SWING PACKAGE

What is JFC : 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.

Swing API is set of extensible GUI Components to ease developer's life to create JAVA based Front End/ GUI Applications. It is build upon top of AWT API and acts as replacement of AWT API as it has almost every control corresponding to AWT controls. Swing component follows a Model-View-Controller architecture to fulfill the following criterias.

- A single API is to be sufficient to support multiple look and feel.
- API is to model driven so that highest level API is not required to have the data.
- API is to use the Java Bean model so that Builder Tools and IDE can provide better services to the developers to use it.

MVC Architecture

Swing API architecture follows loosely based Model View Controller architecture in the following manner.

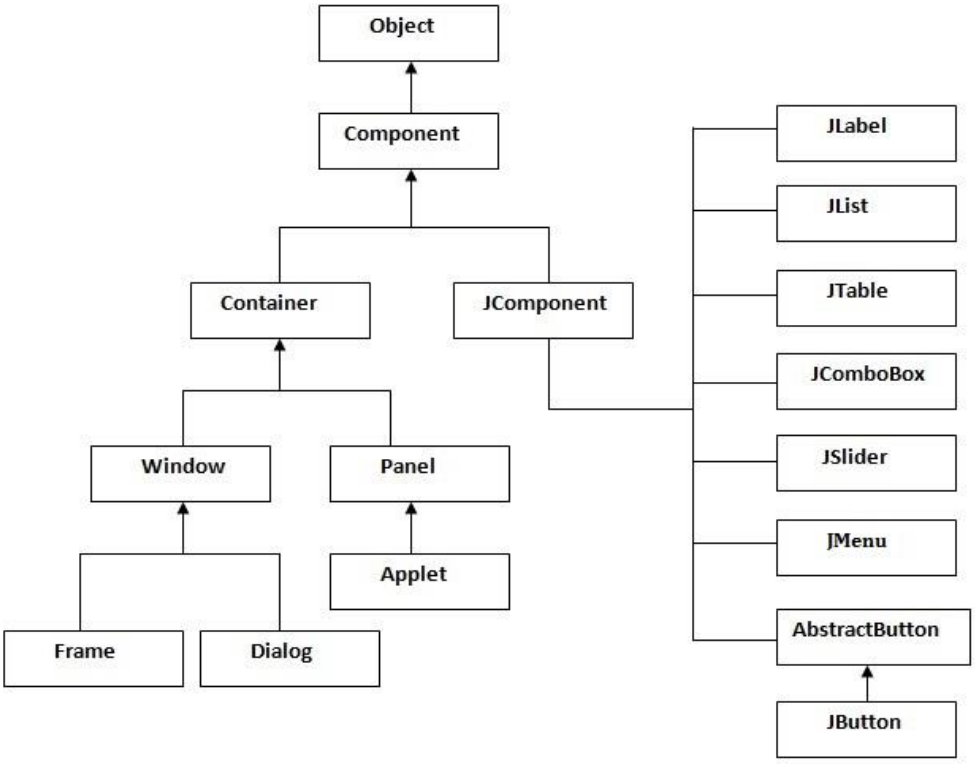
- **Model** : It is a component used to represent value/information/data. E.g Class of javax.swing package.
- **View** : The physical/visual representation of the data present in a model is called view.
- **Controller**: It is a component that takes the input from the user on the view and reflects the changes in Component's data or view.

Swing component have Model as a seperate element and View and Controller part are clubbed in User Interface elements. Using this way, Swing has pluggable look-and-feel architecture.

Swing features

- **Light Weight** - Swing component 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, table controls
- **Highly Customizable** - Swing controls can be customized in 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.

AWT Components	Swing Components
AWT components are platform-dependent.	Java swing components are platform-independent.
AWT components are heavyweight.	Swing components are lightweight.
AWT doesn't support pluggable look and feel.	Swing supports pluggable look and feel.
AWT provides less components than Swing.	Swing provides more powerful components such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
AWT doesn't follows MVC(Model View Controller)	Swing Componets follow MVC architecture.



Class & Description

- **Component** : A Component is the abstract class & base class that represents a graphical user-interface components. It has all the functionality(method) of a gui component.
- **Container** : A Container is a component that provides a rectangular area & it can contain other SWING components like JButton, JTextField, etc.
- **JComponent** : A JComponent is a base class for all swing UI components. In order to use a swing component that inherits from JComponent, component must be in a containment hierarchy whose root is a top-level Swing container.

Every user interface considers the following three main aspects:

- **UI elements :** These are the core visual elements the user eventually sees and interacts with. GWT provides a huge list of widely used and common elements varying from basic to complex which we will cover in this tutorial.
- **Layouts:** They define how UI elements should be organized on the screen and provide a final look and feel to the GUI (Graphical User Interface). This part will be covered in Layout chapter.
- **Behavior:** These are events which occur when the user interacts with UI elements. This part will be covered in Event Handling chapter.

Menu & Button :

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class SimpMenu extends JFrame implements  ActionListener, ItemListener{
JButton b1,b2,b3;
JRadioButton r1, r2;
public SimpMenu() {
```

```
createMenuBar();
setLayout(new FlowLayout());
setTitle("Simple menu");
setMyFont();
b1=new JButton("ok", new ImageIcon("ok.jpg"));
b2=new JButton("submit", new ImageIcon("submit.jpg"));
b3=new JButton("cancel", new ImageIcon("cancel.jpg"));
b1.addActionListener(this);
b2.addActionListener(this);
b3.addActionListener(this);
b1.setRolloverIcon(new ImageIcon("ok1.jpg"));
b2.setRolloverIcon(new ImageIcon("submit1.jpg"));
b3.setRolloverIcon(new ImageIcon("cancel1.jpg"));
b1.setToolTipText("Click Ok button");
b2.setToolTipText("Submit the fields");
b3.setToolTipText("Cancel text of textfield/textarea");
add(b1);        add(b2);
add(b3);

 String country[]={"India","Aus","U.S.A","England","Newzeland"};
JComboBox cb=new JComboBox(country);
add(cb);
JTextArea jt=new JTextArea(10,10);
add(jt);
r1=new JRadioButton("Male");
r2=new JRadioButton("FeMale");
ButtonGroup bg=new ButtonGroup();
bg.add(r1);bg.add(r2);
r1.addItemListener(this);
r2.addItemListener(this);
add(r1); add(r2);

setSize(400, 300);
setLocationRelativeTo(null);
setDefaultCloseOperation(EXIT_ON_CLOSE);
}
private void createMenuBar() {
    ImageIcon icNew = new ImageIcon("new.png");
    ImageIcon icOpn = new ImageIcon("open.png");
    ImageIcon icSav = new ImageIcon("save.jpg");
    ImageIcon icExt = new ImageIcon("exit.png");
JMenuItem ne = new JMenuItem("New", icNew);
 JMenuItem op = new JMenuItem("Open", icOpn);
 JMenuItem sv = new JMenuItem("Save", icSav);
 JMenuItem ex = new JMenuItem("Exit", icExt);

ex.setMnemonic(KeyEvent.VK_E);
ex.setToolTipText("Exit application");
ex.addActionListener((ActionEvent event) -> {
     System.exit(0);
});

JMenu file=new JMenu("File");
file.setMnemonic(KeyEvent.VK_F);
file.add(ne);    file.add(op);    file.add(sv);    file.add(ex);
```

```
 JMenuBar mbar = new JMenuBar();
 mbar.add(file);
 setJMenuBar(mbar);
}
public void actionPerformed(ActionEvent ae){
  if(ae.getActionCommand()=="ok"){
     JOptionPane.showMessageDialog(this,"Ok is clicked");
  }else if(ae.getActionCommand()=="submit"){
     b1.setEnabled(false);
     b2.setVisible(false);
  }else if(ae.getActionCommand()=="cancel") {
     b1.setEnabled(true);
     b2.setVisible(true);
  }
  pack();
}
public void itemStateChanged(ItemEvent e){
  if(r1.isSelected())
     JOptionPane.showMessageDialog(this,"You are male");
  if(r2.isSelected())
    JOptionPane.showMessageDialog(this,"You are female");
}
public static void main(String[] args) {
 SimpMenu sm= new SimpMenu();
 sm.setVisible(true);
}}
```

```
void setMyFont(){
 Font f1=new Font("Arial", Font.BOLD, 24);
 UIManager.put("Button.font", f1);
 UIManager.put("RadioButton.font", f1);
 UIManager.put("ComboBox.font", f1);
 UIManager.put("MenuBar.font", f1);
 UIManager.put("MenuItem.font", f1);
 UIManager.put("Menu.font", f1);
 UIManager.put("TextField.font", f1);
 UIManager.put("PasswordField.font", f1);
 UIManager.put("TextArea.font", f1);
 UIManager.put("ToolTip.font", f1);
}
```

Note: The UIManager class has a method called put, which will accept a String as 1st argument that specifies the type of component on which font will be applied, and 2nd argument specifies the font object. Various types which an be given in 1st argument are given below; Tree.font

Button.font	ToggleButton.font	RadioButton.font	CheckBox.font	ColorChooser.font
ComboBox.font	Label.font	List.font	MenuBar.font	MenuItem.font
RadioButtonMenuItem.font	Menu.font	PopupMenu.font	OptionPane.font	Panel.font
CheckBoxMenuItem.font	ProgressBar.font	ScrollPane.font	Viewport.font	TabbedPane.font
Table.font	TableHeader.font	TextField.font	PasswordField.font	TextArea.font
TextPane.font	EditorPane.font	TitledBorder.font	ToolBar.font	ToolTip.fon

JTable class : It is used to display the data on two dimensional tables of cells.

JTable(): creates a table with empty cells.

JTable(Object[][] rows, Object[] columns): creates a table with the specified data.

```
import java.awt.*;
import javax.swing.*;
import javax.swing.table.*;
class Table extends JFrame{
```

```
 jt.setBackground(Color.orange);
 jt.setForeground(Color.blue);
 jt.setRowHeight(28);
jt.setFont(new Font("Arial", Font.BOLD, 20));
```

```
public Table(){
 setTitle("Table Demo");
 String data[][]={ {"101","Sachin","670000"},
                   {"102","Sehwag","780000"},
                   {"103","Kohli","700000"},
                   {"104","Rohit","780000"},
                   {"105","Dhoni","700000"},
                   {"106","Yuvraj","780000"},
                   {"107","Ashwini","700000"},
                   {"108","Jadeja","780000"},
                   {"109","Zaheer","700000"}};
 String column[]={"ID","NAME","SALARY"};
 JTable jt=new JTable( data, column);
 JTableHeader jth = jt.getTableHeader();
 jth.setFont(new Font("Dialog", Font.BOLD, 22));
 JScrollPane sp=new JScrollPane(jt);
 add("Center", sp);
 setSize(440,350);
 setVisible(true);
 setLocationRelativeTo(null);
}
public static void main(String[] args) {
  Table tab=new Table();
  tab.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}}
```

JColorChooser class : It is used to create a color chooser dialog box so that user can select any color.

JColorChooser(): is used to create a color chooser pane with white color initially.

JColorChooser(Color initialColor): is used to create a color chooser pane with the specified color initially.

public static Color showDialog(Component c, String title, Color initialColor) : is used to show the color-chooser dialog box & returns the object of the color.

```
import java.awt.event.*;
import java.awt.*;
import javax.swing.*;
public class ColorChoose extends JFrame
implements ActionListener{
JButton b;
Container c;
ColorChoose(){
  c=getContentPane();
  c.setLayout(new FlowLayout());
  b=new JButton("color");
  b.addActionListener(this);
  c.add(b);
}
public void actionPerformed(ActionEvent e) {
  Color initialcolor=Color.RED;
  Color color=JColorChooser.showDialog(this,"Select a
color",initialcolor);
  c.setBackground(color);
}
public static void main(String[] args) {
  ColorChoose ch=new ColorChoose();
  ch.setSize(400,400);
  ch.setVisible(true);
  ch.setDefaultCloseOperation(EXIT_ON_CLOSE);
}}
```

JProgressBar class : It is used to display the progress of the task.

- **JProgressBar():** is used to create a horizontal progress bar but no string text.
- **JProgressBar(int min, int max):** is used to create a horizontal progress bar with the specified minimum and maximum value.
- **JProgressBar(int orient):** is used to create a progress bar with the specified orientation, it can be either Vertical or Horizontal by using SwingConstants.VERTICAL and SwingConstants.HORIZONTAL constants.
- **JProgressBar(int orient, int min, int max):** is used to create a progress bar with the specified orientation, minimum and maximum value.

public void setStringPainted(boolean b): is used to determine whether string should be displayed.

public void setString(String s): is used to set value to the progress string.

public void setOrientation(int orientation): is used to set the orientation, it may be either vertical or horizontal by using SwingConstants.VERTICAL and SwingConstants.HORIZONTAL constants..

public void setValue(int value): is used to set the current value on the progress bar.

```
import javax.swing.*;
public class MyProgress extends JFrame{
JProgressBar jb;
int i=0,num=0;
MyProgress(){
 jb=new JProgressBar(0,2000);
 jb.setBounds(40,40,200,30);
 jb.setValue(0);
 jb.setStringPainted(true);
 add(jb);
 setSize(400,400);
 setLayout(null);
 setVisible(true);
}
```

```
public void iterate(){
 while(i<=2000){
  jb.setValue(i);
  i=i+20;
  try{
    Thread.sleep(150);
  }catch(Exception e){ }
 }
}
public static void main(String[] args) {
 MyProgress m=new MyProgress();
  m.iterate();
}}
```

Using Methods of Graphics class:

```
import java.awt.*;
import javax.swing.JFrame;
public class Shape extends Canvas{
public void paint(Graphics g) {
  g.drawString("Hello",40,40);
  setBackground(Color.WHITE);
  g.fillRect(130, 30,100, 80);
  g.drawOval(30,130,50, 60);
  setForeground(Color.RED);
  g.fillOval(130,130,50, 60);
 g.drawArc(30, 200, 40,50,90,60);
 g.fillArc(30, 130, 40,50,180,40);
```

```
Toolkit t=Toolkit.getDefaultToolkit();
 Image i=t.getImage("process.png");
  g.drawImage(i, 120,220,this);
}
public static void main(String[] args) {
  Shape m=new Shape();
  JFrame f=new JFrame();
  f.add(m);
  f.setSize(400,400);
  f.setVisible(true);
}}
```

JSlider class : It is used to create the slider. By using JSlider we can select a value from a specific range.

JSlider(): creates a slider with the initial value of 50 and range of 0 to 100.

JSlider(int orientation): creates a slider with the specified orientation set by either JSlider.HORIZONTAL or JSlider.VERTICAL with the range 0 to 100 and initial value 50.

JSlider(int min, int max): creates a horizontal slider using the given min and max.

JSlider(int min, int max, int value): creates a horizontal slider using the given min, max and value.

JSlider(int orientation, int min, int max, int value): creates a slider using the given orientation, min, max and value.

void setMinorTickSpacing(int n): is used to set the minor tick spacing to the slider.

void setMajorTickSpacing(int n): is used to set the major tick spacing to the slider.

void setPaintTicks(boolean b): is used to determine whether tick marks are painted.

void setPaintLabels(boolean b): is used to determine whether labels are painted.

void setPaintTracks(boolean b): is used to determine whether track is painted.

```
import java.awt.*;
import javax.swing.*;
class Slider extends JFrame{
public Slider() {
  JSlider slider = new JSlider(JSlider.HORIZONTAL,
0, 50, 25);
  slider.setMinorTickSpacing(2);
  slider.setMajorTickSpacing(10);
  slider.setPaintTicks(true);
  slider.setPaintLabels(true);
  // We'll just use the standard numeric labels for now...
  slider.setLabelTable(slider.createStandardLabels(10));
  add(slider, BorderLayout.CENTER);
}
public static void main(String s[]) {
 Slider frame=new Slider();
 frame.pack();
 frame.setVisible(true);
}}
```

JTree class : JTree Class : It is used to display hierarchical data. A JTree has a 'root node' which is the top-most parent for all nodes in the tree. A node is a leaf/node in a tree. A node can have many children nodes. These children nodes themselves can have further children nodes. If a node doesn't have any children node, it is called a leaf node.

The node is represented by TreeNode interface. The interface MutableTreeNode extends TreeNode interface which represents a mutable node. An implementation of this interface in the form of DefaultMutableTreeNode class.

```
import java.awt.*;
import javax.swing.*;
import javax.swing.tree.*;
class Jtre extends JFrame{
JTree  tr;
Jtre(){
setBackground(Color.cyan);
DefaultMutableTreeNode t1, t2, t3, t4, t5, t6, t7, t8, t9,
t10, t11 ,t12, t13;
t1=new DefaultMutableTreeNode("Fruit");
t2=new DefaultMutableTreeNode("Raw");
t3=new DefaultMutableTreeNode("Ripe");
t4=new DefaultMutableTreeNode("Cooked");
t5=new DefaultMutableTreeNode("apple");
t6=new DefaultMutableTreeNode("Cucumber");
t7=new DefaultMutableTreeNode("Coconut");
t8=new DefaultMutableTreeNode("Mango");
t9=new DefaultMutableTreeNode("Jadck Fruit");
t10=new DefaultMutableTreeNode("Pineapple");
t11=new DefaultMutableTreeNode("Pumpkin");
t12=new DefaultMutableTreeNode("Brinjal");
t13=new DefaultMutableTreeNode("Patato");
t2.add(t5);     t2.add(t6);     t2.add(t7);
t3.add(t8);     t3.add(t9);     t3.add(t10);
t4.add(t11);    t4.add(t12);    t4.add(t13);
t1.add(t2);     t1.add(t3);     t1.add(t4);

tr=new JTree(t1);
tr.setFont(new Font("asdf",Font.BOLD,40));
add("Center", tr);
setSize(700,500);
setVisible(true);
}
public static void main(String s[]){
new Jtre();
}}
```

Creating Notepad :

```
import javax.swing.*;
import java.awt.event.*;
public class MyNote implements ActionListener{
        JFrame f;
        JMenuBar mb;
        JMenu file,edit,help;
```

```
        JMenuItem cut,copy,paste,selectAll;
        JTextArea ta;
        MyNote(){
        f=new JFrame();
          cut=new JMenuItem("cut");                  file=new JMenu("File");
          copy=new JMenuItem("copy");                edit=new JMenu("Edit");
          paste=new JMenuItem("paste");              help=new JMenu("Help");
          selectAll=new JMenuItem("selectAll");      edit.add(cut);    edit.add(copy);

          cut.addActionListener(this);               edit.add(paste);edit.add(selectAll);
          copy.addActionListener(this);              mb.add(file);     mb.add(edit);    mb.add(help);
          paste.addActionListener(this);
          selectAll.addActionListener(this);         ta=new JTextArea();
          mb=new JMenuBar();                         ta.setBounds(5,30,460,460);
          mb.setBounds(5,5,400,40);

        f.add(mb);        f.add(ta);
        f.setLayout(null);
        f.setSize(500,500);
        f.setVisible(true);
}
public void actionPerformed(ActionEvent e) {
        if(e.getSource()==cut)
        ta.cut();
        if(e.getSource()==paste)
        ta.paste();
        if(e.getSource()==copy)
        ta.copy();
        if(e.getSource()==selectAll)
        ta.selectAll();
}
public static void main(String[] args) {
   new MyNote();
}}
```

https://www.tutorialspoint.com/swing/swing_controls.htm
http://www.javatpoint.com/java-swing
http://zetcode.com/tutorials/javaswingtutorial/introduction/
http://www.java2s.com/Tutorial/Java/0240__Swing/SetFontandforegroundcolorforaJLabel.htm for ALL
http://www.javatpoint.com/JSlider-class

Asdfsadfdsa