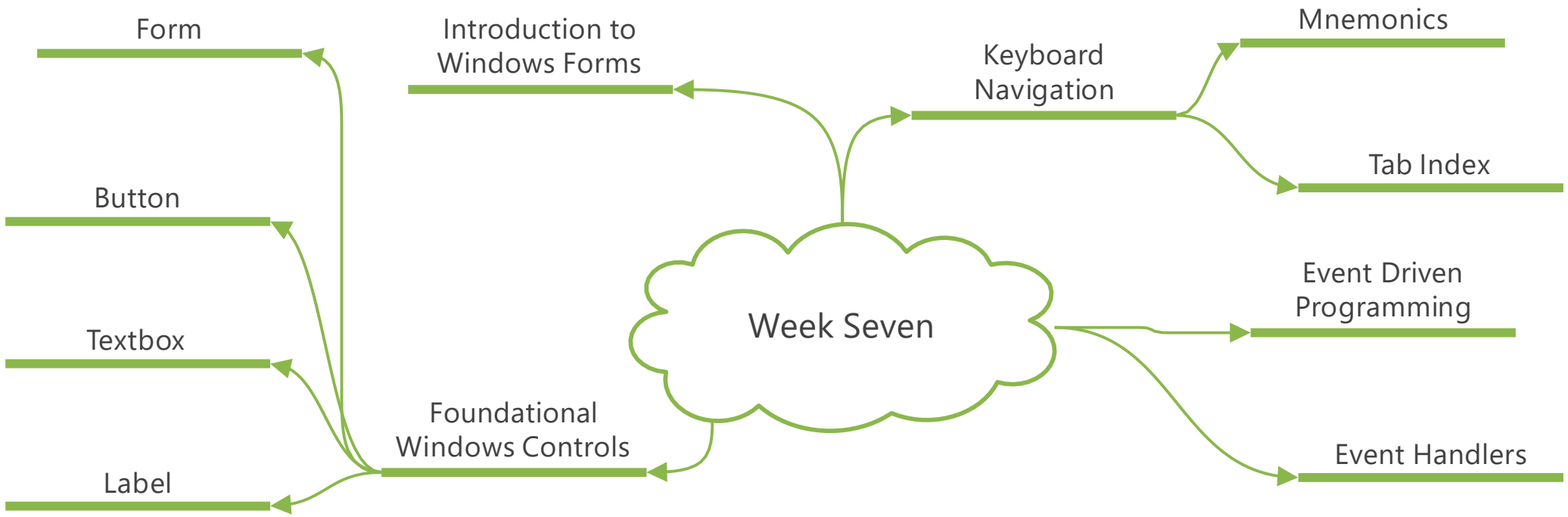
COMP 3602

C# Application Development

Week Seven

Tonight's Learning Outcomes



Assignment 03: To Round or Not to Round

- No set standard
  - Even in banking!
  - Different regulations depending on context and region
- General rule of thumb – *as late in the game as possible*
- Banker's rounding (.Net default)
  - Round half to even
  - Eg 3.5 gets rounded to 4 and so does 4.5
  - Evens out impact over many operations
  - https://en.wikipedia.org/wiki/Rounding#Round_half_to_even

Assignment 03: To Round or Not to Round

```
Invoice Number: 3221766
Invoice Date:   Sep 16, 2019
Discount Date:  Sep 26, 2019
Terms:          1.00% 10 days ADI
-------------------------------------------------------------
Qty SKU         Description                Price PST       Ext
-------------------------------------------------------------
  1 INX5700HT   8-Core XEON CPU         1,437.79  Y   1,437.79
 20 IN2300K     4-Core 3.0GHZ CPU         205.00  Y   4,100.00
-------------------------------------------------------------
                Subtotal:                                5,537.79
                GST:                                       276.89
                PST:                                       387.65
-------------------------------------------------------------
                Total:                                   6,202.33

                Discount:                                   62.02
```

Assignment 03: To Round or Not to Round

```
Invoice Number: 3221766
Invoice Date:   Sep 16, 2019
Discount Date:  Sep 26, 2019
Terms:          1.00% 10 days ADI
-------------------------------------------------------------
Qty SKU       Description              Price PST          Ext
-------------------------------------------------------------
  1 INX5700HT 8-Core XEON CPU       1,437.79  Y      1,437.79
 20 IN2300K   4-Core 3.0GHz CPU       205.00  Y      4,100.00
-------------------------------------------------------------
             Subtotal:                                5,537.79
             GST:                                       276.89
             PST:                                       387.65
-------------------------------------------------------------
             Total:                                   6,202.32

             Discount:                                   62.02
```

Assignment 03: To Round or Not to Round

```
2 references
62  public decimal Total {
63      get {
64          return Subtotal + GstTotal + PstTotal;
65      }
66  }
```

133 % No issues found

Autos

Search (Ctrl+E) Search Depth: 3

Name	Value	Type
GstTotal	276.8895	decimal
PstTotal	387.6453	decimal
Subtotal	5537.79	decimal
this	{COMP2614Assign03.Invoice}	COMP2614Assi...

Call Stack

Name
COMP261
COMP261
COMP261

```
     276.8895
     387.6453
    5537.7900
-------------
    6202.3248

-------------
     Rounded:
      6202.32
```

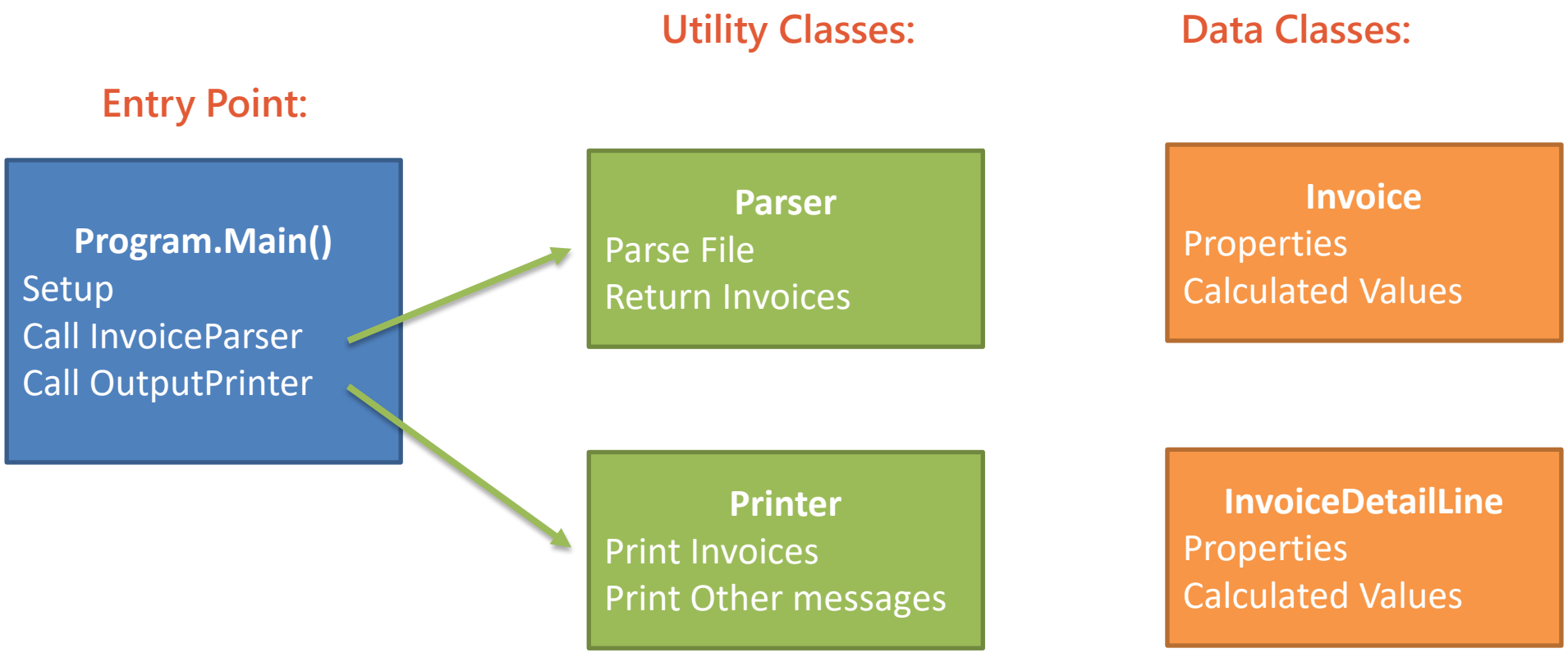
Assignment 03: To Round or Not to Round

No Rounding:	Common rounding:	Banker's rounding:
276.889	276.89	276.89
387.645	387.65	387.64
5537.790	5537.79	5537.79
---	---	---
6202.324	**6202.33**	**6202.32**

Further: What happens if we round by line item? What happens if we round by operation (eg taxes applied)?

This is a great example of the complexities we can face programming about things that are "simple"

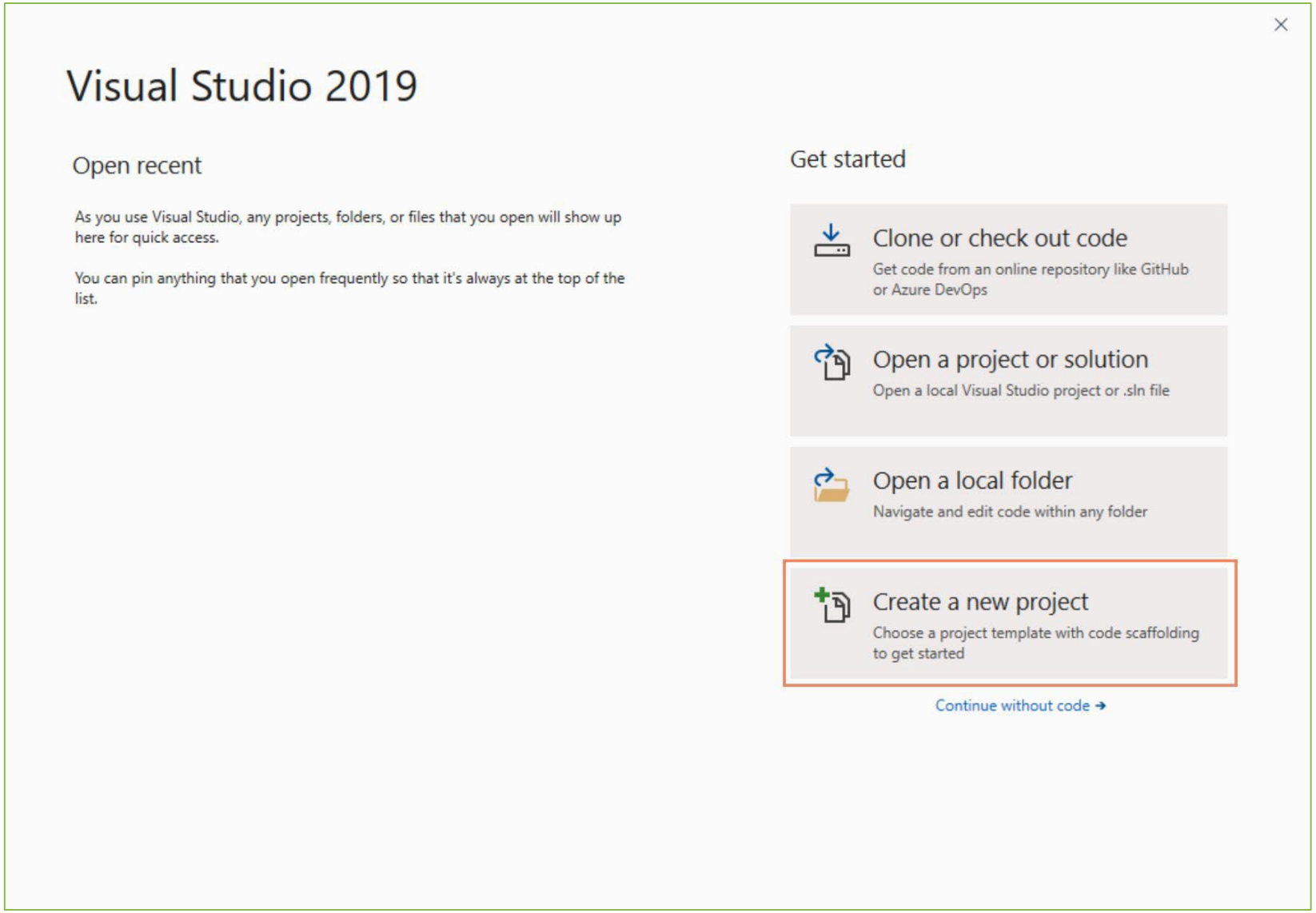
Assignment 03: High-Level Design



Further: What would be the best way to report errors in the Parser class? (Console.WriteLine(), Printer.PrintError(), Exception thrown, other?)

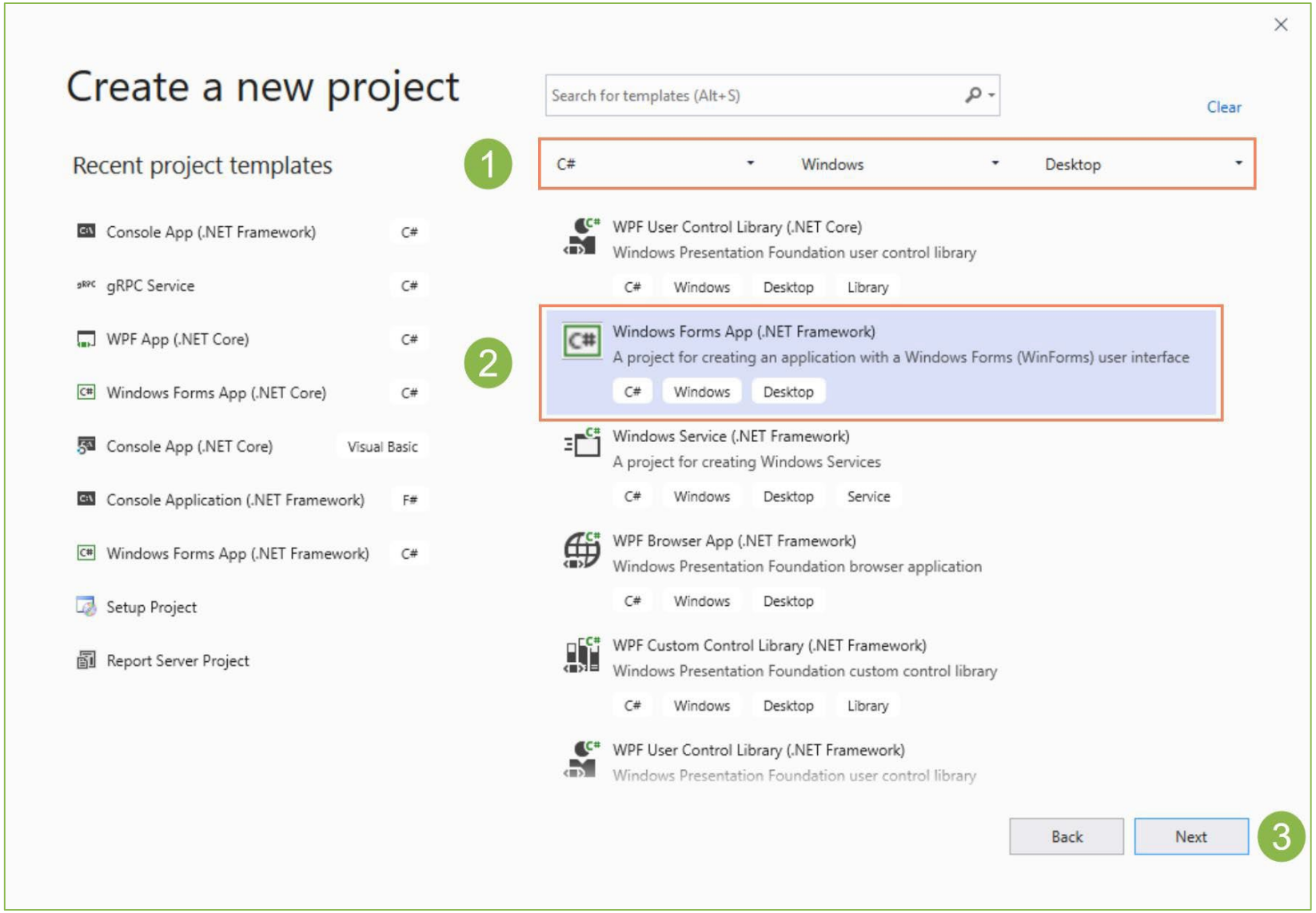
Where should we set Console.Title?

Creating a New WinForms Project with Visual Studio 2019 - A



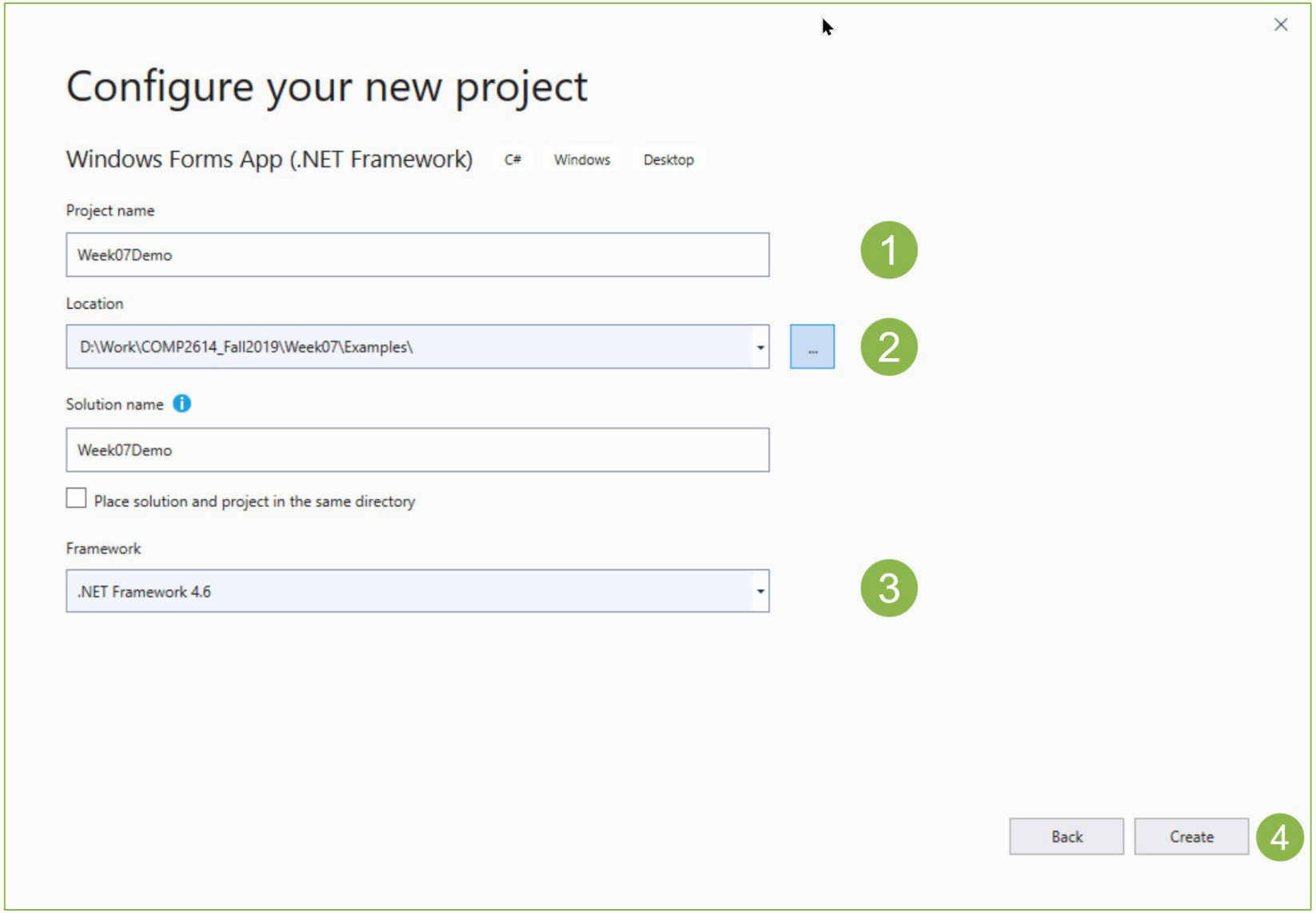
1) Start Visual Studio
2) Select “Create a new project”

Creating a New WinForms Project with Visual Studio 2019 - B



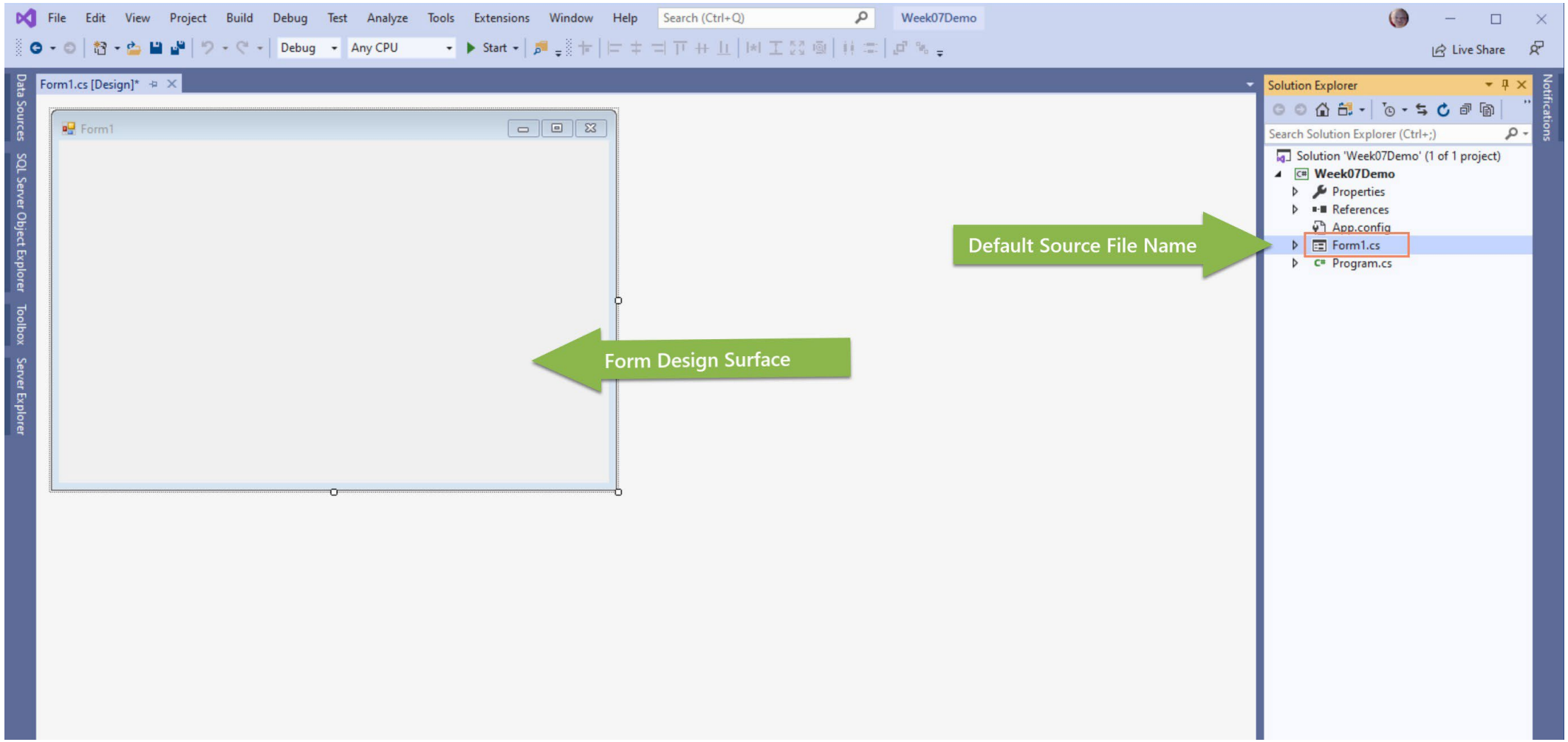
1) Filter by:
   Language: C#
   Platform: Windows
   Project type: Desktop
2) Select Windows Forms App (.NET Framework)
3) Click Next

Creating a New WinForms Project with Visual Studio 2019 - C

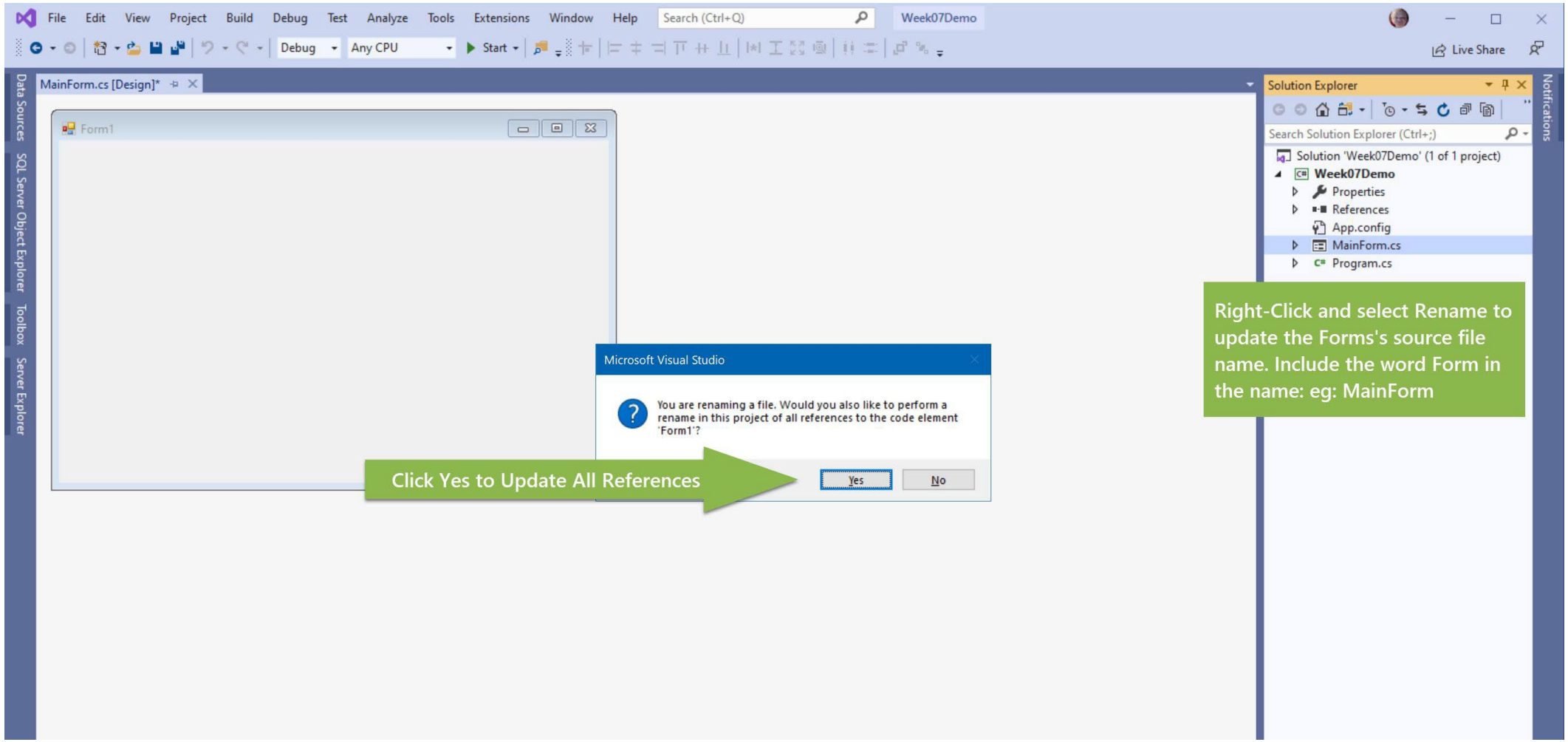


1) Name your project (PascalCase)
2) Navigate to a predetermined location
3) Select Framework Version (Current)
4) Click Create to create the project

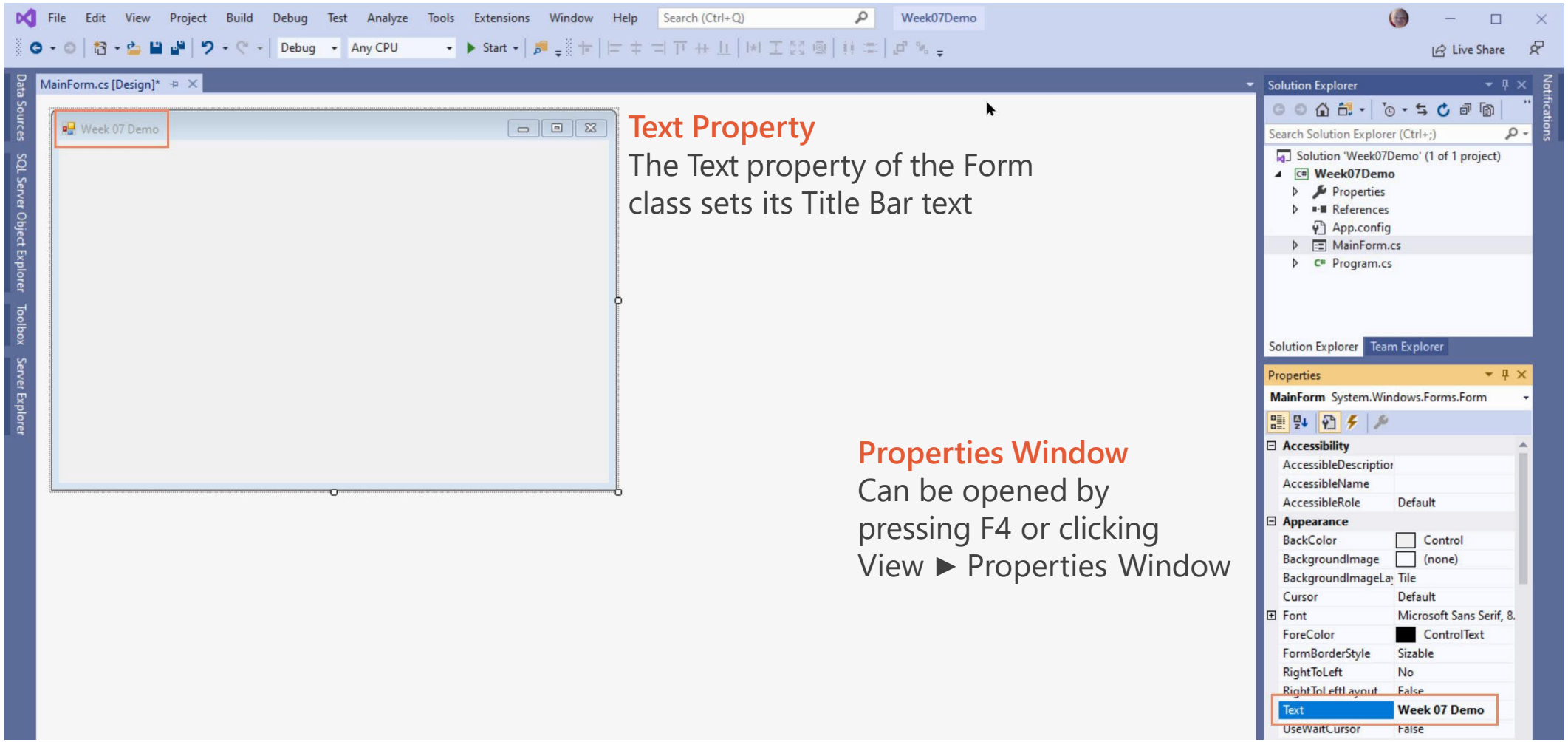
Project Startup



Renaming the Startup Form



Properties Window



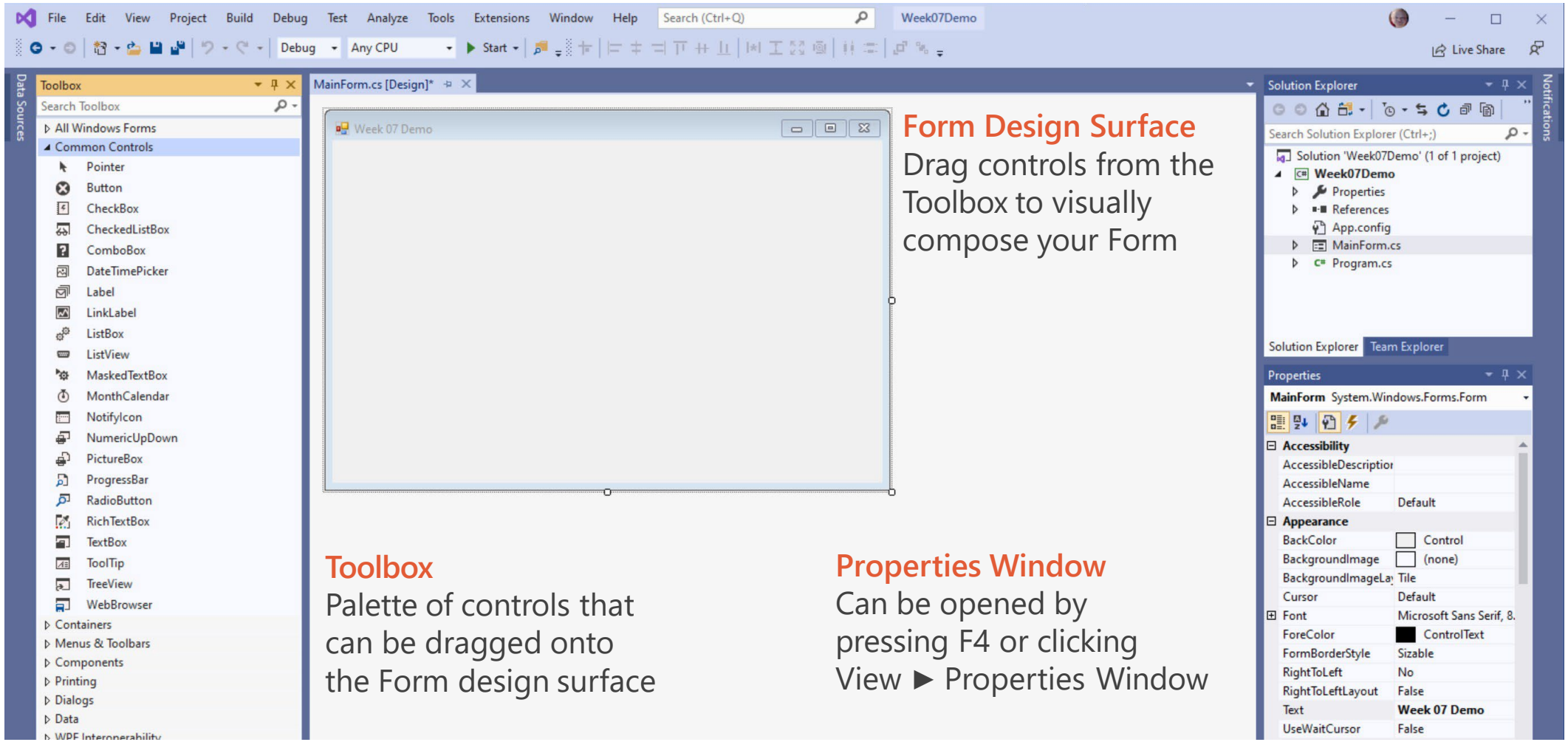
Text Property

The Text property of the Form class sets its Title Bar text

Properties Window

Can be opened by pressing F4 or clicking View ► Properties Window

Form Designer

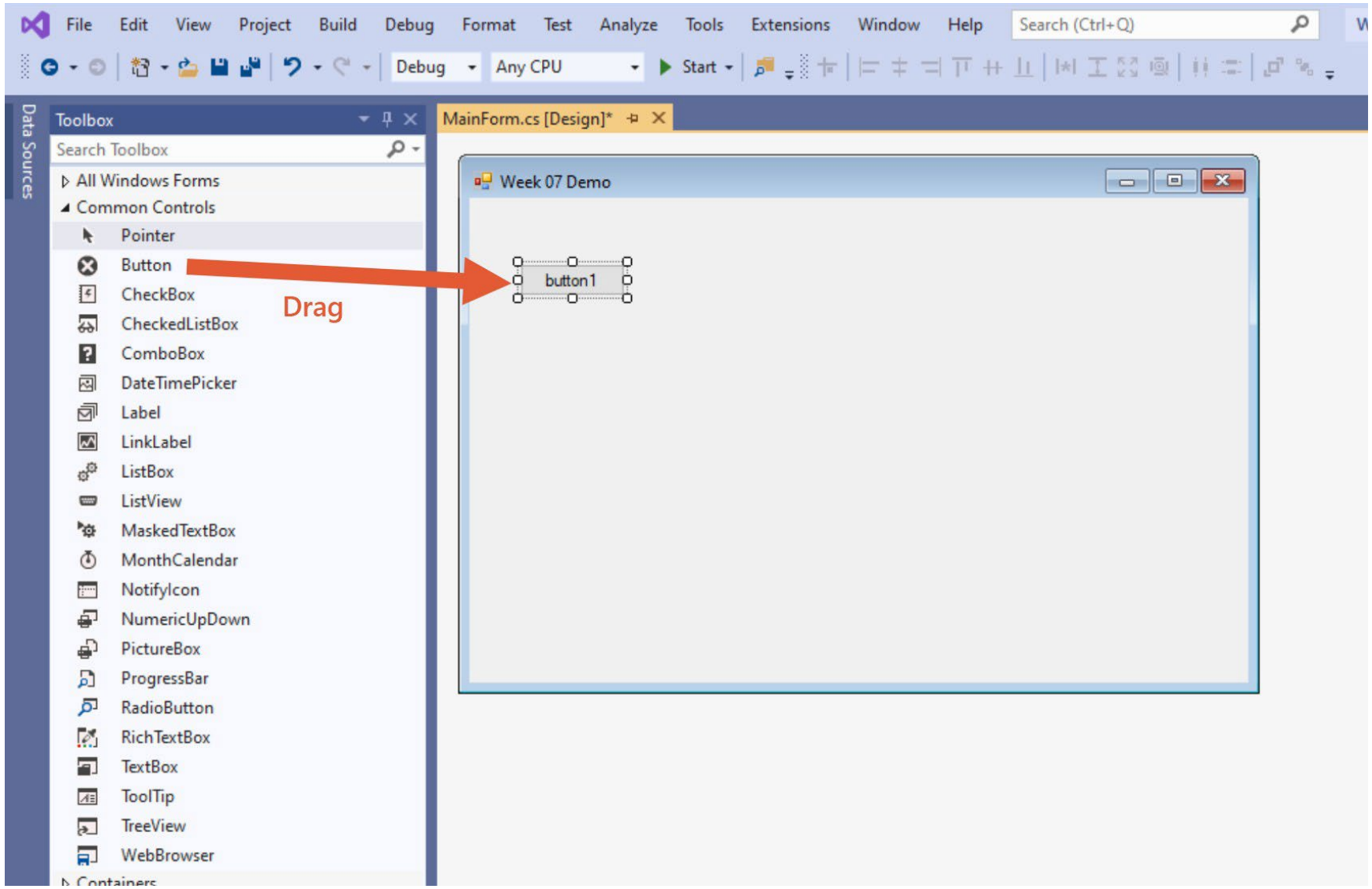


Form Design Surface
Drag controls from the Toolbox to visually compose your Form

Toolbox
Palette of controls that can be dragged onto the Form design surface

Properties Window
Can be opened by pressing F4 or clicking View ► Properties Window

Composing Your Form

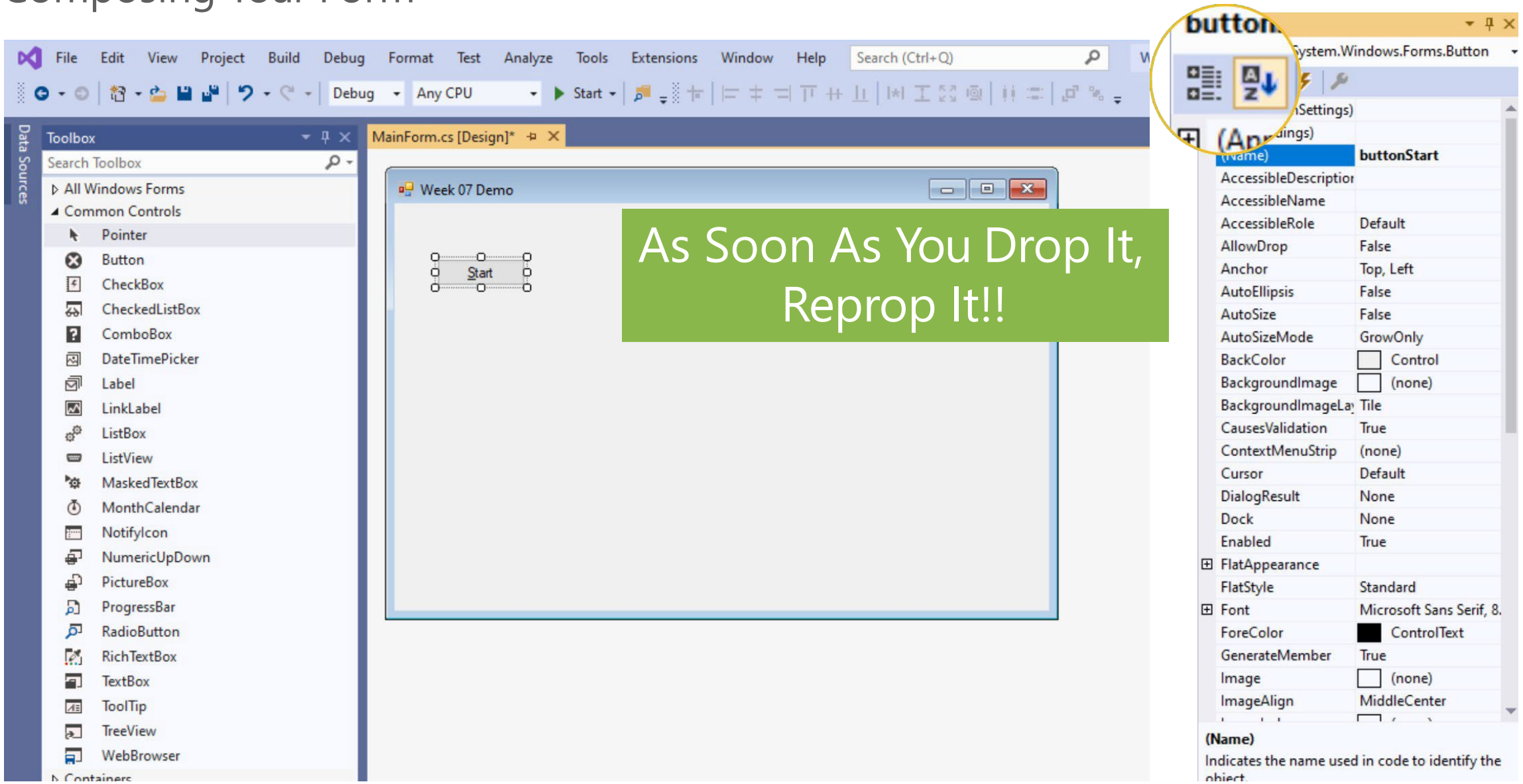


Adding Controls

Click on a Control in the Toolbox and "drag" it onto the Design Surface

Double Clicking on a Control in the Toolbox will also add it to the Form

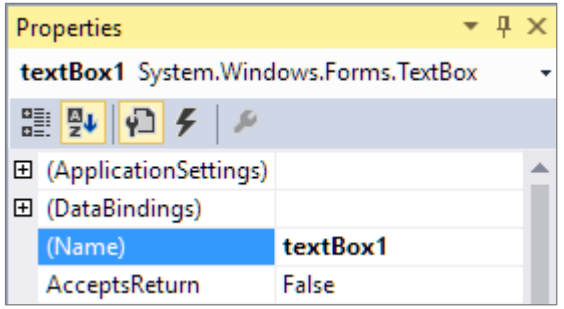
Composing Your Form



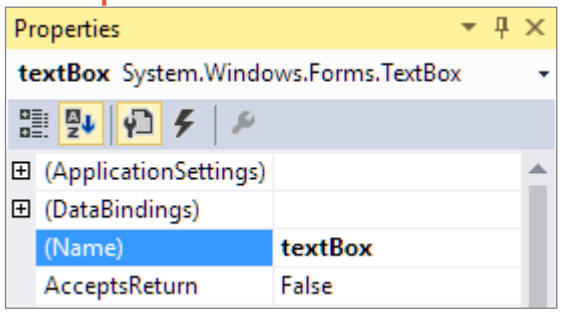
Control Naming Conventions

Control Name	Prefix	Example
Button	button	buttonCancel
Label	label	labelItemPrompt
TextBox	textBox	textBoxInterestRate
CheckBox	checkBox	checkBoxOnOff
RadioButton	radioButton	radioButtonPrintAll
ListBox	listBox	listBoxCountries
ComboBox	comboBox	comboBoxSelectionList
ListView	listView	listViewFileList
TreeView	treeView	treeViewDestinations
DataGridView	dataGridView	dataGridViewProducts
Form	None	MainForm

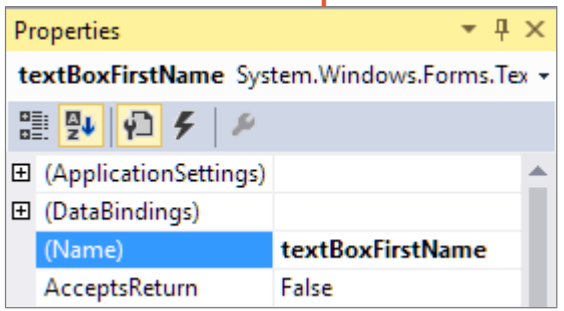
Default IDE name



Backspace over number



Add Control description



Application Startup

```
7   namespace Week07Demo
8   {
        0 references
9       static class Program
10      {
11          /// <summary>
12          /// The main entry point for the application.
13          /// </summary>
14          [STAThread]
            0 references
15          static void Main()
16          {
17              Application.EnableVisualStyles();
18              Application.SetCompatibleTextRenderingDefault(false);
19              Application.Run(new MainForm());
20          }
21      }
22  }
23
```

Windows Forms Applications start with the instantiation of the startup form.

This is done in the Main method – where all C# programs begin their execution.

Essentially, Main has one line of code which creates an instance of the startup Form.

Form Class Source Files

All Forms have two source files:

FormName.cs

FormName.Designer.cs

This separates the IDE generated code from developer written code

MainForm.Designer.cs – IDE Generated Code Goes Here

```
1   namespace Week07Demo
2   {
    3 references
3       partial class MainForm
4       {
5           /// <summary> Required designer variable.
8           private System.ComponentModel.IContainer components
9
10          /// <summary> Clean up any resources being used.
            0 references
            protected override void Dispose(bool disposing)
            {
                if (disposing && (components != null))
                {
                    components.Dispose();
                }
                base.Dispose(disposing);
            }
22
23          #region Windows Form Designer generated code
24
25          /// <summary> Required method for Designer support - do not modify
            1 reference
29          private void InitializeComponent()
30          {
31              this.buttonStart = new System.Windows.Forms.Button();
32              this.SuspendLayout();
33              //
34              // buttonStart
35              //
36              this.buttonStart.Location = new System.Drawing.Point(36, 47);
37              this.buttonStart.Name = "buttonStart";
38              this.buttonStart.Size = new System.Drawing.Size(75, 23);
39              this.buttonStart.TabIndex = 0;
40              this.buttonStart.Text = "&Start";
41              this.buttonStart.UseVisualStyleBackColor = true;
```

InitializeComponent() Method

This method's code is generated by the IDE as you compose your Form with the Visual Designers and called by the Form's constructor.

MainForm.cs – Developer Code Goes Here

```
11  namespace Week07Demo
12  {
        3 references
13      public partial class MainForm : Form
14      {
            1 reference
15          public MainForm()
16          {
17              InitializeComponent();
18          }
19
```

Plan… Plan… Plan…

If you "Fail to Plan" you "Plan to Fail"

Determine the following prior to creating your form:

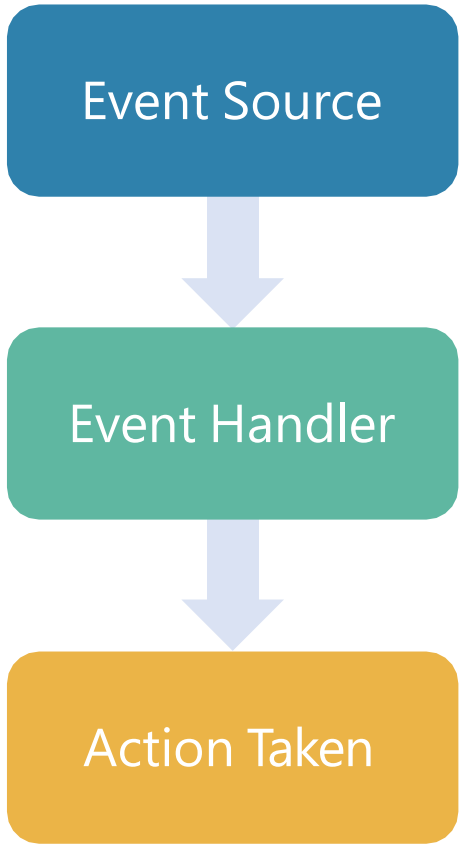
- Control types and quantities
- Control layout
- Control alignment
- Control names (use conventions)
- Keyboard navigation (mnemonics)
- etc…

Design, Sketch, Prototype

Event Driven Programming

Examples of events:

- button click
- keyboard key press
- timer tick
- form closing
- bytes received
- etc...



Windows Forms programs are event-driven, meaning once they start, they usually sit idle waiting for an event to process.

When a program receives an event, it can do one of two things:

- respond with an event handler (take action with a method)
- let the operating system handle the event (ignore it)

Most event-driven programs let the operating system handle the majority of the events they receive

Event Driven Programming

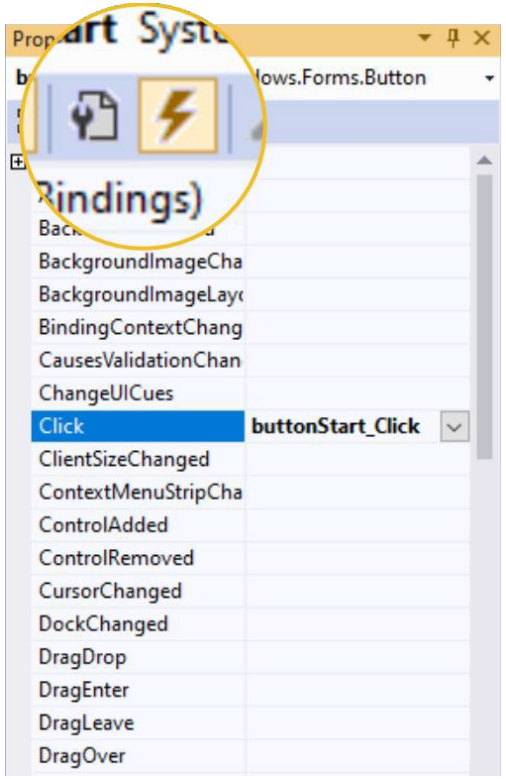
```
//
// buttonSetMaxSizeWidth
//
this.buttonSetMaxSizeWid
this.buttonSetMaxSize
this.buttonSetMaxSize
this.buttonSetMaxSize
this.buttonSetMaxSizeWidth.TabStop
this.buttonSetMaxSizeWidth.Text &Set MaximumSize Width to 120";
this.buttonSetMaxSizeWidth.UseVisualStyleBackColor = true;
this.buttonSetMaxSizeWidth.Click += new System.EventHandler(this.buttonSetMaxSizeWidth_Click);
//
```

Click event

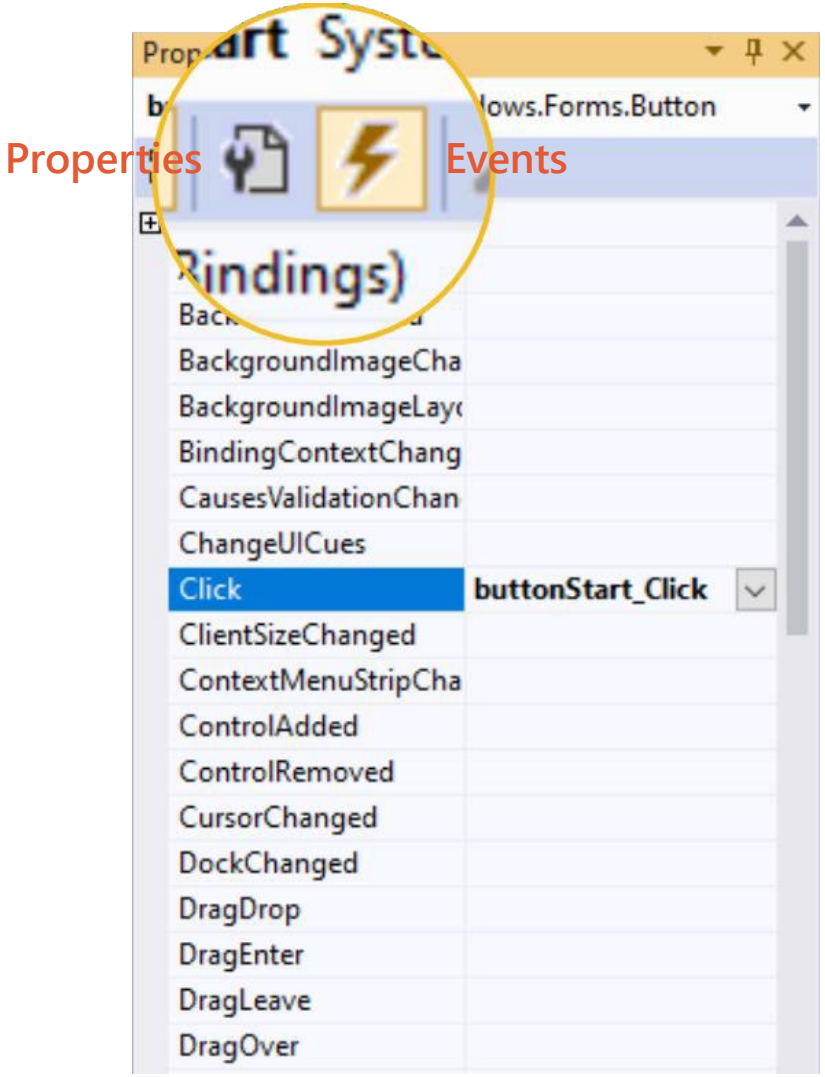
Method "subscribes" to the event

```
1 reference
private void buttonSetMaxSizeWidth_Click(object sender, EventArgs e)
{
    labelAutoSizeTrue.MaximumSize = new Size(120, 0);
}
```

- **events** are defined on the controls
- The are *invoked* when the event occurs
- We need to tell the events *which method* to call when they are invoked



Event Handlers - Creating



- Select the desired control in the designer
- Click on the Events button on the Properties Window Toolbar
- Double Click on the Event you wish to create a handler for
- The IDE will generate an event handler method stub and wire it up to the selected control (see next slide)
- Every control has a “default event” **(For Button, it is the Click event)**
- Double Clicking on a control in the Form Designer is a shortcut for generating a default event handler

Event Handlers – Code Elements

MainForm.cs

```
3 references
public partial class MainForm : Form
{
    1 reference
    public MainForm()
    {
        InitializeComponent();
    }

    1 reference
    private void buttonStart_Click(object sender, EventArgs e)
    {
        // developer event handling code
    }

```

The IDE creates a method stub for developer written event handling code

MainForm.Designer.cs

```
private void InitializeComponent()
{
    this.buttonStart = new System.Windows.Forms.Button();
    this.SuspendLayout();
    //
    // buttonStart
    //
    this.buttonStart.Location = new System.Drawing.Point(36, 47);
    this.buttonStart.Name = "buttonStart";
    this.buttonStart.Size = new System.Drawing.Size(75, 23);
    this.buttonStart.TabIndex = 0;
    this.buttonStart.Text = "&Start";
    this.buttonStart.UseVisualStyleBackColor = true;
    this.buttonStart.Click += new System.EventHandler(this.buttonStart_Click);
    //
```

The IDE "wires up" the Event Handler method to the appropriate control

Do not modify this code

Event Handlers - Parameters

First Parameter

A reference to the Control that fired the event, upcast to type object

Second Parameter

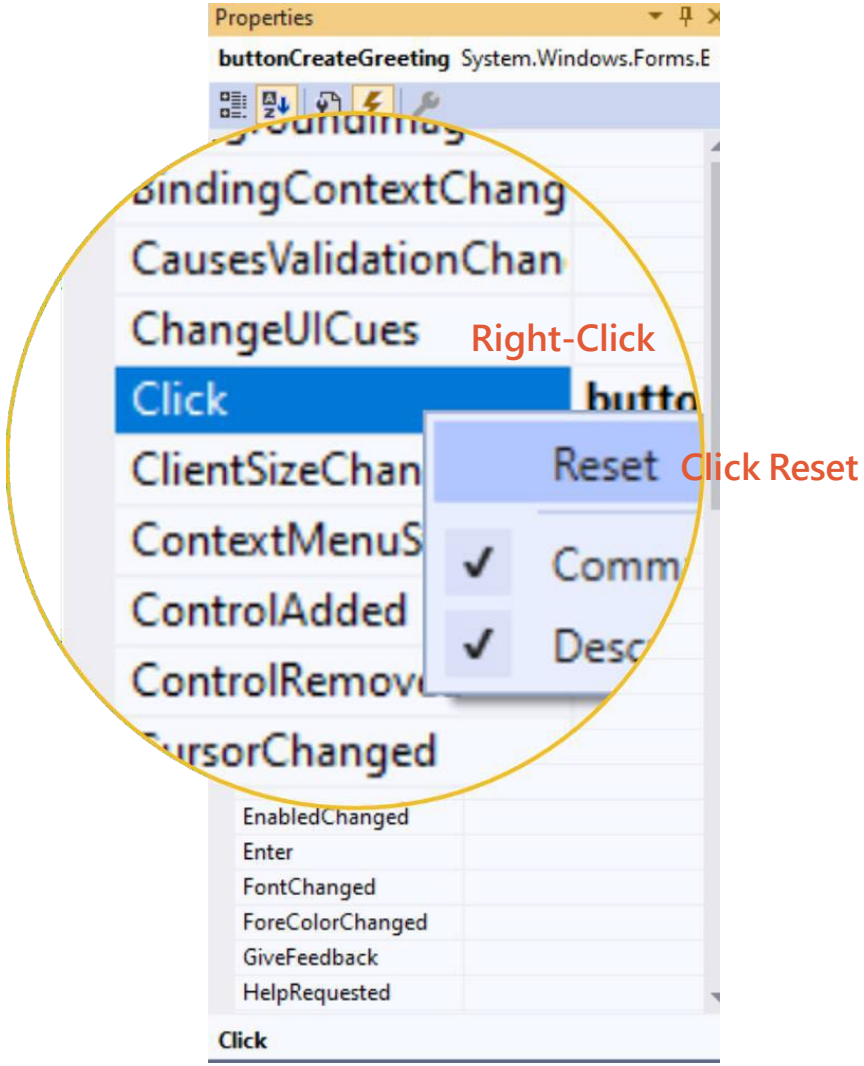
Varies with event type. With MouseMove, XY co-ordinates are passed in to indicate the Mouse position

```
1 reference
private void MainForm_MouseMove(object sender, MouseEventArgs e)
{
    labelGreeting.Text = $"X:{e.X} Y:{e.Y}";
}
```

Clicks
Delta
Equals
GetHashCode
GetType
Location
ToString
X
Y

int MouseEventArgs.Y { ge
Gets the y-coordinate of th

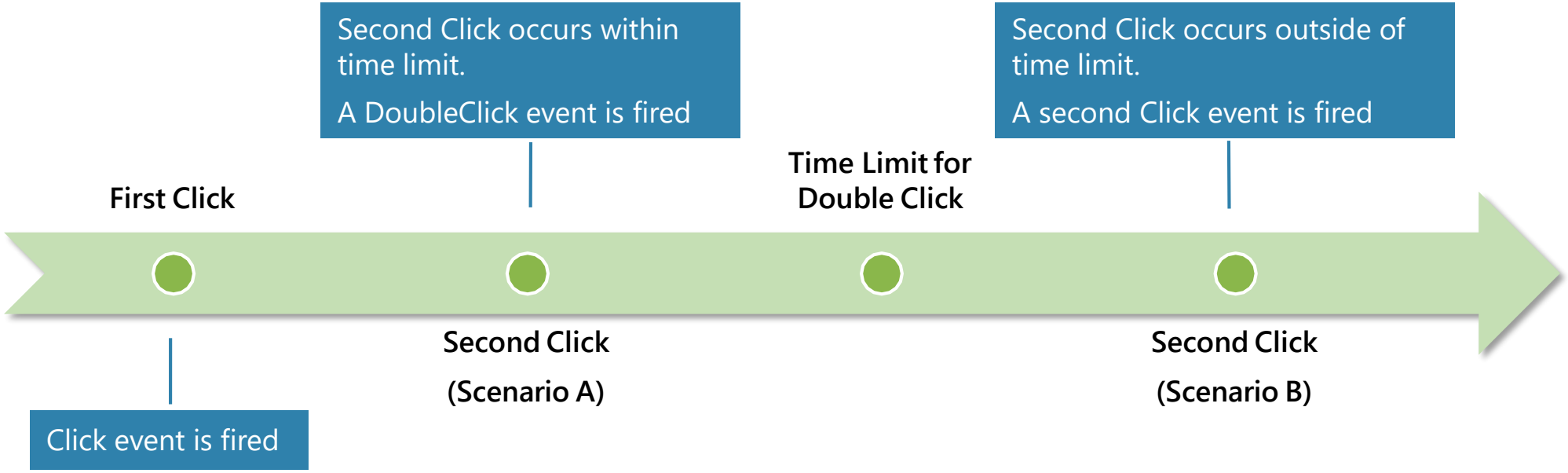
Event Handlers – Deleting



- Always delete unused event handlers.
- Simply deleting the event handler will cause the program to break.
- Go to events tab in the Properties window.
- Right-click on the handler to be deleted and click Reset. This will remove the delegate associated with this handler.
- You can safely delete the event handler once this has been done.

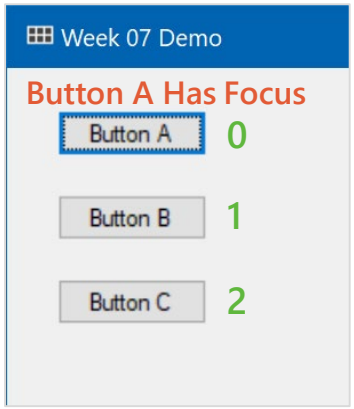
Click and Double Click Events

A DoubleClick event is **always** preceded by a Click event

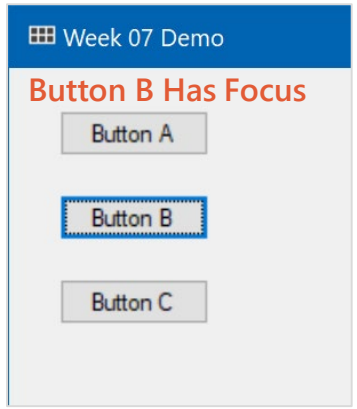


Keyboard Navigation - Focus

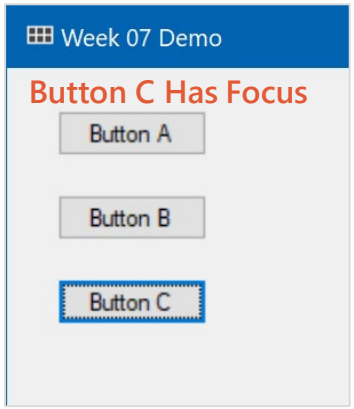
Press Tab -->

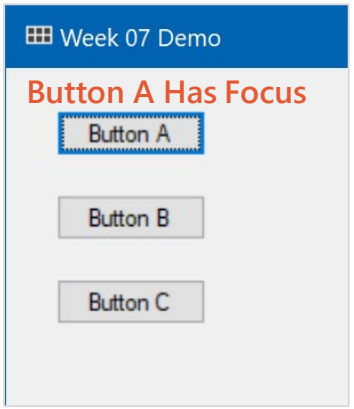


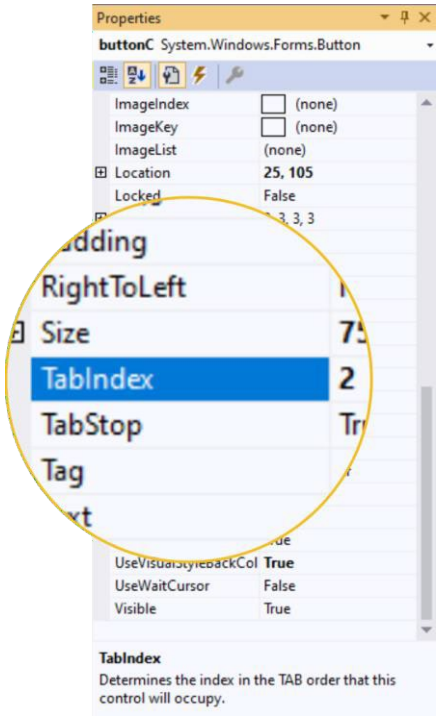
Press Tab -->



Press Tab -->

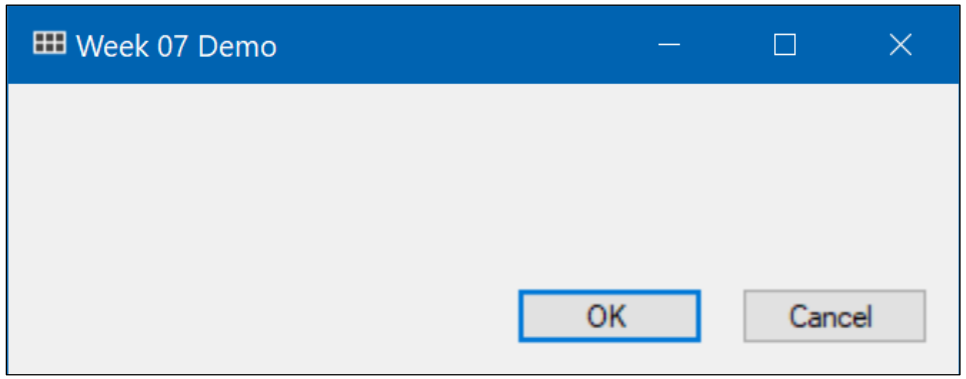


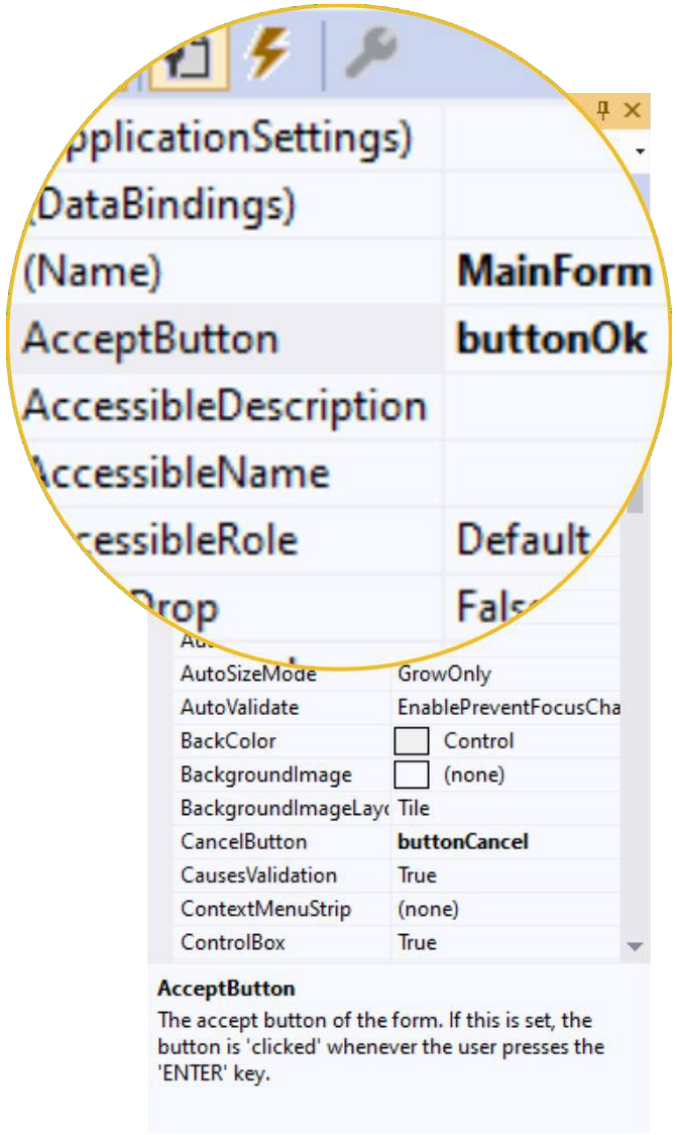




- Active control has focus
- Only one control can have focus at a time
- Pressing Tab will move the focus from one control to another
- The control's TabIndex determines the order in which each control receives focus
- The default TabIndex order is determined by the order each control is added to the Form/container

Keyboard Navigation – Accept and Cancel Buttons

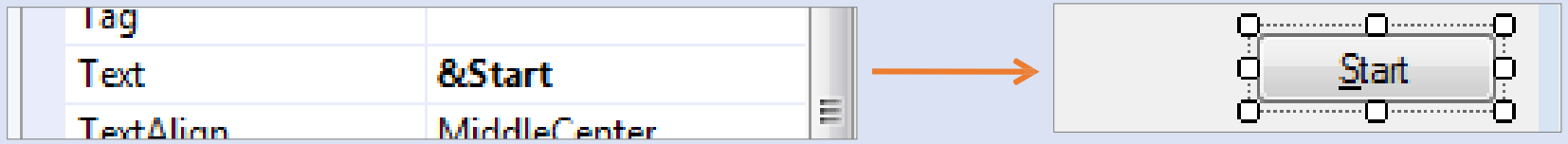




- Each Form can have an Accept and/or Cancel button
- **Accept Button**: The Click event will be fired when the user presses the Enter key
- **Cancel Button**: The Click event will be fired when the user presses the Esc key

Keyboard Navigation – Accessor Keys/Mnemonics

Place an Ampersand '&' prior to the letter you want to set as an accessor key in the Text property of the Control
Accessor Keys should be unique

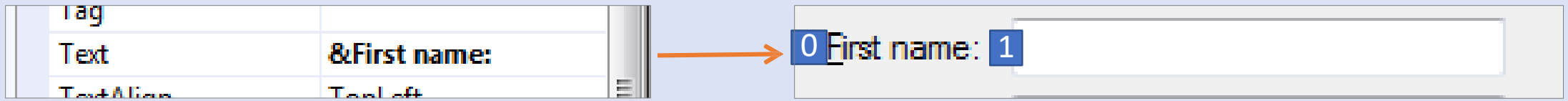


Four ways to fire a Button Click Event:

- Clicking the Button
- Pressing Alt-Key combo (Mnemonic)
- Pressing Enter (AcceptButton)
- Pressing Enter (while focused)
  (Focus takes precedence over Accept)

Keyboard Navigation – Accessor Keys/Mnemonics

At Form startup, the Label at TabIndex 0 will receive focus. Since a Label is a “non-focusable” Control, focus will go to the TextBox at TabIndex 1

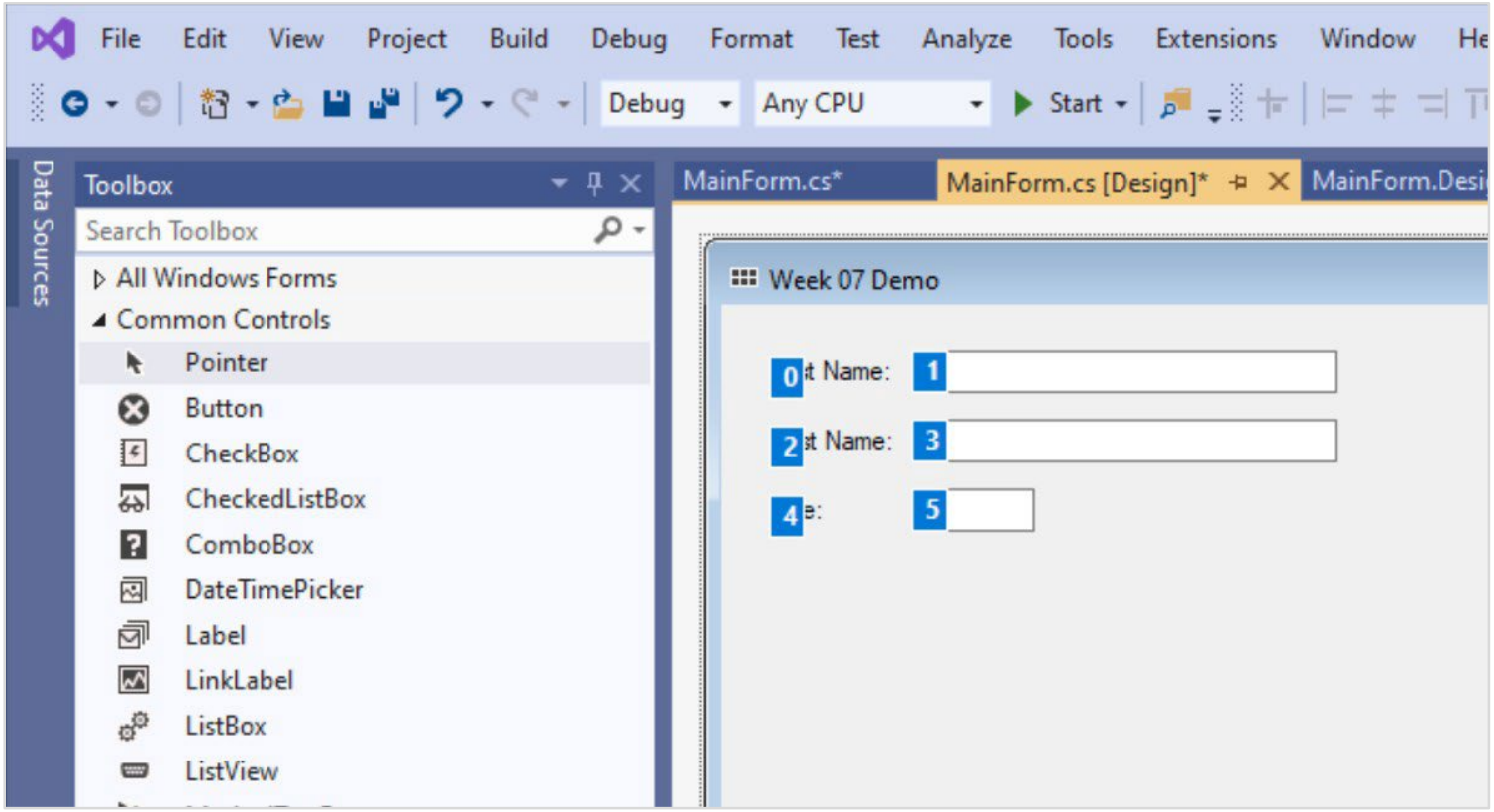


Pressing Alt-Key combination will:

- Click a Button
- Set Focus to the Control (Focusable)
- Set Focus to the Control at TabIndex + 1 (Non-Focusable)

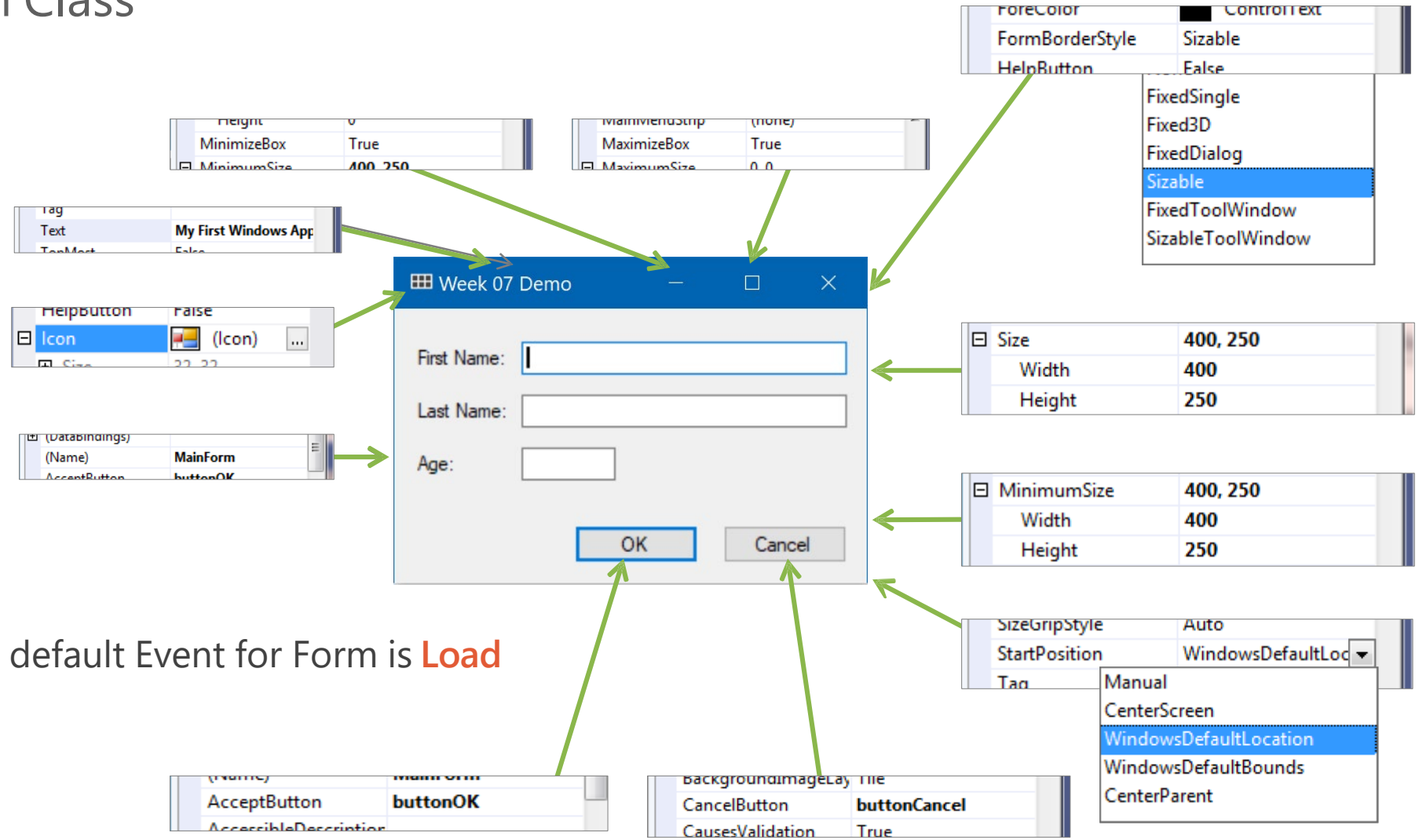
Keyboard Navigation – TabIndex/TabOrder

Click View ► Tab Order to open Tab Order screen



- The Control with the TabIndex zero will receive focus at Form startup
- Well designed Keyboard Navigation can enhance the User Experience by reducing the time wasted moving between the Keyboard and Mouse
- Default Tab Order is the same order that the Controls are added to the Form

Form Class



The default Event for Form is **Load**

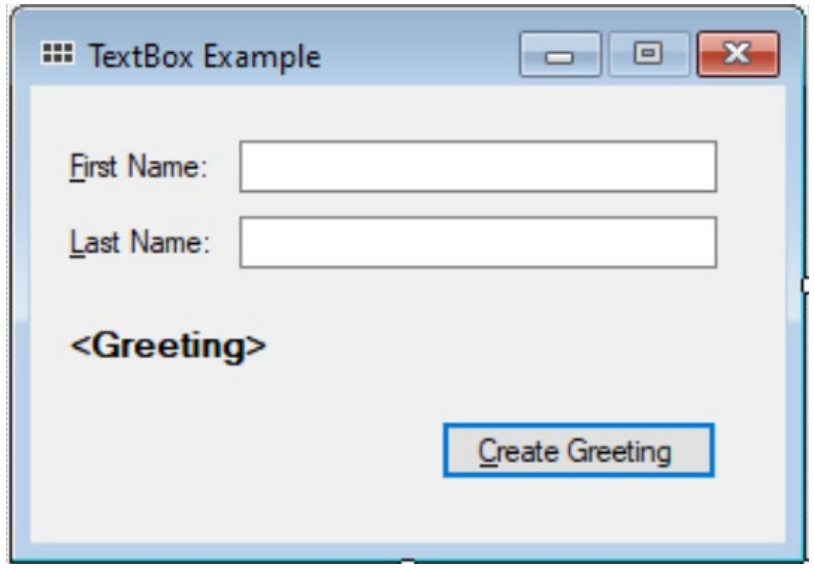
Clicks button when **Enter** key is pressed

Clicks button when **Esc** key is pressed

Label Class

Initializing Values

Tag	
Text	<Greeting>
TextAlign	TopLeft




```
33        private void MainForm_Load(object sender, EventArgs e)
34        {
35            labelGreeting.Text = string.Empty;
36        }
```

Design Time Value

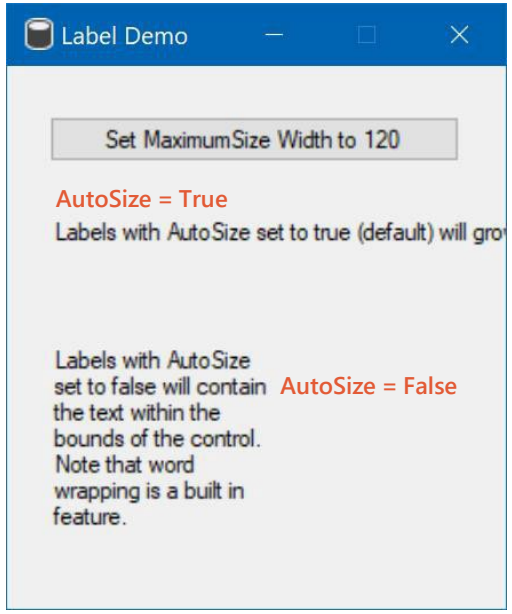
Give each dynamic Label a "Design Time" value even if it is meant to be blank at Form startup. This allows you to see the Label during design to better position and align it.

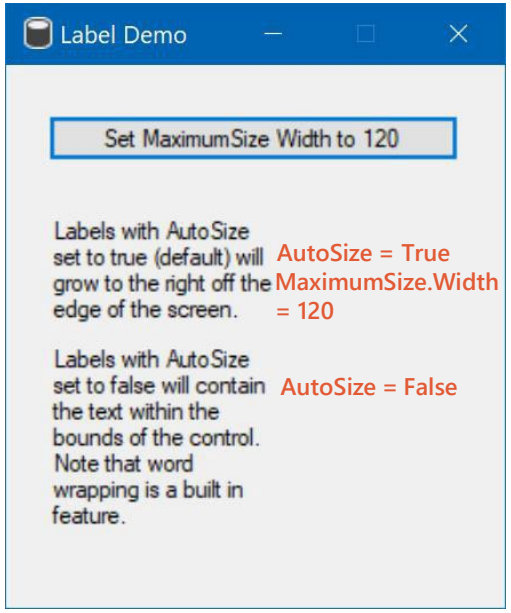
I wrap the value in **< >** to indicate a Design Time value

Clear the Value at Form Startup

The Form_Load event fires once just after instantiation. This is a great place to initialize values and state. Blank the value of your Label here so it shows nothing when the Form first appears.

Label Class – AutoSize Property



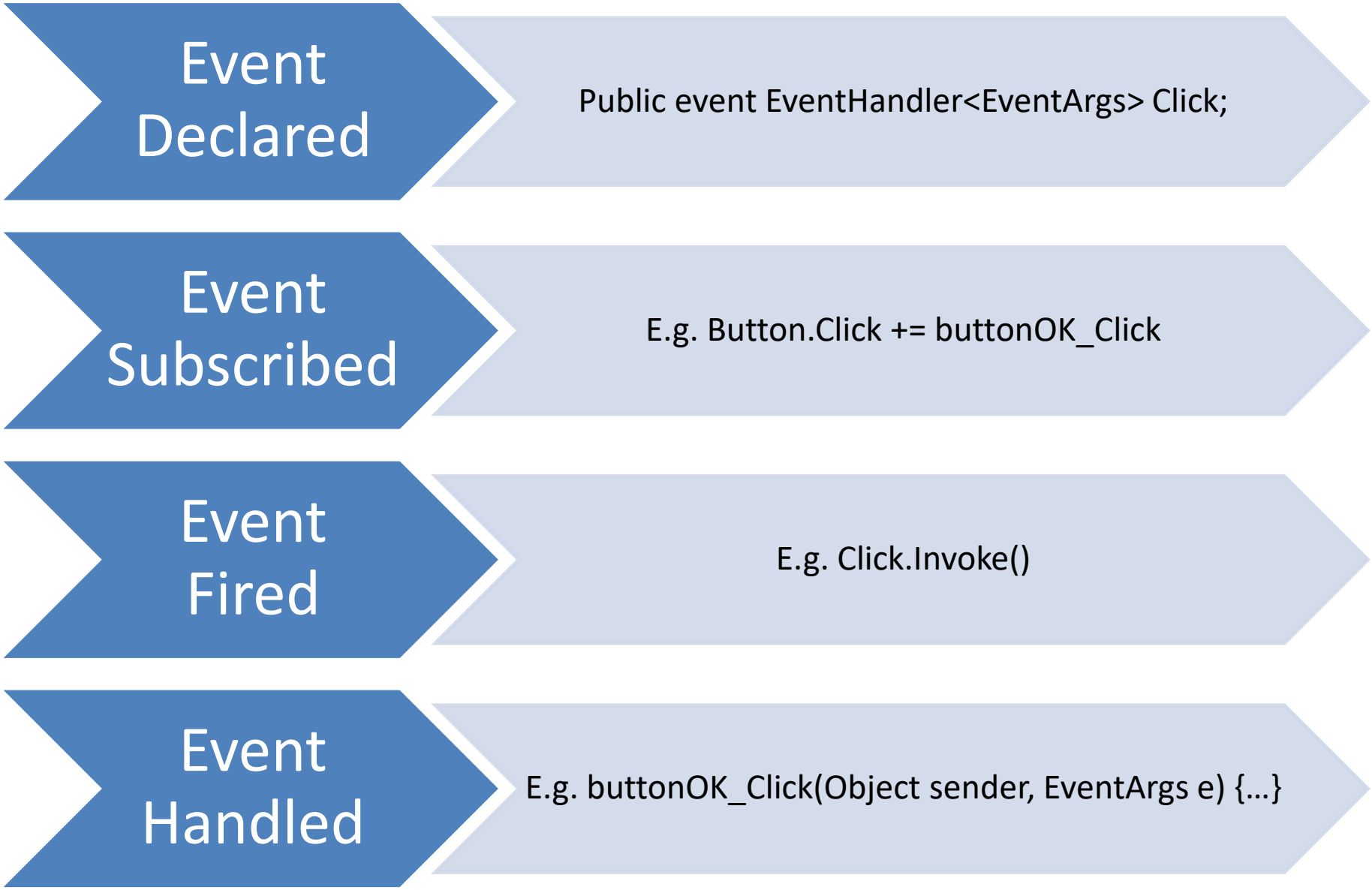


- The Label's AutoSize property is True by default
- Text will flow on a single line to the right, off the edge of the Form
- Setting AutoSize to False will contain the text within the bounds of the control. Wordwrap will occur when the text flows onto a new line.
- Setting the MaximumSize.Width property will also contain text when AutoSize is True

TextBox Class

Properties	Methods	Events
Name	Clear	TextChanged*
Text	Select	Click
Multiline	SelectAll	Enter
Readonly		Leave
BackColor		
ForeColor		
AcceptsReturn		
AcceptsTab		
CharacterCasing		* Default Event

Events – Overview



Events – Publisher

ChequingAccount Class

-string AccountNumber
-decimal Balance

-void Withdraw()
-void Deposit()

I want to tell other code as soon as an account gets overdrawn

Events – Publisher - Declare

ChequingAccount Class

-string AccountNumber
-decimal Balance
-**event AccountOverDrawn**

-void Withdraw()
-void Deposit()

So, I'll declare an event!

Events – Publisher - Fire

ChequingAccount Class

-string AccountNumber
-decimal Balance
-**event AccountOverDrawn**

-void Withdraw()
{
AccountOverDrawn.Invoke();
}
-void Deposit()

Then, in some places in my methods, I can **Invoke** the event (aka **fire** the event)

Events – Publisher

```
class ChequingAccount
{
    1 reference
    public string AccountNumber { get; }
    5 references
    public decimal Balance { get; private set; }


    public event EventHandler<EventArgs> AccountOverDrawn;
```

Publisher

Declare

```
1 reference
public void WithDraw(decimal amount)
{
    if (amount <= 0m)
    {
        throw new ArgumentOutOfRangeException("Amount", "Amount must be greater than zero");
    }

    this.Balance -= amount;

    if (this.Balance < 0.00m)
    {
        if (AccountOverDrawn != null)
        {
            AccountOverDrawn.Invoke(this, new EventArgs());
        }
    }
}
```

Publisher

Invoke (Fire)

Events – Subscriber

MainForm

-ChequingAccount chequingAccount
- …. otherFields

-void Initialize()

I know that I am creating an instance of ChequingAccount, and I know it can tell me if it is overdrawn

Events – Subscriber

MainForm

-ChequingAccount chequingAccount
- …. otherFields

-void Initialize()
-**void Account_AccountOverDrawn()**

So I'll create a method that does some action when the Account is overdrawn

Events – Subscriber

Then, I'll subscribe to the event in my Load event

```
MainForm

-ChequingAccount account
- …. otherFields

-void Initialize()
-void MainForm_Load()
{
  account.AccountOverDrawn += Account_AccountOverDrawn;
}
-void Account_AccountOverDrawn()
```

Events – Wired Up

ChequingAccount Class

-string AccountNumber
-decimal Balance
-**event AccountOverDrawn**

-void Withdraw()
{
AccountOverDrawn.Invoke();
}
-void Deposit()

MainForm

-ChequingAccount chequingAccount
- …. otherFields

-void Initialize()
-void MainForm_Load()
-**void Account_AccountOverDrawn()**

Events - Subscriber

```
1 reference
private void MainForm_Load(object sender, EventA
{
    account = new ChequingAccount("1001", 0.00m);
    account.AccountOverDrawn += Account_AccountOverDrawn;

    updateBalanceDisplay();
}
```

Subscriber

Subscribe

```
1 reference
private void Account_AccountOverDrawn(object sender, EventArgs e)
{
    MessageBox.Show("Account is Overdrawn", "Account Overdrawn", MessageBoxButtons.OK, MessageBoxIcon.Information);
}
```

Handle

Subscriber

Events – Publisher

```
class ChequingAccount
{
    1 reference
    public string AccountNumber { get; }
    5 references
    public decimal Balance { get; private set; }


    public event EventHandler<EventArgs> AccountOverDrawn;
```

Publisher

Declare

```
1 reference
public void WithDraw(decimal amount)
{
    if (amount <= 0m)
    {
        throw new ArgumentOutOfRangeException("Amount", "Amount must be greater than zero");
    }

    this.Balance -= amount;

    if (this.Balance < 0.00m)
    {
        if (AccountOverDrawn != null)
        {
            AccountOverDrawn.Invoke(this, new EventArgs());
        }
    }
}
```

Publisher

Invoke (Fire)

Events - Subscriber

```
1 reference
private void MainForm_Load(object sender, EventA
{
    account = new ChequingAccount("1001", 0.00m);
    account.AccountOverDrawn += Account_AccountOverDrawn;

    updateBalanceDisplay();
}
```

Subscriber

Subscribe

```
1 reference
private void Account_AccountOverDrawn(object sender, EventArgs e)
{
    MessageBox.Show("Account is Overdrawn", "Account Overdrawn", MessageBoxButtons.OK, MessageBoxIcon.Information);
}
```

Handle

Subscriber