Code Listing 14-19 (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 14-29 MouseEvents applet

Using Adapter Classes

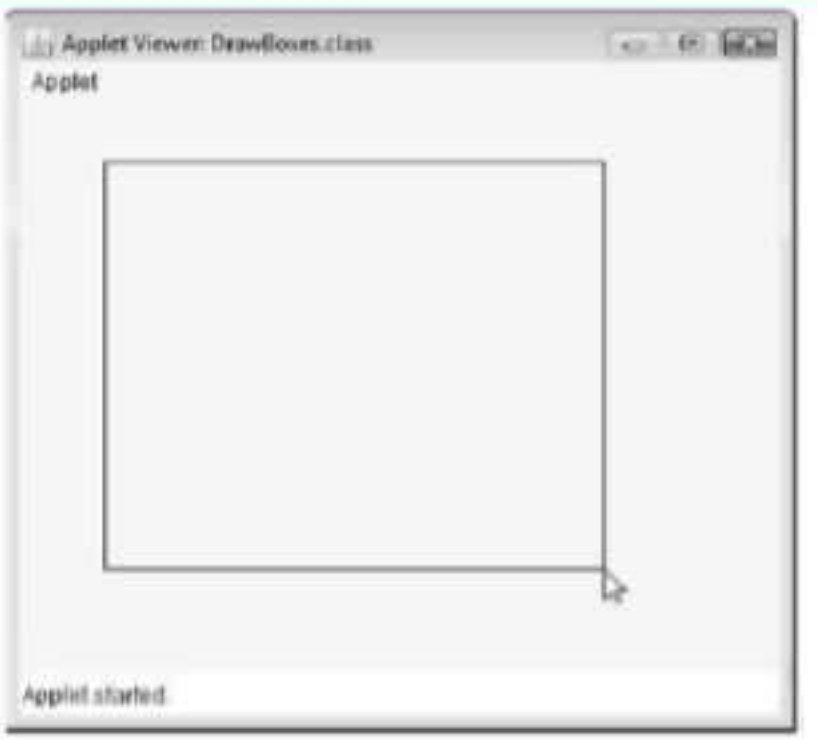
Many times when you handle mouse events, you will not be interested in handling every event that the mouse generates. This is the case with the `DrawBoxes` applet, which handles only mouse pressed and mouse dragged events.

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 14-30 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 14-20 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 14-30 `DrawBoxes` applet



Code Listing 14-20 (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 14-21 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 14-21 (`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*

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

14.27 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?

14.28 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?

14.29 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?

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

14.7 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)
```

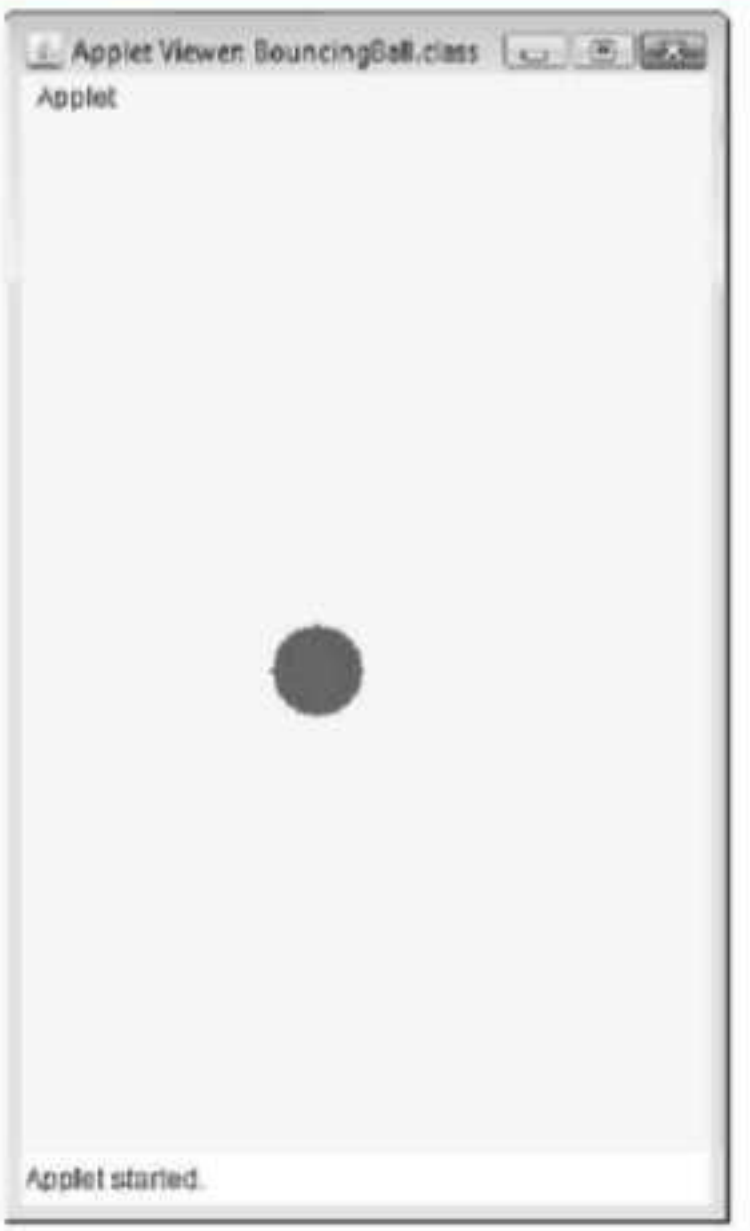
The argument passed into the *`delay`* parameter is the amount of time between action events, measured in milliseconds. A millisecond is a thousandth of a second, so a *`delay`* value of 1000 causes an action event to be generated every second. The argument passed into the *`listener`* parameter is a 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 14-5 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 14-22. This class is an applet that displays a bouncing ball, as shown in Figure 14-31.

Table 14-5 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 14-31 `BouncingBall` applet



Code Listing 14-22 (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;      // Max height of ball
18    private final int MAXIMUM_Y = 400;     // Min 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*

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

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

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

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

14.8 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)

In order 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 14-23 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 14-32 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 14-23 (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 14-32 AudioDemo2 applet

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 14\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*

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

14.36 What is the difference between using the `Applet` method asked for in Checkpoint 14.35, and using an `AudioClip` object to play a sound file?

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

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

14.9 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

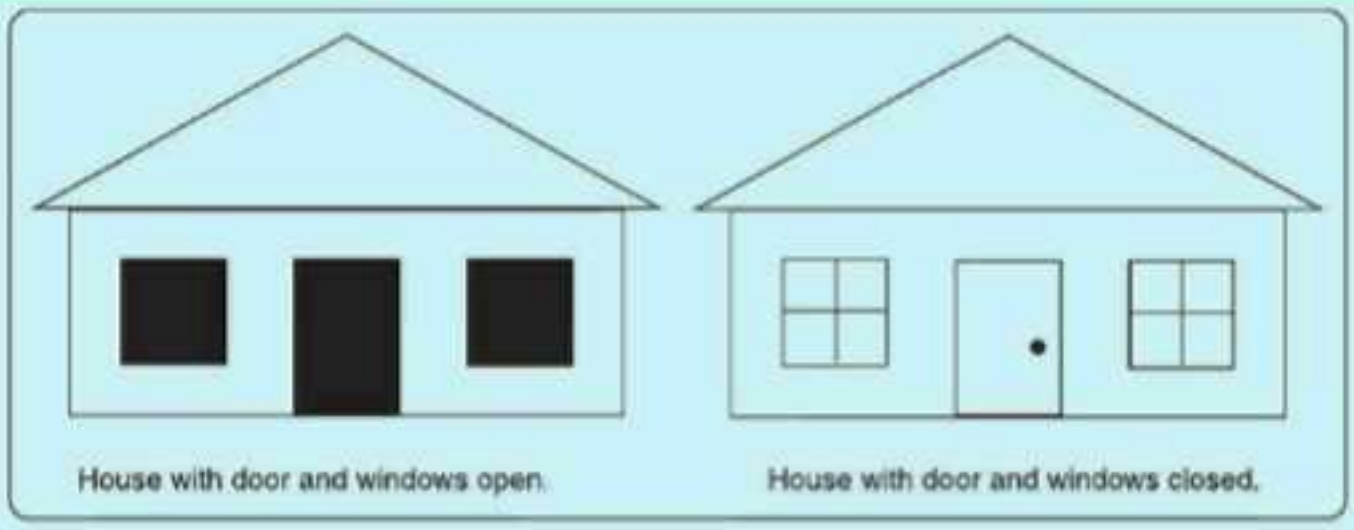
Write an applet that initially displays the word "Hello" in the center of a window. The word should follow the mouse cursor when it is moved inside the window.

VideoNote
The `House` Applet Problem

2. `House` Applet

Write an applet that draws the house shown on the left in Figure 14-33. 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 14-33 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 14-34.

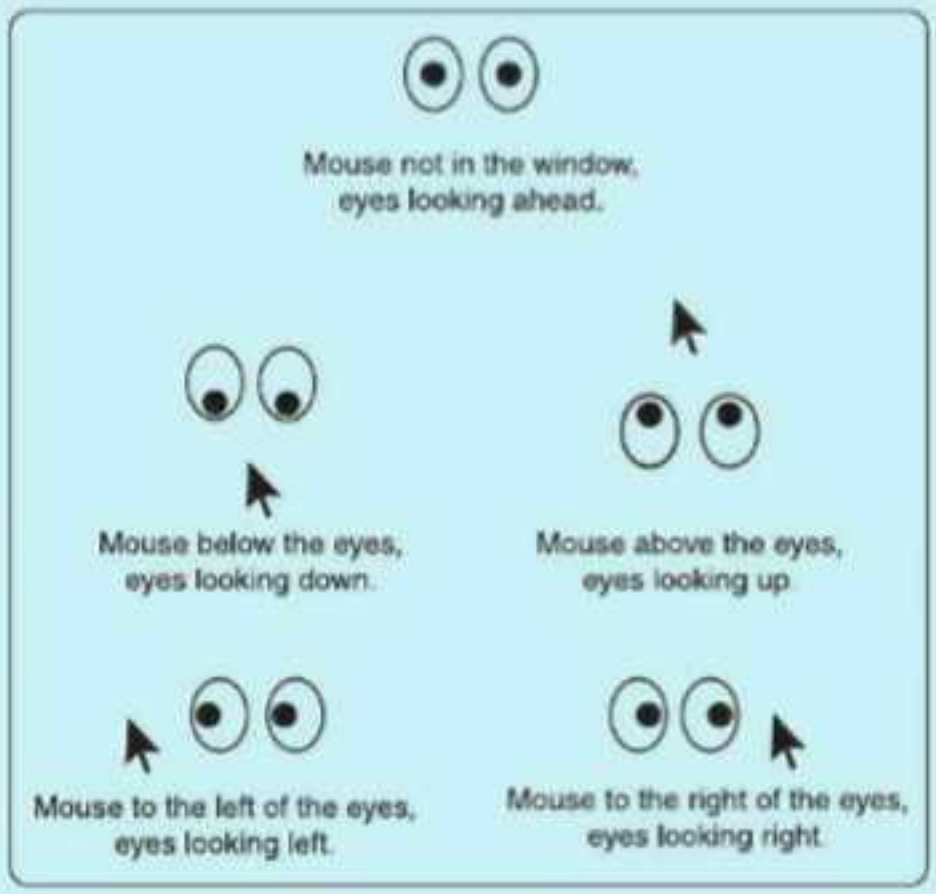
4. `Thermometer` Applet

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.

Figure 14-34 Eyes following the mouse cursor



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.

CHAPTER

15 Recursion

TOPICS

15.1 Introduction to Recursion

CONCEPT: A recursive method is a method that calls itself.

You have seen instances of methods calling other methods. Method A can call method B, which can then call method C. It's also possible for a method to call itself. A method that calls itself is a recursive method. Look at the `message` method in Code Listing 15-1.

Code Listing 15-1 (EndlessRecursion.java)

```
1 /**
2     This class has a recursive method.
3 */
4
5 public class EndlessRecursion
6 {
7    public static void message()
8    {
9       System.out.println("This is a recursive method.");
10      message();
11   }
12 }
```

This method displays the string "This is a recursive method." and then calls itself. Each time it calls itself, the cycle is repeated. Can you see a problem with the method? There's no way to stop the recursive calls. This method is like an infinite loop because there is no code to stop it from repeating.

Like a loop, a recursive method must have some way to control the number of times it repeats. The class in Code Listing 15-2 has a modified version of the `message` method. It passes an integer argument, which holds the number of times the method should call itself.

Code Listing 15-2 (Recursive.java)

```
1 /**
2    This class has a recursive method, message,
3    which displays a message n times.
4 */
5
6 public class Recursive
7 {
8    public static void message(int n)
9    {
10      if (n > 0)
11      {
12         System.out.println("This is a recursive method.");
13         message(n - 1);
14      }
15   }
16 }
```

This method contains an `if` statement that controls the repetition. As long as the `n` parameter is greater than zero, the method displays the message and calls itself again. Each time it calls itself, it passes `n - 1` as the argument. For example, look at the program in Code Listing 15-3.

Code Listing 15-3 (RecursionDemo.java)

```
1 /**
2    This class demonstrates the Recursive.message method.
3 */
4
5 public class RecursionDemo
6 {
7    public static void main(String[] args)
8    {
9       Recursive.message(5);
10   }
```