**Swings**

* **Java Swing** is a part of Java Foundation Classes (JFC) that is *used to create window-based applications*.
* It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.
* Unlike AWT, Java Swing provides platform-independent and lightweight components.
* The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.
* **Difference between AWT and Swing :**

There are many differences between java awt and swing that are given below.

|  |  |  |
| --- | --- | --- |
| No. | Java AWT | Java Swing |
| 1) | AWT components are platform dependent. | Java swing components are platform-independent. |
| 2) | AWT components are heavyweight. | Swing components are lightweight. |
| 3) | AWT doesn't support pluggable look and feel. | Swing supports pluggable look and feel. |
| 4) | AWT provides less components than Swing. | Swing provides more powerful components such as tables, lists, scrollpanes, colorchooser, tabbedpane etc. |
| 5) | AWT doesn't follows MVC. | Swing follows MVC. (Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view. |

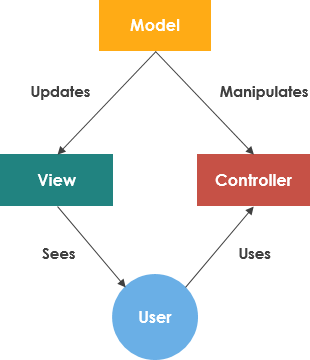
* Swing API is a set of extensible GUI Components to ease the developer's life to create JAVA based Front End/GUI Applications. It is built on top of AWT API and acts as a replacement of AWT API, since it has almost every control corresponding to AWT controls. Swing component follows a Model-View-Controller architecture to fulfil the following criteria.
* A single API is to be sufficient to support multiple looks and feel.
* API is to be model driven so that the highest level API is not required to have data.
* API is to use the Java Bean model so that Builder Tools and IDE can provide better services to the developers for use.

## Swing Features

* **Light Weight** − Swing components are independent of native Operating System's API as Swing API controls are rendered mostly using pure JAVA code instead of underlying operating system calls.
* **Rich Controls** − Swing provides a rich set of advanced controls like Tree, TabbedPane, slider, colorpicker, and table controls.
* **Highly Customizable** − Swing controls can be customized in a very easy way as visual apperance is independent of internal representation.
* **Pluggable look-and-feel** − SWING based GUI Application look and feel can be changed at run-time, based on available values.

## MVC Architecture:

* MVC stands for Model-View-Controller. It is architecture or a software design pattern that makes creating huge applications easy. It does not belong to specific programming language or framework, but it is a concept that you can use in creating any kind of application or software in any programming language.
* Swing API architecture follows loosely based MVC architecture in the following manner.
* The goal of MVC is to help structure the separate the concerns of an application into three parts:



* Model is responsible for managing the data of the application. It receives user input from the controller.
* View means the presentation of the model in a particular format.
* Controller responds to the user input and performs interactions on the data model objects. The controller receives the input, optionally validates it and then passes the input to the model.

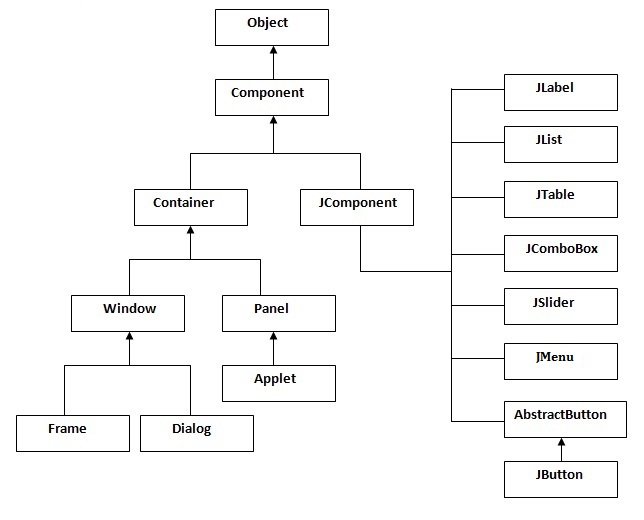
**Advantages of MVC architecture:**

* Development of the application becomes fast.
* Easy for multiple developers to collaborate and work together.
* Easier to update the application.
* Easier to Debug as we have multiple levels properly written in the application.

**Disadvantages of MVC architecture:**

* It is hard to understand the MVC architecture.
* Must have strict rules on methods.

**The hierarchy of java swing API is given below.**



**Swing Components**

* **The list of swing components are:**
* JTree
* JButton
* JLabel
* JTextField
* JTextArea
* JPasswordField
* JCheckBox
* JRadioButton

**1. JTree**

The JTree class is used to display the tree structured data or hierarchical data. JTree is a complex component. It has a 'root node' at the top most which is a parent for all nodes in the tree. It inherits JComponent class.

**JTree class declaration:**

Let's see the declaration for javax.swing.JTree class.

**public** **class** JTree **extends** JComponent **implements** Scrollable,

Accessible commonly used Constructors:

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| JTree() | Creates a JTree with a sample model. |
| JTree(Object[] value) | Creates a JTree with every element of the specified array as the child of a new root node. |
| JTree(TreeNode root) | Creates a JTree with the specified TreeNode as its root, which displays the root node. |

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

**TreeExpansionEvent**,

**TreeSelectionEvent**, and

**TreeModelEvent**.

* **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.

// Demonstration of JTree:

import javax.swing.\*;

import javax.swing.tree.DefaultMutableTreeNode;

public class JTreeDemo

{

    JFrame f;

    JTreeDemo()

    {

        f = new JFrame();

        DefaultMutableTreeNode Style = new DefaultMutableTreeNode("Style");

        DefaultMutableTreeNode Color = new DefaultMutableTreeNode("Color");

        DefaultMutableTreeNode Font = new DefaultMutableTreeNode("Font");

        Style.add(Color);

        Style.add(Font);

        DefaultMutableTreeNode Red=new DefaultMutableTreeNode("Red");

        DefaultMutableTreeNode Blue=new DefaultMutableTreeNode("Blue");

        DefaultMutableTreeNode Black=new DefaultMutableTreeNode("Black");

        DefaultMutableTreeNode Green=new DefaultMutableTreeNode("Green");

        Color.add(Red);

        Color.add(Blue);

        Color.add(Black);

        Color.add(Green);

        DefaultMutableTreeNode Calibri=new DefaultMutableTreeNode("Calibri");

        DefaultMutableTreeNode TimesNewRoman = new DefaultMutableTreeNode("TimesNewRoman");

        Font.add(Calibri);

        Font.add(TimesNewRoman);

        JTree jt=new JTree(Style);

        f.add(jt);

        f.setSize(200,200);

        f.setVisible(true);

    }

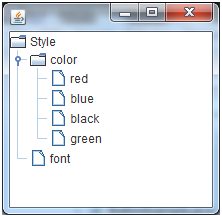
    public static void main(String[] args)

    {

        new JTreeDemo();

    }

}// Output:



Note 1: Syntax for SetSize method

JFrame: setSize(int width, int height)

Note 2 :

The **DefaultMutableTreeNode**class implements the **MutableTreeNode**interface. It represents a node in a tree. One of its constructors is shown here:

DefaultMutableTreeNode(Object *obj*)

Here, *obj*is the object to be enclosed in this tree node. The new tree node doesn’t have a parent or children.

To create a hierarchy of tree nodes, the **add( )**method of **DefaultMutableTreeNode**can be used. Its signature is shown here:

void add(MutableTreeNode*child*)

Here, *child* is a mutable tree node that is to be added as a child to the current node.

**2.JLabel**

The object of JLabel class is a component for placing text in a container. It is used to display a single line of read only text. The text can be changed by an application but a user cannot edit it directly. It inherits JComponent class.

JLabel class declaration

Let's see the declaration for javax.swing.JLabel class.

**public** **class** JLabel **extends** JComponent **implements** SwingConstants, Accessible

Commonly used Constructors:

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| JLabel() | Creates a JLabel instance with no image and with an empty string for the title. |
| JLabel(String s) | Creates a JLabel instance with the specified text. |
| JLabel(Icon i) | Creates a JLabel instance with the specified image. |
| JLabel(String s, Icon i, int horizontalAlignment) | Creates a JLabel instance with the specified text, image, and horizontal alignment. |

Commonly used Methods:

|  |  |
| --- | --- |
| **Methods** | **Description** |
| String getText() | It returns the text string that a label displays. |
| void setText(String text) | It defines the single line of text this component will display. |
| void setHorizontalAlignment(int alignment) | It sets the alignment of the label's contents along the X axis. |
| Icon getIcon() | It returns the graphic image that the label displays. |
| int getHorizontalAlignment() | It returns the alignment of the label's contents along the X axis. |

// Demonstration of JLabel:

import javax.swing.\*;

class JLabelDemo

{

    public static void main(String args[])

    {

        JFrame f= new JFrame("JLabel Demo");

        JLabel l1,l2,l3;

        l1=new JLabel("Hello,");

        l1.setBounds(50,50, 200,30);

        l2=new JLabel("Good Morning everyone!");

        l2.setBounds(50,70, 200,30);

        l3=new JLabel("My name is Tayyab.");

        l3.setBounds(50,90, 200,30);

        f.add(l1);

        f.add(l2);

        f.add(l3);

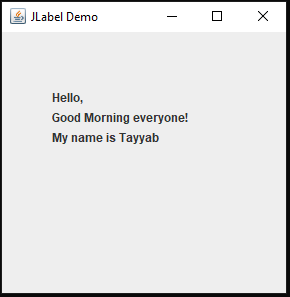
        f.setSize(300,300);

        f.setLayout(null);

        f.setVisible(true);

    }

}// Output:



**setLayout() :** Every container has a default layout manager and has an instance method, setLayout(), that takes a parameter of type LayoutManager and that is used to specify a different layout manager for the container.

**setLayout(null)** : null layout means absolute positioning - you have to do all the work in your code. No layout manager to help you out.

**The setBounds() method** not only specifies the size of the frame, but the location of the upper left corner:

public void setBounds(int x, int y, int width, int height)

This puts the upper left corner at location (x, y), where x the the number of pixels from the left of the screen and y is is the number from the top of the screen.height and width are as before.

# 3.JTable

The JTable class is used to display data in tabular form. It is composed of rows and columns.

JTable class declaration

Let's see the declaration for javax.swing.JTable class.

Commonly used Constructors:

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| JTable() | Creates a table with empty cells. |
| JTable(Object[][] rows, Object[] columns) | Creates a table with the specified data. |

// Demonstration of JTable:

import javax.swing.\*;

public class JTableDemo

{

    JFrame f;

    JTableDemo()

    {

        f=new JFrame();

        String data[][]={ {"101","Amit","67000"},

                          {"102","Jai","78000"},

                          {"101","Sachin","70000"}};

        String column[]={"ID","NAME","SALARY"};

        JTable JT = new JTable(data,column);

        JT.setBounds(30,40,200,300);

        JScrollPane sp=new JScrollPane(JT);

        f.add(sp);

        f.setSize(400,400);

        f.setVisible(true);

    }

    public static void main(String[] args)

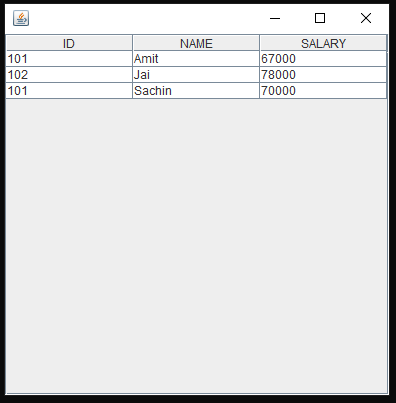
    {

        new JTableDemo();

    }

}

Outptut:



# 4. JCheckBox

The JCheckBox class is used to create a checkbox. It is used to turn an option on (true) or off (false). Clicking on a CheckBox changes its state from "on" to "off" or from "off" to "on ".It inherits JToggleButton class.

JCheckBox class declaration

Let's see the declaration for javax.swing.JCheckBox class.

**public** **class** JCheckBox **extends** JToggleButton **implements** Accessible

Commonly used Constructors:

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| JCheckBox() | Creates an initially unselected check box button with no text, no icon. |
| JChechBox(String s) | Creates an initially unselected check box with text. |
| JCheckBox(String text, boolean selected) | Creates a check box with text and specifies whether or not it is initially selected. |
| JCheckBox(Action a) | Creates a check box where properties are taken from the Action supplied. |

import javax.swing.\*;

public class JCheckBoxExample

{

JCheckBoxExample()

{

JFrame f= new JFrame("CheckBox Example");

JCheckBox checkBox1 = new JCheckBox("C++");

checkBox1.setBounds(100,100, 50,50);

JCheckBox checkBox2 = new JCheckBox("Java", true);

checkBox2.setBounds(100,150, 50,50);

f.add(checkBox1);

f.add(checkBox2);

f.setSize(400,400);

f.setLayout(null);

f.setVisible(true);

}

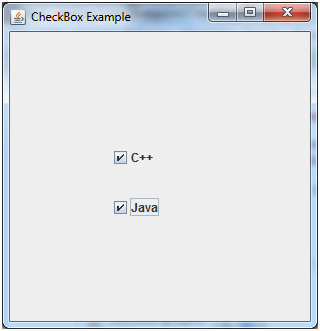
public static void main(String args[])

{

new JCheckBoxExample();

}

}



# 5.JComboBox

The object of Choice class is used to show popup menu of choices. Choice selected by user is shown on the top of a menu. It inherits JComponent class.

JComboBox class declaration

Let's see the declaration for javax.swing.JComboBox class.

**public** **class** JComboBox **extends** JComponent **implements** ItemSelectable, ListDataListener, ActionListener, Accessible

Commonly used Constructors:

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| JComboBox() | Creates a JComboBox with a default data model. |
| JComboBox(Object[] items) | Creates a JComboBox that contains the elements in the specified array. |
| JComboBox(Vector<?> items) | Creates a JComboBox that contains the elements in the specified Vector. |

Commonly used Methods:

|  |  |
| --- | --- |
| **Methods** | **Description** |
| void addItem(Object anObject) | It is used to add an item to the item list. |
| void removeItem(Object anObject) | It is used to delete an item to the item list. |
| void removeAllItems() | It is used to remove all the items from the list. |
| void setEditable(boolean b) | It is used to determine whether the JComboBox is editable. |
| Void addActionListener(ActionListener a) | It is used to add the ActionListener. |
| void addItemListener(ItemListeneri) | It is used to add the ItemListener. |

import javax.swing.\*;

public class JComboBoxExample

{

JFrame f;

JComboBoxExample()

{

f=new JFrame("ComboBox Example");

String country[]={"India","Aus","U.S.A","England","Newzealand"};

JComboBoxcb=new JComboBox(country);

cb.setBounds(50, 50,90,20);

f.add(cb);

f.setLayout(null);

f.setSize(400,500);

f.setVisible(true);

}

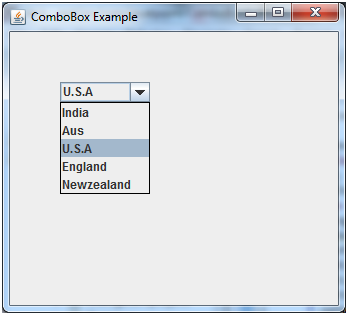
public static void main(String[] args)

{

new JComboBoxExample();

}

}



6.JButton

The JButton class is used to create a labeled button that has platform independent implementation. The application result in some action when the button is pushed. It inherits AbstractButton class.

JButton class declaration

Let's see the declaration for javax.swing.JButton class.

1. **public** **class** JButton **extends** AbstractButton **implements** Accessible

Commonly used Constructors:

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| JButton() | It creates a button with no text and icon. |
| JButton(String s) | It creates a button with the specified text. |
| JButton(Icon i) | It creates a button with the specified icon object. |

Commonly used Methods of AbstractButton class:

|  |  |
| --- | --- |
| **Methods** | **Description** |
| void setText(String s) | It is used to set specified text on button |
| String getText() | It is used to return the text of the button. |
| void setEnabled(boolean b) | It is used to enable or disable the button. |
| void setIcon(Icon b) | It is used to set the specified Icon on the button. |
| Icon getIcon() | It is used to get the Icon of the button. |
| void setMnemonic(int a) | It is used to set the mnemonic on the button. |
| void addActionListener(ActionListener a) | It is used to add the action listener to this object. |

import javax.swing.\*;

public class ButtonExample

{

public static void main(String[] args)

{

JFrame f=new JFrame("Button Example");

JButton b1=new JButton("C");

b1.setBounds(50,50,50,50);

f.add(b1);

JButton b2=new JButton("C++");

b2.setBounds(100,100,100,100);

f.add(b2);

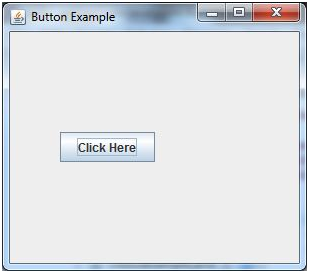
f.setSize(400,400);

f.setLayout(null);

f.setVisible(true);

}

}



**Note : Define setLayout(null)**

\*This indicates no Layout. Items must be manually positioned and arranged.  
This layout should only be used if the window will not and cannot be resized, as the item in the window will stay where they are placed, be that hidden or clumped in one corner of a window.

# 7.JRadioButton

The JRadioButton class is used to create a radio button. It is used to choose one option from multiple options. It is widely used in exam systems or quiz.

It should be added in ButtonGroup to select one radio button only.

JRadioButton class declaration

Let's see the declaration for javax.swing.JRadioButton class.

1. **public** **class** JRadioButton **extends** JToggleButton **implements** Accessible

Commonly used Constructors:

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| JRadioButton() | Creates an unselected radio button with no text. |
| JRadioButton(String s) | Creates an unselected radio button with specified text. |
| JRadioButton(String s, boolean selected) | Creates a radio button with the specified text and selected status. |

Commonly used Methods:

|  |  |
| --- | --- |
| **Methods** | **Description** |
| void setText(String s) | It is used to set specified text on button. |
| String getText() | It is used to return the text of the button. |
| void setEnabled(boolean b) | It is used to enable or disable the button. |
| void setIcon(Icon b) | It is used to set the specified Icon on the button. |
| Icon getIcon() | It is used to get the Icon of the button. |
| void setMnemonic(int a) | It is used to set the mnemonic on the button. |
| void addActionListener(ActionListener a) | It is used to add the action listener to this object. |

import javax.swing.\*;

public class RadioButtonExample

{

JFrame f;

RadioButtonExample()

{

f=new JFrame();

JRadioButton r1=new JRadioButton("A) Male");

JRadioButton r2=new JRadioButton("B) Female");

r1.setBounds(75,50,100,30);

r2.setBounds(75,100,100,30);

ButtonGroupbg=new ButtonGroup();

bg.add(r1);

bg.add(r2);

f.add(r1);

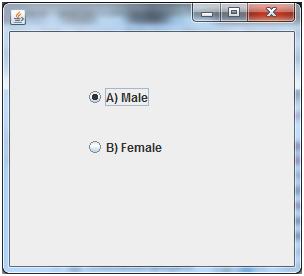
f.add(r2);

f.setSize(300,300);

f.setLayout(null);

f.setVisible(true);

}



Exe 2:

import javax.swing.\*;

import java.awt.event.\*;

class RadioButtonExample2 extends JFrame implements ActionListener

{

JRadioButton rb1,rb2;

JButton b;

RadioButtonExample2()

{

rb1=new JRadioButton("C");

rb1.setBounds(100,50,100,30);

rb2=new JRadioButton("C++");

rb2.setBounds(100,100,100,30);

ButtonGroupbg=new ButtonGroup();

bg.add(rb1);

bg.add(rb2);

b=new JButton("click");

b.setBounds(100,150,80,30);

b.addActionListener(this);

add(rb1);

add(rb2);

add(b);

setSize(300,300);

setLayout(null);

setVisible(true);

}

public void actionPerformed(ActionEvent e)

{

if(rb1.isSelected())

{

JOptionPane.showMessageDialog(this,"You selected C.");

}

if(rb2.isSelected())

{

JOptionPane.showMessageDialog(this,"You selected C++.");

}

}

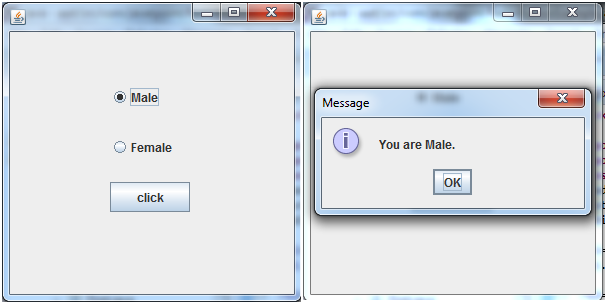
public static void main(String args[])

{

new RadioButtonExample2();

}

}



# 8.JScrollBar

The object of JScrollbar class is used to add horizontal and vertical scrollbar. It is an implementation of a scrollbar. It inherits JComponent class.

JScrollBar class declaration

Let's see the declaration for javax.swing.JScrollBar class.

1. **public** **class** JScrollBar **extends** JComponent **implements** Adjustable, Accessible

Commonly used Constructors:

|  |  |
| --- | --- |
| **Constructor** | **Description** |
| JScrollBar() | Creates a vertical scrollbar with the initial values. |
| JScrollBar(int orientation) | Creates a scrollbar with the specified orientation and the initial values. |
| JScrollBar(int orientation, int value, int extent, int min, int max) | Creates a scrollbar with the specified orientation, value, extent, minimum, and maximum. |

import javax.swing.\*;

class ScrollBarExample

{

ScrollBarExample()

{

JFrame f= new JFrame("Scrollbar Example");

JScrollBar s=new JScrollBar();

s.setBounds(100,100, 50,100);

f.add(s);

f.setSize(400,400);

f.setLayout(null);

f.setVisible(true);

}

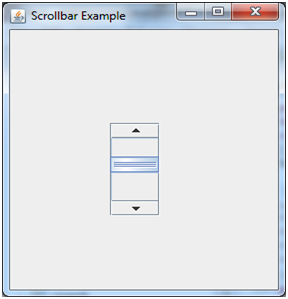
public static void main(String args[])

{

new ScrollBarExample();

}

}



**9. JFrame**

The javax.swing.JFrame class is a type of container which inherits the java.awt.Frame class. JFrame works like the main window where components like labels, buttons, textfields are added to create a GUI.

**import** java.awt.FlowLayout;

**import** javax.swing.JButton;

**import** javax.swing.JFrame;

**import** javax.swing.JLabel;

**import** javax.swing.Jpanel;

**public** **class** JFrameExample

{

**public** **static** **void** main(String s[])

{

 JFrame frame = **new** JFrame("JFrame Example");

        JPanel panel = **new** JPanel();

        panel.setLayout(**new** FlowLayout());

        JLabel label = **new** JLabel("JFrame By Example");

        JButton button = **new** JButton();

        button.setText("Button");

        panel.add(label);

        panel.add(button);

        frame.add(panel);

        frame.setSize(200, 300);

        frame.setLocationRelativeTo(**null**);

        frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

        frame.setVisible(**true**);

    }

}

