CMPE 1666- Intermediate Programming

Lecture 2- Windows Forms Applications

(Acknowledgements- J.D. Silver)

Event Based Programs

- A console application starts execution at Main and runs sequentially to the end of the program.
- Console applications interact with the operating system via Read and Write calls.
- A Windows Form application also has a main program, which runs in a process loop passing events to the form.

Event Based Programs

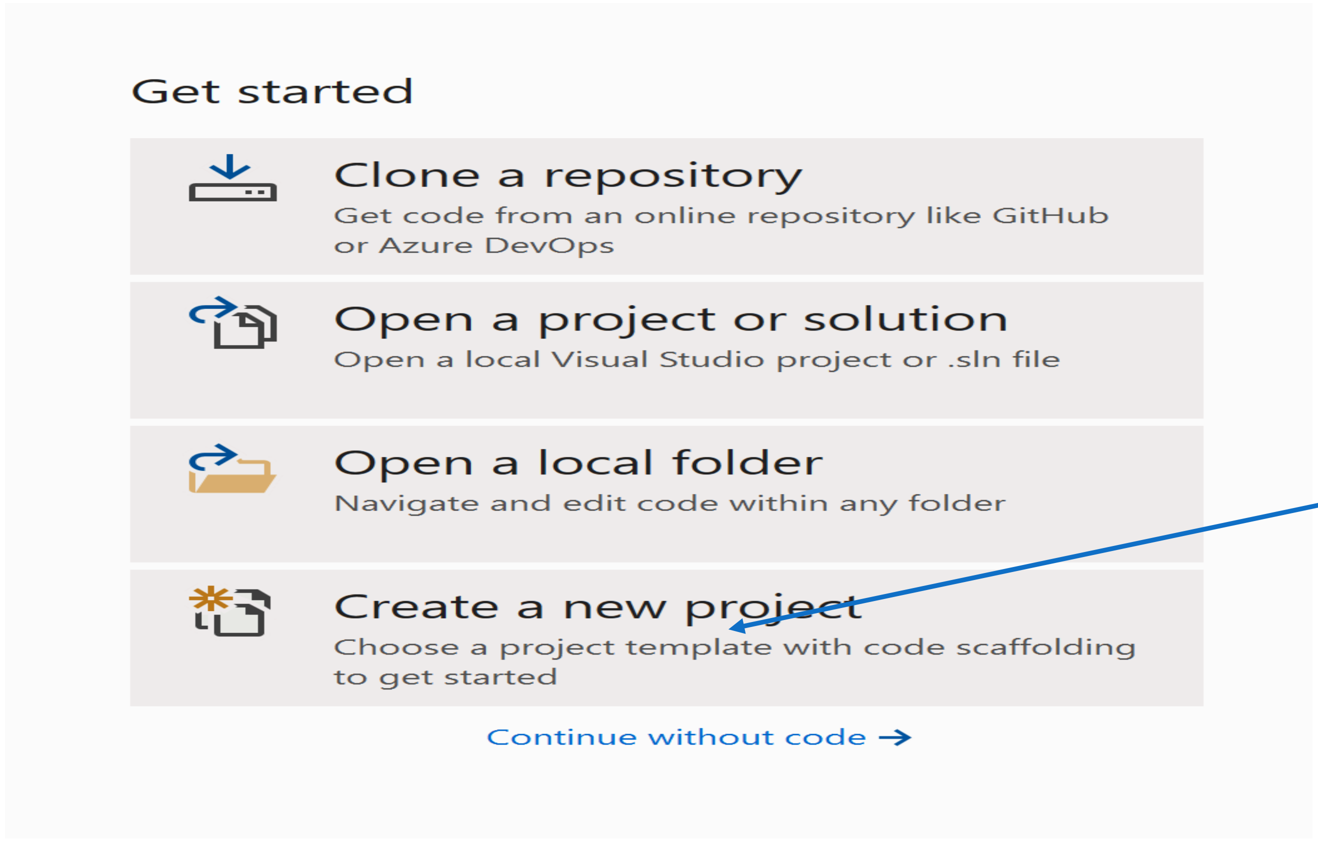
- An event is a notification from the OS that an action has taken place:
  - The form has been loaded
  - The mouse has moved
  - A key is pressed.

Working with our first Form Demo

- To understand the concepts related to Forms, we will create a Form-Based Demo called Lecture2Demo1, by following the steps in the upcoming slides.

Creating A Form-Based Application

- In Visual Studio, A Windows Form Based Application is created in a similar way to a Console Application



Get started

Clone a repository
Get code from an online repository like GitHub or Azure DevOps

Open a project or solution
Open a local Visual Studio project or .sln file

Open a local folder
Navigate and edit code within any folder

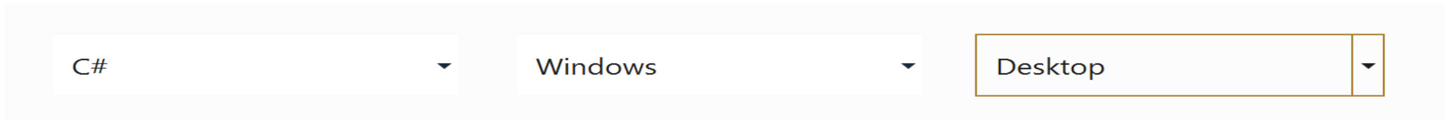
Create a new project
Choose a project template with code scaffolding to get started

Continue without code →

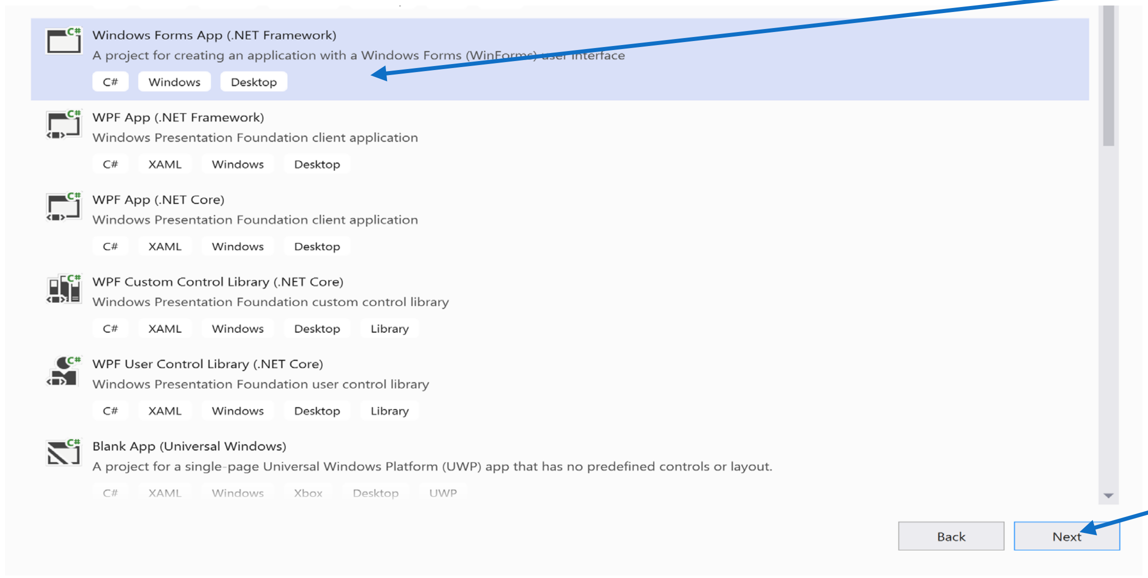
Click on "Create New Project"

Creating A Form-Based Application

- On the next screen Choose C#, Windows and Desktop from the 3 dropdown boxes.



- Then among the options available, choose “**Windows Forms App (.NET Framework)**” and click on **Next**



Windows Forms App (.NET Framework)
A project for creating an application with a Windows Forms (WinForms) user interface
C# Windows Desktop

WPF App (.NET Framework)
Windows Presentation Foundation client application
C# XAML Windows Desktop

WPF App (.NET Core)
Windows Presentation Foundation client application
C# XAML Windows Desktop

WPF Custom Control Library (.NET Core)
Windows Presentation Foundation custom control library
C# XAML Windows Desktop Library

WPF User Control Library (.NET Core)
Windows Presentation Foundation user control library
C# XAML Windows Desktop Library

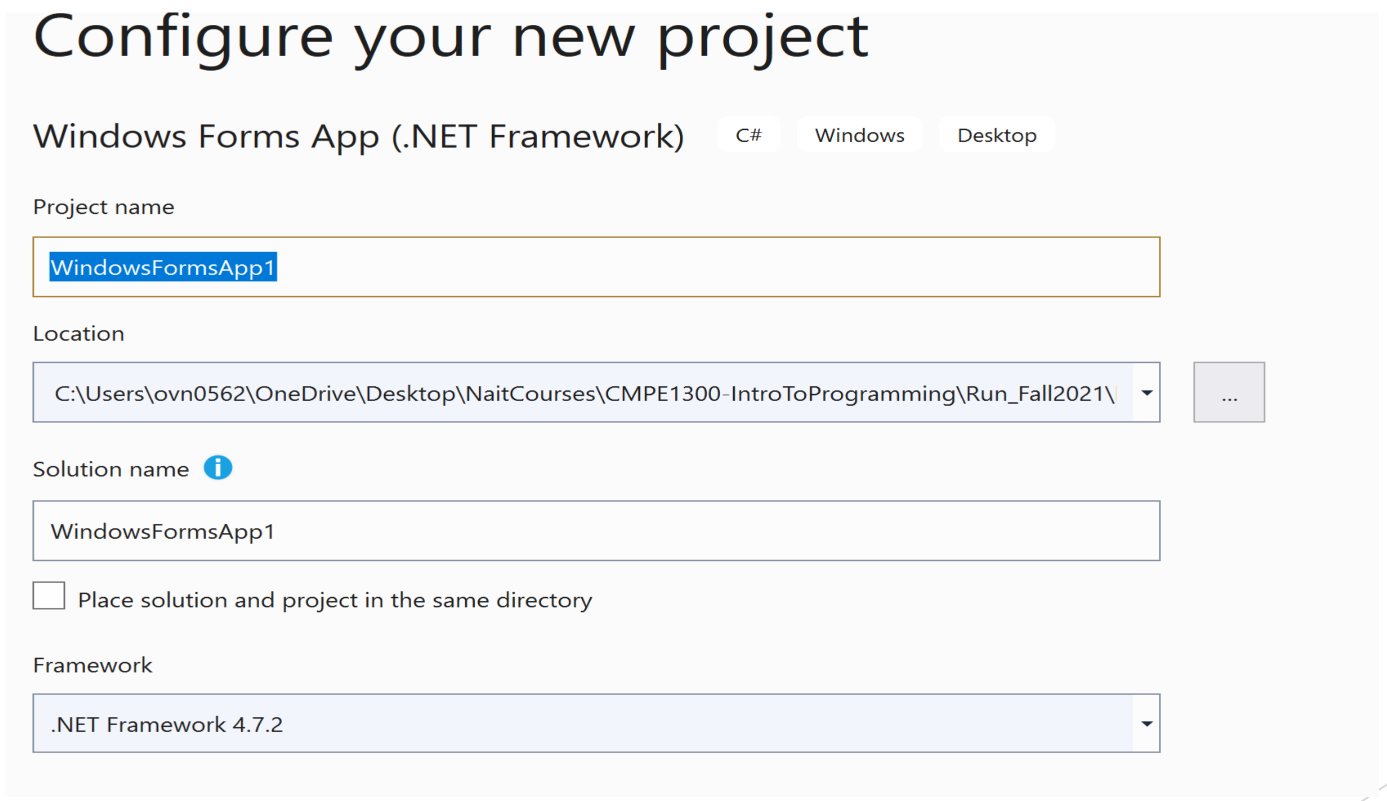
Blank App (Universal Windows)
A project for a single-page Universal Windows Platform (UWP) app that has no predefined controls or layout.
C# XAML Windows Xbox Desktop UWP

Windows Forms App(.NET Framework)

Next

Creating A Form-Based Application

- Finally choose the folder and give the project a name as you would do for a console application.



Configure your new project

Windows Forms App (.NET Framework) C# Windows Desktop

Project name

WindowsFormsApp1

Location

C:\Users\ovn0562\OneDrive\Desktop\NaitCourses\CMPE1300-IntroToProgramming\Run_Fall2021\

Solution name

WindowsFormsApp1

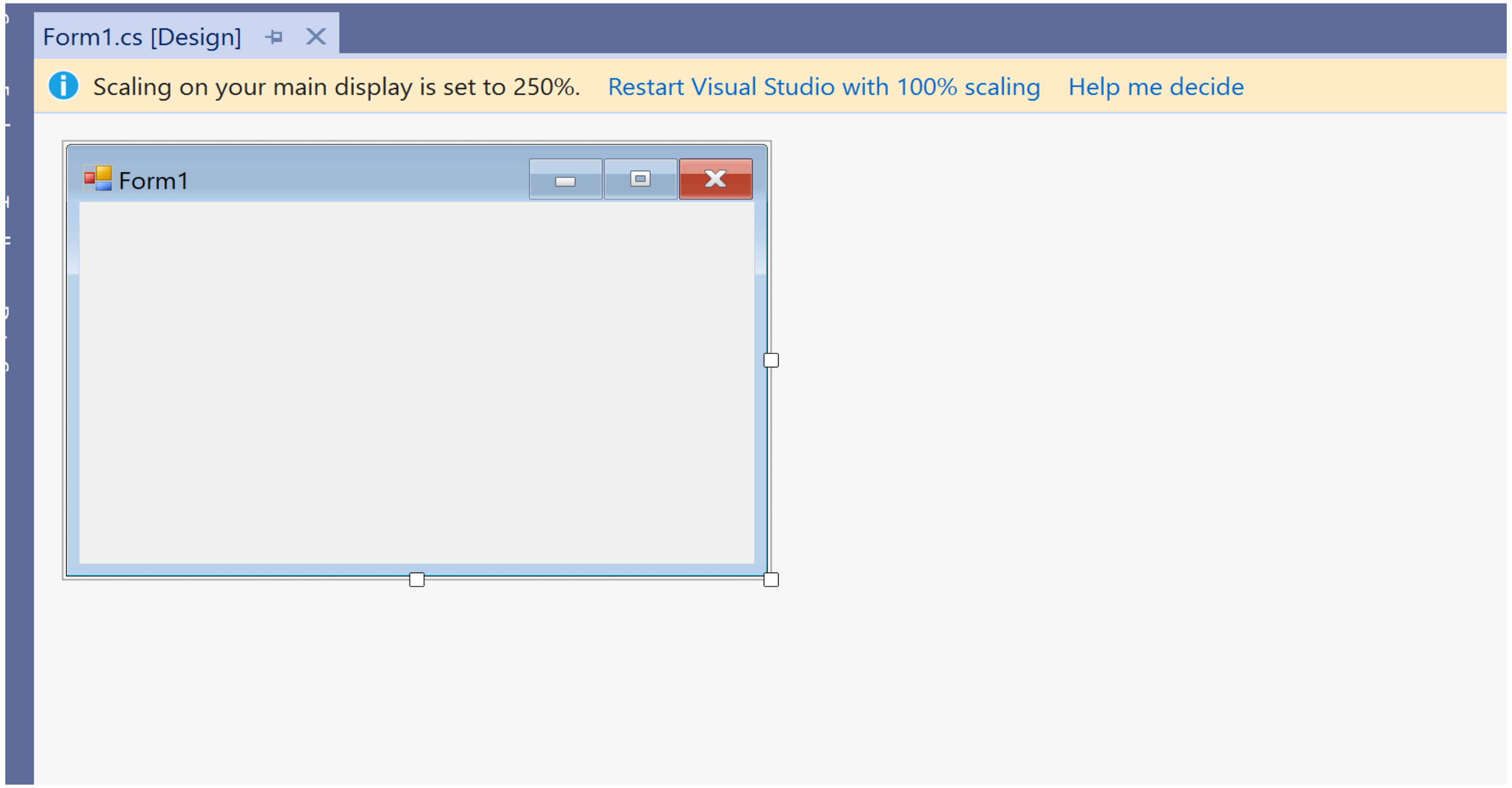
☐ Place solution and project in the same directory

Framework

.NET Framework 4.7.2

Creating A Form-Based Application

- When the application is created, we get a form design.



Form1.cs [Design]

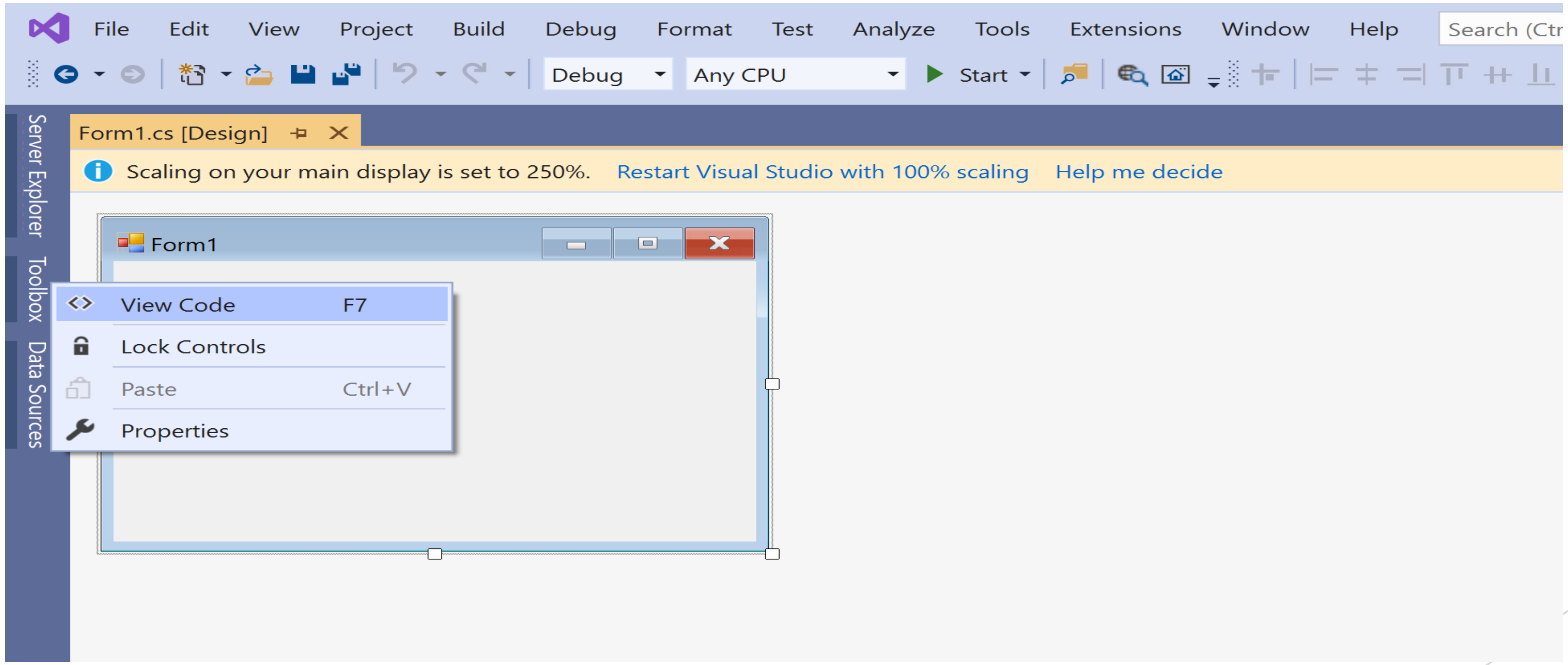
Scaling on your main display is set to 250%. Restart Visual Studio with 100% scaling Help me decide

Form1.cs [Design]

- The Form1.cs[Design] file gives a graphical representation of the form.
- You will drag and drop controls onto the form, and the controls will be added to files as required.

Form Code

- The code for the form resides in the file Form1.CS
- To access the code of Form1.cs, right-click on the form design and select "view code".



Scaling on your main display is set to 250%. Restart Visual Studio with 100% scaling Help me decide

Form Code

- .
- The code does not run sequentially, except within the event handler methods.
- The order of the events may vary, depending on the situation.

Form1.cs

- The code for a form resides within a class for that form in the file Form1.cs.
- This is where you will place your code.
- The code will be organized as event handlers, which are methods that are called by Windows when an event takes place.
- You may also use other methods that are called from the event handlers.

Form1.cs

- Form1.cs contains a partial class called Form1, inherited from class System.Form.
- It already contains a Form **Constructor** which performs initializations for the form.
- The class Form1 is dispersed over The Form1.cs file and the Form1Designer.cs file.

Form1.Designer.cs

- Contains the code generated by Visual Studio when you modify the form or add controls to it.
- Do not modify this code by editing it.
- Use the graphical tools to make modifications, and Visual Studio will adjust the code.

Program.cs

- This is the main program to launch the form.
- Execution starts by setting visual styles and a text format, then calls Application.Run and constructs the form.
- By default, the form is called Form1, although that name can be modified.

Windows Forms Properties

- There are many properties that can be modified for a form object.
- These properties can be modified at design time through the property editor, or during run time by writing program code.

Form Class

- Every form (window) is based on a class which contains member variables and methods.
- Since the form is created as an object by **Main**, the member methods and variables are usually **non-static**.
- A form uses properties to modify its appearance, and event handlers to respond to actions.

Windows Forms Properties

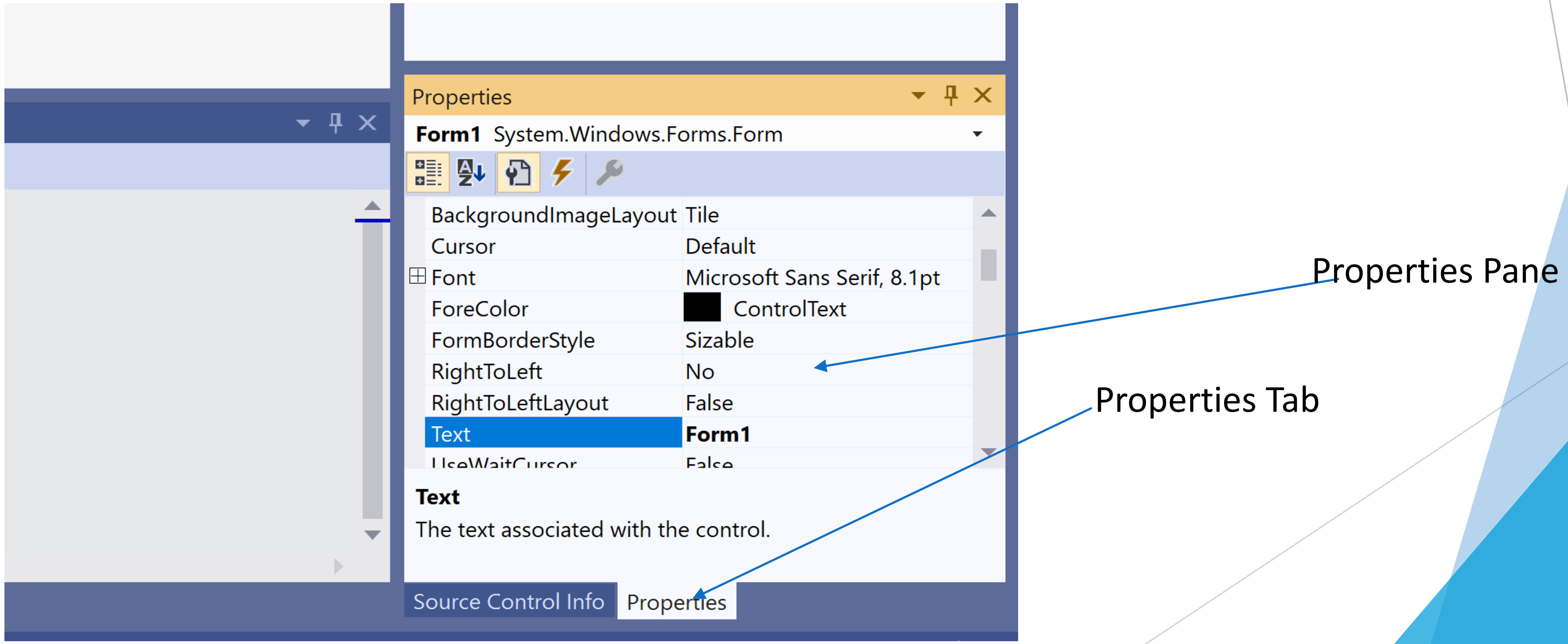
Property	Description
Text	Caption of the form.
FormBorderStyle	Determines if the form can be resized and styled.
Font	Font used for text on the form.
BackColor	Background color for the client area.
MinimizeBox	Determines if minimize box is shown.
MaximizeBox	Determines if maximize box is shown.
Size	Size of the client area in pixels.

Windows Forms Events

- There are many events that a form can respond to with an event handler.
- We will examine the more common form events.

Accessing the properties and events

- To access the properties and events on the form, we use the properties tab at the lower-right part of the screen to obtain the “properties” pane.

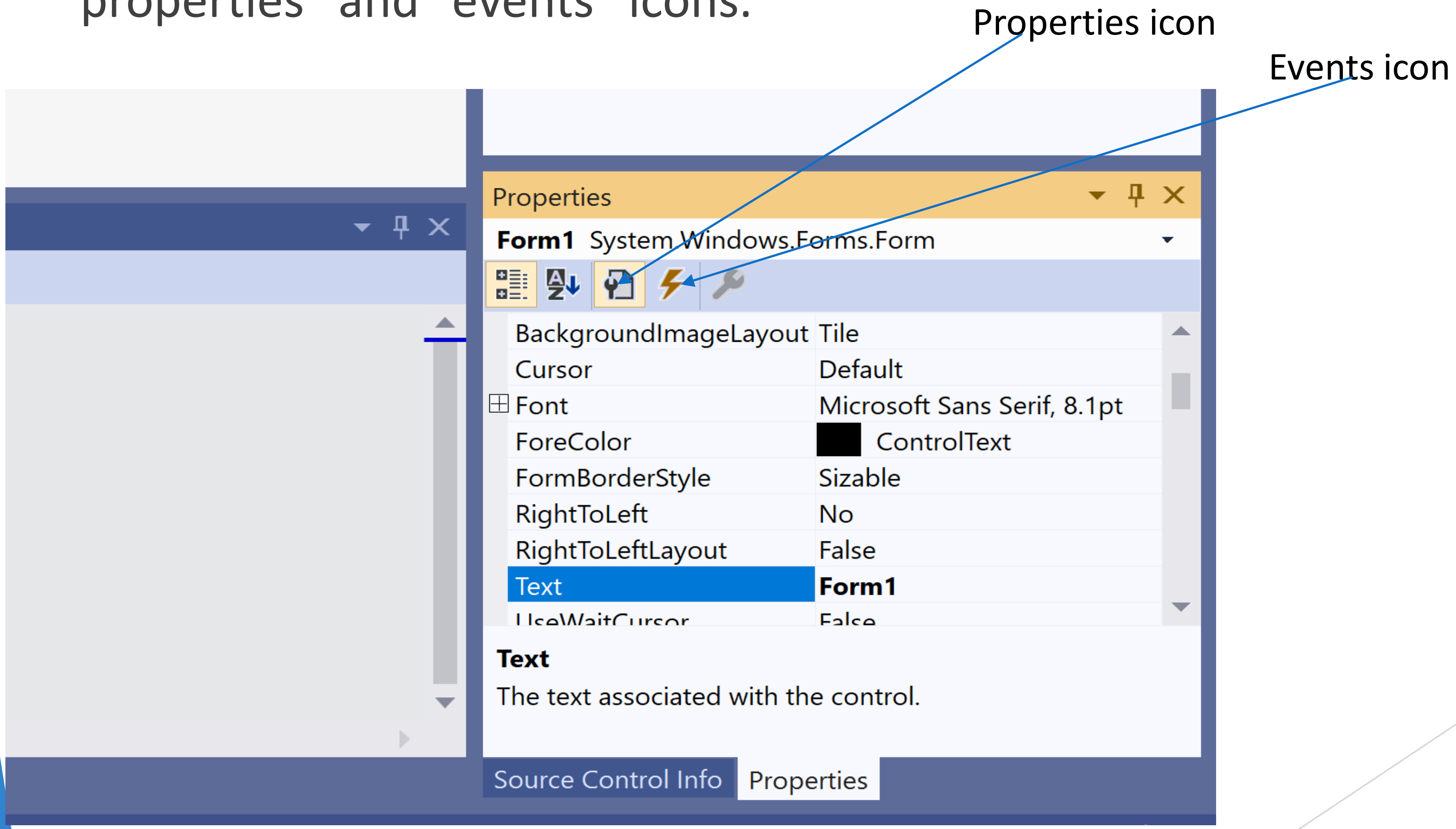


BackgroundImageLayout	Tile
Cursor	Default
⊞ Font	Microsoft Sans Serif, 8.1pt
ForeColor	■ ControlText
FormBorderStyle	Sizable
RightToLeft	No
RightToLeftLayout	False
Text	**Form1**
UseWaitCursor	False

Text
The text associated with the control.

Accessing the properties and events

- We obtain access to the properties and events through the "properties" and "events" icons.



Form1 System.Windows.Forms.Form

BackgroundImageLayout	Tile
Cursor	Default
Font	Microsoft Sans Serif, 8.1pt
ForeColor	■ ControlText
FormBorderStyle	Sizable
RightToLeft	No
RightToLeftLayout	False
Text	Form1
UseWaitCursor	False

Text
The text associated with the control.

Lecture2-demo 1b

- In Lecture2Demo1, change the Form Text To “Windows Forms Demo1”.
- Run the project again and observe the form title.
- Change the MinimizeBox property to false
- Run the project and note that the Minimize button has disappeared.
- Change the MaximizeBox property to false
- Change the background color to HightLight
- Run the project and note that the Maximize button has also disappeared.

Event Handlers

- An event handler is a **non-static** method that has been linked to a Windows event using a delegate.
- The delegates are created automatically in Form1.Designer.cs
- To create an event handler, you double click on the event shown in the Properties pane of Visual Studio

Event Handlers

- An example of an event handler for Form1 Load is shown below:

```
private void Form1_Load(object sender, EventArgs e) {
    System.Diagnostics.Trace.WriteLine("Form1 is loaded")
    }
```

Lecture2-Demo1C

- To Lecture2Demo1, add an event handler for the Load Event of the Form.
- In the event handler, write the following statement:

```
Console.WriteLine(“Form1 is loaded");
```

- Run the program using the “**start Debugging**” button and observe the output in the Output pane.

Event Handlers

- Every event handler has two parameters:
  - object sender is a reference to the object that sent the event.
  - EventArgs e is an object that contains additional data about the event. The data type of this parameter varies depending on the event handled.

Event Handlers

- Never call an event handler from your own code. Only Windows should call the event handler.
- Event handlers are non-static methods, so they cannot access static variables.

Load

- The Load event occurs when the form object is first created (loaded) by Windows in memory.
- Event handlers for Load usually set up resources or the initial state for controls on the form.
- Load is the common event for a form, and the event handler can be added by double clicking on the form.

Shown

- The Shown event occurs only once when the form is first shown on the screen.
- This event can be used to initialize controls on the form after gaining access to data brought in by the form load event.

Paint

- The Paint event occurs whenever the contents of the form’s client area need to be redrawn.
- This event occurs whenever the form is uncovered, maximized, or the size is changed.
- The event handler draws the contents of the client area as needed

FormClosing

- The FormClosing event occurs as the form is closing.
- This event may be initiated by the user or the operating system.
- The value **e.ClosingReason** provides the reason for the closure.
- This event is often used to intercept the form before it closes to prompt the user to save their work to a file.

FormClosing

- If the closing reason is **CloseReason.UserClosing** then you may wish to intercept the event.
- You can prevent the form from closing by setting the **e.Cancel** property to true in **the FormClosing** event handler.

FormClosed

- This event occurs when the form has been closed, and specifies the closing reason.
- This event would be used to release resources used by the form.

Mouse Events

- Mouse events occur on a form when the mouse cursor is in the client area of the form.
- The event arguments provide the mouse cursor location, and the status of the mouse buttons.

MouseMove

- This event occurs when the mouse is moved in the client area of the form.
- The position of the mouse is provided by **e.X** and **e.Y**
- The status of the mouse buttons are provided by **e.Button** as a value of **MouseButtons.Left** etc.

MouseClick/MouseDoubleClick

- The **MouseClick/MouseDoubleClick** event occurs when any mouse button is clicked/double-clicked in the client area.
- The position of the mouse is provided by **e.X** and **e.Y**
- The status of the mouse buttons are provided by **e.Button** as a value of **MouseButtons.Left** etc

MouseEnter/ MouseLeave

- The **MouseEnter/MouseLeave** event occurs when the mouse cursor enters/leaves a window.
- On a form, this event occurs when the mouse enters/leaves the client area.

Console.WriteLine

- Normally, console input/output is not used in a Windows Form application.
- The **Console.WriteLine** method is available but will not send the output to the form.
- Console.WriteLine only produces output in the Output window of Visual Studio for diagnostic purposes.

```
        Console.WriteLine(“Form1 is loaded");
```

- The following statement displays the value stored in iCount to the Visual Studio output window:

```
    Console.WriteLine($”iCount is {iCount}");
```

- An alternative way of writing the same thing is:

```
     Console.WriteLine(”iCount is {0}“,iCount);
```

Diagnostic.Trace.WriteLine

- **Console.WriteLine()** offers buffered output. In a multithread application the order of outputs is not really predictable.
- Instead we can use the **System.Diagnostics.Trace.WriteLine()**
- The following statement sends output to the Visual Studio output window:

```
System.Diagnostics.Trace.WriteLine("Form1 is loaded");
```

- It works with Similar Formatting to **Console.WriteLine()**

string.Format

- The Format method can perform complex formatting (similar to the Console.WriteLine method).
- It returns a single string containing the formatted output.
- **string.Format** is often used to prepare data to be displayed in a label.

string.Format

- The example shown below uses Format to create a single string describing the mouse location:

```
string sOut = string.Format("Mouse at {0},{1}", e.X, e.Y);
Console.WriteLine(sOut);
```

- Most Windows controls only accept a string for display, so Format can be used to display numeric data.

String Interpolation

- String interpolation provides a more readable way of placing values into a string. The statements in the previous slide can be written as:

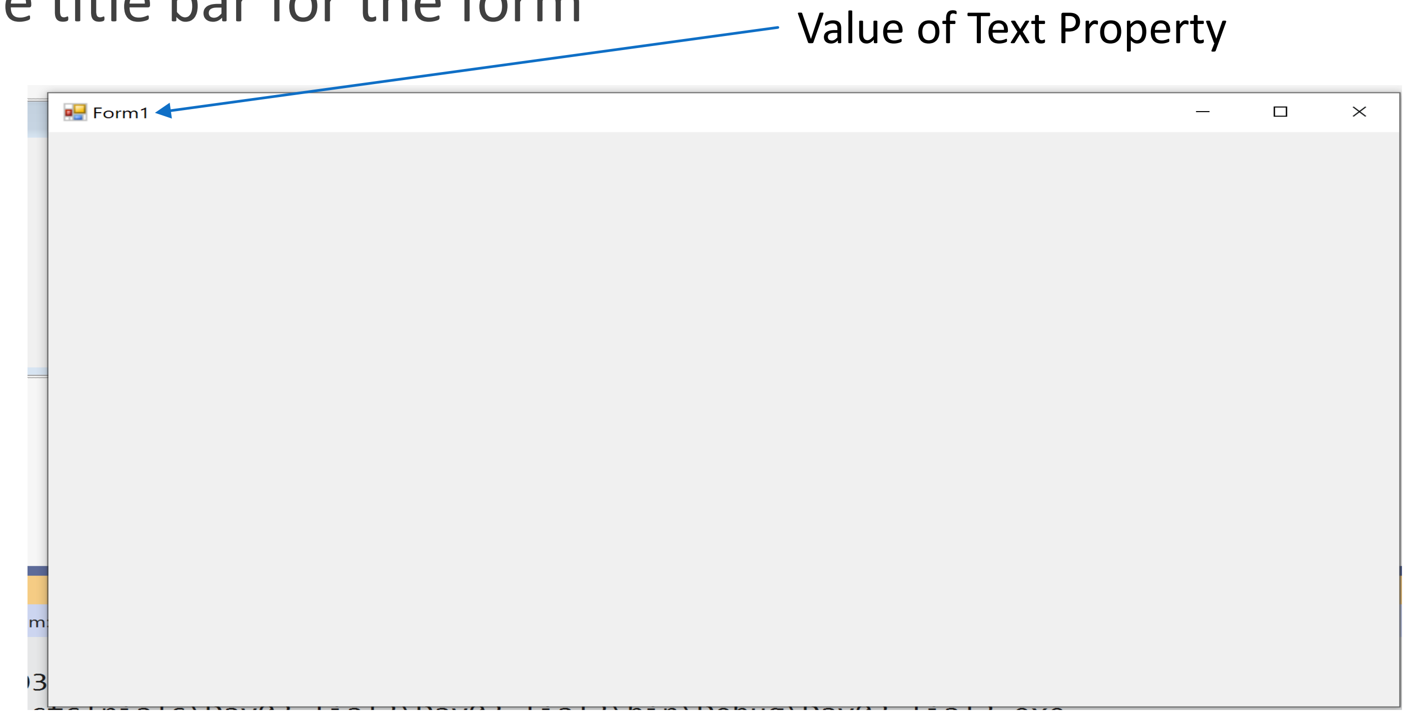
```
string sOut = $"Mouse at {e.X},{e.Y}";
        Console.WriteLine(sOut);
```

ToString

- All C# data types have a **ToString()** method to convert their value to a string equivalent.
- The resulting string can then be assigned to the **Text** property of most Windows controls and forms.
- The Text property only accepts strings.

Changing The Text Property of a Form

- The value of the Text property of the form is what is displayed in the title bar for the form



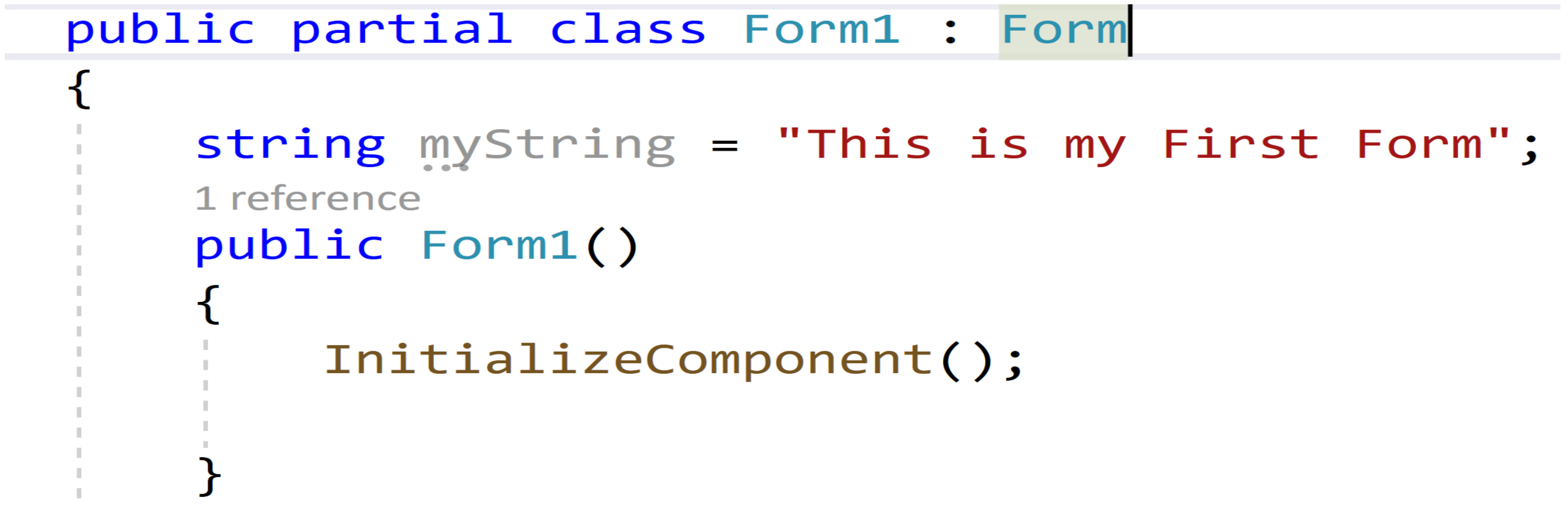
- This value can be changed through the property window or through an event handler.

Changing The MinimizeBox and MaximizeBox properties

- The **MinimizeBox** and **MaximizeBox** properties are of type boolean.
- By default, their values are True, which allows the user to see the **Minimize** and **Maximize** buttons and thus minimize and maximize the form.
- If we change any of these properties to false, its corresponding button will not longer be visible.

Member Variables

- Form1 is a class.
- We can add member variables to the class, in **Form1.cs**.



```
public partial class Form1 : Form
{
    string myString = "This is my First Form";
    1 reference
    public Form1()
    {
        InitializeComponent();

    }
```

Lecture 2-demo2

- Create a Windows Form Project called Lecture2Demo2.
- Include a member variable of type **string**, called **mystring** and initialize it to the value “**This is my first form**”.
- Place a suitable statement in the Form1 constructor (after the Initcomponent) to set the Form **Text** to mystring
- Add an event handler to the form, such that when a user double-clicks on the mouse, the Form Text is changed to “**The mouse has been double-clicked**”

Lecture 2-Exercise1

- Create a project called Lecture2Exercise1
- Add a suitable member variable and an event handler to the form class, so that when the mouse is doubled-clicked the Form Text always displays the number of times the mouse has been double-clicked.
- Your Text should have the general format below:

This form has been double-clicked <num> times

E.g.

This form has been double-clicked 5 times

Lecture 2-Exercise2

- Create a Windows Form project called Lecture2Exercise2.
- Add an event handler such that when the mouse is double-clicked, the text toggles between the following 2 messages:

  This is a nice form

  and

  The mouse has been clicked <n> times

  (where <n> is replaced by an actual value)
- The first time the mouse is double-clicked the first message is displayed. After that, for each double-click, the message changes to the other one (first, second, first, second……….)
- Hint: use an int variable called **toggle** for which you can continuously toggle the value between 0 and 1