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.
21.1 Introduction
21.2 Loading, Displaying and Scaling Images
21.3 Animating a Series of Images
21.4 Image Maps
21.5 Loading and Playing Audio Clips
21.6 Playing Video and Other Media with Java Media Framework
21.7 Wrap-Up
21.8 Internet and Web Resources
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
// Fig. 21.1: LoadImageAndScale.java
// Load an image and display it in its original size and twice its
// original size. Load and display the same image as an ImageIcon.
import java.awt.Graphics;
import java.awt.Image;
import javax.swing.ImageIcon;
import javax.swing.JApplet;

public class LoadImageAndScale extends JApplet {
    private Image image1; // create Image object
    private ImageIcon image2; // create ImageIcon object

    // load image when applet is loaded
    public void init()
    {
        image1 = getImage( getDocumentBase(), "redflowers.png" );
        image2 = new ImageIcon( "yellowflowers.png" );
    } // end method init

    // display image
    public void paint( Graphics g )
    {
        super.paint( g );

        g.drawImage( image1, 0, 0, this ); // draw original image

        // draw image to fit the width and the height less 120 pixels
        g.drawImage( image1, 0, 120, getWidth(), getHeight() - 120, this );
    }
}

Returns location of HTML file as URL object
Method getImage returns Image object for file redflowers.jpg
Create ImageIcon object for file yellowflowers.jpg
Draw image stored in redflowers.jpg
Draw same image scaled to different size
// draw icon using its paintIcon method
image2.paintIcon( this, g, 180, 0 );
} // end method paint
} // 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
// Fig. 21.2: LogoAnimatorJPanel.java
// Animation of a series of images.
import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.Graphics;
import javax.swing.ImageIcon;
import javax.swing.JPanel;
import javax.swing.Timer;

public class LogoAnimatorJPanel extends JPanel {
    private final static String IMAGE_NAME = "deitel"; // base image name
    protected ImageIcon images[]; // array of images
    private final int TOTAL_IMAGES = 30; // number of images
    private int currentImage = 0; // current image index
    private final int ANIMATION_DELAY = 50; // millisecond delay
    private int width; // image width
    private int height; // image height
    private Timer animationTimer; // Timer drives animation

    // constructor initializes LogoAnimatorJPanel by loading images
    public LogoAnimatorJPanel() {
        images = new ImageIcon[TOTAL_IMAGES]; // Will be used to store images to be animated
    }
}
// load 30 images
for ( int count = 0; count < images.length; count++ )
    images[ count ] = new ImageIcon( getClass().getResource("images/" + IMAGE_NAME + count + ".gif") );

// this example assumes all images have the same width and height
width = images[ 0 ].getIconWidth();    // get icon width
height = images[ 0 ].getIconHeight();  // get icon height
}

// display current image
public void paintComponent( Graphics g )
{
    super.paintComponent( g );   // call superclass paintComponent

    images[ currentImage ].paintIcon( this, g, 0, 0 );

    // set next image to be drawn only if timer is running
    if ( animationTimer.isRunning() )
        currentImage = ( currentImage + 1 ) % TOTAL.Images;
}

Create and store ImageIcon for each image
Set next image only if Timer is still running
// start animation, or restart if window is redisplayed
public void startAnimation()
{
    if ( animationTimer == null )
    {
        currentImage = 0; // display first image
        // create timer
        animationTimer =
            new Timer( ANIMATION_DELAY, new TimerHandler() );
        animationTimer.start(); // start timer
    } // end if
    else // animationTimer already exists, restart animation
    {
        if ( ! animationTimer.isRunning() )
            animationTimer.restart();
    } // end else
} // end method startAnimation

// stop animation timer
public void stopAnimation()
{
    animationTimer.stop();
} // end method stopAnimation
// return minimum size of animation
public Dimension getMinimumSize()
{
    return getPreferredSize();
} // end method getMinimumSize

// return preferred size of animation
public Dimension getPreferredSize()
{
    return new Dimension(width, height);
} // end method getPreferredSize

// inner class to handle action events from Timer
private class TimerHandler implements ActionListener
{
    // respond to Timer's event
    public void actionPerformed(ActionEvent actionEvent)
    {
        repaint(); // repaint animator
    } // end method actionPerformed
} // end class TimerHandler

} // end class LogoAnimatorJPanel
// Fig. 21.3: LogoAnimator.java
// Animation of a series of images.
import javax.swing.JFrame;

public class LogoAnimator {
    // execute animation in a JFrame
    public static void main( String args[] ) {
        LogoAnimatorJPanel animation = new LogoAnimatorJPanel();

        JFrame window = new JFrame( "Animator test" ); // set up window
        window.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
        window.add( animation ); // add panel to frame

        window.pack(); // make window just large enough for its GUI
        window.setVisible( true ); // display window

        animation.startAnimation(); // begin animation
    } // end main
} // 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
public class ImageMap extends JApplet
{
    private ImageIcon mapImage;

    private static final String captions[] = { "Common Programming Error",
        "Good Programming Practice", "Graphical User Interface Tip",
        "Performance Tip", "Portability Tip",
        "Software Engineering Observation", "Error-Prevention Tip" };
new MouseAdapter() // anonymous inner class
{
    // indicate when mouse pointer exits applet area
    public void mouseExited(MouseEvent event)
    {
        showStatus("Pointer outside applet");
    } // end method mouseExited
} // end anonymous inner class
); // end call to addMouseListener

addMouseMotionListener(
    new MouseMotionAdapter() // anonymous inner class
    {
        // determine icon over which mouse appears
        public void mouseMoved(MouseEvent event)
        {
            showStatus(translateLocation(event.getX(), event.getY()));
        } // end method mouseMoved
    } // end anonymous inner class
); // end call to addMouseMotionListener

mapImage = new ImageIcon("icons.png"); // get image
```java
public void paint(Graphics g) {
    super.paint(g);
    mapImage.paintIcon(this, g, 0, 0);
} // end method paint

// return tip caption based on mouse coordinates
public String translateLocation(int x, int y) {
    // if coordinates outside image, return immediately
    if (x >= mapImage.getIconWidth() || y >= mapImage.getIconHeight())
        return ""

    // determine icon number (0 - 6)
    double iconWidth = (double) mapImage.getIconWidth() / 7.0;
    int iconNumber = (int)(x / iconWidth);

    return captions[iconNumber]; // return appropriate icon caption
} // end method translateLocation
```
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
// Fig. 21.5: LoadAudioAndPlay.java
// Load an audio clip and play it.
import java.applet.AudioClip;
import java.awt.event.ItemListener;
import java.awt.event.ItemEvent;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.FlowLayout;
import javax.swing.JApplet;
import javax.swing.JButton;
import javax.swing.JComboBox;

public class LoadAudioAndPlay extends JApplet {
    private AudioClip sound1, sound2, currentSound;
    private JButton playJButton, loopJButton, stopJButton;
    private JComboBox soundJComboBox;

    // load the image when the applet begins executing
    public void init() {
        setLayout( new FlowLayout() );
        String choices[] = { "Welcome", "Hi" };
        soundJComboBox = new JComboBox( choices ); // create JComboBox
        soundJComboBox.addItemListener(
            new AudioClip()
        );
new ItemListener() // anonymous inner class
{
// stop sound and change to sound to user's selection
public void itemStateChanged(ItemEvent e)
{
    currentSound.stop();
    currentSound = soundJComboBox.getSelectedItem() == 0 ?
        sound1 : sound2;
} // end method itemStateChanged
} // end anonymous inner class
}; // end addItemListener method call

add(soundJComboBox); // add JComboBox to applet

// set up button event handler and buttons
ButtonHandler handler = new ButtonHandler();

// create Play JButton
playJButton = new JButton("Play");
playJButton.addActionListener(handler);
add(playJButton);
// create Loop JButton
loopButton = new JButton( "Loop" );
loopButton.addActionListener( handler );
add( loopButton );

// create Stop JButton
stopButton = new JButton( "Stop" );
stopButton.addActionListener( handler );
add( stopButton );

// load sounds and set currentSound
sound1 = getAudioClip( getDocumentBase(), "welcome.wav" );
sound2 = getAudioClip( getDocumentBase(), "hi.au" );
currentSound = sound1;
}

// stop the sound when the user switches Web pages
public void stop() {
    currentSound.stop(); // stop AudioClip
}

Load audio clips
private class ButtonHandler implements ActionListener {
    // process play, loop and stop button events
    public void actionPerformed(ActionEvent actionEvent) {
        if (actionEvent.getSource() == playJButton)
            currentSound.play(); // play AudioClip once
        else if (actionEvent.getSource() == loopJButton)
            currentSound.loop(); // play AudioClip continuously
        else if (actionEvent.getSource() == stopJButton)
            currentSound.stop(); // stop AudioClip
    } // end method actionPerformed
} // end class ButtonHandler

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
public class MediaPanel extends JPanel{
    public MediaPanel( URL mediaURL ) {
        try {
            // create a player to play the media specified in the
            // URL
            Player mediaPlayer = Manager.createRealizedPlayer( mediaURL);

            // get the components for the video and the playback controls
            Component video = mediaPlayer.getVisualComponent();
            Component controls = mediaPlayer.getControlPanelComponent();
        }
        catch( CannotRealizeException e ) {
            // handle error when cannot realize player
            System.err.println( e.getMessage() );
        } catch( NoPlayerException e ) { // handle error when no player
            System.err.println( e.getMessage() );
        } catch( IOException e ) { // handle error when cannot read URL
            System.err.println( e.getMessage() );
        } finally {
            setLayout( new BorderLayout() ); // use a BorderLayout
            // Use lightweight components for Swing compatibility
            Manager.setHint( Manager.LIGHTWEIGHT_RENDERER, true );
        }
    }
}
if ( video != null )
    add( video, BorderLayout.CENTER ); // add video component

if ( controls != null )
    add( controls, BorderLayout.SOUTH ); // add controls

    mediaPlayer.start(); // start playing the media clip
} // end try
catch ( NoPlayerException noPlayerException )
{
    System.err.println( "No media player found" );
} // end catch
catch ( CannotRealizeException cannotRealizeException )
{
    System.err.println( "Could not realize media player" );
} // end catch
catch ( IOException iOException )
{
    System.err.println( "Error reading from the source" );
} // end catch
} // end MediaPanel constructor
} // end class MediaPanel
// Fig. 21.7: MediaTest.java
// A simple media player
import java.io.File;
import java.net.MalformedURLException;
import java.net.URL;
import javax.swing.JFileChooser;
import javax.swing.JFrame;

public class MediaTest
{
    // launch the application
    public static void main( String args[] )
    {
        // create a file chooser
        JFileChooser fileChooser = new JFileChooser();

        // show open file dialog
        int result = fileChooser.showOpenDialog( null );

        if ( result == JFileChooser.APPROVE_OPTION ) // user chose a file
        {
            URL mediaURL = null;

            try
            {
                // get the file as URL
                mediaURL = fileChooser.getSelectedFile().toURL();
            } // end try
        } // end if
    } // end main
} // end MediaTest

Retrieve file specified by user
catch ( MalformedURLException malformedURLException )
{
    System.err.println( "Could not create URL for the file" );
} // end catch

if ( mediaURL != null ) // only display if there is a valid URL
{
    JFrame mediaTest = new JFrame( "Media Tester" );
    mediaTest.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );

    MediaPanel mediaPanel = new MediaPanel( mediaURL );
    mediaTest.add( mediaPanel );

    mediaTest.setSize( 300, 300 );
    mediaTest.setVisible( true );
} // end inner if
} // end outer if
} // end main
} // end class MediaTest