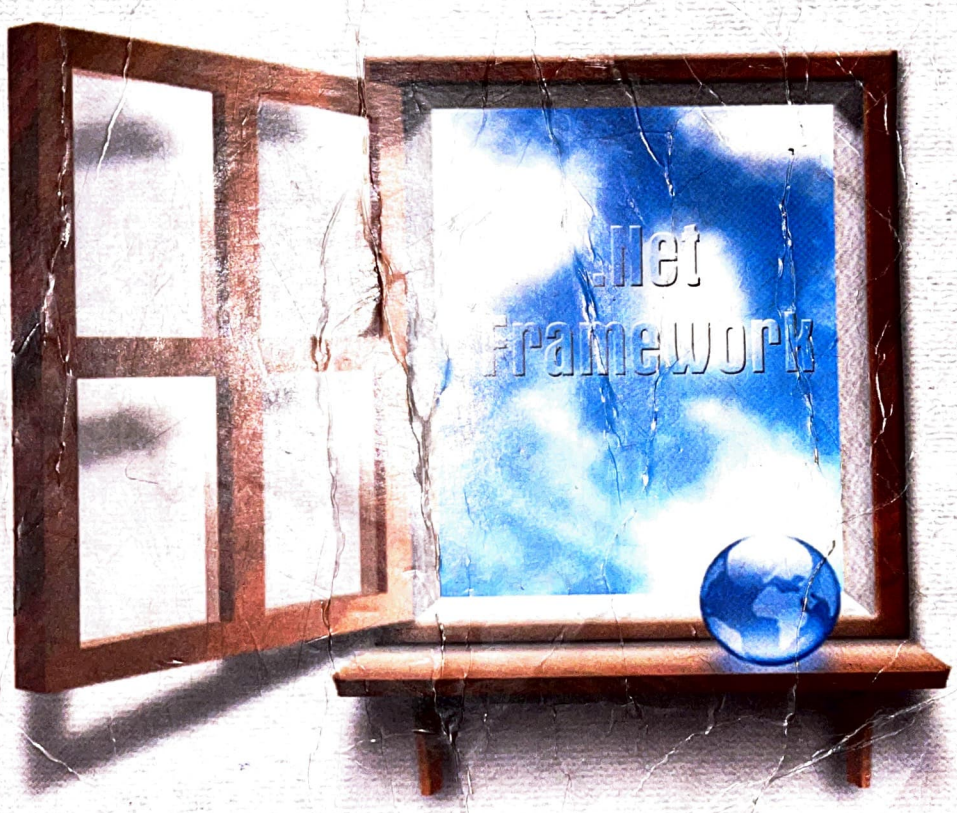
VB.NET PROGRAMMING

Mr. Pragnesh B. Desai



Bharat &
Company

:: CONTENTS ::

Ch. 1 Overview of Microsoft .NET Framework

1.1 What is .NET? 1.2
1.2 The .NET Framework Consists of Two Main Components 1.3
1.3 What is Visual Studio .NET? 1.5
1.4 What is Visual Basic .NET? 1.5

Ch. 2 Visual Basic .NET Programming

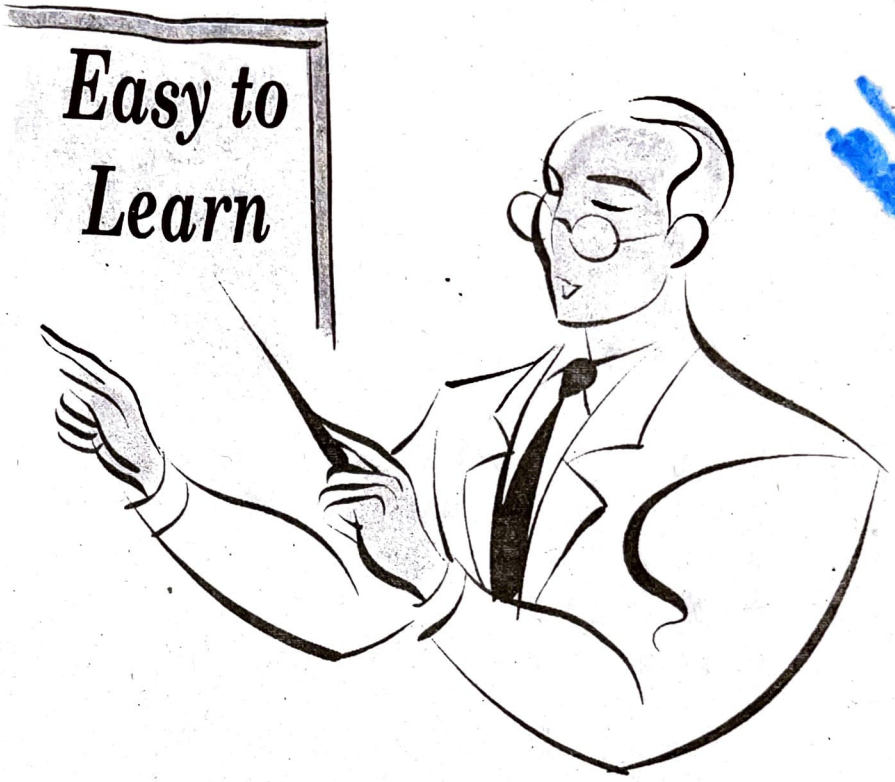
2.1 Computer Application 2.2
2.2 What is IDE? (Integrated Development Environment) 2.2
2.3 Data Type of VB .NET 2005 2.6
2.4 Composite Data Types OR User Define Data Types 2.8
2.5 Working with ToolBox Controls 2.22
2.6 Working with Menus & Dialog Boxes 2.84
2.7 Exception Handling: (Trapping Errors) 2.102
2.8 Using Module & Procedure 2.104
2.9 Using Arrays & Collections 2.109

Ch. 3 Object Oriented Programming

3.1 Object-Oriented Programming 3.2
3.2 Abstraction, Encapsulation & Polymorphism 3.5
3.3 Interfaces & Inheritance 3.8

Ch. 4 Database Access Using ADO.NET

4.1 ADO.NET Overview 4.2
4.2 ADO.NET Object Model OR ADO.NET Architecture 4.3
4.3 Use of ADO.NET Objects Directly in Visual Basic Code. 4.16
4.4 Data Access Components or Data Components 4.23



UNIT 1

OVERVIEW OF MICROSOFT .NET FRAMEWORK

1.1 What is .NET?
1.2 The .NET Framework Consists of Two Main Components.
1.3 What is Visual Studio .NET?
1.4 What is Visual Basic .NET?

1.1 WHAT IS .NET?

→ **.NET is not an operating system. .NET is not a programming language. ".NET is a framework". We cannot define .NET as a 'single thing'.** Microsoft .NET is software that connects information, people, system, and devices.

→ **It is a new, easy, and extensive programming platform. It is not a programming language, but it supports several programming languages. (.NET is a framework)**

→ **The .NET Framework is an environment or platform for developing, deploying and executing applications.**

→ **By default .NET comes with few programming languages including C# (C Sharp), VB.NET, J# and managed C++.** .NET is a common platform for all the .NET supported languages.

→ **.NET framework comes with a common class library,** which can be called from any of the supported languages. So, developers need not learn many libraries when they switch to a different language. Only the syntax is different for each language.

Whether you write code in C# or VB.NET, you are calling methods in the same .NET class libraries. The same .NET framework executes the C# and VB.NET applications. So, there won't be any performance difference based on the language you write code. The common class library in .NET Framework can assist you in developing your applications in a faster, cheaper and easier manner.

→ In future versions of Windows, .NET will be freely distributed as part of operating system and users will never have to install .NET separately. The .NET Framework is included with Windows Server 2003, Windows Server 2008 and Windows Vista, and can be installed on older versions of Windows.

➢ **Version of .Net Framework :**

Version	Release Date	Visual Studio	Default in Windows
.Net Framework 1.0	13-02-2002	Visual Studio .Net	
.Net Framework 1.1	24-04-2003	Visual Studio .Net 2003	Windows Server 2003
.Net Framework 2.0	07-11-2005	Visual Studio 2005	
.Net Framework 3.0	06-11-2006		Window Vista, Windows Server 2008
.Net Framework 3.5	19-11-2007	Visual Studio 2008	Windows 7, Windows Server 2008 R2
.Net Framework 4.0 Beta1	20-05-2009	Visual Studio 2010	

1.2 THE .NET FRAMEWORK CONSISTS OF TWO MAIN COMPONENTS:

[1] COMMON LANGUAGE RUNTIME (CLR)
[2] CLASS LIBRARIES

[1] Common Language Runtime (CLR) :

The CLR is described as the "execution engine" of .NET. It provides the environment within which the programs run. The CLR is the module that actually runs your .NET applications.

It's this CLR that manages the
- → **Execution of programs** and provides important services, such as
- → Code compilation,
- → Memory allocation (memory management),
- → Thread management,
- → Garbage collection
- → Exception handling
- → Security mechanisms etc

❖ **Working of the CLR :**

When the .NET program is compiled, the output of the compiler is not an executable file but a file that contains a special type of code called the Microsoft Intermediate Language (MSIL), So, your compiled executable contains the IL and not really executable machine language.

This MSIL defines a set of portable instructions that are independent of any specific CPU. It is this MSIL which you deliver to your customer in the form of a DLL or EXE.

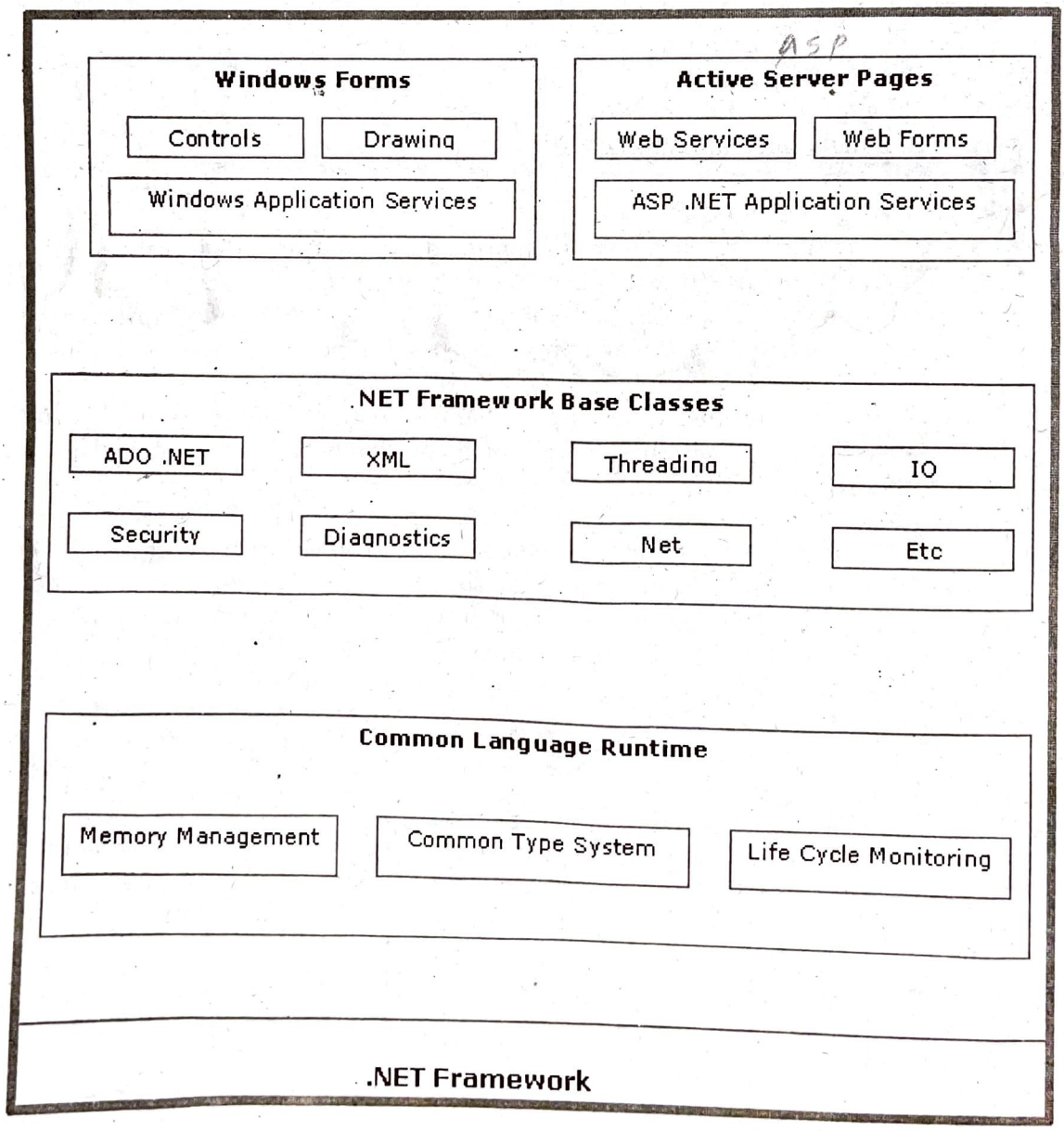
It's the job of the CLR to translate this Intermediate code into an executable. When the .NET application runs, the .NET framework in the target computer take care of the execution. (To run a .NET application, the target computer should have .NET framework installed.) The .NET framework converts the calls to .NET class libraries to the corresponding APIs of the Operating system.

SOURCE COMPUTER		TARGET COMPUTER (client)
C# VB .NET J# Perl **(Compile to Intermediate Language) (MSIL)**	MSIL → → → → → → →	Target (client) computer should have **.NET framework for Windows** installed. **CLR** converts MSIL into native platform code.

[2] Class Libraries :

Class library is the second major entity (part) of the .NET Framework which is designed to integrate with the common language runtime. The class library consists of lots of prewritten code that all the applications created in VB .NET and Visual Studio .NET will use. The presence of many libraries in .NET Framework can assist you in developing your applications in a faster, cheaper and easier manner. The code for all the elements like forms, controls and the rest in VB .NET applications actually comes from the class library.

This class library constitutes various predefined functional sets that are very useful while developing the applications by developers. **There are three main components in this class library and they are :**
[1] ASP.NET. [2] Windows Forms. & [3] ADO.NET.



1.3 WHAT IS VISUAL STUDIO .NET ?

The new Visual Studio.NET is a common development environment for developing, testing & debugging web application, desktop application, mobile application etc. For developing application Visual Studio.NET support languages including C# (C Sharp), VB.NET, J# and managed C++. .NET is a common platform for all the .NET supported languages.

Using Visual Studio .NET we can develop Window Form based application (Desktop application), ASP .NET Web application (Dynamic website), Smart Device application (Mobile or PDA), Console application (DOS base), Web services etc.

1.4 WHAT IS VISUAL BASIC .NET?

Visual Basic .NET is a pure object oriented programming language implemented on Microsoft .NET framework. It makes programmer's life very easy.

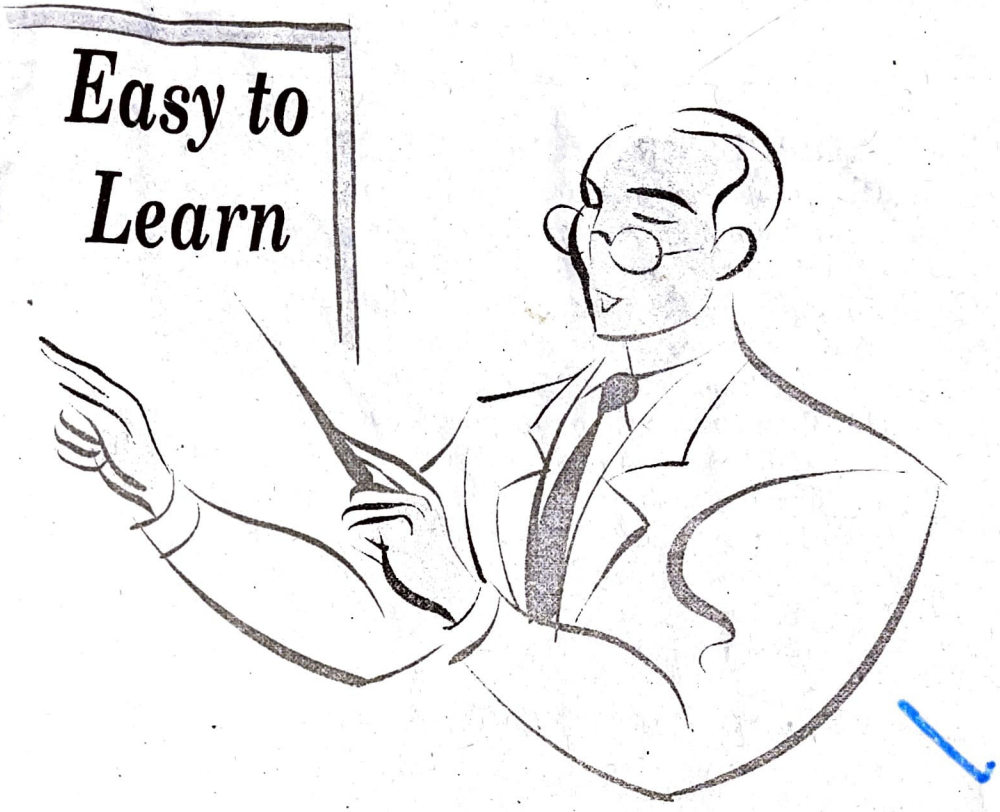
Visual Basic .NET provides the easiest, most productive language and tool for rapidly building Windows and Web applications. Visual Basic .NET comes with enhanced visual designers, increased application performance, and a powerful integrated development environment (IDE). It also supports creation of applications for wireless, Internet-enabled hand-held devices.

➢ **Versions Of Visual Basic .NET :**

- Visual Basic .Net (VB7) Released in 2002
- Visual Basic .Net 2003 (VB7.1)
- Visual Basic .Net 2005 (VB8.0)
- Visual Basic .Net 2008 (VB9.0)
- Visual Basic .Net 2010

➢ **Visual Basic .NET has following features :**

- Rich set of Classes: Visual Basic comes with thousands of built-in classes.
- Provides fully Object Oriented Programming environment.
- Multi Language & multi device support.
- Powerful, Flexible, Simplified Data Access with ADO .NET class.
- XML support: It supports for writing, manipulating & transforming XML documents.
- Simplified Deployment
- With an improved integrated development environment (IDE) you can build robust applications quickly & easily.



UNIT 2

VISUAL BASIC .NET PROGRAMMING

2.1 Computer Application
2.2 What is IDE? (Integrated Development Environment)
2.3 Data Type of VB .NET 2005
2.4 Composite Data Types OR User Define Data Types
2.5 Working with ToolBox Controls
2.6 Working with Menus & Dialog Boxes
2.7 Exception Handling: (Trapping Errors)
2.8 Using Module & Procedure
2.9 Using Arrays & Collections

2.1 COMPUTER APPLICATION

Examples of computer application are Inventory Management System, Payroll Management System, Library Management System, School Management System, AHMS (Automatic Hospital Management System), HRMS (Human Research Management System), Banking Management System, Reservation System etc.

Any computer application divides into two parts :

[1] Program. [2] User Interface.

[1] Programs : Computer need clear cut instructions to tell them what to do? When to do? And how to do? Program is a set of instructions that is used to carry out specific task.

[2] User Interface : Users interact with a computer application via user interface. Windows program use **controls** to provide easy and pleasant user interface. Good user interface will be easy to learn, easy to use, attractive. It means more users friendly.

2.2 WHAT IS IDE? (INTEGRATED DEVELOPMENT ENVIRONMENT)

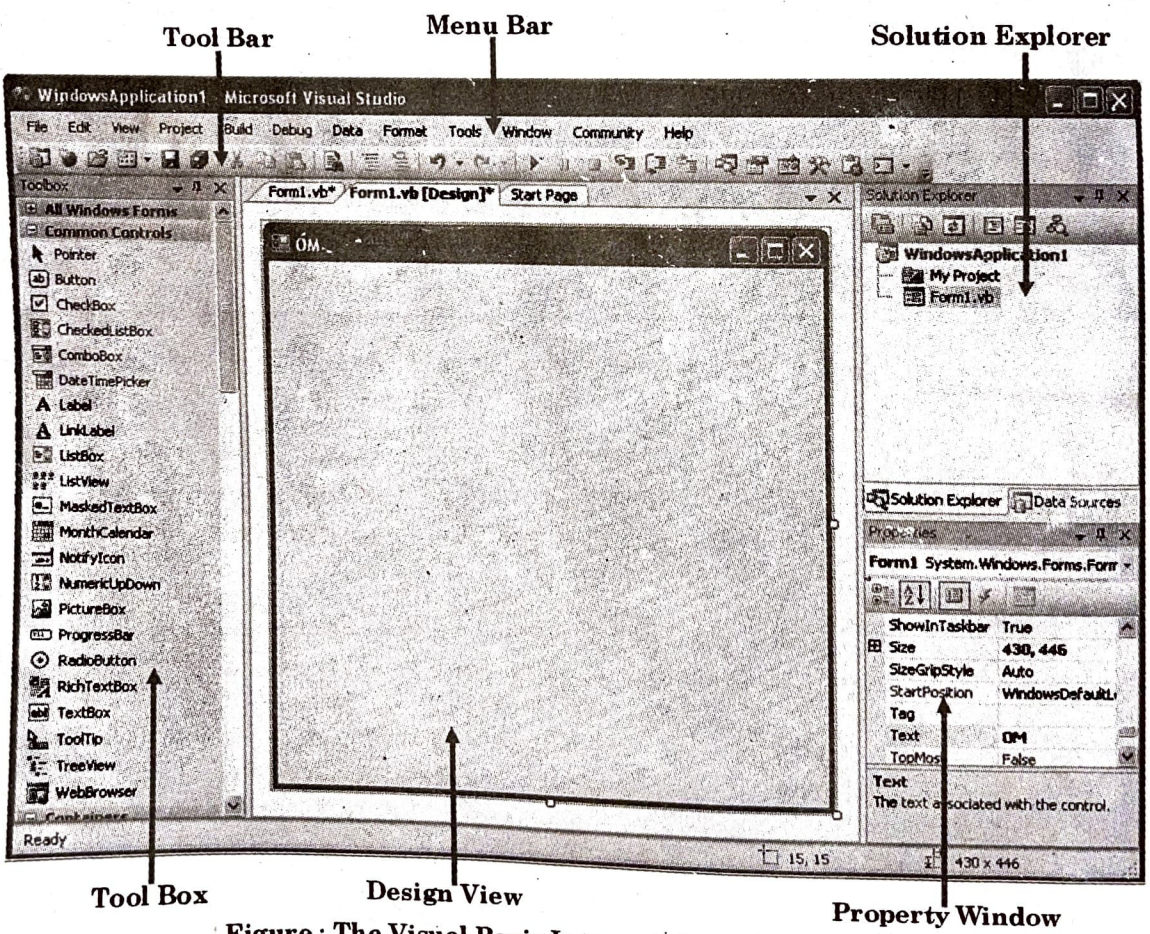


Figure : The Visual Basic Integrated Development Environment.

- IDE stands for Integrated Development Environment. It is as environment design to help developers to build solutions in a quick & efficient way.
- The IDE is what you see when you start Visual Basic, and it's where you develop, edit, compile & debug your applications.
- Visual Studio .NET provides an environment that is common to all Visual Studio languages such as such as VB, Visual C#, Visual J#, Visual C++.

The Visual Basic Integrated Development Environment consist of :

[1] The Menu System
[2] Toolbars
[3] The Toolbox
[4] The Solution Explorer
[5] The Server Explorer
[6] The Properties Window
[7] Form Designer View
[8] Code Designers
[9] IntelliSense

[1] The Menu System : Menu is the one of the common elements. It contains options like File, Edit, View, Project, Build, Debug, Data, Window, etc. The menu system changes as you make selections in the rest of the IDE. For example if you are at form's design view, you'll see a Format menu in the menu bar. If you then select the code view, Format menu disappears

[2] Toolbars : The toolbar is a shortcut to the commonly used commands. By default Standard toolbar is appear on visual studio .NET screen. We can add as well as customize the toolbar as per need.

[3] The Toolbox : (Ctrl+Alt+X) : Provides a set of controls divided into various categories like Common controls, All Windows Forms, Containers, Menus & Toolbars, Data, Components, Printing, Dialogs, Crystal reports & and General.

[4] The Solution Explorer : (Ctrl+Alt+L) : Like Window Explorer, this window gives the list of all the items of the solution you're working with, including all the projects in it, and the items in those projects like forms, classes, database etc.

[5] The Server Explorer (Visual Database Tool) : (Ctrl+Alt+S) : Server Explorer is a great tool to Create, modify, delete, and refresh database connections, Create new database objects, such as diagrams, tables, views, triggers, stored procedures, and user-defined types, View and access database objects.

[6] The Properties Window :(F4) : It will display the list of properties for the selected control. It is use to view or set the property at design time.

[7] Form Designer View :(Shift + F7) : Forms are the windows on which controls are placed. It is the "base" on which user interface is build.

[8] Code Designer : (F7) : Code Designer is used to write or edit the code. You can also switch between Form designer view and Code designer view using the Designer and Code items in the View menu, or you can use the top two buttons at left in the Solution Explorer.

[9] IntelliSense : IntelliSense is what's responsible for those boxes that open as you write your code, listing all the possible options and even completing your typing for you.

IntelliSense is made up of a number of options, including :

[1] List Members : Lists the members of an object.

[2] Parameter Info : Lists the arguments of procedure calls

[3] Quick Info : Displays information in tool tips as the mouse rests on elements in your code.

[4] Complete Word : Completes typed words.

[5] Automatic Brace Matching : Adds parentheses or braces as needed.

- **Variable :** Variables are used to store (hold) values temporarily during the execution of the program. Variable has a name & data type. Dim statement is used to declare variable.

Ex. You can declare variable as follows : Dim Area As Integer
Area = 100

OR In following example you can also declare & initialize the value as follows:
Dim Area As Integer = 100
You can also declare multiple variables as follows :
Dim No1, No2, No3 As Integer

- **Constant :** Constants store values that remain same (constant) throughout the execution of an application. Const statement is used to declare the constant. There is user define constants & system define constants (intrinsic constants).

const ConstantName [As DataType] = initialization

For Example : Const Pi As Double = 3.14
Const Msg = "OMSHIVSAI"

- **Enumeration :** Constants store values that remain same (constant) throughout the execution of an application. **Enumeration is a set of related constants.**

group all constant declare then enum used

Enumerations provide a convenient way to work with sets of related constants, and to associate constant values with names. Visual Basic provides a number of useful enumerations (pre-defined enumeration). For example MsgBoxResult Enumeration, MsgBoxStyle Enumeration, DateFormat Enumeration, etc.

Enum statement is used to create Custom Define Enumeration.

Syntax :
```
Enum enumeration name [ As data type ]
    member list
End Enum
```

default integer

The data type must be one of the integer types. You can specify Byte, Integer, Long, SByte, Short, UInteger, ULong, or UShort. By default is Integer.

two type
① Predefined
② User defined

Ex.1 : The following example assigns a constant to every day of the week.

```
Enum Days
    Sunday = 1
    Monday = 2
    Tuesday = 3
    Wednesday = 4
    Thursday = 5
    Friday = 6
    Saturday = 7
End Enum
```

To use constant in the enumeration, you refer to like this :

```
Msgbox(Days. Sunday)        'It will display 1
Msgbox(Days.Thursday)       'It will display 5
```

If you do not initialize a constant in an enumeration, then by default, the first constant in an enumeration is initialized to 0, and subsequent constants are initialized to a value of one more than the previous constant.

Ex.2 :

```
Enum Days
    Sunday
    Monday
    Tuesday
    Wednesday
    Thursday
    Friday
    Saturday
End Enum
Msgbox(Days. Sunday)        'It will display 0
Msgbox(Days.Thursday)       'It will display 4
```

➤ **Comment :** ' REM

elect
ck comm
and
button

- Comment is a remark or information embedded in the programme, often explanatory text that is ignore by the compiler. For example, What is the purpose of the procedure? What is the use of variables? etc. Comment is used for documentation. Comments in VB starts with an apostrophe (') & Visual Basic ignore whatever follows the apostrophe on the line.

  For Example : ' The following procedure calculates the average.

- You can also give comment using REM keyword followed by a space.

  For Example : REM The following procedure calculates the average.

- You can also give comment using Standard toolbar. First select the lines which you want to give comment, & then select the Comment button from the Standard toolbar.

- To remove the comment there is also button on the Standard toolbar.

➢ **Multiple Statements on one line :**
Colon character is used to write multiple statements in one line.
For Example : Dim A, B, C As Integer
A=10 : B=70: C=60

➢ **One statement on Multiple Line :**
You can write one statement on multiple lines (to continue one statement onto the next line), by using space followed by an underscore character.
For Example : Dim Str As String
Str = "Please enter the Date Of Birth" _
& " in DD/MM/YYYY Format"

2.3 DATA TYPE OF VB .NET 2005

Data Type means, what kind of data can be stored in a variable & how that data is stored. memory storage size allot a variable.

➢ **Elementary Data Types :**
[1] Numeric Data Types
[2] Character Data Types
[3] Miscellaneous Data Types

[1] Numeric Data Types :

(without decimal part)

(a) Integral Numeric types :
It represents only whole numbers **without fractional parts.** (Positive, negative, & zero)
Byte, SByte, Short, Ushort, Integer, UInteger, Long, Ulong

(with decimal part)

(b) No integral Numeric types :
It represents numbers **with both integer and fractional parts.**
Ex. : Single, Double, Decimal

1byte : 8bit

[1] Numeric Data Types :

(a) Integral Numeric Types : It represents only whole numbers without fractional parts. (Positive, negative, & zero).

negative not possible

→ **Byte : (1 Byte) :** Its storage size is 1 Byte. It holds unsigned 1 byte integers ranging in value from 0 to 255. The default value of Byte is 0.

→ **SByte : (1 Byte) :** Its storage size is 1 Byte. It holds Signed 1 byte integers ranging in value from -128 to 127. The default value of SByte is 0.

→ **Short : (2 Byte) :** Its storage size is 2 Byte. It holds signed 2 bytes integers ranging in value from -32,768 through 32,767. The default value of Short is 0.

→ **UShort : (2 Byte) :** Its storage size is 2 Byte. It holds unsigned 2 bytes integers ranging in value from 0 through 65,535. The default value of UShort is 0.

→ **Integer : (4 Byte) :** Its storage size is 4 Byte. The default value of Integer is 0.

→ **UInteger: (4 Byte) :** It is Unsign Integer. Its storage size is 4 Byte. The default value of UInteger is 0.

→ **Long : (8 Byte) :** Its storage size is 8 Byte. The default value of Long is 0.

→ **ULong: (8 Byte) :** (Unsign Long) Its storage size is 8 Byte. The default value of ULong is 0.

(b) Nonintegral Numeric types : It represents numbers with both integer and fractional parts.(With Decimal point).

→ **Single : (4 Byte) :** Its storage size is 4 Byte. Holds signed IEEE 32-bit (4-byte) single-precision floating-point numbers. The default value of Single is 0.

→ **Double : (8 Byte) :** Its storage size is 8 Byte. Holds signed IEEE 64-bit (8-byte) double-precision floating-point numbers. The default value of Double is 0.

→ **Decimal : (16 Byte) :** Its storage size is 16 Byte. The number of digits to the right of the decimal point can range from 0 to 28. The default value of Decimal is 0.

[2] Character Data Types :

(a) Char (2 Byte) : It is used to store only one character. It occupies 2 Byte.

(b) String : The String data type is used to store text. String can contain a series of letters, numbers, spaces, and other characters. Its storage size is depends on implementing platforms. A string can contain from 0 to approximately 2 billion (2 ^ 31) Unicode characters.

The default value of String is Nothing (a null reference). Note that this is not the same as the empty string (value "").

Framework Type : The corresponding type in the .NET Framework is the **System.String** class.

Example : You can declare **Variable Length String** variable as follows:

```
Dim StrMsg As String
StrMsg = "OMSHIVSAI"
```

OR In following example you can also declare & initialize the string as follows : `Dim StrMsg As String = "OMSHIVSAI"`

→ There are quite number of string-handling functions built into Visual Basic 2005. For example Left, Mid, Right etc

→ Besides the string-handling functions built into Visual Basic 2005, many .Net Framework functions are built into the **String** class that Visual Basic 2005 uses. For example we can convert a text string to upper case either by using Visual Basic 2005 UCase function, or the **String** class's ToUpper method.

[3] Miscellaneous Data Types :

(a) Boolean : It is used to store either **True** or **False** value. The default value of **Boolean** is **False**. Its storage size (data width) depends on the implementing platform.

(b) Date (8 Byte) : Its storage size is 8 Bytes. Its value ranging from 1st January 0001 to 31st December 9999. It holds date values, time values, or date and time values. The default value of Date is 0:00:00 (midnight) on 1st January 0001.

(c) Object : An Object data type is a universal data type, which can hold any type of data. The default value of Object is Nothing (a null reference). It occupies 4 Bytes on 32 bit platform & 8 Bytes on 64 bit platform.

2.4 COMPOSITE DATA TYPES OR USER DEFINE DATA TYPES

In addition to the elementary data types Visual Basic also supports, composite data types such as structures, arrays, and classes. You can build composite data types from elementary types and from other composite types.

[1] Option Explicit Statement :

When Option Explicit statement is set to ON, you must explicitly declare all variables using the Dim or ReDim statements. By default it in ON, so you need to declare all the variables before they are used. When Option Explicit statement is set to OFF, all undeclared variables are treated as Object type.

To set Option Explicit in the integrated development environment (IDE) :

(1) Click on the Options form the Tools menu.

(2) Open the Projects and Solutions node.

(3) Select VB Defaults.

(4) Give the desire Option Explicit setting that is either ON or OFF.

Ex.

```
'Force explicit variable declaration.
Option Explicit On
.........
.........
Dim A As Integer
A = 10
'The following assignment statement produces a COMPILER ERROR because
'The variable Square1 is not declared and Option Explicit is set to ON.:
Square1 = 10 ' It will causes ERROR
'If Option Explicit is set to ON, then we must declare the variable before it is
used.
```

[2] Option Strict Statement :

If Option Strict Statement is set to ON, Visual Basic prevents (disallows) implicit narrowing data type conversion. Visual Basic will not automatically convert data type when you assign a value of one type to a variable of another data type. Then Visual Basic will consider that an error. If Option Strict Statement is set to OFF, Visual Basic will do automatically implicit narrowing data type conversion. By default it in OFF.

To set Option Strict in the integrated development environment (IDE) :

(1) Click on the Options form the Tools menu.

(2) Open the Projects and Solutions node.

(3) Select VB Defaults.

(4) Give the desire Option Strict setting that is either ON or OFF.

Ex.

```
'Force explicit variable declaration.
Option Strict On
.........
.........
Dim I As Integer
Dim D As Double
D = 3.14159
I = D
' A compilation ERROR occurs if such a narrowing conversion fails.
' It will disallows implicit conversions from Double To Integer
' To solve above problem either you have to make Option Strict Off 'or you have to
'do specific type conversion as follows.
    I = Cint(D)    ' Convert To Integer.
```

on strict
Implicit conversion
error
data type not list

[3] Option Compare Statement :

The Option Compare statement is used to specify the string comparison method, that is either Binary or Text for a class, module or structure. By default text comparison method is Binary.

Option Compare Binary

OR

Option Compare Text

Option Compare Binary, the following text sort order is produced.

A < B < E < Z < a < b < e < z

Option Compare Text, the following text sort order is produced.

(A=a) < (B=b) < (E=e) < (Z=z)

To set Option Compare in the integrated development environment (IDE)

(1) Click on the Options form the Tools menu.

(2) Open the Projects and Solutions node.

(3) Select VB Defaults.

(4) Give the desire Option Compare setting that is either Binary or Text.

Ex.
```
' Set the string comparison method to Binary
Option Compare Binary
MsgBox("AAA" < "aaa")  'It will return True, because A < a in Binary comparison
```

Ex.
```
' Set the string comparison method to Text
Option Compare Text
MsgBox("AAA" < "aaa")  'It will return False, because A=a in Text comparison
```

[4] Operators :

An operator is a symbol or character which allows us to manipulate data

(1) Arithmetic operators are ^, *, /, \, +, -, Mod
(2) Assignment operators are =, ^=, *=, /=, \=, +=, -=, &=
(3) Comparison or Relational operators are =, <>, <, <=, >, >=, Is, Like
(4) String Concatenation operators are &, +
(5) Logical or Bitwise operators are Not, And, AndAlso, Or, OrElse, Xor
(6) Miscellaneous operators are AddressOf, GetType

- AndAlso & OrElse operator performs logical short-circuiting. In AndAlso, if the first operand of the expression is False, then the remaining logical expression is not evaluated. It saves time. Similarly, In OrElse, if the first operand of the expression is True, then the remaining logical expression is not evaluated. It saves time.
- AddressOf operator is used to return address of procedure.
- GetType operator is used to get information about the type of object, variable, properties, methods & events.

[5] Control Flow Statement or Conditional Statements :

Decision structure or Control Structure
(1) If.....Then....Else Statement
(2) Select Case Statement

(1) If.....Then....Else Statement : Control flow statement or conditional statements are used to change the flow of the programme based on condition. Condition is evaluated, & depending on the result of the condition, the block of code is executed.

Syntax :
```
If condition1 Then
    Statements
ElseIf condition2 Then
    Statements
.............
.............
Else
    Statements
End If
```

When a multiple-line **If...Then...Else** is encountered, First condition is tested. If condition is **True**, the statements following **Then** are executed. If condition is **False**, next **ElseIf** condition is tested. If that condition is true, the statements following the associated **Then** are executed. If no ElseIf condition evaluates to **True**, the statements following **Else** are executed.

Ex.
```
Dim No As Integer
No = 55000
If No < 10 Then
    MsgBox("One Digit No.")
ElseIf No < 100 Then
    MsgBox("Two Digit No.")
ElseIf No < 1000 Then
    MsgBox("Three Digit No.")
Else
    MsgBox("More Than Three Digit No.")
End If
```
'Output will be **More Than Three Digit No.**

(2) Select Case Statement : It is used to execute one of several groups of statements, depending on the value of an expression. In If....Then.....Else statement every If & ElseIf condition is evaluated, while in Select Case statement initially only one time TestExpression is evaluated, and the result of that TestExpression is compare for every cases. By using Select Case statement, code is easier to read & maintain as compare to If... Then.... ElseIf statement.

Syntax :
```
Select Case   TestExpression
        Case  ExpressionList1
              Statements...
        Case  ExpressionList2
              Statements...
        ....................
        ....................
        Case  Else
              ElseStatements
End Select
```

Ex.
```
Dim No As Integer
No = 55000
Select Case No
    Case 0 To 9
        MsgBox("One Digit No.")
    Case Is <= 99
        MsgBox("Two Digit No.")
    Case 100 To 999
```

```
            MsgBox("Three Digit No.")
        Case Else
            MsgBox("More Than Three Digit No.")
    End Select
' Output will be More Than Three Digit No.
```

[6] Looping structure or Looping Statement : Looping statements are used to execute a series of statements repeatedly, until a specified condition is **True or** until a condition is **False** or a specified number of times.

VB .NET supports following Loop structure :

(1) For....Next Loop Statement
(2) Do....Loop Statement
(3) While....End While Statement
(4) For Each....Next Statement

(1) For....Next Loop Statement : For....Next Loop Statement is used to execute a series (block) of statements a specified number of times.

Syntax :

```
For Counter [As datatype] = start To end [Step step]
    [statements]
    [Exit For]
    [statements]
Next [Counter]
```

The Counter variable is originally set to start automatically when the loop begins. Each time through the loop, Counter is incremented or decrement by step and when Counter equals end, the loop ends. If you do not specify the step value, by default it is set to 1.

Exit For statement is used to immediately exits the **For** loop in which it appears. Execution continues with the statement following the **Next** statement. It Transfers control out of the For loop.

Ex. **To print Odd numbers between 1 to 20.**

```
Dim S As String
For I As Integer = 1 To 20 Step 2
    S = S & I & " "
Next
MsgBox(S)
' Output will be 1 3 5 7 9 11 13 15 17 19
```

(2) Do....Loop Statement : Repeats a block of statements while a **Boolean** condition is **True** or until the condition becomes **True**.

Use a **Do...Loop** structure when you want to repeat a set of statements an indefinite number of times, until a condition is satisfied. If you want to repeat the statements a set number of times, the For...Next Statement is usually a better choice.

```
Do
While condition
    [Statements]
    [Exit Do]
    [Statements]
Loop
```

OR

```
Do Until condition
    [Statements]
    [Exit Do]
    [Statements]
Loop
```

While : Repeat the loop till condition is **True.**
Until : Repeat the loop till condition is **False.**

Following Do.....loop statement checks the condition at the end of the loop, so that means so loop statements will executed at least once.

```
Do
    [Statements]
    [Exit Do]
    [Statements]
Loop
While condition
```

```
Do
    [Statements]
    [Exit Do]
    [Statements]
Loop
Until condition
```

Exit Do statement is used to immediately exit the **Do** loop in which it appears (to escape the loop). Execution continues with the statement following the **Loop** statement.

(3) While....End While Statement : It executes the series of statements as long as the given condition is **True**.

```
While condition
    [Statements]
    [Exit While]
    [Statements]
End While
```

Exit While statement is used to immediately exits the **While** loop in which it appears. Execution continues with the statement following the **End While** statement.

(4) For Each...Next Statement : Repeats a group of statements for each element in a collection or array. You can use a For Each loop to iterate through every element of a collection or array. You need not to get loop indices.

```
For Each element [As datatype] In group
    [Statements]
    [Exit For]
    [Statements]
Next [element]
```

element : Used to iterate through the elements of the collection.
group: Required. Object variable. Refers to the collection over which the statements are to be repeated.
Exit For statement is used to transfers control out of the For Each loop. Execution continues with the statement following the **Next** statement.

Ex. 1 :
```
'To disable all the textbox of the current form.
    For Each C As Control In Me.Controls
        If TypeOf C Is TextBox Then
            C.Enabled = False
        End If
    Next
```

Ex.2 :
```
'Declare an array
    Dim StudArr(4) As String
'Store the data
    StudArr(0) = "OM"
    StudArr(1) = "SHIVANGI"
    StudArr(2) = "PRIYAM"
    StudArr(3) = "DATTAVI"
    StudArr(4) = "JAYANI"
'Following For Each.....Next loop iterate through a string 'collection, in this
example, it is used in an array. You have to 'create a control variable that
is of the same type as an element 'in the array. You can search as follows
    For Each Studname As String In StudArr
        If Studname = "Priyam" Then
            MsgBox("Found")
        End If
    Next
' You can add all the elements of array to the ListBox1
    For Each Studname As String In StudArr
        ListBox1.Items.Add(StudName)
    Next
```

[7] With ...End With statement : Using With....End With statement, you can refer to particular object only once, instead of repeating reference to a single object. It is not a Loop.

For example, to change a number of different properties on a single object, place the property assignment statements inside the With...End With, referring to the object only once instead of for each property assignment.

```
With Button1
    .Text = "Exit"
    .BackColor = Color.LightCyan
    .ForeColor = Color.Red
    .Height = 30
    .Width = 80
End With
```

[8] **Namespace : A Namespace is a group of related classes.** It is a good practice to group related classes into a namespace when you create a class library. Namespaces prevent ambiguity and simplify references when using large groups of objects such as class libraries.

The main advantage of using namespaces is, to avoid conflicts (clashes) when you have multiple classes with the same name. For example, .NET comes with a class called "**Button**" class. Suppose you also write another class called "**Button**" in your application and if you refer to the class "**Button**" class, how does the compiler know which Button class you are referring to?

You can refer to the .NET Button Class using the fully qualified name, including the namespace like this : **System.Windows.Forms.Button**

And you can refer to your own Button class like this : **MyApplication.Button**, where MyApplication is your namespace. Each project has its own root namespace, and by default, Visual Basic uses the name of the project for the root namespace.

So, namespaces allow you to avoid name conflicts.
All classes in .NET class library are grouped into namespaces. There are many such classes, organized into various namespaces. Overviews of some of those namespaces are as follows :

- **System** namespace is at the root of the hierarchy. It include fundamental classes & base classes that define commonly used data types, events and event handlers, interfaces, attributes, exceptions, and so on.
- **System.Windows.Forms** namespace contains classes for creating Windows based application.
- **System.Data** namespace contains classes to represent the ADO.NET architecture for access & management of data.
- **System.IO** namespace contains classes that allow reading & writing to file. It also provides basic file & directory support.

- **System.Net** namespace contains classes for sending & receiving data over a network using almost all the available network protocol.
- **System.XML** namespace contains classes to work with XML.
- **System.Collections** namespace contains interfaces and classes that define various collections such as lists, queues, arrays, hash tables, and dictionaries.

➢ **Creating Namespaces :**

When you're creating a large number of classes, it can be helpful to divide them up into their own namespaces. Namespaces prevent ambiguity and simplify references. To create your own namespace, you can use the Namespace statement. Here's an example that declares two namespaces, one inside another :

```
Namespace N1
    Class a
        ' Insert class definition.
    End Class
End Namespace
```

[9] Imports Statement : You can use Imports statements to import a namespace so you don't have to qualify items in that namespace by listing the entire namespace when you refer to item.

Imports [aliasname =] namespace

OR

Imports [aliasname =] namespace.element

The following example imports the Microsoft.VisualBasic.Strings class and assigns an alias, str, that can be used to access the Left method.

```
' Place Imports statements at the top of your program.
Imports str = Microsoft.VisualBasic.Strings
```

To access the Left method, it can use the alias **str** instead of the entire qualification string **Microsoft.VisualBasic.Strings**.

```
MsgBox(str.Left("OMSAIRAM", 2))
```

[11] VB .NET File Extension :

- **.sln** : Solution File, which stores the solution's configuration. This contains one or more projects (similar to VB6 group file).
- **.vbproj** : Represents a Visual Basic project.
- **Vbproj.user** : Contains project level user options.
- **.vb** : Form's code file. (Contains source code).
- **AssemblyInfo.vb** : It contains general information about an assembly, including version information.

The AssemblyInfo.vb file contains extra information, or metadata, about your application.

- **Bin** : Directory for binary executables.
- **Obj** : Directory for debugging binaries.

[12] Inbuilt Functions :

(1) **String Manipulation Functions :** Inbuilt String handling functions are supported by

(a) VB .NET as well as

(b) .NET framework methods are built into the **String** class

Example of VB .NET String handling functions :

(static)

	VB .NET String handling functions	NET framework String class methods
To Find length of a string use :-	Len	String.Length
Format a string use :-	Format	String.Format
Convert to lowercase or uppercase :-	Lcase, Ucase	String. ToLower, String. ToUpper,
Get a substring use :-	Mid	String.SubString
Insert a substring use :-	-	String.Insert
Trim leading or trailing spaces use :-	LTrim, RTrim, Trim,	String.TrimStart, String.TrimEnd, String.Trim,
Search strings use :-	InStr	String.Chars, String.IndexOf,
Compare two strings use :-	StrComp, String.Compare	String.Equals, String.CompareTo
Convert strings use :-	StrConv, CStr	String. ToString
Returns an array of substrings from one string use :-	–	String.Split

Ex. 1 :

```
Dim Str1 As String = "OM SHIV SAI"
Dim Str2 As String = "Whosoever puts their feet on " _
& "SHIRDI soil their suffering will come to an end."
' VB .NET String handling functions (Convert to Lower
' case).
MsgBox(LCase(Str1)) ' Gives output : om shiv sai
' OR
' .NET framework String class methods (Convert to
' Lower case).
```

```
MsgBox(Str1.ToLower())      ' Gives output : om shiv sai
' VB .NET String handling functions (To get the
' SubString).
MsgBox(Mid(Str1, 4, 4))     ' Gives output : SHIV
' OR
' .NET framework String class methods (To get the
' SubString).
MsgBox(Str1.Substring(3, 4))     ' Gives output : SHIV
```

Ex. 2 : Split function returns an array of substrings from one string.

```
Dim Array1() As String = Split("OMSAIRAM : SABKA MALIK EK.")
MsgBox(Array1(0))          ' Gives output : OMSAIRAM
MsgBox(Array1(1))          ' Gives output : :
MsgBox(Array1(2))          ' Gives output : SABKA
MsgBox(Array1(3))          ' Gives output : MALIK
MsgBox(Array1(4))          ' Gives output : EK
```

Ex. 3 :

```
Dim Array1() As String = Split("DEV,DATTA,DATTAR", ",")
MsgBox(Array1(0))          ' Gives output : DEV
MsgBox(Array1(1))          ' Gives output : DATTA
MsgBox(Array1(2))          ' Gives output : DATTAR
```

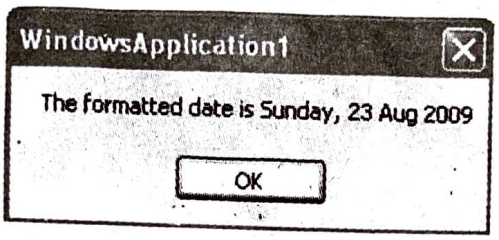
(2) Date & Time Functions : You can specify the time value in either 12-hour or 24-hour format, for example #1:15:30 PM# or #13:15:30#. However, if you do not specify either the minutes or the seconds, you must specify AM or PM. Use the Date data type to contain date values, time values, or date and time values.

Datetime Class :

→ **Property :**

```
Dim DT1 As Date = #11/12/2009 4:30:45 AM# ' Formate of Date is
' mm/dd/yyyy
    MsgBox(DT1)      ' Gives output : 12/11/2009 4:30:45 AM
    MsgBox(DT1.Day)  ' Return Day of Month. Gives output : 12
    MsgBox(DT1.Month)  ' Return Month. Gives output : 11
    MsgBox(DT1.Year)   ' Return Year.  Gives output : 2009
    MsgBox(DT1.DayOfYear)  ' Return Day of Year.  Gives output : 316
    MsgBox(DT1.Hour)  ' Return Hour. Gives output : 4
    MsgBox(DT1.Minute)  ' Return Minute. Gives output : 30
    MsgBox(DT1.Second)  ' Return Second. Gives output : 45
    MsgBox(DT1.Millisecond)  ' Return Millisecond. Gives output : 0
' Now() returns a current date and time according to your
' system.
    MsgBox(Now())    ' Gives output : 12/11/2009 10:46:14 AM
' Today() returns a current date
```

```
    MsgBox(Today( ))    ' Gives output : 12/11/2009
' TimeOfDay() returns current time.
    MsgBox(TimeOfDay())    ' Gives output : 11:56:45 AM
    MsgBox("The formatted date is " & Format(#8/23/2009#, "dddd, d MMM
    yyyy"))
' Gives output as follows.
```



→ **Method :**

```
Dim DT1 As Date = #11/12/2009 4:30:45 AM#    ' Formate of Date is '
mm/dd/yyyy
MsgBox(DT1.AddDays(11)) 'Argument can be -ve or +ve
' Gives output : 23/11/2009 4:30:45 AM
MsgBox(Today.AddDays(11)) 'Argument can be -ve or +ve.  Today ' is
12/11/2009
' Gives output : 23/11/2009
MsgBox(DT1.AddMonths(1))    ' Gives output : 12/12/2009
MsgBox(DT1.AddYears(1))    ' Gives output : 12/11/2010
MsgBox(DT1.AddHours(1))    ' Gives output : 12/11/2009 5:30:45 AM
MsgBox(TimeOfDay.AddHours(1))  'Current time is 1:14:02 PM
' Gives output : 2:14:02 PM
MsgBox(DT1.AddMinutes(9))    ' Gives output : 39
MsgBox(DT1.AddSeconds(9))    ' Gives output : 54
MsgBox(DateTime.DaysInMonth(2009, 11))    ' Gives output : 30
MsgBox(DateTime.IsLeapYear(2009))    ' Gives output : False
```

(3) Other Date & Time function :

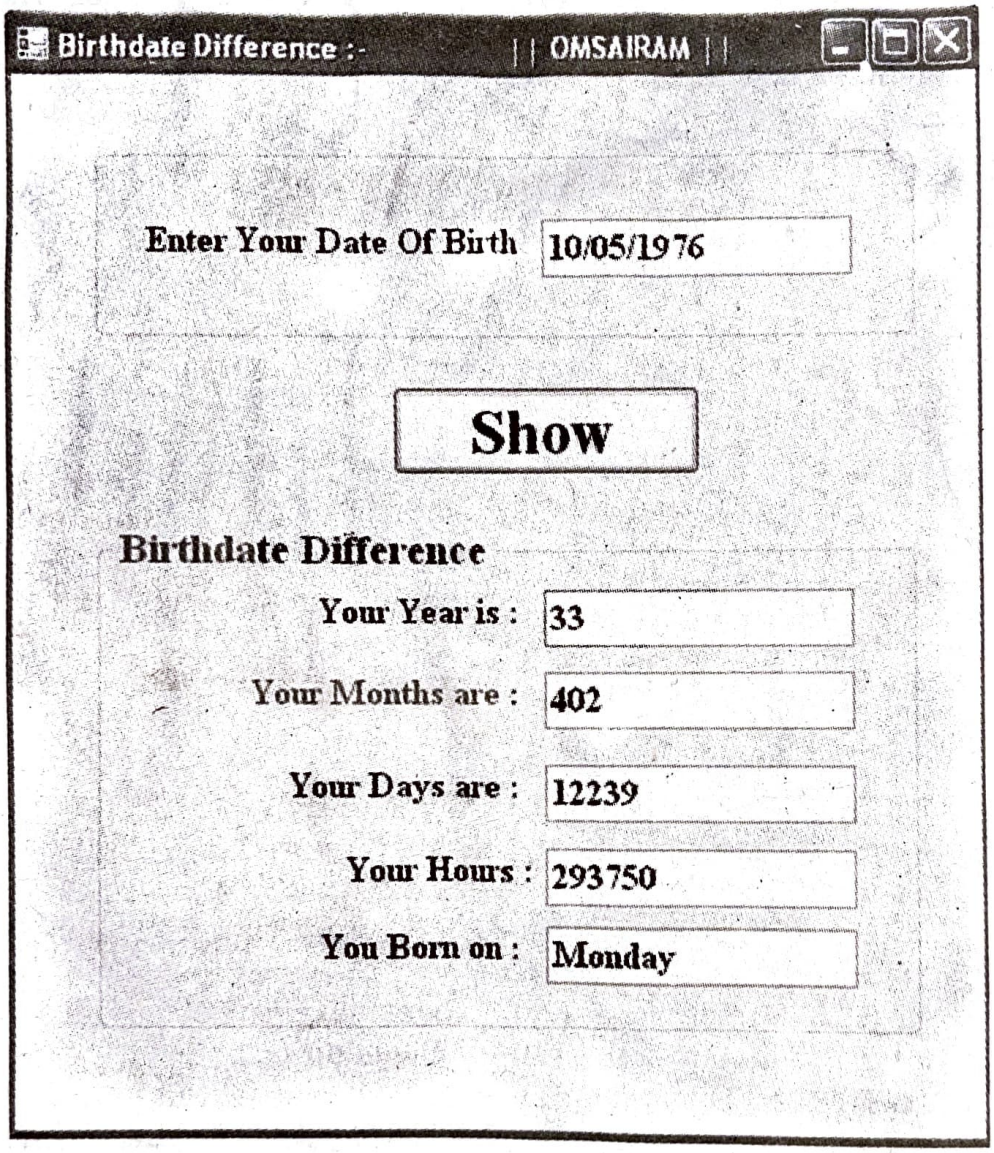
→ **DateAdd :** Returns a Date value to which a specified time interval has been added.

Ex.

```
Dim DT1 As Date = #11/12/2009#    ' Formate of Date is mm/dd/
yyyy
MsgBox(DateAdd(DateInterval.Month, 5, DT1))  ' Gives output : 12/
04/2010
```

→ **DateDiff :** Returns a value specifying the number of time intervals between two Date values.

Ex. Design the interface as shown in figure.




```
'Write down coding in Button1 Click Event as shown below
Dim DOB As Date = Format(CDate(TextBox1.Text), "dd/MM/yyyy")
    TextBox2.Text = DateDiff(DateInterval.Year, DOB, Now)
    TextBox3.Text = DateDiff(DateInterval.Month, DOB, Now)
    TextBox4.Text = DateDiff(DateInterval.Day, DOB, Now)
    TextBox5.Text = DateDiff(DateInterval.Hour, DOB, Now)
    TextBox6.Text = WeekdayName(DOB.DayOfWeek, False,
    FirstDayOfWeek.Monday)
'Output is as shown in figure above.
```

Timespan Class :

- This class contains properties & methods related to time interval.
- It represents time interval in different unit measured as a positive or negative number of days, hours, minutes, seconds, and fractions of a second.
- A tick is equal to 100 nanoseconds, and the value of a TimeSpan object can range from MinValue to MaxValue.

(4) Math Functions :

```
MsgBox(System.Math.Abs(-108))          ' Output : 108
MsgBox(System.Math.Cos(90))            ' Output : -0.44807361612917
MsgBox(System.Math.Pow(5, 2))          ' Output : 25
MsgBox(System.Math.Sqrt(100))          ' Output : 10
MsgBox(System.Math.PI)                 ' Output : 3.14159265358979
MsgBox(System.Math.Log(100, 10))       ' Output : 2
MsgBox(System.Math.Sign(-100))         ' Output : -1
MsgBox(System.Math.Sign(0))            ' Output : 0
MsgBox(System.Math.Sign(100))          ' Output : 1
MsgBox(System.Math.Round(785.7, 0))' Output : 786
```

(5) Data Type Conversion Functions :

Function Name	Convert into
CByte	Convert to **Byte** data type.
CShort	Convert to **Short** data type.
CInt	Convert to **Int** data type.
CLng	Convert to **Long** data type.
CSng	Convert to **Single** data type.
CDbl	Convert to **Double** data type.
CDec	Convert to **Decimal** data type.
CChar	Convert to **Char** data type.
CStr	Convert to **String** type.
CBool	Convert to **Bool** data type.
CDate	Convert to **Date** data type.
CObj	Convert to **Object** type.

Ex.

```
Dim STR As String = "12/11/2009"
Dim DT As Date
DT = CDate(STR)      ' Here CDate function convert String
                     ' data type to Date data type.
```

→ **Ctype Function :**
It is used t convert one data type to another data type.

```
Ex.   Dim STR As String = "12/11/2009"
      Dim DT As Date
      DT = CType(STR, Date) 'Here CDate function convert,
                            ' String data type to Date data type.
```

(6) Type Checking Functions :

→ **IsNumeric :** Returns True if the value is numeric.

```
Ex.   Dim STR As String = "OMSAIRAM"
      MsgBox(IsNumeric(STR))  ' Output : False
```

→ **IsDate :** Returns True if it is expression is valid date.

```
Ex.   Dim  DT As Date  = #11/12/2009#
      MsgBox(IsDate(DT))      ' Output : True
```

→ **IsNothing :** Returns True if the expression has not assigned a value.

```
Ex.   Dim B1 As Byte
      MsgBox(IsNothing(B1))   ' Output : False
```

→ **IsDBNull:** The System.DBNull value indicates that the Object represents missing or nonexistent data. DBNull is not the same as Nothing, which indicates that a variable has not yet been initialized. DBNull is also not the same as a zero-length string ("") (null string).

```
Ex.   Dim B1 As Object = 1
      MsgBox(IsDBNull(B1))    ' Output : False
      B1 = System.DBNull.Value
      MsgBox(IsDBNull(B1))    ' Output : True
```

→ **IsArray :** Returns a Boolean value indicating whether a variable points to an array.

```
Ex.   Dim MyArray(5) As Integer
      MsgBox(IsArray(MyArray))  ' Output : True
      Dim I As Integer
      MsgBox(IsArray(I))        ' Output : False
```

2.5 WORKING WITH TOOLBOX CONTROLS

Visual Basic controls are based on classes provided by the .NET Framework. Windows controls are bases on Control class. Control class is the base class of all windows controls. The Control class is in the **System.Windows.Forms** namespace. **All the controls have properties, methods & events.** Examples of controls are Label, TextBox, Button, CheckBox, RadioButton, ListBox, ComboBox, PictureBox, TreeView, ListView, etc. We can add controls at design time using toolbox or at run time through coding.

➔ **Property :** Characteristics of Control (Object) are called as property. Example is Name, Text, BackColor, ForeColor, Visible, Enable, Font etc. VB assign default property to every new control when you place on a form, you can change property of a control,

- At design time (using Property Window) OR
- At run time (through Coding)

➔ **Method :**

Method is a predefine function associated with controls (objects). OR
Method is a code that is built into interface components (control).
It hides the information detail. (Encapsulation)
Examples of methods are SetBounds, SendToBack, BringToFront, Focus, Clear, Copy, Cut, Paste, SelectAll etc

➔ **Event :** Event are response that are triggered (fired or execute or run), when user interact with control.

OR

Events are reaction of any action.

OR

An event is an action which you can respond (react) to, or "handle," in code. Events can be generated by a user action, such as clicking the mouse or pressing a key; by program code; or by the system. Examples of events are Click, DoubleClick, KeyPress, KeyDown, KeyUp, MouseDown, MouseEnter, MouseHover, MouseLeave, MouseUp, MouseMove, MouseWheel, TextChanged etc.

VB. NET is also called as an Event Driven Programming Language, because code is executed with response to event.

➔ **Advantages of Visual Programming :**

(1) Ready to use Controls. Programmer need not to write code to display require controls.
(2) The controls can be move, resize rename & delete if required.
(3) There is no restriction on the no. of controls that can be placed on the form. All components have ability to recognize events for example button known when it has been clicked.
Example of Visual programming tools (GUI Tools or Front End Tools) are VB, PB (Power Builder), D2K (Developer 2000)

➔ **Disadvantages of Visual Programming :**

(1) Highly graphical on nature, therefore it requires more memories.
(2) Visual development requires the computer of higher configuration. That is more RAM, Higher processor.
(3) It is used only with GUI operating system such as Windows.

[1] Label Control : **Label controls are used to display text or images that cannot be edited by the user.** Label controls are typically used to provide descriptive text for other controls. For example, you can use labels to add descriptive text to list boxes, text boxes, combo boxes, etc. Control class is the base class of all windows controls. The Label Control is in the **System.Windows.Forms** namespace.

Diagram

Label 1

➢ **Property of Label Control :** You can change property of a control At design time (using Property Window) OR At run time (through Coding).

(1) Name : Gets or sets the name of the control. It represents the name of the control. It is used for identification.

(2) Text : Gets or sets the current text of the control. It is the text that is displayed within the control.
To change this property at run time you have to write,
Lable1.Text = "OMSHIVSAI"

(3) BackColor : Gets or sets the background color for the control. To change this property at run time you have to write,
Label1.BackColor = Color.Purple

(4) ForeColor : Gets or sets the foreground color for the control. To change this property at run time you have to write,
Label1.ForeColor = Color.Red

(5) AutoSize : Gets or sets a value indicating whether the control is automatically resized to display its entire contents. By default it is set to True. This property is typically set to true when you use a Label control to display text of various lengths. To change this property at run time you have to write,
Label1.AutoSize = True

(6) Visible : Gets or sets a value indicating whether the control is visible on the form or not. If you set it to false, the control will not be display (visible). To change this property at run time you have to write,
Label1.Visible = False

(7) Enabled : Gets or sets a value indicating whether the user can use the control or not (whether the control can respond to user interaction or not). If it is set to false, the control cannot get user input, means it become gray.
To change this property at run time you have to write,
Label1.Enable = False

(8) **Left** : Gets or sets the distance, in pixels, between the left edge of the control and the left edge of its container's client area.

(9) **Right** : Gets the distance, in pixels, between the right edge of the control and the left edge of its container's client area. At run time you can not change the value of this property. It is read-only. You can change the value of this property indirectly by changing the value of the Left or Width properties or calling the SetBounds method.

(10) **Top** : Gets or sets the distance, in pixels, between the top edge of the control and the top edge of its container's client area.

(11) **Bottom** : Gets the distance, in pixels, between the bottom edge of the control and the top edge of its container's client area. Like Right property, Bottom property is also read-only property. You can change this property value by changing the value of the Top or Height properties or calling the SetBounds method.

(12) **BorderStyle** : Gets or sets the border style (border type) for the control. You can use this property to add a border to the control.

None : No border. (The default value is None) (0)
FixedSingle : A single-line border. (1)
Fixed3D : A three-dimensional border. (2)

To change this property at run time you have to write,

```
Label1.BorderStyle = BorderStyle.Fixed3D
```

(13) **TextAlign** : Gets or sets the alignment of text in the label control. (TopCenter, TopLeft, TopRight, MiddleCenter, MiddleLeft, MiddleRight, BottomCenter, BottomLeft, BottomRight).

System.Drawing.ContentAlignment Enumeration Specifies alignment of content on the drawing surface.

For Example :
```
Label1.TextAlign = ContentAlignment.BottomCenter
```
OR
```
Label1.TextAlign = 512
```

(14) **Size** : Gets or sets the height and width of the control in pixels. 1st argument is width & 2nd argument is height.

For Example :
```
Label1.Size = New System.Drawing.Size(200, 500)
```
OR
```
Label1.Size = New Size(200, 500)
```

(15) **Location** : Gets or sets the coordinates of the upper-left corner of the control with respect to the upper-left corner of its container. Point(x, y)

To change this property at run time you have to write,

```
Label1.Location = New Point(100, 100)
```

(16) Cursor : Gets or sets the cursor that is displayed when the user moves the mouse pointer over the control. To change this property at run time you have to write, Label1.Cursor = Cursors.Hand

(17) FlatStyle : Gets or sets the flat style appearance of the label control.

Flat	:	The control appears flat.
Popup	:	A control appears flat until the mouse pointer moves over it, at which point it appears three-dimensional.
Standard	:	The control appears three-dimensional. The default value is Standard.
System	:	The appearance of the control is determined by the user's operating system.

(18) UseMnemonic : If it is set to True, then the first character preceded by an ampersand (&) (defined in the text property of the label) will be use as a label's mnemonic key. By pressing ALT + the mnemonic character sets the focus to the control that follows the Label in the tab order. By default it is set to True.

(19) Image : Gets or sets the image that is displayed on a Label. The default is a null reference (Nothing in Visual Basic). You can display image at design time (via property window) & run time (via coding). At run time FromFile method of Image class is used to display image on a label control.
For Example : Label1.Image = Image.FromFile("C:\SAI.JPG")

(20) ImageAlign : Gets or sets the alignment of an image that is displayed in the control.
(TopCenter, TopLeft, TopRight, MiddleCenter, MiddleLeft, MiddleRight, BottomCenter, BottomLeft, BottomRight)
To change this property at run time you have to write,
Label1.ImageAlign = ContentAlignment.TopLeft

(21) Locked : Gets or sets a value indicating whether the Label Control can be moved or resized. If it is set to true, then Label Control can not be moved or resized. By default it is False.

(22) Font : Gets or sets the font of the text displayed by the control.
Ex.1 : To change font, font size & font style

```
Dim Fobj As New System.Drawing.Font("Arial", 21, FontStyle.Bold)
Label1.Font = Fobj
```

OR

```
Label1.Font = New System.Drawing.Font("Arial", 21, FontStyle.Bold)
```

OR

```
Label1.Font = New Font("Arial", 21, FontStyle.Bold)
```

Ex.2 : To change only font

```
Label1.Font = New Font( "Arial", Label1.Font.Size, Label1.Font.Style)
```

Ex.3 : To change only font size

```
Label1.Font = New Font( Label1.Font.Name, 25, Label1.Font.Style)
```

Ex.4 : To change only font style to bold

```
Label1.Font = New Font( Label1.Font.Name, Label1.Font.Size,
FontStyle.Bold)
```

➢ **Method of Label Control :**

- **SetBounds :** Sets the bounds of the control to the specified location and size.

  SetBounds(x, y, Width, Height)

  Parameters :

x	The new Left property value of the control.
y	The new Top property value of the control.
width	The new Width property value of the control.
height	The new Height property value of the control.

  For Ex. : Label1.SetBounds(400, 400, 100, 100)

- **BringToFront :** Brings the control to the front of the z-order.

  For Ex. Label1.BringToFront()

- **SendToBack :** Sends the control to the back of the z-order.

  For Ex. Label1.SendToBack()

➢ **Events of Label Control :**

(1) Click : Click event is fire (raised or event handler is executed or triggered) when control is click.

For Ex.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles
Button1.Click
    Me.Text = "|| HARI OM SHRI JAY GANESHAY NAMAHA: OMSHIVSAI ||"
End Sub
```

- Two arguments are common to all events.
- ByVal **sender** As System.Object and ByVal **e** As System.EventArgs.
  → The first argument (parameter), Sender which passes information about the object. (Provides a reference to the object that raised the event).

  Where sender used to find out, which control fires the event. (Name of the control that fires the event.

  For example : Console.WriteLine(Sender. ToString) → Button1

  Where sender argument also used to find out, the type of the control that fires the event.

  For example : Console.WriteLine(Sender. GetType) → Button

→ The second argument (parameter), e which passes information about the action that invokes the event. By referencing the object's properties (and, sometimes, its methods), you can obtain information such as the location of the mouse for mouse events, or you can trap which mouse button was pressed or you can trap which key was pressed or data being transferred in drag-and-drop events. The properties of e object are not same for all the events.

- **Handles clause :** Event declaration has a **Handles clause** that defines which events will be handled. **Handles allows multiple events to be handled by one event handler.** In this case, the following example, either Button1 or Button2 is click same click event will be executed.

Ex.
```
Private Sub Button1_Click(ByVal sender As Object, ByVal e As
System.EventArgs)

Handles Button1.Click, Button2.Click
    MsgBox("SAIBABA : Help Me !")
    End Sub
```

(2) DoubleClick : Occurs (Fires or Executes) when the control is double-clicked.

(3) MouseDown : Occurs when the mouse pointer is over the control and a mouse button is pressed.(Namespace: System.Windows.Forms)
Mouse events occur in the following order :
[1] MouseEnter
[2] MouseMove
[3] MouseHover / MouseDown / MouseWheel
[4] MouseUp
[5] MouseLeave

Ex. 1 :
```
Private Sub Form1_MouseDown(ByVal sender As Object, ByVal e As
System.Windows.Forms.MouseEventArgs)Handles Me.MouseDown
    If e.Button = Windows.Forms.MouseButtons.Left Then
        TextBox1.Text = "Mouse down at : " & e.X & " : " & e.Y
    End If
End Sub
```

Ex.2 :
```
Private Sub Form1_MouseMove(ByVal sender As Object, ByVal e As
System. Windows. Forms. MouseEventArgs) Handles Me.MouseMove
    Me.Text = "Mouse move at: " & e.X & " : " & e.Y
End Sub
```

- **Properties of the MouseEventArgs Object :**

  Button : Gets (Indicates) which mouse button was pressed.

  Clicks : Gets the number of times the mouse button was pressed and released.

  Delta : Gets a signed count of the number of detents the mouse wheel has rotated. A detent is one notch of the mouse wheel.

  Location : Gets the location of the mouse during the generating mouse event.

  X : Gets the x-coordinate of the mouse during the generating mouse event. (The x-coordinate of the mouse click).

  Y : Gets the y-coordinate of the mouse during the generating mouse event. (The y-coordinate of the mouse click).

- **MouseEventArgs.Button Property holds one of these members of MouseButtons enumeration :**

  Left : The left mouse button was pressed.

  Middle : The middle mouse button was pressed.

  None : No mouse button was pressed.

  Right : The right mouse button was pressed.

  XButton1 : The first XButton was pressed.

  XButton2 : The second XButton was pressed.

  With Windows 2000, Microsoft is introducing support for the Microsoft IntelliMouse Explorer, which is a mouse with five buttons. The two new mouse buttons (XBUTTON1 and XBUTTON2) provide backward/forward navigation.

(4) **MouseEnter :** Occurs (Fires or Executes) when the mouse pointer enters the control.

(5) **MouseHover :** Occurs (Fires or Executes) when the mouse pointer hovers over the control.

(6) **MouseLeave :** Occurs (Fires or Executes) when the mouse pointer leaves the control.

(7) **MouseMove :** Occurs (Fires or Executes) when the mouse pointer is moved over the control.

(8) **MouseUp :** Occurs (Fires or Executes) when the mouse pointer is over the control and a mouse button is released.

(9) **MouseWheel :** Occurs (Fires or Executes) when the mouse wheel moves while the control has focus.

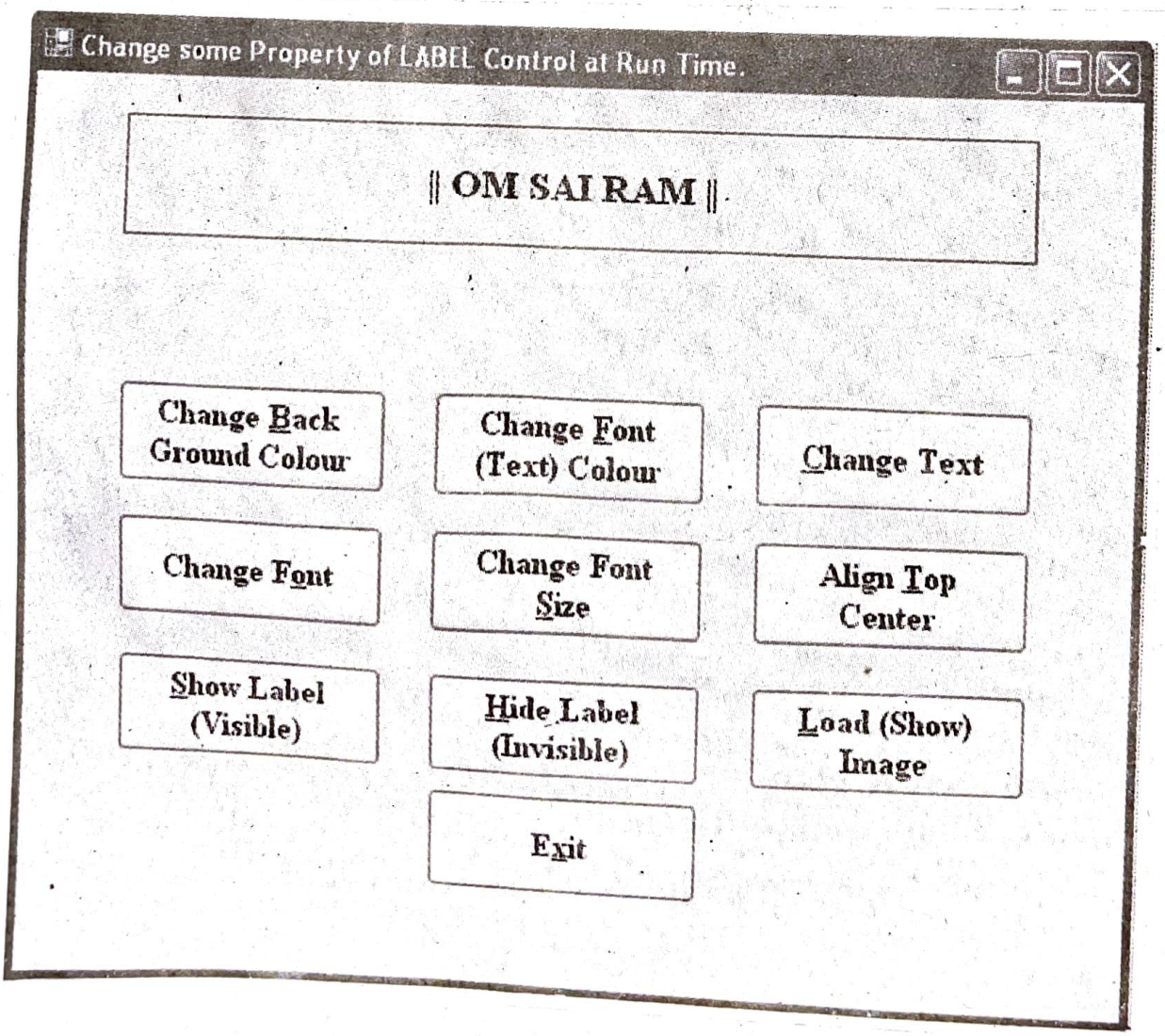
(10) TextChanged : Occurs (Fires or Executes) when the Text property value has changed.

(11) Paint : Occurs when the control is redrawn or updated.

(12) MouseClick :
Note : This event is new in the .NET Framework version 2.0.
Occurs (Fires or Executes) when the control is clicked by the mouse.

(13) MouseDoubleClick :
Note : This event is new in the .NET Framework version 2.0.
Occurs (Fires or Executes) when the control is double clicked by the mouse.

➤ **Practical 1 : Change some Property of LABEL Control at Run Time.**
Design the form as given below. Place one label control & ten buttons.



```
' Coding To Change Background Colour
    Label1.BackColor = Color.Wheat
' Coding To Change Font (Text) Colour
    Label1.ForeColor = Color.Red
' Coding To Change Text
    Label1.Text = "OM"
' Coding To Change Font
    Label1.Font = New Font("Arial", Label1.Font.Size, Label1.Font.Style)
' Coding To Change Font Size
    Label1.Font = New Font(Label1.Font.Name, 20, Label1.Font.Style)
' Coding To Align &Top Center
    Label1.TextAlign = ContentAlignment.TopCenter
' Coding To Show Label  (Visible)
    Label1.Visible = True
' Coding To Hide Label (Invisible)
    Label1.Visible = False
' Coding To Load (Show) Image
    Label1.Image = Image.FromFile("C:\Pictures\OM1.JPG")
' Coding of Exit the form
    Me.Close( )
```

[2] TextBox Control : TextBox control is used to get input from the user or display text to the user. An instance (object) of the "TextBox" class is created when a TextBox control is added to the form. Text boxes can be multiline, have scroll bars, be read only, and have many other attributes. The text displayed by the control is contained in the Text property. The Text property can be set at design time with the Properties Window, at run time in code, or by user input at run time. TextBox control sometime also called as edit control. It is used to accept user input. The main difference between textbox control & label control is that, user can modify the text inside the textbox, but user can not modify the text in a label control.

➢ **Property :**

(1)	**Name** – object name	**(2)**	**Text**	**(3)**	**BackColor**
(4)	**ForeColor**	**(5)**	**Visible**	**(6)**	**Enabled** – unactive
(7)	**Left**	**(8)**	**Right**	**(9)**	**Top**
(10)	**Bottom**	**(11)**	**BorderStyle**	**(12)**	**Size**
(13)	**Location**	**(14)**	**Cursor**		

(All these properties are discussed in Lable control).

(15) TextAlign : Gets or sets how text is aligned in a TextBox control. This **HorizontalAlignment** enumeration is used to align the text to the left, right, or in the center of a control element.

TextBox1.TextAlign = HorizontalAlignment.Center

(16) **MultiLine :** Gets or sets a value indicating whether this is a multiline TextBox control. True if the control is a multiline TextBox control; otherwise, false.
By default, you can enter up to 2048 characters (2KB) in a text box (MultiLine property is set to False). If you set the Multiline property to true, you can enter up to 32 KB of text.

(17) **MaxLength :** Gets or sets the maximum number of characters the user can type or paste into the text box control. The default is 32767.

(18) **HideSelection :** Gets or sets a value indicating whether the selected text in the text box control remains highlighted when the control loses focus. If you set this property to True, then the selected text does not appear highlighted when the text box control loses focus. (By default it is True). You can use this property to keep text highlighted in a text box control while another form or a dialog box has focus, such as a spelling checker dialog box.

(19) **Lines :** Gets or sets the lines of text in a text box control. Lines is a string array where each elements, holds a line of text. You can access individual lines of text through Lines property.

```
TextBox1.Lines(0)
TextBox1.Lines(1)
TextBox1.Lines(2)
TextBox1.Lines(3)
```

(20) **PasswordChar :** Gets or sets the character used to mask characters of a password in a single-line TextBox control.

(21) **CharacterCasing :** Gets or sets whether the TextBox control modifies the case of characters as they are typed. The default is CharacterCasing.Normal. Followings are the CharacterCasing enumeration values

Lower : Converts all characters to lowercase.
Normal : The case of characters is left unchanged.
Upper : Converts all characters to uppercase.
For Ex. :

```
' Change all text entered to be lowercase.
TextBox1.CharacterCasing = CharacterCasing.Lower
```

(22) **ReadOnly :** Gets or sets a value indicating whether text in the text box is read-only. When this property is set to true, the contents of the control cannot be changed by the user at runtime (it becomes read only), but user can still view the ToolTips, user can copy the content of the textbox and also navigate the content using the cursor.

(23) **Locked** : As above, Same as Label Control.

(24) **ScrollBars** : Gets or sets which scroll bars should appear in a multiline TextBox control. One of the ScrollBars enumeration values. None (By default), Horizontal, Vertical or Both For example, at run time you can set as follows: TextBox1.ScrollBars = ScrollBars.Vertical

(25) **SelectedText :(Only available at run time, Dynamic property):** Gets or sets a value indicating the currently selected text in the control. If no text is currently selected in the text box, this property returns a zero-length string.

(26) **SelectionLength : (Only available at run time, Dynamic property):**
Gets or sets the number of characters selected in the text box.

(27) **SelectionStart : (Only available at run time, Dynamic property):** Gets or sets the starting point of text selected in the text box. If no text is selected in the control, this property indicates the insertion point for new text.

(28) **TextLength :(Only available at run time, Dynamic property):** Gets the length of text in the control. It is used to get the number of characters contained in the text of the control.

For Ex.1 : MsgBox(TextBox1.TextLength)

Ex.2 :
```
Dim Strlen As Integer
Strlen = TextBox1.TextLength
```

(29) **WordWrap** : Indicates whether a multiline text box control automatically wraps words to the beginning of the next line when necessary.

If this property is set to true, horizontal scroll bars are not displayed regardless of the ScrollBars property setting. (Default is True)

(30) **TabIndex** : Gets or sets the tab order of the control within its container. It is the no that will reflect the order when user press tab key. A tab index can consist of any valid integer greater than or equal to zero. If more than one control on the same parent control has the same tab index, the z-order of the controls determines the order to cycle through the controls.

For a control to be included in the tab order, its TabStop property must be set to true.

(31) **TabStop :** Gets or sets a value indicating whether the user can give the focus to this control using the TAB key. If it is set to False, than control will not get focus through tab key. **The default is True.**

(32) **AcceptReturn :** Gets or sets a value indicating whether pressing ENTER in a multiline TextBox control creates a new line of text in the control or activates the default button (OK button) on the form.

If you set this property to True, ENTER key creates a new line of text in a multiline version of the control. If you set this property to False, ENTER key activates the default button (for ex. OK button) for the form & user must press CTRL+ENTER to create a new line in a multiline TextBox control. The default is False.

(33) **AcceptTab :** By default (False), so user can move the next control with the tab key. If you set to True, you can insert a tab character in the multiline textbox using TAB key & user must press CTRL+TAB key to move to the next control.

(34) **Font :** (Same as Label control)

➢ **Method of TextBox Control :**

(1) **SetBound** (2) **BringToFront,**

(3) **SendToBack (Same as Label Control)**

(4) **Focus : Focus** method is used to sets input focus to the control. The **Focus** method returns **true** if the control successfully received input focus.
Following example transfer the focus to TextBox1 control.

```
TextBox1.Focus( )
```

(5) **AppendText :** Appends text to the current text of a text box. You can use this method to add text to the existing text in the TextBox control instead of using the concatenation operator (+) to concatenate text to the Text property. It is much faster than its equivalent statement. It works faster when content of text is more.
For Ex. TextBox1.**AppendText**("OM")

(6) **Clear :** Clears all text from the text box control. You can use this method to clear the contents of the control instead of assigning the Text property an empty string.
For Ex. TextBox1.**Clear( )** **OR**
TextBox1.Text = ""

(7) **Copy :** This method Copies the selected text from the TextBox control to the Clipboard.

For Ex.
```
' Ensure that text is selected in the text box.
    If TextBox1.SelectionLength > 0 Then
        ' Copy the selected text to the Clipboard.
            TextBox1.Copy( )
    End If
```

(8) **Cut :** This method removes (Moves) the selected text from the TextBox control to the Clipboard.

For Ex.
```
' Ensure that text is currently selected in the text box.
If TextBox1.SelectedText <> "" Then
    ' Cut the selected text from the control and paste it into the
    ' Clipboard.
        TextBox1.Cut()
End If
```

(9) **Paste :** This method replaces the selected text in the TextBox with the text from the Clipboard.

```
TextBox1.Paste( )
```

(10) **SelectAll :** This method is used to Selects all text in the text box.

➢ **Event of TextBox Control :**

(1)	**Click**	**(2)**	**DoubleClick**	**(3)**	**MouseDown**
(4)	**MouseEnter**	**(5)**	**MouseHover**	**(6)**	**MouseLeave**
(7)	**MouseMove**	**(8)**	**MouseUp**	**(9)**	**MouseWheel**
(10)	**MouseClick**				

(11) MouseDoubleClick(Same as Label Control)

(12) **TextChanged :** This is the default event of the Textbox control. It is triggered (fires or executes) when the text in the TextBox is changed.

(13) **ReadOnlyChanged :** It is triggered (fires or executes) when the value of the ReadOnly property is changed.

(14) **MultilineChanged :** It is triggered (fires or executes) when the value of the Multiline property is changed.

(15) **KeyPress :** Occurs when a key is pressed while the control has focus. KeyPress event occurs (fires or executes or happens) every time a key is pressed.

Key events occur in the following order :

[1] KeyDown [2] KeyPress [3] KeyUp

With each KeyPress event, it returns **e** argument of type

(16) **KeyPressEventArgs :**

KeyPressEventArgs have two properties (KeyChar & Handled).

- The **KeyChar property** of the e arguments returns the key that was pressed. (Returns the character corresponding to the key pressed).
- Set **Handled property** to true to cancel the KeyPress event.

Use the **KeyChar** property to sample keystrokes at run time and to modify keystrokes under special run-time circumstances. For example,You can use KeyChar to disable non-numeric keypresses when the user enters a ZIP code, change all alphabetical keypresses to uppercase in a data entry field, or monitor the keyboard or other key input device for specific key combinations.

- You can get or set the following keys: (KeyChar) a-z, A-Z. , CTRL, Punctuation marks, Number keys, both across the top of the keyboard and on the numeric keypad, ENTER.
- You cannot get or set the following keys: (KeyChar) The TAB key, INSERT and DELETE, HOME, END, PAGE UP and PAGE DOWN,F1-F12. (All Function Keys), ALT, Arrow keys.

Ex. 1 :

```
'To allow only alphabets (A to Z or a to z or backspace(8) or
'spacebar(32) or enter(13)
Private Sub TextBox5_KeyPress(ByVal sender As Object, ByVal e
As System.Windows.Forms.KeyPressEventArgs) Handles
TextBox5.KeyPress
    If (Asc(e.KeyChar) >= 65 And Asc(e.KeyChar) <= 90) Or
    (Asc(e.KeyChar) >= 97 And Asc(e.KeyChar) <= 122) Or
    Asc(e.KeyChar) = 8 Or Asc(e.KeyChar) = 32 Or Asc(e.KeyChar)
    = 13 Then
        e.Handled = False
    Else
        e.Handled = True
    End If
End Sub
```

Ex. 2 :

```
' To allow only NUMBERS ( 0 to 9) or Backspace(8)
Private Sub TextBox6_KeyPress(ByVal sender As Object, ByVal e
As System.Windows.Forms.KeyPressEventArgs) Handles
TextBox6.KeyPress
    If (Asc(e.KeyChar) >= 48 And Asc(e.KeyChar) <= 57) Or
    Asc(e.KeyChar) = 8 Then
        e.Handled = False
    Else
        e.Handled = True
    End If
End Sub
```

(17) **KeyDown & KeyUp :**

KeyDown event occurs when a key is pressed while the control has focus. KeyUp event occurs when a key is released while the control has focus. (The KeyDown event occurs (fires or executes or happens) when the user presses any key. The KeyUp event occurs (fires or executes or happens) when the user releases the key).

With each KeyDown & KeyUp event, it returns **e** argument of type **KeyEventArgs.**

KeyEventArgs have following property (KeyCode, Handled, Modifiers, Alt, control, shift).

Alt	:	Gets a value indicating whether the ALT key was pressed. (Returns True or False)
Control	:	Gets a value indicating whether the CTRL key was pressed.
Shift	:	Gets a value indicating whether the SHIFT key was pressed.
Handled	:	Gets or sets a value indicating whether the event was handled. (Set **Handled property** to true to cancel the KeyPress event).
KeyCode	:	Gets the keyboard code for a KeyDown or KeyUp event.
Modifiers	:	Gets the modifier flags for a KeyDown or KeyUp event. The flags indicate which combination of CTRL, SHIFT, and ALT keys was pressed.

The KeyChar argument does not report function keys. The event that can capture function keys are KeyDown event, which is generated when the key is pressed & the KeyUp event, which is generated when the key is released.

KeyDown event & KeyUp event don't return the KeyChar but returns the KeyCode (A special number that distinguishes each key on the keyboard, also known as scancode), through the e.KeyCode property. The KeyCode is unique for each key, not for each character. Lowercase & Uppercase characters have different ASCII values but the same KeyCode because they are on the same key. The number 4 and $ symbol have the same KeyCode because the same key on the keyboard generates both character.

The KeyCode for the function key F1 is 112 (or the constant Keys.F12) In following example, whenever user press enter key or down arrow key or page down key in textbox1, the focus will be shifted to next control. And whenever user press up arrow key or page up key in textbox1, the focus will be shifted to previous control (as per Tab order).

```
Private Sub TextBox1_KeyDown(ByVal sender As Object, ByVal e As
System.Windows.Forms.KeyEventArgs) Handles TextBox1. KeyDown
    If e.KeyCode = Keys.Enter Or e.KeyCode = Keys.Down Or e.KeyCode
    = Keys.PageDown Then
        SendKeys.Send("{Tab}")
    ElseIf e.KeyCode = Keys.Up Or e.KeyCode = Keys.PageUp Then
        SendKeys.Send("+{Tab}")
    End If
End Sub
```

(18) **Enter :** Fired when control received the focus.

(19) **Leave :** Fired when control loses the focus.

(20) **GotFocus :** Occurs when the control receives focus.

```
Private Sub TextBox1_GotFocus(ByVal sender As Object, ByVal e As
System.EventArgs) Handles TextBox1.GotFocus
    TextBox1.BackColor = Color.Lavender
    TextBox1.Font = New Font("Times New Roman", 8, FontStyle.Bold)
    SendKeys.Send("{Home}+{End}")
End Sub
```

(21) **LostFocus :** Occurs when the control loses focus.

```
Private Sub TextBox1_LostFocus(ByVal sender As Object, ByVal e As
System.EventArgs) Handles TextBox1.LostFocus
    TextBox1.BackColor = Color.White
    TextBox1.Font = New Font("Times New Roman", 8, FontStyle.Regular)
End Sub
```

When you change the focus by using the keyboard (TAB, SHIFT+TAB, and so on) focus events occur in the following order :

[1] Enter [2] GotFocus [3] Leave
[4] Validating [5] Validated [6] LostFocus

When you change the focus by using the mouse or by calling the Focus method, focus events occur in the following order

[1] Enter [2] GotFocus [3] LostFocus
[4] Leave [5] Validating [6] Validated

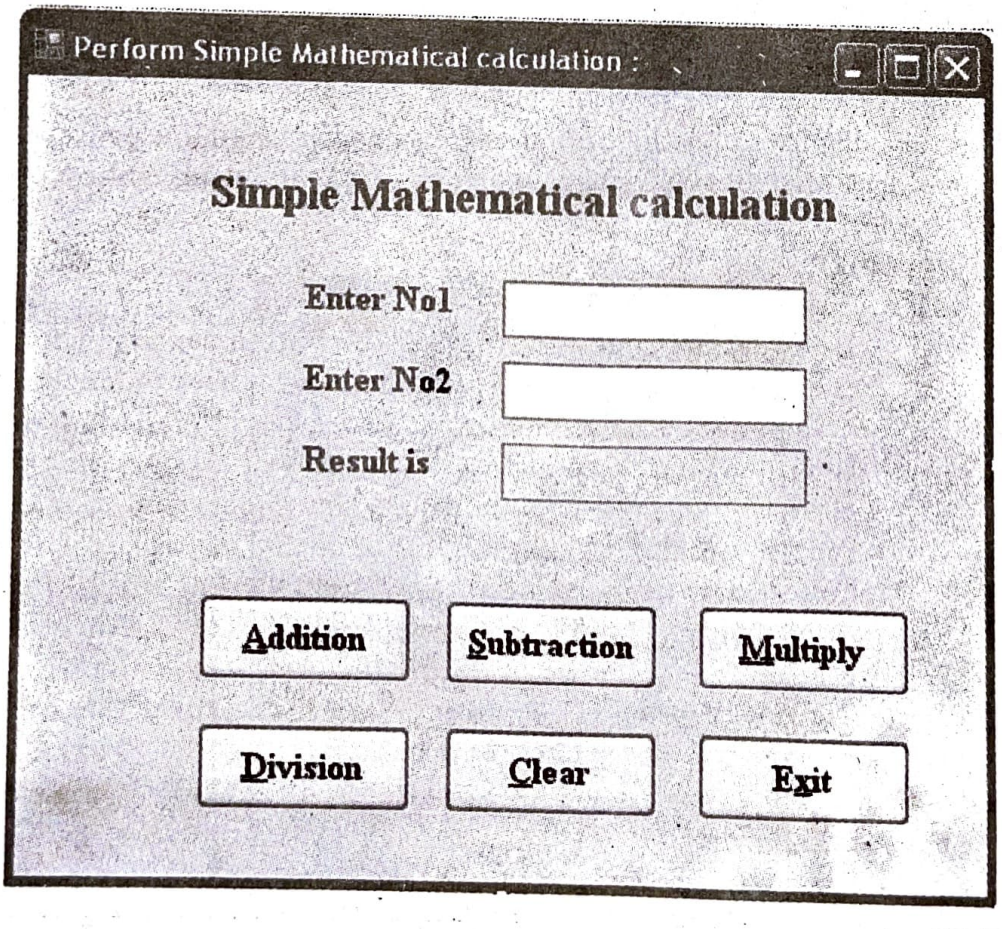
(22) **Validating :** Occurs when the control is validating. Validating Events .NET gives you two validate events instead of one: Validating and Validated. Validating is triggered before the control loses the focus. **For example**, to check TextBox1 should not be left blank. Where the code **e.Cancel = True** prevents the shift of focus away from the textbox if there is nothing in the box.

```
Private Sub TextBox1_Validating(ByVal sender As Object, ByVal e As
System.ComponentModel.CancelEventArgs) Handles
    TextBox1.Validating
    If TextBox1.Text.Trim = String.Empty Then
        e.Cancel = True
    End If
End Sub
```

The second example, In Validating event of the TextBox control, you can validate an e-mail address that the user enters. If the e-mail address is not in the standard format (containing "@" and "."), the validation fails, an ErrorProvider icon is displayed, and the event is canceled.

(23) **Validated :** Occurs when the control is finished validating.
The Validated event, on the other hand, fires after the focus has shifted away from the control, but before other controls get the focus. Thus, you can use Validated to update the state of the other controls on the form.
NOTE : If you set the CausesValidation property of a control to False, the Validating and Validated events will not be triggered.

Practical 2 : Perform Simple Mathematical calculation.
Design the form as given below. There are three TextBox. TextBox1 to enter No1, TextBox2 to enter No2 & TextBox3 to display Result. Set **ReadOnly** property of TextBox3 to True, so user can not change the content of TextBox3.



```
' Coding of Addition Button
        TextBox3.Text = Val(TextBox1.Text) + Val(TextBox2.Text)
' Coding of Subtraction Button
        TextBox3.Text = Val(TextBox1.Text) - Val(TextBox2.Text)
' Coding of Multiply Button
        TextBox3.Text = Val(TextBox1.Text) * Val(TextBox2.Text)
' Coding of Divisiopn Button
        TextBox3.Text = Val(TextBox1.Text) / Val(TextBox2.Text)
' Coding of Clear Button : Here we have cleared the content of all
' the three TextBox using Clear method & set focus to the TextBox1 ' using
Focus method.
    TextBox1.Clear()
    TextBox2.Clear()
    TextBox3.Clear()
    TextBox1.Focus()
' Coding of Exit the form
    Me.Close()
```

[3] Button Control :

Button control is used to perform some action. Action may be Save, Delete, Print, Add New, Modify etc. Whenever the user clicks a button, the Click event handler is invoked. You place code in the Click event handler to perform any action you desire.

The text displayed on the button is contained in the Text property. The Button control can also display images using the Image and ImageList properties.

➢ **Property :**

(1) Name & Text :

It is the text that is display on the button control. The Text property can contain an access key, which allows a user to "click" the control by pressing the ALT key with the access key. If you place ampersand (&) before any letter, so underscore letter will appear, then user can press alter key with that letter to fire the Click event. For example when you set **E&xit** in Text property of button control, it will be display as Exit . So by pressing ALT key & x click event will be fired.

(2)	**BackColor**	**(3)**	**ForeColor**	**(4)**	**AuotSize**
(5)	**Visible**	**(6)**	**Enabled**	**(7)**	**Left**
(8)	**Right**	**(9)**	**Top**	**(10)**	**Bottom**
(11)	**Image**	**(12)**	**ImageAlign**	**(13)**	**TextAlign**
(14)	**Size**	**(15)**	**Location**	**(16)**	**Cursor**
(17)	**TabIndex**	**(18)**	**TabStop**		
(19)	**Font : (Same as label control)**				

(20) **DialogResult :** Gets or sets a value that is return to the parent form when the button is click.
You can set the way a dialog box is closed at either design time or run time. At design time, you can set the DialogResult property for all of the Button controls on a dialog box. At run time, you can set the DialogResult property so you can dynamically handle user responses. Often used when you are creating a dialog boxes.

➢ **Method :**

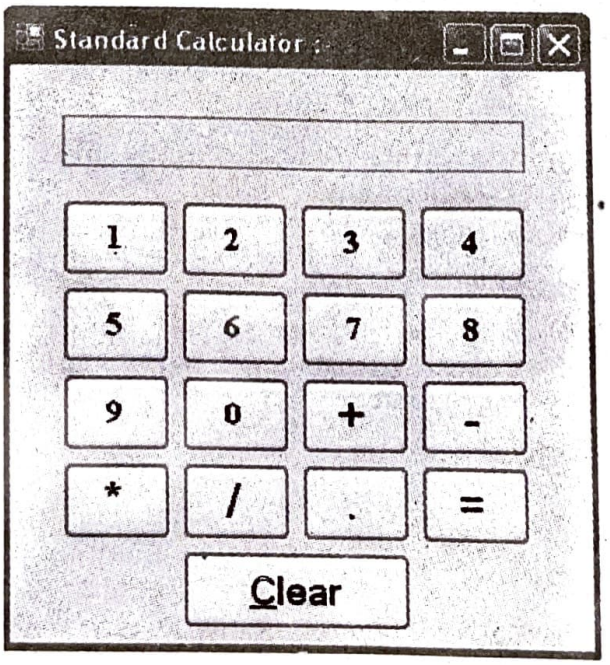
(1) **SetBounds** (2) **BringToFront** (3) **SendToBack**
(4) **Focus**
(5) **PerformClick :** Generates a Click event for a button. This method raises the click event of the button control. This method comes handy when the code in the click event has to be executed **even when the user has not clicked the button.**

➢ **Event :**

(1)	**Click**	(2)	**DoubleClick**	(3)	**MouseDown**
(4)	**MouseEnter**	(5)	**MouseHover**	(6)	**MouseLeave**
(7)	**MouseMove**	(8)	**MouseUp**	(9)	**MouseWheel**
(10)	**MouseClick**	(11)	**MouseDoubleClick**		
(12)	**TextChanged**	(13)	**KeyPress**	(14)	**KeyDown**
(15)	**KeyUp**	(16)	**Enter**	(17)	**Leave**
(18)	**GotFocus**	(19)	**LostFocus**	(20)	**Validating**
(21)	**Validated**				

Practical 3 : Design Standard Calculator.
Design the interface as given below.

(1)



Set **ReadOnly** property of TextBox1 to True.
Set **TextAlign** property of TextBox1 to Right.

Logic :

- Declare three variables in declaration section. No1 to store first number. No2 to store second number. And OP to store status of the operator.

- Whenever plus "+" button is click, that means first number have been enter, now we have to store first number in No1 variable. & clear the TextBox1 for second no. Assign "+" operator to OP variable (for the status).

- When equal to "=" button is click, that means second number have been enter, now we have to store second number in No2 variable. Now to perform the operation, we have to check status of OP variable, whether it has "+" or "-" or "*" or "/". According to the OP status perform that operation.

- For dot (decimal point) we have used InStr function, it Returns an integer specifying the start position of the first occurrence of second string within the first string. It second string (dot /decimal point) in snot present in the first string (number) then, it will return 0. It returns 0 that means, no dot (decimal point) present in the no., so append dot to no.

```
' Coding of variable declaration (Declare it in declaration section)
    Dim NO1, NO2 As Double
    Dim OP As String
' Coding of 1 number button
    TextBox1.AppendText(1)
' Coding of 2 number button
    TextBox1.AppendText(2)
' Coding of 3 number button
    TextBox1.AppendText(3)
' Repeat the same for all other number buttons. (4, 5, 6, 7, 8, 9, & 0).
' Coding of "+" operator Button
    NO1 = Val(TextBox1.Text)
    TextBox1.Clear()
    OP = "+"
' Coding of "/" operator Button
    NO1 = Val(TextBox1.Text)
    TextBox1.Clear()
    OP = "/"
' Repeat the same procedure for "-", & "*" Operators Button
' Coding of "=" Button
    NO2 = Val(TextBox1.Text)
        If OP = "+" Then
```

```
        TextBox1.Text = NO1 + NO2
    ElseIf OP = "-" Then
        TextBox1.Text = NO1 - NO2
    ElseIf OP = "*" Then
        TextBox1.Text = NO1 * NO2
    ElseIf OP = "/" Then
        TextBox1.Text = NO1 / NO2
    End If
' Coding of "."(dot /decimal point) Button
    If InStr(TextBox1.Text, ".") = 0 Then
        TextBox1.AppendText(".")
    End If
' Coding of Clear Button
    TextBox1.Clear()
```

[4] Panel Control: (Container Control)

Panel control is used to group other controls. Typically, you use panels to subdivide a form into groups depending upon the function. Panel Control is also called as Container control. When Panel control is moved, all the controls within it will also be moved.

➢ **Property :**

(1) BorderStyle : Gets or sets the border style (border type) for the control. You can use this property to add a border to the control.

Fixed3D	:	A three-dimensional border.	(2)
FixedSingle	:	A single-line border.	(1)
None	:	No border. (The default value is None)	(0)

(2) BackColor **(3) BackgroundImage**

(4) AutoScroll : Panel control supports scroll bars. To enable scroll bars in a panel control, set its AutoScroll property to True. If set to true then the scrollbar will automatically appears if the control contents are larger than its visible area.

(5) AutoScrollMargin : Gets or sets the size of the auto-scroll margin. A Size that represents the height and width of the auto-scroll margin in pixels.

➢ **Example to add TextBox Controls to the Panel at runtime :**

You can add controls to the panel at run time just as you can add controls to a form. **Form's Controls** collection is used to add controls to a form. **Panel's Controls** collection is used to add controls to a panel. In following example, we have created a Panel1 object from the Panel class, & Textbox1 object from the TextBox class. Then set the desire location & the size of the objects.

```
Public Class Form1
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
    System.EventArgs) Handles MyBase.Load
        Dim Panel1 As New Panel()
        Dim TextBox1 As New TextBox()
        Panel1.Location = New Point(100, 100)
        Panel1.Size = New Size(200, 200)
        Panel1.BorderStyle = BorderStyle.Fixed3D
        TextBox1.Location = New Point(11, 11)
        TextBox1.Size = New Size(170, 100)
        'Adds a Panel1 control to the Form's controls collection.
        Me.Controls.Add(Panel1)
        'Adds a TextBox1 control to the Panel's controls collection.
        Panel1.Controls.Add(TextBox1)
    End Sub
End Class
```

[5] CheckBox Control :

You can use check box controls in groups to display multiple choices from which the user can select one or more. (It is used to select one or more oprtions from the multiple options).

Only one radio button in a group can be selected at a time. While with the check box control, any number of check boxes may be selected.

➢ **Property :**

(1) **Appearance :** If Appearance value is set to **Normal**, the CheckBox appears as a typical CheckBox. If the value is set to **Button**, the CheckBox appears like a toggle button, which can be toggled to an up or down state.

(2) **FlatStyle :** Gets or sets the flat style appearance of the button control. If the FlatStyle property of the RadioButton and CheckBox is set to FlatStyle.System, the user's operating system sets the appearance of the control.

Flat : The control appears flat.

Popup : A control appears flat until the mouse pointer moves over it, at which point it appears three- dimensional.

Standard: The control appears three-dimensional. (By Default)

System : The appearance of the control is determined by the user's operating system.

(3) **ThreeState :** Gets or sets a value indicating whether the CheckBox will allow three check states rather than two. By default is False. If this property is set to True, then the CheckBox has three check states namely Checked, Unchecked & Intermediate. In Intermediate state CheckBox is checked & shaded.

(4) **CheckState :** Gets or sets the state of the CheckBox.

CheckState	Description
Checked	The CheckBox displays a check mark.
Unchecked	The CheckBox is empty.
Indeterminate	The CheckBox displays a check mark and is shaded. Intermediate state is sort of middle state between checked & unchecked.

(5) **Checked :** This property is used to check whether the Checkbox is checked or not.. If this property is true, then the CheckBox is in the checked state. If it is false, then the CheckBox is in unchecked state.

Note : If the ThreeState property is set to true, the Checked property will return true for either a Checked or Indeterminate CheckState.

(6) **Text :** It is the text that is displayed within the control. The Text property can contain an access key, which allows a user to "click" the control by pressing the ALT key with the access key.

➢ **Event :**

(1) **AppearanceChanged :** It occurs or executes when the value of the **Appearance** property changes.

(2) **CheckedChanged :** It occurs when the value of the **Checked** property changes. It is default event of CheckBox control.

(3) **CheckStateChanged :** It occurs when the value of the **CheckState** property changes.

[6] RadioButton Control :

It is used to select one and only one option from the available options in one group.

Only one radio button in a group can be selected at a time. While with the check box control, any number of check boxes may be selected.

➢ **Property :**

(1) **Appearance :** If the Appearance value is set to Normal, then the RadioButton control appears as a circular check box. If the value is set to Button, then the RadioButton appears like a control that can be toggled to an up or down state. Either type can display text, an image, or both.

(2) **FlatStyle & Checked :** This property is used to check whether the RadioButton is selected or not. If this property is true, then RadioButton is selected.

(3) **Text :** It is the text that is displayed within the control. The Text property can contain an access key, which allows a user to "click" the control by pressing the ALT key with the access key.

➢ **Method :**

(1) Performclick : Generates a Click event for the control, simulating a click by a user. This method comes handy when the code in the click event has to be executed **even when the user has not clicked the RadioButton.**

➢ **Event :**

(1) AppearanceChanged : It occurs when the value of the **Appearance** property changes.

(2) CheckedChanged : It occurs when the value of the **Checked** property changes. It is default event of RadioButton control.

[7] GroupBox Control : (Container Control)

GroupBox control is used to group other controls. Typically, you use GroupBox to subdivide a form into groups depending upon the function. GroupBox Control is also called as Container control. When GroupBox control is moved, all the controls within it will also be moved.

➢ **Panel Control versus GroupBox Control :**

The Panel control is similar to the GroupBox control; however, they have following difference.

Panel Control	GroupBox Control
Panel control can have scroll bars.	GroupBox control dose not have scroll bar.
Panel control dose not display a caption (Label) within it. (It does not have Text property)	GroupBox control displays a caption (Label) within it. It is gets or sets with Text property
Panel control does not have **FlatStyle,** but it has **BorderStyle** property.	GroupBox control does not have **BorderStyle,** but it has **FlatStyle** property.

➢ **Property :**

(1) FlatStyle :

Gets or sets the flat style appearance of the group box control.

Flat : The control appears flat.

Popup : A control appears flat until the mouse pointer moves over it, at which point it appears three-dimensional.

Standard : The control appears three-dimensional.

System : The appearance of the control is determined by the user's operating system.

(2) BackColor **(3) BackgroundImage**

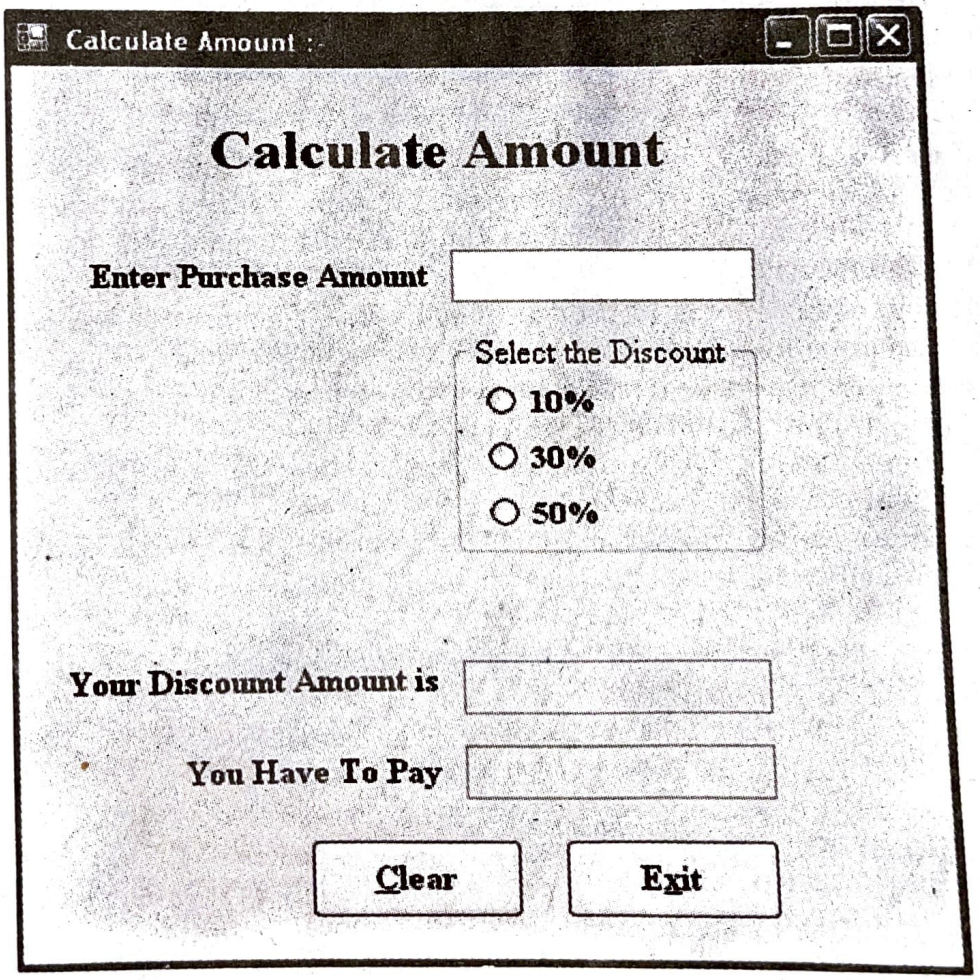
(4) Text : It is used to gets or sets the text (caption) that is displayed within the control.

PRACTICAL 4 : Accept amount from the users. Select Discount from the option button and display the Discount amount and the Pay-amount.
Design the form as given in figure. There are three TextBox. TextBox1 to enter Purchase amount. Set its Name property to TxtPurAmt

Set **ReadOnly** property of TextBox2 & TextBox3 to True, so user can not change the content of TextBox2 & TextBox3.

Set Name property of TextBox2 to TxtDiscountAmt. & Set Name property of TextBox3 to TxtPayAmt.

Place one GroupBox. & place three RadioButton (for discount) in it. Whenever any RadioButton (Discount) is click, Its Discount amount and the Pay-amount should be display in TextBox2 & TextBox3 respectively.



```
' Coding of RadioButton1 (Write in CheckedChanged Event)
    TxtDiscountAmt.Text = Val(TxtPurAmt.Text) * 10 / 100
    TxtPayAmt.Text = Val(TxtPurAmt.Text) - Val(TxtDiscountAmt.Text)
' Coding of RadioButton2 (Write in CheckedChanged Event)
    TxtDiscountAmt.Text = Val(TxtPurAmt.Text) * 30 / 100
    TxtPayAmt.Text = Val(TxtPurAmt.Text) - Val(TxtDiscountAmt.Text)
' Coding of RadioButton3 (Write in CheckedChanged Event)
    TxtDiscountAmt.Text = Val(TxtPurAmt.Text) * 50 / 100
    TxtPayAmt.Text = Val(TxtPurAmt.Text) - Val(TxtDiscountAmt.Text)
' Coding of Clear Button
    RadioButton1.Checked = False
    RadioButton2.Checked = False
    RadioButton3.Checked = False
    TxtPurAmt.Clear()
    TxtDiscountAmt.Clear()
    TxtPayAmt.Clear()
    TxtPurAmt.Focus()
' Coding of Exit the form
    Me.Close()
```

[8] ListBox Control : ListBox control displays a list of items from which the user can select one or more items. If the total number of items exceeds the number that can be displayed, a scroll bar is automatically added to the **ListBox** control.

Advantage of ListBox is (i) user did not have to remember all possible options. (ii) It also prevent wrong entry because, user did not have to type.

→ To add or remove items at design time you can use **Items** property, which is a very handy array of items in the ListBox.

→ To add items at run time, you can use Items.Add **OR** Items.Insert **OR** Items.AddRange method.

At run time you can use add method, to insert an item into the ListBox.

For Ex. ListBox1.Items.Add("Boll Pen")

OR

Or, you can use Insert method, to inserts an item into the list box at the specified index.

ListBox1.Items.Insert(Index As Integer, Item As Object)

For Ex. ListBox1.Items.Insert(3, "Eraser")

OR

Or, you can also use AddRange method, to add a set of items to the list box in a single operation,

For Ex.
```
Dim DataArray(20) As String
.......
.........
ListBox1.Items.AddRange(DataArraty)
```

→ To remove items at run time, you can use Items.Remove **OR** Items.RemoveAt method.

At run time you can use remove method, to remove a specific item from the ListBox.

For example, to remove a currently selected item

```
ListBox1.Items.Remove(ListBox1.SelectedItem)
```

You can also remove an item by passing corresponding object to remove.

```
ListBox1.Items.Remove("Boll Pen")
```

OR

Or, you can use RemoveAt method to delete items from a ListBox. For example, to remove the item at index 5.

```
ListBox1.Items.RemoveAt(5)
```

Or, you can also remove the currently selected item with RemoveAt

```
ListBox1.Items.RemoveAt(ListBox1.SelectedIndex)
```

→ To remove all the item at run time, you can use Clear methods of the Items collection to clear a ListBox.

For Ex. `ListBox1.Items.Clear()`

➢ **Property :**

(1) **MultiColumn :** Gets or sets a value indicating whether the ListBox supports multiple columns.

When the MultiColumn property is set to true, the list box displays items in multiple columns and a horizontal scroll bar appears if total number of items exceeds. When the **MultiColumn** property is set to **false**, the list box displays items in a single column and a vertical scroll bar appears if total number of items exceeds.

(2) **ScrollAlwaysVisible :** When **ScrollAlwaysVisible** is set to **true**, the scroll bar appears regardless of the number of items. (A scroll bar always appears).

(3) **Sorted :** Gets or sets a value indicating whether the items in the ListBox are sorted alphabetically. If **Sorted** property is set to True, then the items are automatically sort strings alphabetically in a **ListBox**.

(4) **Text :** Gets or searches for the text of the currently selected item in the ListBox. This property represents the text of the item that is currently selected.

(5) **SelectionMode :** Gets or sets the method in which items are selected in the ListBox. How many items can be selected at a time and how the user can make multiple-selections. You can set this property to None, One, Multisimple, or MultiExtended.

MultiExtended	Multiple items can be selected, and the user can use the SHIFT, CTRL, and arrow keys to make selections. When the SelectionMode property is set to SelectionMode.MultiExtended, pressing SHIFT and clicking the mouse or pressing SHIFT and one of the arrow keys (UP ARROW, DOWN ARROW, LEFT ARROW, and RIGHT ARROW) extends the selection from the previously selected item to the current item. Pressing CTRL and clicking the mouse selects or deselects an item in the list.
MultiSimple	Multiple items can be selected. When the property is set to SelectionMode.MultiSimple, a mouse click or pressing the SPACEBAR selects or deselects an item in the list.
None	No items can be selected.
One	Only one item can be selected. (By Default)

(6) **Items :** This property is used to view and change the items in the ListBox. You can display the "String collection editor" by the ellipsis (...) button next to the Items property of the control.
The items in the list boxes are stored in the **Items** collection.
You can add items to a ListBox either at design time or at run time. The first item will have index 0, the next item will have index 1, and so on in the ListBox.
At design time you can use **Items** property, which is a very handy array of items in the ListBox.

(7) **SelectedItem :** Gets or sets the currently selected item in the ListBox. This property represents the item that is currently selected.
For a standard ListBox, you can use this property to determine which item is selected in the ListBox. If the SelectionMode property of the ListBox is set to either SelectionMode.MultiSimple or SelectionMode.Multi Extended (which indicates a multiple-selection ListBox) and multiple items are selected in the list, this property can return any selected item.

(8) **SelectedItems :** This property represents a collection of all the items that are currently selected in the ListBox.

(9) **SelectedIndex :** Gets or sets the index of the currently selected item in the ListBox. This property is used to find out the index number of the currently selected item.
If no item is selected in the listbox, then SelectedIndex property will return -1.

(10) **SelectedIndices** : This property represents a collection of all the indexes that are currently selected in the ListBox.

➢ **Methods :**

(1) **ClaerSelected** : Unselects all items in the ListBox. Calling this method is equivalent to setting the SelectedIndex property to minus one (-1). You can use this method to quickly unselect all items in the list.

(2) **GetSelected** : Returns a value indicating whether the specified item is selected. This method returns a True if the specified item is selected & returns a False if the specified item is not selected.

```
ListBox1.GetSelected(Index As Integer) As Boolean
```

(3) **SetSelected** : This method is used to select or unselect the specified item. A boolean True value can be passed to this method to select the item.

```
ListBox1.SetSelected(Index As Integer, Value As Boolean)
```

For Ex.

```
' Select two items from the ListBox.
ListBox1.SetSelected(1, True)
ListBox1.SetSelected(5, True)
```

(4) **BeginUpdate & EndUpdate** : The preferred way to add multiple items to the ListBox is to use the AddRange method. This enables you to add an array of items to the list in a single operation. However, if you want to add items one at a time using the Add method, **you can use the BeginUpdate method to prevent the control from repainting the ListBox each time an item is added to the list.** Once you have completed the task of adding items to the list, call the EndUpdate method to enable the ListBox to repaint. **This way of adding items can prevent flickered drawing of the ListBox when a large number of items are being added to the list.**

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    ListBox1.BeginUpdate( ) ' Prevent the control from repainting the ListBox
    Dim I As Integer
    For I = 1 To 20
        ListBox1.Items.Add("Item " & I.ToString())
    Next I
    ListBox1.EndUpdate( )    ' resume the painting the listbox control.
    ' (Eventhough 1 to 20 items are added to the ListBox1, you can 'view only
    after  EndUpdate method is called)
End Sub
```

(5) **FindString** : Finds the first item in the ListBox that starts with the specified string. If not found then it will return -1.

Syntax. 1 : ListBox.FindString (S as String)

Finds the first item in the ListBox that starts with the specified string.

Syntax. 2 : ListBox.FindString (S as String, StartIndex as integer)

Finds the first item in the ListBox that starts with the specified string. The search starts at a specific starting index.

(6) **FindStringExact** : Finds the first item in the ListBox that exactly matches the specified string. If not found then it will return -1.

ListBox. FindStringExact (S as String) OR

ListBox. FindStringExact (S as String, StartIndex as integer)

➢ **Events :**

(1) **SelectedIndexChanged** : Occurs when the value of the SelectedIndex property has changed.

[9] **CheckedListBox Control** : CheckedListBox control is derived from standard LisxtBox, except that a check box is displayed to the left of each item. It does almost everything that a list box does and also can display a check mark to the left of each item. Differences between the two controls are that in **SelectionMode property** of CheckedListBox control **MultiSimple & MultiExtended** are not supported. (It supports only none & one).

As with standard listbox, you can access the items in a Check List box using the Items property. To check an item, the user has to double click a checkbox by default. If you set CheckOnClick property to True, then only one click is required to check an item.

➢ **Property :**

(1) **CheckOnClick** : If it is set to True, then an item can be checked or unchecked with single click. If it is set to False (By Default), then an item can be checked or unchecked with double click.

(2) **CheckedItems** : This property refers to the collection of all the checked items in the CheckedListBox control.

(3) **CheckedIndices** : This property refers to the collection of indexes of all the checked items in the CheckedListBox control.

➢ **Methods :**

(1) **GetItemChecked** : Returns a value indicating whether the specified item is checked. This method returns a True if the specified item is checked & returns a False if the specified item is not checked.

CheckedListBox1.GetItemChecked(Index As Integer) As Boolean

(2) **SetItemChecked** : This method is used to check or uncheck the specified item. A boolean True value can be passed to this method to check the item.

CheckedListBox1.SetItemChecked (Index As Integer, Value As Boolean)

For Ex.
```
' To check the three items from the ListBox.
CheckedListBox1.SetItemChecked(1, True)
CheckedListBox1.SetItemChecked(5, True)
```

Note : To use three-state checkboxes, you have to use the **GetItemCheckState** & **SetItemChecked** methods instead of GetItemChecked & SetItemChecked methods.

(3) ClaerSelected **(4) GetSelected**
(5) SetSelected **(6) FindString**
(7) FindStringExact

➢ **Events :**

(1) SelectedIndexChanged : (Default event) : Occurs when the value of the SelectedIndex property has changed.

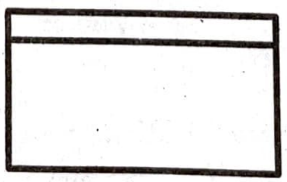
(2) ItemCheck : This event is triggered when the checkedstate of the item is changed.

[10] ComboBox Control : ComboBox control is used to display data in a drop-down combo box. The ComboBox control is made up of two parts. The top part is a text box that allows the user to type a list item. The second part is a list box that displays a list of items from which the user can select one. You can add or remove items to a ComboBox either at design time or at run time. To add or remove items at design time you can use **Items** property, which is a very handy array of items in the ComboBox. To add items at run time, you can use Items.Add **OR** Items.Insert **OR** Items.AddRange method. To remove items at run time, you can use Items.Remove **OR** Items.RemoveAt method. To remove all the item at run time, you can use Items.Clear method.

➢ **Property :**

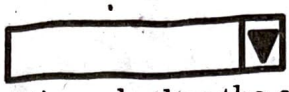
(1) DropDownStyle : Gets or sets a value specifying the style of the combo box. This property represents the style of the combo box control. The different styles are(a) Simple (b) DropDwon & (c) DropDownList.

[a] Simple : It is the ComboBox without drop down arrow. It contain text box & list box that does not drop down. User can select from the list or type in to the text box.



[b] DropDwon : It is the by default style. User can either type the text directly or select the item from the drop down list.

[c] **DropDownList :** Display list box when user click on the drop down arrow. User can only select an item from the list; user can not enter text in the text box.



(2) Focused : Gets a value indicating whether the control has input focus. It returns True if the control has focus.

(3) MaxDropDownItems : This property represents the maximum numbers of items that will be display in the drop down portion when the drop down arrow on the ComboBox control is click.

(4) Text : Gets or searches for the text of the currently selected item in the ComboBox. This property represents the text of the item that is currently selected.

(5) Items : This property is used to view and change the items in the ComboBox. You can display the "String collection editor" by the ellipsis (...) button next to the Items property of the control.

The items in the list boxes are stored in the **Items** collection.

You can add items to a ComboBox either at design time or at run time. The first item will have index 0, the next item will have index 1, and so on in the ComboBox.

At design time you can use **Items** property, which is a very handy array of items in the ComboBox.

(6) SelectedItem : Gets or sets the currently selected item in the ComboBox. This property represents the item that is currently selected.

(7) SelectedIndex : Gets or sets the index of the currently selected item in the ComboBox. This property is used to find out the index number of the currently selected item.

If no item is selected in the ComboBox, then SelectedIndex property will return -1.

Note: There is no **SelectedItems & SelectedIndices** property exists for ComboBox control.

➢ **Methods :**

(1) BeginUpdate **(2) EndUpdate**
(3) FindString **(5) FindStringExact**

➢ **Events :**

(1) SelectedIndexChanged : Occurs when the value of the SelectedIndex property has changed.

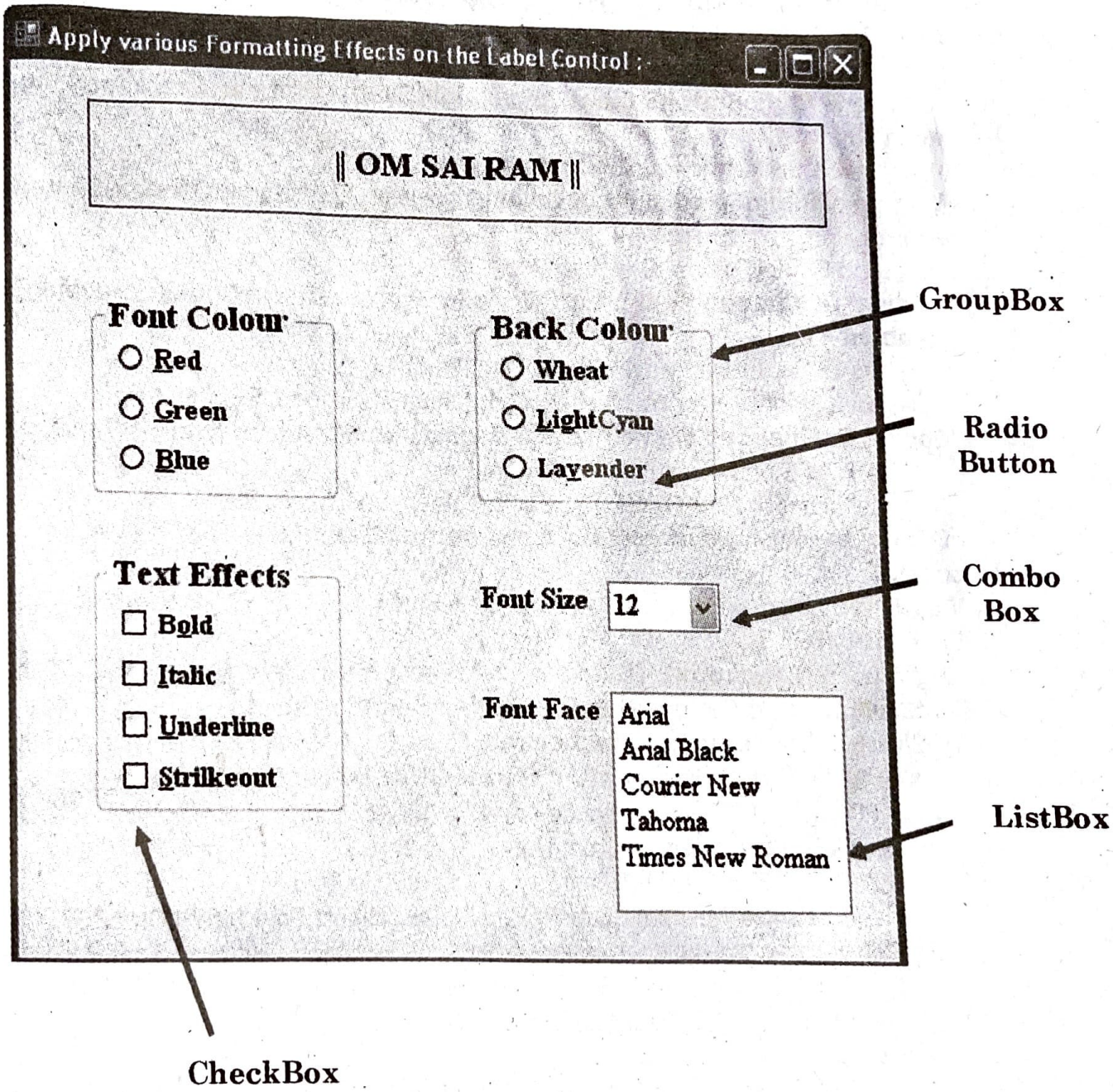
(2) DropDownStyleChanged : It occurs when the value of the DropDownStyle property has changed.

PRACTICAL 5 : Apply various Formatting Effects on the Label Control.

Design the interface as given in figure.

Set **Items** property of ComboBox control to various font size like 12, 14, 18, 20, 22, 30, etc.

Set **Items** property of ListBox control to various fonts like Arial, Arial Black, Courier New, etc.



```
'Coding of RadioButton1 (Font Colour Red) (Write in 'CheckedChanged event)
    Label1.ForeColor = Color.Red
'Coding of RadioButton2 (Font Colour Green) (Write in 'CheckedChanged
event)
    Label1.ForeColor = Color.Green
'Coding of RadioButton3 (Font Colour Blue) (Write in 'CheckedChanged
event)
    Label1.ForeColor = Color.Blue
'Coding of RadioButton4 (Font Colour Wheat) (Write in 'CheckedChanged
event)
    Label1. BackColor = Color.Wheat
'Coding of RadioButton5 (Font Colour LightCyan) (Write in 'CheckedChanged
event)
    Label1.BackColor = Color.LightCyan
'Coding of RadioButton6 (Font Colour Lavender) (Write in 'CheckedChanged
event)
    Label1. BackColor = Color.Lavender
'Coding To Change Font Size (Write in SelectedIndexChanged event 'of
ComboBox)
    Dim FS As Integer = Val(ComboBox1.Text)
    Label1.Font = New Font(Label1.Font.Name, FS, Label1.Font.Style)
'Coding To Change Font (Write in SelectedIndexChanged event of 'ListBox)
    Dim F As String = ListBox1.Text
    Label1.Font = New Font(F, Label1.Font.Size, Label1.Font.Style)
' Coding to give Bold effect (Write in CheckedChanged event of 'CheckBox1
control)
'FontStyle Enumeration has values like Bold=1, Italic=2, Underline=4 '&
Strikeout=8.
'Logic is, if CheckBox1 (Bold) is checked (True), We have to add 'the
FontStyle.Bold to the existing font style of label control.
'If CheckBox1 (Bold) is Unchecked (False), We have to subtract the
'FontStyle.Bold from the existing font style of label control.
    Dim FONT_EFFECT As System.Drawing.FontStyle
    If CheckBox1.Checked = True Then
        FONT_EFFECT = Label1.Font.Style + FontStyle.Bold
        Label1.Font = New Font(Label1.Font.Name, Label1.Font.Size, FONT_EFFECT)
    Else
        FONT_EFFECT = Label1.Font.Style - FontStyle.Bold
        Label1.Font = New Font(Label1.Font.Name, Label1.Font.Size, FONT_EFFECT)
    End If
'Coding to give Italic effect (Write in CheckedChanged event of 'CheckBox2
control)
    Dim FONT_EFFECT As System.Drawing.FontStyle
    If CheckBox2.Checked = True Then
```

```
        FONT_EFFECT = Label1.Font.Style + FontStyle.Italic
        Label1.Font = New Font(Label1.Font.Name, Label1.Font.Size, FONT_EFFECT)
    Else
        FONT_EFFECT = Label1.Font.Style - FontStyle.Italic
        Label1.Font = New Font(Label1.Font.Name, Label1.Font.Size, FONT_EFFECT)
    End If
' Now repeat the same procedure for Underline & Strikeout.
```

PRACTICAL 6: Select the desired products or all the products from one list box to another list box and also count the number of selected products and display in the label.

Design the interface as given in figure. Place two Listbox.

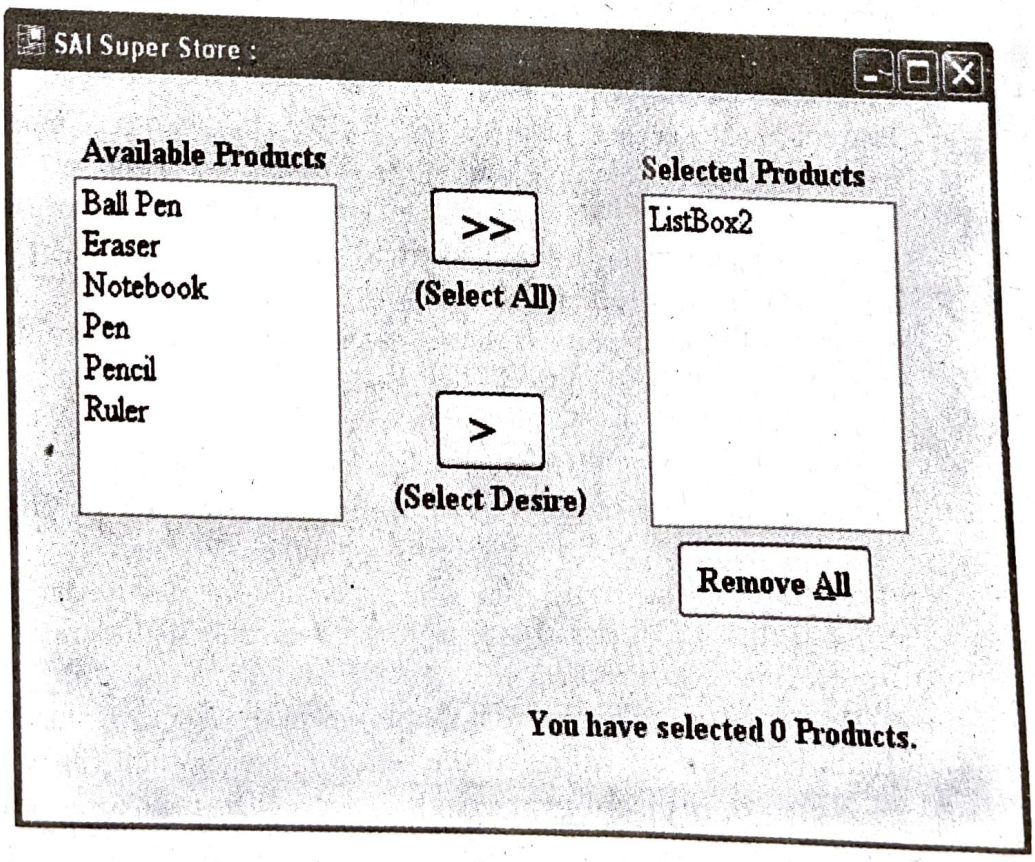
Set **Items** property of ListBox1 control to available products like Ball Pen, Eraser, Notebook, etc

Set **Sorted** property of ListBox1 control to True.

Set **SelectionMode** property of ListBox1 control to MultiExtended.

When Select All button is click, all the available products of ListBox1 will be copy to ListBox12. And no. of selected products should display in Label control (Label5). When Select Desire button is click, no of selected products of ListBox1 will be copy to ListBox12. And no. of selected products should display in Label control (Label5).

When Remove All button is click, ListBox2 should be clear.



```
'Coding of Select All button
    Dim I As Integer
    For I = 0 To ListBox1.Items.Count - 1
        ListBox2.Items.Add(ListBox1.Items.Item(I))
    Next I
        Label5.Text = "You have selected " & ListBox2.Items.Count & " Products."
' Coding of Select Desire button
    Dim I As Integer
    For I = 0 To ListBox1.SelectedItems.Count - 1
        ListBox2.Items.Add(ListBox1.SelectedItems(I))
    Next I
        Label5.Text = "You have selected " & ListBox2.Items.Count & " Products."
' Coding of Remove All button
    ListBox2.Items.Clear()
    Label5.Text = "You have selected 0 Products."
```

[11] **PictureBox Control :** The PictureBox control is used to display graphics in bitmap, GIF, JPEG, metafile, or icon format. To display an image in picture box, you can set the Image property. Image can be set at design time or at run time.

➢ **Property :**

(1) **Image :** Gets or sets the image that the PictureBox displays. Image can be set at design time or at run time. At design time you can add an image to the PictureBox using **Image** property.

At run time you have to set the Image property to an Image object, which you can create using the Image class's FromFile method. This method can load images from bitmap (.bmp), icon (.ico) or metafile (.wmf), JPEG (.jpg), GIF (.gif) files, and other types of files.

```
PictureBox1.Image = Image.FromFile("C:\My Document\OM.bmp")
```

(2) **SizeMode :** Indicates how the image is displayed. This property is used to specify the way the image is displayed. The different size modes that can be specified are Normal, StretchImage, AutoSize, CenterImage or Zoom. The default is Normal.

- **Normal :** The upper-left corner of the image is placed at upper-left in the PictureBox. The image is clipped if it is larger than the PictureBox it is contained in. The default is Normal.
- **StretchImage :** The image within the PictureBox is stretched or shrunk to fit the size of the PictureBox. The image will automatically resize to the size of the PictureBox control.
- **AutoSize :** The PictureBox control is sized equal to the size of the image that it contains. The PictureBox control will automatically resize to the size of the image.
- **CenterImage :** The image is displayed in the center if the PictureBox is larger than the image. If the image is larger than the PictureBox, the picture is placed in the center of the PictureBox and the outside edges are clipped.

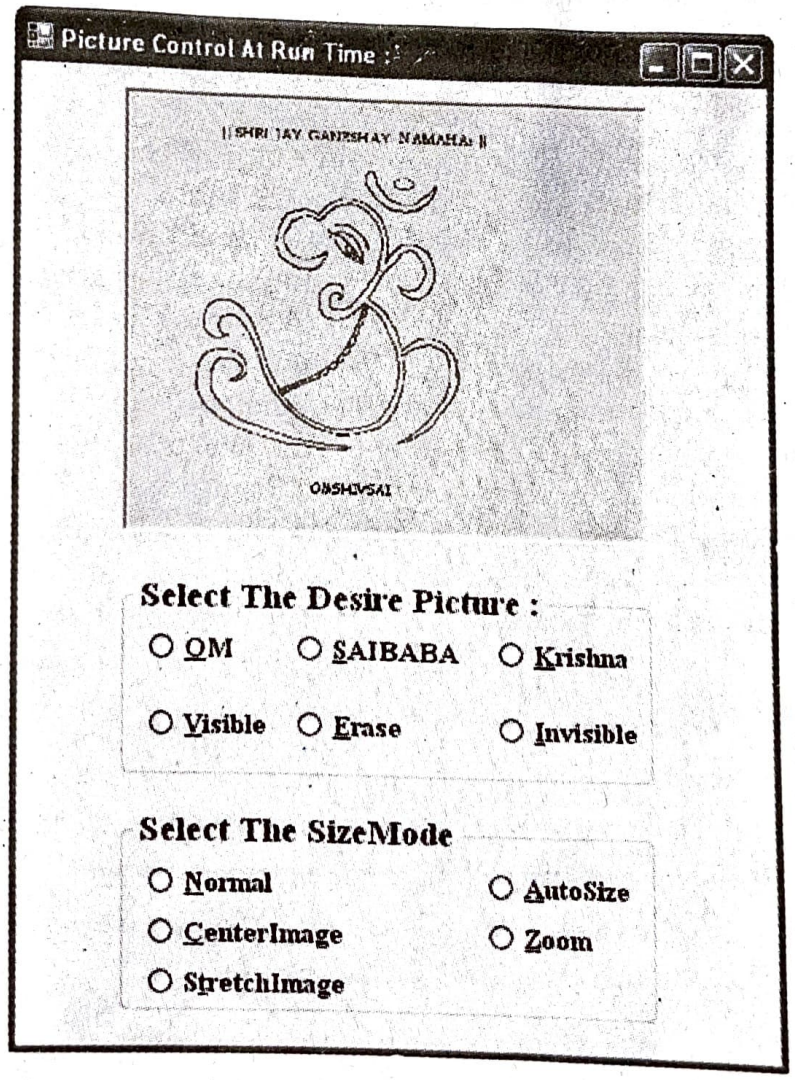
- **Zoom :** The size of the image is increased or decreased maintaining the size ratio.

➢ **Event :**

(1) SizeModeChanged : This event is occurs when SizeMode property is changed. The SizeMode property can be changed programmatically.

PRACTICAL 7: Using option button select various pictures with erase, visible, invisible & SizeMode facilities.

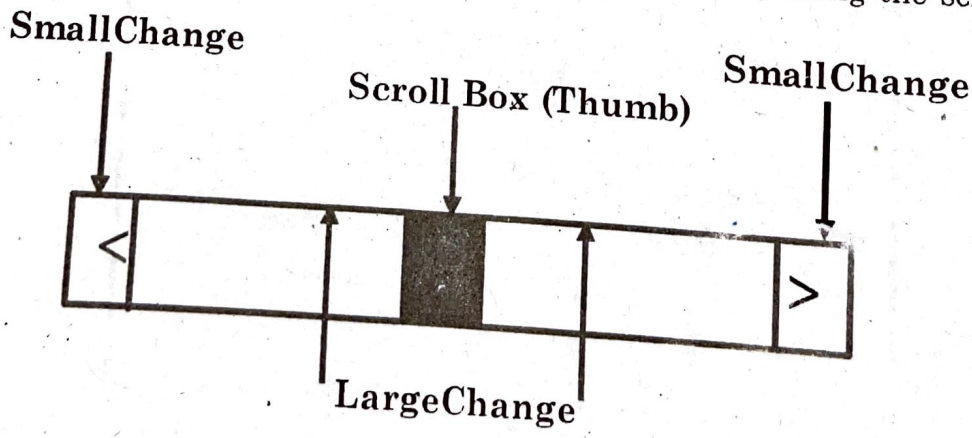
Design the interface as shown in figure. Place one PictureBox control.
Set **BorderStyle** property of PictureBox1 control to Fixed3D.
Set **SizeMode** property of PictureBox1 control to StretchImage, so the image will automatically resize to the size of the PictureBox control.
Set **Image** property of PictureBox1 control to any desire image. (Import /browse the image).
Place two GroupBox, each contain six RadioButtons.



```
'Coding of RadioButton1 (For picture OM) (Write in 'CheckedChanged event)
    PictureBox1.Image = Image.FromFile("C:\Pictures\OM1.JPG")
'Coding of RadioButton2 (For picture SAIBABA) (Write in 'CheckedChanged
event)
    PictureBox1.Image = Image.FromFile("C:\Pictures\ SAIBABA.JPG")
'Coding of RadioButton3 (For picture KRISHNA) (Write in 'CheckedChanged
event)
    PictureBox1.Image = Image.FromFile("C:\Pictures\ KRISHNA.JPG")
'Coding of RadioButton4 (Visible) (Write in CheckedChanged event)
    PictureBox1.Visible = True
'Coding of RadioButton5 (Erase, It removes only pictures) (Write 'in
CheckedChanged event)
    PictureBox1.Image = Nothing
'Coding of RadioButton6 (Invisible) (Write in CheckedChanged 'event)
    PictureBox1.Visible = False
'Coding of RadioButton7 (Normal) (Write in CheckedChanged event)
    PictureBox1.SizeMode = PictureBoxSizeMode.Normal
'Coding of RadioButton8 (CenterImage) (Write in CheckedChanged 'event)
    PictureBox1.SizeMode = PictureBoxSizeMode.CenterImage
'Now repeat the same procedure for other SizeMode
```

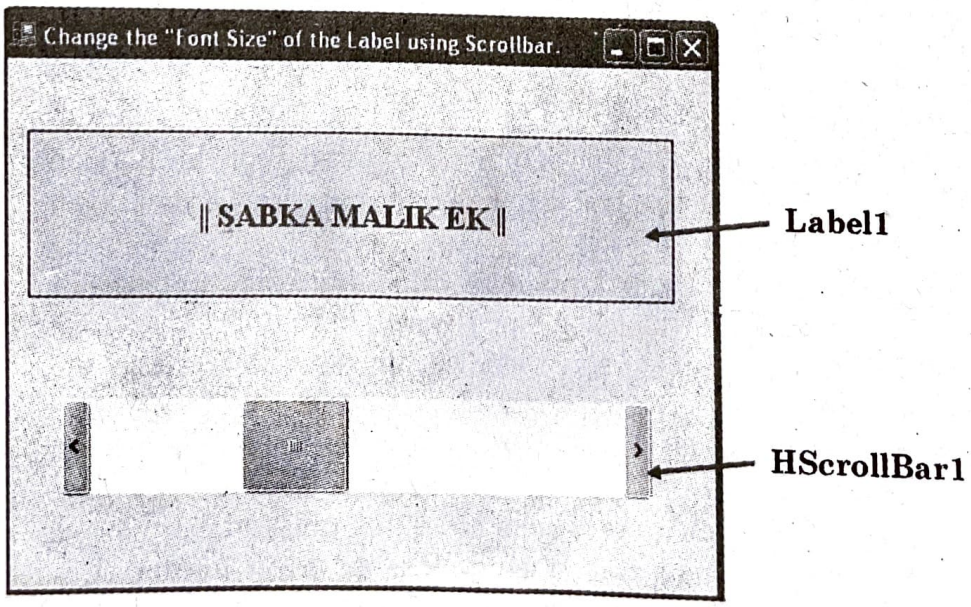
[12] Scroll Bars Control : [HScrollBar (horizontal Scroll Bar) & VScrollBar (vertical Scroll Bar)]

- ScrollBar controls are used to provide easy navigation through a long list of items or a large amount of information by scrolling either horizontally or vertically within an application or control. Scroll bar allows user to select a value by positioning the scroll box (thumb) to desire position (instead of typing a value). Scroll bars are a common element of the Windows interface. Many developers choose to incorporate the ScrollBar control when authoring their own user controls. The HScrollBar (horizontal) and VScrollBar (vertical) controls operate independently from other controls and have their own set of events, properties, and methods. The ScrollBar controls use the Scroll event to monitor the movement of the scroll box (sometimes referred to as the thumb) along the scroll bar.



- **Property :**
  - **(1) Value :** Gets or sets a numeric value that represents the current position of the scroll box (thumb) on the scroll bar control. The Value property (which, by default, is 0) is an integer value corresponding to the position of the scroll box in the scroll bar.
  - **(2) Minimum :** Gets or sets the lower limit of the scrollable range. (By default, is 0).
  - **(3) Maximum :** Gets or sets the upper limit of the scrollable range. (By default, is 100).
  - **(4) SmallChange :** Gets or sets the value to be added to or subtracted from the Value property when user clicks an arrow button. (By default is 1). When the user presses one of the arrow keys or clicks one of the scroll bar arrow buttons, the Value property changes according to the value set in the SmallChange property.
  - **(5) LargeChange :** Gets or sets a value to be added to or subtracted from the Value property when the user clicks in the blank area of the scroll box. (outside the scroll box). The default value is 10
    When the user presses the PAGE UP or PAGE DOWN key or clicks in the scroll bar track on either side of the scroll box, the Value property changes according to the value set in the LargeChange property.
- **Event :**
  - **(1) Scroll :** This event is triggered when the scroll box has been moved (either by the mouse or keyboard).
  - **(2) ValueChanged :** It occurs when the Value property has changed, either by a Scroll event or programmatically.

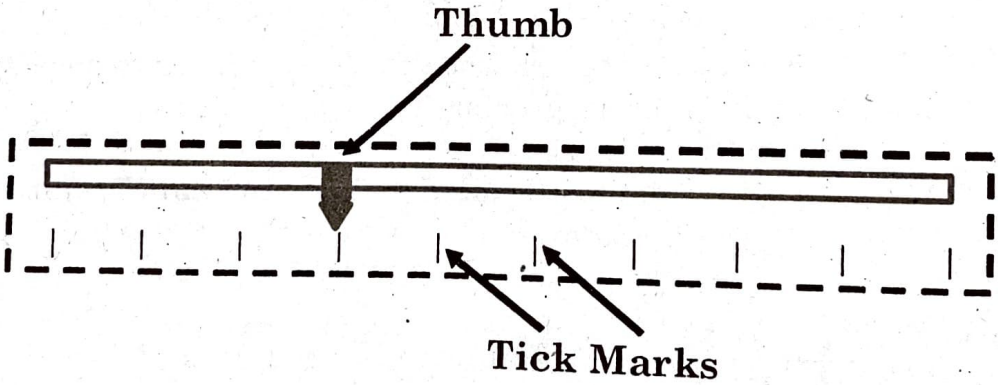
PRACTICAL 8: Change the "Font Size" of the Label control using the Scrollbar control.



Design the interface as shown in figure.
Set **Minimum** property of HScrollBar1 to 1.
Set **Maximum** property of HScrollBar1 to 50.
Set **Value** property of HScrollBar1 to 15.
' Write following Coding in Scroll Event (HScrollBar1_Scroll Event) 'of ScrollBar1. Assign current value of HScrollBar1 to the font 'size of the Label1.

```
Dim V As Integer
V = HScrollBar1.Value
Label1.Font = New Font(Label1.Font.Name, V, Label1.Font.Style)
```

[13] Track Bars Control : TrackBar control is very much like scroll bars but differ in appearance. TrackBar control (also sometimes called a "slider" control) is used for navigating through a large amount of information or for visually adjusting a numeric setting. The TrackBar control has two parts: the thumb, also known as a slider, and the tick marks. The thumb is the part that can be adjusted. Its position corresponds to the Value property. The tick marks are visual indicators that are spaced at regular intervals. The trackbar moves in increments that you specify and can be aligned horizontally or vertically. For example, TrackBar control commonly used in mouse pointer speed setting, volume setting, display properties settings, etc



➢ **Property :**

(1) Orientation : Gets or sets a value indicating the horizontal or vertical orientation of the track bar. Track bar can be display horizontally or vertically.

(2) Value : Gets or sets a numeric value that represents the current position of the slider in the track bar. (By default, is 0)

(3) Minimum : Gets or sets the lower limit of the Track scrollable range. (By default, is 0).

(4) Maximum : Gets or sets the upper limit of the Track scrollable range. (By default, is 10).

(5) SmallChange : Gets or sets the value to be added to or subtracted from the Value property when user clicks an arrow button. (By default is 1)

The value of the SmallChange property is the number of positions the thumb (slider) moves in response to having the LEFT or RIGHT ARROW key pressed.

(6) LargeChange : Gets or sets a value to be added to or subtracted from the Value property when the user clicks on the track bar on either side of the thumb.

The value of the LargeChange property is the number of positions the thumb moves in response to having the PAGE UP or PAGE DOWN key pressed, or in response to mouse clicks on the track bar on either side of the thumb.

(7) TickStyle : Gets or sets a value indicating how to display the tick marks on the track bar.

- **Both :** Tick marks are located on both sides of the control.
- **BottomRight :** Tick marks are located on the bottom of a horizontal control or on the right side of a vertical control. The default is TickStyle.BottomRight
- **None :** No tick marks appear in the control.
- **TopLeft :** Tick marks are located on the top of a horizontal control or on the left of a vertical control.

(8) Tickfrequency : Gets or sets a value that specifies the distance between ticks. The default is 1.

For example, if you have a TrackBar control with a Maximum value of 50, then setting a TickFrequency to 5, here causes the control to draw 10 ticks. In this case, each tick represents five units in the range of values. In short 50 (Maximum) / 5 (TickFrequency) = 10.

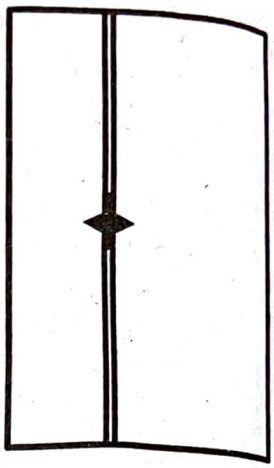
➢ **Event :**

(1) Scroll : This event is triggered when the slider moves (either by the mouse or keyboard).

(2) ValueChanged : It occurs when the Value property of a track bar changes (either by moving the slider or through code).

[14] Splitters Control : Splitter controls are used to resize docked controls at run time. You can use splitters to let the user resize controls. The Splitter control is often used on forms with controls that have varying lengths of data to present, like Windows Explorer, whose data panes contain information of varying widths at different times.

Here is how it works: you add a control to a form and then dock it. Next, you add a splitter control and dock it to the same size of the same container. When you run a program splitter is invisible until the mouse passes over it. As soon as the mouse cursor passes over the splitter, it changes, indicating that the control can be resized.



You can enable or disable the splitter with the **Enabled property**, Set the cursor that appears with the splitter with the **Cursor property** (for vertical splitter the default cursor is VSplit, for horizontal splitter the default cursor is HSplit.

Note : Although **SplitContainer control** replaces and adds functionality to the **Splitter control** of previous versions, Splitter is retained for both backward compatibility and future use if you choose.

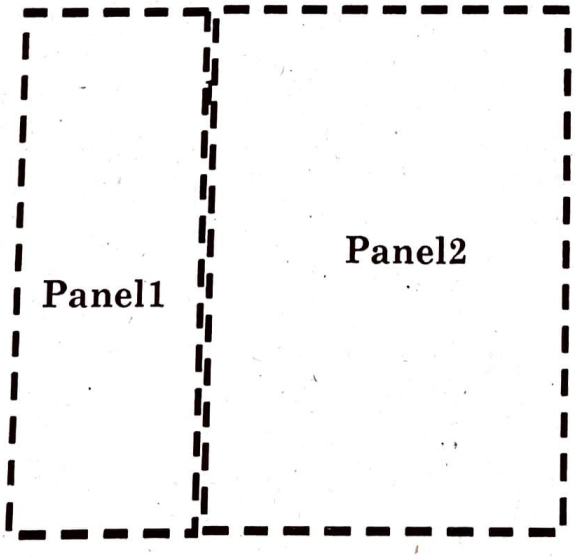
[15] SplitContainer Control : (Container Control)

Note : In the **Toolbox**, **SplitContainer** control replaces the Splitter control that was there in the previous version of Visual Studio. The **SplitContainer** control is much preferred over the **Splitter** control. The **Splitter** class is still included in the .NET Framework for compatibility with existing applications, so use the **SplitContainer** control for new projects.

SplitContainer control contains two panels that are separated by a movable bar. When the mouse pointer is over the bar, the pointer changes shape to show that the bar is movable. You can drop other controls on these panels & arrange them as you like.

Using SplitContainer Control, creating splitter based user interface becomes much easier task.

The **SplitContainer** control is often used on forms with controls that have varying lengths of data to present, like Windows Explorer, whose data panes contain information of varying widths at different times.



- **Property :**
  (1) **FixedPanel :** Gets or sets which SplitContainer panel remains the same size when a user resizes the container. That is either Panel1 or Panel2 or None (By Default).
  (2) **IsSplitterFixed :** Determines if the splitter can be moved with the keyboard or mouse. If it is set to True then, the splitter is fixed. If it is set to False splitter is movable with the keyboard or mouse. The default is False.
  (3) **Orientation : Orientation** property is used to change the orientation of the **SplitContainer** panels from vertical to horizontal or from horizontal to vertical.

- **Event :**
  (1) **SplitterMoving :** Occurs when the splitter control is moving.
  (2) **SplitterMoved :** Occurs when the splitter control has moved. The **SplitterMoved** event is raised when you stop moving the splitter control.

[16] DateTimePicker Control : (Pickers)

DateTimePicker control presents a graphical interface for users to select date information. It is used to select a date and a time and to display the date and time with a specified format.

The DateTimePicker control displays a graphical calendar from which users can select (set) a date required by them. If you click the arrow in the date-time picker, it displays a month calendar, just as a combo box would display a drop-down list. Using the **DateTimePicker** eliminates much of the need for validating date/time values.

- **Property :**
  (1) **MaxDate :** Gets or sets the maximum date and time that can be selected in the control.
  (2) **MinDate :** Gets or sets the minimum date and time that can be selected in the control.
  (3) **Format :** Gets or sets the format of the date and time displayed in the control. The date/time format is based on the user's regional settings in their operating system. The default is Long.
  To display only time in a DateTimePicker, set the **Format** property to Time, and to view up down button set the **ShowUpDown** property to true.
  - **Custom :** The DateTimePicker control displays the date/time value in a custom format. If the format property is set to custom (DateTimePicker Format.Custom), you can create your own format style by setting the **CustomFormat** property.
  - **Long :** The DateTimePicker control displays the date/time value in the long date format set by the user's operating system.

- **Short :** The DateTimePicker control displays the date/time value in the short date format set by the user's operating system.
- **Time :** The DateTimePicker control displays the date/time value in the time format set by the user's operating system.

(4) CustomForamt : Gets or sets the custom date/time format string.

d	:	The one-or-two digit day.
dd	:	The two digit day. Single digit day values are preceded by a zero,
ddd	:	Three character day-of-week abbreviation.
dddd	:	The full day-of-week name.
h	:	The one or two digit hour in 12 hour format.
hh	:	The two digit hour in 12 hour format. Single digit day values are preceded by a zero,
H	:	The one or two digit hour in 24 hour format.
HH	:	The two digit hour in 24 hour format. Single digit day values are preceded by a zero,
MMM	:	The three character month abbreviation. .....Etc....

(5) ShowUpDown : Gets or sets a value indicating whether a spin button control (also known as an up-down control) is used to adjust the date/time value.

➢ **Events :**

(1) DropDown : Occurs when the drop-down calendar is appears (shown).
(2) CloseUp : Occurs when the drop-down calendar is disappears.
(3) ValueChanged : It Occurs when the Value property changes.
(4) FormatChanged : It Occurs when the Format property value has changed.

[17] MonthCalendar Control (Pickers) : MonthCalendar control enables the user to view & to set a date visually. MonthCalendar control presents a graphical interface for users to view and set date information. The control displays a calendar. The selected range of dates is highlighted. You can limit the date and times that can be selected by setting the MinDate and MaxDate properties. You can change the look of the calendar portion of the control by setting the ForeColor, Font, TitleBackColor, TitleForeColor, TrailingForeColor, and BackColor properties.

➢ **Property :**

(1) MaxDate : Gets or sets the maximum allowable date that can be selected in the control.
(2) MinDate : Gets or sets the minimum allowable date that can be selected in the control.
(3) BoldedDates : Gets or sets an array of DateTime specifying which days should be bold.

(4) **FirstDayOfweek :** Gets or sets the first day of the week as displayed in the month calendar.

(5) **ShowToday :** Gets or sets a value indicating whether the today's date is shown at the bottom of the control. (By default is True).

(6) **ShowTodayCircle :** Gets or sets a value indicating whether today's date should appear inside a circled. (By default is True).

(7) **ShowWeekNumbers :** Gets or sets a value indicating whether the month calendar control displays week numbers (1-52) to the left of each row of days. The default is False.

(8) **TodayDate :** Gets or sets the today's date. This value is used by MonthCalendar control as today's date.

➢ **Events :**

(1) **DateChanged :** It occurs when the date in the MonthCalendar control changes.

(2) **DateSelected :** Occurs when the user makes an explicit date selection using the mouse.

This event is similar to the DateChanged event, but it occurs at the end of a date selection made using the mouse. The DateChanged event occurs during any date selection, whether by mouse, keyboard, or code.

[18] **NotifyIcon Control :** Notify icons let you display an icon in the status notification area of the Windows taskbar (at extreme right in the taskbar) called the Windows system tray. Icons in the notification area are shortcuts to processes that are running in the background of a computer, such as a virus protection program or a volume control. Each **NotifyIcon** component displays a single icon in the status area. If you have three background processes and wish to display an icon for each, you must add three **NotifyIcon** components to the form. The key properties of the **NotifyIcon** component are Icon and Visible. The **Icon** property sets the icon that appears in the status area. In order for the icon to appear, the **Visible** property must be set to **true**. You can associate balloon tips, shortcut menus, and ToolTips with a **NotifyIcon** to assist the user.

➢ **Property :**

(1) **Icon :** Gets or sets the current icon. The Icon displayed by the NotifyIcon component. The default value is a null reference (Nothing in Visual Basic).

(2) **ContextMenuStrip :** Gets or sets the shortcut menu (context menu) for the NotifyIcon (tray icon)

(3) **Text :** Gets or sets the ToolTip text which is to be displayed when the mouse pointer rests on a notification area icon. (System tray icon).

(4) **Visible :** Gets or sets a value indicating whether the icon is visible in the notification area of the taskbar.

(5) **BalloonTipIcon :** Gets or sets the icon to display on the balloon tip associated with the NotifyIcon. That is None (by default), Info, Warning & Error.

(6) **BalloonTipText :** Gets or sets the text to display on the balloon tip associated with the NotifyIcon.

(7) **BalloonTipTitle :** Gets or sets the title of the balloon tip display on the NotifyIcon.

➢ **Method :**

(1) **ShowBaloonTip :** Displays a balloon tip in the taskbar for the specified time period.

➢ **Event :**

(1) **Click :** It occurs when the user clicks the NotifyIcon (system tray icon) with the mouse.

(2) **DoubleClick :** It occurs when the user double clicks the NotifyIcon (system tray icon) with the mouse.

(3) **MouseDown :** Occurs when the user presses the mouse button on the icon in the system tray.

(4) **MouseUp** & (5) **MouseMove**

[19] **Tool Tips Control :** Tool Tip control is used to display a small rectangular pop-up window that displays (appears) with explanatory text when the user rests the mouse pointer on the control. Tool tips are used to give quick help (hints) to the user when the mouse pointer rests on the control. Tool tips are components, not controls. So when you add them to a form, they will appear in a component tray. A single ToolTip component typically is used to create ToolTips for multiple controls on a single form. After you create a ToolTip, use a separate call (SetToolTip method) for individual controls.

➢ **Property :**

(1) **Active :** Gets or sets a value indicating whether the ToolTip is currently active. A tip will only appear if the tooltip has been activated. By default it is **true**. With the **Active** property, you can enable or disable the display of ToolTip text.

(2) **AutomaticDelay :** Gets or sets the time (in milliseconds) before the tool tip appears. The default is 500. It specify the time that should elapsed before the toop tip is displayed, when the mouse pointer is moved over the control.

(3) **InitialDelay : (Same as AutomaticDelay) :** With the **InitialDelay** property, you can shorten or lengthen the time that the ToolTip waits before displaying a ToolTip window.

Gets or sets the time that passes before the ToolTip appears.

(4) **AutoPopDelay :** The **AutoPopDelay** property enables you to shorten or lengthen the time that the **ToolTip** window is displayed when the pointer is on a control. For example, if you display extensive Help in a ToolTip window, you can increase the value of this property to ensure that the user has sufficient time to read the text.

The maximum time you can delay a popup is 5000 milliseconds. For longer durations, use the Show method to control the exact moment when the ToolTip is displayed.

(5) **IsBalloon :** Gets or sets a value indicating whether the ToolTip should use a balloon window.

(6) **ToolTipIcon :** Gets or sets a value that defines the type of icon to be displayed alongside the ToolTip text. That is None (by default), Info, Warning & Error.

(7) **ToolTipTitle :** Gets or sets a title for the ToolTip window.

➢ **Method :**

(1) **SetToolTip :** This method is used to set the ToolTip text with the specified control.

Syntax : ToolTip1.SetToolTip(Control, Caption as string)

Example : ToolTip1.SetToolTip(TextBox1, "Plese enter DOB in DD/MM/YYYY format.")

(2) **GetToolTip :** This method is used to get (retrieve) the ToolTip text for the specified control.

Example : MsgBox(ToolTip1.GetToolTip(TextBox1))

It will display : Plese enter DOB in DD/MM/YYYY format.

(3) **RemoveAll :** This method is used to remove all the **Tool Tips text** corresponding to a specific ToolTip component.

(4) **Show :** Sets the text associated with a ToolTip, and then displays it. ToolTip.Show (tool tip **text** as string, Name of **control**, **x** as integer, **y** as integer, **Duration** as integer)

(5) **Hide :** The **Hide** method hides the ToolTip for the specified Control if it is currently being displayed.To instead disable all ToolTip windows associated with the current ToolTipTitle component, set the Active property to **false**.

ToolTip1.Hide(Name of control)

[20] Timer Control : Timer is a component that raises an event at regular user-defined intervals. Timer is a component not control. So when you add them to a form, they will appear in a component tray at design time and it will not appear in a form at run time.

The code written in a **tick event**, is executed at predefined regular interval. The interval property is used to set the time gape between the code executions. (Set in millisecond).

The key methods of the **Timer** component are Start and Stop, which turn the timer on and off. You can also use enabled property of timer component to Start (on) and Stop (off) the timer.

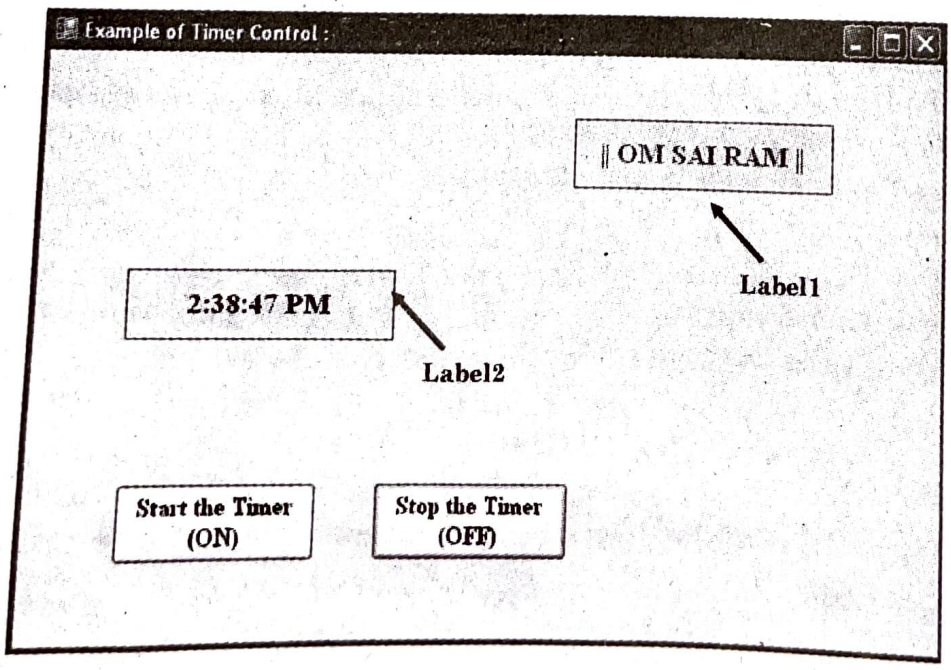
- **Property :**
  (1) Enabled : Gets or sets whether the timer is running. If it is set to true, then the timer is currently enabled (on). If it is set to **false**, then the timer is currently disabled (off). The default is **false**.
  (2) Interval : Gets or sets the time, in milliseconds, that represents the time period between each timer ticks. The interval property is used to set the time gape between the code executions. (Set in millisecond). By default it is 100 milliseconds.
  It is in milliseconds. (1000 milliseconds = 1 second). For example you want to write a code, that should be execute at every 2 seconds of interval, then set interval property to 2000 milliseconds.

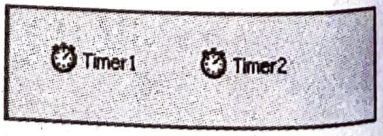
- **Method :**
  (1) Start : It starts the timer. You can also start the timer by setting the Enabled property to true.
  (2) Stop : It stops the timer. You can also stop the timer by setting the Enabled property to false.

- **Event :**
  (1) Tick : This event is triggered (executed) when the specified timer interval has elapsed and the timer is enabled.

PRACTICAL 9 :Move the Label1 from left to right & right to left using Timer Control. Also change the Background & Text colour of Label1 at every two seconds. Label2 should display current time at every one second.

Design the interface as shown in figure. Take two Timer control, two Label Control & two button Control.





Timer1 is used for Label1, to move Label1 & to change colour at every two seconds.
Timer2 is used for Label2, to display current time at every one second.
Set **Interval** property of Timer1 to 2000. & Set **Enable** property of Timer1 to True.
Set **Interval** property of Timer2 to 1000. & Set **Enable** property of Timer2 to True.

```
' Declare the variable in Declaration Section
    Dim Left1 As Boolean
  Dim S(7) As Color
' At Form Load Event assign different colour to Array (S)
  S(0) = Color.Navy
  S(1) = Color.Gray
  S(2) = Color.Khaki
  S(3) = Color.OliveDrab
  S(4) = Color.Violet
  S(5) = Color.Chocolate
  S(6) = Color.WhiteSmoke
  S(7) = Color.Magenta

' Coding of Timer1 (Write in Timer1_Tick Event)
  'Coding To change Colour of Label1.
  Static D As Byte
  Label1.BackColor = S(D)
  Label1.ForeColor = S(D + 1)
  D = D + 1
  If D = 7 Then
     D = 0
  End If
  'Coding To Move Label1 from left to right & right to left.
  If Left1 = False Then
     Label1.Left = Label1.Left + 50   'Move from left to right.
     If Label1.Left >= 400 Then
        Left1 = True
     End If
  ElseIf Left1 = True Then
     Label1.Left = Label1.Left - 50   'Move from right to left.
     If Label1.Left <= 10 Then
        Left1 = False
     End If
  End If

' Coding of Timer2 (Write in Timer2_Tick Event)
  Label2.Text = TimeOfDay
```

```
' Coding of Start the Timer (ON) Button
  Timer1.Start()      ' OR  Timer1.Enabled = True
  Timer2.Start()      ' OR  Timer2.Enabled = True

' Coding of Stop the Timer (OFF) Button
  Timer1.Stop()       ' OR  Timer1.Enabled = False
  Timer2.Stop()       ' OR  Timer2.Enabled = False
```

[21] RichTextBox Control :

- RichTextBox control is used for displaying, entering, and manipulating text with formatting.
- RichTextBox control is an advance version of TextBox. It provides more advanced text formatting features than the standard TextBox control.
- The RichTextBox control does everything the TextBox control does, but it can also display fonts, colors, and links; selected text effects, bullets, alignment, indent, load text and embedded images from a file; and find specified characters, etc.
- Text can be assigned directly to the control, or can be loaded from a Rich Text Format (RTF) or plain text file.
- The RichTextBox control is typically used to provide text manipulation and display features similar to word processing applications such as Microsoft Word.
- You can prevent the user from manipulating some or all of the text in the control by setting the **SelectionProtected** property to true.
- You can use the **Find** method to find strings of text or specific characters.

➢ **Property :**

(1)	**Name**	**(2)**	**Text**	**(3)**	**BackColor**
(4)	**ForeColor**	**(5)**	**Visible**	**(6)**	**Enabled**
(7)	**Left**	**(8)**	**Right**	**(9)**	**Top**
(10)	**Bottom**	**(11)**	**Margin**	**(12)**	**BorderStyle**
(13)	**Size**	**(14)**	**Location**	**(15)**	**Cursor**
(16)	**HideSelection**	**(17)**	**Lines**	**(18)**	**ReadOnly**
(19)	**Locked**	**(20)**	**WordWrap**	**(21)**	**SelectedText**
(22)	**SelectionLength**	**(23)**	**SelectionStart**	**(24)**	**TextLength**

(Same as TextBox Control)

(25) **MultiLine :** Gets or sets a value indicating whether this is a multiline RichTextBox control. (By default is true).

(26) **DetectUrls :** Gets or sets a value indicating whether or not the RichTextBox will automatically format a Uniform Resource Locator (URL) when it is typed into the control.

When this property is set to false the RichTextBox considers the URL as a normal text.

(27) **SelectionIndent :** Indents first line of paragraph.

(28) **SelectionRightIndent :** Sets the right indentation of the paragraph.

(29) **SelectionHangingIndent :** Indents all other lines of paragraph with respect to SelectionIndent.

(30) **SelectionBullet :** Gets or sets a value indicating whether the bullet style is applied to the current selection or insertion point.

(31) **SelectionAlignment :** Gets or sets the alignment to apply to the current selection or insertion point. It can be Left or Center or Right.
For Ex. RichTextBox1.SelectionAlignment = HorizontalAlignment.Center

(32) **SelectionColor :** Gets or sets the text color of the current text selection or insertion point.

(33) **SelectionFont :** Gets or sets the font of the current text selection or insertion point.

(34) **SelectionTabs :** Gets or sets the absolute tab stop positions in a RichTextBox control.

(35) **ZoomFactor :** Gets or sets the current zoom level of the RichTextBox. The value of this property can be between 1/64 and 64.0. A value of 1.0 indicates that no zoom is applied to the control.

➢ **Method of RichTextBox Control :**

(1)	**SetBound**	(2)	**BringToFront**	(3)	**SendToBack**
(4)	**AppendText**	(5)	**Copy**	(6)	**Cut**
(7)	**Paste**	(8)	**SelectAll**	(9)	**Undo**
(10)	**Redo**				

(Same as TextBox Control)

(11) **LoadFile :** Loads the contents of the specified file into the RichTextBox control. The LoadFile method enables you to load an existing RTF or ASCII text file into the RichTextBox control.

For Ex.
```
OpenFileDialog1.ShowDialog()
RichTextBox1.LoadFile(OpenFileDialog1.FileName)
```

(12) **SaveFile :** Saves the contents of the RichTextBox to a file. The SaveFile enables you to save a file to RTF or ASCII text.

For Ex.
```
SaveFileDialog1.ShowDialog()
RichTextBox1.SaveFile(SaveFileDialog1.FileName)
```

(13) **Find :** It searches for specified text within the contents of the RichTextBox control.

For Ex.
```
Dim Posi As Integer = RichTextBox1.Find(s)
```

➢ **Event of RichTextBox Control :**

(1) **LinkClicked :** Triggered (executed) when the user clicks on a link within the text of the RichTextBox control.

(2) **SelectionChanged :** Triggered (executed) when the selection of text within the RichTextBox control has changed.

[22] MaskedTextBox Control :

The MaskedTextBox is an improvement (enhancement) upon text box controls in that they let you specify a format (syntax) for input and avoid wrongly formatted or unexpected inputs. Using the **Mask** property, you can specify the following input without writing any custom validation logic in your application:

- Required input characters.
- Optional input characters.
- The type of input expected at a given position in the mask; for example, a digit, or an alphabetic or alphanumeric character..
- Mask literals, or characters that should appear directly in the MaskedTextBox; for example, the hyphens (-) in a phone number, or the currency symbol in a price.
- Special processing for input characters; for example, to convert alphabetic characters to uppercase.

As the user types input into the masked text box, valid input characters replace their respective prompt characters in a sequential fashion. If the user types an invalid input character, no replacement occurs, but instead a beep is issued if the **BeepOnError** property is set to **true**, and the **MaskInputRejected** event is raised. You can provide your own custom error logic by handing this event.

➢ **Property :**

(1) Mask : Gets or sets the input mask to use at run time. A String representing the current mask. The default value is the empty string which allows any input.

Mask must be a string composed of one or more of the masking elements, as shown in the following table.

Masking element	Description
L	Letter, **required**. Letters a-z and A-Z.
?	Letter, **optional**. Letters a-z and A-Z
&	Character, **required**. Any non-control character.(For ex.. + @ $ & etc & A to Z & 0 to 9)
C	Character, **optional**. Any non-control character.(For ex.. + @ $ & etc & A to Z & 0 to 9)
A	Alphanumeric (Only nos & alphabets), **optional**.
<	Converts all characters that follow to lowercase.
>	Converts all characters that follow to uppercase.
0	Digit, **required**. This element will accept any single digit between 0 and 9.
9	Digit or space, **optional**.
#	Digit or space, optional. If this position is blank in the mask, it will be rendered as a space in the Text property. Plus (+) and minus (-) signs are allowed.

For example, If you want Employee No. that should be start with 3 alphabets, then dash (hyphen) & then followed by 3 digits. Like MAR-007, EDP-101, MAR-112, EDP-786, EXP-008, etc.
In mask property, you have to write : **LLL-000**

(2) **BeepOnError :** If the user types an invalid input character, then masked text box control raises the system beep if the **BeepOnError** property is set to **true**. By default it is set to False.

(3) **TextMaskFormat :** It determines whether literal characters, prompt characters, or both are included in the Text property. When prompt characters are excluded, they are transformed into spaces in the formatted string.

For Ex. In **Mask** property we have set **LLL-000**
And if we enter **SAI-1__** in MaskedTextBox control. Then **Text** property will return will different result based on respective setting in **TextMaskFormat** property.

ExcludePromptAndLiterals	Return only text input by the user. In our example **Text** property will return **SAI1**
IncludeLiterals	Return text input by the user as well as any literal characters defined in the mask. In our example **Text** property will return **SAI-1**
IncludePrompt	Return text input by the user as well as any instances of the prompt character. In our example **Text** property will return **SAI1__**
IncludePromptAndLiterals	Return text input by the user as well as any literal characters defined in the mask and any instances of the prompt character. In our example **Text** property will return **SAI-1__**

➢ **Event :**

(1) **MaskChange :** Occurs after the input mask is changed.

(2) **MaskInputRejected :** If the user types an invalid input character, then **MaskInputRejected** event is raised. You can provide your own custom error logic by handing this event. MaskInputRejected is the default event for the MaskedTextBox class.

[23]LinkLabel Control :

- The LinkLabel control is similar to a Label control with the exception that it can display a hyperlink.
- Multiple hyperlinks can be specified in the text of the control. Each hyperlink can perform a different task within an application..
- In addition to all the properties, methods, and events of the Label control, the LinkLabel control has properties for hyperlinks and link colors. The **LinkColor**, **VisitedLinkColor**, and **ActiveLinkColor** properties set the colors of the link. The **LinkClicked** event executed when a link is clicked within the control.

➢ **Property :**

(1) LinkBehavior : Gets or sets a value that represents the behavior of a link. The default is LinkBehavior.SystemDefault.

AlwaysUnderline	The link always displays with underlined text.
HoverUnderline	The link displays underlined text only when the mouse is hovered over the link text.
NeverUnderline	The link text is never underlined. The link can still be distinguished from other text by use of the LinkColor property of the LinkLabel control.
SystemDefault	The behavior of this setting depends on the options set using the Internet Options dialog box in Control Panel or Internet Explorer.

(2) LinkArea : Gets or sets the range in the text to treat as a link.

Syntax : System.Windows.Forms.LinkArea(Start as Integer, Length as Integer)

Ex.1 : Suppose you have written following text in to the LinkLabel control.
For On Line (Live Darshan) of **SIDDHIVINAYAK, MUMBAI** click here : www.siddhivinayak.org
Now to create a Link to the web site, you can set LinkArea property either at design time or at run time.
So above example our Link starts at 65 character & length is 21 characters. So write following line in form load event.
LinkLabel1.LinkArea = New System.Windows.Forms. Link Area(65, 21)

Output will be as follows :

For On Line (Live Darshan) of SIDDHIVINAYAK, MUMBAI click here : www.siddhivinayak.org

Ex. 2 : Suppose you have written following text in to the LinkLabel control.
For On Line (Live Darshan) of **SHIRDI SAI BABA** click here : www.shrisaibabasansthan.org
Now to create a Link to the web site, you can set LinkArea property either at design time or at run time.
So above example our Link starts at 59 character & length is 27 characters. So write following line in form load event.
LinkLabel2.LinkArea = New System.Windows.Forms.Link Area(59, 27)

Output will be as follows :
For On Line (Live Darshan) of SHIRDI SAI BABA click here : www.shrisaibabasansthan.org

(3) LinkColor : Gets or sets the color used when displaying a normal link. The default color is specified by the system, typically this color is Color.Blue.

(4) LinkVisited :.To display the link as a visited link, you can set the value of this property to true. A visited link is displayed using the color specified in the **VisitedLinkColor** property of the LinkLabel control.

(5) VisitedLinkColor : Gets or sets the color used when displaying a link that that has been previously visited. The default color is specified by the system, typically this color is Color.Purple.

(6) Links : Gets the collection of links contained within the LinkLabel. A LinkLabel control can display any number of links within the text of the control. You can then use the members of the LinkLabel.LinkCollection class to add, remove, and find links in the collection.

➢ **Event :**

(1) Linkclicked : Occurs when a link is clicked within the control.

[23] Form :

- Forms are the main building block of VB project. Forms are the windows on which controls are placed. It is the "base" on which user interface is build.
- A Windows form is in the System.Windows.Forms namespace and the form class is System.Windows.Forms.Form.
- You can also use the Form class to create modal windows such as a dialog box.
- A special kind of form, the multiple-document interface (MDI) form, can contain other forms called MDI child forms. An MDI form is created by setting the IsMdiContainer property to true.
- Current form can be identified by Me keyword.

➢ **SDI & MDI :**

SDI (Single Document Interface) : Notepad, MS Paint is the example of SDI. In SDI you can only edit & view one file at a time. Means you can not view, edit or compare many documents simultaneously.

MDI (Multiple Document Interface) : MDI allows us to work on multiple documents under one roof (same application). Where you can view, edit or compare many documents simultaneously. MS Word, MS Excel is the example of MDI that allows us to work multiple files at a time. Outermost window that hold menu, toolbar it called as a container form (MDI parent form). & innermost windows in which individual documents are display, are called as a MDI child forms.

Modal & Modeless :
Modal forms or modal dialog box will not allow the user to perform any action while it is on display, so user must response to dialog box, by either clicking on ok or cancel or close the dialog box. In MS Word file open, save etc are the examples of modal dialog box. In VB .Net InputBox, MsgBox are the examples of modal dialog box. Modeless forms or Modeless dialog box will allow the user to continue to work in the application while it is on display, so user can continue to work without having to close the dialog box. In MS Word find, find & replace are the examples of Modeless dialog box. In VB .Net Property Window, Solution Explorer, Tool Box are the examples of Modeless dialog box

➢ **Property :**

(1)	**Name**	**(2)**	**BackColor**	**(3)**	**ForeColor**
(4)	**Visible**	**(5)**	**Enabled**	**(6)**	**Height**
(7)	**Width**	**(8)**	**Left**	**(9)**	**Top**
(10)	**Cursor**	**(11)**	**TabStop**		

(12) WindowState :
Gets or sets the form's window state. This property determines the mode in which the form is displayed. The default is FormWindowState.Normal.

Maximized : A maximized window.
Minimized : A minimized window.
Normal : A default sized window.

(13) Text : Gets or sets the text associated with this control. It is the text that is display in the title bar of the form.

(14) BackgroundImage : Gets or sets the background image displayed in the form.

(15) BackgroundImageLayout : Use the **BackgroundImageLayout** property to specify the position and behavior of an image you have placed onto a control. **BackgroundImageLayout** takes effect only if the BackgroundImage property is set. Its value can be Center , None, Stretch, Tile, or Zoom. **Tile** is the default value.

(16) Icon : Gets or sets the icon for the form.

(17) **AcceptButton** : Gets or sets the button control (Name of button control) on the form that is clicked when the user presses the ENTER key..

(18) **CancelButton** : Gets or sets the button control (Name of button control) that is clicked when the user presses the ESC key.

(19) **MdiParent : (Dynamic Property)** : Gets or sets the current MDI parent form of this form.
To create an MDI child form, assign the **Parent Form** (that will be the MDI parent form) to the **MdiParent** property of the child form.

```
Form1.MdiParent = Me
Form1.Show()
```

(20) **MdiChildren : (Dynamic Property)** : Returns an array of forms that represent the MDI child forms that are parented to this form. You can use this property to loop through all the MDI child forms to perform operations.

(21) **IsMdiContainer** : Gets or sets a value indicating whether the form is a container for MDI child forms. By default value is False. An MDI form is created by setting the IsMdiContainer property to true.

(22) **IsMdiChild : (Dynamic Property)** : Gets a value indicating whether the form is an MDI child form. (True or False)

(23) **Controls** : Returns the collection of controls contained within the control.When controls are added to a Form, each of the controls is a member of the **Control.ControlCollection** assigned to the **Controls** property of the form, which is derived from the **Control** class

(24) **DialogResult** : Gets or sets the dialog result for the form. The DialogResult of a form is the value that is returned from the form when it is displayed as a modal dialog box. It can be Cancel, Ignore, No,None, OK, Retry, Yes.

(25) **ShowInTaskbar** : Gets or sets a value indicating whether the form is displayed in the Windows taskbar. If it is set to true then, form is display in the Windows taskbar at run time. By default is True.

(26) **ShowIcon** : Gets or sets a value indicating whether a form's icon is displayed in the caption bar (title bar) of the form. By default is True.

(27) **ControlBox** : Gets or sets a value indicating whether a control box is displayed in the caption bar (title bar) of the form. By default is True.

(28) **MaximizeBox** : Gets or sets a value indicating whether the **Maximize** button is displayed in the caption bar (title bar) of the form. By default is True.

(29) **MinimizeBox** : Gets or sets a value indicating whether the **Minimize** button is displayed in the caption bar (title bar) of the form. By default is True.

(30) **FormBorderStyle** : Gets or sets the border style of the form. The default is **FormBorderStyle.Sizable**.

Fixed3D	:	A fixed, three-dimensional border. (It can not be resize).
FixedDialog	:	A thick, fixed dialog-style border. (It can not be resize & It does not show control box).
FixedSingle	:	A fixed, single-line border. (It can not be resize).
FixedToolWindow	:	A tool window border that is not resizable. A tool window does not appear in the taskbar or in the window that appears when the user presses ALT+TAB.
None	:	No border. It does not show title bar.(min, max, close, control box).
Sizable	:	A resizable border. (By Default). It display title bar, min ,max control box.
SizableToolWindow	:	A resizable tool window border. A tool window does not appear in the taskbar or in the window that appears when the user presses ALT+TAB.

(31) **Modal :** Gets a value indicating whether this form is displayed modally. It is read only property. You can use this property to determine whether a form has been displayed modally or not.

(32) **KeyPreview :** Gets or sets a value indicating whether the form will receive key events before the event is passed to the control that has focus.

When this property is set to **true**, the form will receive all KeyPress, KeyDown, and KeyUp events. After the form's event handlers have completed processing the keystroke, the keystroke is then assigned to the control that has focus.

For example, if the **KeyPreview** property of the form is set to **true** and the currently selected control is a TextBox. When you type in textbox, first all the key events of the form is executed & then **TextBox** control will receive the key that was pressed (then key events of **TextBox** control is executed).

➢ **Methods :**

(1) **SetBound** (2) **BringToFront**

(3) **SendToBack** (4) **Focus**

(5) **Activate :** Activates the form and gives it focus. Activating a form brings it to the front if this is the active application. The form must be visible for this method to have any effect.

(6) **Close :** Closes the form. For example: Me.Close( )

(7) **Show :** Makes the form display by setting the visible property to true.

(8) **ShowDialog :** Shows the form as a modal dialog box.

(9) **Hide :** Hides the form.

(10) **Dispose :** Releases the resources used by the form.

(11) **LayoutMdi :** Arranges the multiple-document interface (MDI) child forms within the MDI parent form. You can arrange in four ways as shown below.

- **ArrangeIcons :** All MDI child icons are arranged within the client region of the MDI parent form.
- **Cascade :** All MDI child windows are cascaded within the client region of the MDI parent form.
- **TileHorizontal :** All MDI child windows are tiled horizontally within the client region of the MDI parent form.
- **TileVertical :** All MDI child windows are tiled vertically within the client region of the MDI parent form.

For Ex. Me.LayoutMdi(MdiLayout.Cascade)

➢ **Events :**

(1)	**Click**	(2)	**DoubleClick**	(3)	**MouseDown**
(4)	**MouseEnter**	(5)	**MouseHover**	(6)	**MouseLeave**
(7)	**MouseMove**	(8)	**MouseUp**	(9)	**MouseWheel**
(10)	**MouseClick**	(11)	**MouseDoubleClick**		
(12)	**TextChanged**	(13)	**KeyPress**	(14)	**KeyDown**
(15)	**KeyUp**	(16)	**Enter**	(17)	**Leave**
(18)	**GotFocus**	(19)	**LostFocus**	(20)	**Validating**
(21)	**Validated**				

(22) **Activated :** Occurs when the form is activated in code or by the user.

(23) **Deactivate :** Occurs when the form loses focus and is not the active form.

(24) **FormClosed :** Occurs after the form is closed. The **FormClosed** event occurs after the form has been closed by the user or by the Close method or the Exit method of the Application class.
Note: This event is new in the .NET Framework version 2.0.

(25) **FormClosing :** Occurs before the form is closed. When a form is closed, all resources created within the object are released and the form is disposed. If you cancel this event, the form remains opened. To cancel the closure of a form, set the Cancel property of the e arguments passed to your event handler to **true**.
Note : This event is new in the .NET Framework version 2.0.

Ex. 1 : Following example shows, whenever user close the form either with Close method, or using close button of the form or using control box, then message box will be display.

```
Private Sub Form1_FormClosing(ByVal sender As Object, ByVal e As
System.Windows.Forms.FormClosingEventArgs) HandlesMe.Form Closing
    If e.CloseReason = CloseReason.UserClosing Then
```

```
            Me.DialogResult = MsgBox("Do U want to exit?",MessageBox
            Buttons.YesNo,"Quit")
            If Me.DialogResult = Windows.Forms.DialogResult.Yes Then
                End
            Else
                e.Cancel = True
            End If
        End If
    End Sub
```

(26) **ControlAded :** Occurs when a new control is added to the Control.ControlCollection. **Control.ControlCollection** Class : Represents a collection of Control objects.

(27) **ControlRemoved :** Occurs when a new control is removed from the Control.ControlCollection.

(28) **Load :** Occurs before a form is displayed for the first time.

(29) **Move :** Occurs when the control (form) is moved.

(30) **Paint :** Occurs when the control (form) is redrawn.

(31) **Resize :** Occurs when the control (form) is resized.

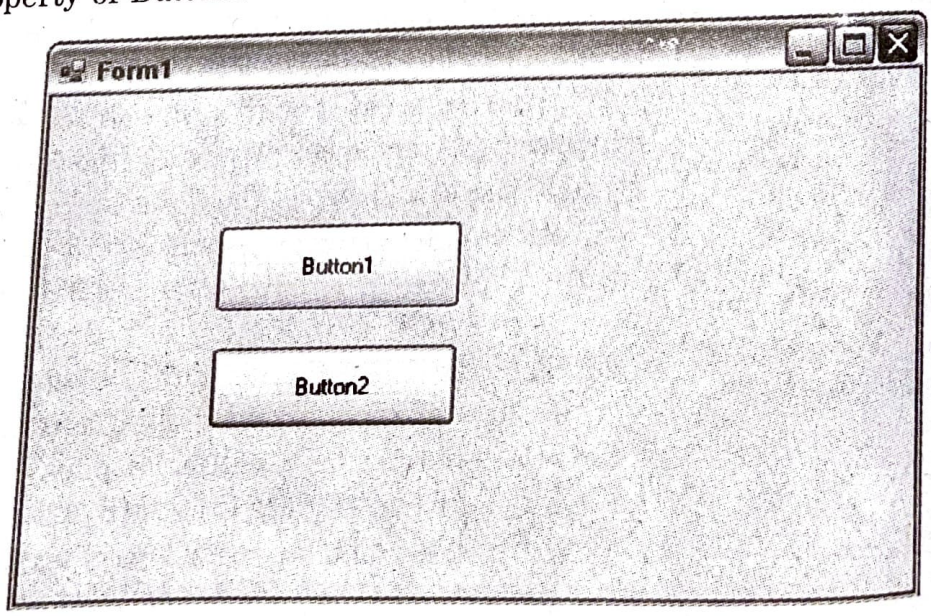
➢ **Anchoring & Docking :**

[a] Anchor Property : The Anchor property of a control allows us to set one or more borders of a control to remain at a fixed distance from the borders of the form (container).

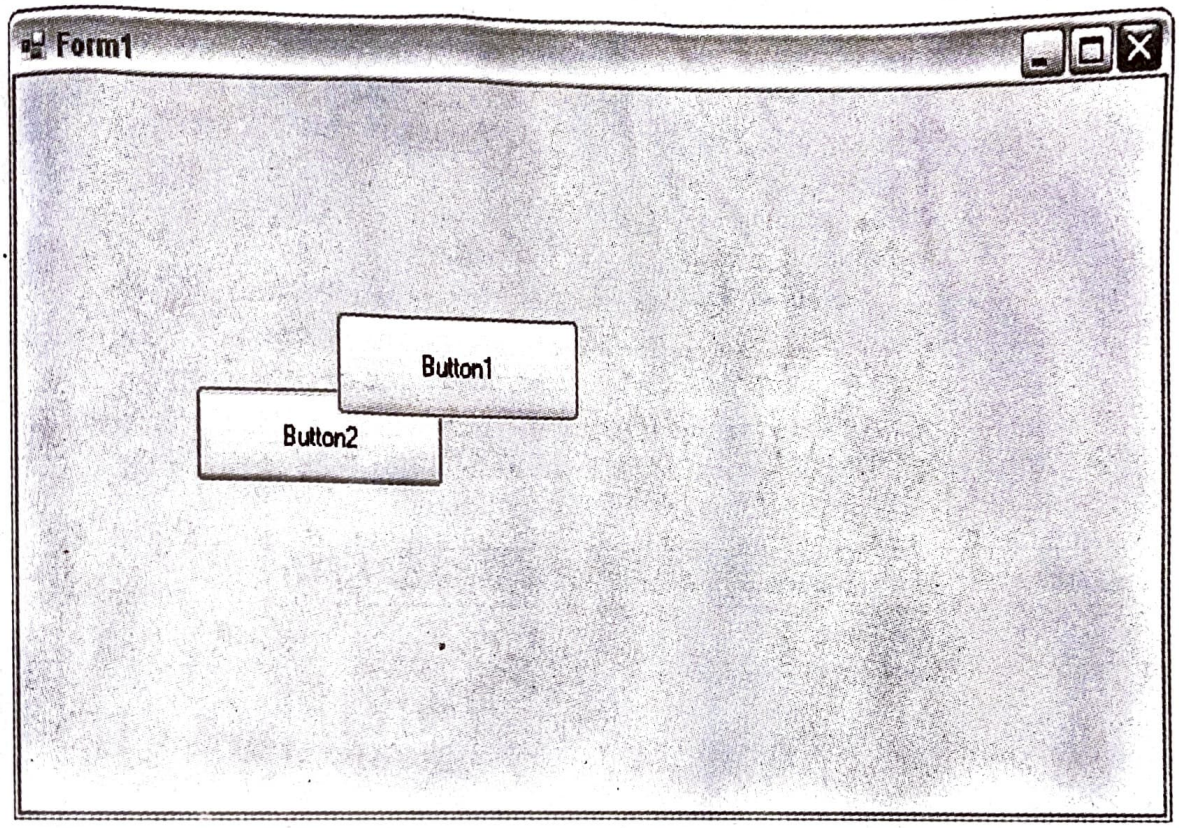
The distance from the anchored edge always remains the same as the distance defined when the control is positioned in the Windows Forms Designer.

Example :

- **At Design Time :** In following form (example) Anchor property of Button1 is set to None (Button1 is not Anchor) While Anchor property of Button2 is set to Top, Left.



- **At Run Time: (Output):** Position of the button1 changes when you resize the form, because it is not Anchor.
  While for Button2 maintain the same distance form Top, Left (Top & Left distance of Button2 remain Fixed). The position of the Button2 from the top, left borders of the form remains same as when the form is resize.



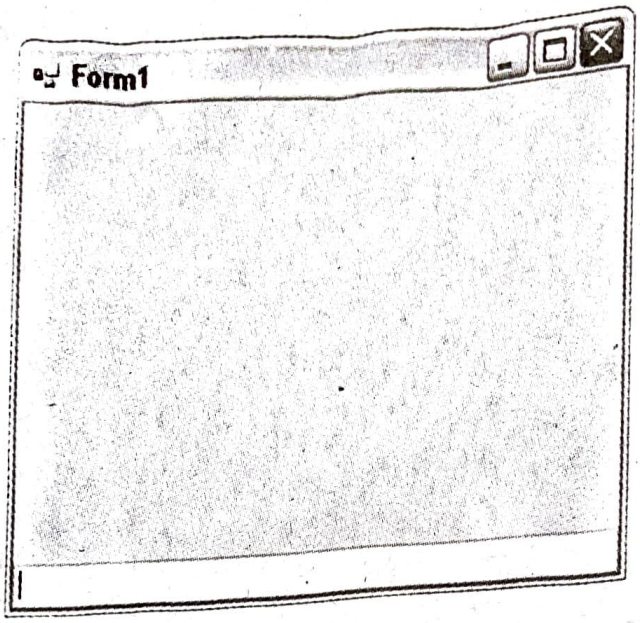
[b] Dock Property : You can dock controls to the edges of your form or have them fill the control's container (either a form or a container control). For example, Windows Explorer docks its TreeView control to the left side of the window and its ListView control to the right side of the window.

Dock property can be set to Left, Fill, Bottom, None, Right, Top

The Difference between **Anchor & Dock** property is that Anchoring is with respect to the position of the control on the form, that is its x,y coordinate value, docking is with respect to the entire edge of the form.

Example :

- **At Design Time :** Set **Dock** property of TextBox control to the **Bottom.**



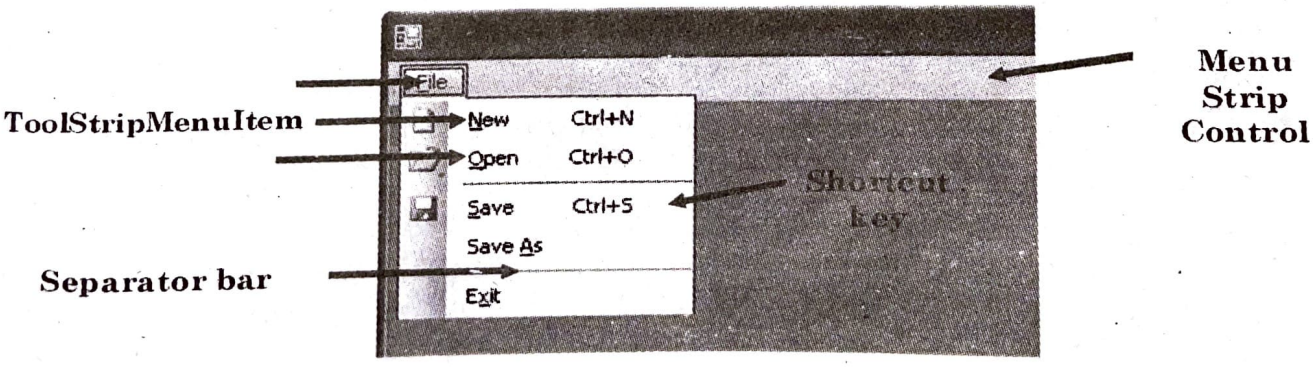
- **At Run Time: (Output) :** Now whenever you resize the form TextBox remains docked to the entire edge of the form.

2.6 WORKING WITH MENUS & DIALOG BOXES

[1] **Menu :** Menu is the one of the common elements of the window user interface. Menu is used to present multiple choices & allow user to perform some action or command. It gives a program a professional look & helps the user to navigate through the application. The menu bar appears immediately below the title bar on the form & contains one or more menu items. Each menu item on the menu bar can display another menu called the sub menu, containing some more menu items. The items on the sub menu are called sub menu items.

- **MenuStrip Control OR MainMenu Control :** The **MenuStrip** control is new addition in Visual Studio 2005 and the .NET Framework version 2.0. With the control, you can easily create menus like those found in Microsoft Office. The **MenuStrip** control represents the container of **ToolStripMenuItem** objects. (All ToolStripMenuItem resides in **MenuStrip).** Any item that resides on a menu is called a menu item & represents an individual part of the menu.
  - **Property :**
    (1) Items : Gets all the items that belong to a ToolStrip. Use the **Items** collection to retrieve all the items that have been added to the **ToolStrip.** Use the DisplayedItems property to retrieve only the items that are currently displayed on the **ToolStrip**.
    (2) MdiWindowListItem : Gets or sets the ToolStripMenuItem that is used to display a list of MDI child forms. Set the name of ToolStripMenuItem (for example Windows), so it will display list of all the open MDI child forms on that ToolStripMenuItem.
  - **Event :**
    (1) ItemClicked : Occurs when the ToolStripItem is clicked.
    (2) ItemRemoved : Occurs when a ToolStripItem is removed from the ToolStripItemCollection.
    (3) ItemAdded : Occurs when a new ToolStripItem is added to the ToolStripItemCollection.

- **ToolStripMenuItem : OR MenuItem :**
  ToolStripMenuItem is a collection of the items in the menu. Any item that resides on a menu is called a menu item & represents an individual part of the menu. Following figure shows individual menuitem (File, New, Open, Save, etc) called as ToolStripMenuItem resides on a MenuStrip control. Using ToolStripMenuItem class you can display a check mark next to a menu item, create a menu separator, assign shortcut keys to menu items, add images to the menu items etc. In order for a ToolStripMenuItem to be displayed, you must add it to a MenuStrip or ContextMenuStrip.



- **Property :**

  (1) Checked : This property indicates whether the specified menu item (ToolStripMenuItem) will have a check mark beside its text. If this property is set to true, then a check mark is placed beside the specified menu item.

  (2) IsMdiWindowListEntry : (Dynamic, Read only property) : If it returns true, then the name of the active child windows will be displayed in the menu item.

  (3) Text : Gets or sets the text that is to be displayed on the item.

  → **To display an access key for a menu command :** Access keys make it possible to select menu items from the keyboard using the Alt key. When you set the Text property for the menu command, enter an ampersand (&) before the letter you want to be underlined as the access key. For example, typing &Open as the Text property of a menu item will result in a menu command that appears as Open.

  → **To create a menu separator :** You can assign a single hyphen (-) to a menu item's Text property. Separator bar is used to divide menu items into logical groups.

  (4) ShortcutKeys : Gets or sets the shortcut keys associated with the ToolStripMenuItem.

  Use the **ShortcutKeys** property to assign a key combination to a menu command, such as CTRL+C for the **Copy** command.

  Short cut key is similar to access key but instead of opening a menu, they run a command when access key is pressed. For example CTRL + C for copy.

  (5) ShowShortcutKeys : Gets or sets a value indicating whether the shortcut keys are displayed next to the ToolStripMenuItem.

  If this value is set to true, then the short cut key such as Ctrl + C would be displayed beside the menu item. The default is true.

  (6) Visible **(7) Enabled** **(8) Image**

- **Method :**

  (1) PerformClick : This method triggers the Click event of the menu item without requiring the user to actually click on the menu item.

- **Event :**

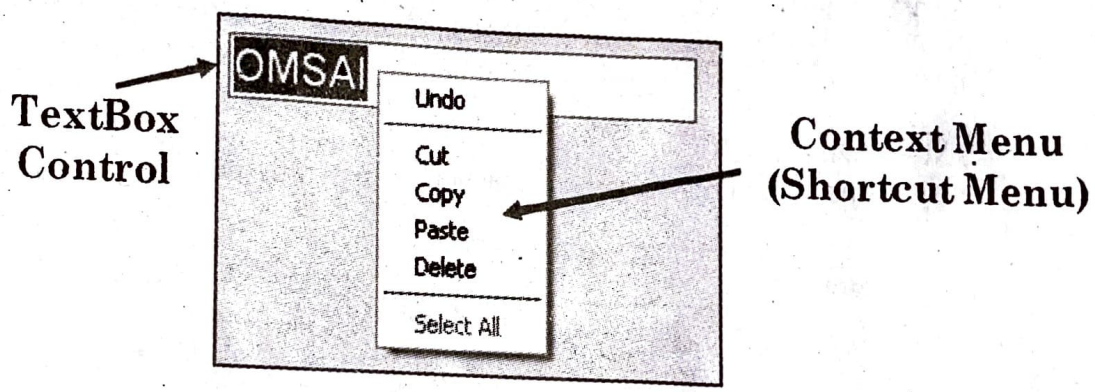
  (1) Click : This event is triggered when the menu item is clicked with the mouse or by pressing the shortcut key or access key.

➢ **ContextMenuStrip OR ContextMenu :**

The **ContextMenuStrip** class represents shortcut menus that are displayed when the user clicks the right mouse button over a control or area of the form. Shortcut menus are typically used to give users access to frequently

used commands. Context Menu is also known as Shortcut menu or Popup menu. Context menus are assigned mostly to the controls. For example, following figure shows example of shortcut menu (context menu) assigned to a TextBox control to provide clipboard features for cut, copy, paste, delete, undo in the TextBox.

Creating context menus is much like creating standard menus. First of all you need to add a **ContextMenuStrip** control to a Windows form. Then adds the menu items as per need. You can assign context menus with other controls (for ex. TextBox control) by setting the control's **ContextMenuStrip** property to the **ContextMenuStrip** control.
In fact, the only major difference standard menus & context menus is that context menus are not divided into separate menus, like File, Edit, Format, Window, etc.



- **Property :**

(1)	**Checked**	(2)	**CheckState**	(3)	**Text**
(4)	**ShortcutKeys**	(5)	**ShowShortcutKeys**	(6)	**Visible**
(7)	**Enabled**				

- **Event :**
  (1) Click : This event is triggered when the menu item is clicked with the mouse or by pressing the shortcut key or access key.

[2] InputBox Function : InputBox function is used to create a modal dialog box, that is used to get (retrieve) one piece of information (a string of text) from the user. InputBox displays a dialog box with title, message, one textbox, one OK & Cancel button. To get a multiple piece of information you have to create your own custom dialog box.

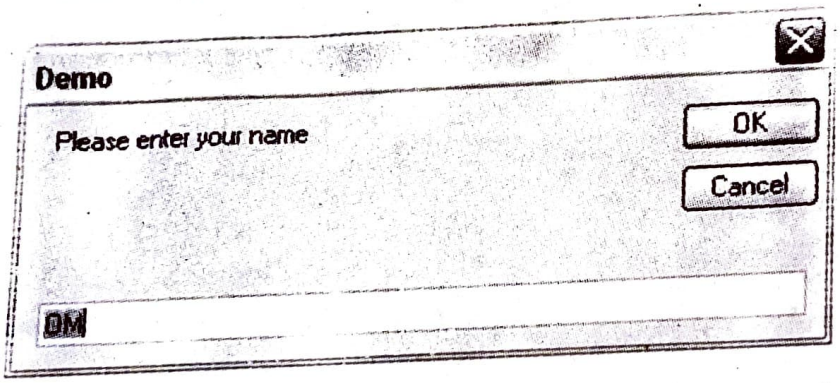
Syntax :

InputBox(**Prompt** As String [, **Title** As String = "", **DefaultResponse** As String = "",**Xpos** As Integer = -1, **YPos** As Integer = -1]) As String

And here are the arguments (parameters) for this function :

- **Prompt :** A string expression displayed as the message in the dialog box. You can separate the lines using a carriage return character **Chr(13)**, a line feed character **Chr(10)**.
- **Title :** (Optional) String expression displayed in the title bar of the dialog box. Note that if you omit Title, the application name is placed in the title bar.
- **DefaultResponse :** (Optional) A string expression displayed in the text box as the default response if no other input is provided. Note that if you omit **DefaultResponse**, the displayed text box is empty.
- **XPos :** (Optional) The distance in pixels of the left edge of the dialog box from the left edge of the screen. Note that if you omit **XPos**, the dialog box is centered horizontally.
- **YPos :** (Optional) The distance in pixels of the upper edge of the dialog box from the top of the screen. Note that if you omit **YPos**, the dialog box is positioned vertically about one-third of the way down the screen.

Ex.
```
Dim ans As String
ans = InputBox("Please enter your name", "Demo", "OM", 300, 300)
```



In above example InputBox will be displayed with the title "Demo" & the default response "OM" in the text box.

If you provide the input name it will be store in the variable named "ans" and if you do not you provide the input name, then the given default response "OM" will be store in the variable named "ans".

[3] MsgBox Function : MsgBox function is used to display a message to the user & get user response (waits for the user to click a button), and then returns an integer value indicating which button of the message box was user clicked. For example yes, no, cancel, ok, etc).

Syntax :

```
MsgBox(Prompt As Object [, Buttons As MsgBoxStyle = MsgBoxStyle.OKOnly,
Title As Object = Nothing ]) As MsgBoxResult
```

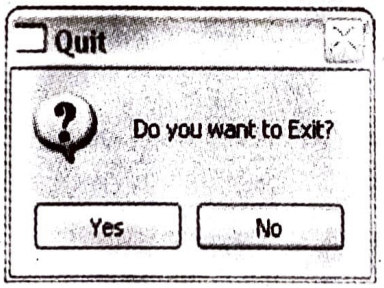
And here are the arguments (parameters) for this function :

- **Prompt :** A string expression displayed as the message in the dialog box. The maximum length is about 1,024 characters You can separate the lines using a carriage return character **Chr(13)**, a line feed character **Chr(10)**.
- **Buttons :** (Optional). The sum of values specifying
  - → the number and type of buttons to display, for example OKOnly, OKCancel, YesNoCancel, YesNo, AbortRetryIgnore
  - → the icon style to use, for example Information, Exclamation, Question, Critical
  - → the identity of the default button, for example DefaultButton1, DefaultButton2, DefaultButton3
  - → and the modality of the message box.

  If you omit Buttons, the default value is zero, then it will display OK button.
- **Title :** (Optional). String expression displayed in the title bar of the dialog box. Note that if you omit Title, the application name is placed in the title bar.
- **Return Value :** This function returns a value from the **MsgBoxResult** enumeration. Here are the possible **MsgBoxResult** values, indicating which button in the message box the user clicked: for example either OK, Cancel, Abort, Retry, Ignore, Yes, No

Ex.

```
Dim Result As Integer
Result = MsgBox("Do you want to Exit?", MsgBoxStyle.YesNo +
MsgBoxStyle.Question + MsgBoxStyle.SystemModal + MsgBoxStyle.
DefaultButton2, "Quit")
If (Result = MsgBoxResult.Yes) Then
    Me.Close()
End If
```



In above example we have provided three arguments explain as follows.

- First argument is the "**Prompt**", that is the string display in the MessageBox.
- Second argument is the "**Buttons**", that is the types of buttons (Yes & No), Icons (Question) to be display in the message box. Along with the value of SystemModal. And the default button is button2 (So the focus is on the botton2, that is "No" in our example).
- Third argument is the "**Title**", we have written "Quit" it is shown in the title bar of the message box.

If user clicks on yes button then the form will be close.

- **MessageBox.Show Method :**

Show method of MessageBox class is used to display a message to the user & get user response (waits for the user to click a button), and then returns an integer value indicating which button of the message box was user clicked. For example yes, no, cancel, ok, etc). This method has many overloaded forms. **MessageBox** class is built into the .NET Framework. The title, message, buttons, and icons displayed in the message box are determined by parameters that you pass to this method.

```
MessageBox.Show(Text As String, Caption As String, Buttons As
MessageBoxButtons,Icon As MessageBoxIcon, DefaultButton As
MessageBoxDefaultButton) As DialogResult
```

Here are the arguments you pass to this method :

(1) Text : It is the text to be display in the message box.

(2) Caption : It is the text to be display in the title bar of the message box.

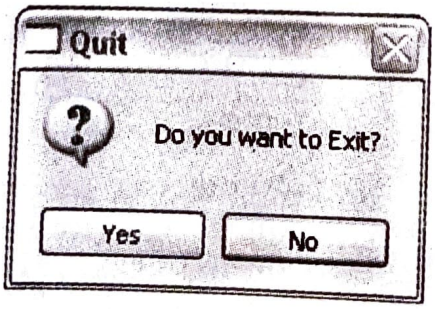
(3) Buttons : One of the **MessageBoxButtons** enumeration values that specifies which buttons to display in the message box. For example AbortRetryIgnore, YesNoCancel, OKCancel, OK

(4) Icon : One of the **MessageBoxIcon** enumeration values that specifies which icon to display in the message box. For example Asterisk, Error, Exclamation, Warning, Information, Question, None

(5) DefaultButton : One of the **MessageBoxDefaultButton** enumeration values that specifies which is the default button for the message box. For example Button1, Button2, Button3

Ex.

```
Dim Result As Integer
Result = MessageBox.Show("Do you want to Exit?", "Quit",MessageBox
Buttons.YesNo,MessageBoxIcon. Question,MessageBoxDefaultButton.
Button2)
    If (Result = DialogResult.Yes) Then
        Me.Close( )
    End If
```



- The result of the **Show** method is a value from the **DialogResult** enumeration, showing what button the user clicked. For example it can be Abort, Cancel, Yes, No, Ignore, Retry

In above example we have provided five arguments explain as follows.

- First argument is the "**Text**", that is the string display in the MessageBox.
- Second argument is the "**Caption**", we have written "Quit" it is shown in the title bar of the message box.
- Third argument is the "**Buttons**", that is the types of buttons (Yes & No) will be display in the message box.
- Fourth argument is the "**Icon**", Question mark will be display in the message box.
- Fifth argument is the "**DefaultButton**". The default button is button2 (So the focus is on the botton2 that is "No" in our example).

If user clicks on yes button then the form will be close.

[4] Dialog Box : Dialog Box are used to

[1] Display information to the user and,

[2] Retrieving information from the user.

Examples of dialog box are Font Dialog Box, File Open Dialog Box, File Save Dialog Box, Find & Replace Dialog Box, Print Dialog Box, etc.

The Built-in Dialog Boxes OR Common Dialog Boxes : There are a number of built-in dialog boxes in Visual Basic, so programmer need not to spend time to develop it. Following are the Built-in Dialog Boxes or Common Dialog Boxes.

- ✓ Folder Browser Dialog
- ✓ Open File Dialog
- ✓ Save File Dialog
- ✓ Font Dialog
- ✓ Color Dialog
- ✓ Print Dialog
- ✓ Print Preview Dialog
- ✓ Page Setup dialog

You use the **ShowDialog** method to display the dialog at run time and can check its return value (such as **Windows.Forms.DialogResult.OK** or **Windows.Forms.DialogResult.Cancel, ...Abort, Ignore, Yes, etc**) to see which button the user has clicked.

Folder Browser Dialog : The FolderBrowserDialog component is a pre define modal dialog box that is used for browsing and selecting folders. Folder Browser dialogs is supported with the **FolderBrowserDialog** class. With this dialog you can prompt the user to browse & select an existing folder or create a new one & select it. You can use this class when you only want to allow the user to select folders, not files. The **FolderBrowserDialog** component is displayed at run time using the **ShowDialog** method. Once the dialog box has been shown, you can use the **SelectedPath** property to get the path of the folder that was selected.

- **Property :**

(1) Description : Gets or sets the descriptive text displayed above the tree view control in the dialog box. The default is an empty string ("").The **Description** property can be used to specify additional information to the user, like instructions.

(2) RootFolder : Gets or sets the root folder where the browsing starts from. Set the **RootFolder** property that will appear within the tree view of the dialog box.

Ex.
```
FolderBrowserDialog1.RootFolder= Environment.SpecialFolder.MyDocuments
```

(3) SelectedPath : Obtains the path of the folder that was selected by the user.

(4) ShowNewFolderButton : Gets or sets a value indicating whether the **New Folder** button appears in the folder browser dialog box. If **ShowNewFolderButton** property is set to True (by default), the dialog box shows a "Make New Folder" button.

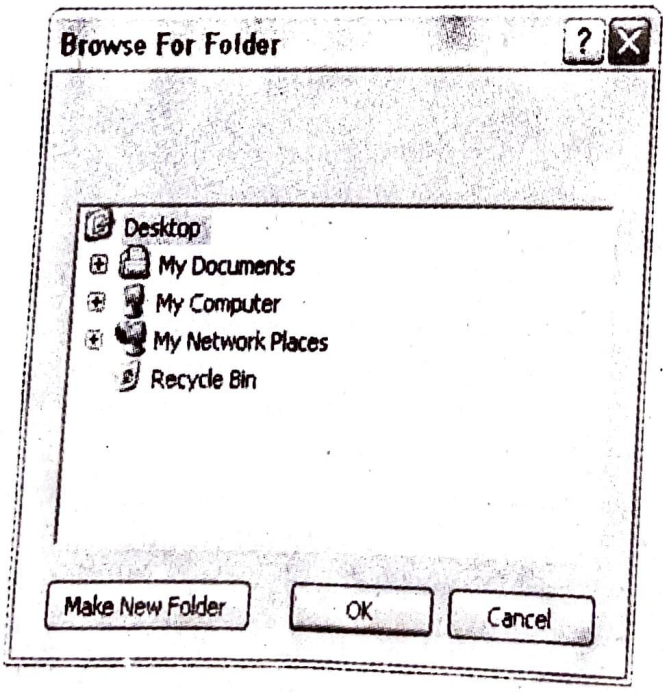
- **Method :**

(1) ShowDialog : It is used to display the dialog box at run time.

- **Event :**

(1) HelpRequest : It occurs when the user clicks the **Help** button on the dialog box.

Ex.
```
If FolderBrowserDialog1.ShowDialog = Windows.Forms.DialogResult.OK Then
    Dim Str As String = FolderBrowserDialog1.SelectedPath
End If
```



➢ **Open File Dialog :** The OpenFileDialog component is a pre define modal dialog box, which allows the user select a file to open. In is similar to the "Open File" dialog box provided by the Windows operating system. Open File dialogs are supported with the **OpenFileDialog** class.

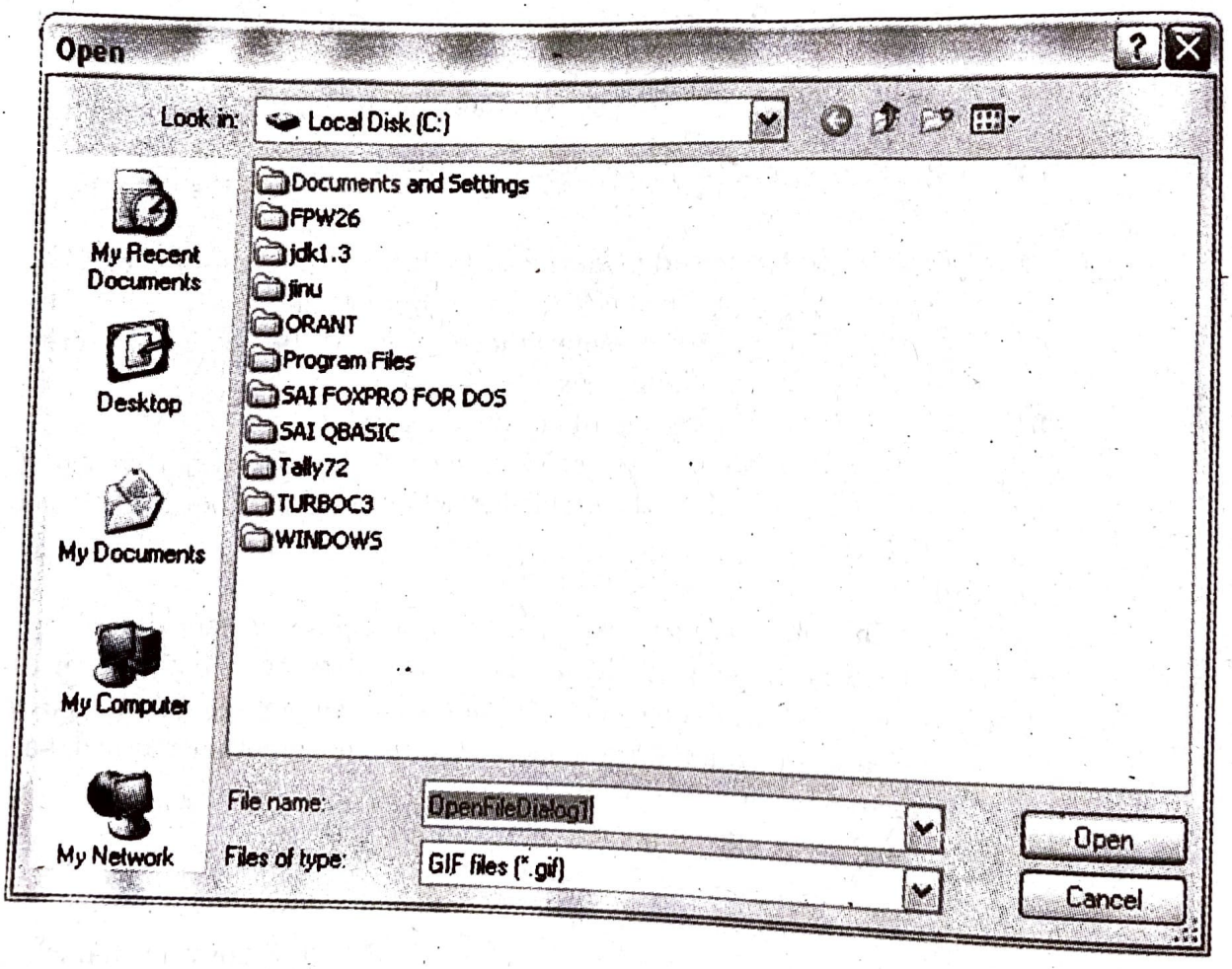
- **Property :**
  - (1) **FileName :** It is used to get the file name & path, that has been selected in the dialog box.
  - (2) **FileNames :** It is used to get the file names & paths, of all the files, that has been selected in the dialog box. It returns an array of string.
  - (3) **Filter :** Gets or sets the file name filter string (which type of the file) for example *.BMP, *.JPG, *.DOC, etc to be display in the dialog box. It sets the choices that appear in the "Save as file type" or "Files of type" box.
  - (4) **FilterIndex :** Gets or sets the index of the filter selected in the file dialog box.
  - (5) **Multiselect :** Gets or sets whether the dialog box allows multiple file selections.
  - (6) **ShowReadOnly :** Gets or sets whether the dialog displays a read-only check box.
  - (7) **ReadOnlyChecked :** Gets or sets whether the read-only checkbox is checked. (True to check & False to uncheck)
  - (8) **ShowHelp :** Gets or sets whether the Help button should be displayed in the dialog box.
  - (9) **Title :** Gets or sets the file dialog box title.
  - (10) **InitialDirectory :** It is used to specify the directory that will be appear by default in the dialog box when the dialog box is displayed.

- **Method :**
  - (1) **ShowDialog :** It is used to display the dialog box at run time.
  - (2) **OpenFile :** The **OpenFile** method is used to open a file from the dialog box. The file is opened in read-only mode for security purposes. To open a file in read/write mode, you must use another method, such as **FileStream**.
  - (3) **Reset :** Resets all options to their default values.

- **Event :**
  - (1) **HelpRequest :** It occurs when the user clicks the **Help** button on the dialog box.
  - (2) **FileOk :** Occurs when the user clicks on the **Open** or **Save** button on a file dialog box.

Example :

```
Private Sub CmdOpen_Click(ByVal sender As System.Object, ByVal
e As System.EventArgs) Handles CmdOpen.Click
    OpenFileDialog1.InitialDirectory = "C:\"
    OpenFileDialog1.Filter = "JPEG files (*.jpg)|*.jpg|GIF files
    (*.gif)|*.gif|All files (*.*)|*.*"
    OpenFileDialog1.FilterIndex = 2        ' For GIF (*.gif)
    If OpenFileDialog1.ShowDialog( )<>Windows.Forms.
    DialogResult.Cancel Then
        PictureBox1.Image = Image.FromFile (OpenFile
        Dialog1.FileName)
        End If
    End Sub
```



➤ **Save File Dialog :**

The SaveFileDialog component is a pre define modal dialog box that is used to specify filename & the location, to save file. It is the same as the standard **Save File** dialog box used by Windows operating system. Save File dialogs are supported by the **SaveFileDialog** class.

You can use the **ShowDialog** method to display the dialog box at run time. You can use the **FileName** property to get the file the user selected.

- **Property :**

(1) **CheckFileExists** (2) **CheckPathExists**
(3) **AddExtension** (4) **DefaultExt** (5) **FileName**
(6) **FileNames** (7) **Filter** (8) **FilterIndex**
(9) **ShowHelp** (10) **Title**
(11) **InitialDirectory**
(12) **CreatePrompt :** Gets or sets a value indicating whether the dialog box prompts the user for permission to create a file if the user specifies a file that does not exist.
For Ex. Specified file name does not exist. Do you want to create it? (Yes or No).
(13) **OverwritePrompt :** Gets or sets a value indicating whether the **Save As** dialog box displays a warning if the user specifies a file name that already exists.
For Ex. Specified file already exists. Do you want to replace it? (Yes or No).

- **Method :**

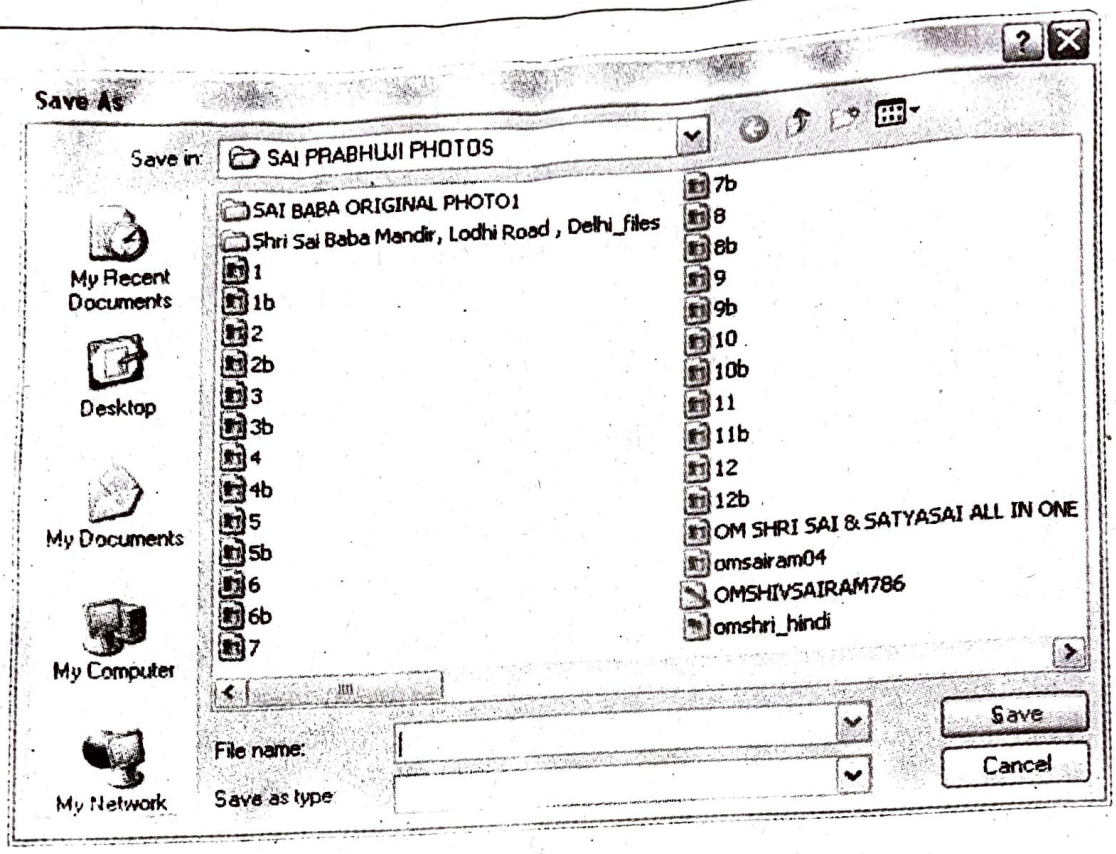
(1) **ShowDialog :** It is used to display the dialog box at run time.
(2) **OpenFile** (3) **Reset**

- **Event :**

(1) **HelpRequest :** It occurs when the user clicks the **Help** button on the dialog box.
(2) **FileOk :** Occurs when the user clicks on the **Open** or **Save** button on a file dialog box.

Ex.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    If SaveFileDialog1.ShowDialog <> Windows.Forms. Dialog Result. Cancel
    Then
        MsgBox("You chose " & SaveFileDialog1.FileName)
    End If
End Sub
```



➢ **Font Dialog :**

The FontDialog component is a pre define modal dialog box that allows user to select font, font size, font style, effects, colour, etc. By default, the dialog box shows list boxes for Font, Font style, and Size; check boxes for effects like Strikeout and Underline; a drop-down list for Script; and a sample of how the font will appear.

- **Property :**
  - (1) **Color :** Gets or sets the selected font color. This property is used to specify the font color to be used.
  - (2) **Font :** Gets or sets the selected font. This property is used to specify the font to be used.
  - (3) **FontMustExist :** Gets or sets a value indicating whether the dialog box specifies an error condition if the user attempts to select a font or style that does not exist.
  - (4) **MaxSize :** Gets or sets the maximum point size a user can select.
  - (5) **MinSize :** Gets or sets the minimum point size a user can select.
  - (6) **ShowApply :** Gets or sets a value indicating whether the dialog box contains an **Apply** button. The default value is **false**.
  - (7) **ShowColor :** The value of this property indicates whether the **Color** choice will be display on the font dialog box. The default value is **false**.
  - (8) **ShowEffects :** Gets or sets a value indicating whether the dialog box contains controls that allow the user to specify strikethrough, underline, and text color options. The default value is **true**.

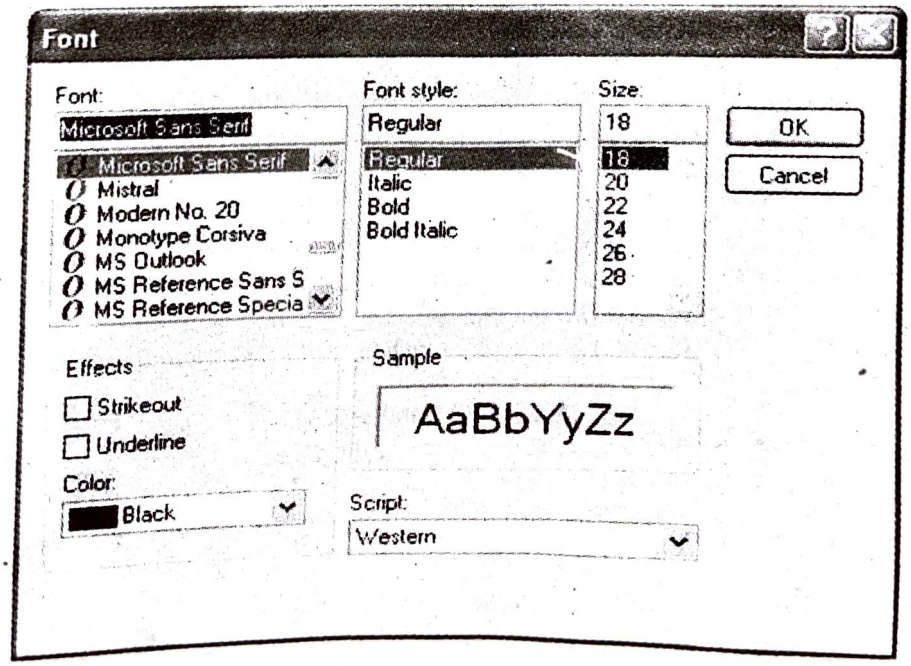
(9) ShowHelp : Gets or sets whether the Help button should be displayed in the dialog box.

- **Method :**
  (1) ShowDialog : It is used to display the dialog box at run time. It runs a common dialog box.
  (2) Reset ; Resets all options to their default values.

- **Event :**
  (1) HelpRequest : It occurs when the user clicks the **Help** button on the dialog box.
  (2) Apply : Occurs when the user clicks the Apply button.

Ex.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    FontDialog1.ShowColor = True
    ' Set a minimum and maximum size to be shown in the FontDialog.
    FontDialog1.MinSize = 18
    FontDialog1.MaxSize = 32
    If FontDialog1.ShowDialog <> Windows.Forms. DialogResult. Cancel
    Then
        RichTextBox1.SelectionFont = FontDialog1.Font
        RichTextBox1.SelectionColor = FontDialog1.Color
```



➢ **Color Dialog :** The ColorDialog component is a pre-define dialog box. The ColorDialog box is used to select a color from a palette and to add custom colors to that palette. The color selected in the dialog box is returned in the **Color** property. If the **AllowFullOpen** property is set to **false**, the "Define Custom Colors" button is disabled and the user is restricted to the predefined colors in the palette. To display the dialog box, you must call its **ShowDialog** method.

- **Property :**
  (1) AllowFullOpen : If the **AllowFullOpen** property is set to **false**, the "Define Custom Colors" button is **disabled**. The default is **true**. It enables or disables the "Define Custom Colors" button.

  (2) Color : Gets or sets the color selected by the user.

  (3) FullOpen : Gets or sets whether the controls used to create custom colors are visible when the dialog box is opened. The default value is **false**.

  (4) ShowHelp : Gets or sets whether the Help button should be displayed in the dialog box.

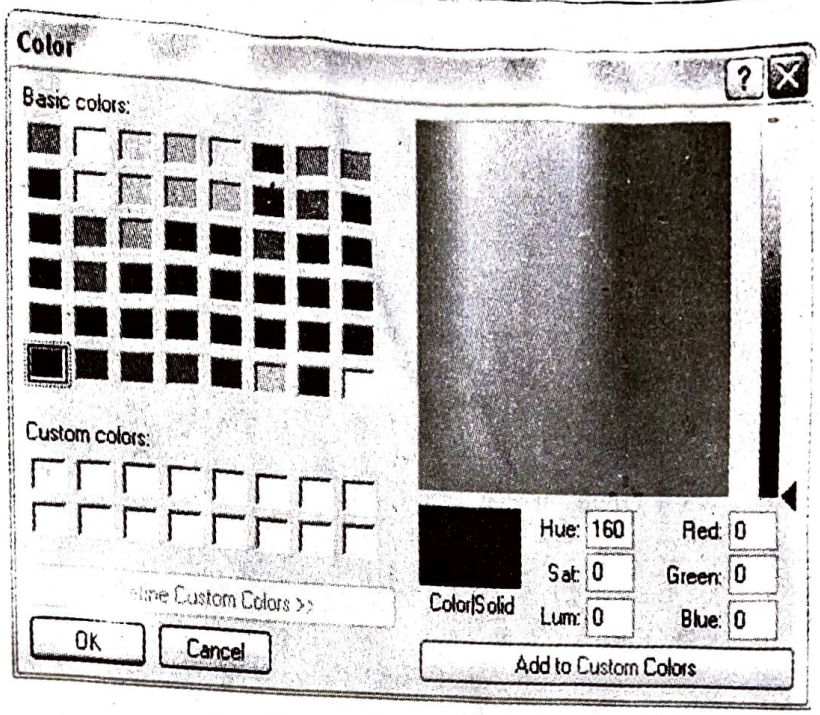
- **Method :**
  (1) ShowDialog : It is used to display the dialog box at run time.

  (2) Reset : Resets all options to their default values.

- **Event :**
  (1) HelpRequest : It occurs when the user clicks the **Help** button on the dialog box.

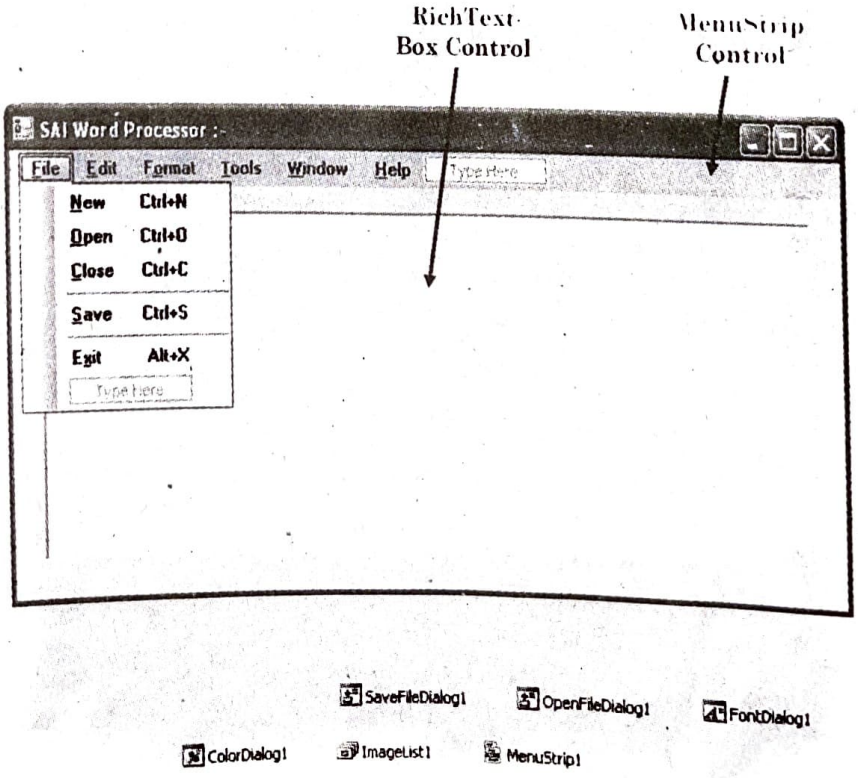
Ex.

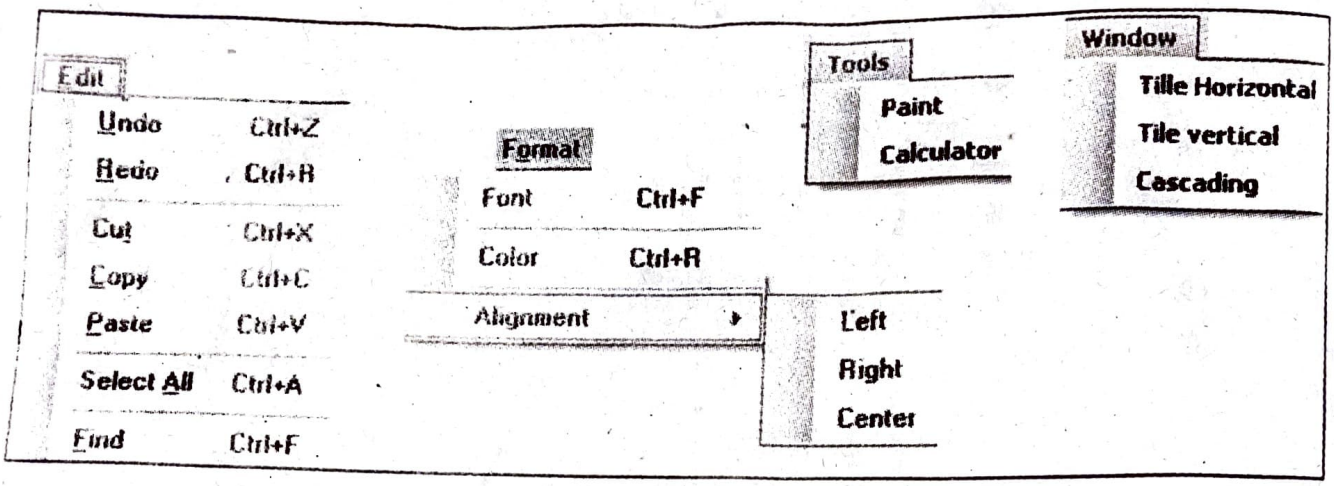
```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
ColorDialog1.FullOpen = True
    If ColorDialog1.ShowDialog <> Windows.Forms.DialogResult. Cancel
    Then
        Label1.Text = "OMSHIVSAI"
        Label1.BackColor = ColorDialog1.Color
    End If
End Sub
```



Practical 10 : Design simple WordPad application using Rich textbox and Menu.

Design interface as given in figure. Place MenuStrip control on to the form. Design menu as shown below. In the **MdiWindowListItem** property of MenuStrip1 set the name of ToolStripMenuItem (for example **WindowsToolStripMenuItem**), so it will display list of all the open MDI child forms on that ToolStripMenuItem.






```
'Coding Of File à New
    RichTextBox1.SetBounds(0, 24, 1020, 680)
    Me.Controls.Add(RichTextBox1)

'Coding Of File → Open
    OpenFileDialog1.InitialDirectory = "D:\"
    OpenFileDialog1.Filter = "Text Files(*.TXT)|*.TXT|Any Files(*.*)|*.*"
    OpenFileDialog1.FilterIndex = 1
    OpenFileDialog1.ShowDialog()
    RichTextBox1.LoadFile(OpenFileDialog1.FileName)

'Coding Of File → Close
    Me.Controls.Remove(RichTextBox1)
    RichTextBox1.Clear()

'Coding Of File → Save
    SaveFileDialog1.ShowDialog()
    RichTextBox1.SaveFile(SaveFileDialog1.FileName)
'Coding Of Edit → Undo
    RichTextBox1.Undo()

'Coding Of Edit → Redo
    RichTextBox1.Redo()

'Coding Of Edit → Cut
    If RichTextBox1.SelectedText <> "" Then
        RichTextBox1.Cut()
    End If
```

```
'Coding Of Edit → Copy
    If RichTextBox1.SelectionLength > 0 Then
        RichTextBox1.Copy()
    End If

'Coding Of Edit → Paste
    RichTextBox1.Paste()

'Coding Of Edit → Select All
    RichTextBox1.SelectAll()

'Coding Of Edit à Find
    Dim S As String = InputBox("Please Enter The String To Find:")
    Dim Pos As Integer = RichTextBox1.Find(S)
    MsgBox("Searched String has been found at Position " & Pos)
    RichTextBox1.SelectionLength = S.Length

'Coding Of Format → Font
    FontDialog1.ShowDialog()
    RichTextBox1.SelectionFont = FontDialog1.Font

'Coding Of Format → Colour
    ColorDialog1.ShowDialog()
    RichTextBox1.SelectionColor = ColorDialog1.Color

'Coding Of Format → Alignment à Center
    RichTextBox1.SelectionAlignment = HorizontalAlignment.Center

'Coding Of Tools → Paint
    Shell("C:\WINDOWS\system32\MSPaint.EXE",AppWinStyle. Maximized Focus)

'Coding Of Tools → Calculator
    Shell("C:\WINDOWS\system32\calc.exe", AppWinStyle.Maximized Focus)

'Coding Of Windows → Tile Horizontal
    Me.LayoutMdi(MdiLayout.TileHorizontal)

'Coding Of Windows → Tile Vertical
    Me.LayoutMdi(MdiLayout.TileVertical)

'Coding Of Windows → Cascase
    Me.LayoutMdi(MdiLayout.Cascade)
```

2.7 EXCEPTION HANDLING : (TRAPPING ERRORS)

There are two ways of handling errors that occur at run time in VB 2005

[A]With Unstructured exception handling : Unstructured exception handling uses the **On Error** statement.

[B]With Structured exception handling : Structured exception handling uses the **Try...Catch...Finally** statement.

Without an **On Error GoTo** or **Try...Catch...Finally** statement, any exception that occurs is fatal and your program will stop.

➢ **Structured exception handling : (Trapping Error Using Structure error handling) :**

Structured exception handling uses the **Try...Catch...Finally** statement :

```
Try
    ' Place statements that may produce a runtime error(that may throw 'an
    exception) in this block.
    [Catch <exception1> As <type1> [When expression1] ]
        ' Place statements that take some action if an exception is thrown.
        [Exit Try] ]
    [Additional Catch blocks]
Finally
    ' This code always runs whether an error occur or not
End Try
```

- Structured exception handling is based on a particular statement, the **Try...Catch...Finally** statement, which is divided into three blocks.
- The **Try** block contains the section of code to be monitored for exceptions. (Contains code that may throw an exception). Place statements that may produce a runtime error.
- The **Catch** block contains code to handle the exceptions that occur. If an error occurs during execution of **Try** clock, Visual Basic examines each **Catch** statement within the **Try...Catch...Finally** until it finds one with a condition that matches that error.
- The code in the **Finally** block always executes last, regardless of whether the code in the **Catch** blocks has executed. Place cleanup code, such as that for closing files and releasing objects, in the **Finally** section.
- If a **Try** statement does not contain any **Catch** blocks, it must contain a **Finally** block. You can have any number of **Catch** statements (however, you must have at least one **Catch** statement or a **Finally** statement).

Ex.
```
Dim x As Double = 5   ' Declare variables.
Dim y As Integer = 0
Try   ' Set up structured error handling.
    x /= y   ' Cause a "Divide by Zero" error.
Catch exp As Exception When y = 0   ' Catch the error.
    MsgBox(ex.ToString)   ' Show friendly error message.
Finally
    Beep()   ' This line is executed no matter what.
End Try
```

- **ToString Method :**
  We can use exception object's **ToString** method to display information about the current exception. .
- **e.message Property :** Gets the message that describes current exception.

➢ **Error Filtering in the Catch Block :**

There are actually two ways to filter exceptions with **Catch** blocks.

[a] You can filter on specific classes of exceptions &

[b] Second, You can filter using the **When** keyword.

[a] You can filter on specific classes of exceptions :

Exceptions are based on the Visual Basic **Exception** class. For example when you divide two numbers and an overflow exception occurs.

Ex.
```
Dim Top As Double = 5
Dim Bottom As Double = 0
Dim Result As Integer
Try
    Result = CType(Top / Bottom, Integer)
Catch Exp As System.OverflowException
    MsgBox("Attempt to divide by zero resulted in overflow")
End Try
```

[b] You can filter using the When keyword :

The second exception-filtering option lets you use the **Catch** statement to filter on any conditional expression, using the **When** keyword. This option is often used to filter by exception number, which you can check with the **Err** object's **Number** property. Here's an example that filters overflow exceptions by number (which is exception number 6 in Visual Basic .NET):

Ex. 1 :
```
Dim A = 1, B = 0, Z As Integer
Try
    Z = A / B
    MsgBox("The answer is :" & Z)
Catch When Err.Number = 6
    MsgBox("Exception: Arithmetic overflow")
End Try
```

Ex. 2 : If you want to add a general exception handler to catch any exceptions not filtered, you can add a **Catch** block for the **Exception** class at the end of the other **Catch** blocks :

```
Dim A = 1, B = 0, Z As Integer
Try
    Z = A / B
    MsgBox("The answer is :" & Z)
Catch exp As System.OverflowException
    MsgBox("Exception: Arithmetic overflow")
Catch exp1 As System.ArgumentException
    MsgBox("Exception: Invalid argument value")
Catch exp2 As Exception
    MsgBox("Exception occurred")
End Try
```

➢ **Err Object :**

- **Err object** Contains information about run-time errors.
- The **Number property** of the Err object contains the number of the most recent run time error, and the **Description property** contains a text message that describes the error in detail.
- You can clear the Err object by using the **Err.Clear method.**
- To generate a run-time error in your code, use the **Raise method.**

2.8 USING MODULE & PROCEDURE

➢ **What is a procedure?**

A procedure is a separate section of code which performs one or more specific tasks & is identified by having a name. **OR**

A named sequence of statements executed as a unit.

➢ **Advantages of procedures :**

- Procedures allow you to break your program into small well-defined tasks, which you can debug more easily than a program without procedures. (Easy to understand, Easy to Debug).
- Avoid repeated task: - procedures are useful to avoid repeated task, ex. (frequently used calculation). Code can be reuse.
- You can use procedures in other program.
- Easier to maintain. (Change can be made once in a procedure).

➢ **Types of procedures :**

Visual Basic uses several types of procedures :

[1] Sub-procedures.

[2] Function procedures.

[3] Event procedures. etc.

[1] Sub Procedures : value not return

A Sub Procedures or general procedure is a block of code that is executed when it is called through code. It performs actions but do not return a value to the calling code.

A **Sub** procedure is a series of Visual Basic statements enclosed by the **Sub** and **End Sub** statements. Each time the procedure is called, its statements are executed, starting with the first executable statement after the **Sub** statement and then returns control to the calling code, (when the **End Sub**, **Exit Sub**, or **Return** statement encountered), but it does not return a value to the calling code.

You can define a **Sub** procedure in modules, classes, and structures. It is **Public** by default.

The syntax for declaring a Sub procedure is as follows :

```
[modifiers] Sub subprocedurename[(parameterlist)]
    ' Statements of the Sub procedure.
End Sub
```

Calling Syntax : The syntax for a call to a Sub procedure is as follows :

```
[Call] subprocedurename [(argumentlist)]
```

Ex.

```
Sub Add(x as integer, y as integer)
    A=Val(x) + Val(y)
    Msgbox(A)
End Sub
```

You can call above procedure is as follows

```
Call Add(776, 10)
```

OR

```
Add(776, 10)
```

[2] Function Procedures : It return a value

Function procedure is similar to sub procedure but function procedure can return a value.It returns a value by assigning it to the procedure name itself.

A **Function** procedure is a series of Visual Basic statements enclosed by the **Function** and **End Function** statements. The **Function** procedure performs a task and then returns control to the calling code (when the first **End Function**, **Exit Function**, or **Return** statement encountered). When it returns control, it also returns a value to the calling code.

You can define a **Function** procedure in a module, class, or structure. It is **Public** by default.

The syntax for declaring a Function procedure is as follows :

```
[modifiers] Function functionname[(parameterlist)] As returntype
' Statements of the Function procedure.
functionname = expression
        OR
Return expression
End Function
```

Calling Syntax : The syntax for a call to a **Sub** procedure is as follows :

```
lvalue = functionname[(argumentlist)]
```

Ex.

```
Function Add1(x as integer, y as integer) As integer
    Ans= Val(x) + Val(y)
    Add1 = Ans
End Function
```

You can call above procedure is as follows

```
MsgBox(Add1(776, 10))
```

➢ **Passing Arguments :**

When variable is passed to a procedure, it is called an argument. There are two ways of passing arguments to procedures.

[1] Passing argument by value (by default)

[2] Passing argument by reference

[1] Passing Arguments By Value. (ByVal) :

When argument is passed by value, then two memory locations are involved. Means copy of variable is passed, if the procedure changes the value, the change affects only the copy & original value of variable remain as it is. By default argument are passed by value (ByVal) in VB.

Ex.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button1.Click
    Dim S As Integer
    S = 4
    ListBox1.Items.Add(S)
    Twice(S)
    ListBox1.Items.Add(S)
End Sub
```

```
Sub Twice(ByVal Z As Integer)
    Z = Z * 2
    ListBox1.Items.Add(Z)
End Sub
```

Output is as follows in the ListBox1 :

4
8
4

[2] Passing Argument By reference :

When argument is passed by reference only one memory location is involved. Means address of variable is passed to a procedure, As a result procedure can access to the original value of variable.

Ex.

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e
As System.EventArgs) Handles Button1.Click
    Dim S As Integer
    S = 4
    ListBox1.Items.Add(S)
    Twice(S)
    ListBox1.Items.Add(S)
End Sub
-----------------------------------------------------------------
Sub Twice(ByRef Z As Integer)
    Z = Z * 2
    ListBox1.Items.Add(Z)
End Sub
```

Output is as follows in the ListBox1 :

4
8
8

- **Scope :**

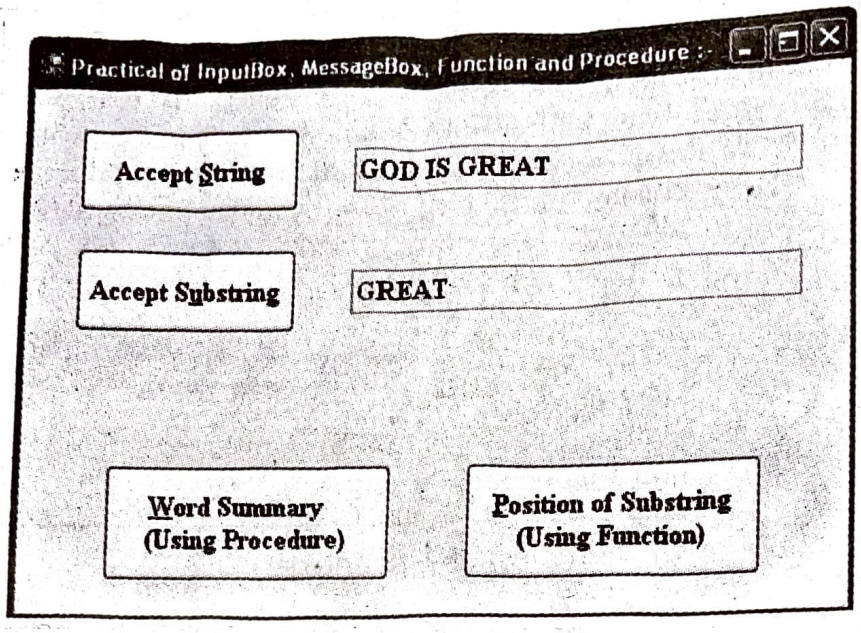
Scope means which part of the application that can see & manipulate the element. The level of the scope can be at Block level scope (narrowest scope) , Procedure level scope, Module level scope, Namespace level scope (widest scope)

Practical 11 : Practical of InputBox, MessageBox, Function & Procedure.

Design interface as shown in figure. Place four buttons. Two buttons is used to accept String & Substring. Accept String & Substring using InputBox. Then display String & Substring in TextBox1 & TextBox2 respectively. Set **ReadOnly** property of TextBox1 & TextBox2 to True.

When word summary button is click, you have to count number of vowels, number of blank spaces using Procedure.

When Position of Substring button is click, you have to find out position of substring in a given string, using Function.




```
' Declare two variables in declaration section.
    Dim Str, SubStr As String
' Coding of Accept string Button
    Str = InputBox("Please Enter String ")
    TextBox1.text = Str

' Coding of Accept Substring Button.
    SubStr = InputBox("Please Enter String ")
    TextBox2.text = SubStr

' Create a Sub Procedure to count blank space, vowels etc.
    Sub WordCountSummary(ByVal Str As String)
    Dim Vowels, BlanbSpc As Integer
        ' Find the number of vowels in string
        For I As Integer = 1 To Str.Length
        If Mid(Str.ToLower, I, 1) = "a" Or Mid(Str.ToLower, I, 1) = "e" Or
            Mid(Str.ToLower, I, 1) = "i" Or Mid(Str.ToLower, I, 1) = "o" Or
            Mid(Str.ToLower, I, 1) = "u" Then
            Vowels = Vowels + 1
        End If
        ' Find the number of blank spaces in the string
        If Mid(Str.ToLower, I, 1) = " " Then
            BlanbSpc = BlanbSpc + 1
        End If
        Next
        MsgBox("Length of the string is :-" & Str.Length & Chr(13) & "Number of
        vowels:-" & Vowels & Chr(13) & "Number of blank spaces:-" & BlanbSpc)
End Sub
```

```
' Coding of Word Summary Button.
' Lets Call Sub Procedure & Pass Accepted string as argument.
    WordCountSummary(Str)

' Create a Function for find out position of substring.
    Function Position(ByVal Str As String, ByVal SubStr As String) As Integer
        Position = InStr(Str, SubStr)
    End Function

' Coding of Position of SubString Button.
' Lets Call Function Procedure & Pass Accepted string, SubString ' as
arguments
    Dim P As Integer
    P = Position(Str, SubStr)
    MsgBox("Position of SubString is at the location " & P)
```

2.9 USING ARRAYS & COLLECTIONS

➢ **Array :** An array is a group of common element accessed under a common name. Each element is referenced with a no, called an index or subscript. In VB index start with zero.

Characteristics of Array :

[1] Array element (index) start with zero (0)
[2] An object array may have different data type in each element.
[3] Array may be fixed size or dynamically allocated at runtime.

❖ **Fixed size Array :**

Dim statement is used to declare an array.

```
Dim Stud(5) As Integer
'Stud array have been declared to hold 6 elements (0 to 5).
MsgBox(LBound(Stud))   0 is the lower bound of this array
 MsgBox(UBound(Stud)) 5 is the upper bound of this array
```

In .NET Lower bound of **Array** is always zero. So Option Base 1 statement is no longer exists.

❖ **Array Dimensions : Multidimensional Array :-**

The array Stud in the preceding example uses one index and is said to be one-dimensional. An array that uses more than one index or subscript is called multidimensional.

The following example declares a variable to hold a two-dimensional array with elements of the Integer Data Type.

Ex. To store 10 student's Roll no with their Marks.

```
Dim Marks(9,9) As Integer
```

Ex. Of Three Dimension Array

```
Dim matrix(3, 0 To 10, 0 To 14)
    0 to 3 = 4
    0 to 10 = 11
    0 to 14 = 15
4 * 11 * 15 = 660 Elements.
```

❖ **Dynamic Array :**

If you do not know the size of array during the declaration of the array, then you can use dynamic Array. You can declare the array as dynamic with empty parentheses (without dimension subscripts). You can change the size of the array at run time (dynamically). It is called dynamic array. Dynamic array can be resized at run time.

Ex. Dim MyArray( ) As String

Then you can allocate actual no of element with a **ReDim** statement

```
ReDim MyArray (x + 1)
```

The **ReDim** statement is used to size or resize a dynamic array that has already been formally declared using a **Private**, **Public**, or **Dim** statement with empty parentheses (without dimension subscripts). You can use the **ReDim** statement repeatedly to change the number of elements and dimensions in an array. **Note:** ReDim statement can not change the dimension of the array. When you use ReDim statement VB will completely erase the array & re-dimension it to the size you specify. Sometimes you may want to change the size of the array without losing the data in array, then you can use **Preserve** Keyword with Redim statement.

❖ **Example of ReDim & Preserve**

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
    Dim x( ) As Integer   'Dynamic Array
    ReDim x(5)            '0 to 5
    For I As Integer = 1 To 5
        x(I) = I
    Next I
    ReDim Preserve x(10)
    For I As Integer = 6 To 10
        x(I) = I
    Next I
    For I As Integer = 1 To 10
        ListBox1.Items.Add(x(I))
    Next I
End Sub
```

You can also initialize the data in array **if you don't give an array an explicit size**. In following dynamic array example, we are initializing an array with the values "OM", "DATTAR", & "MALIK".

```
Dim MyArray() = {"OM", "DATTAR", "MALIK"}
```

In above example we have not give the size of the array, so it is possible to initializing an array.

❖ **Array related Functions :**

- **IsArray Function :**
  Returns a Boolean value indicating whether a variable is an array.
  For Ex.
  ```
  Dim MyArray(4) As Integer
  MsgBox(IsArray(MyArray))  'Returns True
  ```
- **LBound Function :**
  The lower bound for any dimension is always 0.
  For Ex.
  ```
  Dim matrix(3, 0 To 9, 0 To 14)
  MsgBox(LBound(matrix, 1))        'Returns 0
  ```
- **UBound Function :**
  Returns the upper bound. In UBound function second argument is optional, which is the dimension. Use 1 for first dimension, 2 for second, & so on. If it is omitted, 1 is assumed.
  For Ex.
  ```
  Dim matrix(3, 0 To 9, 0 To 14)
  MsgBox(UBound(matrix, 1))   'Returns 3
  MsgBox(UBound(matrix, 2))   'Returns 9
  MsgBox(UBound(matrix, 3))   'Returns 14
  ```

➢ **Collection (Introduction) :**

In dynamic array ReDim takes up more execution time, because it actually creates a new array of the specified size which replaces the existing array. If you have specified a Preserve option. It also copies the elements from the old array to new one. So, if you ReDim the arrays in your application frequently, its performance would be hit considerably.

.NET provides you with a solution to this problem in the form of collections. They are different from arrays in one following respect. Or advantage of Collection over the Array.

- **You don't have to re-allocate memory as number of objects to hold increases or decreases.** (ArrayList collection provides you with an ideal alternative to arrays with **optimized allocation / de-allocation capabilities.**)
- Other advantages of ArrayList collection & other collection classes is, it **have in built methods.** For example ArrayList and other collection classes have in built methods like Sort, Reverse, Search, Find, Add, Remove, Clear etc.

[1] **Visual Basic collection** from **Microsoft.VisualBasic.Collection()**

[2] **.NET Framework collection classes** defined in the **System.Collections** namespace. They are not strongly typed. These collection classes are as follows.

ArrayList Class, BitArray, HashTable, Queue, SortedList, Stack

[3] **.NET Framework collection classes** defined in the **System.Collections.Generic** namespace Which allow users to create strongly typed collections that provide better type safety and performance than non-generic strongly typed collections.

.NET framework provides you with a variety of following collection classes to create generic collection. List, LinkedList, Dictionary, Queue, SortedDictionary, SortedList, Stack

❖ **Collection :**

A Visual Basic **Collection** is an ordered set of items that can be referred to as a unit. It is a lists of items grouped under a common name. Collections are more powerful than arrays, because they have pre define properties & methods attached to them. And you don't have to re-allocate memory as number of objects to hold increases or decreases.

A collection is an object used for grouping and managing related objects. Many different types of collections exist. **Predefined collections** are used in Visual Basic applications for many purposes, For example, every Form has a collection of controls. (You can access this collection through the **form's Controls property**.) You can also create **your own collections** to organize and manipulate objects.

The following example creates a new Visual Basic collection and assigns it to the variable MyVBColl

```
Dim MyVBColl As New Microsoft.VisualBasic.Collection()
```

This **Collection** object is one-based, which means that the index values of the elements range from 1 through the value of the **Count** property. Visual Basic collections hold elements of type Object.

Once you have created a collection, you can do any of the following :

- Add an element with the Add Method.
- Remove an element with the Remove Method.
- Remove all elements with the Clear Method.
- Find out how many elements the collection contains with the Count Property.

- Check whether a specific element is present with the Contains Method.
- Return a specific element from the collection with the Item Property.
- Iterate through the entire collection with the For Each...Next Statement (Visual Basic).

Visual Basic collections are not compatible with the .NET Framework collections available in the **System.Collections**, **System.Collections.Generic**, and **System.Collections.Specialized** namespaces.

- **Methods :**

(1) Add :

Adds an element to a Collection object.

Add(ByVal Item As Object, Optional ByVal Key As String, Optional By Val { Before | After } As Object = Nothing)

➔ **Parameters :**

✓ **Item :**

Required. An object of any type that specifies the element to add to the collection.

✓ **Key :**

Optional. The Key argument is optional in a Visual Basic collection. A unique **String** expression that specifies a key string that can be used as a index.

✓ **Before :**

Optional. An expression that specifies a relative position in the collection.

Ex.

```
Dim MyVBColl As New Microsoft.VisualBasic.Collection()
Dim item1, item2, item3, item4 As String
item1 = "OM"
item2 = "DATTAVI"
item3 = "VIPAL"
item4 = "PRIYAM"
MyVBColl.Add(item1, "1stkey")
MyVBColl.Add(item2, "2ndkey")
MyVBColl.Add(item3, "3rdkey")
MyVBColl.Add(item4, "4thkey")
```

(2) Remove : Removes an element from a Collection object. Note that when an element is removed from a Visual Basic **Collection**, the index values are renumbered from 1 through the value of the Count Property.

```
Remove(ByVal { Key As String | Index As Integer })
' Remove the first element of the Visual Basic collection.
    MyVBColl.Remove(1)
' Remove the element with the key "2ndkey".
    MyVBColl.Remove("2ndkey")
```

(3) **Clear :** Deletes all elements of a Visual Basic Collection object.& resets its Count property to 0.

```
MyVBColl.Clear()
```

(4) **Contains :** Returns a Boolean value indicating whether a Visual Basic Collection object contains an element with a specific key.

- **Property :**
  (1) **Count :** Returns an **Integer** containing the number of elements in a collection. (Read-only)
  (2) **Item :** Returns a specific element of a **Collection** object either by position or by key. (Read-only)
  Ex. `MsgBox(CStr(MyVBColl.Item(1)))`

➢ **ArrayList Collection :**

An **ArrayList** is a kind of collection. It is .NET Framework collections class available (define) in the **System.Collections** namespace.

ArrayList collection provides you with an ideal alternative to arrays with optimized allocation / de- allocation capabilities (You don't have to re-allocate memory as number of objects to hold increases or decreases.). Other advantages of ArrayList collection is, it **has in built methods.** For example ArrayList and other collection classes have in built methods like Sort, Reverse, Search, Find, Add, Remove, Clear etc.

- **Property :**
  (1) **Count :** Gets the number of elements actually contained in the **ArrayList.**
  (2) **Item :** Gets the element at the specified index.
  (3) **Capacity :** Gets the capacity of the ArrayList. It is the number of elements the ArrayList can hold.

- **Method :**
  (1) **Add :** Adds an object to the end of the ArrayList.
  (2) **AddRange :** Adds the elements of a Collection to the end of the ArrayList.
  (3) **Insert :** Inserts an element into the ArrayList at the specified index.
  (4) **InsertRange :** Inserts the elements of a collection into the ArrayList at the specified index.
  (5) **Remove :** Removes the first occurrence of a specific object from the ArrayList.
  (6) **RemoveRange :** Removes a range of elements from the ArrayList. (removes the elements by given range).
  (7) **RamoveAt :** Removes the element at the specified index of the ArrayList. (removes by specified position)
  (8) **IndexOf :** Returns the zero-based index of the first occurrence of a value in the ArrayList.

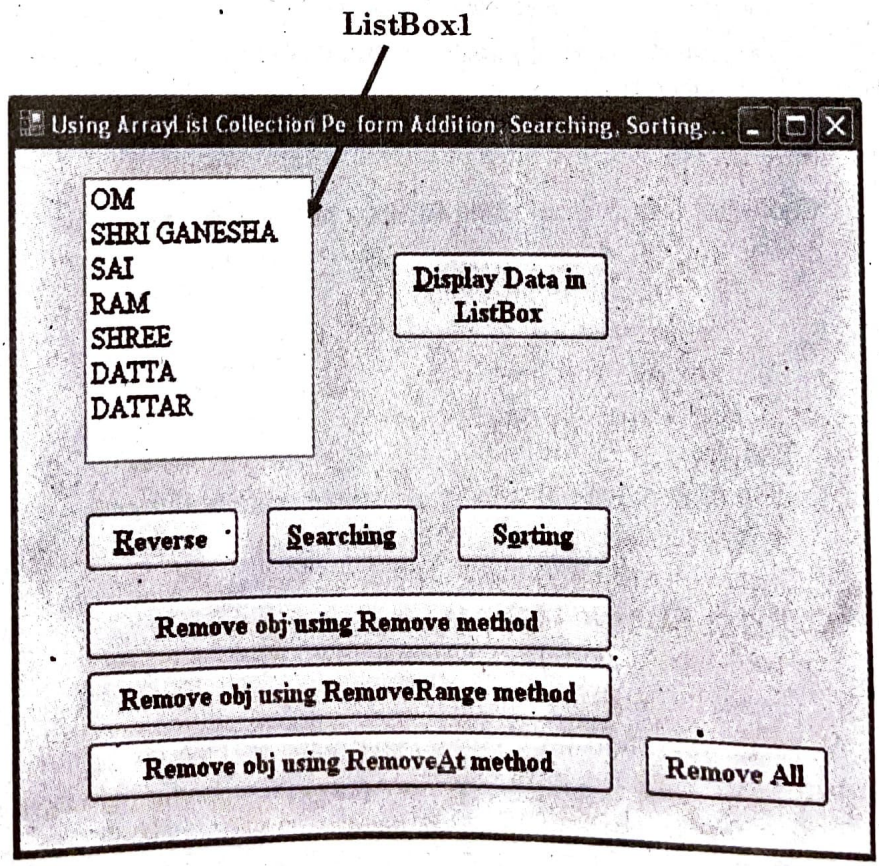
(9) **Clear** : Removes all elements from the ArrayList.
(10) **Contains** : Determines whether an element is in the ArrayList.
(11) **Reverse** : Reverses the order of the elements in the ArrayList or a portion of it.
(12) **Short** : Sorts the elements in the ArrayList or a portion of it.
(13) **ToArray** : Copies the elements of the ArrayList to a new array.

Example of ArrayList :

Practical 12 : Using ArrayList Collection of .NET framework, Perform Addition, Removing, Searching, Sorting, Reversing, etc operations.

Design the interface as shown in figure.

```
' Creates and initializes the object of ArrayList class in declaration section.
Dim MyArrayList As New ArrayList()

'Procedure to Display the data of ArrayList to the ListBox1
    Sub DisplayInListBox()
        ListBox1.Items.Clear()
        For I As Integer = 0 To MyArrayList.Count - 1
            ListBox1.Items.Add(MyArrayList(I))
        Next
    End Sub
```



```
'Adding elements to the collections in Form_Load event.
    MyArrayList.Add("OM")
    MyArrayList.Add("SHRI GANESHA")
    MyArrayList.Add("SAI")
    MyArrayList.Add("RAM")
    MyArrayList.Add("SHREE")
    MyArrayList.Add("DATTA")
    MyArrayList.Add("DATTAR")
    DisplayInListBox( )
'Coding of Reverse Button.
    MyArrayList.Reverse( )
    ListBox1.Items.Clear( )
    DisplayInListBox( )
' Coding of Sorting Button.
    MyArrayList.Sort( )
    ListBox1.Items.Clear( )
    DisplayInListBox( )
' Coding of Searching Button.
    Dim STR As String = InputBox("Please enter the Name to be Search")
    If MyArrayList.Contains(STR) = True Then
        MsgBox("Search item has been Found")
    Else
        MsgBox("Search item has not been Found")
    End If
' Coding of Remove obj using Remove method Button. (selected object from the
ListBoc1 will be remove).
    MyArrayList.Remove(ListBox1.SelectedItem)
    DisplayInListBox()
' Coding of Remove obj using RemoveRange method Button.
    Try
        MyArrayList.RemoveRange(1, 3)
        DisplayInListBox( )
    Catch ex As Exception
        MsgBox(ex.Message)
    End Try
' Coding of Remove obj using RemoveAt method Button.
    MyArrayList.RemoveAt(ListBox1.SelectedItem)
    DisplayInListBox( )
' Coding of RemoveAll Button
    MyArrayList.Clear( )
    DisplayInListBox( )
'Note : Array is in the System namespace. ArrayList is in the System.Collections
namespace. 'List is in the System.Collections.Generic namespace.
```

UNIT 3

OBJECT ORIENTED PROGRAMMING

3.1 Object-Oriented Programming
3.2 Abstraction, Encapsulation & Polymorphism
3.3 Interfaces & Inheritance

3.1 OBJECT-ORIENTED PROGRAMMING

Visual Basic comes with thousands of built-in classes. Windows forms are classes, based on the System.Windows.Forms.Form class. The controls on the Toolbox in Visual Basic represent classes. When you drag a control from the Toolbox onto a form, you are creating an object. For example controls such as text boxes are really based on the TextBox class. In fact, the Integer type is a class in Visual Basic, and variables of that type are in fact objects.

.Net framework follows the Object Oriented approach that is built upon classes and objects. .Net framework supports all the main OOP features like Abstraction, Inheritance, Encapsulation, Polymorphism. The primary advantage of Object Oriented Programming is code reusability. This means that one need not to create code from scratch; instead, existing codes can be re-used and built upon to create new codes.

A **class** can be defined as a template for creating different objects. A **class** contains the characteristics and functionalities that objects can posses. For example in Train class will posses the characteristics (properties) like engine, colour, no_of_coaches etc and functionalities (methods) like move, start, stop etc.

Each object in Visual Basic is defined by a class. A class describes the Fields (variables), Properties, Methods, and Events of an object. Classes are made of fields (variables), properties, methods, and events.

Fields (variables), Properties, Methods, and Events are called the members of a class. Inside the class, members are declared as either Public, Private, Protected, Friend, or Protected Friend. (Access modifier).

It's easy to create classes and objects in Visual Basic. To create a class, you only need to use the **Class** statement, needs to end with **End Class** :

```
Public Class ClassName
    -------------
    -------------
    -------------
End Class
```

Ex.

```
Public Class Train
    -------------
    -------------
    -------------
End Class
```

Objects are instances of classes; you can create as many objects you need once you have defined a class. You must use the **New** keyword to create a new instance of a class : Dim Car As New Vehicle()

OR

Dim Car As Vehicle = New Vehicle()

OR

If you omit the **New** keyword, you're just declaring a new object, and it's not yet created : Dim Car As Vehicle

Before using the object, you must explicitly create it with **New**: Car = New Vehicle( )

- **Fields :** Fields are like variables in that they can be read or set directly. For example, if you have an object named "Car" you could store its price in a field named "Price".

  Field is also called a **member variable** or **class's data members.**

  Ex.
```
Public Class Vehicle
    Public V_Name As String
    Public V_Price As Double
End Class
```

  You can access the field after creating an object (instance).
```
Dim Car As New Vehicle
Car.V_Name = "Maruti Zen"
Car.V_Price = 100000
```

- **Property :** Properties are retrieved and set like fields, but are implemented using property **Get** and property **Set** procedures (method). It is an easy way to guarding to the data in your object.
  - **Set** property procedure used to set (assign / initialize) the value of a property.
  - **Get** property procedure used to return (get / retrieve) the value of a property.
  - Note that you can make properties write-only with the **WriteOnly** keyword (and you must omit the **Get** method)
  - You can make properties read-only with the **ReadOnly** keyword (and you must omit the **Set** method)

Ex.
```
Public Class Vehicle
  Private V_No As String
    Public Property Vehicl_No() As String
      Get
        Return V_No
      End Get
      Set(ByVal value As String)
        V_No = value
      End Set
    End Property
End Class
```

Now you can access the property after creating an object (instance).

```
Dim Car As New Vehicle
Car.Vehicl_No = "GJ-15/786"
MsgBox(Car.Vehicl_No)
```

- **Method :** Methods represent the object's built-in procedures. For example, a class named **Vehicle** may have methods named **Moving, Start, Stop** etc. You define methods by adding procedures, either Sub procedures or functions, to your class.

Ex.

```
Public Class Vehicle
    Public Sub move( )
        MsgBox("Moving")
    End Sub
    Public Function Stop1() As String
        Return "Stoped"
    End Function
End Class
```

Now you can access the method after creating an object (instance).

```
Dim Car As New Vehicle
Car.move( )
MsgBox(Car.Stop1( ))
```

- **Event :** Events allow objects to perform actions whenever a specific occurrence takes place. An example of an event for the "Car" class would be a "Check_Engine" event. Events can be generated by a user action, such as clicking the mouse or pressing a key; by program code; or by the system.

- **Constructors :** Constructor method executes automatically whenever new object (instance) is created.

Constructor is used to initialize an object. Constructor is typically declared with Public keyword. To create a constructor for a class, create a procedure using the **Sub New( ).....End Sub** block anywhere in the class definition.

Ex.

```
Public Class Vehicle
    Private V_Color As String
    Public Sub New(ByVal newValue As String)
        V_Color = newValue
    End Sub
End Class
```

Now when you create a new object, Constructor method executes automatically

```
Dim Car As New Vehicle("Blue")
```

- **Destructors :** Destructor is a method that executes (called by the system) whenever an object goes out of the scope or destroyed. You can use this method to de-allocate resources, disconnect from the Internet, inform other objects that the current object is going to be destroyed, and more.

  In VB.NET, The destructor is written using the **Finalize** method. **Finalize** method is called automatically by the system when an object is destroyed (which means that you should not explicitly call **Finalize** yourself).

  You have to use the **Overrides** keyword with **Finalize**, because you're actually overriding the **Finalize** method built into the **Object** class. A class can have many overloaded constructors but only one destructor.

  Ex.
```
Public Class Vehicle
    Protected Overrides Sub Finalize()
        MsgBox("Object is now going to be destroyed")
        Beep( )
    End Sub
End Class
```

  You have to use the **Overrides** keyword with **Finalize**, because you're actually overriding the **Finalize** method built into the **Object** class. A class can have many overloaded constructors but only one destructor.

3.2 ABSTRACTION, ENCAPSULATION & POLYMORPHISM

True object-orient programming requires objects to support three qualities: encapsulation, inheritance, and polymorphism.

Encapsulation is the process of packaging information in such a manner that relevant information is made visible & other details of the object is hidden.

Encapsulation or information hiding refers to the facts that object hides the detail how they work. Developers hide the complexity. For example car driver should not bother about car engine.
When you set the text property of the CommandButton control, you do not know how the CommandButton internally repaints the character. Another example is caption of the form. It hides implementation details of your objects, a practice called data hiding.

Data Abstraction refers to act of representing essential features without including the background detail or explanation.

Polymorphism is the ability to have more than one form. (One name many forms). **Overloading** is the process of declaring methods (Sub procedure or Function procedure), or property or constructor of a class **having the same name but**

different signatures. The signature of the method contains the name of the method, the numbers of arguments & types of arguments. In VB.NET, (1) Functions (2) Procedure (3) Property & (4) Constructors within a class can be overloaded.

- **Access Modifier :**
  Public, Private, Protected, Friend, or Protected Friend :
  - **Public :**
    Gives public access, which means there are no restrictions on their accessibility.
  - **Private :**
    Gives private access, which means they are accessible only from within the same class, module or structure.
  - **Protected :**
    Gives protected access, which means they (elements) are accessible only from within the same class, or from a class derived from that class. Note that you can use **Protected** only at class level (which means you can't use it inside a procedure), because you use it to declare members of a class.
  - **Friend :**
    Gives friend access, which means they are accessible from within the program that contains their declaration, as well as anywhere else in the same assembly. (methods, property, fields, events are accessible throughout the program in which the class is defined)
  - **Protected Friend :**
    The Protected and Friend keywords together in the declaration statement specify that the elements are accessible either from derived classes or from within the same assembly, or both.

- **Shared(Shared Data Members) :**
  No object is needed to access the shared data members. Shared keyword is used to make shared data members.

  For example, You have a class named **Mathematics**, and declare a shared variable named **Pi** :

```
Public Class Mathematics
    Public Shared Pi As Double = 3.1415926535
End Class
```

  Now you can use **Pi** as a class variable with the **Mathematics** class directly, no object needed : `Msgbox(Mathematics.Pi)`

The variables you declare with the **Shared** keyword are called shared is because they're shared over all instances of the class. For example if you have a shared data member named **X1** to the **Mathematics** class, and each instance(object) of this class(Mathematics) will now see the same value stored in **X1**.

➢ **Shared Methods : No object is needed to access the shared methods. Shared** keyword is used to make shared methods.

For example : **Mathematics** class have Shared method name as **Add**. Then you can use the shared method without crating an object. Only you have to use ClassName.SharedMethodName **(Mathematics.Add)**

Ex.
```
Public Class Mathematics
    Shared Function Add(ByVal x As Integer, ByVal y As Integer) As Integer
        Return x + y
    End Function
End Class
```

In program, you can use the shared method without crating an object as follows.

```
Mathematics.Add(776,10)
```

- **Same way, also you can create Shared Property & Event.**

➢ **Overloading Methods, Properties, Constructor : Polymorphism** is the ability to have more than one form. (One name many forms).

Overloading is the process of declaring methods (Sub procedure or Function procedure), or property or constructor of a class **having the same name but different signatures.** The signature of the method contains the name of the method, the numbers of arguments & types of arguments.

In VB.NET, (1) Functions (2) Procedure (3) Property & (4) Constructors within a class can be overloaded.

Ex.
```
Public Class Maths
    Function Area(ByVal L As Integer) As Integer
        Return L * L
    End Function
    Function Area(ByVal B As Integer, ByVal H As Integer) As Integer
        Return 1 / 2 * B * H
    End Function
End Class
```

Now uou can access the overloaded Area method by passing different arguments.

```
Dim Maths1 As New Maths
MsgBox("Area of Square is = " & Maths1.Area(10))
MsgBox("Area of Triange is = " & Maths1.Area(10, 5))
```

3.3 INTERFACES & INHERITANCE

➢ **Inheritance :** Inheritance means heredity. Some of the characteristics of the son or daughter are common to their parents. **Inheritance is the process use to derive one class from another.**

Inheritance is the process of creating classes based on existing classes. Existing class is called as **base class** or parent class. New class is called as the **derived class** or child class.

Inheritance is based on the principle of reusability (reusability of code). Instead of creating a class from scratch, we can add new methods, properties & events to the existing classes, thus saving time & effort.

Inherits statement is used to derived a new class from the existing class. Derived classes inherit, and can extend, the properties, methods, events, fields, and constants defined in the base class

```
Class   DerivedClassName
        Inherits  BaseClassName
```

Ex.

```
Public Class Building
    Public Address As String
    Public Floors As Integer
End Class
Public Class Apartment
    Inherits Building
    Public name As String
End Class
```

Now you can access the members of the class as follows

```
Dim Apt1 As New Apartment
Apt1.name = "OM"
Apt1.Address = "Goa"
Apt1.Floors = 3
```

➢ **Overriding Base Class Members : Overriding** is the process of overwriting or replacing the methods or properties of a base class in a derived class. In case of method overriding, the return type, name & arguments of the method in the derived class must be the same as that of the method being overridden. The base class method must be marked with the keyword **Overridable** and the derived class method must be marked with the keyword **Overrides**. The following modifiers are used to control how properties and methods are overridden.

- **Overridable :** This keyword is used to mark, property or method of a base class as overridable.

- **Overrides :** This keyword is used to mark, property or method of a derived class as overrides. (It means property or method of a base class is going to overwriting or replacing).

- **NotOverridable :** Prevents a property or method from being overridden. Note that public methods are **NotOverridable** by default.

- **MustOverride :** Requires that a derived class override the property or method. **MustOverride** methods must be declared in **MustInherit** classes.

Ex.

```
Public Class MyBaseClass
    Public Overridable Sub Test(ByVal X As Integer, ByVal Y As Integer)
        MsgBox(X + Y)
    End Sub
End Class
Public Class MyDeriveClass
    Inherits MyBaseClass
        Public Overrides Sub Test(ByVal X As Integer, ByVal Y As Integer)
            MsgBox(X * Y)
        End Sub
End Class
```

Now you can access the Test method as follows

```
Dim BaseObj As New MyBaseClass
BaseObj.Test(2, 5)        'Answer will be 7
Dim DeriObj As New MyDeriveClass
DeriObj.Test(2, 5)        'Answer will be 10
```

Abstract Base Class: Use MustInherit Keyword to create Abstract Class:
An abstract class is a class that cannot be instantiated. In other words, **You can not create an object of abstract base class.** It is generally used as a template to derive other classes. An abstract class typically has abstract methods which are only method declarations without any implementation.

In VB.NET all classes are inheritable. The **MustInherit** keyword makes the base class an abstract class. **You can not create an object of abstract base class.**

Following **example** defining an abstract base class, that has one abstract method & one non- abstract method.

```
Public MustInherit Class AbsBaseClass
    Public Sub Test()
        MsgBox("Abstract Base Class")
    End Sub
    MustOverride Sub AnothrTest()
```

```
End Class
Public Class child
    Inherits AbsBaseClass
        Public Overrides Sub AnothrTest()
            MsgBox("Implementing abstract base method")
        End Sub
End Class
```

Now you can access as follows

```
Dim childobj As New child
childobj.Test()
childobj.AnothrTest()
```

- **Final Class :** Final class in VB.NET is marked with the **NotInheritable** keyword. Final class can not have derived class but can be used to create objects. So you can not inherit from this class. **Means this is the class from which no further inheritance is allowed.**

Ex.

```
Public NotInheritable Class MyFinalClass
    Public Sub Test()
        MsgBox("Final Class")
    End Sub
End Class
Public Class child
    Inherits MyFinalClass  ' à It will result compil error
End Class
```

- When you inherits final class, It will result compil error. Because of NotInheritable keyword,
- Inheritance Modifiers are MustInherit & NotInheritable

- **Shadowed :** For example, you can declare a property that shadows an inherited method with the same name. Shadowed members cannot be inherited. **Shadows** keyword can not be used with the **Overloads** keyword.

You can hide a variable by shadowing it, that is, by redefining it with a variable of the same name. If the variable you want to hide is defined at class level, you can shadow it through inheritance by redeclaring it with the Shadows keyword in a derived class

For example, imagine that a module defines a public variable named **Data**, and a procedure inside the module declares a local variable also named **Data**. When you use the name **Data** in the procedure, you use the local variable, but references to **Data** outside the procedure use the public variable. Here, the procedure-level variable **Data** shadows the module-level variable **Data**.

```
Class Class1
    Public Pi As Double = 3.1415926535
End Class
Class Class2
    Inherits Class1
        Public Shadows ReadOnly Property Pi() As Double
            Get
                Return 3.1415926535
            End Get
        End Property
End Class
```

- In above example if you do not write Shadows keword then following error will occurs.Property "Pi" conflicts with the variable "Pi" in the base class "class1" & should be declare "Shadows"

➢ **MyBase Keyword : MyBase** is commonly used to access base class members that are overridden or shadowed in a derived class. In particular, **MyBase.New** is used to explicitly call a base class constructor from a derived class constructor.

You can use the **MyBase** keyword to access methods in a base class when **overriding** methods in a derived class. For example, suppose you are designing a derived class that overrides a method inherited from the base class; in this case, the **overridden** method can call the original method in the base class using **MyBase**.

Ex.

```
Public Class Animal
    Public Overridable Sub Eats()
        MsgBox("Eats Foods Base Class Eats Method")
    End Sub
End Class
Public Class Cow
    Inherits Animal
        Public Overrides Sub Eats( )
            MsgBox("Eats Grass Derived Eats Method ")
        End Sub
        Public Sub Eating( )
            MyBase.Eats( )
            MsgBox("Is a herbivourous animal")
        End Sub
End Class
'Creates Object as follows
    Dim C As New Cow( )
    C.Eating( )
    C.Eats( )
```

Output : Eats Foods Base Class Eats Method
Is a herbivourous animal
Eats Grass Derived Eats Method

The code in above example creates a base class animal that defines one **Overridabel** method Eats( ). This Eats( ) method is later **Overrides** by a child class cow. Derived class has also another method Eating( ), in that we want to access the original method Eat( ) of the base class, and not the **Overrides** method of the child class. To indicate that it is the base class method we wish to invoke, We have to invoke wite the **MyBase** keyword.

- **MyClass Keyword : MyClass keyword works opposite to the way in which MyBase keyword works.**

Assume that, If we wish to call the base class version of the method (an **overridable** method), instead of (overridden method in) the derived class version, we must use the **MyClass** keyword.

```
Public Class Animal
    Public Overridable Sub Eats()
        MsgBox("Animal Eats Food")
    End Sub
    Public Sub EatingHabits()
        MsgBox("Calling method directly")
        Eats()
        MsgBox("Calling method using MyClass")
        MyClass.Eats()
    End Sub
End Class
Public Class Cow
    Inherits Animal
        Public Overrides Sub Eats()
            MsgBox("Cow Eats Grass")
        End Sub
End Class
```

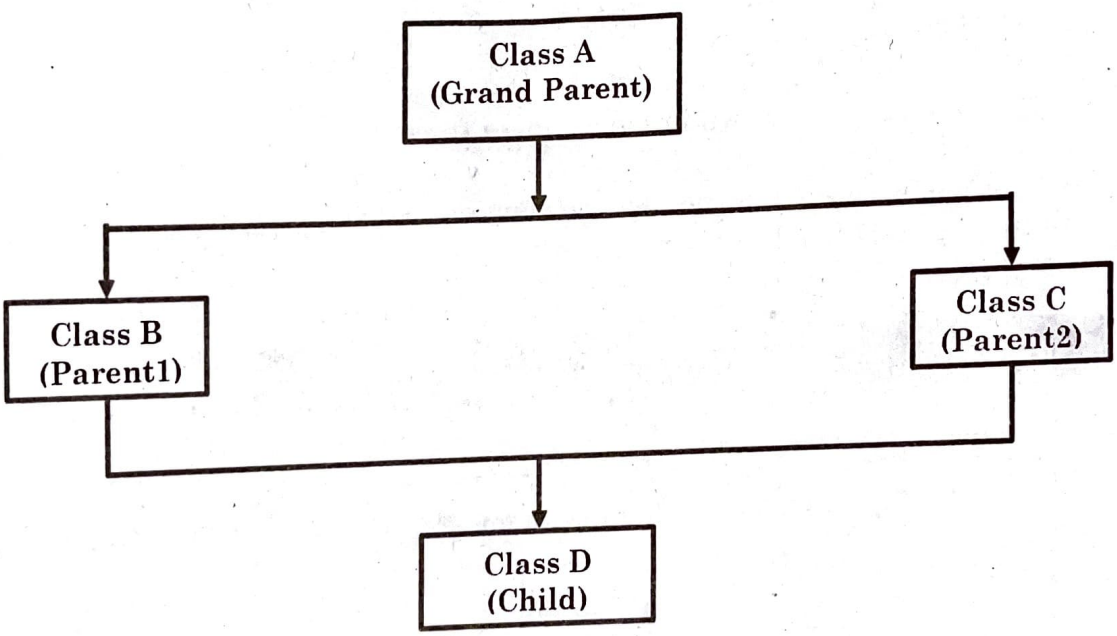
```
'Creates Object as follows
    Dim C As New Cow
    C.Eats()
    C.EatingHabits()
```

Output : Cow Eats Grass
Calling method directly
Cow Eats Grass
Calling method using MyClass
Animal Eats Food

'When MyClass.Eats() method is called it called the base class (an **overridable** method, instead 'of the the derived class overridden method.

- **Interfaces** : VB .Net language does not support the concept of multiple inheritance. VB .Net requires **interface** between two base class for deriving new class.

Interface is a method used to implement multiple inheritance.
Like classes, interface defines a set of properties, methods, & events. But interface does not contain any implementation code (just declaration). Interface definitions are enclosed within the **Interface & End Interface** statements.
If we want to use interface, then we need to write "**Implements**" keyword. In that class we have to write implementation code.



Using **interface**, we can make child class D, which is derived from two classes (B & D).

Ex.
```
Public Interface ABC
    Function MyFunc(ByVal I As Integer) As Integer
    Sub MySub( )
End Interface
Public Interface DEF
    Function MyFunc1(ByVal I As Integer) As Integer
End Interface
```

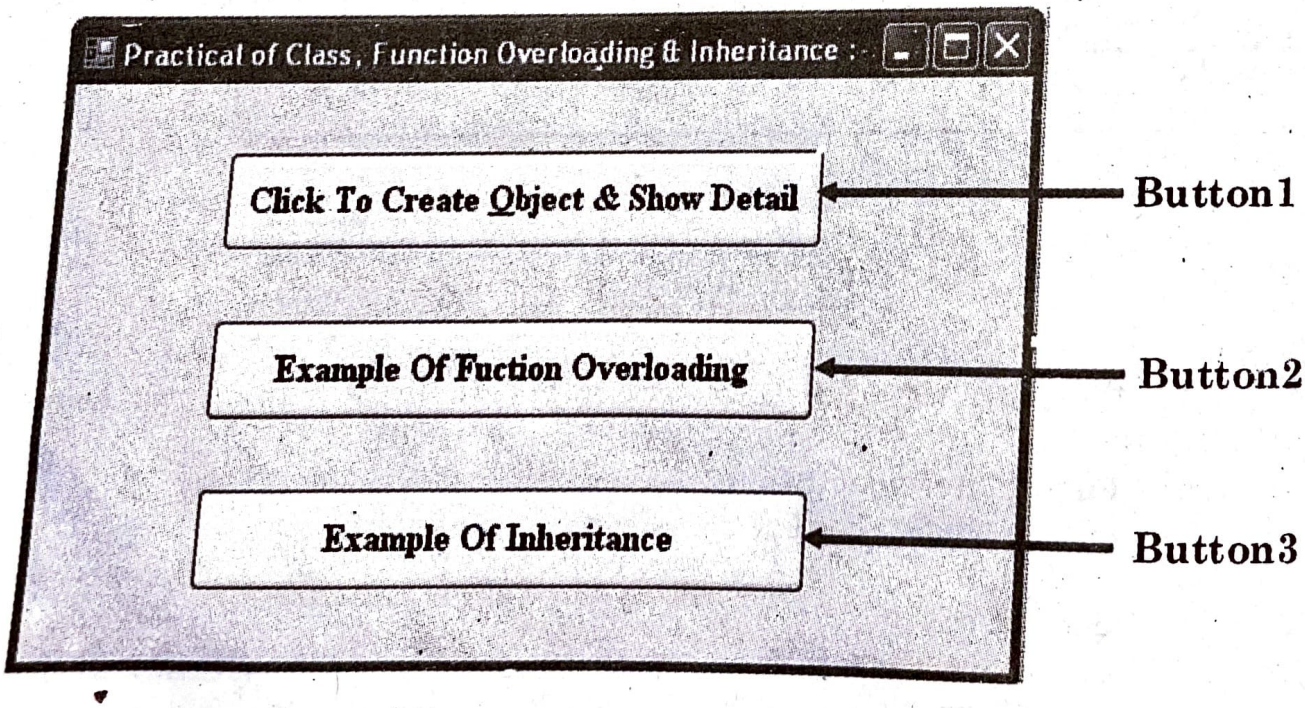
Above example, defines two interfaces named **ABC & DEF. ABC** interface contains one function & one sub procedure definition. **DEF** interface contains one sub procedure definition. (It has been just declared (No implementation).

'If we want to use interface, then we need to write "**Implements**" keyword. & we have to define implementation code.

```
Public Class XYZ
    Implements ABC, DEF
    Public Sub MySub() Implements ABC.MySub
        '.........Logic.........
    End Sub
    Public Sub MySub1() Implements DEF.MySub1
        '.........Logic.........
    End Sub
End Class
```

➢ **Practical 13 :**
[A]Example of Class (Field, Property, Method, Constructor, Destructor)
[B]Function overloading
[C]Inheritance

Design interface as shown in figure. Place three buttons.
Button1 is used to create an Object & show detail.
Button2 is used for Function Overloading example.
Button3 is used for Inheritance example.



'First create a class named Vehicle, with Data member, Property, 'Fuction, Constructor, Desctructor.

```
Public Class vehical
    '.............Create Fields or Data member.....
    Private v_no As String
    Private v_color As String
    Public v_name As String
    Public v_prise As Double
    '.............Create Property (v_no)............
    Public Property vehical_no() As String
        Get
            Return v_no
        End Get
        Set(ByVal value As String)
            v_no = value
        End Set
    End Property
    '.............Create Function with return value.........
    Public Function stop1() As String
        Return "stoped"
    End Function
    '.............Create Function with return value..........
    Public Function scolor() As String
        Return v_color
    End Function
    '.............Create Constructor.............
    Public Sub New(ByVal newvalue As String)
        v_color = newvalue
    End Sub
    '.............Create a Destructor.............
    Protected Overrides Sub finalize()
        MsgBox("Object is now going to be Destroyed")
        Beep()
    End Sub
End Class
```

'Coding of Button1 (click to create an Object & show detail button).
'Create an object of Vehical class.

```
Dim car As New vehical("Sky Blue")
car.v_name = "Maruti Zen"
car.v_prise = 255000
car.vehical_no = "GJ-15/786"
MsgBox(car.scolor())
MsgBox(car.stop1())
```

```
MsgBox("NAME of CAR is " & car.v_name & Chr(13) & "PRIZE of that CAR is:-" &
car.v_prise & Chr(13) & "CAR NUMBER IS:-" & car.vehical_no)
```

'Now for Function Overloading, First we have created Maths class. 'It contains two function having same named Area, but with 'different arguments.

```
Public Class Maths
    Function Area(ByVal L As Integer) As Integer
        Return L * L
    End Function
    Function Area(ByVal B As Integer, ByVal H As Integer) As Integer
        Return 1 / 2 * B * H
    End Function
End Class
```

'Coding of Button2 (Example of Function Overloading button).
'Create an object of Class Maths.

```
Dim Maths1 As New Maths
MsgBox("Area of Square is = " & Maths1.Area(10))
MsgBox("Area of Triange is = " & Maths1.Area(10, 5))
```

'Now for Inheritance, First we have created a base class or parent 'class named as "Building" class. Also created a derived class or 'child class named "Apartment".

```
Public Class Building
    Public address As String
    Public Floor As Integer
End Class
Public Class Apartment
    Inherits building
    Public name As String
End Class
```

'Coding of Button3 (Example of Inheritance button).
'Create an Object of class Apartment.

```
Dim Apt1 As New Apartment
Apt1.name = "SAI Apartment"
Apt1.address = "Tithal Road, Valsad"
Apt1.Floor = 1
MsgBox(Apt1.name & Chr(13) & Apt1.address & Chr(13) & Apt1.Floor)
```

UNIT 4

DATABASE ACCESS USING ADO.NET

4.1 ADO.NET Overview
4.2 ADO.NET Object Model OR ADO.NET Architecture
4.3 Use of ADO.NET Objects Directly in Visual Basic Code.
4.4 Data Access Components or Data Components

4.1 ADO.NET OVERVIEW

- ADO.NET is an **easy-to-use, a faster & more efficient data access technology** for building database applications for Windows & the Web.

- This technology provides a user-friendly interface for accessing data in any type of database & accessible to developers coding in most **any type of language**.

- ADO.NET is a set of **highly object-oriented, extensible library of classes** for accessing & manipulating database. The data can be form SQL server, Oracle, Text files on your disk, XML streams, or virtually any where else.

- ADO.NET has been designed for Internet use, meaning **that it use the same method for accessing local, client-server, and Internet based data sources.**

- **ADO.NET components** are at the core (heart) of developing database applications & provide the necessary classes that **enable you to connect to a data store & to retrieve, manipulate & update data.** Many components are available in ADO.NET, which is implemented under the **System.Data** namespace.

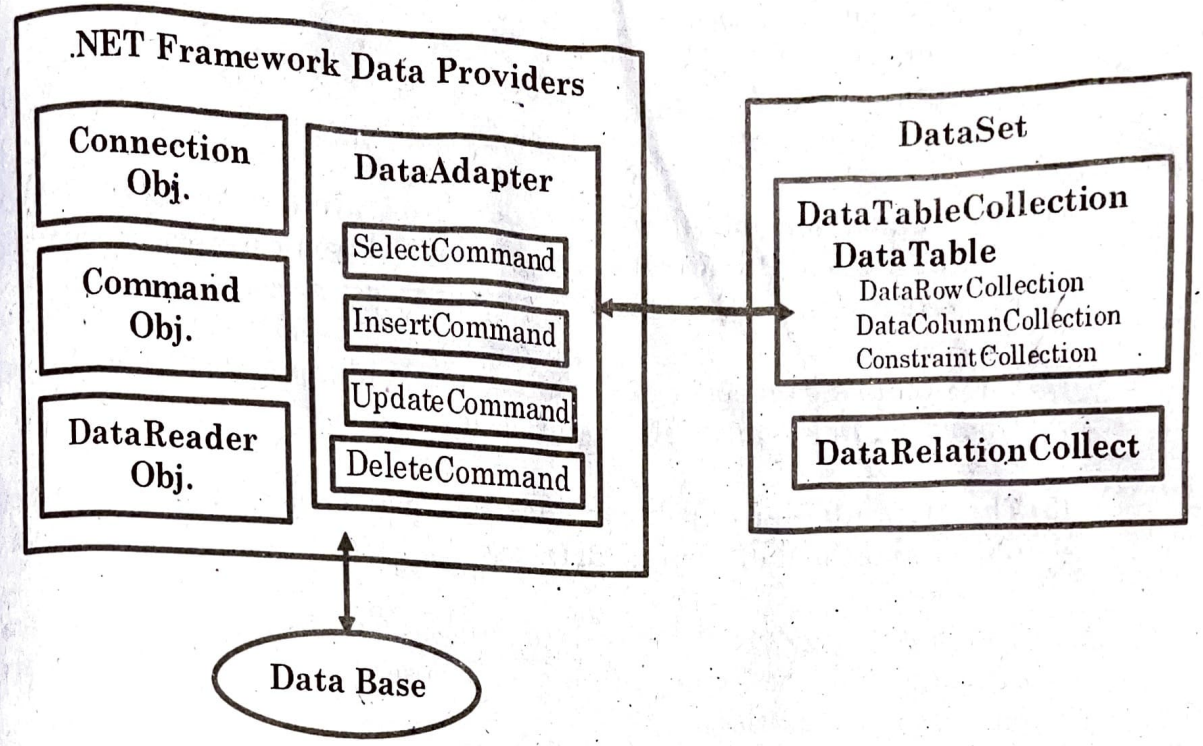
- This namespace, in turn contains some child namespaces. The most important of these are **System.Data.SqlClient** (for accessing SQL server database), **System.Data.Oledb** (for accessing OLEDB compliant databases), **System.Data.OracleClient** (is used exclusively for accessing Oracle databases, provides optimum performance when accessing Oracle databases, Works with Oracle client software version 8.1.7 and later), **System.Data.Odbc** (this namespace provides access to older ODBC data sources that do not support the OLEDB technology such as MySQL or Sybase.

- The **System.Data.SqlClient, System.Data.Oledb, System.Data. OracleClient** and **System.Data.Odbc** namespaces are known as **data providers** in ADO.NET. The default data provider that Visual Basic .NET works with is Microsoft's SQL Server, version 7.0 or later.

- ADO .NET provides a **disconnected database access model (disconnected architecture).** ADO .NET **conserves (preserve / save) system resources and provides maximum security** for databases and also has less impact on system performance

Visual DataBase Tools are Server Explorer / Database Explorer (A graphical tool), Data Adapter Configuration Wizard (A graphical tool) etc

4.2 ADO.NET OBJECT MODEL OR ADO.NET ARCHITECTURE



ADO .NET provides a disconnected database access model (disconnected architecture). This means that you connect to the database, get the data that you need, & then disconnect from the database. You work with the data offline in the DataSet or DataTable components. **This provides high scalability for your applications as it allows more clients to connect to the database & retrieve data.** This is particularly important in Web applications, where you could potentially have thousands of clients requesting data simultaneously.

ADO.NET Components : (ADO.NET classes are divided into two components.) There are two components of ADO.NET that you can use to access and manipulate data :

[1] .NET Framework data providers
[2] The DataSet

[1] .NET Framework Data Providers : The .NET Framework Data Providers provides methods for connecting to a database, retrieving data from the database, committing changes to the database, and executing database commands.

The **System.Data.SqlClient, System.Data.Oledb, System.Data. OracleClient** and **System.Data.Odbc** namespaces are known as **data providers** in ADO.NET. The default data provider that Visual Basic .NET works with is Microsoft's SQL Server, version 7.0 or later.

The core (foundation) of above four data providers are four Objects as described below.(Core Objects of .NET Framework Data Providers are as described below.)

(1) Connection Object
(2) Command Object
(3) DataAdapter Object
(4) DataReader Object

(1) The Connection object provides connectivity (physical connection) to a data source (database). Using its method, you can open & close the connection, change the database & manage transactions.

(2) The Command object is used to execute SQL statements (Select, Insert, Update & Delete) as well as stored procedure.

(3) The DataAdapter object provides the bridge between the DataSet object and the data source (database) for retrieving and saving data.

The DataAdapter's sole purpose is to retrieves data from the database, then populates (fill) the Datasets & also used to send (propagate) the Datasets changes to the database.

The DataAdapter contains four command objects: SelectCommand, InsertCommand, UpdateCommand, and DeleteCommand. The DataAdapter uses the SelectCommand to fill a DataSet & the remaining three commands to transmit changes back to the data source.

(4) The DataReader in addition to datasets, there are also data readers, which are extremely fast, read-only, forward only low-overhead way of retrieving information from the database. You can only move through records with in ascending order. It provides a high-performance stream of data from the data source (database). If we only need single direction, read only loading of DataSets or controls, we can bypass the Adapter entirely & utilize a DataReader. DataReader cannot be created directly in code; they are created only by calling the ExecuteReader method of a Command object.

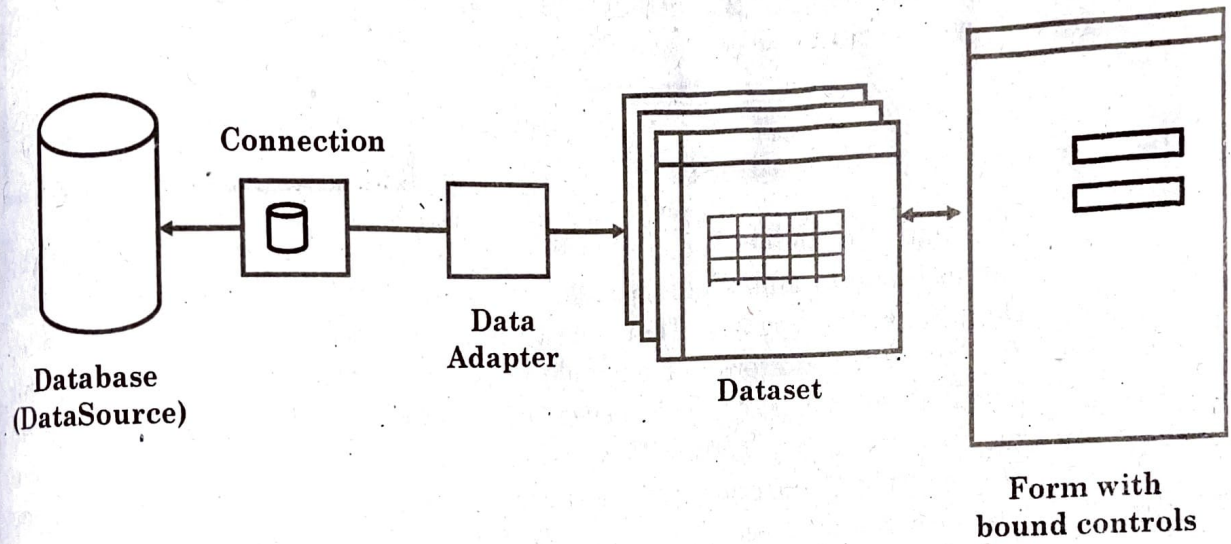
[2] **The DataSet :** It is the major component of ADO.NET. DataSet is a memory-resident representation of data. **A dataset is a disconnected cache of data, and, that is stored in memory.** DataSet is always disconnected from the data source. It can contain data from multiple sources.

As shown in figure of ADO.NET object model, The DataSet composed of two primary objects: the **DataTableCollection**, accessed through Tables property, and **DataRelationCollection** accessed through the Relations property. The DataTableCollection contains zero or more DataTable objects, which are in turn

made up of three collections: DataColumnColection, DataRowCollection, and ConstraintCollection. The DataRelationCollection contains zero or more DataRelation objects.

The dataset is a disconnected, in-memory representation of data. A DataSet is a local copy of relational data & all operations that need to be carried out on the data are done on this local copy (independent of the database) with no connection to the original data store. An advantage of this is that we do not need to have a continuous connection to the database.

Steps : To receive & modify information of a database



In ADO .NET various objects are used to receive & modify information in a database.

[1] First a **connection** is made, which specify the connection information about the database.

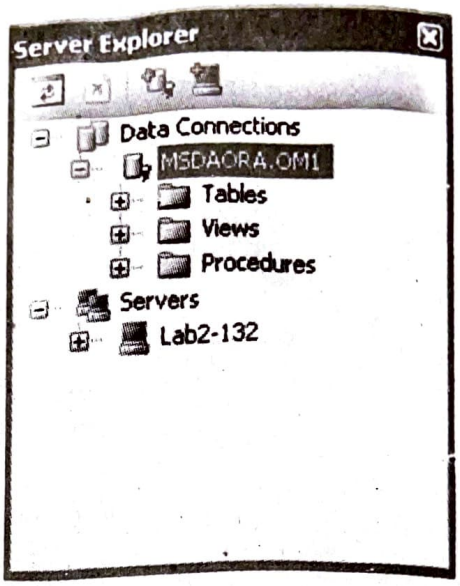
[2] Next a **data adapter** is created, which manages receiving data from the database and posting data changes. (Bridge between the DataSet & DataSource)

[3] Then a **dataset** is created, which is a representation of one or more database tables, you plan to work with in your program. (You don't manipulate the actual data, but rather a copy of it.)

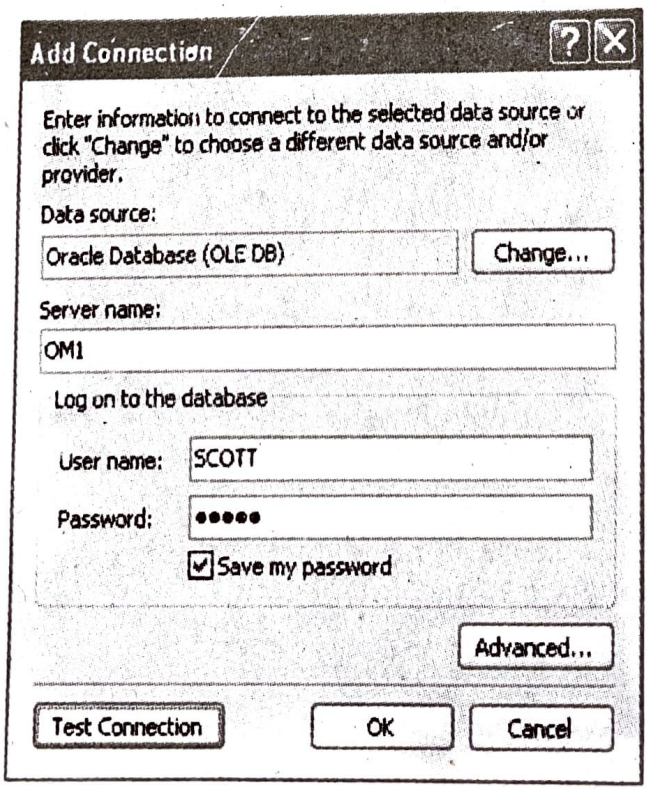
[4] Information in the dataset can then be **bound** to controls on a form.

(a) Establish a connection to a database [Server Explorer / Database Explorer (A graphical tool)] :

Server Explorer or Database Explorer is a graphical tool that lets you establish connections to local, client-server, or Internet-based data sources. Using Server Explorer you can Create, modify, delete, and refresh database connections. To display the Server Explorer if it's not already visible, then click on Server Explorer menu item from the View menu. or press Ctrl+Alt+S shortcut key. It is shown in the figure.



To create the connection, right-click on the Data Connections icon in the Server Explorer and select the Add Connection item. Then it will open Choose Data Source Dialog box. Then select the desire data source & data provider & click on OK button. For example select "Oracle Database (OLE DB)" for Oracle database. Then type server name, user name & password. (Click on change button to select a different data source and/or data provider.) Then click on Test Connection button. If the connection is ok, you'll see a message box with the message "Test connection succeeded".



(b) Creating Data Adapter [Data Adapter Configuration Wizard (A graphical tool)] : Visual Studio provides several mechanisms for creating data adapter in a program. One method is to use of a graphical tool called the Data Adapter Configuration Wizard. This tool is easy to use.

If OledbDataAdapter object is not available in the tool box, then right click on toolbox & click Choose Items. The Choose Toolbox dialog box will be open. From the .NET Framework components tab, select the OledbDataAdapter object & click ok. Now drag the OledbDataAdapter object from the Toolbox to the main form.

Right click on OleDbDataAdapter1 & select Configure Data Adapter; Then Data Adapter Configuration Wizard will be open. In wizard first choose your data connection. Then click on next button to choose a command type for the adapter. Click on next button in that you can write SQL statement or use Query Builder to construct query. And last click on Finish button to close the Data Adapter Configuration Wizard.

(c) Creating DataSet : A DataSet can contain information from one or more database tables. Now to create a new dataset using this data adapter, as we've done before, Click on Data menu & select **Generate Dataset**. It will open Generate Dataset dialog box. Then you can choose the DataSet either existing or new. Then select the tables, which you want to add to the DataSet. And last click on OK button. Then it will create DataSet11.

(d) Now Binding with DataGridView : Drag one DataGridView on to the form. Now connect the new DataSet to a DataGridView using the **DataSource** and **DataMember** properties.

- Now On the form there in one button with caption "Load Data" to load data into the DataGridView. And write down following code.

```
DataSet11.Clear()
OleDbDataAdapter1.Fill(DataSet11)
```

- When the user edits the data in the data grid control, that control automatically updates the DataSet. To send the new updated data back to the database, you have to use the Data Adapter's **Update** method

Now On the form there in other button with caption "Update Data". And write down following code.

```
OleDbDataAdapter1.Update(DataSet11)
```

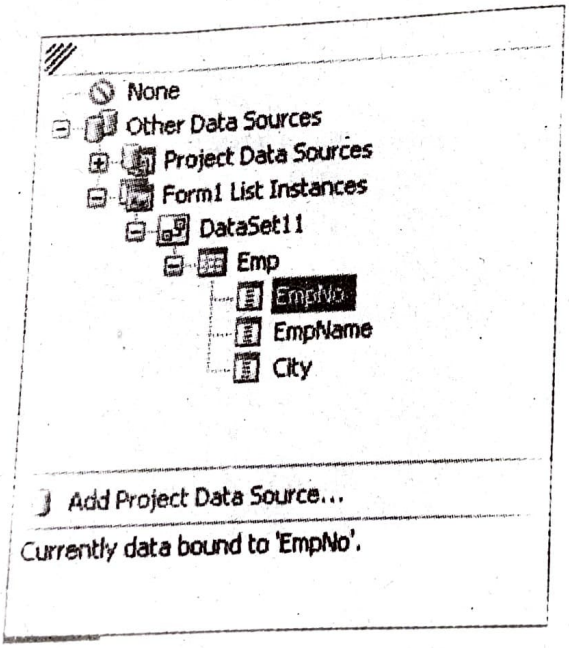
- **Data Binding :** It is the process ot attaching the Data source to a data aware control is called as Data Binding.
  .NET supports two types of Data Binding.
  [1] Simple Data Binding &
  [2] Complex Data Binding.

[1] Simple Data Binding : In Simple Data Binding, only a single value (one data element) from a DataSet can be bound to control at a time. In this type of Data Binding, any value from the DataSet can be bound to any property of a control.

Simple Binding at design time : For example EmpName field in the Employee table in a DataSet11, can be boud to Text property of the TextBox.

First select the TextBox control, then exapand its DataBinding property in the Properties window. Then click on Text property, then click on drop down arrow. The property window display a list of data sources that you can bind to the TextBox.(As shown in figure).

For Ex. : DataSet11 - Emp.EmpNo



Simple Binding in code (at run time) : At run time, we can use a control's **DataBindings** property, which holds a collection of **Binding** objects. **DataBindings** collection has **Add** method.Which adds a Binding to the collection.

```
TextBox1.DataBindings.Add("Text", DataSet11, "Employee. EmpName")
```

[2] **Complex Data Binding :** In Complex Data Binding, a control is bound to a complete DataSet (more than one data element) instead of just one column from it. Most controls support only simple data binding but some controls, such as DataGridView, ComboBox, ListBox & CheckedListBox support complex data binding.

For complex data binding, following properties can be use.

(a) **DataSource :** The data source, a dataset such as **DataSet11**.

(b) **DataMember :** A table in a dataset such as the **Employee** table. Data grids use this property to determine which table they should display.

(c) **DisplayMember :** The field you want a control to display, such as the Dept_Name. List boxes use the **DisplayMember** and **ValueMember** properties instead of a **DataMember** property.

(d) **ValueMember :** The field you want the control to return in properties like **SelectedValue**, such as Dept_No. List boxes use the **DisplayMember** and **ValueMember** properties instead of a **DataMember** property. For example you can use the **SelectedValue** property of the control to get the selected Dept_No.

Here at run time we can bind the DataSet11 to a ComboBox control, using DisplayMember property, by Dept_Name field in Department table. So Combobox will display Department Names.

Now to get the respective department No, When Department name is selected, you have to set **ValueMember** property to Dept_No. You can access the Department No using the **SelectedValue** property.

```
ComboBox1.DataSource = DataSet11
ComboBox1.DisplayMember = "Department.DeptName"
ComboBox1.ValueMember = "Department.Dept_No"
MsgBox(ComboBox1.SelectedValue)
```

To bind a dataset, **DataSet11**, to a DataGridView1, showing the Employee table : DataGrid1.DataSource = DataSet11

```
DataGrid1.DataMember = "Employee"
```

❖ **Different Data Sources :**

Many components can act as a data source for data binding. For example following data sources are supported by ADO.NET

DataSet, DataTable, DataView, DataViewManager etc

➢ **Record Navigation :**

[1] CurrencyManager Object : ADO.NET keeps track of information about the current record & the total number of records by using an object named the **CurrencyManager**.

[2] BindingContext Object : There is a CurrencyManager object for each DataSet, and each Windows form has a **BindingContext** Object that keeps track of all the CurrencyManager objects on the form.

[3] Position property : A zero-based index that specifies a position of current record in the DataSet. We can use a **BindingContext** object's **Position** property to move (navigate) through a dataset.

To go to the first item, set the **Position** to 0. To go to the end of the list, set the **Position** to the value of the Count property minus 1.

The PositionChanged event occurs when the **Position** property value changes.

[4] Count property : It specify the total no of records in the DataSet.

❖ **Moving to the First Record :** To move to the first record in the DataBinding example, set the **Position** property to **0** :

```
Me.BindingContext(DataSet11, "Employee").Position = 0
```

❖ **Moving to the Last Record :** To move to the last record, set the **Position** property to the total count of the records in the dataset minus one:

```
Me.BindingContext(DataSet11, " Employee ").Position = Me.Binding
Context(Data Set11,"Employee").Count - 1
```

❖ **Moving to the Next Record :** To move to the next record, Increment the **Position** property (this property won't increment if we're at the end of the dataset) :

```
Me.BindingContext(DataSet11, "Employee").Position  += 1
```

OR

```
Me.BindingContext(DataSet11, "Employee"). Position  =
Me.BindingContext(Data Set11,"Employee"). Position +1
```

❖ **Moving to the Previous Record :** To move to the previous record, Decrement the **Position** property (this property won't decrement if we're at the beginning of the dataset) :

```
Me.BindingContext(DataSet11, "Employee").Position  -= 1
```

OR

```
Me.BindingContext(DataSet11, "Employee"). Position  =
Me.BindingContext(DataSet11,"Employee"). Position -1
```

➢ **DataGridView Control or DataGrid Control : The DataGridView control provides a powerful and flexible way to display data in a tabular format.** You can use the **DataGridView** control to show read-only views of a small amount of data, or if you are enabling a user to edit a table with millions of records. With the **DataGridView** control, you can display and edit tabular data from many different kinds of data sources (database). The **DataGridView** control can also be used in unbound mode, with no underlying data store. The **DataGridView** control is the new control that replaces the **DataGrid** control; however, the **DataGrid** control is retained for both backward compatibility and future use.

- **Property :**

(1) DataSource : Gets or sets the name of data source that the DataGridView is used for displaying data. The data source can be of DataSet, DataTable etc. For example **DataGridView1.DataSource = "DataSet11"**

(2) DataMember : Gets or sets the name of the table or list of data, the DataGridView should display. You do not need to set this property when binding to a data source that contains a single table or list DataGrid1.DataMember = "Employee"

(3) Columns : By using the **DataGridViewColumnCollection** returned by this property, you can add columns, remove columns, and count the columns contained in the DataGridView control :

```
DataGrid1.Columns.Add(columnName, headerText)
```

Where columnName is the name by which the column will be referred. HeaderText is the text for the column's header.

```
DataGridView1.Columns.Add("Col1", "Employee No.")
```

- To change the Column name & Styles using the **column index**

```
DataGridView1.Columns(0).HeaderText = "OMSHIVSAI1"
```

- To change the width of the column do as follow

```
DataGridView1.Columns(0).Width = 300
```

(4) Rows : A **DataGridViewRowCollection** that contains all the rows in the DataGridView.

(5) AllowUserToAddRows : Default is True.

(6) AllowUserToDeleteRows : Default is True.

(7) AllowUserToOrderColumns : Default is False.

(8) AllowUserToResizeColumns : Default is True.

(9) AllowUserToResizeRows : Default is True.

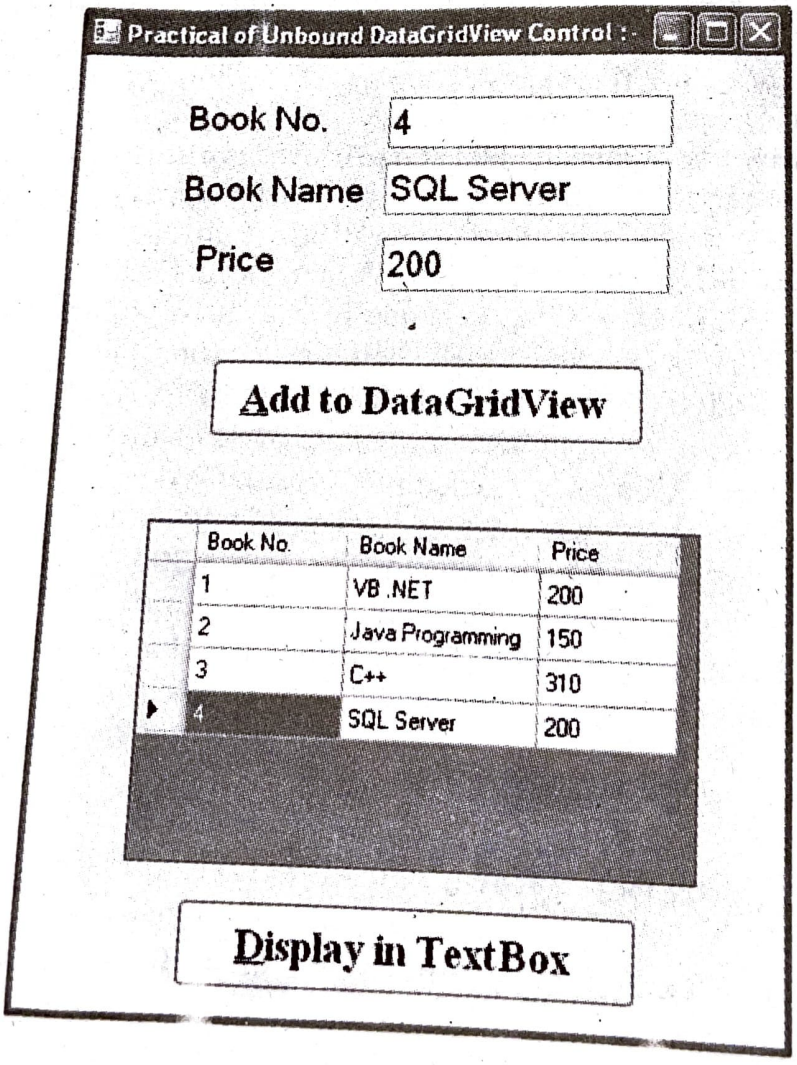
(10) **AlternatingRowsDefaultCellStyle :** Gets or sets the default cell style applied to odd-numbered rows of the DataGridView.

(11) **ColumnHeadersVisible :** Default is True.

(12) **ReadOnly :** Default is False.

Practical 15 : Practical of Unbound DataGridView control.
Suppose we have three TextBox controls on the form for Book No, Book Name & Price. When values are entered & **Add to DataGridView** Button is click, It should added in DataGridView control.

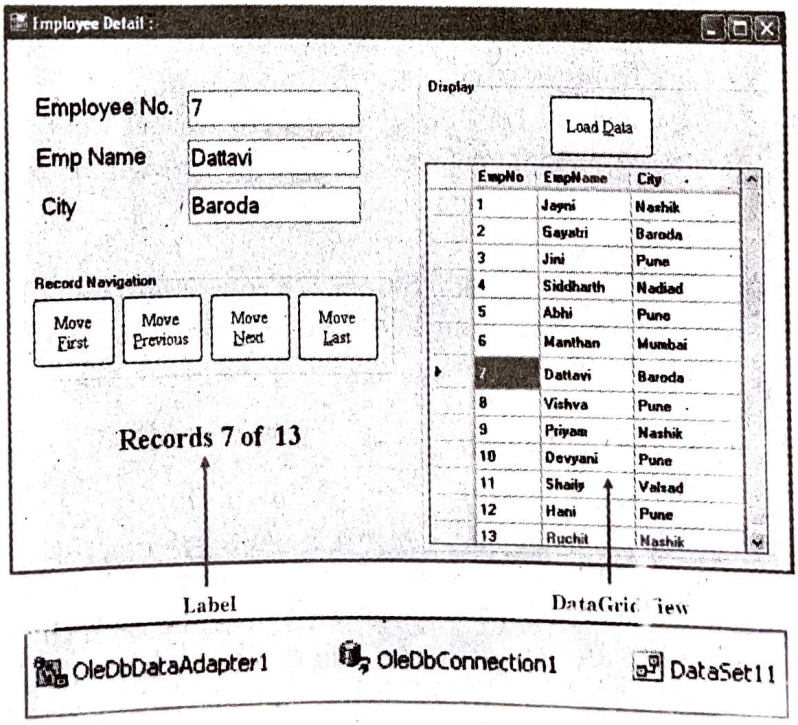
When **Display in TextBox Button** is click, then selected row should be display in three TextBox control.



```
'Coding of Form Load Event.
'In Form Load Event, add three columns & set its width.
    DataGridView1.AllowUserToAddRows = False
    DataGridView1.Columns.Add("Col1", "Book No.")
    DataGridView1.Columns.Add("Col2", "Book Name")
    DataGridView1.Columns.Add("Col3", "Price")
    DataGridView1.Columns(0).Width = 80
    DataGridView1.Columns(1).Width = 100
    DataGridView1.Columns(2).Width = 70
'Coding of Add to DataGridView Button.
    DataGridView1.Rows.Add()
    Dim X As Integer = DataGridView1.Rows.Count
    X = X - 1
    DataGridView1.Rows(X).Cells(0).Value = TextBox1.Text
    DataGridView1.Rows(X).Cells(1).Value = TextBox2.Text
    DataGridView1.Rows(X).Cells(2).Value = TextBox3.Text
'Coding of Display in TextBox Button.
'Selected row should be display in to the TextBox control.
    Dim CurrentRow = DataGridView1.CurrentRow.Index
    TextBox1.Text = DataGridView1.Rows(CurrentRow).Cells(0).Value. ToString
    TextBox2.Text = DataGridView1.Rows(CurrentRow).Cells(1).Value. ToString
    TextBox3.Text = DataGridView1.Rows(CurrentRow).Cells(2).Value. ToString
```

Practical 16 : Create Connection with MS Access database and display data in the data grid view and also perform record navigation operation.

MS-Access Table: Emp(EmpNo, EmpName, City)



Step 1 : First Make the MS-Access table named "Emp" with the three fields. (EmpNo, EmpName & City). Enter some records.

Step 2 : Design the interface as shown in figure. First place three TextBox controls & three labels. Place one GroupBox. Set Text property of GroupBox1 to "**Record Navigation**". Place four buttons in GroupBox1 & set Text property of buttons as shown in figure.

Drag one Label control (Label4), & set its Text property to "**Record 0 of 0**".

Place second GroupBox on the right side. Set Text property of GroupBox2 to "**Display**". Place one buttons in GroupBox2 & set Text property of button5 to "**&Load Data**". Place DataGridView control in GroupBox2.

Step 3 : **Create a connection with MS Access Database using Server Explorer.** Click on Server Explorer menu item from the View menu. OR Press Ctrl+Alt+S shortcut key.

To create the connection, right-click on the Data Connections icon in the Server Explorer and select the Add Connection item. Then it will open Choose Data Source Dialog box. Then select the **"Microsoft Access Database File"** from Data source list & select data provider. Click on OK button. Click on Browse button to select your MS-Access database file. Then click on Test Connection button. If the connection is ok, you'll see a message box with the message "Test connection succeeded".

Step 4 : **Create a OledbDataAdapter object.**

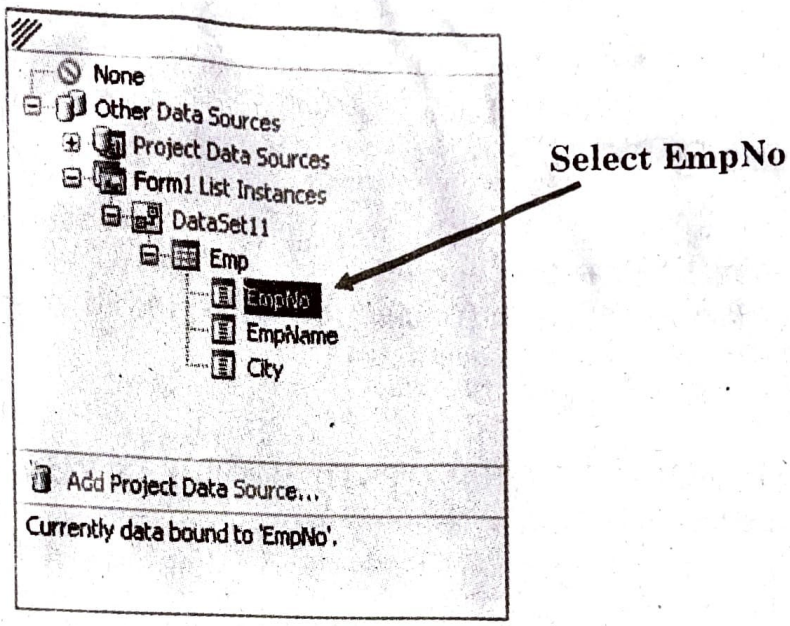
If OledbDataAdapter object is not available in the tool box, then right click on toolbox & click Choose Items. The Choose Toolbox dialog box will be open. From the .NET Framework components tab, select the OledbDataAdapter object & click ok. Now drag the OledbDataAdapter object from the Toolbox to the main form.

Right click on OleDbDataAdapter1 & select Configure Data Adapter; Then Data Adapter Configuration Wizard will be open. In wizard first choose your data connection. Then click on next button to choose a command type for the adapter. Click on next button in that you can write SQL statement (**Select * From Emp**) or use Query Builder to construct query. And last click on Finish button to close the Data Adapter Configuration Wizard. **OleDbDataAdapter1** has been created.

Step 5 : **Generate a DataSet.**

Now to create a new dataset using this data adapter (OleDbDataAdapter1), as we've done before. Click on Data menu & select **Generate Dataset**. It will open Generate Dataset dialog box. Then click on New Radio Button. Check the Emp(OleDbDataAdapter1) to add to the DataSet. And click on OK button. Then it will create **DataSet11.**

Step 6 : **Now, Binding of three TextBox Controls with DataSet11.**
First select the TextBox1, then exapand its DataBinding property in the Properties window. Then click on Text property, then click on drop down arrow. The property window display a list of data sources that you can bind to the TextBox. (As shown in figure).
For Ex. DataSet11 - Emp.EmpNo



Similarly, binds the TextBox2 with EmpName & TextBox2 with City.

Step 7 :

```
'Coding of Form Load Event.
' Fill the DataSet11
    DataSet11.Clear()
    OleDbDataAdapter1.Fill(DataSet11)

'Coding Of Load Data Button.
'Bind the DataSet11 to the DataGridView control.
    DataGridView1.DataSource = DataSet11
    DataGridView1.DataMember = "Emp"

'Display ON THE LABEL CONTROL Procedure.
    Public Sub DisplayOnLabel()
        Dim CurrentPosition, TotalRecord As Integer
        CurrentPosition = Me.BindingContext(DataSet11,"EMP").Position +1
        TotalRecord = Me.BindingContext(DataSet11, "EMP").Count
        Label4.Text = "Records " & CurrentPosition & " of " & TotalRecord
    End Sub
```

```
'Coding Of Move First Button.
    Me.BindingContext(DataSet11, "Emp").Position = 0
    DisplayOnLabel()

'Coding Of Move Previous Button.
    Me.BindingContext(DataSet11, "Emp").Position -= 1
    DisplayOnLabel()

'Coding Of Move Next Button.
    Me.BindingContext(DataSet11, "Emp").Position += 1
    DisplayOnLabel()

'Coding Of Move Last Button.
    Me.BindingContext(DataSet11, "Emp").Position = Me.BindingContext
    (DataSet11,"Emp").Count - 1
    DisplayOnLabel()
```

4.3 USE OF ADO.NET OBJECTS DIRECTLY IN VISUAL BASIC CODE.

There are two ways to access & manipulate data of database. First method is visually (with graphical tools) & second method is via coding (using ADO.NET objects directly in coding).

ADO .NET objects are **Connection Object, Command object, DataAdapter object, DataSet objects, DataReader Object, DataTable Object, DataRow Object, DataColumn Object, etc.**

[1] **Connection Objects : The Connection object provides connectivity (physical connection) to a data source (database).** Using its method, you can open & close the connection, change the database & manage transactions. Connection class exist for **ODBC** (OdbcConnection), **OLE DB** (OledbConnection), **SQL Server** (SqlConnection) & **Oracle** (OracleConnection)

- **Property :**

  (1) ConnectionString : Gets or sets the string that is used to connect (open) a database when the Open method is executed.

  ConnectionString property of OledbConnection object has arguments like Provider, Data Source, Database, User ID, Password.

  Provider parameter specifies the driver that uses to communicate with the database. The most common drivers are **Microsoft.Jet.OLEDB.4.0** for Access, **SQLOLEDB** for SQL server & **MSDAORA** for Oracle. Because the SqlConnection & OracleConnection classes have been designed specifically for SQL Server & Oracle, the Provider parameter is not needed for those classes.

Example ConnectionString property of OleDbConnection class to connect MS Access :

```
Imports System.Data.Oledb
.....................................................
Dim Con as OleDbConnection
.....................................................
Con = New OleDbConnection("Provider= Microsoft.Jet.OLEDB.4.0; Data
Source= 'D:\SAI_DB.mdb'")
Con.Open( )
.....................................................
```

Example ConnectionString property of OleDbConnection class To connect Oracle :

```
Imports System.Data.OleDb
.....................................................
Dim Con As New OleDbConnection
.....................................................
Con.ConnectionString = "Provider=MSDAORA;DataSource= OMSAI1;User
ID=SCOTT;PASSWORD=TIGER"
Con.Open
.....................................................
```

(2) ConnectionTimeout : The maximum time the Connection object attempts to make the connection before throwing an exception. (before terminating the attempt and generating an error). By default is 15 seconds.

(3) DataSource : It is used to specify the server name of the computer on which the database is running. When connecting to an Access database, this specifies the path & database name.

(4) State : Gets (returns) the current state of the connection. For example we get Closed, if the connection is closed. & we get Open, if the connection is open.

(5) ServerVersion : Gets the version of the server.

(6) Provider : This property represents the name of the provider. Because the SqlConnection & OracleConnection classes have been designed specifically for SQL Server & Oracle, the Provider parameter is not needed for those classes.

- **Method :**

(1) Open : Opens a database connection with the property settings specified by the ConnectionString.

(2) Close : Closes the connection to the data source. After the connection is close no transaction can be perform on the database data.

```
Con.Close( )
Con.Dispose( )   'Releases the resources used by the connection object.
Con = Nothing    'Release your reference to the connection object
```

(3) BeginTransaction : Starts (begins) a database transaction.

[2] Command Objects : The Command object is used to execute SQL statements (Select, Insert, Update & Delete) as well as stored procedure. In addition to the DML statements, you can also execute DDL statements that change the structure of the database. You can also use the **Parameters** collection in the Command class to pass parameters to stored procedures or SQL statements. When you set CommandText & Connection property, now you can execute the command object by following two ways.

First method is, you can assign name of command object to SelectCommand / UpdateCommand / InsertCommand / DeleteCommand property of Data Adapter & execute its SQL with the Data Adapter's Fill method.

OR

Second method is, You can also use the command's object built-in method to execute commands in a database. (No Data Adapter is needed). (ExecuteReader, ExecuteNonQuery, ExecuteScalar.

The .NET Framework Data Provider for OLE DB includes an **OleDbCommand object**, the .NET Framework Data Provider for SQL Server includes a **SqlCommand object**, the .NET Framework Data Provider for ODBC includes an **OdbcCommand object**, and the .NET Framework Data Provider for Oracle includes an **OracleCommand object**.

- **Property :**
  - **(1) CommandText :** It is the string, contains either SQL statements or name of the Stores procedure to be executed.
  - **(2) CommandType :** It represents the type of the **Command** object. Depending upon the command type Command object executes the command. The different command types are as follows.

    StoredProcedure : The name of a stored procedure.
    TableDirect : The name of a table.
    Text : SQL statements. (Default.)
    For Ex. Cmd.CommandType = CommandType.**Text**
  - **(3) Connection :** The name of the active Connection object, through which the command is to be executed.
  - **(4) Parameters :** Gets the command parameters. The parameters property contains a collection of parameters for the SQL statements or stored procedure.
- **Methods :**
  - **(1) ExecuteNonQuery :** Executes commands that do not return data rows. But it returns no of rows affected by the commands. (such as **SQL INSERT, DELETE, UPDATE,** and **SET** statements).
  - **(2) ExecuteScalar :** Calculates and returns a single value, such as a sum, min, max from a database. Used for aggregate function.
  - **(3) ExecuteReader :**Executes SQL commands that return rows. Note that this method does not return a dataset, it creates a data reader, which is much simpler.
  - **(4) Cancel :** Cancels the execution of the command.

Ex. 1 :
```
Imports System.Data.OleDb
Dim Con As New OleDbConnection
Dim Cmd As OleDbCommand
Con.ConnectionString = "Provider=MSDAORA;Data Source= OMSAI1;User
ID=SCOTT; PASSWORD=TIGER"
Con.Open
Dim SAI_STR As String = "Insert Into Employee Values (....."
Cmd = New OleDbCommand(SAI_STR, Con)
Cmd.ExecuteNonQuery()
```

OR

Ex. 2 :
```
Imports System.Data.OleDb
Public Con As New OleDbConnection
Dim Cmd As OleDbCommand
Con.ConnectionString = "Provider=MSDAORA;Data Source=
OMSAI1;UserID=SCOTT;PASSWORD=TIGER"
    Con.Open
    Dim SQL_Str As String = "Insert Into Employee Values(....."
    Cmd = New OleDbCommand( )
    Cmd.Connection = Con
    Cmd.CommandText = SQL_Str
    Cmd.CommandType = CommandType.Text
    Cmd.ExecuteNonQuery( )
```

[3] Data Adapter Objects : The DataAdapter object provides the bridge between the DataSet object and the data source (database) for retrieving and saving data.

The DataAdapter's sole purpose is to retrieves data from the database, then populates (fill) the Datasets & also used to send (propagate) the Datasets changes to the database. (The **DataAdapter** object has **Fill** method to load data from the data source into the dataset, and the **Update** method to send changes you've made in the dataset back to the data source).

The DataAdapter contains four command objects: SelectCommand, InsertCommand, UpdateCommand, and DeleteCommand. The DataAdapter uses the SelectCommand to fill a DataSet & the remaining three commands to transmit changes back to the data source.

The .NET Framework Data Provider for OLE DB includes an **OleDbDataAdapter** object; the .NET Framework Data Provider for Oracle includes an **OracleDataAdapter** object. You can assign name of command object to SelectCommand / UpdateCommand / InsertCommand / DeleteCommand property of Data Adapter & execute its SQL with the Data Adapter's Fill method.

- **Property :**
  (1) SelectCommand : The name of the Command object used to retrieve rows from the data source.
  (2) InsertCommand : The name of the Command object used to insert rows in the data source.
  (3) UpdateCommand : The name of the Command object used to update rows in the data source.
  (4) DeleteCommand : The name of the Command object used to delete rows in the data source.

- **Methods :**
  (1) Fill : The **Fill** method which loads data from the data source (database) into the Dataset. If the **Connection** is closed before **Fill** is called, it is opened to retrieve data and then closed.
  Syntax 1 : OleDbDataAdapter.Fill(DataSet)
  Syntax 2 : OleDbDataAdapter.Fill(DataTable)
  Syntax 3 : OleDbDataAdapter.Fill(DataSet, TableName)
  (2) Update : Update method is used to send changes you've made in the dataset back to the data source (database).
  Syntax 1 : OleDbDataAdapte.Update(DataSet)
  Syntax 2 : OleDbDataAdapter.Update(DataTable)

Ex.
```
Imports System.Data.OleDb
Dim Con As New OleDbConnection
Dim DA As OleDbDataAdapter
Dim DT As DataTable
Con.ConnectionString = "Provider=MSDAORA;Data Source=
OMSAI1;User ID=SCOTT;PASSWORD=TIGER"
Con.Open
DA = New OleDbDataAdapter("SELECT * FROM STUD", CON)
DT = New DataTable
DA.Fill(DT)
DataGridView1.DataSource = DT
```

[4] DataSet Objects : It is the major component of ADO.NET. DataSet is a memory-resident representation of data. **A dataset is a disconnected cache of data, and, that is stored in memory.** DataSet is always disconnected from the data source. It can contain data from multiple sources. **The dataset is a disconnected, in-memory representation of data.**

As shown in figure of ADO.NET object model, The DataSet composed of two primary objects: the **DataTableCollection**, accessed through Tables property, and **DataRelationCollection** accessed through the Relations property. The **DataTableCollection** contains zero or more DataTable objects, which are in turn

made up of three collections: DataColumnColection, DataRowCollection, and ConstraintCollection. The DataRelationCollection contains zero or more DataRelation objects.

- **Property :**
  (1) **Namespace :** Gets/sets the namespace of the dataset.
  (2) **Relations :** It is the collection of DataRelation objects, which defines the relationship of the DataTables within the dataset.
  (3) **Tables :** It is the collection of DataTables contained in the dataset.

- **Method :**
  (1) **AcceptChanges :** Accepts (Commits) all the pending changes made to the dataset.
  (2) **RejectChanges :** Roll back all changes pending in the DateSet. Rolls back the changes made to the dataset since it was created or since the **AcceptChanges** method was called.
  (3) **Copy :** Copies the structure & contents of the DataSet.
  (4) **Clear :** Empties all the tables in the DataSet.
  (5) **Reset :** Returns the DataSet back to its original state.
  (6) **CreateDataReader :** Returns a DataTableReader from the DataSet, allowing you to read-only, forward-only access to the data. The DataTableReader is functionally identical DataReader.

Ex.
```
Imports System.Data.OleDb
Dim Con As OleDbConnection
Dim DA As OleDbDataAdapter
Dim DS as  DataSet
Con.ConnectionString = "Provider=MSDAORA;Data Source= OMSAI1;User
ID=SCOTT; PASSWORD=TIGER"
Con.Open
DA = New OleDbDataAdapter("SELECT * FROM STUD", CON)
DS = New DataSet( )
DA.Fill(DS,"STUD")
DataGridView1.DataSource = DS
DataGridView1.DataMember="STUD"
```

[5] **DataReader Objects :** The **DataReader** in addition to datasets, there are also data readers, which are extremely fast, read-only, forward only low-overhead way of retrieving information from the database. You can only move through records with in ascending order means you can not go backward. It provides a high-performance stream of data from the data source (database). If we only need single direction, read only loading of DataSets or controls, we can bypass the Adapter entirely & utilize a DataReader. DataReader cannot be created directly in code; they are created only by calling the **ExecuteReader** method of a Command object.

DataReader is almost always used in conjunction with a Data **Command**. The only exception is the **TableReader** object, a variation of the **DataReader** that is used with the DataSet.

DataReader is appropriate when you are processing rows individually and then discarding them. For example, Report generators, you get a performance benefit from **DataReader. DataReader** is best suited for retrieving huge amounts of data, as the data is not cached in the memory.

- **Property :**
  - **(1) FieldCount :** Gets the number of columns in the current row.
  - **(2) HasRows :** Returns a Boolean value indicating whether the DataReader contains rows of data.
  - **(3) IsClosed :** Indicates whether the DataReader is closed.
  - **(4) Item :** Gets the value of a column(field). For example Select EmpNo, EmpName & City from Employee, then EmpNo is Item(0), EmpName is Item(2) & City is Item(3).
- **Method :**
  - **(1) Close :** Closes the data reader.
  - **(2) GetName :** Gets (returns)the name of the specified column.
  - **(3) GetValue :** Gets a field's value (column's value) in its native format.
  - **(4) GetValues :** Gets all columns in the current row.
  - **(5) IsDBNull :** Indicates if a column contains nonexistent (or missing) values.

Ex. Create Emp Table in Oracle. Emp(EmpNo, EmpName, City). We want to display only the name of all employees into the ListBox1, so we have to write code as follows.

```
Imports System.Data.OleDb
Public Class Form1
        Dim Con As New OleDbConnection
        Dim Cmd As OleDbCommand
        Dim DR As OleDbDataReader
    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
    System.EventArgs) Handles MyBase.Load
        Con.ConnectionString = "Provider=MSDAORA;Data
        Source=OMSAI1;User ID = SCOTT; PASSWORD =TIGER"
        Con.Open( )
        Cmd = New OleDbCommand("SELECT EmpNo, EmpName, City
        FROM EMP", Con)
        DR = Cmd.ExecuteReader( )
        'Now we want to add all Emp. Name into the ListBox1
        While DR.Read
            ListBox1.Items.Add(DR.Item(1))
        End While
    End Sub
End Class
```

[6] **DataTable Objects :** Data can be stored in memory in DataTable objects. (Datasets are made up of **DataTable** objects.) Data in the DataSet is stored in the form of DataTable Objects.

The DataTableCollection contains zero or more DataTable objects, which are in turn made up of three collections: DataColumnColection, DataRowCollection, and ConstraintCollection (used to ensure integrity of data , ForeignKeyConstraint & UniqueConstraint)

[7] **DataRow Objects : DataRow** objects represent rows in a **DataTable** object. You use **DataRow** objects to get access to, insert, delete, and update the records in a table.

[8] **DataColumn Objects : DataColumn** objects represent the columns, that is, the fields, in a data table.

[9] **DataRelation Objects :** To make those relations explicit, you have to use **DataRelation** objects. The **DataRelation** class supports data relations between data tables.

4.4 DATA ACCESS COMPONENTS OR DATA COMPONENTS

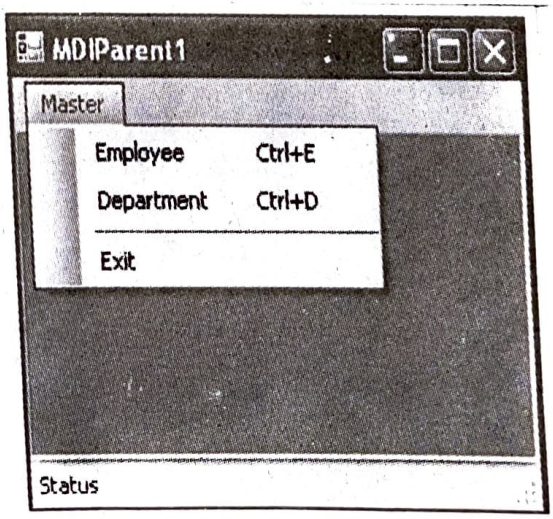
There are three main data access components in Visual Basic 2005 that you need for retrieving & viewing data from the database:

[1] BindingSource [2] TableAdapter [3] DataSet

Practical 17 : Design Menu driven application with Add New, Save, Modify, Delete, Search facility with Oracle database.Oracle Table: Emp(EmpNo Primary Key, EmpName, City)

Step 1 : Insert MDIParent Form. (Click on Add windows forms from the Project menu, then select MDI Parent Form.

Step 2 : Desing following Menu on the MDIParent1 Form as shown below.



Step 3 : Now take Form1 as a Employee Form & write code in **EmployeeMenuItem click event** of the MDIParent1 as follows. Double click on Employee menu option (**EmployeeMenuItem)**

```
Form1.MdiParent = Me
Form1.Show()
```

Step 4 : Create table in Oracle as following.
Emp(EmpNo Primary Key, EmpName, City).

Step 5 : Design the Employee Form (Form1) as shown in figure.
Panel1 contains three Textbox & three Label controls.
Set **Text** property of TextBox1 control to TxtEmpNo
Set **Text** property of TextBox2 control to TxtEmpName
Set **Text** property of TextBox3 control to TxtCity

GropuBox1 contains four Buttons control, for Add New, Save, Modify & Delete operation.
Set **Text** property of Button1 control to CmdAddNew
Set **Text** property of Button2 control to CmdSave
Set **Text** property of Button3 control to CmdModify
Set **Text** property of Button4 control to CmdDelete
Set **Enable** property of CmdSave, CmdModify & CmdDelete buttons to **False**.

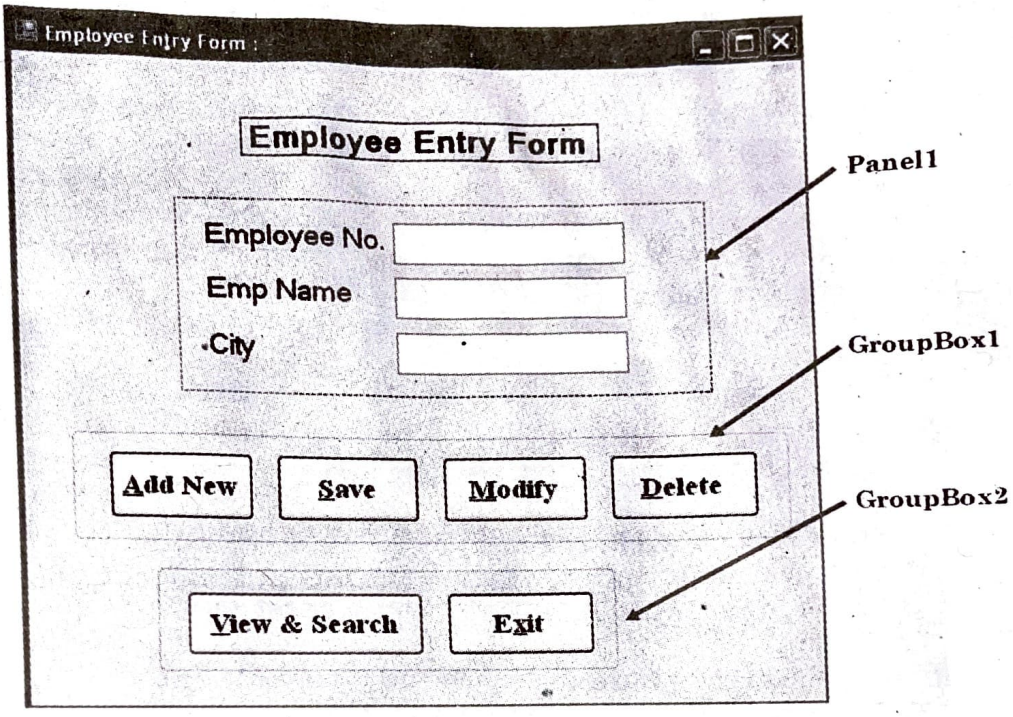
- When Add New button is click, disable the Add New button & enabled the Save button. Focus should be shifted to TxtEmpNo
- When Save button is click, disable the Save button & enabled the Add New button.

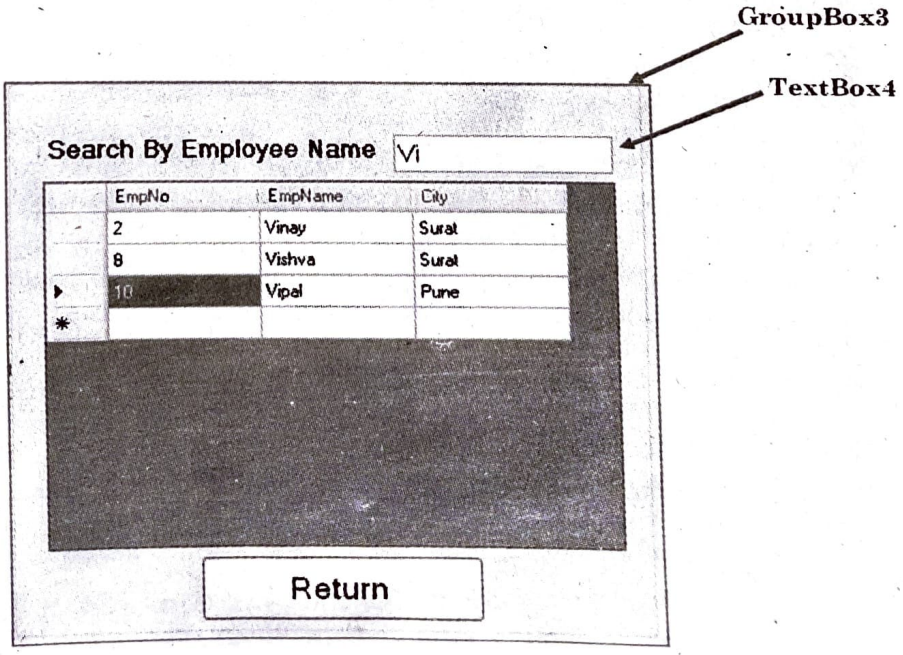
GropuBox2 contains two Buttons control, for "View & Search" & Exit Button.
Set **Text** property of Button5 control to CmdView
Set **Text** property of Button6 control to CmdExit

GropuBox3 contains one DataGridView control, one TextBox (TextBox4), one Label & one Button control as shown in figure.
Set **Text** property of TextBox4 control to TxtSearch
Set **Text** property of Button7 control to CmdReturn
Set **Visible** property of **GropuBox3** to False.

- When View & Search button is click, show **GropuBox3** (Make it visible). So user can view the data. If user wants to modify or delete the employee record, then user has to select the desire record form the DataGridView1, & Click on return button, so that selected record will be display into the TextBox controls. Hide the **GropuBox3** and enable the Modify, Delete buttons.

- When Modify button is click, perform modify operation & disable the Modify button.





```
'General Declaration Section
    Imports System.Data.OleDb            ' For MS Access, Old version of Oracle
    'Imports System.Data.SqlClient       ' For MS SQL Server Version 7.0 or Later
    'Imports System.Data.OracleClient    ' For Oracle Version 8.1.7 & Later

Public Class Form1
    Dim Con As New OleDbConnection
    Dim DA As New OleDbDataAdapter
    Dim DT As New DataTable
    Dim Cmd As New OleDbCommand
        'CREATE A SUB PROCEDURE TO FILL THE DATAGRIDVIEW
        Public Sub DataGridViewFill()
            DA = New OleDbDataAdapter("SELECT * FROM EMP", Con)
            Dim DT As New DataTable
            DA.Fill(DT)
            DataGridView1.DataSource = DT
        End Sub

'Coding of Form Load Event
    Con.ConnectionString = "Provider=MSDAORA;Data Source=OMSAI1;User ID=
    SCOTT; PASSWORD=TIGER"
    Con.Open()
'Coding of Add New Button
    TxtEmpNo.Clear()
    TxtEmpName.Clear()
    TxtCity.Clear()
    TxtEmpNo.Focus()
    CmdAddNew.Enabled = False
    CmdSave.Enabled = True

'Coding of Save Button
    Dim SQL_Str As String = "INSERT INTO EMP VALUES(" &
    Val(TxtEmpNo.Text) & ",'" & TxtEmpName.Text & "','" & TxtCity.Text & "')"
    Cmd = New OleDbCommand(SQL_Str, Con)
    Cmd.ExecuteNonQuery()
    CmdAddNew.Enabled = True
    CmdSave.Enabled = False

' Coding of View & Search Button
    GroupBox3.Visible = True
    DataGridViewFill()
' Seach By Employee Name
' Whenever any text changes in TextBox4 of GropuBox3, its employee ' name
should be search & Display in DataGridView1.
```

```
DA = New OleDbDataAdapter("SELECT * FROM EMP WHERE EMPNAME LIKE '" &
TxtSearch.Text & "%'", Con)
DT = New DataTable
DA.Fill(DT)
DataGridView1.DataSource = DT
```

'Coding of Return Button. Selected record from the DataGridView1 'will be display into the TextBox controls. Hide the GropuBox3 and 'enable the Modify, Delete buttons.

```
GroupBox3.Visible = False
Dim Current_Row_Index As Integer
Current_Row_Index = DataGridView1.CurrentRow.Index
TxtEmpNo.Text = DataGridView1.Rows(Current_Row_Index).Cells("EMPNO").
Value.ToString
TxtEmpName.Text = DataGridView1.Rows(Current_Row_Index).Cells("EMP
NAME").Value.ToString
TxtCity.Text = DataGridView1.Rows(Current_Row_Index).Cells("CITY"). Value.ToString
CmdModify.Enabled = True
CmdDelete.Enabled = True
```

'Coding of Modify Button

```
Dim SQL_Str As String = "UPDATE EMP SET EMPNO=" & Val(TxtEmpNo.Text) &
",EMPNAME='" & TxtEmpName.Text & "',CITY='" & TxtCity.Text & "' WHERE
EMPNO=" & Val(TxtEmpNo.Text)
    Cmd = New OleDbCommand(SQL_Str, Con)
    Cmd.ExecuteNonQuery()
    CmdModify.Enabled = False
    CmdDelete.Enabled = False
```

'Coding of Delete Button

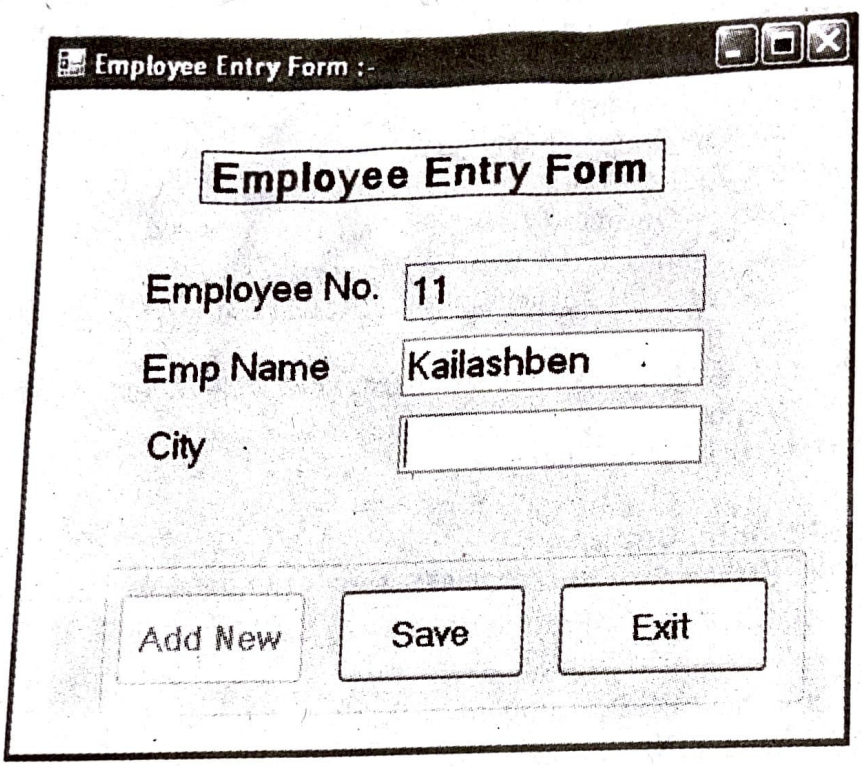
```
Dim SQL_Str As String = "DELETE FROM EMP WHERE EMPNO=" & Val(TxtEmpNo.Text)
Cmd = New OleDbCommand(SQL_Str, Con)
Cmd.ExecuteNonQuery()
CmdModify.Enabled = False
CmdDelete.Enabled = False
```

'Coding of Exit Button

```
Dim Result As Integer
Result = MsgBox("Do you want to Exit?", MsgBoxStyle.YesNo + MsgBoxStyle. Question
+ MsgBoxStyle.SystemModal + MsgBoxStyle.DefaultButton2, "Quit")
If (Result = MsgBoxResult.Yes) Then
    Me.Close()
End If
End Class
```

Practical 18 : **Design application with Add New, Save facility with Oracle database. Employee No should be generated automatically. Oracle Table: Emp(EmpNo Primary Key, EmpName, City)**

This practical is very similar to previous, but here we have to generate Emp No. automatically & in this parctical we are only taking Add New & Save facility.



- Create table in Oracle as following.
  Emp(EmpNo Primary Key, EmpName, City).

- Design interface as shown in figure.
  Set **Text** property of TextBox1 control to TxtEmpNo
  Set **Text** property of TextBox2 control to TxtEmpName
  Set **Text** property of TextBox3 control to TxtCity

Set **ReadOnly** property of TextBox1 (TxtEmpNo) control to True, because user is not going to enter employee no. When Add New Button is click, employee no. should be appear automatically & focus should be shifted to TxtEmpName. Disable the Add New button & Enable the Save button.

```
'General Declaration Section
Imports System.Data.OleDb            ' For MS Access, Old version of Oracle
'Imports System.Data.SqlClient       ' For MS SQL Server Version 7.0 or Later
'Imports System.Data.OracleClient    ' For Oracle Version 8.1.7 & Later
```

```
Public Class Form1
    Dim Con As New OleDbConnection
    Dim DA As New OleDbDataAdapter
    Dim DT As New DataTable
    Dim Cmd As New OleDbCommand
'Coding of Form Load Event
    Con.ConnectionString = "Provider=MSDAORA;Data Source=OMSAI1;User ID =
    SCOTT; PASSWORD=TIGER"
    Con.Open( )

'Coding of Add New Button
' In SQL Query we have selected the employee no. by descending order,
' If there is no record in the DataTable,
' then set emp no. to 1 else set last emp no. plus 1
    DA = New OleDbDataAdapter("SELECT EMPNO FROM EMP ORDER
    BY EMPNO DESC", Con)
    DT = New DataTable
    DA.Fill(DT)
    If DT.Rows.Count = 0 Then
        TxtEmpNo.Text = 1
    Else
        TxtEmpNo.Text = Val(DT.Rows(0).Item(0).ToString) + 1
    End If
    TxtEmpName.Clear()
    TxtCity.Clear()
    TxtEmpName.Focus()
    CmdAddNew.Enabled = False
    CmdSave.Enabled = True

'Coding of Save Button
    Dim SQL_Str As String = "INSERT INTO EMP VALUES(" & Val(TxtEmpNo.Text)
    & ",'" & TxtEmpName.Text & "','" & TxtCity.Text & "')"
    Cmd = New OleDbCommand(SQL_Str, Con)
    Cmd.ExecuteNonQuery()
    CmdAddNew.Enabled = True
    CmdSave.Enabled = False
'Coding of Exit Button
    Dim Result As Integer
    Result = MsgBox("Do you want to Exit?", MsgBoxStyle.YesNo + MsgBoxStyle.Question
    + MsgBoxStyle.SystemModal + MsgBoxStyle. DefaultButton2, "Quit")
    If (Result = MsgBoxResult.Yes) Then
        Me.Close()
    End If
End Class
```

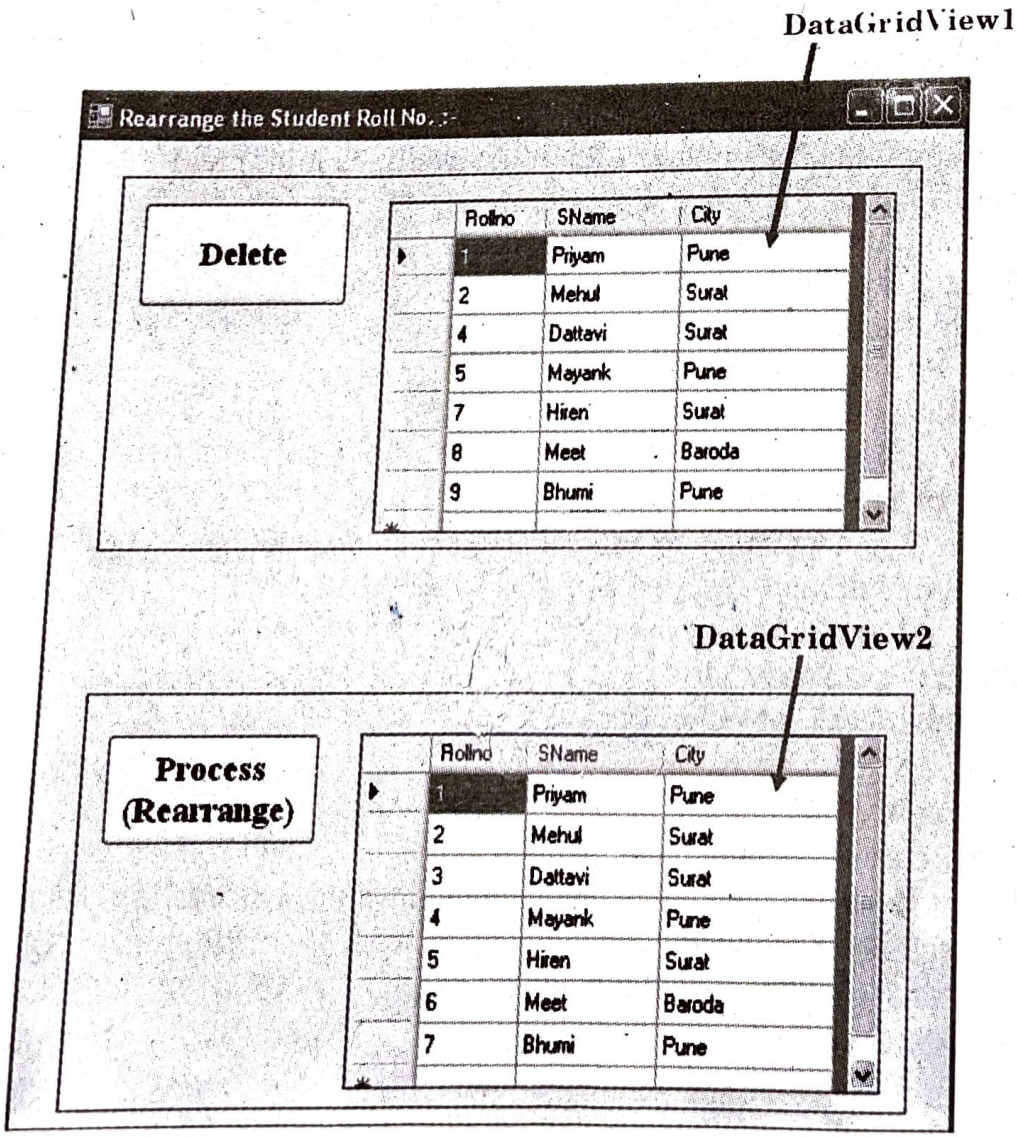
Practical 19 : **Create STUD table with fields (Rollno Primary Key, SName, City) in Oracle database. Delete some records from STUD table. When Process (Rearange) button is click, arrange the rollno in sequance.**

Create STUD table with fields (Rollno Primary Key, SName, City) in Oracle Now suppose we have 1 to 9 students in Stud Table.

When Delete button is click, selected record from the DataGridView1 should be deleted,. For example **we have deleted roll no. 3 & 6 as shown in DataGridView1.**

When Process (Rearange) button is click, New Arrange roll no is shown in DataGridView2.

We have declare variable "c" as integer. The Individual records of DataTable is process, & "c" variable is incremented every time. Old RollNo is updated by new value of "c".



```
'General Declaration Section.
    Imports System.Data.OleDb                    ' For MS Access, Old version of Oracle
    'Imports System.Data.SqlClient               ' For MS SQL Server Version 7.0 or Later
    'Imports System.Data.OracleClient            ' For Oracle Version 8.1.7 & Later
Public Class Form1
    Dim Con As New OleDbConnection
    Dim DA As New OleDbDataAdapter
    Dim DT As New DataTable
    Dim Cmd As New OleDbCommand

'CREATE A SUB PROCEDURE TO FILL THE DATAGRIDVIEW.
    Public Sub DataGridViewFill()
        DA = New OleDbDataAdapter("SELECT * FROM STUD ORDER BY ROLLNO",
        Con)
        DT = New DataTable
        DA.Fill(DT)
        DataGridView1.DataSource = DT
    End Sub

'Coding of Form Load Event.
    Con.ConnectionString = "Provider=MSDAORA;Data Source=OMSAI1;User ID =
    SCOTT; PASSWORD=TIGER"
    Con.Open( )
    DataGridViewFill( )

'Coding of Delete Button (Selected record from the DataGridView1 should be
deleted)
    Dim CurrentRowIndex As Integer = DataGridView1.CurrentRow.Index
    Dim RNO As Integer = DataGridView1.Rows(CurrentRowIndex).Cells
    ("Rollno").Value.ToString
    Dim SQL_STR As String = "DELETE FROM STUD WHERE ROLLNO=" & RNO
    Cmd = New OleDbCommand(SQL_STR, Con)
    Cmd.ExecuteNonQuery()
    'Display in DataGridView1.
    DataGridViewFill()

'Coding of Process (Rearrange) button.
    Dim SQL_Str As String = "SELECT * FROM STUD ORDER BY ROLLNO"
    DT = New DataTable
    DA = New OleDbDataAdapter(SQL_Str, Con)
    DA.Fill(DT)
    Dim C As Integer = 0
    For I As Integer = 0 To DT.Rows.Count - 1
        C = C + 1
```

```
        Dim SQL_Str1 As String = "UPDATE STUD SET ROLLNO=" & C & " WHERE
        ROLLNO=" & DT.Rows(I).Item(0).ToString()
        Cmd = New OleDbCommand(SQL_Str1, Con)
        Cmd.ExecuteNonQuery()
    Next
    'Now, display the rearranged records in the DataGridView2.
    DA = New OleDbDataAdapter("SELECT * FROM STUD ORDER BY ROLLNO", Con)
    DT = New DataTable
    DA.Fill(DT)
    DataGridView2.DataSource = DT
End Class
```

[1] ImageList
[2] TreeView Control
[3] ListView control
[4] Tool Strips Control
[5] StatusStrip Control
[6] Progress Bar
[7] Tab Control

[1] ImageList : ImageList component is used to store (collect) images, these images can then be used by other controls, such as the ListView, TreeView, or ToolBar, CheckBox, RadioButton, Label Control etc.

The controls such as list views, tree views, toolbars, tab controls, checkboxes, buttons, radio buttons, and labels have **ImageList** and **ImageIndex** property.

For example ImageList property of Button control is used to get or set the name of ImageList control from where images to be used.

For example ImageIndex property of Button control is used to get or set the index no. (which image from the ImageList control should be display in to the button control). Setting ImageIndex to 0 will display the first image from the ImageList.

You also can use image lists with controls that only have **Image** or **BackgroundImage** properties. For example, picture boxes only have an Image property, so to load an image from an image list into a picture box, you have to access images in the list using the Images collection. For example loading image 0 from ImageList1 into a picture box, we have to code as follows.

```
PictureBox1.Image = ImageList1.Images(0)
```

- **Property :**

(1) **Images :** This property represents the collection of images that are contained in the ImageList. Images can be added & removed at design time as well as run time. Each individual image can be accessed by its index value or by its key. At design time when you click **images** property in the property window, an Image Collection Editor dialog Box will be open, & you can add new images to the image list by clicking the **Add button** or remove them by clicking the **Remove button**.

(2) **ImageSize :** Gets or sets the size of the images in the image list. The Size that defines the **height** and **width**, in pixels, of the images in the list. The default size is 16 by 16. The maximum size is 256 by 256.

Ex. :

```
' Construct the ImageList.
ImageList1 = New ImageList
' Set the ImageSize property to a larger size (the default is 16 x 16).
ImageList1.ImageSize = New Size(112, 112)
' Add two images to the list.
ImageList1.Images.Add(Image.FromFile("c:\Photo\OM.bmp"))
ImageList1.Images.Add (Image.FromFile("C:\Photo\SAI.bmp"))
```

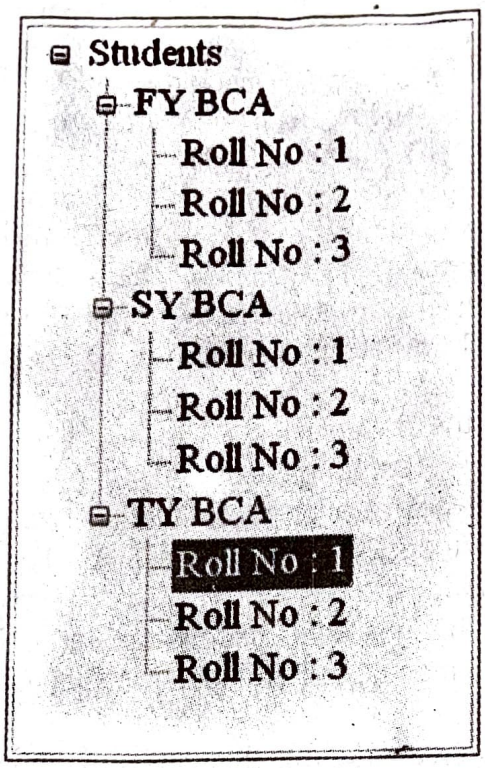
[2] **TreeView Control :** TreeView control is used to display hierarchy nodes. (Nodes are in the form text or data arranged in a hierarchical order). An example of TreeView control is the left pane of the Windows Explorer.

We can also display a tree view with checkboxes next to the nodes, if the tree view's **CheckBoxes** property is set to **True**. You can then check or uncheck nodes by setting the node's **Checked** property to **True** or **False**.

Nodes can be added & removed at design time as well as run time. At design time when you click **nodes** property in the property window, the **TreeNode Editor dialog Box** will be open, & you can add new nodes to the TreeView control by clicking the **Add Root or Add Child Button** & remove them by clicking the **Remove button**. At run time we can use the **Nodes** collection's **Add** or **Remove** methods to add or remove nodes in code.

Tree views also support various properties for navigating through them, node by node. In particular, you can use the following properties: **FirstNode**, **LastNode**, **NextNode**, **PrevNode**, **NextVisibleNode**, **PrevVisibleNode**.

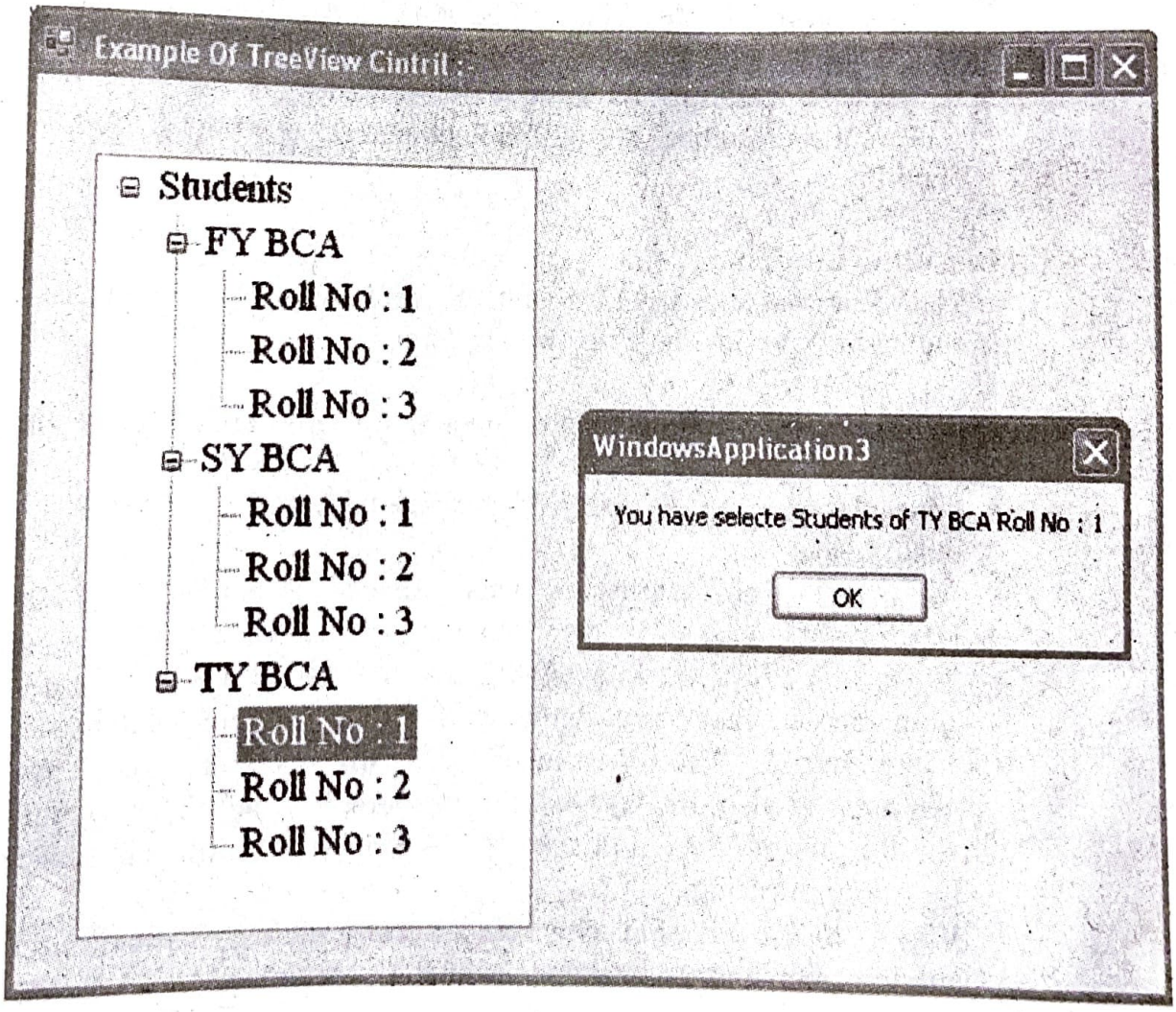
The user can expand the **TreeNode** by clicking the plus-sign (+) button, if one is displayed next to the **TreeNode**, or you can expand the **TreeNode** by calling the TreeNode.Expand method. To expand all the child tree node levels in the Nodes collection, call the ExpandAll method. You can collapse the child **TreeNode** level by calling the TreeNode.Collapse method.



- **Property :**
  - **(1) CheckBoxes :** Gets/sets whether checkboxes should be displayed next to tree nodes.
  - **(2) HotTracking :** Gets/sets whether a tree node label should change its appearance when the mouse pointer moves over it.
  - **(3) ImageList :** Gets/sets the name of the ImageList control used for the tree view control.
  - **(4) ImageIndex :** It is used to get or set the index no. (which image from the ImageList control should be display in to the button control).
  - **(5) SelectedImageIndex :** Gets/sets the image index for the image to display when a node is selected.
  - **(6) LabelEdit :** Gets/sets whether tree node text can be edited.
  - **(7) Nodes :** Gets the collection of tree nodes. It represents collection of all the nodes in the TreeView control. At design time you can use this property to add / remove the root & child nodes.
  - **(8) SelectedNode :** Gets/sets the node that is selected. It represents the list of currently selected node in the TreeView control.
  - **(9) PathSeparator :** Gets/sets the string the tree node uses as a path delimiter.The default is\.
  - **(10) Sorted :** Gets/sets if the tree nodes should be sorted.

- **Method :**
  (1) CollapseAll : Collapses all the tree nodes. When all the nodes in the tree are collapsed, only the parent nodes are displayed.
  (2) ExpandAll : Expands all the tree nodes. When the nodes in the tree are expanded all the parent nodes & child nodes of each parent node are displayed.
- **Event :**
  (1) AfterSelect : It is the default event of TreeView control. It occurs (executes / fires) when a tree node is selected.
  (2) BeforeSelect : It occurs (executes / fires) before a node is selected.
  (3) AfterCollapse : It occurs (executes) when a tree node is collapsed.
  (4) BeforeCollapse : It occurs (executes) before a node is collapsed.
  (5) AfterExpand : Occurs when a tree node is expanded.
  (6) BeforeExpand : Occurs before a node is expanded.
  (7) AfterCheck : Occurs when a node checkbox is checked.
  (8) BeforeCheck : Occurs before a node checkbox is checked.

TreeNode Class : Tree views display nodes in a hierarchical structure, and each node in a tree view is a TreeNode object.
Ex. :



```
' Coding of Form Load Event
    TreeView1.Nodes.Clear()
    TreeView1.Nodes.Add("OM1Stud", "Students")
        TreeView1.Nodes("OM1Stud").Nodes.Add("FY BCA")
    TreeView1.Nodes("OM1Stud").Nodes(0).Nodes.Add("Roll No : 1")
    TreeView1.Nodes("OM1Stud").Nodes(0).Nodes.Add("Roll No : 2")
    TreeView1.Nodes("OM1Stud").Nodes(0).Nodes.Add("Roll No : 3")

    TreeView1.Nodes("OM1Stud").Nodes.Add("SY BCA")
    TreeView1.Nodes("OM1Stud").Nodes(1).Nodes.Add("Roll No : 1")
    TreeView1.Nodes("OM1Stud").Nodes(1).Nodes.Add("Roll No : 2")
    TreeView1.Nodes("OM1Stud").Nodes(1).Nodes.Add("Roll No : 3")

    TreeView1.Nodes("OM1Stud").Nodes.Add("TY BCA")
    TreeView1.Nodes("OM1Stud").Nodes(2).Nodes.Add("Roll No : 1")
    TreeView1.Nodes("OM1Stud").Nodes(2).Nodes.Add("Roll No : 2")
    TreeView1.Nodes("OM1Stud").Nodes(2).Nodes.Add("Roll No : 3")

' Coding of TreeView1_AfterSelect Event
    If e.Node.Level = 2 Then
        Dim RootNode As String = TreeView1.SelectedNode.Parent.Parent. Text
        Dim N1 As String = TreeView1.SelectedNode.Parent.Text
        Dim SelectedNode As String = e.Node.Text
        MsgBox("You have selecte " & RootNode & " of " & N1 & " " & SelectedNode)
    End If
```

[3] ListView Control :

A **ListView** control is used to display a list of items. Example of ListView control is the right pane in the Windows Explorer.

ListView control displays a list of items in a five different views. **View** property is used to set views mode.

- **View.LargeIcon :** It dislays items with large size icons (32×32 pixels) with a label below it.
- **View.SmallIcon :** It dislays items with a small icon (6×16 pixels) with a label to its right.
- **View.List :** It dislays items with a small icon with a label to its right. Each items are arranged in columns wise with no column headers.
- **View.Details :** The report mode (or details mode) displays items in multiple columns. It also displays column headers. The left-most column contains a small icon and label, and subsequent columns contain sub items as specified by the application.
- **View.Tile :** Each item appears as a full-sized icon with the item label and subitem information to the right of it.

The key property of the **ListView** control is **Items**, which contains the items displayed by the control. The **SelectedItems** property contains a collection of the items currently selected in the control. You can sort the items in a list view with the **Sorting** property

- **Property :**
  (1) AllowColumnReorder : Gets/sets whether the user may drag column headers.
  (2) Columns : Gets a **collection** of columns. It represents all the columns in the ListView control Columns can be added & removed at design time as well as run time. At design time **columns** property is used to add or remove columnHeader by using the **ColumnHeader Collection Editor dialog Box**.
  (3) GridLines : Gets/sets whether grid lines are drawn between items and their subitems.
  (4) Items : Gets the list items. It represents all the items in the ListView control. Items can be added & removed at design time as well as run time. At design time **Items** property is used to add or remove Items by using the ListViewItem Collection Editor dialog Box.
  (5) SelectedIndices : Gets the indices of the selected items.
  (6) SelectedItems : Gets the selected items.
  (7) Sorting : Gets/sets the sort order for the items. Ascending, Descending or None.
  (8) View : Gets/sets the current view mode. LargeIcon, SmallIcon, Detail, List, or Tile.

- **Method :**
  (1) Clear : Removes all items from the **list view**.

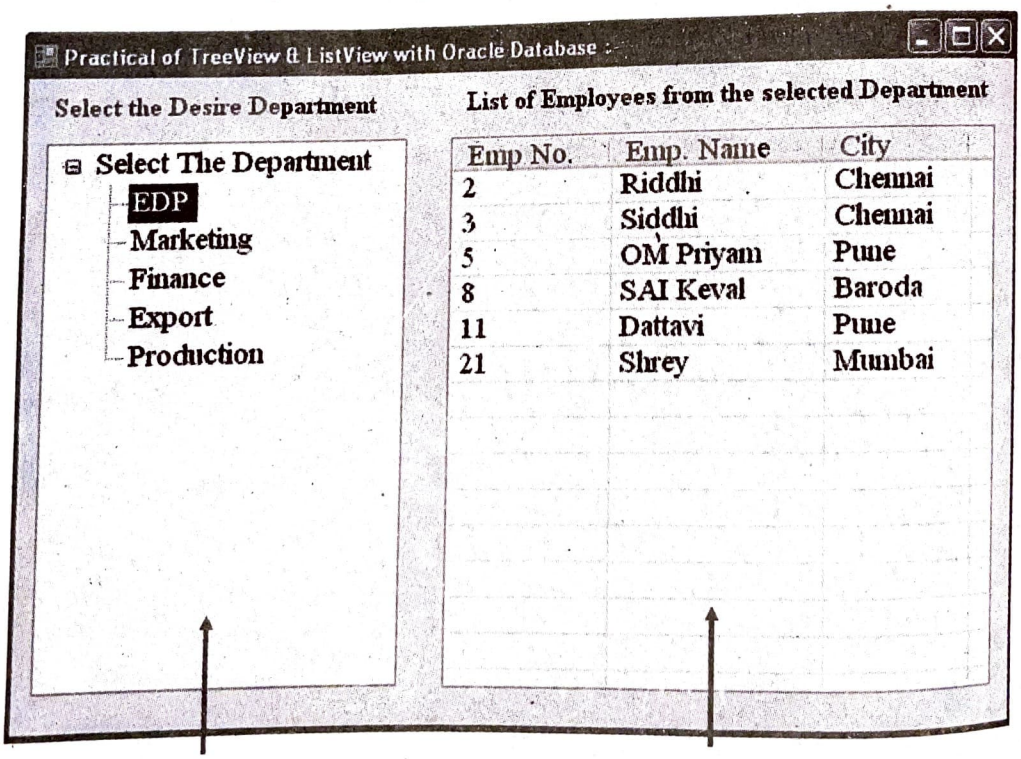
- **Event :**
  (1) SelectedIndexChanged : It is the default event of ListView control. It occurs (fires) when the selected index changes.
  (2) ColumnClick : This event is occurs (executes) when a column header is clicked

Ex. :
```
ListView1.View = View.Details
ListView1.Columns.Add("EmpNO", 100)
ListView1.Columns.Add("Name", 100)
ListView1.Columns.Add("City", 100)
ListView1.Items.Add("1")
ListView1.Items(0).SubItems.Add("Rahim")
ListView1.Items(0).SubItems.Add("Goa")
ListView1.Items.Add("2")
ListView1.Items(1).SubItems.Add("Ram")
ListView1.Items(1).SubItems.Add("Baroda")
```

Output :

EmpNo	Name	City
1	Rahim	Goa
2	Ram	Baroda

Practical 20 : **Fill the TreeView control with DeptName from Dept Table. Whenever any DeptName is selected from TreeView1. List of employees from the selected departmet should be display in the ListView control. Create two table in Oracle database as shown below.**
Oracle Table 1 : Dept(DeptNo Primary Key, DeptName)
Oracle Table 2 : Emp(EmpNo Primary Key, EmpName, City, DeptNo Foreign Key)



TreeView1 **ListView1**

Step 1 : Create two table in Oracle Database.
Oracle Table 1 : Dept(DeptNo Primary Key, DeptName)
Oracle Table 2 : Emp(EmpNo Primary Key, EmpName, City, DeptNo Foreign Key)

Step 2 :

```
'General Declaration Section
    Imports System.Data.OleDb        ' For MS Access, Old version of Oracle
    'Imports System.Data.SqlClient   ' For MS SQL Server Version 7.0 or Later
    'Imports System.Data.OracleClient  ' For Oracle Version 8.1.7 & Later

Public Class Form1
    Dim Con As New OleDbConnection
    Dim DA As New OleDbDataAdapter
    Dim DT As New DataTable
    Dim Cmd As New OleDbCommand

'Coding of Form Load Event
    Con.ConnectionString = "Provider=MSDAORA;Data Source=OMSAI1; User
    ID=SCOTT; PASSWORD=TIGER"
    Con.Open()
    'Fill the TreeView control with DeptName from Dept Table.
    TreeView1.Nodes.Clear()
    TreeView1.Nodes.Add("OMDept", "Select The Department")
    DA = New OleDbDataAdapter("Select DeptName from Dept" & """", Con)
    Dim DT As New DataTable
    DA.Fill(DT)
    If DT.Rows.Count = 0 Then
        MsgBox("There is no records exist in the Department table.")
    Else
        For I As Integer = 0 To DT.Rows.Count - 1
            TreeView1.Nodes("OMDept").Nodes.Add(DT.Rows(I).Item(0). ToString)
        Next I
    End If

    'Design the ListView control's column Header.
    ListView1.View = View.Details
    ListView1.GridLines = True
    ListView1.Columns.Add("Emp No.", 100)
    ListView1.Columns.Add("Emp. Name ", 130)
    ListView1.Columns.Add("City", 90)

'Coding of TreeView1_AfterSelect Event
    If TreeView1.SelectedNode.Text <> "Select The Department" Then
        Dim SAI_SQL As String
```

```
        SAI_SQL = "SELECT EMP.EmpNo, EMP.EmpName, EMP.City FROM DEPT, EMP
        WHERE  DEPT.DeptNo=Emp.DeptNo    AND  DEPT.DeptName='" &
        e.Node.Text.Trim & "'"
        DA = New OleDbDataAdapter(SAI_SQL, Con)
        Dim DT1 As New DataTable
        DA.Fill(DT1)
        If DT1.Rows.Count = 0 Then
            MsgBox("There is no employees exist in the selected Department.")
        Else
            ListView1.Items.Clear()
            For I As Integer = 0 To DT1.Rows.Count - 1
                ListView1.Items.Add(DT1.Rows(I).Item(0).ToString)
                ListView1.Items(I).SubItems.Add(DT1.Rows(I).Item(1). ToString)
                ListView1.Items(I).SubItems.Add(DT1.Rows(I).Item(2). ToString)
            Next I
        End If
    End If
End Class
```

[4] Tool Strips Control or Toolbar Control :

The ToolStrip and its associated classes enable you to create toolbars. The toolbar is generally situated immediately below the menu bar & has become a standard feature in many windows applications. Toolbars contain toolbar buttons, which provides quick access to the most frequently used menu commands in an application. An application can have multiple toolbars. The **Items** property of the tool strip control allows a user to add items to the tool strip. Following types of items you can add into the tool strip. For example

- Push Buttons.
- Drop-down buttons (that can display a drop down menu),
- Split buttons (that are a combination of drop-down buttons & normal push button),
- Separators which display empty horizontal space to separate other buttons &
- Other controls, like combo boxes, textboxes & progress bars.

When **ShowItemToolTips** property is set to true, Toolbars can display tool tips when the user points the mouse pointer at a toolbar button.

- **Property :**

  (1) Items : It is the **Items** collection used to retrieve all the items that have been added to the **ToolStrip**. The DisplayedItems property is used to retrieve only the items that are currently displayed on the **ToolStrip**. Items can be added & removed at design time as well as run time. At design time **Items** property is used to add or remove items by using an Items Collection Editor dialog Box.

(2) **Orientation :** Gets the orientation of the ToolStripPanel. Horizontal or Vertical (By default).

(3) **ShowItemToolTips :** When **ShowItemToolTips** property is set to true, Toolbars can display tool tips when the user points the mouse pointer at a toolbar button.

(4) **ImageList :** Gets or sets the name of image list control that contains the images to be displayed on a **ToolStrip** item.

- **Event :**

(1) **ItemClicked** : It is the default event of ToolStrip control. It occurs (executes / fires) when the **ToolStripItem** is clicked.

(2) **ItemAdded :** It occurs when a new ToolStripItem is added to the ToolStripItemCollection.

- **ToolStripItem Class :**

The items in a **ToolStrip** are objects of the **ToolStripItem class**, and the **collection** of those items is stored in the ToolStrip's **Items** property.

- **Property :**

(1) **DisplayStyle :** Gets or sets whether text and images are displayed on a **ToolStripItem**. It can be **Image** or **ImageAndText** or **None** or **Text.** By default is **Image.**

(2) **Image :** Gets or sets the image that is displayed on a **ToolStripItem**.

(3) **ImageAlign** (4) **ImageIndex**

(5) **Pressed :** It returns **True** if the state of the item is pressed; otherwise, **false**.

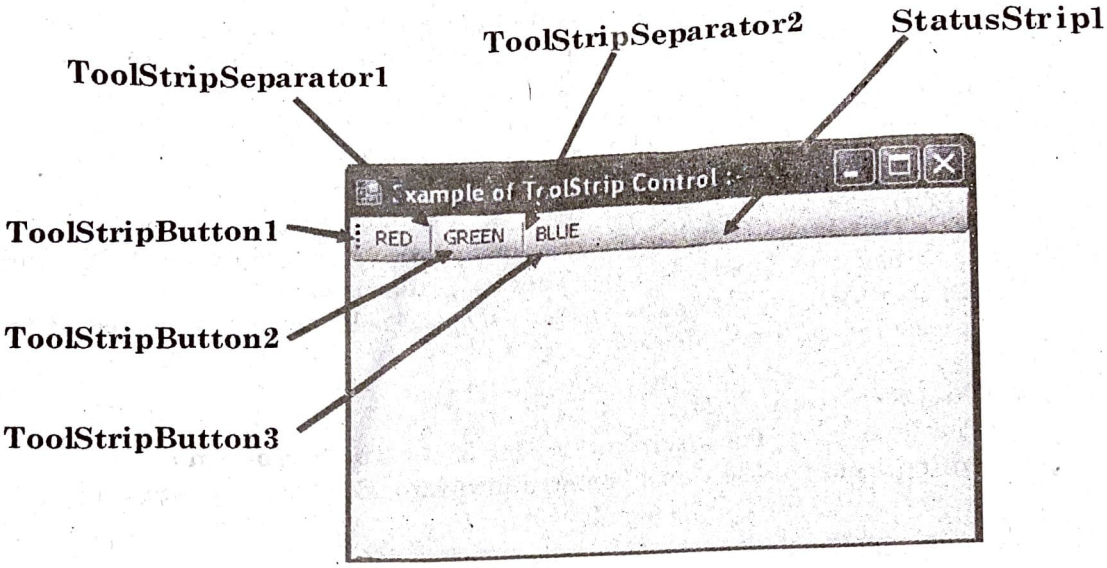
(6) **Text :** Gets or sets the text that is to be displayed on the item.,

(7) **TextAlign** (8) **TextDirection** (9) **ToolTipText**

(10) **Visible :** Gets or sets a value indicating whether the item is displayed into the toolstrip control.

- **Event :**

(1) **Click :** Occurs when the ToolStripItem is clicked.



Step 1 : Place one StatusStrip control.

Step 2 : **Coding of Form Load Event**, Add three buttons & two seperator bar as shown in figure.

```
Dim B1 As New ToolStripButton
B1.Text = "RED"
ToolStrip1.Items.Add(B1)
Dim S1 As New ToolStripSeparator
ToolStrip1.Items.Add(S1)

Dim B2 As New ToolStripButton
B2.Text = "GREEN"
ToolStrip1.Items.Add(B2)

Dim S2 As New ToolStripSeparator
ToolStrip1.Items.Add(S2)

Dim B3 As New ToolStripButton
B3.Text = "BLUE"
ToolStrip1.Items.Add(B3)
```

Step 3 : **Coding of ToolStrip1 ItemClicked event, For example if RED button is click, then chcnge the background colour of the form to RED.**

```
If e.ClickedItem.Text = "RED" Then
    Me.BackColor = Color.Red
ElseIf e.ClickedItem.Text = "GREEN" Then
    Me.BackColor = Color.Green
ElseIf e.ClickedItem.Text = "BLUE" Then
    Me.BackColor = Color.Blue
End If
```

[5] StatusStrip Control or StatusBar Control :

Status bars usually appear at the bottom of a window and it informs the user about the status of the application. For example page number, line number, status of the insert key or information that relates to that object's operation within your application, etc.

A StatusBar can be divided into many sections each of which is referred to as a **panel** or a **pane**. The **Items** property of the StatusStrip control allows a user to add items to the StatusStrip. Following types of items you can add into the StatusStrip control. For example

- ToolStripStatusLabel objects, each of which displays text, an icon, or both.
- ToolStripDropDownButton
- ToolStripSplitButton
- ToolStripProgressBar

Each panel is a **StatusbarPanel** object. You also can set the text in any such object directly, like this: **StatusbarPanel1.Text = "Please enter valid Date"**. To handle panel clicks, you use the **StatusBar** class's **PanelClick** event.

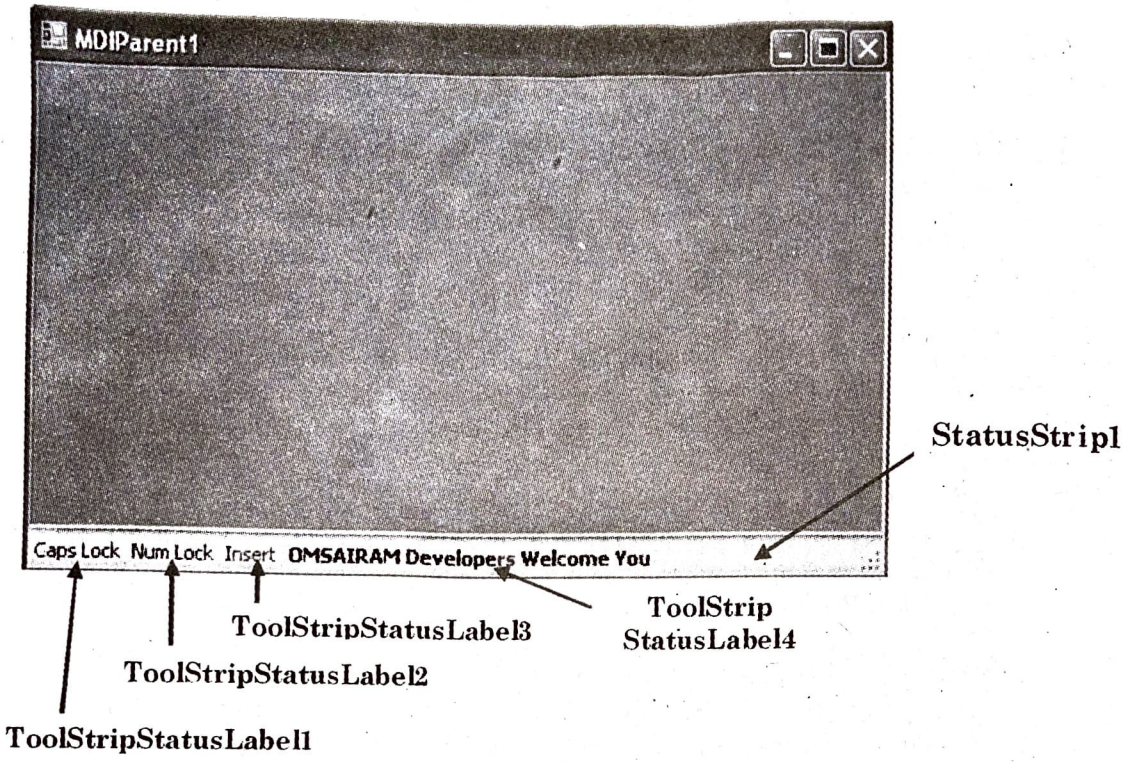
- **Property :**
  (1) Text : Gets/sets the status bar text.
  (2) BackgroundImage : Gets/sets the background image in a status bar.
  (3) Items : It is the Items collection used to retrieve all the items that have been added to the StatusStrip control. Items can be added & removed at design time as well as run time. At design time **Items** property is used to add or remove items by using an Items Collection Editor dialog Box.

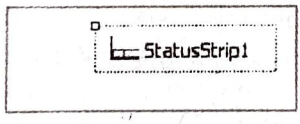
- **Event :**
  (1) ItemClicked : It is the default event of StatusStrip control. It occurs (executes / fires) when the **ToolStripItem** is clicked.

Ex. :

```
ToolStripStatusLabel1.Text = "OMSAIRAM "
ToolStripStatusLabel1.ForeColor = Color.Red
ToolStripStatusLabel2.Text = TimeOfDay
ToolStripStatusLabel2.ForeColor = Color.Red
```

Practical 21 : **Design a Status bar control. Which has a four ToolStripStatusLabel for status of Caps Lock, Num Lock, Insert key & to display text message.**





Step 1 : Place one StatusStrip Control on MDIParent form. Click on Items property of StatusStrip1 control, then it will open "Items Collection Editor dialog Box".

Step 2 : Click on **Add button**, then it will add **ToolStripStatusLabel1**, Set its **Text** property to **Caps Lock**.

Step 3 : Click on **Add button**, then it will add **ToolStripStatusLabel2**, Set its **Text** property to **Num Lock**.

Step 4 : Click on **Add button**, then it will add **ToolStripStatusLabel3**, Set its **Text** property to **Insert**.

Step 5 : Click on **Add button**, then it will add **ToolStripStatusLabel4**.

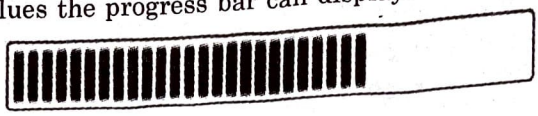
Step 6 : Write coding as follows.

```
'General Declaration Section.
    Imports System.Windows.Forms

    Public Class MDIParent1
        ` Sub Procedure to Check Status of Key.
        Private Sub Check_KeyLock()
            If Control.IsKeyLocked(Keys.CapsLock) = True Then
                StatusStrip1.Items(0).Enabled = True
            Else
                StatusStrip1.Items(0).Enabled = False
            End If
            If Control.IsKeyLocked(Keys.NumLock) = True Then
                StatusStrip1.Items(1).Enabled = True
            Else
                StatusStrip1.Items(1).Enabled = False
            End If
            If Control.IsKeyLocked(Keys.Insert) = True Then
                StatusStrip1.Items(2).Enabled = True
            Else
                StatusStrip1.Items(2).Enabled = False
        End If
    End Sub

'Coding of MFIParent1 Load event
    Private Sub MDIParent1_Load(ByVal sender As System.Object, ByVal e As
    System.EventArgs) Handles MyBase.Load
        Check_KeyLock()
        StatusStrip1.Items(3).Text = "OMSAIRAM Developers Welcome You"
    End Sub
'Coding of MFIParent1 KeyUp event
    Private Sub MDIParent1_KeyUp(ByVal sender As Object, ByVal e As
    System.Windows.Forms.KeyEventArgs) Handles Me.KeyUp
        Check_KeyLock()
    End Sub
End Class
```

[6] Progress Bar Control :

Progress Bar is used to show the progress of some operation (by displaying rectangles in a horizontal bar) & shows how long it will take time to finish the operation.

The main properties of a progress bar are **Value**, **Minimum**, and **Maximum**. You use the **Minimum** and **Maximum** properties to set the maximum and minimum values the progress bar can display.



- **Property :**
  (1) Maximum : Gets/sets the progress bar's maximum value. (Default is 0).
  (2) Minimum : Gets/sets the progress bar's minimum value. (Default is 100).
  (3) Value : Gets/sets the current value of the progress bar. (Default is 0).
  (4) Step : Gets/sets the value by which the **PerformStep** method will increase a progress bar's value. By default Step value is 10.

- **Method :**
  (1) Increment : Increments the position of the progress bar by a given amount.
  (2) PerformStep : Increments the value of the progress bar by the Step property.

Example : Place one Timer Control.
Set enable property of Timer control to True.
Set Interval property of Timer Control to 1000.
Place one ProgressBar control.
Write down following code in **Tick event of Timer1**

```
ProgressBar1.Value = ProgressBar1.Value + 10
    If ProgressBar1.Value = ProgressBar1.Maximum Then
        Timer1.Enabled = False
    End If
```

OR Write down following code in **Tick event of Timer1**

```
ProgressBar1.PerformStep()
```

PerformStep method will increment the value of the progress bar by the step property, By default step value is 10. It will not go beyond the maximum value.

[7] Tab Control :

If your interface has so many options, and space is a problem, then you can divide them up by category and give each category its own tab page. Tab control is used to divides the form visually into several tabs.

Tab control is designed to help you conserve space. Each tab in a TabControl is a TabPage, which contains the control or picture. When a tab is clicked, it displays its page. Example of Tab control is Display Properties of control panel in the Windows operating system, some dialog box may use Tab control. Additionally, the **TabControl** can be used to create property pages, which are used to set a group of related properties.

The main property of the **TabControl** is **TabPages**, which contains the individual tab pages in the control. TabPage can be added & removed at design time as well as run time. At design time **TabPages** property is used to Add or Remove TabPages by using the TabPage Collection Editor dialog Box.

At run time, you can add new tab pages with the **TabPages** collection's **Add** method, and remove them with the **Remove** method.

- **Property :**

  (1) DisplayRectangle : Gets the bounding rectangle of the tab pages.

  (2) HotTrack : Gets/sets whether the tabs should change appearance when the mouse is over them.

  (3) Multiline : Gets/sets whether the tab control can show more than one row of tabs.

  (4) TabPages : Gets the **collection** of tab pages. TabPage can be added & removed at design time as well as run time. At design time **TabPages** property is used to Add or Remove TabPages by using the TabPage Collection Editor dialog Box.

  (5) TabCount : Gets the number of tabs in the TabControl.

  (6) SelectedTab : Gets/sets the selected tab page. (Tab page that is currently selected).
  For Ex. : TabControl1.SelectedTab = TabControl1.TabPages(3)

  (7) SelectedIndex : Gets/sets selected tab page's index.

Ex. 1 :

```
' At run time to set the Text property of Tabpages1 you have to write.
TabControl1.TabPages.Item(0).Text = "Education"
```

Ex. 2 :

```
' At run time, you can add new tab pages to Tab control with the TabPages
collection's Add method.
Dim TP As New TabPage
TP.Text = "Experience"
TabControl1.TabPages.Add(TP)
```

➢ **Coding To Call Crystal Report :**

- First insert the Crystal Report from the : Project-->Add Windows Forms--> Crystal Report. Then Create a new connection, & design the desire report.
- Now take one form (SAI_VIEWER_FORM2) in our example & Place Crystal Report Viewer Control.
- To call report on to the menu item or button control, do as follows.
- Create the Connection object (Con), DataAdapter object (DA), DataTable object (DT).

```
' Coding To Call CrystalReport1
Dim ObjCrystRepo As New CrystalReport1
Dim SAI_SQL_STR As String = "SELECT * FROM EMPLOYEE"
DA = New OleDbDataAdapter(SAI_SQL_STR, Con)
DT = New DataTable
DA.Fill(DT)
ObjCrystRepo.SetDataSource(DT)
SAI_VIEWER_FORM2.CrystalReportViewer1.ReportSource = C
SAI_VIEWER_FORM2.ShowDialog()
ObjCrystRepo.Dispose()
```

SECOND YEAR BCA (SEM. – IV) EXAMINATION
OCTOBER / NOVEMBER - 2008
PAPER : 402 .NET PROGRAMMING
(NEW COURSE)

Q.1 Answer in short. 14

[1] What is Visual Studio .NET?
[2] Explain any two property of Listbox control which is available at run time.
[3] Write a code to insert a picture on Button control at run time and also align a picture at bottom right.
[4] What is the extension of Visual Basic Solution File & Form's code file?
[5] What is Namespace? Give example?
[6] Explain Option Strict statement with suitable example.
[7] What is CLR?

Q.2 Any two. 14

[1] What is IDE? Explain the IDE of Visual Basic.
[2] Explain the .NET Framework?
[3] Explain the Sub Procedure & Function with suitable example.

Q.3 Any two. 14

[1] Explain the Keyboard events with suitable example.
[2] Explain the use of following controls.
  [a] Panel control
  [b] ListView Control
  [c] LinkLabel Control
[3] Explain the ComboBox control with its property & methods.

Q.4 Any two. 14

[1] Explain MsgBox & InputBox function with suitable example.
[2] Write short note on exception handling?
[3] Explain the ScrollBar control with its property, methods & events.

Q.5 Any two. 14

[1] Write down short note on Object Oriented Programming.
[2] Write steps to connect oracle database, & write a steps to execute DML statements.
[3] Write short note on ADO.NET Object Model.
[4] Explain String handling functions & methods with examples.

View → Server explorer

SECOND YEAR BCA (SEM. – IV) EXAMINATION
MARCH / APRIL - 2009
PAPER : 402 .NET PROGRAMMING
(NEW COURSE)

Accesskey Alter
Control key control
Shortkey

Q.1 Answer in short. 10

[1] What is the use of assembly?
[2] How to declare the constant?
[3] Is it necessary to declare the variable?
[4] Give any one form controls method.
[5] How to load a picture in picture box during run time?
[6] Define shadowing.
[7] Give any two properties of list box which is used to connect to a database.
[8] Define context menu.
[9] What is a data set?
[10] What is data binding?

Dimple D. Patel

Dimple D. Patel

Q.2 [1] Compare select-case with IF-ELSE IF. 6

OR

[1] Write a note on string functions. 6
[2] Write notes on data types. 4
[3] Write a note on conversion functions. 3

OR

[3] Explain user define data type. 3

Q.3 Answer (Any three) : 12

[A] Define exception. Explain on structured exception.
[B] Write a note on used defined functions.
[C] Write on link label & its properties & events.
[D] Discuss on color dialog

Q.4 Any three. 15

[A] Tree node - Discuss
[B] Compare – Overload & Overriding
[C] Navigation in data sets.
[D] Adding controls during run time.

Q.5 [A] What is an interface? How to create it? 10
[B] Give the method used in data sets to manipulate records.

OR

[A] Write about significance of MDI application. 4
[B] Short notes on data row & data column. 6

Q.6 Any two. 10

[A] Write about classes and objects.
[B] Short notes on module, how to create it?
[C] Create tool bar & discuss on properties, events.