Windows Forms and Controls

Introduction to Windows Forms and Controls

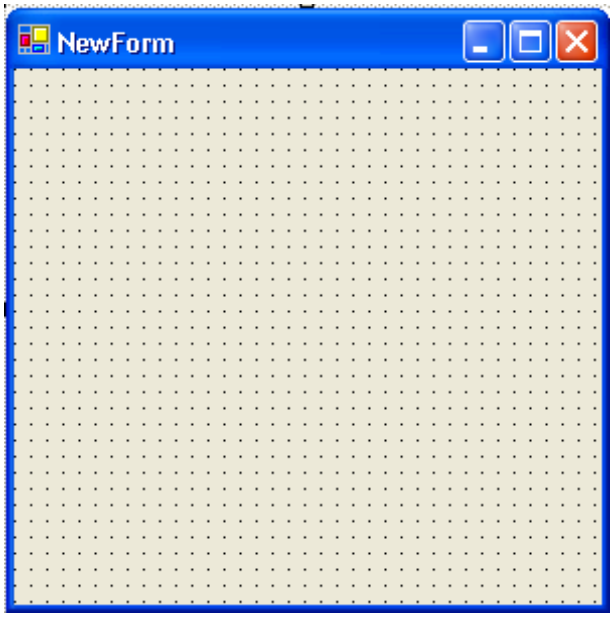
Windows Forms are the basic building surface of windows applications. Controls are nuggets of functionality that we can add to a Form. We have already made several windows forms and used controls such as the Button and TextBox to create interfaces for our simple applications. The objective of this chapter is to look at how windows forms and controls can be created and launched programmatically. Also, we will look at how data can be transformed to Forms and Controls and how it can be retrieved from them.

As with most elements that we have seen up to now, Forms and Controls are C# classes that inherit from special .NET classes. These .NET classes provide the Forms and Controls we will develop with most the functionality needed.

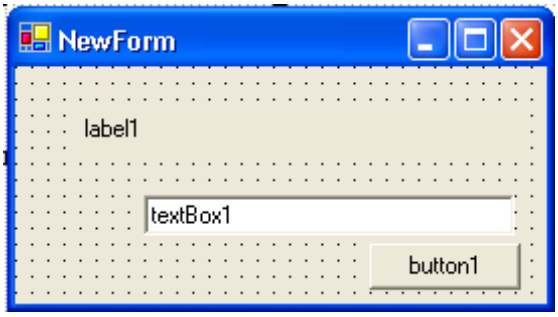
To start, use Visual Studio to create a new application called WindowsFormsAndControls. As expected, Visual Studio already has created a Form1 for us to start building our application from.

Creating a new Form

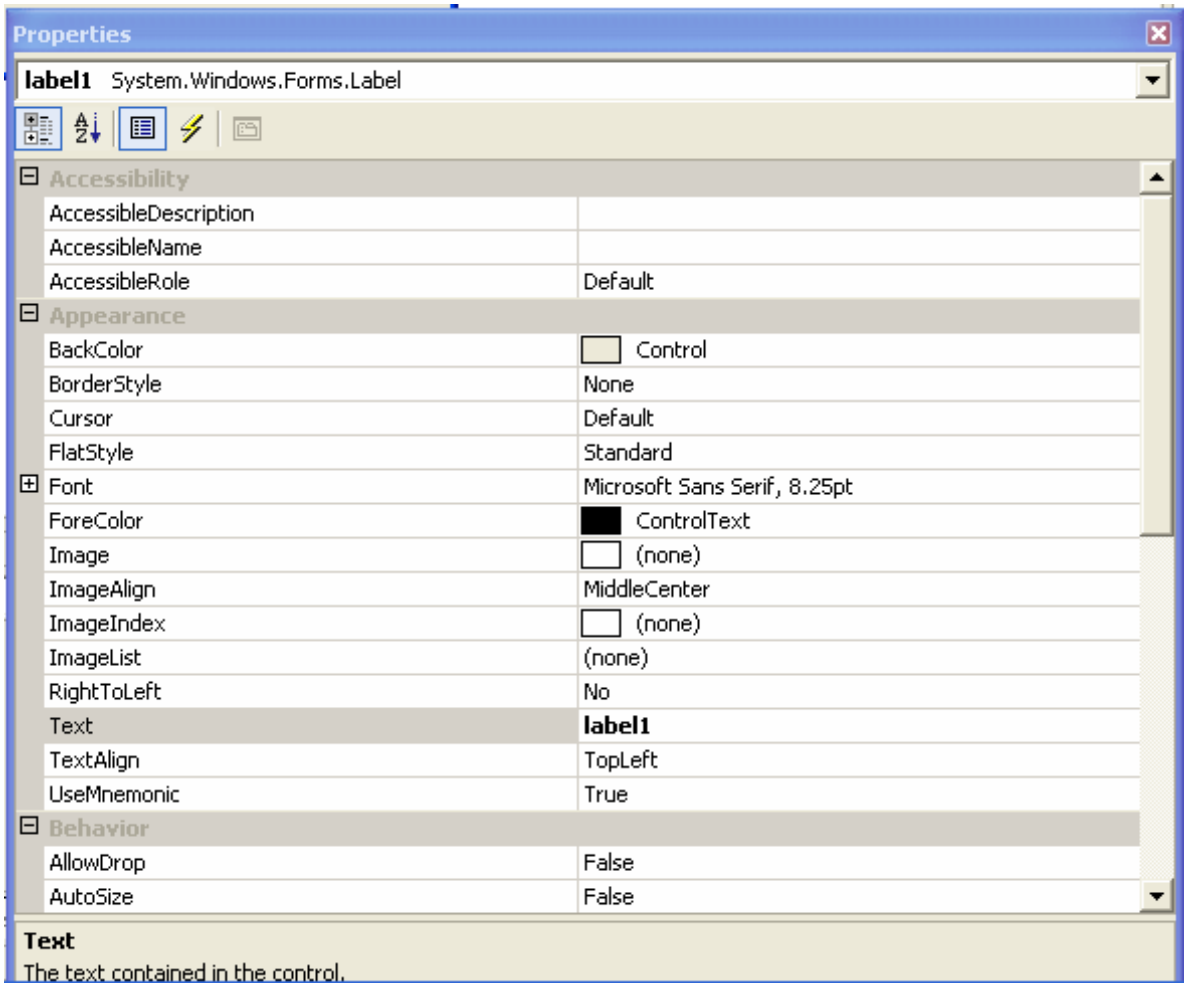
To start with the creation of a new Form in Visual Studio select File, Add New Item, and then Windows Form, and name the Form "NewForm." Visual Studio will create a form and show you the following designer view.



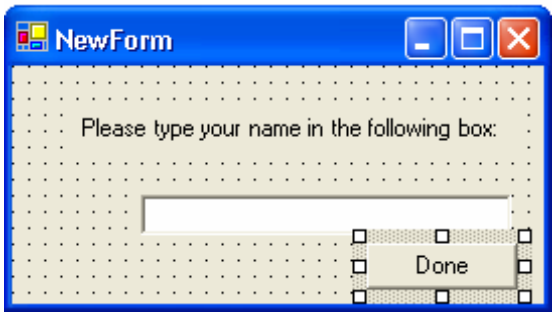
Now drag and drop a Label, a TextBox, and a Button from the Toolbox onto your NewForm and resize it so that it looks as follows:



Right-Click on label1 and select its properties to get:



Change the Text property from “label1” to “Please type your name in the following box:”. Then Click on the TextBox and delete the text in the Text property. After that, Click on the Button and change the Text property to “Done”. Your form should now look like this:



Now from the View menu select Code to see the C# code that generates this Form. You should see something like this:

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;

namespace WindowsFormsAndControls
{
	/// <summary>
	/// Summary description for NewForm.
	/// </summary>
	public class NewForm : System.Windows.Forms.Form
	{
		private System.Windows.Forms.Label label1;
		private System.Windows.Forms.TextBox textBox1;
		private System.Windows.Forms.Button button1;
		/// <summary>
		/// Required designer variable.
		/// </summary>
		private System.ComponentModel.Container components = null;

		public NewForm()
		{
			//
			// Required for Windows Form Designer support
			//
			InitializeComponent();

			//
			// TODO: Add any constructor code after InitializeComponent call
			//
		}

		/// <summary>
		/// Clean up any resources being used.
		/// </summary>
		protected override void Dispose( bool disposing )
		{
			if( disposing )
			{
				if(components != null)
				{
					components.Dispose();
				}
			}
			base.Dispose( disposing );
```



```
        }

        #region Windows Form Designer generated code
        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
                this.label1 = new System.Windows.Forms.Label();
                this.textBox1 = new System.Windows.Forms.TextBox();
                this.button1 = new System.Windows.Forms.Button();
                this.SuspendLayout();
                //
                // label1
                //
                this.label1.Location = new System.Drawing.Point(32, 24);
                this.label1.Name = "label1";
                this.label1.Size = new System.Drawing.Size(224, 23);
                this.label1.TabIndex = 0;
                this.label1.Text = "Please type your name in the following box:";
                //
                // textBox1
                //
                this.textBox1.Location = new System.Drawing.Point(64, 64);
                this.textBox1.Name = "textBox1";
                this.textBox1.Size = new System.Drawing.Size(184, 20);
                this.textBox1.TabIndex = 1;
                this.textBox1.Text = "";
                //
                // button1
                //
                this.button1.Location = new System.Drawing.Point(176, 88);
                this.button1.Name = "button1";
                this.button1.TabIndex = 2;
                this.button1.Text = "Done";
                //
                // NewForm
                //
                this.AutoScaleBaseSize = new System.Drawing.Size(5, 13);
                this.ClientSize = new System.Drawing.Size(264, 118);
                this.Controls.Add(this.button1);
                this.Controls.Add(this.textBox1);
                this.Controls.Add(this.label1);
                this.Name = "NewForm";
                this.Text = "NewForm";
                this.ResumeLayout(false);

        }
        #endregion
    }
}
```

Note that Visual Studio first attached several namespaces to this form, the most important is System.Windows.Forms that contains the Form class that all forms inherit from. Next Visual Studio put our Form in the namespace of our application so that it can be accessed from our application WindowsFormsAndControls. Then the NewForm was declared as a class that inherits from System.Windows.Forms.Form.

Next, label1, textBox1 and button1 are declared as variables. Since these are declared outside of any methods of NewForm, they will act as field variables or global variables, and will, therefore, be accessible to all methods within NewForm. For our application declare a new private field variable name, and public property Name as follows:

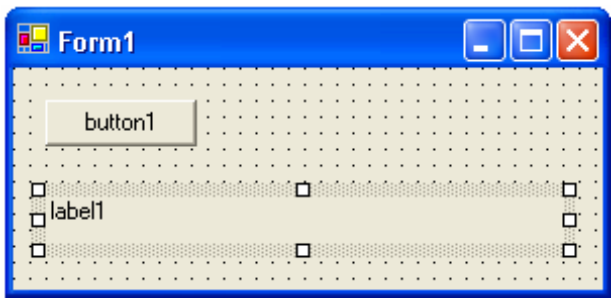
```
private System.Windows.Forms.Label label1;
private System.Windows.Forms.TextBox textBox1;
private System.Windows.Forms.Button button1;

private string name;
public string Name
{
	get
	{
		return name;
	}
	set
	{
		name = value;
	}
}
```

Now go back to Designer view by selecting Designer from the View menu. Double-Click on the button to get the method that is run on its Click event. To this method add the following code to assign the content of the TextBox to our variable name and then close NewForm.

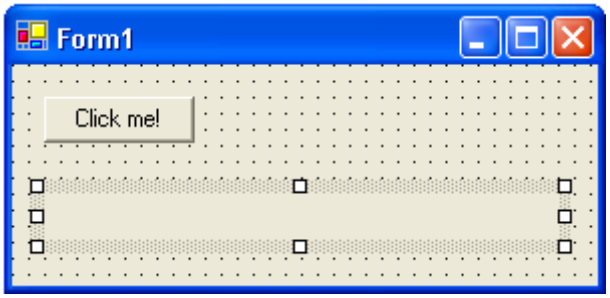
```
private void button1_Click(object sender, System.EventArgs e)
{
	name = this.textBox1.Text;
	this.Close();
}
```

Using Forms

The next task after creating a form is to learn how to use it. Return to the Designer view of Form1 of your application and drag and drop a button and a label from the toolbox onto the form and resize it to get:



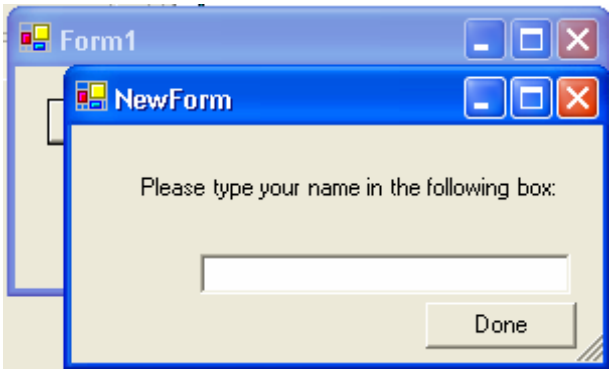
Change the Text property of the button to “Click me!” and remove the text from the Text property of the label. You should now have:



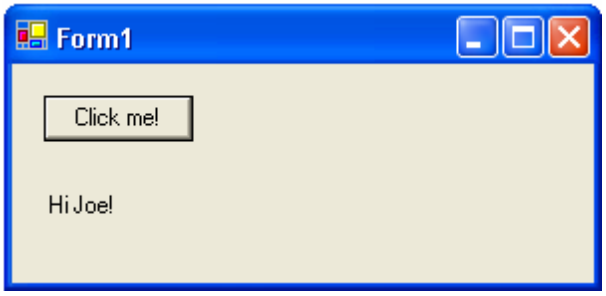
Double-Click on the button to get the method that will run on its Click event. In this method type the following:

```
private void button1_Click(object sender, System.EventArgs e)
{
	NewForm myForm = new NewForm();
	myForm.ShowDialog();
	this.label1.Text = "Hi "+myForm.Name+"!";
}
```

Note that in the first line we have declared a NewForm with the name myForm and then have initialized it. In the second line we show myForm with the method ShowDialog(), which will open myForm and freeze Form1 until we close myForm. This is the most common way of opening a form since it guarantees that we don’t access any of the properties of myForm in Form1 until we have completed this form. Finally, in the third line we make label1 show a string based on the results of myForm. If we now build and run our application and then click on the button we will get the NewForm looking like this:



If we type the name “Joe” in it and press Done, the following window will appear:



Now if we click the button again, we will get a NewForm with a blank TextBox again. Let us change our code for the button so that our previous text in the TextBox of the form appears every time we click the "Click me!" button. We first have to provide a way for Form1 to remember the output of NewForm from one click of the button to the other. We need a variable to keep the name in, but we can't declare this variable in the method of the button since it will go out of scope between clicks and will be lost. So we make our variable a field variable of Form1. We go to the start of Form1 and declare a private string newFormName as follows:

```
public class Form1 : System.Windows.Forms.Form
{
	private System.Windows.Forms.Button button1;
	private System.Windows.Forms.Label label1;
	private string newFormName = "";
```

The next thing is to give the value stored in newFormName to myForm before it is opened, and then to change the value of newFromName after myForm is closed. This can be done by adding these lines to the Click even of the button on Form1:

```
NewForm myForm = new NewForm();

myForm.Name = newFormName;

myForm.ShowDialog();
this.label1.Text = "Hi "+myForm.Name+"!";

newFormName = myForm.Name;
```

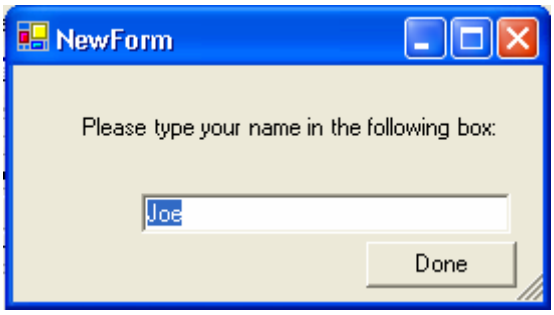
If you now build and run the application, you will notice that it still does not show the old name in the NewForm. The problem is that we never told NewForm to set the value of its TextBox equal to the value of the Name property when we set its value. This can be done by going to NewForm and changing the Name property to the following:

```
public string Name
{
	get
	{
		return name;
	}
	set
	{
		name = value;
```



```
            this.textBox1.Text = name;
        }
}
```

Now if you build and run the program you will note that each time you will have the previous input displayed in the NewForm TextBox such as:



More about Forms

As was shown above, the transfer of data between forms is conducted in the standard ways available for classes. Two primary ways to access the variables of a form is through its public field variables and through its properties. This is because a form is just another class.

In the example above, we opened the form with the "ShowDialog" method, which froze all windows associated with the application until we closed this new form. Another way to open a window is to use the "Show" method, which allows both windows to be active at the same time. To see how this works change the "Click me!" button Click event method as follows:

```
private void button1_Click(object sender, System.EventArgs e)
{
        NewForm myForm = new NewForm();
        myForm.Name = newFormName;

        myForm.Show();

        this.label1.Text = "Hi "+myForm.Name+"!";
        newFormName = myForm.Name;
}
```

Build and run the program again. What happens? The application is no longer working! Why? Because, before this change Form1 would wait at the line "myForm.ShowDialog();" until the Done button was Clicked and Text of the TextBox stored in the name variable so that when the program got to the last two lines

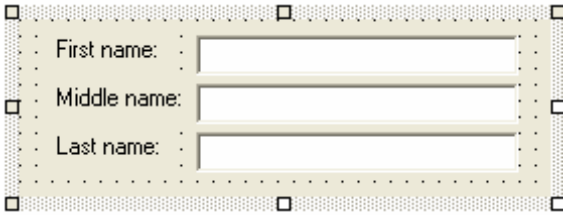
```
        this.label1.Text = "Hi "+myForm.Name+"!";
        newFormName = myForm.Name;
```

the property myForm.Name had the value typed into the TextBox. But, now the program, being much faster than we are, opens the myForm when it reaches myForm.Show() and imedeately proceeds to the next command lines and executes them even before we have completed the NewForm. This is the reason why normally the ShowDialog() method is used, it forces the program to stop execution of the original code until the form is completed.

There are times when there is no reason to stop the execution of a form when a new form is being displayed and in these instances the Show method can be useful. An example would be a form that simply displays certain values without returning any data.

Making user controls

Just like making a new form, we can also make user controls that can be used over and over. To make a user control, from the File menu in Visual Studio select Add New Item and then User Control and name it MyControl. Visual Studio will create for you a blank control. Drag and drop three labels and three text boxes, change the properties and resize it to look as follows:



View the code of the control and add the following struct in the namespace command, but outside the user control:

```
public struct nameStruct
{
	public string FirstName;
	public string MiddleName;
	public string LastName;

	public string FullName()
	{
		string fullName = FirstName+" "+MiddleName+" "+LastName;
		return fullName;
	}
}
```

This will create a little container that we can store the three parts of a name and transfer it as one unit. It also has a method that constructs the full name string. Now, inside the user control add the following private variable and property.

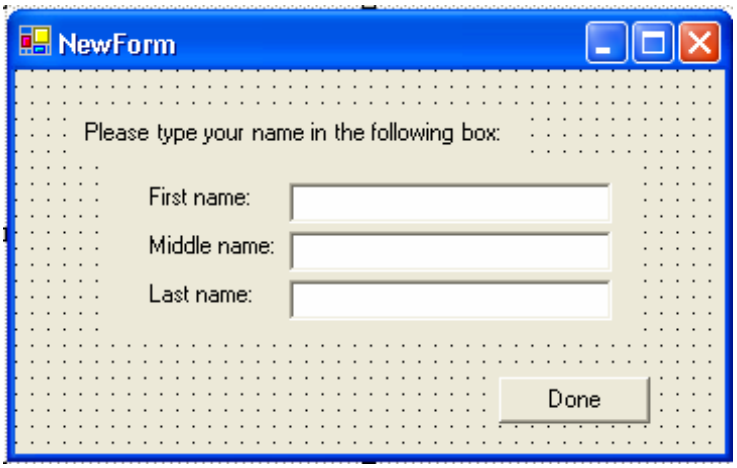
```
private nameStruct myName;
public nameStruct MyName
```



```
{
	get
	{
		myName.FirstName = this.textBox1.Text;
		myName.MiddleName = this.textBox2.Text;
		myName.LastName = this.textBox3.Text;
		return myName;
	}
	set
	{
		myName = value;
		this.textBox1.Text = myName.FirstName;
		this.textBox2.Text = myName.MiddleName;
		this.textBox3.Text = myName.LastName;
	}
}
```

As you can see, this control will get the first, middle and last name and store it in myName and make it available through the property MyName. It also will initialize the Text properties of the three TextBoxes on the set segment of the property.

Now close myControl designer, build the application, and then go to the NewForm designer, remove the text box and drag and drop myControl from the toolbox onto NewForm and resize it until you get:



As you can see, our control, which is a composite of three controls appears as one single unit. Now remove the previous field variables and properties you added in NewForm with the following variable and property:

```
private nameStruct myName;
public nameStruct MyName
{
	get
	{
		return myName;
	}
	set
```

```
        {
                myName = value;
                myControl1.MyName = myName;
        }
}
```

Next, change the method for the Click event of the Done button by

```
this.myName = this.myControl1.MyName;
this.Close();
```

As you can see, we are now passing the nameStruct myName that contains the first, middle and last name back and forth through the elements of our program.

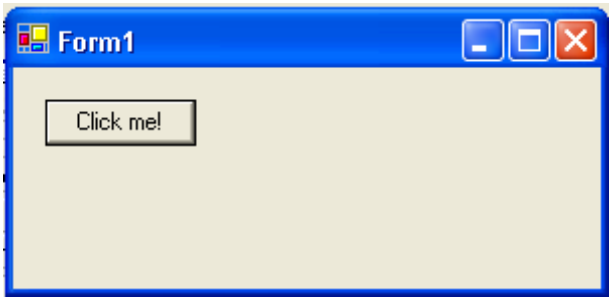
The next thing we need to do is change the Name variable in Form1 to a myStruct type so that we can get, set, and use it in Form1. This is done by replacing the previous field variable we added in Form1 by:

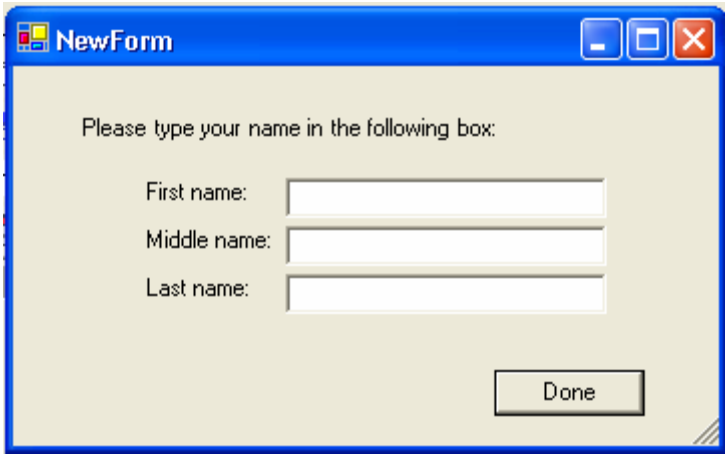
```
private nameStruct newFormName;
```

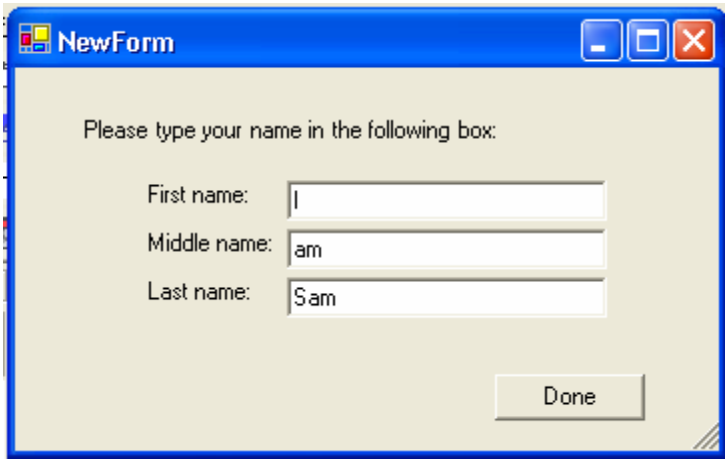
Finally, the button Click event method needs to be changed in Form1 to

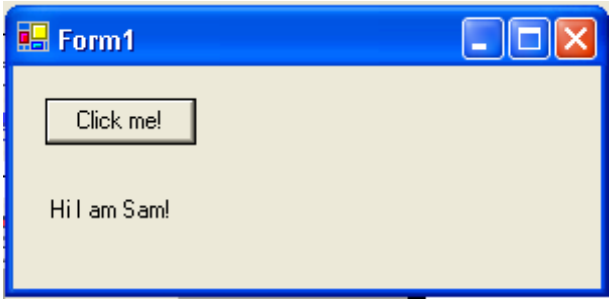
```
private void button1_Click(object sender, System.EventArgs e)
{
        NewForm myForm = new NewForm();
        myForm.MyName = newFormName;

        myForm.ShowDialog();
        this.label1.Text = "Hi "+myForm.MyName.FullName()+"!";

        newFormName = myForm.MyName;
}
```

Now if you build and run the program you will see a sequence of windows









Summary

In this chapter we looked at Forms and User Controls and showed how to make and use them. In the process we noted that both Forms and User Controls are classes so that we can use the same data transfer methods that we learned for a class with Forms and User Controls. Specifically, we used public variables and properties to transfer data to and from Forms and User Controls.

In the process of the examples, we also created a struct to store the elements of a name (first, middle, and last name) and a method to construct the full name from these

elements. This single variable was passed back and forth between our forms and user controls.

We also looked at the Form method ShowDialog() and why it is needed to stop the execution of the calling Form until the control is released by the new form.

Normally User Controls are created when a certain functionality will be used over and over, as one might expect that gathering names and displaying them might occur in many places. Creation of a User Control that collects the name and packages it into a compact and function unit such as the struct we created could be used in many different places.

