



UNIT-IV

LAYOUT MANAGEMENT-

SWING

Presented by,
Rahul Raj. M
Guest Faculty
DCA,CUSAT

LAYOUT MANAGEMENT

- Layout refers to the **arrangement of components within the container**. In another way, it could be said that layout is **placing the components at a particular position within the container**. The task of laying out the controls is done **automatically by the Layout Manager**.
- The layout manager automatically positions all the components within the container. Even if you do not use the layout manager, the components are still positioned by the **default layout manager**.
- Java provides various layout managers to position the controls. **Properties like size, shape, and arrangement varies** from one layout manager to the other.

AWT Layout Manager Classes

1. **BorderLayout-** The BorderLayout arranges the components to fit in the five regions: **east, west, north, south, and center**.
2. **GridLayout-** The GridLayout manages the components in the form of a **rectangular grid**.
3. **GroupLayout-** The GroupLayout **hierarchically groups the components** in order to position them in a Container.
4. **FlowLayout-** The FlowLayout is the **default layout**. It layout the components in a **directional flow**.
5. **CardLayout-** The CardLayout object treats each **component** in the container **as a card**. Only **one card is visible at a time**.

6. **GridBagLayout-** This is the most flexible layout manager class. The object of GridBagLayout **aligns** the component **vertically, horizontally, or along their baseline without requiring the components of the same size.**
7. **SpringLayout-** A SpringLayout **positions the children** of its associated container **according to a set of constraints.**
8. Scroll Panel Layout:

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:

- **public static final int NORTH**
- **public static final int SOUTH**
- **public static final int EAST**
- **public static final int WEST**
- **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();
    }
}
```

GridLayout

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

Constructors of GridLayout class

GridLayout():

- creates a grid layout with **one column per component in a row**.

GridLayout(int rows, int columns):

- creates a grid layout with the given rows and columns but **no gaps** between the components.

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**.


```

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);

//creating grid layout of 3 row and 3 columns
    f.setLayout(new GridLayout(3,3));
    f.setSize(300,300);
    f.setVisible(true);
}
public static void main(String[] args) {
    new MyGridLayout();
}
}

```



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

- public static final int **LEFT**
 - public static final int **RIGHT**
 - public static final int **CENTER**
 - public static final int **LEADING**
 - public static final int **TRAILING**
-
- **LEADING** –left to right organization
 - **TRAILING** – left to right organization



Constructors of FlowLayout class

FlowLayout():

- creates a flow layout with centered alignment and a **default 5 unit horizontal and vertical gap**.

FlowLayout(int align):

- creates a flow layout with the **given alignment and a default 5 unit horizontal and vertical gap**.

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 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();
    }
}
```



BoxLayout

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

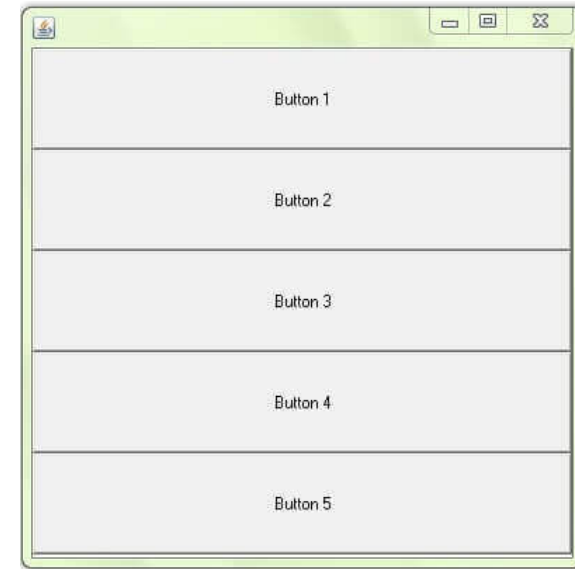
Fields of BoxLayout class

- public static final int **X_AXIS**
- public static final int **Y_AXIS**
- public static final int **LINE_AXIS**
- public static final int **PAGE_AXIS**

Constructor of BoxLayout class

BoxLayout(Container c, int axis):

- creates a box layout that arranges the components with the given axis.



- **LINE_AXIS** - Components are laid out the way words are laid out in a line, based on the container's `ComponentOrientation` property. If the container's `ComponentOrientation` is horizontal then components are laid out horizontally, otherwise they are laid out vertically. For horizontal orientations, if the container's `ComponentOrientation` is left to right then components are laid out left to right, otherwise they are laid out right to left. For vertical orientations components are always laid out from top to bottom.
- **PAGE_AXIS** - Components are laid out the way text lines are laid out on a page, based on the container's `ComponentOrientation` property. If the container's `ComponentOrientation` is horizontal then components are laid out vertically, otherwise they are laid out horizontally. For horizontal orientations, if the container's `ComponentOrientation` is left to right then components are laid out left to right, otherwise they are laid out right to left. For vertical orientations components are always laid out from top to bottom.

```

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.*;

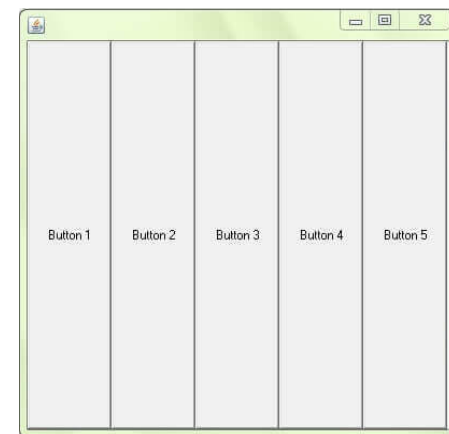
public class BoxLayoutExample2 extends Frame {
    Button buttons[];

    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();
    }
}
```



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

CardLayout():

- creates a card layout with zero horizontal and vertical gap.

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.

```
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() {

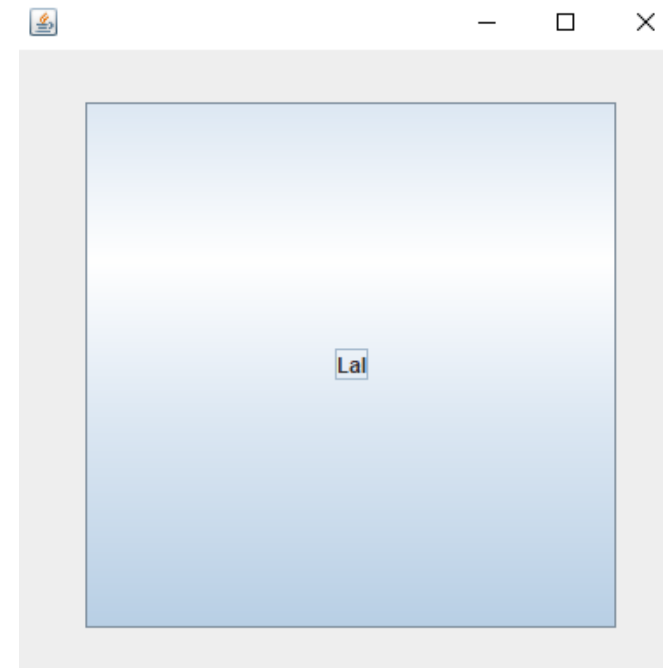
        c=getContentPane();
        card=new CardLayout(40,30);
        //create CardLayout object with 40 hor space and 30 ver space
        c.setLayout(card);

        b1=new JButton("Lal");
        b2=new JButton("Bal");
        b3=new JButton("Pal");
        b1.addActionListener(this);
        b2.addActionListener(this);
        b3.addActionListener(this);

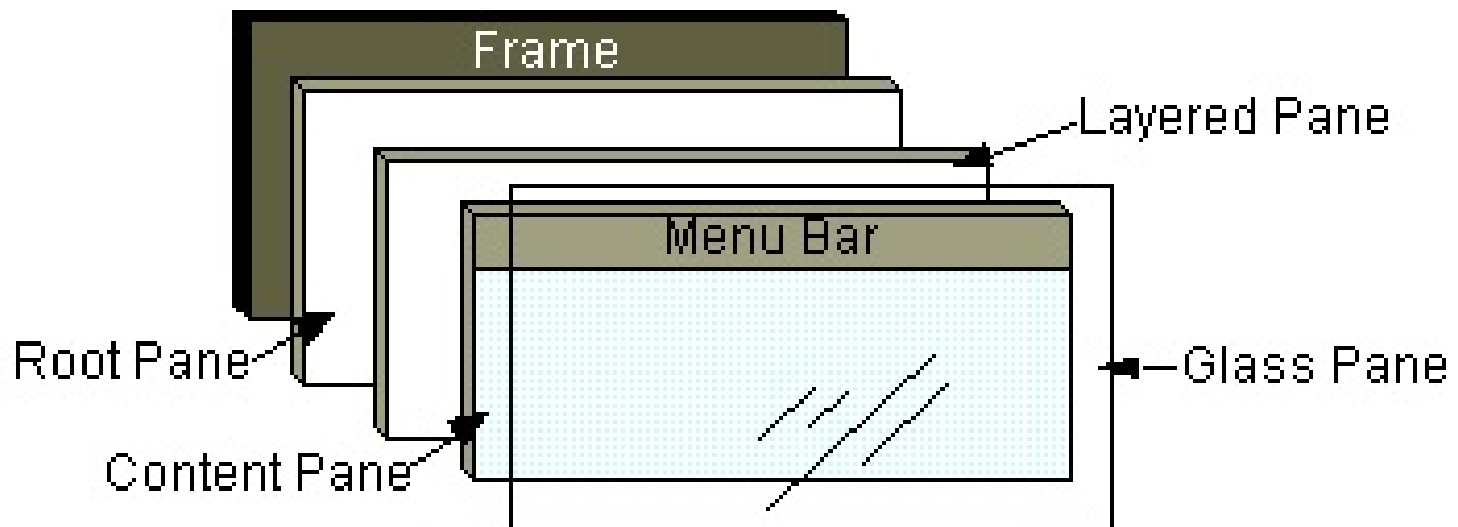
        c.add("a",b1);c.add("b",b2);c.add("c",b3);

    }
}
```

```
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) ;  
}
```



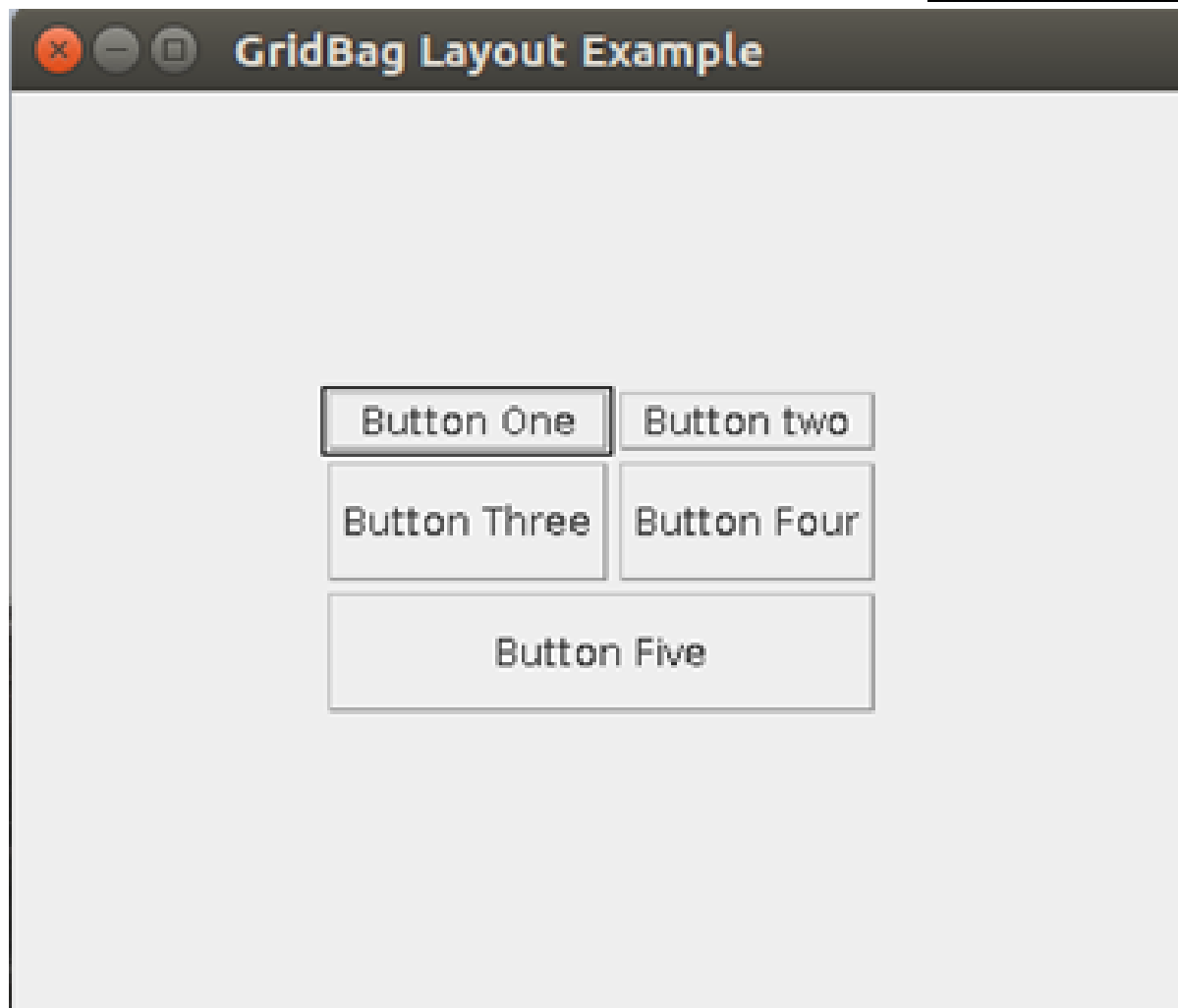
- Every Swing **top level container** (and `JInternalFrame`) has what's called a **JRootPane**. This is responsible for actually **managing** the **overall layout** of the window.
- The **root pane** has a number of layers, one of which is the content pane. When you add something to a frame, it is automatically added to the content pane for you.



- **Glass Pane:** Makes the frame as a transparent glass
- **Layered Pane:** Contain different layers of panes, each suitable for different usages such as pop-ups, dialogues etc.

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.

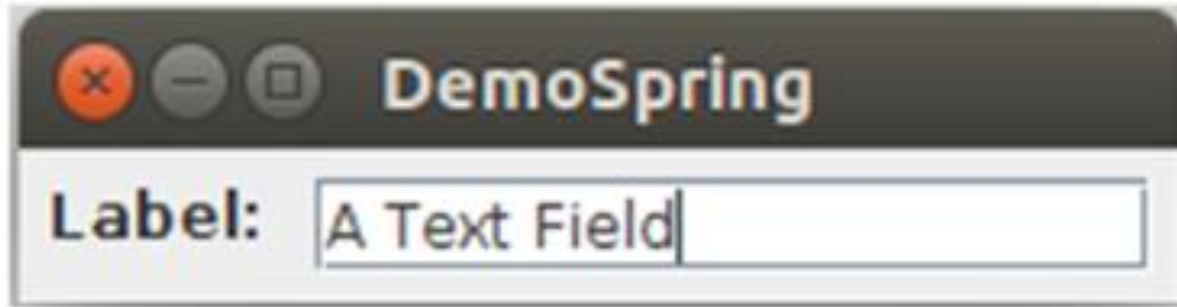


GroupLayout

- GroupLayout groups its **components** and **places** them in a Container **hierarchically**. The grouping is done by instances of the Group class.
- **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 where as **ParallelGroup** aligns its **child on top of each other**.
- The GroupLayout class provides methods such as **createParallelGroup()** and **createSequentialGroup()** to create groups.
- GroupLayout treats each **axis independently**. That is, there is a group representing **the horizontal axis**, and a group representing the **vertical axis**. Each component must exist in both a horizontal and vertical group, otherwise an **IllegalStateException** is thrown during layout, or when the minimum, preferred or maximum size is requested.

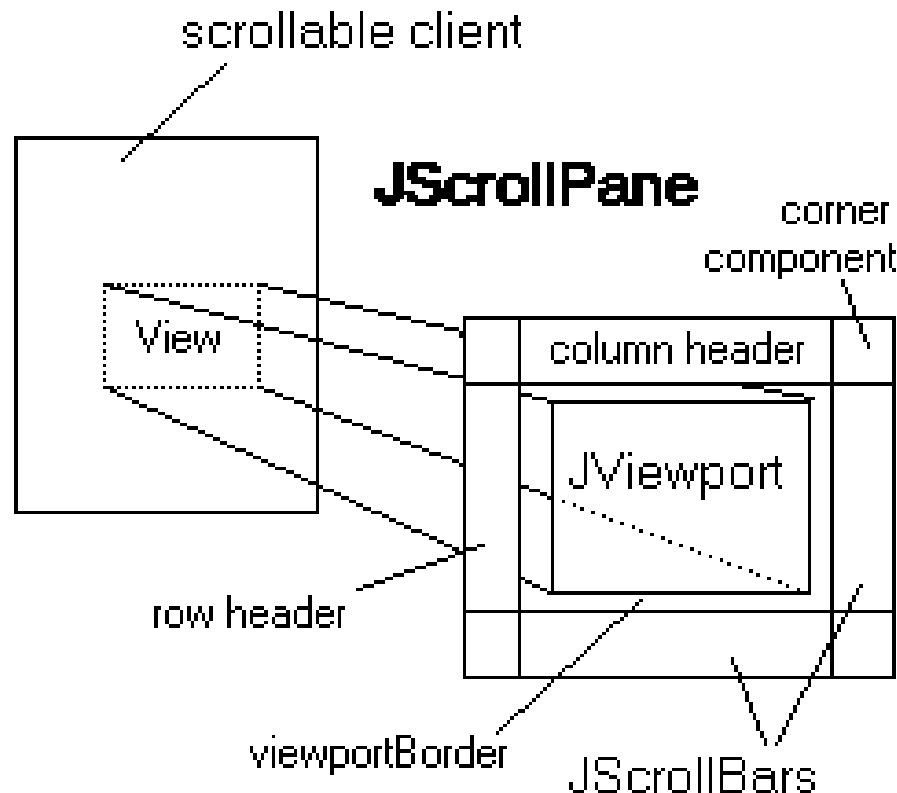
SpringLayout

- A SpringLayout **arranges** the children of its associated **container** according to **a set of constraints**.
- **Constraints** are nothing but **horizontal and vertical distance** between two component edges. Every constraints are represented by a SpringLayout. Constraint object.
- **Each child** of a SpringLayout container, as well as the container itself, has exactly **one set of constraints** associated with them.
- Each edge position is dependent on the position of the other edge. If **a constraint is added** to create new edge than the **previous binding is discarded**. SpringLayout doesn't automatically set the location of the components it manages.



ScrollPaneLayout

- The layout manager used by **JScrollPane**. **ScrollPaneLayout** is responsible for nine components: a **viewport**, two **scrollbars**, a **row header**, a **column header**, and four "corner" components.



```
import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JScrollPane;

public class ScrollPaneDemo extends JFrame
{
    public ScrollPaneDemo() {
        super("ScrollPane Demo");
        ImageIcon img = new ImageIcon("child.png");

        JScrollPane png = new JScrollPane(new JLabel(img));

        getContentPane().add(png);
        setSize(300,250);
        setVisible(true);
    }

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

Output:



Components

- Swing components are basic building blocks of an application.
- Swing has a wide range of various components, including buttons, check boxes, sliders, and list boxes.

JButton

```
import javax.swing.*;  
  
public class FirstSwingExample {  
    public static void main(String[] args) {  
        JFrame f=new JFrame();//creating instance of JFrame  
  
        JButton b=new JButton("click");//creating instance of JButton  
        b.setBounds(130,100,100, 40);//x axis, y axis, width, height  
  
        f.add(b);//adding button in JFrame  
  
        f.setSize(400,500);//400 width and 500 height  
        f.setLayout(null);//using no layout managers  
        f.setVisible(true);//making the frame visible  
    }  
}
```



```
import javax.swing.*;

public class ButtonExample{

    ButtonExample(){

        JFrame f=new JFrame("Button Example");

        JButton b=new JButton(new ImageIcon("D:\\icon.png"));

        b.setBounds(100,100,100, 40);

        f.add(b);

        f.setSize(300,400);

        f.setLayout(null);

        f.setVisible(true);

        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    }

    public static void main(String[] args) {

        new ButtonExample();

    }

}
```

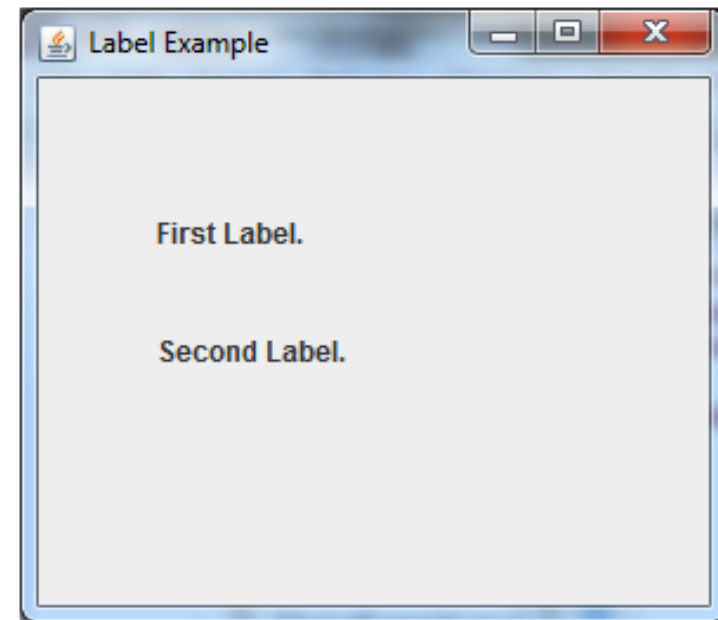


- The **EXIT_ON_CLOSE** operation **exits the program when your user closes the frame**. This behavior is appropriate for this program because the program has only one frame, and closing the frame makes the program useless.

JLabel

```
import javax.swing.*;

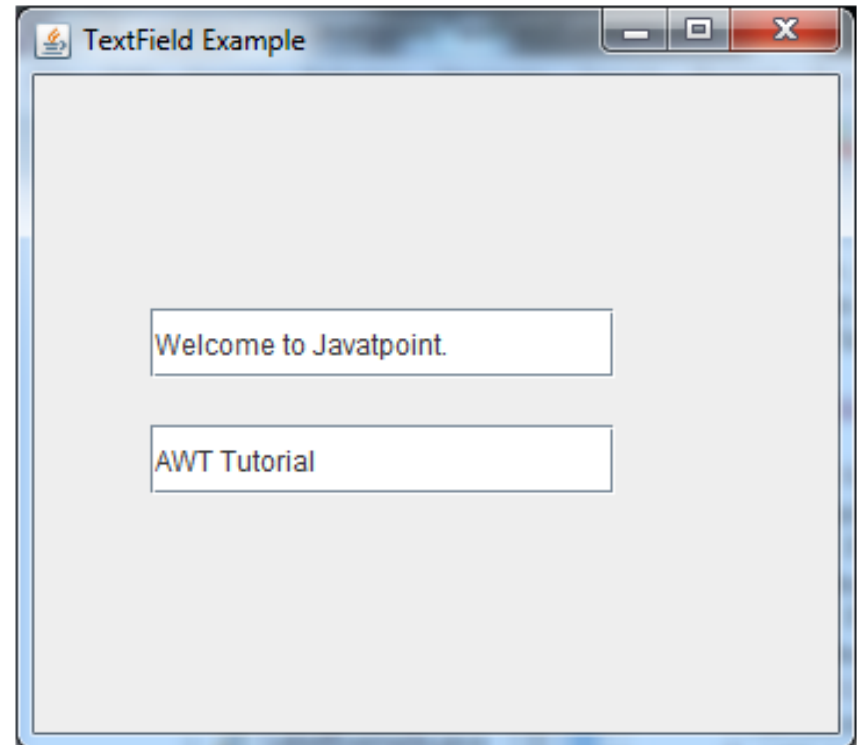
class LabelExample
{
    public static void main(String args[])
    {
        JFrame f= new JFrame("Label Example");
        JLabel l1,l2;
        l1=new JLabel("First Label.");
        l1.setBounds(50,50, 100,30);
        l2=new JLabel("Second Label.");
        l2.setBounds(50,100, 100,30);
        f.add(l1); f.add(l2);
        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```



JTextField

```
import javax.swing.*;

class TextFieldExample
{
    public static void main(String args[])
    {
        JFrame f= new JFrame("TextField Example");
        JTextField t1,t2;
        t1=new JTextField("Welcome to Javatpoint.");
        t1.setBounds(50,100, 200,30);
        t2=new JTextField("AWT Tutorial");
        t2.setBounds(50,150, 200,30);
        f.add(t1); f.add(t2);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
}
```

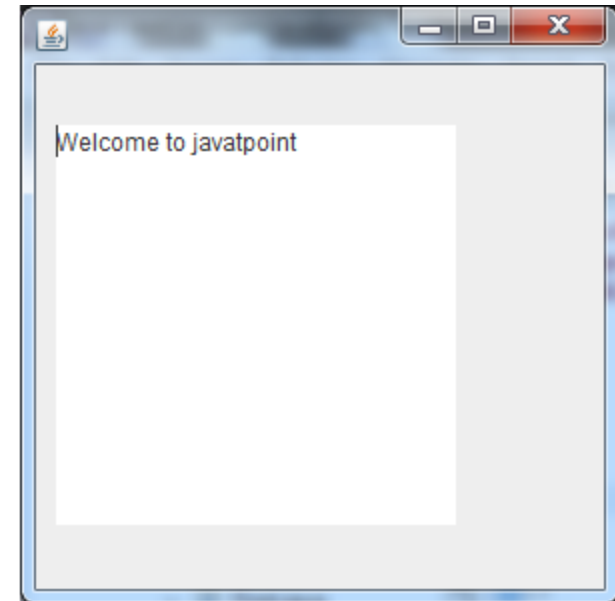


JTextArea

```
import javax.swing.*;

public class TextAreaExample
{
    TextAreaExample(){
        JFrame f= new JFrame();
        JTextArea area=new JTextArea("Welcome to javatpoint");
        area.setBounds(10,30, 200,200);
        f.add(area);
        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);
    }

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



JPasswordField

```
import javax.swing.*;

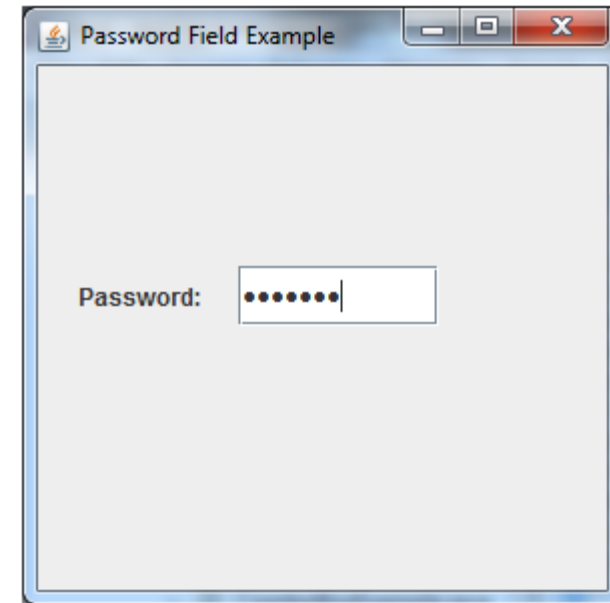
public class PasswordFieldExample {

    public static void main(String[] args) {

        JFrame f=new JFrame("Password Field Example");
        JPasswordField value = new JPasswordField();
        JLabel l1=new JLabel("Password:");
        l1.setBounds(20,100, 80,30);
        value.setBounds(100,100,100,30);
        f.add(value); f.add(l1);
        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);

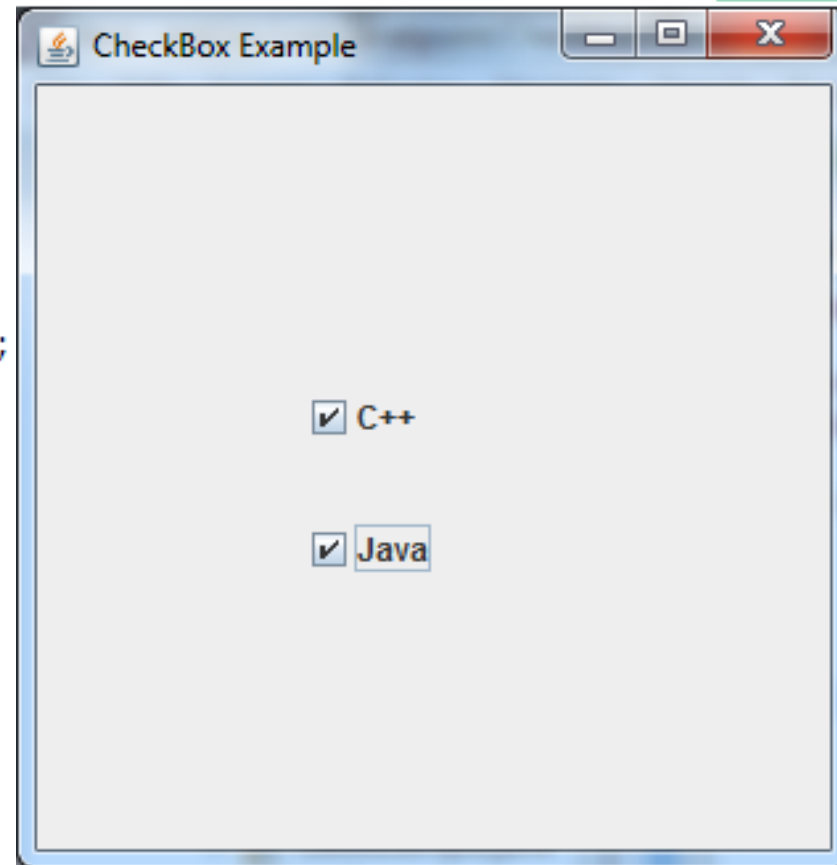
    }

}
```



JCheckBox

```
import javax.swing.*;  
  
public class CheckBoxExample  
{  
    CheckBoxExample(){  
        JFrame f= new JFrame("CheckBox Example");  
        JCheckBox checkBox1 = new JCheckBox("C++");  
        checkBox1.setBounds(100,100, 50,50);  
        JCheckBox checkBox2 = new JCheckBox("Java", true);  
        checkBox2.setBounds(100,150, 50,50);  
        f.add(checkBox1);  
        f.add(checkBox2);  
        f.setSize(400,400);  
        f.setLayout(null);  
        f.setVisible(true);  
    }  
  
    public static void main(String args[])  
    {  
        new CheckBoxExample();  
    }  
}
```



JRadioButton

```
import javax.swing.*;

public class RadioButtonExample {

    JFrame f;

    RadioButtonExample(){

        f=new JFrame();

        JRadioButton r1=new JRadioButton("A) Male");
        JRadioButton r2=new JRadioButton("B) Female");

        r1.setBounds(75,50,100,30);
        r2.setBounds(75,100,100,30);

        ButtonGroup bg=new ButtonGroup();
        bg.add(r1);bg.add(r2);

        f.add(r1);f.add(r2);

        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);

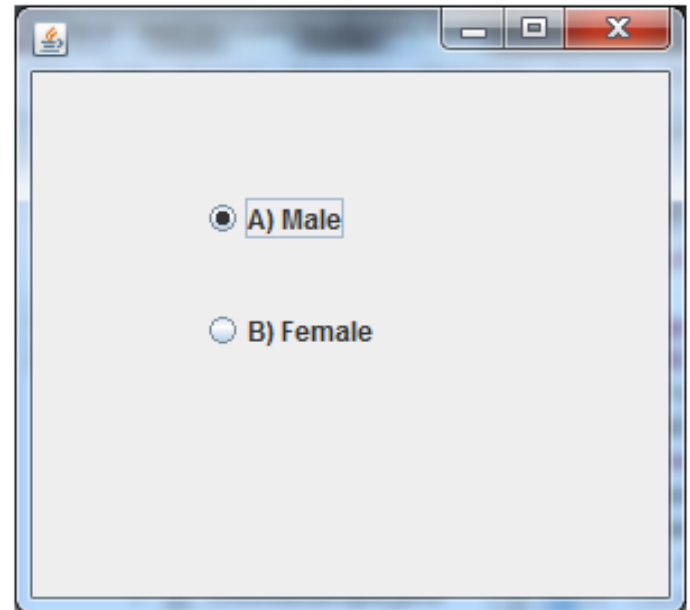
    }

    public static void main(String[] args) {

        new RadioButtonExample();

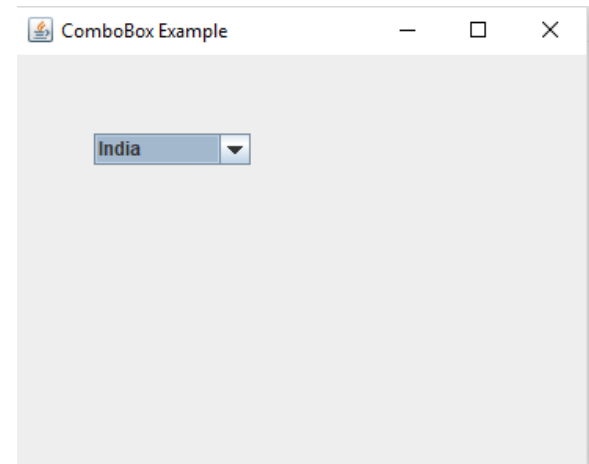
    }

}
```



JComboBox

```
import javax.swing.*;
public class ComboBoxExample {
    JFrame f;
    ComboBoxExample() {
        f=new JFrame("ComboBox Example");
        String country[]={"India", "Bangladesh", "Myanmar",
            "Srilanka", "Thailand", "Nepal", "Bhutan"};
        JComboBox cb=new JComboBox(country);
        cb.setBounds(50, 50, 100, 20);
        f.add(cb);
        f.setLayout(null);
        f.setSize(400, 500);
        f.setVisible(true);
    }
    public static void main(String[] args) {
        new ComboBoxExample();
    }
}
```

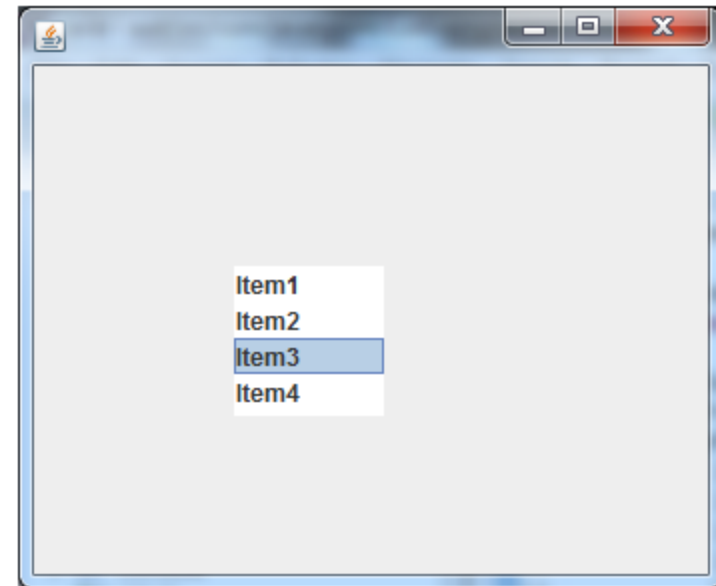


JList

```
import javax.swing.*;

public class ListExample
{
    ListExample(){
        JFrame f= new JFrame();
        DefaultListModel<String> l1 = new DefaultListModel<>();
        l1.addElement("Item1");
        l1.addElement("Item2");
        l1.addElement("Item3");
        l1.addElement("Item4");
        JList<String> list = new JList<>(l1);
        list.setBounds(100,100, 75,75);
        f.add(list);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }

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



- **DefaultListModel<E>:** A list model which stores list as vector.

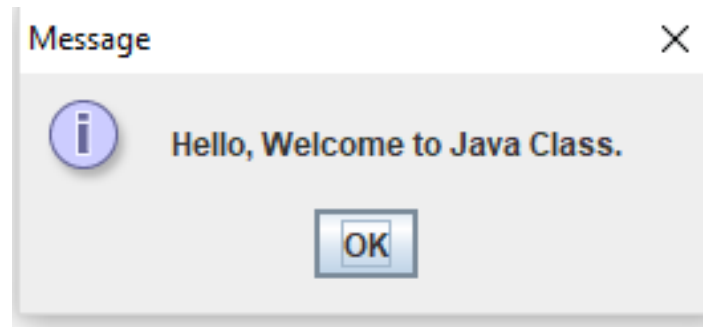
Dialogue Box

```
import javax.swing.*;

public class OptionPaneExample {
    JFrame f;

    OptionPaneExample() {
        f=new JFrame();
        JOptionPane.showMessageDialog(f,"Hello, Welcome to Java Class.");
    }

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

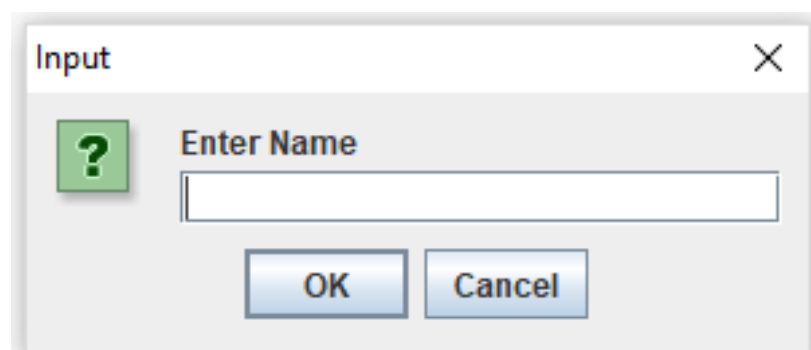


```
import javax.swing.*;

public class OptionPaneExample {
    JFrame f;

    OptionPaneExample(){
        f=new JFrame();
        String name=JOptionPane.showInputDialog(f,"Enter Name");
    }

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



Jtable

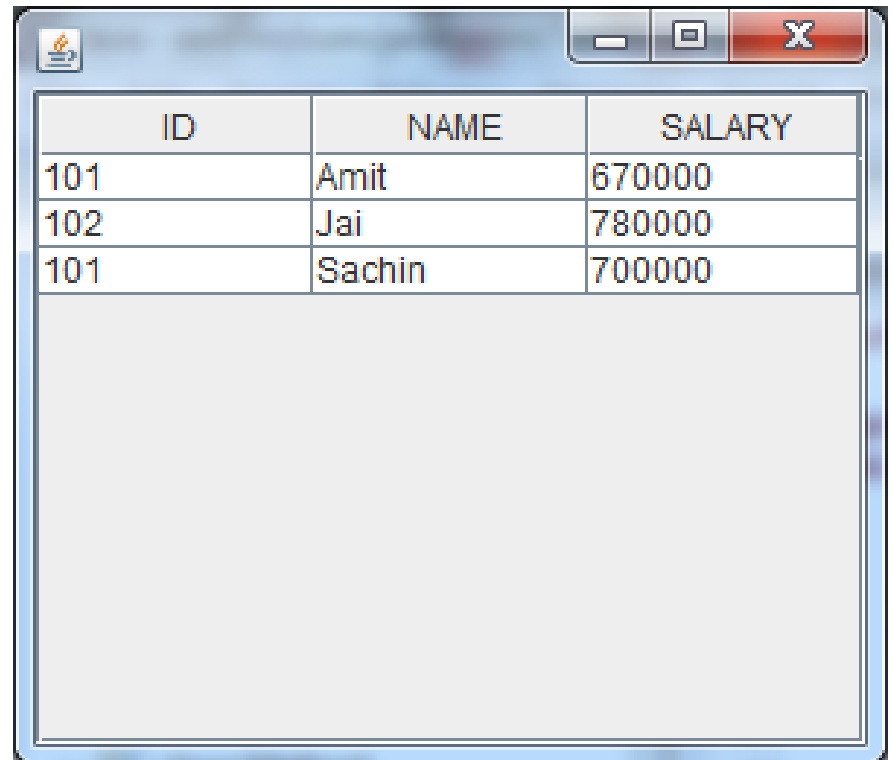
```
import javax.swing.*;

public class TableExample {
    JFrame f;

    TableExample(){
        f=new JFrame();
        String data[][]={ {"101","Amit","670000"},
                           {"102","Jai","780000"},
                           {"101","Sachin","700000"}};

        String column[]={"ID","NAME","SALARY"};
        JTable jt=new JTable(data,column);
        jt.setBounds(30,40,200,300);
        JScrollPane sp=new JScrollPane(jt);
        f.add(sp);
        f.setSize(300,400);
        f.setVisible(true);
    }

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



ID	NAME	SALARY
101	Amit	670000
102	Jai	780000
101	Sachin	700000