Swing

AWT is used for creating GUI in Java. However, the AWT components are internally depends on native methods like C functions and operating system equivalent and hence Problems related to portability arise (look and feel. Ex. Windows window and MAC window). Also, AWT components are heavy weight. It means AWT components take more system resources like memory and processor time.

Due to this, Java soft people felt it is better to redevelop AWT package without internally taking the help of native methods. Hence all the classes of AWT are extended to form new classes and a new class library is created. This library is called JFC (Java Foundation Classes).

Swing is a framework or API that is used to create GUI (or) window-based applications.It is an advanced version of AWT (Abstract Window 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:

S.N	AWT	Swing
1.	AWT components are platform-dependent.	Swing components are platform-independent.
2.	AWT components are heavyweight.	Swing components are lightweight.
3.	AWT provides less components than Swing.	Swing provides more powerful components such as tables, lists, scrollpanes, colorchooser and` etc.
4.	AWT doesn't follow MVC	Swing follows MVC.

Commonly used Methods of Component class:

Method	Description
add(Component c)	inserts a component on this component.
setSize(int width,int height)	sets the size (width and height) of the component.

setLayout(LayoutManager m)	defines the layout manager for the component.
setVisible(boolean status)	changes the visibility of the component, by default false.
setTitle(String text)	Sets the title for component

Components and Containers:

A Swing GUI consists of two key items: components and containers. However, this distinction is mostly conceptual because all containers are also components.
A container holds a group of components. Thus, a container is a special type of component that is designed to hold other components.Furthermore, in order for a component to be displayed, it must be held within a container.
Thus all Swing GUIs will have at least one container.

Components:

In general, Swing components are derived from the JComponent class. JComponent provides the functionality that is common to all components.
All of Swing's components are represented by classes defined within the package javax.swing.
The following figure shows the hierarchy of classes of javax.swing :

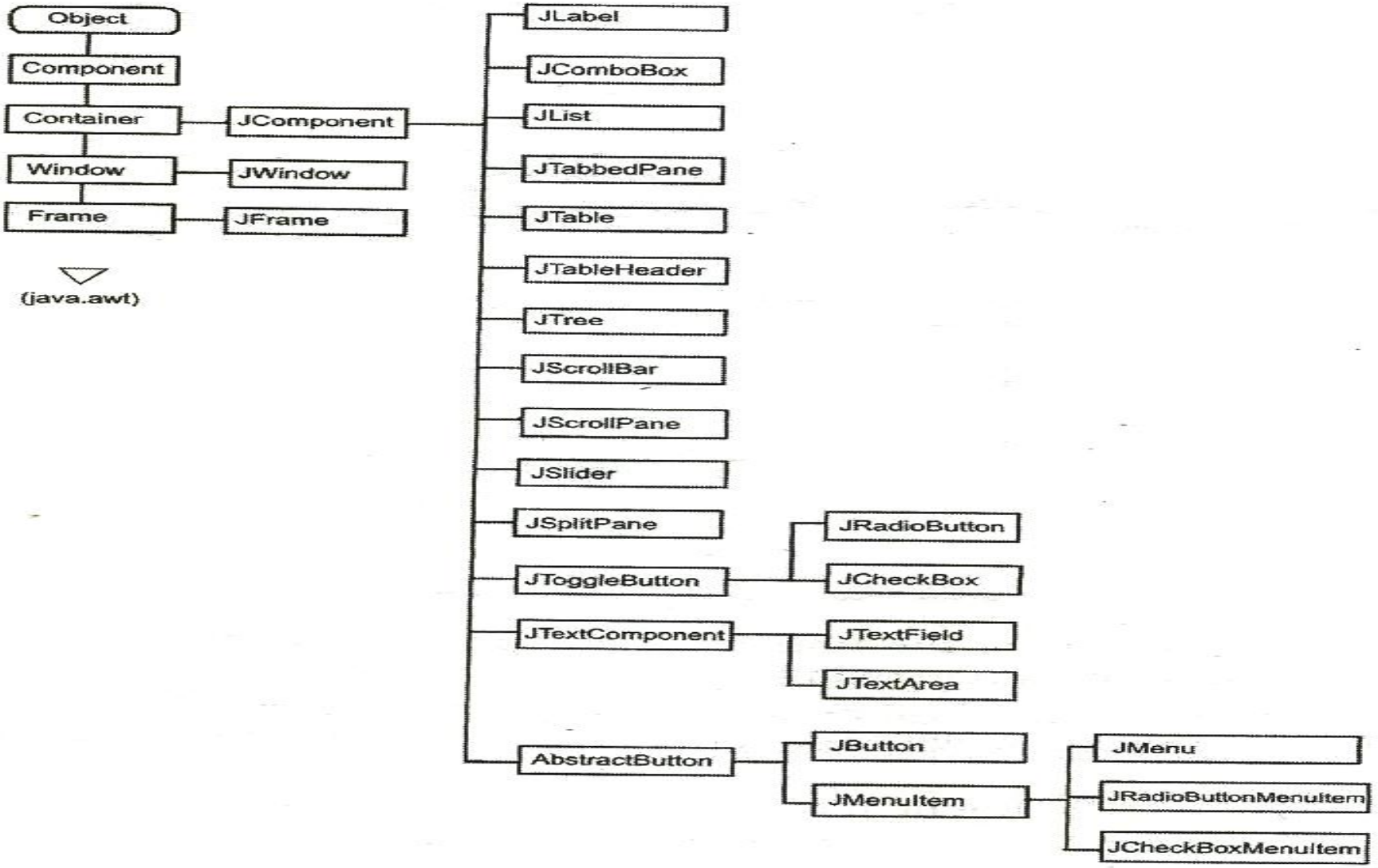
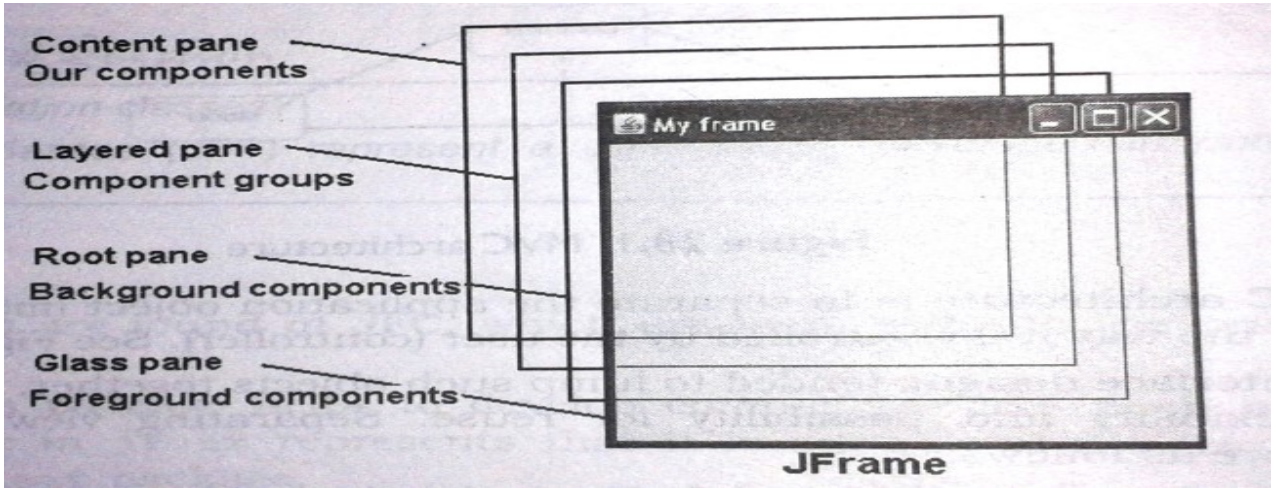


Figure : Hierarchy of classes of javax.swing

Containers:

In Swing, containers are components that can hold other components, such as buttons, labels, text fields, and more. Swing defines two main types of containers:

1. **Top-level containers/ Root containers:** These are the highest-level containers in a Swing application. They are the windows or frames that typically represent the main application window. Examples of top-level containers include JFrame, JDialog, JWindow and JApplet. Whenever we create a top level container four sub-level containers are automatically created:
    1. Glass pane (JGlass)
    2. Root pane (JRootPane)
    3. Layered pane (JLayeredPane)
    4. Content pane



Glass pane: This is the first pane and is very close to the monitor's screen. Any components to be displayed in the foreground are attached to this glass pane. To reach this glass pane we use getGlassPane() method of JFrame class, which return Component class object.
Root Pane: This pane is below the glass pane. Any components to be displayed in the backgrounds are displayed in this frame. To go to the root pane, we can use getRootPane() method of JFrame class, which returns JRootPane object.
Layered pane: This pane is below the root pane. When we want to take several components as a group, we attach them in the layered pane. We can reach this pane by calling getLayeredPane() method of JFrame class which returns JLayeredPane class object.
Content pane: This is bottom most of all, Individual components are attached to this pane. To reach this pane, we can call getContentPane() method of JFrame class which returns a Container class object.

2. **Lightweight containers:** These are containers that can be nested inside top-level containers or other lightweight containers. Lightweight containers are used to organize and layout the components within a window or panel. Examples of lightweight containers include JPanel, JScrollPane, and JSplitPane. They are lightweight in the sense that they do not have their own native windowing resources, unlike top-level containers. Instead, they rely on the host top-level container for rendering and handling user input.

JFrame:

Frame represents a window with a title bar and borders.
Frame becomes the basis for creating the GUIs for an application because all the components go into the frame.
To create a frame, we have to create an object to JFrame class in swing as:

JFrame jf=new JFrame(); // create a frame without title

JFrame jf=new JFrame("title"); // create a frame with title

To close the frame, use setDefaultCloseOperation(constant) method of JFrame class where constant values are: **JFrame.EXIT_ON_CLOSE, JFrame.DISPOSE_ON_CLOSE, JFrame.DO_NOTHING_ON_CLOSE and JFrame.HIDE_ON_CLOSE**

```
// JFrame Example
import javax.swing.*;
class FrameDemo
{
public static void main(String arg[])
{
JFrame jf=new JFrame("JFrame Example");
jf.setSize(200,200);
jf.setVisible(true);
jf.setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE );
}
}
```

```
// JFrame with background
import javax.swing.*;
import java.awt.*;
class FrameDemo
{
public static void main(String arg[])
{
JFrame jf=new JFrame("JFrame Example");
jf.setSize(200,200);
jf.setVisible(true);
Container c=jf.getContentPane();
c.setBackground(Color.green);
}
}
```

JApplet:

Fundamental to Swing is the JApplet class, which extends Applet and is used to create Applet.

JComponent:

The class JComponent is the base class for all Swing components except top-level containers.

```
import javax.swing.JComponent;
import java.awt.Color;
import java.awt.Graphics;

public class MyCustomComponent extends JComponent {

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);

        // Set the background color
        setBackground(Color.YELLOW);

        // Draw a red rectangle
        g.setColor(Color.RED);
        g.fillRect(10, 10, 100, 50);
    }
}
```

Components of Swing:

JLabel:

- Jlabel is used to display a text
- JLabel(string str)
- JLabel(Icon i)
- JLabel(String s, Icon i, int align)
  CENTER, LEFT, RIGHT, LEADING, TRAILING

Important Methods:

- Icon getIcon()
- String getText()
- void setIcon(Icon icon)
- void setText(String s)

JText Fields

It provides functionality that is common to Swing text components. It allows you to edit one line of text. Some of its constructors are shown here:

- JTextField( )
- JTextField(int cols)
- JTextField(String s, int cols)
- JTextField(String s)
  Here, s is the string to be presented, and cols is the number of columns

JPasswordField:

The JPasswordField class is a text component specialized for password entry. It allows the editing of a single line of text.

Syntax:

```
JPasswordField pwd = new JPasswordField();
pwd.setBounds(100,50,80,30);
```

JTextArea:

A JTextArea is a Swing component used to create a multi-line text input or display area in a Java GUI application. It allows users to enter and edit multiple lines of text.

Some of its constructors are:

- JTextArea()
- JTextArea(String text)
- JTextArea(int rows, int columns)
- JTextArea(String text, int rows, int columns)

The JButton Class

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. Some of its constructors are shown here:

- JButton(Icon i)
- JButton(String s)
- JButton(String s, Icon i)

Here, s and i are the string and icon used for the button.
Swing defines four types of buttons: **JButton**, **JToggleButton**, **JCheckBox**, and **JRadioButton**.
All are subclasses of the AbstractButton class, which extends JComponent. Thus, all buttons share a set of common traits.

JToggleButton

- A useful variation on the push button is called a toggle button.
- A toggle button looks just like a push button, but it acts differently because it has two states: pushed and released.
- That is, when you press a toggle button, it stays pressed rather than popping back up as a regular push button does.
- When you press the toggle button a second time, it releases (pops up). Therefore, each time a toggle button is pushed, it toggles between its two states.

Constructor :

- **JToggleButton(String str)**

JToggleButton generates an item event.
The easiest way to determine a toggle button's state is by calling the **isSelected( )** method

JCheckBox:

The JCheckBox class, which provides the functionality of a check box, is a concrete implementation of AbstractButton. Its immediate super class is JToggleButton, which provides support for two-state buttons (true or false). Some of its constructors are shown here:

- 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)

Here, i is the icon for the button. The text is specified by s. If state is true, the check box is initially selected. Otherwise, it is not.

The state of the check box can be changed via the following method:

void setSelected(boolean state)

When a check box is selected or deselected, an item event is generated. This is handled by **itemStateChanged( )**. Inside **itemStateChanged( )**, the **getItem( )** method gets the JCheckBox object that generated the event. The **getText( )** method gets the text for that check box and uses it to set the text inside the text field.

JRadioButton:

Radio buttons are supported by the JRadioButton class, which is a concrete implementation of AbstractButton. Its immediate superclass is JToggleButton, which provides support for two-state buttons. Some of its constructors are shown here:

- JRadioButton(Icon i)
- JRadioButton(Icon i, boolean state)
- JRadioButton(String s)
- JRadioButton(String s, boolean state)
- JRadioButton(String s, Icon i)
- JRadioButton(String s, Icon i, boolean state)

Here, i is the icon for the button. The text is specified by s. If the state is true, the button is initially selected. Otherwise, it is not.
Radio buttons must be added to the ButtonGroup class to act as mutually exclusive.
Radio button presses generate action events that are handled by **actionPerformed( )**.
The **getActionCommand( )** method returns the text that is associated with a radio button and uses it to set the text field.

JComboBox :
Swing provides a combo box (a combination of a text field and a drop-down list) through the JComboBox class, which extends JComponent.
Two of JComboBox's constructors are shown here:

- JComboBox( )
- JComboBox(Vector v)

Items are added to the list of choices using addItem() method :
void addItem(Object obj)
Here, obj is the object to be added to the combo box.

JList:

JList class is useful to create a list which displays a list of items and allows the user to select one or more items.

Constructors

- JList()
- JList(Object arr[])
- JList(Vector v)

Methods

- getSelectedIndex() – returns selected item index
- getSelectedValue() – to know which item is selected in the list
- getSelectedIndices() – returns selected items into an array
- getSelectedValues() – returns selected items names into an array

JList generates ListSelectionEvent

– ListSelectionListener

• void valueChanged(ListSelectionEvent)

JPanel:

The JPanel is the simplest container class. It provides space in which an application can attach any other component.

Syntax:

```
JPanel panel=new JPanel();
panel.setBounds(40,80,200,200);
panel.setBackground(Color.gray);
JButton b1=new JButton("Button 1");
b1.setBounds(50,100,80,30);
panel.add(b1);
```

JDialog:

The JDialog control represents a top level window with a border and a title used to take some form of input from the user.

Unlike JFrame, it doesn't have maximize and minimize buttons.

Syntax:

```
JFrame f= new JFrame();
JDialog d=new JDialog(f , "Dialog", true);
JButton b = new JButton ("OK");
d.add(b);
```

JTabbedPane:

JTabbedPane encapsulates a tabbed pane. It manages a set of components by linking them with tabs.

The general procedure to use a tabbed pane is outlined here:

1. Create an instance of JTabbedPane.
2. Add each tab by calling addTab( ).

3. Add the tabbed pane to the content pane.

JSplitPane:

JSplitPane is a Swing component in Java that provides a way to divide a GUI component into two or more resizable and scrollable panes. It is commonly used to create a split view or split-pane layout in Swing applications, allowing users to adjust the size and visibility of multiple components within a single container.

Example:

```
import javax.swing.*;

public class Demo {
    public static void main(String[] args) {
        JFrame frame = new JFrame("JSplitPane Example");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setSize(400, 300);

        JPanel leftPanel = new JPanel();
        JPanel rightPanel = new JPanel();

        JSplitPane splitPane = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,
```

```
leftPanel, rightPanel);

        frame.add(splitPane);

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

Layout Managers:

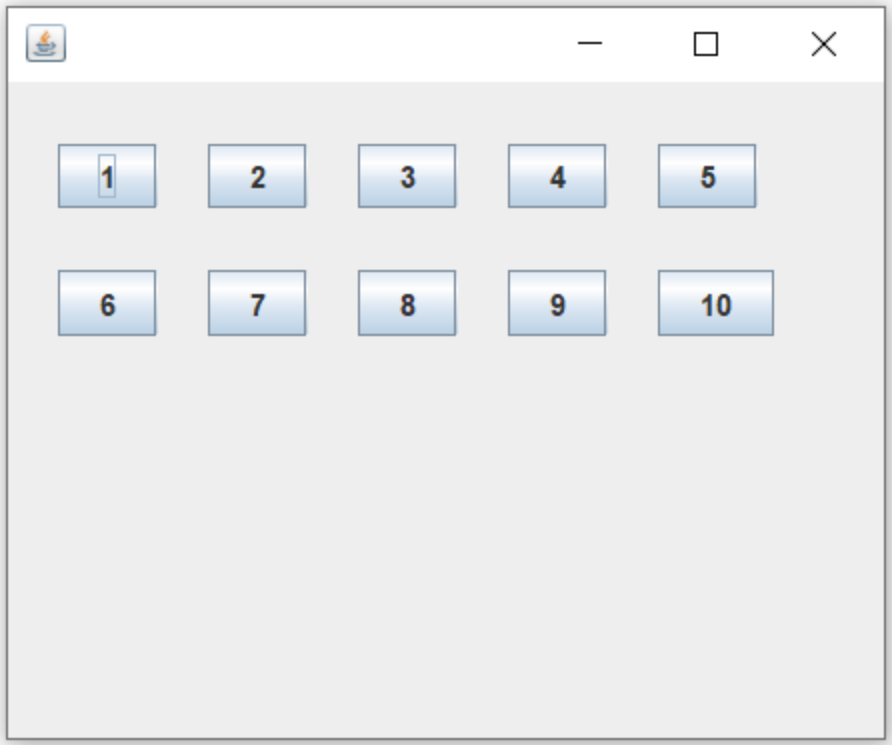
FlowLayout:

FlowLayout is the default layout manager.
FlowLayout implements a simple layout style, which is similar to how words flow in a text editor.
The direction of the layout is governed by the container's component orientation property, which, by default, is left to right, top to bottom.
Here are the constructors for FlowLayout:

- FlowLayout( )
- FlowLayout(int how)
- FlowLayout(int how, int horz, int vert)

The second form lets you specify how each line is aligned. Valid values for how are as follows:

- FlowLayout.LEFT
- FlowLayout.CENTER
- FlowLayout.RIGHT
- FlowLayout.LEADING
- FlowLayout.TRAILING



BorderLayout:

The BorderLayout class implements a common layout style for top-level windows.

The four sides are referred to as north, south, east, and west. The middle area is called the center.

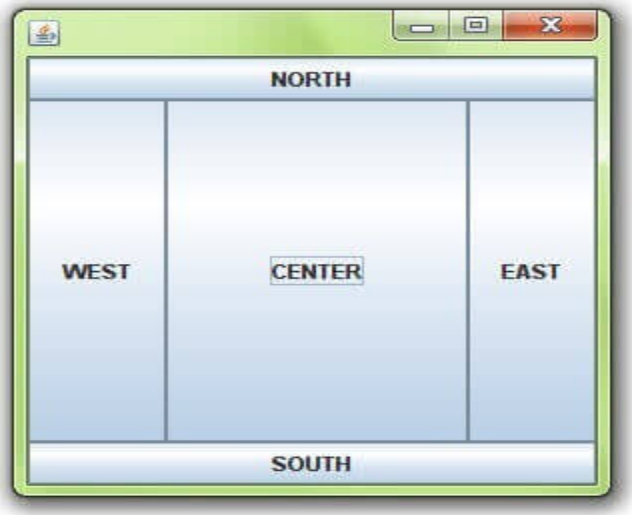
Here are the constructors defined by BorderLayout:

- BorderLayout( )

- BorderLayout(int horz, int vert)

The first form creates a default border layout.
The second allows you to specify the horizontal and vertical space left between components in horz and vert, respectively.
BorderLayout defines the following constants that specify the regions:

- BorderLayout.EAST
- BorderLayout.WEST
- BorderLayout.NORTH
- BorderLayout.SOUTH
- BorderLayout.CENTER



GridLayout

GridLayout lays out components in a two-dimensional grid.
When you instantiate a GridLayout, you define the number of rows and columns.
The constructors supported by GridLayout are shown here:

- GridLayout( )
- GridLayout(int numRows, int numColumns)
- GridLayout(int numRows, int numColumns, int horz, int vert)

The first form creates a single-column grid layout.
The second form creates a grid layout with the specified number of rows and columns.
The third form allows you to specify the horizontal and vertical space left between components in horz and vert, respectively.

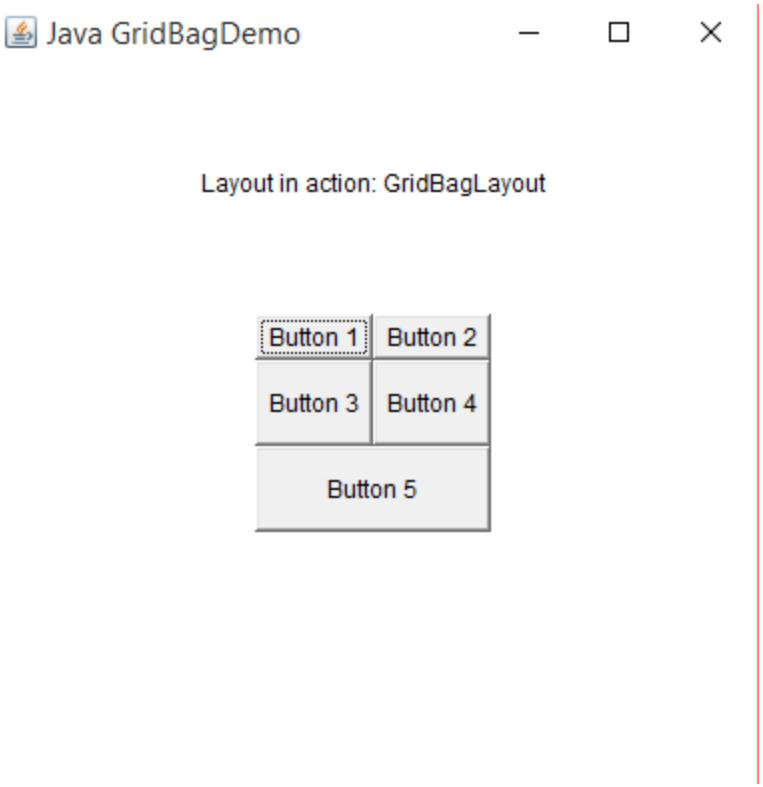
GridBagLayout

- The key to the grid bag is that each component can be a different size, and each row in the grid can have a different number of columns.
- The location and size of each component in a grid bag are determined by a set of constraints linked to it. The constraints are contained in an object of type GridBagConstraints.
- GridBagLayout defines only one constructor, which is shown here:
  GridBagLayout( )
- The fill property of the GridBagConstraints class in Java's GridBagLayout allows you to specify how a component should expand within its cell
  `GridBagConstraints.NONE`
  `GridBagConstraints.HORIZONTAL`
  `GridBagConstraints.VERTICAL`
  `GridBagConstraints.BOTH`
- The ipady field in the GridBagConstraints class is used to specify the internal padding in the vertical (Y-axis)
- The ipadx field in the GridBagConstraints class is used to specify the internal padding in the horizontal (X-axis)
- gridx and gridy is an integer value that represents the column position where the component should be placed
- The weightx determine how much of the extra space within a container is allocated to each row and column
- By default, both these values are zero

```
import java.awt.*;
import java.awt.event.*;
public class GridBagLayoutDemo {
   private Frame f;
   private Label headerLabel;
   private Label statusLabel;
   private Panel controlPanel;
   private Label msglabel;
public GridBagLayoutDemo(){
      f = new Frame("Java GridBagDemo");
      f.setSize(400,400);
      f.setLayout(new GridLayout(3, 1));
      headerLabel = new Label();
      headerLabel.setAlignment(Label.CENTER);
      controlPanel = new Panel();
      controlPanel.setLayout(new FlowLayout());
      f.add(headerLabel);
      f.add(controlPanel);
      f.setVisible(true);
   }
private void showGridBagLayoutDemo(){
      headerLabel.setText("Layout in action: GridBagLayout");
```

```
    Panel panel = new Panel();
    panel.setBackground(Color.darkGray);
    panel.setSize(300,300);
    GridBagLayout gb = new GridBagLayout();
panel.setLayout(gb);
    GridBagConstraints gbc = new GridBagConstraints();

    gbc.fill = GridBagConstraints.HORIZONTAL;
    gbc.gridx = 0;
    gbc.gridy = 0;
    panel.add(new Button("Button 1"),gbc);
    gbc.gridx = 1;
    gbc.gridy = 0;
    panel.add(new Button("Button 2"),gbc);

    gbc.fill = GridBagConstraints.HORIZONTAL;
    gbc.ipady = 20;
    gbc.gridx = 0;
    gbc.gridy = 1;
    panel.add(new Button("Button 3"),gbc);

    gbc.gridx = 1;
```

```
      gbc.gridy = 1;
      panel.add(new Button("Button 4"),gbc);
gbc.gridx = 0;
      gbc.gridy = 2;
      gbc.fill = GridBagConstraints.HORIZONTAL;
      gbc.gridwidth = 2;
      panel.add(new Button("Button 5"),gbc);

      controlPanel.add(panel);
      f.setVisible(true);
   }
       public static void main(String[] args){
      GridBagLayoutDemo  gridLayoutDemo = new GridBagLayoutDemo();
      gridLayoutDemo.showGridBagLayoutDemo();
   }
}
```


Java GridBagDemo
Layout in action: GridBagLayout
Button 1
Button 2
Button 3
Button 4
Button 5

Menu

- A top-level window can have a menu bar associated with it.
- A menu bar displays a list of top-level menu choices. Each choice is associated with a drop-down menu.
- This concept is implemented in the AWT by the following classes: MenuBar, Menu, and MenuItem.
- Following are the constructors for Menu:
  Menu( )
  Menu(String optionName)
  Menu(String optionName, boolean removable)
  Here, optionName specifies the name of the menu selection. If removable is true, the menu can be removed and allowed to float free. Otherwise, it will remain attached to the menu bar. (Removable menus are implementation-dependent.)
- The first form creates an empty menu.
- Individual menu items are of type MenuItem. It defines these constructors:
  MenuItem( )
  MenuItem(String itemName)
  MenuItem(String itemName, MenuShortcut keyAccel)
  Here, itemName is the name shown in the menu, and keyAccel is the menu shortcut for this item.
- You can disable or enable a menu item by using the setEnabled( ) method.
- You can determine an item's status by calling isEnabled( ).
- You can change the name of a menu item by calling setLabel( ) and get its name by using method getLabel()
- You can create a checkable menu item by using a subclass of MenuItem called CheckboxMenuItem. It has these constructors:
  CheckboxMenuItem( )
  CheckboxMenuItem(String itemName)
  CheckboxMenuItem(String itemName, boolean on)

- You can obtain the status of a checkable item by calling getState( ). You can set it to a known state by using setState( ).
- Once you have created a menu item, you must add the item to a Menu object by using add( )
- Once you have added all items to a Menu object, you can add that object to the menu bar by using this version of add( )

```
import java.awt.*;
import java.awt.event.*;

public class MenuDemo {
   public MenuDemo() {
      Frame f = new Frame("DashBoard");
      f.setSize(400, 400);
      MenuBar menubar = new MenuBar();
      Menu fileMenu = new Menu("File",true);
      Menu editMenu = new Menu("Edit");
      Menu helpMenu = new Menu("Help");
      menubar.add(fileMenu);
      menubar.add(editMenu);
      menubar.add(helpMenu);
      f.setMenuBar(menubar);


      MenuItem newfile = new MenuItem("New");
      MenuItem openfile = new MenuItem("Open");
      MenuItem newsave = new MenuItem("Save");
      MenuItem newsaveas = new MenuItem("SaveAs");
```

```
        MenuItem newexit = new MenuItem("Exit");
        CheckboxMenuItem c1 = new CheckboxMenuItem("Check 1", true);
        CheckboxMenuItem c2 = new CheckboxMenuItem("Check 2");
        fileMenu.add(newfile);
        fileMenu.add(openfile);
        fileMenu.add(newsave);
        fileMenu.add(newsaveas);
        fileMenu.add(newexit);
        fileMenu.add(c1);
        fileMenu.add(c2);


        newsaveas.setEnabled(false);

        f.setLocationRelativeTo(null);
        f.setVisible(true);

    }

    public static void main(String[] args) {
        new MenuDemo();
    }
}
```