**Layout Manager in Java**

## **Layout Manager in Java with Examples**

1. **Layout Manager in Java**
2. **Types of Layout Managers**
3. **Flow Layout**
4. **Border Layout**
5. **Card Layout**
6. **Grid Layout**
7. **Grid Bag Layout**

##### **Layout Manager in Java:**

The layout will specify the format or the order in which the components have got to be placed on the container. Layout Manager may be a class or component that’s responsible to rearrange the components on the container consistent with the required layout. A layout manager automatically arranges your controls within a window by using some sort of algorithm.

Each Container object features a layout manager related to it. A layout manager is an instance of any class that implements the LayoutManager interface. The layout manager is about by the setLayout( ) method. If no call to setLayout( ) is formed, then the default layout manager is employed. Whenever a container is resized (or sized for the primary time), the layout manager is employed to position each of the components within it.

The setLayout( ) method has the subsequent general form: **void setLayout(LayoutManager layoutObj**)

Here, layoutObj may be a regard to the specified layout manager. If you would like to disable the layout manager and position components manually, pass null for layoutObj. If you are doing this, you’ll get to determine the form and position of every component manually, using the setBounds() method defined by Component.

##### **Types of Layout Managers**

AWT package provides following types of Layout Managers:

1. Flow Layout
2. Border Layout
3. Card Layout
4. Grid Layout
5. GridBag Layout

##### **Flow Layout**

This layout will display the components in sequence 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.

**Note:** If the row contains only one component then the component is aligned in the center position of that row.

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

##### **Example to demonstrate Flow Layout in Java**

import java.awt.\*;

import javax.swing.\*;

public class FlowLayoutDemo

{

JFrame f;

FlowLayoutDemo ()

{

f = new JFrame ();

JLabel l1 = new JLabel ("Enter Name");

JTextField tf1 = new JTextField (10);

JButton b1 = new JButton ("SUBMIT");

f.add (l1);

f.add (tf1);

f.add (b1);

f.setLayout (new FlowLayout (FlowLayout.RIGHT));

//setting flow layout of right alignment

f.setSize (300, 300);

f.setVisible (true);

}

public static void main (String[]args)

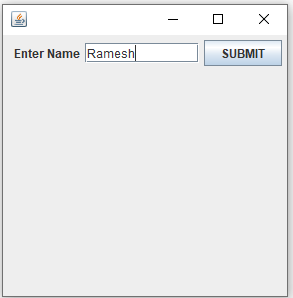
{

new FlowLayoutDemo ();

}

}

###### **Output:**



##### **Border Layout in Java**

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. The above regions are the predefined static constants belonging to the BorderLayout class. Whenever other regions’ spaces are not in use automatically container selects as a center region default and the component occupies the surrounding region’s spaces of the window and which damages the look and feel of the user interface.

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

##### **Example to demonstrate Border Layout in Java**

import java.awt.\*;

public class BorderLayoutDemo

{

public static void main (String[]args)

{

Frame f1 = new Frame ();

f1.setSize (250, 250);

Button b1 = new Button ("Button1");

Button b2 = new Button ("Button2");

Button b3 = new Button ("Button3");

Button b4 = new Button ("Button4");

Button b5 = new Button ("Button5");

f1.add (b1, BorderLayout.NORTH);

f1.add (b2, BorderLayout.EAST);

f1.add (b3, BorderLayout.WEST);

f1.add (b4, BorderLayout.SOUTH);

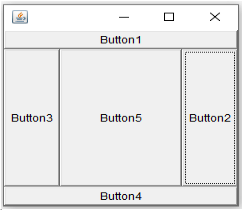
f1.add (b5);

f1.setVisible (true);

}

}

###### **Output:**



**In the above application, frame class contains two types of add methods:**

1. **add(component):**This method default aligns components in the center region.
2. **add(component, region name):**Internally add method handovers a given component (i.e object) to the container and container user peer class of that component to communicate with OS library and then created component is aligned on the window.

##### **Card Layout in Java**

A card layout represents a stack of cards displayed on a container. At a time only one card can be visible and each can contain the 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 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.

##### **Example to demonstrate Card Layout in Java**

import java.awt.\*;

import javax.swing.\*;

import javax.swing.JButton;

import java.awt.event.\*;

public class CardLayoutDemo extends JFrame implements ActionListener

{

JButton b1, b2, b3, b4, b5;

CardLayout cl;

Container c;

CardLayoutDemo ()

{

b1 = new JButton ("Button1");

b2 = new JButton ("Button2");

b3 = new JButton ("Button3");

b4 = new JButton ("Button4");

b5 = new JButton ("Button5");

c = this.getContentPane ();

cl = new CardLayout (10, 20);

c.setLayout (cl);

c.add ("Card1", b1);

c.add ("Card2", b2);

c.add ("Card3", b3);

b1.addActionListener (this);

b2.addActionListener (this);

b3.addActionListener (this);

setVisible (true);

setSize (400, 400);

setTitle ("Card Layout");

setDefaultCloseOperation (JFrame.EXIT\_ON\_CLOSE);

}

public void actionPerformed (ActionEvent ae)

{

cl.next (c);

}

public static void main (String[]args)

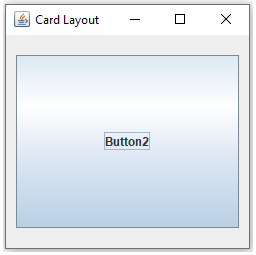
{

new CardLayoutDemo ();

}

}

###### **Output:**



##### **Grid Layout in Java**

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);**

##### **Example to demonstrate Grid Layout in Java**

import java.awt.\*;

import javax.swing.\*;

public class GridLayoutDemo

{

public static void main (String[]args)

{

Frame f1 = new Frame ();

f1.setSize (250, 250);

GridLayout ob = new GridLayout (2, 2);

f1.setLayout (ob);

Panel p1 = new Panel ();

Label l1 = new Label ("Enter name");

TextField tf = new TextField (10);

Button b1 = new Button ("Submit");

p1.add (l1);

p1.add (tf);

p1.add (b1);

f1.add (p1);

Panel p2 = new Panel ();

f1.add (p2);

Panel p3 = new Panel ();

f1.add (p3);

Label l2 = new Label ("Welcome to Java");

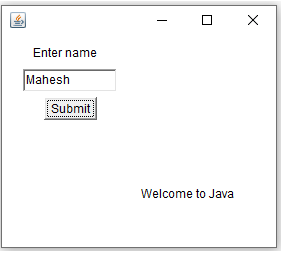
f1.add (l2);

f1.setVisible (true);

}

}

###### **Output:**



##### **Grid Bag Layout in Java**

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, specify the size, etc. In this Layout manager, for each grid, we need to define grid properties or constraints. Based on grid properties, the layout manager aligns a component on the grid, and also we can span multiple grids also as per the requirement. Gris properties are defined using GridBagConstraints class.

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

**Note:**We can specify the location (or) the size with the help of GridBagConstraints.

##### **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.

**FIRST\_LINE\_START PAGE\_START FIRST\_LINE\_END**  
**LINE\_START CENTER LINE\_END**  
**LAST\_LINE\_START PAGE\_END LAST\_LINE\_END**

           The above format is equaled to one grid and so we can specify components in any position of the grid.

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

##### **Example to demonstrate GridBag Layout Manager in Java**

import java.awt.\*;

public class GridBagLayoutDemo

{

public static void main (String[]args)

{

Frame f1 = new Frame ();

f1.setSize (250, 250);

GridBagLayout gb = new GridBagLayout ();

f1.setLayout (gb);

GridBagConstraints gc = new GridBagConstraints ();

Button b1 = new Button ("Button1");

Button b2 = new Button ("Button2");

Button b3 = new Button ("Button3");

gc.fill = GridBagConstraints.HORIZONTAL;

gc.weightx = 0.5;

gc.weighty = 0.5;

gc.gridx = 0;

gc.gridy = 0;

f1.add (b1, gc);

gc.gridx = 1;

gc.gridy = 0;

f1.add (b2, gc);

gc.gridx = 2;

gc.gridy = 0;

f1.add (b3, gc);

Button b4 = new Button ("Button4");

gc.gridx = 0;

gc.gridy = 1;

gc.gridwidth = 3;

gc.ipady = 40;

Button b5 = new Button ("Button5");

gc.gridx = 2;

gc.gridy = 3;

gc.insets = new Insets (10, 0, 10, 0);

f1.add (b5, gc);

f1.pack ();

f1.setVisible (true);

}

}

**Output:**

