University of Anbar
College of Computer Science
and Information Technology
Computer Network Systems Department

Visual Programming I

Lecture Nine

Third Stage

First Course 2021 - 2022

Seddiq Qais Abd Al-Rahman

MSc Computer Science

co.sedeikaldossary@uoanbar.edu.iq

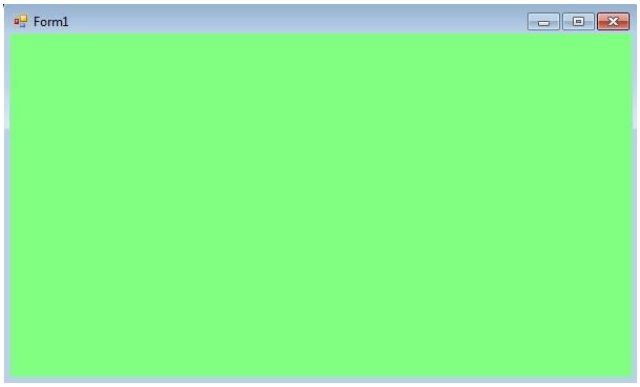
Visual Programming

ComboBox in C#

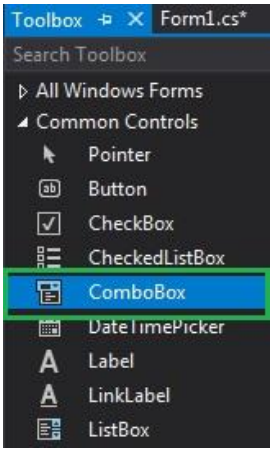
In Windows Forms, ComboBox provides two different features in a single control, it means ComboBox works as both TextBox and ListBox. In ComboBox, only one item is displayed at a time and the rest of the items are present in the drop-down menu. The ComboBox is a class in C# and defined under *System.Windows.Forms* Namespace. You can create ComboBox using the two different ways:

1. Design-Time: It is the easiest method to create a ComboBox control using the following steps:

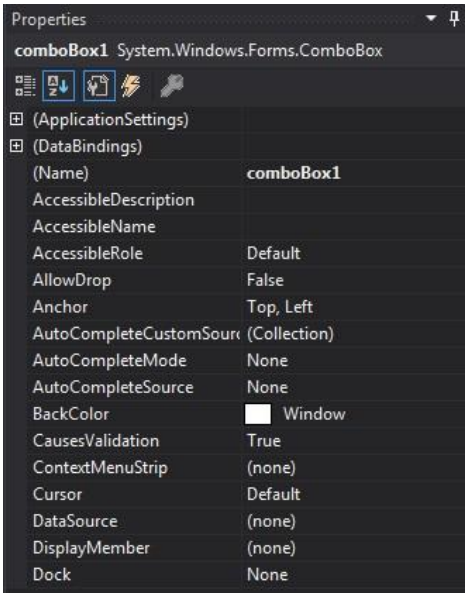
- **Step 1:** Create a windows form as shown in the below image: **Visual Studio -> File -> New -> Project -> WindowsFormApp**



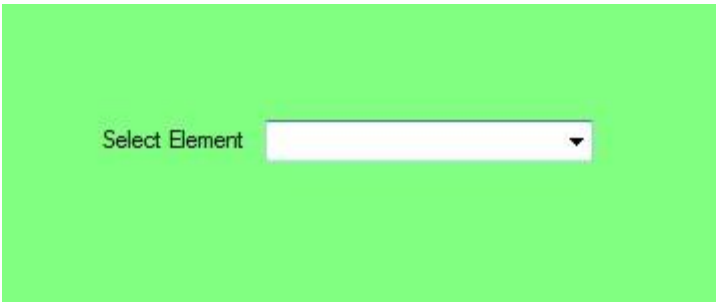
- **Step 2:** Drag the ComboBox control from the ToolBox and drop it on the windows form. You are allowed to place a ComboBox control anywhere on the windows form according to your need.



Step 3: After drag and drop you will go to the properties of the ComboBox control to set the properties of the ComboBox according to your need.



Output:



Run-Time: It is a little bit trickier than the above method. In this method, you can set create your own ComboBox control using the ComboBox class. Steps to create a dynamic ComboBox:

- **Step 1:** Create a combobox using the ComboBox() constructor is provided by the ComboBox class.

```
// Creating combobox using ComboBox class
ComboBox mybox = new ComboBox();
```

- **Step 2:** After creating ComboBox, set the properties of the ComboBox provided by the ComboBox class.

```
// Set the location of the ComboBox
mybox.Location = new Point(327, 77);

// Set the size of the ComboBox
mybox.Size = new Size(216, 26);

// Add items in the ComboBox
mybox.Items.Add("C#");
mybox.Items.Add("Java");
```

```
mybox.Items.Add("Scala");
mybox.Items.Add("C");
mybox.Items.Add("C++");
```

- **Step 3:** And last add this ComboBox control to form using Add() method.

```
// Add this ComboBox to the form
this.Controls.Add(mybox);
```

Example:

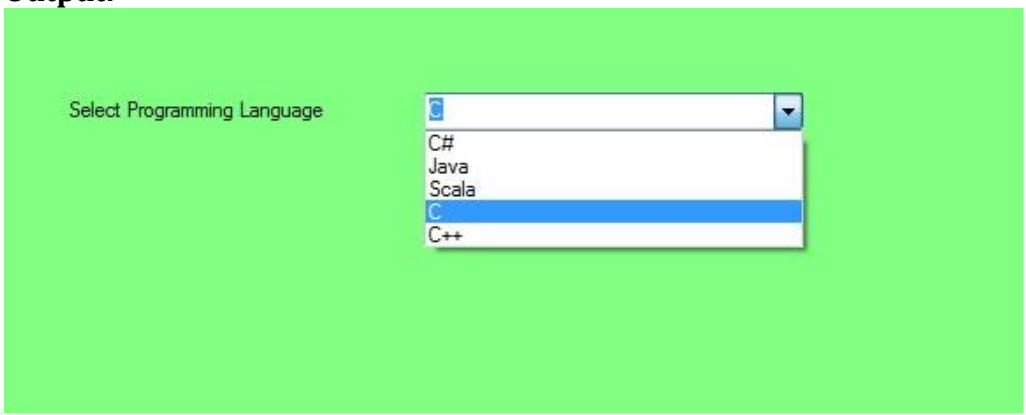
filter_none

brightness_4

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
namespace WindowsFormsApp18 {
public partial class Form1 : Form {
   public Form1()
   {
      InitializeComponent();
   }
   private void Form1_Load(object sender, EventArgs e)
   {
      // Creating and setting the properties of label
      Label l = new Label();
      l.Location = new Point(122, 80);
      l.AutoSize = true;
      l.Text = "Select Programming Language";
      // Adding this label to the form
      this.Controls.Add(l);
      // Creating and setting the properties of comboBox
      ComboBox mybox = new ComboBox();
      mybox.Location = new Point(327, 77);
      mybox.Size = new Size(216, 26);
      mybox.Items.Add("C#");
      mybox.Items.Add("Java");
      mybox.Items.Add("Scala");
      mybox.Items.Add("C");
```

```
            mybox.Items.Add("C++");
            // Adding this ComboBox to the form
            this.Controls.Add(mybox);
        }
    }
}
```

Output:



Important Properties of the ComboBox

PROPERTY	DESCRIPTION
BackColor	This property is used to set the background color for the ComboBox control.
DropDownHeight	This property is used to set the height in pixels of the drop-down portion of the ComboBox control.
DropDownStyle	This property is used to set a value specifying the style of the ComboBox control.
DropDownWidth	This property is used to set the width of the of the drop-down portion of a ComboBox control.
Font	This property is used to set the font of the text displayed by the ComboBox control.
ForeColor	This property is used to set the foreground color of the ComboBox control.
Height	This property is used to set the height of the ComboBox control.
Items	This property is used to get an object representing the collection of the items contained in this ComboBox control.
MaxDropDownItems	This property is used to set the maximum number of items to be shown in the drop-down portion of the ComboBox control.
MaxLength	This property is used to set the number of characters a user can type into the ComboBox control.
Name	This property is used to set the name of the ComboBox control.
SelectedItem	This property is used to set currently selected item in the ComboBox.
Size	This property is used to set the height and width of the ComboBox control.

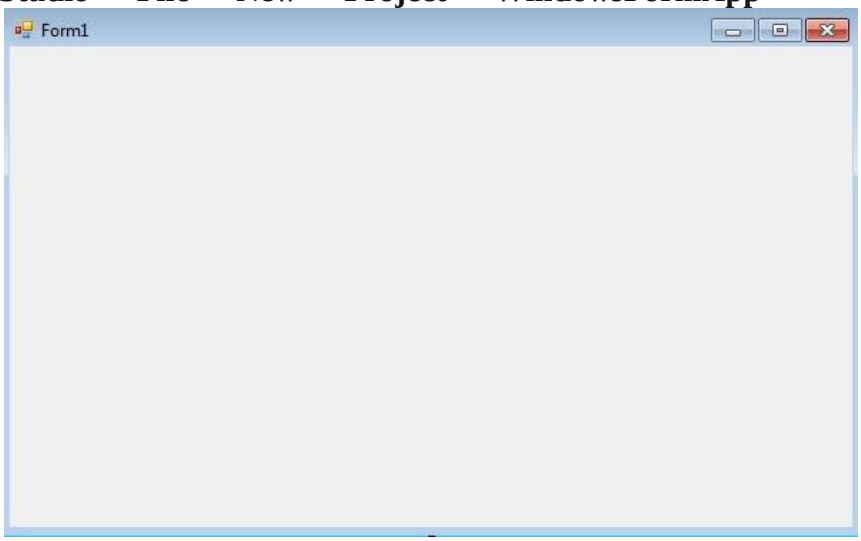
Sorted	This property is used to set a value indicating whether the items in the combo box are sorted.
Text	This property is used to set the text associated with this ComboBox control.
Visible	This property is used to set a value indicating whether the control and all its child controls are displayed.

C# | NumericUpDown Class

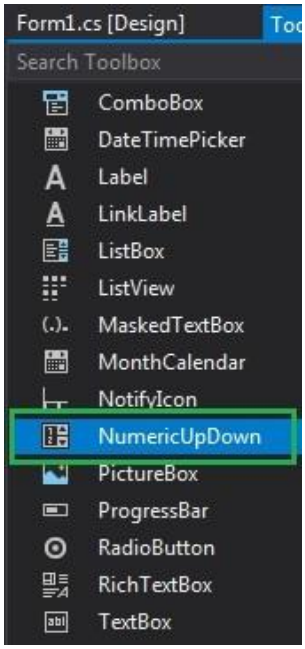
In Windows Forms, NumericUpDown control is used to provide a Windows spin box or an up-down control which displays the numeric values. Or in other words, NumericUpDown control provides an interface which moves using up and down arrow and holds some pre-defined numeric value. The NumericUpDown class is used to represent the windows numeric up-down box and also provide different types of properties, methods, and events. It is defined under **System.Windows.Forms** namespace. In C# you can create a NumericUpDown in the windows form by using two different ways:

1. Design-Time: It is the easiest way to create a NumericUpDown as shown in the following steps:

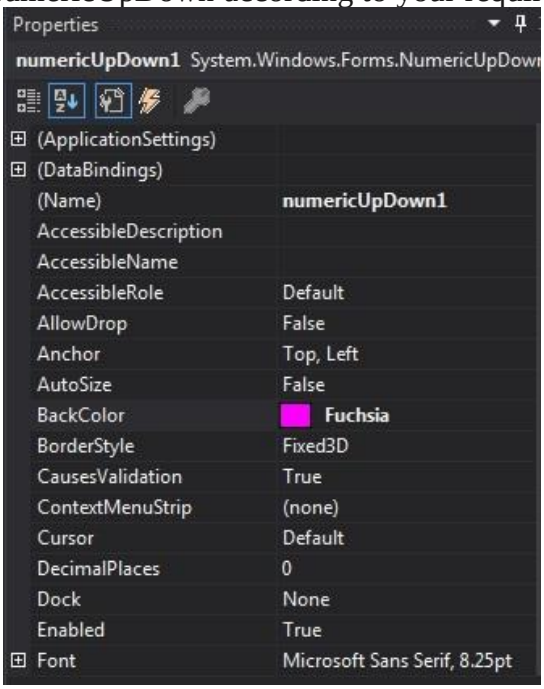
- **Step 1:** Create a windows form as shown in the below image:
  Visual Studio -> File -> New -> Project -> WindowsFormApp



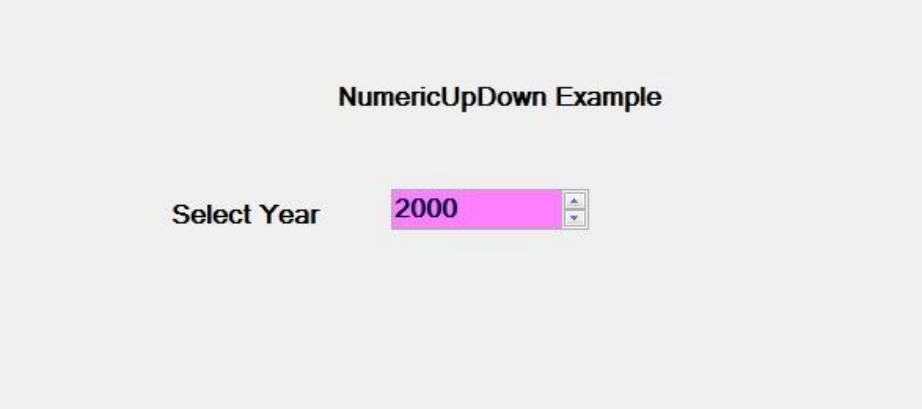
- **Step 2:** Next, drag and drop the NumericUpDown control from the toolbox to the form.



- **Step 3:** After drag and drop you will go to the properties of the NumericUpDown control to modify NumericUpDown according to your requirement.



Output:



2. Run-Time: It is a little bit trickier than the above method. In this method, you can create a NumericUpDown control programmatically with the help of syntax provided by the NumericUpDown class. The following steps show how to set the create NumericUpDown dynamically:

- **Step 1:** Create a NumericUpDown control using the NumericUpDown() constructor is provided by the NumericUpDown class.

```
// Creating a NumericUpDown control
NumericUpDown nbox = new NumericUpDown();
```

- **Step 2:** After creating a NumericUpDown control, set the property of the NumericUpDown control provided by the NumericUpDown class.

```
// Setting the properties of NumericUpDown control
nbox.Location = new Point(386, 130);
nbox.Size = new Size(126, 26);
nbox.Font = new Font("Bodoni MT", 12);
nbox.Value = 18;
nbox.Minimum = 18;
nbox.Maximum = 30;
nbox.BackColor = Color.LightGreen;
nbox.ForeColor = Color.DarkGreen;
nbox.Increment = 1;
nbox.Name = "MySpinBox";
```

- **Step 3:** And last add this NumericUpDown control to the form using the following statement:

```
// Adding this control
// to the form
this.Controls.Add(nbox);
```

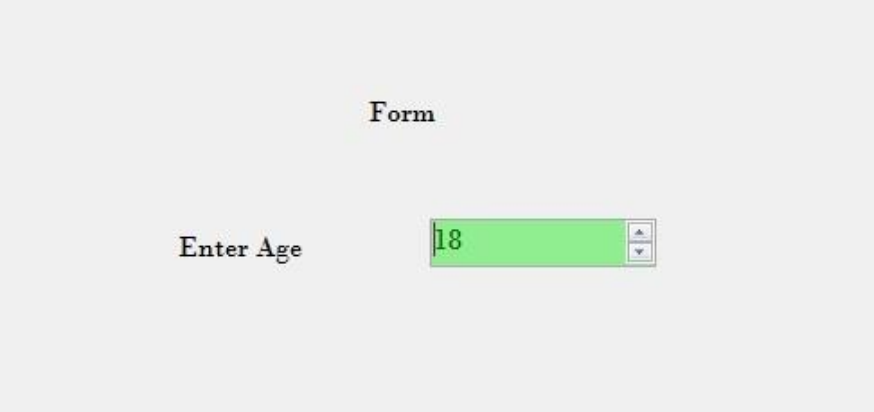
Example:
filter_none

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
```

```
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
namespace WindowsFormsApp42 {
public partial class Form1 : Form {
   public Form1()
   {
      InitializeComponent();
   }
   private void Form1_Load(object sender, EventArgs e)
   {
      // Creating and setting the
      // properties of the labels
      Label l1 = new Label();
      l1.Location = new Point(348, 61);
      l1.Size = new Size(215, 20);
      l1.Text = "Form";
      l1.Font = new Font("Bodoni MT", 12);
      this.Controls.Add(l1);
      Label l2 = new Label();
      l2.Location = new Point(242, 136);
      l2.Size = new Size(103, 20);
      l2.Text = "Enter Age";
      l2.Font = new Font("Bodoni MT", 12);
      this.Controls.Add(l2);
      // Creating and setting the
      // properties of NumericUpDown
      NumericUpDown nbox = new NumericUpDown();
      nbox.Location = new Point(386, 130);
      nbox.Size = new Size(126, 26);
      nbox.Font = new Font("Bodoni MT", 12);
      nbox.Value = 18;
      nbox.Minimum = 18;
      nbox.Maximum = 30;
      nbox.BackColor = Color.LightGreen;
      nbox.ForeColor = Color.DarkGreen;
      nbox.Increment = 1;
      nbox.Name = "MySpinBox";
      // Adding this control
      // to the form
      this.Controls.Add(nbox);
   }
}
```

}

Output:



Constructor

CONSTRUCTOR	DESCRIPTION
NumericUpDown()	This Constructors is used to initialize a new instance of the NumericUpDown class.

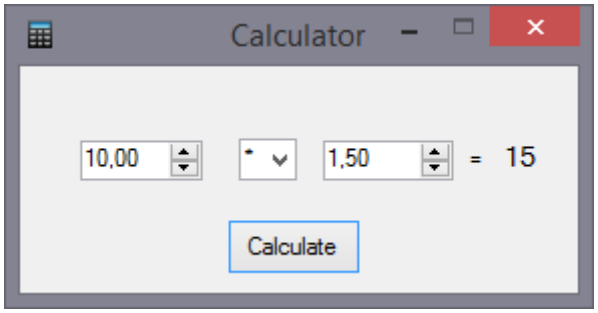
Properties

PROPERTY	DESCRIPTION
AutoSize	This property is used to get or set a value that indicates whether the control resizes based on its contents.
BackColor	This property is used to get or set the background color for the control.
BorderStyle	This property indicates the border style for the control.
Font	This property is used to get or set the font of the text displayed by the control.
ForeColor	This property is used to get or set the foreground color of the control.
Height	This property is used to get or set the height of the control.
Location	This property is used to get or set the coordinates of the upper-left corner of the NumericUpDown control relative to the upper-left corner of its form.
Name	This property is used to get or set the name of the control.
TabStop	This property is used to get or set a value that shows whether the user can press the TAB key to provide the focus to the NumericUpDown.
Size	This property is used to get or set the height and width of the control.
Text	This property is used to get or set the text to be displayed in the NumericUpDown control.
TextAlign	This property is used to get or set the alignment of the text in the spin box (also known as an up-down control).
Visible	This property is used to get or set a value indicating whether the control and all its child controls are displayed.
Width	This property is used to get or set the width of the control.

UpDownAlign	This property is used to get or set the alignment of the up and down buttons on the spin box (also known as an up-down control).
ThousandsSeparator	This property is used to get or set a value indicating whether a thousands separator is displayed in the spin box (also known as an up-down control) when appropriate.
Hexadecimal	This property is used to get or set a value indicating whether the spin box (also known as an up-down control) should display the value it contains in hexadecimal format.
Increment	This property is used to get or set the value to increment or decrement the spin box (also known as an up-down control) when the up or down buttons are clicked.

Simple Calculator in C# .NET Windows Forms

We introduced Windows Forms and created a window with a text label. In today's C# .NET tutorial, we're going to take a look at events and create a simple calculator. It'll look like this:



Form preparation

Create a new Windows Forms project named Calculator. We'll rename the form to CalculatorForm. We usually start our applications with form design. From the Toolbox, we'll drag a few controls into it. We're going to need:

- 2x Label
- 1x Button
- 2x NumericUpDown
- 1x ComboBox

Label

We already know the Label, it's simply a text label.
If we don't use a control in the code, we don't have to name it. If we do, we should set the Name property of it (in the Properties window, the property name is in parentheses (Name)), then we'll be able to access this control using this name. I recommend switching the properties from the categorized view to alphabetical order (first 2 icons in the Properties window), you'll find properties faster. Name is the name of the control, Text is what is written on the control. This logically implies that we can have multiple controls with the same text on a form, but they have to have different names.

One label will only serve as a label with the text "=", so let's set it. The second Label will be used for displaying the result, and since we want to enter the value into it programmatically, we'll set its Name property to resultLabel. We'll set the text to "0". We can also increase the font to 10.

Button

The button is simply a button that calls a method (more precisely, an event) when clicked. In this case, we'll name this button as calculateButton and we set its Text to "Calculate". We'll assign the event to the button later.

NumericUpDown

NumericUpDown is the first control for entering a value we're going to introduce. By default, we can only enter an integer in it. We can change this behavior by setting the DecimalPlaces property, which specifies the number of decimal places. We'll set this value to 2 for both controls we put in our form. We also set their Minimum and Maximum properties. In our case, the minimum should be some low value and the maximum some high value, let's say -1000000 and 1000000. To use the maximum values of a given data type, we'd have to set the limits in the form code using the MaxValue and MinValue properties of the respective data type.

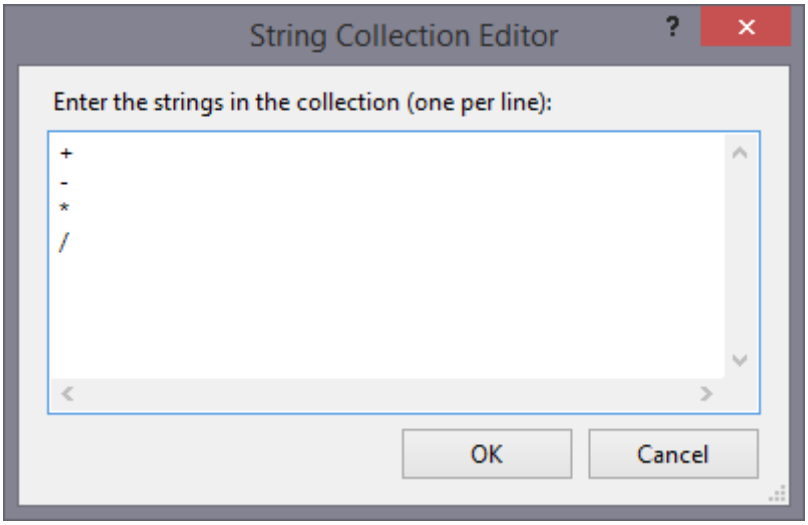
The advantage of entering numbers using this control is that the user isn't able to enter any nonsense value. If we parsed the number from a TextBox (which we'll show you in the next lessons), our application might crash when it gets an invalid value. It's always easier to choose the right control than to check the user's input.

We'll name the controls as number1NumericUpDown and number2NumericUpDown. Note that the name should always contain the control type. For example, we can have both ageLabel and ageNumericUpDown, where ageLabel is the label of the age field, and ageNumericUpDown is the field. Moreover, it makes easier to orientate in the code. Sometimes names as numberNmr, calculateBtn, etc. are used as well.

ComboBox

We're almost there. ComboBox is a drop-down list with several predefined items. The items can be either added in the designer or specified in the code, even while the program is running. This applies to all controls - all their properties from the designer can also be set in the code. However, some advanced properties can only be set from the code and aren't present in the designer.

We'll name the control operationComboBox and then click the "..." button in the Items property. Inside the newly opened window, we'll list the options that can be selected in the comboBox. We write each option on a separated line. In our case, those options are +, -, *, /.



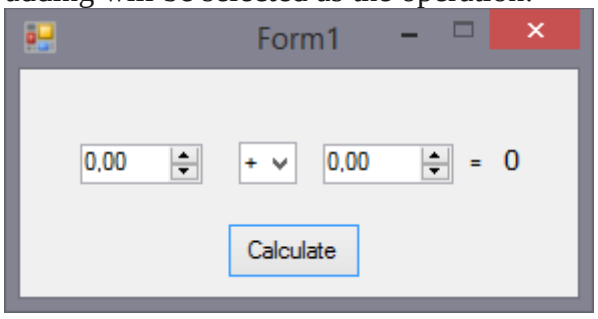
The items don't have to be strings, we can pass objects as well. We'll show this later.
Unfortunately, the selected item can only be set from the code.
We'll arrange the controls on the form as shown at the beginning of this lesson.

Form code

We'll move to the source code of the form. We already know we do this by pressing Ctrl + Alt + 0 or by right-clicking on the form and selecting View Code.
Inside the form constructor, below the InitializeComponents() method call, we'll set the selected item of the operationComboBox. To do this, we'll set the SelectedIndex property to 0, thus the first item:

```
public CalculatorForm()
{
    InitializeComponent();
    operationComboBox.SelectedIndex = 0;
}
```

Of course, we can access all the form's items from the form.
Into the constructor, we write the code that should be executed right after the form is created.
When you run the app, adding will be selected as the operation:



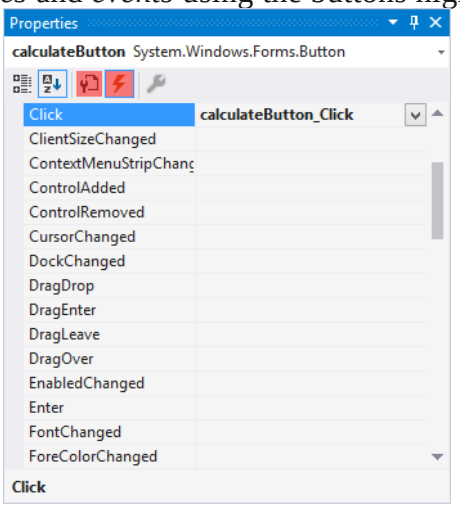
Event handler

Now all we have to do is to respond to the button's click event. We'll move back from the code to the form again, then we'll double-click the button. A new method will be automatically added into the code:

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

```
}
```

The method header should remind you of EventHandler. In the CalculatorForm.Designer.cs file, we can find code that assigns this method to the button's event. In case you don't understand the previous sentences, it doesn't mind at all. All you need to know is this method is called when the button is clicked.
Let's go back to the designer (Shift + F7) to select the button. In the Properties window, we can switch between *properties* and *events* using the buttons highlighted in red below:



Here we see our Click event. This is the place from which we're able to remove it or add it again. Some controls have special events for which we can generate methods from here.

- Never delete events just by removing the handler method from the code. The designer would stop working, and you'd have to fix its file (specifically, remove the assignment of a no longer existent method to the event). The only correct way is to use the designer.

Calculation

Let's move to the calculation itself. The code won't be complicated at all. We'll simply just use conditions for the operationComboBox items and calculate the result inside the event handler of the button accordingly. Then we'll set the result as the text of resultLabel. We shouldn't forget to handle division by zero.
The event handling method's code may look like this:

```
private void calculateButton_Click(object sender, EventArgs e)
{
    // variables setup
    string operation = operationComboBox.SelectedItem.ToString();
    double number1 = Convert.ToDouble(number1NumericUpDown.Value);
    double number2 = Convert.ToDouble(number2NumericUpDown.Value);
    double result = 0;

    // calculation
    if (operation == "+")
        result = number1 + number2;
    else if (operation == "-")
        result = number1 - number2;
```

```
        else if (operation == "*")
            result = number1 * number2;
        else if (operation == "/")
        {
            if (number2 != 0)
                result = number1 / number2;
            else
                MessageBox.Show("You can't divide by zero");
        }
        resultLabel.Text = result.ToString();
    }
```

First, we store the values from the controls in variables to make the code more readable. We access the selected comboBox item using the SelectedItem property, which is of the object type. This means we have to convert it to string in our case. Similarly, we could also use just the item index using SelectedIndex. Since NumericUpDown returns the value in its Value property which is of the decimal type, we must convert it to the double type using the Convert class.

For the case of zero divisor, we display a MessageBox using the static class of the same name and calling the Show() method. Finally, we display the result in resultLabel. Unlike the console, where we could simply print the numbers, we must first convert the numbers to string here.

You can also set the Icon property of the form (by selecting the icon file), Text to "Calculator" and StartPosition to CenterScreen. Like this, the form will be created in the center of the screen. If we set FormBorderStyle to FixedSingle, the form can't be resized, which fits our application. We can also disable window maximization using the MaximizeBox property.

References:

- Tony Gaddis, “Starting out with Visual C#.”, Fourth edition, Boston, Pearson Inc., 2017.
- Salvatore A. Buono, “C# and Game Programming: A Beginner's Guide.” Second Edition, Boca Raton, CRC Press Inc., 2019.
- Eric Butow and Tommy Ryan, "C#: Your Visual Blueprint for Building .NET Applications.", Hungry Minds Inc., New York, 2002.
- Faraz Rasheed, “Programmer's Heaven: C# School.”, First Edition, Fuengirola, Synchron Data, 2006.