5.User Interface With AWT and Swing

There are multiple ways to develop GUI-based applications in java, out of which the most popular ones are AWT and Swing.

1. AWT

AWT stands for Abstract Window Toolkit. It is a platform-dependent API to develop GUI (Graphical User Interface) or window-based applications in Java. It was developed by heavily Sun Microsystems In 1995. It is heavy-weight in use because it. It contains a large number of classes and methods, which are used for creating and managing GUI.

2. Swing:

Swing is a lightweight Java graphical user interface (GUI) that is used to create various applications. Swing has platform-independent components. It enables the user to create buttons and scroll bars. Swing includes packages for creating desktop applications in Java. Swing components are written in Java language.

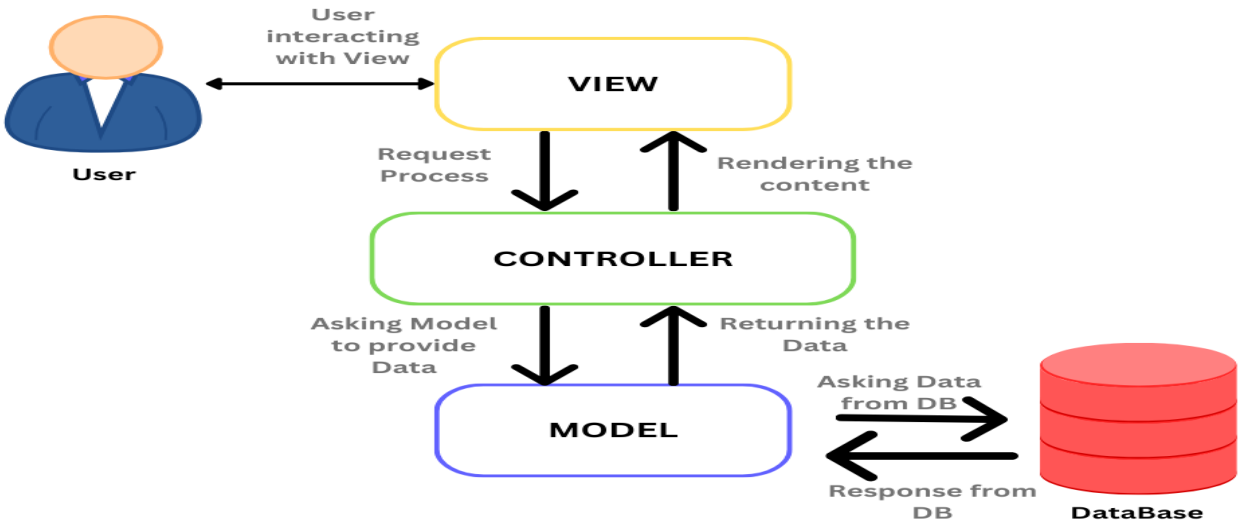
Difference between AWT and Swing

AWT	Swing
Java AWT is an API to develop GUI applications in Java	Swing is a part of Java Foundation Classes and is used to create various applications.
The components of Java AWT are heavy weighted.	The components of Java Swing are light weighted.
Java AWT has comparatively less functionality as compared to Swing.	Java Swing has more functionality as compared to AWT.
The execution time of AWT is more than Swing.	The execution time of Swing is less than AWT.
The components of Java AWT are platform dependent.	The components of Java Swing are platform independent.
MVC pattern is not supported by AWT.	MVC pattern is supported by Swing.
AWT provides comparatively less powerful components.	Swing provides more powerful components.

AWT	Swing
AWT components require java.awt package	Swing components requires javax.swing package
AWT stands for Abstract windows toolkit .	Swing is also called as JFC(java Foundation classes). It is part of oracle's JFC.
Using AWT , you have to implement a lot of things yourself .	Swing has them built in.

MVC Architecture and swing

MVC (Model-View-Controller) architecture is a universal pattern of a structure in which an application is divided into three parts which are all dedicated to certain parts of the whole application. This pattern is normally used in software development to create organized and easy-to-maintain code.



- **Model:** Model stands as the data layer for the application. It is directly involved in managing the data as well as the control of the application's logic and rules.
- **View:** The View is in the presentation tier. It plays a role of presenting the information given by the Model to the user and transferring the user commands to the Controller. The View is used to display the data to the user in a readable and manageable way using the interface created by the Controller.
- **Controller:** The Controller works in the middle between the Model and the View. It takes the input from the View, sometimes modifies it with the help of the Model, and sends it back to the View. the results back to the View.

Layout Managers

A container can hold one or more components.It has responsibility to arranging components on a container like frame or a panel.

The Layout managers enable us to control the way in which visual components are arranged in the GUI forms by determining the size and position of components within the containers.

Setting the layout manager

You can install a new layout manager at any time by using the setLayout() method.

Syntax:

setLayout(LayoutManager obj)

Example:

setLayout(new FlowLayout());

1. Flow Layout

This layout will display the components from left to right, from top to bottom. The components will always be displayed in the first line and if the first line is filled, these components are displayed on the next line automatically.

In this Layout Manager, initially, the container assumes 1 row and 1 column of the window. Depending on the number of components and size of the window, the number of rows and columns count is decided dynamically.

Creation of Flow Layout
FlowLayout f1 = new FlowLayout();
FlowLayout f1 = new FlowLayout(int align);
FlowLayout f1 = new FlowLayout(int align, int hgap, int vgap);

2. Border Layout
This layout will display the components along the border of the container. This layout contains five locations where the component can be displayed. Locations are North, South, East, west, and Center. The default region is the center.

Creation of BorderLayout
BorderLayout bl = new BorderLayout();
BorderLayout bl = new BorderLayout(int vgap, int hgap);

3. Grid Layout

The layout will display the components in the format of rows and columns statically. The container will be divided into a table of rows and columns. The intersection of a row and column cell and every cell contains only one component, and all the cells are of equal size. According to Grid Layout Manager, the grid cannot be empty.

Creation of Grid Layout Manager in Java

GridLayout gl = new GridLayout(int rows, int cols);

GridLayout gl = new GridLayout(int rows, int cols, int vgap, int hgap);

4. Box Layout

The BoxLayout class is used to arrange the components either vertically (along Y-axis) or horizontally (along X-axis). In BoxLayout class, the components are put either in a single row or a single column.

BoxLayout(Container c, int axis): Creates a BoxLayout class that arranges the components with the X-axis or Y-axis.

5. Card Layout

A card layout represents a stack of cards displayed on a container. At a time, only one card can be visible, each containing only one component.

Creation of Card Layout in Java

CardLayout cl = new CardLayout();

CardLayout cl = new CardLayout(int hgap, int vgap);

To add the components in CardLayout, we use the add method:

add(“Cardname”, Component);

Methods of CardLayout in Java

1. **first(Container):** It is used to flip to the first card of the given container.
2. **last(Container):** It is used to flip to the last card of the given container.
3. **next(Container):** It is used to flip to the next card of the given container.
4. **previous(Container):** It is used to flip to the previous card of the given container.
5. **show(Container, cardname):** It is used to flip to the specified card with the given name.

6. **Grid Bag Layout**

In GridLayout manager, there is no grid control, i.e., inside a grid, we cannot align the component in a specific position. To overcome this problem, we have an advanced Layout Manager, i.e., Grid Bag Layout Manager. This layout is the most efficient layout that can be used for displaying components. In this layout, we can specify the location, size, etc. In this Layout manager, we need to define grid properties or constraints for each grid. Based on grid properties, the layout manager aligns a component on the grid, and we can also span multiple grids as per the requirement. Gris properties are defined using the GridBagConstraints class.

Creation of GridBagLayout: GridBagLayout gbl = new GridBagLayout();

Properties of GridBagConstraints:

1. **gridx, gridy:** For defining x and y coordinate values, i.e., specifying grid location.
2. **gridwidth, grid height:** For defining the number of grids to span a document.
3. **fill:** Used whenever component size is greater than the area (i.e., VERTICAL or HORIZONTAL).

4. **ipadx, ipady:** For defining the width and height of the components, i.e., for increasing component size.
5. **insets:** For defining the surrounding space of the component, i.e., top, left, right, bottom.
6. **anchor:** Used whenever component size is smaller than area, i.e., where to place in a grid.

7. **weightx, weighty: T**hese are used to determine how to distribute space among columns(weightx) and among rows(weighty), which is important for specifying resizing behavior.

Insets

In Java, Insets objects define the padding or space around a component, specifying the amount of space (in pixels) to leave at each edge (top, left, bottom, and right).

Insets(int top,int left,int bottom,int right)

Containers and Components

1.Containers classes

The Container is a component that will be used to extend other components such as AWT Container classes-window, panel, Frame, Dialog, and Applet.

Swing container classes-JFrame,JPanel,JDialog.

- **Window:** The Window is a Container that doesn't include borders and a menu bar.
- **Panel:** The Panel is also a Container that doesn't include a title bar, menu, or border. It is a container that holds components like buttons, textfield, etc.
- **Frame:** The Frame is a container used while creating an AWT application. It can have components like title bar, menu bars, borders and also buttons, scroll bar, etc.
- **Dialog:** The Dialog box is a container that will display the message that we want to display on the screen.

2.Component classes

The Component class is the superclass of all components. A component class can be linked with a page, components of web applications.

Types of Components in Component Class

The components in a component class are as follows :

1. **Container**
2. **Button**
3. **Label**
4. **Checkbox**
5. **Choice**
6. **List**

1.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.

Constructors

Constructor	Description
JFrame()	It constructs a new frame that is initially invisible.
JFrame(String title)	It creates a new, initially invisible Frame with the specified title.

Methods:

setTitle(String title)-Sets the title of the JFrame.
setVisible(boolean b)-Sets the visibility of the JFrame.
setSize(int width, int height)-Sets the size of the JFrame to the specified width and height.

Example:

```
public class MyJFrame {
    // main function
    public static void main(String[] args)
    {
        // Create a new JFrame
        JFrame frame = new JFrame("My First
JFrame");
        frame.setSize(300,
                    200); // Set the size of the frame
        // Make the frame visible
        frame.setVisible(true);
    }
}
```

Adding and removing components
add(),remove(),removeAll()

Syntax: add(Components obj)
remove(Component obj)

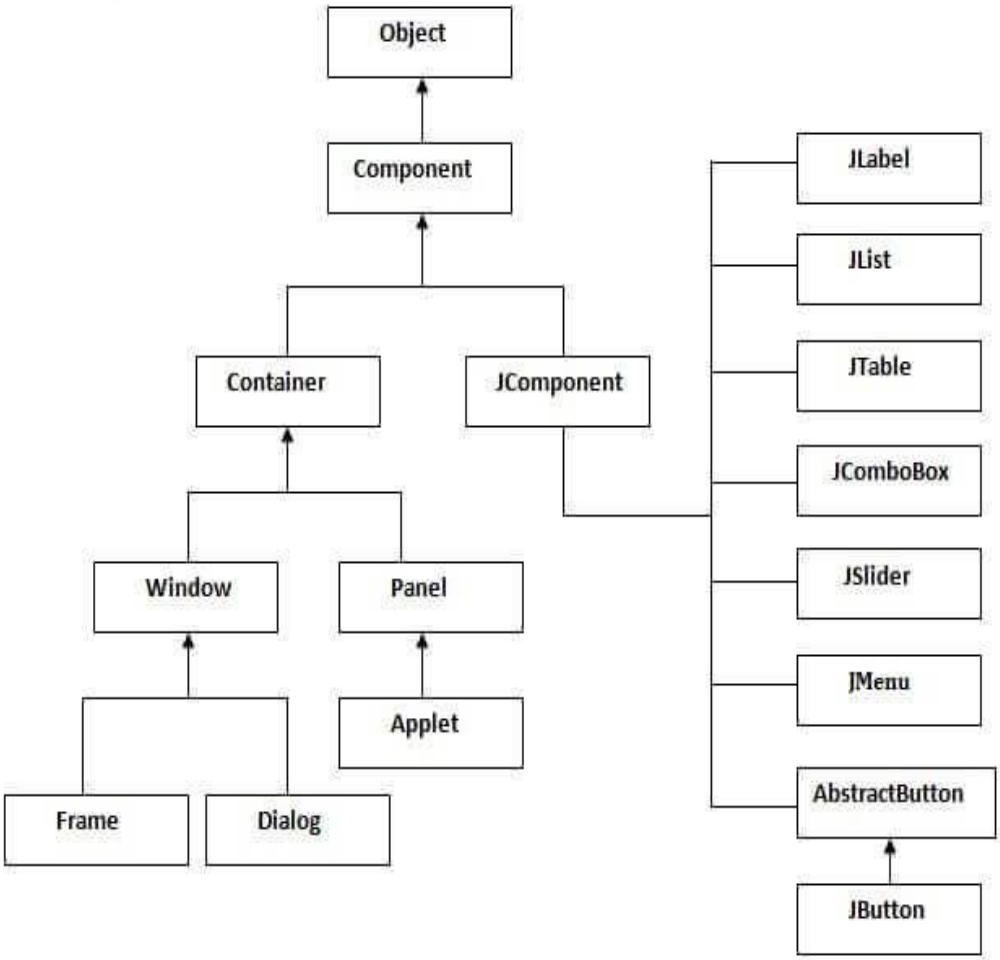
Example:

```
public class MyJFrame {
   public static void main(String[] args)
   {
     JFrame f = new JFrame("My First JFrame");
f.setSize(300, 200);
       f.setVisible(true);

JButton b=new JButton(“submit”);
f.add(b);
   }
}
```

2.JComponent

The JComponent class is the base class of all Swing components. Swing components whose names begin with "J". For example, JButton, JScrollPane, JPanel, JTable etc.



1. JButton

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

```
JButton b=new JButton("Click Here");
f.add(b);
f.setSize(400,400);
f.setLayout(null);
f.setVisible(true);
```

JButton b=new JButton(new ImageIcon("D:\\icon.png"));

2. Jlabel

```
Eg JFrame f= new JFrame("Label Example");
    JLabel l1,l2;
    l1=new JLabel("First Label.");
    l1.setBounds(50,50, 100,30);
    l2=new JLabel("Second Label.");
    l2.setBounds(50,100, 100,30);
    f.add(l1); f.add(l2);
    f.setSize(300,300);
    f.setLayout(null);
    f.setVisible(true);
```

3. JTextfield

```
Eg  JFrame f= new JFrame("TextField
Example");
        JTextField t1, t2;
        t1 = new JTextField("Welcome to
Javatpoint.");
        t1.setBounds(50,100, 200,30);
        t2 = new JTextField("AWT Tutorial");
        t2.setBounds(50,150, 200,30);
        f.add(t1);
        f.add(t2);
        f.setSize(400,400);
        f.setLayout(null);   f.setVisible(true);
```

4. JTextArea

```
Eg  JFrame f= new JFrame();
        JTextArea area=new JTextArea("Welcome
to javatpoint");
        area.setBounds(10,30, 200,200);
        f.add(area);
        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);
```

5. JCheckbox

```
Eg  JFrame f= new JFrame("CheckBox
Example");
        JCheckBox checkBox1 = new
JCheckBox("C++");
        checkBox1.setBounds(100,100, 50,50);
        f.add(checkBox1);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
```

6. JRadioButton

```
Eg JFrame 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);
ButtonGroup bg=new ButtonGroup();
bg.add(r1);bg.add(r2);
f.add(r1);f.add(r2);
f.setSize(300,300);
f.setLayout(null);
f.setVisible(true);

7. JList
   Eg
   JFrame f=new JFrame();
   String [] arr={“java”,”c”,”c++”};
   JlList l=new JList(arr);

8. JComboBox
   Eg
   JFrame f=new JFrame("ComboBox Example");
   String country[]={"India","Aus","U.S.A","England","Newzealand"};
   JComboBox cb=new JComboBox(country);
   cb.setBounds(50, 50,90,20);

```
f.add(cb);
f.setLayout(null);
f.setSize(400,500);
f.setVisible(true);
```

9. JMenu

Eg

```
JFrame f= new JFrame("Menu and MenuItem Example");
JMenuBar mb=new JMenuBar();
JMenu menu=new JMenu("Menu");
JMenu submenu=new JMenu("Sub Menu");
JMenuItem  i1=new JMenuItem("Item 1");
JMenuItem i2=new JMenuItem("Item 2");
JMenuItem i3=new JMenuItem("Item 3");
JMenuItem i4=new JMenuItem("Item 4");
JMenuItem i5=new JMenuItem("Item 5");
menu.add(i1); menu.add(i2); menu.add(i3);
submenu.add(i4); submenu.add(i5);
menu.add(submenu);
mb.add(menu);
f.setJMenuBar(mb);
f.setSize(400,400);
f.setLayout(null);
f.setVisible(true);
```