Java LayoutManagers

The LayoutManagers are used to arrange components in a particular manner. LayoutManager is an interface that is implemented by all the classes of layout managers. There are following classes that represents the layout managers:

1. java.awt.BorderLayout
2. java.awt.FlowLayout
3. java.awt.GridLayout
4. java.awt.CardLayout
5. java.awt.GridBagLayout
6. javax.swing.BoxLayout
7. Java BorderLayout

The BorderLayout is used to arrange the components in five regions: north, south, east, west and center. Each region (area) may contain one component only. It is the default layout of frame or window. The BorderLayout provides five constants for each region:

1. **public static final int NORTH**
2. **public static final int SOUTH**
3. **public static final int EAST**
4. **public static final int WEST**
5. **public static final int CENTER**

Constructors of BorderLayout class:

* **BorderLayout():** creates a border layout but with no gaps between the components.
* **JBorderLayout(int hgap, int vgap):** creates a border layout with the given horizontal and vertical gaps between the components.

**import** java.awt.\*;

**import** javax.swing.\*;

**public** **class** Border {

JFrame f;

Border(){

    f=**new** JFrame();

    JButton b1=**new** JButton("NORTH");;

    JButton b2=**new** JButton("SOUTH");;

    JButton b3=**new** JButton("EAST");;

    JButton b4=**new** JButton("WEST");;

    JButton b5=**new** JButton("CENTER");;

    f.add(b1,BorderLayout.NORTH);

    f.add(b2,BorderLayout.SOUTH);

    f.add(b3,BorderLayout.EAST);

    f.add(b4,BorderLayout.WEST);

    f.add(b5,BorderLayout.CENTER);

    f.setSize(300,300);

    f.setVisible(**true**);

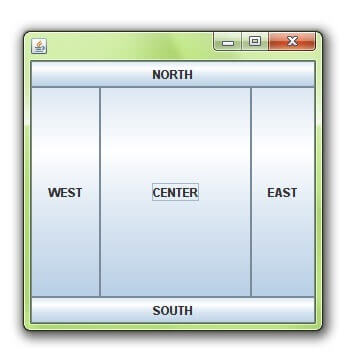
}

**public** **static** **void** main(String[] args) {

**new** Border();

}

}



# Java FlowLayout

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

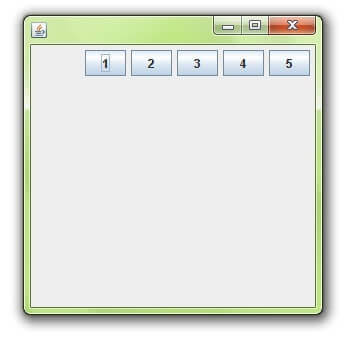
### Fields of FlowLayout class

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.

### Example of FlowLayout class



**import** java.awt.\*;

**import** javax.swing.\*;

**public** **class** MyFlowLayout{

JFrame f;

MyFlowLayout(){

    f=**new** JFrame();

    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");

    f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);

    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** MyFlowLayout();

}

}

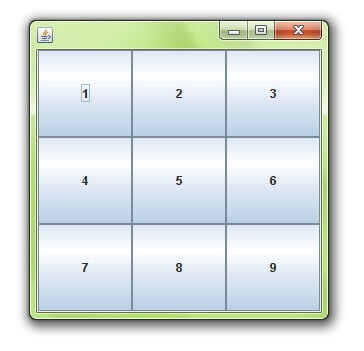
# Java GridLayout

The GridLayout is used to arrange the components in rectangular grid. One component is displayed in each rectangle.

### Constructors of GridLayout class

1. **GridLayout():** creates a grid layout with one column per component in a row.
2. **GridLayout(int rows, int columns):** creates a grid layout with the given rows and columns but no gaps between the components.
3. **GridLayout(int rows, int columns, int hgap, int vgap):** creates a grid layout with the given rows and columns alongwith given horizontal and vertical gaps.

### Example of GridLayout class



**import** java.awt.\*;

**import** javax.swing.\*;

**public** **class** MyGridLayout{

JFrame f;

MyGridLayout(){

    f=**new** JFrame();

    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");

    f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);

    f.add(b6);f.add(b7);f.add(b8);f.add(b9);

    f.setLayout(**new** GridLayout(3,3));

    //setting grid layout of 3 rows and 3 columns

    f.setSize(300,300);

    f.setVisible(**true**);

}

**public** **static** **void** main(String[] args) {

**new** MyGridLayout();

}

}

# Java BoxLayout

The BoxLayout is used to arrange the components either vertically or horizontally. For this purpose, BoxLayout provides four constants. They are as follows:

#### Note: BoxLayout class is found in javax.swing package.

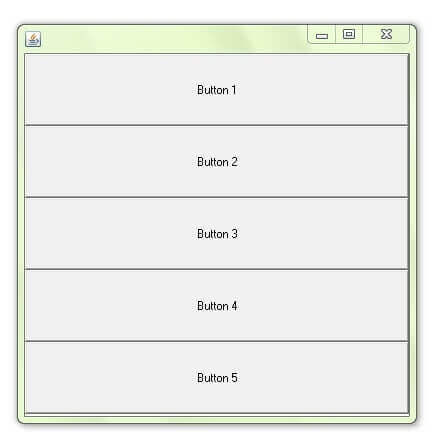
### Fields of BoxLayout class

1. **public static final int X\_AXIS**
2. **public static final int Y\_AXIS**
3. **public static final int LINE\_AXIS**
4. **public static final int PAGE\_AXIS**

### Constructor of BoxLayout class

1. **BoxLayout(Container c, int axis):** creates a box layout that arranges the components with the given axis.

### Example of BoxLayout class with Y-AXIS:



**import** java.awt.\*;

**import** javax.swing.\*;

**public** **class** BoxLayoutExample1 **extends** Frame {

 Button buttons[];

**public** BoxLayoutExample1 () {

   buttons = **new** Button [5];

**for** (**int** i = 0;i<5;i++) {

      buttons[i] = **new** Button ("Button " + (i + 1));

      add (buttons[i]);

    }

setLayout (**new** BoxLayout (**this**, BoxLayout.Y\_AXIS));

setSize(400,400);

setVisible(**true**);

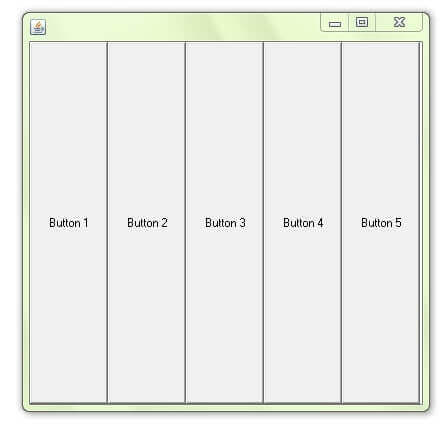
}

**public** **static** **void** main(String args[]){

BoxLayoutExample1 b=**new** BoxLayoutExample1();

}

}



**import** java.awt.\*;

**import** javax.swing.\*;

**public** **class** BoxLayoutExample2 **extends** Frame {

 Button buttons[];

**public** BoxLayoutExample2() {

   buttons = **new** Button [5];

**for** (**int** i = 0;i<5;i++) {

      buttons[i] = **new** Button ("Button " + (i + 1));

      add (buttons[i]);

    }

setLayout (**new** BoxLayout(**this**, BoxLayout.X\_AXIS));

setSize(400,400);

setVisible(**true**);

}

**public** **static** **void** main(String args[]){

BoxLayoutExample2 b=**new** BoxLayoutExample2();

}

}

# Java CardLayout

The CardLayout class manages the components in such a manner that only one component is visible at a time. It treats each component as a card that is why it is known as CardLayout.

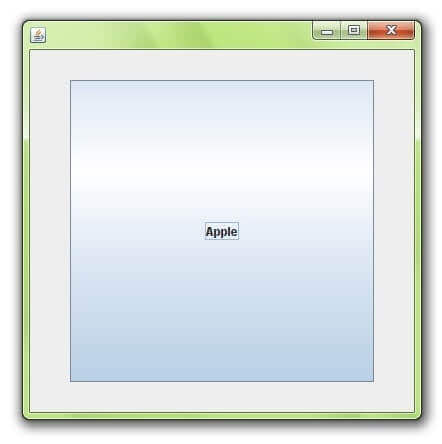
### Constructors of CardLayout class

1. **CardLayout():** creates a card layout with zero horizontal and vertical gap.
2. **CardLayout(int hgap, int vgap):** creates a card layout with the given horizontal and vertical gap.

### Commonly used methods of CardLayout class

* **public void next(Container parent):** is used to flip to the next card of the given container.
* **public void previous(Container parent):** is used to flip to the previous card of the given container.
* **public void first(Container parent):** is used to flip to the first card of the given container.
* **public void last(Container parent):** is used to flip to the last card of the given container.
* **public void show(Container parent, String name):** is used to flip to the specified card with the given name.

### Example of CardLayout class



**import** java.awt.\*;

**import** java.awt.event.\*;

**import** javax.swing.\*;

**public** **class** CardLayoutExample **extends** JFrame **implements** ActionListener{

CardLayout card;

JButton b1,b2,b3;

Container c;

    CardLayoutExample(){

        c=getContentPane();

        card=**new** CardLayout(40,30);

//create CardLayout object with 40 hor space and 30 ver space

        c.setLayout(card);

        b1=**new** JButton("Apple");

        b2=**new** JButton("Boy");

        b3=**new** JButton("Cat");

        b1.addActionListener(**this**);

        b2.addActionListener(**this**);

        b3.addActionListener(**this**);

        c.add("a",b1);c.add("b",b2);c.add("c",b3);

    }

**public** **void** actionPerformed(ActionEvent e) {

    card.next(c);

    }

**public** **static** **void** main(String[] args) {

        CardLayoutExample cl=**new** CardLayoutExample();

        cl.setSize(400,400);

        cl.setVisible(**true**);

        cl.setDefaultCloseOperation(EXIT\_ON\_CLOSE);

    }

}

# Java GridBagLayout

The Java GridBagLayout class is used to align components vertically, horizontally or along their baseline.

The components may not be of same size. Each GridBagLayout object maintains a dynamic, rectangular grid of cells. Each component occupies one or more cells known as its display area. Each component associates an instance of GridBagConstraints. With the help of constraints object we arrange component's display area on the grid. The GridBagLayout manages each component's minimum and preferred sizes in order to determine component's size.

### Example

**import** java.awt.Button;

**import** java.awt.GridBagConstraints;

**import** java.awt.GridBagLayout;

**import** javax.swing.\*;

**public** **class** GridBagLayoutExample **extends** JFrame{

**public** **static** **void** main(String[] args) {

            GridBagLayoutExample a = **new** GridBagLayoutExample();

        }

**public** GridBagLayoutExample() {

    GridBagLayoutgrid = **new** GridBagLayout();

            GridBagConstraints gbc = **new** GridBagConstraints();

            setLayout(grid);

            setTitle("GridBag Layout Example");

            GridBagLayout layout = **new** GridBagLayout();

**this**.setLayout(layout);

    gbc.fill = GridBagConstraints.HORIZONTAL;

    gbc.gridx = 0;

    gbc.gridy = 0;

**this**.add(**new** Button("Button One"), gbc);

    gbc.gridx = 1;

    gbc.gridy = 0;

**this**.add(**new** Button("Button two"), gbc);

    gbc.fill = GridBagConstraints.HORIZONTAL;

    gbc.ipady = 20;

    gbc.gridx = 0;

    gbc.gridy = 1;

**this**.add(**new** Button("Button Three"), gbc);

    gbc.gridx = 1;

    gbc.gridy = 1;

**this**.add(**new** Button("Button Four"), gbc);

    gbc.gridx = 0;

    gbc.gridy = 2;

    gbc.fill = GridBagConstraints.HORIZONTAL;

    gbc.gridwidth = 2;

**this**.add(**new** Button("Button Five"), gbc);

            setSize(300, 300);

            setPreferredSize(getSize());

            setVisible(**true**);

            setDefaultCloseOperation(EXIT\_ON\_CLOSE);

        }

}

Output:

