CHAPTER 17

Applets, Images, and Sound

In this chapter, you will:

- Learn about applets
- Write an HTML document to host an applet
- Use the `init()` method
- Work with `JApplet` components
- Understand the `JApplet` life cycle
- Understand multimedia and use images
- Add sound to `JApplets`

Introducing Applets

An **applet** is a program that is called within another application—often a Web browser. It can contain any number of components, such as buttons, text fields, and pictures, and it can respond to user-initiated events, such as mouse clicks or keyboard presses. Many of an applet's behaviors come from methods that reside in a Java class named `JApplet`—the programmer can create additional behaviors. The name *applet* means "little application." An applet is "little" in that it is not a full-blown program; it relies on other programs to execute it.

A Java applet is like a Java application in several ways:

- You save both applets and applications with a .java file extension.
- You compile both applets and applications into bytecode using the `javac` command, and the bytecode is stored in a file with a .class file extension.
- Both applets and applications can contain any number of methods you define and any number of variables and constants. Both can contain decisions, loops, arrays, and all the other language elements you have learned about in Java.
- Both can (and applets almost always do) contain GUI elements such as buttons and labels, and event listeners that respond to user-initiated actions.

An applet also is different from an application in several ways:

- Unlike applications, applets descend from the `JApplet` class.
- Unlike applications, applets run from another application. You do not use the `java` command to execute an applet.
- Unlike applications, applets do not contain a `main()` method. In this chapter, you will learn about the methods every applet contains.
- Unlike an application that uses a `JFrame`, you do not set a default close operation for a `JApplet`.
- Applets cannot delete, read, or create files on the user's system.
- Applets cannot run any other program on the user's system.

Understanding the `JApplet` Class

`JApplet` is a `Swing` class from which you can inherit. A `JApplet` is a `Component`, and it is also a `Container`. Figure 17-1 shows the relationship of `JApplet` to its ancestors. When you create a `JApplet`, you gain access to over 200 methods through inheritance.

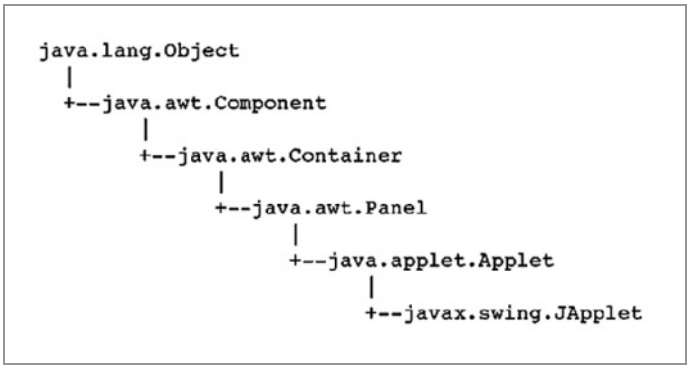

Figure 17-1 Inheritance hierarchy of the `JApplet` class

You can import the `JApplet` class to a program by using one of the following statements:

```
import javax.swing.JApplet;
import javax.swing.*;
```

Running an Applet

To view an applet, it must be called from another document; frequently the document is written in HTML. **HTML**, or **Hypertext Markup Language**, is a simple language used to create Web pages for the Internet. HTML contains many commands that allow you to format text on a Web page, import graphic images, and link your page to other Web pages. You run applets within a page on the Internet, an intranet, or a local computer.

The current version of HTML is **Extensible Hypertext Markup Language (XHTML)**. The differences are minor, although XHTML is stricter. Most Web developers are continuing to use HTML until XHTML matures and some compatibility issues are resolved. You can find more information and tutorials on both HTML and XHTML at *www.w3schools.com*.

You can run an applet in one of two ways:

- You can run an applet in a Web browser, such as Internet Explorer, Firefox, Opera, or Safari. To do so, you open the applet-hosting HTML document in the browser. For example, in Internet Explorer, you click File on the menu bar, click Open, and type the complete path for the HTML document that you created. After you press Enter, the applet appears on your screen. The applet can be run within a page on the Internet or an intranet, but you do not have to connect to the Internet to run an applet in your browser—you can simply use the browser locally.
- You can also view your applet on a local computer by using the **Applet Viewer**, which is a program that comes with the Java Development Kit (JDK) that provides a convenient environment in which to test your applets. To use the Applet Viewer, type the **`appletviewer` command** at the command line, followed by the full HTML filename,

including the extension. When you press Enter, the Applet Viewer window opens and displays the applet.

When you save an HTML file, you can use .html or .htm as a file extension. In some older versions of DOS and Windows, filename extensions were not allowed to be more than three characters. Current Web browsers and servers accept files with .htm and .html extensions. Examples in this book use the four-character .html extension.

TWO TRUTHS & A LIE

Introducing Applets

1. An applet can contain any number of components, such as buttons, text fields, and pictures, and can respond to user-initiated events, such as mouse clicks or keyboard presses.
2. Applets and applications are similar in that both have .java file extensions and you compile both with the `javac` command.
3. Applets and applications are different in that applets do not use loops and arrays.

The false statement is #3. Applets use all the language constructs in Java.

Writing an HTML Document to Host an Applet

When you create an applet, you do the following:

- Write the applet in Java and save it with a .java file extension, just as when you write a Java application.
- Compile the applet into bytecode using the `javac` command, just as when you write a Java application.
- Write an HTML document that includes a statement to call your compiled Java class.
- Load the HTML document into a Web browser (such as Internet Explorer) or run the Applet Viewer program, which in turn uses the HTML document.

Java in general and applets in particular are popular topics among programmers, partly because users can execute applets using a Web browser on the Internet. A **Web browser** is a program that allows you to display HTML documents on your computer screen. Web documents often contain Java applets.

Because applets are sent over the Internet and run from other applications, applet code is not trusted. A malicious programmer might try to include code that contains a virus, reads data from your files, or performs other unwanted tasks. Therefore, applet code runs in a constrained area called a sandbox. A **sandbox** is a safe area in which a program can run, much like a real sandbox is an area in a yard where children can play safely.

Fortunately, to run a Java applet, you don't need to learn much HTML; you need to learn only two pairs of HTML commands, called **tags**. The tag that begins every HTML document is **<html>**. Like all tags, this opening tag is surrounded by angle brackets. The `html` within the tag is an HTML keyword that specifies that an HTML document follows the keyword. The tag that ends every HTML document is **</html>**. The preceding slash indicates that the tag is a closing tag. The following is the simplest HTML document you can write:

```
<html>
</html>
```

Unlike Java, HTML is not case sensitive, so you can use <HTML> in place of `<html>`. However, this book uses the all-lowercase convention when displaying HTML code. With the growing importance of XML and XHTML, many programmers recommend putting all HTML tags in lowercase because XML and XHTML are case sensitive, even though HTML is not.

The simple HTML document, containing just the pair of `html` tags, begins and ends and does nothing in between; you can create an analogous situation in a Java method by typing an opening curly brace and following it immediately with the closing curly brace. HTML documents generally contain more statements. For example, to run an applet from an HTML document, you add an `<object>` and `</object>` tag pair. Usually, you place three attributes within the `<object>` tag: `code`, `width`, and `height`. **Tag attributes**, sometimes referred to as arguments, promote activity or describe the features of the tag; with arguments, the HTML tag can do something in a certain way. Note the following example:

```
<object code = "AClass.class" width = 300 height = 200> </object>
```

The three `object` tag attributes in the previous example are described with their corresponding arguments in the following list:

- `code` = is followed by the name of the compiled applet you are calling.
- `width` = is followed by the width of the applet on the screen.
- `height` = is followed by the height of the applet on the screen.

The name of the applet you call must be a compiled Java applet (with a .class file extension). The width and height of an applet are measured in pixels. For monitors that display 1024 pixels horizontally and 768 pixels vertically, a statement such as `width = 512 height = 384` creates an applet that occupies approximately one-fourth of most screens (half the height and half the width).

When you assign a height and width to an applet, keep in mind that a browser's menu bar and screen elements (such as the toolbar and the scroll bars) take up some of the screen viewing area for an applet.

Instead of the <object> and </object> tag pair, you can use <applet> and </applet> in your HTML applet host documents. However, the World Wide Web Consortium recommends using object rather than applet because it is more general, and it includes new and future media types in addition to applets. For more information, including definitions of and recommendations for using all the other HTML tags, visit *www.w3.org*.

Figure 17-2 shows an HTML file that could be used to run a JApplet named JHello. The applet will be 450 pixels wide by 150 pixels tall.

```
<html>
<object code = "JHello.class" width = 450 height = 150>
</object>
</html>
```

Figure 17-2 The TestJHello.html file

In Figure 17-2, the JHello.class file resides in the same folder as the HTML file, so no path is necessary in the object code statement. Later in this chapter, you will learn how to create the JHello class.

TWO TRUTHS & A LIE

Writing an HTML Document to Host an Applet

1. You must run applets within a Web page.
2. To view an applet, it must be called from another document, such as one written in HTML.
3. All HTML tags are surrounded by angle brackets.

The false statement is #1. You run applets within a page on the Internet or an intranet; you also can run an applet on a local computer from another program called Applet Viewer.

Using the init() Method

The JApplet class uses several methods that are invoked by a Web browser when the browser runs an applet. In a Java application, the main() method calls other methods that you write, but in contrast, an applet does not contain a main() method. With an applet, the browser or

Applet Viewer calls several methods automatically at different times. The following five methods are inherited from JApplet, are called automatically, and constitute the life cycle of an applet:

- public void init()
- public void start()
- public void paint()
- public void stop()
- public void destroy()

You can override any of these methods within a JApplet that you write. If you fail to implement one or more of these methods, your JApplet will use the versions that belong to the parent JApplet class. To create a Java applet that does anything useful, you must code statements within at least one of these methods.

For example, you can create a JApplet using only the init() method. The **init() method** is the first method called in any applet. You use it to perform initialization tasks, such as setting variables to initial values or placing applet components on the screen. In general, tasks that you would place in the constructor for a JFrame in an application are the same types of tasks you place in an init() method in a JApplet.

You must code the init() method's header as follows:

```
public void init()
```

Figure 17-3 shows the JApplet that displays "Hello. Who are you?" in a JLabel on the screen. The default layout manager for a JApplet is BorderLayout, so when the JLabel is added to the applet without a specified region, the JLabel occupies the entire JApplet surface.

```
import javax.swing.*;
import java.awt.*;
public class JHello extends JApplet
{
   JLabel greeting = new JLabel("Hello. Who are you?");
   public void init()
   {
      add(greeting);
   }
}
```

Figure 17-3 The JHello JApplet

After you write this JApplet, you do the following:

- Save this file as JHello.java, and compile it.
- Run the Applet Viewer using the command appletviewer TestJHello.html and the saved file shown in Figure 17-2.

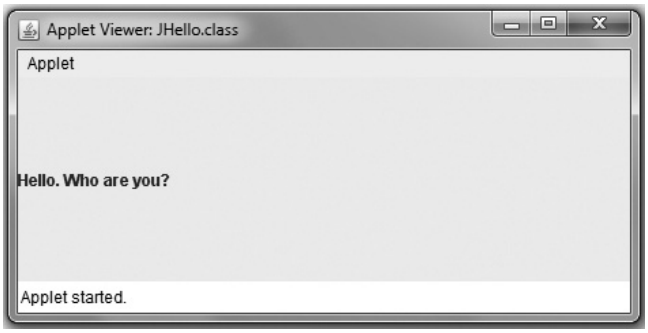

Figure 17-4 Output of the `JHello JApplet` when run in the Applet Viewer

Figure 17-4 shows how the applet appears on your screen. After you view a `JApplet`, you can click the close button on the Applet Viewer to end the application. Unlike a `JFrame`, you do not need to set a default close operation with a `JApplet`.

Watch the video *Writing and Running an Applet*.

TWO TRUTHS & A LIE

Using the `init()` Method

1. In a Java applet, the `main()` method is the first to execute; it calls other methods that you write.
2. The following five methods can automatically be called by every applet: `init()`, `start()`, `paint()`, `stop()`, and `destroy()`.
3. You can create a `JApplet` using only the `init()` method, which is the first method called in any applet.

The false statement is #1. In a Java application, the `main()` method calls other methods that you write, but in contrast, an applet does not contain a `main()` method.

You Do It

Creating an HTML Document to Host an Applet

In this section, you create a simple HTML document that you will use to display the applet you will create in the next section. You will name the applet `JGreet`, and it will occupy a screen area of 450 by 100 pixels.

1. Open a new file in your text editor. Type the opening HTML tag:

   ```
   <html>
   ```

2. On the next line, type the opening `object` tag that contains the applet's name and dimensions:

   ```
   <object code = "JGreet.class" width = 450 height = 200>
   ```

3. On the next line, type the applet's closing tag:

   ```
   </object>
   ```

4. On the next line, type the closing HTML tag:

   ```
   </html>
   ```

5. Save the file as **TestJGreet.html**. Just as when you create a Java application, be certain that you save the file as text only and use an .html extension. The .html file extension is required and makes the file easy to identify as an HTML file.

Creating and Running a `JApplet`

Next, you create the `JGreet` applet for which you prepared the HTML document.

1. Open a new text file in your text editor. Enter the following `import` statements you need for the `JApplet`. You need the `javax.swing` package because it defines `JApplet`, and you need the `java.awt` package because it defines `Container`.

   ```
   import javax.swing.*;
   import java.awt.*;
   ```

2. Next, enter the `JGreet JApplet`. It contains a `Container` that holds a `JLabel`. The `init()` method adds the `JLabel` to the `Container`.

(continues)

(continued)

```
public class JGreet extends JApplet
{
   Container con = getContentPane();
   JLabel greeting = new JLabel("Greetings!");
   public void init()
   {
      con.add(greeting);
   }
}
```

3. Save the file as **JGreet.java**. Then compile the class. If necessary, correct any errors and compile again.
4. At the command line, type **appletviewer TestJGreet.html**, and then press **Enter**. The applet appears on your screen, as shown in Figure 17-5.

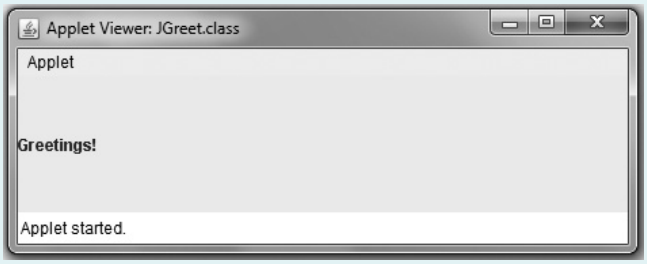

Figure 17-5 Execution of the `JGreet JApplet`

5. Use the mouse pointer to drag any corner of the Applet Viewer window to resize it. Notice that if you increase or decrease the window's height, the window is redrawn on the screen and the `JLabel` is automatically repositioned to remain centered within the window. If you make the window narrower by dragging its right border to the left, the `JLabel` eventually becomes partially obscured when the window becomes too narrow for the display.

If your operating system does not allow you to make the window narrow enough to obscure part of the greeting, make the string in the label longer—for example, "Greetings to you and all your family!" Recompile the applet and then use the `appletviewer` command to display the HTML document again. The string displayed will be long enough for you to observe the effects when you narrow the width of the window.

6. Close the Applet Viewer by clicking the **Close button** in the upper-right corner of the window.

(continues)

(continued)

Running a JApplet in Your Web Browser

1. Open any Web browser, such as Internet Explorer. You do not have to connect to the Internet; you will use the browser locally. (If you do not have a Web browser installed on your computer, skip to the end of Step 3.)
2. Click **File** on the menu bar, click **Open**, type the complete path for the HTML document that you created to access JGreet.class (for example, **C:\Java\Chapter17\TestJGreet.html**), and then press **Enter**. Instead of typing the filename, you can click the **Browse** button, browse to the file location, and then click **OK**. You might have to agree to several security messages before you can view the applet. The applet should appear in the browser on your screen. If you receive an error message, verify that the path and spelling of the HTML file are correct.
3. Close the browser.

Some applets might not work correctly with your browser. Java was designed with a number of security features so that when an applet is displayed on the Internet, the applet cannot perform malicious tasks, such as deleting a file from your hard drive. If an applet does nothing to compromise security, testing it using the Web browser or the `appletviewer` command achieves the same results. For now, you can get your applets to perform better by using the Applet Viewer window because the output does not depend on the browser type or version.

Working with JApplet Components

The output in Figures 17-4 and 17-5 is not very attractive or interesting. Fortunately, all the techniques that you used with `JFrames` in Chapters 14, 15, and 16 can also be used with `JApplets`. For example, you can do the following:

- Change the font and color of labels
- Use layout managers
- Add multiple GUI components
- Change the background color
- Add listeners for user events
- Add images and sounds

For example, Figure 17-6 shows how you can write a `JApplet` that displays a prompt, allows the user to enter text and press a button, and displays output. The `JApplet` also uses color and interesting fonts. The `init()` method sets up the applet surface and enables both the

button and text box to generate events to which the JApplet can respond. The setLayout() method is used to set the content pane's layout manager to FlowLayout; without the setLayout() statement, the content pane would use the default BorderLayout.


```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.awt.Color;
public class JHello2 extends JApplet implements ActionListener
{
   JLabel greeting = new JLabel("Hello. Who are you?");
   Font font1 = new Font("Teen", Font.BOLD, 36);
   Font font2 = new Font("Teen", Font.ITALIC, 48);
   JTextField answer = new JTextField(10);
   JButton pressMe = new JButton("Press me");
   JLabel personalGreeting = new JLabel(" ");
   Container con = getContentPane();
   public void init()
   {
      greeting.setFont(font1);
      personalGreeting.setFont(font2);
      con.add(greeting);
      con.add(answer);
      con.add(pressMe);
      con.setLayout(new FlowLayout());
      con.setBackground(Color.YELLOW);
      pressMe.addActionListener(this);
      answer.addActionListener(this);
   }
   public void actionPerformed(ActionEvent e)
   {
      String name = answer.getText();
      con.remove(greeting);
      con.remove(pressMe);
      con.remove(answer);
      personalGreeting.setText("Hello, " + name + "!  ");
      con.add(personalGreeting);
      con.setBackground(Color.PINK);
      validate();
   }
}
```

Figure 17-6 The JHello2 class that adds a JLabel when the user clicks the JButton

The code in the actionPerformed() method in the JHello2 class shows that after the user clicks the button, the name is retrieved from the text field; the label, button, and text field are removed from the applet surface; and the setText() method sets the JLabel text for personalGreeting to "Hello, " + name + "! ". Then the personal greeting is added to the

JApplet's Container, and the background color is changed. You learned about all of the same methods when you studied JFrames in Chapters 14 and 15. Figure 17-7 shows a typical execution.

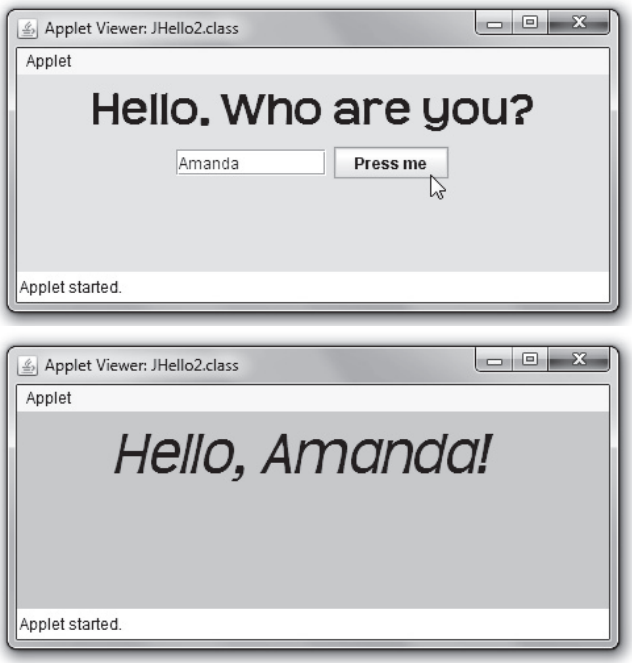

Figure 17-7 Typical execution of the JHello2 JApplet

In the actionPerformed() method in Figure 17-6, the final statement following the addition of the JLabel is validate(). As you learned in Chapter 15, invoking the validate() method after adding one or more JComponents to an applet ensures that the Components draw themselves on the screen. It isn't necessary to call the validate() method every time you add a JComponent to a JApplet. For example, when you add components in the init() or start() methods, you do not have to call validate(). When you add components in other methods (frequently event-handling methods), you must call validate().

The validate() method is complex. Even the online Java documentation refers to its performance as "voodoo."

Watch the video *Working with JApplet Components*.

TWO TRUTHS & A LIE

Working with JApplet Components

1. In an applet, you can change fonts and colors of GUI components just as you can in an application.
2. JApplets can contain an actionPerformed() method to respond to mouse clicks.
3. Invoking the validate() method in an applet allows it to pass security restrictions.

The false statement is #3. Invoking the validate() method after adding one or more JComponents to an applet ensures that the Components draw themselves on the screen.

You Do It

Creating a More Complicated JApplet

Next, you change the font of the text in your JGreet applet, add components, and make other changes to make a more complicated and useful JApplet.

1. Open the **JGreet.java** file in your text editor, and change the class name to **JGreet2**. Immediately save the file using the filename **JGreet2.java**.
2. After the declaration of the greeting JLabel, declare a Font object named bigFont by typing the following:

```
Font bigFont = new Font("Times Roman", Font.ITALIC, 24);
```

3. Within the init() method, set the greeting font to bigFont by typing the following:

```
greeting.setFont(bigFont);
```

4. Along with the other declarations, declare two JLabels, two empty JTextFields, and a JButton with the label **View Greeting** by typing the following:

(continues)

(continued)

```
JLabel firstLabel = new JLabel("Please enter your first name:");
JLabel lastLabel = new JLabel("Please enter your last name:");
JTextField firstField = new JTextField("",10);
JTextField lastField = new JTextField("",10);
JButton viewButton = new JButton("View Greeting");
```

5. Set the new layout manager to a flow layout with the following statement:

```
FlowLayout flow = new FlowLayout();
```

6. Within the `init()` method, set a layout manager as follows:

```
con.setLayout(flow);
```

7. Add all the newly created components to the applet by typing the following:

```
con.add(firstLabel);
con.add(firstField);
con.add(lastLabel);
con.add(lastField);
con.add(viewButton);
```

8. On the next line, request focus for the first-name text field by typing:

```
firstField.requestFocus();
```

9. Save the file and compile it.

10. Open the **TestJGreet.html** document you created earlier, and change the class name in the `object code` statement to **`JGreet2.class`**. Save the file as **TestJGreet2.html**. Execute the **appletviewer TestJGreet2.html** command. The output is shown in Figure 17-8. Confirm that you can type characters into the `JTextFields` and that you can click the `JButton` using the mouse. You haven't coded any action to take place as a result of a `JButton` click yet, but the components should function.

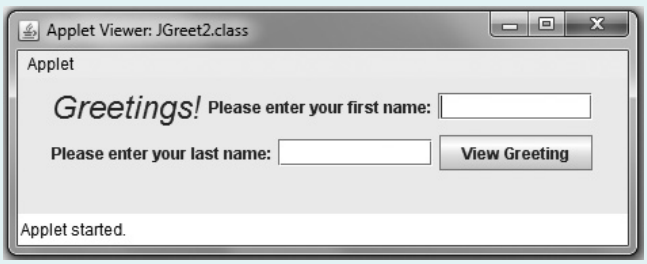

Figure 17-8 The `JGreet2 JApplet`

11. Close the Applet Viewer window.

(continues)

(continued)

Making the JApplet's Button Respond to Events

Next, you make your applet an event-driven program by adding functionality to the applet. When the user enters a name and clicks the JButton, the JApplet displays a personalized greeting.

1. Open the **JGreet2.java** file in your text editor, and change the class name to JGreet3. Immediately save the file as **JGreet3.java**.
2. Add a third import statement to your program by typing the following:

```
import java.awt.event.*;
```

3. Add the following phrase to the class header:

```
implements ActionListener
```

4. Prepare your JApplet for JButton-sourced events by typing the following statement within the init() method:

```
viewButton.addActionListener(this);
```

5. Following the init() method, add the following actionPerformed() method. In the method, declare two Strings—one to hold the user's first name and another for the last name—and then use the getText() method on the JTextFields to retrieve values for these Strings. Using the Strings, display a personalized question for the user.

```
public void actionPerformed(ActionEvent thisEvent)
{
   String firstName = firstField.getText();
   String lastName = lastField.getText();
   greeting.setText("How are you, " + firstName + " " +
      lastName + "?");
}
```

6. Save the file and compile the program. Edit the file **TestJGreet2.html** to change the class reference to **JGreet3.class**, and then save the file as **TestJGreet3.html**. Run the applet using the **appletviewer TestJGreet3.html** command.
7. Type your name in the JTextFields, and then click the **View Greeting** button. The personalized message should appear, similar to the one in Figure 17-9.

(continues)

(continued)

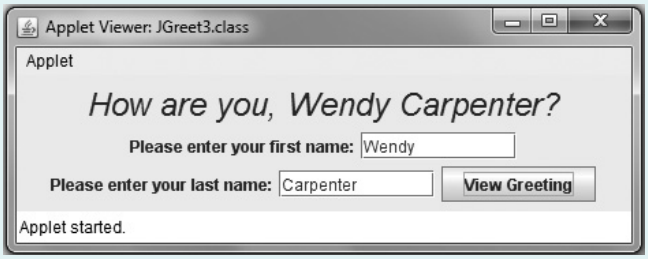

Figure 17-9 Typical execution of the `JGreet3` `JApplet`

8. Drag the mouse to highlight the first or last name (or both) in the Applet Viewer window, and then type a different name. Click the **View Greeting** button. A greeting that uses the new name appears.
9. Close the Applet Viewer window.

Understanding the `JApplet` Life Cycle

Applets are popular because they are easy to use in Web pages. Because applets execute in a browser, the `JApplet` class contains methods that are automatically called by the browser. You already are familiar with the `paint()` method. In Chapter 16 you learned that the system can request that a component's contents be repainted or that the application can request repainting. The `paint()` method is always called after the `init()` and `start()` methods execute. It is also called if an applet needs repainting—if, for example, the user covers part of an applet with another open window and then uncovers it. The method works the same way with `JApplets` as with `JFrames`. The method header is as follows:

```
public void paint(Graphics g)
```

As you learned in Chapter 16, the `paint()` method provides you with a `Graphics` object. If you override the `paint()` method in your applet, you can use the automatically provided `Graphics` object to draw shapes and strings.

The other four methods that are automatically called in a `JApplet` are `init()`, `start()`, `stop()`, and `destroy()`.

The `init()` Method

You have already seen examples of `JApplets` that contain `init()` methods. When a Web page containing a `JApplet` is loaded in the browser, or when you run the `appletviewer` command within an HTML document that calls a `JApplet`, the applet's `init()` method executes. The `init()` method might be your version (if you have written one to override the version in the

parent class) or the automatically provided version. You should write your own `init()` method when you have any initialization tasks to perform, such as setting up user interface components.

The `start()` Method

The **`start()` method** executes after the `init()` method, and it executes again every time the applet becomes active after it has been inactive. For example, if you run a `JApplet` using the `appletviewer` command and then minimize the Applet Viewer window, the `JApplet` becomes inactive. When you restore the window, the `JApplet` becomes active again. On the Internet, users can leave a Web page, visit another page, and then return to the first site. Again, the `JApplet` becomes inactive and then active. When you write your own `start()` method, you must include any actions you want your `JApplet` to take when a user revisits the `JApplet`. For example, you might want to resume some animation that you suspended when the user left the applet.

The `stop()` Method

When a user leaves a Web page (perhaps by minimizing a window or traveling to a different Web page), the **`stop()` method** is invoked. You override the existing empty `stop()` method only if you want to take some action when a `JApplet` is no longer visible. You usually don't need to write your own `stop()` methods.

The `destroy()` Method

The **`destroy()` method** is called when the user closes the browser or Applet Viewer. Closing the browser or Applet Viewer releases any resources the `JApplet` might have allocated. As with the `stop()` method, you do not usually have to write your own `destroy()` methods.

Advanced Java programmers override the `stop()` and `destroy()` methods when they want to add instructions to "suspend a thread," or stop a chain of events that were started by a `JApplet` but are not yet completed.

Every `JApplet` has the same life cycle outline, as shown in Figure 17-10. When the applet executes, the `init()` method runs, followed by the `start()` method. If the user leaves the `JApplet`'s page, the `stop()` method executes. When the user returns, the `start()` method executes. The `stop()` and `start()` sequence might continue any number of times until the user closes the browser (or Applet Viewer), which invokes the `destroy()` method.

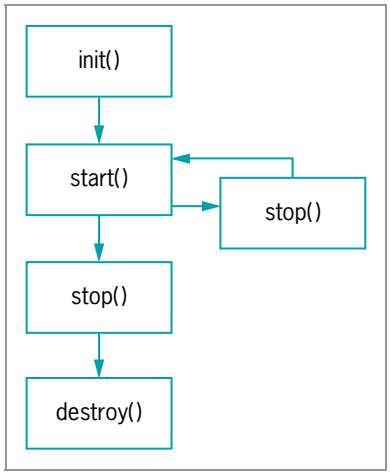

Figure 17-10 The `JApplet` life cycle

TWO TRUTHS & A LIE

Understanding the `JApplet` Life Cycle

1. The `paint()` method is always called after the `init()` and `start()` methods execute or if an applet needs repainting.
2. When a Web page containing a `JApplet` is loaded in the browser, the applet's `start()` method is the first to execute.
3. When a user leaves a Web page, the `stop()` method executes.

The false statement is #2. When a Web page containing a `JApplet` is loaded in the browser, or when you run the `appletviewer` command within an HTML document that calls a `JApplet`, the applet's `init()` method is the first to execute.

You Do It

Understanding the Applet Life Cycle

To demonstrate the life cycle methods in action, you can write a `JApplet` that overrides all four methods: `init()`, `start()`, `stop()`, and `destroy()`. When you run this applet, you can observe the number of times each method executes.

(continues)

(continued)

1. Open a new text file in your text editor, and then type the following `import` statements:

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
```

2. To make the `JApplet` include a `JButton` that the user can click, and to implement an `ActionListener`, type the following header for a `JLifeCycle` applet and include an opening curly brace for the class:

```
public class JLifeCycle extends JApplet implements ActionListener
{
```

3. Declare the following six `JLabel` objects, which display each of the six methods that execute during the lifetime of the applet:

```
JLabel messageInit = new JLabel("init ");
JLabel messageStart = new JLabel("start ");
JLabel messageDisplay = new JLabel("display ");
JLabel messageAction = new JLabel("action ");
JLabel messageStop = new JLabel("stop ");
JLabel messageDestroy = new JLabel("destroy ");
```

4. Declare a `JButton` by typing the following:

```
JButton pressButton = new JButton("Press");
```

5. Declare six integers that hold the number of occurrences of each of the six methods by typing the following code:

```
int countInit, countStart, countDisplay, countAction,
   countStop, countDestroy;
```

6. Start the `init()` method by adding a container and flow layout manager with the following statements:

```
public void init()
{
  Container con = getContentPane();
  con.setLayout (new FlowLayout());
```

7. Add the following statements, which add 1 to `countInit`, place the components within the applet, and then call the `display()` method:

(continues)

(continued)

```
        ++countInit;
        con.add(messageInit);
        con.add(messageStart);
        con.add(messageDisplay);
        con.add(messageAction);
        con.add(messageStop);
        con.add(messageDestroy);
        con.add(pressButton);
        pressButton.addActionListener(this);
        display();
    }
```

8. Add the following `start()` method, which adds 1 to `countStart` and calls `display()`:

```
    public void start()
    {
        ++countStart;
        display();
    }
```

9. Add the following `display()` method, which adds 1 to `countDisplay`, displays the name of each of the six methods with the current count, and indicates how many times the method has executed:

```
    public void display()
    {
        ++countDisplay;
        messageInit.setText("init " + countInit);
        messageStart.setText("start " + countStart);
        messageDisplay.setText("display " + countDisplay);
        messageAction.setText("action " + countAction);
        messageStop.setText("stop " + countStop);
        messageDestroy.setText("destroy " + countDestroy);
    }
```

10. Add the following `stop()` and `destroy()` methods. Each adds 1 to the appropriate counter and calls `display()`:

```
    public void stop()
    {
        ++countStop;
        display();
    }
    public void destroy()
    {
        ++countDestroy;
        display();
    }
```

(continues)

(continued)

11. When the user clicks `pressButton`, the following `actionPerformed()` method executes; it adds 1 to `countAction` and displays it:

```
public void actionPerformed(ActionEvent e)
{
   ++countAction;
   display();
}
```

12. Add the closing curly brace for the class. Save the file as **JLifeCycle.java**. Compile the class.

    Take a moment to examine the code you created for the `JLifeCycle` applet. Each method adds 1 to one of the six counters, but you never explicitly call any of the methods except `display()`; each of the other methods is called automatically.

Creating an HTML Document to Host the `JApplet`

Next, you create an HTML document so you can test the `JLifeCycle` applet.

1. Open a new file in your text editor, and then enter the following HTML code:

```
<html>
<object code = "JLifeCycle.class" width = 460 height = 60>
</object>
</html>
```

2. Save the file as **TestJLifeCycle.html**.
3. Run the HTML document using the following command:

```
appletviewer TestJLifeCycle.html
```

   Figure 17-11 shows the output. When the applet begins, the `init()` method is called, so 1 is added to `countInit`. The `init()` method calls `display()`, so 1 is added to `countDisplay`. Immediately after the `init()` method executes, the `start()` method is executed, and 1 is added to `countStart`. The `start()` method calls `display()`, so 1 more is added to `countDisplay`. The first time you see the applet, `countInit` is 1, `countStart` is 1, and `countDisplay` is 2. The methods `actionPerformed()`, `stop()`, and `destroy()` have not yet been executed.

(continues)

(continued)

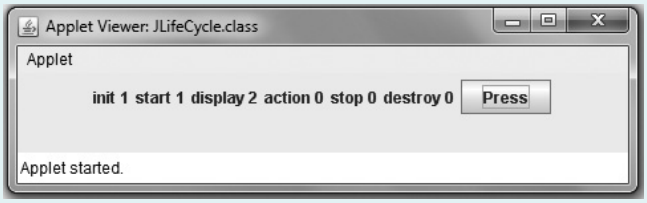

Figure 17-11 The `JLifeCycle JApplet` when it first executes

4. Click the **Minimize** button to minimize the Applet Viewer window, and then click the **Taskbar** button to restore it. The applet now looks like Figure 17-12. The `init()` method still has been called only once, but when you minimized the applet, the `stop()` method executed, and when you restored it, the `start()` method executed. Therefore, `countStop` is now 1 and `countStart` has increased to 2. In addition, because `start()` and `stop()` call `display()`, `countDisplay` is increased by 2 and now holds the value 4.

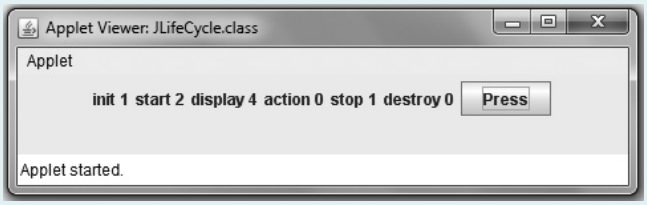

Figure 17-12 The `JLifeCycle JApplet` after minimizing and restoring

5. Minimize and restore the Applet Viewer window again. Now, the `stop()` method has executed twice, the `start()` method has executed three times, and the `display()` method has executed a total of six times, as shown in Figure 17-13.

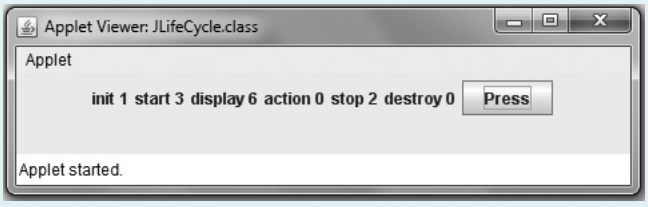

Figure 17-13 The `JLifeCycle JApplet` after minimizing and restoring twice

(continues)

(continued)

6. Click the **Press** button. The count for the `actionPerformed()` method is now 1, and `actionPerformed()` calls `display()`, so `countDisplay` is now 7, as shown in Figure 17-14.

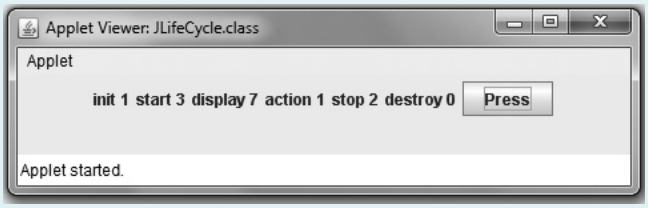

Figure 17-14 The `JLifeCycle JApplet` after minimizing and restoring twice, and then pressing the button

7. Continue to minimize, maximize, and click the **Press** button. Note the changes that occur with each activity until you can correctly predict the outcome. Notice that the `destroy()` method is not executed until you close the applet, and then it is too late to observe an increase in `countDestroy`.
8. Close the Applet Viewer.

Understanding Multimedia and Using Images

Multimedia describes the use of sound, images, graphics, and video in computer programs. Most computers that are sold today are "multimedia ready"—that is, they have CD-RW and DVD-RW drives, audio boards, and video capabilities. Java provides extensive multimedia tools, including the following:

- Java programmers can use the Java 2D or the Java 3D Application Programming Interface (API) to create 3D graphics applications.
- The Java Media Framework (JMF) API allows you to add audio and video media to an application.
- Java Sound allows you to play, record, and modify audio files.
- The Java Advanced Imaging API provides image-manipulation capabilities.
- The Java Speech API allows a user to input speech commands and allows an application to produce speech output.

Adding Images to JApplets

An **image** is a likeness of a person or thing. Images abound on the Internet in all shapes, colors, and sizes. Image formats supported by Java include:

- Graphics Interchange Format (GIF), which can contain a maximum of 256 different colors
- Joint Photographic Experts Group (JPEG), which is commonly used to store photographs and is a more sophisticated way to represent a color image
- Portable Network Graphics (PNG), which is more flexible than GIF and stores images in a lossless form. (PNG was originally designed to be a portable image storage form for computer-originated images.) **Lossless data compression** is a set of rules that allows an exact replica of data to be reconstructed from a compressed version. If you have ever worked with a .zip file, you have worked with lossless data compression.

The `Image` class provides many of Java's image capabilities; this class loads images that have been stored in one of the allowed `Image` formats. The `Image` class, which you can find in the `java.awt` package, is an abstract class. Recall that an abstract class is one from which you cannot create any objects, but which you can use as an interface or from which you can inherit. Because `Image` is abstract, you must create `Image` objects indirectly using the `getImage()` method.

The `getImage()` method is used to load an `Image` into the named `Image` in the applet. One version of the `getImage()` method can take up to two arguments: a location where the image is stored and its filename. For example, you can create and load an `Image` named `companyLogo` with a statement such as the following:

```
companyLogo = getImage(getCodeBase(), "logo.gif");
```

The `getCodeBase()` call returns the Uniform Resource Locator (URL) where the code is located. That is, it finds the directory from which the code is running.

You can use the applet `paint()` method to display `Image` object images. The `drawImage()` method is a `Graphics` method that uses the following four arguments:

- The first argument is a reference to the `Image` object in which the image is stored.
- The second argument is the x-coordinate where the image appears on the applet.
- The third argument is the y-coordinate where the image appears on the applet.
- The fourth argument is a reference to an `ImageObserver` object.

An `ImageObserver` object can be any object that implements the `ImageObserver` interface. Because the `Component` class implements the `ImageObserver` interface, all `Components`, including `JApplets`, inherit this implementation. Usually, the `ImageObserver` object is the object on which the image appears—in this case, the `JApplet`. Recall from Chapter 4 that the `this` reference refers to the current object using a method. Frequently, with the `drawImage()` method, you use the `this` reference to indicate that you want the `Image` drawn on the current `JApplet`. For example, the code to display the `companyLogo` image in the upper-left corner of the `JApplet` is as follows:

```
g.drawImage(companyLogo, 0, 0, this);
```

You can use an overloaded version of the `Graphics` method `drawImage()`to output a scaled image. This method takes six arguments. Notice that the first three arguments are the same as those for the four-argument version of the `drawImage()` method. In the overloaded version:

- The first argument is a reference to the `Image` object in which the image is stored.
- The second and third arguments are the x- and y-coordinates where the image appears on the applet.
- The fourth and fifth arguments are the width and height of the scaled object.
- The sixth argument uses the `this` reference to implement the `ImageObserver` object.

For example, the following code displays the `companyLogo` image at coordinates 0, 120 using the full width of the `JApplet`, but 100 pixels less than the height:

```
g.drawImage(companyLogo, 0, 120, getWidth(), getHeight() - 100, this);
```

Figure 17-15 shows a `JApplet` that draws an image twice. The `JCompanyImage JApplet` uses an image file named CompanyLogo.png, which holds a logo that is originally 287 pixels wide by 129 pixels high. Within the `paint()` method in the `JApplet`, the image is drawn first in its "natural," or original, size in the upper-left corner. Then it is redrawn lower at twice its original size. Figure 17-16 shows the contents of the HTML document that displays the applet; the output of the `JCompanyImage` applet is shown in Figure 17-17.

```
import java.awt.*;
import java.applet.*;
import javax.swing.*;
public class JCompanyImage extends JApplet
{
   Image companyLogo;
   final int WIDTH = 287;
   final int HEIGHT = 129;
   final int FACTOR = 2;
   public void init()
   {
      companyLogo = getImage(getCodeBase(), "CompanyLogo.png");
   }
   public void paint(Graphics g)
   {
      super.paint(g);
      // Draw image at its natural size
      g.drawImage(companyLogo, 0, 0, this);
      // Draw the image scaled - twice as large
      g.drawImage(companyLogo, 0, HEIGHT, WIDTH * FACTOR,
         HEIGHT * FACTOR, this);
   }
}
```

Figure 17-15 The `JCompanyImage JApplet`

```
<html>
<object code = "JCompanyImage.class" width = 600 height = 390>
</object>
</html>
```

Figure 17-16 The `TestJCompanyImage` HTML document

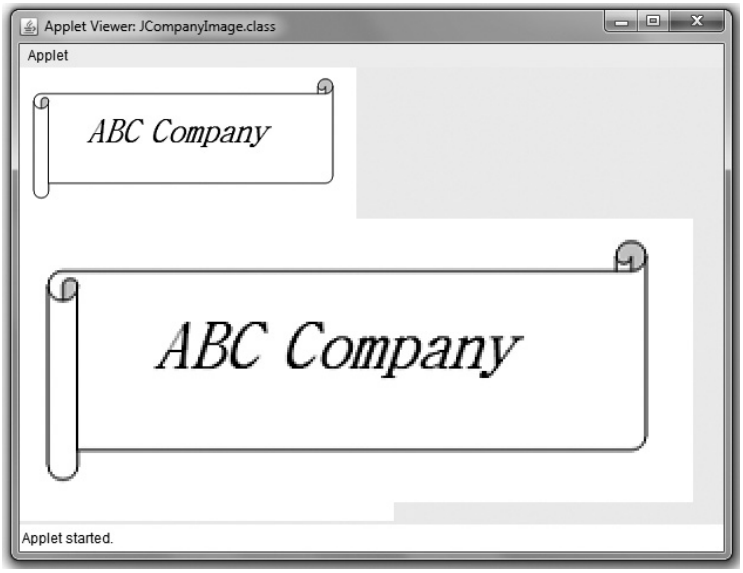

Figure 17-17 Output of the `JCompanyImage JApplet`

You can examine the CompanyLogo.png file in the Chapter 17 folder of your downloadable student files.

Using `ImageIcons`

You can also use the `ImageIcon` class to create images in your applications and applets. In general, working with the `ImageIcon` class is simpler than working with `Image`. You can use all the `Image` methods with an `ImageIcon`, plus many additional methods. Unlike with the `Image` class, you can create `ImageIcon` objects directly. Also, unlike with `Images`, you can place an `ImageIcon` on a `Component`, such as a `JPanel`, `JLabel`, or `JButton`, as well as on a `JApplet` or `JFrame`. For example, the following statements create a `JButton` that contains a picture of an arrow:

```
ImageIcon arrowPicture = new ImageIcon("arrow.gif");
JButton arrowButton = new JButton(arrowPicture);
```

Behind the scenes, each `ImageIcon` object uses an `Image` object to hold the image data and a `MediaTracker` object to keep track of the image's loading status. As you are aware if you have visited many Web pages, some images can require a good deal of time to load. To improve performance, the `Image` `get()` methods return immediately while the image continues to load, so that your application does not have to wait before performing other operations.

You can also use the `paintIcon()` method to display `ImageIcon` images. This method requires four arguments:

- The first argument is a reference to the `Component` on which the image appears—`this` in the following example.
- The second argument is a reference to the `Graphics` object used to render the image—`g` in the following example.
- The third and fourth arguments are the x- and y-coordinates for the upper-left corner of the image.

The code to display the `arrowPicture` `ImageIcon` using the `paintIcon()` method is:

```
arrowPicture.paintIcon(this, g, 180, 0);
```

You can retrieve an `ImageIcon`'s width and height with methods named `getIconWidth()` and `getIconHeight()`; each returns an integer. Figure 17-18 contains a `JApplet` that manipulates an `ImageIcon`'s width and height to achieve display effects. In the `JBear` `JApplet`, an `ImageIcon` is created using a .gif file. In the `init()` method, the width and height of the `ImageIcon` are stored in variables named `width` and `height`. In the `actionPerformed()` method, the width and height of the image are doubled with every button click. Figure 17-19 shows the `JApplet` when it starts, after the user has clicked the button once, and after the user has clicked the button twice.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class JBear extends JApplet implements ActionListener
{
   private ImageIcon image = new ImageIcon("bear.gif");
   private JButton closerButton = new JButton("Oh my!");
   private int width, height;
   Container con = getContentPane();
   public void init()
   {
      con.setLayout(new FlowLayout());
      closerButton.addActionListener(this);
      con.add(closerButton);
      width = image.getIconWidth();
      height = image.getIconHeight();
   }
   public void actionPerformed(ActionEvent event)
   {
      width = width * 2;
      height = height * 2;
      repaint();
   }
   public void paint(Graphics g)
   {
      super.paint(g);
      g.drawImage(image.getImage(), 0, 0, width, height, this);
   }
}
```

Figure 17-18 The JBear JApplet

Figure 17-19 Output of the `JBear` `JApplet`

The first statement in the `paint()` method of the `JBear` applet passes the `Graphics` object to the `JApplet` class `paint()` method so that the applet surface is cleared. If you eliminate the call to `super.paint()` and then get a series of new bear images, all the previous versions remain on the screen "behind" the newest, larger one.

Notice that the `drawImage()` method call in the `paint()` method uses `image.getImage()` as an argument. An `ImageIcon` cannot be drawn to scale, but an `Image` can, so you use the `getImage()` method to return a scalable `Image` reference for the `ImageIcon` object.

TWO TRUTHS & A LIE

Understanding Multimedia and Using Images

1. Multimedia describes the use of sound, images, graphics, and video in computer programs; most computers that are sold today are "multimedia ready."
2. Image formats supported by Java include GIF, JPEG, and PNG.
3. You can use the `Picture` class to create images in your applications and applets.

The false statement is #3. You can use the `Image` or `ImageIcon` class to create images in your applications and applets; you can even place an `ImageIcon` on a `Component`, such as a `JPanel` or `JButton`.

You Do It

Displaying Images

In the next steps, you add an animated image to an applet.

1. Open a new file in your text editor, and then enter the first few lines of the `JMonkey JApplet`:

```
import java.awt.*;
import javax.swing.*;
public class JMonkey extends JApplet
{
```

2. Declare an `ImageIcon` and initialize it with the monkey.gif file that is stored in your downloadable student files. The file contains an animated image of a chimpanzee shaking its head from side to side. Declare variables to hold the width and height of the image, the content pane for the applet, and horizontal and vertical placement positions.

```
private ImageIcon image = new ImageIcon("monkey.gif");
private int width, height;
Container con = getContentPane();
int x = 30;
int y = 30;
```

(continues)

(continued)

3. Write the `init()` method to set the layout manager and get the width and height of the image.

```
public void init()
{
   con.setLayout(new FlowLayout());
   width = image.getIconWidth();
   height = image.getIconHeight();
}
```

4. The `paint()` method calls the parent's method, then draws a string and draws the image 20 pixels lower. Add the method shown below, and then add a closing curly brace for the class.

```
   public void paint(Graphics g)
   {
      super.paint(g);
      g.drawString("No, no, no", x, y);
      g.drawImage(image.getImage(), x, y + 20,
         width, height, this);
   }
}
```

5. Save the file as **JMonkey.java**, and compile it.

6. Write an HTML document to host the applet as follows:

```
<html>
<object code = "JMonkey.class" width = 200 height = 150>
</object>
</html>
```

7. Save the HTML document as **TestJMonkey.html**. Run the applet. The output should look like Figure 17-20. On your screen, you will see the animated monkey move.

Figure 17-20 Execution of the `JMonkey` applet

Adding Sound to JApplets

Java programs can play audio clips on computers that have speakers and a sound card (which includes most computers that are sold today). Java supports sound using methods from the Applet class, rather than JApplet. You can use these methods to retrieve and play sound files that use various sound formats. These formats include the Windows Wave file format (.wav), Sun Microsystems Audio file format (.au), and Music and Instrument Digital Interface file format (.midi or .mid).

The simplest way to retrieve and play a sound is to use the play() method of the Applet class. The play() method retrieves and plays the sound as soon as possible after it is called. The play() method takes one of two forms:

- play() with one argument—The argument is a URL object that loads and plays an audio clip when both the URL object and the audio clip are stored at the same URL.
- play() with two arguments—The first argument is a URL object, and the second argument is a folder path name that loads and plays the audio file. The first argument is often a call to a getCodeBase() method or getDocumentBase() method to retrieve the URL object; the second argument is the name of the audio clip within the folder path that is stored at that URL.

Used with the codebase attribute, which indicates the filename of the applet's main class file, the getCodeBase() and getDocumentBase() methods direct the browser to look in a different folder for the applet and other files it uses. This is necessary when the desired files are in a different location than the Web page containing the applet. By calling getCodeBase() in an applet, you get a URL object that represents the folder in which the applet's class file is stored. For example, the following statement retrieves and plays the tune.au sound file, which is stored in the same place as the applet:

```
play(getCodeBase(),"tune.au");
```

The getDocumentBase() method returns an absolute URL naming the directory of the document in which the applet is stored. It is sometimes used instead of getCodeBase() as a matter of preference. An applet is restricted to reading files only from the server that hosts it.

To play a sound more than once, or to start or stop the sound, you must load the sound into an AudioClip object using the applet's newAudioClip() method. AudioClip is part of the java.awt.Applet class and must be imported into your program. Like the play() method, the getAudioClip() method can take one or two arguments. The first argument (or only argument, if there is only one) is a URL argument that identifies the sound file; the second argument is a folder path reference needed for locating the file.

The following statement loads the sound file from the previous example into the clip object:

```
AudioClip aClip = new AudioClip(getCodeBase(), "audio/tune.au");
```

Here, the sound file reference indicates that the tune.au sound file is in the audio folder. After you have created an AudioClip object, you can use the play() method to call and play the sound, the stop() method to halt the playback, and the loop() method to play

the sound repeatedly. Multiple `AudioClip` items can play at the same time, and the resulting sound is mixed together to produce a composite.

In the next "You Do It" section, you will create a `JApplet` that plays a sound.

Watch the video *Using Images and Sound in an Applet*.

TWO TRUTHS & A LIE

Adding Sound to `JApplets`

1. Java programs can play audio clips on computers that have speakers and a sound card.
2. Java supports sound using methods from the `JApplet` class.
3. Sound formats that applets can use include the Windows Wave file format (.wav), Sun Audio file format (.au), and Music and Instrument Digital Interface file format (.midi or .mid).

The false statement is #2. Java supports sound using methods from the `Applet` class (rather than `JApplet`).

You Do It

Playing Sounds

Next, you will use the `loop()` method and an `AudioClip` to play a sound continually in an applet. You will also create and add a `Graphics2D` object.

1. Open a new file in your text editor, and then enter the first few lines of the `JSound JApplet`:

```
import java.awt.*;
import java.applet.*;
import javax.swing.*;
public class JSound extends JApplet
{
```

(continues)

(continued)

2. Enter the following statement to declare an `AudioClip` object named `sound`:

```
AudioClip sound;
```

3. Create the `init()` method and an `AudioClip` object to play the mysteryTune.au sound file by entering the following code:

```
public void init()
{
   sound = getAudioClip(getCodeBase(),"mysteryTune.au");
}
```

You can find the mysteryTune.au file in the Chapter 17 folder of your downloadable student files.

4. Create the following `start()` method. This method uses the `loop()` method to play the mysteryTune.au sound file continually:

```
public void start()
{
   sound.loop();
}
```

5. Create the following `stop()` method to halt the mysteryTune.au sound file:

```
public void stop()
{
   sound.stop();
}
```

6. Create a `Graphics` object using `paint(Graphics g)`, and then use a cast to change the graphics context to a `Graphics2D` object. Use the `drawString()` method to create a message that appears on the screen while the `JApplet` plays. Add a closing curly brace for the class.

```
   public void paint(Graphics g)
   {
      super.paint(g);
      Graphics2D g2D = (Graphics2D)g;
      g2D.drawString("Listen to the mystery tune ",
         10, 10);
   }
}
```

7. Save the file as **JSound.java**, and then compile it.

8. Open a new file in your text editor, and then enter the following HTML document to test the `JApplet`:

(continues)

(continued)

```
<html>
<object code = "JSound.class" width = 200 height = 50>
</object>
</html>
```

9. Save the HTML document as **TestJSound.html**, and then run it using the **appletviewer TestJSound.html** command. The output should look like Figure 17-21. If speakers are installed on your system and they are on, you should also be able to hear sound playing continually.

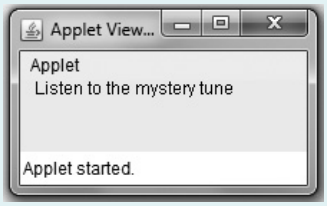

Figure 17-21 Output of the `JSound JApplet`

Don't Do It

- Don't forget a matching closing tag for every opening tag in an HTML document.
- Don't forget to use the .class extension with the name of a `JApplet` you want to execute from an HTML document.
- Don't add a `main()` method to a `JApplet`; it will not execute automatically like it does in an application.
- Don't try to execute an applet using the `java` command. An applet doesn't contain a `main()` method and won't execute like an application.

Key Terms

An **applet** is a Java program that is called from another application.

`JApplet` is a `Swing` class from which you can inherit to create your own applet.

HTML, or **Hypertext Markup Language**, is a simple language used to create Web pages for the Internet.

Extensible Hypertext Markup Language (XHTML) is an extension of HTML.

Applet Viewer is a program that comes with the JDK that allows you to view applets without using a Web browser.

The **`appletviewer` command** allows you to view an applet in a viewing program that comes with the JDK.

A **Web browser** is a program that allows you to display HTML documents on your computer screen.

A **sandbox** is a safe area in which a program can run without causing harm to other areas of a system.

Tags are HTML commands.

`<html>` is the tag that begins every HTML document.

`</html>` is the tag that ends every HTML document.

Tag attributes, sometimes referred to as arguments, promote activity or describe the features of an HTML tag.

The **`init()` method** is the first method called in any applet.

The **`start()` method** executes after the `init()` method, and it executes again every time the applet becomes active after it has been inactive.

The **`stop()` method** is invoked in an applet when a user leaves a Web page (perhaps by minimizing a window or traveling to a different Web page).

The **`destroy()` method** is called in an applet when the user closes the browser or Applet Viewer.

Multimedia describes the use of sound, images, graphics, and video in computer programs.

An **image** is a likeness of a person or thing.

Lossless data compression is a set of rules that allows an exact replica of data to be reconstructed from a compressed version.

Chapter Summary

- An applet is a program that is called within another application—often a Web browser or a program called the Applet Viewer. Applets descend from the `JApplet` class and do not contain a `main()` method.
- To view an applet, it must be called from another document written in Hypertext Markup Language (HTML), which is a simple language used to create Web pages for the Internet. When you create an applet, you write it in Java, save it with a .java file extension, compile it, and write an HTML document that includes a statement to call your compiled Java

class. You then load the HTML document into a Web browser or run the Applet Viewer program, which in turn uses the HTML document.

- With an applet, the browser calls several methods automatically at different times. The five methods that are included in every applet are `init()`, `start()`, `paint()`, `stop()`, and `destroy()`. You can override any of those methods within a `JApplet` that you write.
- When you write a `JApplet`, you can add components, use fonts and colors, change the layout manager, and add event listeners, in much the same way you do in a `JFrame`.
- An applet's `paint()` method is always called after the `init()` and `start()` methods execute. It is also called if an applet needs repainting. When a Web page containing a `JApplet` is loaded in the browser, or when you run the `appletviewer` command within an HTML document that calls a `JApplet`, the applet's `init()` method executes. The `start()` method executes after the `init()` method, and it executes again every time the applet becomes active after it has been inactive. When a user leaves a Web page, the `stop()` method is invoked. The `destroy()` method is called when the user closes the browser or Applet Viewer.
- Multimedia describes the use of sound, images, graphics, and video in computer programs. Image formats supported by Java include Graphics Interchange Format (GIF), Joint Photographic Experts Group (JPEG), and Portable Network Graphics (PNG). The `Image` class provides many of Java's image capabilities. You can also use the `ImageIcon` class to create images in your applications and applets.
- Java programs can play audio clips on computers that have speakers and a sound card. Supported sound formats include the Windows Wave file format (.wav), Sun Audio file format (.au), and Music and Instrument Digital Interface file format (.midi or .mid). The simplest way to retrieve and play a sound is to use the `play()` method of the `Applet` class.

Review Questions

1. An applet is like a Java application in all of the following ways *except* _____.
   a. you save it with a .java file extension
   b. you compile it into bytecode, creating a file with a .class extension
   c. it can contain decisions, loops, arrays, and all the other language elements of Java
   d. it requires a `main()` method

2. How is a `JApplet` different from an application that instantiates a `JFrame`?
   a. `JFrames` use graphics.
   b. `JApplets` run from another program.
   c. `JFrames` are larger than `JApplets`.
   d. The user cannot close a `JApplet`; the program must issue a command to do so.

3. A program that allows you to display HTML documents on your computer screen is a _____.

   a. search engine
   b. compiler
   c. browser
   d. server

4. HTML contains commands that allow you to do all of the following except _____.

   a. add `JButtons` to `JApplets`
   b. format text on a Web page
   c. import graphic images to a Web page
   d. link your page to other Web pages

5. Pairs of HTML commands are called _____.

   a. tickets
   b. labels
   c. tags
   d. keywords

6. The name of any applet called using `code` within an HTML document must use the extension _____.

   a. .exe
   b. .code
   c. .java
   d. .class

7. Which `JApplet` method can you override in your extended version?

   a. `init()`
   b. `stop()`
   c. `paint()`
   d. all of the above

8. A `JApplet` is a(n) _____.

   a. `Container`
   b. `Component`
   c. `Object`
   d. all of the above

9. The first method called in any `JApplet` is the _____ method.

   a. `init()`
   b. `start()`
   c. `begin()`
   d. `main()`

10. A `JApplet`'s `init()` method is closest in purpose to a `JFrame`'s _____.

    a. `main()` method
    b. `actionPerformed()` method
    c. `start()` method
    d. constructor

11. Which of the following `JApplet` methods is likely to execute the greatest number of times?

    a. `init()`
    b. `start()`
    c. `destroy()`
    d. `main()`

12. The `paint()` method is automatically called _____.

    a. before the `init()` method
    b. after the `start()` method
    c. from the `actionPerformed()` method
    d. never

13. The `start()` method called in any `JApplet` is called _____.

    a. as the first method when an applet starts
    b. when the user closes the browser
    c. when a user revisits an applet
    d. when a user leaves a Web page

14. To respond to user events within a `JApplet`, you must _____.

    a. prepare the applet to accept event messages
    b. import the `java.applet.*` package
    c. tell your applet how to respond to events
    d. accomplish both a and c

15. When a user leaves a Web page, the `JApplet` method that executes is _____.

    a. `stop()`
    b. `destroy()`
    c. `kill()`
    d. `exit()`

16. _____ describes the use of sound, images, graphics, and video in computer programs.

    a. Inheritance
    b. Multimedia
    c. Art
    d. Graphics

17. Image formats supported by Java include all of the following except _____.

    a. GIF
    b. JPEG
    c. MIDI
    d. PNG

18. Unlike using the `Image` class, objects of the `ImageIcon` class _____.
    a. are harder to work with
    b. cannot be extended
    c. can be placed on components such as `JButtons`
    d. cannot be created by calling the class constructor

19. The `Applet` class method that retrieves sound is _____.
    a. `sound()`
    b. `music()`
    c. `record()`
    d. `play()`

20. The method that plays a sound repeatedly is _____.
    a. `play()`
    b. `loop()`
    c. `repeat()`
    d. `continue()`

Exercises

Programming Exercises

For each `JApplet` you create in the following exercises, create an HTML host document named **Test** plus the `JApplet` name. For example, the host document for the **JRiddle.java** file is named **TestJRiddle.html**.

1. Create a `JApplet` with a `JLabel` that contains a riddle and a `JButton`. When the user clicks the button, display the answer in a large font on another `JLabel`. Save the file as **JRiddle.java**.

2. a. Create a `JApplet` that asks a user to enter a password into a `JTextField` and to then press Enter. Compare the password to *Rosebud*; if it matches exactly, display "Access Granted". If not, display "Access Denied". Save the file as **JPasswordA.java**.

   b. Modify the password applet in Exercise 2a to ignore differences in case between the typed password and *Rosebud*. Save the file as **JPasswordB.java**.

   c. Modify the password applet in Exercise 2b to compare the password to a list of five valid passwords: *Rosebud*, *Redrum*, *Jason*, *Surrender*, or *Dorothy*. Save the file as **JPasswordC.java**.

3. Create a `JApplet` that contains a `JLabel` and `JButton`. When the user clicks the `JButton`, change the font typeface, style, and size on the `JLabel`. Save the file as **JChangeFont.java**.

4. Create a `JApplet` that contains a `JButton` and a `JTextField`. When the user clicks the `JButton`, display "Today is ", the date, and the time in the `JTextField`. Save the file as **JDateAndTime.java**.

5. Create a `JApplet` that allows a user to type a product number in a `JTextField`. Compare the product number to an array of at least eight valid product numbers. If the typed number is valid, display a product description and price; otherwise, display an appropriate error message. Save the file as **JProductFinder.java**.

6. Create a `JApplet` that contains two parallel arrays with at least five friends' names and phone numbers. Allow the user to enter either a name or phone number and to click a `JButton` to display the other. Include a `JLabel` to describe each `JTextField`. Save the file as **JFriendsPhones.java**.

7. Create a `JApplet` that initially displays a single `JButton`. When the user clicks the `JButton`, display a `JLabel` that prompts the user to enter an integer, a `JTextField` into which the user can type the integer, and a second `JButton` that contains the text *Double Me*. When the user clicks the second button, the integer is doubled and the answer is displayed in the `JTextField`. Save the file as **JDouble.java**.

8. Create a `JApplet` named `JBMICalculator` that allows the user to enter a height in inches and a weight in pounds. When the user clicks a `JButton`, the body mass index (BMI) is displayed. BMI is calculated by dividing weight by height squared and multiplying the result by 703. Save the file as **JBMICalculator.java**.

9. Write a `JApplet` that uses the `ImageIcon` class to place `ImageIcon` objects on four `JButtons`. Download any free JPEG or GIF files from the Internet; if necessary, reduce the size of the images to approximately 30 by 30 pixels. Alternatively, you can use the four files named up.jpg, down.jpg, left.jpg, and right.jpg in your downloadable student files. Each time a `JButton` is clicked, display a different message below the `JButtons`. Save the file as **JButtonIcons.java**.

10. Create a `JApplet` that paints an `ImageIcon` the first time its `paint()` method is called, and then randomly draws small filled ovals in the background color over the image each time a `JButton` is clicked. The resulting effect is that the image seems to be erased by an increasing number of small overlapping ovals. For example, if you place an `ImageIcon` at the coordinates contained in variables named `startPosX` and `startPosY`, you can create a series of 15-by-10 filled ovals placed randomly on the `ImageIcon`'s surface using the following `for` loop:

```
for(int count = 0; count < 20; ++count)
{
   int x = (int) (Math.random() * imageWidth) + startPosX;
   int y = (int) (Math.random() * imageHeight) + startPosY;
   g.fillOval(x, y, 10, 10);
}
```

Save the file as **JEraseImage.java**.

11. Create a `JApplet` for Business Associates, a consulting firm whose motto is "Let Business Associates take care of your business." Include two `JButtons`: clicking one `JButton` plays the tune "Taking Care of Business" continuously, and clicking the other stops the music. You can find the business.mid file in your downloadable student files. Save the file as **JBusiness.java**.

12. Write an applet that prompts the user for a color name. If it is not red, white, or blue, throw an `Exception`. Otherwise, change the applet's background color appropriately. Create an HTML document to host the `JApplet`. Save the file as **RWBApplet.java**.

13. Write a `JApplet` that uses a `JPanel` to show the messages "Mouse Entered Applet" and "Mouse Exited Applet" when the mouse enters and exits the applet. Also, when the mouse is clicked on the applet, a message "Mouse Clicked Here" should appear near the clicked location. Save the file as **JMouse.java**.

Debugging Exercises

1. Each of the following files in the Chapter17 folder of your downloadable student files has syntax and/or logic errors. In each case, determine the problem and fix the program. After you correct the errors, save each file using the same filename preceded with *Fix*. For example, DebugSeventeen1.java will become FixDebugSeventeen1.java. You can test each applet with four downloadable student files named TestFixDebugSeventeen1.html through TestFixDebugSeventeen4.html.

   a. DebugSeventeen1.java
   b. DebugSeventeen2.java
   c. DebugSeventeen3.java
   d. DebugSeventeen4.java

Game Zone

1. In Chapter 2, you created a *Mad Libs* game in which the user entered several words out of context that were then inserted into a rhyme or story, producing humorous effects. Modify the game so it becomes an applet. In turn, prompt the user for each required word. After all the words are entered, display the completed rhyme or story. Figure 17-22 shows the first and last screens displayed during a typical execution. Save the applet as **JMadLib.java**.

Figure 17-22 First and last screens in a typical execution of the `JMadLib` applet

2. In Chapter 5, you created a Rock Paper Scissors game. Now create it as a `JApplet` in which the user can click one of three buttons labeled "Rock", "Paper", or "Scissors". The computer's choice is still randomly generated. Figure 17-23 shows a typical execution after the user has played a few games. Save the applet as **JRockPaperScissors.java**.

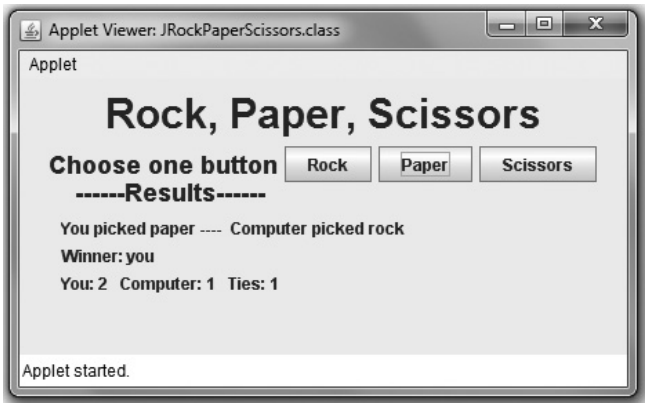

Figure 17-23 Typical execution of the `JRockPaperScissors` applet

3. In earlier chapters, you created and used a `Card` class in which each object represents a playing `Card`, and in Chapter 8 you constructed a deck of 52 unique `Card`s. Create a `JApplet` that uses such a deck. Start the player with a $10 stake. Randomly deal a card to the computer, and allow the player to make one of four bets:

   - $5 that the player's card will be higher than the computer's card
   - $10 that the player's card will be higher than the computer's card
   - $5 that the player's card will be lower than the computer's card
   - $10 that the player's card will be lower than the computer's card

   After the player makes a bet, deal the player's card and add or subtract the correct amount from the player's winnings based on the results. When the computer's and player's cards are the same value, the computer wins. The game ends when the first of these events happens:

   - The player goes broke (the winnings go to $0 or below).
   - The player wins $100.
   - All 52 cards are exhausted without a winner.

As shown in the game in progress in Figure 17-24, when the player is making a bet, enable four betting buttons. After the player makes a bet, disable the betting buttons while the player examines the outcome, and enable an OK button. When the player is ready to resume play and clicks OK, disable the OK button and, if the game isn't over, enable the four betting buttons. Save the game as **JCardBet.java**.

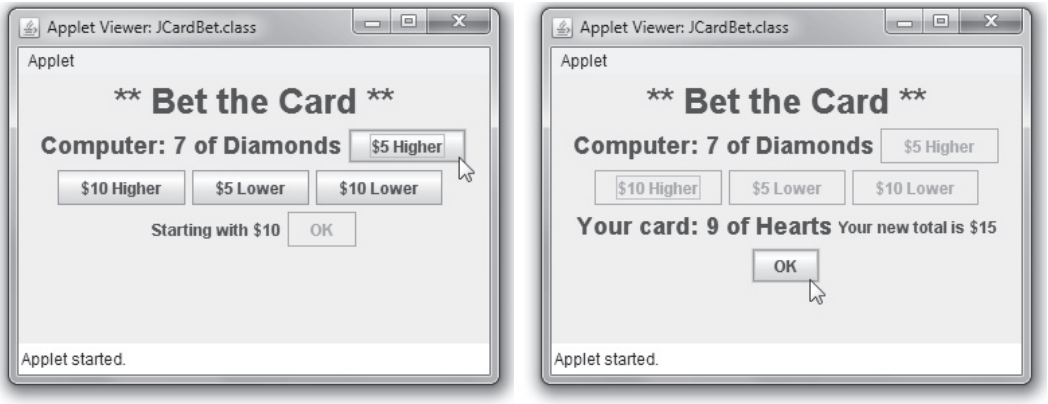

Figure 17-24 Typical execution of the `JCardBet` applet

4. a. In Chapters 10 and 11, you created an `Alien` class and `Martian` and `Jupiterian` classes that descend from it. Add a `draw()` method for each child class. The `draw()` method accepts a `Graphics` object and x- and y- starting coordinates. The method draws the `Aliens` in any way you choose, using lines, ovals, rectangles, and so on. Using the `drawString()` method, include a description that names each drawing. Save the files as **Martian.java** and **Jupiterian.java**.

b. Create an applet that instantiates a `Martian` and a `Jupiterian`. In the applet's `paint()` method, draw each type of `Alien`. Save the file as **JDemoAliens.java**. Figure 17-25 shows some sample `Aliens`, but your `Aliens` might look very different.

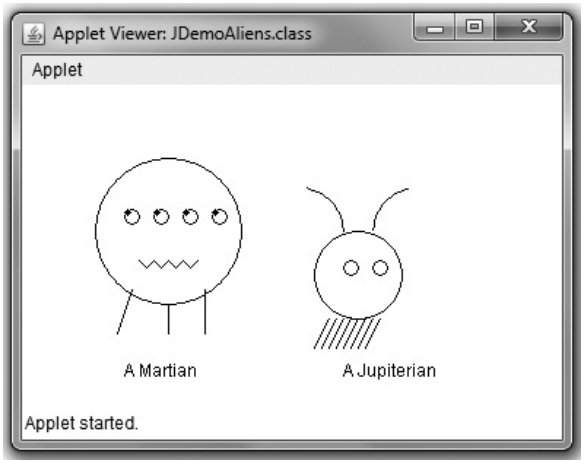

Figure 17-25 Some aliens

c. Create an applet that contains an Alien Hunt game. Place eight numbered buttons in the applet. Randomly assign `Martians` to six of the buttons and `Jupiterians` to two. (*Hint*: You do not need to create an `Alien` array; you can simply create an array that randomly contains *0*s and *1*s, representing `Martians` and `Jupiterians`.) The object of the game is to find all the `Martians` before finding both `Jupiterians`. When a user clicks a button, display the `Alien` represented by the button. If the user clicks two `Jupiterians` before clicking six `Martians`, the player loses the game. When this happens, display two `Jupiterians` and a message telling the user that Earth has been destroyed. Disable any button after it has been selected. Save the game as **JAlienHunt.java**.

5. a. In Chapter 4, you created a `Die` class that you can use to instantiate objects that hold one of six values. Create a `GraphicDie` class that descends from `Die` but adds a `drawDie()` method that draws a representation of the die on the screen. Design the method so it accepts a `Graphics` object as well as x- and y-coordinate positions where the drawing should be placed. Create the drawing of a `Die` based on its value and using the `drawRect()` and `fillOval()` methods. Save the file as **GraphicDie.java**.

b. Create a `JGraphicDie JApplet` that instantiates a `GraphicDie` object. In the `JApplet`'s `paint()` method, pass the method's `Graphics` object and two values to the `GraphicDie` object's `drawDie()` method. Save the file as **JGraphicDie.java**.

c. In Chapter 8, you created a `FiveDice3` game in which a player's random roll of five dice is compared to the computer's roll and a winner is determined. Now create an applet that plays the game. At each roll, initiated by a player's button click, display the player's five dice and the computer's five dice. Save the file as **JFiveDice.java**. Figure 17-26 shows a typical game.

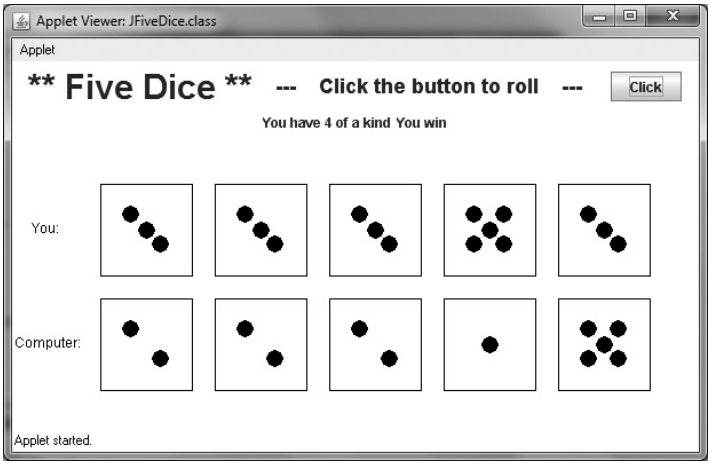

Figure 17-26 The `JFiveDice` applet

Case Problems

1. Throughout this book you have created applications for Carly's Catering. Now create a `JApplet` that advertises the business. Use at least one animation file and one sound file in your `JApplet`. Save the `JApplet` as **JCarlys.java**, and save the HTML host file as **TestJCarlys.html**.

2. Throughout this book you have created applications for Sammy's Seashore Rentals. Now create a `JApplet` that advertises the business. Use at least one animation file and one sound file in your `JApplet`. Save the `JApplet` as **JSammys.java**, and save the HTML host file as **TestJSammys.html**.