

COURSE: ADVANCED JAVA PROGRAMMING (22517)



Unit 2 Swings

By

Mr. Vijay M Bande
Lecturer in Computer Engineering
Government Polytechnic Khamgaon

COURSE OUTCOMES(CO's)

- a) **Develop Programs using GUI Framework (AWT and Swings)**
- b) **Handle events of AWT and Swings Components.**
- c) Develop programs to handle events in Java Programming
- d) Develop Java programs using networking basics.
- e) Develop programs using Database.
- f) Develop programs using Servlets.

UNIT OUTCOMES(UO's)

Unit	Unit Outcomes (UOs) (in cognitive domain)	Topics and Sub-topics
Unit-II Swings	<p>2a. Differentiate between AWT and Swing on the given aspect.</p> <p>2b. Develop Graphical user interface (GUI) programs using swing components for the given problem.</p> <p>2c. Use the given type of button in Java based GUI.</p> <p>2d. Develop Graphical user interface (GUI) programs using advanced swing components for the given problem.</p>	<p>2.1 Introduction to swing:Swing features, Difference between AWT and Swing.</p> <p>2.2 Swing Components: JApplet,Icons and Labels, Text Fields, Combo Boxes.</p> <p>2.3 Buttons: The JButton, Check Boxes, Radio Buttons.</p> <p>2.4 Advanced Swing Components: Tabbed Panes, Scroll Panes, Trees, Tables, Progress bar, tool tips.</p> <p>2.5 MVC Architecture.</p>

THE ORIGINS OF SWING

- Swing did not exist in the early days of Java.
- AWT defines a basic set of controls, windows, dialog boxes that support usable, but limited graphical interface.
- One reason for limited nature of the AWT is that
 - **Look and feel of a component** is defined by the platform, not by Java.
 - AWT Components referred to as **heavyweight**
- Introduced in **1997**, **Swing** was included as part of the **Java Foundation Classes(JFC)**.



SWING IS BUILT ON THE AWT

- **Swing** is *built on the foundation* of the **AWT**.
- **Swing** also uses the *same event handling* mechanism as the **AWT**.



TWO KEY SWING FEATURES

○ Swing Components are Lightweight.

- This means that, they are written entirely in Java, and do not map directly to platform-specific peers.

○ Swing Supports a Pluggable Look and Feel

- It is possible to design custom look and feel. Also look and feel will changed dynamically at run time.

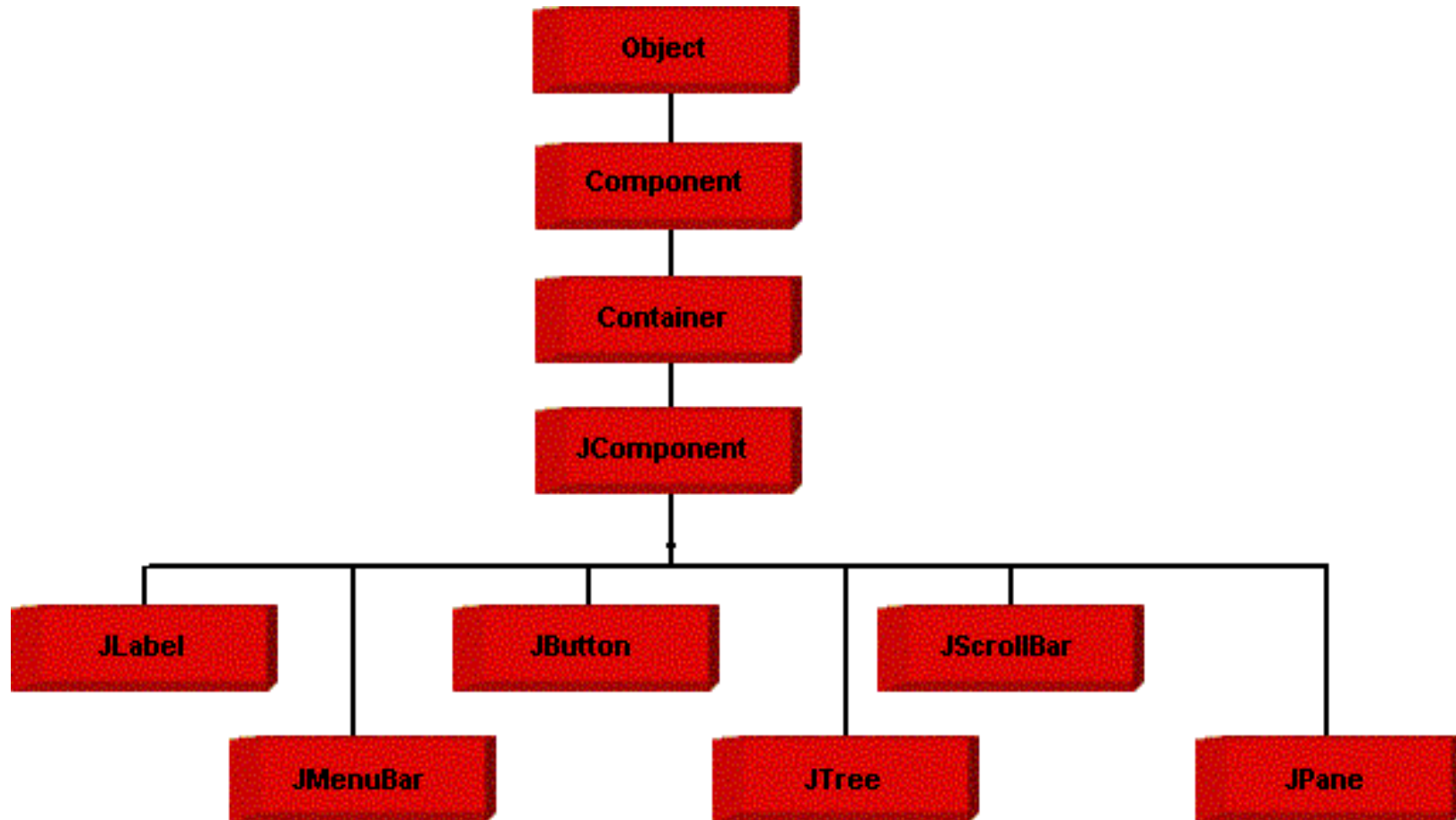


INTRODUCTION TO SWING


- Package : *javax.swing.**
- **Swing** is set of classes which provides more powerful and flexible components as compare to AWT.
- Build on top of **AWT API** and acts as replacement of AWT API.
- Swing component follows a ***Model-View- Controller***
- Swing Components are implemented **using Java** and so they are **platform independent**.
- Called **lightweight components**



INTRODUCTION TO SWING



THE MVC CONNECTION

- 100 % Java implementations of components.
 - *Use MVC architecture.*
 - **Model** represents state information associated with the component.
 - For example, in case of check box, the model contains field that indicates if the box is checked or unchecked.
 - **View** determines how the component is displayed on the screen,
 - **Controller** determines how the component reacts to the user,
 - For example, when user clicks a check box, the controller reacts by changing the model to reflect user's choice.
- 

THE MVC CONNECTION

- Swing uses a modified version of MVC that combines the view and the controller into a single logical entity called UI delegate.
- For this reason, Swing's approach is called as either **Model-Delegate architecture** or the **Separable Model architecture**.



DIFFERENCE BETWEEN AWT AND SWING

- *AWT uses Applet and Frame* while **Swing uses JApplet and JFrame for GUI.**
- *AWT is platform dependent code* while **Swing code is platform independent.**
- **Swing has bigger collection of classes and interfaces** as compare to AWT.
- In Swing extra feature to Button: **Provide Image.**
- Swing provides: **Tree, Table, Scrollpanes, Tabbedpanes** etc new feature which not available in AWT.



COMPONENTS AND CONTAINERS

- In general, Swing components are derived from the **JComponent** class.
- **JComponent** supports pluggable look and feel.
- **JComponent** inherits the AWT classes **Container** and **Component**.
- All Swing classes are represented by classes defined within the package **javax.swing**
- **Point to remember:** that all component classes begin with the letter **J**.



IMPORTANT CLASSES BY SWING

JApplet	JButton	JCheckBox	JCheckBoxMenuItem
JColorChooser	JComboBox	JComponent	JDesktopPane
JDialog	JEditorPane	JFileChooser	JFormattedTextField
JFrame	JInternalFrame	JLabel	JLayer
JLayeredPane	JList	JMenu	JMenuBar
JMenuItem	JOptionPane	JPanel	JPasswordField
JPopupMenu	JProgressBar	JRadioButton	JRadioButtonMenuItem
JRootPane	JScrollBar	JScrollPane	JSeparator
JSlider	JSpinner	JSplitPane	JTabbedPane

JTable	JTextArea	JTextField	JTextPane
JToggleButton	JToolBar	JToolTip	JTree
JViewport	JWindow		



COMPONENT AND CONTAINER

- Swing defines two types of containers:
 - Top-Level Containers
 - Lightweight Containers
- Top-Level Containers
 - JFrame
 - JApplet
 - JWindow
 - JDialog
- These containers **do not** inherit **JComponent** class
- The top level containers are **heavyweight**.



COMPONENT AND CONTAINER

- Lightweight containers do inherit **JComponent** class.
- An example of lightweight container is **JPanel**.



THE TOP-LEVEL CONTAINER PANES

- Each top level container defines set of panes.
- At the top of the hierarchy is an instance of **JRootPane**.
- **JRootPane** is a lightweight container whose purpose is to manage the other panes.
- The panes that comprise the root pane are called
 - **glass pane**
 - **content pane**
 - **layered pane**
- The pane with which your application will interact the most is the content pane, because this is the pane to which you will add visual components.

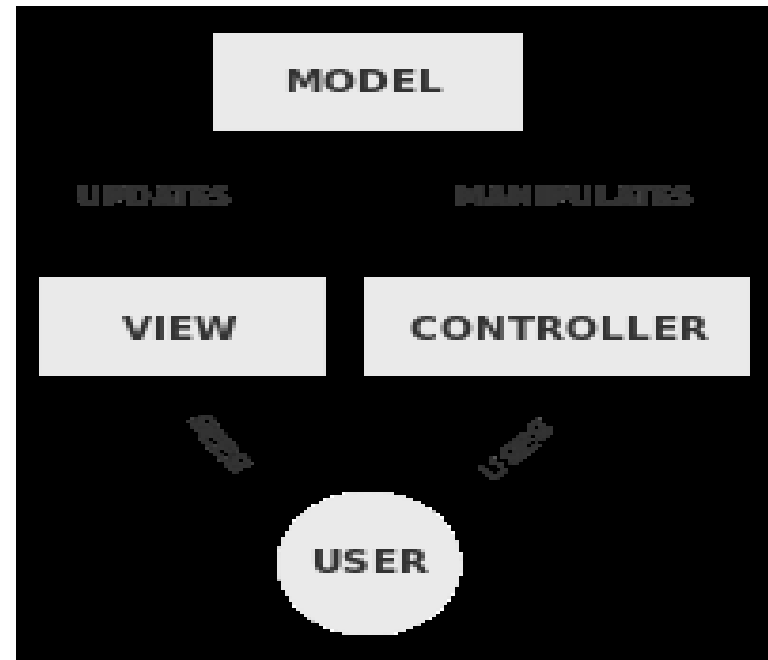
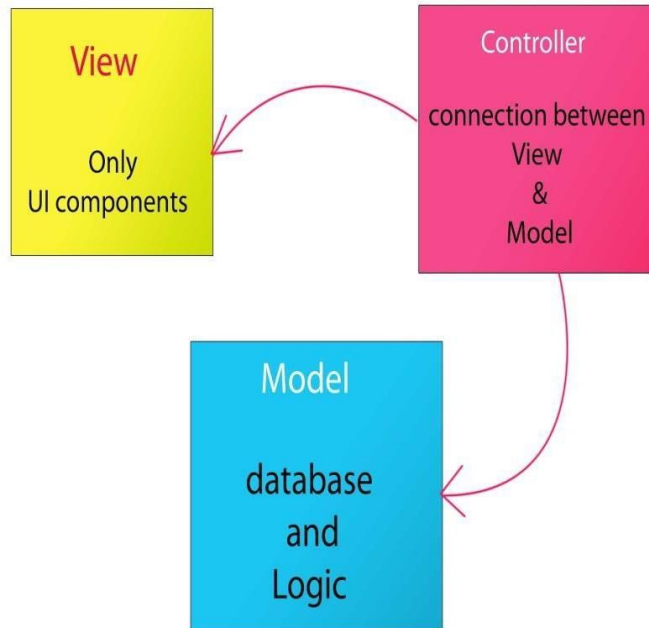
MVC ARCHITECTURE

- Software design pattern for software development.
- Model:
 - Major function of this layer to maintain the data.
 - Database and logic.
- View:
 - Used to display full or partial data.
 - User Interface
- Controller:
 - Control the interaction and communication between Model and view.
 - Communication logic/integration logic



MVC ARCHITECTURE

MVC Architecture (basic)



JAPPLET AND JFRAME

- Extends JApplet/JFrame class.
- Design UI in init() or Constructor method.
- Add all components on Container instead on JApplet/JFrame.
- **getContentPane()** method returns the container object.
- Call container add() method to add components.
- **For JFrame close operation:**
void setDefaultCloseOperation(int what)
- Parameters (The value passed in *what* determines what happens when the window is closed.):
 - **DISPOSE_ON_CLOSE**
 - **EXIT_ON_CLOSE**
 - **HIDE_ON_CLOSE**
 - **DO_NOTHING_ON_CLOSE**



JFRAME (EXAMPLE)

```
//A Simple Swing Application
import javax.swing.*;
public class SwingDemo
{
    SwingDemo()
    {
        //Create a new JFrame Container
        JFrame jfrm=new JFrame("A Simple Swing Application");

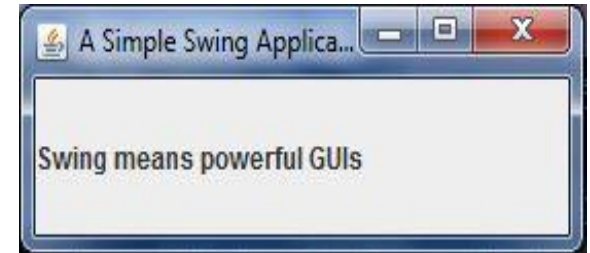
        //Give the frame an initial size
        jfrm.setSize(275,100);

        //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)
    {
        new SwingDemo();
    }
}
```



EXPLORING SWING

- Some of important **Swing** component classes :

JButton	JCheckBox	JComboBox	JLabel
JList	JRadioButton	JScrollPane	JTabbedPane
JTable	TextField	JToggleButton	JTree

- These components are lightweight, which means that they all are derived from **JComponent**.
- ButtonGroup** class, which encapsulates a mutually exclusive set of Swing Buttons.
- ImageIcon**, which encapsulates a graphics image.

JLABEL AND IMAGEICON

- Small display area for text, image or both.
- Passive component
- extends **JComponent**.
- **Constructors:**
 - **JLabel**(Icon *i*)
 - **JLabel**(String *s*)
 - **JLabel**(String *s*, Icon *i*, int *align*)
-where align argument is either **LEFT**, **RIGHT**,
CENTER, **LEADING** or **TRAILING**
- These constants are defined in the **SwingConstants** interface
- **ImageIcon:**
 - ImageIcon(String *filename*)
 - ImageIcon(URL *url*)



JLABLE AND IMAGEICON

- The **ImageIcon** class implements the Icon interface that declares the methods
 - **int** **getIconHeight()**
 - **int** **getIconWidth()**
- **Other methods (Setter and Getter Methods):**
 - The icon and text associated with the label can be obtained by the following methods:
 - **Icon** **getIcon()**
 - **String** **getText()**
 - The icon and text associated with a label can be set by these methods:
 - **void** **setIcon(Icon i)**
 - **void** **setText(String s)**



JTEXTFIELD

- **java.lang.Object**
 - **java.awt.Component**
 - **java.awt.Container**
 - **javax.swing.JComponent**
 - **javax.swing.text.JTextComponent**
 - **javax.swing.JTextField**



JTEXTFIELD

- **JTextField** allows you to edit one line of text.
- It is derived from **JTextComponent**.
- **JTextField** uses the **Document** interface for its model.
- Three of **JTextField**'s constructors:
 - **JTextField**(*int cols*)
 - *JTextField*(*String s, int cols*)
 - *JTextField*(*String s*)



JTEXTFIELD

- **JTextField** generates events in response to user interaction.
- For example,
 - An **ActionEvent** is fired when the user presses **ENTER**.
 - A **CaretEvent** is fired each time the caret(i.e. cursor) changes position.
- To obtain the text currently in the text field, call **getText()**.
- Note:** **CaretEvent** is packaged in **javax.swing.event**



THE SWING BUTTONS

- **Swing** defines **four** types of buttons:
 - 1) **JButton**
 - 2) **JToggleButton**
 - 3) **JCheckBox**
 - 4) **JRadioButton**
- All are subclasses of the **AbstractButton** class which extends **JComponent**.



THE SWING BUTTONS

- **AbstractButton** contains many methods that allow you to control the behaviour of buttons.
 - `void setDisabledIcon(Icon di)`
 - `void setPressedIcon(Icon pi)`
 - `void setSelectedIcon(Icon si)`
 - `void setRolloverIcon(Icon ri)`
- The text associated with the button can be read and written via following methods.
 - **`String getText()`**
 - **`void setText(String str)`**
- The model used by all button is defined by **ButtonModel** interface.



JBUTTON

- The **JButton** class provides the functionality of a **push button**.
- **JButton** allows an icon, a string or both to be associated with the push button.
- **Constructors:**
 - **`JButton(Icon i)`**
 - **`JButton(String s)`**
 - **`JButton(String s, Icon i)`**
- When the button is pressed, an **ActionEvent** is generated.



JTOGGLEBUTTON

- A toggle button looks like a push button, but it acts differently because it has two states:
 - pushed
 - released
- Toggle buttons are objects of the **JToggleButton** class.
- **JToggleButton** extends **AbstractButton**.
- **JToggleButton** is superclass for two other swing components that also represent two-state controls.
 - **JCheckBox**
 - **JRadioButton**



JTOGGLEBUTTON

- **JToggleButton** defines several constructors:
 - **JToggleButton(String str)**
- By **default**, the button is in the off position.
- Like **JButton**, **JToggleButton** generates **action event** each time it is pressed.
- Unlike **JButton**, **JToggleButton** also generates an **item event**.
- When a **JToggleButton** is pressed in, it is selected. When it is popped out, it is deselected.



JCHECKBOX

- java.lang.Object
 - java.awt.Component
 - java.awt.Container
 - javax.swing.JComponent
 - javax.swing.AbstractButton
 - javax.swing.JToggleButton
 - javax.swing.JCheckBox
- JCheckBox(Icon i)
- JCheckBox(Icon i, boolean state)
- JCheckBox(String s)
- JCheckBox(String s, boolean state)
- JCheckBox(String s, Icon i)
- JCheckBox(String s, Icon i, boolean state)



JCHECKBOX

- `void setSelected(boolean state)`
- **ItemEvent** is generated.
- **ItemListener** interface is needed to handle **ItemEvent**.
- `public itemStateChnaged()` used to override.



JRADIOBUTTON

- java.lang.Object
 - java.awt.Component
 - java.awt.Container
 - javax.swing.JComponent
 - javax.swing.AbstractButton
 - javax.swing.JToggleButton
 - javax.swing.JRadioButton

- 1) **JRadioButton(Icon i)**
- 2) **JRadioButton(Icon i, boolean state)**
- 3) **JRadioButton(String s)**
- 4) **JRadioButton(String s, boolean state)**
- 5) **JRadioButton(String s, Icon i)**
- 6) **JRadioButton(String s, Icon i, boolean state)**



JRADIOBUTTON

- **ButtonGroup** class is used to add radio button in group.
- **ActionEvent** is generated.
- **ActionListener** Listener interface is needed to handle **ActionEvent**.
- **public void actionPerformed()** used to override.

