



User Interface Components with Swing

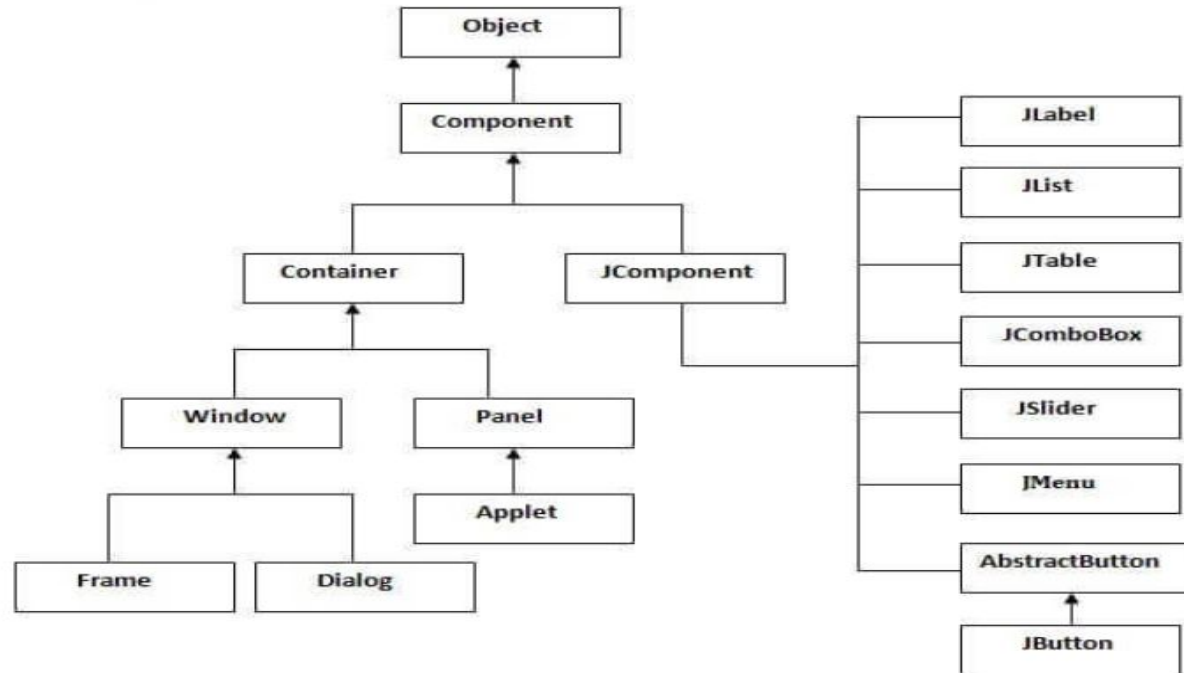
Prepared by: Aakash Raj Shakya




Swing in Java

1. Swing in java is part of Java foundation class which is lightweight and platform independent.
2. It is used for creating window based applications.
3. It includes components like button, scroll bar, text field etc.
4. It is a lightweight GUI toolkit which has a wide variety of widgets for building optimized window based applications.
5. It is platform independent unlike AWT and has lightweight components.
6. The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

Hierarchy of Java Swing classes





All the components in swing like JButton, JComboBox, JList, JLabel are inherited from the JComponent class which can be added to the container classes.

Containers are the windows like frame and dialog boxes.

Basic swing components are the building blocks of any GUI application.

Methods like setLayout override the default layout in each container.

Containers like JFrame and JDialog can only add a component to itself.



The Model-View-Controller Architecture

Swing uses the model-view-controller architecture (MVC) as the fundamental design behind each of its components.

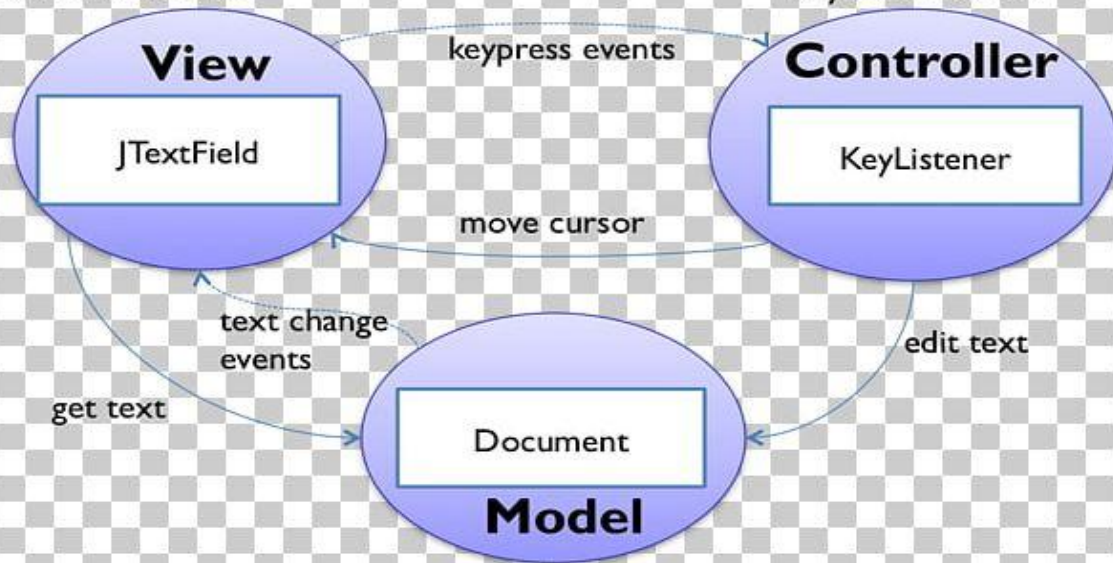
Model - Contains all the business logic and access to data layer.

View - It is the representation of what we see (or what a user sees).

Controller - Mediator to send and receive information.

UITextField is a JComponent that can be added to a view tree

KeyListener is a listener for keyboard events



Document represents a mutable string of characters



Layout Management

1. 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.
2. It is an interface that is implemented by all classes if layout managers and determines the size and position of the components within a container.
3. Although components can provide size and alignment hints, a container's layout manager has the final say on the size and position of the components within the container.




Types of Layout managers

1. BorderLayout
2. GridLayout
3. GridBagLayout
4. CardLayout
5. FlowLayout
6. GroupLayout



BorderLayout

1. The BorderLayout is used to arrange the components in five regions: north, south, east, west and center.
2. Each region (area) may contain one component only.
3. It is the default layout of frame or window.
4. The BorderLayout provides five constants for each region:
 - a. `public static final int NORTH`
 - b. `public static final int SOUTH`
 - c. `public static final int EAST`
 - d. `public static final int WEST`
 - e. `public static final int CENTER`



North		
West	Center	East
South		



GridLayout

1. The GridLayout is used to arrange the components in rectangular grid.
2. One component is displayed in each rectangle.
3. Constructors of GridLayout class
 - a. **GridLayout()**: creates a grid layout with one column per component in a row.
 - b. **GridLayout(int rows, int columns)**: creates a grid layout with the given rows and columns but no gaps between the components.
 - c. **GridLayout(int rows, int columns, int hgap, int vgap)**: creates a grid layout with the given rows and columns along with given horizontal and vertical gaps.



```
public class MyGridLayout {
    private static JFrame frame;

    public static void main(String[] args) {
        frame = new JFrame("List");

        JButton b1=new JButton("Button 1");
        JButton b2=new JButton("Button 2");
        JButton b3=new JButton("Button 3");
        JButton b4=new JButton("Button 4");
        JButton b5=new JButton("Button 5");
        JButton b6=new JButton("Button 6");

        frame.add(b1);
        frame.add(b2);
        frame.add(b3);
        frame.add(b4);
        frame.add(b4);
        frame.add(b5);
        frame.add(b6);

        frame.setLayout(new GridLayout(3, 3));

        frame.setVisible(true);
        frame.setSize(500, 500);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```






GridBagLayout

1. The Java GridBagLayout class is used to align components vertically, horizontally or along their baseline.
2. The components may not be of same size.
3. It is one of the most flexible layout manager but one of the trickiest to use.
4. Each GridBagLayout object maintains a dynamic, rectangular grid of cells.
5. Each component occupies one or more cells known as its display area.
6. With the help of constraints object we arrange component's display area on the grid.
7. The GridBagLayout manages each component's minimum and preferred sizes in order to determine component's size.
8. Each component managed by a GridBagLayout is associated with an instance of GridBagConstraints.



```
public class GridBagLayoutDemo {  
    public static void main(String[] args) {  
        JFrame frame = new JFrame("Grid bag layout example");  
  
        GridBagLayout bagLayout = new GridBagLayout();  
        frame.setLayout(bagLayout); //Note if we don't add the layout here then adding constraints below will throw an exception  
  
        GridBagConstraints constraints = new GridBagConstraints();  
  
        constraints.fill = GridBagConstraints.HORIZONTAL;  
  
        constraints.gridx = 0;  
        constraints.gridy = 0;  
        frame.add(new JButton("First Button"), constraints);  
  
        constraints.gridx = 1;  
        constraints.gridy = 0;  
        frame.add(new JButton("Second Button"), constraints);  
  
        constraints.ipady = 10; //Gives padding vertical padding to the component  
        constraints.gridx = 0;  
        constraints.gridy = 1;  
        frame.add(new JButton("Third Button"), constraints);  
  
        constraints.gridx = 1;  
        constraints.gridy = 1;  
        frame.add(new JButton("Fourth Button"), constraints);  
    }  
}
```

```
constraints.ipady = 20;
constraints.gridx = 0;
constraints.gridy = 2;
constraints.gridwidth = 2;
frame.add(new JButton("Fifth Button"), constraints);

frame.setVisible(true);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setSize(500, 600);
}
```



CardLayout

1. The CardLayout class manages the components in such a manner that only one component is visible at a time.
2. It treats each component as a card that is why it is known as CardLayout.
3. The first component added to a CardLayout object is the visible component when the container is first displayed.
4. The ordering of cards is determined by the container's own internal ordering of its component objects.



FlowLayout

The Java FlowLayout class is used to arrange the components in a line, one after another (in a flow). It is the default layout of the applet or panel.

Fields of FlowLayout class

1. `public static final int LEFT`
2. `public static final int RIGHT`
3. `public static final int CENTER`
4. `public static final int LEADING`
5. `public static final int TRAILING`



Constructors of `FlowLayout` class

1. **`FlowLayout()`**: creates a flow layout with centered alignment and a default 5 unit horizontal and vertical gap.
2. **`FlowLayout(int align)`**: creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap.
3. **`FlowLayout(int align, int hgap, int vgap)`**: creates a flow layout with the given alignment and the given horizontal and vertical gap.

```
import java.awt.*;
import javax.swing.*;

public class FlowLayoutExample {

    JFrame frameObj;

    // constructor
    FlowLayoutExample() {
        // creating a frame object
        frameObj = new JFrame();

        // creating the buttons
        JButton b1 = new JButton("1");
        JButton b2 = new JButton("2");
        JButton b3 = new JButton("3");
        JButton b4 = new JButton("4");
        JButton b5 = new JButton("5");
        JButton b6 = new JButton("6");
        JButton b7 = new JButton("7");
        JButton b8 = new JButton("8");
        JButton b9 = new JButton("9");
        JButton b10 = new JButton("10");

        // adding the buttons to frame
        frameObj.add(b1);
        frameObj.add(b2);
        frameObj.add(b3);
        frameObj.add(b4);
        frameObj.add(b5);
        frameObj.add(b6);
        frameObj.add(b7);
        frameObj.add(b8);
```

```
frameObj.add(b9);
frameObj.add(b10);

// parameter less constructor is used
// therefore, alignment is center
// horizontal as well as the vertical gap is 5 units.
frameObj.setLayout(new FlowLayout());

frameObj.setSize(500, 500);
frameObj.setVisible(true);
frameObj.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
}

// main method
public static void main(String args[]) {
    new FlowLayoutExample();
}
}
```



1

2

3

4

5

6

7

8

9

10



GroupLayout

- The class GroupLayout hierarchically groups the components in order to position them in a Container.
- Group is an abstract class, and two concrete classes which implement this Group class are SequentialGroup and ParallelGroup.
- SequentialGroup positions its child sequentially one after another whereas ParallelGroup aligns its child on top of each other.
- The GroupLayout class provides methods such as createParallelGroup() and createSequentialGroup() to create groups.

Class Constructor

GroupLayout(Container host)


```
import java.awt.*;
import javax.swing.*;

public class GroupLayoutExample {

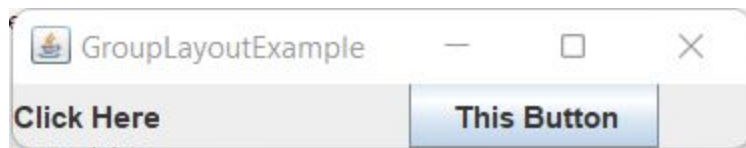
    public static void main(String[] args) {
        JFrame frame = new JFrame("GroupLayoutExample");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        Container contentPanel = frame.getContentPane();
        GroupLayout groupLayout = new GroupLayout(contentPanel);

        contentPanel.setLayout(groupLayout);

        JLabel clickMe = new JLabel("Click Here");
        JButton button = new JButton("This Button");

        groupLayout.setHorizontalGroup(
            groupLayout.createSequentialGroup()
                .addComponent(clickMe)
                .addGap(10, 20, 100)
                .addComponent(button));
        groupLayout.setVerticalGroup(
            groupLayout.createParallelGroup(GroupLayout.Alignment.BASELINE)
                .addComponent(clickMe)
                .addComponent(button));

        frame.pack();
        frame.setVisible(true);
    }
}
```





Using Menus with Frames

Menus are an integral part of GUIs.

They allow the user to perform actions without unnecessarily cluttering a GUI with extra components. In Swing GUIs, menus can be attached only to objects of the classes that provide method **setJMenuBar**. Two such classes are **JFrame** and **JApplet**.

The classes used to declare menus are **JMenuBar**, **JMenu**, **JMenuItem**, **JCheckBoxMenuItem** and **JRadioButtonMenuItem**.



Class JMenuBar (a subclass of JComponent) contains the methods necessary to manage a menu bar, which is a container for menus.

Class JMenu (a subclass of javax.swing.JMenuItem) contains the methods necessary for managing menus. Menus contain menu items and are added to menu bars or to other menus as submenus. When a menu is clicked, it expands to show its list of menu items.

Class JMenuItem (a subclass of javax.swing.AbstractButton) contains the methods necessary to manage menu items. A menu item is a GUI component inside a menu that, when selected, causes an action event. A menu item can be used to initiate an action, or it can be a submenu that provides more menu items from which the user can select. Submenus are useful for grouping related menu items in a menu.

Class JCheckBoxMenuItem (a subclass of javax.swing.JMenuItem) contains the methods necessary to manage menu items that can be toggled on or off. When a JCheckBoxMenuItem is selected, a check appears to the left of the menu item. When the JCheckBoxMenuItem is selected again, the check is removed.



Class JRadioButtonMenuItem (a subclass of `javax.swing.JMenuItem`) contains the methods necessary to manage menu items that can be toggled on or off like `JCheckBoxMenuItems`. When multiple `JRadioButtonMenuItems` are maintained as part of a `Button-Group`, only one item in the group can be selected at a given time. When a `JRadioButtonMenuItem` is selected, a filled circle appears to the left of the menu item. When another `JRadioButtonMenuItem` is selected, the filled circle of the previously selected menu item is removed.

JPopupMenu:

Many of today's computer applications provide so-called context-sensitive pop-up menus. In Swing, such menus are created with class **JPopupMenu** (a subclass of `JComponent`). These menus provide options that are specific to the component for which the popup trigger event was generated. On most systems, **the pop-up trigger event occurs when the user presses and releases the right mouse button.**