GRAPHICS PROGRAMMING – LAYOUT MANAGERS

CHAPTER 29

We create several components like push buttons, checkboxes, radio buttons etc., in GUI. After creating these components, they should be placed in the frame (in AWT) or container (in swing). While arranging them in the frame or container, they can be arranged in a particular manner by using layout managers. JavaSoft people have created a `LayoutManager` interface in `java.awt` package which is implemented in various classes which provide various types of layouts to arrange the components.

Important Interview Question

What is a layout manager?

A layout manager is a class that is useful to arrange components in a particular manner in a frame or container.

The following classes represent the layout managers in Java:

- FlowLayout
- BorderLayout
- CardLayout
- GridLayout
- GridBagLayout
- BoxLayout

To set a particular layout, we should first create an object to the layout class and pass the object to `setLayout()` method. For example, to set `FlowLayout` to the container that holds the components, we can write:

```
FlowLayout obj = new FlowLayout();
c.setLayout(obj);  //c is container
```

FlowLayout

FlowLayout is useful to arrange the components in a line one after the other. When a line is filled with components, they are automatically placed in the next line. This is the default layout in applets and panels.

To create FlowLayout, we can use the following ways:

- `FlowLayout obj = new FlowLayout();`

This creates flow layout. By default, the gap between components will be 5 pixels and the components are centered in the first line.

- `FlowLayout obj = new FlowLayout(int alignment);`

Here, the alignment of components can be specified. To arrange the components starting from left to right, we can use `FlowLayout.LEFT`. To adjust the components towards right, we can use `FlowLayout.RIGHT` and for center alignment, we can use `FlowLayout.CENTER`.

- `FlowLayout obj = new FlowLayout(int alignment, int hgap, int vgap);`

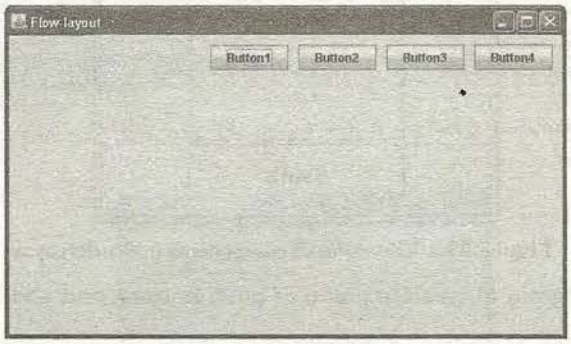
Here, the `hgap` and `vgap` specify the space between components. `hgap` represents horizontal gap and `vgap` represents vertical gap in pixels.

Program 1: Write a program to create a group of push buttons and arrange them in the container using flow layout manager. The buttons are right justified.

```
//FlowLayout demo
import java.awt.*;
import javax.swing.*;
class FlowLayoutDemo extends JFrame
{
        FlowLayoutDemo()
        {
                //create content pane
                Container c = getContentPane();

                //create FlowLayout object with alignment: right
                //and 10px horizontal and vertical gap
        FlowLayout obj = new FlowLayout(FlowLayout.RIGHT,10,10);

                //set the layout to content pane
                c.setLayout(obj);

                //create 4 push buttons
                JButton b1,b2,b3,b4;
                b1 = new JButton("Button1");
                b2 = new JButton("Button2");
                b3 = new JButton("Button3");
                b4 = new JButton("Button4");

                //when we add the buttons to c, they are added as per flow layout
                c.add(b1);
                c.add(b2);
                c.add(b3);
                c.add(b4);
        }
        public static void main(String args[])
        {
                //create the frame
                FlowLayoutDemo demo = new FlowLayoutDemo();
                demo.setSize(400,400);
                demo.setTitle("Flow layout");
                demo.setVisible(true);
                demo.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }
}
```

Output:

```
C:\> javac FlowLayoutDemo.java
C:\> java FlowLayoutDemo
```



BorderLayout

BorderLayout is useful to arrange the components in the 4 borders of the frame as well as in the center. The borders are identified with the names of directions. The top border is specified as 'North', the right side border 'East', the bottom one as 'South' and the left one as 'West'. The center is represented as 'Center'. See Figure 29.1.

To create a BorderLayout, we can use the following ways:

- `BorderLayout obj = new BorderLayout();`

This creates a BorderLayout object without any gaps between the components.

- `BorderLayout obj = new BorderLayout(int hgap, int vgap);`

Here, `hgap` represents horizontal gap and `vgap` represents vertical gap between components in pixels.

While adding the components to the container the direction should be specified, as:

```
c.add("North", component);  //c is container
```

Here, the component is added in the container in North direction.

We can also add the component in North direction, as shown here:

```
c.add(component, BorderLayout.NORTH);
```

In the preceding statement, we are specifying the direction with the constant, `BorderLayout.NORTH`. Similarly, other constants `EAST`, `WEST`, `SOUTH`, `CENTER` can be used.

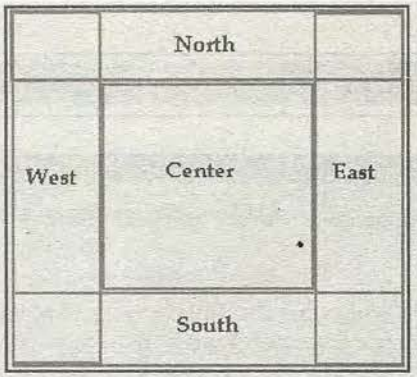


Figure 29.1 Direction of components in BorderLayout

Program 2: Write a program to create a group of push buttons and add them to the container by using BorderLayout.

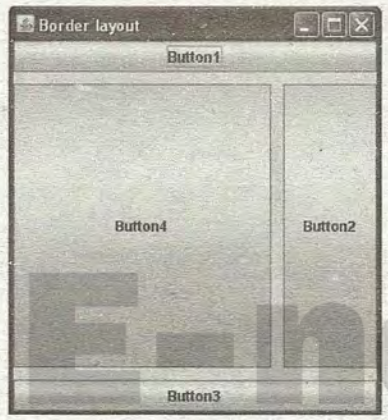
```
//BorderLayout demo
import java.awt.*;
import javax.swing.*;
class BorderLayoutDemo extends JFrame
{
      BorderLayoutDemo()
      {
            //create content pane
            Container c = getContentPane();

            //create border layout with 10px horizontal and vertical
            //gap between components
            BorderLayout obj = new BorderLayout(10,10);

            //set border layout to c
            c.setLayout(obj);

            //create 4 push buttons
            JButton b1,b2,b3,b4;
            b1 = new JButton("Button1");
            b2 = new JButton("Button2");
            b3 = new JButton("Button3");
            b4 = new JButton("Button4");

            //add buttons to c
            c.add("North",b1);
            c.add("East", b2);
            c.add("South",b3);
            c.add("Center",b4);

/*
            the above statements can be re-written as:
            c.add(b1, BorderLayout.NORTH);
            c.add(b2, BorderLayout.EAST);
            c.add(b3, BorderLayout.SOUTH);
            c.add(b4, BorderLayout.CENTER);
*/

      }
      public static void main(String args[])
      {
            //create the frame
            BorderLayoutDemo demo = new BorderLayoutDemo();
            demo.setSize(400,400);
```

```
                demo.setVisible(true);
                demo.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        }
    }
```

Output:

```
C:\> javac BorderLayoutDemo.java
C:\> java BorderLayoutDemo
```

In preceding program, the components are arranged in the four borders as well as in the center of the container.



CardLayout

A CardLayout object is a layout manager which treats each component as a card. Only one card is visible at a time, and the container acts as a stack of cards. The first component added to a CardLayout object is the visible component when the container is first displayed.

To create CardLayout object, we can use the following ways:

- `CardLayout obj = new CardLayout();`

Here, the card layout object is created without any gaps between the components.

- `CardLayout obj = new CardLayout(int hgap, int vgap);`

The preceding statement creates a card layout with the specified horizontal and vertical gaps between the components.

- While adding components to the container, we can use add() method as:

```
c.add("cardname", component);
```

- To retrieve the cards one by one, the following methods can be used:
  - `void first(container)` : to retrieve the first card.
  - `void last(container)` : to retrieve the last card.
  - `void next(container)` : to go to the next card.
  - `void previous(container)` : to go back to previous card.

- void show(container, "cardname") : to see a particular card with the name specified.

Program 3: Write a program to create a group of push buttons and add them to the container using CardLayout.

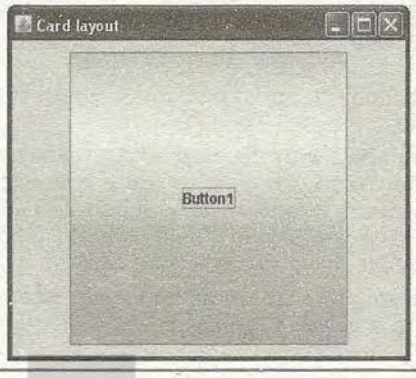
```
//Card layout demo
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class CardLayoutDemo extends JFrame implements ActionListener
{
      //vars
      Container c;
      CardLayout card;
      JButton b1,b2,b3,b4;

      CardLayoutDemo()
      {
            //create container
            c = getContentPane();

            //create CardLayout object with 50 px horizontal space
            //and 10 px vertical space
            card = new CardLayout(50,10);

            //set the layout to card layout
            c.setLayout(card);

            //create 4 push buttons
            b1 = new JButton("Button1");
            b2 = new JButton("Button2");
            b3 = new JButton("Button3");
            b4 = new JButton("Button4");

            //add each button to c on a separate card
            c.add("First card",b1);
            c.add("Second card",b2);
            c.add("Third card",b3);
            c.add("Fourth card", b4);

            //add action listeners to buttons
            b1.addActionListener(this);
            b2.addActionListener(this);
            b3.addActionListener(this);
            b4.addActionListener(this);

      }
      public void actionPerformed(ActionEvent ae)
      {
            //when a button is clicked show the next card
            card.next(c);

            /* To show a particular card, e.g. Third card, we can use as:
            card.show(c, "Third card");
            */
      }

      public static void main(String args[])
      {
            //create frame
            CardLayoutDemo demo = new CardLayoutDemo();
            demo.setSize(400,400);
            demo.setTitle("Card layout");
            demo.setVisible(true);
            demo.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      }
```

```
}
```

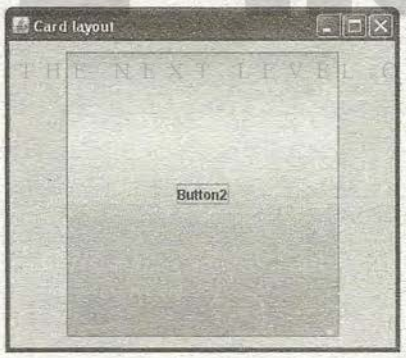
Output:

```
C:\> javac CardLayoutDemo.java
C:\> java CardLayoutDemo
```

In preceding program, the components are arranged on 4 cards whose names are First card, Second card, Third card and Fourth card. When a button on a card is clicked, the next card is displayed.



Observe the preceding output. If the Button1 is clicked there, the next card with Button2 will be displayed as shown here.



Using a Layout Inside Another Layout

It is possible to use a layout inside another layout. For example, we can set a particular layout to a group of components and that group can be added to a frame or container by using another layout. In the previous program, we used card layout to display push buttons on each card. Suppose we want to display a group of components on the second card. How can we do this? First, we should write a class which extends JPanel class. JPanel class is useful to create a panel to which a group of components can be attached, as see here:

```
class MyPanel extends JPanel
```

The default layout used by a panel is flow layout. If we want any other layout, we can set it using `setLayout()` method.

In MyPanel class, we should create the components and attach them to panel, as:

```
this.add(component1);
this.add(component2);
```

Now, create an object to MyPanel and pass that object as a component to the container by using card layout, as:

```
c.add("Second card", new MyPanel());
```

Since, we are creating MyPanel object, and it contains a group of components, all those components are attached to the Second card.

In Program 4, we create a container with the card layout manager. Then, a push button is added to the first card. In case of second card, we want to add a group of components, a text field, a check box and a push button. We want to add these components to the second card by using border layout. For this purpose, we create a panel and add the components by using border layout manager. Then, we add the panel object to the second card.

Program 4: Write a program to add a group of components: a text field, a check box and a push button, using border layout to a panel. Then the panel object is added in the container which uses card layout.

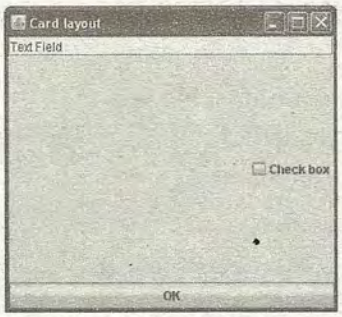
```
//Using border layout inside card layout.
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
class LayoutsDemo extends JFrame implements ActionListener
{
      //vars
      Container c;
      CardLayout card;
      JButton b1;

      LayoutsDemo()
      {
            //create container
            c = getContentPane();

            //create CardLayout object w
            card = new CardLayout();

            //set the layout to card layout
            c.setLayout(card);

            //create a push buttons
            b1 = new JButton("Button1");

            //add button to c on first card
            c.add("First card",b1);

            //add panel object to c on second card
            //MyPanel is the sub class of Panel class
            c.add("Second card",new MyPanel());

            //add action listeners to buttons
            b1.addActionListener(this);

      }
      public void actionPerformed(ActionEvent ae)
      {
```

```
                //when a button is clicked show the second card
        card.next(c);
        }

        public static void main(String args[])
        {
                //create frame
                LayoutsDemo demo = new LayoutsDemo();
                demo.setSize(400,400);
                demo.setTitle("Card layout");
                demo.setVisible(true);

                demo.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }
}

class MyPanel extends JPanel
{
        //vars
        JTextField tf;
        JCheckBox cb;
        JButton b;

        MyPanel()
        {
                //set border layout to panel
                this.setLayout(new BorderLayout());

                //create components
                tf = new JTextField("Text Field", 15);
                b = new JButton("OK");
                cb = new JCheckBox("Check box");

                //add them to panel
                this.add("North", tf);
                this.add("South", b);
                this.add("East", cb);
        }
}
```

Output:

```
C:\> javac LayoutsDemo.java
C:\> java LayoutsDemo
```

When the preceding button on first card is clicked, the user is led to second card where three components are displayed by using border layout, as shown here.



GridLayout

GridLayout is useful to divide the container into a two-dimensional grid form that contains several rows and columns. The container is divided into equal-sized rectangles, and one component is placed in each rectangle.

To create `GridLayout` object, we can write as:

- `GridLayout obj = new GridLayout();`

This creates a grid layout with a default of one column per component, in a single row.

- `GridLayout obj = new GridLayout(int rows, int cols);`

This creates a grid layout with specified number of rows and columns.

- `GridLayout obj = new GridLayout(int rows, int cols, int hgap, int vgap);`

Here, `hgap` represents horizontal gap between components and `vgap` represents vertical gap between components.

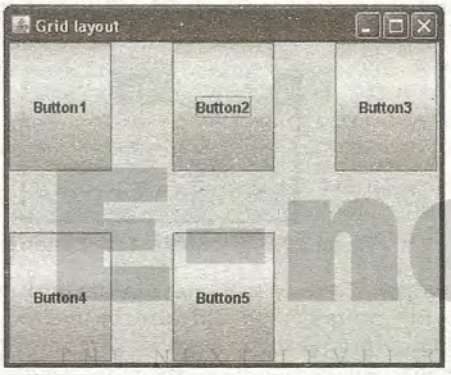
Program 5: Write a program to create five push buttons and add them to the content pane using border layout manager.

```
//GridLayout demo
import java.awt.*;
import javax.swing.*;
class GridLayoutDemo extends JFrame
{
      GridLayoutDemo()
      {
            //create container
            Container c = getContentPane();

            //create grid layout with 2 rows, 3 cols and 50 px
            //gap between components
            GridLayout grid = new GridLayout(2,3,50,50);
            c.setLayout(grid);

            //create 5 push buttons
            JButton b1 = new JButton("Button1");
            JButton b2 = new JButton("Button2");
            JButton b3 = new JButton("Button3");
            JButton b4 = new JButton("Button4");
            JButton b5 = new JButton("Button5");

            //add buttons to c
            c.add(b1);
            c.add(b2);
            c.add(b3);
            c.add(b4);
```

```
            c.add(b5);
        }
        public static void main(String args[])
        {
            //create a frame
            GridLayoutDemo demo = new GridLayoutDemo();
            demo.setSize(500,400);
            demo.setTitle("Grid layout");
            demo.setVisible(true);
            demo.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }
}
```

Output:

```
C:\> javac GridLayoutDemo.java
C:\> java GridLayoutDemo
```



GridBagLayout

GridBagLayout class represents grid bag layout manager where the components are arranged in rows and columns. This layout is more flexible as compared to other layouts since in this layout, the components can span more than one row or column and the size of the components can be adjusted to fit the display area. The intersection of rows and columns where a component can be placed is called a 'grid' or 'display area'.

When positioning the components by using grid bag layout, it is necessary to apply some constraints or conditions on the components regarding their position, size and space in or around the components etc. Such constraints are specified using `GridBagConstraints` class.

To create grid bag layout, we can create an object to `GridBagLayout` class, as:

```
GridBagLayout obj = new GridBagLayout();
```

To apply some constraints on the components, we should first create an object to `GridBagConstraints` class, as:

```
GridBagConstraints cons = new GridBagConstraints();
```

This will create constraints for the components with default values. The other way to specify the constraints is by directly passing their values while creating the `GridBagConstraints` object, as:

```
GridBagConstraints cons = new GridBagConstraints(int gridx, int gridy,
int gridwidth, int gridheight, double weightx, double weighty, int anchor,
int fill, Insets insets, int ipadx, int ipady);
```

Let us now understand each of the constraints.

- `GridBagConstraints.gridx, GridBagConstraints.gridy`: They represent the row and column positions of the component at upper left corner of the component. See Figure 29.2.

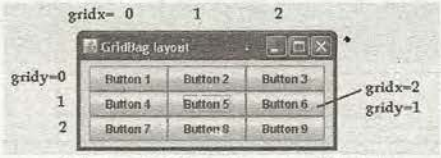


Figure 29.2 gridx and gridy values

- `GridBagConstraints.gridwidth, GridBagConstraints.gridheight`: Specify the number of columns (for gridwidth) or rows (for gridheight) in the component's display area. The default value is 1. See Figure 29.3.

Figure 29.3 gridwidth=2 for Button4

- `GridBagConstraints.weightx, GridBagConstraints.weighty`: When the frame is resized, the components inside the container should also be resized or not – is determined by the weightx and weighty constraints. When these values are not set, by default, they take 0.0. This means the components size will not change when the frame is resized. The components will have their original size. If the weightx and weighty values are set to a value from 0.0 to 1.0, then the components size will also change along with the size of the frame. weightx is for resizing the component horizontally and weighty is for resizing the component vertically. Generally, weights are specified with 0.0 and 1.0 as the extremes, the numbers in between are used as necessary. Larger numbers indicate that the component's row or column should get more space. Figure 29.4 depicts components when resized and not resized.

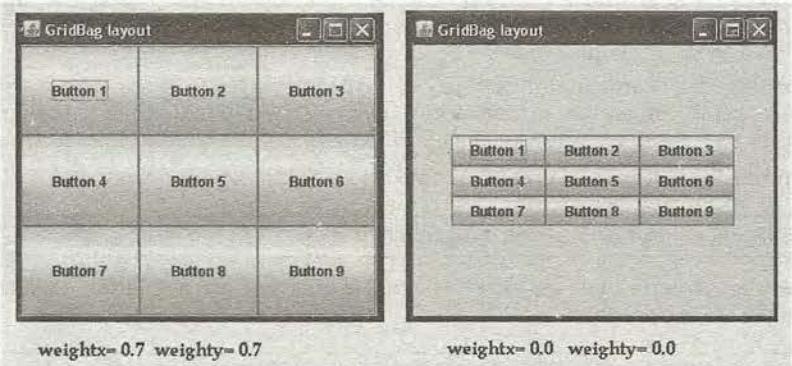


Figure 29.4 components resized and not resized

- `GridBagConstraints.anchor`: When the display area is larger than the component, anchor constraint will determine where to place the component in the display area. The positions of the component in the display area are shown in the Figure 29.5 here. The default value is `GridBagConstraints.CENTER`.

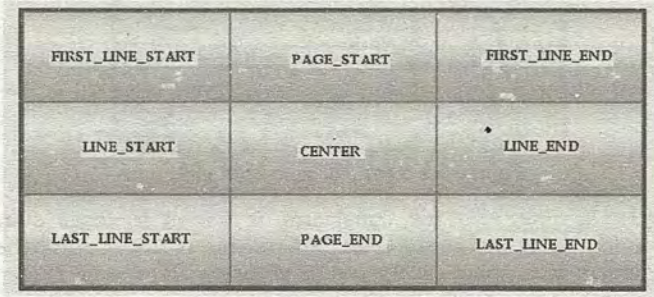


Figure 29.5 The anchor values of a component

- `GridBagConstraints.fill`: While the weightx and weighty constraints are useful to resize the component according to the frame's size, 'fill' is useful to resize the component according to the space available in its display area. If the display area is larger than the component, then the component should stretch and occupy the display area horizontally or vertically and it is decided by the fill constraint. Possible values are as follows:

`GridBagConstraints.NONE` (the default) `GridBagConstraints.HORIZONTAL` (make the component wide enough to fill its display area horizontally, but don't change its height) `GridBagConstraints.VERTICAL` (make the component tall enough to fill its display area vertically, but don't change its width) `GridBagConstraints.BOTH` (make the component fill its display area entirely). The effect of fill constraint is shown in Figures 29.6(a), (b) and (c).

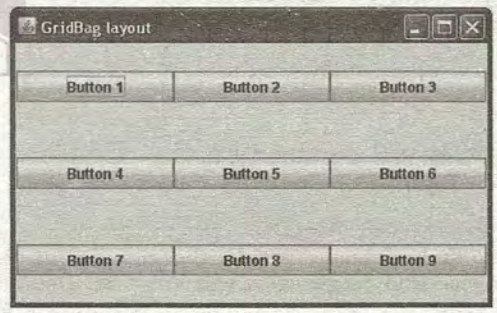


Figure 29.6 (a) Filling horizontally

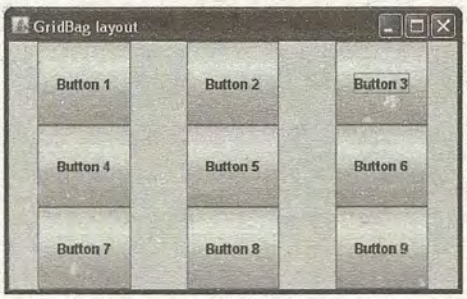


Figure 29.6 (b) Filling vertically

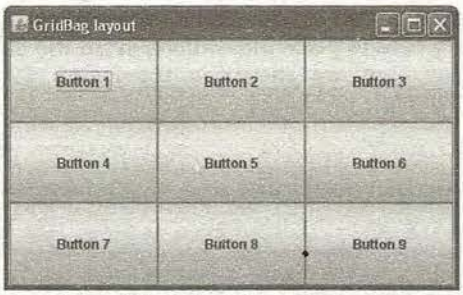


Figure 29.6 (c) Filling in both directions

- `GridBagConstraints.insets`: This constraint is useful to leave some space around the component at the four edges of component. This space is left around the component and the boundary of its display area. insets is the object of Insets class, so it is created as:

```
Insets insets = new Insets(5,10,15,20);
```

Here, we are leaving 5px at top of the component, 10px at left, 15px at bottom and 20px at the right of the component. See Figure 29.7.

By default, it is given as:

```
Insets insets = new Insets(0,0,0,0);
```

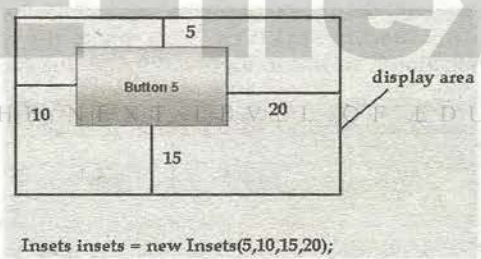


Figure 29.7 The effect of insets constraint

- `GridBagConstraints.ipadx, GridBagConstraints.ipady`: `ipadx` and `ipady` are useful t leave space horizontally and vertically within the component. After adding the space, th components size width-wise and height-wise will increase. Default value is 0. What happen when `ipadx` and `ipady` values are set to a component is shown in Figure 29.8.

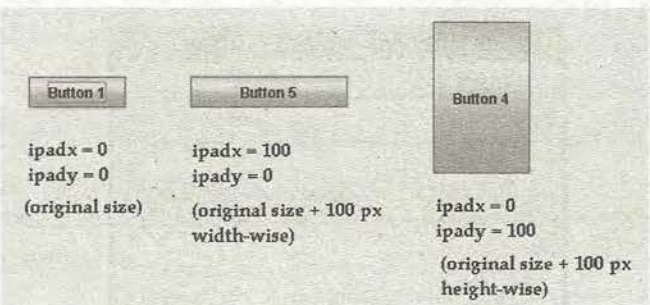


Figure 29.8 The effect of ipadx and ipady constraints

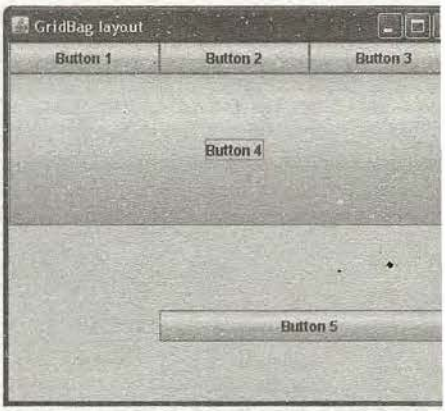
Program 6: This program is designed to display 5 push buttons using grid bag layout manager at certain positions. Please observe the output of the program first and then observe one by one the constraints set to the buttons.

```
//GridBagLayout demo
import java.awt.*;
import javax.swing.*;
class GridBagLayoutDemo extends JFrame
{
      //vars
      GridBagLayout gbag;
      GridBagConstraints cons;

      GridBagLayoutDemo()
      {
            //get the content pane
            Container c = getContentPane();

            //create GridBagLayout object
            gbag = new GridBagLayout();

            //set gridbag layout to content pane
            c.setLayout(gbag);

            //create GridBagConstraints object
            cons = new GridBagConstraints();

            //create 5 push buttons
            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");

            //for all buttons, use horinzontal filling
            cons.fill = GridBagConstraints.HORIZONTAL;

            //display button1 at x,y coordinates 0,0
            cons.gridx = 0;
            cons.gridy = 0;

            //resize all the components when the frame is resized
            cons.weightx = 0.7;
            //cons.weighty = 0.7;

            //set the above constraints to button1
            gbag.setConstraints(b1,cons);

            //add button1 to content pane
            c.add(b1);

            //display button2 at x,y coordinates 1,0
            cons.gridx = 1;
            cons.gridy = 0;

            //remaining constraints applicable as set for previous button
            //set constraints to button2
            gbag.setConstraints(b2,cons);
            c.add(b2);

            //display button3 at x,y coordinates 2,0
            cons.gridx = 2;
            cons.gridy = 0;

            //remaining constraints applicable as set for previous button
            //set constraints to button3
            gbag.setConstraints(b3,cons);
```

```
                c.add(b3);

                //display button4 at x,y coordinates 0,1
                cons.gridx = 0;
                cons.gridy = 1;

                //add 100 px height-wise
                cons.ipady = 100;

                //let button4 occupy 3 columns width-wise
                cons.gridwidth = 3;

                //remaining constraints applicable as set for previous button
                //set constraints to button4
                gbag.setConstraints(b4,cons);
                c.add(b4);

                //display button5 at x,y coordinates 1,2
                cons.gridx = 1;
                cons.gridy = 2;

                //reset the ipady value to 0
                cons.ipady = 0;

                //leave space above the button for resizing vertically
                cons.weighty = 0.8;

                //position the button starting from center of bottom line
                cons.anchor = GridBagConstraints.PAGE_END ;

                //leave 50px space at bottom of button5
                cons.insets = new Insets(0,0,50,0);

                //let the button occupy 2 columns width
                cons.gridwidth = 2;

                //set constraints to button5
                gbag.setConstraints(b5,cons);
                c.add(b5);

        }

        public static void main(String args[])
        {
                //create the frame
                GridBagLayoutDemo demo = new GridBagLayoutDemo();
                demo.setSize(400,400);
                demo.setTitle("GridBag layout");
                demo.setVisible(true);
                demo.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }
}
```

Output:

```
C:\> javac GridBagLayoutDemo.java
C:\> java GridBagLayoutDemo
```



BoxLayout

BoxLayout of `javax.swing` package allows multiple components t
horizontally. The components will not wrap so, for example, a ver
will stay vertically arranged when the frame is resized. We can cr
the combinations shown below:

- `BoxLayout box = new BoxLayout(JPanel object, axis-o`

Here, all the components which are inside a JPanel object are arr
This 'axis-orientation' can be:

- `BoxLayout.X_AXIS`: Here, components are arranged along x-a
- `BoxLayout.Y_AXIS`: Here, components are arranged along y-a
- `BoxLayout.LINE_AXIS`: Here, components are arranged like li
- `BoxLayout.PAGE_AXIS`: Here, components are arranged like li

Program 7: Write a program to understand the usage of BoxLayout

```
//BoxLayout Demo
import java.awt.*;
import javax.swing.*;
class BoxLayoutDemo extends JFrame
{
      BoxLayoutDemo()
      {
            //create container and set flow layout
            Container c = getContentPane();
            c.setLayout(new FlowLayout());

            //create a JPanel object which holds co
            MyPanel1 mp1 = new MyPanel1();

            //add the panel to content pane
            c.add(mp1);

            //create another JPanel object which ho
            //of components
            MyPanel2 mp2 = new MyPanel2();

            //add the panel to content pane
            c.add(mp2);
      }
```

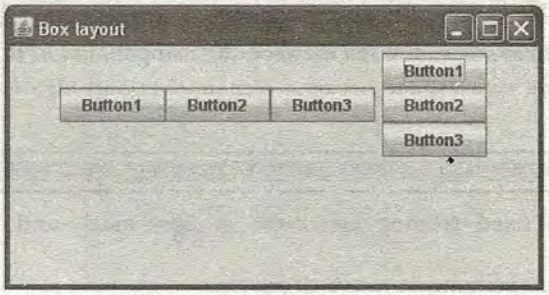
```
        public static void main(String args[])
        {
                //create the frame
                BoxLayoutDemo demo = new BoxLayoutDemo();
                demo.setSize(400,400);
                demo.setTitle("Box layout");
                demo.setVisible(true);
                demo.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }
}
class MyPanel1 extends JPanel
{
        MyPanel1()
        {
                //create BoxLayout object to arrange components along x-axis
                BoxLayout box1 = new BoxLayout(this, BoxLayout.X_AXIS);

                //set box layout to JPanel
                setLayout(box1);

                //create 3 push buttons and add them to panel using box layout
                JButton b1, b2, b3;
                b1 = new JButton("Button1");
                b2 = new JButton("Button2");
                b3 = new JButton("Button3");

                add(b1);
                add(b2);
                add(b3);
        }
}

class MyPanel2 extends JPanel
{
        MyPanel2()
        {
                //create BoxLayout object to arrange components along y-axis
                BoxLayout box2 = new BoxLayout(this, BoxLayout.Y_AXIS);

                //set box layout to JPanel
                setLayout(box2);

                //create 3 push buttons and add them to panel using box layout
                JButton b1, b2, b3;
                b1 = new JButton("Button1");
                b2 = new JButton("Button2");
                b3 = new JButton("Button3");

                add(b1);
                add(b2);
                add(b3);
        }
}
```

Output:

```
C:\> javac BoxLayoutDemo.java
C:\> java BoxLayoutDemo
```

This program uses box layout to arrange some components along x-axis and some more components along y-axis. The components which are arranged along x-axis are added inside a JPanel object where box layout is used and that object is added to content pane. Similarly, the components which are arranged along y-axis are added inside another JPanel object where box

layout is used and that object is also added to content pane. The two JPanel objects are created which are added to the content pane by using flow layout.



Box Class

BoxLayout is implemented with the help of Box class defined in `javax.swing` package. We can imagine the Box class object as an invisible area where the components can be placed by using box layout. Box class provides basically two types of boxes, horizontal box and vertical box.

The following methods help us to work with box layout:

- To create a box where the components are arranged from left to right, we can use `createHorizontalBox()` method, as:

```
Box b = Box.createHorizontalBox();
```

- In case, we want to arrange the components vertically from top to bottom, we can use `createVeriticalBox()` method.

```
Box b = Box.createVerticalBox();
```

- Now, the components can be added to Box object using `add()` method.

```
b.add(component);
```

- The size of components can be set by using `setPreferredSize()`, `setMinimumSize()` and `setMaximumSize()` methods of `JComponent` class.
  - `setPreferredSize()` : This method tells the container to use some preferred size for the component so that the component fits nicely in the container.
  - `setMinimumSize()` : This method sets some minimum size of the component.
  - `setMaximumSize()` : This method sets the maximum size of the component.

Similarly, to retrieve the size of components, we can use `getPreferredSize()`, `getMinimumSize()` and `getMaximumSize()` methods.

- By default, there will be no space left between the components in a box layout. To add space we need fillers. There are three types of fillers defined in Box class.
  - **Struts:** to leave some space between components.

```
b.add(Box.createHorizontalStrut(20));
```

This strut leaves horizontal space of 20px between the components in the box.

```
b.add(Box.createVerticalStrut(20));
```

This method adds a vertical space of 20px between the components in the box.

- **Rigid areas:** to leave some space between components and also to set the vertical size of the box.

```
b.add(Box.createRigidArea(new Dimension(5,20));
```

This method leaves a fixed rectangular space of 5px width and 20px height between the components.

- **Glue:** to separate components as much as possible.

```
b.add(Box.createHorizontalGlue())
```

This method throws the components maximum distance that is possible horizontally.

```
b.add(Box.createVerticalGlue())
```

This method adds vertical glue in the box so that the components are kept at a maximum distance that is possible vertically.

Let us design a program to understand how to use Box class:

- First, we create a horizontal box to place components, for example, a label and a text field. For this purpose, we create the components, i.e., a label and text field, as:

```
JLabel l1= new JLabel("Enter Name:");
JTextField t1= new JTextField(20);
```

Then, we set the maximum size of the text field to the preferred size, as:

```
t1.setMaximumSize(t1.getPreferredSize());
```

Then, we create a horizontal box and add the components in the box, as:

```
Box horiz1 = Box.createHorizontalBox();
horiz1.add(l1);
horiz1.add(t1);
```

Suppose, we wish to leave some horizontal space between the label and text field, we can use strut as:

```
horiz1.add(Box.createHorizontalStrut(20));  //gap 20 px
```

- In the same way, we create a second horizontal box to place another label and a text field. The same steps described earlier can be used in this case also.
- Finally, We create a third horizontal box where we want to place two push buttons.
- Now, we create a vertical box and add the preceding three horizontal boxes inside it, as:

```
Box vert = Box.createVerticalBox();
vert.add(horiz1);
vert.add(horiz2);
vert.add(horiz3);
```

- To leave some vertical space between the three horizontal boxes, we can add a vertical strut between the boxes as:

```
vert.add(Box.createVerticalStrut(100));  //gap of 100 px
```

The preceding steps are shown in Figure 29.9 and are implemented in Program 8.

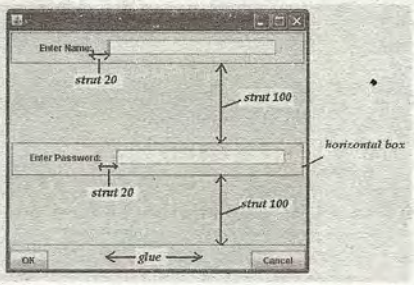


Figure 20.9 Components arranged using Box class.

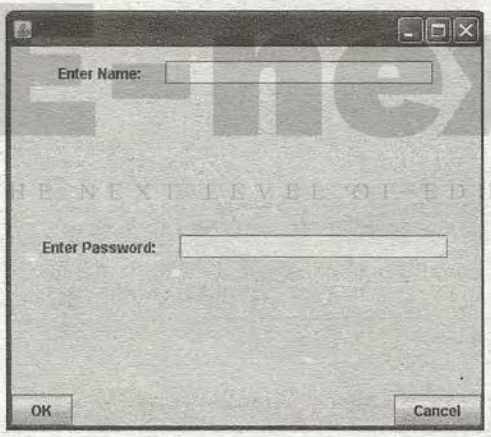
Program 8: Write a program to create three horizontal boxes with components and finally add them inside another vertical box.

```
//Box layout manager
import java.awt.*;
import javax.swing.*;

class BoxDemo extends JFrame
{
      BoxDemo()
      {
            //create a label and a text field
            JLabel l1= new JLabel("Enter Name:");
            JTextField t1= new JTextField(20);
            t1.setMaximumSize(t1.getPreferredSize());

            //create top horizontal box and add above components to it
            Box horiz1 = Box.createHorizontalBox();
            horiz1.add(l1);
            horiz1.add(Box.createHorizontalStrut(20));  //gap 20 pxl
            horiz1.add(t1);

            //create a label and a text field
            JLabel l2= new JLabel("Enter Password:");
            JTextField t2= new JTextField(20);
            t2.setMaximumSize(t2.getPreferredSize());

            //create a middel horizontal box and add above components to it
            Box horiz2 = Box.createHorizontalBox();
            horiz2.add(l2);
            horiz2.add(Box.createHorizontalStrut(20)); //gap 20 px
            horiz2.add(t2);

            //create two push buttons
            JButton b1= new JButton("OK");
            JButton b2= new JButton("Cancel");

            //construct the bottom horizontal box and add components in it
            Box horiz3 = Box.createHorizontalBox();
            horiz3.add(b1);
            horiz3.add(Box.createHorizontalGlue()); //throw buttons apart
            horiz3.add(b2);
```

```
                //add the three horizontal boxes inside a vertical box
                Box vert = Box.createVerticalBox();
                vert.add(horiz1);
                vert.add(Box.createVerticalStrut(100));  //gap of 100 px
                vert.add(horiz2);
                vert.add(Box.createVerticalStrut(100));
                vert.add(horiz3);

                //add the vertical box to the content pane
                Container c = getContentPane();
                c.add(vert);
        }
        public static void main(String args[])
        {
                //create frame
                BoxDemo bd = new BoxDemo();
                bd.setSize(400,350);
                bd.setVisible(true);
                bd.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        }
}
```

Output:

```
C:\> javac BoxDemo.java
C:\> java BoxDemo
```



Conclusion

Layout managers are useful to arrange a group of components in a particular manner in a container as wished by the programmer. FlowLayout is the simplest layout manager and GridBagLayout is the most flexible and complex layout manager among all other layouts. While working with a group of components, the programmer should first plan his output and accordingly decide which layout should be used in a particular application. Then only he can start writing his program.