Chapter 3: Using GUI Objects and the Visual Studio IDE
Creating a **Form** in the IDE

- **Forms** are graphical user interface (GUI) objects that provide an interface for collecting, displaying, and delivering information.

- To create a **Form**:
  - Select New Project after starting Visual Studio
  - Then choose Windows Forms Application
  - After you click OK in the New Project window, you see the IDE main window
Creating a **Form** in the IDE (cont’d.)

**Figure 3-11** The Form when it first appears and after a user has entered two integers and clicked the Button
Creating a **Form** in the IDE (cont’d.)
Creating a Form in the IDE (cont’d.)

- The name of the application shows in three places: the title bar, the Solution Explorer, and the Properties window.
- The main menu lies horizontally across the top.
- The Toolbox tab provides a list of controls you can drag onto a Form.
- The Form Designer appears in the center of the screen.
- The Solution Explorer allows viewing and management of project files and settings.
Creating a **Form** in the IDE (cont’d.)

- The **Properties window** is for configuring properties and events on controls in your user interface.
- The **Error list** displays messages about compiler errors in your code.
Creating a Form in the IDE (cont’d.)

• The Program.cs file contains the automatically generated `Main()` method of the application

• Form1.cs contains other automatically generated code
  – The code describes what tasks you will assign to the controls in your application

• When you create a Windows Form project, Visual C# adds a `Form` to the project and calls it `Form1`
Creating a **Form** in the IDE (cont’d.)

- In the Properties window, you can change the appearance, size, and color of a window
- The **Form** class contains around 100 properties
- Property names appear in alphabetical order in groups (except for **Name**)

Microsoft Visual C# 2012, Fifth Edition
Using the Toolbox to Add a **Button** to a **Form**

• When you open the IDE, the left border displays a Toolbox tab
  – Open the toolbox, and a list of tool groups is displayed
  – The tool groups include the controls you have seen when using a Windows application

• You can drag controls (such as a **Button**) onto a **Form**

• You can relocate a control by dragging it, or delete it by selecting it and pressing the Del key
Using the Toolbox to Add a Button to a Form (cont’d.)

• A Button is a clickable object that allows a user to interact with a GUI program

• By default, the first button dragged onto the Form is named button1
  – You should probably change the name to something meaningful

• When you click on a control, the Properties window shows its properties
Using the Toolbox to Add a **Button** to a **Form** (cont’d.)

**Figure 3-5** A Button dragged onto a Form
Using the Toolbox to Add a **Button** to a **Form** (cont’d.)

<table>
<thead>
<tr>
<th>Property Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BackColor</td>
<td>Gets or sets the background color for the control</td>
</tr>
<tr>
<td>Enabled</td>
<td>Gets or sets whether the control is enabled</td>
</tr>
<tr>
<td>Font</td>
<td>Gets or sets the current font for the control</td>
</tr>
<tr>
<td>ForeColor</td>
<td>Gets or sets the foreground color of the control</td>
</tr>
<tr>
<td>Name</td>
<td>Gets or sets the name of the control</td>
</tr>
<tr>
<td>Size</td>
<td>Gets or sets the size of the control</td>
</tr>
<tr>
<td>Text</td>
<td>Gets or sets the text associated with the control</td>
</tr>
<tr>
<td>Visible</td>
<td>Gets or sets a value indicating whether the control is visible</td>
</tr>
</tbody>
</table>

Table 3-2  Selected properties of **Buttons**
Adding functionality to a Button is easy with the IDE

- After dragging the Button onto a Form, you double-click on it to create a method that executes when the user clicks the Button

You can view the code by selecting View and then Code from the main menu
Using the Toolbox to Add a **Button to a Form** (cont’d.)

**Figure 3-6** The automatically generated code after clicking `button1`
Using the Toolbox to Add a Button to a Form (cont’d.)

- You will see many generated statements—some confusing, most of which you can ignore.
- Write code between the curly braces of the `button1_Click()` method.
- If you change the `Name` property of the `button1` object, a subsequently created `Click()` method will have the same name.
- You can write anything you want in the `Click()` methods.
Some examples of tasks that could be performed:

- Perform calculations
- Verify user input
- Display messages
- Read information from or write information to files
- Add, modify, retrieve, or delete database information
- Open other forms
Adding **Labels** and **TextBoxes** to a Form

- **Labels** are controls that you use to display text to communicate with an application’s user.
- You can drag a **Label** onto a **Form**, just like a **Button**.
- You can change its **Text** property to display any text you like.
  - Depending on the amount of text, you may need to resize the label.
- In Figure 3-7, “Enter a number” has been assigned to **labell’s Text** property.
Adding **Labels** and **Textboxes** to a Form (cont’d.)

![Image of Visual Studio with a label on a form](image)

**Figure 3-7** A Label on a Form
• **TextBoxes** are controls through which a user can enter input data in a GUI application

• If a user is supposed to enter data in a TextBox, you frequently want to start with its Text property empty

• When a user types a value into a TextBox, it becomes the value of the Text property
  
  – It is a string by default
  
  – If the user is supposed to enter a number, you’ll have to convert it, just as in a console application
Adding **Labels and TextBoxes** to a Form (cont’d.)

![Image of a form with a label and text box](image)

**Figure 3-8** A Label and TextBox on a form
The form in Figure 3-9 is meant to have the user enter two numbers in the TextBoxes.

When the user clicks the Button, the sum of the two numbers will be displayed in label3.

You must create the code to do the addition.

– It goes in the button1_Click() method.
Adding **Labels** and **TextBoxes** to a **Form** (cont’d.)

**Figure 3-9**  A Form with several controls
Adding **Labels** and **TextBoxes** to a **Form** (cont’d.)

```csharp
private void button1_Click(object sender, EventArgs e)
{
    int num1;
    int num2;
    int sum;
    num1 = Convert.ToInt32(textBox1.Text);
    num2 = Convert.ToInt32(textBox2.Text);
    sum = num1 + num2;
    label3.Text = "Sum is " + sum;
}
```

**Figure 3-10** The `button1_Click()` method that calculates the sum of the entries in two `TextBoxes`
Adding **Labels** and **TextBoxes** to a Form (cont’d.)

![Image of a form with two text boxes and a button]

**Figure 3-11** The Form when it first appears and after a user has entered two integers and clicked the Button.
Understanding Focus and Tab Control

• **Focus** defines which control is active and will accept keyboard input
  – A **Button** that has focus will have a bright line around it
  – A **TextBox** that has focus will have the cursor inside it

• A control can be given focus by:
  – The user clicking on the control
  – The user pressing the Tab key, which will set the focus on the next control as defined by the tab order
• **Tab order** is represented by a sequential number known as the **TabIndex**
  – **TabIndex**es begin with 0
• The default tab order is determined by the order in which the controls are added to the form
• Unless otherwise specified, the first control to receive focus will have **TabIndex 0**
The `TabIndex` for a control can be changed in the Properties list for that control.
Changing a **Label**’s Font

- Every control that can contain text has a `Font` property.

![Font properties](image-url)
The default font is Microsoft Sans Serif

**Proportional fonts versus fixed-pitch fonts, or monospaced fonts**

- The width of proportional font characters varies based on the letter
  - The letter w is wider than the letter i
  - Microsoft Sans Serif is a proportional font

- All fixed-pitch font characters have the same width
  - Courier New is a fixed-pitch font
Naming Forms and Controls

• Provide reasonable Name property values for all the controls you place on a Form
  – Conventionally, start control names with a lowercase letter and use camel casing
  – Start Form names with an uppercase letter
    • A Form is a class, and C# class names conventionally start with an uppercase letter
  – Use the type of object in the name

• To rename a control after you have generated code for it, use the code refactoring feature
Naming **Forms** and Controls (cont’d.)

![Figure 3-12 Choosing the Rename option from the Refactor menu](image)

Microsoft Visual C# 2012, Fifth Edition
Naming **Forms and Controls** (cont’d.)

- When you click Rename, a dialog box opens
  - Type the new method name
- A Preview Changes dialog box will highlight the change
  - Confirm the change by clicking Apply
Naming **Forms** and Controls (cont’d.)

![Image of the Rename dialog box]

**Figure 3-13** The Rename dialog box
Correcting Errors

• As in console-based programs, you will often generate syntax errors
• If you build or run a program with an error, you will see “Build failed” in the lower-left corner of the IDE
• You will also see an error dialog box
  – Always click No in response to “Would you like to continue and run the last successful build?”
• Errors will be listed in the error list at the lower-left corner of the screen
• Double-clicking an error message will take you to the location in the code where the error was found
Correcting Errors (cont’d.)

![DiaLog box that appears when an error occurs during compilation](image)

**Figure 3-14** Dialog box that appears when an error occurs during compilation
Correcting Errors (cont’d.)

![Error List](image)

```
Figure 3-15  The error list
```

```csharp
; expected
22 21 AddTwoNumbers
```
Deleting an Unwanted Event-Handling Method

- When you are working in the Form Designer, it is easy to inadvertently double-click a control and create an event-handling method you don’t want
  - You can leave the event-handling method empty, but that’s considered bad form
  - You cannot just delete the method because, behind the scenes, other code will have been created that refers to the method
  - Instead, go to the Properties window of the control, and then click on its Events button, select the button you want to eliminate, and delete the method name
Deleting an Unwanted Event-Handling Method (cont’d.)

![Deleting an event from the Properties window](image)

**Figure 3-16** Deleting an event from the Properties window
You Do It

• Working With the Visual Studio IDE
• Providing Functionality for a Button
• Adding a Second Button to a Form