CMPE 1666
Intermediate Programming

Lecture 2B- Windows Form Controls

Introduction To Form Controls

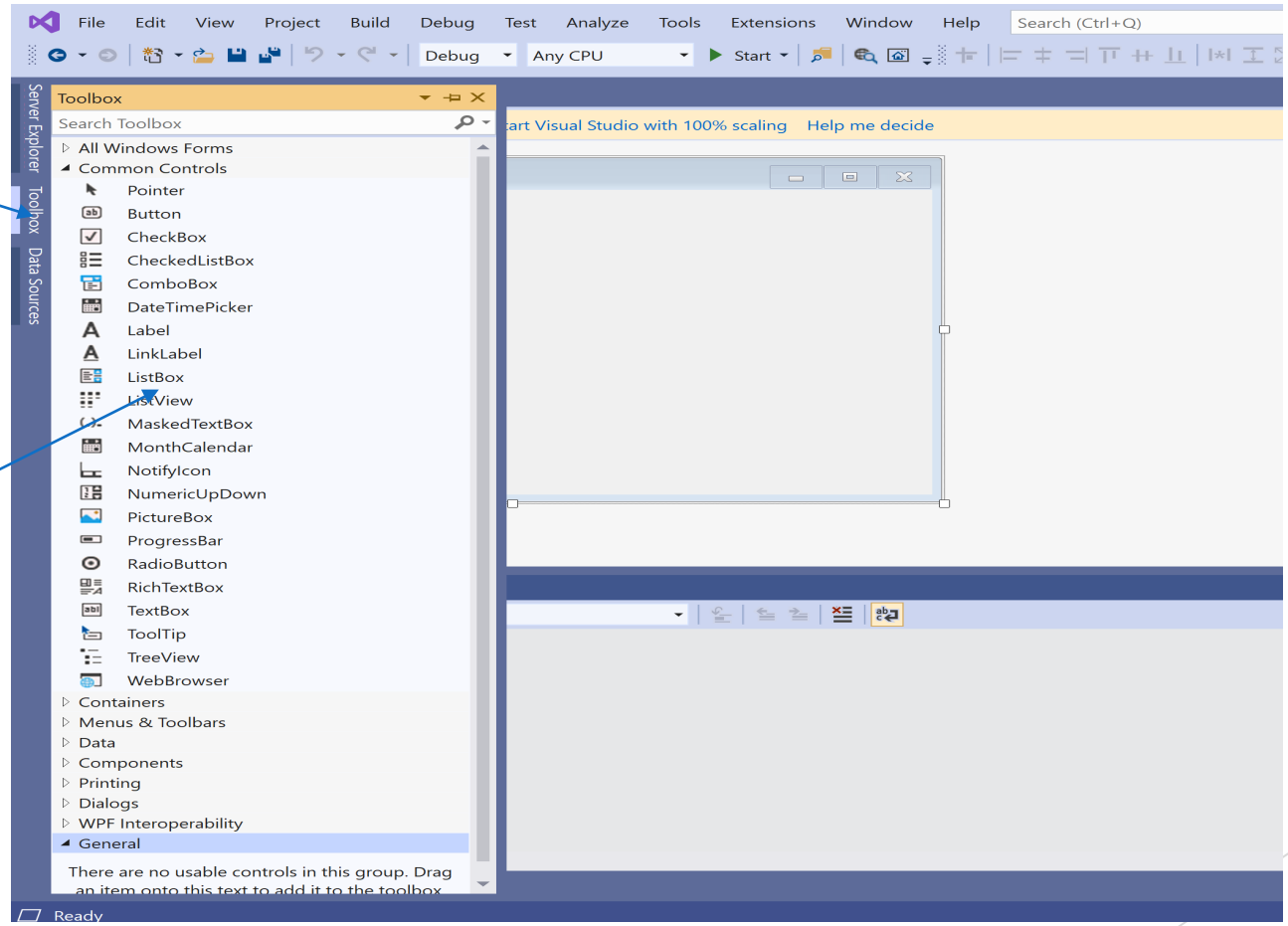
- Form Controls are components that can be placed on Forms for building our applications.
- Form controls provide for user interactions with the form.
- The different components have events and properties.
- The properties can be manipulated through event-handlers.

Introduction To Form Controls

- The Form controls we are interested in, at this point, are:
  - Buttons
  - Labels
  - ListBoxes
  - TextBoxes
  - RaioButtons
  - CheckBoxes
- A long list of existing controls and their properties are given in the slide deck on Windows Form controls (Detailed Descriptions), available on your Moodle site.
- You will refer to these slides for the exercises that will follow.

Obtaining the controls

- When our form is in the design mode, we click on ToolBox to obtain the list of controls

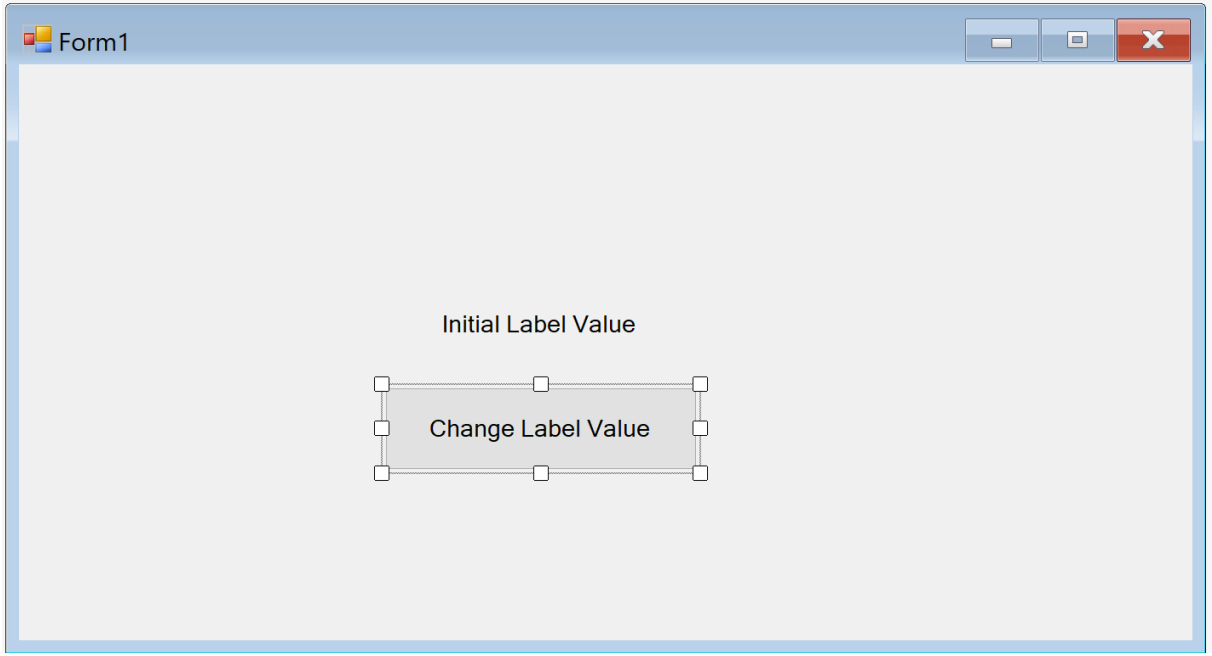


Working With The Controls

- In a typical Windows Form application, users:
  - set values for controls like radio buttons and checkboxes,
  - perform inputs on controls such as textboxes and
  - click on a button
- The output is based on the state of a combination of the controls.
- You are going to work with the controls and use the Windows Forms Controls Slide deck (From your Moodle site) as reference.

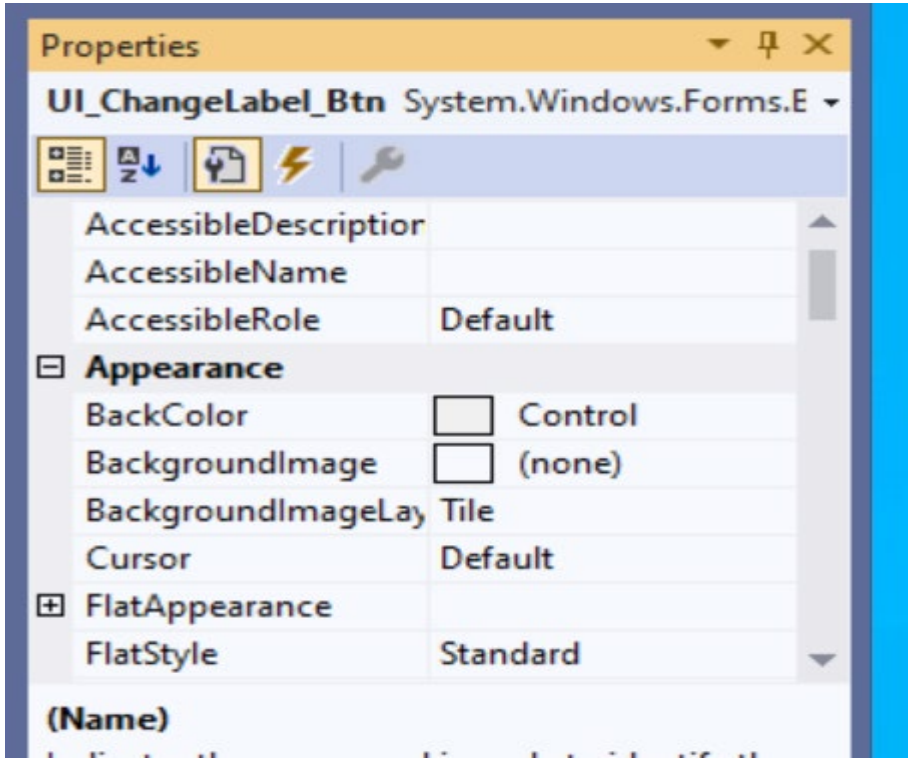
Lecture 2B-Demo1

- Develop a Windows Form Application that has a label and a button as shown.
- The Initial Text on the Label must be “Initial Label Value”
- The button name must be **UI_Change_Btn** and the text must be **“Change Label Value”**. The name of the label must be **UI_Display_Lbl**.
- When the button is clicked, the text of the label must change to **“Hello World!”**



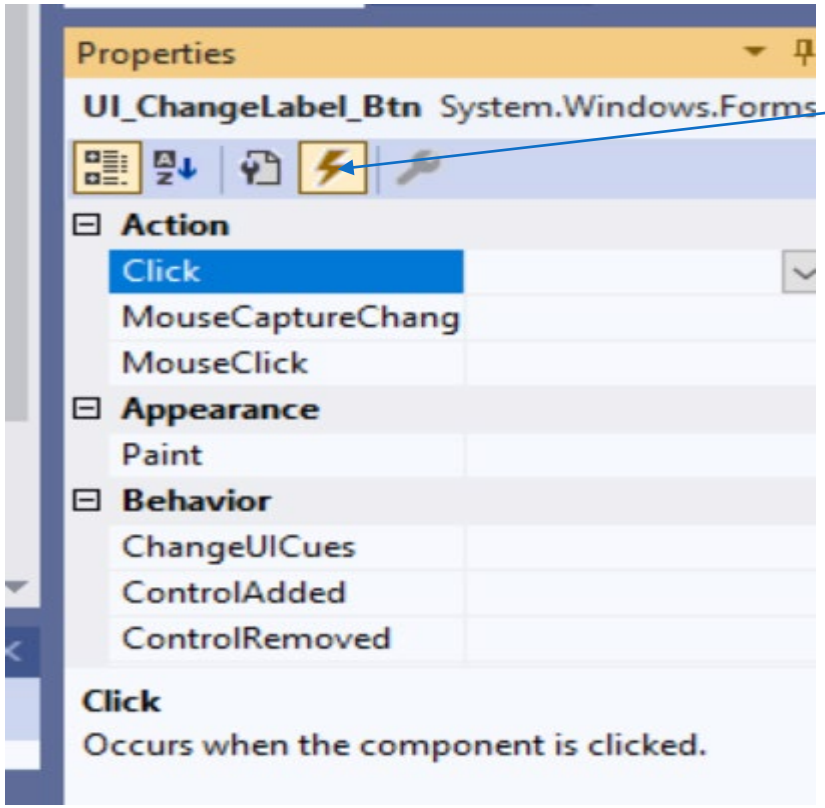
Changing Properties

- Like for the Form, the properties of the controls can be accessed and modified through the property pane or through code in the program



Adding Event Handlers

- Again, like for the form, event handlers can be automatically generated for each control, by using the event icon in the property pane and double-clicking on the event of interest.



Event Icon

Common Event

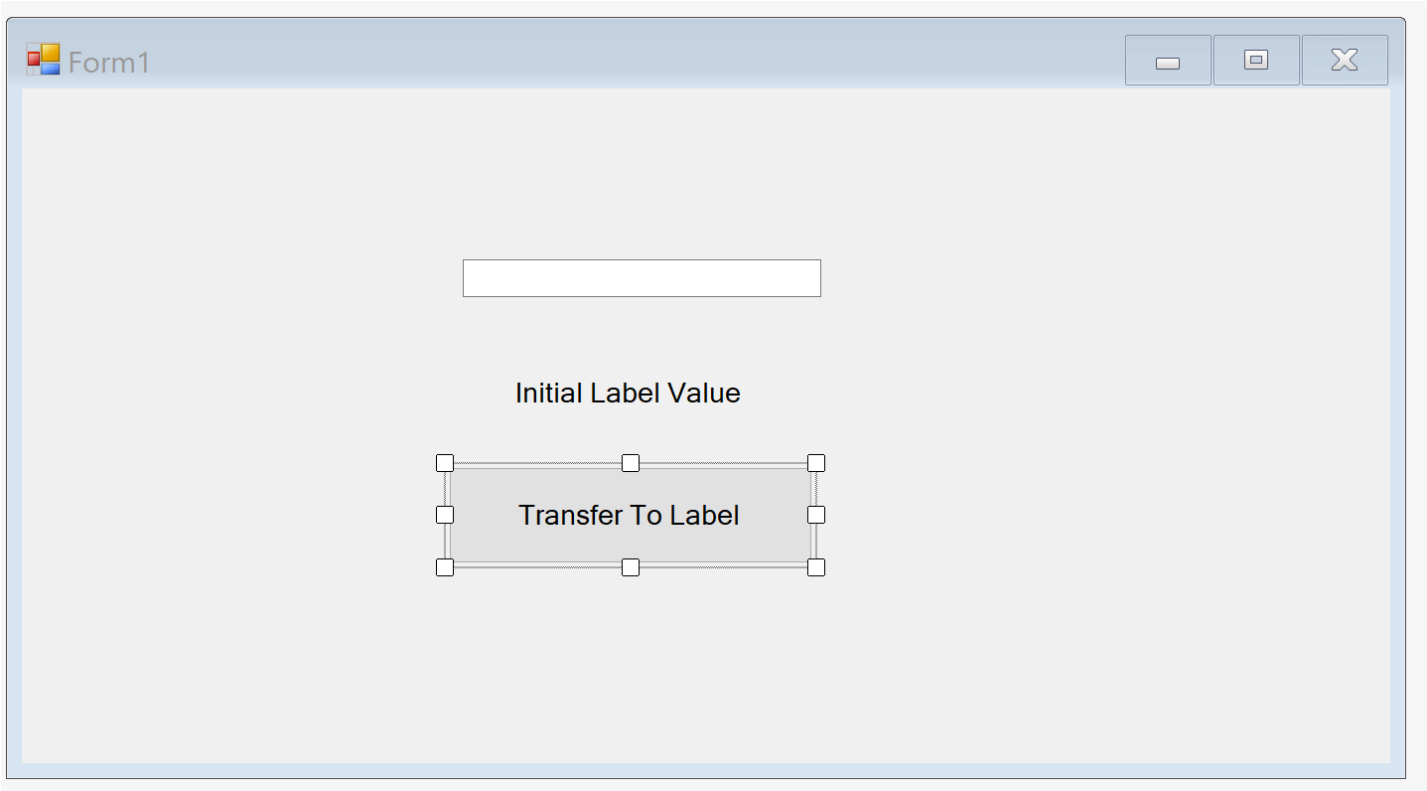
- Each control also has a “**common**” event. The event handler for that event can be generated through double-clicking on the control.

Naming Your Control Components

- It’s important to give meaningful and consistent names to our control components.
- We’ll adopt the following practice for naming the components:
  - All our control components will start with the term **UI_** followed by a word or a combination that indicates the purpose followed by underscore followed by a short form of the type of control, e.g. **UI_Change_Btn**, **UI_Name_Tbx**

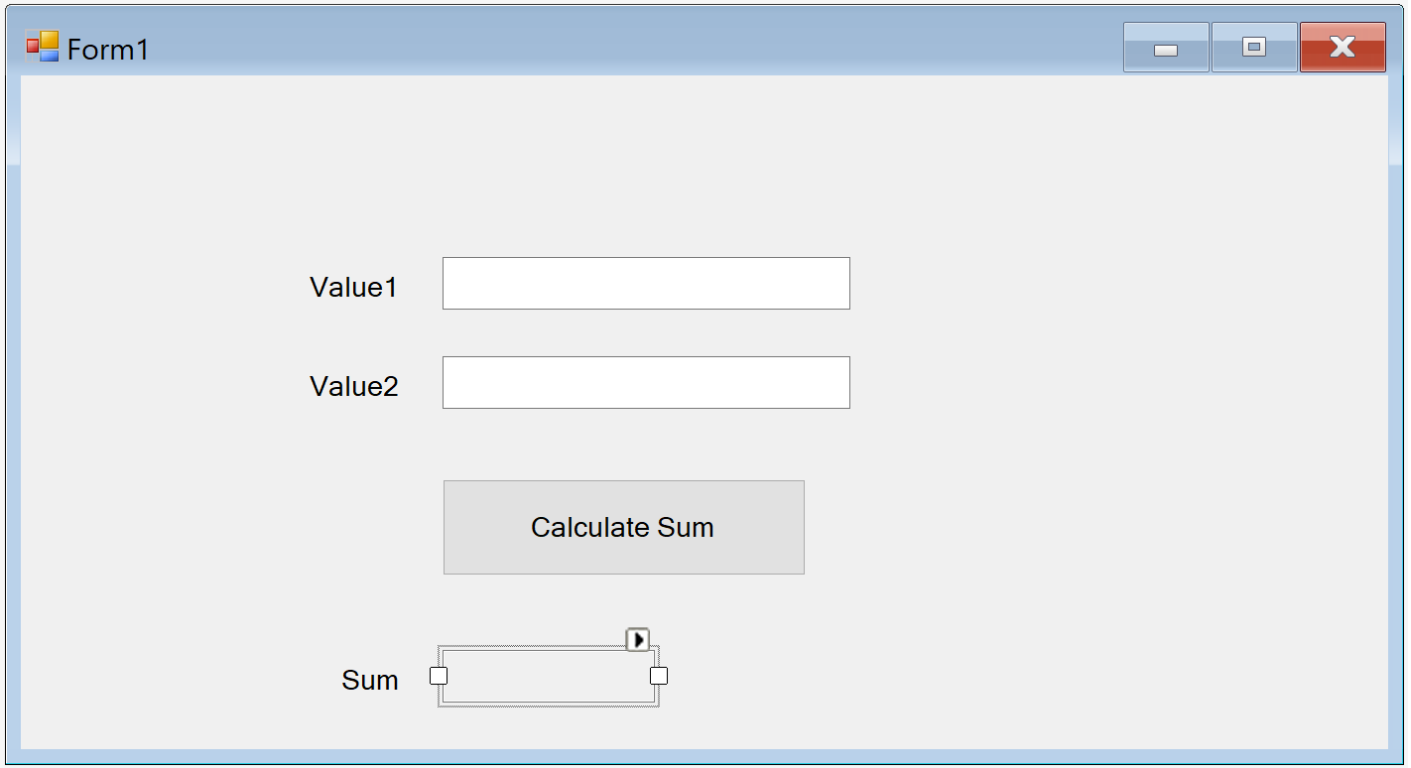
Lecture 2B-Demo2

- Write a Windows Application that contains a textbox, a label and a button as shown.
- When the button is clicked, the value of the textbox is transferred to the label and the textbox becomes empty.
- Choose appropriate names for your controls.



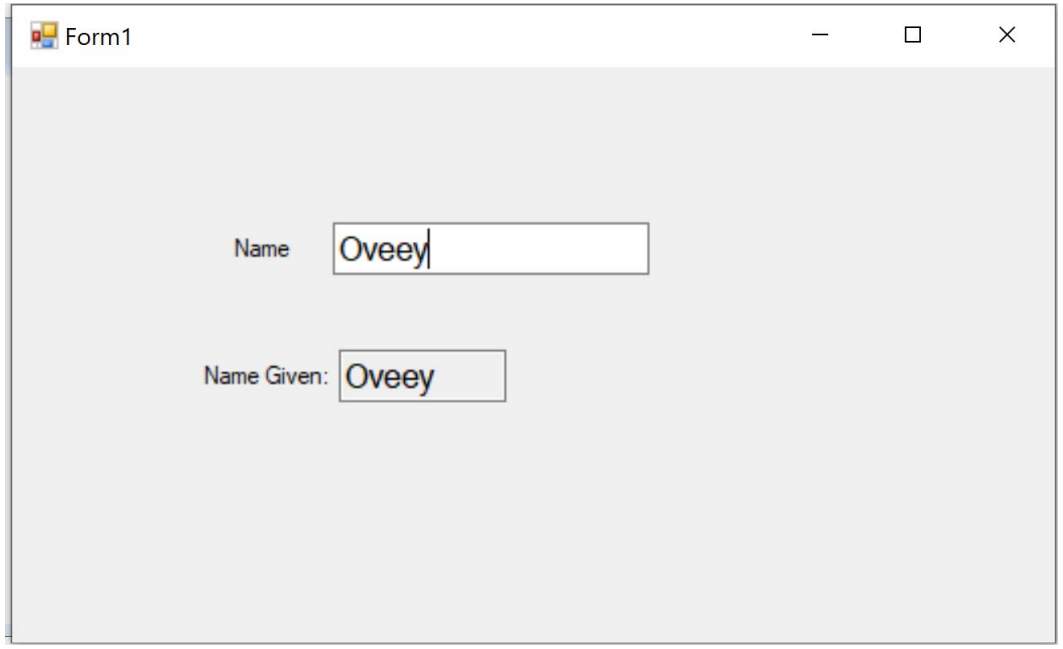
Lecture 2B- Demo 3

- Write a Windows Form application that allows the user to input 2 (numerical) values and displays their sum in a Read-Only text box.



Lecture 2B – Demo 4

- Write a Windows Form Program that allows the user to input a name in a text box.
- As the name is being typed, it also appears in a read-only text box.
- Hint: Use the **TextChanged** event

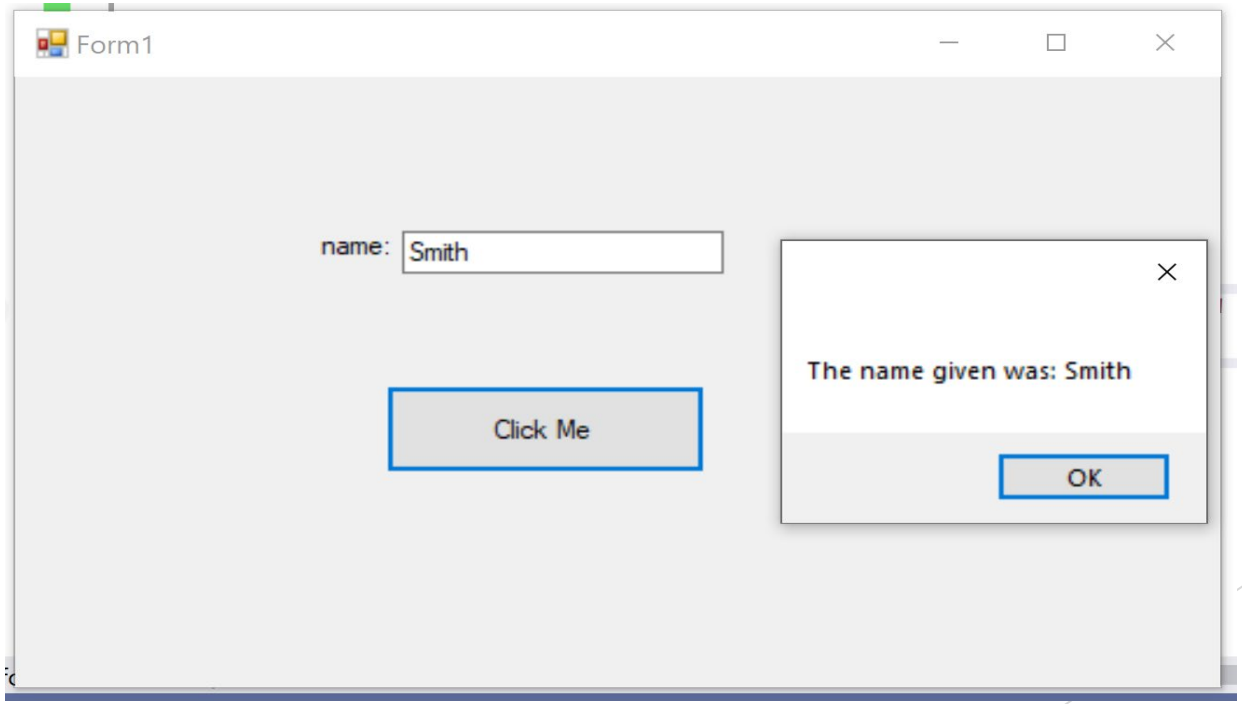


MessageBox

- The MessageBox is not a control, as such.
- However, it’s a useful Tool for displaying error messages.
- There are 4 versions of the MessageBox, with differing number of parameters, given in the Windows Controls slide deck.

Lecture 2B- Demo 5

- Write a Windows Form Program that has a textbox and a button.
- The textbox allows the user to input a name.
- When the user clicks on the button, a message box will pop up, giving the message “The name given was: “ + the name. E.g. “The name given was Smith”



Timer Component

- The Timer component is not a visible control.
- A Timer can be set to fire a **TimerTick** event at specified intervals.
- Timers are often used to implement animation or periodic events.
- A note of caution when working with timers – Always ensure that the **enabled** property is set to true.

Timer Component

Property	Description
Name	The timer name
Enabled	If true, TimerTick events are fired. If false, no TimerTick events occur
Interval	The number of milliseconds to pass between TimerTick events

Timer Component

- The Timer has only one event, Tick, which occurs once for every timing period if the Timer is enabled.
- Double-click on the Timer component (or the Tick event) to add a **Tick** event listener.

Lecture 2B- Demo 6

- In this demo, we are going to include a timer component and change the Form Text to a count value that increments on every timer tick.
- Create a Windows Form application
- Add a timer component
- Change the **enabled** property of the timer to True
- Set the interval property to 200 milliseconds.
- In the **Form1** class include an **int** attribute (variable) called **count**.
- In the form constructor, initialize **count** to 0
- Create an event handler for the tick event of the timer. It should change the Text of the form to display "**count is:** " + the value of count.

Stopwatch

- The **Stopwatch** class allows us to start a stopwatch so as to track time.
- Its part of the **System.Diagnostics** binaries.
- An example of creating a stopwatch, is as follows:

```
System.Diagnostics.Stopwatch sw = new System.Diagnostics.Stopwatch();
```

- The above code must be inserted inside the class where the Stopwatch object will be used.

Stopwatch properties

- The Stopwatch class has number of methods and properties.
- We'll look at 3 of them currently.

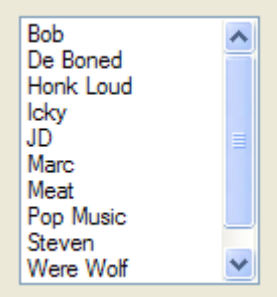
Property/method	Desciption
Start()	Starts the stopwatch
Stop()	Stops the Stopwatch
EllapsedMilliseconds	The number of milliseconds that have elapsed since the stopwatch started

Lecture2B-Demo 6B- Including a stopwatch

- Modify the program for Demo 6 as follows:
  - Add a stopwatch object to the **Form1** class
  - In the Form load event handler, start the stopwatch
  - In the Form Text, in addition to the count, display the number of milliseconds elapsed
  - Include a button in the Form. Add an event handler so that when the button is clicked, the stopwatch stops.

ListBox

- The ListBox will display a list of items from which the user can select one, or multiple items.



- However, currently, we are only interested in adding items to the list box and clearing the list box

List Box

- Items can be added during program execution using Items.Add

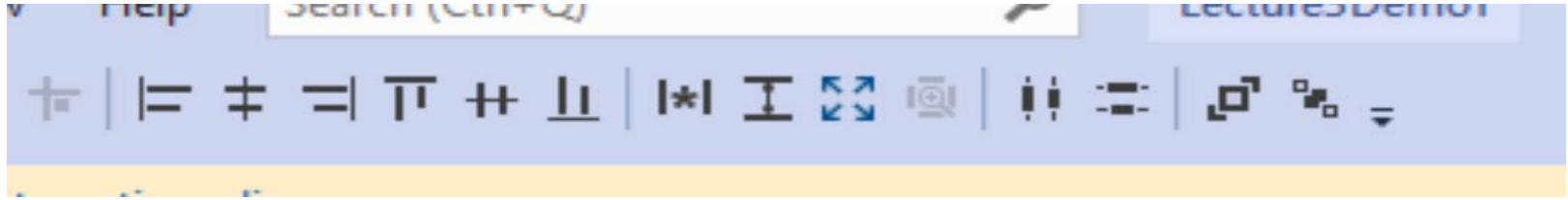
```
listBox1.Items.Add("New Item");
```

Lecture2B- Demo6C

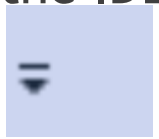
- Modify the program for Demo 6, so that:
  - It also includes a list box
  - every time the **count** variable reaches a multiple of 20, the value is added to the list box

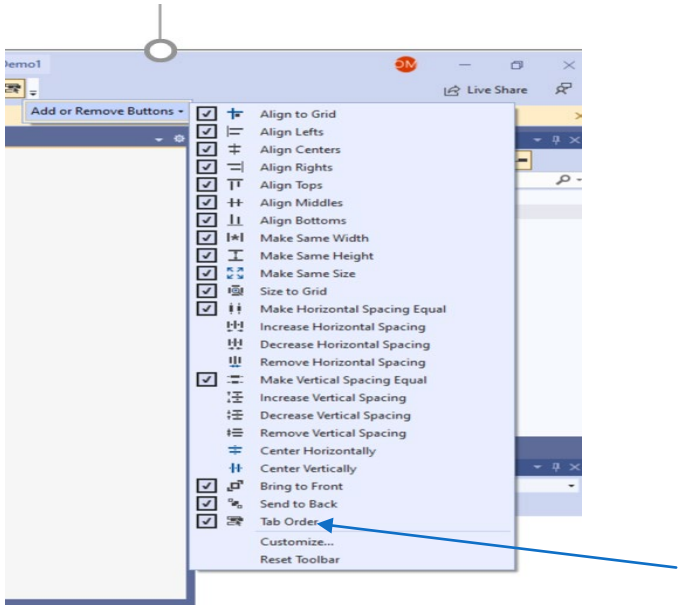
Professional Design Layout

- We want our design to have a professional layout. This implies:
  - Alignment of controls (horizontally or vertically)
  - Consistent size among the same or similar controls
  - Having consistent spacing between controls
- The following set of tools help us with developing a consistent professional layout.



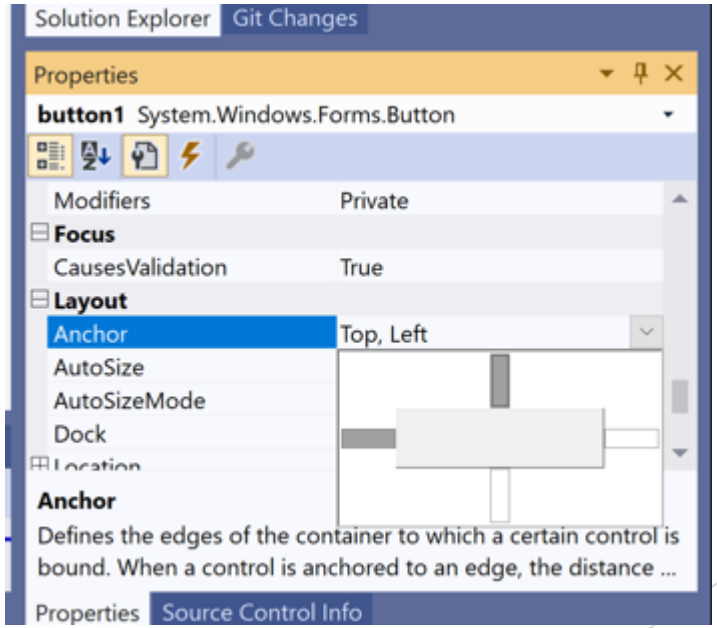
Tab Order

- The controls on your form will have a tab order.
- This is the order in which focus will move when you press the tab key.
- The default tab order will be the order in which we have added the controls.
- To change the tab order, we use the “Tab Order” button. 
- Unfortunately this button doesn’t appear by default on the IDE, but we can add it through the “Add/Remove Buttons” tool. 



Anchoring

- Anchoring consists of pinning our controls to the sides of our form.
- It determines the positioning of our controls as the size of our form changes.
- The default anchoring is top and left, meaning that our control is pinned to the top and left of our form.
- We use the anchor property to change it.



Watch the Video on Layout, **Tab Order and Anchoring**

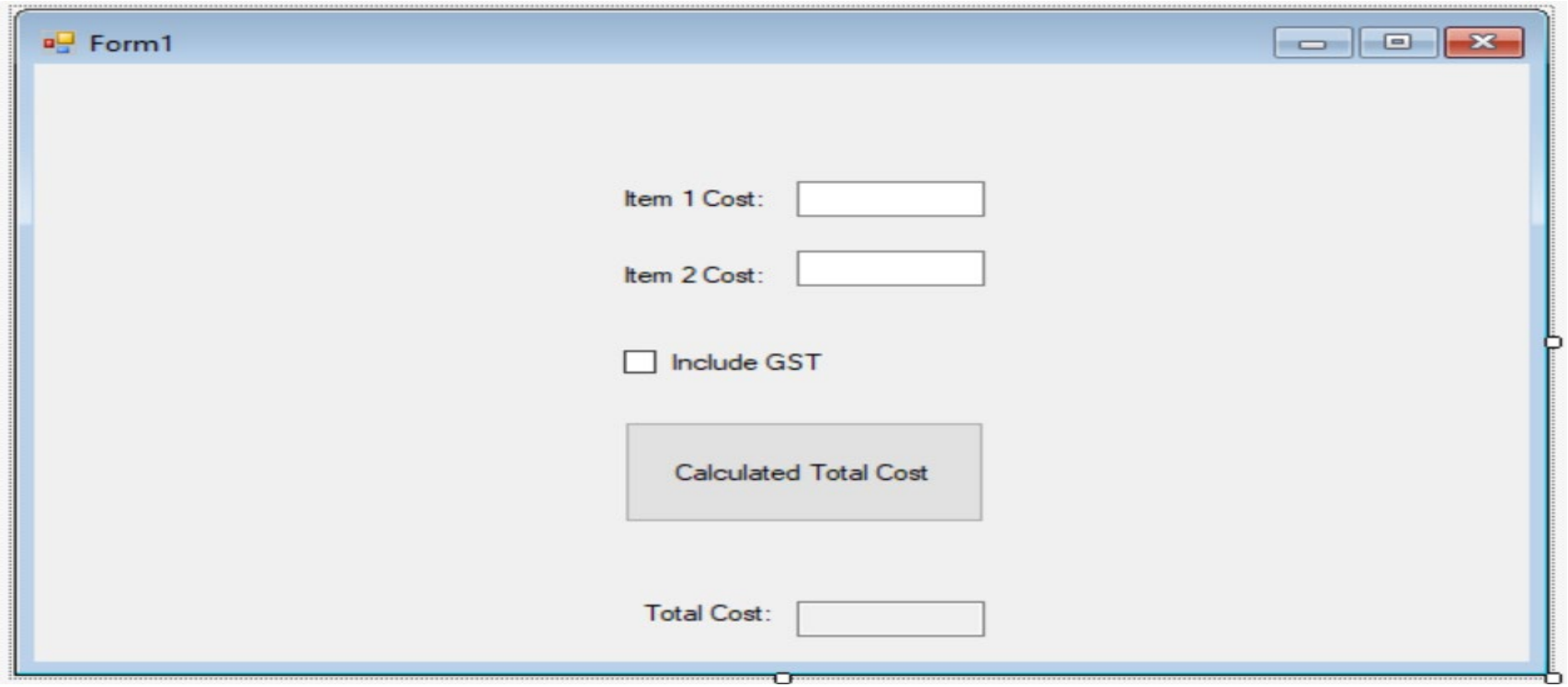
CheckBoxes

- The CheckBox can be used to make true/false selections.

- The **Checked** property specifies whether the CheckBox is checked.
- The common event for a checkbox is the **CheckedChanged** event.

Lecture 2B- Demo 7

- Develop an application with the UI below.



- For uniformity with my code, it's recommended that you name the textboxes UI_Item1Cost_Tbx, UI_Item2Cost_Tbx and UI_TotalCost_Tbx
- Name the button UI_TotalCost_Btn and the checkbox UI_GST_Cbx
- Implement the method and event handler requested in the next slide.

Lecture 2B- Demo 7-contd

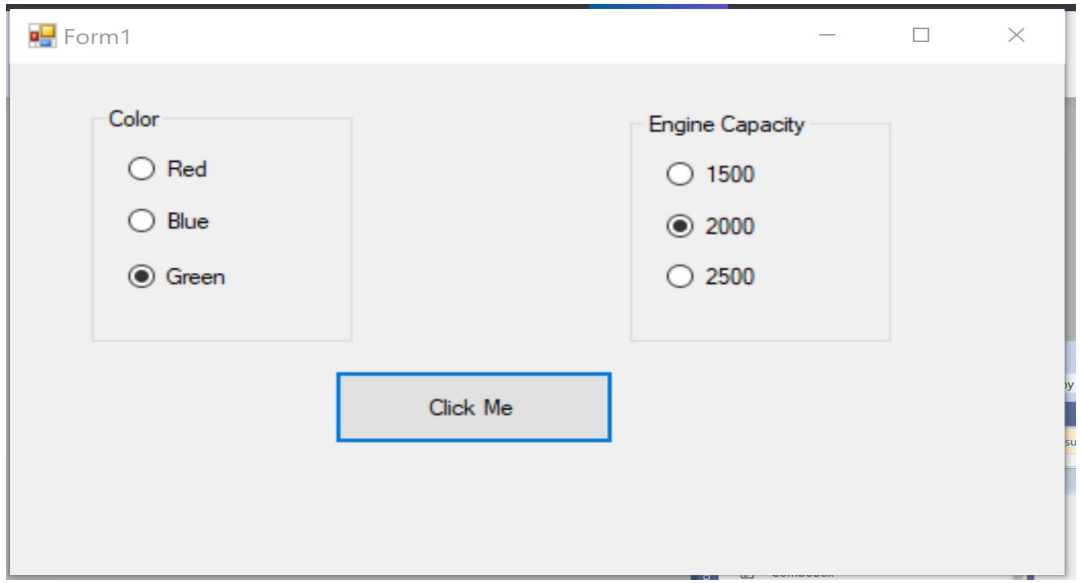
- Write a method **CalculateTotalCost()** that has as parameters 2 strings and a boolean. It converts the 2 string values to decimal type (you can assume the values are valid) and it calculates and returns the sum of the 2 values. If the boolean value is true, it adds another 5% of the total as GST.
- Write an event handler for the **Click** event of the button and another one for the **CheckedChanged** event of the checkbox so that they each call **CalculateTotalCost()** passing as arguments the values in the 2 textboxes and the checked status of the checkbox. It displays the result in the ReadOnly textbox.

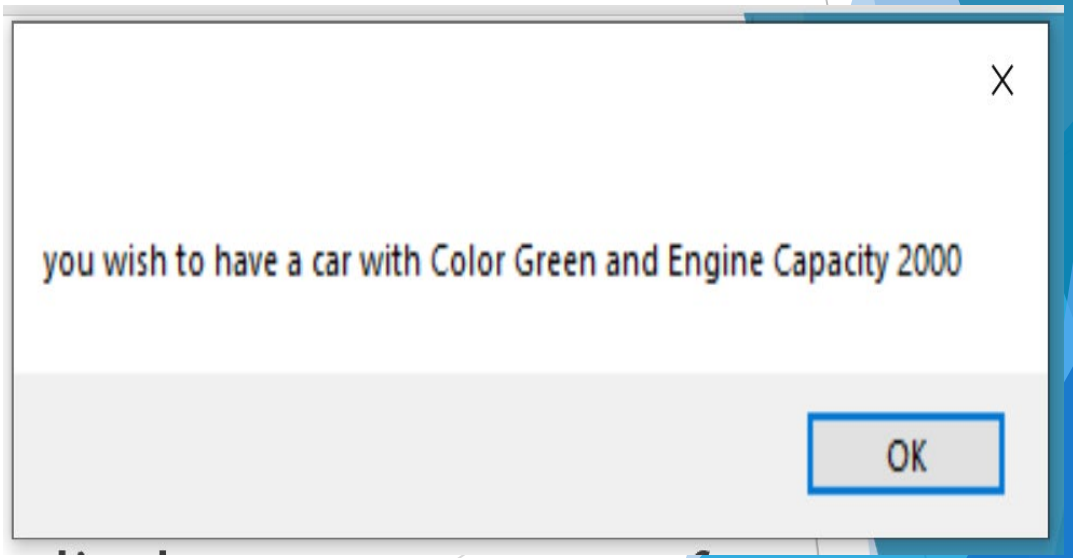
Radio Buttons

- Radio Buttons are different from the other controls as only one of them can be in the checked state at a time.
- We can verify the **Checked** property of a radio button to know whether it's checked.
- The common event for Radio Buttons is **CheckedChanged**
- Since only one Radio Button can be in the Checked state at a given time, if we want to have multiple sets of Radio Buttons, we have to group each set under a GroupBox.
- **GroupBox** is found under the list of **containers**.

Lecture 2B- Demo 8

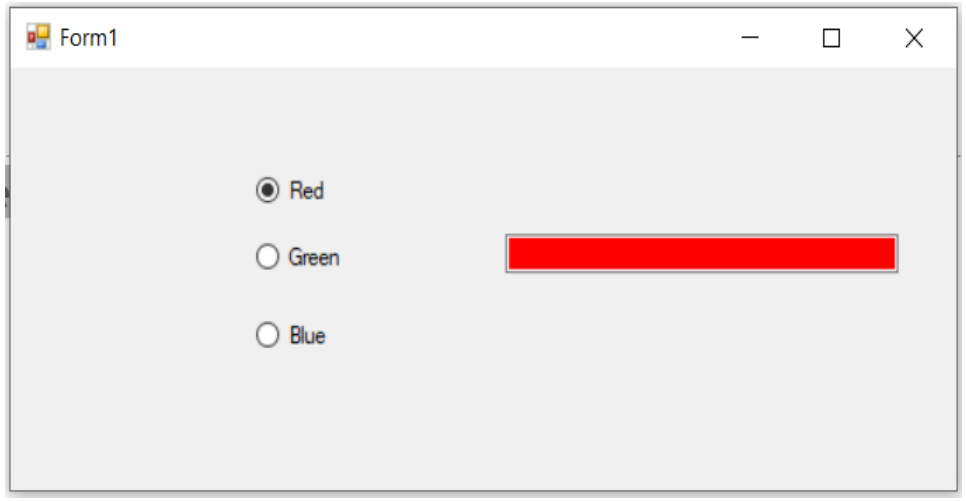
- Write a Windows Form Program that displays two groups of radio buttons- One for a color and another for engine capacity of a car.
- The form must also have a button. When the button is clicked, a message box appears giving the car color and engine capacity desired.





Event Consolidation

- Consider the application with the UI below.
- When the user clicks on one of the radio buttons, the color Changes based on the button that is clicked.



- We can have one event handlers for the CheckChanged event of each button as shown in the next slide.
- However, we can instead have just one event handler that is used for all the 3 events. This is known as event consolidation

Event Consolidation

- The three event handlers for the 3 boxes would be as below

```
private void UI_Red_Radio_CheckedChanged(object sender, EventArgs e)
        {
           if (UI_Red_Radio.Checked)
               UI_RdOnly_Tbx.BackColor = Color.Red;
        }
        private void UI_Green_Radio_CheckedChanged(object sender, EventArgs e)
        {
            if (UI_Green_Radio.Checked)
                  UI_RdOnly_Tbx.BackColor = Color.Green;
        }
        private void UI_Blue_radio_CheckedChanged(object sender, EventArgs e)
        {      if (UI_Blue_Radio_Checked)
                     UI_RdOnly_Tbx.BackColor = Color.Blue;
        }
```

Event Consolidation

- Instead of 3 event handlers, we can have just one event handler handling all 3 cases.
- This is known as event consolidation

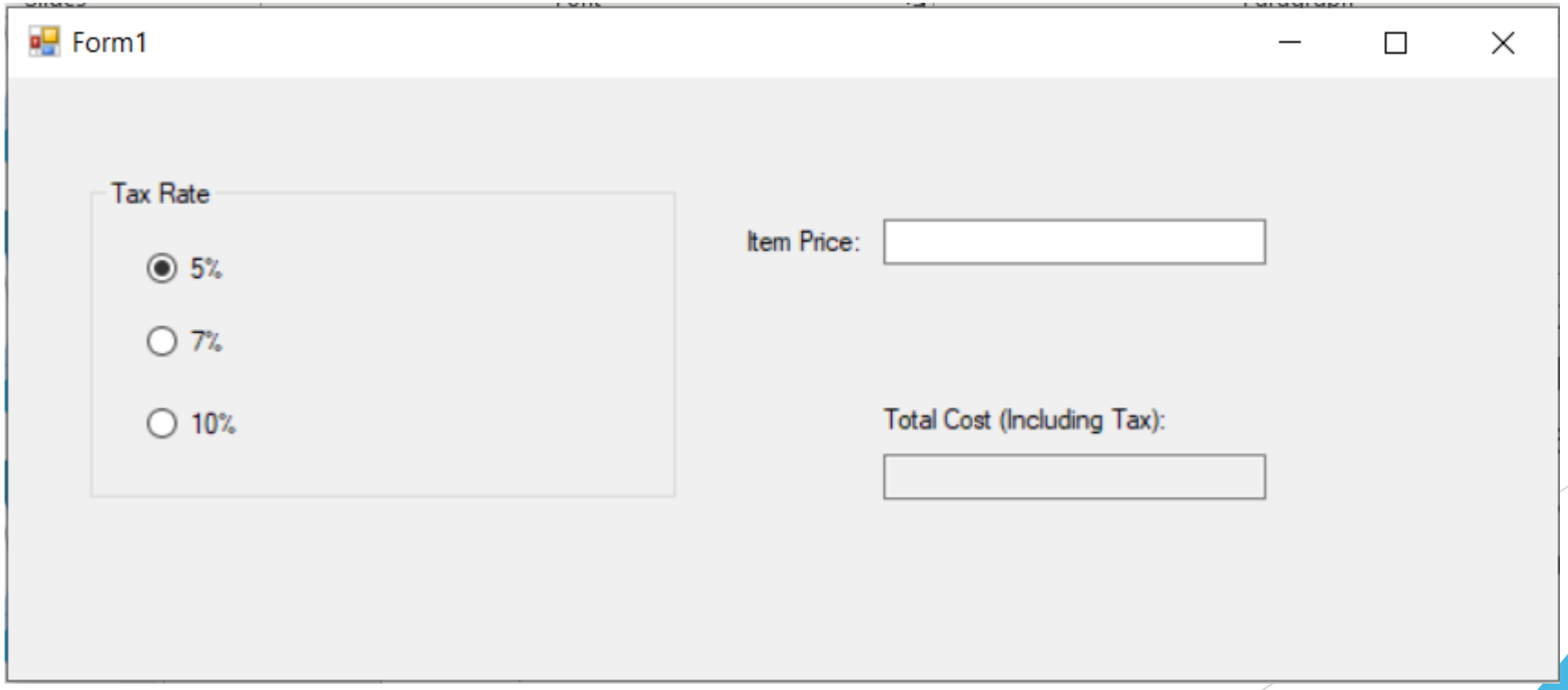
```
private void UI_Radio_CheckedChanged(object sender, EventArgs e)
{
      if UI_Red_radio.Checked
         UI_RdOnly_Tbx.BackColor = Color.Red;
      else if UI_Green_radio.Checked
          UI_RdOnly_Tbx.BackColor = Color.Green;
      else
         UI_RdOnly_Tbx.BackColor = Color.Blue;
      }
```

Event Consolidation

- Even if the benefit of a single event handler may not be obvious in the shown example, it may in general avoid having the same code repeated in each event handler.

Lecture2B-Demo 9

- Write a form-based application with the control below. As the user enters the value in the upper textbox, if the value is a valid double, the program must calculate the value including tax and display in the Read-Only textbox. The tax is based on the radio button that is checked. If the user clicks on a different radio button, the total value must again be recalculated accordingly. Use event consolidation.



Adding Event Handlers Manually in Code

- We can register event handlers directly in the code (without using the designer), through the steps below:
  - We type in the name of the control followed by “.”
  - Select the required event from the pop-up list
  - Type in +=<tab><tab> (i.e press the “tab” key twice).
- Code for registering event handlers are placed in the Form constructor

Lecture2B- Demo10

- Create an application with the UI shown.
- In the code, manually add an event handler for the **CheckedChanged** event of the checkbox, so that it will first verify that the Name and Course textboxes are not empty.
- If both textboxes are not empty, it will verify if the checkbox is checked it will display in the output label: <Name> enrolled for <course>. Otherwise it will display <Name> not enrolled for course.
- If one or more of the textboxes is empty, inform the user accordingly using a messagebox.