Lecture No. 40-43 Graphical User Interface

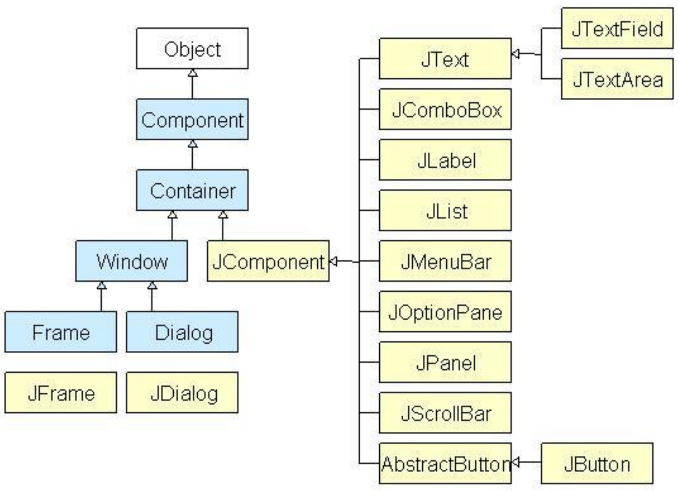
Swing

Java Swing is a lightweight Graphical User Interface (GUI) toolkit that includes a rich set of widgets. It includes package lets you make GUI components for your Java applications, and It is platform independent.

The Swing library is built on top of the Java Abstract Window Toolkit (**AWT**), an older, platform dependent GUI toolkit. You can use the Java GUI components like button, textbox, etc. from the library and do not have to create the components from scratch.

No.	Java AWT	Java Swing
1)	AWT components are **platform-dependent**.	Java swing components are **platform-independent**.
2)	AWT components are **heavyweight**.	Swing components are **lightweight**.
3)	AWT **doesn't support pluggable look and feel**.	Swing **supports pluggable look and feel**.
4)	AWT provides **less components** than Swing.	Swing provides **more powerful components** such as tables, lists, scrollpanes, colorchooser, tabbedpane etc.
5)	AWT **doesn't follows MVC**(Model View Controller) where model represents data, view represents presentation and controller acts as an interface between model and view.	Swing **follows MVC**.

Swing Class Hierarchy

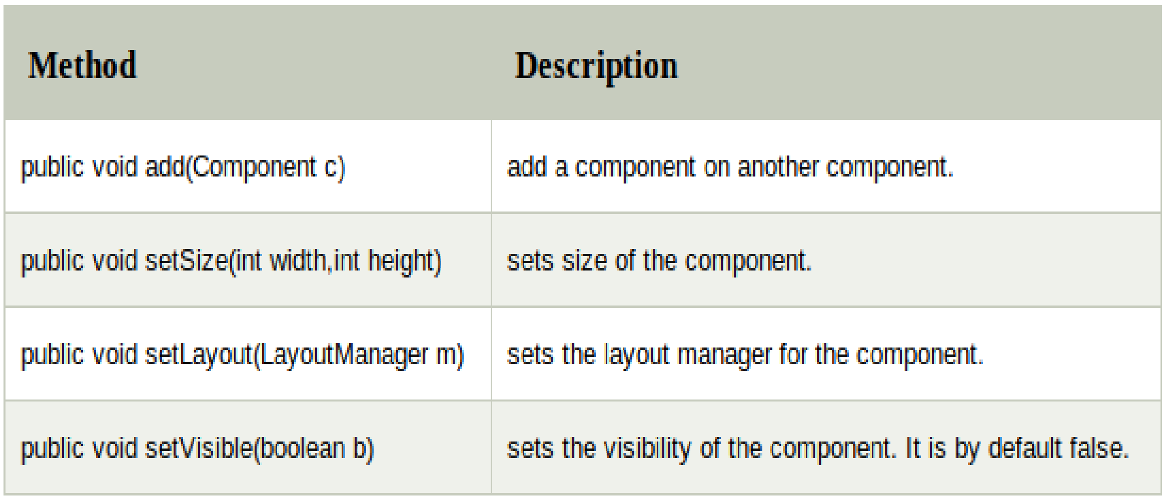


What is a container class?

Container classes are classes that can have other components on it. So for creating a GUI, we need at least one container object. There are 3 types of containers.

1. **Panel**: It is a pure container and is not a window in itself. The sole purpose of a Panel is to organize the components on to a window.
2. **Frame**: It is a fully functioning window with its title and icons.
3. **Dialog**: It can be thought of like a pop-up window that pops out when a message has to be displayed. It is not a fully functioning window like the Frame.

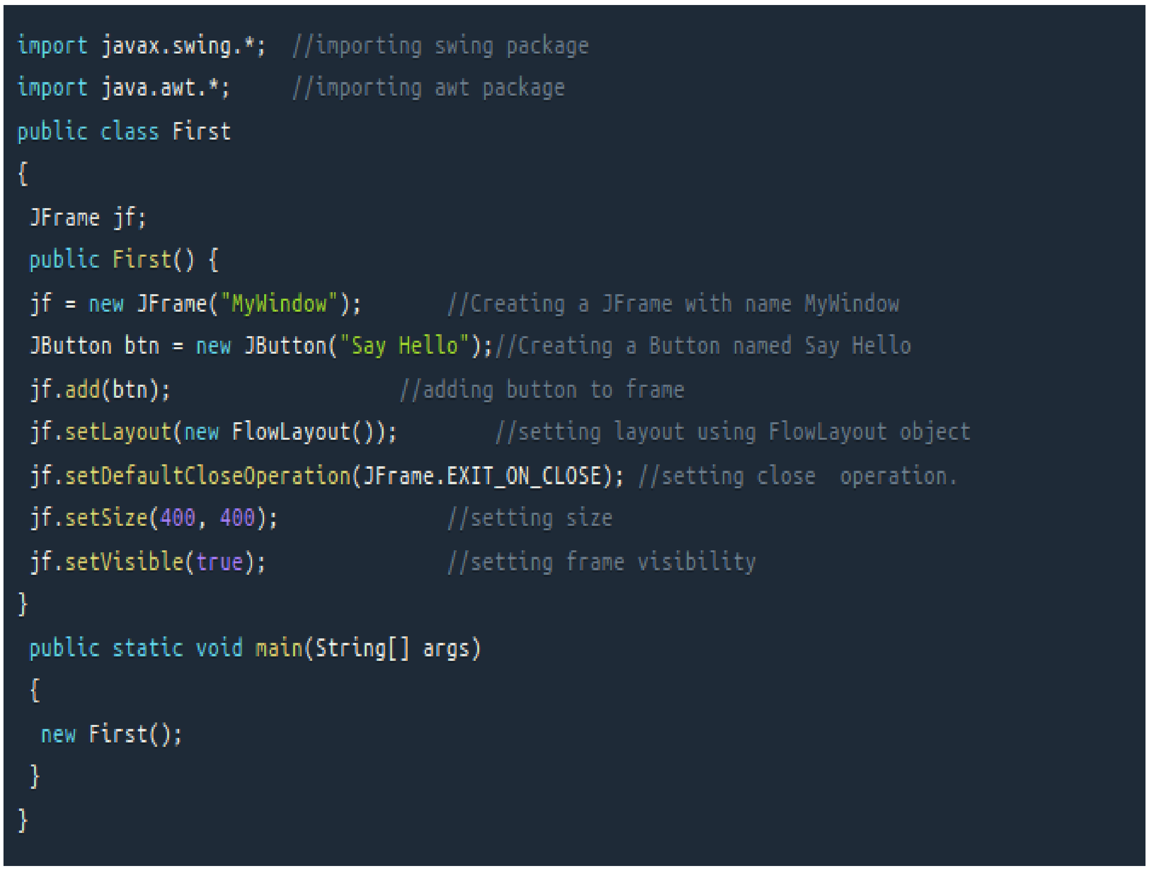
Commonly used methods of Container class



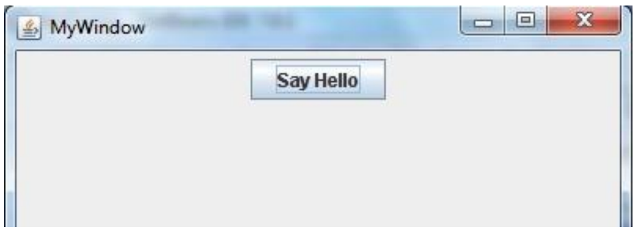
Method	Description
public void add(Component c)	add a component on another component.
public void setSize(int width,int height)	sets size of the component.
public void setLayout(LayoutManager m)	sets the layout manager for the component.
public void setVisible(boolean b)	sets the visibility of the component. It is by default false.

Creating a JFrame

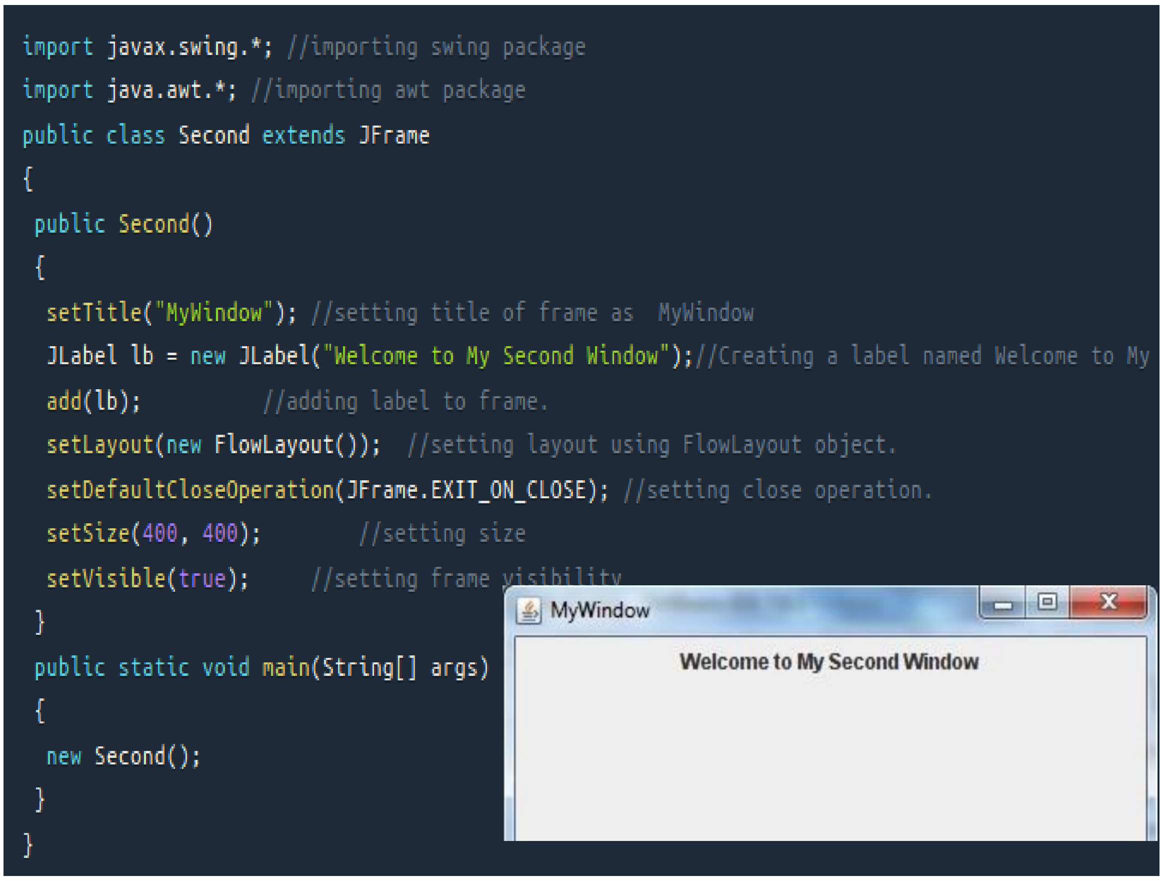
1. **By instantiating Jframe – Example**



```java
import javax.swing.*;  //importing swing package
import java.awt.*;     //importing awt package
public class First
{
 JFrame jf;
 public First() {
 jf = new JFrame("MyWindow");       //Creating a JFrame with name MyWindow
 JButton btn = new JButton("Say Hello");//Creating a Button named Say Hello
 jf.add(btn);                    //adding button to frame
 jf.setLayout(new FlowLayout());        //setting layout using FlowLayout object
 jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //setting close  operation.
 jf.setSize(400, 400);              //setting size
 jf.setVisible(true);               //setting frame visibility
}
 public static void main(String[] args)
 {
  new First();
 }
}
```



2. By extending JFrame class – Example



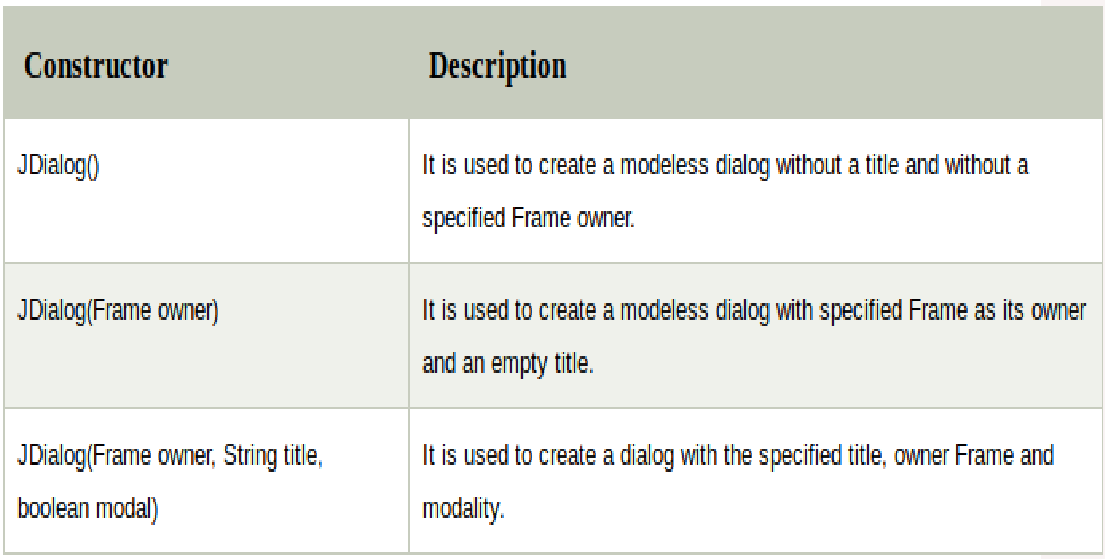
```
import javax.swing.*; //importing swing package
import java.awt.*; //importing awt package
public class Second extends JFrame
{
 public Second()
 {
  setTitle("MyWindow"); //setting title of frame as  MyWindow
  JLabel lb = new JLabel("Welcome to My Second Window");//Creating a label named Welcome to My
  add(lb);          //adding label to frame.
  setLayout(new FlowLayout());  //setting layout using FlowLayout object.
  setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //setting close operation.
  setSize(400, 400);        //setting size
  setVisible(true);     //setting frame visibility
 }
 public static void main(String[] args)
 {
  new Second();
 }
}
```

```
MyWindow
Welcome to My Second Window
```

JDialog

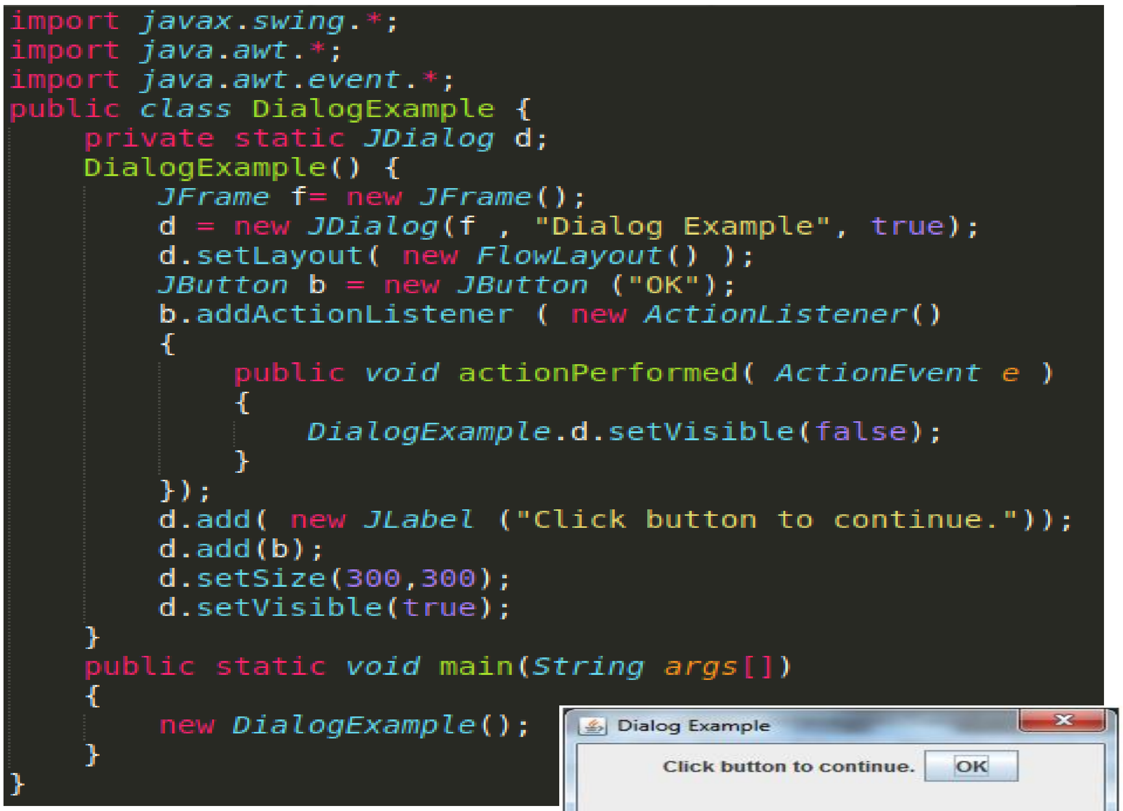
The JDialog control represents a top level window with a border and a title used to take some form of input from the user. It inherits the Dialog class. Unlike JFrame, it doesn't have maximize and minimize buttons.

Commonly used Constructors

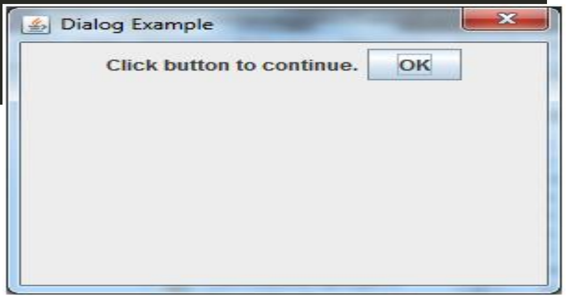


Constructor	Description
JDialog()	It is used to create a modeless dialog without a title and without a specified Frame owner.
JDialog(Frame owner)	It is used to create a modeless dialog with specified Frame as its owner and an empty title.
JDialog(Frame owner, String title, boolean modal)	It is used to create a dialog with the specified title, owner Frame and modality.

Example:



```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class DialogExample {
    private static JDialog d;
    DialogExample() {
        JFrame f= new JFrame();
        d = new JDialog(f , "Dialog Example", true);
        d.setLayout( new FlowLayout() );
        JButton b = new JButton ("OK");
        b.addActionListener ( new ActionListener()
        {
            public void actionPerformed( ActionEvent e )
            {
                DialogExample.d.setVisible(false);
            }
        });
        d.add( new JLabel ("Click button to continue."));
        d.add(b);
        d.setSize(300,300);
        d.setVisible(true);
    }
    public static void main(String args[])
    {
        new DialogExample();
    }
}
```



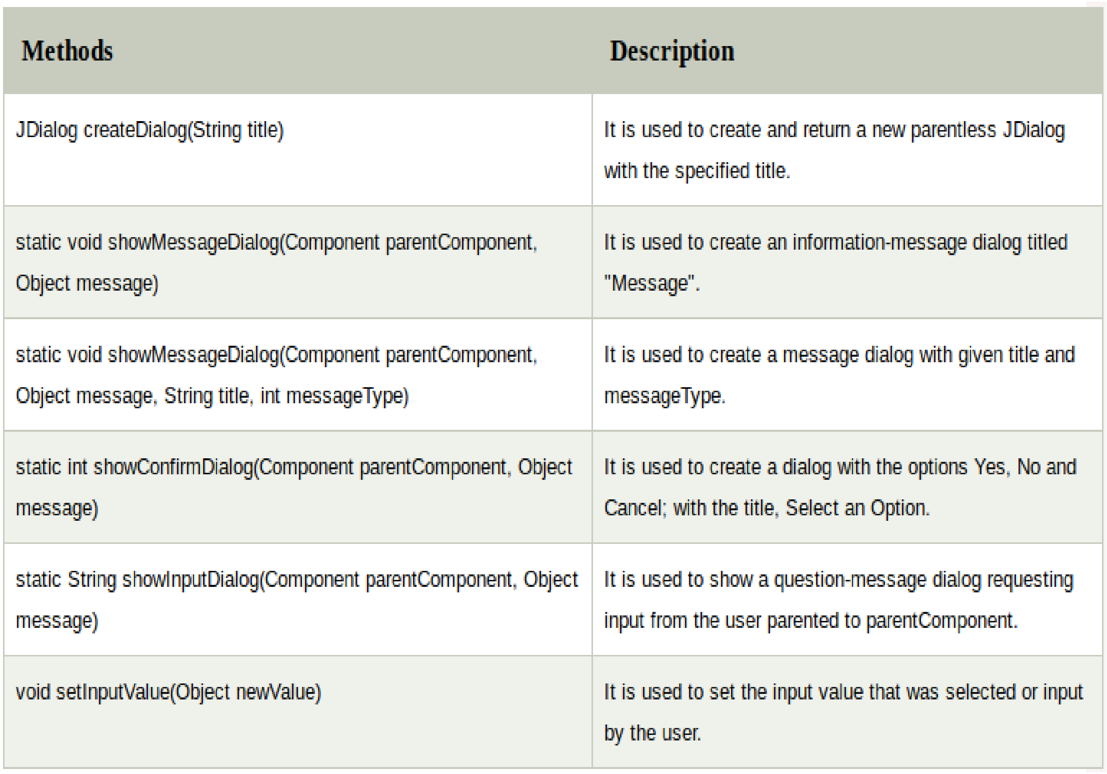
JOptionsPane

The JOptionPane class is used to provide standard dialog boxes such as message dialog box, confirm dialog box and input dialog box. These dialog boxes are used to display information or get input from the user. The JOptionPane class inherits JComponent class.

Commonly used constructors

Constructor	Description
JOptionPane()	It is used to create a JOptionPane with a test message.
JOptionPane(Object message)	It is used to create an instance of JOptionPane to display a message.
JOptionPane(Object message, int messageType	It is used to create an instance of JOptionPane to display a message with specified message type and default options.

Common Methods



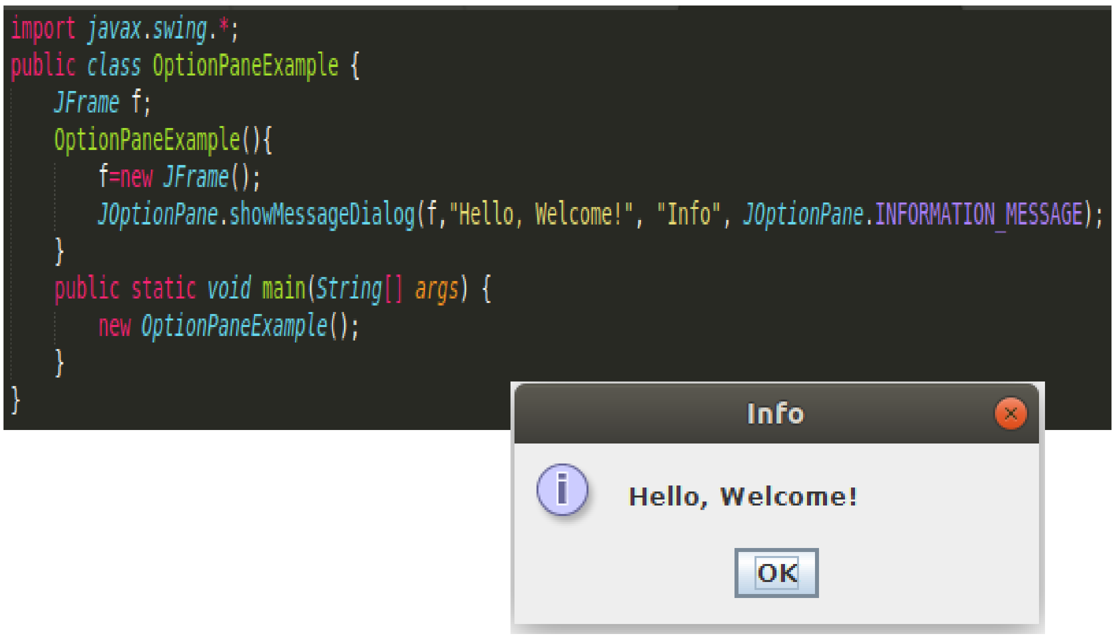
Methods	Description
JDialog createDialog(String title)	It is used to create and return a new parentless JDialog with the specified title.
static void showMessageDialog(Component parentComponent, Object message)	It is used to create an information-message dialog titled "Message".
static void showMessageDialog(Component parentComponent, Object message, String title, int messageType)	It is used to create a message dialog with given title and messageType.
static int showConfirmDialog(Component parentComponent, Object message)	It is used to create a dialog with the options Yes, No and Cancel; with the title, Select an Option.
static String showInputDialog(Component parentComponent, Object message)	It is used to show a question-message dialog requesting input from the user parented to parentComponent.
void setInputValue(Object newValue)	It is used to set the input value that was selected or input by the user.

By passing the following values as the fourth argument in showMessageDialog method, we can display different kinds of messages

1. JOptionPane.WARNING_MESSAGE
2. JoptionPane.ERROR_MESSAGE
3. JoptionPane.Plain_MESSAGE

Examples

Message Dialog



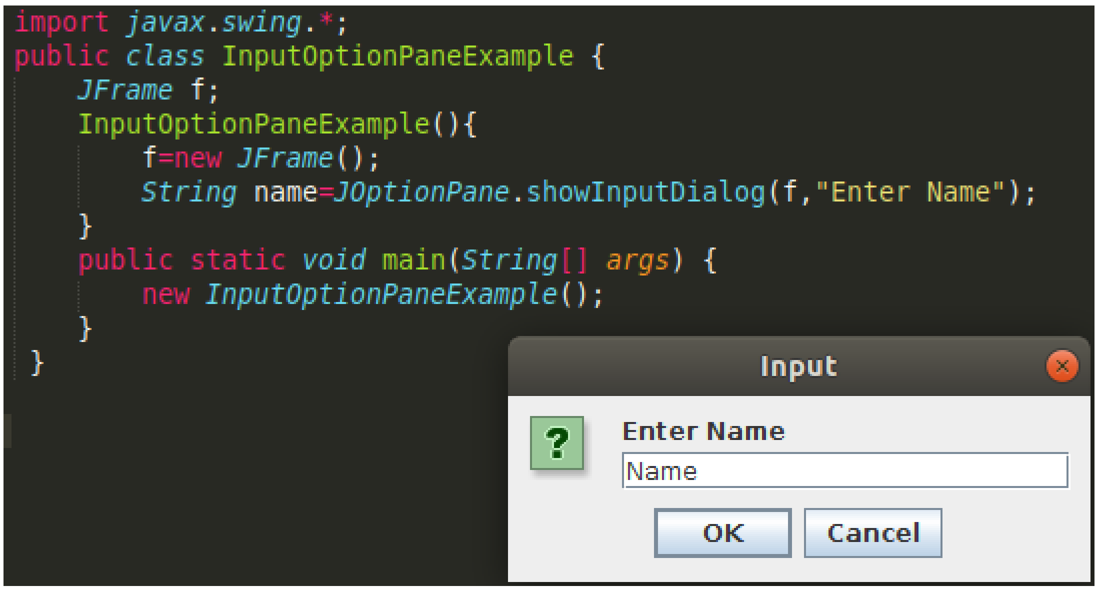
```
import javax.swing.*;
public class OptionPaneExample {
    JFrame f;
    OptionPaneExample(){
        f=new JFrame();
        JOptionPane.showMessageDialog(f,"Hello, Welcome!", "Info", JOptionPane.INFORMATION_MESSAGE);
    }
    public static void main(String[] args) {
        new OptionPaneExample();
    }
}
```

Info

Hello, Welcome!

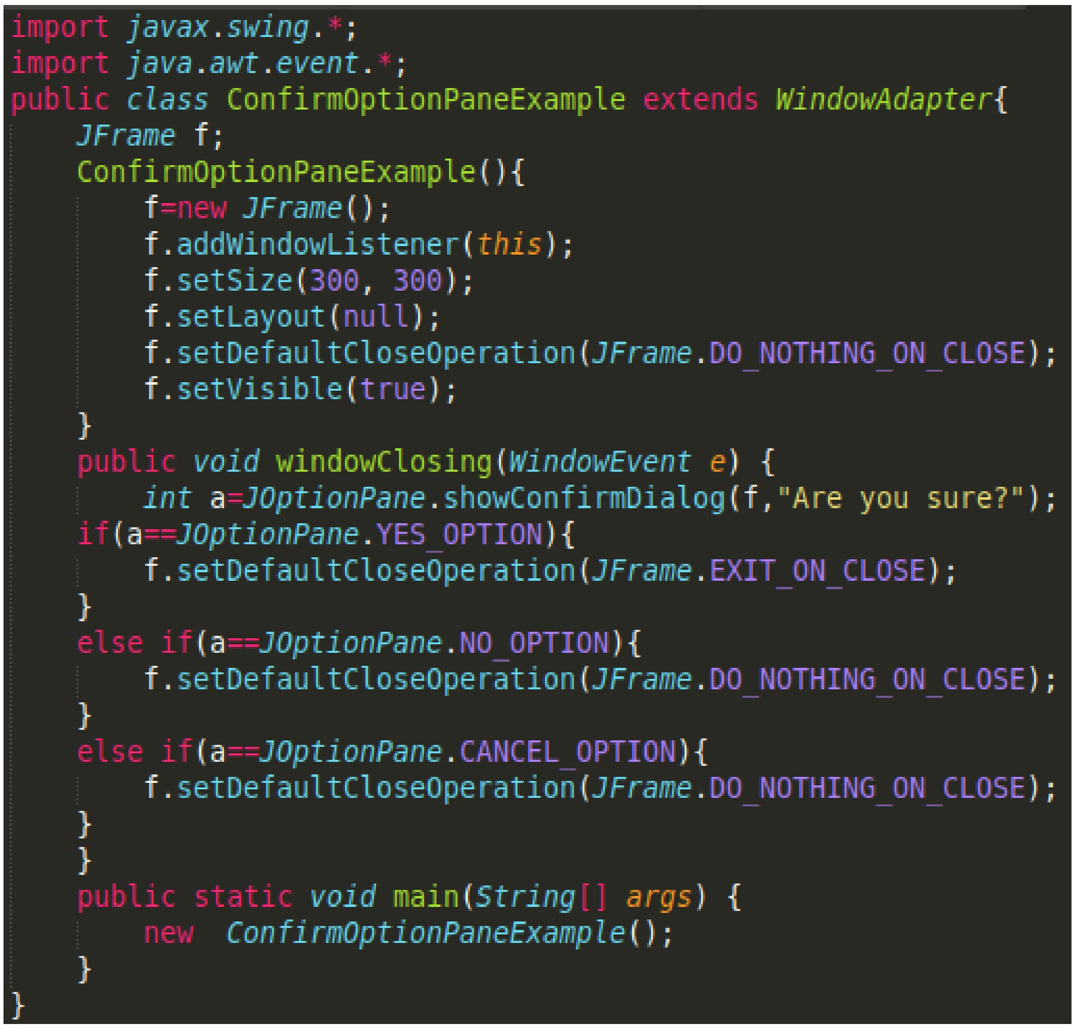
OK

Input Dialog

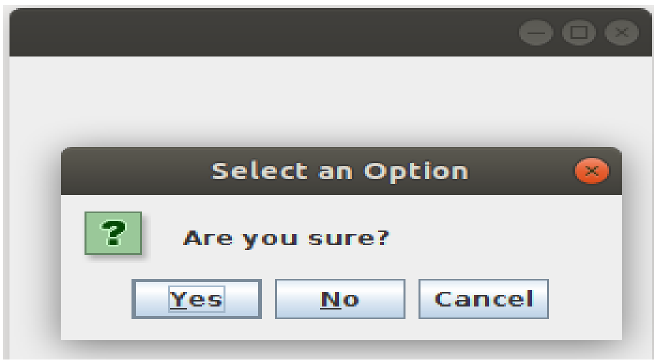


```
import javax.swing.*;
public class InputOptionPaneExample {
    JFrame f;
    InputOptionPaneExample(){
        f=new JFrame();
        String name=JOptionPane.showInputDialog(f,"Enter Name");
    }
    public static void main(String[] args) {
        new InputOptionPaneExample();
    }
}
```

Confirm Dialog

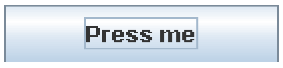


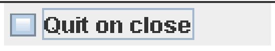
```
import javax.swing.*;
import java.awt.event.*;
public class ConfirmOptionPaneExample extends WindowAdapter{
    JFrame f;
    ConfirmOptionPaneExample(){
        f=new JFrame();
        f.addWindowListener(this);
        f.setSize(300, 300);
        f.setLayout(null);
        f.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
        f.setVisible(true);
    }
    public void windowClosing(WindowEvent e) {
        int a=JOptionPane.showConfirmDialog(f,"Are you sure?");
    if(a==JOptionPane.YES_OPTION){
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    else if(a==JOptionPane.NO_OPTION){
        f.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
    }
    else if(a==JOptionPane.CANCEL_OPTION){
        f.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
    }
    }
    public static void main(String[] args) {
        new  ConfirmOptionPaneExample();
    }
}
```

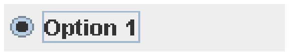


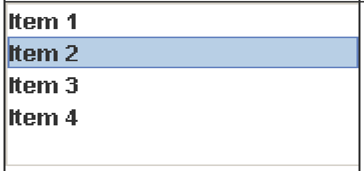
Components

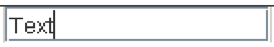
Swing components are basic building blocks of an application. Swing has a wide range of various components, including buttons, check boxes, sliders, and list boxes.

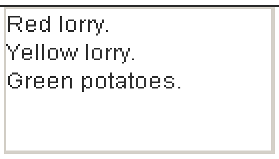
Component	Common constructor parameters	Important methods
 JButton	• Text • Icon • Text/icon • Action	

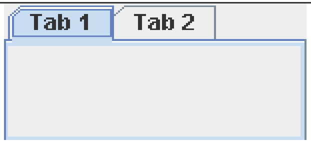
 JCheckBox	• Text
	• Text/selected
		`setSelected()`
		`isSelected()`

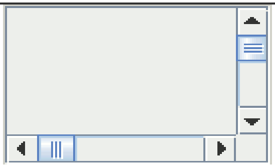
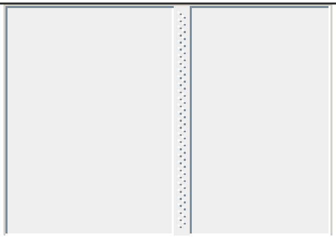
 JRadioButton	• Text
	• Text/selected
		`setSelected()`
		`isSelected()`
		`ButtonGroup.add(...)`

 JList	• Array of items
	• ListModel
		`getSelectedValue()`
		`getSelectedItem()`

 JTextField	• No columns
	• Initial text
		`getText()`
		`setText()`
		`getDocument()`

 JTextArea	• No rows/columns • Rows/cols/text	

 JTabbedPane	• Tab placement	`addTab()`
		`insertTab()`
		`getSelectedComponent()`
		`setSelectedComponent()`

 JScrollPane	• Component • Component, scrollbar options	`setViewportView()`
 JSplitPane	• Orientation, 2 components *For **orientation**, use* *`JScrollPane.HORIZONTAL_SPLIT`* *or* *`JScrollPane.VERTICAL_SPLIT`.*	`setDividerLocation()`

Layout Managers

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

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

1. BorderLayout

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

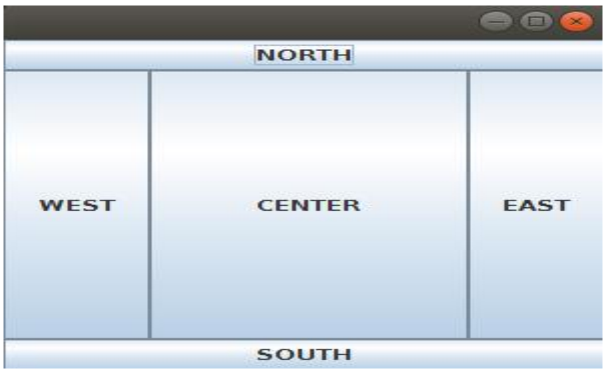
1. public static final int NORTH
2. public static final int SOUTH
3. public static final int EAST
4. public static final int WEST
5. public static final int CENTER

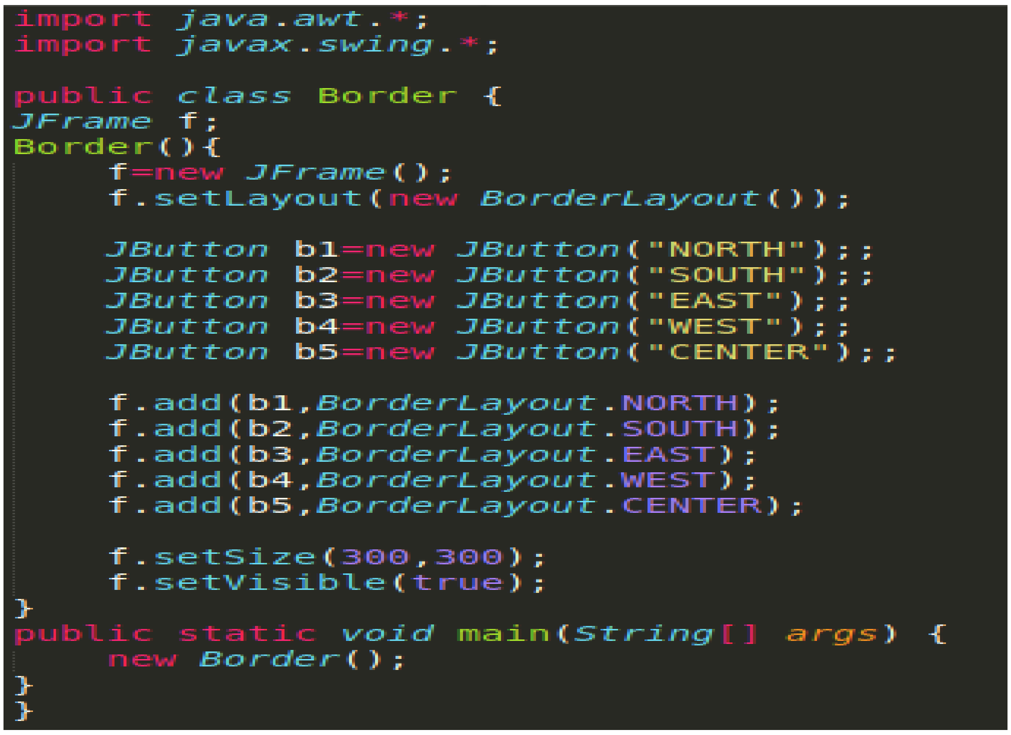
Constructors used:

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

public class Border {
JFrame f;
Border(){
    f=new JFrame();
    f.setLayout(new BorderLayout());

    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();
}
}
```

2. FlowLayout

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

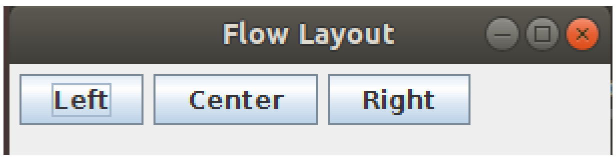
Fields of FlowLayout class

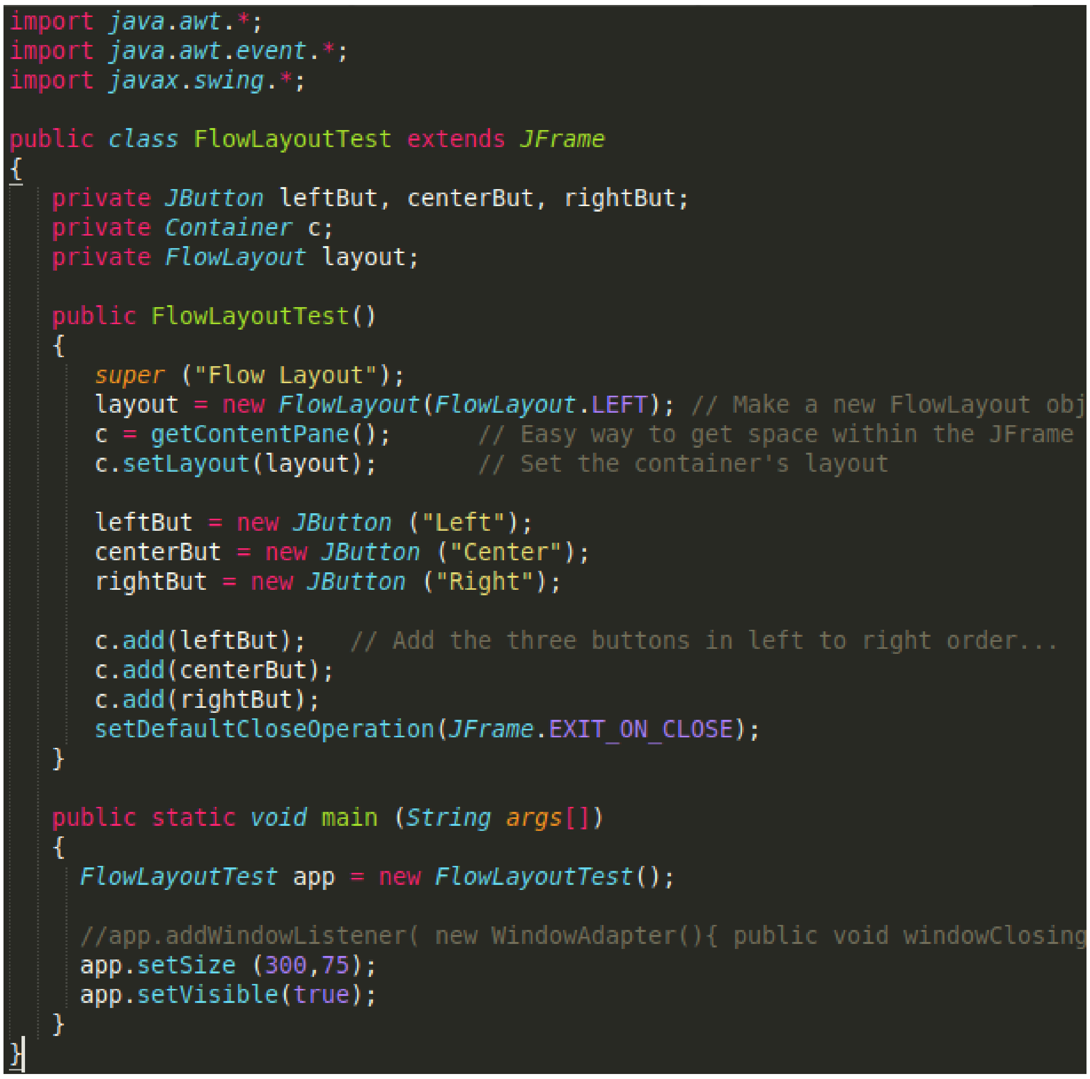
1. public static final int LEFT
2. public static final int RIGHT
3. public static final int CENTER
4. public static final int LEADING
5. public static final int TRAILING

Constructors of FlowLayout class

1. **FlowLayout():** creates a flow layout with centered alignment and a default 5 unit horizontal and vertical gap.
2. **FlowLayout(int align):** creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap.
3. **FlowLayout(int align, int hgap, int vgap):** creates a flow layout with the given alignment and the given horizontal and vertical gap.

Example:





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

public class FlowLayoutTest extends JFrame
{
    private JButton leftBut, centerBut, rightBut;
    private Container c;
    private FlowLayout layout;

    public FlowLayoutTest()
    {
        super ("Flow Layout");
        layout = new FlowLayout(FlowLayout.LEFT); // Make a new FlowLayout obj
        c = getContentPane();       // Easy way to get space within the JFrame
        c.setLayout(layout);        // Set the container's layout

        leftBut = new JButton ("Left");
        centerBut = new JButton ("Center");
        rightBut = new JButton ("Right");

        c.add(leftBut);   // Add the three buttons in left to right order...
        c.add(centerBut);
        c.add(rightBut);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

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

       //app.addWindowListener( new WindowAdapter(){ public void windowClosing
       app.setSize (300,75);
       app.setVisible(true);
    }
}
```

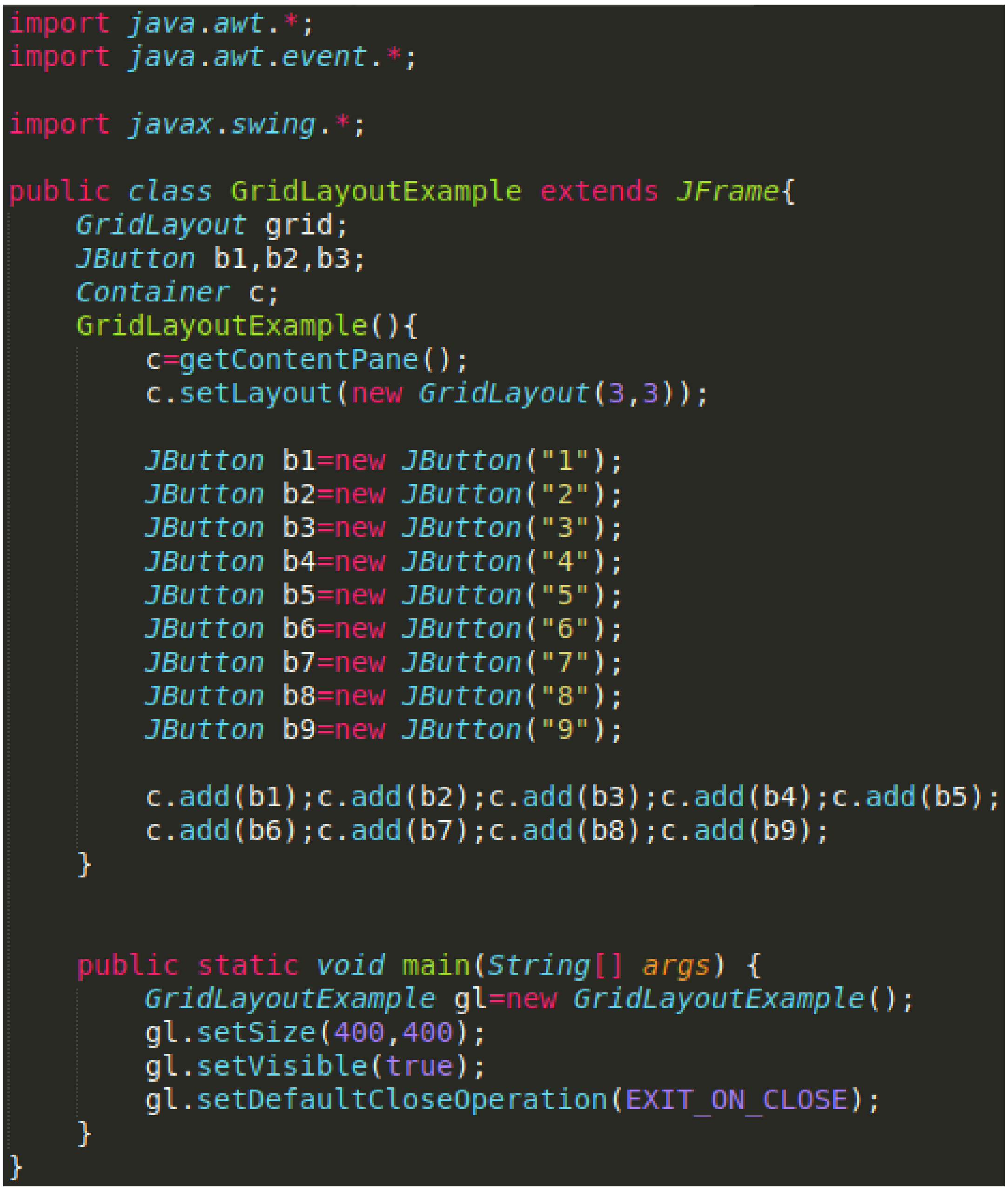
3. **GridLayout**

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

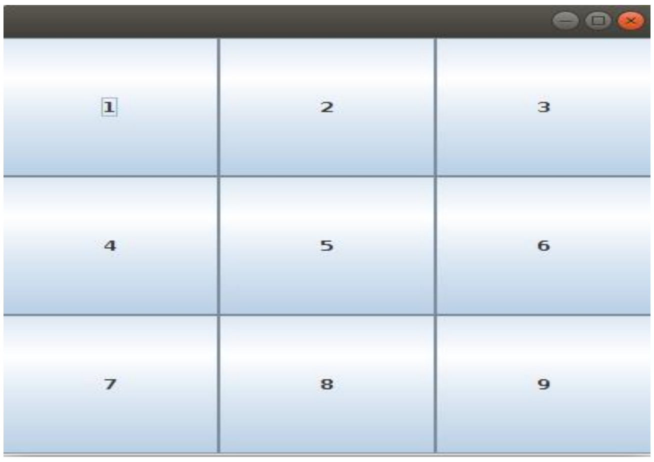
Constructors of GridLayout class

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

Example:



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

import javax.swing.*;

public class GridLayoutExample extends JFrame{
    GridLayout grid;
    JButton b1,b2,b3;
    Container c;
    GridLayoutExample(){
        c=getContentPane();
        c.setLayout(new GridLayout(3,3));

        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");

        c.add(b1);c.add(b2);c.add(b3);c.add(b4);c.add(b5);
        c.add(b6);c.add(b7);c.add(b8);c.add(b9);
    }


    public static void main(String[] args) {
        GridLayoutExample gl=new GridLayoutExample();
        gl.setSize(400,400);
        gl.setVisible(true);
        gl.setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
}
```



4. **CardLayout**

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

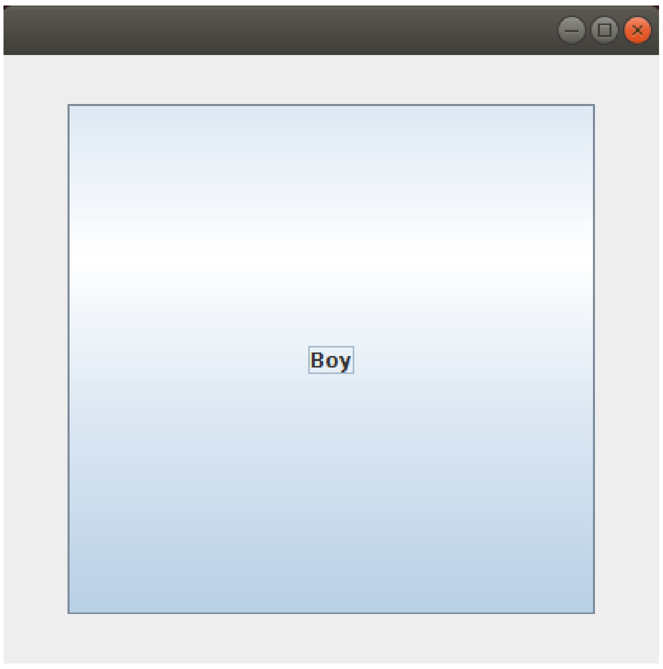
Constructors of CardLayout class

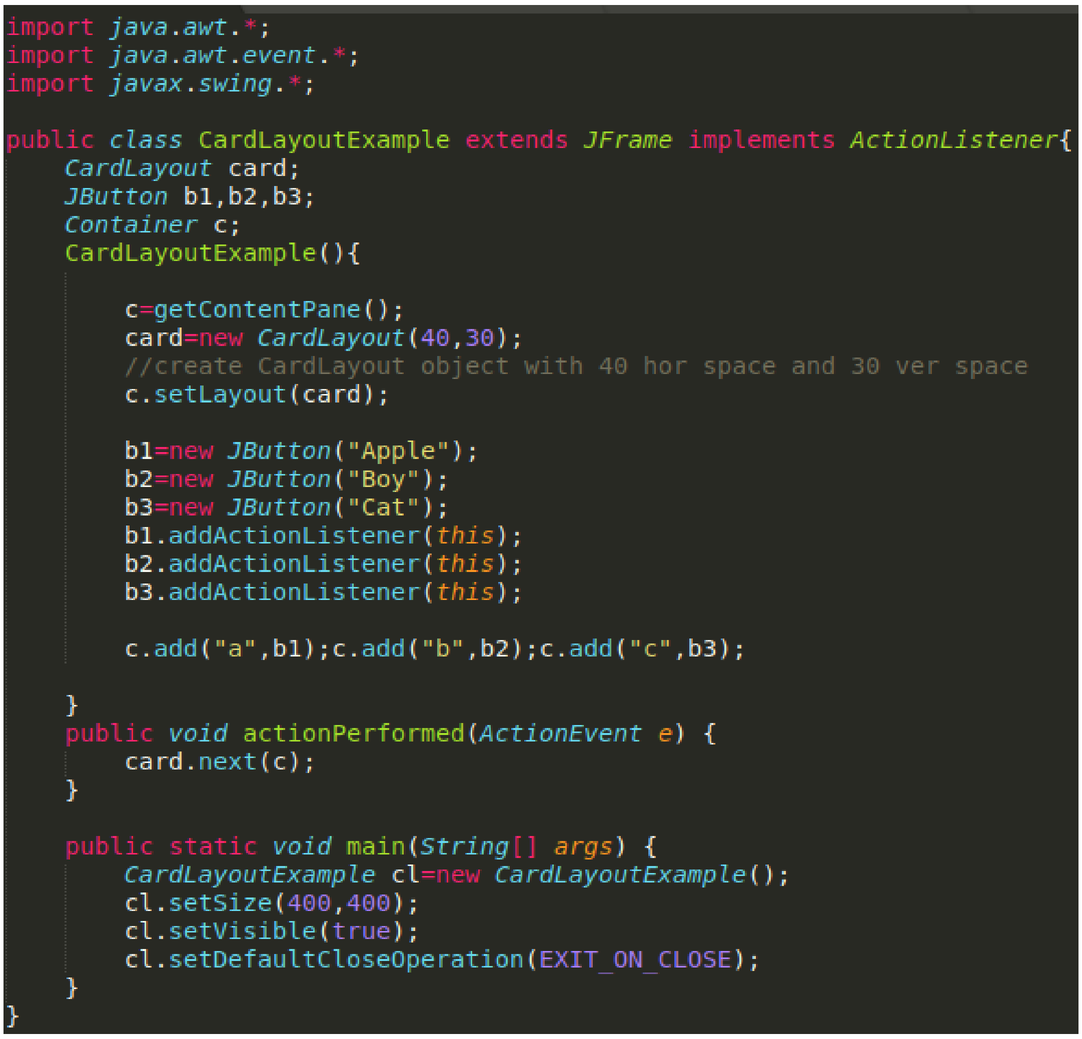
1. **CardLayout():** creates a card layout with zero horizontal and vertical gap.
2. **CardLayout(int hgap, int vgap):** creates a card layout with the given horizontal and vertical gap.

Commonly used methods of CardLayout class

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

Example:



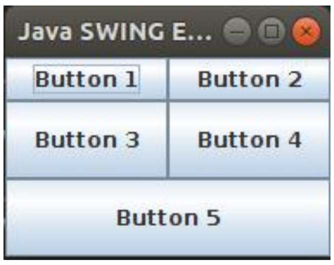


```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class CardLayoutExample extends JFrame implements ActionListener{
    CardLayout card;
    JButton b1,b2,b3;
    Container c;
    CardLayoutExample(){

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

        b1=new JButton("Apple");
        b2=new JButton("Boy");
        b3=new JButton("Cat");
        b1.addActionListener(this);
        b2.addActionListener(this);
        b3.addActionListener(this);

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

    }
    public void actionPerformed(ActionEvent e) {
        card.next(c);
    }

    public static void main(String[] args) {
        CardLayoutExample cl=new CardLayoutExample();
        cl.setSize(400,400);
        cl.setVisible(true);
        cl.setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
}
```

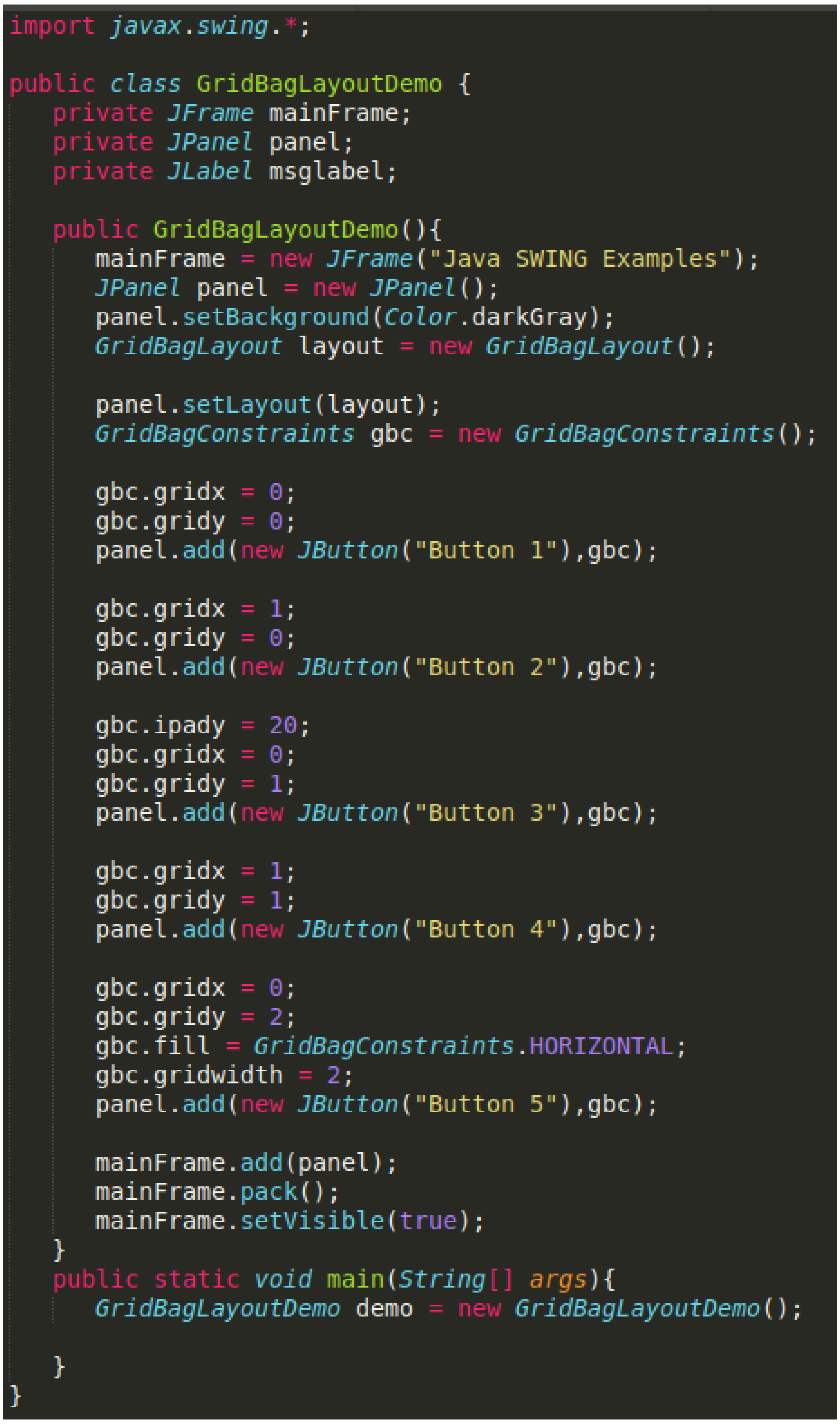
5. GridBagLayout

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

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



Example:



```java
import javax.swing.*;

public class GridBagLayoutDemo {
   private JFrame mainFrame;
   private JPanel panel;
   private JLabel msglabel;

   public GridBagLayoutDemo(){
      mainFrame = new JFrame("Java SWING Examples");
      JPanel panel = new JPanel();
      panel.setBackground(Color.darkGray);
      GridBagLayout layout = new GridBagLayout();

      panel.setLayout(layout);
      GridBagConstraints gbc = new GridBagConstraints();

      gbc.gridx = 0;
      gbc.gridy = 0;
      panel.add(new JButton("Button 1"),gbc);

      gbc.gridx = 1;
      gbc.gridy = 0;
      panel.add(new JButton("Button 2"),gbc);

      gbc.ipady = 20;
      gbc.gridx = 0;
      gbc.gridy = 1;
      panel.add(new JButton("Button 3"),gbc);

      gbc.gridx = 1;
      gbc.gridy = 1;
      panel.add(new JButton("Button 4"),gbc);

      gbc.gridx = 0;
      gbc.gridy = 2;
      gbc.fill = GridBagConstraints.HORIZONTAL;
      gbc.gridwidth = 2;
      panel.add(new JButton("Button 5"),gbc);

      mainFrame.add(panel);
      mainFrame.pack();
      mainFrame.setVisible(true);
   }
   public static void main(String[] args){
      GridBagLayoutDemo demo = new GridBagLayoutDemo();

   }
}
```

6. BoxLayout

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

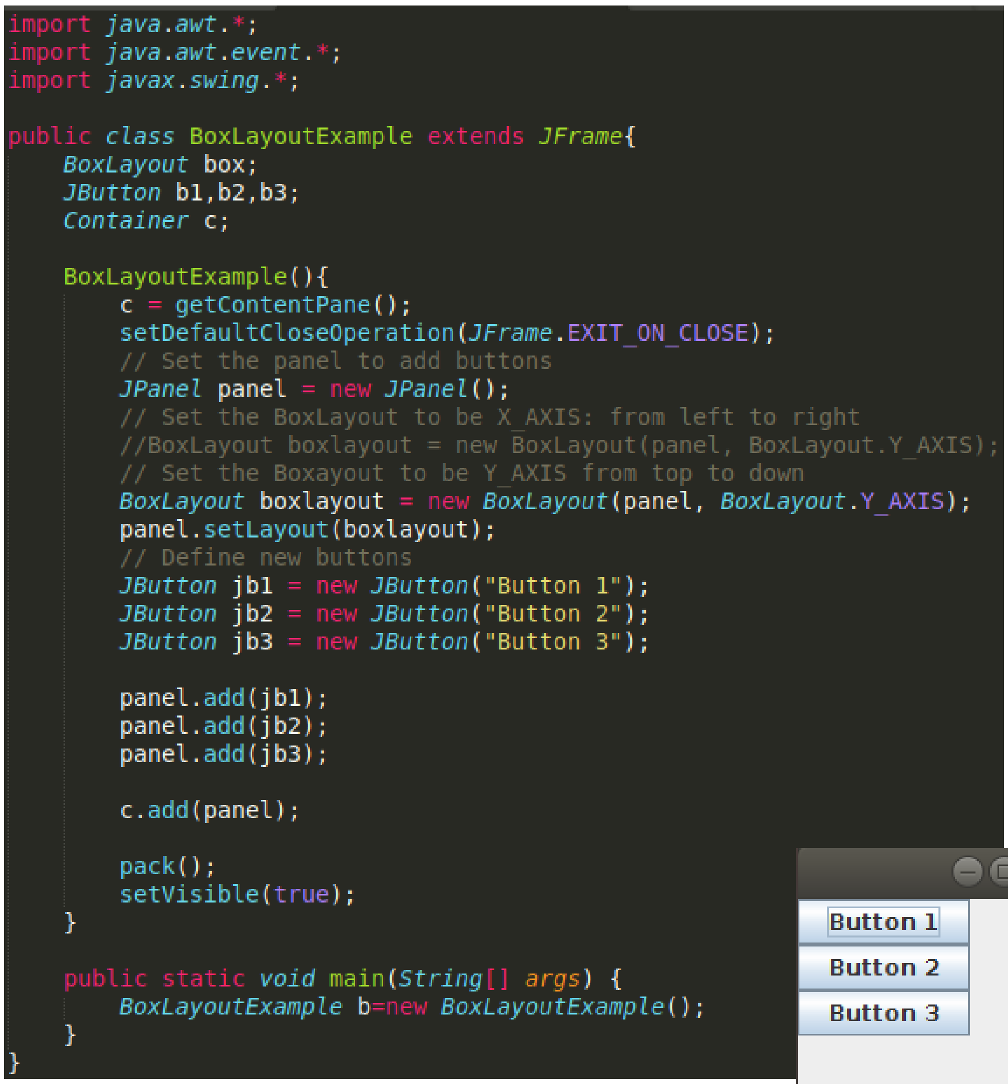
Fields of BoxLayout class

1. public static final int X_AXIS
2. public static final int Y_AXIS
3. public static final int LINE_AXIS
4. public static final int PAGE_AXIS

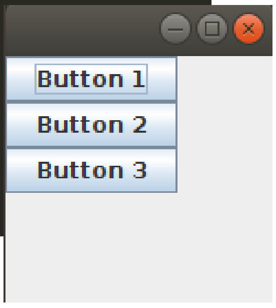
Constructor of BoxLayout class

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

Example:



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

public class BoxLayoutExample extends JFrame{
    BoxLayout box;
    JButton b1,b2,b3;
    Container c;

    BoxLayoutExample(){
        c = getContentPane();
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // Set the panel to add buttons
        JPanel panel = new JPanel();
        // Set the BoxLayout to be X_AXIS: from left to right
        //BoxLayout boxlayout = new BoxLayout(panel, BoxLayout.Y_AXIS);
        // Set the Boxayout to be Y_AXIS from top to down
        BoxLayout boxlayout = new BoxLayout(panel, BoxLayout.Y_AXIS);
        panel.setLayout(boxlayout);
        // Define new buttons
        JButton jb1 = new JButton("Button 1");
        JButton jb2 = new JButton("Button 2");
        JButton jb3 = new JButton("Button 3");

        panel.add(jb1);
        panel.add(jb2);
        panel.add(jb3);

        c.add(panel);

        pack();
        setVisible(true);
    }

    public static void main(String[] args) {
        BoxLayoutExample b=new BoxLayoutExample();
    }
}
```



Event Handling

Any program that uses GUI (graphical user interface) such as Java application written for windows, is event driven.

Event describes the change in state of any object. **For Example:** Pressing a button, Entering a character in Textbox, Clicking or Dragging a mouse, etc.

Event listeners represent the interfaces responsible to handle events. Every method of an event listener method has a single argument as an object which is the subclass of EventObject class.

For example, mouse event listener methods will accept instance of MouseEvent, where MouseEvent derives from EventObject.

Event handling has three main components,

- **Events :** An event is a change in state of an object.
- **Events Source :** Event source is an object that generates an event.
- **Listeners :** A listener is an object that listens to the event. A listener gets notified when an event occurs.

Steps to perform Event Handling

Following steps are required to perform event handling:

1. Implement appropriate interface in the class.
2. Register the component with the Listener

Registration Methods

For registering the component with the Listener, many classes provide the registration methods. For example:

- **Button**
  public void addActionListener(ActionListener a)
- **MenuItem**
  public void addActionListener(ActionListener a)
- **TextField**
  public void addActionListener(ActionListener a)
  public void addTextListener(TextListener a)
- **TextArea**
  public void addTextListener(TextListener a)
- **Checkbox**

public void addItemListener(ItemListener a)

- **Choice**

  public void addItemListener(ItemListener a)

- **List**

  public void addActionListener(ActionListener a)

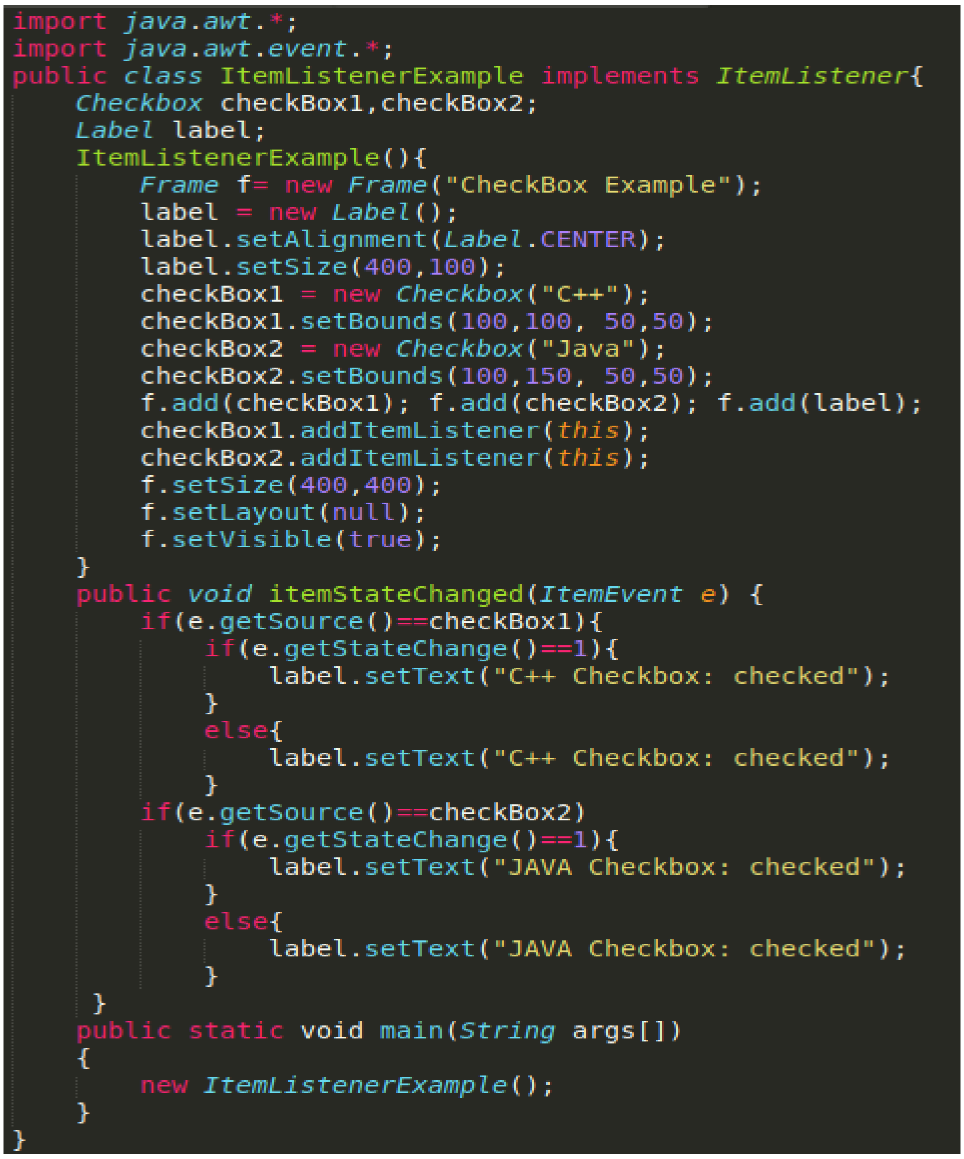
  public void addItemListener(ItemListener a)

Java Event Handling Code

We can put the event handling code into one of the following places:

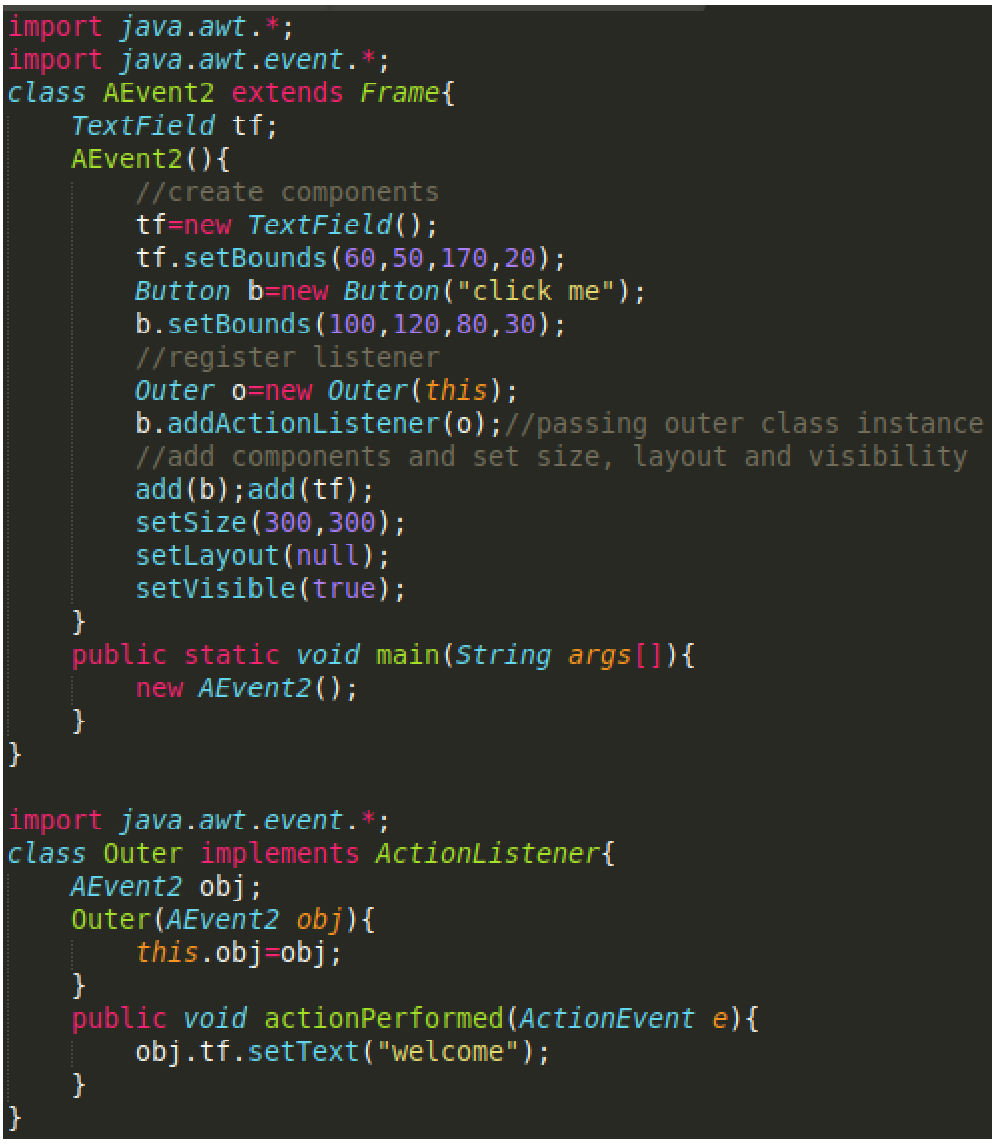
1. Within class
2. Other class
3. Anonymous class

1. Implement within Class



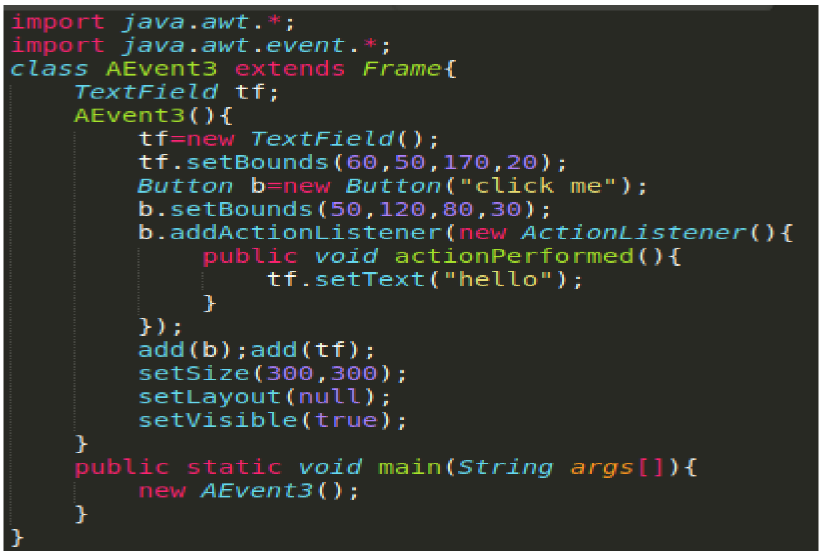
```java
import java.awt.*;
import java.awt.event.*;
public class ItemListenerExample implements ItemListener{
    Checkbox checkBox1,checkBox2;
    Label label;
    ItemListenerExample(){
        Frame f= new Frame("CheckBox Example");
        label = new Label();
        label.setAlignment(Label.CENTER);
        label.setSize(400,100);
        checkBox1 = new Checkbox("C++");
        checkBox1.setBounds(100,100, 50,50);
        checkBox2 = new Checkbox("Java");
        checkBox2.setBounds(100,150, 50,50);
        f.add(checkBox1); f.add(checkBox2); f.add(label);
        checkBox1.addItemListener(this);
        checkBox2.addItemListener(this);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
    }
    public void itemStateChanged(ItemEvent e) {
        if(e.getSource()==checkBox1){
            if(e.getStateChange()==1){
                label.setText("C++ Checkbox: checked");
            }
            else{
                label.setText("C++ Checkbox: checked");
            }
        if(e.getSource()==checkBox2)
            if(e.getStateChange()==1){
                label.setText("JAVA Checkbox: checked");
            }
            else{
                label.setText("JAVA Checkbox: checked");
            }
     }
    public static void main(String args[])
    {
        new ItemListenerExample();
    }
}
```

2. Implement in some Other Class



```java
import java.awt.*;
import java.awt.event.*;
class AEvent2 extends Frame{
    TextField tf;
    AEvent2(){
        //create components
        tf=new TextField();
        tf.setBounds(60,50,170,20);
        Button b=new Button("click me");
        b.setBounds(100,120,80,30);
        //register listener
        Outer o=new Outer(this);
        b.addActionListener(o);//passing outer class instance
        //add components and set size, layout and visibility
        add(b);add(tf);
        setSize(300,300);
        setLayout(null);
        setVisible(true);
    }
    public static void main(String args[]){
        new AEvent2();
    }
}

import java.awt.event.*;
class Outer implements ActionListener{
    AEvent2 obj;
    Outer(AEvent2 obj){
        this.obj=obj;
    }
    public void actionPerformed(ActionEvent e){
        obj.tf.setText("welcome");
    }
}
```

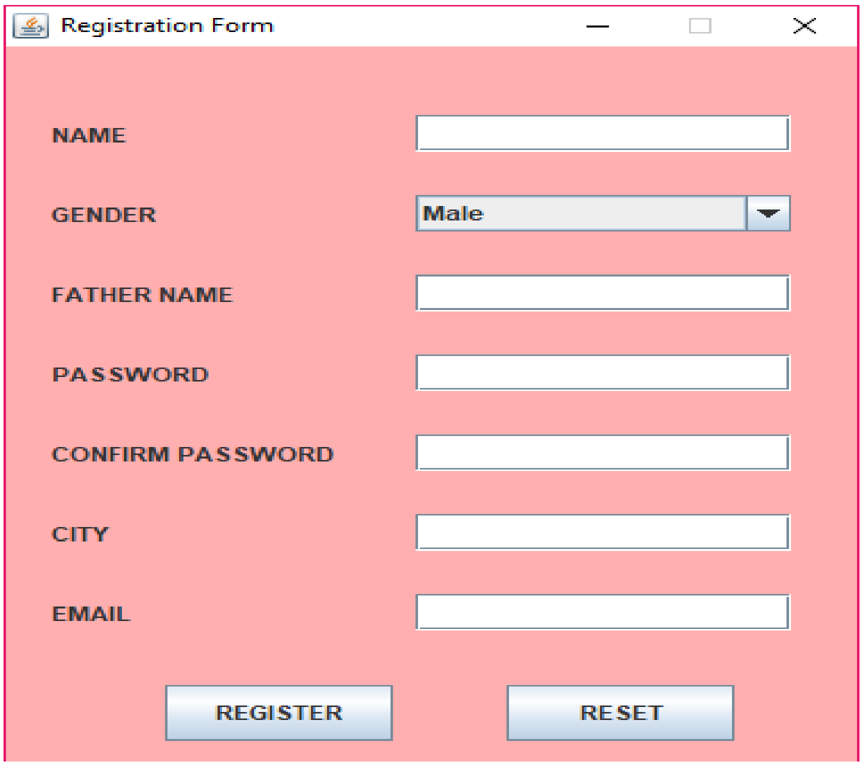
3. Anonymous Inner Class



```java
import java.awt.*;
import java.awt.event.*;
class AEvent3 extends Frame{
    TextField tf;
    AEvent3(){
        tf=new TextField();
        tf.setBounds(60,50,170,20);
        Button b=new Button("click me");
        b.setBounds(50,120,80,30);
        b.addActionListener(new ActionListener(){
            public void actionPerformed(){
                tf.setText("hello");
            }
        });
        add(b);add(tf);
        setSize(300,300);
        setLayout(null);
        setVisible(true);
    }
    public static void main(String args[]){
        new AEvent3();
    }
}
```

Class Tasks

Task#01: Create a form as follows.



Task#02: Use Layout Managers such that the button panel always appears at the bottom.

Task#03: Add code such that on clicking the **Register** button, a dialog box pops up which displays all the user entered information.(Use JOptionPane) and on clicking **Reset** button, all the TextFields and ComboBox selections become null.