Module-5: Swings, Exploring Swing and Case Studies

Introducing Swing

Swing is a set of classes that provides more powerful and flexible GUI components than the earlier Abstract Window Toolkit (AWT). Swing provides buttons, lists, menus, and text areas that are platform-independent.

The Origins of Swing

Swing was developed to address the shortcomings of the AWT. AWT components rely on native code counterparts (peers), which leads to platform-specific limitations and the "lowest common denominator" feature set. Swing was introduced with the Java Foundation Classes (JFC) to provide a better alternative.

Swing Is Built on the AWT

Swing is built on top of the AWT, not a complete replacement. It reuses the core infrastructure of the AWT, including its event model and basic features, but replaces the AWT's component set with its own.

Two Key Swing Features

1. **Lightweight Components:** Swing components are "lightweight," meaning they are written entirely in Java and do not rely on native peers. This makes them more efficient, flexible, and consistent across different platforms.
2. **Pluggable Look and Feel (PLAF):** Swing separates the logic of a component from its appearance. This allows the look and feel of a Swing application to be changed dynamically at runtime. Several look and feels are available, such as Metal (the Java default), Windows, and Motif.

The MVC Connection

Swing components use a modified version of the Model-View-Controller (MVC) architecture.

- **Model:** Stores the state of the component.
- **View:** Determines how the component is displayed.
- **Controller:** Dictates how the component reacts to user input.

Swing combines the view and controller into a single logical entity called the **UI delegate**. This approach is often called Model-Delegate or Separable Model architecture.

Components and Containers

- **Component:** An object that has a graphical representation. `JComponent` is the top-level class for most Swing components.
- **Container:** A component that can hold other components. The `Container` class (from AWT) is the superclass for Swing containers.

Swing Containers:

1. **Top-Level Containers:** Do not inherit from `JComponent`. Every Swing GUI must have a top-level container.
    - `JFrame`: A typical window.
    - `JApplet`: An applet container.
    - `JDialog`: For creating dialog boxes.
2. **General-Purpose Containers:** Inherit from `JComponent`.
    - `JPanel`: A general-purpose container for grouping components.

The Swing Packages

Swing's classes are located in the `javax.swing` package and its subpackages, such as `javax.swing.event`, `javax.swing.tree`, and `javax.swing.table`.

A Simple Swing Application

All Swing applications are created on the **event dispatching thread**. The main thread should use `SwingUtilities.invokeLater()` to queue the GUI creation code for execution on this thread.

Example: A basic Swing application using `JFrame` and `JLabel`.

```
// 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();
            }
        });
    }
}
```

7. Exploring Swing

This section covers several of the most commonly used Swing components.

`JLabel` and `ImageIcon`

`JLabel` is a component for placing text in a container. It can also display an icon.

Constructors:

- `JLabel(Icon icon)`
- `JLabel(String str)`
- `JLabel(String str, Icon icon, int horizontalAlignment)`

Example:

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


class JLabelDemo {
    public JLabelDemo() {
        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) {
        SwingUtilities.invokeLater(() -> new JLabelDemo());
    }
}
```

JTextField

The JTextField class implements a single-line text-entry area.

Constructors:

- JTextField(int numChars)
- JTextField(String str)
- JTextField(String str, int numChars)

Example:

```
import java.awt.*;

import java.awt.event.*;

import javax.swing.*;


class JTextFieldDemo {

   public JTextFieldDemo() {

      JFrame jfrm = new JFrame("JTextFieldDemo");

      jfrm.setLayout(new FlowLayout());

      jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      jfrm.setSize(260, 120);


      JTextField jtf = new JTextField(15);

      jfrm.add(jtf);


      JLabel jlab = new JLabel();

      jfrm.add(jlab);
```

```
        // Add an action listener for the text field.
        jtf.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent ae) {
                // Get text from the text field.
                jlab.setText(jtf.getText());
            }
        });

        jfrm.setVisible(true);
    }
    // ... main method
}
```

The Swing Buttons

Swing provides four types of buttons:

- JButton: A standard push button.
- JToggleButton: A toggled button with on/off states.
- JCheckBox: A check box.
- JRadioButton: A radio button (mutually exclusive).

JButton Example:

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

```
class JButtonDemo implements ActionListener {
  JLabel jlab;

  public JButtonDemo() {
    // ... frame setup ...
    jfrm.setSize(220, 90);

    JButton jbtnUp = new JButton("Up");
    JButton jbtnDown = new JButton("Down");

    // Add action listeners.
    jbtnUp.addActionListener(this);
    jbtnDown.addActionListener(this);

    jfrm.add(jbtnUp);
    jfrm.add(jbtnDown);

    jlab = new JLabel("Press a button.");
    jfrm.add(jlab);

    jfrm.setVisible(true);
```

```
    }


    // Handle button events.
    public void actionPerformed(ActionEvent ae) {
        if(ae.getActionCommand().equals("Up")) {
            jlab.setText("You pressed Up.");
        } else {
            jlab.setText("You pressed Down.");
        }
    }
    // ... main method
}
```

JTabbedPane

A JTabbedPane manages a set of components by linking them with tabs. Clicking a tab brings its associated component to the front.

Example:

```
// Demonstrate JTabbedPane.
import javax.swing.*;
public class JTabbedPaneDemo {
    public JTabbedPaneDemo() {
        JFrame jfrm = new JFrame("JTabbedPaneDemo");
        jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
        jfrm.setSize(400, 200);


        JTabbedPane jtp = new JTabbedPane();
        jtp.addTab("Cities", new CitiesPanel());
        jtp.addTab("Colors", new ColorsPanel());
        jtp.addTab("Flavors", new FlavorsPanel());
        jfrm.add(jtp);


        jfrm.setVisible(true);
    }
    // ... main method and panel classes (e.g., CitiesPanel extends JPanel)
}
```

JScrollPane

A JScrollPane provides a scrollable view of a component.

Example:

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


class JScrollPaneDemo {
    public JScrollPaneDemo() {
        JFrame jfrm = new JFrame("JScrollPaneDemo");
```

```
        jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jfrm.setSize(400, 400);

        // Add 400 buttons to a panel.
        JPanel jp = new JPanel();
        jp.setLayout(new GridLayout(20, 20));
        for(int i = 0; i < 400; i++) {
            jp.add(new JButton("Button " + i));
        }

        // Create the scroll pane.
        JScrollPane jsp = new JScrollPane(jp);

        // Add the scroll pane to the content pane.
        jfrm.add(jsp, BorderLayout.CENTER);

        jfrm.setVisible(true);
    }
   // ... main method
}
```

JList

A JList provides a scrollable list of items from which the user can select one or more entries.

Example:

```
import javax.swing.*;

import javax.swing.event.*;

import java.awt.*;


class JListDemo implements ListSelectionListener {

   JList<String> jlst;

   JLabel jlab;

   String[] cities = { "New York", "Chicago", "Houston", "Denver",

                       "Los Angeles", "Seattle", "London", "Paris" };


   public JListDemo() {

      // ... frame setup ...

      jlst = new JList<>(cities);


      // Set list to single-selection mode.

      jlst.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);


      JScrollPane jscrlp = new JScrollPane(jlst);

      jscrlp.setPreferredSize(new Dimension(120, 90));
```

```
        jlab = new JLabel("Please choose a city");

        // Add selection listener for the list.
        jlst.addListSelectionListener(this);

        jfrm.add(jscrlp);
        jfrm.add(jlab);
        jfrm.setVisible(true);
    }

    // Handle list selection events.
    public void valueChanged(ListSelectionEvent le) {
        int idx = jlst.getSelectedIndex();
        if (idx != -1)
            jlab.setText("Current selection: " + cities[idx]);
        else
            jlab.setText("Please choose a city");
    }
    // ... main method
}
```

JComboBox

A JComboBox is similar to a JList, but it displays a drop-down list.

Example:

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

class JComboBoxDemo {
   String[] timepieces = { "Hourglass", "Analog", "Digital", "Stopwatch" };
   public JComboBoxDemo() {
      // ... frame setup ...
      JComboBox<String> jcb = new JComboBox<>(timepieces);
      jfrm.add(jcb);

      JLabel jlab = new JLabel(new ImageIcon("hourglass.png"));
      jfrm.add(jlab);

      jcb.addActionListener(new ActionListener() {
         public void actionPerformed(ActionEvent ae) {
            String s = (String) jcb.getSelectedItem();
            jlab.setIcon(new ImageIcon(s.toLowerCase() + ".png"));
         }
      });
```

```
        jfrm.setVisible(true);
    }
    // ... main method
}
```

Trees

A JTree is a component that displays hierarchical data. Each node in the tree is a DefaultMutableTreeNode.

Example:

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

class JTreeDemo {
    public JTreeDemo() {
        // ... frame setup ...
        DefaultMutableTreeNode top = new DefaultMutableTreeNode("Options");

        DefaultMutableTreeNode a = new DefaultMutableTreeNode("A");
        top.add(a);
        a.add(new DefaultMutableTreeNode("A1"));
        a.add(new DefaultMutableTreeNode("A2"));
```

```
        DefaultMutableTreeNode b = new DefaultMutableTreeNode("B");
        top.add(b);
        b.add(new DefaultMutableTreeNode("B1"));
        b.add(new DefaultMutableTreeNode("B2"));

        JTree tree = new JTree(top);

        JScrollPane jsp = new JScrollPane(tree);
        jfrm.add(jsp);

        jfrm.setVisible(true);
    }
    // ... main method
}
```

JTable

A JTable displays data in a two-dimensional table.

Example:

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

class JTableDemo {
```

```
    // Column headers.
    String[] colHeads = { "Name", "Extension", "ID#" };
    // Data.
    Object[][] data = {
        { "Gail", "4567", "865" },
        { "Ken", "7566", "555" },
        { "Viviane", "5634", "587" },
        // ... more data
    };

    public JTableDemo() {
        JFrame jfrm = new JFrame("JTableDemo");
        jfrm.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        jfrm.setSize(300, 200);

        JTable table = new JTable(data, colHeads);

        JScrollPane jsp = new JScrollPane(table);

        jfrm.add(jsp);
        jfrm.setVisible(true);
    }
```

```
    // ... main method
}
```