21

Multimedia: Applets and Applications

The wheel that squeaks the loudest … gets the grease.

— John Billings (Henry Wheeler Shaw)

We'll use a signal I have tried and found far-reaching and easy to yell. Waa-hoo!

— Zane Grey

There is a natural hootchy-kootchy motion to a goldfish.

— Walt Disney

Between the motion and the act falls the shadow.

— Thomas Stearns Eliot

OBJECTIVES

In this chapter you will learn:

- How to get and display images.
- How to create animations from sequences of images.
- How to create image maps.
- How to get, play, loop and stop sounds, using an **`AudioClip`**.
- How to play video using interface **`Player`**.

Outline

21.1 Introduction

- **Multimedia – the "sizzle" of Java**
  - **Sound, images, graphics and video**
  - **An enormous programming field**
  - **Demands extraordinary computing power**
- **Many computer users now want three-dimensional, high-resolution, color graphics**
- **Java provides extensive multimedia facilities, including:**
  - **Java 3D API – for creating 3D graphics applications**
  - **JMF API – for adding audio and video to an application**
  - **Java Sound API – for playing, recording and modifying audio**
  - **Java Speech API – for inputting and outputting voice commands**

21.2 Loading, Displaying and Scaling Images

- Classes `Image` and `ImageIcon` – used to load and display images
- Displaying images
  - `Graphics` method `drawImage` – used to draw image referenced by `Image` object (can be scaled)
  - `ImageIcon` method `paintIcon` can be used to draw image referenced by `ImageIcon` object
- Loading images
  - `Applet` method `getImage` loads an `Image` into an applet
  - Applet method `getDocumentBase` returns location of applet's HTML file on Internet
  - `ImageObserver`s receive notifications as `Image` is loaded and update image on screen if it was not complete when displayed
- Java supports several image formats, including GIF, JPEG and PNG

```
1  // Fig. 21.1: LoadImageAndScale.java
2  // Load an image and display it in its original size and twice its
3  // original size. Load and display the same image as an ImageIcon.
4  import java.awt.Graphics;
5  import java.awt.Image;
6  import javax.swing.ImageIcon;
7  import javax.swing.JApplet;
8
9  public class LoadImageAndScale extends JApplet
10 {
11    private Image image1; // create Image object
12    private ImageIcon image2; // create ImageIcon object
13
14    // load image when applet is loaded
15    public void init()
16    {
17       image1 = getImage( getDocumentBase(), "redflowers.png" );
18       image2 = new ImageIcon( "yellowflowers.png" );
19    } // end method init
20
21    // display image
22    public void paint( Graphics g )
23    {
24       super.paint( g );
25
26       g.drawImage( image1, 0, 0, this ); // draw original image
27
28       // draw image to fit the width and the height less 120 pixels
29       g.drawImage( image1, 0, 120, getWidth(), getHeight() - 120, this );
30
```

Returns location of HTML file as URL object

Create ImageIcon object for file yellowflowers.jpg

Draw image stored in redflowers.jpg

Draw same image scaled to different size

```
31       // draw icon using its paintIcon method
32       image2.paintIcon( this, g, 180, 0 );
33    } // end method paint
34 } // end class LoadImageAndScale
```

Portability Tip 21.1

Class `Image` is an `abstract` class—as a result, programs cannot instantiate class `Image` to create objects. To achieve platform independence, the Java implementation on each platform provides its own subclass of `Image` to store image information.

21.3 Animating a Series of Images

- Animation can be created by displaying a sequence of similar images
- `Timer` object can be used to specify when each image is displayed
- `Timer` objects generate `ActionEvents` at fixed intervals
  - Method `start` – `Timer` should start generating events
  - Method `stop` – `Timer` should stop generating events
  - Method `restart` – `Timer` should start generating events again
- `Component` method `getPreferredSize` determines the preferred width and height of a component
- `Component` method `getMinimumSize` determines the minimum width and height of a component

```
1  // Fig. 21.2: LogoAnimatorJPanel.java
2  // Animation of a series of images.
3  import java.awt.Dimension;
4  import java.awt.event.ActionEvent;
5  import java.awt.event.ActionListener;
6  import java.awt.Graphics;
7  import javax.swing.ImageIcon;
8  import javax.swing.JPanel;
9  import javax.swing.Timer;
10
11 public class LogoAnimatorJPanel extends JPanel
12 {
13    private final static String IMAGE_NAME = "deitel"; // base image name
14    protected ImageIcon images[]; // array of images
15    private final int TOTAL_IMAGES = 30; // number of images
16    private int currentImage = 0; // current image index
17    private final int ANIMATION_DELAY = 50; // millisecond delay
18    private int width; // image width
19    private int height; // image height
20
21    private Timer animationTimer; // Timer drives animation
22
23    // constructor initializes LogoAnimatorJPanel by loading images
24    public LogoAnimatorJPanel()
25    {
26       images = new ImageIcon[ TOTAL_IMAGES ];
27
```

Will be used to store images to be animated

```
28       // load 30 images
29       for ( int count = 0; count < images.length; count++ )
30          images[ count ] = new ImageIcon( getClass().getResource(
31             "images/" + IMAGE_NAME + count + ".gif" ) );
32
33       // this example assumes all images have the same width and height
34       width = images[ 0 ].getIconWidth();   // get ic
35       height = images[ 0 ].getIconHeight(); // get icon height
36    } // end LogoAnimatorJPanel constructor
37
38    // display current image
39    public void paintComponent( Graphics g )
40    {
41       super.paintComponent( g ); // call superclass paintComponent
42
43       images[ currentImage ].paintIcon( this, g, 0, 0 );
44
45       // set next image to be drawn only if timer is running
46       if ( animationTimer.isRunning() )
47          currentImage = ( currentImage + 1 ) % TOTAL_IMAGES;
48    } // end method paintComponent
49
```

Create and store `ImageIcon` for each image

Set next image only if `Timer` is still running

```
50    // start animation, or restart if window is redisplayed
51    public void startAnimation()
52    {
53       if ( animationTimer == null )
54       {
55          currentImage = 0; // display first image
56
57          // create timer
58          animationTimer =
59             new Timer( ANIMATION_DELAY, new TimerHandler() );
60
61          animationTimer.start(); // start timer
62       } // end if
63       else // animationTimer already exists, restart animation
64       {
65          if ( ! animationTimer.isRunning() )
66             animationTimer.restart();
67       } // end else
68    } // end method startAnimation
69
70    // stop animation timer
71    public void stopAnimation()
72    {
73       animationTimer.stop();
74    } // end method stopAnimation
75
```

Create `Timer` so images will be displayed at intervals of length `ANIMATION_DELAY`

Allow `Timer` to start generating events

Allow `Timer` to start generating events again

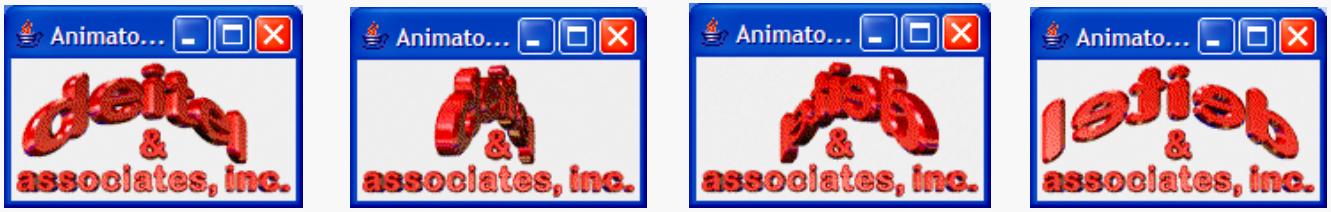
Stop `Timer` from generating events

```
76    // return minimum size of animation
77    public Dimension getMinimumSize()
78    {
79       return getPreferredSize();
80    } // end method getMinimumSize
81
82    // return preferred size of animation
83    public Dimension getPreferredSize()
84    {
85       return new Dimension( width, height );
86    } // end method getPreferredSize
87
88    // inner class to handle action events from Timer
89    private class TimerHandler implements ActionListener
90    {
91       // respond to Timer's event
92       public void actionPerformed( ActionEvent actionEvent )
93       {
94          repaint(); // repaint animator
95       } // end method actionPerformed
96    } // end class TimerHandler
97 } // end class LogoAnimatorJPanel
```

Define minimum size for `JPanel`

Define preferred size for `JPanel`

```
1  // Fig. 21.3: LogoAnimator.java
2  // Animation of a series of images.
3  import javax.swing.JFrame;
4
5  public class LogoAnimator
6  {
7     // execute animation in a JFrame
8     public static void main( String args[] )
9     {
10       LogoAnimatorJPanel animation = new LogoAnimatorJPanel();
11
12       JFrame window = new JFrame( "Animator test" ); // set up window
13       window.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
14       window.add( animation ); // add panel to frame
15
16       window.pack();  // make window just large enough for its GUI
17       window.setVisible( true );   // display window
18
19       animation.startAnimation();  // begin animation
20    } // end main
21 } // end class LogoAnimator
```



Software Engineering Observation 21.1

When creating an animation for use in an applet, provide a mechanism for disabling the animation when the user browses a new Web page different from the one on which the animation applet resides.

Look-and-Feel Observation 21.1

The default size of a `JPanel` object is 10 pixels wide and 10 pixels tall.

Look-and-Feel Observation 21.2

When subclassing `JPanel` (or any other `JComponent`), override method `getPreferredSize` if the new component is to have a specific preferred width and height.

Look-and-Feel Observation 21.3

If a new GUI component has a minimum width and height (i.e., smaller dimensions would render the component ineffective on the display), override method `getMinimumSize` to return the minimum width and height as an instance of class `Dimension`.

Look-and-Feel Observation 21.4

For many GUI components, method `getMinimumSize` is implemented to return the result of a call to the component's `getPreferredSize` method.

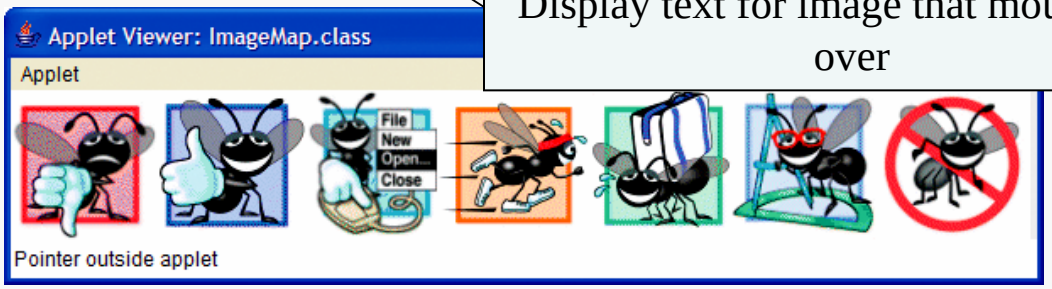
21.4 Image Maps

- **Image maps used to create interactive Web pages**
- **Contains hot areas user can click to accomplish a task**
- **When user positions mouse pointer over hot area, normally a descriptive message is displayed**
- **`Applet` method `showStatus` displays text in an applet container's status bar**

```
1  // Fig. 21.4: ImageMap.java
2  // Demonstrating an image map.
3  import java.awt.event.MouseAdapter;
4  import java.awt.event.MouseEvent;
5  import java.awt.event.MouseMotionAdapter;
6  import java.awt.Graphics;
7  import javax.swing.ImageIcon;
8  import javax.swing.JApplet;
9
10 public class ImageMap extends JApplet
11 {
12    private ImageIcon mapImage;
13
14    private static final String captions[] = { "Common Programming Error",
15       "Good Programming Practice", "Graphical User Interface Tip",
16       "Performance Tip", "Portability Tip",
17       "Software Engineering Observation", "Error-Prevention Tip" };
18
19    // sets up mouse listeners
20    public void init()
21    {
22       addMouseListener(
23
```

```
24          new MouseAdapter() // anonymous inner class
25          {
26             // indicate when mouse pointer exits applet area
27             public void mouseExited( MouseEvent event )
28             {
29                showStatus( "Pointer outside applet" );
30             } // end method mouseExited
31          } // end anonymous inner class
32       ); // end call to addMouseListener
33
34       addMouseMotionListener(
35
36          new MouseMotionAdapter() // anonymous inner class
37          {
38             // determine icon over which mouse appears
39             public void mouseMoved( MouseEvent event )
40             {
41                showStatus( translateLocation(
42                   event.getX(), event.getY() ) );
43             } // end method mouseMoved
44          } // end anonymous inner class
45       ); // end call to addMouseMotionListener
46
47       mapImage = new ImageIcon( "icons.png" ); // get image
48    } // end method init
49
```

```
50    // display mapImage
51    public void paint( Graphics g )
52    {
53       super.paint( g );
54       mapImage.paintIcon( this, g, 0, 0 );
55    } // end method paint
56
57    // return tip caption based on mouse coordinates
58    public String translateLocation( int x, int y )
59    {
60       // if coordinates outside image, return immediately
61       if ( x >= mapImage.getIconWidth() || y >= mapImage.getIconHeight() )
62          return "";
63
64       // determine icon number (0 - 6)
65       double iconWidth = ( double ) mapImage.getIconWidth() / 7.0;
66       int iconNumber = ( int )( ( double ) x / iconWidth );
67
68       return captions[ iconNumber ]; // return appropriate icon caption
69    } // end method translateLocation
70 } // end class ImageMap
```



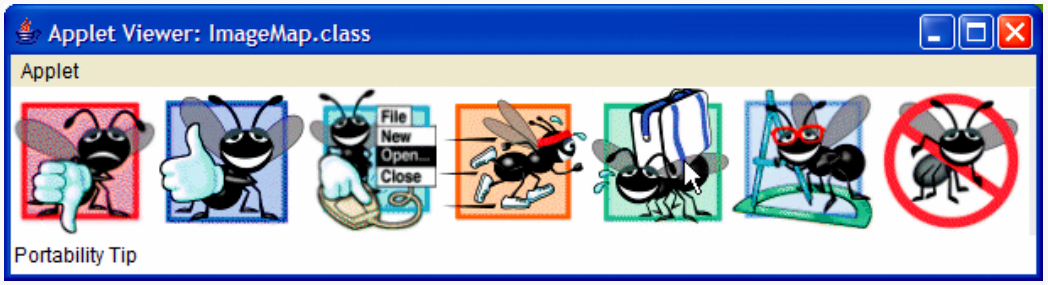
Method called when mouse is moved

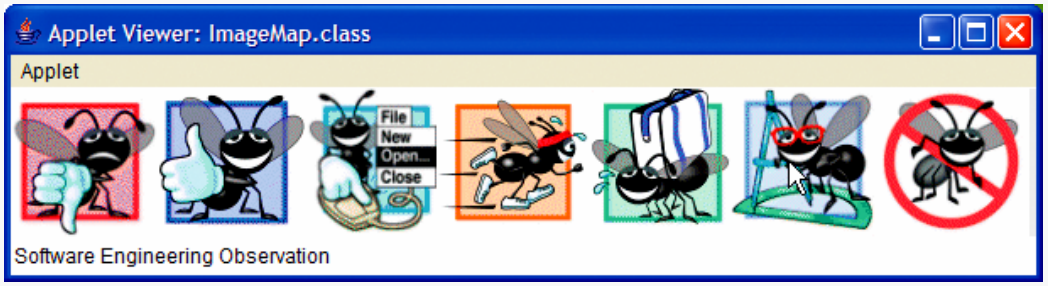
Current mouse coordinates

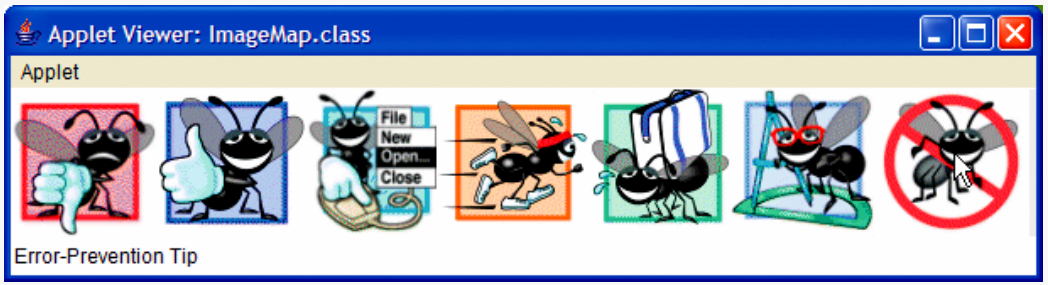
Do nothing if mouse is not over an icon

Determine which icon the mouse is over

Display text for image that mouse is over







21.5 Loading and Playing Audio Clips

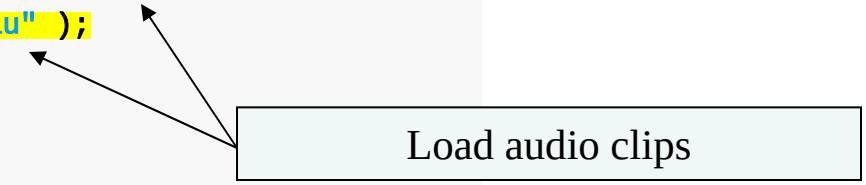
- **Java programs can play and manipulate audio clips**
- **Playing sounds in an applet**
  - **`Applet`'s `play` method – loads sound and plays once**
  - **`AudioClip`'s `play`, `loop` and `stop` methods**
  - **Additional capabilities provided by JMF and Java Sound APIs**
- **Loading sounds in an applet**
  - **`Applet` method `getAudioClip` – retrieves sound, returns reference to an `AudioClip`**
  - **`Applet`'s `play` method loads sound**
- **Supported file formats include Sun Audio file format, Windows Wave file format, MIDI file format**

```
1  // Fig. 21.5: LoadAudioAndPlay.java
2  // Load an audio clip and play it.
3  import java.applet.AudioClip;
4  import java.awt.event.ItemListener;
5  import java.awt.event.ItemEvent;
6  import java.awt.event.ActionListener;
7  import java.awt.event.ActionEvent;
8  import java.awt.FlowLayout;
9  import javax.swing.JApplet;
10 import javax.swing.JButton;
11 import javax.swing.JComboBox;
12
13 public class LoadAudioAndPlay extends JApplet
14 {
15    private AudioClip sound1, sound2, currentSound;
16    private JButton playJButton, loopJButton, stopJButton;
17    private JComboBox soundJComboBox;
18
19    // load the image when the applet begins executing
20    public void init()
21    {
22       setLayout( new FlowLayout() );
23
24       String choices[] = { "Welcome", "Hi" };
25       soundJComboBox = new JComboBox( choices ); // create JComboBox
26
27       soundJComboBox.addItemListener(
28
```

`AudioClip` used to represent audio files

```
29          new ItemListener() // anonymous inner class
30          {
31             // stop sound and change to sound to user's selection
32             public void itemStateChanged( ItemEvent e )
33             {
34                currentSound.stop();
35                currentSound = soundJComboBox.getSelectedIndex() == 0 ?
36                   sound1 : sound2;
37             } // end method itemStateChanged
38          } // end anonymous inner class
39       ); // end addItemListener method call
40
41       add( soundJComboBox ); // add JComboBox to applet
42
43       // set up button event handler and buttons
44       ButtonHandler handler = new ButtonHandler();
45
46       // create Play JButton
47       playJButton = new JButton( "Play" );
48       playJButton.addActionListener( handler );
49       add( playJButton );
50
```

```
51       // create Loop JButton
52       loopJButton = new JButton( "Loop" );
53       loopJButton.addActionListener( handler );
54       add( loopJButton );
55
56       // create Stop JButton
57       stopJButton = new JButton( "Stop" );
58       stopJButton.addActionListener( handler );
59       add( stopJButton );
60
61       // load sounds and set currentSound
62       sound1 = getAudioClip( getDocumentBase(), "welcome.wav" );
63       sound2 = getAudioClip( getDocumentBase(), "hi.au" );
64       currentSound = sound1;
65    } // end method init
66
67    // stop the sound when the user switches Web pages
68    public void stop()
69    {
70       currentSound.stop(); // stop AudioClip
71    } // end method stop
72
```



```
73    // private inner class to handle button events
74    private class ButtonHandler implements ActionListener
75    {
76       // process play, loop and stop button events
77       public void actionPerformed( ActionEvent actionEvent )
78       {
79          if ( actionEvent.getSource() == playJButton )
80             currentSound.play(); // play AudioClip once
81          else if ( actionEvent.getSource() == loopJButton )
82             currentSound.loop(); // play AudioClip continuously
83          else if ( actionEvent.getSource() == stopJButton )
84             currentSound.stop(); // stop AudioClip
85       } // end method actionPerformed
86    } // end class ButtonHandler
87 } // end class LoadAudioAndPlay
```

Play clip

Play clip multiple times

End playing of audio clip

Look-and-Feel Observation 21.5

When playing audio clips in an applet or application, provide a mechanism for the user to disable the audio.

21.6 Playing Video and Other Media with Java Media Framework

- **A simple video can concisely and effectively convey a great deal of information**
- **JMF API enables Java programmers to play, edit, stream and capture popular media types**
- **Supported file types include Microsoft Audio/Video Interleave, Macromedia Flash2 movies, MPEG-1 videos and QuickTime movies**

Creating a Simple Media Player

- Interface `Player` used to play video
- Class `Manager` declares utility methods for accessing system resources to play and manipulate media
- Manager method `createRealizedPlayer` obtains a `Player` for a specified media clip
- Loading and playing video
  - `Player` method `getVisualComponent` gets component that displays visual aspect of media file
  - `Player` method `getControlPanelComponent` gets component that provides playback and media controls
  - `Player` method start begins playing media file

```
1  // Fig. 21.6: MediaPanel.java
2  // A JPanel the plays media from a URL
3  import java.awt.BorderLayout;
4  import java.awt.Component;
5  import java.io.IOException;
6  import java.net.URL;
7  import javax.media.CannotRealizeException;
8  import javax.media.Manager;
9  import javax.media.NoPlayerException;
10 import javax.media.Player;
11 import javax.swing.JPanel;
12
13 public class MediaPanel extends JPanel
14 {
15    public MediaPanel( URL mediaURL )
16    {
17       setLayout( new BorderLayout() ); // use a BorderLayout
18
19       // Use lightweight components for Swing compatibility
20       Manager.setHint( Manager.LIGHTWEIGHT_RENDERER, true );
21
22       try
23       {
24          // create a player to play the media specified in the
25          Player mediaPlayer = Manager.createRealizedPlayer( med
26
27          // get the components for the video and the playback controls
28          Component video = mediaPlayer.getVisualComponent();
29          Component controls = mediaPlayer.getControlPanelComponent();
30
```

Use a lightweight renderer

Create `Player` for file specified by `mediaURL`

Retrieve components to display video and controls to pause and run video

```
31          if ( video != null )
32             add( video, BorderLayout.CENTER ); // add video component
33
34          if ( controls != null )
35             add( controls, BorderLayout.SOUTH ); // add controls
36
37          mediaPlayer.start(); // start playing the media clip
38       } // end try
39       catch ( NoPlayerException noPlayerException )
40       {
41          System.err.println( "No media player found" );
42       } // end catch
43       catch ( CannotRealizeException cannotRealizeException )
44       {
45          System.err.println( "Could not realize media player" );
46       } // end catch
47       catch ( IOException iOException )
48       {
49          System.err.println( "Error reading from the source" );
50       } // end catch
51    } // end MediaPanel constructor
52 } // end class MediaPanel
```

```
1  // Fig. 21.7: MediaTest.java
2  // A simple media player
3  import java.io.File;
4  import java.net.MalformedURLException;
5  import java.net.URL;
6  import javax.swing.JFileChooser;
7  import javax.swing.JFrame;
8
9  public class MediaTest
10 {
11    // launch the application
12    public static void main( String args[] )
13    {
14       // create a file chooser
15       JFileChooser fileChooser = new JFileChooser();
16
17       // show open file dialog
18       int result = fileChooser.showOpenDialog( null );
19
20       if ( result == JFileChooser.APPROVE_OPTION ) // user chose a file
21       {
22          URL mediaURL = null;
23
24          try
25          {
26             // get the file as URL
27             mediaURL = fileChooser.getSelectedFile().toURL();
28          } // end try
```

Retrieve file specified by user

```
29          catch ( MalformedURLException malformedURLException )
30          {
31             System.err.println( "Could not create URL for the file" );
32          } // end catch
33
34          if ( mediaURL != null ) // only display if there is a valid URL
35          {
36             JFrame mediaTest = new JFrame( "Media Tester" );
37             mediaTest.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
38
39             MediaPanel mediaPanel = new MediaPanel( mediaURL );
40             mediaTest.add( mediaPanel );
41
42             mediaTest.setSize( 300, 300 );
43             mediaTest.setVisible( true );
44          } // end inner if
45       } // end outer if
46    } // end main
47 } // end class MediaTest
```