TOPIC:

INTRODUCING SWING
EXPLORING SWING

Swing – Key Features

•Swing eliminates a number of the limitations inherent in the AWT, Swing does not replace it. Instead, Swing is built on the foundation of the AWT.

•Two key features: lightweight components and a pluggable look and feel. Together they provide an elegant, yet easy-to-use solution to the problems of the AWT.

•A Swing GUI consists of two key items: components and containers. However, this distinction is mostly conceptual because all containers are also components.

refer page 1451-1456 of pdf,10 edition

Components and Containners

- Swing components are derived from the JComponent class.
- JComponent inherits the AWT classes Container and Component. Thus, a Swing component is built on and compatible with an AWT component.
- All of Swing's components are represented by classes defined within the package javax.swing.
- For example, the class for a label is JLabel; the class for a push button is JButton; and the class for a scroll bar is JScrollBar.
- Swing defines two types of containers. The first are top-level containers: JFrame, JApplet, JWindow, and JDialog. These containers do not inherit JComponent.
- The second type of containers supported by Swing are lightweight containers. Lightweight containers do inherit JComponent. An example of a lightweight container is JPanel, which is a general-purpose container.

Swing Application- An Example

```
// A simple Swing application.

import javax.swing.*;

class SwingDemo {

  SwingDemo() {

    // Create a new JFrame container.
    JFrame jfrm = new JFrame("A Simple Swing Application");

    // Give the frame an initial size.
    jfrm.setSize(275, 100);

    // Terminate the program when the user closes the application.
    jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    // Create a text-based label.
    JLabel jlab = new JLabel(" Swing means powerful GUIs.");

    // Add the label to the content pane.
    jfrm.add(jlab);

    // Display the frame.
    jfrm.setVisible(true);
  }

  public static void main(String args[]) {
    // Create the frame on the event dispatching thread.
    SwingUtilities.invokeLater(new Runnable() {
      public void run() {
        new SwingDemo();
      }
    });
  }
}
```

```
javac SwingDemo.java
```

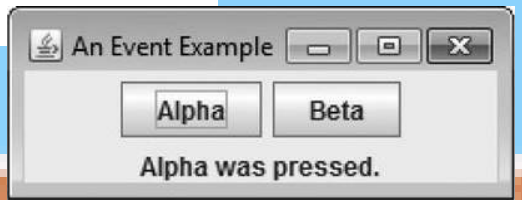
```
java SwingDemo
```

Event Handling

- The preceding example showed the basic form of a Swing program, but it left out one important part: event handling.
- Because JLabel does not take input from the user, it does not generate events, so no event handling was needed.
- However, the other Swing components do respond to user input and the events generated by those interactions need to be handled.
- Events can also be generated in ways not directly related to user input. For example, an event is generated when a timer goes off.

Event Handling - An Example

```
// Handle an event in a Swing program.

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

class EventDemo {

  JLabel jlab;

  EventDemo() {

    // Create a new JFrame container.
    JFrame jfrm = new JFrame("An Event Example");

    // Specify FlowLayout for the layout manager.
    jfrm.setLayout(new FlowLayout());

    // Give the frame an initial size.
    jfrm.setSize(220, 90);

    // Terminate the program when the user closes the application.
    jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    // Make two buttons.
    JButton jbtnAlpha = new JButton("Alpha");
    JButton jbtnBeta = new JButton("Beta");

    // Add action listener for Alpha.
    jbtnAlpha.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ae) {
        jlab.setText("Alpha was pressed.");
      }
    });

    // Add action listener for Beta.
    jbtnBeta.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ae) {
        jlab.setText("Beta was pressed.");
      }
    });

    // Add the buttons to the content pane.
    jfrm.add(jbtnAlpha);
    jfrm.add(jbtnBeta);

    // Create a text-based label.
    jlab = new JLabel("Press a button.");

    // Add the label to the content pane.
    jfrm.add(jlab);

    // Display the frame.
    jfrm.setVisible(true);
  }

  public static void main(String args[]) {
    // Create the frame on the event dispatching thread.
    SwingUtilities.invokeLater(new Runnable() {
      public void run() {
        new EventDemo();
      }
    });
  }
}
```



Painting in Swing

- Swing also lets you write directly into the display area of a frame, panel, or one of Swing's other components, such as JLabel.
- To write output directly to the surface of a component, you will use one or more drawing methods defined by the AWT, such as drawLine( ) or drawRect( ).
- The AWT class Component defines a method called paint( ) that is used to draw output directly to the surface of a component.
- paintComponent( ), paintBorder( ), and paintChildren( ) methods paint the indicated portion of a component and divide the painting process into its three distinct, logical actions.

Painting - An Example

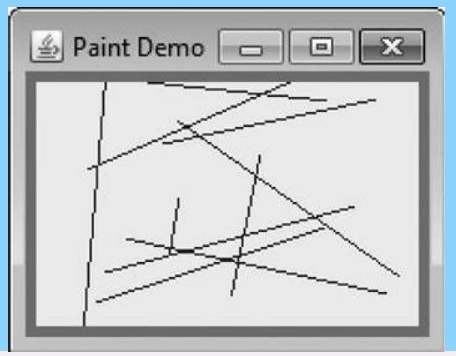
```
// Paint lines to a panel.

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

// This class extends JPanel. It overrides
// the paintComponent() method so that random
// lines are plotted in the panel.
class PaintPanel extends JPanel {
  Insets ins; // holds the panel's insets

  Random rand; // used to generate random numbers

  // Construct a panel.
  PaintPanel() {

    // Put a border around the panel.
    setBorder(
      BorderFactory.createLineBorder(Color.RED, 5));

    rand = new Random();
  }

      // Override the paintComponent() method.
      protected void paintComponent(Graphics g) {
        // Always call the superclass method first.
        super.paintComponent(g);

        int x, y, x2, y2;
```

Painting - An Example

```
    // Get the height and width of the component.
    int height = getHeight();
    int width = getWidth();

    // Get the insets.
    ins = getInsets();

    // Draw ten lines whose endpoints are randomly generated.
    for(int i=0; i < 10; i++) {
      // Obtain random coordinates that define
      // the endpoints of each line.
      x = rand.nextInt(width-ins.left);
      y = rand.nextInt(height-ins.bottom);
      x2 = rand.nextInt(width-ins.left);
      y2 = rand.nextInt(height-ins.bottom);

      // Draw the line.
      g.drawLine(x, y, x2, y2);
    }
  }
}

// Demonstrate painting directly onto a panel.
class PaintDemo {

  JLabel jlab;
  PaintPanel pp;

  PaintDemo() {

    // Create a new JFrame container.
    JFrame jfrm = new JFrame("Paint Demo");

    // Give the frame an initial size.
    jfrm.setSize(200, 150);

    // Terminate the program when the user closes the application.
    jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

Painting - An Example

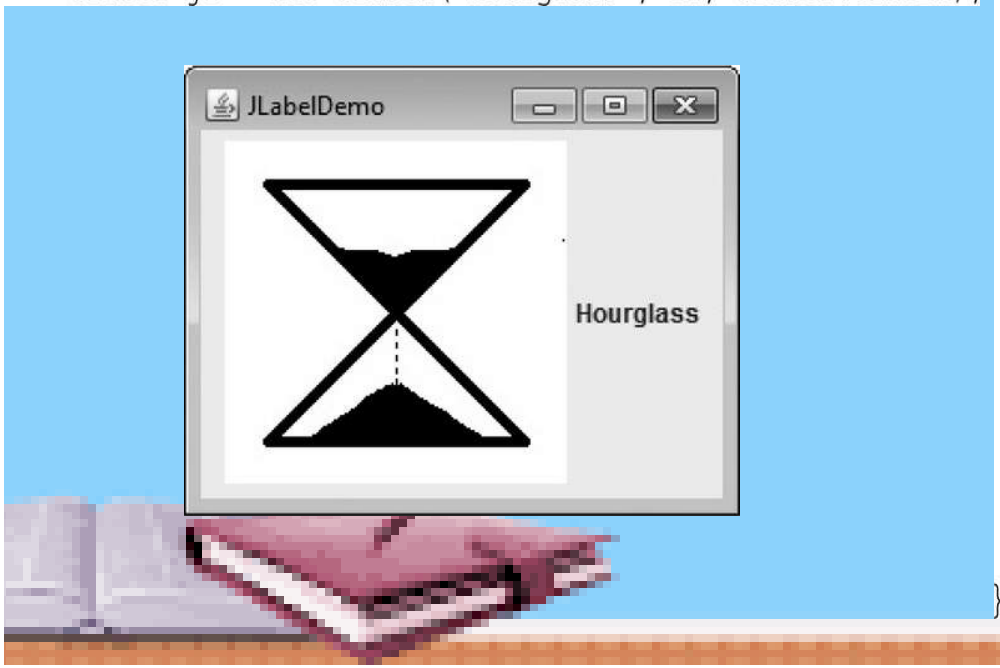
```
        // Create the panel that will be painted.
        pp = new PaintPanel();

        // Add the panel to the content pane. Because the default
        // border layout is used, the panel will automatically be
        // sized to fit the center region.
        jfrm.add(pp);

    // Display the frame.
    jfrm.setVisible(true);
  }

  public static void main(String args[]) {
    // Create the frame on the event dispatching thread.
    SwingUtilities.invokeLater(new Runnable() {
      public void run() {
        new PaintDemo();
      }
    });
  }
}
```



JLabel and ImageIcon

- JLabel can be used to display text and/or an icon. It is a passive component in that it does not respond to user input.
- JLabel defines several constructors.
- The easiest way to obtain an icon is to use the ImageIcon class.
- ImageIcon implements Icon and encapsulates an image. Thus, an object of type ImageIcon can be passed as an argument to the Icon parameter of JLabel's constructor.
- There are several ways to provide the image, including reading it from a file or downloading it from a URL.

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

public class JLabelDemo {

   public JLabelDemo() {

    // Set up the JFrame.
    JFrame jfrm = new JFrame("JLabelDemo");
    jfrm.setLayout(new FlowLayout());
    jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    jfrm.setSize(260, 210);

    // Create an icon.
    ImageIcon ii = new ImageIcon("hourglass.png");

    // Create a label.
    JLabel jl = new JLabel("Hourglass", ii, JLabel.CENTER);

    // Add the label to the content pane.
    jfrm.add(jl);

    // Display the frame.
    jfrm.setVisible(true);
  }

  public static void main(String[] args) {
    // Create the frame on the event dispatching thread.

    SwingUtilities.invokeLater(
      new Runnable() {
        public void run() {
          new JLabelDemo();
        }
      }
    );

  }
}
```

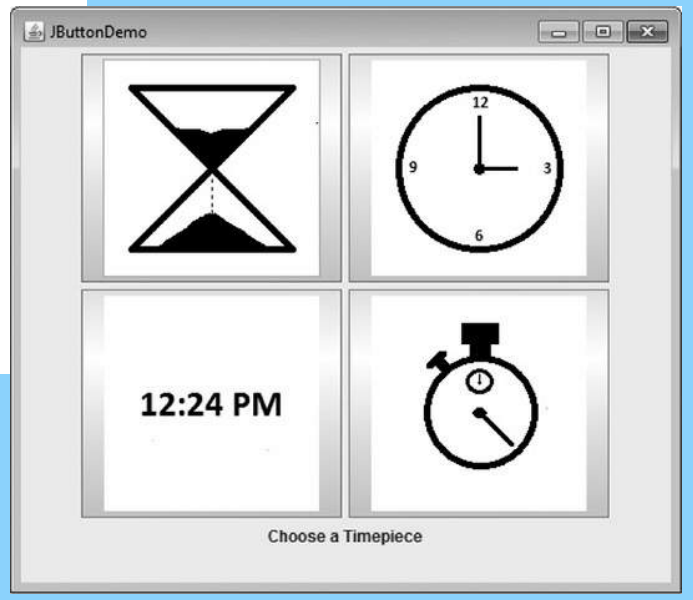


Swing Buttons

- Swing defines four types of buttons: JButton, JToggleButton, JCheckBox, and JRadioButton.
- All are subclasses of the AbstractButton class, which extends JComponent. Thus, all buttons share a set of common traits.
- AbstractButton contains many methods that allow you to control the behavior of buttons.
- For example, you can define different icons that are displayed for the button when it is disabled, pressed, or selected. Another icon can be used as a rollover icon, which is displayed when the mouse is positioned over a button.

```
// Demonstrate an icon-based JButton.
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class JButtonDemo implements ActionListener {
  JLabel jlab;

  public JButtonDemo() {

    // Set up the JFrame.
    JFrame jfrm = new JFrame("JButtonDemo");
    jfrm.setLayout(new FlowLayout());
    jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    jfrm.setSize(500, 450);

    // Add buttons to content pane.
    ImageIcon hourglass = new ImageIcon("hourglass.png");
    JButton jb = new JButton(hourglass);
    jb.setActionCommand("Hourglass");
    jb.addActionListener(this);
    jfrm.add(jb);

    ImageIcon analog = new ImageIcon("analog.png");
    jb = new JButton(analog);
    jb.setActionCommand("Analog Clock");
    jb.addActionListener(this);
    jfrm.add(jb);

    ImageIcon digital = new ImageIcon("digital.png");
    jb = new JButton(digital);
    jb.setActionCommand("Digital Clock");
    jb.addActionListener(this);
    jfrm.add(jb);

    ImageIcon stopwatch = new ImageIcon("stopwatch.png");
    jb = new JButton(stopwatch);
    jb.setActionCommand("Stopwatch");
    jb.addActionListener(this);
    jfrm.add(jb);
```

```
    // Create and add the label to content pane.
    jlab = new JLabel("Choose a Timepiece");
    jfrm.add(jlab);

    // Display the frame.
    jfrm.setVisible(true);
  }

  // Handle button events.
  public void actionPerformed(ActionEvent ae) {
    jlab.setText("You selected " + ae.getActionCommand());
  }

  public static void main(String[] args) {
    // Create the frame on the event dispatching thread.

    SwingUtilities.invokeLater(
      new Runnable() {
        public void run() {
          new JButtonDemo();
        }
      }
    );

  }
}
```



References/Resources:

•Balaguruswamy, E. (2014). Programming with JAVA: A Primer. 5th edition. India: McGraw Hill Education

•Horstmann, C. S. (2017). Core Java - Vol. I – Fundamentals (Vol. 10). Pearson Education

•Schildt, H. (2018). Java: The Complete Reference. 10th edition. McGraw-Hill Education.

NOTE: Please go through the reference book for details on the above topic and feel free to mail your doubts or discuss anything.