C# Windows Forms Application

Windows Forms is a Graphical User Interface(GUI) class library which is bundled in *.Net Framework*.

Its main purpose is to provide an easier interface to develop the applications for desktop, tablet, PCs. It is also termed as the **WinForms**.

The applications which are developed by using Windows Forms or WinForms are known as the **Windows Forms Applications** that runs on the desktop computer.

WinForms can be used only to develop the Windows Forms Applications not web applications.

WinForms applications can contain the different type of controls like labels, list boxes, tooltip etc.

Creating a Windows Forms Application Using Visual Studio 2017

First, open the Visual Studio then Go to **File -> New -> Project** to create a new project and then select the language as *Visual C#* from the left menu.
Click on *Windows/Desktop Forms App(.NET Framework)* in the middle of current window. After that give the project name and Click **OK**.

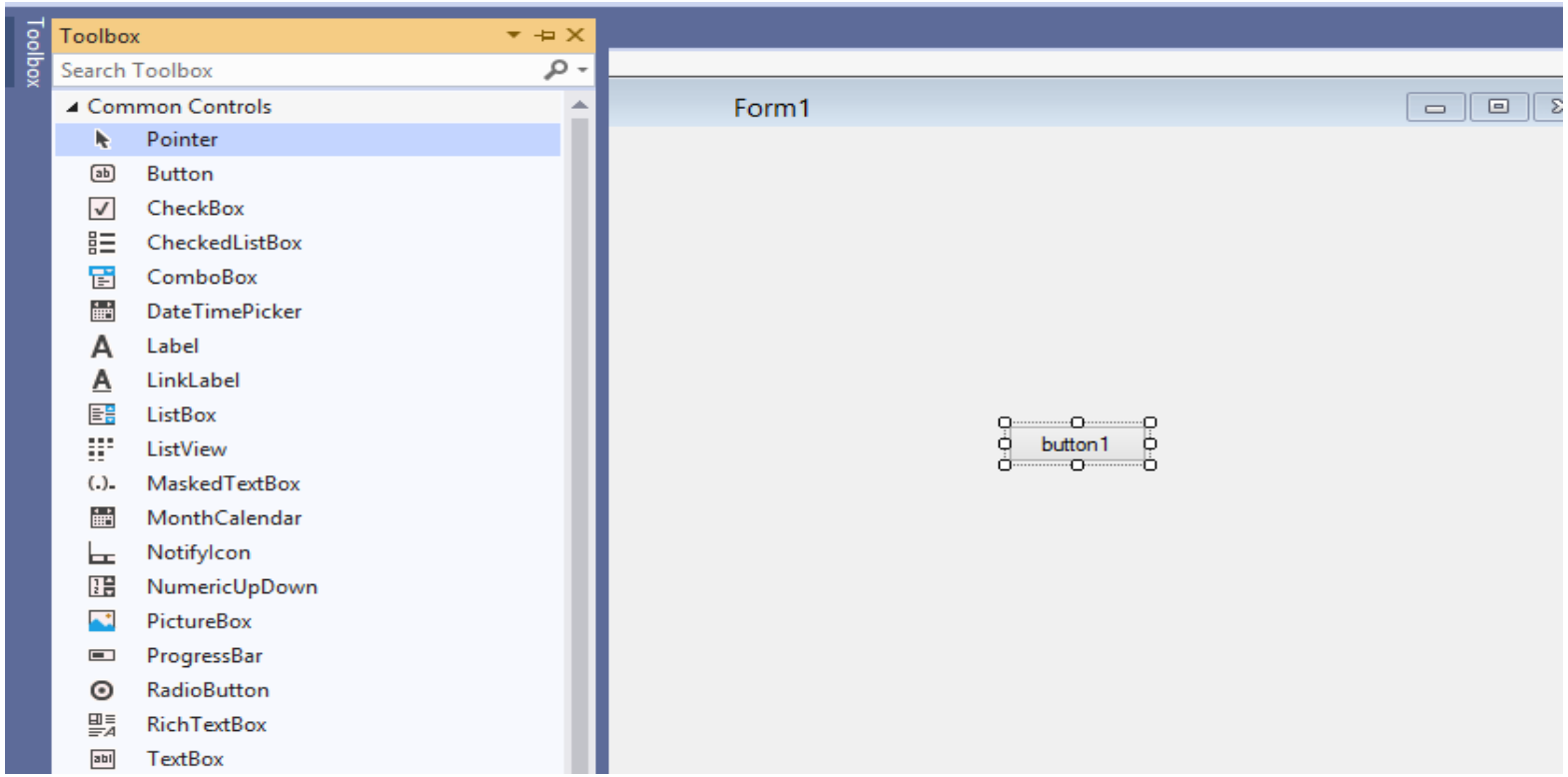
Here the solution is like a container which contains the projects and files that may be required by the program.

After that following window will display which will be divided into three parts as follows:

- **Editor Window or Main Window:** Here, you will work with forms and code editing. You can notice the layout of form which is now blank. You will double click the form then it will open the code for that.
- **Solution Explorer Window:** It is used to navigate between all items in solution. For example, if you will select a file form this window then particular information will be display in the property window.
- **Properties Window:** This window is used to change the different properties of the selected item in the Solution Explorer. Also, you can change the properties of components or controls that you will add to the forms.

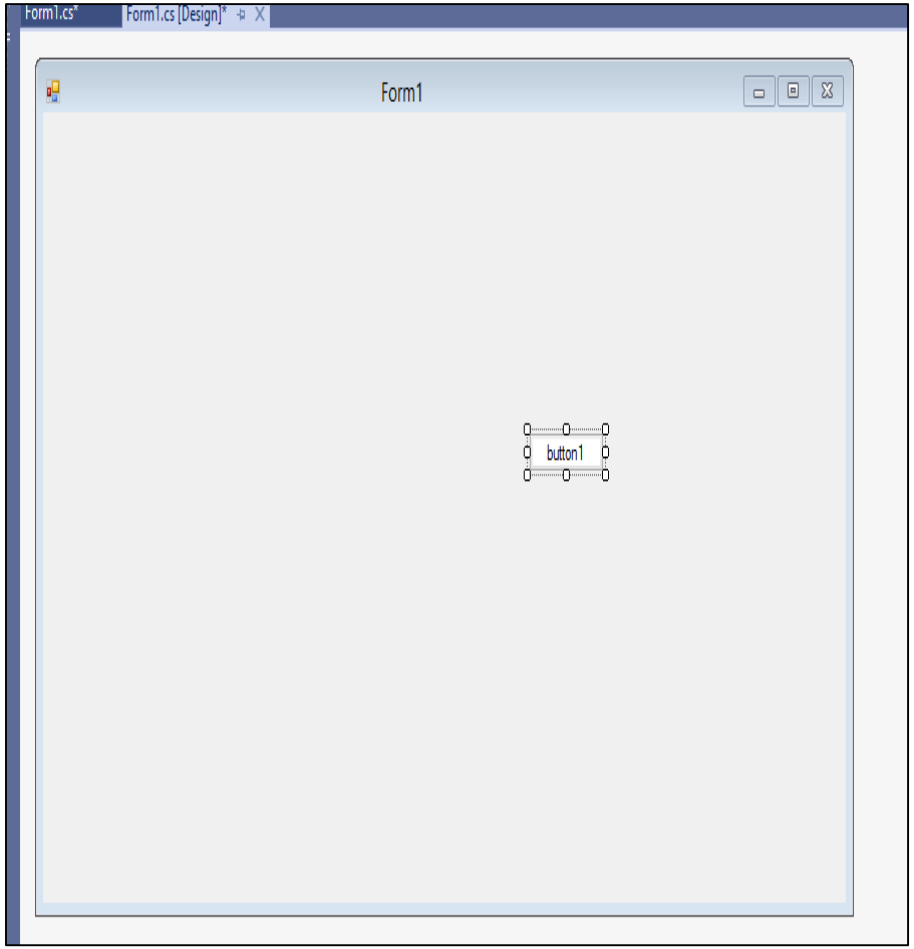
You can also reset the window layout by setting it to default. To set the default layout, go to **Window -> Reset Window Layout** in Visual Studio Menu.

Now **to add the controls to your WinForms application** go to **Toolbox** tab present in the extreme left side of Visual Studio. Here, you can see a list of controls. To access the most commonly used controls go to **Common Controls** present in Toolbox tab.

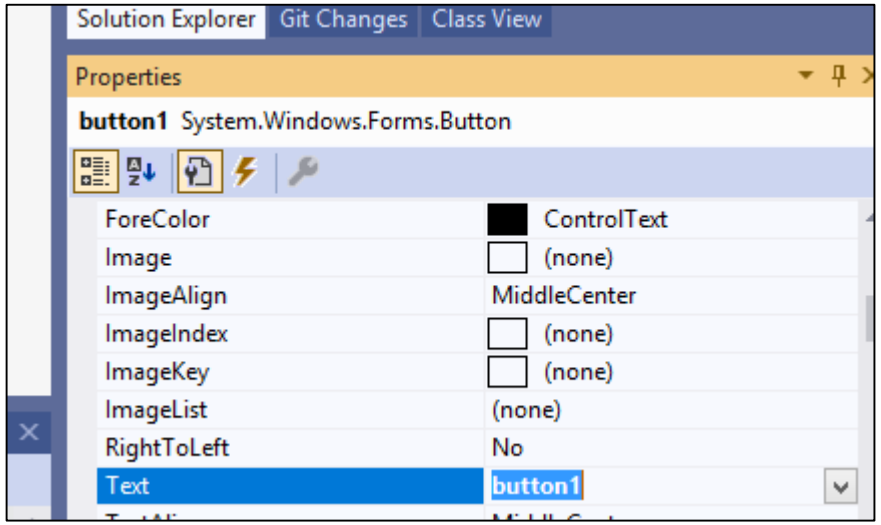


Now drag and drop the controls that you needed on created Form. For example, if you can add TextBox, ListBox, Button etc. as shown below. By clicking on the particular dropped control you can see and change its properties present in the right most corner of Visual Studio.

BUTTON



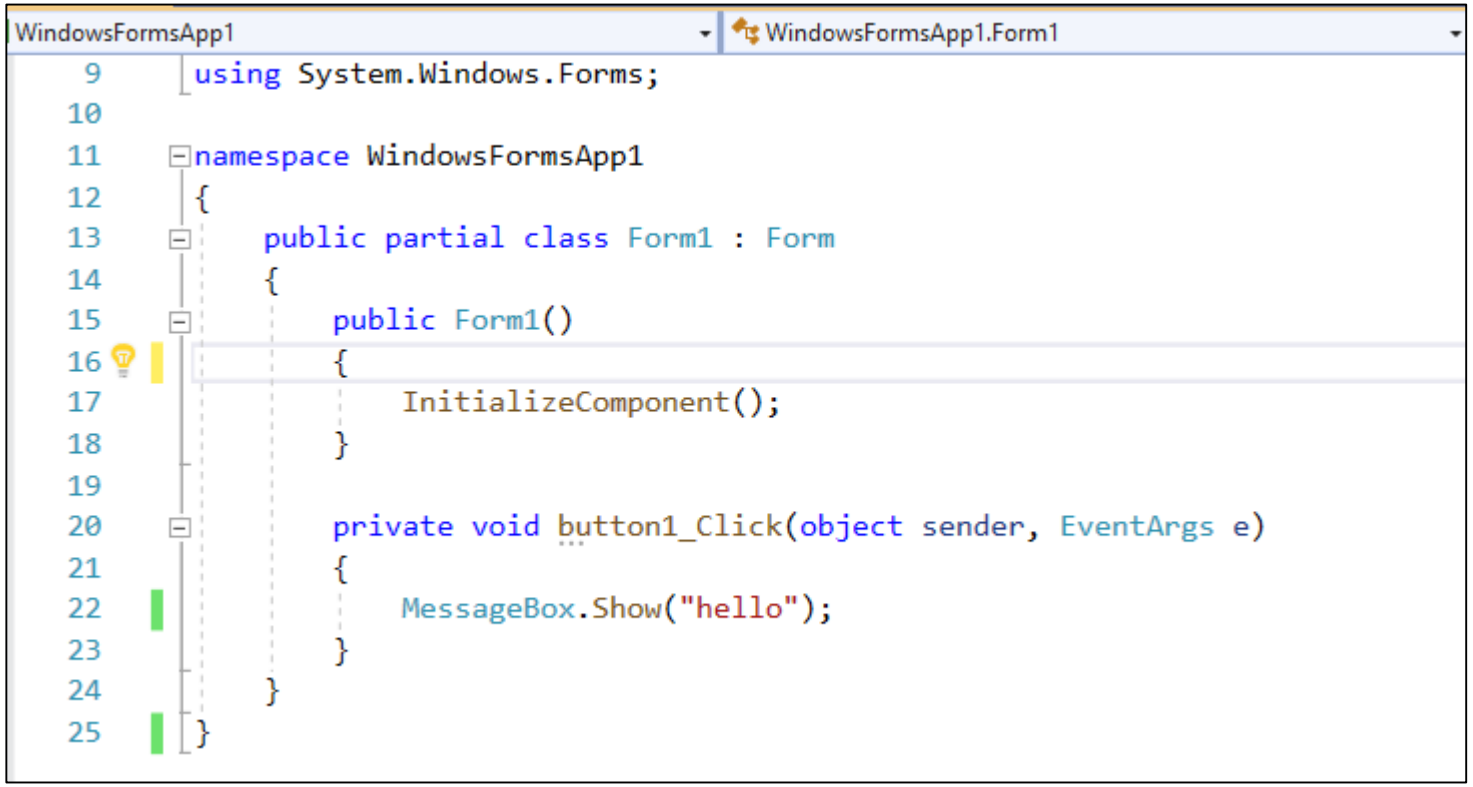
BUTTON PROPERTIES



To run the program you can use an **F5 key** or **Play button** present in the toolbar of Visual Studio. To stop the program you can use pause button present in the ToolBar. You can also run the program by going to **Debug->Start Debugging** menu in the menubar.

Adding the Event Handlers

Example 1:button click event with message:

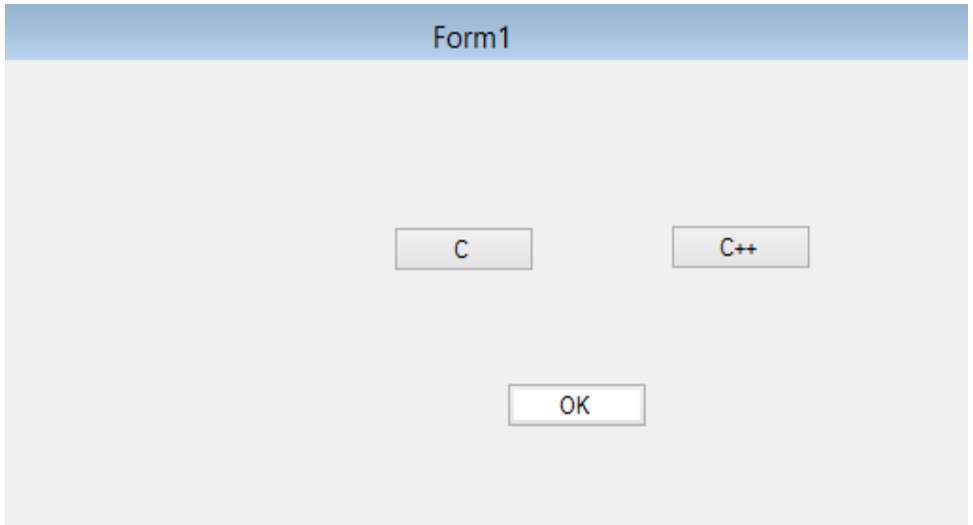


Events

When a user clicks a button or presses a button, you as the programmer of the application, want to be told that this has happened. To do so, controls use events. The Control class defines a number of events that are common to the controls

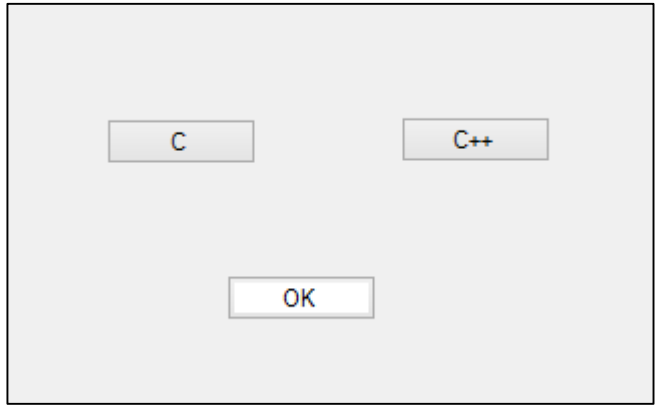
Name	Description
Click	Occurs when a control is clicked. In some cases, this event will also occur when a user presses Enter.
DoubleClick	Occurs when a control is double-clicked. Handling the Click event on some controls, such as the Button control will mean that the DoubleClick event can never be called.
DragDrop	Occurs when a drag-and-drop operation is completed, in other words, when an object has been dragged over the control, and the user releases the mouse button.
DragEnter	Occurs when an object being dragged enters the bounds of the control.
DragLeave	Occurs when an object being dragged leaves the bounds of the control.
DragOver	Occurs when an object has been dragged over the control.
KeyPress	Occurs when a key becomes pressed, while a control has focus. This event always occurs after KeyDown and before KeyUp. The difference between KeyDown and KeyPress is that KeyDown passes the keyboard code of the key that has been pressed, while KeyPress passes the corresponding char value for the key.
KeyUp	Occurs when a key is released while a control has focus. This event always occurs after KeyDown and KeyPress.
GotFocus	Occurs when a control receives focus. Do not use this event to perform validation of controls. Use Validating and Validated instead.
LostFocus	Occurs when a control looses focus. Do not use this event to perform validation of controls. Use Validating and Validated instead.

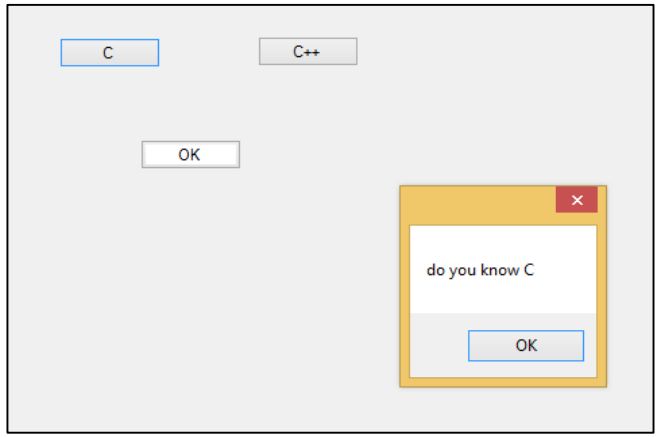
MouseDown	Occurs when the mouse pointer is over a control and a mouse button is pressed. This is not the same as a Click event because MouseDown occurs as soon as the button is pressed and before it is released.
MouseMove	Occurs continually as the mouse travels over the control.
MouseUp	Occurs when the mouse pointer is over a control and a mouse button is released.
Paint	Occurs when the control is drawn.

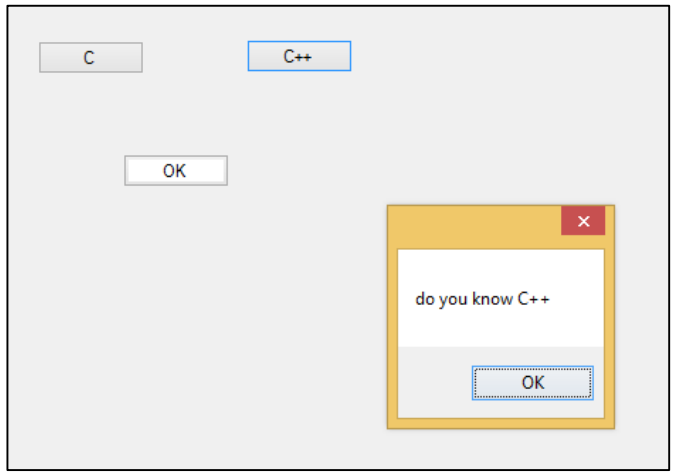

Form1
C
C++
OK

Example:

```
using System;
using System.Windows.Forms;
namespace WindowsFormsApp1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            Application.Exit();
        }

        private void button3_Click(object sender, EventArgs e)
        {
            MessageBox.Show("do you know C++");
        }

        private void button2_Click(object sender, EventArgs e)
        {
            MessageBox.Show("do you know C");
        }
    }
}
```







The Button Control

Button Properties

Name	Availability	Description
FlatStyle	Read/Write	The style of the button can be changed with this property. If you set the style to PopUp, the button will appear flat until the user moves the mouse pointer over it. When that happens, the button pops up to its normal 3D look.
Enabled	Read/Write	We'll mention this here even though it is derived from Control, because it's a very important property for a button. Setting the Enabled property to false means that the button becomes grayed out and nothing happens when you click it.
Image	Read/Write	Allow you to specify an image (bitmap, icon etc.), which will be displayed on the button.
ImageAlign	Read/Write	With this property, you can set where the image on the button should appear.

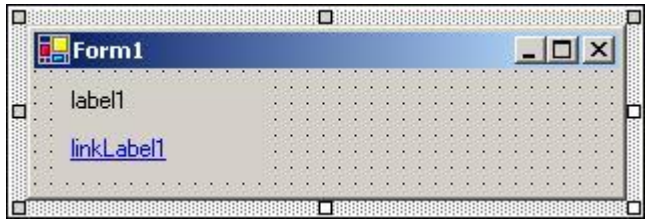
Property	Description
BackColor	Using BackColor property you can set the background color of the button.
BackgroundImage	Using BackgroundImage property you can set the background image on the button.
AutoEllipsis	Using AutoEllipsis property you can set a value which shows that whether the ellipsis character (…) appears at the right edge of the control which denotes that the button text extends beyond the specified length of the button.
AutoSize	Using AutoSize property you can set a value which shows whether the button resizes based on its contents.
Enabled	Using Enabled property you can set a value which shows whether the button can respond to user interaction.
Events	Using Events property you can get the list of the event handlers that are applied on the given button.
Font	Using Font property you can set the font of the button.
FontHeight	Using FontHeight property you can set the height of the font.
ForeColor	Using ForeColor property you can set the foreground color of the button.
Height	Using Height property you can set the height of the button.
Image	Using Image property you can set the image on the button.
Margin	Using Margin property you can set the margin between controls.
Name	Using Name property you can set the name of the button.
Padding	Using Padding property you can set the padding within the button.
Visible	Using Visible property you can set a value which shows whether the button and all its child buttons are displayed.

The Label and LinkLabel Controls

The Label control is probably the most used control of them all. Look at any Windows application and you'll see them on just about any dialog you can find. The label is a simple control with one purpose only: to present a caption or short hint to explain something on the form to the user.

Out of the box, Visual Studio.NET includes two label controls that are able to present them selves to the user in two distinct ways:

- Label, the standard Windows label
- LinkLabel, a label like the standard one (and derived from it), but presents itself as an internet link (a hyperlink)



Name	Availability	Description
BorderStyle	Read/Write	Allows you to specify the style of the border around the Label. The default is no border.
DisabledLinkColor	Read/Write	(LinkLabel only) The color of the LinkLabel after the user has clicked it.
FlatStyle	Read/Write	Controls how the control is displayed. Setting this property to PopUp will make the control appear flat until the user moves the mouse pointer over the control. At that time, the control will appear raised.
Image	Read/Write	This property allows you to specify a single image (bitmap, icon, and so on.) to be displayed in the label.
ImageAlign	Read/Write	(Read/Write) Where in the Label the image is shown.
LinkArea	Read/Write	(LinkLabel only) The range in the text that should be displayed as a link.
LinkColor	Read/Write	(LinkLabel only) The color of the link.
Links	Read only	(LinkLabel only) It is possible for a LinkLabel to contain more than one link. This property allows you to find the link you want. The control keeps track of the links displayed in the text.
LinkVisited	Read only	(LinkLabel only) Returns whether a link has been visited or not.
Text	Read/Write	The text that is shown in the Label.
TextAlign	Read/Write	Where in the control is the text shown.

The TextBox Control:

Text boxes should be used when you want the user to enter text that you have no knowledge of at design time (for example the name of the user). The primary function of a text box is for the user to enter text, but any characters can be entered, and it is quite possible to force the user to enter numeric values only.

Out of the box .NET comes with two basic controls to take text input from the user: TextBox and RichTextBox .Both controls are derived from a base class called TextBoxBase which itself is derived from Control.

TextBoxBase provides the base functionality for text manipulation in a text box, such as selecting text, cutting to and pasting from the Clipboard, and a wide range of events.

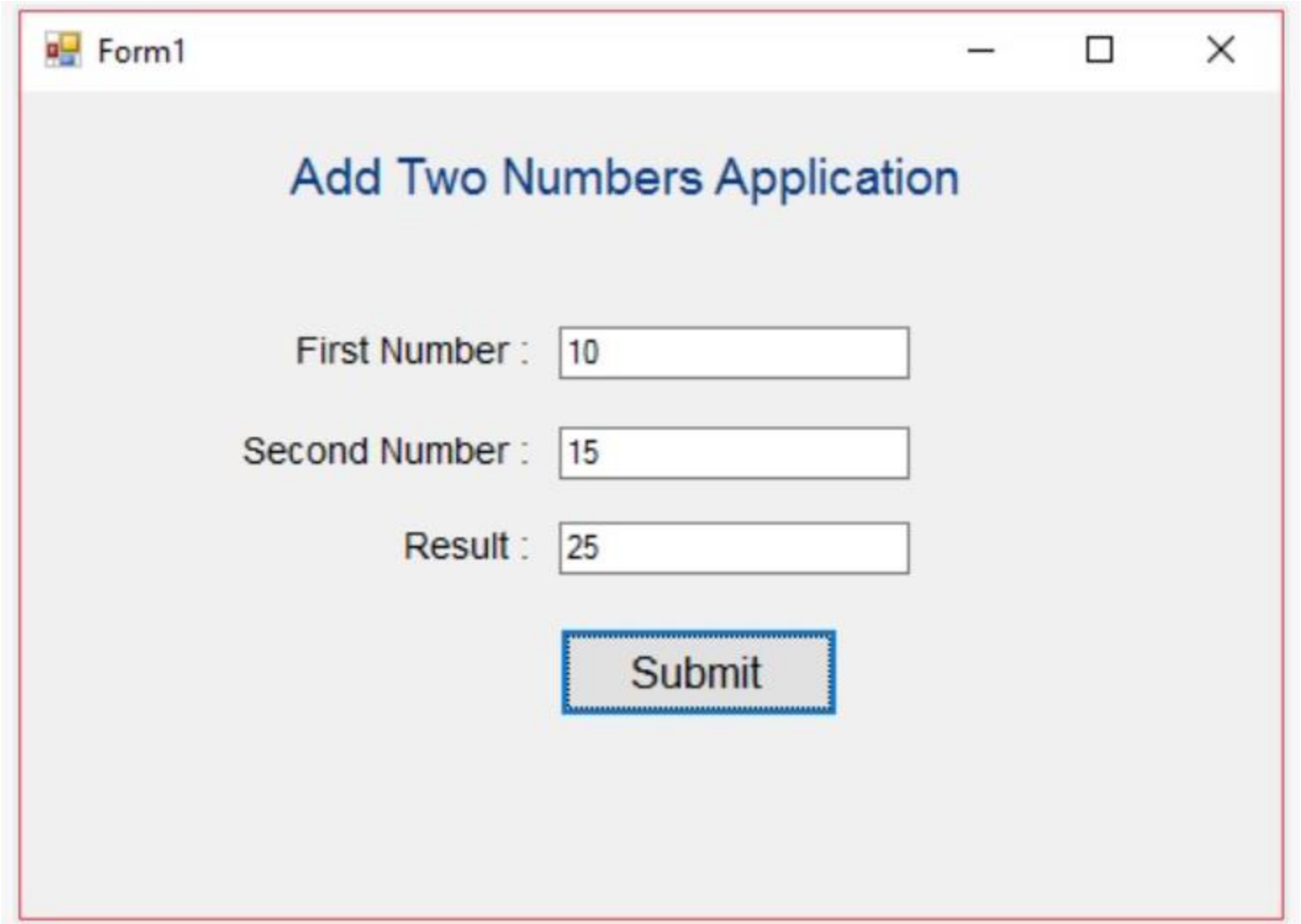
TextBox Properties

Name	Availability	Description
CausesValidation	Read/Write	When a control that has this property set to true is about to receive focus, two events are fired: Validating and Validated. You can handle these events in order to validate data in the control that is losing focus. This may cause the control never to receive focus. The related events are discussed below.
CharacterCasing	Read/Write	A value indicating if the TextBox changes the case of the text entered. The possible values are: q Lower: All text entered into the text box is converted lower case. q Normal: No changes are made to the text. q Upper: All text entered into the text box is converted to upper case.
MaxLength	Read/Write	A value that specifies the maximum length in characters of any text, entered into the TextBox. Set this value to zero it the maximum limit is limited only by available memory.
Multiline	Read/Write	Indicates if this is a multiline control. A multiline control is able to show multiple lines of text.
PasswordChar	Read/Write	Specifies if a password character should replace the actual characters entered into a single line textbox. If the Multiline property is true then this has no effect.
ReadOnly	Read/Write	A Boolean indicating if the text is read only.
ScrollBars	Read/Write	Specifies if a multilane text box should display scrollbars.
SelectedText	Read/Write	The text that is selected in the text box.
SelectionLength	Read/Write	The number of characters selected in the text. If this value is set to be larger than the total number of characters in the text, it is reset by the control to be the total number of characters minus the value of SelectionStart.
SelectionStart	Read/Write	The start of the selected text in a text box.
WordWrap	Read/Write	Specifies if a multiline text box should automatically wrap words if a line exceeds the width of the control.

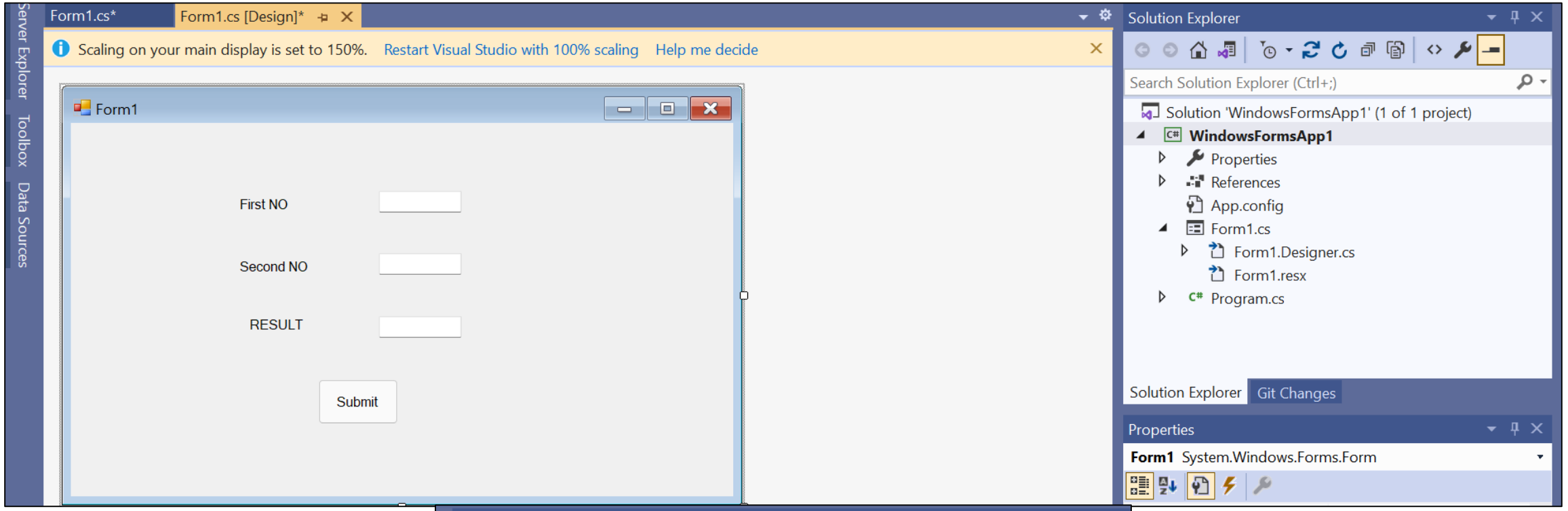
The TextBox control provides these events (all of which are inherited from Control):

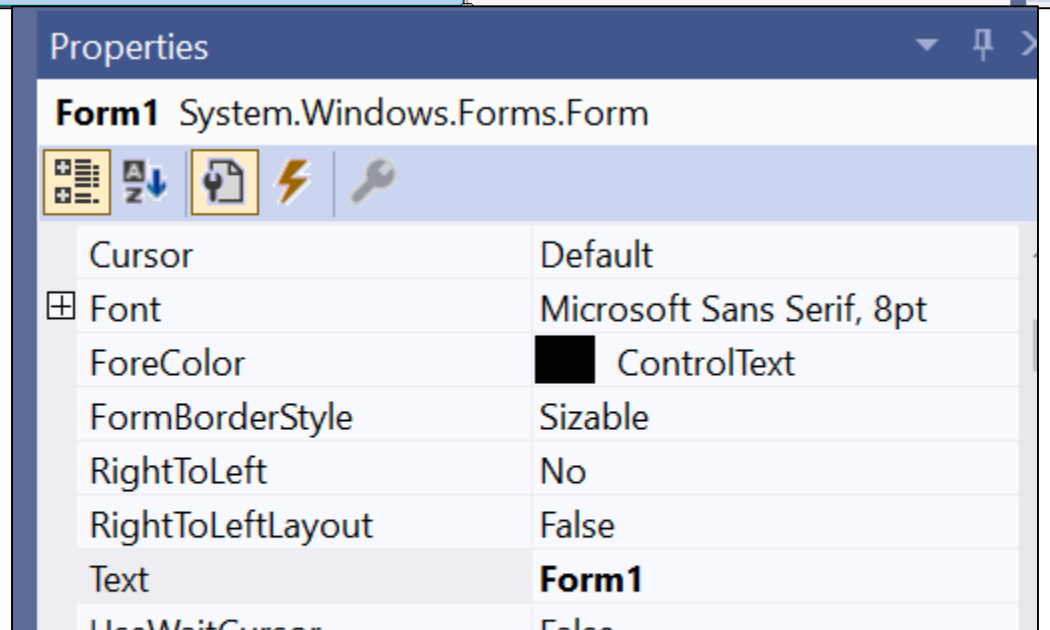
Name	Description
Enter GotFocus Leave Validating Validated LostFocus	These six events occur in the order they are listed here. They are known as "Focus Events" and are fired whenever a controls focus changes, with two exceptions. Validating and Validated are only fired if the control that receives focus has the CausesValidation property set to true. The reason why it's the receiving control that fires the event is that there are times where you do not want to validate the control, even if focus changes. An example of this is if the user clicks a Help button.
KeyDown KeyPress KeyUp	These three events are known as "Key Events". They allow you to monitor and change what is entered into your controls. KeyDown and KeyUp receive the key code corresponding to the key that was pressed. This allows you to determine if special keys such as Shift or Control and F1 were pressed. KeyPress, on the otherhand, receives the character corresponding to a keyboard key. This means that the value for the letter "a" is not the same as the letter "A". It is useful if you want to exclude a range of characters, for example, only allowing numeric values to be entered.
Change	Occurs whenever the text in the textbox is changed, no matter what the change.

Example:


Form1
Add Two Numbers Application
First Number :
10
Second Number :
15
Result :
25
Submit

Design view


Form1.cs*
Form1.cs [Design]*
Scaling on your main display is set to 150%.
Restart Visual Studio with 100% scaling
Help me decide
Form1
First NO
Second NO
RESULT
Submit
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'WindowsFormsApp1' (1 of 1 project)
WindowsFormsApp1
Properties
References
App.config
Form1.cs
Form1.Designer.cs
Form1.resx
Program.cs
Solution Explorer
Git Changes
Properties
Form1 System.Windows.Forms.Form


Properties
Form1 System.Windows.Forms.Form
Cursor
Default
Font
Microsoft Sans Serif, 8pt
ForeColor
ControlText
FormBorderStyle
Sizable
RightToLeft
No
RightToLeftLayout
False
Text
Form1

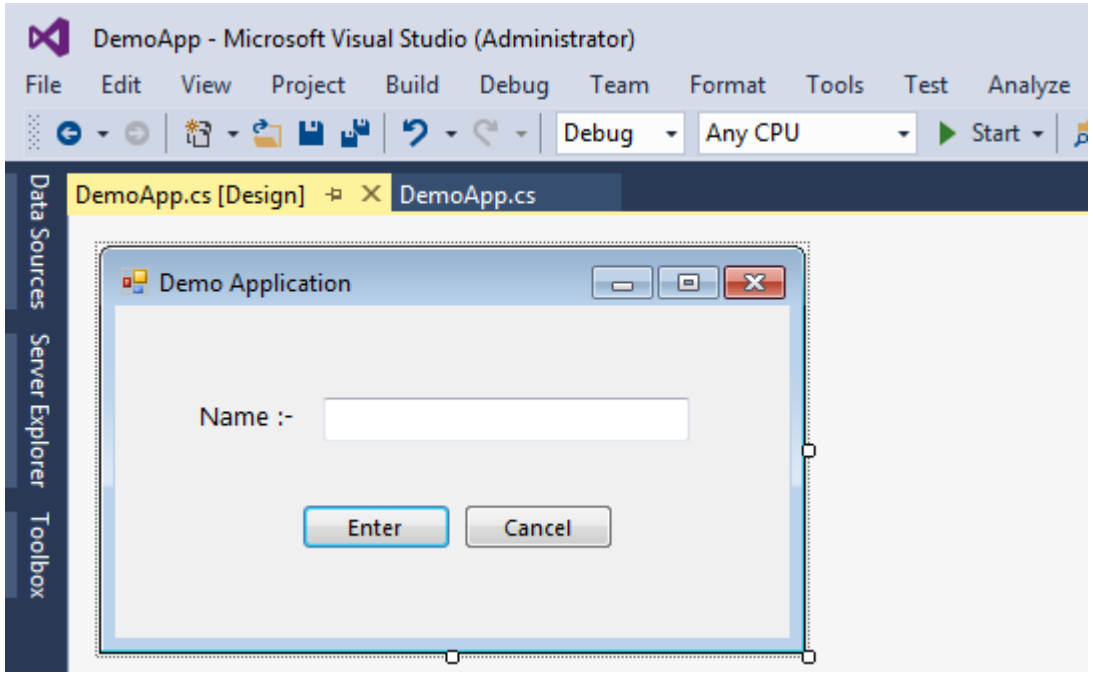
```
using System;
using System.Windows.Forms;
namespace WindowsFormsApp1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Load(object sender, EventArgs e)
        {

        }

                private void button1_Click(object sender, EventArgs e)
        {
            int a, b, c;
            a = Convert.ToInt32(textBox1.Text);
            b = Convert.ToInt32(textBox2.Text);
            c = a + b;
            textBox3.Text = c.ToString();
        } }}
```

Errorprovider class:

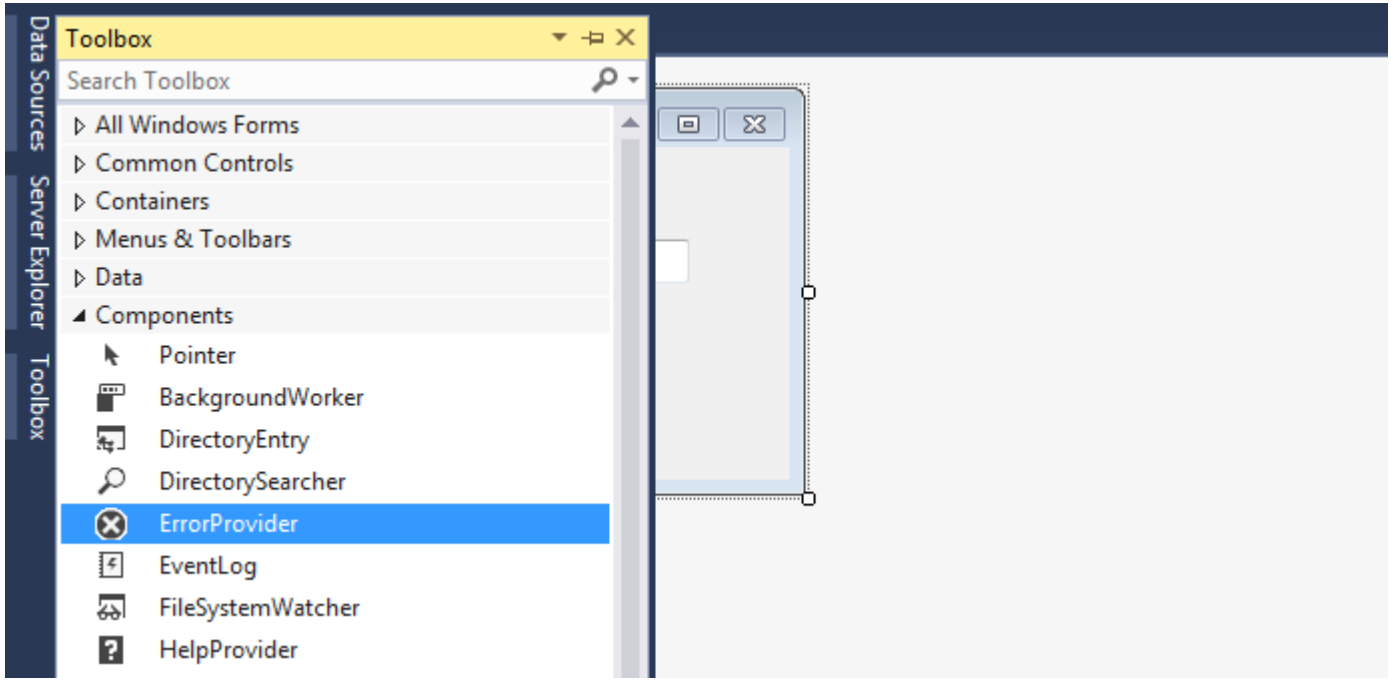
Provides a user interface for indicating that a control on a form has an error associated with it.

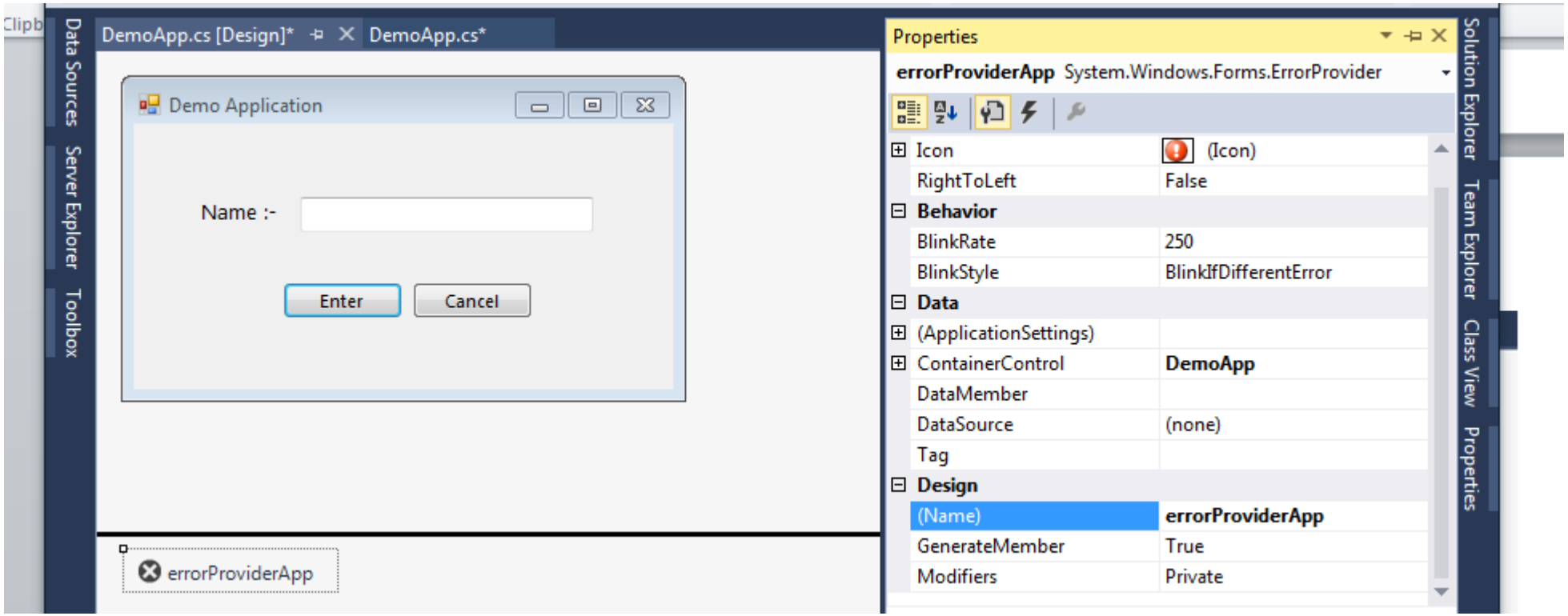
to invoke validate input to Windows Forms controls and display user-friendly informative error messages.

Step 1: Create a Windows form application.

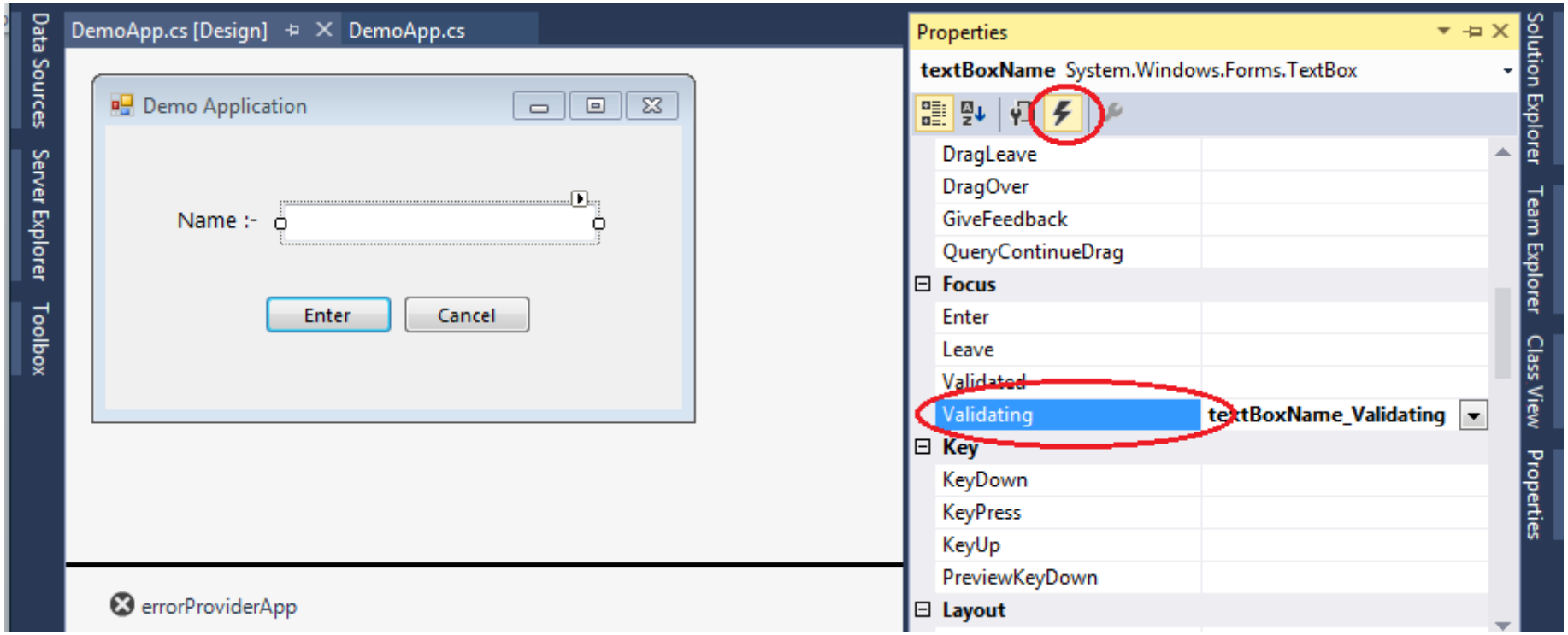


Step 2: Choose “ErrorProvider” form toolbox.




Data Sources
Server Explorer
Toolbox
DemoApp.cs [Design]*
DemoApp.cs*
Demo Application
Name :-
Enter
Cancel
errorProviderApp
Properties
errorProviderApp System.Windows.Forms.ErrorProvider
Icon
(Icon)
RightToLeft
False
Behavior
BlinkRate
250
BlinkStyle
BlinkIfDifferentError
Data
(ApplicationSettings)
ContainerControl
DemoApp
DataMember
DataSource
(none)
Tag
Design
(Name)
errorProviderApp
GenerateMember
True
Modifiers
Private
Solution Explorer
Team Explorer
Class View
Properties

Step 3: Select the Text box and go to its properties.



In properties choose “Events” and under focus double click on “validating”.
Now we have the text box validation method.

```
private void textBox1_Validating(object sender, CancelEventArgs e)
        {
            if (string.IsNullOrWhiteSpace(textBox1.Text))
            {
                e.Cancel = true;
                textBox1.Focus();
                errorProvider1.SetError(textBox1, "Name should not be left blank!");
            }
            else
            {
                e.Cancel = false;
                errorProvider1.SetError(textBox1, "");
            }
        }
```

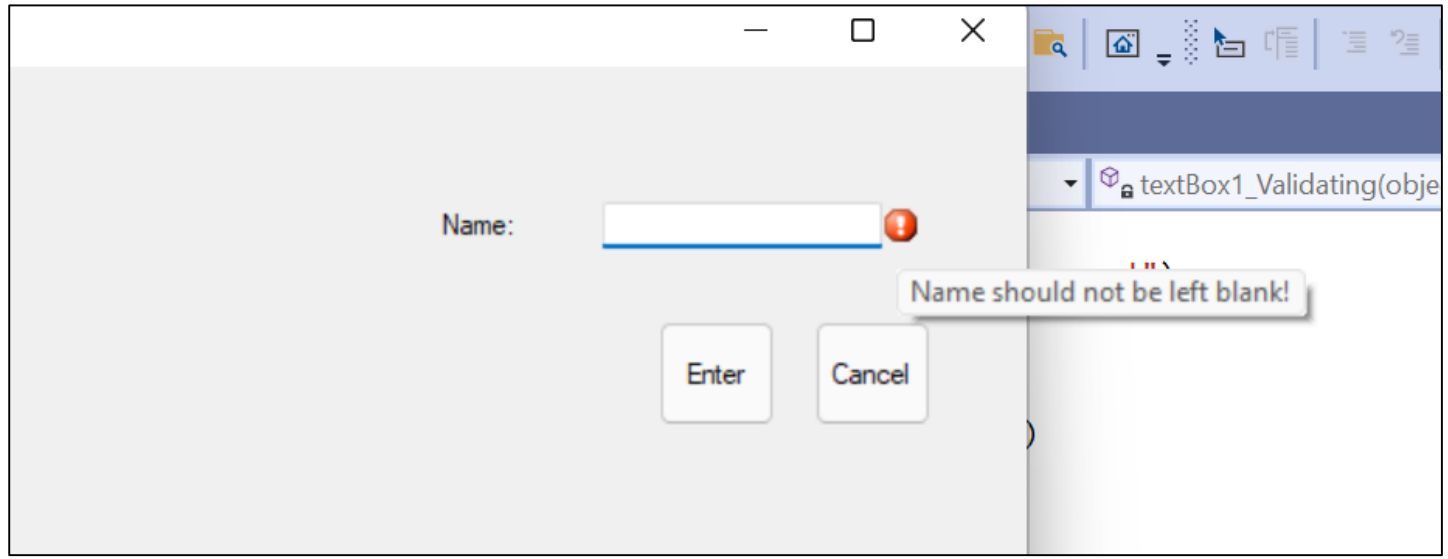
Step 4: Now validation should be triggered on Enter key press. Add following code to Enter key click method.

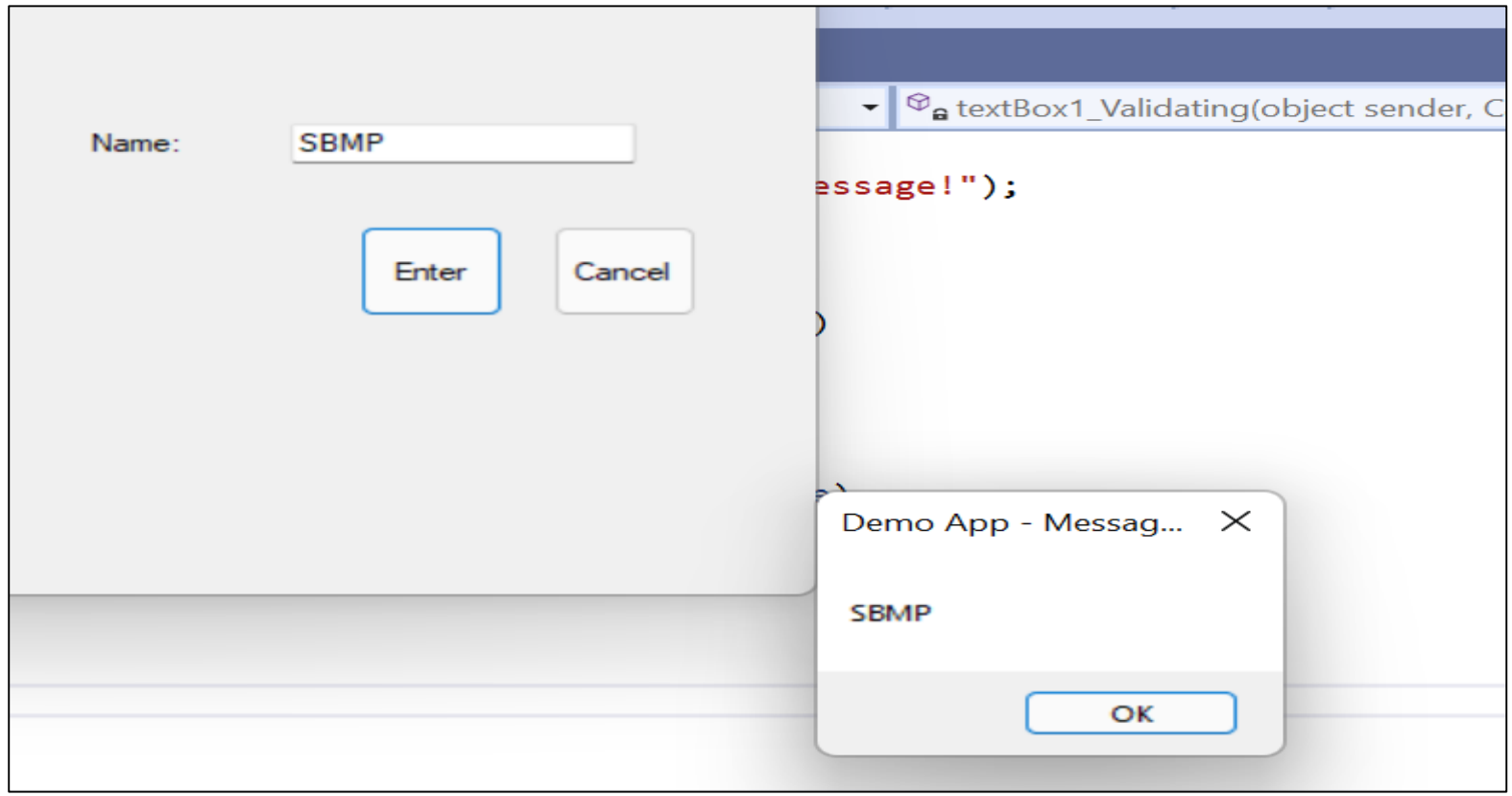
```
private void button1_Click(object sender, EventArgs e)
        {
            if (ValidateChildren(ValidationConstraints.Enabled))
            {
                MessageBox.Show(textBox1.Text, "Demo App - Message!");
            }
        }
```

Entire code:

```
namespace WindowsFormsApp3
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void textBox1_Validating(object sender, CancelEventArgs e)
        {
            if (string.IsNullOrWhiteSpace(textBox1.Text))
            {
                e.Cancel = true;
                textBox1.Focus();
                errorProvider1.SetError(textBox1, "Name should not be left blank!");
            }
            else
            {
                e.Cancel = false;
                errorProvider1.SetError(textBox1, "");
            } }
```

```
        private void button1_Click(object sender, EventArgs e)
        {
            if (ValidateChildren(ValidationConstraints.Enabled))
            {
                MessageBox.Show(textBox1.Text, "Demo App - Message!");
            }
        }
       private void label1_Click(object sender, EventArgs e)
        {

        }

        private void button2_Click(object sender, EventArgs e)
        {
            Application.Exit();
        }
    }
}
```


Name:
Name should not be left blank!
Enter
Cancel
textBox1_Validating(obje


Name:
SBMP
Enter
Cancel
textBox1_Validating(object sender, C
essage!");
Demo App - Messag...
SBMP
OK

TextChanged and KeyDown

```
C# program that uses TextBox and TextChanged event
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 WindowsFormsApplication1
{
   public partial class Form1 : Form
   {
            public Form1()
            {
               InitializeComponent();
            }
            private void textBox1_TextChanged(object sender, EventArgs e)
            {

               // This changes the main window text when you type into the TextBox.
               this.Text = textBox1.Text;
            }}}
```

Windows Forms class that uses KeyDown on TextBox

```
namespace WindowsFormsApplication1
{
   public partial class Form1 : Form
   {
            public Form1()
            {
               InitializeComponent();
            }

            private void textBox1_KeyDown(object sender, KeyEventArgs e)
            {
               // Detect the KeyEventArg's key enumerated constant.
               if (e.KeyCode == Keys.Enter)
               {
                        MessageBox.Show("You pressed enter! Good job!");
               }
               else if (e.KeyCode == Keys.Escape)
               {
                        MessageBox.Show("You pressed escape! What's wrong?");
               }
            }
   }
}
```

The RadioButton and CheckBox Controls

RadioButton Properties

Name	Availability	Description
Appearance	Read/Write	A RadioButton can be displayed either as a label with a circular check to the left, middle or right of it, or as a standard button. When it is displayed as a button, the control will appear pressed when selected and 3D otherwise.
AutoCheck	Read/Write	When this property is true, a check mark is displayed when the user clicks the radio button. When it is false, the check mark is not displayed by default.
CheckAlign	Read/Write	By using this property, you can change the alignment of the radio button. It can be left, middle, and right.
Checked	Read/Write	Indicates the status of the control. It is true if the control has a check mark, and false otherwise.

RadioButton Events

Name	Description
CheckChanged	This event is sent when the check of the RadioButton changes. If there is more than one RadioButton control on the form or within a group box, this event will be sent twice, first to the control, which was checked and now becomes unchecked, then to the control which becomes checked.
Click	This event is sent every time the RadioButton is clicked. This is not the same as the change event, because clicking a RadioButton twice or more times in succession only changes the checked property once – and only if it wasn't checked already.

CheckBox Properties

Name	Availability	Description
CheckState	Read/Write	Unlike the RadioButton, a CheckBox can have three states: Checked, Indeterminate, and Unchecked. When the state of the check box is Indeterminate, the control check next to the label is usually grayed, indicating that the current value of the check is not valid or has no meaning under the current circumstances. An example of this state can be seen if you select several files in the Windows Explorer and look at their properties. If some files are readonly and others are not, the readonly checkbox will be checked, but greyed – indeterminate.
ThreeState	Read/Write	When this property is false, the user will not be able to change the CheckBox' state to Indeterminate. You can, however, still change the state of the check box to Indeterminate from code.

CheckBox Events

Name	Description
CheckedChanged	Occurs whenever the Checked property of the check box changes. Note that in a CheckBox where the ThreeState property is true, it is possible to click the check box without changing the Checked property. This happens when the check box changes from checked to indeterminate state.
CheckedStateChanged	Occurs whenever the CheckedState property changes. As Checked and Unchecked are both possible values of the CheckedState property, this event will be sent whenever the Checked property changes. In addition to that, it will also be sent when the state changes from Checked to Indeterminate.

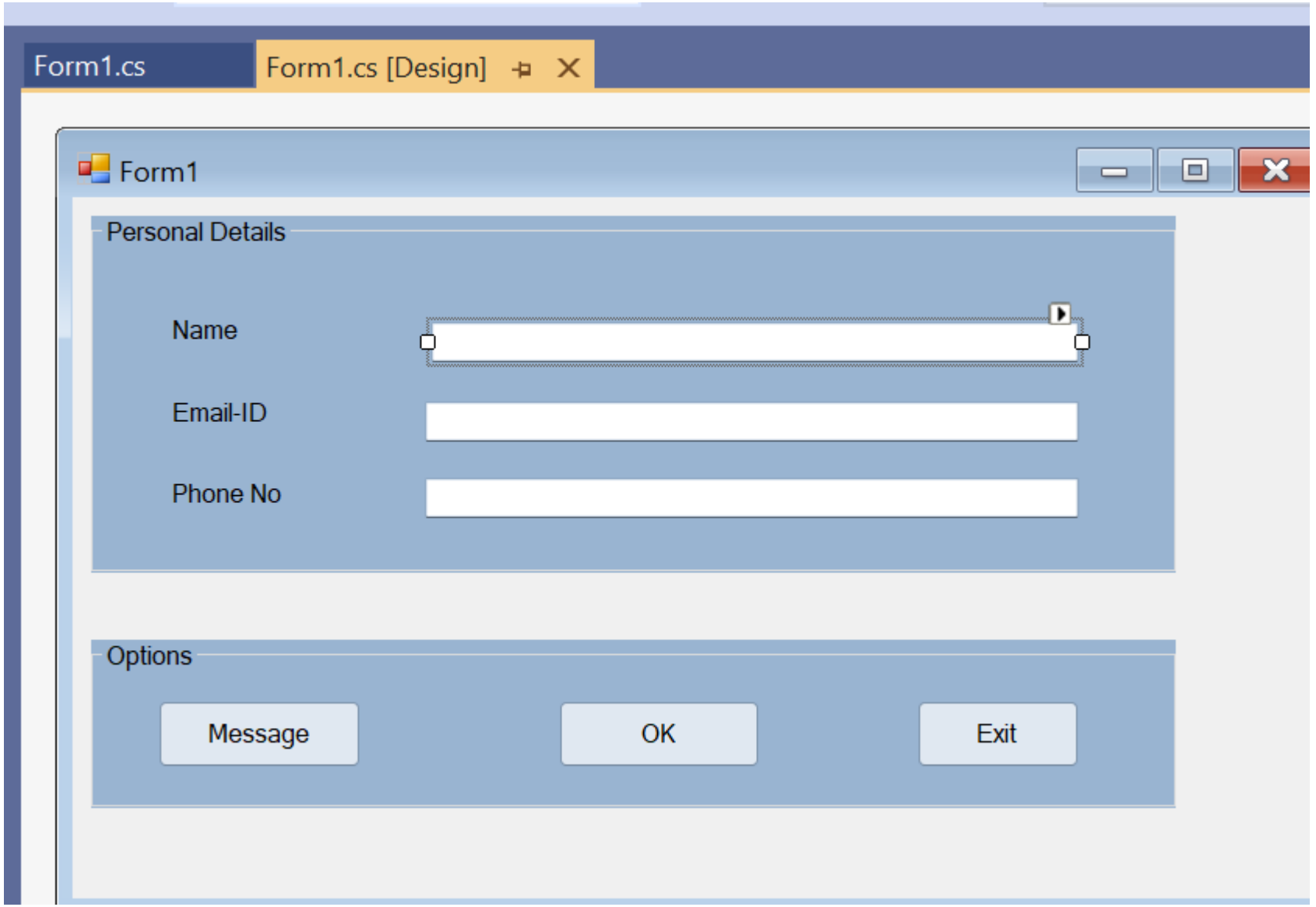
The GroupBox Control:

- This control is often used in conjunction with the RadioButton and CheckBox controls to display a frame around, and a caption above, a series of controls that are logically linked in some way.

- Using the group box is as simple as dragging it onto a form, and then dragging the controls it should contain onto it (but not the other way round – you can't lay a group box over some pre-exisiting controls).

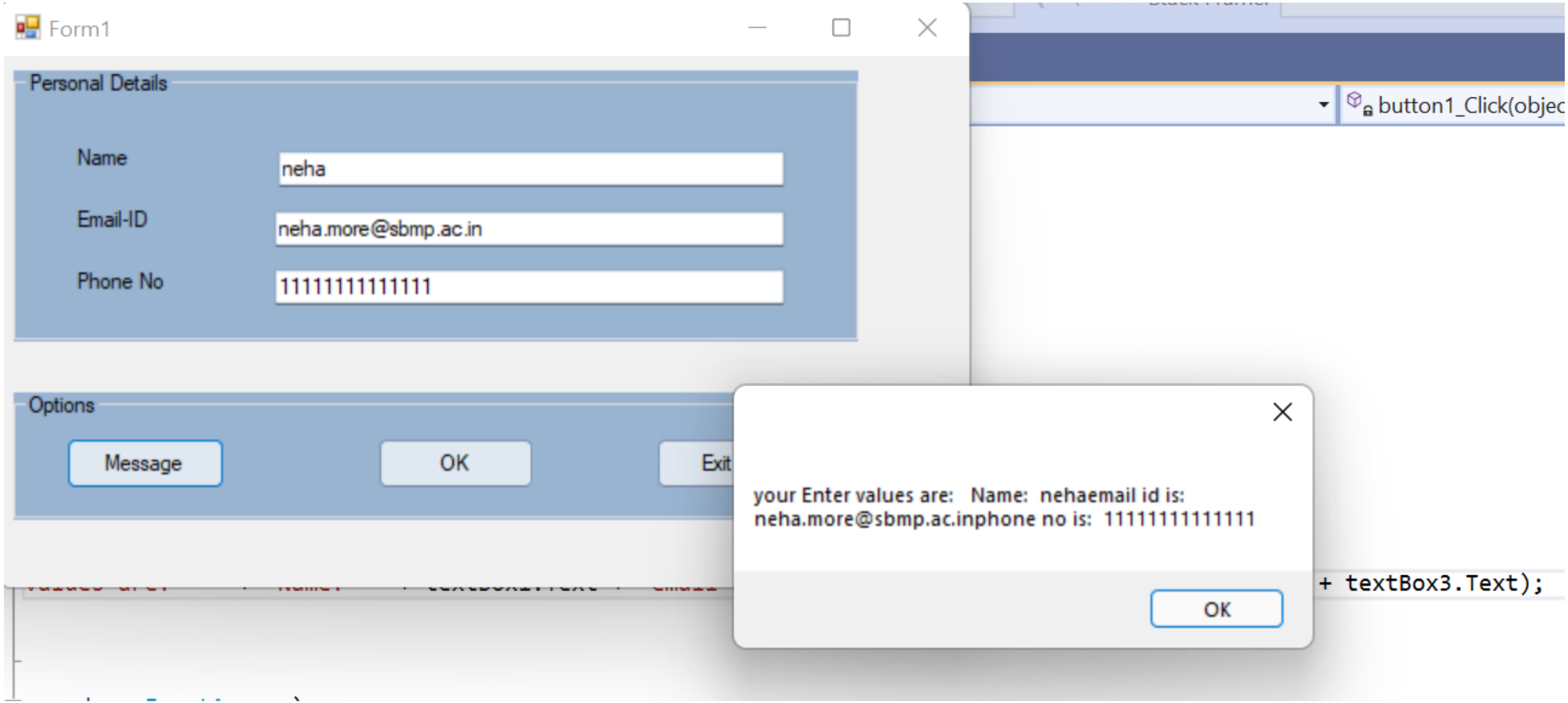
- When a control is placed on a form, the form is said to become the parent of the control, and hence the control is the child of the form. When you place a GroupBox on a form, it becomes a child of a form. As a group box can itself contain controls, it becomes the parent of these controls. The effect of this is that moving the GroupBox will move all of the controls placed on it.

Design a form which includes two group box from toolbar:



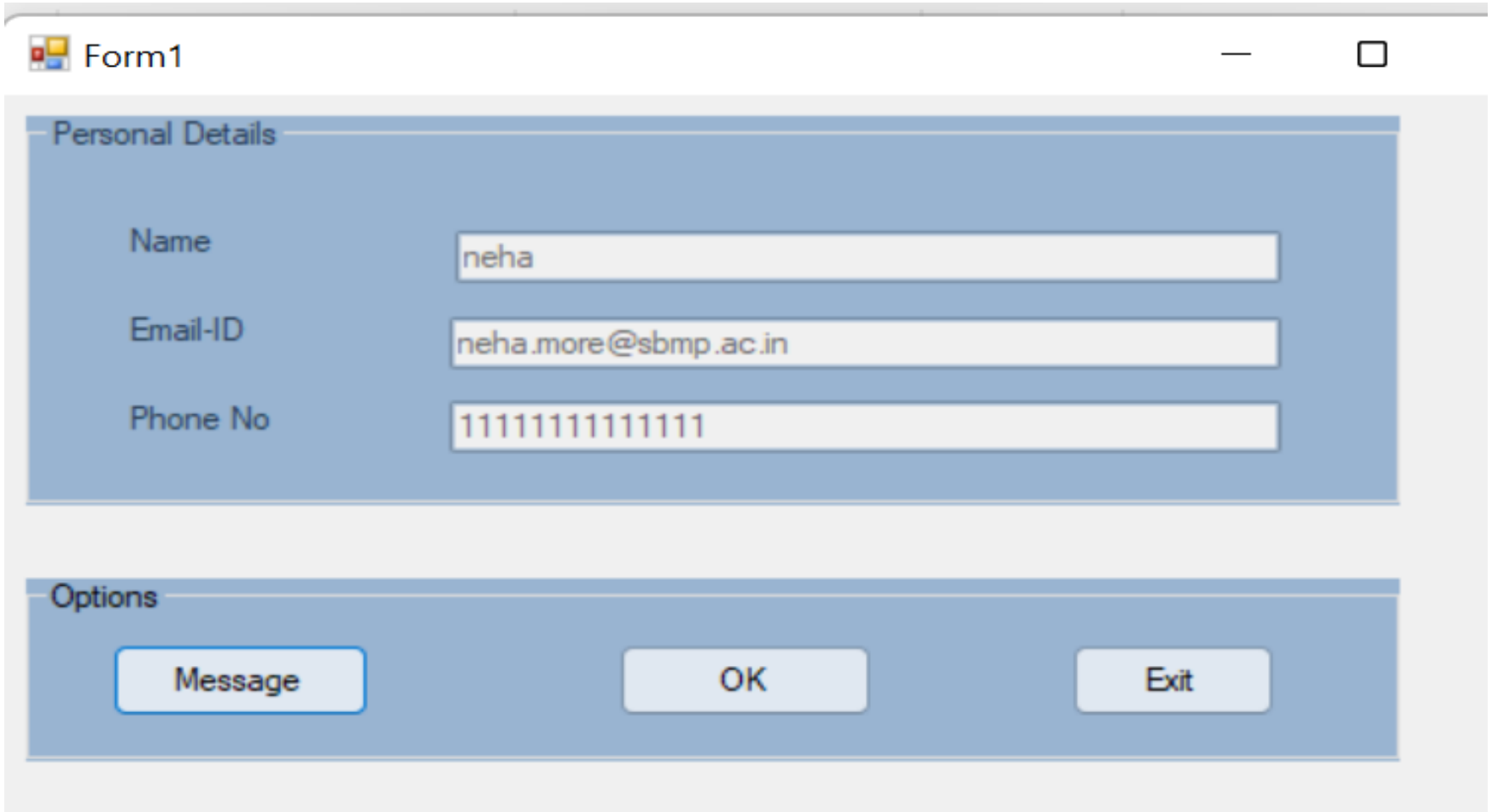
Code for Example:

```
namespace groupcontrol
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void button1_Click(object sender, EventArgs e)
        {
            MessageBox.Show("your Enter values are:    " + "Name:   " + textBox1.Text +
        "email id is: " + textBox2.Text + "phone no is:   " + textBox3.Text);
            groupBox1.Enabled = false;
        }

        private void button3_Click(object sender, EventArgs e)
        {
            Application.Exit();
        }

        private void button2_Click(object sender, EventArgs e)
        {
            groupBox1.Enabled = true;
         }
     }
}
```



Click on **OK of Message box -----Personal Details are disabled**

Click on OK button personal Details are enabled



The RichTextBox Control

- Like the normal TextBox, the RichTextBox control is derived from TextBoxBase. Because of this, it shares a number of features with the TextBox, but is much more diverse.

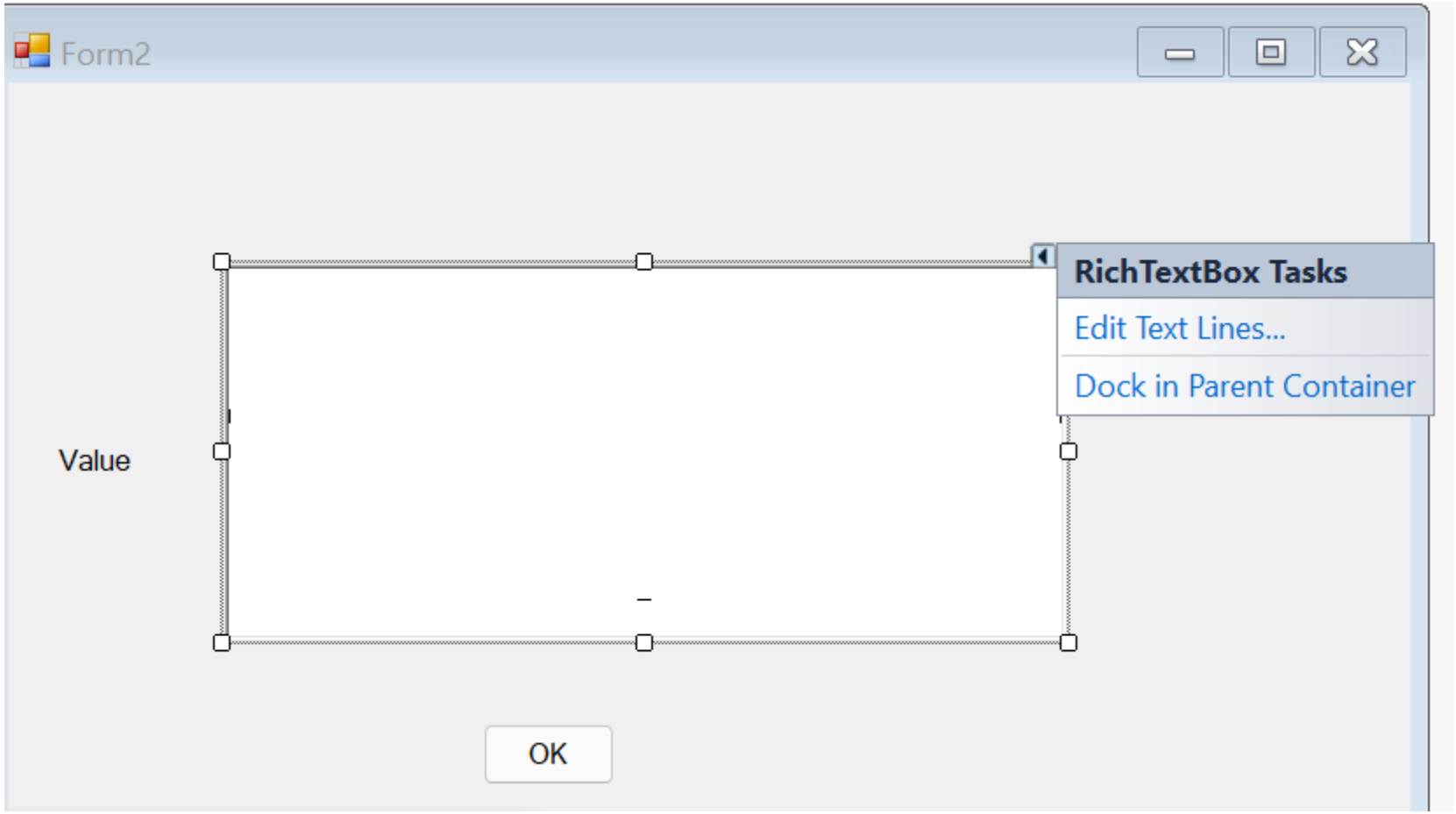
- Where a TextBox is commonly used with the purpose of obtaining short text strings from the user, the RichTextBox is used to display and enter formatted text (for example bold, underline, and *italic*).

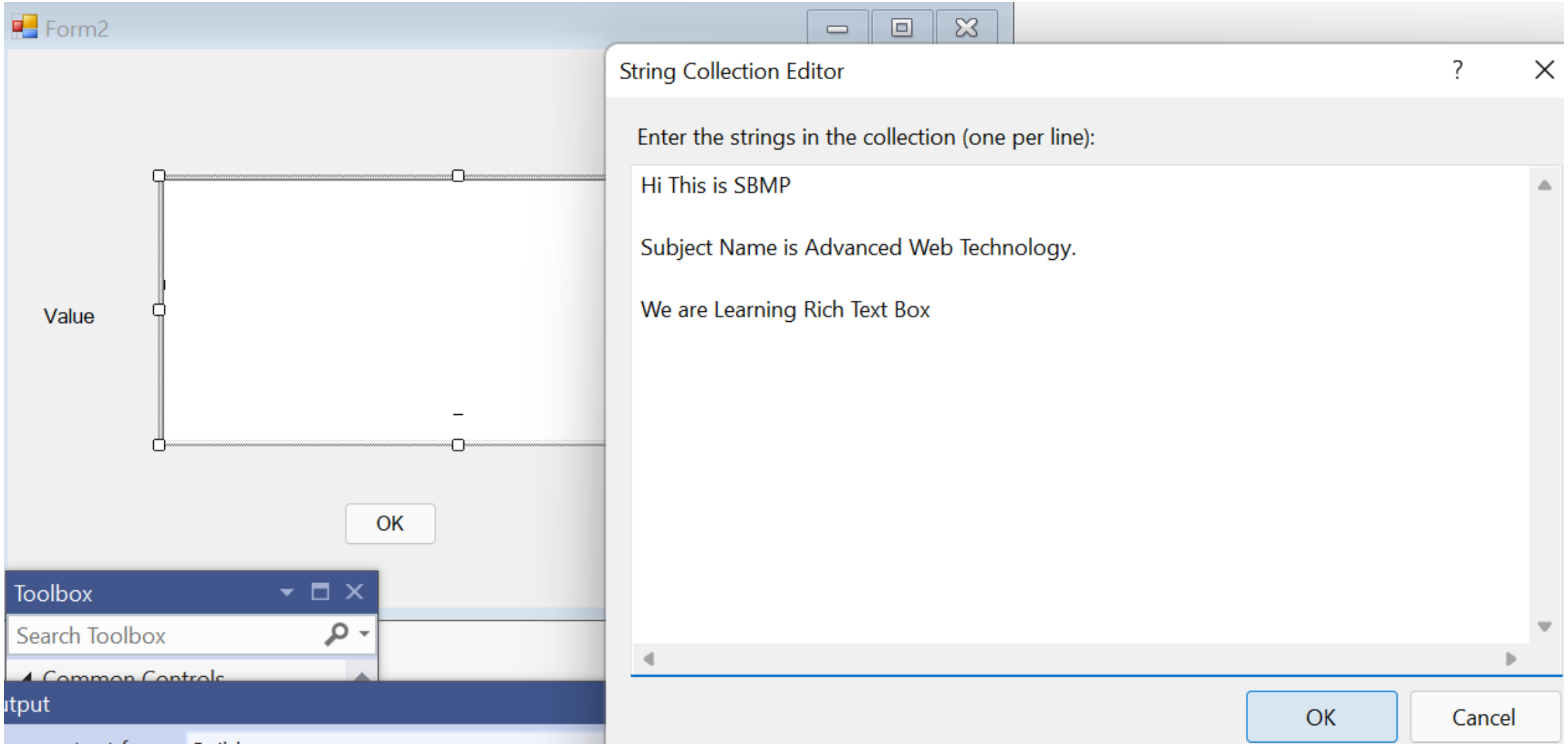
- RichTextBox control is a textbox which gives you rich text editing controls and advanced formatting features also includes a loading rich text format (RTF) files. Or in other words, RichTextBox controls allows you to display or edit flow content, including paragraphs, images, tables, etc.

- The RichTextBox class is used to represent the windows rich text box and also provide different types of properties, methods, and events. It is defined under System.Windows.Forms namespace.

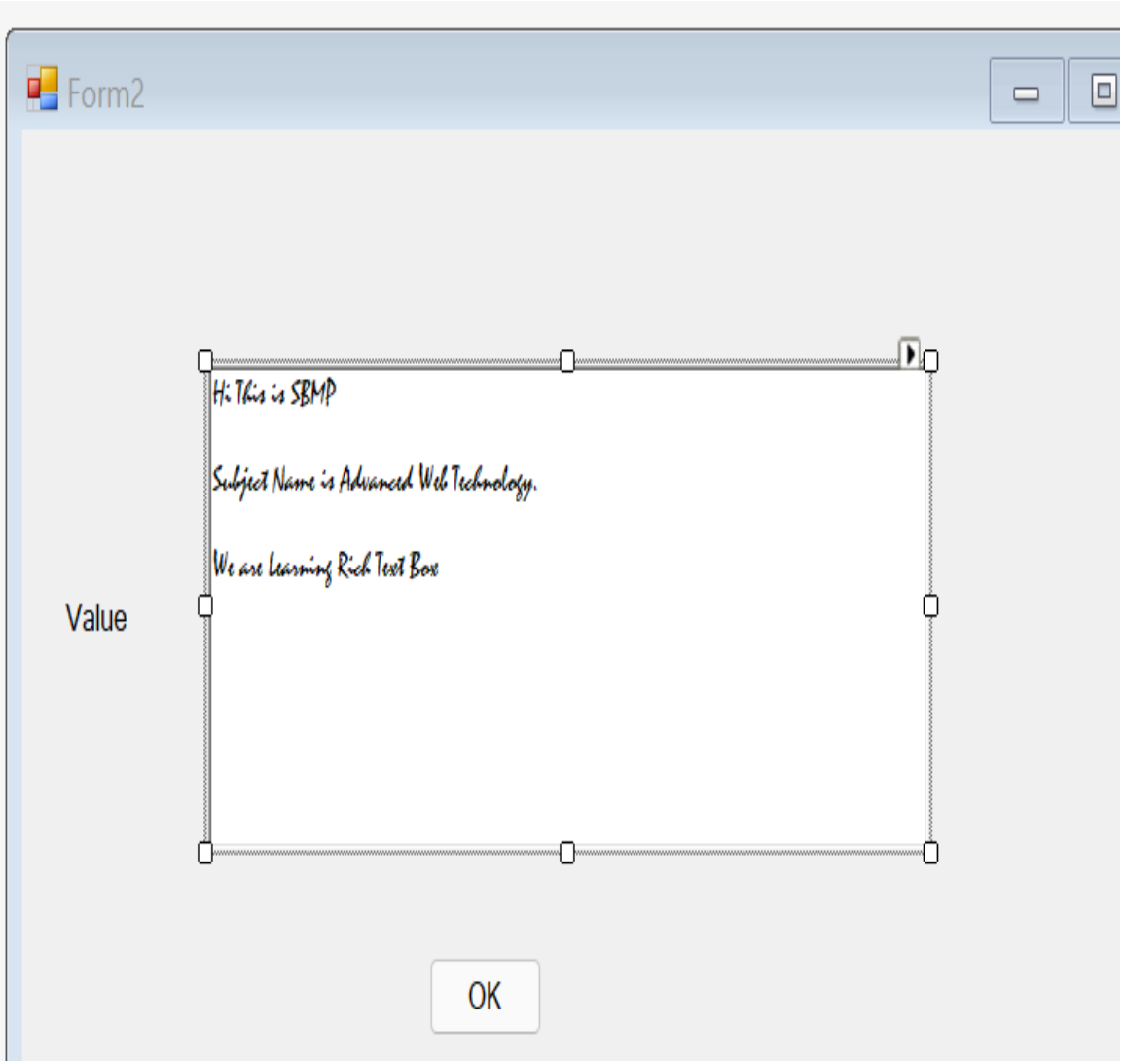
- The RichTextBox is similar to the TextBox, but it has additional formatting capabilities. Whereas the TextBox control allows the Font property to be set for the entire control, the RichTextBox allows you to set the Font, as well as other formatting properties, for selections within the text displayed in the control.

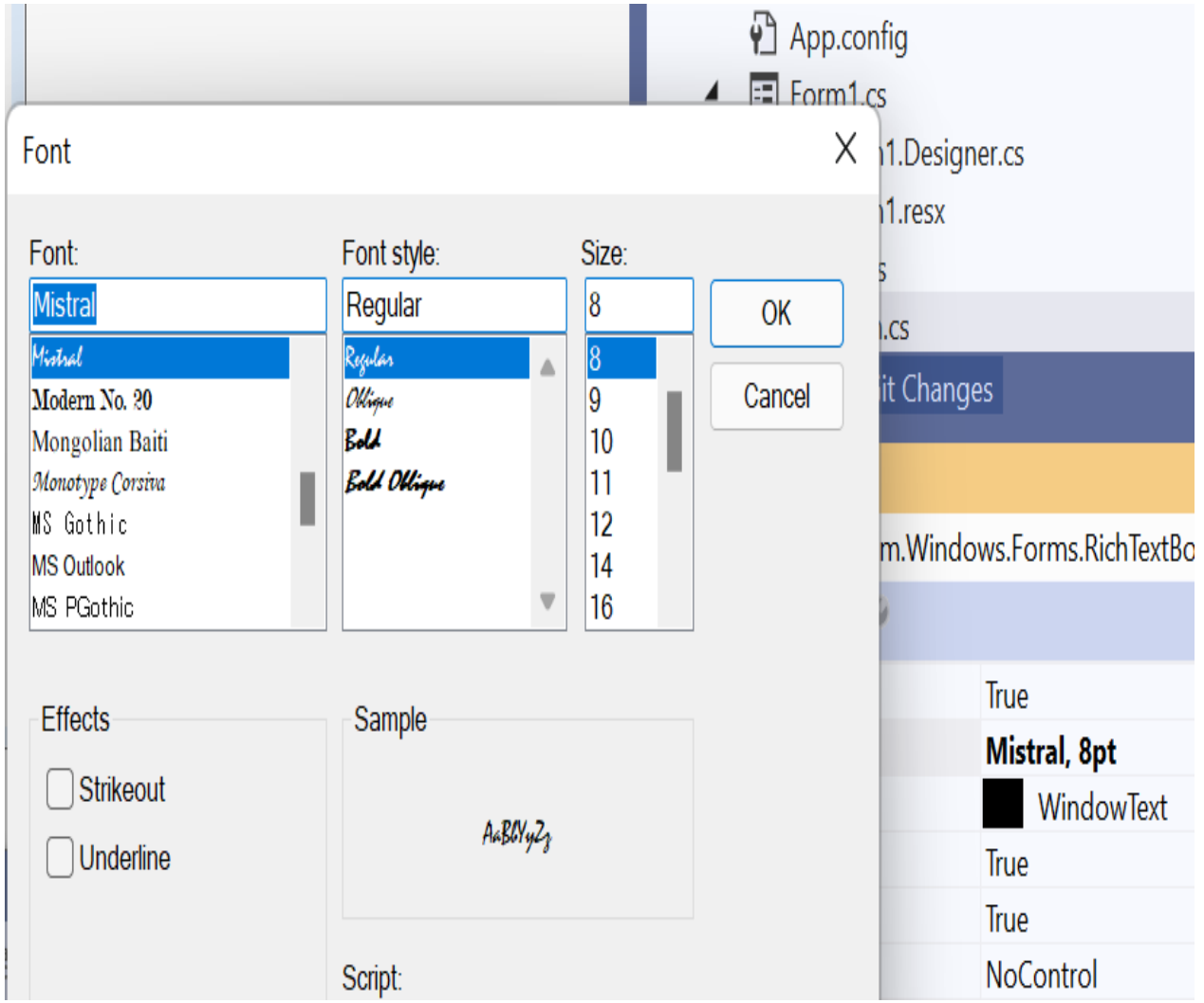
Adding rich text box on form



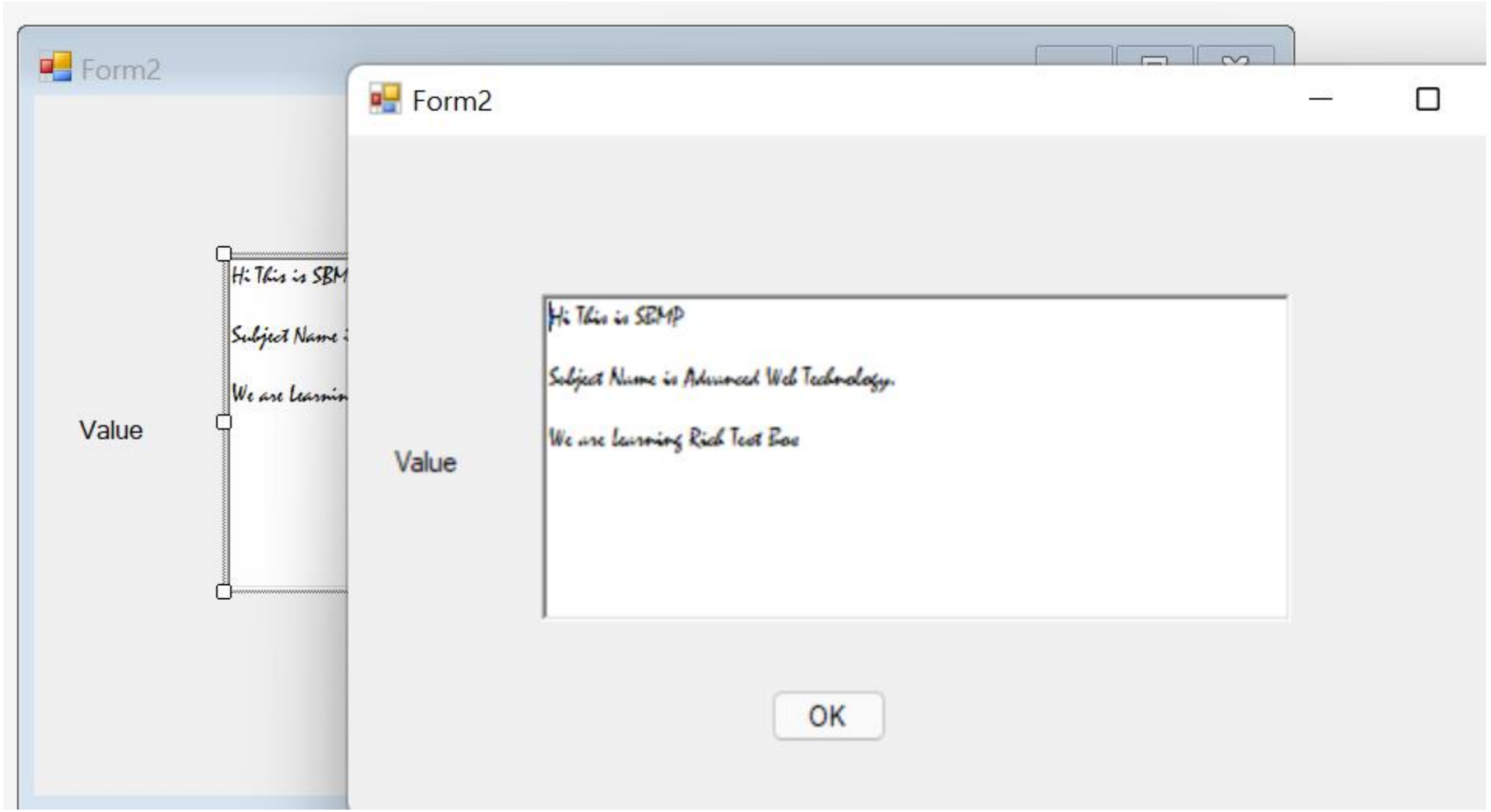


Click on “OK” all text will come inside the text box.


Form2
Hi This is SBMP
Subject Name is Advanced Web Technology.
We are Learning Rich Text Box
Value
OK


App.config
Form1.cs
1.Designer.cs
1.resx
.cs
it Changes
m.Windows.Forms.RichTextBo
Font
Font:
Mistral
Font style:
Regular
Size:
8
OK
Cancel
Mistral
Modern No. 20
Mongolian Baiti
Monotype Corsiva
MS Gothic
MS Outlook
MS PGothic
Regular
Oblique
Bold
Bold Oblique
8
9
10
11
12
14
16
Effects
Strikeout
Underline
Sample
AaBbYyZz
Script:
True
Mistral, 8pt
WindowText
True
True
NoControl

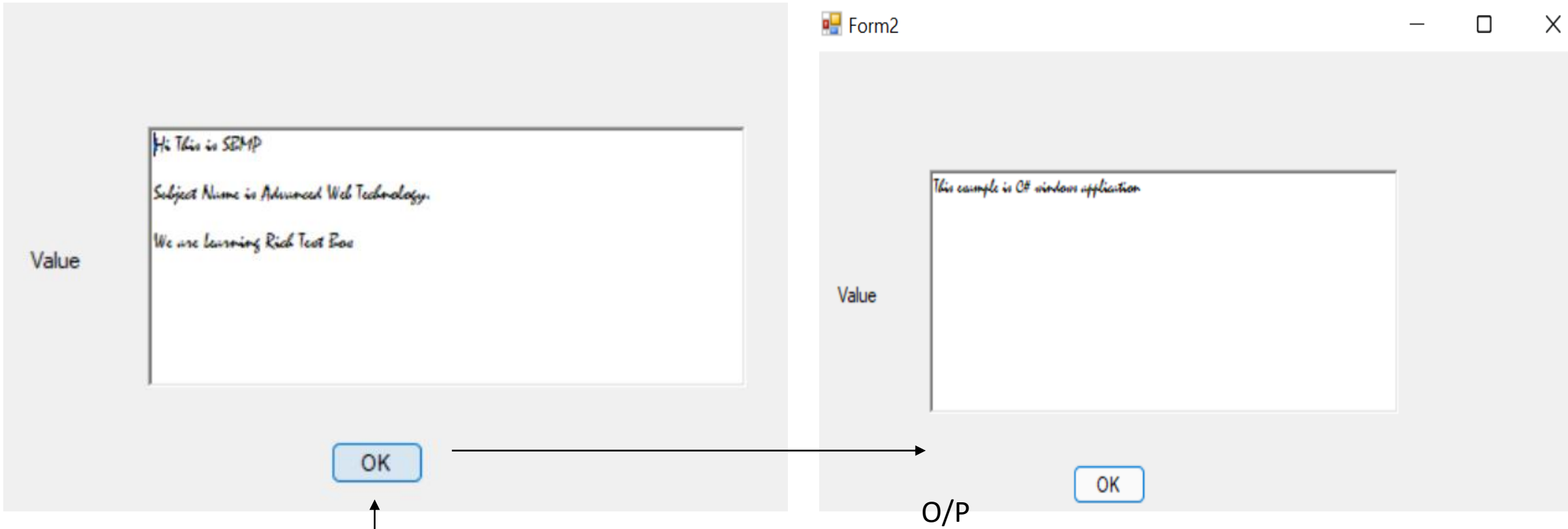
Run


Form2
Value
Hi This is SBM
Subject Name
We are Learnin
Form2
Value
Hi This is SBMP
Subject Name is Advanced Web Technology.
We are Learning Rich Text Box
OK

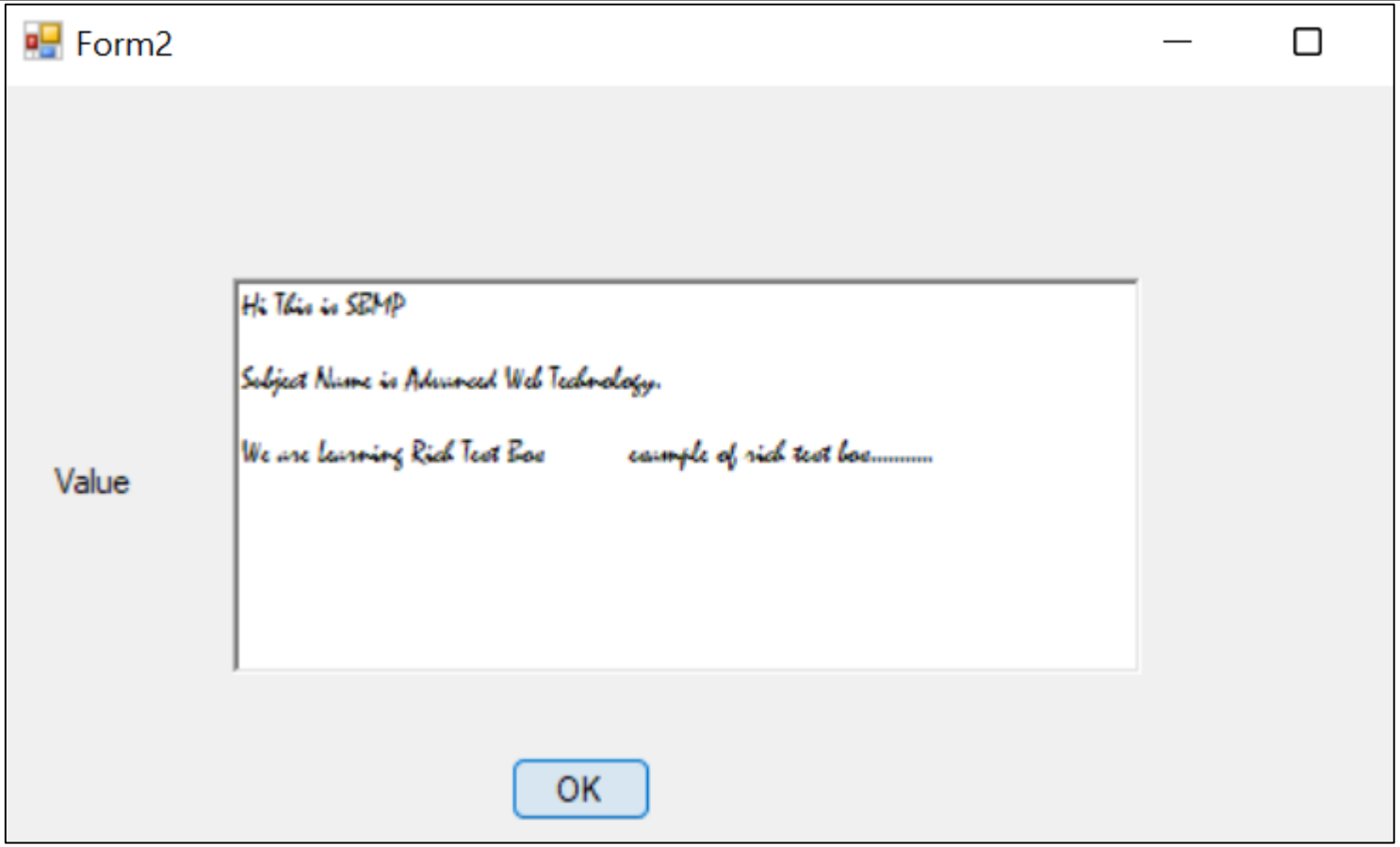
Code:

```
namespace groupcontrol
{
    public partial class Form2 : Form
    {
        public Form2()
        {
            InitializeComponent();
        }

        private void richTextBox1_TextChanged(object sender, EventArgs e)
        {

        }

        private void button1_Click(object sender, EventArgs e)
        {
            richTextBox1.Text = "This example is C# windows application";
        }
    }
}
```

Output


Hi This is SEMP
Subject Name is Advanced Web Technology.
We are Learning Rich Text Box
Value
OK
Click on OK
Form2
This example is C# window application
Value
OK
O/P

```
private void button1_Click(object sender, EventArgs e)
        {
            //richTextBox1.Text = "This example is C# windows application";
            richTextBox1.AppendText("          example of rich text box.....");
        }
```



Constructor

Constructor	Description
RichTextBox()	This Constructors is used to initialize a new instance of the RichTextBox 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.
Name	This property is used to get or set the name of the control.
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 RichTextBox 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.
ZoomFactor	This property is used to get or set the current zoom level of the RichTextBox.
ShowSelectionMargin	This property is used to get or set a value indicating whether a selection margin is displayed in the RichTextBox.
SelectedText	This property is used to get or set the selected text within the RichTextBox.
ScrollBars	This property is used to get or set the type of scroll bars to display in the RichTextBox control.
Multiline	This property is used to get or set a value indicating whether this is a multiline RichTextBox control.

USER CONTROL IN C#

- C# user control is defined as an implementation in programming language of C# to provide an empty control and this control can be leveraged to create other controls.

- This implementation provides additional flexibility to re-use controls in a large-scale project.

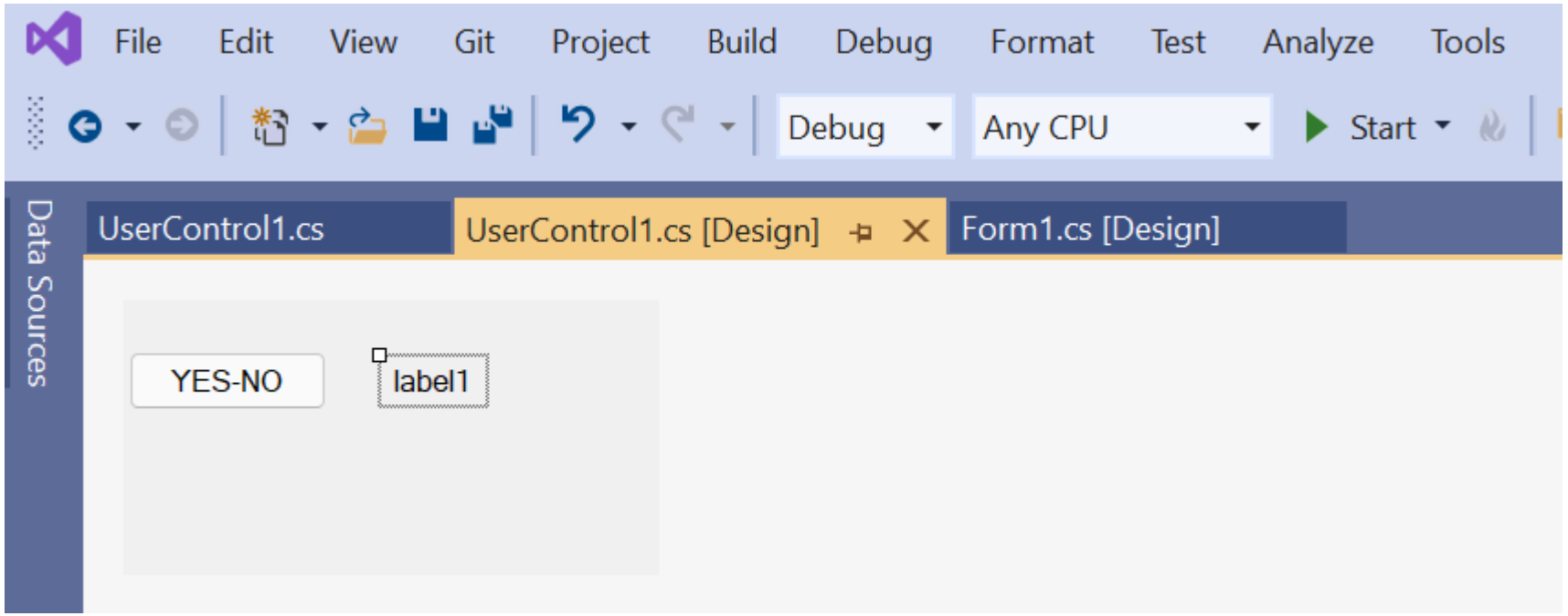
- Not only it is about re-usage of controls, but it also helps users to find and pinpoint a bug and then making it easier to resolve the bug in a shorter time.

- Through user control, one can make any changes in the code at just a single place and the effect will be seen in every web form or the form in the windows application that is attached to the user control.

- Extra effort and time are the very big benefits using the technique of user control.

Adding User Control:

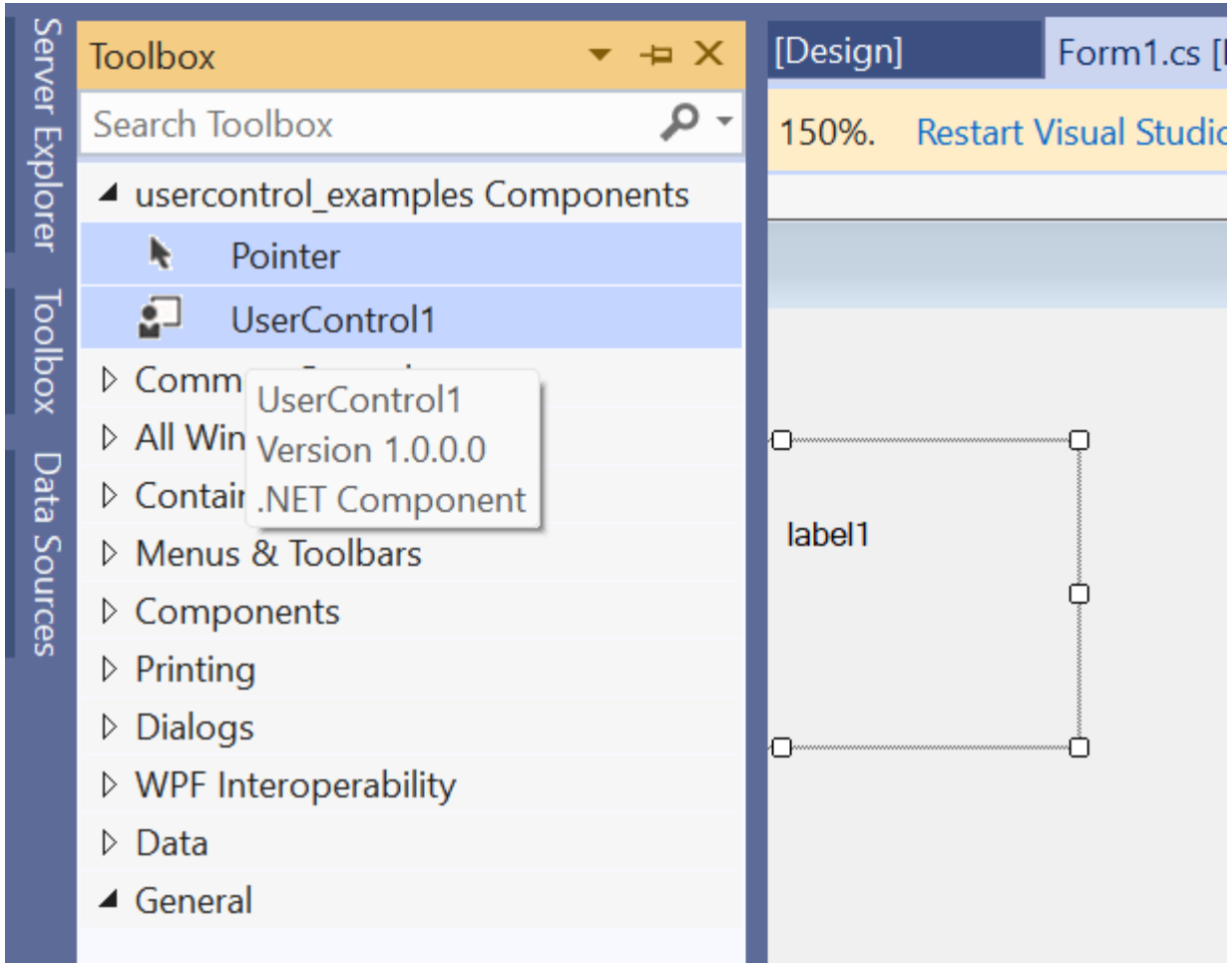
- Right click on Solution Explorer
- Add item→usercontrol



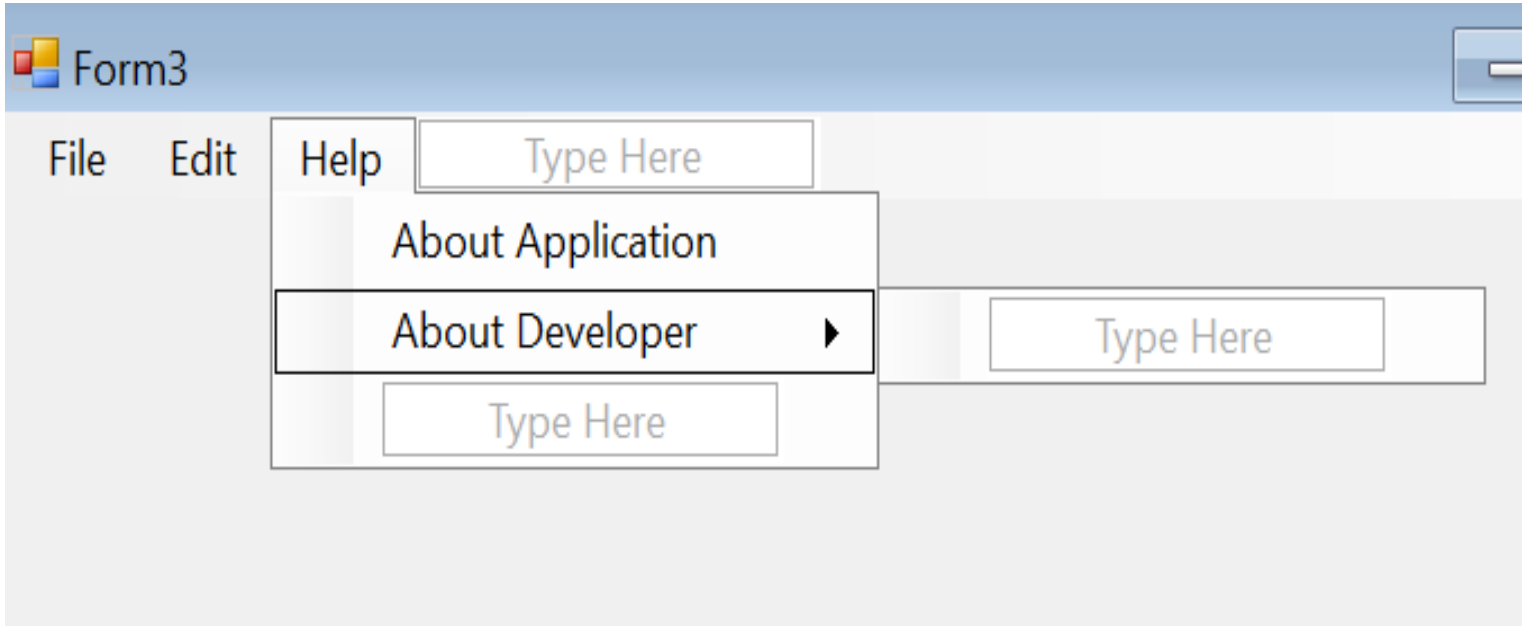
Usercontrol1.cs

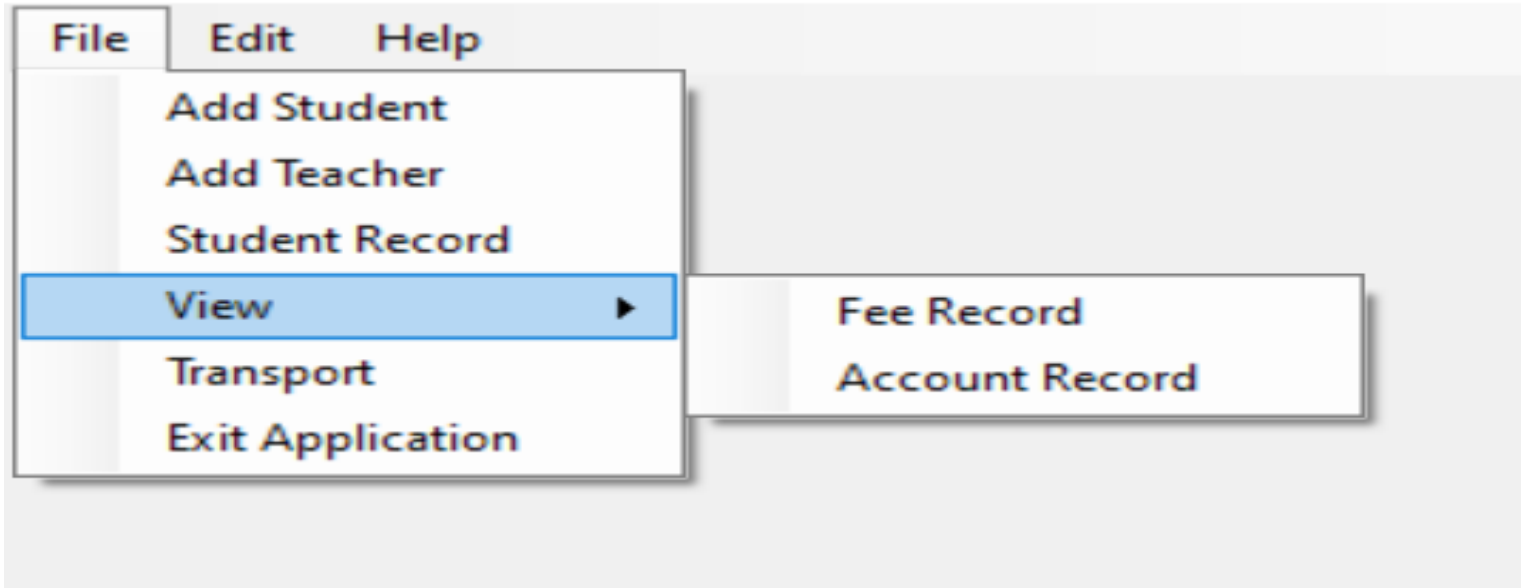
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace usercontrol_examples
{
    public partial class UserControl1 : UserControl
    {
        public UserControl1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            if (label1.Text == string.Empty || label1.Text == "No")
            {
                label1.Text = "yes";

            }
            else
            {
                label1.Text = "No";
            }
        }
    }
}
```

- Click on toolbox→UserControl1



Menustrip in c# windows application





Code for Menu:

```
namespace groupcontrol
{
    public partial class Form3 : Form
    {
        public Form3()
        {
            InitializeComponent();
        }
        private void aboutApplicationToolStripMenuItem_Click(object sender, EventArgs e)
        {
            MessageBox.Show("All rights reserved", "About application", MessageBoxButtons.OK,
MessageBoxIcon.Information);

        }

        private void aboutDeveloperToolStripMenuItem_Click(object sender, EventArgs e)
        {
            MessageBox.Show("This application is deve;oped by ABC", "About Developer",
MessageBoxButtons.OKCancel, MessageBoxIcon.Information);
        }
    }
}
```

MessageBox parameter

```
private void aboutApplicationToolStripMenuItem_Click(object sender, EventArgs e)
{
    MessageBox.Show("All rights reserved", "About application", MessageBoxButtons., MessageBoxIcon.In

}

private void aboutDeveloperToolStripMenuItem_Click(object sender, EventArgs e)
{
    MessageBox.Show("This application is deve;oped by ABC", "About Developer", Mes
```

- ★ OK
- ★ YesNo
- ★ YesNoCancel
- ★ OKCancel

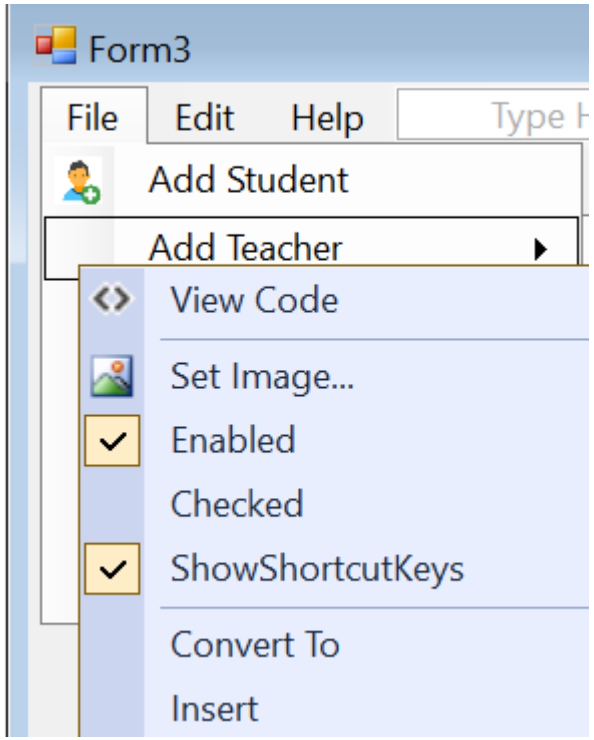
```
private void aboutApplicationToolStripMenuItem_Click(object sender, EventArgs e)
{
    MessageBox.Show("All rights reserved", "About application", MessageBoxButtons.OK, MessageBoxIcon.);

}

private void aboutDeveloperToolStr
{
    MessageBox.Show("This application is deve;oped by ABC", "About Developer", MessageBoxButtons.OKCa
}
```

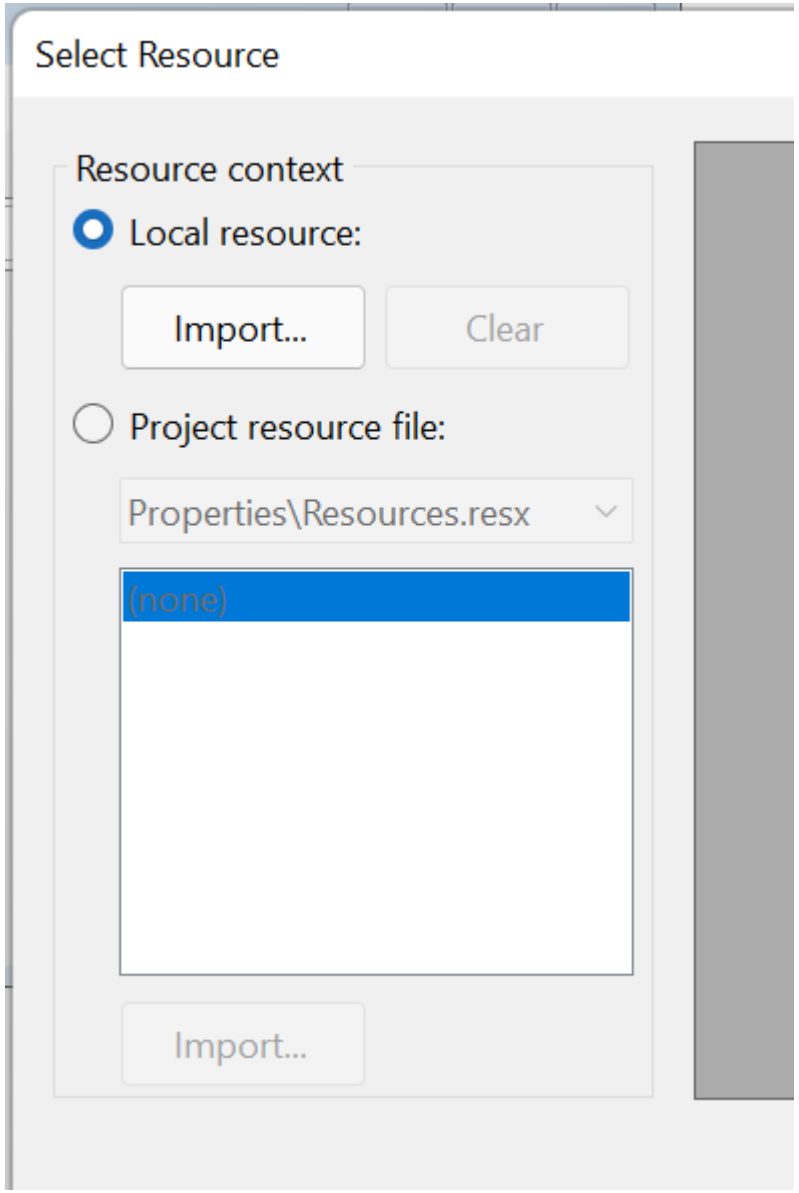
MessageBoxIcon.Error = 16
The message box contains a symbol consisting of white X in a circle with a red background.
★ IntelliCode suggestion based on this context

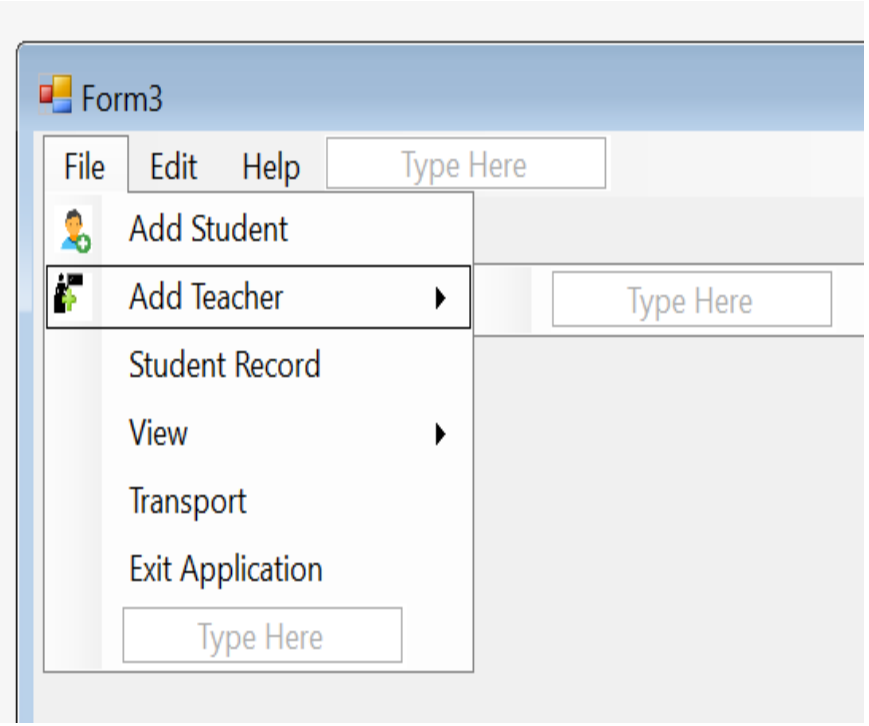
- ★ Error
- ★ Information
- ★ Warning
- ★ Exclamation
- ★ Question

Syntax: MessageBox.Show(message, title, buttons._____, MessageBoxIcon._____);

Adding Icon


Form3
File
Edit
Help
Add Student
Add Teacher
View Code
Set Image...
Enabled
Checked
ShowShortcutKeys
Convert To
Insert


Select Resource
Resource context
Local resource:
Import...
Clear
Project resource file:
Properties\Resources.resx
(none)
Import...

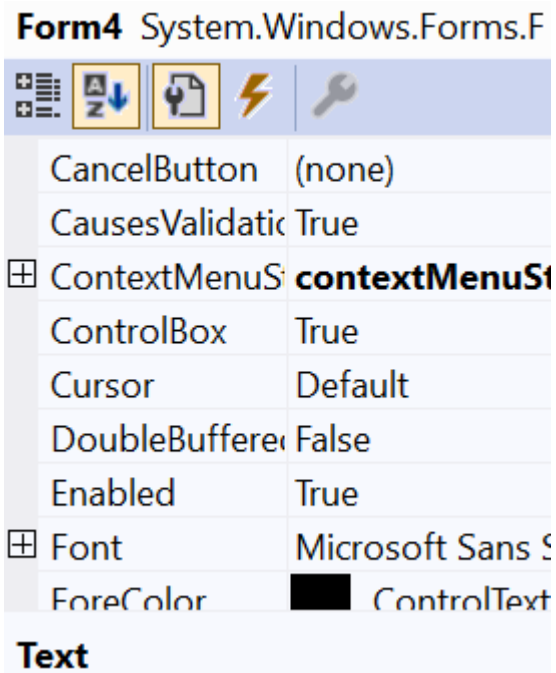

Form3
File
Edit
Help
Type Here
Add Student
Add Teacher
Student Record
View
Transport
Exit Application
Type Here
Type Here

Context Menustrip

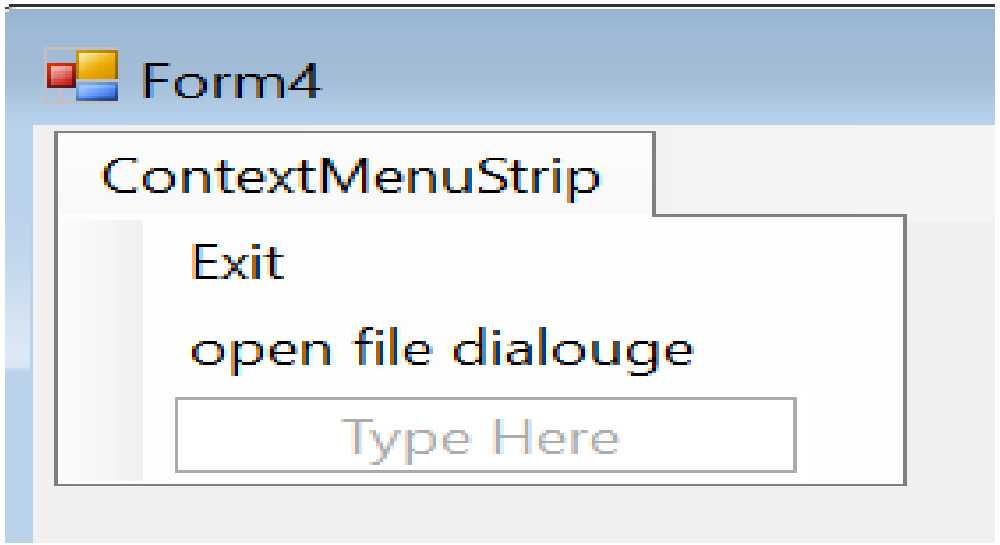
The Menu which gets displayed while user right clicks on the application is what called '**Context Menu**'. We can add the Context Menu to our form the same way we added the MenuStrip control. But this time we should pick the "ContextMenuStrip Control" from the toolbox and drop it to our form.

The important step is assigning the context Menu to the form and you can do that by using the contextMenu property of the windows form.

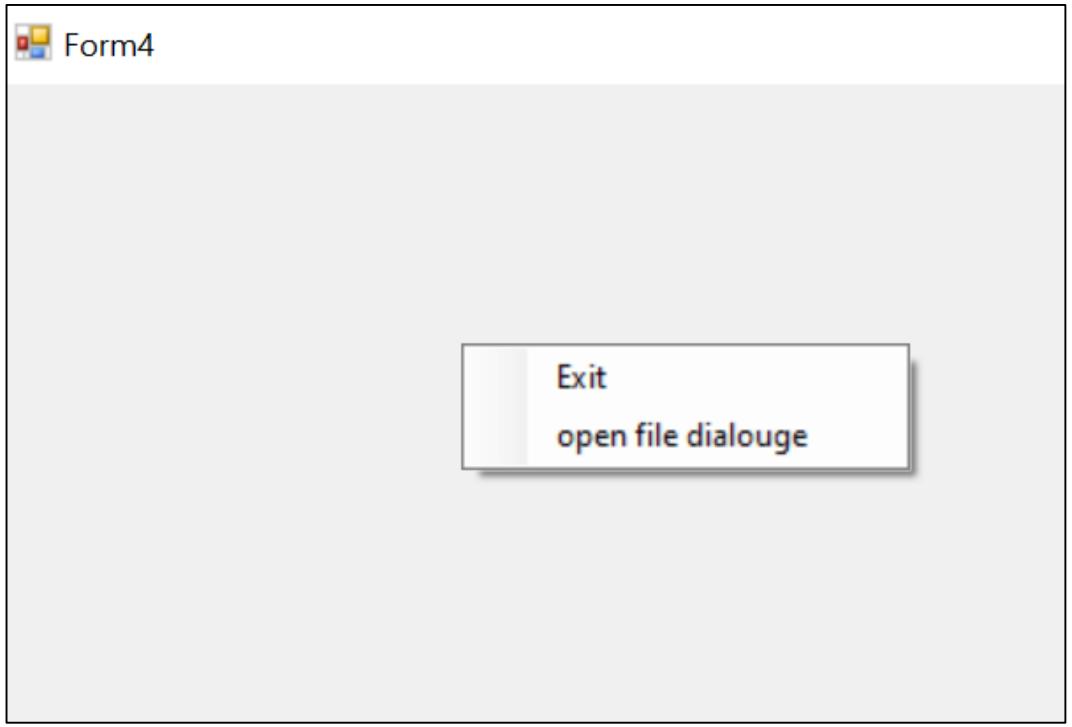
Select Form and then go to properties



Example:



O/P→right click on form

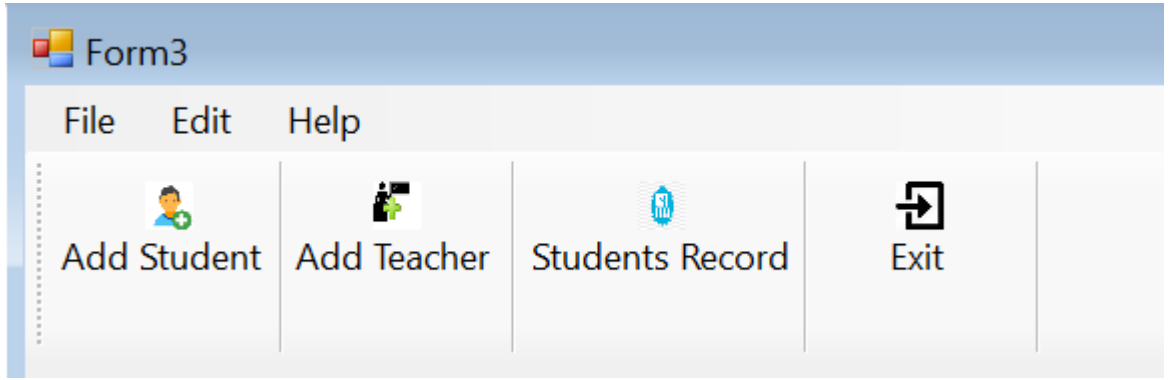


Coding for context Menuscript control:

```
namespace groupcontrol
{
    public partial class Form4 : Form
    {
        public Form4()
        {
            InitializeComponent();
            //this.ContextMenuStrip = contextMenuStrip2;
        }
        private void exitToolStripMenuItem_Click(object sender, EventArgs e)
        {
            //Application.Exit();
            this.Close();
        }

        private void openFileDialougeToolStripMenuItem_Click(object sender, EventArgs e)
        {
            OpenFileDialog dialog = new OpenFileDialog();
            dialog.ShowDialog();
        }
    }
}
```

ToolStrip in C#

ToolStrip control provides functionality of Windows toolbar controls in Visual Studio. ToolStrip class represents a ToolStrip control in Windows Forms and serves as a base class of MenuStrip, StatusStrip, and ContextMenuStrip classes.



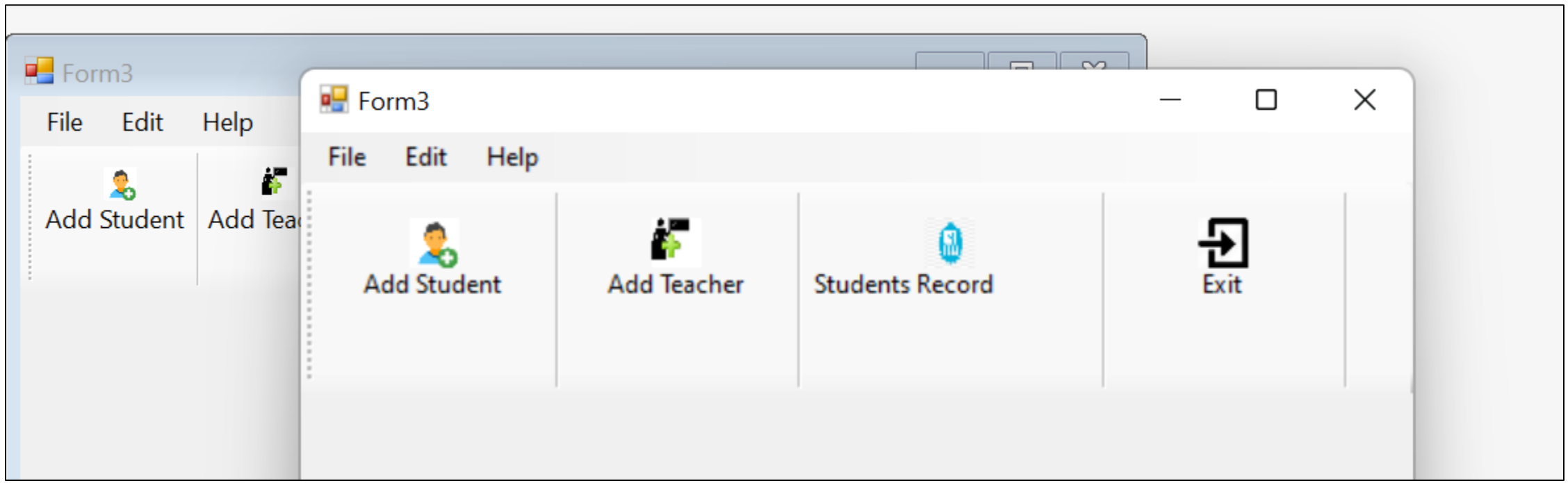
Drag and drop toolstrip from toolbox

Adding an Image-→Right click on created toolbar→Set Image
(can set size—height and width from properties)

Give Name to tool from properties-→we can assign name and image both by selecting tool-→right click-→Display style→Select image/None/text/image and text both

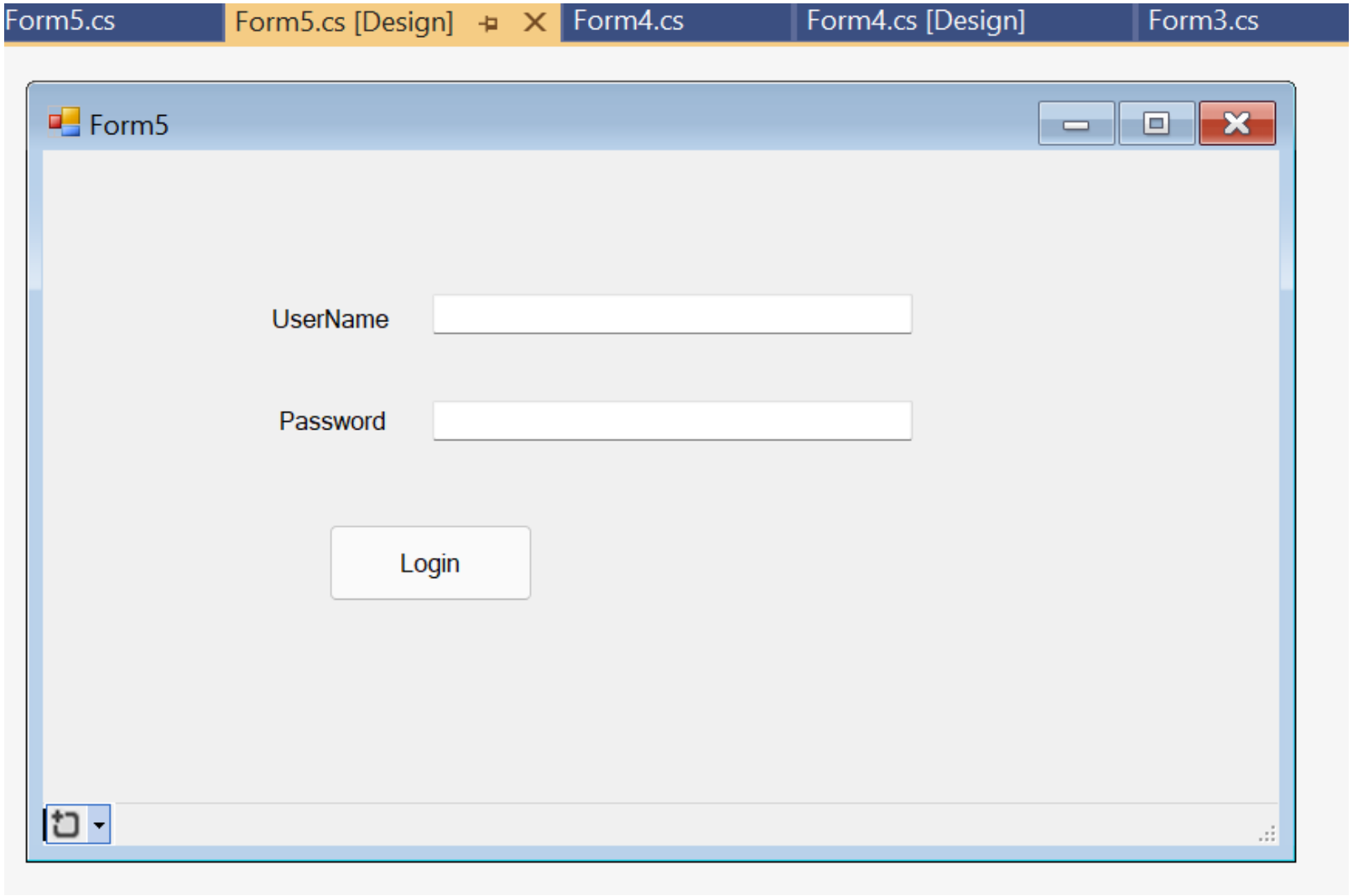
Selected text and image both then from properties select TextImage Relation

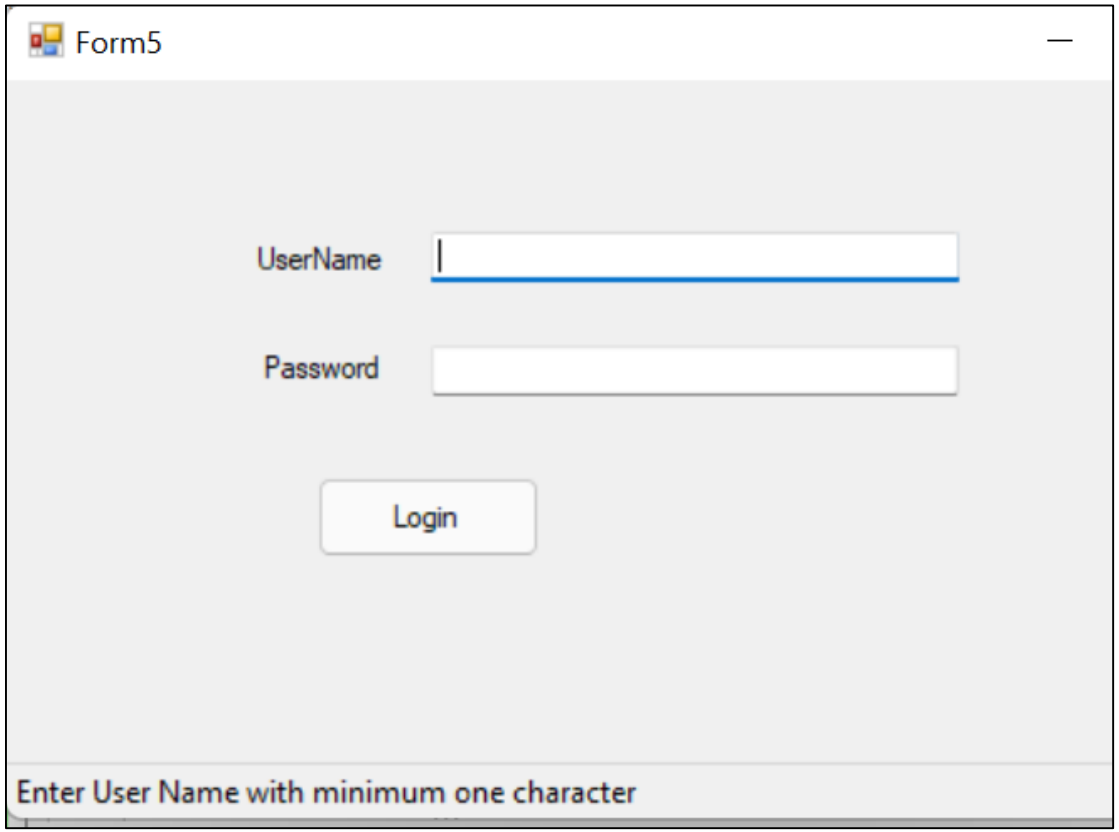
OUTPUT:

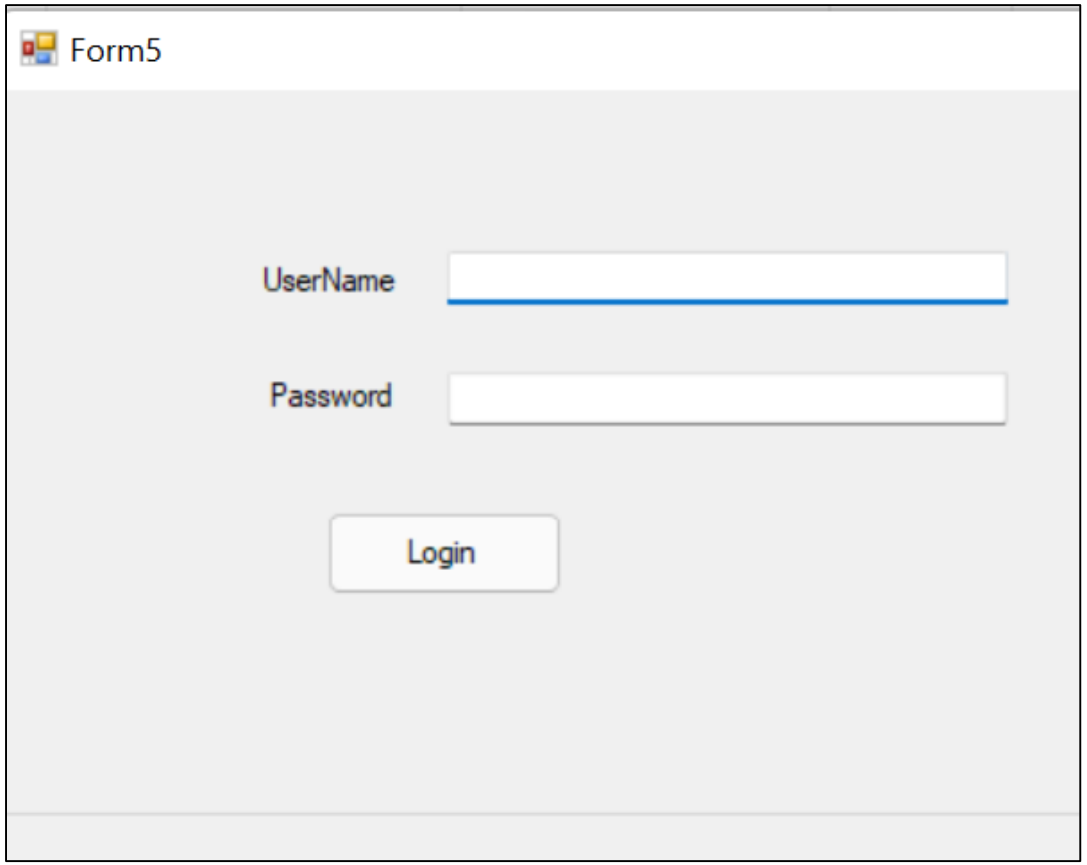

Form3
File
Edit
Help
Add Student
Add Tea
Form3
File
Edit
Help
Add Student
Add Teacher
Students Record
Exit

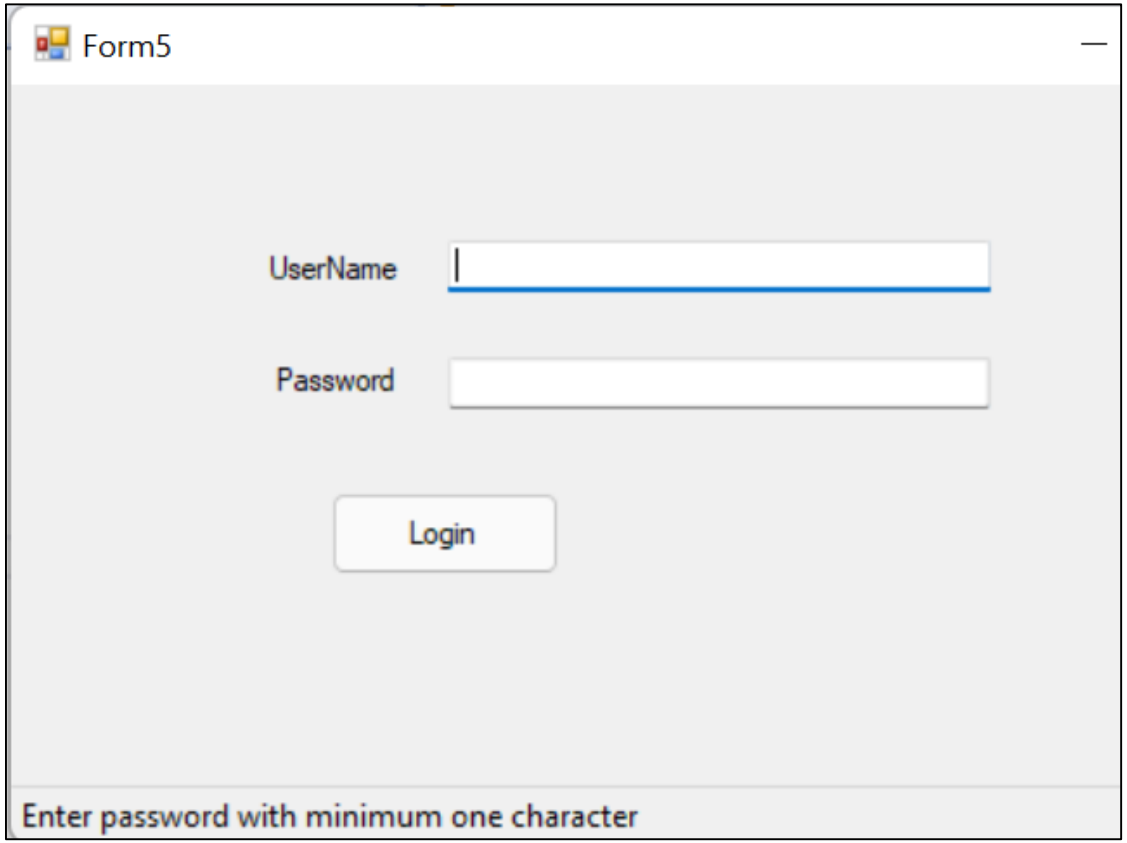
Status bar in c#

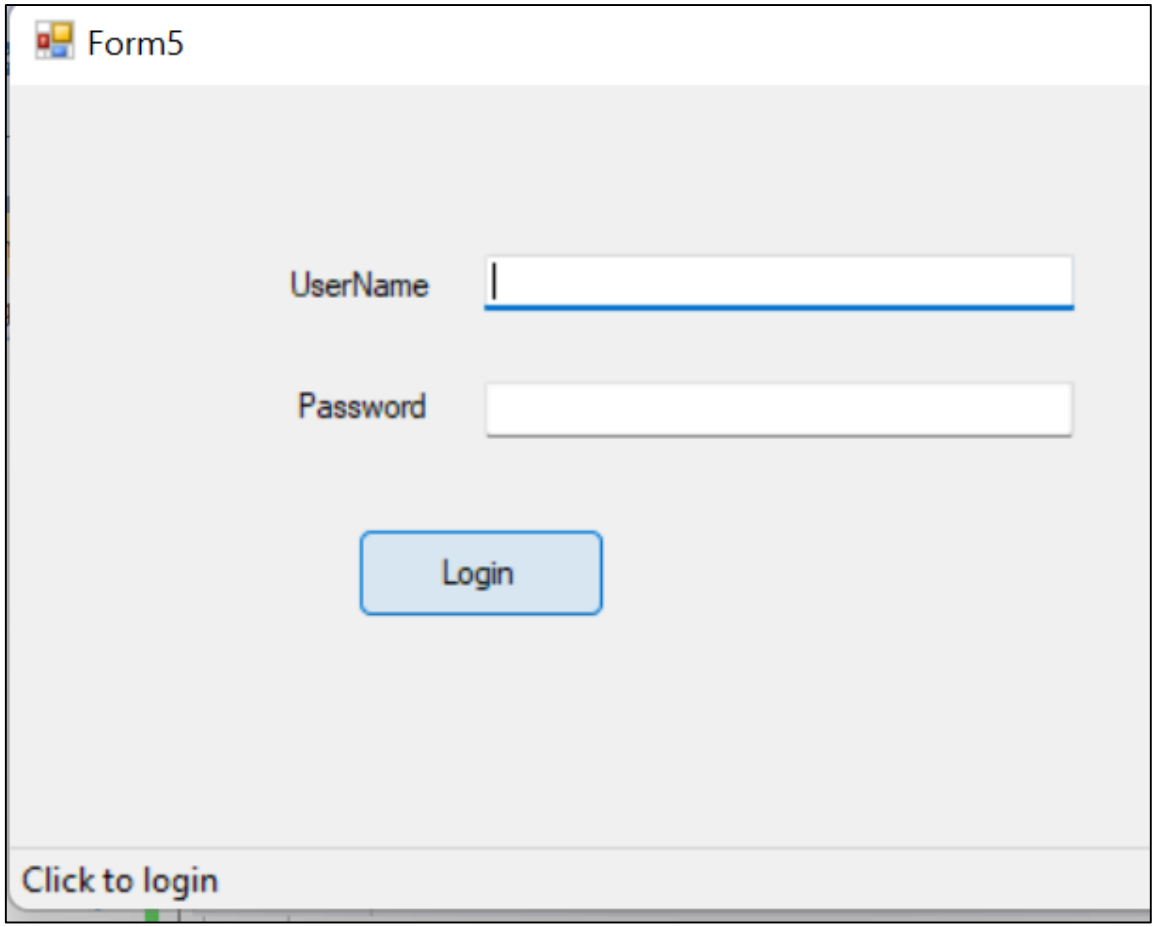
status bar control, placed on the bottom of a Form is used to display some text that represents a status of the application and user actions.




Form5
UserName
Password
Login
Enter User Name with minimum one character


Form5
UserName
Password
Login


Form5
UserName
Password
Login
Enter password with minimum one character


Form5
UserName
Password
Login
Click to login

```
namespace groupcontrol
{
    public partial class Form5 : Form
    {
        public Form5()
        {
            InitializeComponent();
        }

        private void label1_Click(object sender, EventArgs e)
        {

        }

        private void button1_Click(object sender, EventArgs e)
        {

        }
```

```
private void textBox1_MouseHover(object sender, EventArgs e)
        {
            toolStripStatusLabel2.Text = "Enter User Name with minimum one character";
        }

        private void textBox1_MouseLeave(object sender, EventArgs e)
        {
            toolStripStatusLabel2.Text = "";
        }

        private void textBox2_MouseHover(object sender, EventArgs e)
        {
            toolStripStatusLabel2.Text = "Enter password with minimum one character";
        }
        private void button1_MouseHover(object sender, EventArgs e)
        {
            toolStripStatusLabel2.Text = "Click to login";
        }
        private void button1_MouseLeave(object sender, EventArgs e)
        {
            toolStripStatusLabel2.Text = "";
        }
    }
}
```

Graphics in C#:

Drawing in C# is achieved using the Graphics Object. The Graphics Object takes much of the pain out of graphics drawing by abstracting away all the problems of dealing with different display devices and screens resolutions

Persistent Graphics

- An important point is that it simply creating a Graphics Object for a component and then drawing on that component, does not create persistent graphics. In fact what will happen is that as soon as the window is minimized or obscured by another window the graphics will be erased.
- For this reason, steps need to be taken to ensure that any graphics are persistent. Two mechanisms are available for achieving this. One is to repeatedly perform the drawing in the Paint() event handler of the control (which is triggered whenever the component needs to be redrawn), or to perform the drawing on a bitmap image in memory and then transfer that image to the component whenever the Paint() event is triggered.

What Are the Built-In Classes Used for Drawing Graphics?

Class	Description
Graphics	•The Graphics class allows you to draw shapes and lines onto the canvas. It includes methods such as:DrawLine(Pen, Point1, Point2) •DrawRectangle(x, y, width, height) •DrawPolygon(Pen, PointF[])
Pen	•The Pen class allows you to specify the properties of a 'pen' tip which you can use to draw your shapes. You can specify properties such as color, thickness, or dash style. Methods include:SetLineCap(LineCap, LineCap, DashCap)
Color	A color object made up of R (red), G (green), and B (blue) values. You will need a color object for many of the built-in methods that create shapes.
SolidBrush, HatchBrush, TextureBrush	These brush classes derive from the "Brush" interface. These classes allow you to color in blank spaces on the canvas. You can also choose to fill the spaces using different patterns or textures. You can specify properties such as the color.
Rectangle, Line, Polygon, Ellipse	You can create objects based on these shapes, and use them when calling methods such as DrawRectangle(). Instead of passing the x, y, width, and height as arguments, you can choose to pass an existing Rectangle object instead.

Add a Paint on Form Load Event Handler

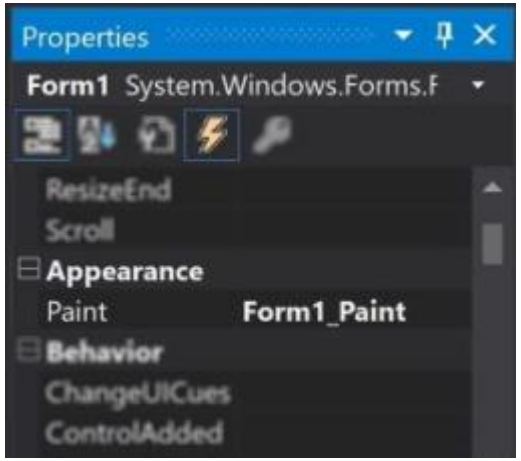
1. Add a Paint function for the form.

```
private void Form1_Paint(object sender, PaintEventArgs e)
{
// Code goes here
}
```

2. Go into the Design View Tab.

3.In the Properties window, select the lightning icon to open the "Events" tab.

4.In "Paint", under "Appearance", select the Form1_Paint function. This will execute the function when you run the application.



Draw Lines

Inside the Form1_Paint() function, create a Color object with the color you want the line to be. Then, create a Pen object to draw the line with.

```
Color black = Color.FromArgb(255, 0, 0, 0); ----------[(alpha, red, green, and blue) ]
Pen blackPen = new Pen(black);
```


[The alpha value **indicates the transparency of the color]**

The DrawLine() method from the Graphics class will draw a line using the pen. This will start drawing a line from an x, y position to another x, y position.

```
e.Graphics.DrawLine(blackPen, 300, 200, 800, 200);
```

You can modify the properties for the pen object to change its width, dash style, and start or end cap.

```
blackPen.Width = 20;
blackPen.DashStyle = System.Drawing.Drawing2D.DashStyle.Dash;
blackPen.StartCap = System.Drawing.Drawing2D.LineCap.ArrowAnchor;
e.Graphics.DrawLine(blackPen, 300, 200, 800, 200);
```

Rectangles and Circles

```
Color red = Color.FromArgb(255, 255, 0, 0);
Pen redPen = new Pen(red);
redPen.Width = 5;
e.Graphics.DrawRectangle(redPen, 100, 100, 500, 200);
```

The arguments are the x and y coordinates for the top-left of the rectangle, along with its width and height.

You can also create a rectangle using the Rectangle Class. First, create a Rectangle object. The arguments are also the x and y coordinates for the top-left corner, width, and height.

```
Rectangle rectangle = new Rectangle(100, 350, 500, 200);
```

Use the DrawRectangle() function to draw the rectangle.

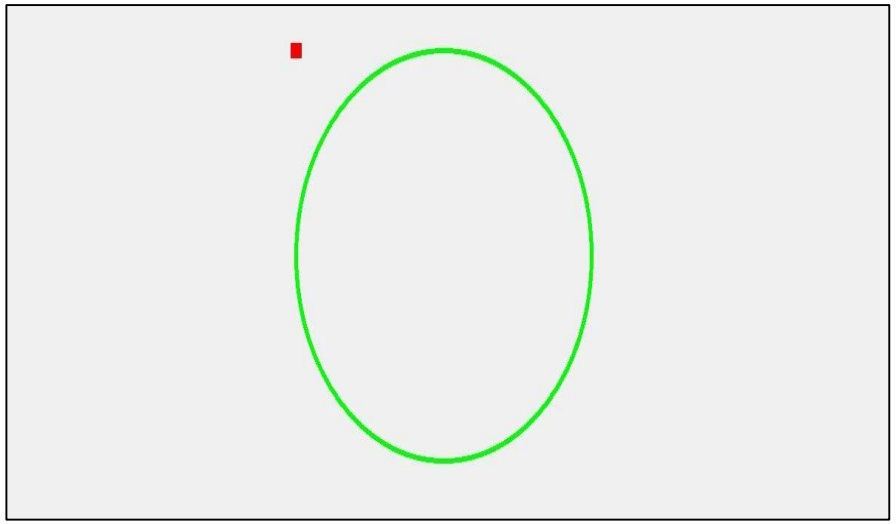
```
e.Graphics.DrawRectangle(redPen, rectangle);
```

x
The x-coordinate of the upper-left corner of the rectangle to draw.
y
The y-coordinate of the upper-left corner of the rectangle to draw.
width
Width of the rectangle to draw.
height
Height of the rectangle to draw.

Use the DrawEllipse() function to draw a circle

```
Color green = Color.FromArgb(255, 0, 255, 0);
Pen greenPen = new Pen(green);
greenPen.Width = 5;
e.Graphics.DrawEllipse(greenPen, 400, 150, 400, 400)
```

When you draw a circle, the x and y coordinates (x=400, y=150) refer to the top-left corner of the circle, not the center of the circle.



x
The x-coordinate of the upper-left corner of the bounding rectangle that defines the ellipse.

y
The y-coordinate of the upper-left corner of the bounding rectangle that defines the ellipse.

width
Int32
Width of the bounding rectangle that defines the ellipse.

height
Int32
Height of the bounding rectangle that defines the ellipse.

To draw other shapes such as triangles or hexagons, use the DrawPolygon() method. Here you can specify a list of coordinates to represent the points of the shape.

```
Color blue = Color.FromArgb(255, 0, 0, 255);
Pen bluePen = new Pen(blue);
bluePen.Width = 5;
PointF[] coordinatesForTriangle = new PointF[] {
new PointF(400, 150),
new PointF(300, 300),
new PointF(500, 300)
};
e.Graphics.DrawPolygon(bluePen, coordinatesForTriangle);
```

How to Use the Brush Class to Fill In Shapes With Color

You can use the FillRectangle(), FillEllipses() or FillTriangle() methods to create shapes with a solid color.

First, create a brush object

```
Color purple = Color.FromArgb(255, 128, 0, 0);
SolidBrush solidBrush = new SolidBrush(purple);
```

Use the FillRectangle(), FillEllipses() or FillTriangle() methods. They work the same way as the draw functions above, except instead of a Pen, they use a Brush object.

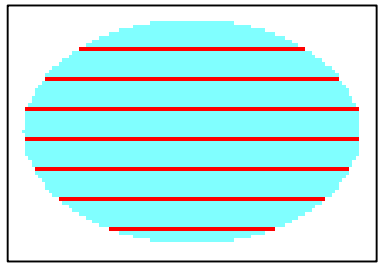
```
e.Graphics.FillRectangle(solidBrush, 50, 50, 200, 250);
e.Graphics.FillEllipse(solidBrush, 300, 50, 200, 200);
e.Graphics.FillPolygon(solidBrush, new PointF[] { new PointF(700, 150), new PointF(600,
300), new PointF(800, 300) });
```

You can also input a shape object directly instead of providing coordinates

```
Rectangle rectangle = new Rectangle(100, 350, 500, 200);
e.Graphics.FillRectangle(solidBrush, rectangle);
```

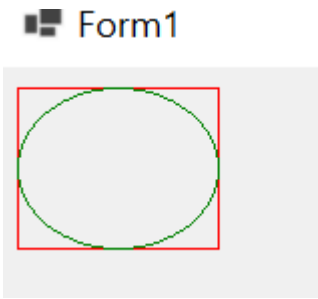
Use the HatchBrush to fill the shape using a different fill style, such as a horizontal or vertical pattern.

```
Color blue = Color.FromArgb(255, 0, 0, 255);
Color green = Color.FromArgb(255, 0, 255, 0);
HatchBrush hatchBrush = new HatchBrush(HatchStyle.Horizontal, red, blue);
e.Graphics.FillRectangle(hatchBrush, 50, 50, 200, 250);
```



Program 1:

```
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 WinFormsApp1_paint
{
    public partial class Form1 : Form
    {
        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            Graphics g = e.Graphics;
            //int w = this.ClientSize.Width;
            //int h = this.ClientSize.Height;
            //pen,brush,font,Image
            g.DrawRectangle(Pens.Red, 10, 10, 100, 80);
            g.DrawEllipse(Pens.Green, 10, 10, 100, 80);
        }}}
```



Width and the height of the working area

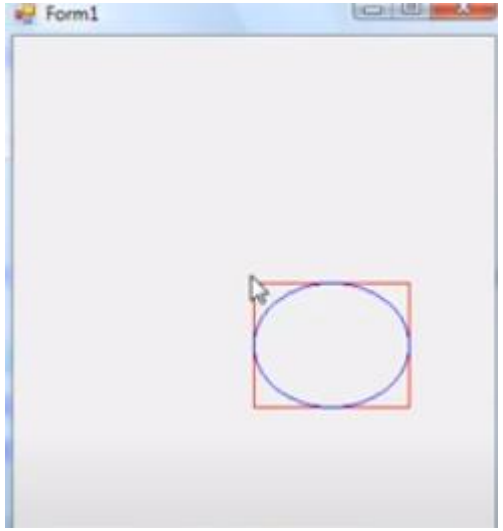
A *pen* is an object that defines line-drawing characteristics. Pens are used to define color, line width, and line style (solid, dashed, and so on), and pens are used with almost all the drawing methods

Creating a using pens

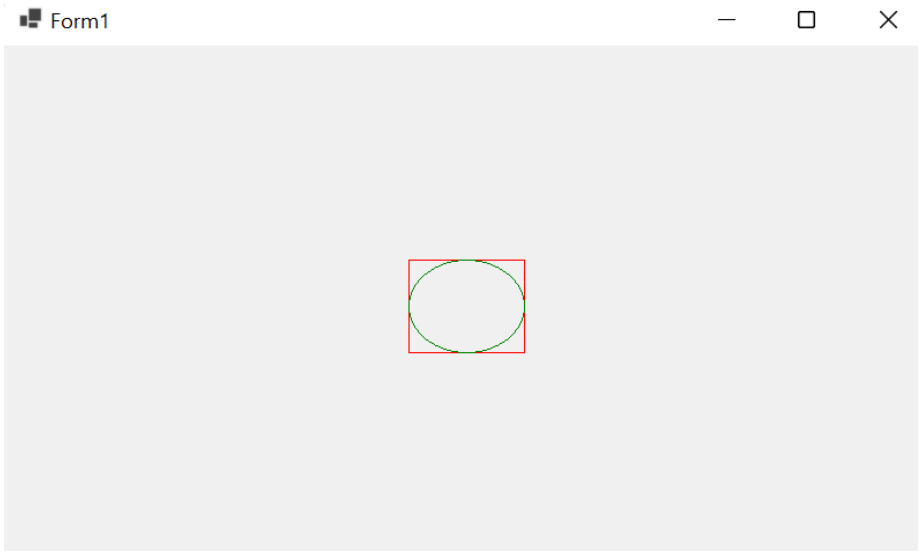
```
Pen pn1 = Pens.Red;
Pen pn2 = Pens.Blue;
Pen pn3 = Pens.Green;
```

```
g.DrawRectangle(Pens.Red, w/2, h/2, 100, 80);
g.DrawEllipse(Pens.Green, w/2, h/2, 100, 80);
```

Diagram will start from the center



```
namespace WinFormsApp1_painting
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void Form1_Paint(object sender, PaintEventArgs e)
        {
            Graphics g = e.Graphics;
            int w = this.ClientSize.Width;
            int h = this.ClientSize.Height;

            //pen,brush,font,Image
            g.DrawRectangle(Pens.Red, w/2-50, h/2-40,100, 80);
            g.DrawEllipse(Pens.Green, w / 2-50, h / 2-40, 100, 80);

        }
    }
}
```



Draw an arc

Syntax: *public void DrawArc (System.Drawing.Pen pen, System.Drawing.Rectangle rect, float startAngle, float sweepAngle)*

Parameters:

pen*: Determines the color, width, and style of the arc.*

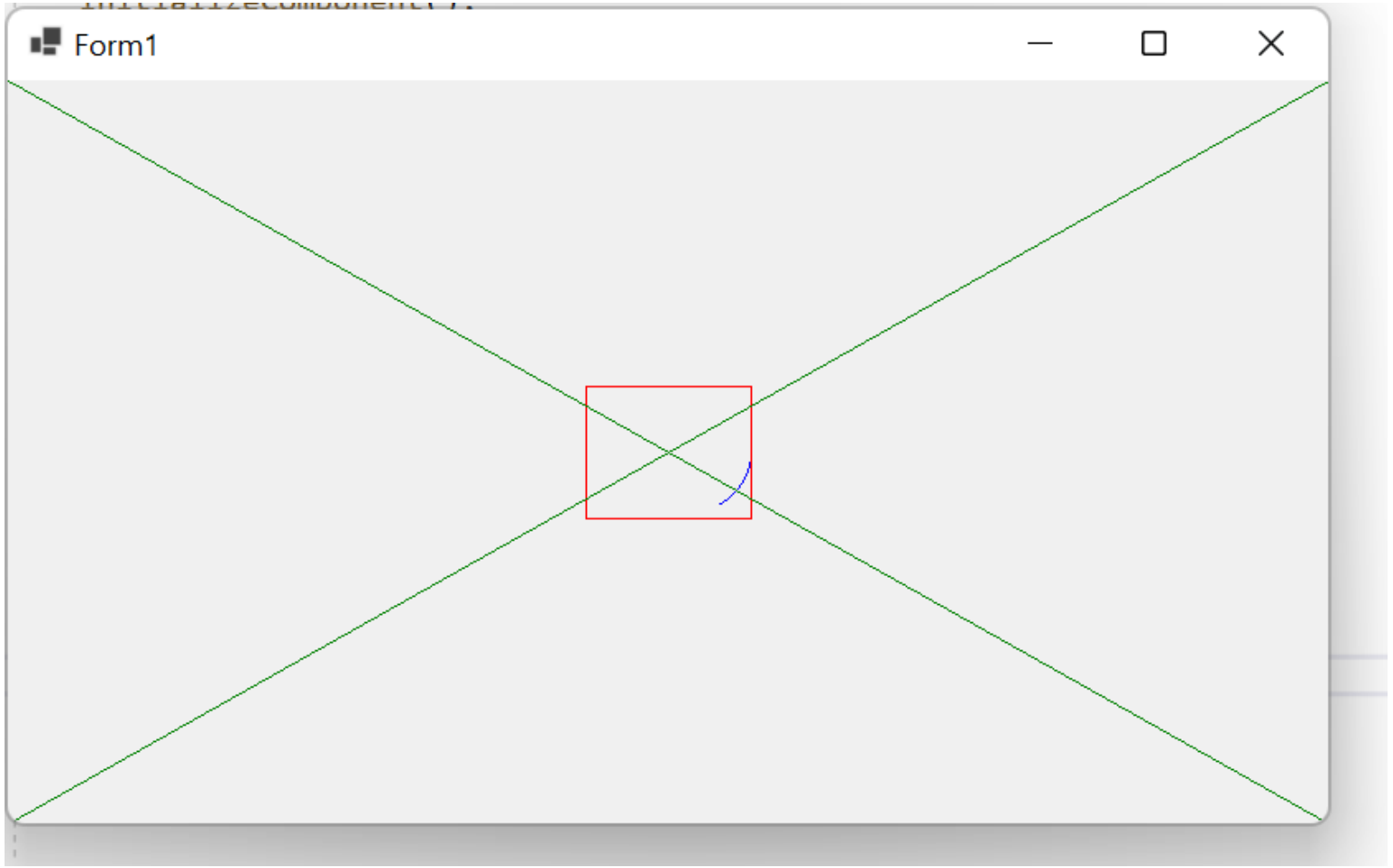
rect*: Determines the smallest rectangle in which the ellipse just perfectly fits into.*

startAngle*: Angle in degrees measured clockwise from the x-axis to the starting point of the arc.*

sweepAngle*: Angle in degrees measured clockwise from the startAngle parameter to ending point of the arc*

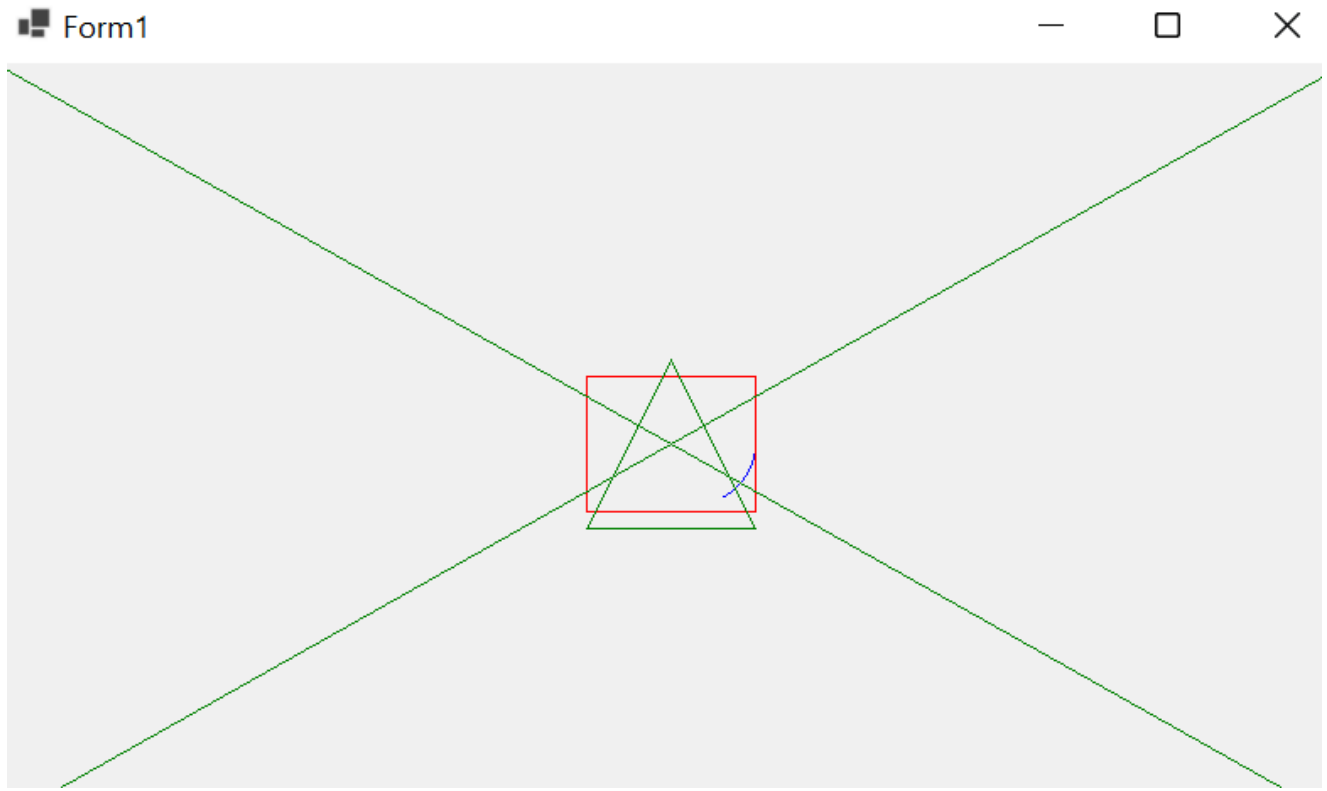
```
g.DrawArc(p, w / 2 - 50, h / 2 - 40, 100, 80, 0, 45);
```

```
p.Dispose();------method is primarily for releasing unmanaged resources
p = new Pen(Color.Green);
g.DrawRectangle(Pens.Red, w / 2 - 50, h / 2 - 40, 100, 80);
g.DrawLine(p, 0, 0, w, h);
g.DrawLine(p, 0, h, w, 0);
```

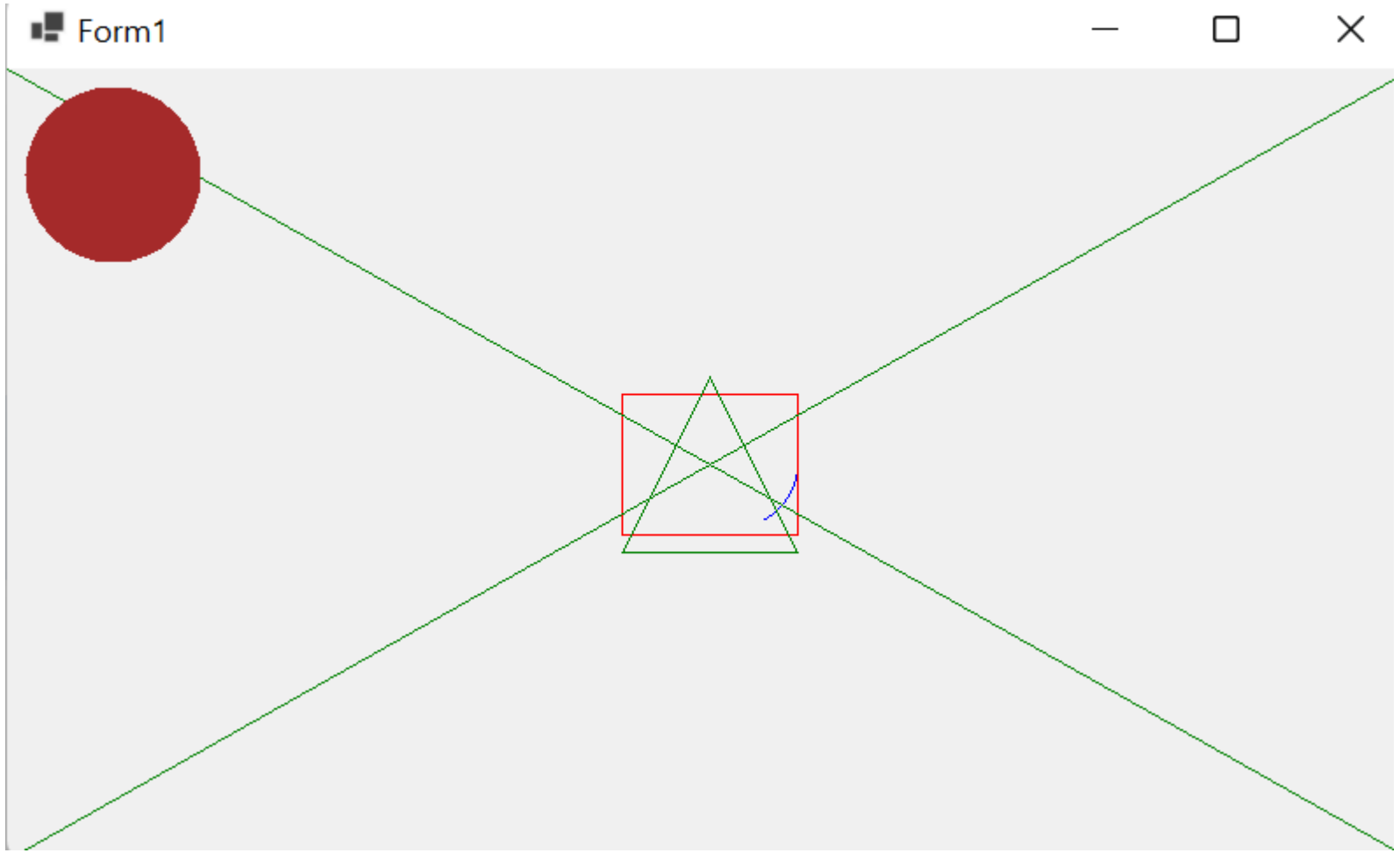


Traingle:

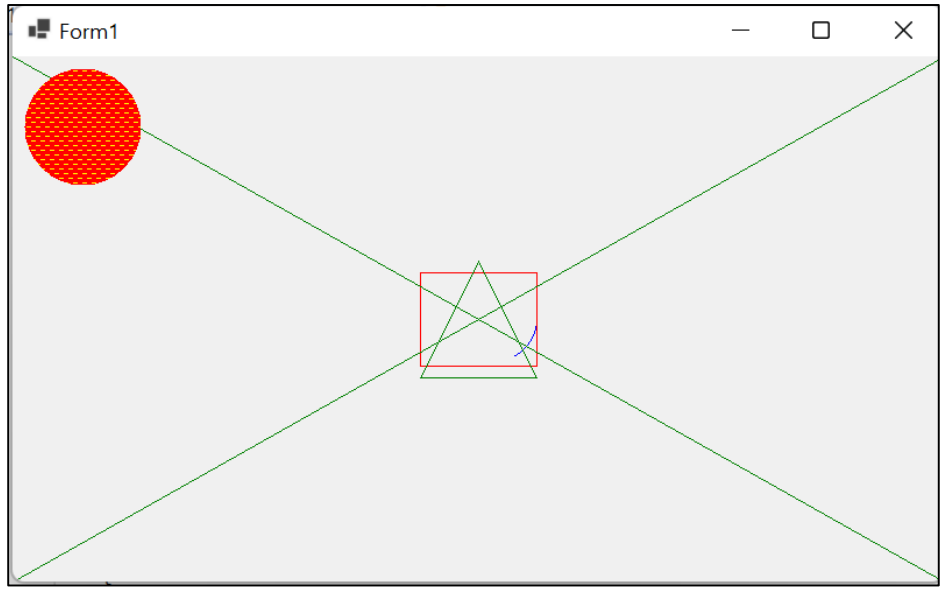
```
Point[] pts = new Point[3];
pts[0].X = w / 2;
pts[0].Y = h / 2 - 50;
pts[1].X = w / 2 + 50;
pts[1].Y = h / 2 + 50;
pts[2].X = w / 2 - 50;
pts[2].Y = h / 2 + 50;
g.DrawPolygon(p, pts);
```

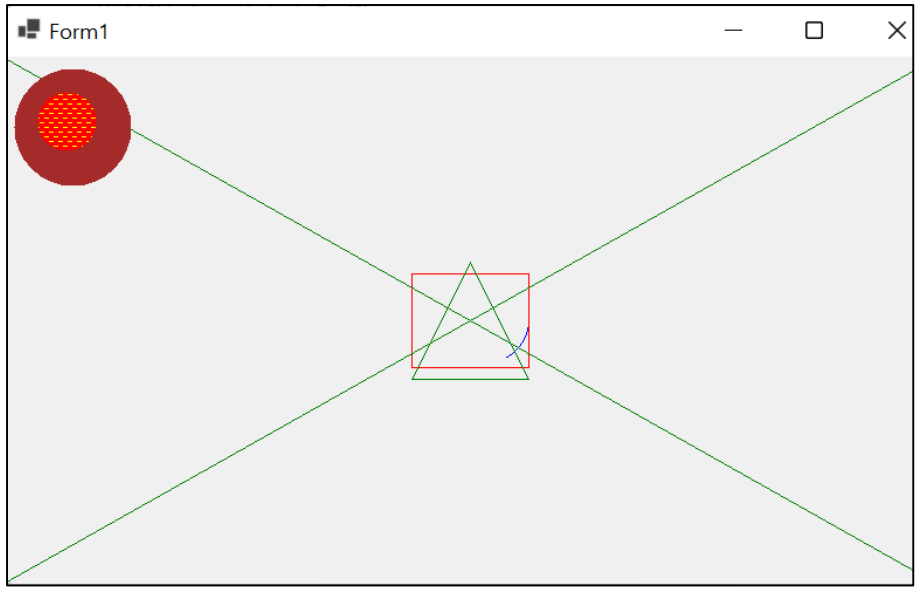


```
Brush br;
Color col = Color.FromArgb(50, 255, 127, 12);
col = Color.FromName("Brown");
br = new SolidBrush(col);
g.FillEllipse(br, 10, 10, 100, 100);
```



```
br = new HatchBrush(HatchStyle.DashedHorizontal, Color.Yellow, Color.Red);
g.FillEllipse(br, 10, 10, 100, 100);
```






```
br = new HatchBrush(HatchStyle.DashedHorizontal, Color.Yellow,
Color.Red);
            g.FillEllipse(br, 30, 30, 50, 50);
```

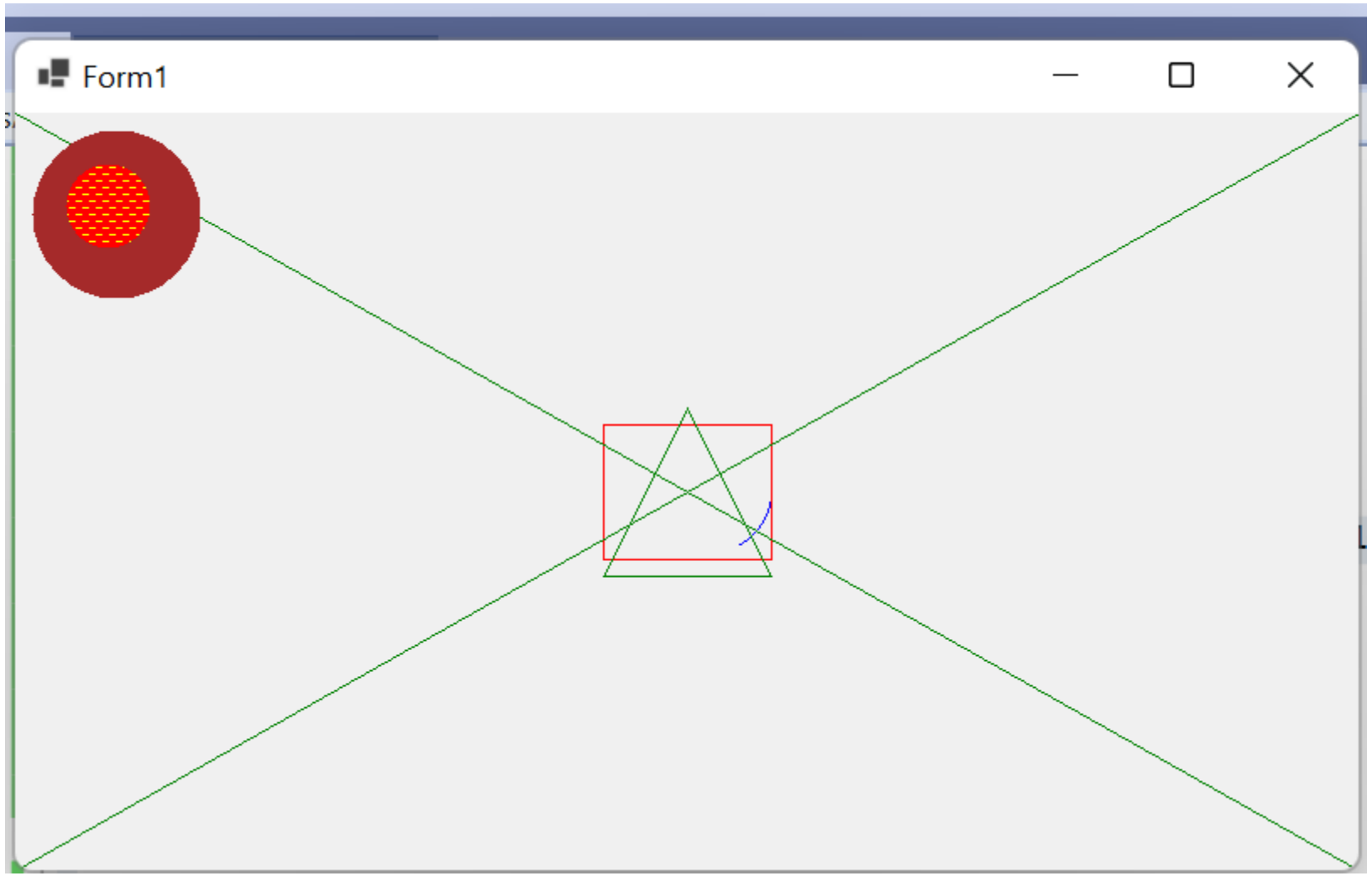
Full program:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Drawing.Drawing2D;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace WinFormsApp1_painting
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
```

```
private void Form1_Paint(object sender, PaintEventArgs e)
        {
            Graphics g = e.Graphics;
            int w = this.ClientSize.Width;
            int h = this.ClientSize.Height;

            //pen,brush,font,Image
            Pen p = new Pen(Color.Blue);
            g.DrawArc(p, w / 2 - 50, h / 2 - 40, 100, 80, 0, 45);
            p.Dispose();
            p = new Pen(Color.Green);
            g.DrawRectangle(Pens.Red, w / 2 - 50, h / 2 - 40, 100, 80);
            g.DrawLine(p, 0, 0, w, h);
            g.DrawLine(p, 0, h, w, 0);
```

```
Point[] pts = new Point[3];
            pts[0].X = w / 2;
            pts[0].Y = h / 2 - 50;
            pts[1].X = w / 2 + 50;
            pts[1].Y = h / 2 + 50;
            pts[2].X = w / 2 - 50;
            pts[2].Y = h / 2 + 50;
            g.DrawPolygon(p, pts);
            Brush br;
            Color col = Color.FromArgb(50, 255, 127, 12);
            col = Color.FromName("Brown");
            br = new SolidBrush(col);
            g.FillEllipse(br, 10, 10, 100, 100);
            br = new HatchBrush(HatchStyle.DashedHorizontal, Color.Yellow, Color.Red);
            g.FillEllipse(br, 30, 30, 50, 50);
```

```
br = new HatchBrush(HatchStyle.DashedHorizontal, Color.Yellow, Color.Red);
            g.FillEllipse(br, 30, 30, 50, 50);


            //g.DrawRectangle(Pens.Red, w/2-50, h/2-40,100, 80);
            //g.DrawEllipse(Pens.Green, w / 2-50, h / 2-40, 100, 80);

        }

        private void Form1_Load(object sender, EventArgs e)
        {

        }

        private void Form1_Resize(object sender, EventArgs e)
        {
            Invalidate();
        }
    }
}
```


Form1

https://www.youtube.com/watch?v=CMCBHK6WUoA