Introduction to Swing:

Swing:

Java Swing is a part of **Java Foundation Classes (JFC)** that *is used to create window-based applications*. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java.

Unlike AWT, Java Swing provides platform-independent and lightweight components.

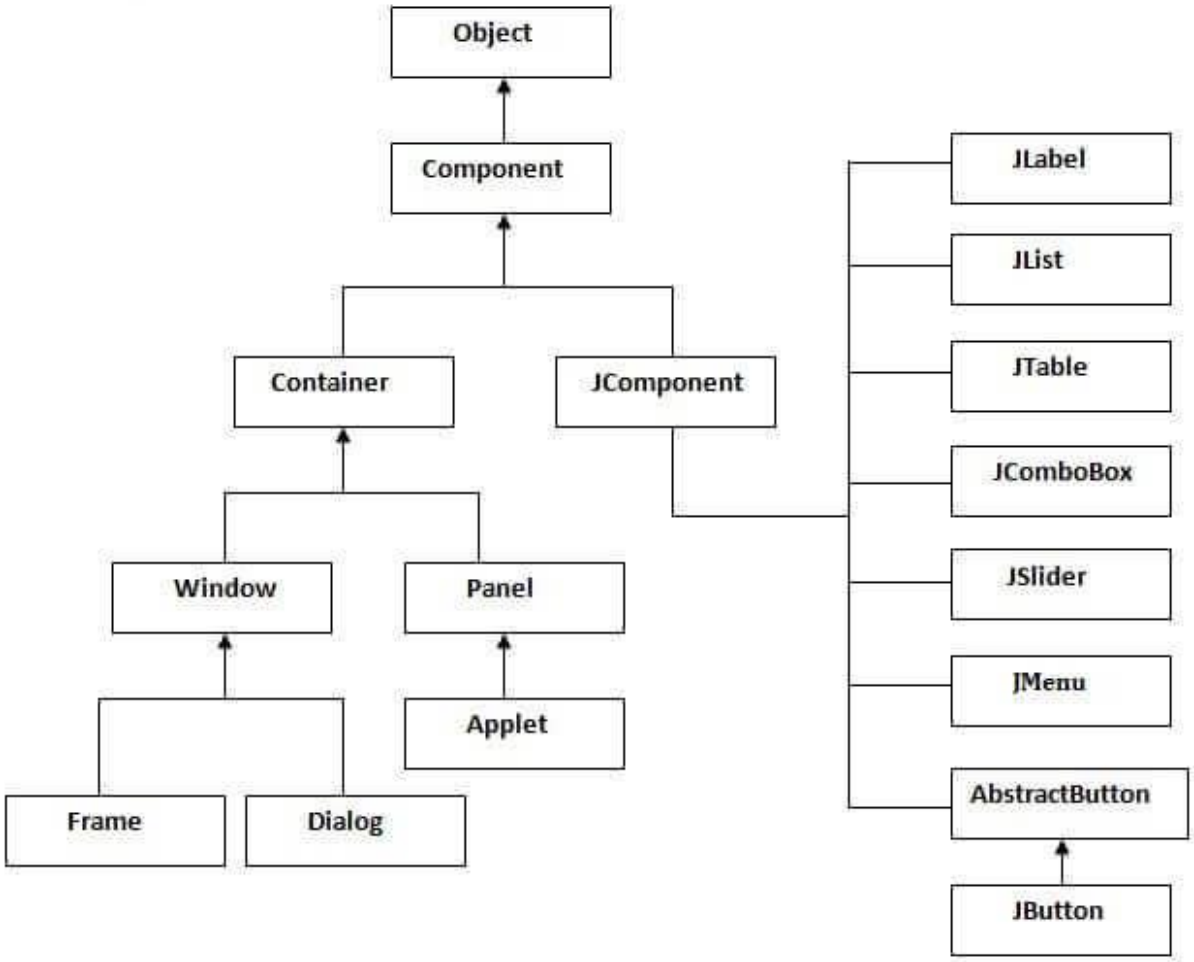
The **javax.swing** package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

What is JFC

The Java Foundation Classes (JFC) are a set of GUI components which simplify the development of desktop applications.

Hierarchy of Java Swing classes

The hierarchy of java swing API is given below.



Java Swing Examples

There are two ways to create a frame:

- By creating the object of Frame class (association)
- By extending Frame class (inheritance)

We can write the code of swing inside the main(), constructor or any other method.

- By creating the object of Frame class (association)

1.By creating the object of Frame class (association)

Simple Java Swing Example

Let's see a simple swing example where we are creating one button and adding it on the JFrame object inside the main() method.

```
import javax.swing.*;
public class App{

    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
    }

}
```

Example of Swing by Association inside constructor

We can also write all the codes of creating JFrame, JButton and method call inside the java constructor.

```
import javax.swing.*;
public class App{
    App()
    {
        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
    }

    public static void main(String[] args)
    {
        App app=new App();
```

```
    }

}
```

2.Simple example of Swing by inheritance

We can also inherit the JFrame class, so there is no need to create the instance of JFrame class explicitly.

```
import javax.swing.*;
public class App extends JFrame{
    JFrame f;
    App()
    {
        JButton b=new JButton("click");//create button
        b.setBounds(130,100,100, 40);

        add(b);//adding button on frame
        setSize(400,500);
        setLayout(null);
        setVisible(true);
    }

    public static void main(String[] args)
    {
        App app=new App();
    }

}
```

GUI Elements:

1. JLabel
2. JRadioButton
3. ButtonGroup
4. JCheckBox
5. JTextField
6. JTextArea
7. JButton
8. Border

9. JComboBox
10. JTabbedPane
11. JPasswordField
12. Look and Feel Management in Java Swing

JLabel

The object of the **JLabel** class may be a component for putting text in a container. It's used to display one line of read-only text. The text is often changed by an application but a user cannot edit it directly. It inherits the **JComponent** class.

Syntax:

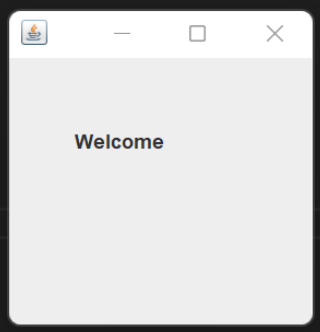
JLabel jl = new JLabel();

JLabel Constructors

- **JLabel():** It is used to create a JLabel instance with no image and with an empty string for the title.
- **JLabel(String s):** It is used to create a JLabel instance with the specified text.
- **JLabel(Icon i):** It is used to create a JLabel instance with the specified image.
- **JLabel(String s, Icon I, int horizontalAlignment):** It is used to create a JLabel instance with the specified text, image, and horizontal alignment.

Example:

```
import javax.swing.*;

public class App extends JFrame {

    Run | Debug
    public static void main(String[] args)
    {
        JFrame a=new JFrame();
        JLabel lbl = new JLabel(text: "Welcome");
        lbl.setBounds(x: 40,y: 40,width: 90,height: 20);
        a.add(lbl);
        a.setSize(width: 200,height: 200);
        a.setLayout(manager: null);
        a.setVisible(b: true);

    }
}
```



Code:

```
import javax.swing.*;
```

```
public class App extends JFrame {

    public static void main(String[] args)
    {
        JFrame a=new JFrame();
        JLabel lbl = new JLabel("Welcome");
        lbl.setBounds(40,40,90,20);
        a.add(lbl);
        a.setSize(200,200);
        a.setLayout(null);
        a.setVisible(true);

    }
}
```

JRadioButton

This component allows the user to select only one item from a group item. By using the JRadioButton component you can choose one option from multiple options.

Syntax**: JRadioButton jrb = new JRadioButton();**

JRadioButton Constructors

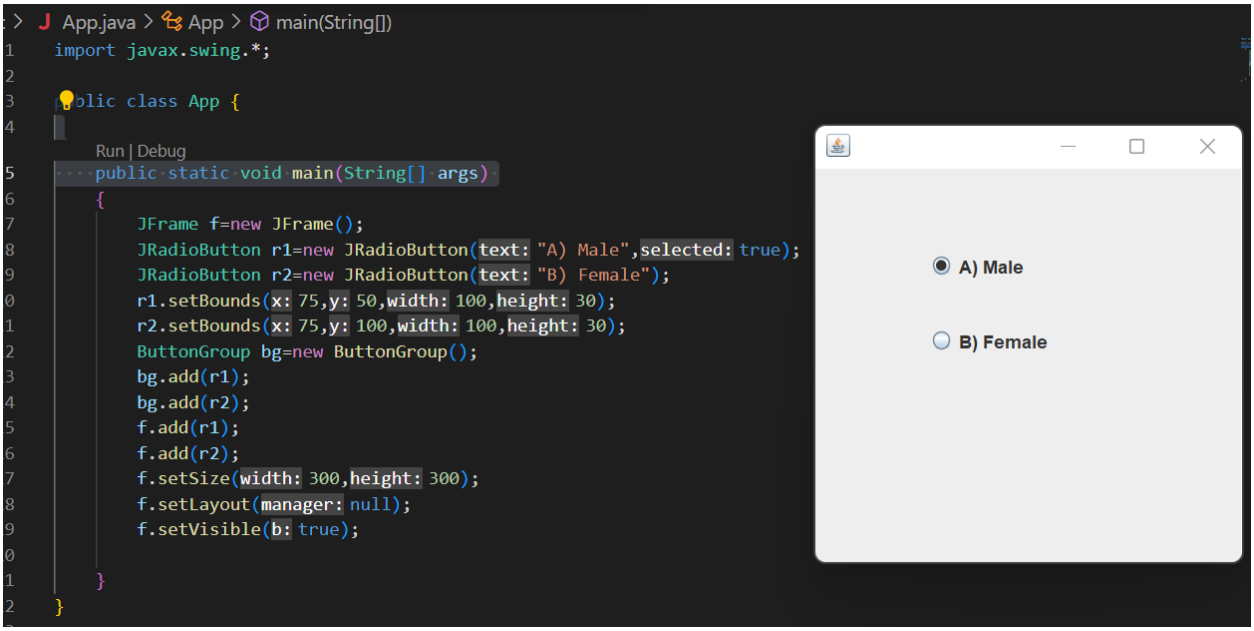
- **JRadioButton():** It is used to create an unselected radio button with no text.
- **JRadioButton(Label):** It is used to create an unselected radio button with specified text.
- **JRadioButton(Label, boolean):** It is used to create a radio button with the specified text and selected status.

Code:

```
import javax.swing.*;

public class App {

    public static void main(String[] args)
    {
        JFrame f=new JFrame();
        JRadioButton r1=new JRadioButton("A) Male",true);
        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);

    }
}
```

Output:



JPasswordField

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

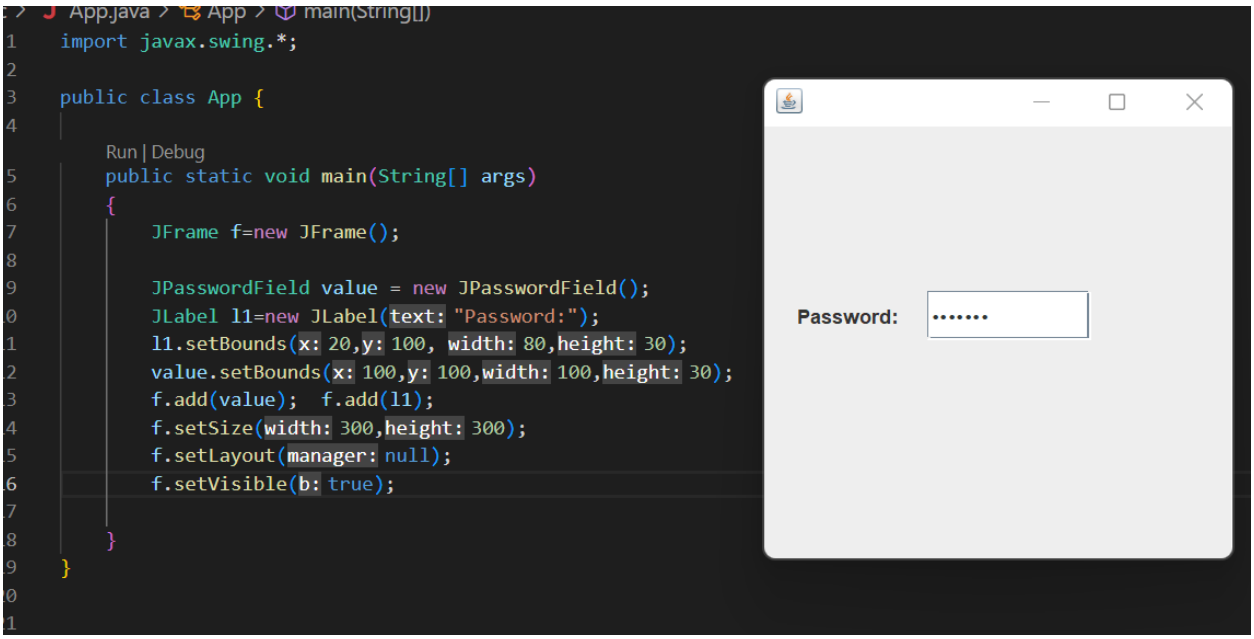
Constructor	Description
JPasswordField()	Constructs a new JPasswordField, with a default document, null starting text string, and 0 column width.
JPasswordField(int columns)	Constructs a new empty JPasswordField with the specified number of columns.

JPasswordField(String text)	Constructs a new JPasswordField initialized with the specified text.
JPasswordField(String text, int columns)	Construct a new JPasswordField initialized with the specified text and columns.

Code:

```
import javax.swing.*;

public class App {

    public static void main(String[] args)
    {
        JFrame f=new JFrame();

        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);

    }
}
```

Output:



JCheckBox

The JCheckBox class is used to create a checkbox. It is used to turn an option on (true) or off (false). Clicking on a CheckBox changes its state from "on" to "off" or from "off" to "on ".It inherits **JToggleButton** class.

Constructor	Description
JJCheckBox()	Creates an initially unselected check box button with no text, no icon.
JChechBox(String s)	Creates an initially unselected check box with text.
JCheckBox(String text, boolean selected)	Creates a check box with text and specifies whether or not it is initially selected.
JCheckBox(Action a)	Creates a check box where properties are taken from the Action supplied.

Code:

```
import javax.swing.*;

public class App {

    public static void main(String[] args)
    {
        JFrame f=new JFrame();

        JCheckBox checkBox1 = new JCheckBox("C++");
        checkBox1.setBounds(100,100, 50,50);
        JCheckBox checkBox2 = new JCheckBox("Java", true);
        checkBox2.setBounds(100,150, 100,50);
        f.add(checkBox1);
        f.add(checkBox2);
        f.setSize(200,200);
        f.setLayout(null);
        f.setVisible(true);

    }
}
```

Output:

JComboBox

It inherits the JComponent class and is used to show pop up menu of choices.

```
import javax.swing.*;

public class App {

    public static void main(String[] args)
    {
        JFrame f=new JFrame();
        String courses[] = { "core java","advance java", "java servlet"};
        JComboBox c = new JComboBox(courses);
        c.setBounds(40,40,90,20);
        f.add(c);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);

    }
}
```

Output:

JTextArea

The object of a JTextArea class is a multi line region that displays text. It allows the editing of multiple line text. It inherits JTextComponent class

Commonly used Constructors:

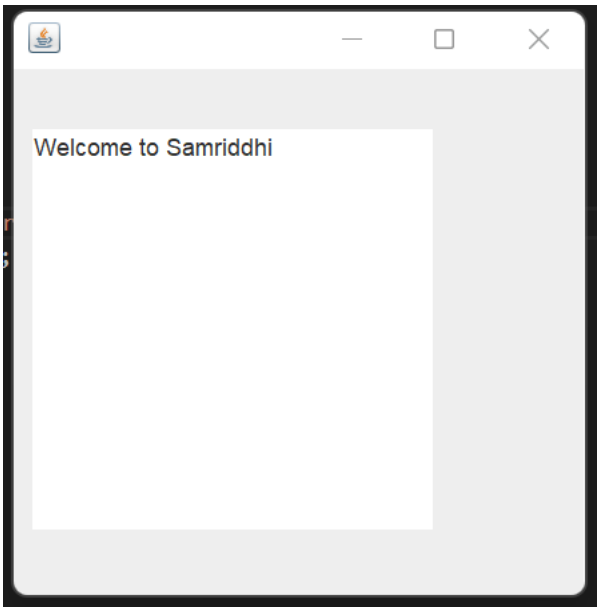
Constructor	Description
JTextArea()	Creates a text area that displays no text initially.
JTextArea(String s)	Creates a text area that displays specified text initially.
JTextArea(int row, int column)	Creates a text area with the specified number of rows and columns that displays no text initially.
JTextArea(String s, int row, int column)	Creates a text area with the specified number of rows and columns that displays specified text.

Code:

```
import javax.swing.*;
```

```java
public class App {

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

    }
}
```

Output



JScrollPane

A JscrollPane is used to make scrollable view of a component. When screen size is limited, we use a scroll pane to display a large component or a component whose size can change dynamically.

Commonly Used Method

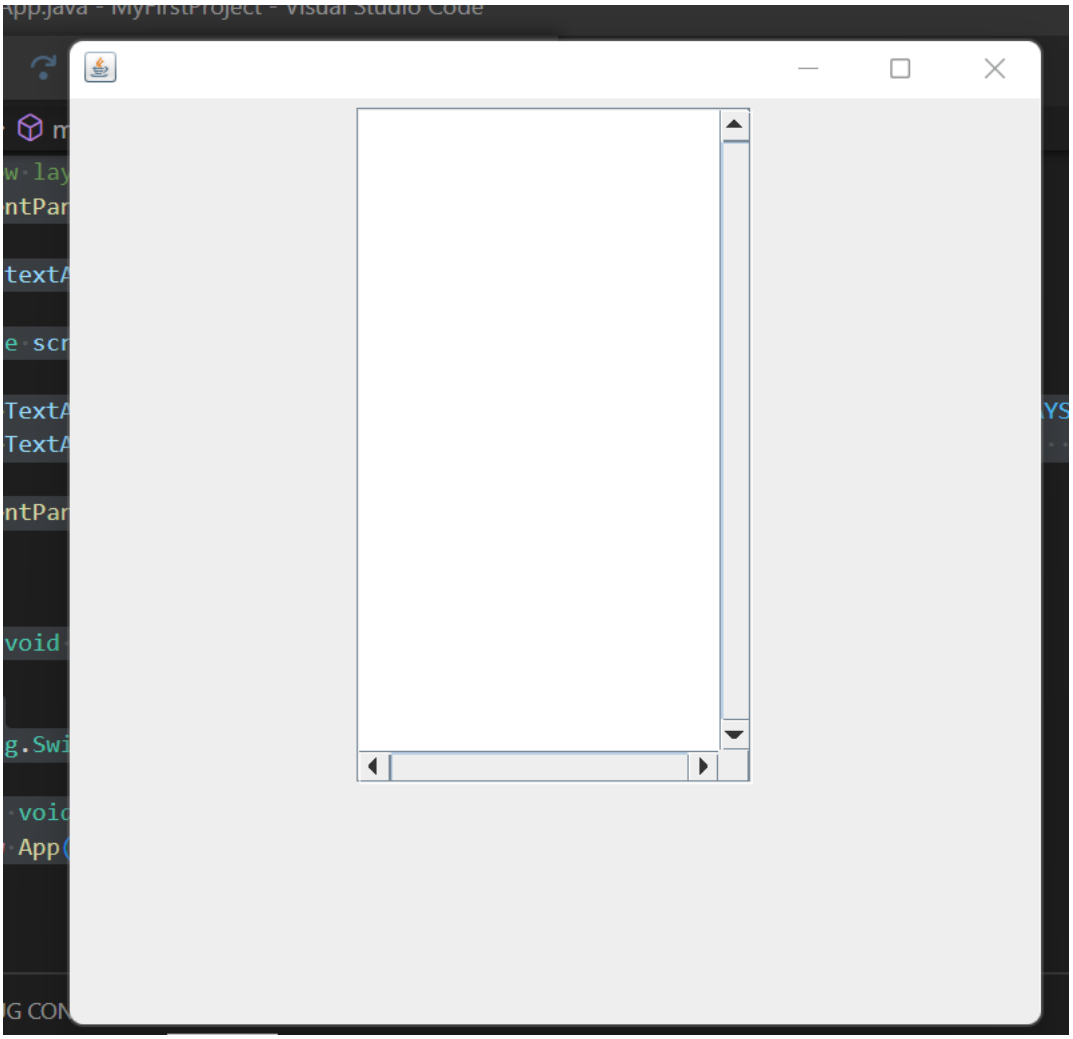
Constructor	Purpose
JScrollPane()	It creates a scroll pane. The Component parameter, when present, sets the scroll pane's client. The two int parameters, when present, set the vertical and horizontal scroll bar policies (respectively).
JScrollPane(Component)	
JScrollPane(int, int)	
JScrollPane(Component, int, int)	

Code:

```
import javax.swing.*;
import java.awt.FlowLayout;



public class App {
    App()
    {
        JFrame f=new JFrame();
        // Display the window.
        f.setSize(500, 500);
        f.setVisible(true);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // set flow layout for the frame
        f.getContentPane().setLayout(new FlowLayout());

        JTextArea textArea = new JTextArea(20, 20);

        JScrollPane scrollableTextArea = new JScrollPane(textArea);

        scrollableTextArea.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_
ALWAYS);
        scrollableTextArea.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_ALWA
YS);

        f.getContentPane().add(scrollableTextArea);
    }

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

```
        javax.swing.SwingUtilities.invokeLater(new Runnable() {

            public void run() {
               new App();
            }
        });
    }
```

Output:



Border Swing Control in Java

The border is an interface using which we can apply a border to every component. To create the borders we have to use the methods available in **BorderFactory** class. We can apply the created border to any component by using the **SetBorder()** method.

Syntax: **Component.setBorder(Border);**

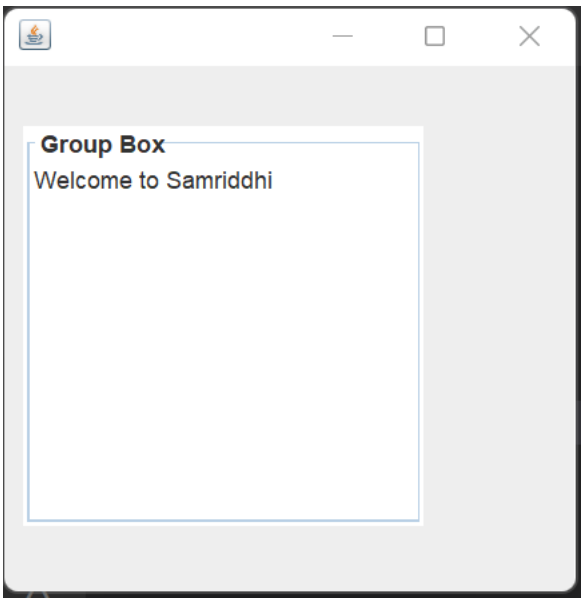
Methods of Border

- **Border createLineBorder(Color, int):** It is used to create a line border. Here, the Color object specifies the color of the line and int specifies the width in pixels of the line.
- **Border createEtchedBorder(int, Color, Color):** It is used to create an etched border. Here, Color arguments specify the highlight and shadow colors to be used.
  Here, int arguments allow the border methods to be specified as
  either EtchedBorder.RAISED or EtchedBorder.LOWERED. The methods without
  the int arguments create a lowered etched border.
- **Border createBevelBorder(int, Color, Color):** It is used to create a raised or lowered beveled border, specifying the colors to use. Here, the integer argument can be either BevelBorder.RAISED or BevelBorder.LOWERED. Here, Color specifies the highlight and shadow colors.
- **MatteBorder createMatteBorder(int, int, int, int, Icon):** It is used to create a matte border. Here, the integer arguments specify the number of pixels that the border occupies at the top, left, bottom, and right (in that order) of whatever component uses it. Here, the color argument specifies the color which with the border should fill its area. Here, the icon argument specifies the icon which with the border should tile its area.
- **TitledBorder createTitledBorder(Border, String, int, int, Font, Color):** Create a titled border. Here, the string argument specifies the title to be displayed. Here, the optional font and color arguments specify the font and color to be used for the title's text. Here, the border argument specifies the border that should be displayed along with the title. Here, the integer arguments specify the number of pixels that the border occupies at the top, left, bottom, and right (in that order) of whatever component uses it.
- **CompoundBorder createCompoundBorder(Border, Border):** Combine two borders into one. Here, the first argument specifies the outer border; the second, the inner border.

Code:

```
import javax.swing.*;

public class App {

    public static void main(String[] args)
    {
        JFrame f=new JFrame();
        JPanel j=new JPanel();
        JTextArea area=new JTextArea("Welcome to Samriddhi");
        area.setBorder(BorderFactory.createTitledBorder("Group Box"));
        area.setBounds(10,30, 200,200);
        f.add(j);
        f.add(area);
        f.setSize(300,300);
        f.setLayout(null);
        f.setVisible(true);

    }
}
```

Output:

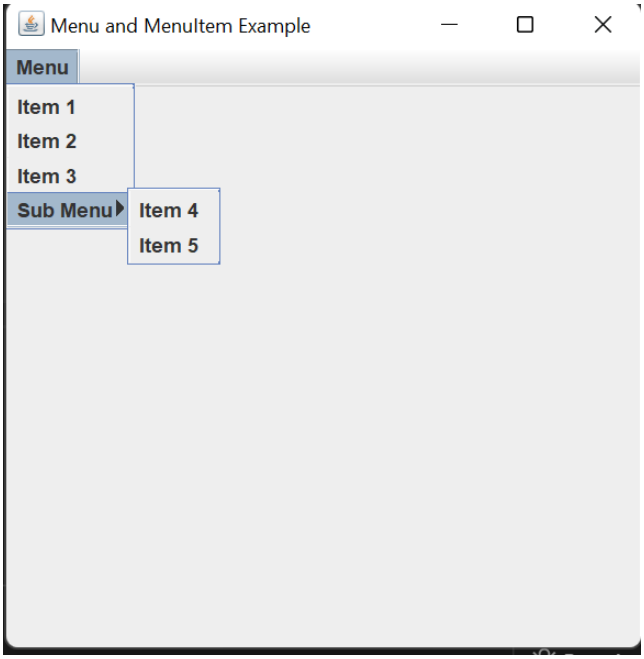


Menu, Menu Item:

Code:

```
import javax.swing.*;

public class App {

    public static void main(String[] args)
    {
        JMenu menu, submenu;
        JMenuItem i1, i2, i3, i4, i5;

        JFrame f= new JFrame("Menu and MenuItem Example");
        JMenuBar mb=new JMenuBar();
        menu=new JMenu("Menu");
        submenu=new JMenu("Sub Menu");
        i1=new JMenuItem("Item 1");
        i2=new JMenuItem("Item 2");
        i3=new JMenuItem("Item 3");
        i4=new JMenuItem("Item 4");
        i5=new JMenuItem("Item 5");
        menu.add(i1);
        menu.add(i2);
        menu.add(i3);
        submenu.add(i4);
        submenu.add(i5);
        menu.add(submenu);
        mb.add(menu);
        f.setJMenuBar(mb);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);

    }
}
```

Output:



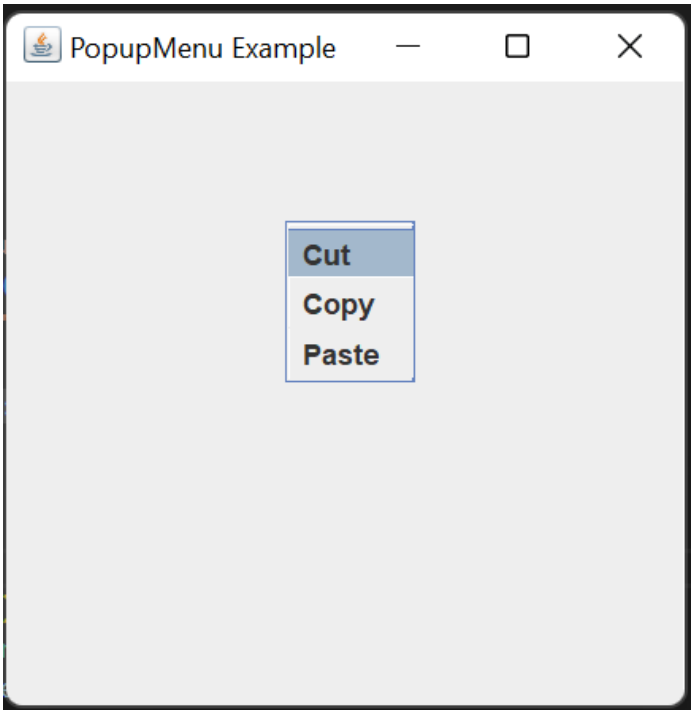
Jpopup Menu

Code:

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

public class App {

    public static void main(String[] args)
    {
        JFrame f= new JFrame("PopupMenu Example");
        JPopupMenu popupmenu = new JPopupMenu("Edit");
         JMenuItem cut = new JMenuItem("Cut");
         JMenuItem copy = new JMenuItem("Copy");
         JMenuItem paste = new JMenuItem("Paste");
         popupmenu.add(cut);
         popupmenu.add(copy);
         popupmenu.add(paste);

         f.addMouseListener(new MouseAdapter() {
            public void mouseClicked(MouseEvent e) {
                popupmenu.show(f , e.getX(), e.getY());
            }
         });
```

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

    }
}
```

Output:

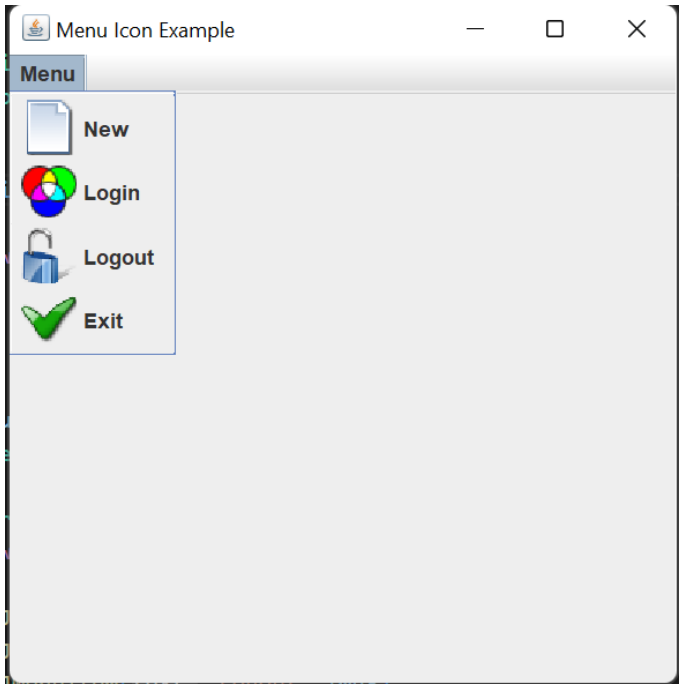


Adding Icon in Menu:

```
import javax.swing.*;
public class App {

    public static void main(String[] args) {

    JFrame f=new JFrame("Menu Icon Example");
      ImageIcon img=new ImageIcon("new.png");
      ImageIcon img2=new ImageIcon("Login.png");
      ImageIcon img3=new ImageIcon("Logout.png");
      ImageIcon img4=new ImageIcon("Exit.png");
      JMenu menu;
        JMenuItem i1, i2, i3, i4;
```

```
        JMenuBar mb=new JMenuBar();
        menu=new JMenu("Menu");

        i1=new JMenuItem("New",img);
        i2=new JMenuItem("Login",img2);
        i3=new JMenuItem("Logout",img3);
        i4=new JMenuItem("Exit",img4);

        menu.add(i1);
        menu.add(i2);
        menu.add(i3);
        menu.add(i4);

        mb.add(menu);
        f.setJMenuBar(mb);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);

    }
}
```

Output:



Disable Menu Item

```
i1.setEnabled(false);
```

Code:

```
import javax.swing.*;
public class App {

    public static void main(String[] args) {

    JFrame f=new JFrame("Menu Icon Example");
      ImageIcon img=new ImageIcon("new.png");
      ImageIcon img2=new ImageIcon("Login.png");
      ImageIcon img3=new ImageIcon("Logout.png");
      ImageIcon img4=new ImageIcon("Exit.png");
      JMenu menu;
        JMenuItem i1, i2, i3, i4;

        JMenuBar mb=new JMenuBar();
        menu=new JMenu("Menu");

        i1=new JMenuItem("New",img);
        i2=new JMenuItem("Login",img2);
        i3=new JMenuItem("Logout",img3);
        i4=new JMenuItem("Exit",img4);

        menu.add(i1);
        menu.add(i2);
        menu.add(i3);
        menu.add(i4);
        i1.setEnabled(false);

        mb.add(menu);
        f.setJMenuBar(mb);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);

    }
}
```

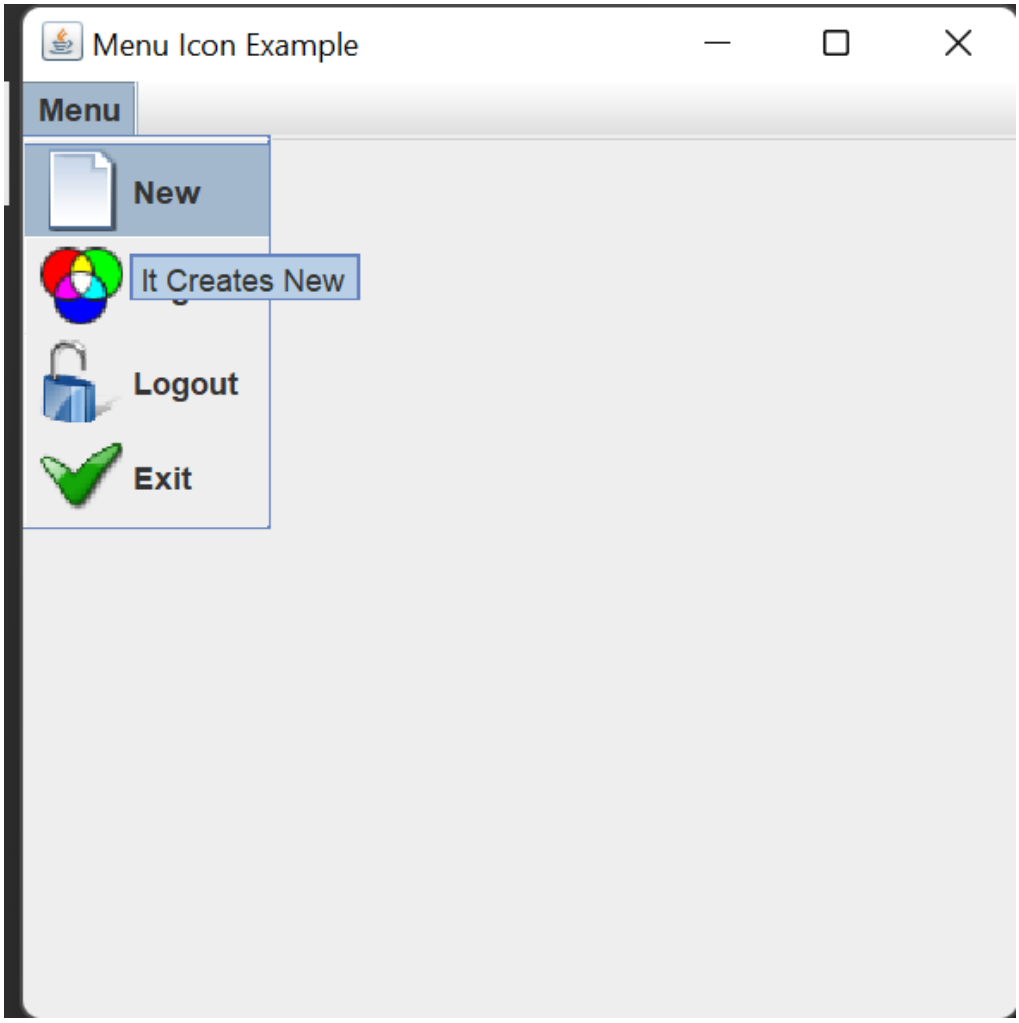
Adding Tooltip:

```
i1.setToolTipText("It Creates New");
```

Code:

```
import javax.swing.*;
public class App {

    public static void main(String[] args) {

    JFrame f=new JFrame("Menu Icon Example");
      ImageIcon img=new ImageIcon("new.png");
      ImageIcon img2=new ImageIcon("Login.png");
      ImageIcon img3=new ImageIcon("Logout.png");
      ImageIcon img4=new ImageIcon("Exit.png");
      JMenu menu;
        JMenuItem i1, i2, i3, i4;

        JMenuBar mb=new JMenuBar();
        menu=new JMenu("Menu");

        i1=new JMenuItem("New",img);
        i2=new JMenuItem("Login",img2);
        i3=new JMenuItem("Logout",img3);
        i4=new JMenuItem("Exit",img4);

        menu.add(i1);
        menu.add(i2);
        menu.add(i3);
        menu.add(i4);
        i1.setToolTipText("It Creates New");

        mb.add(menu);
        f.setJMenuBar(mb);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);

    }
}
```

Output:

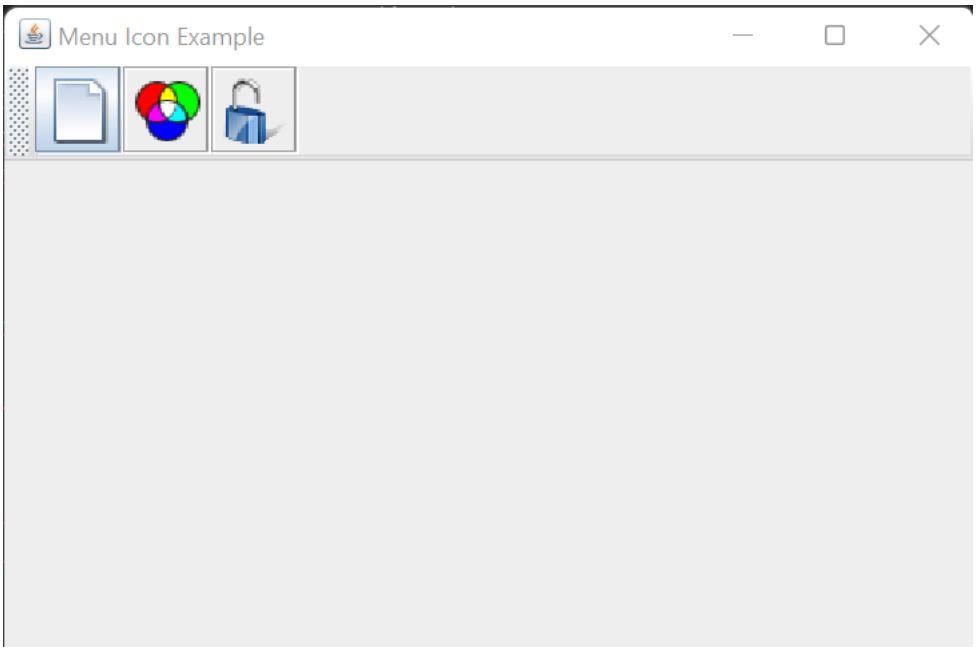


Toolbar :

```
import javax.swing.*;
import java.awt.*;
public class App {
```

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

    JFrame f=new JFrame("Menu Icon Example");
      ImageIcon img1=new ImageIcon("new.png");
      ImageIcon img2=new ImageIcon("Login.png");
      ImageIcon img3=new ImageIcon("Logout.png");

      JToolBar toolbar = new JToolBar();
      // create a panel
      JPanel p = new JPanel();

      JButton newBtn = new JButton(img1);
      JButton openBtn = new JButton(img2);
      JButton saveBtn = new JButton(img3);

      toolbar.add(newBtn);
      toolbar.add(openBtn);
      toolbar.add(saveBtn);

      // add panel to toolbar

      toolbar.add(p);
      // add toolbar to frame
      f.add(toolbar, BorderLayout.NORTH);

      // set the size of the frame
      f.setSize(500, 500);
      f.setVisible(true);

    }
}
```

Output:



Keyboard Mnemonics and Accelerators:

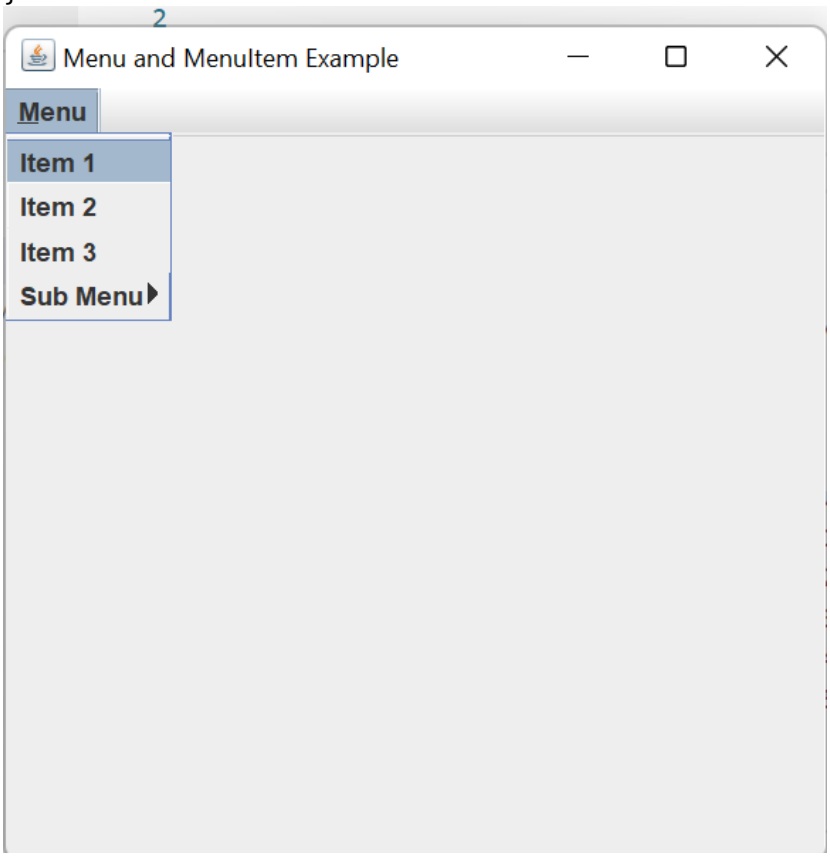
Keyboard Mnemonics:

A menu mnemonic is a single-letter abbreviation for a menu command. When the menu has already been pulled down, the user can type this single key to invoke that menu item. The mnemonic for a menu item is typically indicated by underlining the letter of the shortcut in the menu item name, which means that you must select a shortcut letter that appears in the menu item label. Mnemonics must be unique within a menu, of course, but multiple menu panes can reuse mnemonics. Items in a menu bar may also have mnemonics. You specify a mnemonic for a menu or a menu item with the setMnemonic() method.

Example:

```
import javax.swing.*;

public class App {
    public static void main(String[] args) {

        JMenu menu, submenu;
        JMenuItem i1, i2, i3, i4, i5;

        JFrame f= new JFrame("Menu and MenuItem Example");
        JMenuBar mb=new JMenuBar();
        menu=new JMenu("Menu");
        menu.setMnemonic('M');
        submenu=new JMenu("Sub Menu");
        i1=new JMenuItem("Item 1");
        i2=new JMenuItem("Item 2");
```

```
        i3=new JMenuItem("Item 3");
        i4=new JMenuItem("Item 4");
        i5=new JMenuItem("Item 5");
        menu.add(i1);
        menu.add(i2);
        menu.add(i3);
        submenu.add(i4);
        submenu.add(i5);
        menu.add(submenu);
        mb.add(menu);
        f.setJMenuBar(mb);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);

    }
}
```



Layout Management:

The LayoutManagers are used to arrange components in a particular manner. The **Java LayoutManagers** facilitates us to control the positioning and size of the components in GUI forms. LayoutManager is an interface that is implemented by all the classes of layout managers. There are the following classes that represent the layout managers:

The Abstract Windowing Toolkit (AWT) has the following five layout managers:

- java.awt.BorderLayout
- java.awt.FlowLayout
- java.awt.GridLayout
- java.awt.CardLayout
- java.awt.GridBagLayout

Some of these are used in the swing:

- javax.swing.BoxLayout
- javax.swing.ScrollPanelLayout
- javax.swing.GroupLayout
- javax.swing.SpringLayout

No Layout:

```
import javax.swing.*;
public class App extends JFrame{
    JFrame f;
    App()
    {
        JButton b=new JButton("click");//create button
        b.setBounds(130,100,100, 40);

        add(b);//adding button on frame
        setSize(400,500);
        setLayout(null);
        setVisible(true);
    }

    public static void main(String[] args)
    {
        App app=new App();
```

```
    }

}
```

Flow Layout:

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

Fields of FlowLayout class

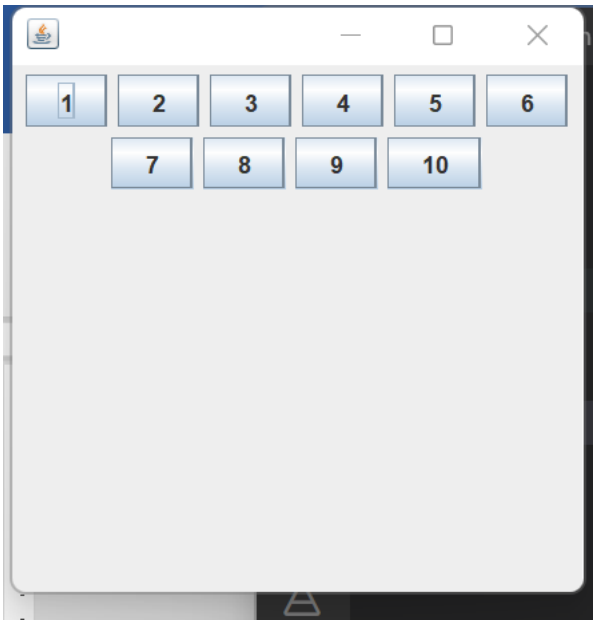
- public static final int LEFT(3)
- public static final int RIGHT(2)
- public static final int CENTER(1)
- public static final int LEADING
- public static final int TRAILING

Example:

```
import javax.swing.*;
import java.awt.*;
public class App{
    JFrame f;
    App()
    {
        // creating a frame object

     f=new JFrame();
    // creating the buttons
   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");
   JButton b10 = new JButton("10");
   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.add(b10);
```

```
    // parameter less constructor is used
    // therefore, alignment is center
    // horizontal as well as the vertical gap is 5 units.
    f.setLayout(new FlowLayout());

    f.setSize(300, 300);
    f.setVisible(true);
    }

    public static void main(String[] args)
    {
        App app=new App();
    }

}
```

Output:



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 a 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:

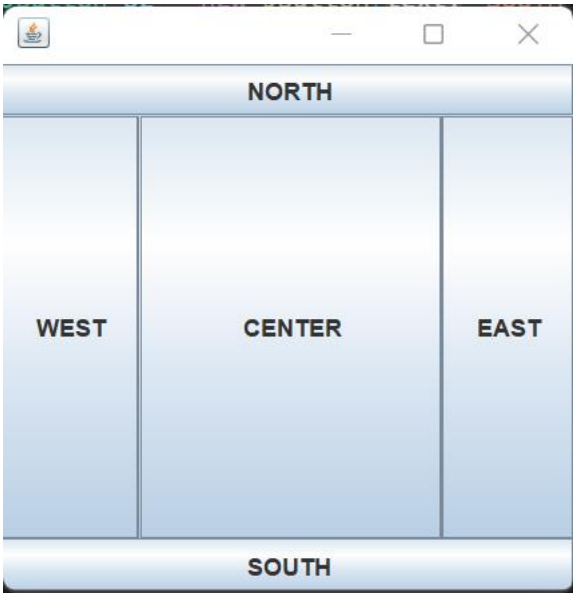
- **BorderLayout():** creates a border layout but with no gaps between the components.
- **BorderLayout(int hgap, int vgap):** creates a border layout with the given horizontal and vertical gaps between the components.

Example:

```
import javax.swing.*;
import java.awt.*;
public class App{
    JFrame f;
    App()
    {

     f=new JFrame();
     // creating buttons
     JButton b1 = new JButton("NORTH");; // the button will be labeled as NORTH
     JButton b2 = new JButton("SOUTH");; // the button will be labeled as SOUTH
     JButton b3 = new JButton("EAST");; // the button will be labeled as EAST
     JButton b4 = new JButton("WEST");; // the button will be labeled as WEST
     JButton b5 = new JButton("CENTER");; // the button will be labeled as
CENTER

     f.add(b1, BorderLayout.NORTH); // b1 will be placed in the North
Direction
     f.add(b2, BorderLayout.SOUTH);  // b2 will be placed in the South
Direction
     f.add(b3, BorderLayout.EAST);  // b2 will be placed in the East
Direction
     f.add(b4, BorderLayout.WEST);  // b2 will be placed in the West
Direction
     f.add(b5, BorderLayout.CENTER);  // b2 will be placed in the Center

     f.setSize(300, 300);
```

```
     f.setVisible(true);
    }

    public static void main(String[] args)
    {
        App app=new App();
    }

}
```

Output:



Grid Layout:

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

Constructors of GridLayout class

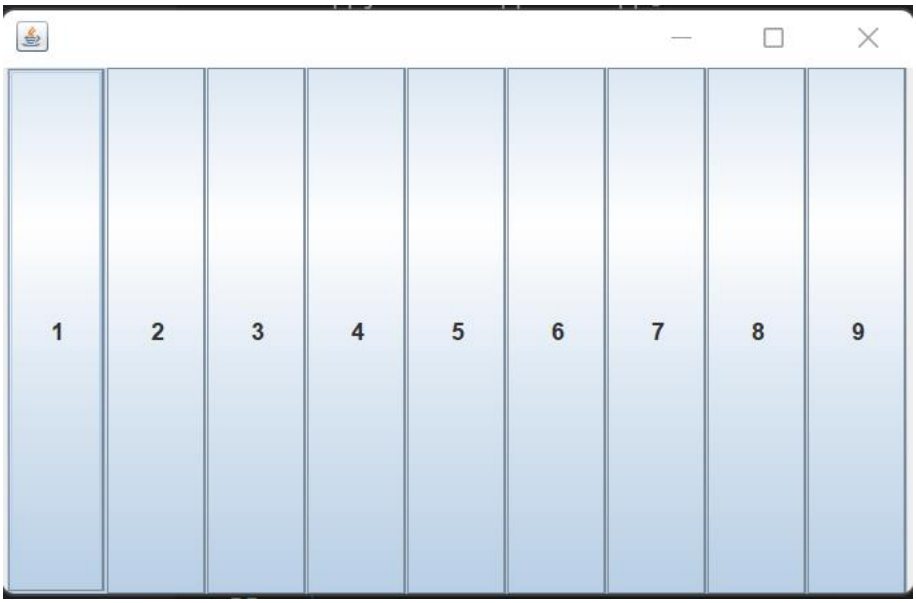
- **GridLayout():** creates a grid layout with one column per component in a row.
- **GridLayout(int rows, int columns):** creates a grid layout with the given rows and columns but no gaps between the components.

- **GridLayout(int rows, int columns, int hgap, int vgap):** creates a grid layout with the given rows and columns along with given horizontal and vertical gaps.

Example:

```
import javax.swing.*;
import java.awt.*;
public class App{
    JFrame f;
    App()
    {

     f=new JFrame();

     // creating 9 buttons
     JButton btn1 = new JButton("1");
     JButton btn2 = new JButton("2");
     JButton btn3 = new JButton("3");
     JButton btn4 = new JButton("4");
     JButton btn5 = new JButton("5");
     JButton btn6 = new JButton("6");
     JButton btn7 = new JButton("7");
     JButton btn8 = new JButton("8");
     JButton btn9 = new JButton("9");
     f.add(btn1); f.add(btn2); f.add(btn3);
f.add(btn4); f.add(btn5); f.add(btn6);
f.add(btn7); f.add(btn8); f.add(btn9);
// setting the grid layout using the parameterless constructor
f.setLayout(new GridLayout());

f.setSize(300, 300);
f.setVisible(true);
    }

    public static void main(String[] args)
    {
        App app=new App();
    }

}
```

Output:



GridBagLayout:

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

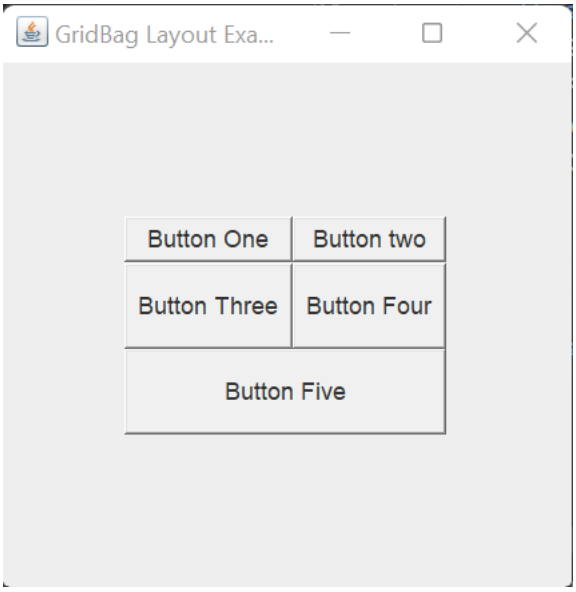
The components may not be of the same size. Each GridBagLayout object maintains a dynamic, rectangular grid of cells. Each component occupies one or more cells known as its display area.

Example:

```
import java.awt.Button;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import javax.swing.*;
public class App extends JFrame{
    App()
    {
        GridBagLayout grid = 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);

    }

    public static void main(String[] args)
    {
        App ap=new App();
    }

}
```

Output:



GroupLayout

GroupLayout *groups its components and places them in a Container hierarchica*lly. The grouping is done by instances of the Group class.

Example:

```
import java.awt.*;

import javax.swing.*;
public class App extends JFrame{



    public static void main(String[] args)
    {
        JFrame frame = new JFrame("GroupLayoutExample");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        Container contentPanel = frame.getContentPane();
        GroupLayout groupLayout = new GroupLayout(contentPanel);

        contentPanel.setLayout(groupLayout);

        JLabel clickMe = new JLabel("Click Here");
        JButton button = new JButton("This Button");

        groupLayout.setHorizontalGroup(
                    groupLayout.createSequentialGroup()
                                .addComponent(clickMe)
```

```
                                .addGap(10, 20, 100)
                                .addComponent(button));
        groupLayout.setVerticalGroup(
                        groupLayout.createParallelGroup(GroupLayout.Alignment.BASELI
NE)
                                .addComponent(clickMe)
                                .addComponent(button));
        frame.pack();
        frame.setVisible(true);
    }

}
```

Output: