UNIT-V-PART-II

UNIT – V

Event Handling : Events, Event sources, Event Listeners, Event classes, Event listener interface, Handling mouse and keyboard events, Adapter classes, The AWT class hierarchy, AWT controls- labels, buttons, scrollbars, text field, check box, check box groups, choices, handling lists, dialogs, Menubar, layout manager- Flow, Border, Grid, Card

Abstract Window Toolkit(AWT)

Java AWT is an API that contains large number of classes and methods to create and manage graphical user interface (GUI) applications.

The AWT was designed to provide a common set of tools for GUI design that could work on a variety of platforms.

The tools provided by the AWT are implemented using each platform's native GUI toolkit, hence preserving the look and feel of each platform. This is an advantage of using AWT.

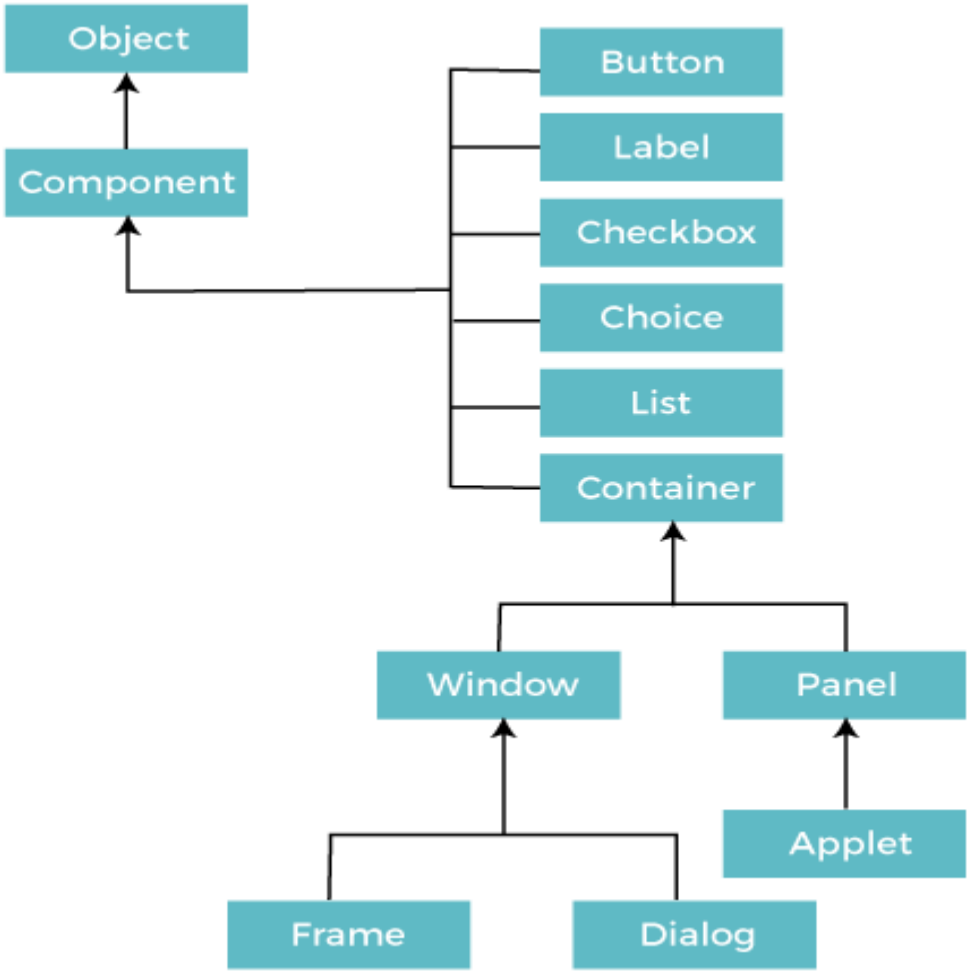
But the disadvantage of such an approach is that GUI designed on one platform may look different when displayed on another platform that means AWT component are platform dependent.

AWT is the foundation upon which Swing is made i.e Swing is a improved GUI API that extends the AWT.

But now a days AWT is merely used because most GUI Java programs are implemented using Swing because of its rich implementation of GUI controls and light-weighted nature.

Java AWT Hierarchy

The hierarchy of Java AWT classes are given below, all the classes are available in **java.awt** package.



Component class

Component class is at the top of AWT hierarchy. It is an abstract class that encapsulates all the attributes of visual component. A component object is responsible for remembering the current foreground and background colors and the currently selected text font.

Container

Container is a component in AWT that contains another component like button, text field, tables etc. **Container** is a subclass of component class. **Container** class keeps track of components that are added to another component.

Panel

Panel class is a concrete subclass of **Container**. Panel does not contain title bar, menu bar or border. It is container that is used for holding components.

Window class

Window class creates a top level window. Window does not have borders and menubar.

Frame

Frame is a subclass of **Window** and have resizing canvas. It is a container that contain several different components like button, title bar, textfield, label etc.

In Java, most of the AWT applications are created using **Frame** window. Frame class has two different constructors,

Creating a Frame

There are two ways to create a Frame. They are,

By Instantiating Frame class

By extending Frame class

Creating Frame Window by Instantiating Frame class

```
import java.awt.*;
public class Testawt
{
  Testawt()
  {
    Frame fm=new Frame();   //Creating a frame
    Label lb = new Label("welcome to java graphics");  //Creating a label
    fm.add(lb);                //adding label to the frame
    fm.setSize(300, 300);  //setting frame size.
```

```
    fm.setVisible(true);     //set frame visibilty true
  }
  public static void main(String args[])
  {
    Testawt ta = new Testawt();
  }
}
```

O/P:-

Creating Frame window by extending Frame class

```
package testawt;

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

public class Testawt extends Frame
{
    public Testawt()
    {
```

```
        Button btn=new Button("Hello World");
        add(btn);                //adding a new Button.
        setSize(400, 500);       //setting size.
        setTitle("java"); //setting title.
        setLayout(new FlowLayout());          //set default layout for frame.
        setVisible(true);         //set frame visibilty true.
    }

    public static void main (String[] args)
    {
        Testawt ta = new Testawt(); //creating a frame.
    }
}
```

O/P:-

AWT Button

In Java, AWT contains a Button Class. It is used for creating a labelled button which can perform an action.

AWT Button Classs Declaration:

public class Button extends Component implements Accessible

Example:

Lets take an example to create a button and it to the frame by providing coordinates.

```
import java.awt.*;
public class ButtonDemo1
{
public static void main(String[] args)
{
   Frame f1=new Frame("java ==> Button Demo");
   Button b1=new Button("Press Here");
   b1.setBounds(80,200,80,50);
   f1.add(b1);
   f1.setSize(500,500);
   f1.setLayout(null);
   f1.setVisible(true);
}
}
```

O/P:-

AWT Label

In Java, AWT contains a Label Class. It is used for placing text in a container. Only Single line text is allowed and the text can not be changed directly.

Label Declaration:

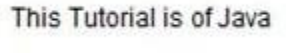
public class Label extends Component implements Accessible

Example:

In this example, we are creating two labels to display text to the frame.

```
import java.awt.*;
class LabelDemo1
{
  public static void main(String args[])
  {
    Frame l_Frame= new Frame("JAVA ==> Label Demo");
    Label lab1,lab2;
    lab1=new Label("Welcome to JAVA ");
    lab1.setBounds(50,50,200,30);
    lab2=new Label("This Tutorial is of Java");
    lab2.setBounds(50,100,200,30);
l_Frame.add(lab1);
l_Frame.add(lab2);
l_Frame.setSize(500,500);
l_Frame.setLayout(null);
l_Frame.setVisible(true);
}
}
```

o/p:-



AWT TextField

In Java, AWT contains aTextField Class. It is used for displaying single line text.

TextField Declaration:

public class TextField extends TextComponent

Example:

We are creating two textfields to display single line text string. This text is editable in nature, see the below example.

```
import java.awt.*;
class TextFieldDemo1{
public static void main(String args[]){
   Frame TextF_f= new Frame("java ==>TextField");
TextField text1,text2;
   text1=new TextField("Welcome to java");
   text1.setBounds(60,100, 230,40);
   text2=new TextField("This tutorial is of Java");
   text2.setBounds(60,150, 230,40);
TextF_f.add(text1);
TextF_f.add(text2);
TextF_f.setSize(500,500);
TextF_f.setLayout(null);
TextF_f.setVisible(true);
}
}
```

o/p:-

AWT TextArea

In Java, AWT contains aTextArea Class. It is used for displaying multiple-line text.

TextArea Declaration:

public class TextArea extends TextComponent

Example:

In this example, we are creating a TextArea that is used to display multiple-line text string and allows text editing as well.

```
import java.awt.*;
public class TextAreaDemo1
{
  TextAreaDemo1()
  {
    Frame textArea_f= new Frame();
    TextArea area=new TextArea("Welcome to java");
    area.setBounds(30,40, 200,200);
    textArea_f.add(area);
    textArea_f.setSize(300,300);
    textArea_f.setLayout(null);
    textArea_f.setVisible(true);
  }
  public static void main(String args[])
  {
    new TextAreaDemo1();
  }
}
```

o/p:-

Welcome to java

AWT Checkbox

In Java, AWT contains a Checkbox Class. It is used when we want to select only one option i.e true or false. When the checkbox is checked then its state is "on" (true) else it is "off"(false).

Checkbox Syntax

public class Checkbox extends Component implements ItemSelectable, Accessible

Example:

In this example, we are creating checkbox that are used to get user input. If checkbox is checked it returns true else returns false.

```
import java.awt.*;
public class CheckboxDemo1
{
  CheckboxDemo1(){
    Frame checkB_f= new Frame("java ==>Checkbox Example");
    Checkbox ckbox1 = new Checkbox("Yes", true);
    ckbox1.setBounds(100,100, 60,60);
    Checkbox ckbox2 = new Checkbox("No");
    ckbox2.setBounds(100,150, 60,60);
    checkB_f.add(ckbox1);
    checkB_f.add(ckbox2);
    checkB_f.setSize(400,400);
    checkB_f.setLayout(null);
    checkB_f.setVisible(true);
  }
  public static void main(String args[])
  {
```

```
    new CheckboxDemo1();
  }
}
```

```
 awt-checkbox
 awt-checkbox
```

AWT CheckboxGroup

In Java, AWT contains aCheckboxGroup Class. It is used to group a set of Checkbox. When Checkboxes are grouped then only one box can be checked at a time.

CheckboxGroup Declaration:

public class CheckboxGroup extends Object implements Serializable

Example:

This example creates a checkboxgroup that is used to group multiple checkbox in a single unit. It is helpful when we have to select single choice among the multiples.

```
import java.awt.*;
public class CheckboxGroupDemo
{
  CheckboxGroupDemo(){
    Frame ck_groupf= new Frame("java ==>CheckboxGroup");
    CheckboxGroupobj = new CheckboxGroup();
    Checkbox ckBox1 = new Checkbox("Yes", obj, true);
    ckBox1.setBounds(100,100, 50,50);
    Checkbox ckBox2 = new Checkbox("No", obj, false);
```

```
    ckBox2.setBounds(100,150, 50,50);
    ck_groupf.add(ckBox1);
    ck_groupf.add(ckBox2);
    ck_groupf.setSize(400,400);
    ck_groupf.setLayout(null);
    ck_groupf.setVisible(true);
  }
  public static void main(String args[])
  {
    new CheckboxGroupDemo();
  }
}
```

o/p:-

AWT Choice

In Java, AWT contains a Choice Class. It is used for creating a drop-down menu of choices. When a user selects a particular item from the drop-down then it is shown on the top of the menu.

Choice Declaration:

public class Choice extends Component implements ItemSelectable, Accessible

Example:

In this example, we are creating drop-down menu that is used to get user choice from multiple choices.

```
import java.awt.*;
public class ChoiceDemo
{
  ChoiceDemo()
  {
    Frame choice_f= new Frame();
    Choice obj=new Choice();
    obj.setBounds(80,80, 100,100);
    obj.add("Red");
    obj.add("Blue");
    obj.add("Black");
    obj.add("Pink");
    obj.add("White");
    obj.add("Green");
    choice_f.add(obj);
    choice_f.setSize(400,400);
    choice_f.setLayout(null);
    choice_f.setVisible(true);
  }
  public static void main(String args[])
  {
    new ChoiceDemo();
  } }
```

AWT List

In Java, AWT contains a List Class. It is used to represent a list of items together. One or more than one item can be selected from the list.

List Declaration:

public class List extends Component implements ItemSelectable, Accessible

Example:

In this example, we are creating a list that is used to list out the items.

```
import java.awt.*;
public class ListDemo
{
  ListDemo()
  {
    Frame list_f= new Frame();
    List obj=new List(6);
    obj.setBounds(80,80, 100,100);
    obj.add("Red");
    obj.add("Blue");
    obj.add("Black");
    obj.add("Pink");
    obj.add("White");
    obj.add("Green");
    list_f.add(obj);
    list_f.setSize(400,400);
    list_f.setLayout(null);
```

```
    list_f.setVisible(true);
  }
  public static void main(String args[])
  {
    new ListDemo();
  }
}
```

AWT Canvas

In Java, AWT contains a Canvas Class. A blank rectangular area is provided. It is used when a user wants to draw on the screen.

Declaration:

public class Canvas extends Component implements Accessible

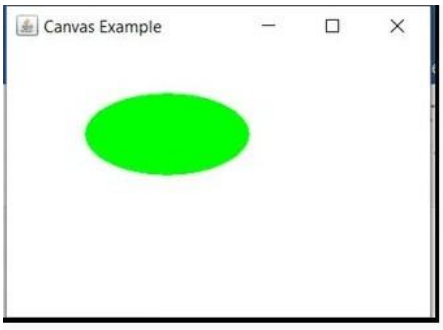
Example:

The canvas is used to provide a place to draw using mouse pointer. We can used it to get user architectural user input.

```
import java.awt.*;
public class CanvasDemo1
{
  public CanvasDemo1()
  {
    Frame canvas_f= new Frame("java==> Canvas");
    canvas_f.add(new CanvasDemo());
    canvas_f.setLayout(null);
    canvas_f.setSize(500, 500);
    canvas_f.setVisible(true);
  }
```

```
  public static void main(String args[])
  {
    new CanvasDemo1();
  }
}
class CanvasDemo extends Canvas
{
  public CanvasDemo() {
    setBackground (Color.WHITE);
    setSize(300, 200);
  }
  public void paint(Graphics g)
  {
    g.setColor(Color.green);
    g.fillOval(80, 80, 150, 75);
  }
}
```

o/p:-



AWT MenuItem and Menu

In Java, AWT contains a MenuItem and Menu Class.MenuItem is used for adding Lable in Menu. The menu is used to create a drop-down of menu components

MenuItem declaration

public class MenuItem extends MenuComponent implements Accessible

Example:

In this example, we are creating a menu item that contains sub menu as well. We use MenuItem and Menu class for creating menu.

```
import java.awt.*;
class MenuDemo1
{
  MenuDemo1()
  {
    Frame menu_f= new Frame("java ==> Menu and MenuItem Demo");
    MenuBarmenu_bar=new MenuBar();
    Menu menu11=new Menu("Menu");
    Menu sub_menu1=new Menu("Sub Menu =>");
    MenuItem a1=new MenuItem("Red");
    MenuItem a2=new MenuItem("Light Red");
    MenuItem a3=new MenuItem("Drak Red");
    MenuItem b1=new MenuItem("Green");
    MenuItem b2=new MenuItem("Light Green");
```

```
    MenuItem b3=new MenuItem("Dark Green");
    menu11.add(a1);
    sub_menu1.add(a2);
    sub_menu1.add(a3);
    menu11.add(b1);
    sub_menu1.add(b2);
    sub_menu1.add(b3);
    menu11.add(sub_menu1);
    menu_bar.add(menu11);
    menu_f.setMenuBar(menu_bar);
    menu_f.setSize(400,400);
    menu_f.setLayout(null);
    menu_f.setVisible(true);
  }
  public static void main(String args[])
  {
    new MenuDemo1();
  }
}
```

o/p:-

AWT Panel

In Java, AWT contains a Panel. The panel provides a free space where components can be placed.

Panel declaration

public class Panel extends Container implements Accessible

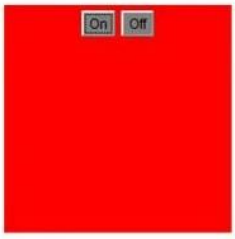
Example:

Lets create a panel to add components like: button, textbox etc. the panel provides a place to add awt components.

```
import java.awt.*;
public class PanelDemo1{
 PanelDemo1()
 {
  Frame panel_f= new Frame("studytonight ==> Panel Demo");
  Panel panel11=new Panel();
  panel11.setBounds(40,80,200,200);
  panel11.setBackground(Color.red);
  Button box1=new Button("On");
  box1.setBounds(50,100,80,30);
  box1.setBackground(Color.gray);
  Button box2=new Button("Off");
  box2.setBounds(100,100,80,30);
  box2.setBackground(Color.gray);
  panel11.add(box1);
  panel11.add(box2);
  panel_f.add(panel11);
  panel_f.setSize(400,400);
  panel_f.setLayout(null);
  panel_f.setVisible(true);
 }
```

```
  public static void main(String args[])
  {
    new PanelDemo1();
  }
}
```

o/p:-



AWT Dialog

In Java, AWT contains a Dialog. It is a type of window which is having a border and a title. But it does not have any maximize and minimize button.

Declaration

public class Dialog extends Window

Example:

In this example, we are creating a dialogue box. The dialogue box is used to provide information to the user.

```
import java.awt.*;
import java.awt.event.*;
public class DialogDemo1
{
```

```
 private static Dialog dialog_d;
 DialogDemo1()
 {
  Frame dialog_f= new Frame();
  dialog_d = new Dialog(dialog_f , "studytonight ==> Dialog Demo", true);
  dialog_d.setLayout( new FlowLayout() );
  Button dialog_b = new Button ("OK");
  dialog_b.addActionListener ( new ActionListener()
  {
    public void actionPerformed( ActionEvent e )
    {
      DialogDemo1.dialog_d.setVisible(false);
    }
  });
  dialog_d.add( new Label ("Welcome to studytonight. Click on button to continue."));
  dialog_d.add(dialog_b);
  dialog_d.setSize(300,300);
  dialog_d.setVisible(true);
 }
 public static void main(String args[])
 {
  new DialogDemo1();
 }
}
```

Layout Managers

In Java, Layout Managers is used for arranging the components in order. LayoutMananger is an interface which implements the classes of the layout manager.

Below are some of the class which are used for the representation of layout manager.

1. java.awt.BorderLayout
2. java.awt.FlowLayout
3. java.awt.GridLayout
4. java.awt.CardLayout
5. javax.swing.BoxLayo

Border Layout

BorderLayout is used, when we want to arrange the components in five regions.

The five regions can be north, south, east, west and the centre.

There are 5 types of constructor in Border Layout. They are as following:

1. public static final int NORTH

2. public static final int SOUTH

3. public static final int EAST

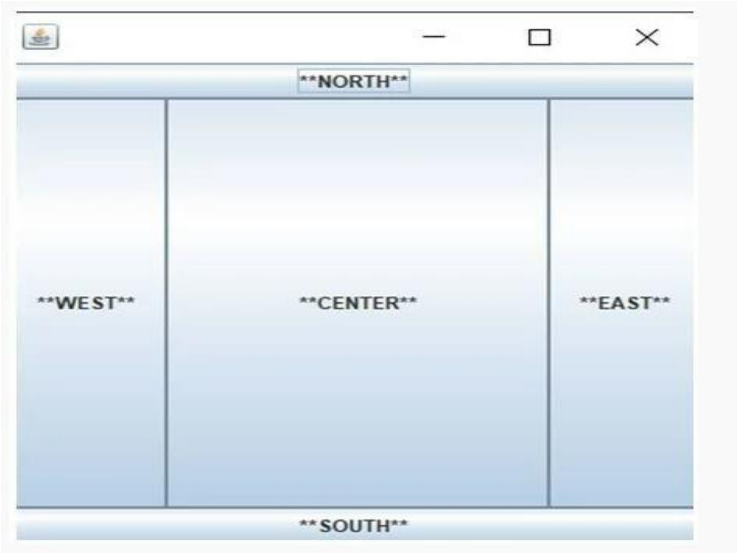
4. public static final int WEST

5. public static final int CENTER

Example:-

```
import java.awt.*;
import javax.swing.*;
public class BorderDemo1
{
JFrame frame;
BorderDemo1()
{
        frame=new JFrame();
        JButton box1=new JButton("**NORTH**");;
        JButton box2=new JButton("**SOUTH**");;
        JButton box3=new JButton("**EAST**");;
        JButton box4=new JButton("**WEST**");;
        JButton box5=new JButton("**CENTER**");;
        frame.add(box1,BorderLayout.NORTH);
        frame.add(box2,BorderLayout.SOUTH);
        frame.add(box3,BorderLayout.EAST);
```

```
            frame.add(box4,BorderLayout.WEST);
            frame.add(box5,BorderLayout.CENTER);
            frame.setSize(400,400);
            frame.setVisible(true);
}
public static void main(String[] args)
{
    new BorderDemo1();
}
}
```



Grid Layout

Grid Layout is used, when we want to arrange the components in a rectangular grid.

There are 3 types of constructor in Grid Layout. They are as following:

1. GridLayout()

2. GridLayout(int rows, int columns)

3. GridLayout(int rows, int columns, inthgap, int vgap)

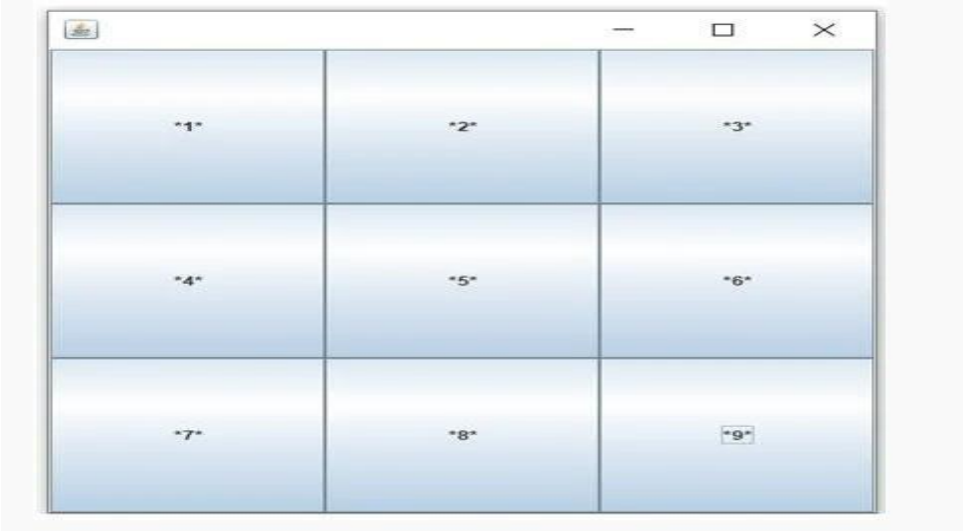
Example:

```
import java.awt.*;
import javax.swing.*;
public class GridDemo1
{
JFrame frame1;
GridDemo1(){
    frame1=new JFrame();
JButton box1=new JButton("*1*");
JButton box2=new JButton("*2*");
```

```
JButton box3=new JButton("*3*");
JButton box4=new JButton("*4*");
JButton box5=new JButton("*5*");
JButton box6=new JButton("*6*");
JButton box7=new JButton("*7*");
JButton box8=new JButton("*8*");
JButton box9=new JButton("*9*");

    frame1.add(box1);
    frame1.add(box2);
    frame1.add(box3);
    frame1.add(box4);
    frame1.add(box5);
    frame1.add(box6);
    frame1.add(box7);
    frame1.add(box8);
    frame1.add(box9);
    frame1.setLayout(new GridLayout(3,3));
    frame1.setSize(500,500);
```

```
    frame1.setVisible(true);
}
public static void main(String[] args)
 {
    new GridDemo1();
}
}
```

o/p:-



Flow Layout

Flow Layout is used, when we want to arrange the components in a sequence one after another.

There are 3 types of constructor in the Flow Layout. They are as following:

1. FlowLayout()

2. FlowLayout(int align)

3. FlowLayout(int align, inthgap, intvgap)

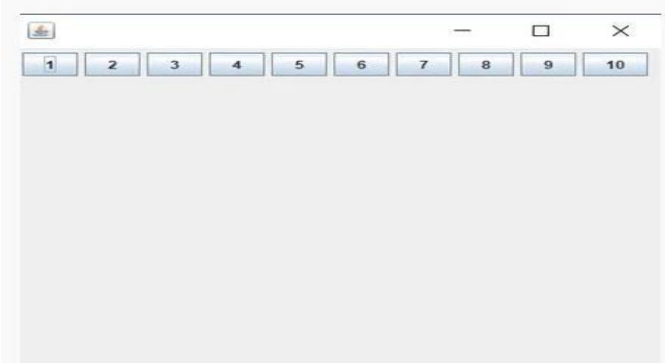
Example:-

```
import java.awt.*;

import javax.swing.*;

public class FlowDemo1

{

JFrame frame1;

FlowDemo1()

{

 frame1=new JFrame();

JButton box1=new JButton("1");

JButton box2=new JButton("2");

JButton box3=new JButton("3");

JButton box4=new JButton("4");

JButton box5=new JButton("5");
```

```
JButton box6=new JButton("6");
JButton box7=new JButton("7");
JButton box8=new JButton("8");
JButton box9=new JButton("9");
JButton box10=new JButton("10");
    frame1.add(box1);
    frame1.add(box2);
    frame1.add(box3);
    frame1.add(box4);
    frame1.add(box5);
    frame1.add(box6);
    frame1.add(box7);
    frame1.add(box8);
    frame1.add(box9);
    frame1.add(box10);
    frame1.setLayout(new FlowLayout(FlowLayout.LEFT));
    frame1.setSize(400,400);
    frame1.setVisible(true);
}
```

```
public static void main(String[] args)
 {
    new FlowDemo1();
}
}
```

o/p:-



Box Layout

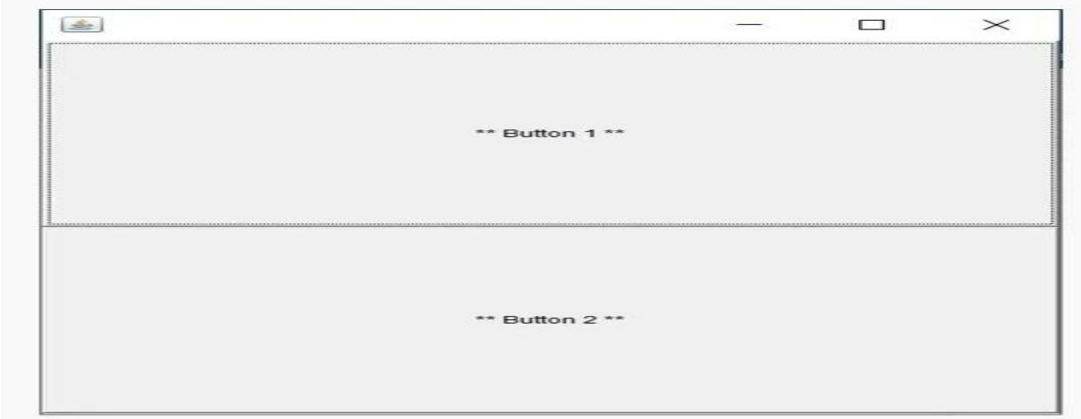
Box Layout is used, when we want to arrange the components vertically or horizontally.

BoxLayout(Container c, int axis)is the only constructor in the Box Layout

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


public class BoxDemo1 extends Frame
{
```

```
    Button buttonBox[];
    public BoxDemo1 ()
    {
            buttonBox = new Button [2];
            for (inti = 0;i<2;i++)
            {
    buttonBox[i] = new Button ("** Button " + (i + 1)+" **");
    add (buttonBox[i]);
      }
    setLayout (new BoxLayout (this, BoxLayout.Y_AXIS));
    setSize(500,500);
    setVisible(true);
    }

public static void main(String args[])
{
    BoxDemo1 obj=new BoxDemo1();
}
}
```

o/p:-



Card Layout

Card Layout is used, when we want to see only one component at a time.

There are 2 types of constructor in the Card Layout. They are as following:

1. CardLayout()

2. CardLayout(inthgap, intvgap)

Example:

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class CardDemo1 extends JFrame implements ActionListener
{
CardLayoutc_Card;
JButton box1,box2,box3;
Container d;
    CardDemo1()
{
            d=getContentPane();
            c_Card=new CardLayout(40,30);
            d.setLayout(c_Card);

             box1=new JButton("Java1");
            box2=new JButton("java2");
            box3=new JButton("java3");
            box1.addActionListener(this);
            box2.addActionListener(this);
```

```
        box3.addActionListener(this);


        d.add("P",box1);
        d.add("Q",box2);
        d.add("R",box3);
    }
    public void actionPerformed(ActionEvent e)
{
        c_Card.next(d);
    }
    public static void main(String[] args)
{
        CardDemo1 obj =new CardDemo1();
        obj.setSize(500,500);
        obj.setVisible(true);
        obj.setDefaultCloseOperation(EXIT_ON_CLOSE);
    }
}
```