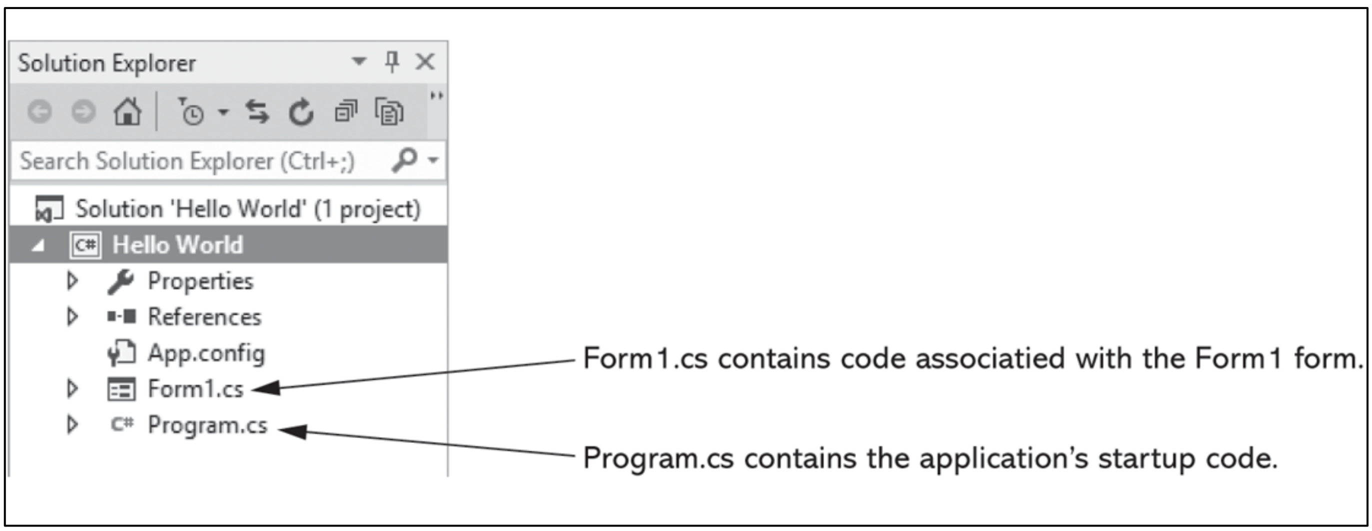
Creating the GUI for Your First Visual C# Application

In this chapter you will create your first Visual C# application, which will be an event-driven program. This section introduces you to Visual C# code and shows how to program an application to respond to button clicks.

A file that contains program code is called a **source code file**. When you start a C# Windows Forms Application project, Visual Studio automatically creates several source code files and adds them to the project. If you look at the Solution Explorer, as shown in Figure 2-1, you will see the names of two source code files: **Form1.cs** and **Program.cs.** (C# source code files always end with the **.cs** extension.)

Figure 2-1 Source code files shown in the Solution Explorer



The Program.cs file holds the application's startup code and handles essential initialization. Modifying it may prevent the application from running.

The Form1.cs file contains the code associated with the Form1 form. Any actions or behaviors related to Form1, such as responding to button clicks or other user interactions, are implemented in this file.

The Code Editor

It will be helpful for you to know how this code is organized, however, because later you will add your own code to this file. C# code is primarily organized in three ways, as shown in the Figure 2-2:

A **namespace** is a container that holds classes.

A **class** is a container that holds methods (among other things).

A **method** is a group of one or more programming statements that performs some operation.

Figure 2-2 Organization of the Form1.cs code

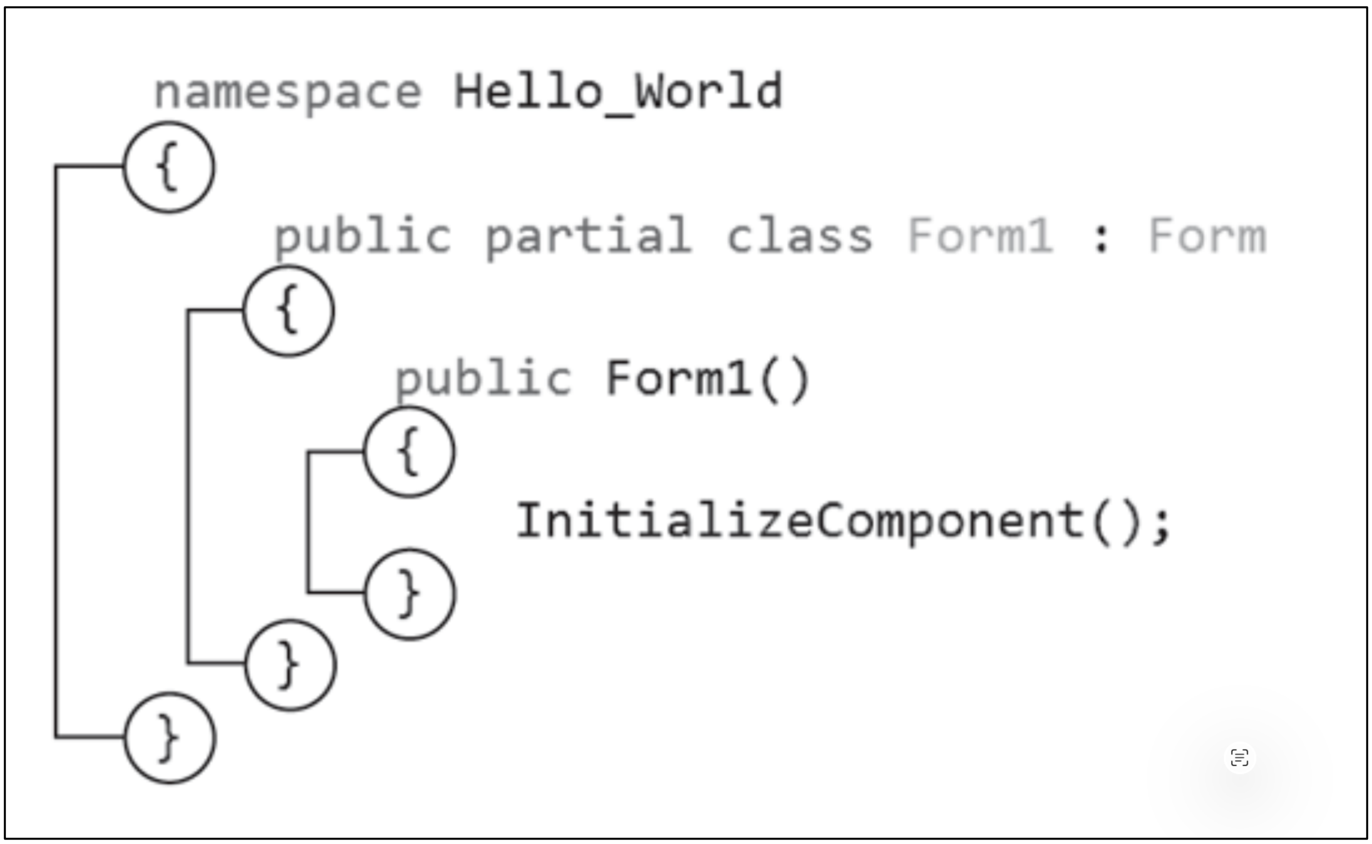
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

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

- C# applications use the .NET Framework, with code organized into namespaces indicated by using directives.
- The `namespace Hello_World { }` block defines a namespace for the project.
- The `public partial class { }` block defines a class and its contents.
- The `public Form1() { }` block defines a method and its code.

Code containers (namespaces, classes, methods) use paired braces `{ }` to enclose their code , as seen in Figure 2-3.

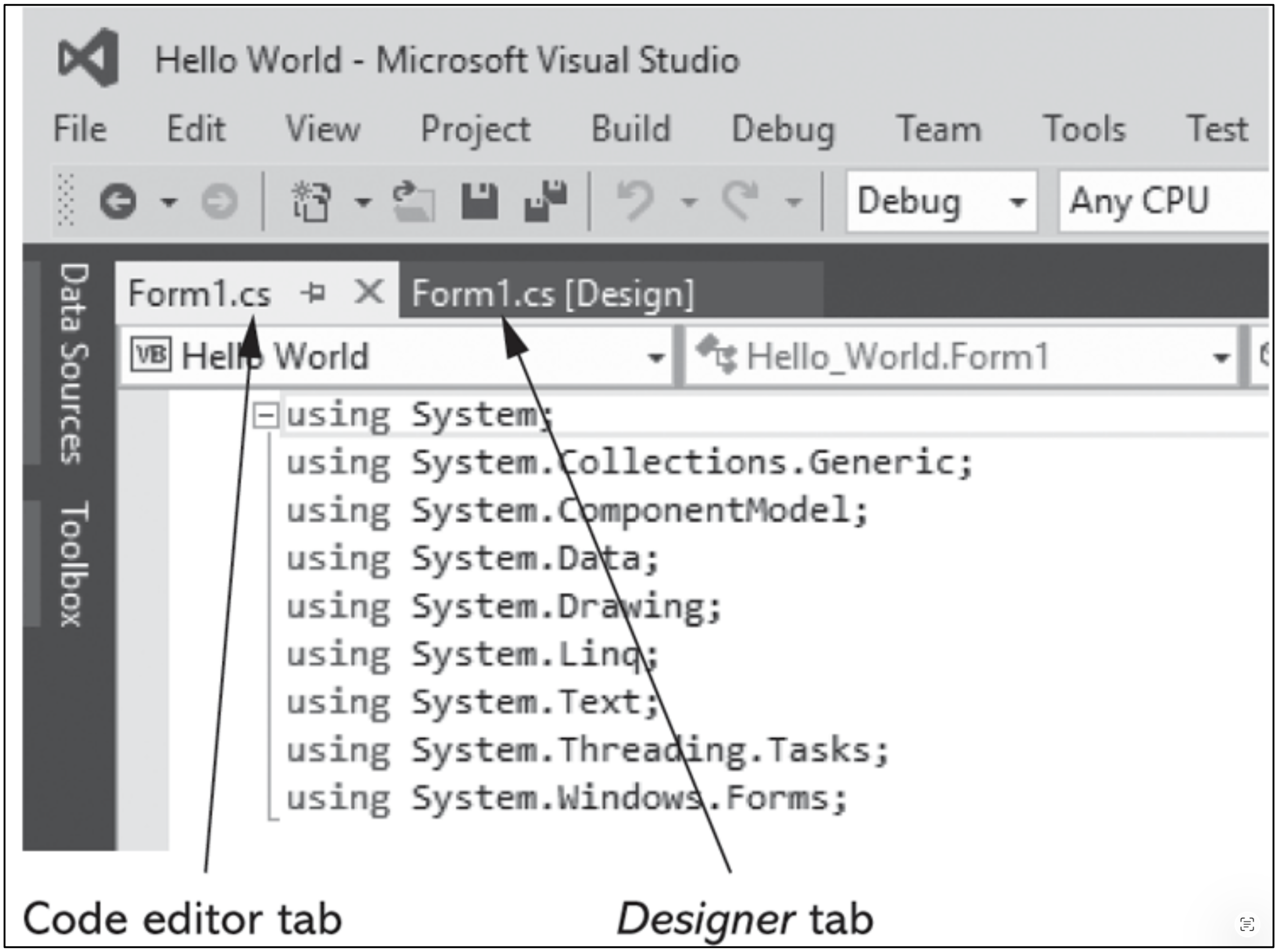
Figure 2-3 Corresponding braces



Switching Between The Code Editor And The Designer

The code editor and Designer share the same window, and you can switch between them using tabs like Form1.cs, As shown in the Figure 2-4.

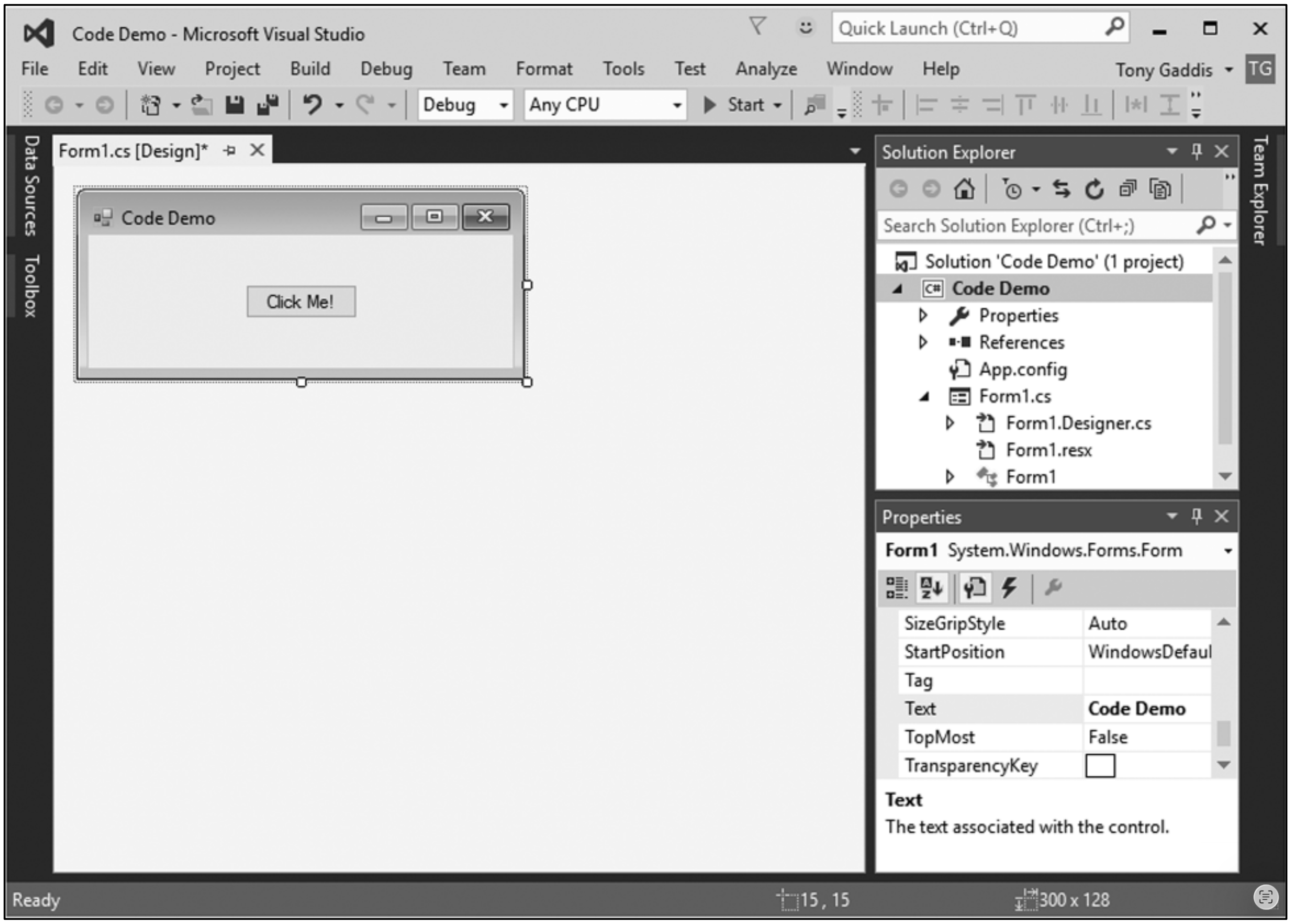
Figure 2-4 Code editor and Designer tabs



Adding Your Own Code To A Project

Suppose you have created a project named *Code Demo* and set up the project's form with a **Button** control, as shown in Figure 2-5. The Button control's **Name** is *myButton*, and its **Text** property is set to *Click Me!*.

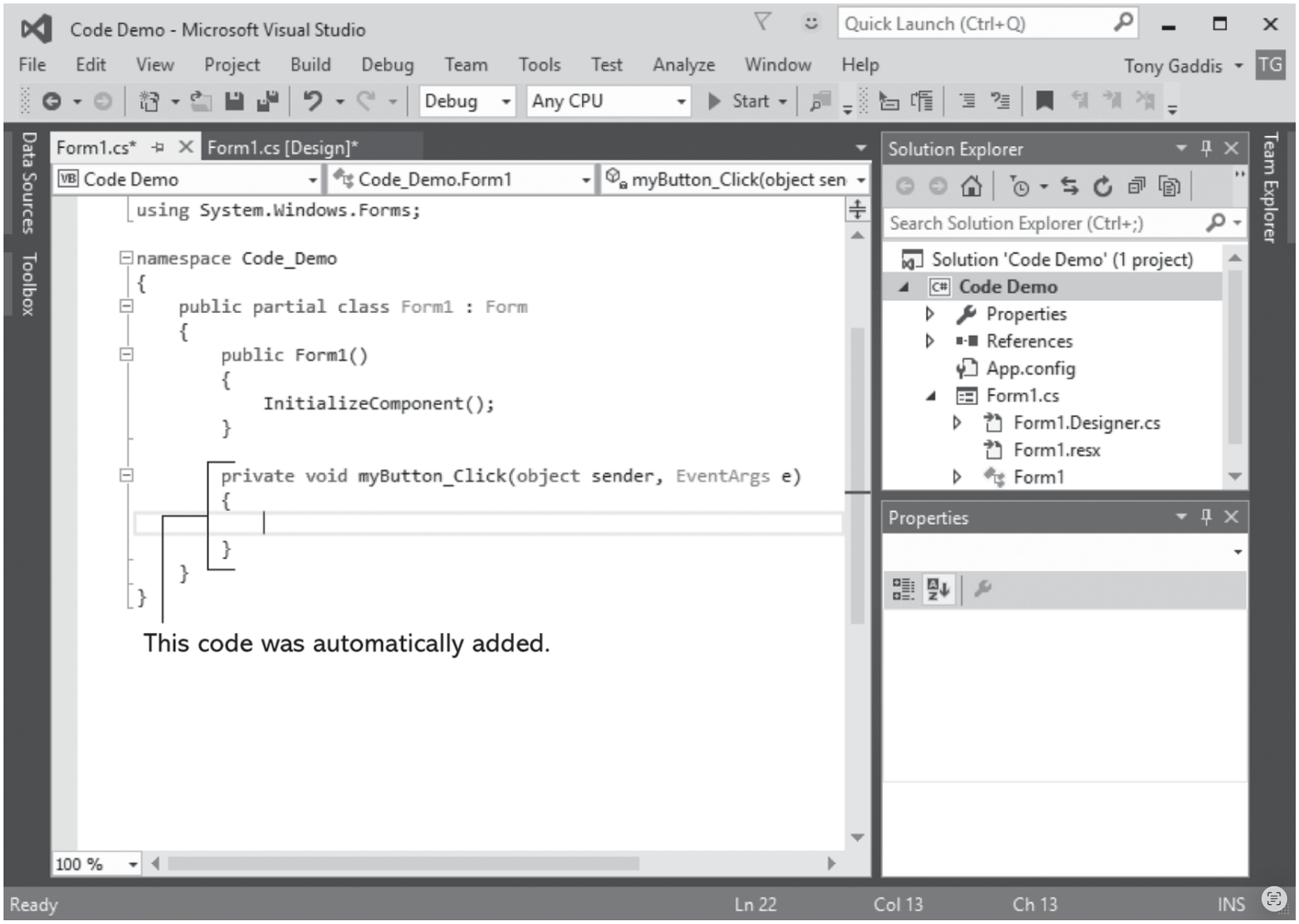
Figure 2-5 A form with a Button control



Suppose you want the application to display the message *Thanks for clicking the button!* when the user clicks the button. To accomplish that, you need to write a special type of method known as an event handler. An **event handler** is a method that executes when a specific event takes place while an application is running.

To create the event handler, you double-click the myButton control in the Designer. This opens the Form1.cs file in the code editor, as shown in Figure 2-6, with some new code added to it.

Figure 2-6 The code window opened with event handler code generated



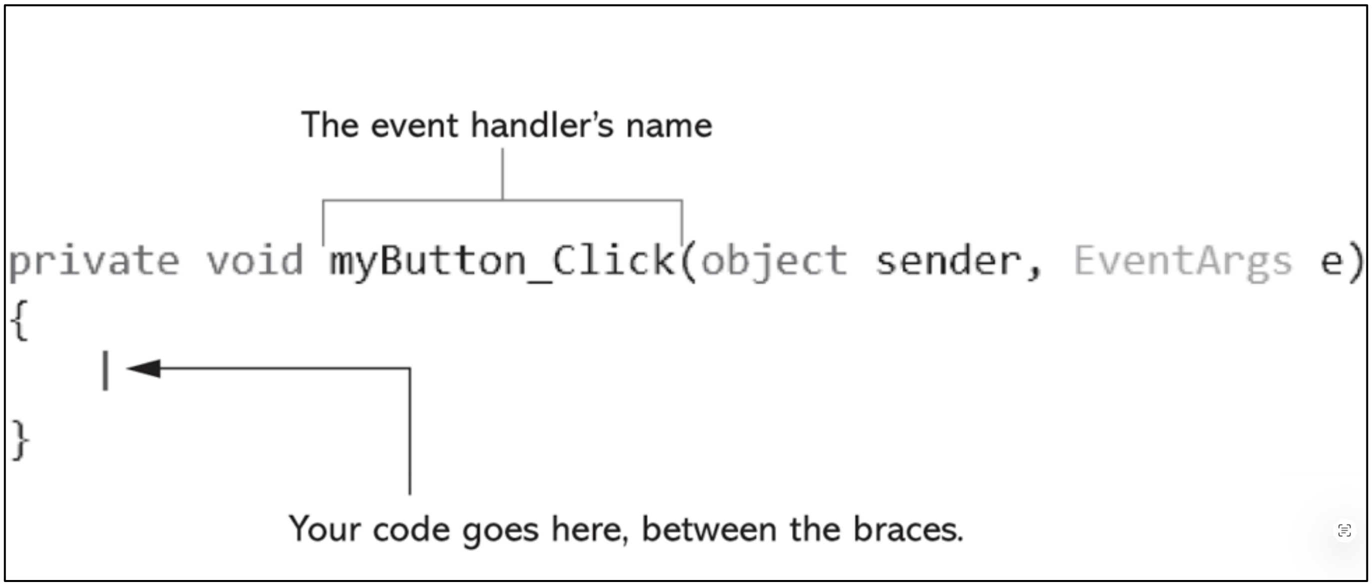
At this point, you need to understand only the following concepts:

- Event handler names follow the convention ***controlName_Event***, e.g., `myButton_Click` for a Click event on *myButton*.

controlName_Event

- Visual Studio generates an empty event handler `{ }` where you add the code to run when the event occurs. See the Figure 2-7.

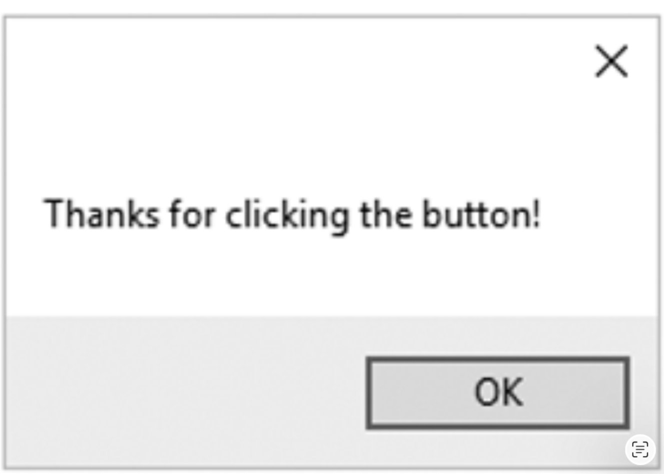
Figure 2-7 A closer look at the event handler code



Message Boxes

A message box is a small window, sometimes referred to as a dialog box, that displays a message. Figure 2-8 shows an example of a message box displaying the message *Thanks for clicking the button!* Notice that the message box also has an OK button. When the user clicks the OK button, the message box closes.

Figure 2-8 A message box



The .NET Framework provides a method named **MessageBox.Show** that you can use in Visual C# to display a message box. If you want to execute the MessageBox.Show method, you write a statement known as a method call.

The following statement shows an example of how you would call the MessageBox.Show method to display the message box shown in Figure 2-8:

```
MessageBox.Show("Thanks for clicking the
button!");
```

When using the MessageBox.Show method in C#, three key elements must be considered:

1- String: The message to be displayed is written as a string inside parentheses.
2- Quotation Marks: The string must be enclosed in double quotation marks (" ").
3- Semicolon: Each statement ends with a semicolon (;), which marks the end of the instruction in C#.

After typing the statement as shown in the Figure 2-9 and When the application runs, it will display the form shown in Figure 2-10.

Figure 2-9 Event handler code for displaying a message box

Form1.cs Form1.cs [Design]

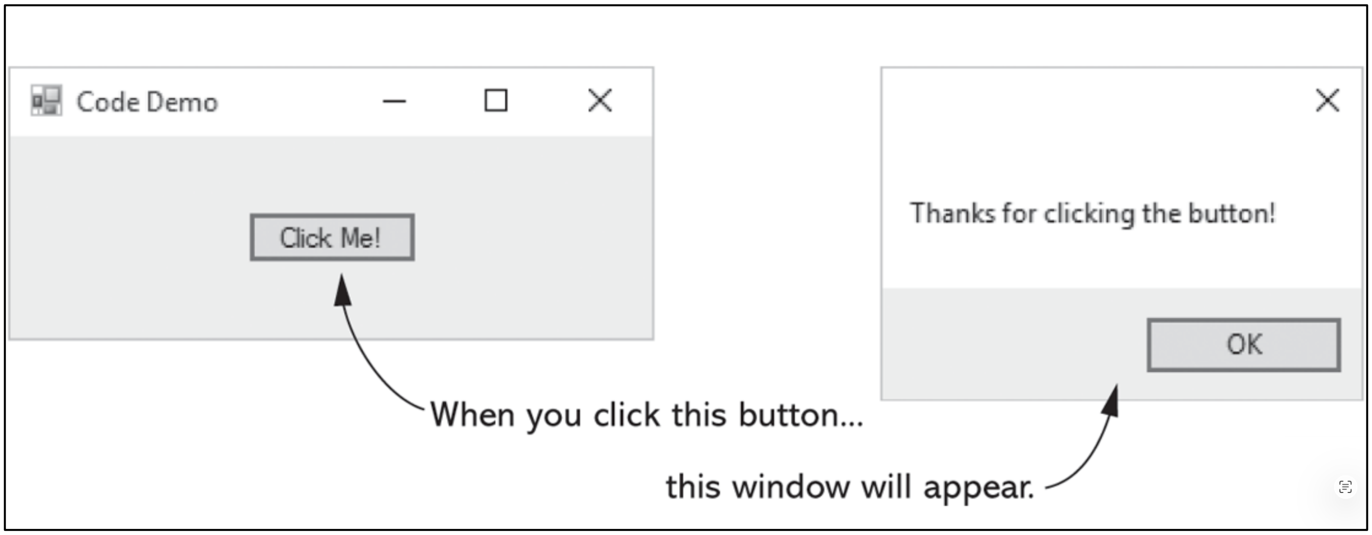
Code Demo | Code_Demo.Form1 | myButton_Click(object se

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

        private void myButton_Click(object sender, EventArgs e)
        {
            MessageBox.Show("Thanks for clicking the button!");
        }
    }
}
```

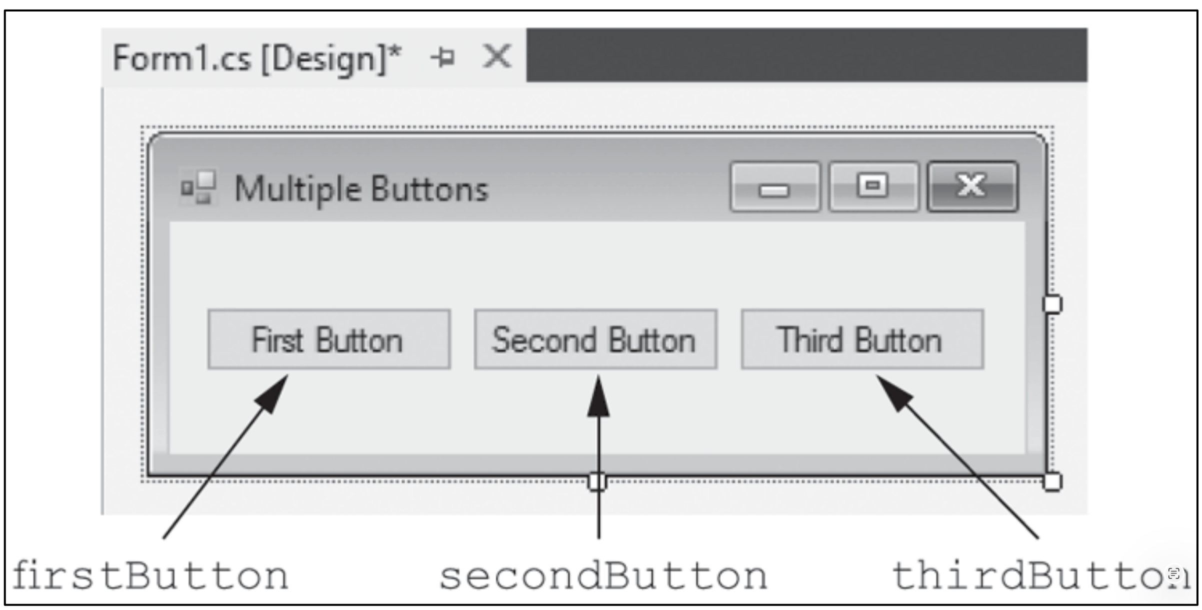
Figure 2-10 The Code Demo project running



Multiple Buttons with Event Handlers

The form shown in Figure 2-11 has three Button controls. The controls are named firstButton, secondButton, and thirdButton.

Figure 2-11 A form with multiple Button controls



Asst. Lect. Ahmed A. Idris
Computer Science Department

To create Click event handlers for the buttons, you simply double-click each Button control in the Designer and an empty event handler will be created in the form's source code file. The names of the Click event handlers will be firstButton_Click, secondButton_Click, and thirdButton_Click. Figure 2-12 shows an example of the form's source code after the three event handlers have been created and a MessageBox.Show statement has been added to each one.

Figure 2-12 Source code with three Click event handlers

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

        private void firstButton_Click(object sender, EventArgs e)
        {
            MessageBox.Show("You clicked the first button.");
        }

        private void secondButton_Click(object sender, EventArgs e)
        {
            MessageBox.Show("You clicked the second button.");
        }

        private void thirdButton_Click(object sender, EventArgs e)
        {
            MessageBox.Show("You clicked the third button.");
        }
    }
}
```

Click event handler for `firstButton`

Click event handler for `secondButton`

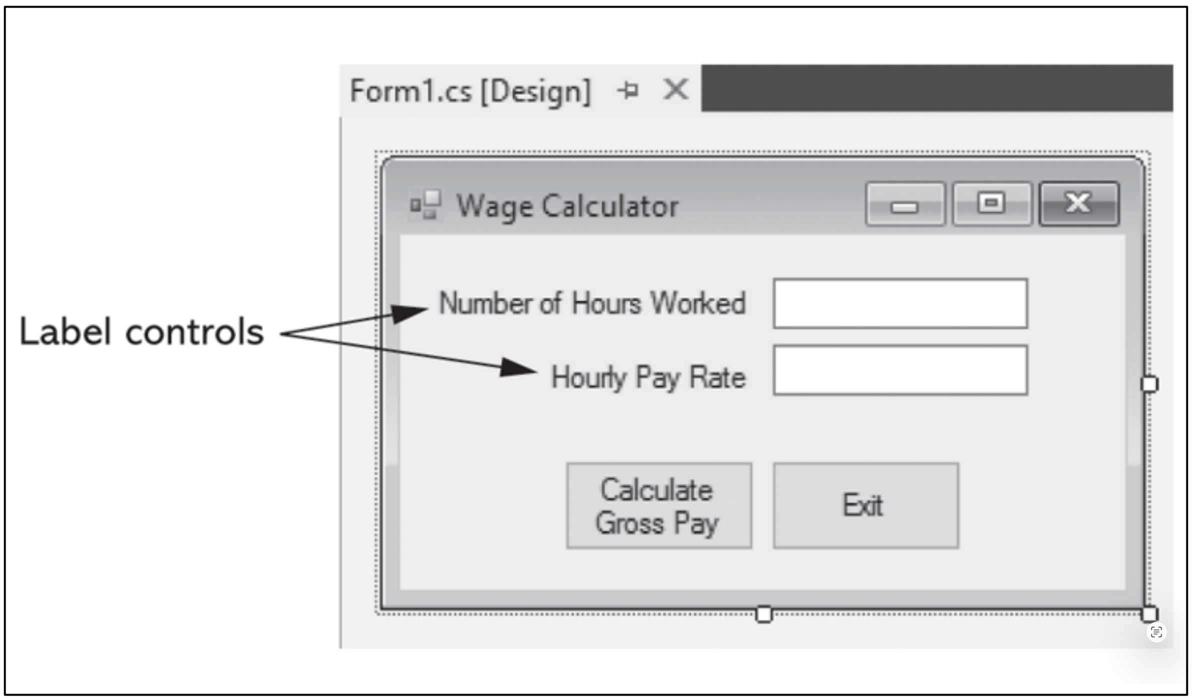
Click event handler for `thirdButton`

Exercise: Build a C# Windows Forms application that displays the message "Hello, World" upon clicking a button labeled "Press Here!".

Label Controls

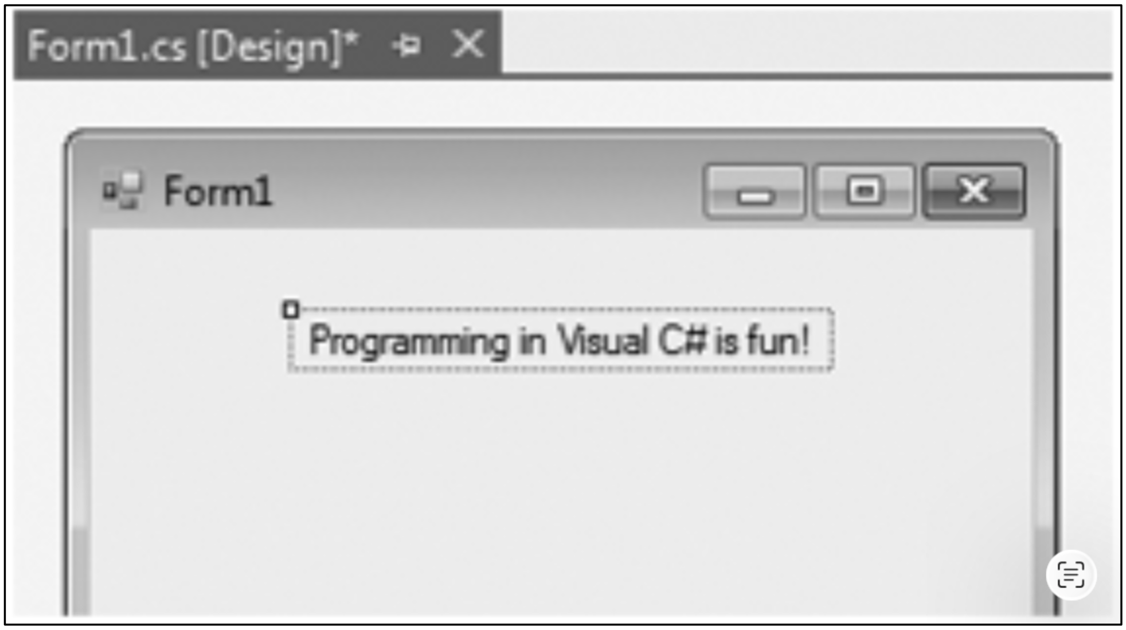
A Label control displays text on a form and used to display unchanging text, or program output. When you want to display text on a form, you use a Label control. Figure 2-13 shows an example of a form with two Label controls.

Figure 2-13 A form with Label controls



When a Label control is created, it is automatically assigned a default name (e.g., label1, label2). Initially, the control's Text property matches its name, causing the label to display that name by default. The Text property can be modified through the Properties window in the Designer to display custom text, such as *"Programming in Visual C# is fun!"*, as shown in the Figure 2-14.

Figure 2-14 A Label control displaying a message



The Font Property

To change the appearance of a Label control's text in C#, you can modify its Font property. This property lets you set the font type, style, and size, as shown in Figure Figure 2-15. By clicking the ellipses button next to the Font property in the Properties window, a Font dialog box opens, allowing you to select the desired attributes shown in the Figure 2-16. Once confirmed, the Label's text updates to reflect the chosen font settings, see the Figure 2-17.

Font: Lucida Handwriting

Font Style: Italic

Size: 10 point

Figure 2-15 The Font property

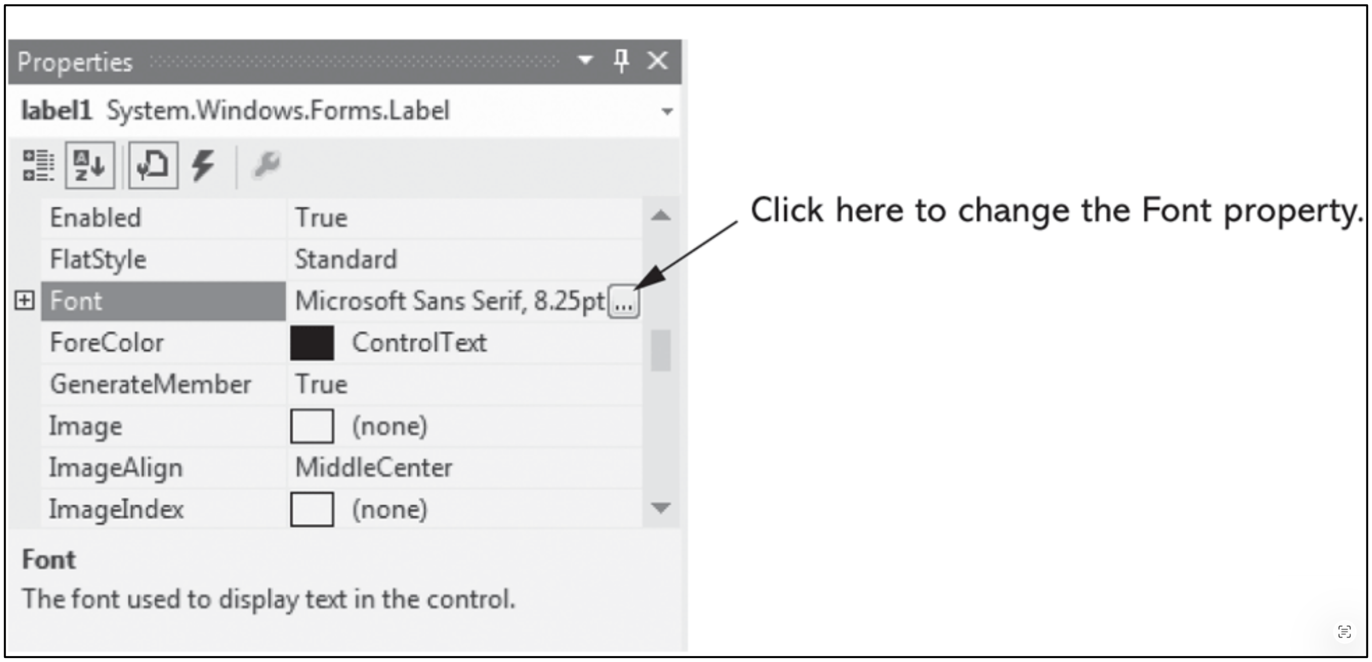


Figure 2-16 The Font dialog box

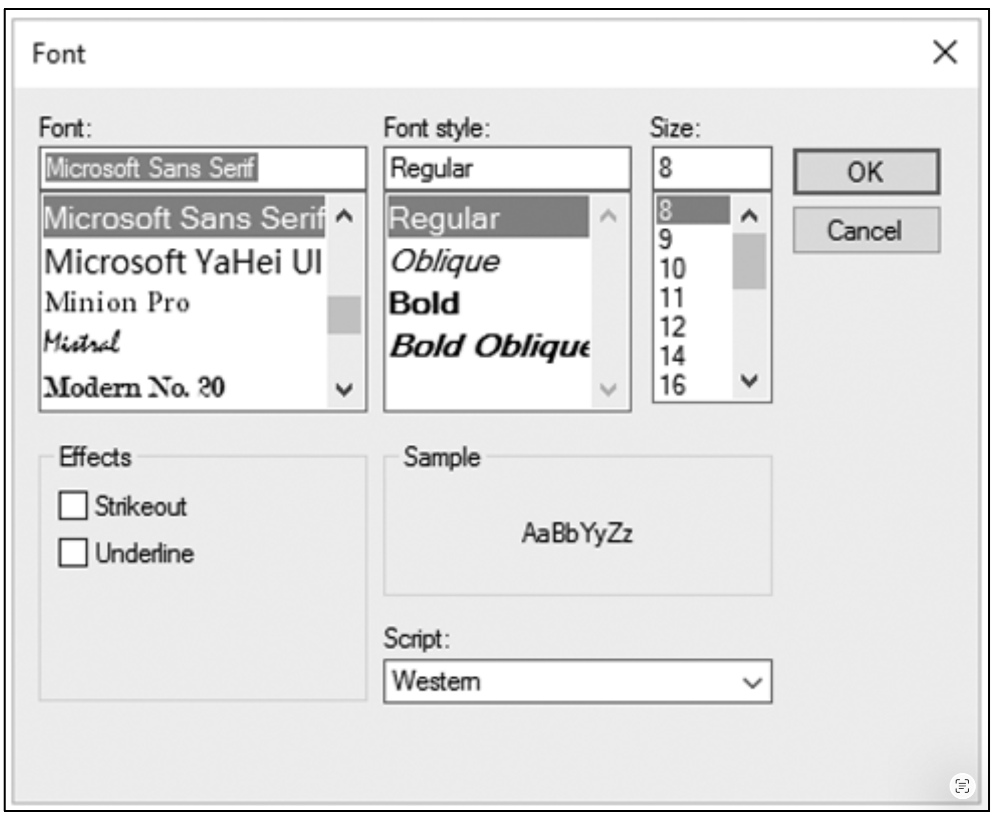
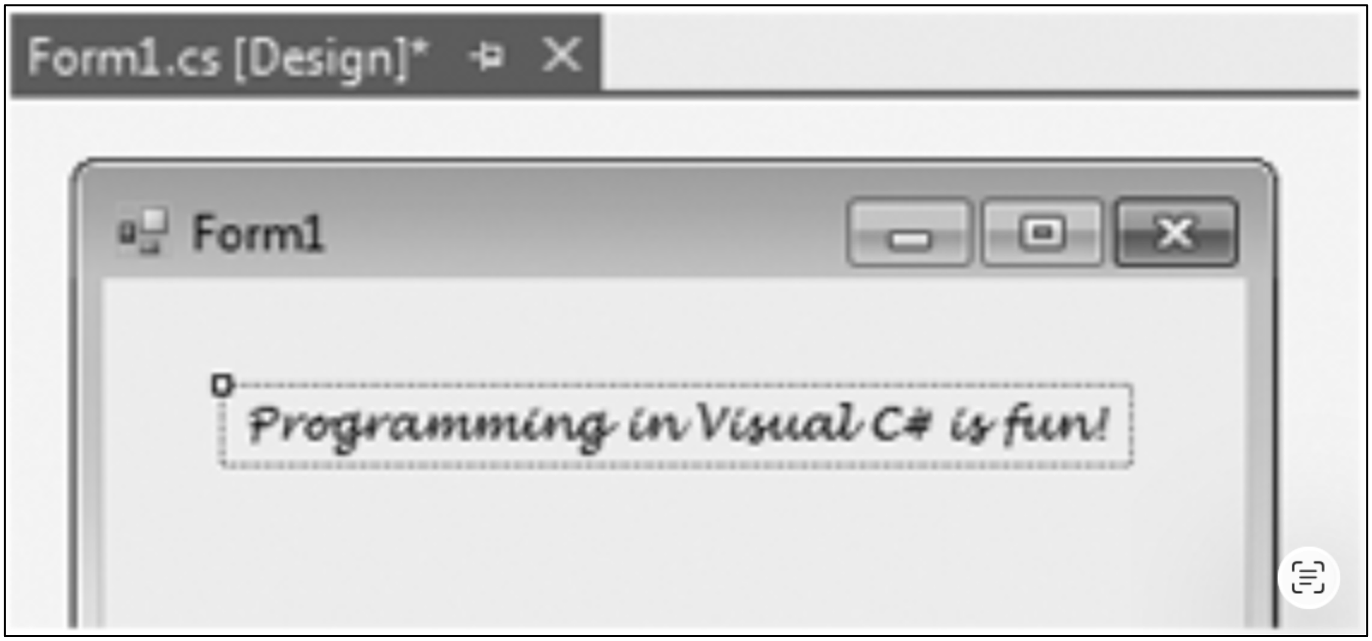


Figure 2-17 A label's appearance with altered font attributes



The BorderStyle Property

The BorderStyle property of a Label control determines the appearance of its border. It can be set to None (no border), FixedSingle (thin border), or Fixed3D (recessed 3D look), see the Figure 2-18. These settings affect how the label appears both in the Designer and at runtime, Figure 2-19.

Figure 2-18 BorderStyle selections

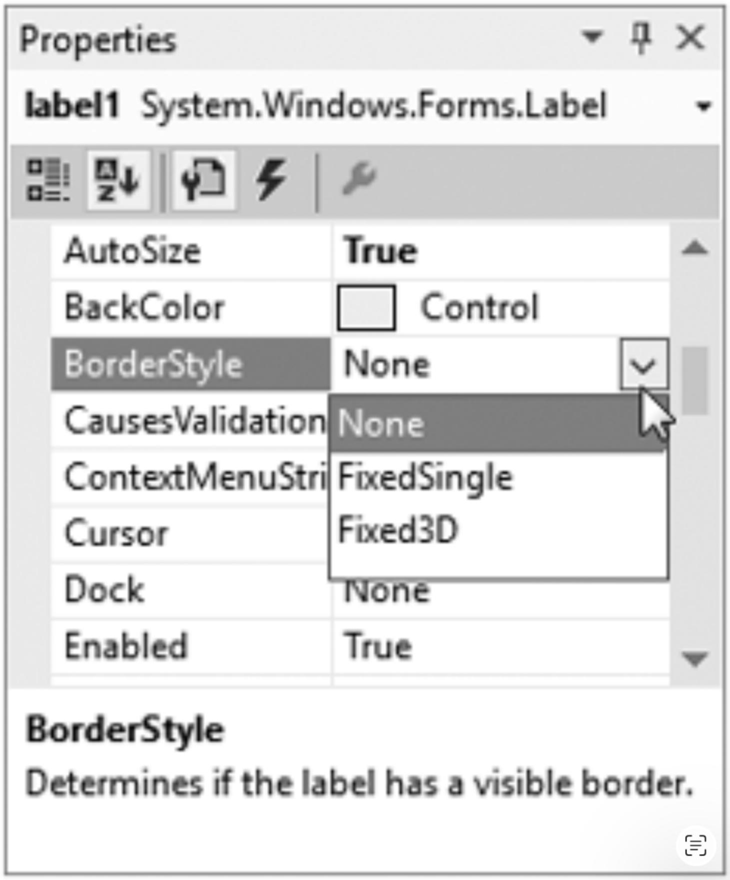
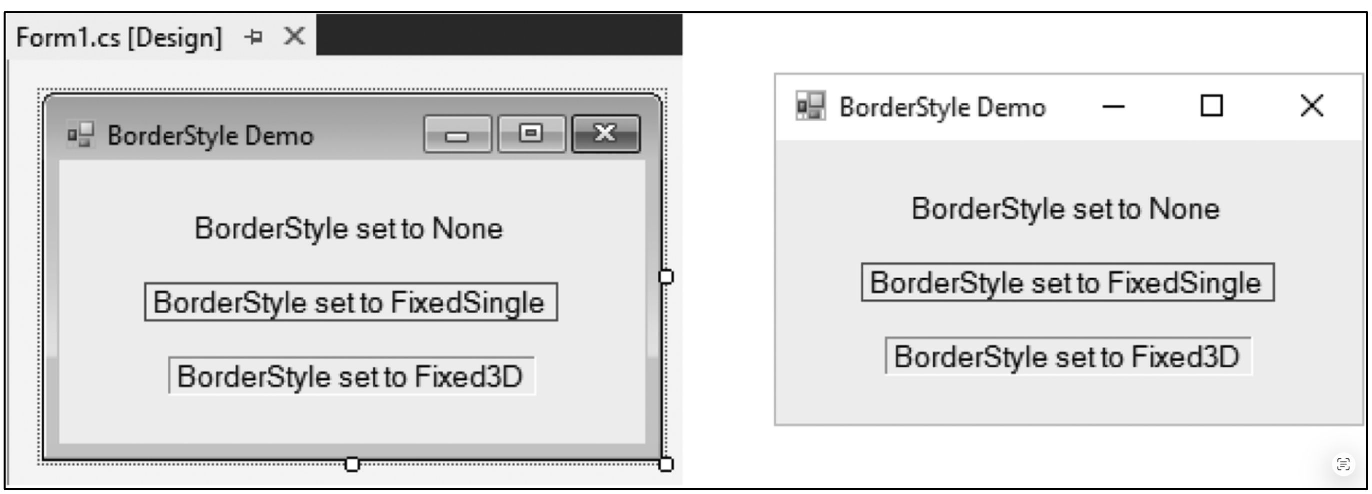


Figure 2-19 BorderStyle examples



Using Code to Display Output in a Label Control

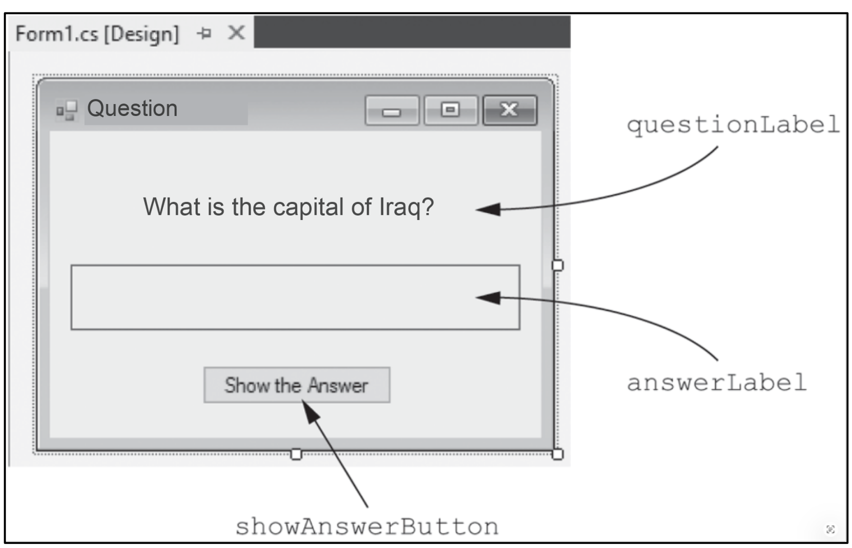
In addition to displaying unchanging text on a form, Label controls are also useful for displaying output while an application is running.

In programming, an assignment statement is used to store a value in a control's property. For instance, in the statement:

```
outputLabel.Text = "Thank you very much";
```

The assignment operator (=) assigns the **string** value on the right to the Text property of the outputLabel control. When executed, this results in the specified text being displayed within the label on the form. Let's create an application that displays a capital city question and reveals the answer when the user clicks a button. As shown in the Figure 2-20.

Figure 2-20 Capital City Question



As shown in the figure, the form has the three controls:

A Label control named **questionLabel**. This label displays the capital question.

A Label control named **answerLabel**. This label initially appears empty, but will be used to display the answer to the capital question.

A Button control named **showAnswerButton**. When the user clicks this button, the answer to the capital question is displayed.

In the code editor, we see the code shown in Figure 2-21.

Figure 2-21 Form1.cs code

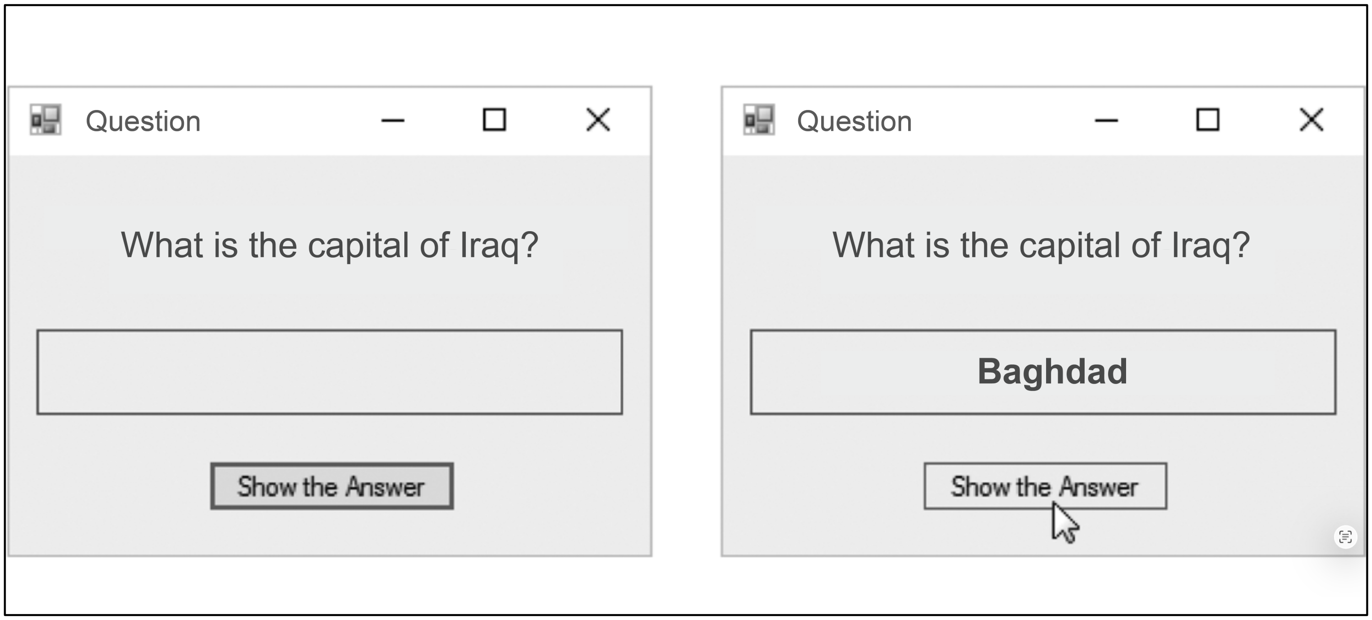
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Capital Question
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void showAnswerButton_Click(object sender, EventArgs e)
        {
            answerLabel.Text = “Baghdad” ;
        }
    }
}
```

When you run the application, the form appears as shown on the left in Figure 2-22. Click the Show the Answer button and the answer to the capital question appears as shown on the right in the figure.

Figure 2-22 The capital application running



The Text Property Accepts Strings Only

The Label control's Text property can accept strings only. You cannot assign a number to the Text property.

The following statement will cause an error because it is attempting to store the number 5 in the resultLabel control's Text property:

```
resultLabel.Text = 5;  ← ERROR!
```

This does not mean that you cannot display a number in a label, however. If you put quotation marks around the number, it becomes a string. The following statement will work:

```
resultLabel.Text = "5";
```

Clearing a Label

In code, if you want to clear the text that is displayed in a Label control, simply assign an empty string ("") to the control's Text property, as shown here:

```
answerLabel.Text = "";
```

Exercise: The Language Translator application, see the Figure 2-23.

Figure 2-23: The Language Translator application