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

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

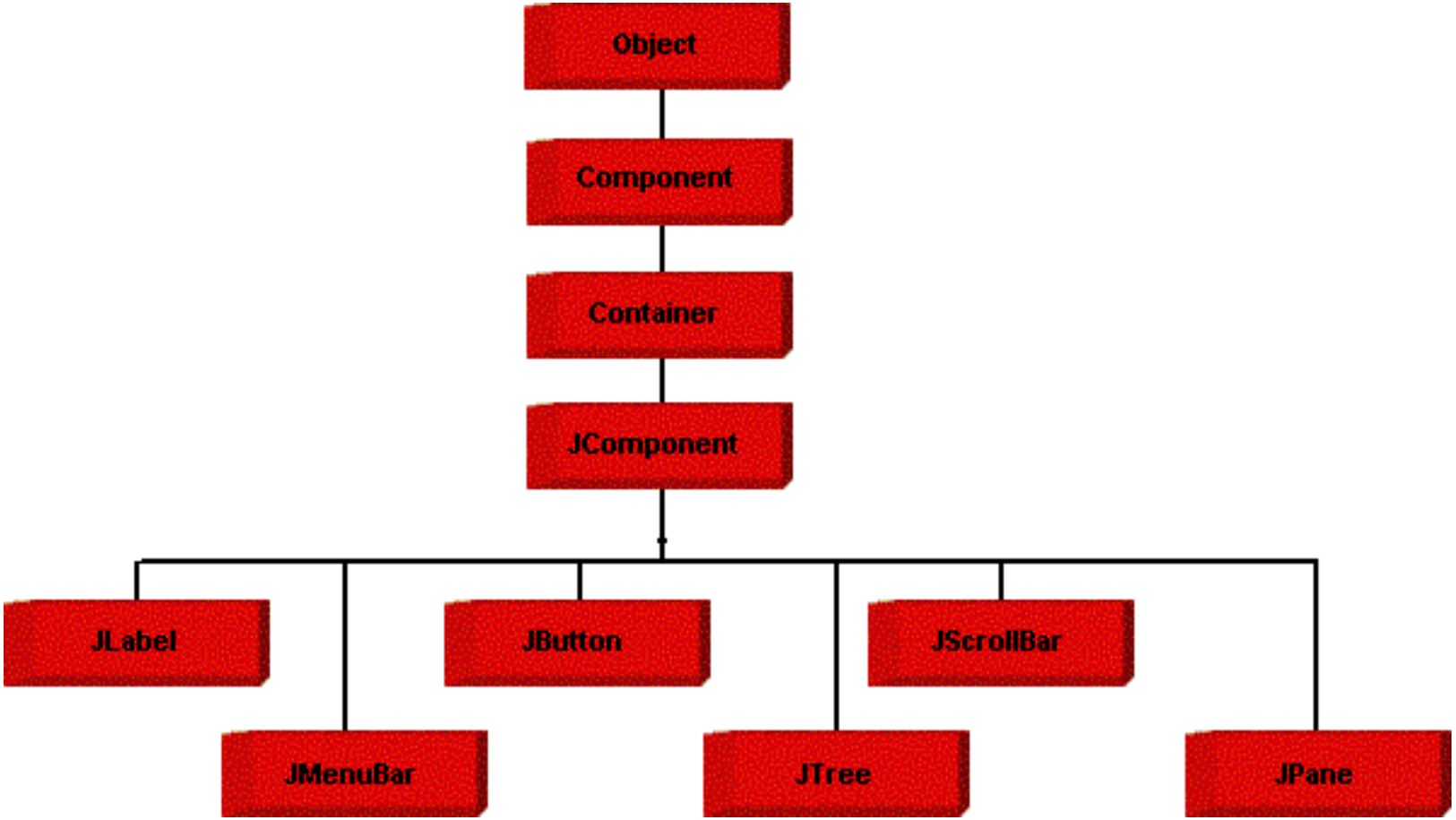
- **Swing Components are Lightweight.**
  - This means that, they are written entirely in Java, and do not map directly to platform-specific peers.

- **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

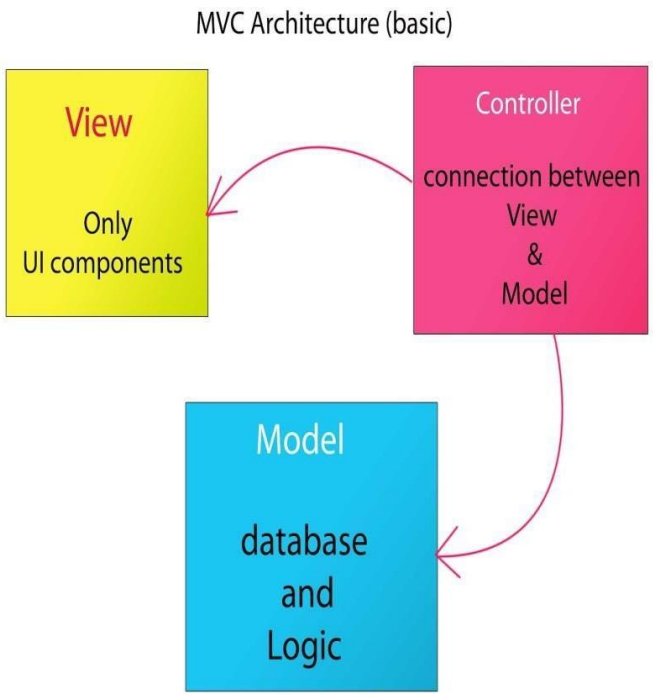
- Software design pattern for software development.

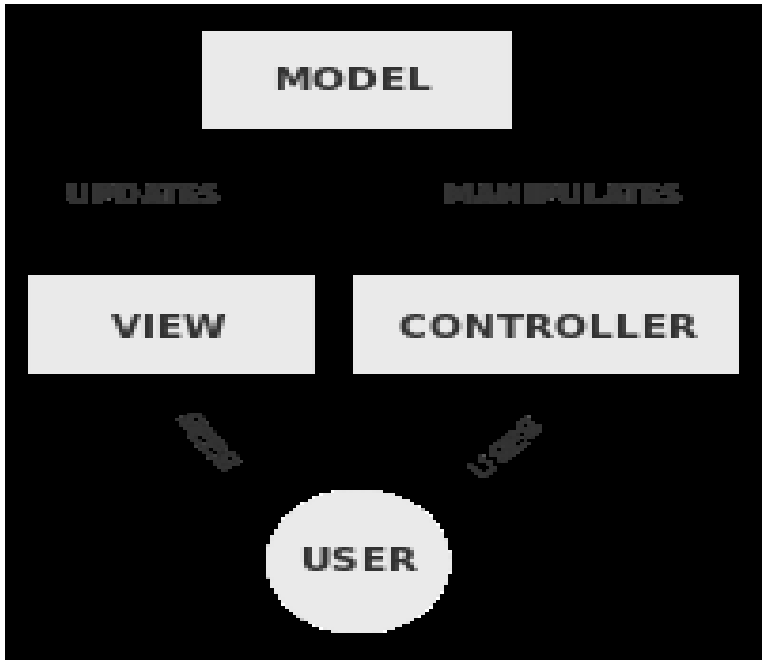
- Model:
  - Major function of this layer to maintain the data.
  - Database and logic.

- 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





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)

```java
//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	**JTextField**	**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.

JLABLE 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 actionPerofrmed() used to override.

JTABBEDPANE

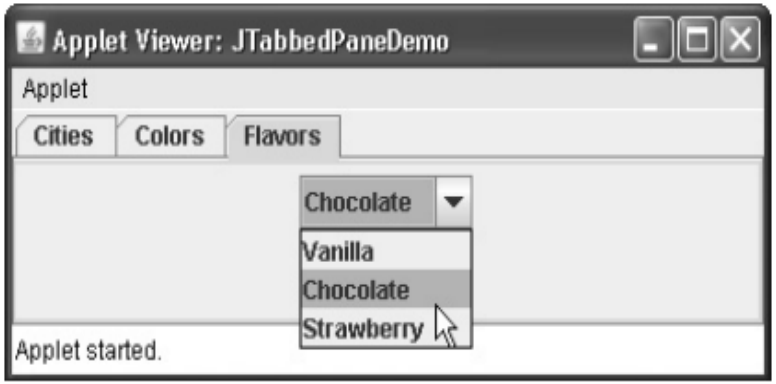
- *A **tabbed pane** is a component that appears as a group of folders in a file cabinet.*
- Each folder has a title.
- When a user selects a folder, its contents become visible.
- Only one of the folders may be selected at a time.
- Tabbed panes are commonly used for setting configuration options.
- subclass of **JComponent.**

JTABBEDPANE

- Tabs are defined via the following method :
  - **void addTab(String *str, Component comp)***
    - str: name for the tab,
    - comp: component, it can be JPanel

- **Steps to create JTabbedPane**:
  1. Create a **JTabbedPane** object.
  2. Call **addTab( )** to add a tab to the pane.
  3. Repeat step 2 for each tab.
  4. Add the tabbed pane to the content pane of the **JApplet** or **JFrame**

JTABBEDPANE



JSCROLLPANE

- A *scroll pane is a component that presents a rectangular area in which a component may be viewed.*
- *The viewable area of a scroll pane is called the* ***viewport.***
- Horizontal and/or vertical scroll bars may be provided if necessary.
- subclass of **JComponent**
- **Constructor:**
  - **JScrollPane(Component *comp)***
  - **JScrollPane(int *vsb, int hsb)***
  - **JScrollPane(Component *comp, int vsb, int hsb)***

    -comp: Component, vsb and hsb: Scrollbar constant

JSCROLLPANE

- These vsb and hsb constants are defined by **ScrollPaneConstants** interface.
  - **HORIZONTAL_SCROLLBAR_ALWAYS**
  - **HORIZONTAL_SCROLLBAR_AS_NEEDED**
  - **VERTICAL_SCROLLBAR_ALWAYS**
  - **VERTICAL_SCROLLBAR_AS_NEEDED**

- **Here are steps to follow to use a scroll pane**
  - Create the component to be added
  - Create an instance of **JScrollPane**, passing to it object to scroll.
  - Add the scroll pane to the content pane.

JLIST

- It supports the selection of one or more item from a list.
- In Swing, the basic list class is called **JList**.
- One of Constructors of **JList**
  - **JList**(E[] items)

    -this will creates a **JList** that contains the items in the array specified by items.

- By default, a **JList** allows the user to select multiple ranges of items within the list, but you can change this behavior by calling **setSelectionMode().**

JLIST

- **Syntax:**
  - void setSelectionMode(int mode)

    -here mode specifies the selection mode. It must be one of these values defined by **ListSelectionModel**.

    1) SINGLE_SELECTION
    2) SINGLE_INTERVAL_SELECTION
    3) MULTIPLE_INTERVAL_SELECTION

- You can obtain the index of the first item selected, by calling **getSelectedIndex().**
- You can obtain the value associated with the selection by calling **getSelectedValue().**

JLIST

- **JList** generates a **ListSelectionEvent** when the user makes or changes a selection.

- It is handled by implementing **ListSelectionListener.**

- This listener specifies only one method called **valueChanged()** which is shown here:
  - void valueChanged(ListSelectionEvent lse)

JCOMBOBOX

- Combination of text field and drop-down list.
- subclass of **JComponent**.
- Only one entry can view at a time.
- You can also create a combo box that lets the user enter selection into the text field.
- Constructor:
  - **JComboBox( )**
  - **JComboBox(Vector *v*)**
- **void addItem(Object obj)**: Used to add object in ComboBox

JC OMBOBOX

- JComboBox **uses** the **ComboBoxModel.**
- Mutable(those whose entries can be changed) use the **MutableComboBoxModel**.

- To obtain the item selected in the list, is to call **getSelectedItem()** on the ComboBox.
  - **Object getSelectedItem()**

JCOMBOBOX: EVENT HANDLING

- **ItemEvent** is generated.
- implements ItemListener interface
- Override: itemStateChnaged(ItemEvent ie) method defined by ItemListener.

JTREE

- A *tree is a component that presents a hierarchical view of data.*

- Trees are implemented in Swing by the **JTree** class, which extends **JComponent.**

- **Constructors:**
  - JTree(Hashtable *ht)*
  - JTree(Object *obj[ ])*
  - JTree(TreeNode *tn)*
  - JTree(Vector *v)*

JTREE

- **JTree** is packaged **in javax.swing**, its support **classes** and **interfaces** are packaged in **javax.swing.tree.**

- **JTree** relies on two models:
  - TreeModel and
  - TreeSelectionModel

- A **JTree** generates a variety of events, but three relate specifically to trees:

  1. TreeExpansionEvent,
  2. TreeSelectionEvent, and
  3. TreeModelEvent

JTREE

- **TreeExpansionEvent** events occur when a node is expanded or collapsed.
- A **TreeSelectionEvent** is generated when the user selects or deselects a node within the tree.
- A **TreeModelEvent** is fired when the data or structure of the tree changes.
- The listeners for these events are
  - TreeExpansionListener,
  - TreeSelectionListener, and
  - TreeModelListener
  - The tree event classes and listener interfaces are packaged in **javax.swing.event.**

JTREE

- The **TreeNode** interface declares methods that obtain information about a tree node.
- For example, it is possible to obtain a reference to the parent node or an enumeration of the child nodes.
- The **MutableTreeNode** interface extends **TreeNode.**
- It declares methods that can insert and remove child nodes or change the parent node.
- The **DefaultMutableTreeNode** class implements the **MutableTreeNode** interface.
- It represents a node in a tree. One of its constructors is shown here:

DefaultMutableTreeNode(Object obj)

JTREE

- To create a hierarchy of tree nodes, the **add( )** method of **DefaultMutableTreeNode** can be used.
  - void add(MutableTreeNode *child)
- Tree Expansion event described by class:
  - **TreeExpansionEvent (Package: javax.swing.event)**
- The **getPath( )** method of this class returns a TreePath.
  - TreePath getPath( )
- **TreeExpansionListener** interface provides the following two methods
  - void treeCollapsed(TreeExpansionEvent *tee)*
  - void treeExpanded(TreeExpansionEvent *tee)*

JTREE

- **Steps to create JTree:**
  1. Create a **JTree** object.
  2. Create a **JScrollPane** object.
  3. Add the tree to the scroll pane.
  4. Add the scroll pane to the content pane of the applet.

JTREE

- A **JTree** object generates events when a node is expanded or collapsed.

- The **addTreeExpansionListener( )** and **removeTreeExpansionListener( )** methods allow listeners to register and unregister for these notifications.

- **Signature for these methods:**
  - void addTreeExpansionListener(TreeExpansionListener *tel)*
  - void removeTreeExpansionListener(TreeExpansionListener *tel)*

JTABLE

- A *table is a component that displays rows and columns of data.*
- You can drag the cursor on column boundaries to resize columns.
- You can also drag a column to a new position.
- subclass of **JComponent**
- **Constructor:**
  - JTable(Object *data[ ][ ], Object colHeads[ ])*

    -data is a two-dimensional array of the information

    -colHeads is a one-dimensional array with the column headings.

JTABLE

- **Steps to create JTable**
  1. Create a **JTable** object.
  2. Create a **JScrollPane** object.
  3. Add the table to the scroll pane.
  4. Add the scroll pane to the content pane of the JApplet or JFrame.

JTABLE

- **JTable** relies on three models.
  - **TableModel (present in javax.swing.table)**

    -displaying data in two dimensional format
  - **TableColumnModel (present in javax.swing.table)**

    -specifies characteristics of column
  - **ListSelectionModel**

    -determines how items are selected.

- A **JTable** can generate several different events.
  - **ListSelectionEvent**
    - A **ListSelectionEvent** is generated when the user selects something in the table.
  - **TableModelEvent**

    - A **TableModelEvent** is fired when that table's data changes in some way.

PROGRESSBAR

- **JProgressBar** is a part of Java Swing package.
- **JProgressBar** inherits your **JComponent** class.
- **JProgressBar** visually displays the progress of some specified task.
- **JProgressBar** shows the percentage of completion of specified task.
- The progress bar fills up as the task reaches it completion.
- In addition to show the percentage of completion of task, it can also display some text .

PROGRESSBAR

- Constructors of **JProgressBar**:

- **JProgressBar()** :
  - It is used to create a horizontal progress bar but no string text.

- **JProgressBar(int orientation)** :
  - Orientation-SwingConstants.VERTICAL
  - Orientation-SwingConstants.HORIZONTAL

- **JProgressBar(int min, int max)** :
  - creates an progress bar with specified minimum and
  - maximum value.

- **JProgressBar(int orientation, int min, int max)** :

JPROGRESSBAR

- **Commonly used methods of JProgressBar are :**
  - **int getMaximum()**
  - **int getMinimum()**
  - **String getString()**
  - **void setMaximum(int n)**
  - **void setMinimum(int n)**
  - **void setValue(int n)**
  - **void setString(String s)**

TOOLTIP

- We can add tooltip text to almost all the components of Java Swing by using the following method
  - **setToolTipText(String s).**
    - This method sets the tooltip of the component to this specified string s .
- When the cursor enters the boundary of that component a popup appears and text is displayed .
- More Methods:
  - **getToolTipText()** : returns the tooltip text for that component .

JSEPERATOR

- **JSeparator** is a part of Java Swing framework.

- It is used to create a dividing line between two components.

- More specifically, it is mainly used to create dividing lines between menu items in a JMenu.

- **Constructor of the class are**:
  - **JSeparator()**: Creates a new horizontal separator.
  - **JSeparator(int o)**: Creates a new separator with the specified horizontal or vertical orientation.

JSEPARATOR

- Commonly used Methods:

  1) setOrientation(int o) - Sets the orientation of the separator.

  2) getOrientation() - returns the orientation of the separator.

  3)addSeparator() - adds a separator in JMenu or JPopupMenu.

VOCABULARY

- AWT – The Abstract Window Toolkit provides basic graphics tools (tools for putting information on the screen)
- Swing – A much better set of graphics tools
- Container – a graphic element that can hold other graphic elements (and is itself a Component)
- Component – a graphic element (such as a Button or a TextArea) provided by a graphics toolkit
- listener – A piece of code that is activated when a particular kind of event occurs
- layout manager – An object whose job it is to arrange Components in a Container