UNIT-4: SWINGS

Swing is built on top of AWT and is entirely written in Java, using AWT's lightweight component support. In particular, unlike AWT, t he architecture of Swing components makes it easy to customize both their appearance and behavior. Components from AWT and Swing can be mixed, allowing you to add Swing support to existing AWT-based programs. For example, swing components such as JSlider, JButton and JCheckbox could be used in the same program with standard AWT labels, textfields and scrollbars.

1. Three parts

Component set (subclasses of JComponent) Support classes, Interfaces

Swing Components and Containers

Swing components are basic building blocks of an application. Swing toolkit has a wide range of various widgets. Buttons, check boxes,sliders, list boxes etc. Everything a programmer needs for his job. In this section of the tutorial, we will describe several useful components.

JLabel Component

JLabel is a simple component for displaying text, images or both. It does not react to input events.

JCheckBox

JCheckBox is a widget that has two states. On and Off. It is a box with a label JSlider

JSlider is a component that lets the user graphically select a value by sliding a knob within a bounded interval

JComboBox

Combobox is a component that combines a button or editable field and a drop-down list. The user can select a value from the drop-down list, which appears at the user's request.

JProgressBar

A progress bar is a widget that is used, when we process lengthy tasks. It is animated so that the user knows, that our task is progressing

JToggleButton

JToggleButton is a button that has two states. Pressed and not pressed. You toggle between these two states by clicking on it

Containers

Swing contains a number of components that provides for grouping other components together.

In AWT, such components extended java.awt.Container and included Panel, Window, Frame, and Dialog.

1.1 A Simple Container

JPanel is Swing's version of the AWT class Panel and uses the same default layout, FlowLayout. JPanel is descended directly from JComponent.

JFrame is Swing's version of Frame and is descended directly from that class. The components added to the frame are referred to as its contents; these are managed by the contentPane. To add a component to a JFrame, we must use its contentPane instead.

JInternalFrame is confined to a visible area of a container it is placed in. It can be iconified , maximized and layered.

JWindow is Swing's version of Window and is descended directly from that class. Like Window, it uses BorderLayout by default.

JDialog is Swing's version of Dialog and is descended directly from that class. Like Dialog, it uses BorderLayout by default. Like JFrame and JWindow,

JDialog contains a rootPane hierarchy including a contentPane, and it allows layered and glass panes. All dialogs are modal, which means the current

thread is blocked until user interaction with it has been completed. JDialog class is intended as the basis for creating custom dialogs; however, some

of the most common dialogs are provided through static methods in the class JOptionPane.

Jlabel and ImageIcon

Syntax : public class **JLabel**

extends JComponent

implements SwingConstants, Accessible

- It is a display area for a short text string or an image, or both.
- A label does not react to input events. As a result, it cannot get the keyboard focus.
- A label can display a keyboard alternative as a convenience for a nearby component that has a keyboard alternative but can't display it.
- A JLabel object can display either text, an image, or both.
- By default, labels are vertically centered in their display area.
- Text-only labels are leading edge aligned, by default; image-only labels are horizontally centered, by default.

• Can use the setIconTextGap method to specify how many pixels should appear between the text and the image. The default is 4 pixels.

ImageIcon

Syntax :

public **ImageIcon**(String filename)

{

this(filename, filename);

}

Creates an ImageIcon from the specified file. The image will be preloaded by using MediaTracker to monitor the loading state of the image.

The specified String can be a file name or a file path. When specifying a path, use the Internet-standard forward-slash ("/") as a separator. (The string is converted to an URL, so the forward-slash works on all systems.)

For example, specify:

new ImageIcon("images/myImage.gif")

The description is initialized to the filename string.

Example of JLabel with ImageIcon :

import java.awt.FlowLayout; **import** java.awt.HeadlessException;

import javax.swing.Icon;

```
import javax.swing.ImageIcon; import javax.swing.JFrame; import
javax.swing.JLabel;

public class Main extends JFrame {

public Main() throws HeadlessException { setSize(300, 300);

setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); setLayout(new
FlowLayout(FlowLayout.LEFT));

Icon icon = new ImageIcon("a.png");

JLabel label1 = new JLabel("Full Name :", icon, JLabel.LEFT);

JLabel label2 = new JLabel("Address :", JLabel.LEFT); label2.setIcon(new
ImageIcon("b.png"));

getContentPane().add(label1); getContentPane().add(label2);

}

public static void main(String[] args) { new Main().setVisible(true);

}

}
```

JTextField

- JTextField is a lightweight component that allows the editing of a single line of text.
- JTextField is intended to be source-compatible with java.awt.TextField where it is reasonable to do so. This component has capabilities not found in the java.awt.TextField class.
- JTextField has a method to establish the string used as the command string for the action event that gets fired.
- The java.awt.TextField used the text of the field as the command string for the ActionEvent.
- JTextField will use the command string set with the setActionCommand method if not null, otherwise it will use the text of the field as a compatibility with java.awt.TextField.

2. Swing Package

Syntax :

public class **JButton** extends AbstractButton implements Accessible

An implementation of a "push" button. Buttons can be configured, and to some degree controlled, by Actions. Using an Action with a button has many benefits beyond directly configuring a button.

```
package com.ack.gui.swing.simple;

import java.awt.*;

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

public class SimpleSwingButtons extends JFrame {

public static void main( String[] argv ) {

SimpleSwingButtons myExample = new SimpleSwingButtons( "Simple Swing Buttons" );

}

public SimpleSwingButtons( String title ) {

super( title );

setSize( 150, 150 );

add WindowListener( new WindowAdapter() { public void windowClosing( WindowEvent we ) { dispose();

System.exit( 0 );

}

} );

init();

setVisible( true );

}
```

```
private void init() {
JPanel my_panel = new JPanel();
my_panel.setLayout( new GridLayout( 3, 3 ) ); for( int i = 1; i < 10; i++ ) {
ImageIcon icon = new ImageIcon( i + ".gif" ); JButton jb = new JButton( icon );
jb.setToolTipText( i + ".gif" );
my_panel.add( jb );
}
getContentPane().add( my_panel );
my_panel.setBorder( BorderFactory.createEtchedBorder() );
}
}
```

JTabbedpane

Syntax : public class JTabbedPane

extends JComponent

implements Serializable, Accessible, SwingConstants

- A component that lets the user switch between a group of components by clicking on a tab with a given title and/or icon.
- Tabs/components are added to a TabbedPane object by using the addTab and insertTab methods.
- A tab is represented by an index corresponding to the position it was added in, where the first tab has an index equal to 0 and the last tab has an index equal to the tab count minus
- The TabbedPane uses a Single SelectionModel to represent the set of tab indices and the currently selected index. If the tab count is greater than 0, then there will always be a selected index, which by default will be initialized to the first tab. If the tab count is 0, then the selected index will be -1.

JScrollPane

Syntax : public class JScrollPane

extends JComponent

implements ScrollPaneConstants, Accessible

- Provides a scrollable view of a lightweight component.
- A JScrollPane manages a viewport, optional vertical and horizontal scroll bars, and optional row and column heading viewports.
- The JViewport provides a window, or "viewport" onto a data source -- for example, a text file. That data source is the "scrollable client" (aka data model) displayed by the JViewport view.
- A JScrollPane basically consists of JScrollBars, a JViewport, and the wiring between them, as shown in the diagram at right.

JList

Syntax : public class JList

extends JComponent

implements Scrollable, Accessible

A component that allows the user to select one or more objects from a list. A separate model, ListModel, represents the contents of the list.

// Create a JList that displays the strings in data[]

String[] data = {"one", "two", "three", "four"}; JList dataList = new JList(data);

JComboBox

Syntax : public class JComboBox

extends JComponent

implements ItemSelectable, ListDataListener, ActionListener, Accessible

- A component that combines a button or editable field and a drop-down list.
- The user can select a value from the drop-down list, which appears at the user's request.
- If you make the combo box editable, then the combo box includes an editable field into which the user can type a value.

JTable

Syntax : public class JTable

extends JComponent

implements TableModelListener, Scrollable, TableColumnModelListener, ListSelectionListener, CellEditorListener, Accessible

- The JTable is used to display and edit regular two-dimensional tables of cells.
- The JTable has many facilities that make it possible to customize its rendering and editing but provides defaults for these features so that simple tables can be set up easily.
- For example, to set up a table with 10 rows and 10 columns of numbers:

```
TableModel dataModel = new AbstractTableModel() {
public int getColumnCount() { return 10; }
public int getRowCount() { return 10;}
public Object getValueAt(int row, int col) { return new Integer(row*col); }
}
JTable table = new JTable(dataModel); JScrollPane scrollpane = new JScrollPane(table);
```

1) Swing demo

```
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.getContentPane().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();
      }
    });
  }
}
```

2)

```
// Demonstrate JToggleButton.
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
/*
  <applet code="jtoggle" width=200 height=80>
  </applet>
*/
```

```
public class jtoggle extends JApplet {
 JLabel jlab;
 JToggleButton jtbn;
 public void init() {
  try {
   SwingUtilities.invokeAndWait(
     new Runnable() {
       public void run() {
         makeGUI();
       }
     }
   );
  } catch (Exception exc) {
   System.out.println("Can't create because of " + exc);
  }
}

 private void makeGUI() {
  // Change to flow layout.
  getContentPane().setLayout(new FlowLayout());
  // Create a label.
  jlab = new JLabel("Button is off.");
  // Make a toggle button.
  jtbn =  new JToggleButton("On/Off");
  // Add an item listener for the toggle button.
```

```
    jtbn.addItemListener(new ItemListener() {
      public void itemStateChanged(ItemEvent ie) {
        if(jtbn.isSelected())
          jlab.setText("Button is on.");
        else
          jlab.setText("Button is off.");
      }

    });
    // Add the toggle button and label to the content pane.
     getContentPane().add(jtbn);
     getContentPane().add(jlab);
  }
}
```

3)

```
// Demonstrate JTextField.
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
/*
  <applet code="JTextFieldDemo" width=300 height=50>
  </applet>
*/
public class jtext extends JApplet {
  JTextField jtf;
```

```
public void init() {
  try {
    SwingUtilities.invokeAndWait(
      new Runnable() {
        public void run() {
          makeGUI();
        }
      }
    );
  } catch (Exception exc) {
    System.out.println("Can't create because of " + exc);
  }
}

private void makeGUI() {
  // Change to flow layout.
  getContentPane().setLayout(new FlowLayout());
  // Add text field to content pane.
  jtf = new JTextField(15);
  getContentPane().add(jtf);
  jtf.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent ae) {
      // Show text when user presses ENTER.
      showStatus(jtf.getText());
    }
```

```
    });
  }
}
```

4) //Demonstrate JScrollPane.

```
import java.awt.*;
import javax.swing.*;
/*
  <applet code="jscroll" width=300 height=250>
  </applet>
*/
public class jscroll extends JApplet {
  public void init() {
    try {
      SwingUtilities.invokeAndWait(
        new Runnable() {
          public void run() {
            makeGUI();
          }
        }
      );
    } catch (Exception exc) {
      System.out.println("Can't create because of " + exc);
    }
  }
```

```
  private void makeGUI() {
    // Add 400 buttons to a panel.
    JPanel jp = new JPanel();
    jp.setLayout(new GridLayout(20, 20));
    int b = 0;
    for(int i = 0; i < 20; i++) {
      for(int j = 0; j < 20; j++) {
        jp.add(new JButton("Button " + b));
        ++b;
      }
    }
    // Create the scroll pane.
    JScrollPane jsp = new JScrollPane(jp);

    // Add the scroll pane to the content pane.
    // Because the default border layout is used,
    // the scroll pane will be added to the center.
    getContentPane().add(jsp, BorderLayout.CENTER);
  }
}
```

5) // Demonstrate JList.

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

```
import java.awt.event.*;
/*
  <applet code="jlist" width=200 height=120>
  </applet>
*/
public class jlist extends JApplet {
  JList jlst;
  JLabel jlab;
  JScrollPane jscrlp;
  // Create an array of cities.
  String Cities[] = { "New York", "Chicago", "Houston",
                      "Denver", "Los Angeles", "Seattle",
                      "London", "Paris", "New Delhi",
                      "Hong Kong", "Tokyo", "Sydney" };
  public void init() {
    try {
      SwingUtilities.invokeAndWait(
        new Runnable() {
          public void run() {
            makeGUI();
          }
        }
      );
    } catch (Exception exc) {
      System.out.println("Can't create because of " + exc);
```

```
  }
}
private void makeGUI() {
  // Change to flow layout.
  getContentPane().setLayout(new FlowLayout());
  // Create a JList.
  jlst = new JList(Cities);
  // Set the list selection mode to single-selection.
  jlst.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);

  // Add the list to a scroll pane.
  jscrlp = new JScrollPane(jlst);
  // Set the preferred size of the scroll pane.
  jscrlp.setPreferredSize(new Dimension(120, 90));
  // Make a label that displays the selection.
  jlab = new JLabel("Choose a City");

  // Add selection listener for the list.
  jlst.addListSelectionListener(new ListSelectionListener() {
    public void valueChanged(ListSelectionEvent le) {
      // Get the index of the changed item.
      int idx = jlst.getSelectedIndex();
      // Display selection, if item was selected.
      if(idx != -1)
        jlab.setText("Current selection: " + Cities[idx]);
```

```
        else // Othewise, reprompt.
          jlab.setText("Choose a City");

      }
    });

    // Add the list and label to the content pane.
    getContentPane().add(jscrlp);
    getContentPane().add(jlab);
  }

6// Demonstrate JComboBox.
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
/*
  <applet code="jcombo" width=300 height=100>
  </applet>
*/

public class jcombo extends JApplet {
  JLabel jlab;
  ImageIcon france, germany, italy, japan;
  JComboBox jcb;
```

```
String flags[] = { "France", "Germany", "Italy", "Japan" };
public void init() {
  try {
    SwingUtilities.invokeAndWait(
      new Runnable() {
        public void run() {
          makeGUI();
        }
      }
    );
  } catch (Exception exc) {
    System.out.println("Can't create because of " + exc);
  }
}
private void makeGUI() {
  // Change to flow layout.
  getContentPane().setLayout(new FlowLayout());

  // Instantiate a combo box and add it to the content pane.
  jcb = new JComboBox(flags);
  getContentPane().add(jcb);
  // Handle selections.
  jcb.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent ae) {
```

```
        String s = (String) jcb.getSelectedItem();
        jlab.setIcon(new ImageIcon(s + ".gif"));
      }
    });
    // Create a label and add it to the content pane.
    jlab = new JLabel(new ImageIcon("france.gif"));
     getContentPane().add(jlab);
  }
}
```