

Windows Forms

CSIS 3540

Client Server Systems

Class 05

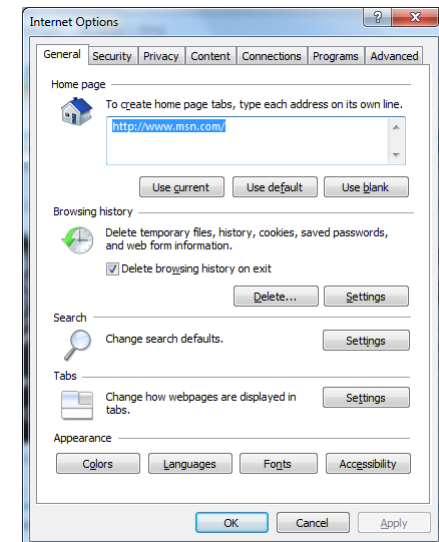
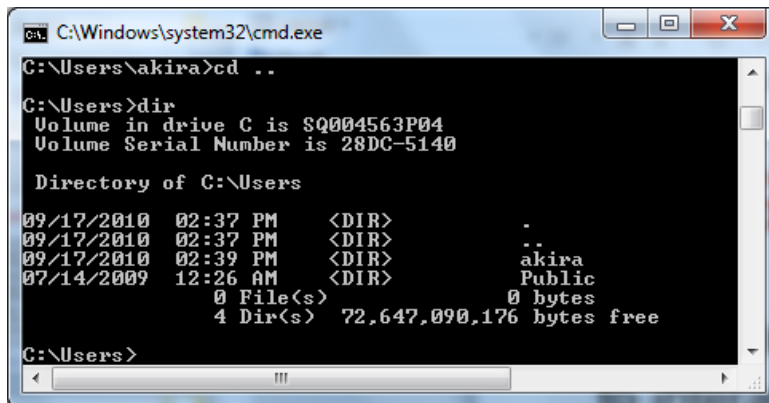
©Michael Hrybyk and others
NOT TO BE REDISTRIBUTED

Topics

- GUI
- Controls and Naming Conventions
- Forms
- Buttons
- Labels
- Textbox
- Groupbox
- Radio Buttons and Check Boxes
- ListBox and ListView
- DataGridView

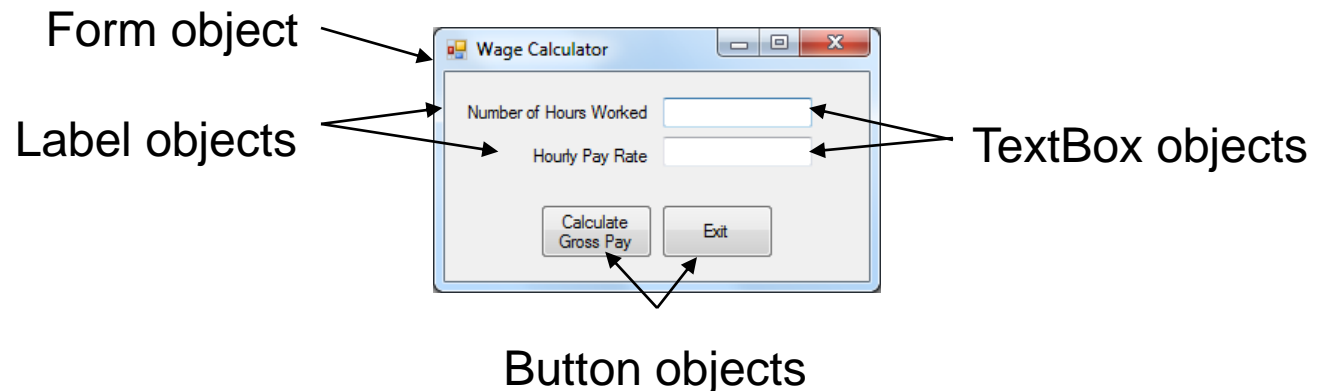
Graphical User Interface

- User interfaces allow users to interact with the computer. Categories are:
 - Command line interface (aka console interface)
 - Graphical user interface (GUI)- now the most commonly used



Objects

- Program objects have properties (or fields) and methods
 - Properties – data stored in an object
 - Methods – the operations an object can perform

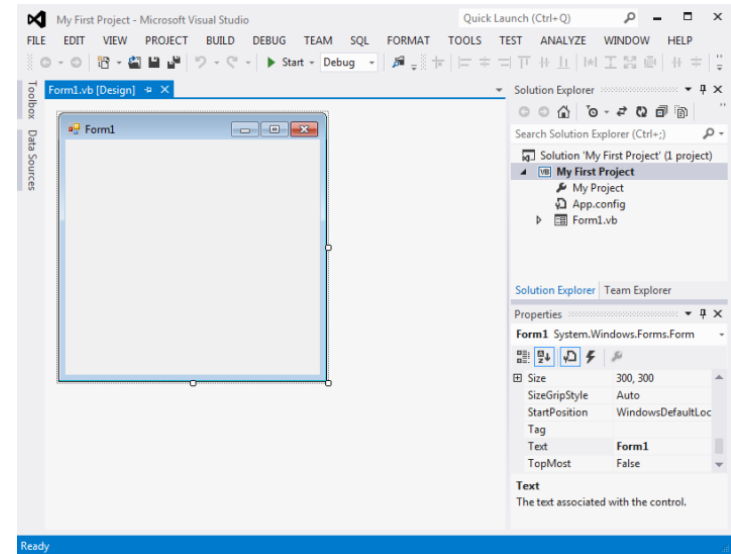


Controls

- Objects that are visible in a program GUI are known as controls
 - Commonly used controls are Labels, Buttons, and TextBoxes
 - They enhance the functionality of your programs

Getting Started with Forms and Controls

- A Visual C# application project starts with creating its GUI with
 - Designer
 - Toolbox
 - Property window
- In the Designer, an empty form is automatically created
 - An application's GUI is made of forms and controls
 - Each form and control in the application's GUI must have a name as ID. The default blank form is named "Form1" automatically.
- NAMING RULES
 - Come up with a **Form Name**
 - Hello World
 - Rename Program.cs to **FormNameProgram.cs**
 - No spaces!
 - HelloWorldProgram.cs
 - Rename Form1.cs to **FormNameForm.cs**
 - No spaces!
 - HelloWorldForm.cs
 - This will rename the internal form name in the code as well.
 - In Form Properties -> Text
 - Rename Form1 **Form Name**
 - **Note use of spaces as in original name**
 - This will now be displayed in the top left of the form
 - Can also be done in code
- See BasicWindowsForm

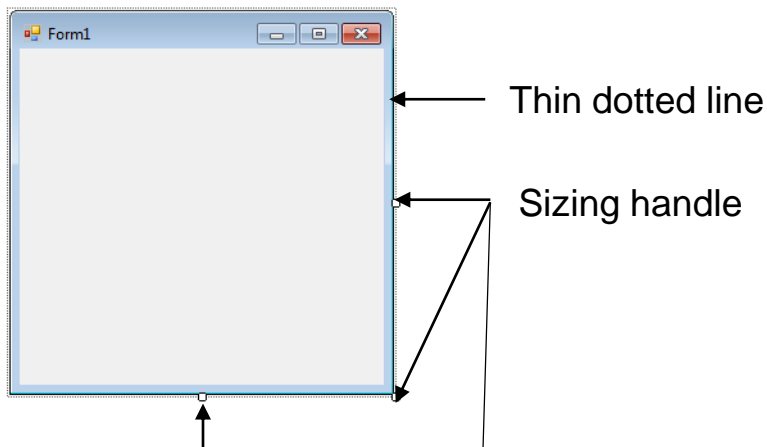


Visual Studio files

- Form.cs (or whatever you rename it)
 - When you click on this under solution, Designer opens.
 - Allows you to dragndrop controls.
 - F7 (or right click on Form.cs) will View Code
 - You will need this to complete any program!
- Designer.cs (will have been renamed)
 - Contains all Designer-generated code.
 - Do NOT edit this.
 - Good demonstration of use of events and callbacks.
 - Use of partial class
- Program.cs (remember to rename this)
 - Do NOT edit this.
 - Simply calls the form from Main.

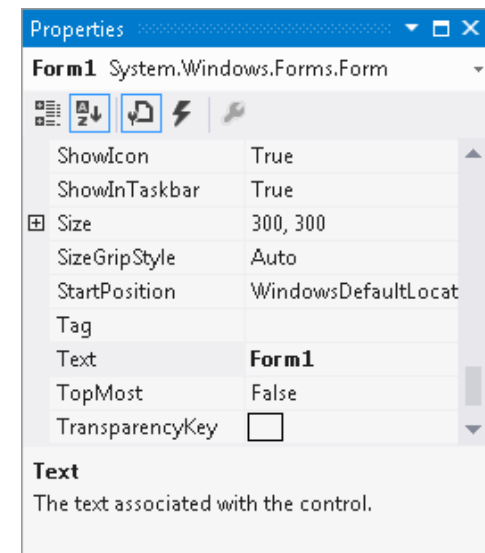
Form's Bounding Box and Sizing Handles

- The default empty form has a dimension (size) of 300 pixels wide by 300 pixels high
- A form in Designer is enclosed with thin dotted lines called the bounding box
- The bounding box has small sizing handles; you can use them to resize the form



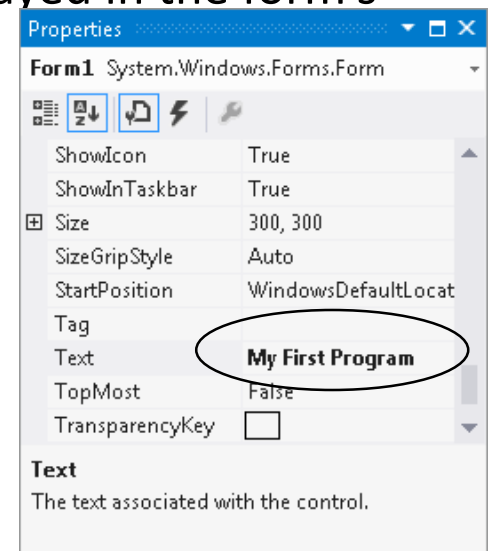
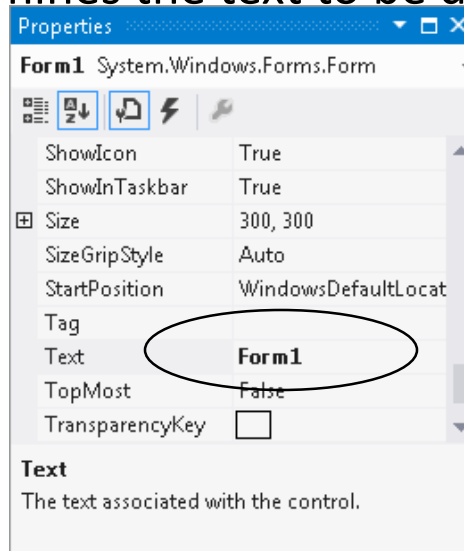
The Property Window

- The appearance and other characteristics of a GUI object are determined by the object's properties
- The Properties window lists all properties
 - When selecting an object, its properties are displayed in Properties windows
 - Each property has 2 columns:
 - Left: property's name
 - Right: property's value



Changing a Property's Value

- Select an object, such as the Form, by clicking it once
- Click View and select Properties if the Properties window is not available
- Can set properties in code as well (preferred).
- Find the property's name in the list and change its value
 - The Text property determines the text to be displayed in the form's title bar
 - Example: Change the value from "Form1" to "My First Program"



Adding Controls to a Form

- In the Toolbox, select the Control (e.g. a Button), then you can either:
 - double click the Button control
 - click and drag the Button control to the form
- On the form, you can
 - resize the control using its bounding box and sizing handles
 - move the control's position by dragging it
 - change its properties in the Properties window
 - although this should be done in code.

Rules for Naming Controls

- Controls' name are identifiers of the controls
- The naming conventions are:
 - Name must be in camelCase
 - All other characters can be alphanumerical characters or underscores
 - The name cannot contain spaces
- Examples of good names are:
 - buttonShowDay
 - labelDisplayTotal
 - labelScore
- **MUST USE camelCase!**

Naming Conventions

- Objects, Methods, Interfaces

- PascalCase –

- Concatenated descriptive words
 - Each word capitalized

- Example:

- MyClass
 - myClass.MyMethod()

- Variables, Instances, Controls

- camelCase –

- Concatenated descriptive words
 - first word lower case, all other words capitalized

- Variables

- i, n, nDays, maxHours

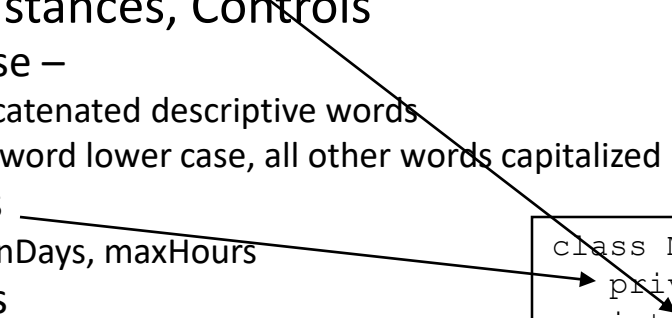
- Instances

- MyClass myHours;

- Controls

- textBox1 should be textBox<description>
 - textBoxInputHours

- Do not use 'Hungarian'



```
class MyClass {  
    private int maxHours;  
    int GetMaxHours() {  
        return(maxHours);  
    }  
}  
...  
Class MyClass myHours;  
...  
employeeMaxHours = myHours.GetMaxHours();
```

The diagram shows two arrows originating from the list items. One arrow points from 'Variables' to the line 'private int maxHours;' in the code block. The other arrow points from 'Instances' to the line 'Class MyClass myHours;' in the same code block.

Creating the GUI for Your First Visual C# Application

- Components: a Form and a Button control
- Purpose:
 - Create the application's GUI
 - Write the code that causes “Hello World” to appear when the user clicks the button

Organization of the Form1.cs

- A sample of Form1.cs:
 1. The using directives indicate which namespaces of .NET Framework this program will use.
 2. The user-defined namespace of the project not .NET Framework namespaces
 3. Class declaration
 4. A method
- C# code is organized as methods, which are contained inside classes, which are contained inside namespaces

```
using System;  
using System.Collections.Generic;  
using System.ComponentModel;  
using System.Data;  
using System.Drawing;  
using System.Linq;  
using System.Text;  
using System.Windows.Forms;  
  
namespace Hello_World  
{  
    {  
        public partial class Form1 : Form  
        {  
            public Form1()  
            {  
                InitializeComponent();  
            }  
        }  
    }  
}
```

1

2

3

4

Adding Your Code

- GUI applications are event-driven which means they interact with users
- An event is a user's action such as mouse clicking, key pressing, etc.
- Double clicking a control, such as Button, will link the control to a default Event Handler
 - An event handler is a method that executes when a specific event takes place
 - A code segment similar to the following will be created automatically:

```
private void MyButton_Click(object sender, EventArgs e)
{

}
```


Message Boxes

- A message box (aka dialog box) displays a message
- The .NET Framework provides a method named **MessageBox.Show**
 - C# can use it to pop up a window and display a message. A sample code is (bold line):

```
private void MyButton_Click(object sender, EventArgs e)
{
    MessageBox.Show("Thanks for clicking the button!");
}
```

- Placing it in the myButton_Click event handler can display the string in the message box when the button is clicked

Hello World Application

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace Hello_World
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void MyButton_Click(object sender, EventArgs e)
        {
            MessageBox.Show("Thanks for clicking the button!");
        }
    }
}
```

Label Controls

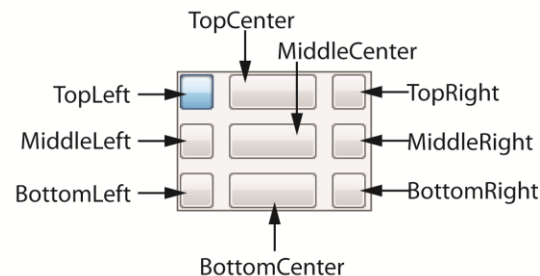
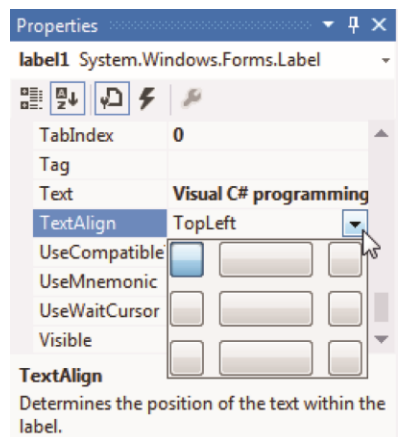
- A Label control displays text on a form and can be used to display unchanging text or program output
- Commonly used properties are:
 - Text: gets or sets the text associated with Label control
 - Name: gets or sets the name of Label control
 - Font: allows you to set the font, font style, and font size
 - BorderStyle: allows you to display a border around the control's text
 - AutoSize: controls the way they can be resized
 - TextAlign: set the text alignments
- Set these in Properties window OR directly in code.

Handling Text Alignments

- The TextAlign property supports the following values:

<u>TopLeft</u>	<u>TopCenter</u>	<u>TopRight</u>
<u>MiddleLeft</u>	<u>MiddleCenter</u>	<u>MiddleRight</u>
<u>BottomLeft</u>	<u>BottomCenter</u>	<u>BottomRight</u>

- You can select them by clicking the down-arrow button of the TextAlign property



Using Code to Display Output in a Label Control

- By adding the following bold line to a Button's event handler, a Label control can display output of the application.

```
private void ShowAnswerButton_Click(object sender, EventArgs e)
{
    answerLabel.Text = "Theodore Roosevelt";
}
```

- Notice that
 - the equal sign (=) is known as assignment operator
 - the item receiving value must be on the left of = operator
 - the Text property accepts string only
 - if you need to clear the text of a Label, simply assign an empty string ("") to clear the Text property
- See BasicWindowsForm

Writing the Code to Close an Application's Form

- To close an application's form in code, use the following statement:

```
this.Close();
```

- A commonly used practice is to create an Exit button and manually add the code to it:

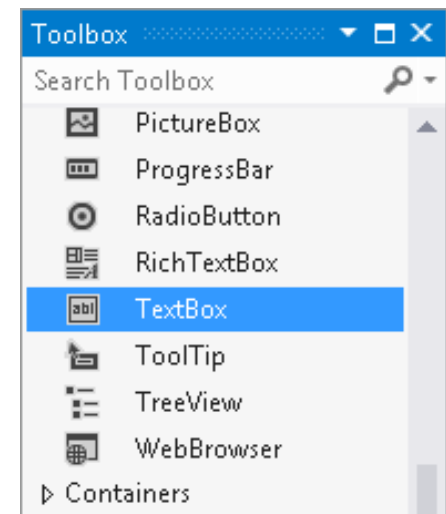
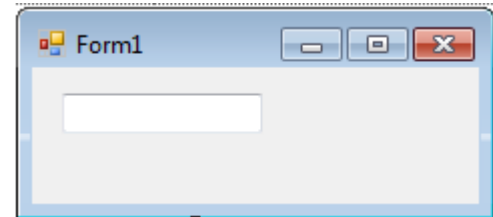
```
private void ExitButton_Click(object sender, EventArgs e)
{
    // Close the form.
    this.Close();
}
```

Reading Input with TextBox Control

- TextBox control

- a rectangular area
- can accept keyboard input from the user
- locates in the Common Control group of the Toolbox
- double click to add it to the form
- default name is `textBoxn`

where n is 1, 2, 3, ...



The Text Property

- A TextBox controls **Text** property stores the user inputs

- **Text** property accepts only string values, e.g.

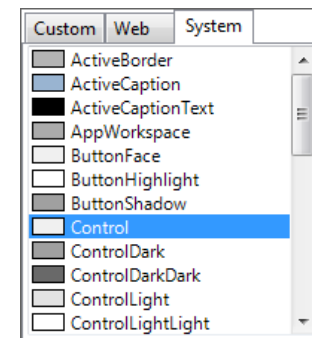
```
textBox1.Text = "Hello";
```

- To clear the content of a TextBox control, assign an empty string("")

```
textBox1.Text = "";
```

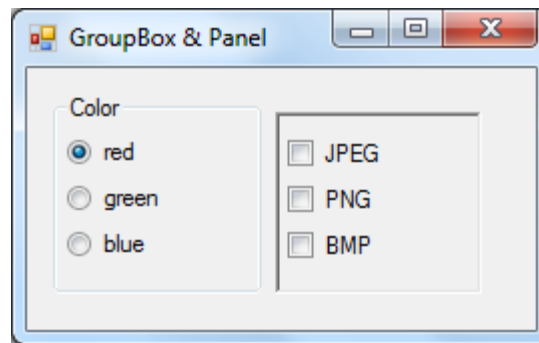

Setting Colors

- Forms and most controls have a BackColor property
- Controls that can display Text also have a ForeColor property
- These color-related properties support a drop-down list of colors
- The list has three tabs:
 - Custom: display a color palette
 - Web: list colors displayed with consistency in Web browsers
 - System: list colors defined in current Windows
- You can set colors in color
 - The .NET Framework provides numerous values that represent colors
`messageLabel.BackColor = Color.Black;`
`messageLabel.ForeColor = Color.Yellow;`



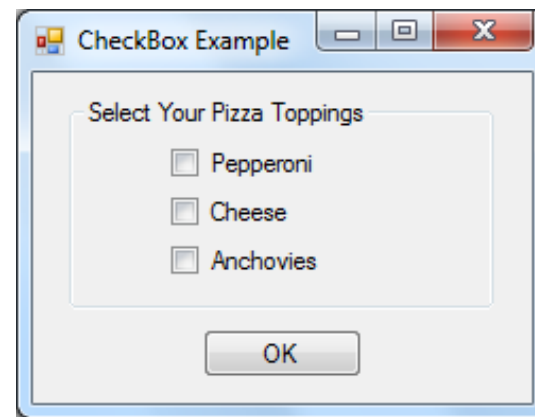
GroupBoxes vs. Panels

- A GroupBox control is a container with a thin border and an optional title that can hold other controls
- A Panel control is also a container that can hold other controls
- There are several primary differences between a Panel and GroupBox:
 - A panel cannot display a title and does not have a Text property, but a GroupBox supports these two properties.
 - A panel's border can be specified by its BorderStyle property, while the GroupBox cannot be



Radio Buttons and Check Boxes

- **Radio buttons** allow users to select one choice from several possible choices
 - Clicking on a radio button, the program automatically deselects any others. This is known as mutually exclusive selection
- **Check boxes** allows users to select multiple options



RadioButton Control

- The .NET Framework provides the RadioButton Control for you to create a group of radio buttons
- Common properties are:
 - **Text**: holds the text that is displayed next to the radio button
 - **Checked**: a Boolean property that determines whether the control is selected or deselected

Working with Radio Buttons in Code

- In code, use a decision structure to determine whether a RadioButton is selected. For example,

```
If (choiceRadioButton.Checked) { } else { }
```

- You do not need to use the == operator, although the following is equivalent to the above

```
If (choiceRadioButton.Checked == true) { } else { }
```

CheckBox Control

- The .NET Framework provide the CheckBox Control for you to give the user an option, such as true/false or yes/no
- Common properties are:
 - **Text**: holds the text that is displayed next to the radio button
 - **Checked**: a Boolean property that determines whether the control is selected or deselected
- In code, use a decision structure to determine whether a CheckBox is selected. For example,

```
If (option1CheckBox.Checked) { } else { }
```

The CheckedChanged Event

- See **MoreControls**
- Anything a RadioButton or a CheckBox control's Checked property changes, a CheckedChanged event is raised
- You can create a CheckedChanged event handler and write codes to respond to the event
- Double click the control in the Designer to create an empty CheckedChanged event handler similar to:

```
private void YellowRadioButton_CheckedChanged(object sender, EventArgs e) { }
```

Introduction to List Boxes

- A list box displays a list of items and allow the user to select an item from the list
- In C#, you can use the **ListBox** control to create a list box
- Commonly used properties are:
 - **Text**: Gets or searches for the text of the currently selected item in the ListBox
 - **Items**: Gets/Adds the items of the ListBox
 - **SelectedItem**: Gets or sets the currently selected item in the ListBox. When the user selects an item, the item is stored in this property. You can use the following to get the selected item from a list box:

```
selectedFruit = fruitListBox.SelectedItem.ToString();
```

- **SelectedIndex**: Gets or sets the zero-based index of the currently selected item in a ListBox

SelectedItem

- An exception will occur if you try to get the value of a ListBox's **SelectedItem** property when no item is selected
- Items in a list box have an index starting with 0
 - The first item has index 0, the nth item has index n-1
 - The SelectedIndex property keeps the index. If no item is selected the, value of SelectedIndex is -1
- The following is an example to use **SelectedIndex** property to make sure that an item is selected before you get the value from SelectedItem.

```
if (fruitListBox.SelectedIndex != -1) { }
```

More About ListBoxes

- ListBox controls have various methods and properties that you can use in code to manipulate the ListBox's contents
- The **Items.Add** method allows you to add an item to the ListBox control

```
ListBoxName.Items.Add(Item);
```

- where *ListBoxName* is the name of the ListBox control; *Item* is the value to be added to the Items property
- The **Items.Clear** method can erase all the items in the Items property

```
employeeListBox.Items.Clear();
```

- The **Count** property reports the number of items stored in the ListBox
- See MoreControls

ListBox selection properties

- Note use of SelectionMode, which can be single or multi.
- MultiExtended allows the user to use ctrl and shift

```
// allow use of ctrl and shift
listBoxCheckedSchools.SelectionMode = SelectionMode.MultiExtended;
listBoxCheckedSchools.SelectedIndexChanged += ListBox_SelectedIndexChanged;

// simple multi select enabled
listBoxButtonSchools.SelectionMode = SelectionMode.MultiSimple;
listBoxButtonSchools.SelectedIndexChanged += ListBox_SelectedIndexChanged;
```

SelectedItems processing

- Loop through all selected items and process
- Note that SelectedItems are objects returned, so a cast or toString() is needed to display.

```
private void ListBox_SelectedIndexChanged(object sender, EventArgs e)
{
    ListBox listBox = sender as ListBox;

    // use a for loop
    for (int i = 0; i < listBox.SelectedItems.Count; i++)
        MessageBox.Show(listBox.SelectedItems[i].ToString());

    // use foreach since we know it is a string
    foreach (object o in listBox.SelectedItems)
        MessageBox.Show(o.ToString());
}
```

DateTime and Date Picker

- DateTime is a class, with values for Day, Year, Month, etc.
 - These are all values (numeric)
 - They can be operated upon
- Use the Date Picker control to get user input
 - Use the Value property to get the DateTime chosen.
- Use DateTime properties to manipulate the date

The Load Event

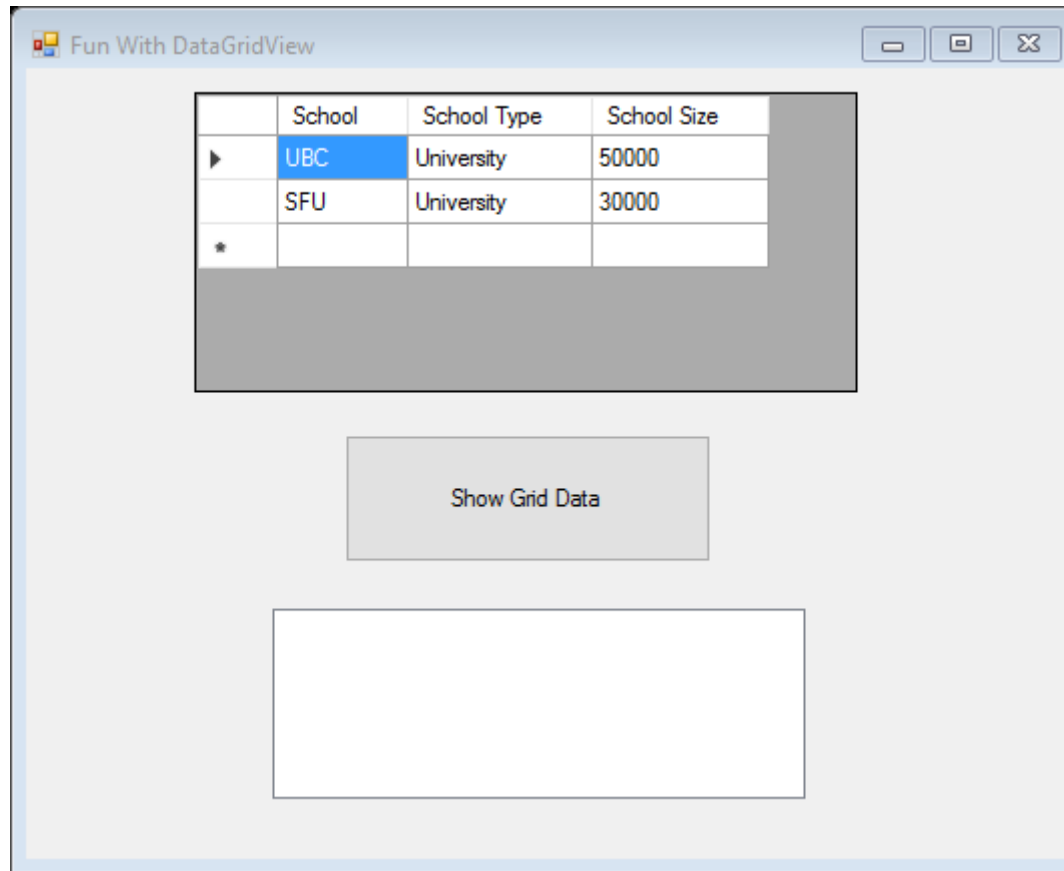
- When running an application, the application's form is loaded into memory and an event known as **Load** takes place
- To create a Load event handler, simply double click the form in the Designer
- An empty Load event handler looks like:

```
private void Form1_Load(object sender, EventArgs e) { }
```

- Any code you write inside the Load event will execute when the form is launched. For example,

```
private void Form1_Load(object sender, EventArgs e)
{
    MessageBox.Show("Prepare to see the form!");
}
```

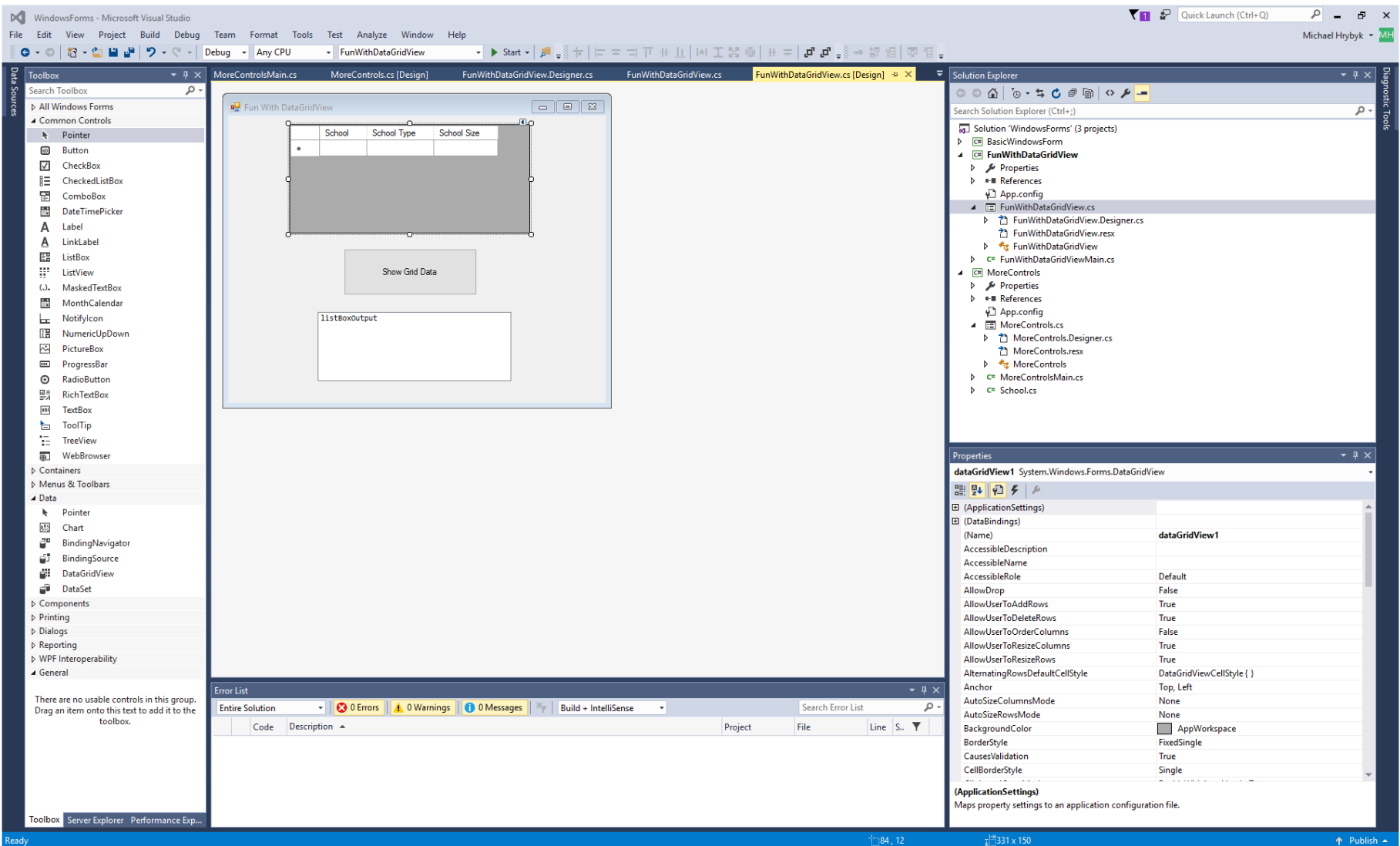
DataGridView control



Setting up DataGridView control

- Open properties
- Set up columns
 - make sure width and type are set correctly
 - Can do this in Designer or in code (much like ListView)
- Use Add() method to initialize if necessary or for additional data.
- Allow editing and deleting if needed
 - Not required for simple reports
 - Can read and write easily from the control!
- THIS CONTROL IS USED A LOT FOR THE REST OF THE COURSE!

DataGridView



DataGridView properties

DataGridView Tasks

Choose Data Source: (none) ▼

[Edit Columns...](#)

[Add Column...](#)

☒ Enable Adding

☒ Enable Editing

☒ Enable Deleting

☐ Enable Column Reordering

[Dock in Parent Container](#)

Edit Columns

Selected Columns:

- School
- School Type
- School Size

Unbound Column Properties

Appearance	
DefaultCellStyle	DataGridViewCellStyle { }
HeaderText	School
ToolTipText	
Visible	True

Behavior	
ContextMenuStrip	(none)
MaxInputLength	32767
ReadOnly	False
Resizable	True
SortMode	Automatic

Data	
DataPropertyName	(none)

Design	
(Name)	gridViewTextBoxColumnSchoc
ColumnType	DataGridViewTextBoxColumn

Layout	
AutoSizeMode	ColumnHeader
DividerWidth	0
FillWeight	100
Frozen	False
MinimumWidth	5
Width	65

DataPropertyName
The name of the data source property or database column to which the DataGridViewColumn is bound.

OK Cancel

Adding and Showing data

```
public partial class FunWithDataGridView : Form
{
    public FunWithDataGridViewForm()
    {
        InitializeComponent();

        // shows how to add rows to the grid
        dataGridViewSchools.Rows.Add(new string[] { "UBC", "University", "50000" });
        dataGridViewSchools.Rows.Add(new string[] { "SFU", "University", "30000" });
    }

    // button press - display grid data in a listbox
    private void buttonShowGridData_Click(object sender, EventArgs e)
    {
        listBoxOutput.Items.Clear();

        StringBuilder sb = new StringBuilder(); // build a string

        // iterate through rows then columns to get cells
        foreach (DataGridViewRow row in dataGridView1.Rows)
        {
            sb.Clear(); // clear the string

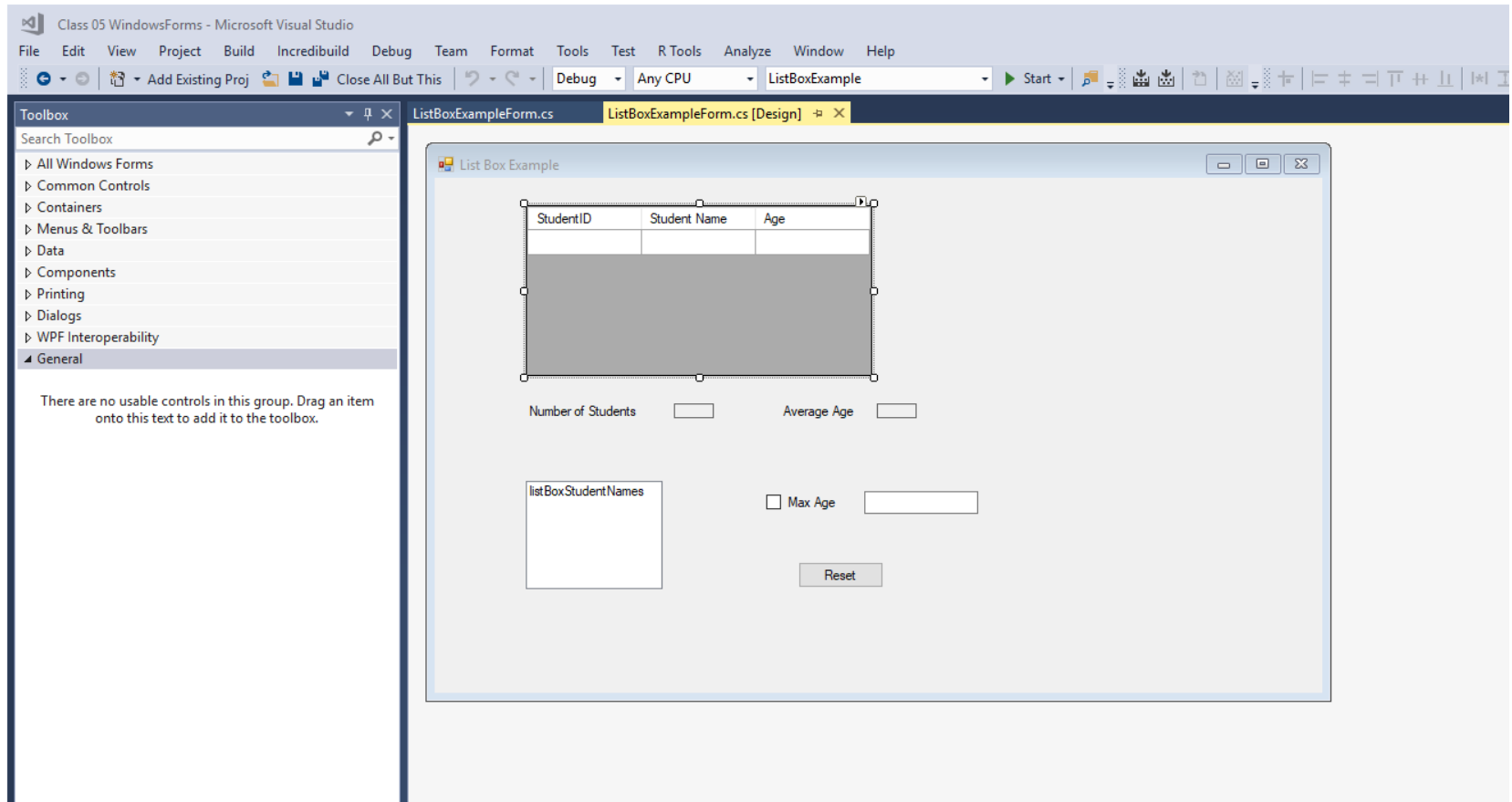
            // for every cell in the row, add the cell value to the string
            foreach (DataGridViewCell cell in row.Cells)
            {
                // make sure the cell value exists
                if (cell.Value != null)
                {
                    sb.Append(cell.Value.ToString());

                    // if we are not at the last cell, add a :
                    if (cell.ColumnIndex != row.Cells.Count - 1)
                        sb.Append(":");
                }
            }
            listBoxOutput.Items.Add(sb.ToString());
        }
    }
}
```

Putting it all together - ListBoxExample

- Create the main form
- Add controls using designer and toolbox
 - In this example
 - DataGridView
 - ListBox
 - Checkbox
 - Labels for output
 - Button
- Could do this programmatically, but much easier to use designer for this.
- Initialize control properties in the Form constructor or in the Load() method.

ListBoxExample form



Set up DataGridView in code

- Initialize any properties (rather than using designer)
- Set up all columns
- Called in Form constructor

```
private void InitializeDataGridViewStudents()
{
    // control should be readonly, nothing can be changed
    dataGridViewStudents.ReadOnly = true;
    dataGridViewStudents.AllowUserToAddRows = false;
    dataGridViewStudents.RowHeadersVisible = false;

    // autofill the columns to the control width
    dataGridViewStudents.AutoSizeColumnsMode = DataGridViewAutoSizeColumnsMode.Fill;
    dataGridViewStudents.Width = 400;

    // set up the columns and column names. first clear anything previously set.
    dataGridViewStudents.Columns.Clear();

    DataGridViewTextBoxColumn[] columns = new DataGridViewTextBoxColumn[] {
        new DataGridViewTextBoxColumn() { Name = "StudentID" },
        new DataGridViewTextBoxColumn() { Name = "Student Name" },
        new DataGridViewTextBoxColumn() { Name = "Age" },
    };

    dataGridViewStudents.Columns.AddRange(columns);
}
```

Set up ListBox control

- Set up properties
- Add values to the control to display
- Called in form constructor

```
/// <summary>
/// Display a unique set of student names in the listbox
/// Also, set any other control properties.
/// </summary>
private void InitializeListBoxStudentNames()
{
    listBoxStudentNames.Items.Clear(); // clear anything that was set up previously

    // ensure listbox selection can be multiple and can use shift, ctl-a, etc

    listBoxStudentNames.SelectionMode = SelectionMode.MultiSimple; // this is REQUIRED!!

    // get a list of unique student names and add to listbox

    // listBoxStudentNames.Items.AddRange(students.OrderBy(x => x.StudentName).Select(x =>
    x.StudentName).Distinct().ToArray());

    var names = from student in students
                orderby student.StudentName
                select student.StudentName;

    listBoxStudentNames.Items.AddRange(names.Distinct().ToArray());
}
```

Set up various control event handlers

- All user input is via ListBox, CheckBox, TextBox, or Button
 - Set event handlers for each
 - Done in form constructor
 - Each event handler basically calls DisplayStudent(), which populates the DataGridView control
- Reset button behavior
 - Need to clear ListBox selections, CheckBox, TextBox

```
// set all controls to defaults, and set listbox event handler
```

```
ResetControlsToDefault();
```

```
// set the event handlers for checkbox, textbox and button
```

```
checkBoxMaxAge.CheckedChanged += CheckBoxMaxAge_CheckedChanged;
```

```
buttonReset.Click += ButtonReset_Click;
```

```
textBoxMaxAge.TextChanged += TextBoxMaxAge_TextChanged;
```


Reset controls

- We don't want events firing when we do a simple reset
- Turn off event handlers (deregister)
- Clear everything, but set ListBox to all selected
 - In another app, you might simply clear the selected
- Display data in DataGridView
- Turn the handlers back on (reregister)

```
// unregister event handler
listBoxStudentNames.SelectedIndexChanged -= ListBoxStudentNames_SelectedIndexChanged;

// set all of the listbox names to selected
for (int i = 0; i < listBoxStudentNames.Items.Count; i++)
    listBoxStudentNames.SetSelected(i, true);

// clear the checkbox and associated text
checkBoxMaxAge.Checked = false;
textBoxMaxAge.Clear();

// redisplay students, which will see that all have been selected
DisplayStudents();

// register the event handler again
listBoxStudentNames.SelectedIndexChanged += ListBoxStudentNames_SelectedIndexChanged;
```

Display

- Get the selected students from the ListBox control
- Get max age from TextBox if CheckBox checked
- Use LINQ to find all students that are selected and less than max age
- Populate DataGridView control with result

```
/// <summary>
/// Given selected students and a specified max age, display the students matching
/// these.
/// Then show statistics - number of students displayed, and their average age
/// </summary>
public void DisplayStudents()
{
    // see if maxage checkbox was checked, if so get the max age

    int maxAge = 0;
    if(checkBoxMaxAge.Checked == true)
        int.TryParse(textBoxMaxAge.Text, out maxAge);

    // get the list of student names selected
    // note use of SelectedItems[index]
    // STUDY THIS

    List<string> selectedNames = new List<string>();

    for (int i = 0; i < listBoxStudentNames.SelectedItems.Count; i++)
        selectedNames.Add(listBoxStudentNames.SelectedItems[i].ToString());

    // now build a query joining names with students, filtering by maxage
    // note if Checked is true AND less than maxAge OR Checked is false makes the where clause evaluate to true
    // and the student is selected

    var selectedStudents = from student in students
                           join name in selectedNames on student.StudentName equals name
                           where (checkBoxMaxAge.Checked == true && student.StudentAge < maxAge) || checkBoxMaxAge.Checked == false
                           select student;

    // now display the students in the datagridview

    dataGridViewStudents.Rows.Clear(); // clear old data first

    foreach (Student student in selectedStudents)
    {
        dataGridViewStudents.Rows.Add(student.StudentID, student.StudentName, student.StudentAge);
    }

    // show statistics, note use of lambda for average age

    int numberOfStudents = selectedStudents.Count();
    labelDisplayNumberOfStudents.Text = numberOfStudents.ToString();

    if (numberOfStudents > 0)
        labelDisplayAverageAge.Text = selectedStudents.Average(s => s.StudentAge).ToString();
    else labelDisplayAverageAge.Text = "0";
}
```