How to design a form

Most developers use a tool known as a *GUI builder* to develop forms. As I mentioned earlier, NetBeans includes a popular GUI builder known as the *Swing GUI Builder* that lets you build GUIs using Swing components. In this topic, you'll learn how to use the Swing GUI Builder to design a form for the Future Value application.

How to create a project for a GUI application

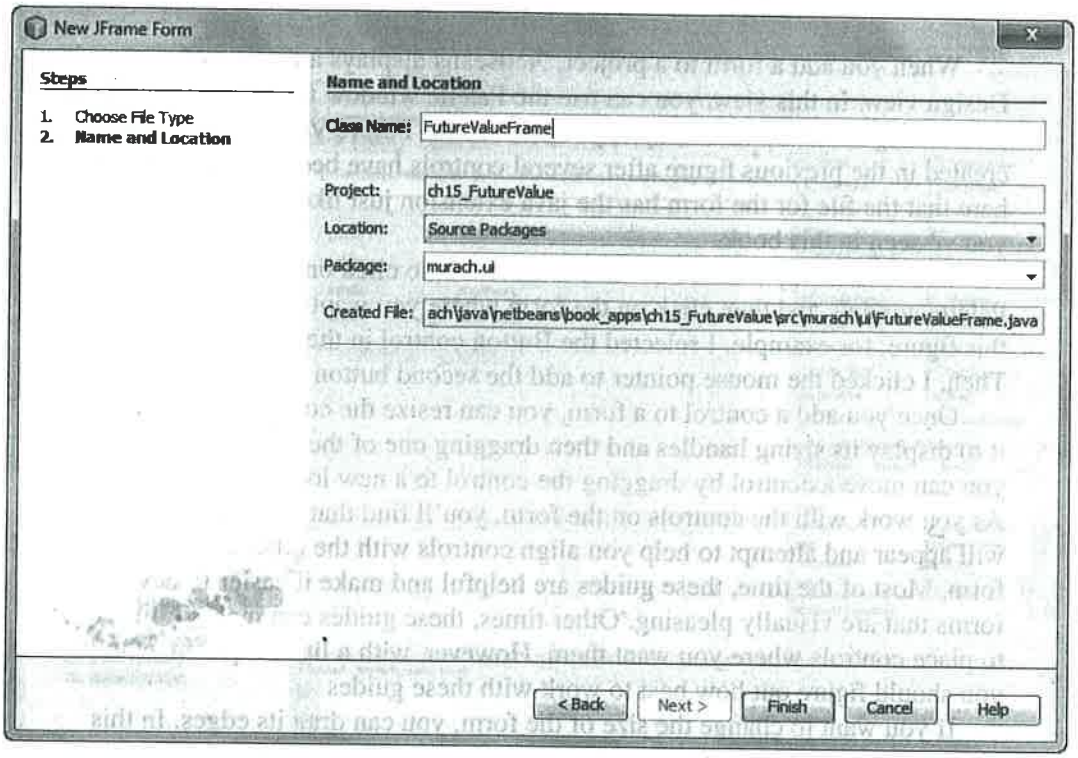
To create a project for a GUI application, you use the same technique you use for any other Java application. When you create a project for a GUI application, though, you should create it without a main class. That's because NetBeans automatically adds a main method to a form that you add to the project. Then, you can use that main method to display the form.

How to add a form to a project

Figure 15-3 shows the dialog box for adding a form to a project. Notice here that the form, which I've named FutureValueFrame, is being stored in a package named murach.ui (ui stands for user interface). When you develop GUI applications, you'll typically store the forms for the application in a separate package. Before you add the first form to a project, then, you should create that package.

For most GUI applications, you base the form on the JFrame class. However, it's also possible to base a form on other classes. If, for example, you want to create a form that can be displayed in a browser, you can base the form on the JApplet class. You'll learn more about this class and how you use it to create applets in chapter 17 of this book.

The New JFrame Form dialog box



Description

- Before you add the first form to a project, you should create a package to hold the forms for the application.
- To add a form to a package, right-click on the package and select the New→JFrame Form command. Then, enter a class name in the resulting dialog box.

Notes

- When you create a project for a GUI application, you should create it without a main class. Then, you can use the class that defines the first form of the application as the main class.
- If the JFrame Form command isn't available from the New menu, the GUI Builder plugin may not be activated. To activate this plugin, select the Tools→Plugins command and display the Installed tab of the resulting dialog box. Then, locate the GUI Builder plugin, select the check box for this plugin, and click the Activate button.

Figure 15-3 How to add a form to a project

How to add controls to a form

When you add a form to a project, NetBeans displays a blank form in Design view. In this view, you can use the Palette window to add controls to the form. In figure 15-4, for example, you can see the Future Value form that was created in the previous figure after several controls have been added to it. Notice here that the file for the form has the java extension just like all the other files you've seen in this book.

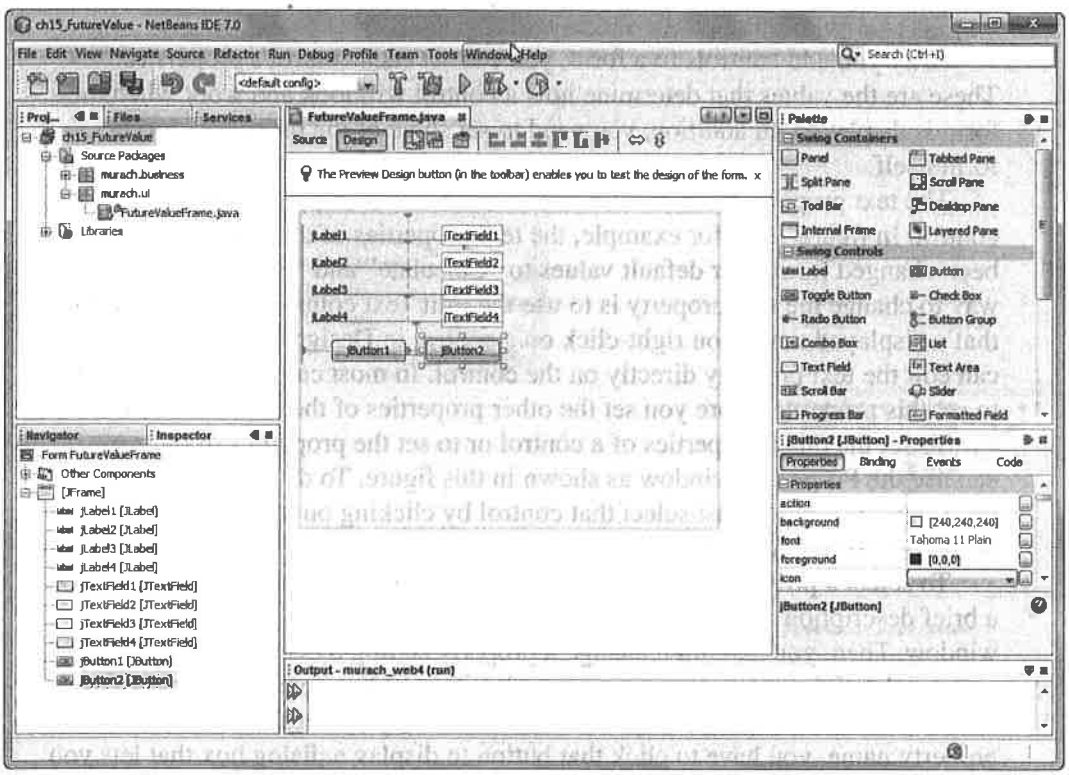
The easiest way to add a control to a form is to click on the control in the Palette window and then click on the form where you want to add the control. In this figure, for example, I selected the Button control in the Palette window. Then, I clicked the mouse pointer to add the second button to the form.

Once you add a control to a form, you can resize the control by clicking on it to display its sizing handles and then dragging one of the handles. Similarly, you can move a control by dragging the control to a new location on the form. As you work with the controls on the form, you'll find that alignment guides will appear and attempt to help you align controls with the other controls on the form. Most of the time, these guides are helpful and make it easier to develop forms that are visually pleasing. Other times, these guides can make it difficult to place controls where you want them. However, with a little experimentation, you should figure out how best to work with these guides.

If you want to change the size of the form, you can drag its edges. In this figure, for example, the form is too tall and too wide for the controls. As a result, it makes sense to drag the bottom and right sides of the form to make it smaller.

If you need to work with several controls at the same time, you can do that by holding down the Ctrl key as you click on the controls. Or, you can click on a blank spot in the form and drag an outline around the controls you want to select. Then, you can work with the selected controls as a group. For example, you can align a group of selected controls by clicking one of the alignment buttons in the toolbar at the top of the form window.

A form after some controls have been added to it



Description

- To open a form, double-click on the .java file for the form in the Projects window.
- You can display a form in two views in NetBeans. Design view shows a graphical representation of the form, and Source view shows the source code for the form.
- To switch between Design view and Source view, click on the Design and Source buttons in the toolbar at the top of the form window.
- To add a control to a form, select the control in the Palette window and then click on the form where you want to place the control. If the Palette window isn't displayed, you can display it by using the Window→Palette command.
- To select a control, click on it. To move a control, drag it. To size a control, select it and drag one of its handles. To change the size of the form, drag its edges.
- To select a group of controls, hold down the Ctrl key as you click on each control. Or, click on a blank spot in the form and drag an outline around the controls.
- To move a group of controls, drag one of them. To align a group of controls, use the buttons in the toolbar at the top of the form window.

Figure 15-4 How to add controls to a form

How to set properties

After you add controls to a form, you can set each control's *properties*. These are the values that determine how a control will look and work when the form is displayed. In addition, you need to set some of the properties for the form itself.

The text property for a control determines what is displayed in or on the control. In figure 15-5, for example, the text properties of the two buttons have been changed from their default values to "Calculate" and "Exit." The easiest way to change the text property is to use the Edit Text command from the menu that's displayed when you right-click on a control in Design view. Then, you can edit the text property directly on the control. In most cases, it makes sense to set this property before you set the other properties of the controls.

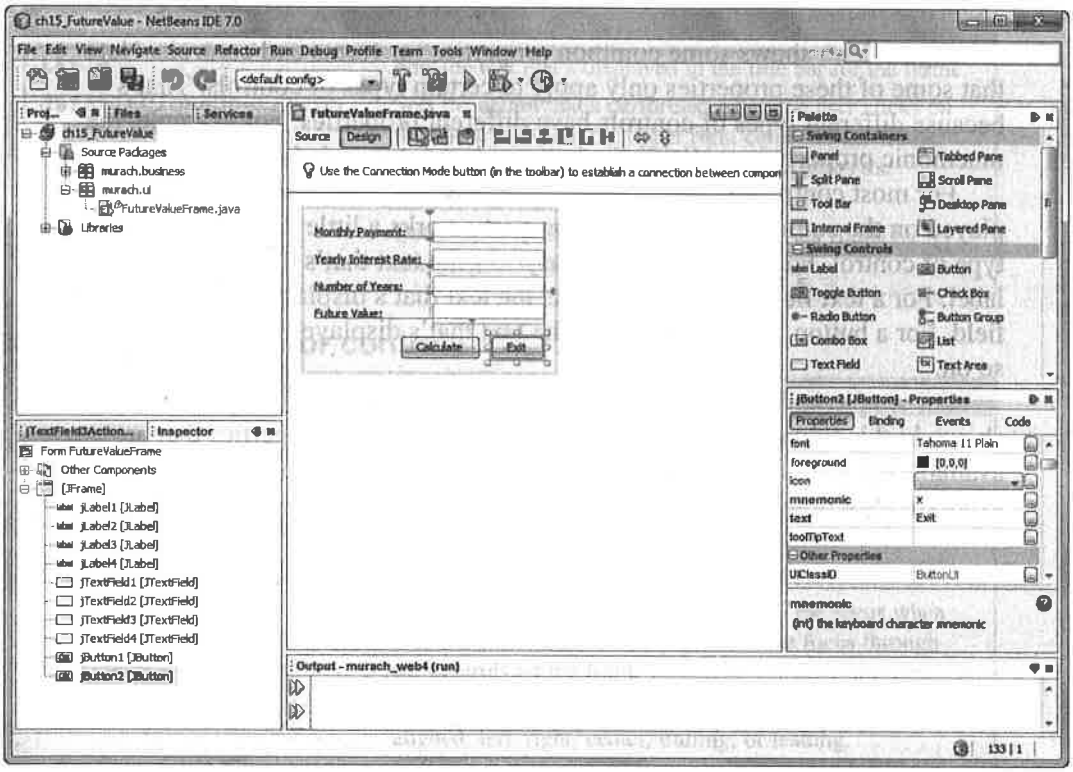
To set the other properties of a control or to set the properties of a form, you can use the Properties window as shown in this figure. To display the properties for a specific control, just select that control by clicking on it. To display the properties for the form, click any blank area of the form.

To select a property in the Properties window, just click on it. When you do, a brief description of that property is displayed at the bottom of the Properties window. Then, you can often change a property setting by entering a new value to the right of the property name or by selecting a value from a combo box or check box. However, if an ellipsis button (...) is displayed to the right of the property name, you have to click that button to display a dialog box that lets you change the property.

As you work with properties, you'll find that most are set the way you want them by default. In addition, some properties are set interactively as you size and position the form and its controls in Design view. As a result, you usually need to change just a few properties for each control.

When you change a property from the default value, the property is displayed in bold in the Properties window. As a result, you can easily identify all properties that you've changed. In this figure, for example, the mnemonic and text properties for the Exit button have been changed from their defaults.

A form after the properties have been set



Description

- To set a property for a control, select the control and then use the Properties window to change the property.
- You can also change the text property for a control by right-clicking the control, selecting the Edit Text command, and entering new text for the control.
- To set a property for more than one control at the same time, select the controls and use the Properties window to change the property.
- To set a property for a form, click outside the controls to select the JFrame object.
- The properties that have been changed from their default values are displayed in bold in the Properties window.
- To sort the properties of a control by category or name, right-click in the Properties window and select the Sort by Category or Sort by Name command.
- To search for a specific property when the focus is in the Properties window, type the starting letter or letters of the property. This starts the Quick Search feature.
- A description of the currently selected property is normally displayed at the bottom of the properties window. If this description isn't shown, right-click in the Properties window and select Show Description Area.

Figure 15-5 How to set properties

Common properties for forms and controls

Figure 15-6 shows some common properties for forms and controls. Note that some of these properties only apply to certain types of controls. That's because different types of controls have different properties. For example, the mnemonic property is available for buttons, but not for labels or text fields.

For most controls, you use the text property to specify the text that's displayed on the control. However, this property works a little differently for each type of control. For a label, this property sets the text that's displayed by the label. For a text field, this property sets the text that's displayed within the text field. For a button, this property sets the text that's displayed on the button. And so on.

For simple applications, you'll probably need to use just the properties shown in this figure. If you want to learn about the other properties that are available for a control, though, you can select the control and then use the Properties window to review its properties.

Common properties for forms

Property	Description
`title`	Sets the text that's displayed in the title bar for the frame.
`defaultCloseOperation`	Sets the action that's performed when the user clicks on the Close button in the upper right corner of the frame. The default value, EXIT_ON_CLOSE, causes the application to exit.
`resizable`	Determines whether the user can resize the frame by dragging its edges.

Common properties for controls

Property	Description
`text`	Sets the text that's displayed on the control.
`editable`	Determines whether the user can edit the text that's stored in the control. Typically used for text fields and other controls that contain text.
`enabled`	Determines whether a control is enabled or disabled.
`focusable`	Determines whether the control accepts the focus when the user presses the Tab key to move the focus through the controls on the form.
`horizontalAlignment`	Determines how the text on a button or in a text field is aligned: left, right, center, trailing, or leading.
`mnemonic`	Specifies a keyboard character that allows the user to quickly access the control by holding down the Alt key and pressing the specified character. Typically used for buttons.
`preferredSize`	Sets the width and height in pixels for the control.

Description

- To learn about the properties that are available for a control, you can select the control, use the Properties window to scroll through its properties, and read the descriptions for each property.

Figure 15-6 Common properties for forms and controls

How to add code to a form

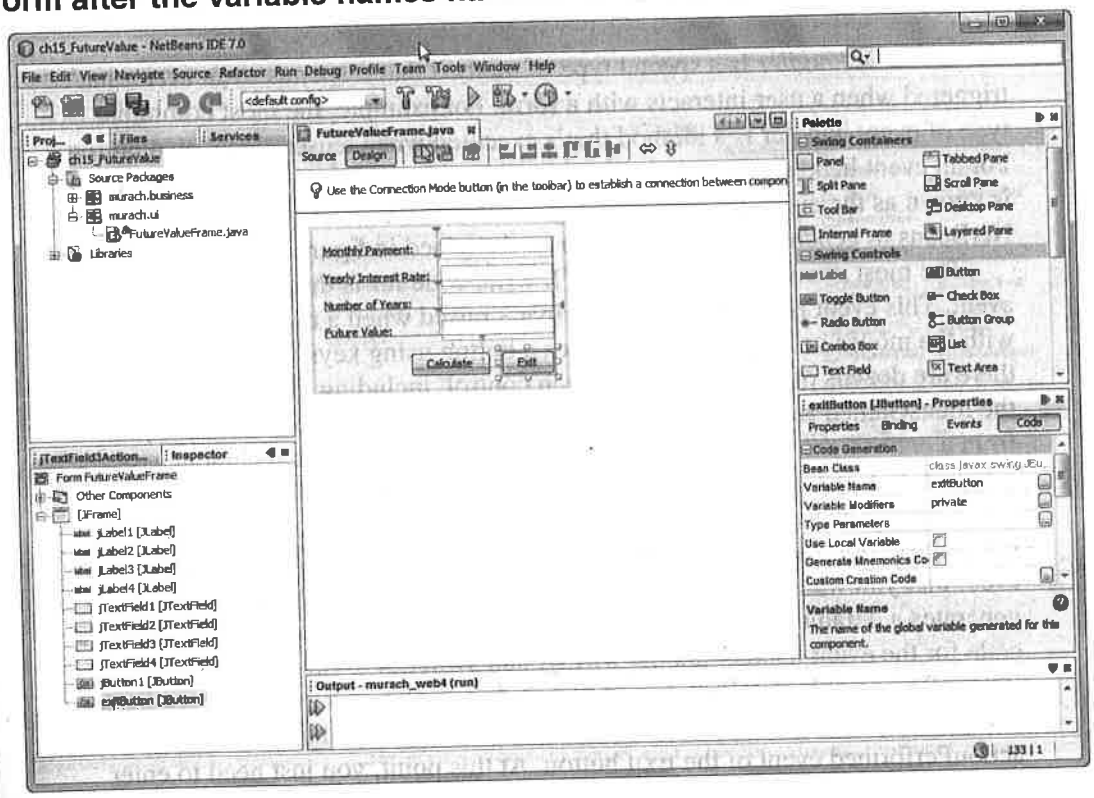
After you add controls to a form and set their properties, you can run the form and it should display properly. However, you won't be able to interact with the controls on the form until you add the required Java code. You'll learn how to do that in the topics that follow. But first, you'll learn how to set the variable names that you'll use to refer to controls in code.

How to set the variable name for a control

When you add a control to a form, NetBeans creates a generic variable name for the control. For example, it uses JTextField1 for the name of the first JTextField control you add to the form, JTextField2 for the name of the second JTextField control you add, and so on. Before you write code that uses any of the controls on a form, you should change these generic names to meaningful names that are easier to remember and use.

As figure 15-7 shows, you can set the variable name for a control using the Properties window. To do that, you select the control in Design view, click on the Code button at the top of the Properties window to access the properties that affect code generation, and then edit the default value for the Variable Name property. In this figure, for example, I changed the default value of JButton2 to exitButton.

A form after the variable names have been set for the controls



Typical variable names for controls

```
paymentTextField
calculateButton
messageLabel
```

Description

- When you add controls to a form, default variable names are given to the controls. If you're going to refer to a control in your Java code, you should change its name so it's easier to remember and use.
- To change the name of the variable that's used for a control, select the control, click the Code button in the Properties window, and then change the Variable Name property.

Figure 15-7 How to set the variable name for a control

How to create an event handler for a control

An *event handler* is a special type of method that responds to an *event* that's triggered when a user interacts with a form. For example, the most common type of event handler is a method that's executed when a user clicks on a button. For an event handler to work, it must be connected, or *wired*, to the event. This is known as the *event wiring*, and it's generated automatically when you use NetBeans to generate an event handler as described in figure 15-8.

The most common event that you'll write code for is the actionPerformed event. This event is a high-level event that's raised when a user clicks a button with the mouse or when a user activates a button using keystrokes. However, there are dozens of other events for each control, including low-level events like the focusGained and focusLost events that occur when the focus is moved to or from a control. As you will see, you can use the same general techniques to work with all types of events.

To create an event handler for a control, you can select the control, click on the Events button at the top of the Properties window, click to the right of the event that you want to handle, and then press the Enter key. Then, NetBeans generates a default name for the method that will handle the event, generates the code for the event handler and its wiring, and switches to Source view. In this figure, for example, NetBeans has generated the name exitButtonActionPerformed for the event handler that will handle the actionPerformed event of the Exit button. At this point, you just need to enter the code for that event handler.

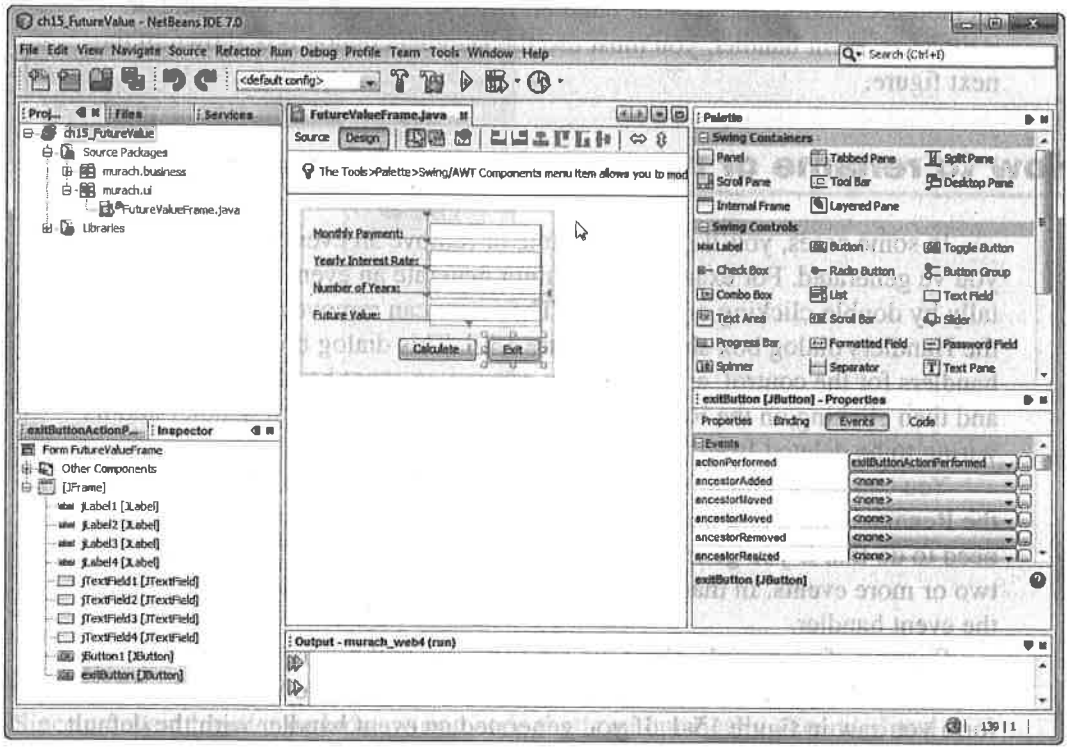
If you don't want to use the generated name for an event handler, you can change it in the Properties window before you press the Enter key. If, for example, you want to create an event handler named textFieldFocusGained for the focusGained event of a text field, you can enter "textFieldFocusGained" to the right of the focusGained event and press the Enter key. Later, if you want to wire another text field to the event handler named textFieldFocusGained, you can enter that event handler name in the Properties window for the focusGained event of the text field. That way, the same event handler will be used for the same event of two different text fields.

When you generate event handlers, you should realize that they're added to your code in the sequence in which you generate them. Unfortunately, NetBeans doesn't let you change that sequence. If you want the event handlers for an application to appear in a specific sequence, then, you will need to generate them in that sequence.

Below the screen in this figure, you can see the code for the event handler that's generated for the Exit button on the Future Value form. Here, the name of the method is the name of the control (exitButton) followed by the name of the event (ActionPerformed). This shows that it's important to set the variable name for the control before you start an event handler for a control. That way, the variable name will be reflected in the name of the event handler.

This figure also shows the wiring for the event handler that's generated for the Exit button. This wiring is stored in a generated region of code, and you

The actionPerformed event for the Exit button



The code that's generated for the actionPerformed event of the Exit button

```
private void exitButtonActionPerformed(java.awt.event.ActionEvent evt) {
    // TODO add your handling code here:
}
```

The generated code that wires the event handler to the event

```
exitButton.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        exitButtonActionPerformed(evt);
    }
});
```

Description

- To create an event handler for the default event of a control, double-click on the control.
- To create an event handler for other events, select the control, click the Events button in the Properties window, and select the name for the event handler from the combo box list.
- To create an event handler with a custom name, select the control, click the Events button in the Properties window, click in the combo box for the event that you want to handle, enter a custom name for the event handler, and press the Enter key.
- To wire an event to an existing event handler, enter the name of the event in the combo box for the event that you want to handle.

Figure 15-8 How to create an event handler for a control

can't edit it manually. In addition, you can't manually rename or remove the method declaration for an event handler. Instead, if you need to rename or remove an event handler, you must use the Handlers dialog box shown in the next figure.

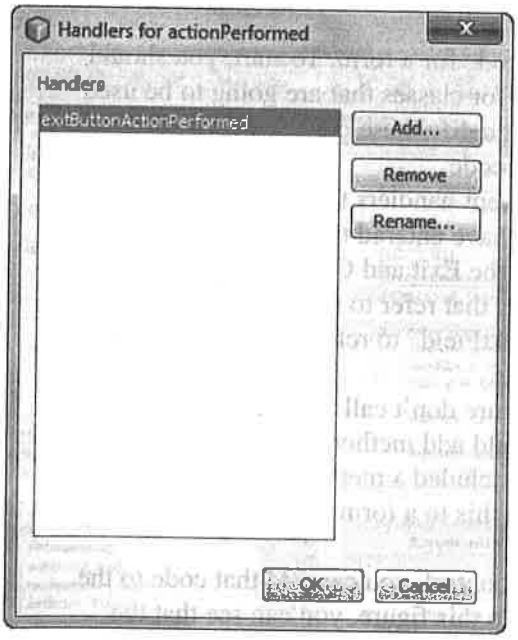
How to rename or remove an event handler

In some cases, you'll need to rename or remove an event handler that you've generated. For example, you might generate an event handler accidentally by double-clicking on a control. Then, you can remove the handler using the Handlers dialog box shown in figure 15-9. This dialog box lists all the event handlers for the control, and it lets you remove an event handler by selecting it and then clicking on the Remove button. This causes the event handler and its wiring to be deleted from the source code for the form.

You can use a similar technique to rename an event handler, but you click the Rename button and then enter a new name for the event handler. You may need to do that if you generate an event handler that you want to use to handle two or more events. In that case, you'll want the name to reflect the purpose of the event handler.

Suppose, for example, that you want to execute the same code for the focusGained event of the first three text fields on the Future Value Calculator form you saw in figure 15-1. If you generated an event handler with the default name for the first text field, the event handler would be named monthlyPaymentTextFieldFocusGained. Then, you could rename this event handler to something like textFieldFocusGained to make it clear that it's used for all three text fields. Of course, you could also create the event handler with a custom name to begin with so you wouldn't have to rename it.

The Handlers dialog box



Description

- Since an event handler includes wiring code that's automatically generated and stored in a different location than the event handler code, you should always use the Handlers dialog box if you need to rename or remove an event handler. That way, both the event handler and its wiring are updated or removed in a single operation.
- To rename or remove an event handler, select the control, click on the Events button in the Properties window, and click on the ellipsis button to the right of the event that you want to work with. Then, use the Handlers dialog box to remove or rename the event handler.

Figure 15-9 How to rename or remove an event handler

How to enter the code for a form

Figure 15-10 shows how to enter the code for a form. To start, you should enter any import statements that you need for classes that are going to be used by the form. That way, you won't have to qualify these classes with their package names when you refer to them in code.

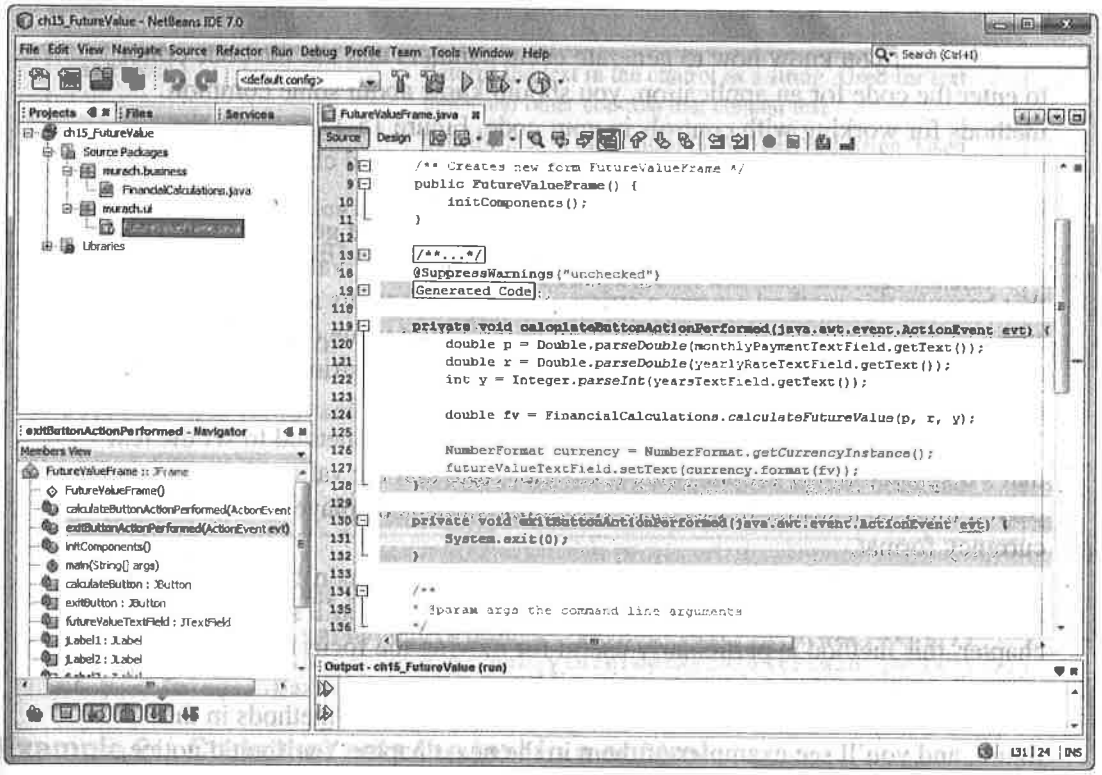
Next, you can enter the code for the event handlers that you've generated for the form. In this figure, for example, I have entered the code for the event handlers for the actionPerformed event of the Exit and Calculate buttons. As I entered the code, I used the variable names that refer to the text field controls. For example, I used the name "paymentTextField" to refer to the entry in the Monthly Payment text field.

Although the event handlers in this figure don't call any other methods, you can add methods to a form just as you would add methods to any other class. For example, this application could have included a method that calculates the future value. When you add a method like this to a form class, you typically place it near the event handlers that call it.

If you need to use code to initialize a control, you can add that code to the end of the constructor for the form class. In this figure, you can see that the constructor contains a single statement that calls the initComponents method. This method contains generated code that creates the controls on the form and sets the properties for the form and its controls so they're displayed correctly. If you want to set any additional properties, you can add that code after the call to the initComponents method. For example, you might want to load the items that will be displayed in a list box or combo box. You'll learn more about these controls in the next chapter.

The Generated Code region of a form contains the code that's generated as you create the form in Design view. Since you can't edit this code directly, you rarely need to view it. If you're curious, however, you can review this code to see how it works. To do that, just expand this region by clicking on the plus sign (+) that's displayed to its left.

The source code after two event handlers have been coded



Description

- You can use the code editor to enter the code for an event handler just as you would enter any other code.
- To refer to the controls on the form, you use the variable names that you assigned to the controls. You can review these variable names in the Navigator window for the frame. You can also review the declarations for these variables at the end of the source code for the form.
- To import a class that will be used by the form, code an import statement after the package statement that's at the start of the form class just like you would for any other class.
- If you need to initialize a control, you can add code to the constructor for the class after the call to the initComponents method.
- You can also add methods other than event handlers to a form. When you do that, you typically place the methods following the event handlers that call them.
- NetBeans shades all generated code, and you can't manually edit this code. If you need to change this code, use Design view as described in previous topics.

Figure 15-10 How to enter the code for a form

Common methods for controls

Now that you know how to generate event handlers and use the code editor to enter the code for an application, you should know about some common methods for working with controls in your code. Figure 15-11 presents the methods that are used in the applications in this chapter and chapter 16. It also shows some examples that use some of these methods.

The first example includes three statements that use the getText method to get the text from the three enabled text fields on the Future Value form. Notice that because this method returns a string, all three statements must convert the string to a numeric value so it can be used in the future value calculation. Here, the first two strings are converted to doubles and the third string is converted to an int.

The statement in the second example uses the setText method to set the text that's displayed in the Future Value text field. Note that the value that's displayed must be a string. In this case, the future value is displayed using the currency format.

The statement in the last example uses the requestFocusInWindow method to move the focus to the Monthly Payment text field. As you'll see later in this chapter, this method is particularly useful for moving the focus to a control when the user enters a value that isn't valid.

You shouldn't have any trouble understanding the other methods in this figure, and you'll see examples of them in the next chapter. You should notice, though, that three of these methods—setEditable, setEnabled, and setFocusable—have property counterparts that you saw in figure 15-6. When I designed the Future Value form, for example, I set the editable property of the Future Value text field so the user can't edit the text in this field. Another way to do that would be to use the setEditable method in the constructor for the Future Value form like this:

```
futureValueTextField.setEditable(false);
```

Since the editable property doesn't change as the program executes, though, it's easier to set the editable property as you're designing the form. In contrast, if a property like this will change as the program executes, you can use the related method to change the setting in code.

Common methods for controls

Method	Description
`getText()`	Returns the text in the control as a string. Used for text fields and other controls that contain text.
`setText(String)`	Sets the text in the control to the specified string. Used for text fields and other controls that contain text.
`requestFocusInWindow()`	Moves the focus to the control.
`setEditable(boolean)`	If the boolean value is true, the control is editable. Otherwise, it's not. Used for text fields and other controls that contain text.
`setEnabled(boolean)`	If the boolean value is true, the control is enabled so the user can interact with it. Otherwise, it's disabled.
`setFocusable(boolean)`	If the boolean value is true, the control can receive the focus. Otherwise, it can't.
`selectAll()`	Selects all the text in a control. Used for text fields and other controls that contain text.

Example 1: Code that gets the text from three controls

```
double p = Double.parseDouble(monthlyPaymentTextField.getText());
double r = Double.parseDouble(yearlyInterestRateTextField.getText());
int y = Integer.parseInt(yearsTextField.getText());
```

Example 2: Code that sets the text in a control

```
futureValueTextField.setText(currency.format(futureValue));
```

Example 3: Code that moves the focus to a control

```
monthlyPaymentTextField.requestFocusInWindow();
```

Description

- To learn more about the methods that are available for a control, refer to the API documentation for the control.
- The getText, setText, setEditable, and selectAll methods are defined by the JTextComponent class. The setEnabled and requestFocusInWindow methods are defined by the JComponent class. And the setFocusable method is defined by the Component class.

Figure 15-11 Common methods for controls

How to display and center a form

Figure 15-12 starts by showing the main method that's generated by NetBeans for a form. Although this code may seem complicated, most of it works the way you want. As a result, you typically only need to make some minor changes to the statements within the run method. In this figure, for example, the generated code creates a new form and displays it using a single statement like this:

```
new FutureValueFrame().setVisible(true);
```

This displays the form in the upper left corner of the screen.

If you want to display the form in the center of the screen, you can do that as shown in the second example. Here, the run method starts by creating a FutureValueFrame object and assigning it to a variable named frame. Then, the setVisible method of that frame is used to display the frame, and the setLocationRelativeTo method is used to center the frame on the screen.

Most of the time, that's all you need to know about displaying a form. If you're curious about how the rest of this generated code works, though, you can review the notes at the bottom of this figure along with the information in the related chapters.

Two methods for displaying a form

Method	Description
`setVisible(boolean)`	Shows this component if the boolean value is true. Otherwise, this method hides the component.
`setLocationRelativeTo(component)`	Sets the location of this component relative to the specified component. If the component is null, this method centers the frame on the screen. Otherwise, it centers the frame on the specified component.

A main method for a form that's generated by NetBeans

```
public static void main(String args[]) {
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            new FutureValueFrame().setVisible(true);
        }
    });
}
```

A main method for a form that displays and centers the form

```
public static void main(String args[]) {
    java.awt.EventQueue.invokeLater(new Runnable() {
        public void run() {
            FutureValueFrame frame = new FutureValueFrame();
            frame.setVisible(true);
            frame.setLocationRelativeTo(null);
        }
    });
}
```

Description

- By default, NetBeans generates a main method for a form that creates the form and displays it in the upper left corner of the user's screen.
- You can modify this code to display the form in the center of the screen as shown in the second example above.

Notes

- The code that's generated by NetBeans for a form creates a new *thread* for the form, which is a single flow of execution through the program. To create this thread, the code creates a new class that implements the Runnable interface. This interface has a single method named run. For more information about working with threads, see chapter 22.
- The class that implements the Runnable interface is coded within the invokeLater method of the EventQueue class. As a result, this class is a type of inner class as described in chapter 10. Since this inner class doesn't have a name, it's known as an *anonymous class*.
- The invokeLater method of the EventQueue class puts the thread for the form in a queue. Then, it runs the thread when the thread reaches the front of the queue.

Figure 15-12 How to display and center a form

How to validate Swing input data

In chapter 5, you learned how to validate input data for console applications. Then, in chapter 7, you saw an example of a Validator class that uses static methods to help validate data for console applications. In the topics that follow, you'll see that you can use similar techniques to validate the input data for Swing applications.

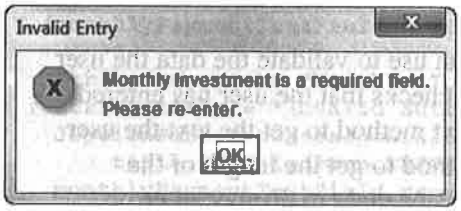
How to display error messages

When you validate data in a Swing application, you need to be able to display an error message to inform the user that an invalid entry has been detected. The easiest way to do that is to display the error message in a separate dialog box as shown at the top of figure 15-14. To display a dialog box like this, you use the showMessageDialog method of the JOptionPane class as described in this figure.

The four parameters accepted by this method are the parent component that determines the location of the dialog box, the message displayed in the dialog box, the title of the dialog box, and the message type, which determines the icon that's displayed in the dialog box. The parent component can be the control that's being validated or the frame that contains the control. It can also be null, in which case the dialog box is centered on the screen.

The code example in this figure shows how to display a simple error message. To start, the first statement defines a string for the message to display in the dialog box, and the second statement defines a string for the title of the dialog box. Then, the third statement calls the showMessageDialog method. Here, the parent component argument is set to the this keyword, which causes the dialog box to be displayed within the current frame. For that to work, this code must appear in a class that inherits the Component class or a class derived from it, such as JFrame or JApplet. Finally, the message type argument is set to the ERROR_MESSAGE field of the JOptionPane class. This causes the dialog box to display an error icon.

An error message displayed in a JOptionPane dialog box



The showMessageDialog method of the JOptionPane class

Syntax

```
showMessageDialog(parentComponent, messageString,
                  titleString, messageTypeInt);
```

Arguments

Argument	Description
`parent`	An object representing the component that's the parent of the dialog box. If you specify null, the dialog box will appear in the center of the screen.
`message`	A string representing the message to be displayed in the dialog box.
`title`	A string representing the title of the dialog box.
`messageType`	An int that indicates the type of icon that will be used for the dialog box. You can use the fields of the JOptionPane class for this argument.

Fields used for the message type parameter

Icon displayed	Field
(none)	PLAIN_MESSAGE
	INFORMATION_MESSAGE
	WARNING_MESSAGE
	ERROR_MESSAGE
	QUESTION_MESSAGE

Code that displays the dialog box shown above

```
String message = "Monthly Investment is a required field.\n"
               + "Please re-enter.";
String title = "Invalid Entry";
JOptionPane.showMessageDialog(this,
    message, title, JOptionPane.ERROR_MESSAGE);
```

Description

- The showMessageDialog method is a static method of the JOptionPane class that is commonly used to display dialog boxes with error messages for data validation.
- To close a dialog box, the user can click the OK button or the close button in the upper right corner of the dialog box.
- You can also use the JOptionPane class to accept input from the user. For more information, see the API documentation for this class.

Figure 15-14 How to display error messages

How to validate the data entered into a text field

Figure 15-15 shows two techniques you can use to validate the data the user enters into a text field. The first code example checks that the user has entered data into the field. To do that, it uses the getText method to get the text the user entered as a string. Then, it uses the length method to get the length of the string. If the length is zero, it uses the JOptionPane class to display an error message in a dialog box. Then, it calls the text field's requestFocusInWindow method to move the focus to the text field after the user closes the dialog box.

The second example shows how to check that the user entered a numeric value. Here, the parseDouble method of the Double class is used to parse the text entered by the user to a double value. This conversion is placed within a try statement. Then, if a NumberFormatException occurs, the catch block catches the exception, displays an error message, and moves the focus to the text field.

Example 1: Code that checks if an entry has been made

```
if (monthlyPaymentTextField.getText().length() == 0)
{
    String message = "Monthly Investment is a required field.";
    String title = "Invalid Entry";
    JOptionPane.showMessageDialog(this, message,
        title, JOptionPane.ERROR_MESSAGE);
    monthlyPaymentTextField.requestFocusInWindow();
    return;
}
```

Example 2: Code that checks if an entry is a valid number

```
try
{
    double d = Double.parseDouble(monthlyPaymentTextField.getText());
}
catch (NumberFormatException e)
{
    String message = "Monthly Investment must be a valid number.";
    String title = "Invalid Entry";
    JOptionPane.showMessageDialog(this, message,
        title, JOptionPane.ERROR_MESSAGE);
    monthlyPaymentTextField.requestFocusInWindow();
    return;
}
```

Description

- Like console applications, GUI applications should validate all data entered by the user before processing the data.
- When an entry is invalid, the application can display an error message and give the user another chance to enter valid data.
- To test whether a value has been entered into a text field, you can use the getText method of the text field to get a string that contains the text the user entered. Then, you can check whether the length of that string is zero by using its length method.
- To test whether a text field contains valid numeric data, you can code the statement that converts the data in a try block and use a catch block to catch a NumberFormatException.

Figure 15-15 How to validate the data entered into a text field