CHAPTER 25

Applets and More

TOPICS

NOTE: Java applets are an older technology, and Oracle has announced that beginning with Java 9, the browser plug-in needed to support Java applets has been discontinued. This is a legacy chapter from the book, and is provided for students and instructors making the transition from applets to other technologies. As you read this chapter, keep in mind that newer technologies exist as alternatives to applets.

25.1 Introduction to Applets

CONCEPT: An applet is a Java program that is associated with a Web page and is executed in a Web browser as part of that Web page.

Recall from Chapter 1 that there are two types of programs you can create with Java: applications and applets. An *application* is a stand-alone program that runs on your computer. So far in this text, we have concentrated exclusively on writing applications.

Applets are Java programs that are usually part of a Web site. If a user opens the Web site with a Java-enabled browser, the applet is executed inside the browser window. It appears to the user that the applet is part of the Web site. This is how it works: Applets are stored on a Web server along with the site's Web pages. When a user accesses a Web page on a server with his or her browser, any applets associated with the Web page are transmitted over the Internet from the server to the user's system. This is illustrated in Figure 25-1. Once the applets are transmitted, the user's system executes them.

Applets are important because they can be used to extend the capabilities of a Web page. Web pages are normally written in Hypertext Markup Language (HTML). HTML is limited, however, because it merely describes the content and layout of a Web page, and creates links

to other files and Web pages. HTML does not have sophisticated abilities such as performing math calculations and interacting with the user. A programmer can write a Java applet to perform these types of operations and associate it with a Web page. When someone visits the Web page, the applet is downloaded to the visitor's browser and executed.

Figure 25-1 Applets are transmitted along with Web pages

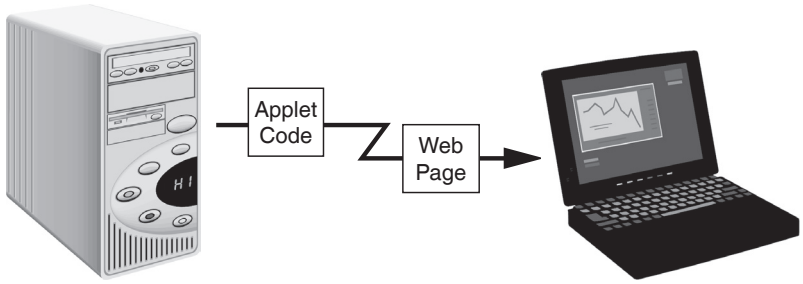
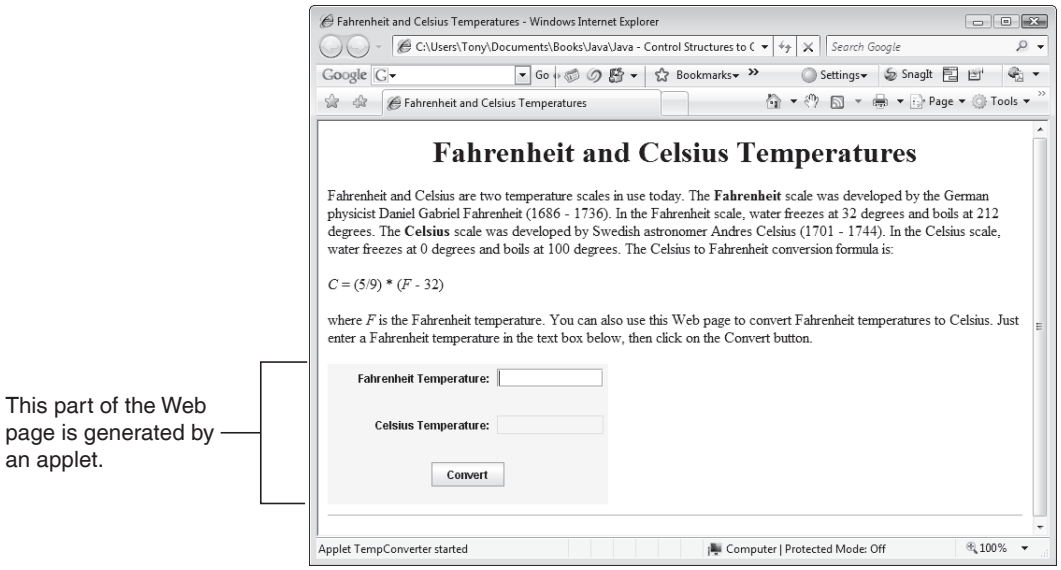


Figure 25-2 shows an example of a Web page that has an applet. In the figure, the Web page is being viewed with Internet Explorer. This Web page briefly explains the Fahrenheit and Celsius temperature scales. The area with the text boxes and the button at the bottom of the page is generated by an applet. To see a Fahrenheit temperature converted to Celsius, the user can enter the Fahrenheit temperature into the top text box and click the Convert button. The Celsius temperature will be displayed in the read-only text box.

An applet does not have to be on a Web server to be executed. The Web page shown in Figure 25-2 is in the source code folder *Chapter 19\TempConverter*. Open the *TempConverter.html* file in your Web browser to try it. Later in this chapter, we will take a closer look at this Web page and its applet.

Figure 25-2 A Web page with an applet (Microsoft Corporation)



Most Web browsers have a special version of the JVM for running applets. For security purposes, this version of the JVM greatly restricts what an applet can do. Here is a summary of the restrictions placed on applets:

- Applets cannot delete files, read the contents of files, or create files on the user's system.
- Applets cannot run any other program on the user's system.
- Applets cannot execute operating system procedures on the user's system.
- Applets cannot retrieve information about the user's system, or the user's identity.
- Applets cannot make network connections with any system except the server from which the applet was transmitted.
- If an applet displays a window, it will automatically have a message such as "Warning: Applet Window" displayed in it. This lets the user know that the window was not displayed by an application on his or her system.

These restrictions might seem severe, but they are necessary to prevent malicious code from attacking or spying on unsuspecting users. If an applet attempts to violate one of these restrictions, an exception is thrown.

Checkpoint

MyProgrammingLab™ *www.myprogramminglab.com*

25.1 How is an applet that is associated with a Web page executed on a user's system?

25.2 Why do applets run in a restricted environment?

25.2 A Brief Introduction to HTML

CONCEPT: When creating a Web page, you use Hypertext Markup Language (HTML) to create a file that can be read and processed by a Web browser.

Hypertext Markup Language (HTML) is the language that Web pages are written in. Although it is beyond the scope of this text to teach you everything about HTML, this section will give you enough of the fundamentals so that you can write simple Web pages. You will need to know a little about HTML in order to run Java applets. If you are already familiar with HTML, this section is optional.

Before we continue, let's look at the meanings of the terms *hypertext* and *markup language*.

Hypertext

Web pages can contain regular text and hypertext, which are both displayed in the browser window. In addition, *hypertext* can contain a link to another Web page, or perhaps another location in the same Web page. When the user clicks on the hypertext, it loads the Web page or the location that the hypertext is linked to.

Markup Language

Although HTML is called a language, it is not a programming language like Java. Instead, HTML is a *markup language*. It allows you to "mark up" a text file by inserting special instructions. These instructions tell the browser how to format the text and create any hypertext links.

To make a Web page, you create a text file that contains HTML instructions, which are known as *tags*, as well as the text that should be displayed on the Web page. The resulting file is known as an *HTML document*, and it is usually saved with the *.html* file name extension. When a Web browser reads the HTML document, the tags instruct it how to format the text, where to place images, what to do when the user clicks on a link, and more.

Most HTML tags come in pairs. The first is known as the opening tag and the second is known as the closing tag. The general format of a simple tag is as follows:

```
<tag_name>
Text
</tag_name>
```

In this general format, *tag_name* is the name of the tag. The opening tag is *<tag_name>* and the closing tag is *</tag_name>*. Both the opening and closing tags are enclosed in angle brackets (< >). Notice that in the closing tag, the tag name is preceded by a forward slash (/). The Text that appears between the opening and closing tags is text that is formatted or modified by the tags.

Document Structure Tags

Some of the HTML tags are used to establish the structure of an HTML document. The first of the structure tags that you should learn is the <html></html> tag. This tag marks the beginning and ending of an HTML document. Everything that appears between these tags, including other tags, is the content of the Web page. When you are writing an HTML document, place an <html> tag at the very beginning, and an </html> tag at the very end.

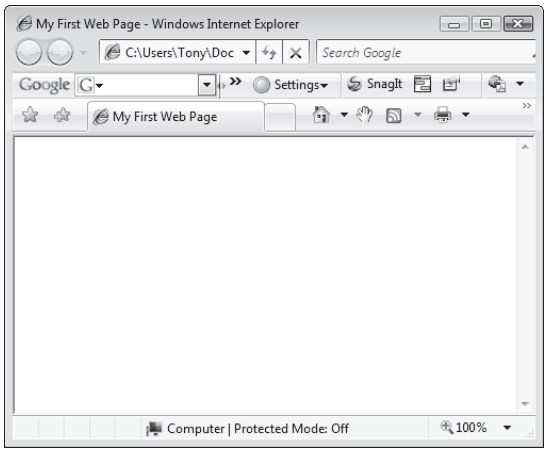
The next tag is <head></head>. Everything that appears between <head> and </head> is considered part of the document head. The *document head* is a section of the HTML file that contains information about the document. For example, key words that search engines use to identify a document are often placed in the document's head. The only thing that we will use the document head for is to display a title in the Web browser's title bar. You do this with the <title></title> tag. Any text that you place between <title> and </title> becomes the title of the page and is displayed in the browser's title bar. Code Listing 25-1 shows the contents of an HTML document with the title "My First Web Page".

Notice that the <title></title> tag is inside of the <head></head> tag. The only output displayed by this Web page is the title. Figure 25-3 shows how this Web page appears when opened in a browser.

Code Listing 25-1 (BasicWebPage1.html)

```
<html>
<head>
    <title>My First Web Page</title>
</head>
</html>
```

Figure 25-3 Web page with a title only (Microsoft Corporation)

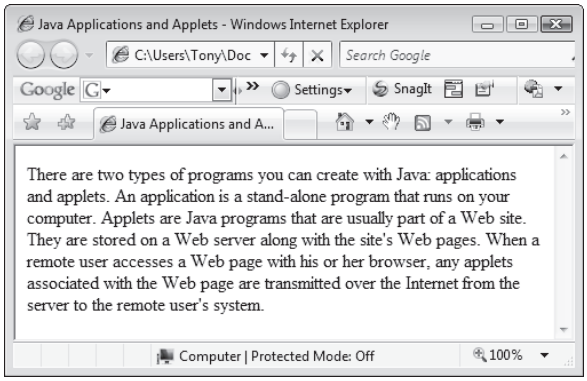


After the document head comes the document body, which is enclosed in the <body></body> tag. The *document body* contains all of the tags and text that produce output in the browser window. Code Listing 25-2 shows an HTML document with text placed in its body. Figure 25-4 shows the document when opened in a browser.

Code Listing 25-2 (BasicWebPage2.html)

```
<html>
<head>
    <title>Java Applications and Applets</title>
</head>
<body>
    There are two types of programs you can create with Java: applications
    and applets. An application is a stand-alone program that runs on your
    computer. Applets are Java programs that are usually part of a Web site.
    They are stored on a Web server along with the site's Web pages. When a
    remote user accesses a Web page with his or her browser, any applets
    associated with the Web page are transmitted over the Internet from the
    server to the remote user's system.
</body>
</html>
```

Figure 25-4 Web page produced by *BasicWebPage2.html* (Microsoft Corporation)



Text Formatting Tags

The text displayed in the Web page in Figure 25-4 is unformatted, which means it appears as plain text. There are many HTML tags that you can use to change the appearance of text. For example, there are six different header tags that you can use to format text as a heading of some type. The `<h1></h1>` tag creates a level one header. A level one header appears in boldface, and is much larger than regular text. The `<h2></h2>` tag creates a level two header. A level two header also appears in boldface, but is smaller than a level one header. This pattern continues with the `<h3></h3>`, `<h4></h4>`, `<h5></h5>`, and `<h6></h6>` tags. The higher a header tag's level number is, the smaller the text that it formats appears. For example, look at the following HTML:

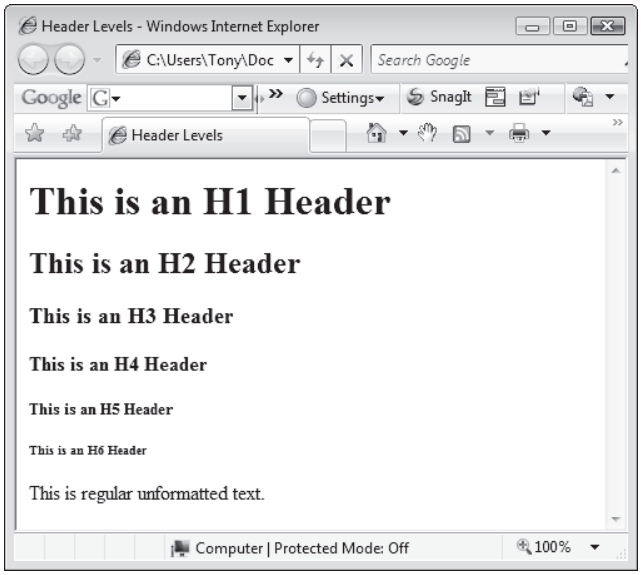
```
<h1>This is an h1 Header</h1>
<h2>This is an h2 Header</h2>
<h3>This is an h3 Header</h3>
<h4>This is an h4 Header</h4>
<h5>This is an h5 Header</h5>
<h6>This is an h6 Header</h6>
This is regular unformatted text.
```

When this appears in the body of an HTML document, it produces the Web page shown in Figure 25-5.

You can use the `<center></center>` tag to center a line of text in the browser window. To demonstrate, we will add the following line to the document that was previously shown in Code Listing 25-2:

```
<center><h1>Java</h1></center>
```

Figure 25-5 Header levels (Microsoft Corporation)

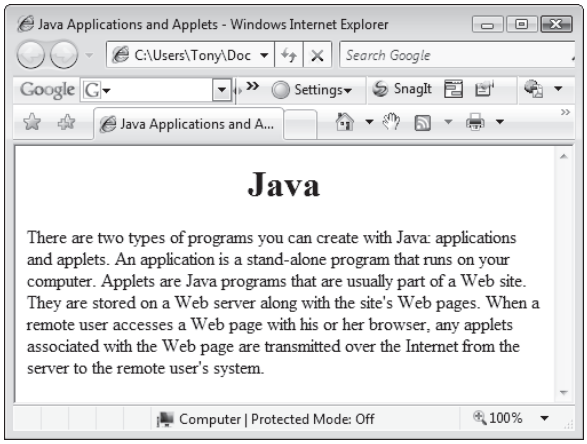


This will cause the word "Java" to appear centered and as a level one header. The modified document is shown in Code Listing 25-3, and the Web page it produces is shown in Figure 25-6.

Code Listing 25-3 (BasicWebPage3.html)

```
<html>
<head>
    <title>Java Applications and Applets</title>
</head>
<body>
    <center>
        <h1>Java</h1>
    </center>
    There are two types of programs you can create with Java: applications
    and applets. An application is a stand-alone program that runs
    on your computer. Applets are Java programs that are usually
    part of a Web site. They are stored on a Web server along with
    the site's Web pages. When a remote user accesses a Web page
    with his or her browser, any applets associated with the Web
    page are transmitted over the Internet from the server to the
    remote user's system.
</body>
</html>
```

Figure 25-6 Web page produced by *BasicWebPage3.html* (Microsoft Corporation)



Notice that in the HTML document, the word "Java" is enclosed in two sets of tags: the `<center>` tags and the `<h1>` tags. It doesn't matter which set of tags is used first. If we had written the line as follows, we would have gotten the same result:

```
<h1><center>Java</center></h1>
```

You can display text in boldface by using the `<b></b>` tag, and in italics by using the `<i></i>` tag. For example, the following will cause the text "Hello World" to be displayed in boldface:

```
<b>Hello World</b>
```

The following will cause "Hello World" to be displayed in italics:

```
<i>Hello World</i>
```

The following will display "Hello World" in boldface and italics:

```
<b><i>Hello World</i></b>
```

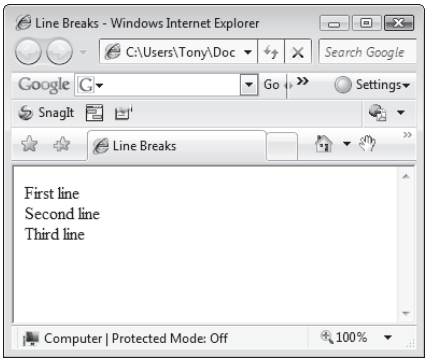
Creating Breaks in Text

We will look at three HTML tags that are used to create breaks in a document's text. These three tags are unique from the ones we previously studied because they do not occur in pairs. When you use one of these tags, you only insert an opening tag.

The `<br />` tag causes a line break to appear at the point in the text where it is inserted. It is often necessary to insert `<br />` tags in an HTML document because the browser usually ignores the newline characters that are created when you press the Enter key. For example, if the following line appears in the body of an HTML document, it will cause the output shown in Figure 25-7.

```
First line<br />Second line<br />Third line
```

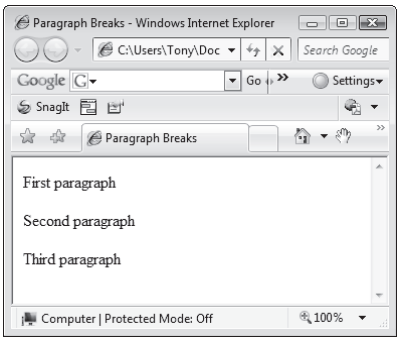
Figure 25-7 Line breaks in an HTML document (Microsoft Corporation)



The <p /> tag causes a paragraph break to appear at the point in the text where it is inserted. A paragraph break typically inserts more space into the text than a line break. For example, if the following line appears in the body of an HTML document, it will cause the output shown in Figure 25-8.

```
First paragraph<p />Second paragraph<p />Third paragraph
```

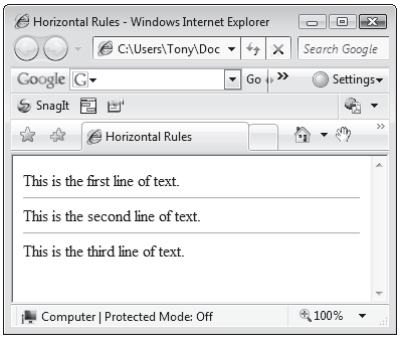
Figure 25-8 Paragraph breaks in an HTML document (Microsoft Corporation)



The <hr /> tag causes a horizontal rule to appear at the point in the text where it is inserted. A horizontal rule is a thin, horizontal line that is drawn across the Web page. For example, if the following text appears in the body of an HTML document, it will cause the output shown in Figure 25-9.

```
This is the first line of text.
<hr />
This is the second line of text.
<hr />
This is the third line of text.
```

Figure 25-9 Horizontal rules in a Web page (Microsoft Corporation)

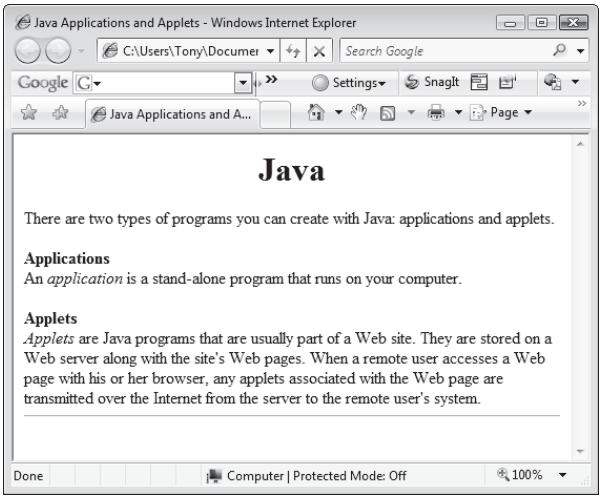


The HTML document shown in Code Listing 25-4 demonstrates each of the tags we have discussed. The Web page it produces is shown in Figure 25-10.

Code Listing 25-4 (BasicWebPage4.html)

```
<html>
<head>
    <title>Java Applications and Applets</title>
</head>
<body>
    <center>
        <h1>Java</h1>
    </center>
    There are two types of programs you can create with Java: applications
    and applets.
    <p />
    <b>Applications</b>
    <br />
    An <i>application</i> is a stand-alone program that runs on
    your computer.
    <p />
    <b>Applets</b>
    <br />
    <i>Applets</i> are Java programs that are usually part of a
    Web site. They are stored on a Web server along with the site's
    Web pages. When a remote user accesses a Web page with his or
    her browser, any applets associated with the Web page are
    transmitted over the Internet from the server to the remote
    user's system.
    <hr />
</body>
</html>
```

Figure 25-10 Web page produced by *BasicWebPage4.html* (Microsoft Corporation)



Inserting Links

As previously mentioned, a link is some element in a Web page that can be clicked on by the user. When the user clicks the link, another Web page is displayed, or some sort of action is initiated. We now look at how to insert a simple link that causes another Web page to be displayed. The tag that is used to insert a link has the following general format:

```
<a href="Address">Text</a>
```

The *Text* that appears between the opening and closing tags is the text that will be displayed in the Web page. When the user clicks on this text, the Web page that is located at *Address* will be displayed in the browser. This address is often referred to as a *uniform resource locator* (*URL*). Notice that the address is enclosed in quotation marks. Here is an example:

```
<a href="http://www.pearsonhighered.com/gaddis/">Click here to go to the
textbook's web site.</a>
```

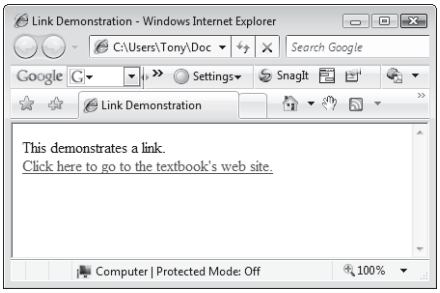
The HTML document shown in Code Listing 25-5 uses this link, and Figure 25-11 shows how the page appears in the browser.

Code Listing 25-5 (LinkDemo.html)

```
<html>
<head>
    <title>Link Demonstration</title>
</head>
<body>
    This demonstrates a link.
    <br />
    <a href="http://www.aw.com/gaddis">Click here to go to the textbook's
    web site.</a>
</body>
</html>
```

The text that is displayed by a link is usually highlighted in some way to let the user know that it is not ordinary text. In Figure 25-11, the link text is underlined. When the user clicks on this text, the browser displays the Web page at `www.aw.com/gaddis`

Figure 25-11 Web page produced by *LinkDemo.html* (Microsoft Corporation)



Checkpoint

MyProgrammingLab™ *www.myprogramminglab.com*

25.3 What tag marks the beginning and end of an HTML document?

25.4 What tag marks the beginning and end of an HTML document's head section?

25.5 What statement would you use in an HTML document to display the text "My Web Page" in the browser's title bar? What section of the HTML document would this statement be written in?

25.6 What tag marks the beginning and end of an HTML document's body section?

25.7 What statement would you write in an HTML document to display the text "Student Roster" as a level one header?

25.8 What statement would you write in an HTML document to display the text "My Resume" in bold and centered on the page?

25.9 What statement would you write in an HTML document to display the text "Hello World" in bold and italic?

25.10 What tag causes a line break? What tag causes a paragraph break? What tag displays a horizontal rule?

25.11 Suppose you wanted to display the text "Click Here" as a link to the Web site `http://java.sun.com`. What statement would you write to create the text?

25.3 Creating Applets with Swing

CONCEPT: You extend a class from `JApplet` to create an applet, just as you extend a class from `JFrame` to create a GUI application.

By now you know almost everything necessary to create an applet. That is because applets are very similar to GUI applications. You can think of an applet as a GUI application that runs under the control of a Web browser. Instead of displaying its own window, an applet

appears in the browser's window. The differences between GUI application code and applet code are summarized here:


VideoNote
Creating an Applet

- A GUI application class inherits from `JFrame`. An applet class inherits from `JApplet`. The `JApplet` class is part of the `javax.swing` package.
- A GUI application class has a constructor that creates other components and sets up the GUI. An applet class does not normally have a constructor. Instead, it has a method named `init` that performs the same operations as a constructor. The `init` method accepts no arguments and has a `void` return type.
- The following methods, which are commonly called in a GUI application's constructor, are not called in an applet:

  ```
  setTitle
  setSize
  setDefaultCloseOperation
  pack
  setVisible
  ```

The methods listed here are used in a GUI application to affect the application's window in some way. They are not usually applicable to an applet because the applet does not have a window of its own.

- There is no static `main` method needed to create an instance of the applet class. The browser creates an instance of the class automatically.

Let's look at a simple applet. Code Listing 25-6 shows an applet that displays a label.

Code Listing 25-6 (SimpleApplet.java)

```
1  import javax.swing.*;
2  import java.awt.*;
3
4  /**
5     This is a simple applet.
6  */
7
8  public class SimpleApplet extends JApplet
9  {
10    /**
11       The init method sets up the applet, much
12       like a constructor.
13    */
14
15    public void init()
16    {
17       // Create a label.
18       JLabel label =
19            new JLabel("This is my very first applet.");
20
```

```
21        // Set the layout manager.
22        setLayout(new FlowLayout());
23
24        // Add the label to the content pane.
25        add(label);
26     }
27  }
```

This code is very much like a regular GUI application. Although this class extends `JApplet` instead of `JFrame`, you still add components to the content pane and use layout managers in the same way.

Running an Applet

The process of running an applet is different from that of running an application. To run an applet, you create an HTML document with an `applet` tag, which has the following general format:

```
<applet code="Filename.class" width=Wide height=High></applet>
```

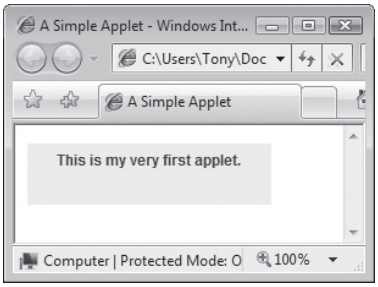
In the general format, *`Filename.class`* is the name of the applet's *.class* file. This is the file that contains the compiled byte code. Note that you do not specify the *.java* file, which contains the Java source code. You can optionally specify a path along with the file name. If you specify only the file name, it is assumed that the file is in the same directory as the HTML document. *`Wide`* is the width of the applet in pixels, and *`High`* is the height of the applet in pixels. When a browser processes an `applet` tag, it loads specified byte code and executes it in an area that is the size specified by the *`Wide`* and *`High`* values.

The HTML document shown in Code Listing 25-7 uses an `applet` tag to load the applet shown in Code Listing 25-6. This document specifies that the applet should be displayed in an area that is 200 pixels wide by 50 pixels high. Figure 25-12 shows this document when it is displayed in a Web browser.

Code Listing 25-7 (`SimpleApplet.html`)

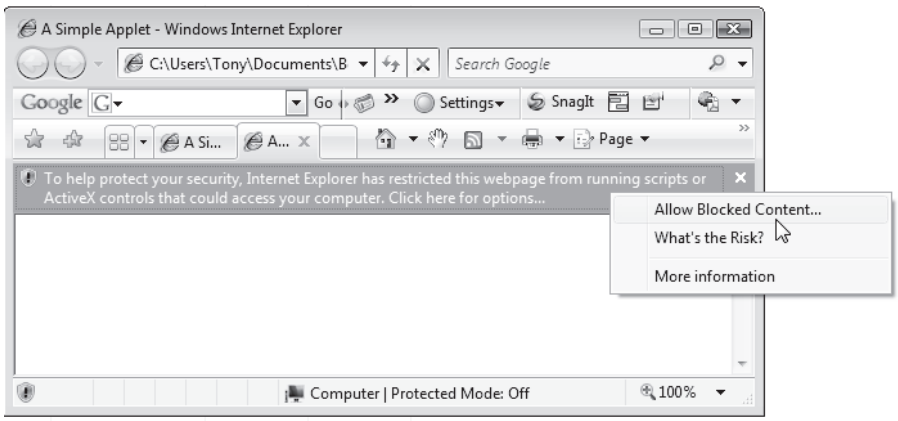
```
<html>
<head>
    <title>A Simple Applet</title>
</head>
<body>
    <applet code="SimpleApplet.class" width="200" height="50">
    </applet>
</body>
</html>
```

Figure 25-12 The Web page produced by *SimpleApplet.html* (Microsoft Corporation)



NOTE: When you load a Web page that uses an applet into your browser, you will most likely get a security warning. For example, Figure 25-13 shows the warning you get from Internet Explorer. To run the applet, click the warning message and then select Allow Blocked Content . . . from the pop-up menu that appears.

Figure 25-13 Security warning in Internet Explorer (Microsoft Corporation)



Running an Applet with `appletviewer`

The Sun JDK comes with an applet viewer program that loads and executes an applet without the need for a Web browser. This program can be run from a command prompt with the `appletviewer` command. When you run the program, you specify the name of an HTML document as a command line argument. For example, the following command passes `SimpleApplet.html` as the command line argument:

```
appletviewer SimpleApplet.html
```

This command executes any applet that is referenced by an `applet` tag in the file *SimpleApplet.html*. The window shown in Figure 25-14 will be displayed.

Figure 25-14 Applet executed by `appletviewer` (Oracle Corporate Counsel)

NOTE: The applet viewer does not display any output generated by text or tags in the HTML document. It only executes applets. If the applet viewer opens an HTML document with more than one `applet` tag, it will execute each applet in a separate window.

Handling Events in an Applet

In an applet, events are handled with event listeners exactly as they are in GUI applications. To demonstrate, we will examine the `TempConverter` class, which is shown in Code Listing 25-8. This class is the applet displayed in the Web page we examined at the beginning of this chapter. It has a text field where the user can enter a Fahrenheit temperature and a Convert button that converts the temperature to Celsius and displays it in a read-only text field. The temperature conversion is performed in an action listener class that handles the button's action events.

Code Listing 25-8 (`TempConverter.java`)

```
1  import javax.swing.*;
2  import java.awt.*;
3  import java.awt.event.*;
4
5  /**
6     The TempConverter class is an applet that converts
7     Fahrenheit temperatures to Celsius.
8  */
9
10 public class TempConverter extends JApplet
11 {
12    private JPanel fPanel;         // To hold a text field
13    private JPanel cPanel;         // To hold a text field
14    private JPanel buttonPanel;    // To hold a button
15    private JTextField fahrenheit; // Fahrenheit temperature
16    private JTextField celsius;    // Celsius temperature
17
18    /**
19       init method
20    */
21
22    public void init()
23    {
24       // Build the panels.
25       buildFpanel();
```

```
26       buildCpanel();
27       buildButtonPanel();
28
29       // Create a layout manager.
30       setLayout(new GridLayout(3, 1));
31
32       // Add the panels to the content pane.
33       add(fPanel);
34       add(cPanel);
35       add(buttonPanel);
36    }
37
38    /**
39       The buildFpanel method creates a panel with a text
40       field in which the user can enter a Fahrenheit
41       temperature.
42    */
43
44    private void buildFpanel()
45    {
46       // Create the panel.
47       fPanel = new JPanel();
48
49       // Create a label to display a message.
50       JLabel message1 =
51                 new JLabel("Fahrenheit Temperature:");
52
53       // Create a text field for the Fahrenheit temp.
54       fahrenheit = new JTextField(10);
55
56       // Create a layout manager for the panel.
57       fPanel.setLayout(new FlowLayout(FlowLayout.RIGHT));
58
59       // Add the label and text field to the panel.
60       fPanel.add(message1);
61       fPanel.add(fahrenheit);
62    }
63
64    /**
65       The buildCpanel method creates a panel that
66       displays the Celsius temperature in a
67       read-only text field.
68    */
69
70    private void buildCpanel()
71    {
72       // Create the panel.
73       cPanel = new JPanel();
```

```
74
75       // Create a label to display a message.
76       JLabel message2 =
77               new JLabel("Celsius Temperature:");
78
79       // Create a text field for the celsius temp.
80       celsius = new JTextField(10);
81
82       // Make the text field read-only.
83       celsius.setEditable(false);
84
85       // Create a layout manager for the panel.
86       cPanel.setLayout(new FlowLayout(FlowLayout.RIGHT));
87
88       // Add the label and text field to the panel.
89       cPanel.add(message2);
90       cPanel.add(celsius);
91    }
92
93    /**
94       The buildButtonPanel method creates a panel with
95       a button that converts the Fahrenheit temperature
96       to Celsius.
97    */
98
99    private void buildButtonPanel()
100   {
101      // Create the panel.
102      buttonPanel = new JPanel();
103
104      // Create a button with the text "Convert".
105      JButton convButton = new JButton("Convert");
106
107      // Add an action listener to the button.
108      convButton.addActionListener(new ButtonListener());
109
110      // Add the button to the panel.
111      buttonPanel.add(convButton);
112   }
113
114   /**
115      Private inner class that handles the action event
116      that is generated when the user clicks the convert
117      button.
118   */
119
120   private class ButtonListener implements ActionListener
```

```
121    {
122       public void actionPerformed(ActionEvent e)
123       {
124          double ftemp, ctemp;  // To hold the temperatures
125
126          // Get the Fahrenheit temperature and convert it
127          // to a double.
128          ftemp = Double.parseDouble(fahrenheit.getText());
129
130          // Calculate the Celsius temperature.
131          ctemp = (5.0 / 9.0) * (ftemp - 32);
132
133          // Display the Celsius temperature.
134          celsius.setText(String.format("%.1f",  ctemp));
135       }
136    }
137 }
```

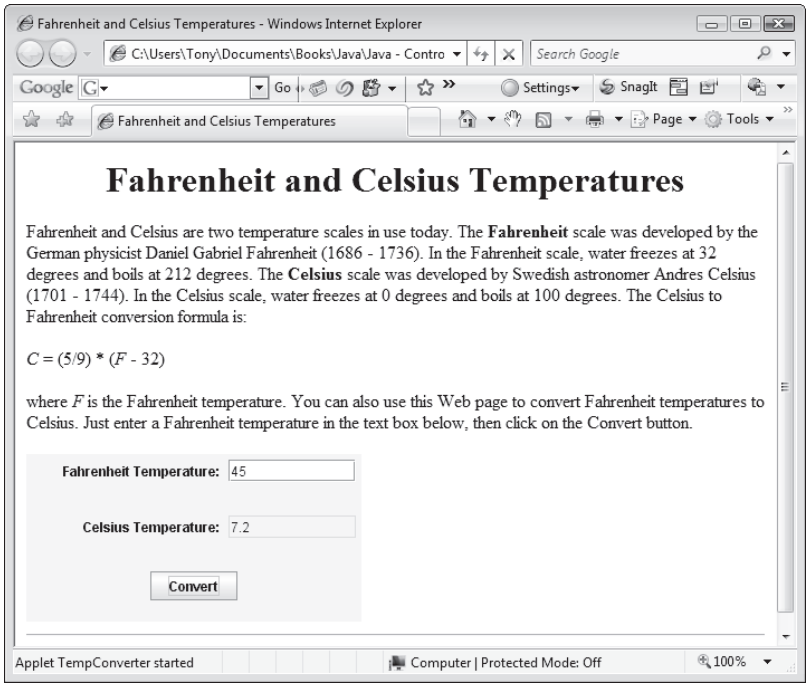
Code Listing 25-9 shows the contents of `TempConverter.html`, an HTML document that uses this applet. Figure 25-15 shows the Web page produced by this document. In the figure, the user has entered a Fahrenheit temperature and converted it to Celsius.

Code Listing 25-9 (TempConverter.html)

```
<html>
<head>
    <title>Fahrenheit and Celsius Temperatures</title>
</head>
<body>
    <center>
        <h1>Fahrenheit and Celsius Temperatures</h1>
    </center>
    Fahrenheit and Celsius are two temperature scales in use today.
    The <b>Fahrenheit</b> scale was developed by the German physicist
    Daniel Gabriel Fahrenheit (1686 - 1736). In the Fahrenheit scale,
    water freezes at 32 degrees and boils at 212 degrees. The
    <b>Celsius</b> scale was developed by Swedish astronomer Andres Celsius
    (1701 - 1744). In the Celsius scale, water freezes at 0 degrees and
    boils at 100 degrees. The Celsius to Fahrenheit conversion formula
    is:
    <p />
    <i>C</i> = (5/9) * (<i>F</i> - 32)
    <p />
    where <i>F</i> is the Fahrenheit temperature. You can also use
    this Web page to convert Fahrenheit temperatures to Celsius.
    Just enter a Fahrenheit temperature in the text box below, then
```

```
    click on the Convert button.
    <p />
    <applet code="TempConverter.class" width="300" height="150">
    </applet>
    <hr />
</body>
</html>
```

Figure 25-15 Web page produced by *TempConverter.html* (Microsoft Corporation)



Checkpoint

MyProgrammingLab™ *www.myprogramminglab.com*

25.12 Instead of `JFrame`, an applet class is extended from what class?

25.13 Instead of a constructor, an applet class uses what method?

25.14 Why is there no need for a static `main` method to create an instance of an applet class?

25.15 Suppose the file *MyApplet.java* contains the Java source code for an applet. What tag would you write in an HTML document to run the applet in an area that is 400 pixels wide by 200 pixels high?

25.4 Using AWT for Portability

CONCEPT: Applets that use Swing components may be incompatible with some browsers. If you want to make sure that an applet is compatible with all Java-enabled browsers, use AWT components instead of Swing.

Java provides two libraries of classes that GUI components may be created from. Recall from Chapter 17 that these libraries are AWT and Swing. AWT is the original library that has been part of Java since its earliest version. Swing is an improved library that was introduced with Java 2. All of the GUI applications in Chapters 17 and 18, as well as the applets we have studied so far in this chapter, use Swing classes for their components.

Some browsers, do not directly support the Swing classes in applets. These browsers require a *plug-in*, which is software that extends or enhances another program, in order to run applets that use Swing components. Fortunately, this plug-in is automatically installed on a computer when the Sun JDK is installed. If you have installed the JDK, you should be able to write applets that use Swing and run them with no problems.

If you are writing an applet for other people to run on their computers, however, there is no guarantee that they will have the required plug-in. If this is the case, you should use the AWT classes instead of the Swing classes for the components in your applet. Fortunately, the AWT component classes are very similar to the Swing classes, so learning to use them is simple if you already know how to use Swing.

There is a corresponding AWT class for each of the Swing classes that you have learned so far. The names of the AWT classes are the same as those of the Swing classes, except the AWT class names do not start with the letter `J`. For example, the AWT class to create a frame is named `Frame`, and the AWT class to create a panel is named `Panel`. Table 25-1 lists several of the AWT classes. All of these classes are in the `java.awt` package.

Table 25-1 Several AWT classes

AWT Class	Description	Corresponding Swing Class
`Applet`	Used as a superclass for all applets. Unlike `JApplet` objects, `Applet` objects do not have a content pane.	`JApplet`
`Frame`	Creates a frame container that may be displayed as a window. Unlike `JFrame` objects, `Frame` objects do not have a content pane.	`JFrame`
`Panel`	Creates a panel container.	`JPanel`
`Button`	Creates a button that may be clicked.	`JButton`
`Label`	Creates a label that displays text.	`JLabel`
`TextField`	Creates a single line text field, which the user may type into.	`JTextField`
`Checkbox`	Creates a check box that may be selected or deselected.	`JCheckBox`

The Swing classes were intentionally designed with constructors and methods that are similar to those of their AWT counterparts. In addition, events are handled in the same way for each set of classes. This makes it easy for you to use either set of classes without learning a completely different syntax for each. For example, Code Listing 25-10 shows a version of the `TempConverter` applet that has been rewritten to use AWT components instead of Swing components.

Code Listing 25-10 (AWTTempConverter.java)

```
1  import java.applet.Applet;
2  import java.awt.*;
3  import java.awt.event.*;
4
5  /**
6     The AWTTempConverter class is an applet that converts
7     Fahrenheit temperatures to Celsius.
8  */
9
10 public class AWTTempConverter extends Applet
11 {
12    private Panel fPanel;          // To hold a text field
13    private Panel cPanel;          // To hold a text field
14    private Panel buttonPanel;     // To hold a button
15    private TextField fahrenheit;  // Fahrenheit temperature
16    private TextField celsius;     // Celsius temperature
17
18    /**
19       init method
20    */
21
22    public void init()
23    {
24       // Build the panels.
25       buildFpanel();
26       buildCpanel();
27       buildButtonPanel();
28
29       // Create a layout manager.
30       setLayout(new GridLayout(3, 1));
31
32       // Add the panels to the applet.
33       add(fPanel);
34       add(cPanel);
35       add(buttonPanel);
36    }
```

```
37
38    /**
39       The buildFpanel method creates a panel with a text
40       field in which the user can enter a Fahrenheit
41       temperature.
42    */
43
44    private void buildFpanel()
45    {
46       // Create the panel.
47       fPanel = new Panel();
48
49       // Create a label to display a message.
50       Label message1 = new Label("Fahrenheit Temperature:");
51
52       // Create a text field for the Fahrenheit temp.
53       fahrenheit = new TextField(10);
54
55       // Create a layout manager for the panel.
56       fPanel.setLayout(new FlowLayout(FlowLayout.RIGHT));
57
58       // Add the label and text field to the panel.
59       fPanel.add(message1);
60       fPanel.add(fahrenheit);
61    }
62
63    /**
64       The buildCpanel method creates a panel that
65       displays the Celsius temperature in a
66       read-only text field.
67    */
68
69    private void buildCpanel()
70    {
71       // Create the panel.
72       cPanel = new Panel();
73
74       // Create a label to display a message.
75       Label message2 = new Label("Celsius Temperature:");
76
77       // Create a text field for the Celsius temp.
78       celsius = new TextField(10);
79
80       // Make the text field read-only.
81       celsius.setEditable(false);
82
```

```
83          // Create a layout manager for the panel.
84          cPanel.setLayout(new FlowLayout(FlowLayout.RIGHT));
85
86          // Add the label and text field to the panel.
87          cPanel.add(message2);
88          cPanel.add(celsius);
89       }
90
91       /**
92          The buildButtonPanel method creates a panel with
93          a button that converts the Fahrenheit temperature
94          to Celsius.
95       */
96
97
98       private void buildButtonPanel()
99       {
100         // Create the panel.
101         buttonPanel = new Panel();
102
103         // Create a button with the text "Convert".
104         Button convButton = new Button("Convert");
105
106         // Add an action listener to the button.
107         convButton.addActionListener(new ButtonListener());
108
109         // Add the button to the panel.
110         buttonPanel.add(convButton);
111      }
112
113      /**
114         Private inner class that handles the action event
115         that is generated when the user clicks the convert
116         button.
117      */
118
119      private class ButtonListener implements ActionListener
120      {
121         public void actionPerformed(ActionEvent e)
122         {
123            double ftemp, ctemp;  // To hold the temperatures
124
125            // Get the Fahrenheit temperature and convert it
126            // to a double.
127            ftemp = Double.parseDouble(fahrenheit.getText());
128
```

```
129          // Calculate the Celsius temperature.
130          ctemp = (5.0 / 9.0) * (ftemp - 32);
131
132          // Display the Celsius temperature.
133          celsius.setText(String.format("%.1f",  ctemp));
134       }
135    }
136 }
```

The only modifications that were made were as follows:

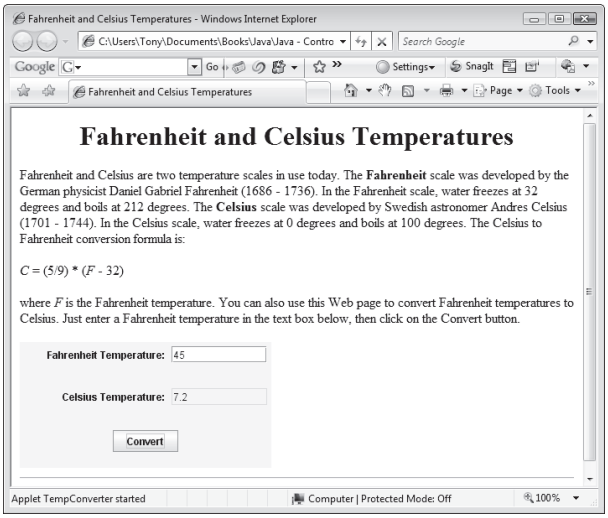
- The `JApplet`, `JPanel`, `JLabel`, `JTextField`, and `JButton` classes were replaced with the `Applet`, `Panel`, `Label`, `TextField`, and `Button` classes.
- The `import javax.swing.*;` statement was removed.

To run the applet in a browser, the `APPLET` tag in the *TempConverter.html* file must be modified to read as follows:

```
<applet code="AWTTempConverter.class" width=300 height=150>
</applet>
```

Once this change is made, the *TempConverter.html* file produces the Web page shown in Figure 25-16.

Figure 25-16 Web page running the `AWTTempConverter` applet (Microsoft Corporation)



Checkpoint

MyProgrammingLab™ *www.myprogramminglab.com*

25.16 To create an applet using AWT, what class do you inherit your applet class from?

25.17 In Swing, if an object's class extends `JFrame` or `JApplet`, you add components to its content pane. How do you add components to an object if its class extends `Frame` or `Applet`?

25.5 Drawing Shapes

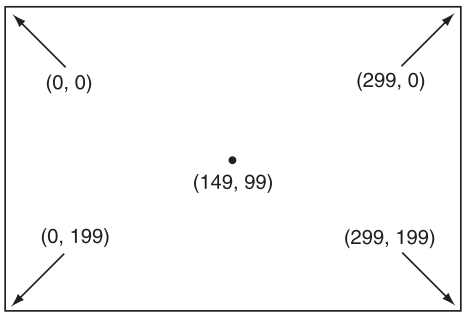
CONCEPT: **Components have an associated `Graphics` object that may be used to draw lines and shapes.**

In addition to displaying standard components such as buttons and labels, Java allows you to draw lines and graphical shapes such as rectangles, ovals, and arcs. These lines and shapes are drawn directly on components. This allows a frame or a panel to become a canvas for your drawings. Before we examine how to draw graphics on a component, however, we must discuss the *XY* coordinate system. You use the *XY* coordinate system to specify the location of your graphics.

The *XY* Coordinate System

The location of each pixel in a component is identified with an *X* coordinate and a *Y* coordinate. The coordinates are usually written in the form (*X*, *Y*). The *X* coordinate identifies a pixel's horizontal location, and the *Y* coordinate identifies its vertical location. The coordinates of the pixel in the upper-left corner of a component are usually (0, 0). The *X* coordinates increase from left to right, and the *Y* coordinates increase from top to bottom. For example, Figure 25-17 illustrates a component such as a frame or a panel that is 300 pixels wide by 200 pixels high. The *X* and *Y* coordinates of the pixels in each corner, as well as the pixel in the center of the component are shown. The pixel in the center of the component has an *X* coordinate of 149 and a *Y* component of 99.

Figure 25-17 *X* and *Y* coordinates on a 300 pixel wide by 200 pixel high component



When you draw a line or shape on a component, you must indicate its position using *X* and *Y* coordinates.

Graphics Objects

Each component has an internal object that inherits from the `Graphics` class, which is part of the `java.awt` package. This object has numerous methods for drawing graphical shapes on the surface of the component. Table 25-2 lists some of these methods.

Table 25-2 Some of the `Graphics` class methods

Method	Description
`void setColor(Color c)`	Sets the drawing color for this object to that specified by the argument.
`Color getColor()`	Returns the current drawing color for this object.
`void drawLine(int x1, int y1, int x2, int y2)`	Draws a line on the component starting at the coordinate (*x1*, *y1*) and ending at the coordinate (*x2*, *y2*). The line will be drawn in the current drawing color.
`void drawRect(int x, int y, int width, int height)`	Draws the outline of a rectangle on the component. The upper-left corner of the rectangle will be at the coordinate (*x*, *y*). The *width* parameter specifies the rectangle's width in pixels, and *height* specifies the rectangle's height in pixels. The rectangle will be drawn in the current drawing color.
`void fillRect(int x, int y, int width, int height)`	Draws a filled rectangle. The parameters are the same as those used by the `drawRect` method. The rectangle will be filled with the current drawing color.
`void drawOval(int x, int y, int width, int height)`	Draws the outline of an oval on the component. The shape and size of the oval is determined by an invisible rectangle that encloses it. The upper-left corner of the rectangle will be at the coordinate (*x*, *y*). The *width* parameter specifies the rectangle's width in pixels, and *height* specifies the rectangle's height in pixels. The oval will be drawn in the current drawing color.
`void fillOval(int x, int y, int width, int height)`	Draws a filled oval. The parameters are the same as those used by the `drawOval` method. The oval will be filled in the current drawing color.
`void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)`	This method draws an arc, which is considered to be part of an oval. The shape and size of the oval are determined by an invisible rectangle that encloses it. The upper-left corner of the rectangle will be at the coordinate (*x*, *y*). The *width* parameter specifies the rectangle's width in pixels, and *height* specifies the rectangle's height in pixels. The arc begins at the angle *startAngle*, and ends at the angle *arcAngle*. The arc will be drawn in the current drawing color.
`void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)`	This method draws a filled arc. The parameters are the same as those used by the `drawArc` method. The arc will be filled with the current drawing color.

(table continues next page)

Table 25-2 Some of the `Graphics` class methods (continued)

Method	Description
`void drawPolygon(int[] xPoints, int[] yPoints, int numPoints)`	This method draws the outline of a closed polygon on the component. The *`xPoints`* array contains the *X*-coordinates for each vertex, and the *`yPoints`* array contains the *Y* coordinates for each vertex. The argument passed into *`numPoints`* is the number of vertices in the polygon.
`void fillPolygon(int[] xPoints, int[] yPoints, int numPoints)`	This method draws a filled polygon. The parameters are the same as those used by the `drawPolygon` method. The polygon will be filled with the current drawing color.
`void drawstring(String str, int x, int y)`	Draws the string passed into *`str`* using the current font. The bottom left of the string is drawn at the coordinates passed into *`x`* and *`y`*.
`void setFont(Font f)`	Sets the current font, which is used by the `drawString` method.

In order to call any of these methods, you must get a reference to a component's `Graphics` object. One way to do this is to override the `paint` method. You can override the `paint` method in any class that extends as follows:

- `JApplet`
- `JFrame`
- Any AWT class, including `Applet` and `Frame`

The `paint` method is responsible for displaying, or "painting," a component on the screen. This method is automatically called when the component is first displayed and is called again any time the component needs to be redisplayed. For example, when the component is completely or partially obscured by another window, and the obscuring window is moved, then the component's `paint` method is called to redisplay it. The header for the `paint` method is:

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

Notice that the method's argument is a `Graphics` object. When this method is called for a particular component, the `Graphics` object that belongs to that component is automatically passed as an argument. By overriding the `paint` method, you can use the `Graphics` object argument to draw your own graphics on the component. For example, look at the applet class in Code Listing 25-11.

This class inherits from `JApplet`, and it overrides the `paint` method. The `Graphics` object that is passed into the `paint` method's `g` parameter is the object that is responsible for drawing the entire applet window. Notice that in line 29 the method first calls the superclass version of the `paint` method, passing the object `g` as an argument. When overriding the `paint` method, you should always call the superclass's `paint` method before doing anything else. This ensures that the component will be displayed properly on the screen.

Code Listing 25-11 (LineDemo.java)

```
1 import javax.swing.*;
2 import java.awt.*;
3
4 /**
5    This class is an applet that demonstrates how lines
6    can be drawn.
7 */
8
9 public class LineDemo extends JApplet
10 {
11    /**
12       init method
13    */
14
15    public void init()
16    {
17       // Set the background color to white.
18       getContentPane().setBackground(Color.white);
19    }
20
21    /**
22       paint method
23       @param g The applet's Graphics object.
24    */
25
26    public void paint(Graphics g)
27    {
28       // Call the superclass paint method.
29       super.paint(g);
30
31       // Draw a red line from (20, 20) to (280, 280).
32       g.setColor(Color.red);
33       g.drawLine(20, 20, 280, 280);
34
35       // Draw a blue line from (280, 20) to (20, 280).
36       g.setColor(Color.blue);
37       g.drawLine(280, 20, 20, 280);
38    }
39 }
```

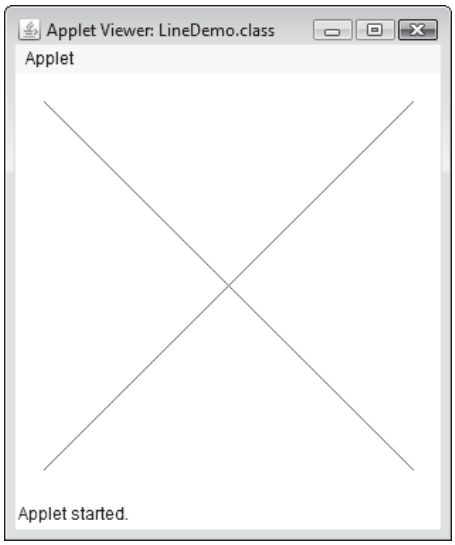
In line 32 the method sets the drawing color to red. In line 33 a line is drawn from the coordinates (20, 20) to (280, 280). This is a diagonal line drawn from the top-left area of the applet window to the bottom-right area. Next, in line 36, the drawing color is set to blue. In line 37 a line is drawn from (280, 20) to (20, 280). This is also a diagonal line. It is drawn from the top-right area of the applet window to the bottom-left area.

We can use the *LineDemo.html* file, which is in the same folder as the applet class, to execute the applet. The following line in the file runs the applet in an area that is 300 pixels wide by 300 pixels high:

```
<applet code="LineDemo.class" width=300 height=300>
</applet>
```

Figure 25-18 shows the applet running in the applet viewer.

Figure 25-18 `LineDemo` applet (Oracle Corporate Counsel)



Notice that the `paint` method is not explicitly called by the applet. It is automatically called when the applet first executes. As previously mentioned, it is also called any time the applet window needs to be redisplayed.

Code Listing 25-12 shows the `RectangleDemo` class, an applet that draws two rectangles: one as a black outline and one filled with red. Each rectangle is 120 pixels wide and 120 pixels high. The file *RectangleDemo.html*, which is in the same folder as the applet class, executes the applet with the following tag:

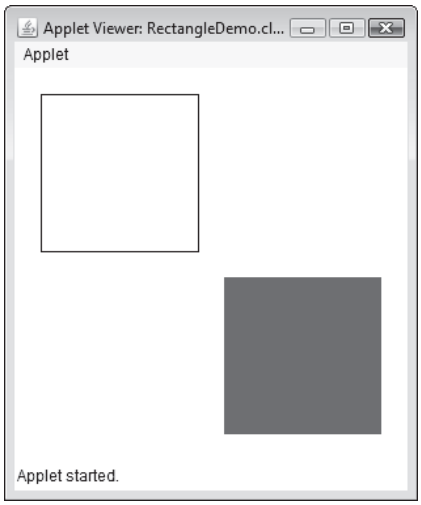
```
<applet code="RectangleDemo.class" width=300 height=300>
</applet>
```

Figure 25-19 shows the applet running in the applet viewer.

Code Listing 25-12 (RectangleDemo.java)

```
1  import javax.swing.*;
2  import java.awt.*;
3
4  /**
5     This class is an applet that demonstrates how
6     rectangles can be drawn.
7  */
8
9  public class RectangleDemo extends JApplet
10 {
11    /**
12       init method
13    */
14
15    public void init()
16    {
17       // Set the background color to white.
18       getContentPane().setBackground(Color.white);
19    }
20
21    /**
22       paint method
23       @param g The applet's Graphics object.
24    */
25
26    public void paint(Graphics g)
27    {
28       // Call the superclass paint method.
29       super.paint(g);
30
31       // Draw a black unfilled rectangle.
32       g.setColor(Color.black);
33       g.drawRect(20, 20, 120, 120);
34
35       // Draw a red filled rectangle.
36       g.setColor(Color.red);
37       g.fillRect(160, 160, 120, 120);
38    }
39 }
```

Figure 25-19 `RectangleDemo` applet (Oracle Corporate Counsel)



Code Listing 25-13 shows the `OvalDemo` class, an applet that draws two ovals. An oval is enclosed in an invisible rectangle that establishes the boundaries of the oval. The width and height of the enclosing rectangle defines the shape and size of the oval. This is illustrated in Figure 25-20.

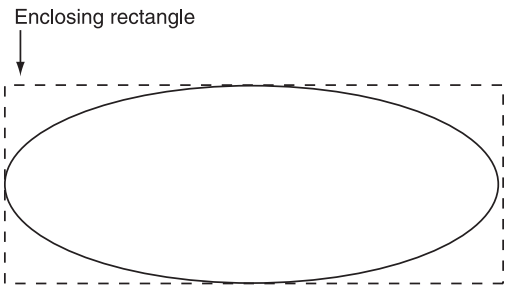
When you call the `drawOval` or `fillOval` method, you pass the *X* and *Y* coordinates of the enclosing rectangle's upper-left corner, and the width and height of the enclosing rectangle as arguments.

Code Listing 25-13 (`OvalDemo.java`)

```
1  import javax.swing.*;
2  import java.awt.*;
3
4  /**
5     This class is an applet that demonstrates how
6     ovals can be drawn.
7  */
8
9  public class OvalDemo extends JApplet
10 {
11    /**
12       init method
13    */
14
15    public void init()
16    {
```

```
17          // Set the background color to white.
18          getContentPane().setBackground(Color.white);
19       }
20
21       /**
22          paint method
23          @param g The applet's Graphics object.
24       */
25
26       public void paint(Graphics g)
27       {
28          // Call the superclass paint method.
29          super.paint(g);
30
31          // Draw a black unfilled oval.
32          g.setColor(Color.black);
33          g.drawOval(20, 20, 120, 75);
34
35          // Draw a green filled oval.
36          g.setColor(Color.green);
37          g.fillOval(80, 160, 180, 75);
38       }
39    }
```

Figure 25-20 An oval and its enclosing rectangle

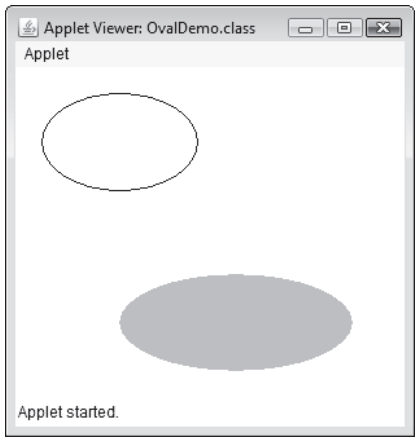


The file *OvalDemo.html*, which is in the same folder as the applet class, executes the applet with the following tag:

```
<applet code="OvalDemo.class" width=300 height=255>
</applet>
```

Figure 25-21 shows the applet running in the applet viewer.

Figure 25-21 `OvalDemo` applet (Oracle Corporate Counsel)



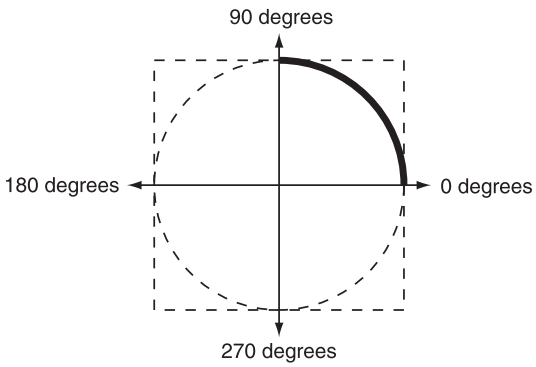
TIP: To draw a circle, simply draw an oval with an enclosing rectangle that is square. In other words, the enclosing rectangle's width and height should be the same.

The `drawArc` method draws an arc, which is part of an oval. You pass the same arguments to `drawArc` as you do to `drawOval`, plus two additional arguments: the arc's starting angle and ending angle. The angles are measured in degrees, with 0 degrees being at the 3 o'clock position. For example, look at the following statement:

```
g.drawArc(20, 20, 100, 100, 0, 90);
```

This statement creates an enclosing rectangle with its upper-left corner at (20, 20) and with a width and height of 100 pixels each. The oval constructed from this enclosing rectangle is a circle. The arc that is drawn is the part of the oval that starts at 0 degrees and ends at 90 degrees. Figure 25-22 illustrates this arc. The dashed lines show the enclosing rectangle and the oval. The thick black line shows the arc that will be drawn.

Figure 25-22 An arc



Code Listing 25-14 shows the `ArcDemo` class, which is an applet that draws four arcs: two unfilled and two filled. The filled arcs are drawn with the `fillArc` method.

The file *ArcDemo.html*, which is in the same folder as the applet class, executes the applet with the following tag:

```
<applet code="ArcDemo.class" width=300 height=220>
</applet>
```

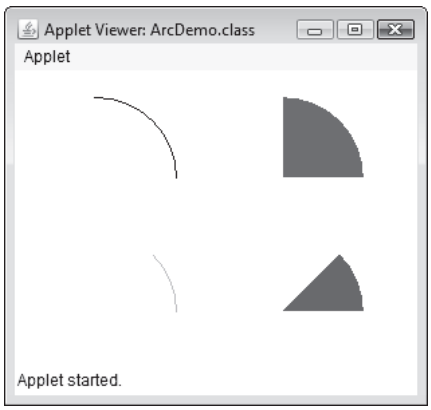
Figure 25-23 shows the applet running in the applet viewer.

Code Listing 25-14 (ArcDemo.java)

```
1  import javax.swing.*;
2  import java.awt.*;
3
4  /**
5     This class is an applet that demonstrates how
6     arcs can be drawn.
7  */
8
9  public class ArcDemo extends JApplet
10 {
11    /**
12       init method
13    */
14
15    public void init()
16    {
17       // Set the background color to white.
18       getContentPane().setBackground(Color.white);
19    }
20
21    /**
22       paint method
23       @param g The applet's Graphics object.
24    */
25
26    public void paint(Graphics g)
27    {
28       // Call the superclass paint method.
29       super.paint(g);
30
31       // Draw a black unfilled arc from 0 degrees
32       // to 90 degrees.
33       g.setColor(Color.black);
34       g.drawArc(0, 20, 120, 120, 0, 90);
35
36       // Draw a red filled arc from 0 degrees
37       // to 90 degrees.
```

```
38        g.setColor(Color.red);
39        g.fillArc(140, 20, 120, 120, 0, 90);
40
41        // Draw a green unfilled arc from 0 degrees
42        // to 45 degrees.
43        g.setColor(Color.green);
44        g.drawArc(0, 120, 120, 120, 0, 45);
45
46        // Draw a blue filled arc from 0 degrees
47        // to 45 degrees.
48        g.setColor(Color.blue);
49        g.fillArc(140, 120, 120, 120, 0, 45);
50     }
51  }
```

Figure 25-23 ArcDemo applet (Oracle Corporate Counsel)



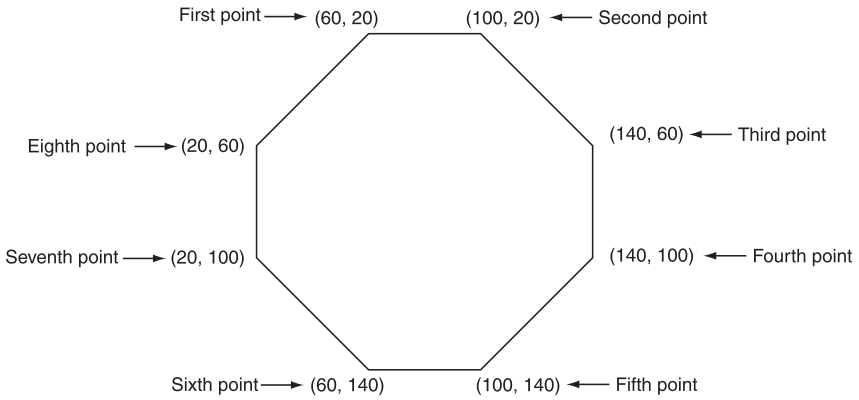
The `drawPolygon` method draws an outline of a closed polygon and the `fillPolygon` method draws a closed polygon filled with the current drawing color. A polygon is constructed of multiple line segments that are connected. The point where two line segments are connected is called a *vertex*. These methods accept two `int` arrays as arguments. The first array contains the *X* coordinates of each vertex, and the second array contains the *Y* coordinates of each vertex. The third argument is an `int` that specifies the number of vertices, or connecting points.

For example, suppose we use the following arrays as arguments for the *X* and *Y* coordinates of a polygon:

```
int[] xCoords = {60, 100, 140, 140, 100, 60, 20, 20 };
int[] yCoords = {20, 20, 60, 100, 140, 140, 100, 60 };
```

The first point specified by these arrays is (60, 20), the second point is (100, 20), and so forth. There are a total of eight points specified by these arrays, and if we connect each of these points we get the octagon shown in Figure 25-24.

Figure 25-24 Points of each vertex in an octagon



If the last point specified in the arrays is different from the first point, as in this example, then the two points are automatically connected to close the polygon. The `PolygonDemo` class in Code Listing 25-15 draws a filled polygon using these arrays as arguments.

Code Listing 25-15 (`PolygonDemo.java`)

```
1  import javax.swing.*;
2  import java.awt.*;
3
4  /**
5     This class is an applet that demonstrates how a
6     polygon can be drawn.
7  */
8
9  public class PolygonDemo extends JApplet
10 {
11    /**
12       init method
13    */
14
15    public void init()
16    {
17       // Set the background color to white.
18       getContentPane().setBackground(Color.white);
19    }
20
21    /**
22       paint method
23       @param g The applet's Graphics object.
24    */
25
26    public void paint(Graphics g)
```

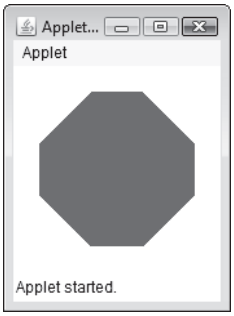
```
27    {
28       int[] xCoords = {60, 100, 140, 140,
29                        100, 60, 20, 20 };
30       int[] yCoords = {20, 20, 60, 100,
31                        140, 140, 100, 60 };
32
33       // Call the superclass paint method.
34       super.paint(g);
35
36       // Set the drawing color.
37       g.setColor(Color.red);
38
39       // Draw the polygon.
40       g.fillPolygon(xCoords, yCoords, 8);
41    }
42 }
```

The file *PolygonDemo.html*, which is in the same folder as the applet class, executes the applet with the following tag:

```
<applet code="PolygonDemo.class" width=160 height=160>
</applet>
```

Figure 25-25 shows the applet running in the applet viewer.

Figure 25-25 PolygonDemo applet (Oracle Corporate Counsel)



The `drawString` method draws a string as a graphic. The string is specified by its first argument, a `String` object. The *X* and *Y* coordinates of the lower-left point of the string are specified by the second and third arguments. For example, assuming that `g` references a `Graphics` object, the following statement draws the string `"Hello World"`, starting at the coordinates 100, 50:

```
g.drawstring("Hello World", 100, 50);
```

You can set the font for the string with the `setFont` method. This method accepts a `Font` object as its argument. Here is an example:

```
g.setFont(new Font("Serif", Font.ITALIC, 20));
```

The `Font` class was covered in Chapter 18. Recall that the `Font` constructor's arguments are the name of a font, the font's style, and the font's size in points. You can combine font styles with the + operator, as follows:

```
g.setFont(new Font("Serif", Font.BOLD + Font.ITALIC, 24));
```

The `GraphicStringDemo` class in Code Listing 25-16 demonstrates the `drawString` method. It draws the same octagon that the `PolygonDemo` class drew, and then draws the string `"STOP"` over it to create a stop sign. The string is drawn in a bold 35-point san serif font.

Code Listing 25-16 (GraphicStringDemo.java)

```
1  import javax.swing.*;
2  import java.awt.*;
3
4  /**
5     This class is an applet that demonstrates how a
6     string can be drawn.
7  */
8
9  public class GraphicStringDemo extends JApplet
10 {
11    /**
12       init method
13    */
14
15    public void init()
16    {
17       // Set the background color to white.
18       getContentPane().setBackground(Color.white);
19    }
20
21    /**
22       paint method
23       @param g The applet's Graphics object.
24    */
25
26    public void paint(Graphics g)
27    {
28       int[] xCoords = {60, 100, 140, 140,
29                        100, 60, 20, 20 };
30       int[] yCoords = {20, 20, 60, 100,
31                        140, 140, 100, 60 };
32
```

```
33        // Call the superclass paint method.
34        super.paint(g);
35
36        // Set the drawing color.
37        g.setColor(Color.red);
38
39        // Draw the polygon.
40        g.fillPolygon(xCoords, yCoords, 8);
41
42        // Set the drawing color to white.
43        g.setColor(Color.white);
44
45        // Set the font and draw "STOP".
46        g.setFont(new Font("SansSerif", Font.BOLD, 35));
47        g.drawString("STOP", 35, 95);
48     }
49  }
```

The file *GraphicStringDemo.html*, which is in the same folder as the applet class, executes the applet with the following tag:

```
<applet code="GraphicStringDemo.class" width=160 height=160>
</applet>
```

Figure 25-26 shows the applet running in the applet viewer.

Figure 25-26 `GraphicStringDemo` applet (Oracle Corporate Counsel)

The `repaint` Method

As previously mentioned, you do not call a component's `paint` method. It is automatically called when the component must be redisplayed. Sometimes, however, you might want to force the application or applet to call the `paint` method. You do this by calling the `repaint` method, which has the following header:

```
public void repaint()
```

The `repaint` method clears the surface of the component and then calls the `paint` method. You will see an applet that uses this method in a moment.

Drawing on Panels

Each of the preceding examples uses the entire `JApplet` window as a canvas for drawing. Sometimes, however, you might want to confine your drawing space to a smaller region within the window, such as a panel. To draw on a panel, you simply get a reference to the panel's `Graphics` object and then use that object's methods to draw. The resulting graphics are drawn only on the panel.

Getting a reference to a `JPanel` component's `Graphics` object is similar to the technique you saw in the previous examples. Instead of overriding the `JPanel` object's `paint` method, however, you should override its `paintComponent` method. This is true not only for `JPanel` objects, but also for all Swing components except `JApplet` and `JFrame`. The `paintComponent` method serves for `JPanel` and most other Swing objects the same purpose as the `paint` method: It is automatically called when the component needs to be redisplayed. When it is called, the component's `Graphics` object is passed as an argument. Here is the method's header:

```
public void paintComponent(Graphics g)
```

When you override this method, first you should call the superclass's `paintComponent` method to ensure that the component is properly displayed. Here is an example call to the superclass's version of the method:

```
super.paintComponent(g);
```

After this you can call any of the `Graphics` object's methods to draw on the component. As an example, we look at the `GraphicsWindow` class in Code Listing 25-17. When this applet is run (via the *GraphicsWindow.html* file, which is in the same folder as the applet class), the window shown in Figure 25-27 is displayed. A set of check boxes is displayed in a `JPanel` component on the right side of the window. The white area that occupies the majority of the window is a `DrawingPanel` object. The `DrawingPanel` class inherits from `JPanel`, and its code is shown in Code Listing 25-18. When one of the check boxes is selected, a shape appears in the `DrawingPanel` object. Figure 25-28 shows how the applet window appears when all of the check boxes are selected.

Figure 25-27 `GraphicsWindow` applet (Oracle Corporate Counsel)

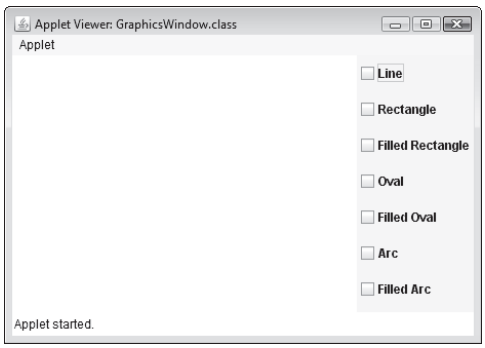
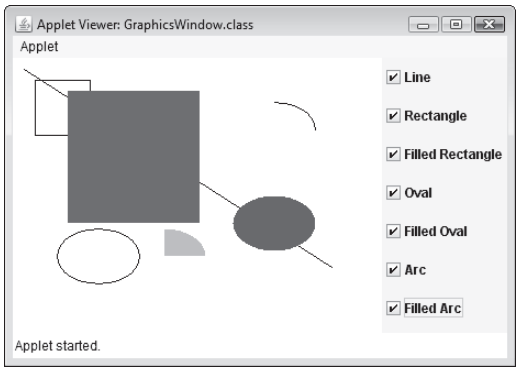


Figure 25-28 GraphicsWindow applet with all graphics selected (Oracle Corporate Counsel)



Code Listing 25-17 (GraphicsWindow.java)

```
1  import javax.swing.*;
2  import java.awt.*;
3  import java.awt.event.*;
4
5  /**
6     This class displays a drawing panel and a set of
7     check boxes that allow the user to select shapes.
8     The selected shapes are drawn on the drawing panel.
9  */
10
11 public class GraphicsWindow extends JApplet
12 {
13    // Declare an array of check box components
14    private JCheckBox[] checkBoxes;
15
16    // The following titles array contains the
17    // titles of the check boxes.
18    private String[] titles = { "Line", "Rectangle",
19                                "Filled Rectangle",
20                                "Oval", "Filled Oval",
21                                "Arc", "Filled Arc" };
22
23    // The following will reference a panel to contain
24    // the check boxes.
25    private JPanel checkBoxPanel;
26
27    // The following will reference an instance of the
28    // DrawingPanel class. This will be a panel to draw on.
29    private DrawingPanel drawingPanel;
```

```
30
31    /**
32       init method
33    */
34
35    public void init()
36    {
37       // Build the check box panel.
38       buildCheckBoxPanel();
39
40       // Create the drawing panel.
41       drawingPanel = new DrawingPanel(checkBoxes);
42
43       // Add the check box panel to the east region
44       // and the drawing panel to the center region.
45       add(checkBoxPanel, BorderLayout.EAST);
46       add(drawingPanel, BorderLayout.CENTER);
47    }
48
49    /**
50       The buildCheckBoxPanel method creates the array of
51       check box components and adds them to a panel.
52    */
53
54    private void buildCheckBoxPanel()
55    {
56       // Create the panel.
57       checkBoxPanel = new JPanel();
58       checkBoxPanel.setLayout(new GridLayout(7, 1));
59
60       // Create the check box array.
61       checkBoxes = new JCheckBox[7];
62
63       // Create the check boxes and add them to the panel.
64       for (int i = 0; i < checkBoxes.length; i++)
65       {
66          checkBoxes[i] = new JCheckBox(titles[i]);
67          checkBoxes[i].addItemListener(
68                                    new CheckBoxListener());
69          checkBoxPanel.add(checkBoxes[i]);
70       }
71    }
72
73    /**
74       A private inner class to respond to changes in the
75       state of the check boxes.
76    */
77
```

```
78    private class CheckBoxListener implements ItemListener
79    {
80       public void itemStateChanged(ItemEvent e)
81       {
82          drawingPanel.repaint();
83       }
84    }
85 }
```

Code Listing 25-18 (DrawingPanel.java)

```
1  import javax.swing.*;
2  import java.awt.*;
3
4  /**
5     This class creates a panel that example shapes are
6     drawn on.
7  */
8
9  public class DrawingPanel extends JPanel
10 {
11    // Declare a check box array.
12    private JCheckBox[] checkBoxArray;
13
14    /**
15       Constructor
16    */
17
18    public DrawingPanel(JCheckBox[] cbArray)
19    {
20       // Reference the check box array.
21       checkBoxArray = cbArray;
22
23       // Set the background color to white.
24       setBackground(Color.white);
25
26       // Set the preferred size of the panel.
27       setPreferredSize(new Dimension(300, 200));
28    }
29
30    /**
31       paintComponent method
32       @param g The panel's Graphics object.
33    */
34
35    public void paintComponent(Graphics g)
```

```
36    {
37       // Call the superclass paintComponent method.
38       super.paintComponent(g);
39
40       // Draw the selected shapes.
41       if (checkBoxArray[0].isSelected())
42       {
43          g.setColor(Color.black);
44          g.drawLine(10, 10, 290, 190);
45       }
46       if (checkBoxArray[1].isSelected())
47       {
48          g.setColor(Color.black);
49          g.drawRect(20, 20, 50, 50);
50       }
51       if (checkBoxArray[2].isSelected())
52       {
53          g.setColor(Color.red);
54          g.fillRect(50, 30, 120, 120);
55       }
56       if (checkBoxArray[3].isSelected())
57       {
58          g.setColor(Color.black);
59          g.drawOval(40, 155, 75, 50);
60       }
61       if (checkBoxArray[4].isSelected())
62       {
63          g.setColor(Color.blue);
64          g.fillOval(200, 125, 75, 50);
65       }
66       if (checkBoxArray[5].isSelected())
67       {
68          g.setColor(Color.black);
69          g.drawArc(200, 40, 75, 50, 0, 90);
70       }
71       if (checkBoxArray[6].isSelected())
72       {
73          g.setColor(Color.green);
74          g.fillArc(100, 155, 75, 50, 0, 90);
75       }
76    }
77 }
```

Let's take a closer look at the applet's code. First, notice in lines 14 through 21 of the `GraphicsWindow` class (in Code Listing 25-17) that two of the class's fields are array reference variables. The `checkBoxes` variable references an array of `JCheckBox` components, and the `titles` variable references an array of strings. The strings in the `titles` array are the titles that the check boxes will display.

The first statement in the `init` method, line 38, is a call to the `buildCheckBoxPanel` method, which creates a panel for the check boxes, creates the array of check boxes, adds an item listener to each element of the array, and adds each element to the panel.

After the `buildCheckBoxPanel` method executes, the `init` method creates a `DrawingPanel` object with the statement in line 41. Notice that the `checkBoxes` variable is passed to the `DrawingPanel` constructor. The `drawingPanel` object needs a reference to the array so its `paintComponent` method can determine which check boxes are selected and draw the corresponding shape.

The only times that the `paintComponent` method is automatically called is when the component is initially displayed and when the component needs to be redisplayed. In order to display a shape immediately when the user selects a check box, we need the check box item listener to force the `paintComponent` method to be called. This is accomplished by the statement in line 82, in the `CheckBoxListener` class's `itemStateChanged` method. This statement calls the `drawingPanel` object's `repaint` method, which causes the `drawingPanel` object's surface to be cleared, and then causes the object's `paintComponent` method to execute. Because it is in the item listener, it is executed each time the user clicks on a check box.

Checkpoint

MyProgrammingLab™ *www.myprogramminglab.com*

25.18 In an AWT component, or a class that extends `JApplet` or `JFrame`, if you want to get a reference to the `Graphics` object, do you override the `paint` or `paintComponent` method?

25.19 In a `JPanel` object, do you override the `paint` or `paintComponent` method to get a reference to the `Graphics` object?

25.20 When are the `paint` and `paintComponent` method called?

25.21 In the `paint` or `paintComponent` method, what should be done before anything else?

25.22 How do you force the `paint` or `paintComponent` method to be called?

25.23 When using a `Graphics` object to draw an oval, what invisible shape is the oval enclosed in?

25.24 What values are contained in the two arrays that are passed to a `Graphics` object's `drawPolygon` method?

25.25 What `Graphics` class methods do you use to perform the following tasks?

a) Draw a line.
b) Draw a filled rectangle.
c) Draw a filled oval.
d) Draw a filled arc.
e) Set the drawing color.
f) Draw a rectangle.
g) Draw an oval.
h) Draw an arc.
i) Draw a string.
j) Set the font.

25.6 Handling Mouse Events

CONCEPT: **Java allows you to create listener classes that handle events generated by the mouse.**

Handling Mouse Events

The mouse generates two types of events: mouse events and mouse motion events. To handle mouse events you create a *mouse listener* class and/or a *mouse motion listener* class. A mouse listener class can respond to any of the follow events:

- The mouse button is pressed.
- The mouse button is released.
- The mouse button is clicked (pressed, then released without moving the mouse).
- The mouse cursor enters a component's screen space.
- The mouse cursor exits a component's screen space.

A mouse listener class must implement the `MouseListener` interface, which is in the `java.awt.event` package. The class must also have the methods listed in Table 25-3.

Table 25-3 Methods required by the `MouseListener` interface

Method	Description
`public void mousePressed(MouseEvent e)`	If the mouse cursor is over the component and the mouse button is pressed, this method is called.
`public void mouseClicked(MouseEvent e)`	A mouse click is defined as pressing the mouse button and releasing it without moving the mouse. If the mouse cursor is over the component and the mouse is clicked on, this method is called.
`public void mouseReleased(MouseEvent e)`	This method is called when the mouse button is released after it has been pressed. The `mousePressed` method is always called before this method.
`public void mouseEntered(MouseEvent e)`	This method is called when the mouse cursor enters the screen area belonging to the component.
`public void mouseExited(MouseEvent e)`	This method is called when the mouse cursor leaves the screen area belonging to the component.

Notice that each of the methods listed in Table 25-3 accepts a `MouseEvent` object as its argument. The `MouseEvent` object contains data about the mouse event. We will use two

of the `MouseEvent` object's methods: `getX` and `getY`. These methods return the *X* and *Y* coordinates of the mouse cursor when the event occurs.

Once you create a mouse listener class, you can register it with a component using the `addMouseListener` method, which is inherited from the `Component` class. The appropriate methods in the mouse listener class are automatically called when their corresponding mouse events occur.

A mouse motion listener class can respond to the following events:

- The mouse is dragged (the button is pressed and the mouse is moved while the button is held down).
- The mouse is moved.

A mouse motion listener class must implement the `MouseMotionListener` interface, which is in the `java.awt.event` package. The class must also have the methods listed in Table 25-4. Notice that each of these methods also accepts a `MouseEvent` object as an argument.

Table 25-4 Methods required by the `MouseMotionListener` interface

Method	Description
`public void mouseDragged(MouseEvent e)`	The mouse is dragged when its button is pressed and the mouse is moved while the button is held down. This method is called when a dragging operation begins over the component. The `mousePressed` method is always called just before this method.
`public void mouseMoved(MouseEvent e)`	This method is called when the mouse cursor is over the component and it is moved.

Once you create a mouse motion listener class, you can register it with a component using the `addMouseMotionListener` method, which is inherited from the `Component` class. The appropriate methods in the mouse motion listener class are automatically called when their corresponding mouse events occur.

The `MouseEvents` class, shown in Code Listing 25-19, is an applet that demonstrates both a mouse listener and a mouse motion listener. The file *MouseEvents.html*, which is in the same folder as the applet class, can be used to start the applet. Figure 25-29 shows the applet running. The window displays a group of read-only text fields that represent the different mouse and mouse motion events. When an event occurs, the corresponding text field turns yellow. The last two text fields constantly display the mouse cursor's *X* and *Y* coordinates. Run this applet and experiment by clicking the mouse inside the window, dragging the mouse, moving the mouse cursor in and out of the window, and moving the mouse cursor over the text fields.

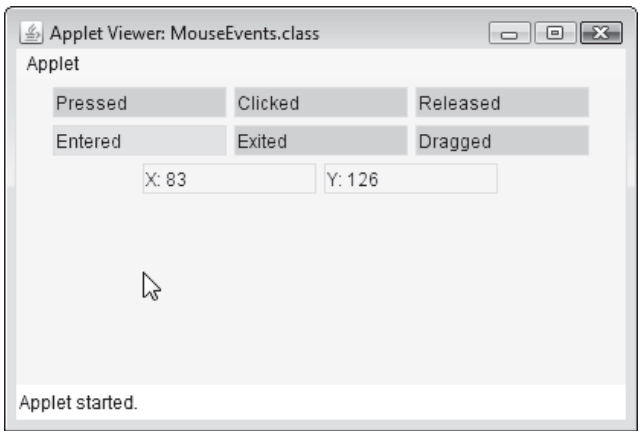
Code Listing 25-19 (MouseEvents.java)

```
1  import javax.swing.*;
2  import java.awt.event.*;
3  import java.awt.*;
4
5  /**
6     This applet shows the mouse events as they occur.
7  */
8
9  public class MouseEvents extends JApplet
10 {
11    private JTextField[] mouseStates;
12    private String[] text = {
13                 "Pressed", "Clicked", "Released",
14                 "Entered", "Exited", "Dragged",
15                 "X:", "Y:" };
16
17    /**
18       init method
19    */
20
21    public void init()
22    {
23       // Create a layout manager.
24       setLayout(new FlowLayout());
25
26       // Create the array of text fields.
27       mouseStates = new JTextField[8];
28       for (int i = 0; i < mouseStates.length; i++)
29       {
30          mouseStates[i] = new JTextField(text[i], 10);
31          mouseStates[i].setEditable(false);
32          add(mouseStates[i]);
33       }
34
35       // Add a mouse listener to this applet.
36       addMouseListener(new MyMouseListener());
37
38       // Add a mouse motion listener to this applet.
39       addMouseMotionListener(new MyMouseMotionListener());
40    }
41
42    /**
43       The clearTextFields method sets all of the text
44       backgrounds to light gray.
45    */
```

```
46
47    public void clearTextFields()
48    {
49       for (int i = 0; i < 6; i++)
50          mouseStates[i].setBackground(Color.lightGray);
51    }
52
53    /**
54       Private inner class that handles mouse events.
55       When an event occurs, the text field for that
56       event is given a yellow background.
57    */
58
59    private class MyMouseListener
60                          implements MouseListener
61    {
62       public void mousePressed(MouseEvent e)
63       {
64          clearTextFields();
65          mouseStates[0].setBackground(Color.yellow);
66       }
67
68       public void mouseClicked(MouseEvent e)
69       {
70          clearTextFields();
71          mouseStates[1].setBackground(Color.yellow);
72       }
73
74       public void mouseReleased(MouseEvent e)
75       {
76          clearTextFields();
77          mouseStates[2].setBackground(Color.yellow);
78       }
79
80       public void mouseEntered(MouseEvent e)
81       {
82          clearTextFields();
83          mouseStates[3].setBackground(Color.yellow);
84       }
85
86       public void mouseExited(MouseEvent e)
87       {
88          clearTextFields();
89          mouseStates[4].setBackground(Color.yellow);
90       }
91    }
92
```

```
93    /**
94       Private inner class to handle mouse motion events.
95    */
96
97    private class MyMouseMotionListener
98                          implements MouseMotionListener
99    {
100      public void mouseDragged(MouseEvent e)
101      {
102         clearTextFields();
103         mouseStates[5].setBackground(Color.yellow);
104      }
105
106      public void mouseMoved(MouseEvent e)
107      {
108         mouseStates[6].setText("X: " + e.getX());
109         mouseStates[7].setText("Y: " + e.getY());
110      }
111   }
112 }
```

Figure 25-29 MouseEvents applet (Oracle Corporate Counsel)



Using Adapter Classes

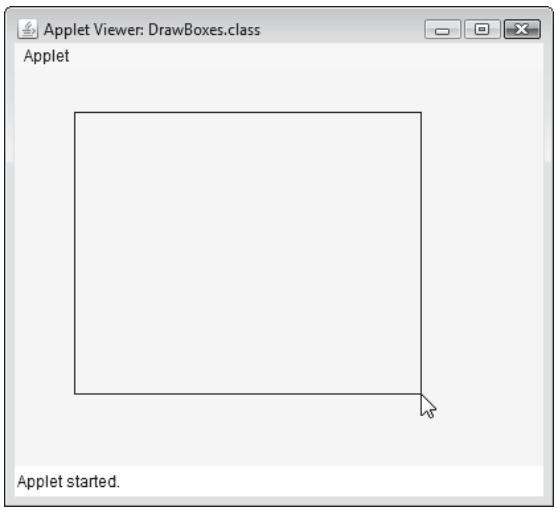
Many times when you handle mouse events, you will not be interested in handling every event that the mouse generates. This is the case with the `DrawBoxes` applet, which handles only mouse pressed and mouse dragged events.

This applet lets you draw rectangles by pressing the mouse button and dragging the mouse inside the applet window. When you initially press the mouse button, the position of the

mouse cursor becomes the upper-left corner of a rectangle. As you drag the mouse, the lower-right corner of the rectangle follows the mouse cursor. When you release the mouse cursor, the rectangle stops following the mouse. Figure 25-30 shows an example of the applet's window. You can run the applet with the *DrawBoxes.html* file, which is in the same folder as the applet class. Code Listing 25-20 shows the code for the `DrawBoxes` class.

NOTE: To draw the rectangle, you must drag the mouse cursor to the right and below the position where you initially pressed the mouse button.

Figure 25-30 DrawBoxes applet (Oracle Corporate Counsel)



Code Listing 25-20 (DrawBoxes.java)

```
1 import javax.swing.*;
2 import java.awt.event.*;
3 import java.awt.*;
4
5 /**
6    This applet demonstrates how mouse events and mouse
7    motion events can be handled. It lets the user draw
8    boxes by dragging the mouse.
9 */
10
11 public class DrawBoxes extends JApplet
12 {
13    private int currentX = 0;      // Mouse cursor's X position
14    private int currentY = 0;      // Mouse cursor's Y position
15    private int width = 0;         // The rectangle's width
```

```
16    private int height = 0;       // The rectangle's height
17
18    /**
19       init method
20    */
21
22    public void init()
23    {
24       // Add a mouse listener and a mouse motion listener.
25       addMouseListener(new MyMouseListener());
26       addMouseMotionListener(new MyMouseMotionListener());
27    }
28
29    /**
30       paint method
31       @param g The applet's Graphics object.
32    */
33
34    public void paint(Graphics g)
35    {
36       // Call the superclass's paint method.
37       super.paint(g);
38
39       // Draw a rectangle.
40       g.drawRect(currentX, currentY, width, height);
41    }
42
43    /**
44       Mouse listener class
45    */
46
47    private class MyMouseListener
48                        implements MouseListener
49    {
50       public void mousePressed(MouseEvent e)
51       {
52          // Get the mouse cursor coordinates.
53          currentX = e.getX();
54          currentY = e.getY();
55       }
56
57       //
58       // The following methods are unused, but still
59       // required by the MouseListener interface.
60       //
61
62       public void mouseClicked(MouseEvent e)
63       {
```

```
64       }
65
66       public void mouseReleased(MouseEvent e)
67       {
68       }
69
70       public void mouseEntered(MouseEvent e)
71       {
72       }
73
74       public void mouseExited(MouseEvent e)
75       {
76       }
77    }
78
79    /**
80       Mouse Motion listener class
81    */
82
83    private class MyMouseMotionListener
84                   implements MouseMotionListener
85    {
86       public void mouseDragged(MouseEvent e)
87       {
88          // Calculate the size of the rectangle.
89          width = e.getX() - currentX;
90          height = e.getY() - currentY;
91
92          // Repaint the window.
93          repaint();
94       }
95
96       /**
97          The mouseMoved method is unused, but still
98          required by the MouseMotionListener interface.
99       */
100
101      public void mouseMoved(MouseEvent e)
102      {
103      }
104   }
105}
```

Notice in the mouse listener and mouse motion listener classes that several of the methods are empty. Even though the applet handles only two mouse events, the `MyMouseListener` and `MyMouseMotionListener` classes must have all of the methods required by the interfaces they implement. If any of these methods are omitted, a compiler error results.

The Java API provides an alternative technique for creating these listener classes, which eliminates the need to define empty methods for the events you are not interested in. Instead of implementing the `MouseListener` or `MouseMotionListener` interfaces, you can extend your classes from the `MouseAdapter` or `MouseMotionAdapter` classes. These classes implement the `MouseListener` and `MouseMotionListener` interfaces and provide empty definitions for all of the required methods. When you extend a class from one of these adapter classes, it inherits the empty methods. In your extended class, you can override the methods you want and forget about the others. Both the `MouseAdapter` and `MouseMotionAdapter` classes are in the `java.awt.event` package.

The `DrawBoxes2` class shown in Code Listing 25-21 is a modification of the `DrawBoxes` class previously shown. In this version, the `MyMouseListener` class extends `MouseAdapter` and the `MyMouseMotionListener` class extends `MouseMotionAdapter`. This applet operates exactly the same as the `DrawBoxes` applet. The only difference is that this class does not have the empty methods in the listener classes.

NOTE: Java provides an adapter class for all of the interfaces in the API that have more than one method.

Code Listing 25-21 (`DrawBoxes2.java`)

```
1  import javax.swing.*;
2  import java.awt.event.*;
3  import java.awt.*;
4
5  /**
6     This applet demonstrates how the mouse adapter
7     classes can be used.
8  */
9
10 public class DrawBoxes2 extends JApplet
11 {
12    private int currentX = 0;      // Mouse cursor's X position
13    private int currentY = 0;      // Mouse cursor's Y position
14    private int width = 0;         // The rectangle's width
15    private int height = 0;        // The rectangle's height
16
17    /**
18       init method
19    */
20
21    public void init()
22    {
23       // Add a mouse listener and a mouse motion listener.
24       addMouseListener(new MyMouseListener());
25       addMouseMotionListener(new MyMouseMotionListener());
26    }
```

```
27
28    /**
29       paint method
30       @param g The applet's Graphics object.
31    */
32
33    public void paint(Graphics g)
34    {
35       // Call the superclass's paint method.
36       super.paint(g);
37
38       // Draw a rectangle.
39       g.drawRect(currentX, currentY, width, height);
40    }
41
42    /**
43       Mouse listener class
44    */
45
46    private class MyMouseListener extends MouseAdapter
47    {
48       public void mousePressed(MouseEvent e)
49       {
50          // Get the coordinates of the mouse cursor.
51          currentX = e.getX();
52          currentY = e.getY();
53       }
54    }
55
56    /**
57       Mouse Motion listener class
58    */
59
60    private class MyMouseMotionListener
61                            extends MouseMotionAdapter
62    {
63       public void mouseDragged(MouseEvent e)
64       {
65          // Calculate the size of the rectangle.
66          width = e.getX() - currentX;
67          height = e.getY() - currentY;
68
69          // Repaint the window.
70          repaint();
71       }
72    }
73 }
```

Checkpoint

MyProgrammingLab™ *www.myprogramminglab.com*

25.26 What is the difference between a mouse press event and a mouse click event?

25.27 What interface would a listener class implement to handle a mouse click event? A mouse press event? A mouse dragged event? A mouse release event? A mouse move event?

25.28 What type of object do mouse listener and mouse motion listener methods accept? What methods do these types of objects provide for determining a mouse cursor's location?

25.29 If a class implements the `MouseListener` interface but does not need to use all of the methods specified by the interface, can the definitions for those methods be left out? If not, how are these methods dealt with?

25.30 What is an adapter class, and how does it make some programming tasks easier?

25.7 `Timer` Objects

CONCEPT: A `Timer` object regularly generates action events at programmer-specified time intervals.

`Timer` objects automatically generate action events at regular time intervals. This is useful when you want a program to perform an operation at certain times or after an amount of time has passed.

`Timer` objects are created from the `Timer` class, which is in the `javax.swing` package. Here is the general format of the `Timer` class's constructor:

```
Timer(int delay, ActionListener listener)
```

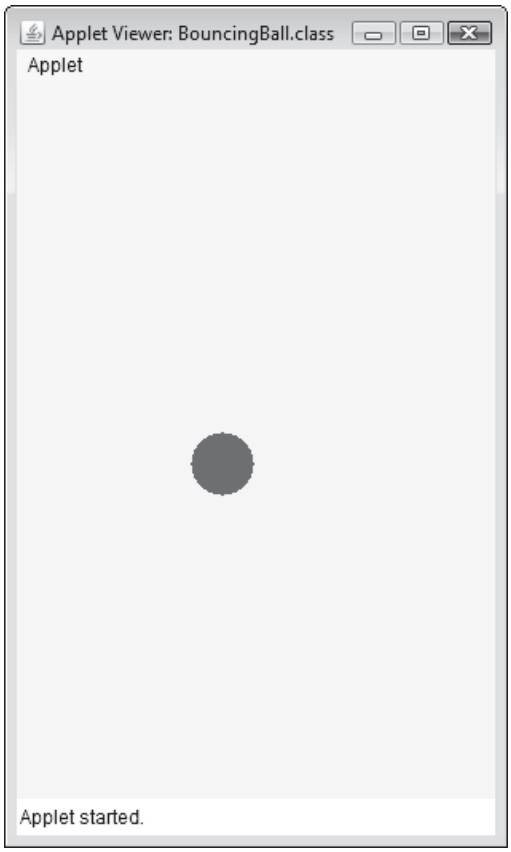
The argument passed into the *`delay`* parameter is the amount of time between action events, measured in milliseconds. A millisecond is a thousandth of a second, so a *`delay`* value of 1000 causes an action event to be generated every second. The argument passed into the *`listener`* parameter is a reference to an action listener that is to be registered with the `Timer` object. If you want to add an action listener at a later time, you can pass `null` as this argument, then use the `Timer` object's `addActionListener` method to register an action listener. Table 25-5 lists the `Timer` class's methods.

An application can use a `Timer` object to execute code automatically at regular time intervals. For example, a `Timer` object can be used to perform simple animation by moving a graphic image across the screen by a certain amount at regular time intervals. This is demonstrated in the `BouncingBall` class, shown in Code Listing 25-22. This class is an applet that displays a bouncing ball, as shown in Figure 25-31.

Table 25-5 Timer class methods

Method	Description
`void addActionListener (ActionListener listener)`	Registers the object referenced by *`listener`* as an action listener.
`int getDelay()`	Returns the current time delay in milliseconds.
`Boolean isRunning()`	Returns `true` if the `Timer` object is running. Otherwise, it returns `false`.
`void setDelay(int delay)`	Sets the time delay. The argument is the amount of the delay in milliseconds.
`void start()`	Starts the `Timer` object, which causes it to generate action events.
`void stop()`	Stops the `Timer` object, which causes it to stop generating action events.

Figure 25-31 `BouncingBall` applet (Oracle Corporate Counsel)



Code Listing 25-22 (BouncingBall.java)

```
1  import javax.swing.*;
2  import java.awt.event.*;
3  import java.awt.*;
4
5  /**
6     This applet uses a Timer object to animate
7     a bouncing ball.
8  */
9
10 public class BouncingBall extends JApplet
11 {
12    private final int X = 109;            // Ball's X coordinate
13    private final int WIDTH = 40;         // Ball's width
14    private final int HEIGHT = 40;        // Ball's height
15    private final int TIME_DELAY = 30;    // Time delay
16    private final int MOVE = 20;          // Pixels to move ball
17    private final int MINIMUM_Y = 50;     // Min height of ball
18    private final int MAXIMUM_Y = 400;    // Max height of ball
19    private int y = 400;                  // Ball's Y coordinate
20    private boolean goingUp = true;       // Direction indicator
21    private Timer timer;                  // Timer object
22
23
24    /**
25       init method
26    */
27
28    public void init()
29    {
30       timer = new Timer(TIME_DELAY, new TimerListener());
31       timer.start();
32    }
33
34    /**
35       paint method
36       @param g The applet's Graphics object.
37    */
38
39    public void paint(Graphics g)
40    {
41       // Call the superclass paint method.
42       super.paint(g);
43
44       // Set the drawing color to red.
45       g.setColor(Color.red);
46
```

```
47          // Draw the ball.
48          g.fillOval(X, y, WIDTH, HEIGHT);
49       }
50
51       /**
52           Private inner class that handles the Timer object's
53           action events.
54       */
55
56       private class TimerListener implements ActionListener
57       {
58          public void actionPerformed(ActionEvent e)
59          {
60             // Update the ball's Y coordinate.
61             if (goingUp)
62             {
63                if (y > MINIMUM_Y)
64                   y -= MOVE;
65                else
66                   goingUp = false;
67             }
68             else
69             {
70                if (y < MAXIMUM_Y)
71                   y += MOVE;
72                else
73                   goingUp = true;
74             }
75
76             // Force a call to the paint method.
77             repaint();
78          }
79       }
80 }
```

The `BouncingBall` class's `init` method creates a `Timer` object with the following statement in line 30:

```
timer = new Timer(TIME_DELAY, new TimerListener());
```

This initializes the object with a time delay of 30 milliseconds (the value of `TIME_DELAY`) and registers an instance of the `TimerListener` class as an action listener. This means that once the object is started, every 30 milliseconds it generates an action event, causing the action listener's `actionPerformed` method to execute. The next statement in the `init` method, in line 31, starts the `Timer` object as follows:

```
timer.start();
```

This causes the `Timer` object to commence generating action events. The `TimerListener` class's `actionPerformed` method calculates the new position of the bouncing ball and repaints the screen.

Checkpoint

MyProgrammingLab™ *www.myprogramminglab.com*

25.31 What type of events do `Timer` objects generate?

25.32 How are the time intervals between a `Timer` object's action events measured?

25.33 How do you cause a `Timer` object to begin generating action events?

25.34 How to you cause a `Timer` object to cease generating action events?

25.8 Playing Audio

CONCEPT: Sounds that have been stored in an audio file may be played from a Java program.

Java applets can play audio that is stored in a variety of popular sound file formats. The file formats directly supported are as follows:

- *.aif* or *.aiff* (Macintosh Audio File)
- *.au* (Sun Audio File)
- *.mid* or *.rmi* (MIDI File)
- *.wav* (Windows Wave File)

To play audio files, your computer must be equipped with a sound card and speakers. One way to play an audio file is to use the `play` method, which the `JApplet` class inherits from the `Applet` class. The version of the method that we will use is as follows:

```
void play(URL baseLocation, String fileName)
```

The argument passed to *`baseLocation`* is a `URL` object that specifies the location of the file. The argument passed to *`fileName`* is the name of the file. The sound that is recorded in the file is played one time.

When calling the `play` method, it is common to use either the `getDocumentBase` or `getCodeBase` method (both of which the `JApplet` class inherits from the `Applet` class) to get a `URL` object for the first argument. The `getDocumentBase` method returns a `URL` object containing the location of the HTML file that invoked the applet. Here is an example of a call to the `play` method, using a call to `getDocumentBase` for the first argument:

```
play(getDocumentBase(), "mysound.wav");
```

This statement will load and play the *mysound.wav* sound file, stored at the same location as the HTML file that invoked the applet.

The `getCodeBase` method returns a URL object containing the location of the applet's *.class* file. Here is an example of its use:

```
play(getCodeBase(), "mysound.wav");
```

This statement will load and play the *mysound.wav* sound file, stored at the same location as the applet's *.class* file. The *AudioDemo1* folder contains an example applet that plays a sound file using the `play` method.

NOTE: If the sound file specified by the arguments to the play method cannot be found, no sound will be played.

Using an `AudioClip` Object

The `Applet` class's `play` method loads a sound file, plays it one time, and then releases it for garbage collection. If you need to load a sound file to be played multiple times, you should use an *`AudioClip` object*.

An `AudioClip` object is an object that implements the `AudioClip` interface. The `AudioClip` interface is in the `java.applet` package, and it specifies the following three methods: `play`, `loop`, and `stop`. The `play` method plays a sound one time. The `loop` method repeatedly plays a sound, and the `stop` method causes a sound to stop playing.

The `Applet` class's `getAudioClip` method can be used to create an `AudioClip` object for a given sound file as follows:

```
AudioClip getAudioClip(URL baseLocation, String fileName)
```

The argument passed to *`baseLocation`* is a `URL` object that specifies the location of a sound file, and the argument passed to *`fileName`* is the name of the file. The method returns an `AudioClip` object that can be used to play the sound file.

As before, we can use the `getDocumentBase` or `getCodeBase` method to get a `URL` object for the first argument. Here is an example of a statement that uses the `getAudioClip` method:

```
AudioClip clip = getAudioClip(getDocumentBase(), "mysound.wav");
```

This statement declares `clip` as an `AudioClip` reference variable. The object returned by the `getAudioClip` method will load the *mysound.wav* file, stored at the same location as the HTML file that invoked the applet. The address of the object will be assigned to `clip`. The following statement can then be used to play the sound file:

```
clip.play();
```

The sound file can be played repeatedly with the following statement:

```
clip.loop();
```

Any time the sound file is being played, the following statement can be used to stop it:

```
clip.stop();
```

The `AudioDemo2` class shown in Code Listing 25-23 is an applet that uses an `AudioClip` object to play a sound file. The file *AudioDemo2.html* can be used to start the applet. Figure 25-32 shows the applet running. The Play button calls the `AudioClip` object's `play` method, causing the sound file to play once. The Loop button calls the `loop` method, causing the

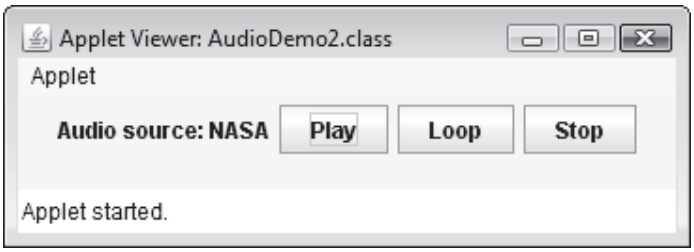
sound file to be played repeatedly. The Stop button stops the sound file from playing. The sound file that is played is a famous NASA transmission from the Moon. NASA provides a wealth of public domain audio, video, and image files. You can find such items by going to www.nasa.gov, and then search the site using search terms such as "audio clips", "video clips", etc.

Code Listing 25-23 (AudioDemo2.java)

```
1  import java.awt.*;
2  import java.applet.*;
3  import java.awt.event.*;
4  import javax.swing.*;
5
6  /**
7     This applet uses the AudioClip class to play a
8     sound. Sound source: NASA
9  */
10
11 public class AudioDemo2 extends JApplet
12 {
13    private JLabel credit;          // Displays NASA credit
14    private JButton playButton;     // Plays the sound clip
15    private JButton loopButton;     // Loops the clip
16    private JButton stopButton;     // Stops the clip
17    private AudioClip sound;        // Holds the sound clip
18
19    /**
20       init method
21    */
22
23    public void init()
24    {
25       // Create a layout manager.
26       setLayout(new FlowLayout());
27
28       // Make the credit label and add it.
29       credit = new JLabel("Audio source: NASA");
30       add(credit);
31
32       // Make the buttons and add them.
33       makeButtons();
34
35       // Get an AudioClip object for the sound file.
36       sound = getAudioClip(getDocumentBase(), "step.wav");
37    }
38
```

```
39    /**
40       The makeButtons method creates the Play, Loop, and
41       Stop buttons, and adds them to the content pane.
42    */
43
44    private void makeButtons()
45    {
46       // Create the Play, Loop, and Stop buttons.
47       playButton = new JButton("Play");
48       loopButton = new JButton("Loop");
49       stopButton = new JButton("Stop");
50
51       // Register an action listener with each button.
52       playButton.addActionListener(new ButtonListener());
53       loopButton.addActionListener(new ButtonListener());
54       stopButton.addActionListener(new ButtonListener());
55
56       // Add the buttons to the content pane.
57       add(playButton);
58       add(loopButton);
59       add(stopButton);
60    }
61
62    /**
63       Private inner class that handles the action event
64       that is generated when the user clicks one of the
65       buttons.
66    */
67
68    private class ButtonListener implements ActionListener
69    {
70       public void actionPerformed(ActionEvent e)
71       {
72          // Determine which button was clicked and
73          // perform the selected action.
74          if (e.getSource() == playButton)
75             sound.play();
76          else if (e.getSource() == loopButton)
77             sound.loop();
78          else if (e.getSource() == stopButton)
79             sound.stop();
80       }
81    }
82 }
```

Figure 25-32 AudioDemo2 applet (Oracle Corporate Counsel)



Playing Audio in an Application

The previous examples show how to play an audio file in an applet. You can play audio in an application as well. The process of getting a reference to an `AudioClip` object is different, however, in a class that does not extend `JApplet`. In the *Chapter 19\AudioDemo3* source code folder you will find a Swing application named *AudioFrame.java* that demonstrates how to do it. The following code segment is from the application.

```
43      // Create a file object for the step.wav file.
44      File file = new File("step.wav");
45
46      // Get a URI object for the audio file.
47      URI uri = file.toURI();
48
49      // Get a URL for the audio file.
50      URL url = uri.toURL();
51
52      // Get an AudioClip object for the sound
53      // file using the Applet class's static
54      // newAudioClip method.
55      sound = Applet.newAudioClip(url);
```

In line 44, we create a `File` object representing the audio file. Then, in line 47 we call the File class's `toURI` method to create a `URI` object representing the audio file. The `URI` class is in the `java.net` package. (URI stands for Uniform Resource Identifier.)

Then, in line 50, we call the `URI` class's `toURL` method to create a `URL` object representing the audio file. Note that if this method cannot construct a `URL` it throws a checked exception—`MalformedURLException`. The `MalformedURLException` class is in the `java.net` package.

Last, in line 55, we call the `Applet` class's static `newAudioClip` method, passing the `URL` object as an argument. The method returns a reference to an `AudioClip` object which can be used as previously demonstrated to play the audio file.

Checkpoint

MyProgrammingLab™ *www.myprogramminglab.com*

25.35 What `Applet` method can you use to play a sound file?

25.36 What is the difference between using the `Applet` method asked for in Checkpoint 25.35, and using an `AudioClip` object to play a sound file?

25.37 What methods does an `AudioClip` object have? What do they do?

25.38 What is the difference between the `Applet` class's `getDocumentBase` and `getCodeBase` methods?

25.9 Common Errors to Avoid

- **Forgetting a closing tag in an HTML document.** Most HTML tags have an opening tag and a closing tag. The page will not appear properly if you forget a closing tag.
- **Confusing the `<head></head>` tag with `<h1></h1>` or another header tag.** The `<head></head>` tag marks a document's head section, whereas the `<h1></h1>` tag marks a header, which is large bold text.
- **Using *X* and/or *Y* coordinates that are outside of the component when drawing a shape.** If you use coordinates that are outside the component to draw a shape, the shape will not appear.
- **Not calling the superclass's `paint` or `paintComponent` method.** When you override the `paint` or `paintComponent` method, the overriding method should call the superclass's version of the method before doing anything else.
- **Overriding the `paint` method with a component extended from `JComponent`.** You should override the `paint` method only with AWT components, `JFrame` components, or `JApplet` components.
- **Not calling the `repaint` method to redisplay a window.** When you update the data used to draw shapes on a component, you must call the `repaint` method to force a call to the `paint` or `paintComponent` method.
- **Not providing empty definitions for the unneeded methods in a mouse listener or mouse motion listener class.** When writing mouse listeners or mouse motion listeners, you must provide definitions for all the methods specified by the listener interfaces. To avoid this you can write a listener as a class that inherits from an adapter class.
- **Forgetting to start a `Timer` object.** A `Timer` object does not begin generating action events until it is started with a call to its `start` method.

Review Questions and Exercises

Multiple Choice and True/False

1. This section of an HTML document contains all of the tags and text that produce output in the browser window.
   a. head
   b. content
   c. body
   d. output

2. You place the `<title></title>` tag in this section of an HTML document.
   a. head
   b. content
   c. body
   d. output
3. Everything that appears between these tags in an HTML document is the content of the Web page.
   a. `<content></content>`
   b. `<html></html>`
   c. `<head></head>`
   d. `<page></page>`
4. To create a level one header you use this tag.
   a. `<level1></level1>`
   b. `<header1></header1>`
   c. `<h1></h1>`
   d. `<head></head>`
5. When using Swing to write an applet, you extend the applet's class from this class.
   a. `Applet`
   b. `JApplet`
   c. `JFrame`
   d. `JAppletFrame`
6. When using AWT to write an applet, you extend the applet's class from this class.
   a. `Applet`
   b. `JApplet`
   c. `JFrame`
   d. `JAppletFrame`
7. This applet method is invoked instead of a constructor.
   a. `startUp`
   b. `beginApplet`
   c. `invoke`
   d. `init`
8. The Sun JDK comes with this program, which loads and executes an applet without the need for a Web browser.
   a. `applettest`
   b. `appletload`
   c. `appletviewer`
   d. `viewapplet`
9. A class that inherits from `Applet` or `Frame` does not have one of these.
   a. an `add` method
   b. an `init` method
   c. a content pane
   d. a layout manager

10. What location on a component usually has the coordinates (0, 0)?
    a. upper-right corner
    b. upper-left corner
    c. center
    d. lower-right corner
11. In a class that extends `JApplet` or `JFrame` you override this method to get a reference to the `Graphics` object.
    a. `paint`
    b. `paintComponent`
    c. `getGraphics`
    d. `graphics`
12. In a class that extends `JPanel` you override this method to get a reference to the `Graphics` object.
    a. `paint`
    b. `paintComponent`
    c. `getGraphics`
    d. `graphics`
13. The `drawLine` method is a member of this class.
    a. `JApplet`
    b. `Applet`
    c. `JFrame`
    d. `Graphics`
14. To force the `paint` method to be called to update a component's display, you _____.
    a. call the `paint` method
    b. call the `repaint` method
    c. call the `paintAgain` method
    d. do nothing; you cannot force the `paint` method to be called
15. A class that implements this interface can handle mouse dragged events.
    a. `MouseListener`
    b. `ActionListener`
    c. `MouseMotionListener`
    d. `MouseDragListener`
16. A class that implements this interface can handle mouse click events.
    a. `MouseListener`
    b. `ActionListener`
    c. `MouseMotionListener`
    d. `MouseDragListener`
17. This `MouseEvent` method returns the *X* coordinate of the mouse cursor at the moment the mouse event is generated.
    a. `getXCoord`
    b. `getMouseX`
    c. `getPosition`
    d. `getX`

18. If a class implements a standard API interface that specifies more than one method but does not need many of the methods, this should be used instead of the interface.
    a. your own detailed versions of the needed methods
    b. an adapter class
    c. a different interface
    d. there is no other choice
19. A `Timer` object's time delay between events is specified in this unit of time.
    a. seconds
    b. microseconds
    c. milliseconds
    d. minutes
20. A `Timer` object generates this type of event.
    a. action events
    b. timer events
    c. item events
    d. interval events
21. The following `Applet` class method returns a `URL` object with the location of the HTML file that invoked the applet.
    a. `getHTMLlocation`
    b. `getDocumentBase`
    c. `getAppletBase`
    d. `getCodeBase`
22. The following `Applet` class method returns a `URL` object with the location of the applet's *.class* file.
    a. `getHTMLlocation`
    b. `getDocumentBase`
    c. `getAppletBase`
    d. `getCodeBase`
23. **True or False:** Applets cannot create files on the user's system.
24. **True or False:** Applets can read files on the user's system.
25. **True or False:** Applets cannot make network connections with any system except the server from which the applet was transmitted.
26. **True or False:** Applets can retrieve information about the user's system or the user's identity.
27. **True or False:** The `<h6>` tag produces larger text than the `<h1>` tag.
28. **True or False:** You use a static `main` method to create an instance of an applet class.
29. **True or False:** In a class that extends `JApplet`, you add components to the content pane.
30. **True or False:** In an applet, events are handled differently than in a GUI application.
31. **True or False:** An object of the `Frame` class does not have a content pane.
32. **True or False:** In an overriding `paint` method, you should never call the superclass's version of the `paint` method.

33. **True or False:** Once a `Timer` object has been started, it cannot be stopped without shutting down the program.
34. **True or False:** The `Applet` class's play method loads and plays an audio file once and then releases the memory it occupies for garbage collection.
35. **True or False:** The `loop` and `stop` methods, for use with audio files, are part of the `Applet` class.

Find the Error

Find the errors in the following code:

1.
```
<applet code="MyApplet.java" width=100 height=50>
</applet>
```
2.
```
public void paint(Graphics g)
{
   drawLine(0, 0, 100, 100);
}
```
3.
```
// Force a call to the paint method.
paint();
```
4.
```
public class MyPanel extends JPanel
{
   public MyPanel()
   {
      // Constructor code...
   }
   public void paint(Graphics g)
   {
      //paint method code...
   {
}
```
5.
```
private class MyMouseListener implements MouseListener
{
   public void mouseClicked(MouseEvent e)
   {
      mouseClicks += 1;
   }
}
```
6.
```
private class MyMouseListener implements MouseAdapter
{
   public void mouseClicked(MouseEvent e)
   {
      mouseClicks += 1;
   }
}
```

Algorithm Workbench

1. Write the text and HTML tags necessary to display "My Home Page" as a level one header, centered in the browser window.
2. You have written an applet and saved the source code in a file named `MyApplet.java`. Write the HTML tag needed to execute the applet in an area that is 300 pixels wide by 200 pixels high. Assume that the compiled applet code is stored in the same directory as the HTML document.
3. Look at the following GUI application class and indicate by line number the changes that should be made to convert this to an applet using Swing:

```
 1 public class SimpleWindow extends JFrame
 2 {
 3    public SimpleWindow()
 4    {
 5       // Set the title.
 6       setTitle("A Simple Window");
 7
 8       // Specify what happens when the close button is clicked.
 9       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
10
11       // Add a label.
12       JLabel label = new JLabel("This is a simple window.");
13       add(label);
14
15       // Pack and display the window.
16       pack();
17       setVisible(true);
18    }
19 }
```

4. Assume that `g` references a `Graphics` object. Write code that performs the following:
   a. Draws an outline of a rectangle that is 100 pixels wide by 200 pixels high, with its upper-left corner at (50, 75).
   b. Draws a filled rectangle that is 300 pixels wide by 100 pixels high, with its upper-left corner at (10, 90).
   c. Draws a blue outline of an oval with an enclosing rectangle that is 100 pixels wide by 50 pixels high, with its upper-left corner at (10, 25).
   d. Draws a red line from (0, 5) to (150, 175).
   e. Draws the string "Greetings Earthling". The lower-left point of the string should be at (80, 99). Use a bold, 20-point serif font.
   f. Draws a polygon with vertices at the following points: (10, 10), (10, 25), (50, 25), and (50, 10). What shape does this code result in?

5. Rewrite the following mouse motion listener so it uses an adapter class:

```
private class MyMouseMotionListener implements MouseMotionListener
{
   public void mouseDragged(MouseEvent e)
   {
   }
   public void mouseMoved(MouseEvent e)
   {
      mouseMovements += 1;
   }
}
```

6. Assume that a class has an inner class named `MyTimerListener` that can be used to handle the events generated by a `Timer` object. Write code that creates a `Timer` object with a time delay of one half second. Register an instance of `MyTimerListener` with the class.

Short Answer

1. When a user accesses a Web page on a remote server with his or her browser, and that Web page has an applet associated with it, is the applet executed by the server or by the user's system?
2. List at least three security restrictions imposed on applets.
3. Why are applets sometimes necessary in Web page development?
4. Why isn't it necessary to call the `setVisible` method to display an applet?
5. Why would you ever need to use the older AWT library instead of Swing to develop an applet?
6. A panel is 600 pixels wide by 400 pixels high. What are the *X* and *Y* coordinates of the pixel in the upper-left corner? The upper-right corner? The lower-left corner? The lower-right corner? The center of the panel?
7. When is a component's `paint` or `paintComponent` method called?
8. What is an adapter class? How does it make some programming tasks more convenient? Under what circumstances does the Java API provide an adapter class?
9. Under what circumstances would you want to use an `AudioClip` object to play a sound file, rather than the `Applet` class's play method?

Programming Challenges

MyProgrammingLab™ *Visit www.myprogramminglab.com to complete many of these Programming Challenges online and get instant feedback.*

1. `FollowMe` Applet

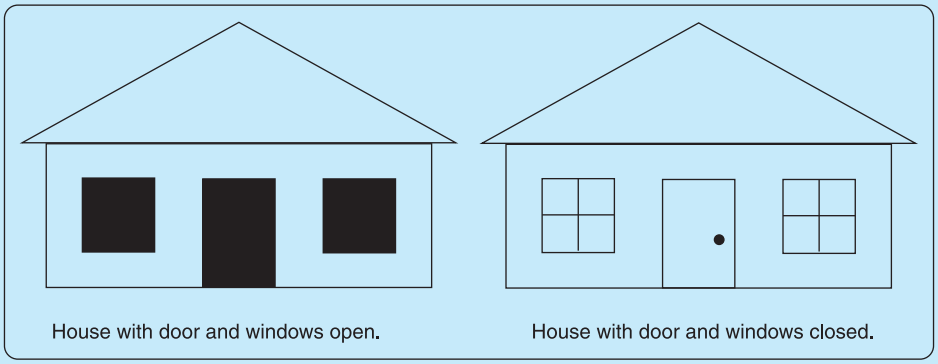
Write an applet that initially displays the word "Hello" in the center of a window. The word should follow the mouse cursor when it is moved inside the window.

VideoNote
The House Applet Problem

2. House Applet

Write an applet that draws the house shown on the left in Figure 25-33. When the user clicks on the door or windows, they should close. The figure on the right shows the house with its door and windows closed.

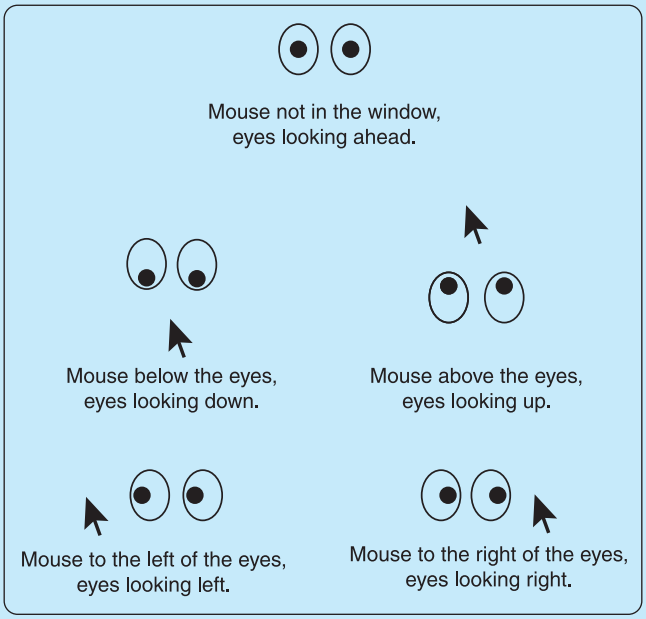
Figure 25-33 House drawing



3. WatchMe Applet

Write an applet that displays a drawing of two eyes in the center of its window. When the mouse cursor is not inside the window, the eyes should look ahead. When the mouse cursor is inside the window, the eyes should follow the cursor. This is illustrated in Figure 25-34.

Figure 25-34 Eyes following the mouse cursor



4. `Thermometer` Applet

Write an applet that displays a thermometer. The user should be able to control the temperature with a slider component. When the user moves the slider, the thermometer should show the corresponding temperature.

5. Polygon Drawer

Write an applet that lets the user click on six points. After the sixth point is clicked, the applet should draw a polygon with a vertex at each point the user clicked.

6. `GridFiller` Applet

Write an applet that displays a 4×4 grid. When the user clicks on a square in the grid, the applet should draw a filled circle in it. If the square already has a circle, clicking on it should cause the circle to disappear.

7. `DrinkMachine` Applet

Write an applet that simulates a soft drink vending machine. The simulated machine dispenses the following soft drinks: cola, lemon-lime soda, grape soda, root beer, and bottled water. These drinks cost $0.75 each to purchase.

When the applet starts, the drink machine should have a supply of 20 of each of the drinks. The applet should have a text field where the user can enter the amount of money he or she is giving the machine. The user can then click on a button to select a drink to dispense. The applet should also display the amount of change it is giving back to the user. The applet should keep track of its inventory of drinks and inform the user whether he or she has selected a drink that is out of stock. Be sure to handle operator errors such as selecting a drink with no money entered and selecting a drink with an inadequate amount of money entered.

8. `Stopwatch` Applet

Write an applet that simulates a stopwatch. It should have a Start button and a Stop button. When the Start button is clicked the applet should count the seconds that pass. When the Stop button is clicked, the applet should stop counting seconds.

9. Slideshow Application

Write an application that displays a slideshow of images, one after the other, with a time delay between each image. The user should be able to select up to 10 images for the slide show and specify the time delay in seconds.