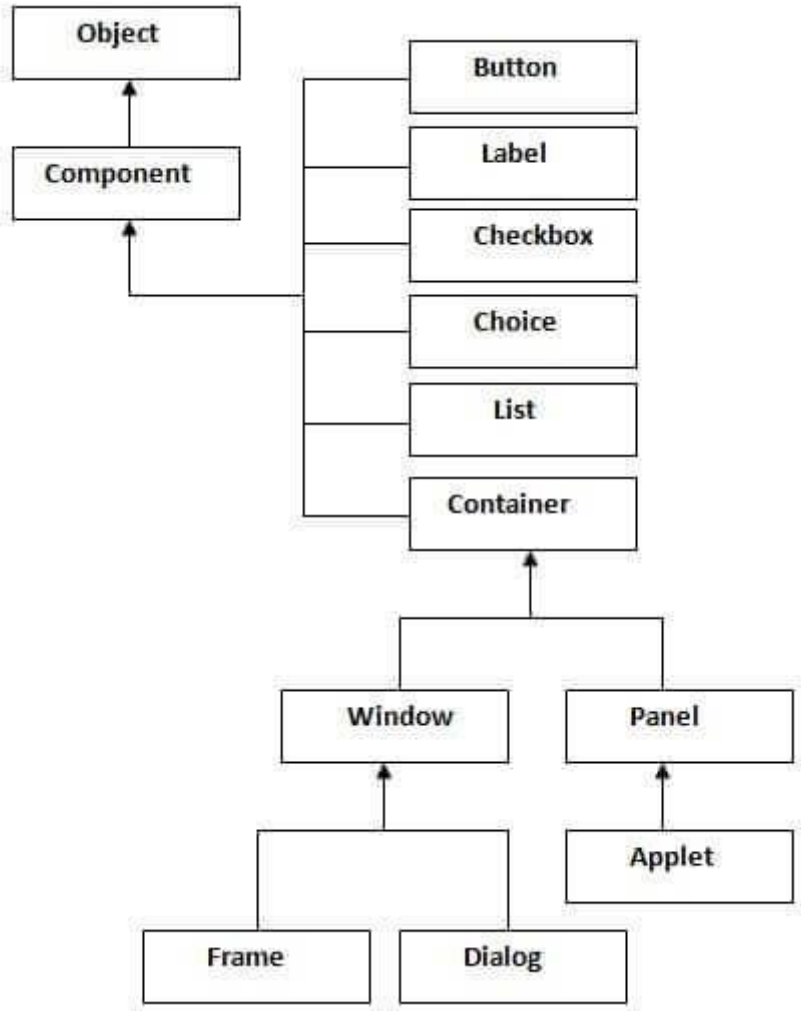
1. Java AWT Hierarchy

Java AWT (Abstract Window Toolkit) is an API to develop GUI or window-based applications in java.
Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavyweight i.e. its components are using the resources of OS.
The java.awt package provides classes for AWT api such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.

The hierarchy of Java AWT classes are given below.



- Container

The Container is a component in AWT that can contain another components like buttons, textfields, labels etc. The classes that extends Container class are known as container such as Frame, Dialog and Panel.

- Window

The window is the container that have no borders and menu bars. You must use frame, dialog or another window for creating a window.

- Panel

The Panel is the container that doesn't contain title bar and menu bars. It can have other components like button, textfield etc.

- Frame

The Frame is the container that contain title bar and can have menu bars. It can have other components like button, textfield etc.

2. Java AWT Example

To create simple awt example, you need a frame. There are two ways to create a frame in AWT.

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

AWT Example by Inheritance

Let's see a simple example of AWT where we are inheriting Frame class. Here, we are showing Button component on the Frame.

```
import java.awt.*;
class First extends Frame{
First(){
Button b=new Button("click me");
b.setBounds(30,100,80,30);// setting button position
add(b);//adding button into frame
setSize(300,300);//frame size 300 width and 300 height
setLayout(null);//no layout manager
setVisible(true);//now frame will be visible, by default not visible
}
```

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

The setBounds(int xaxis, int yaxis, int width, int height) method is used in the above example that sets the position of the awt button.

3. AWT Controls

Every AWT controls inherits properties from Component class.

AWT Component Class

1. Program On Label

The object of Label class is a component for placing text in a container. It is used to display a single line of read only text. The text can be changed by an application but a user cannot edit it directly.

```
import java.awt.*;
class LabelExample{
public static void main(String args[]){
   Frame f= new Frame("Label Example");
   Label l1,l2;
   l1=new Label("First Label.");
   l1.setBounds(50,100, 100,30);
   l2=new Label("Second Label.");
   l2.setBounds(50,150, 100,30);
   f.add(l1); f.add(l2);
   f.setSize(400,400);
   f.setLayout(null);
   f.setVisible(true);
}
}
```

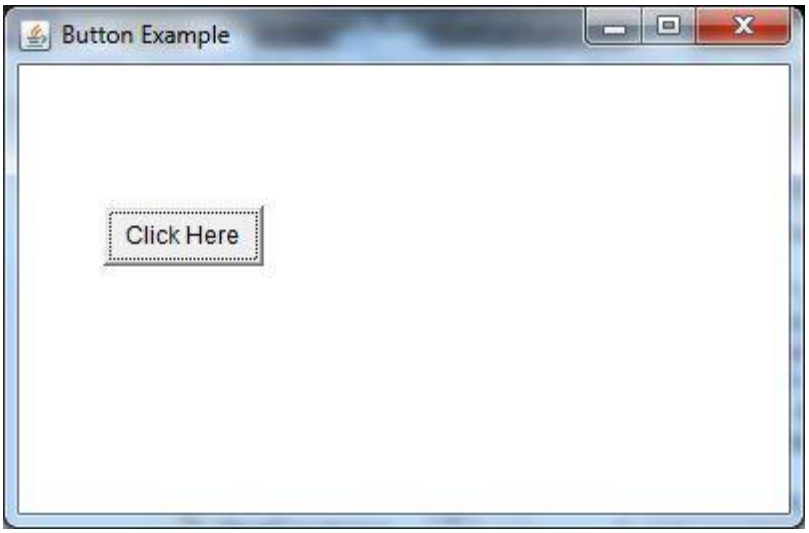
Output :

2. Program On Button

The button class is used to create a labeled button that has platform independent implementation. The application result in some action when the button is pushed.

```
import java.awt.*;
public class ButtonExample {
public static void main(String[] args) {
   Frame f=new Frame("Button Example");
   Button b=new Button("Click Here");
   b.setBounds(50,100,80,30);
   f.add(b);
   f.setSize(400,400);
   f.setLayout(null);
   f.setVisible(true);
}
}
```

Output :



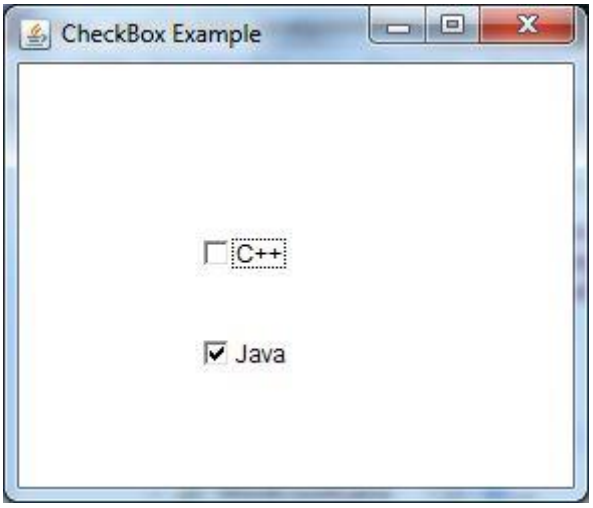
3. Program On Checkbox

The Checkbox 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".

```
import java.awt.*;
public class CheckboxExample
{
    CheckboxExample(){
     Frame f= new Frame("Checkbox Example");
```

```
        Checkbox checkbox1 = new Checkbox("C++");
        checkbox1.setBounds(100,100, 50,50);
        Checkbox checkbox2 = new Checkbox("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();
}
}
```

Output :



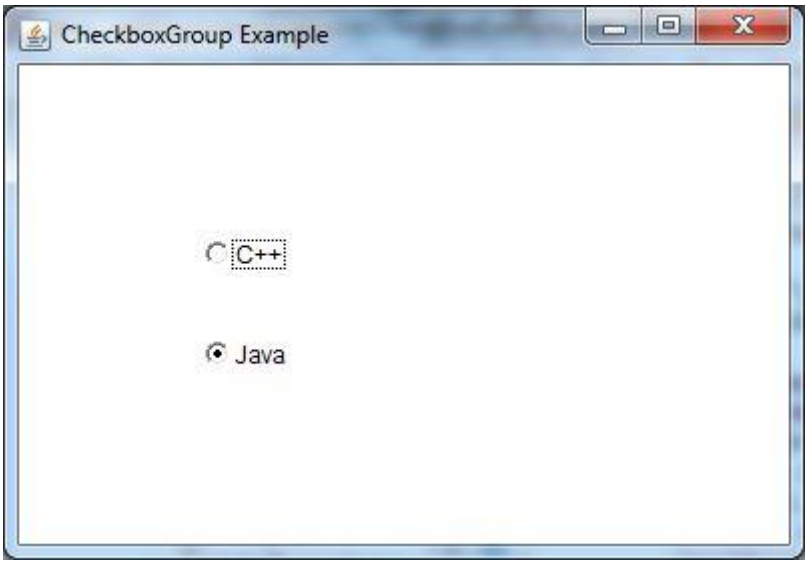
4. Program On Checkbox Group

The object of CheckboxGroup class is used to group together a set of Checkbox. At a time only one check box button is allowed to be in "on" state and remaining check box button in "off" state. It inherits the object class.

```
import java.awt.*;
public class CheckboxGroupExample
{
     CheckboxGroupExample(){
     Frame f= new Frame("CheckboxGroup Example");
      CheckboxGroup cbg = new CheckboxGroup();
      Checkbox checkBox1 = new Checkbox("C++", cbg, false);
      checkBox1.setBounds(100,100, 50,50);
```

```
        Checkbox checkBox2 = new Checkbox("Java", cbg, 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 CheckboxGroupExample();
}
}
```

Output :

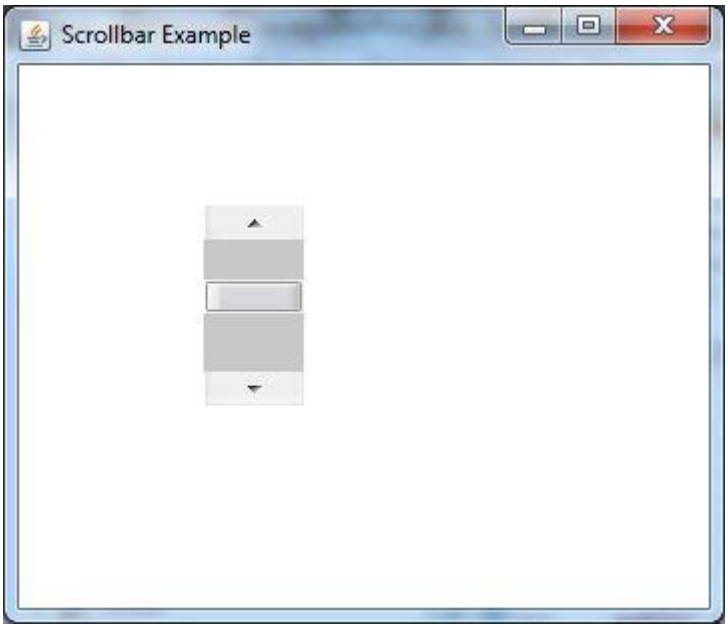


5. Program On Scrollbar

```
import java.awt.*;
class ScrollbarExample{
ScrollbarExample(){
        Frame f= new Frame("Scrollbar Example");
        Scrollbar s=new Scrollbar();
        s.setBounds(100,100, 50,100);
        f.add(s);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
}
```

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

Output :

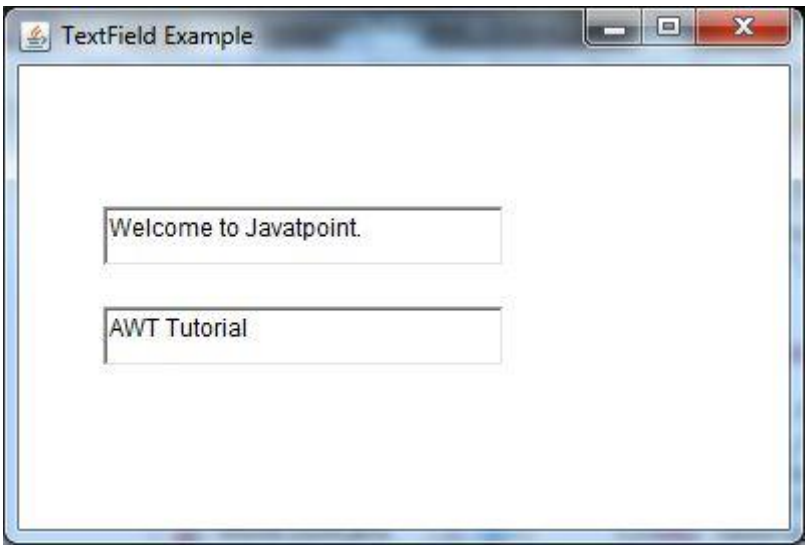


6. Program On TextField

The object of a TextField class is a text component that allows the editing of a single line text. It inherits TextComponent class.

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

Output :

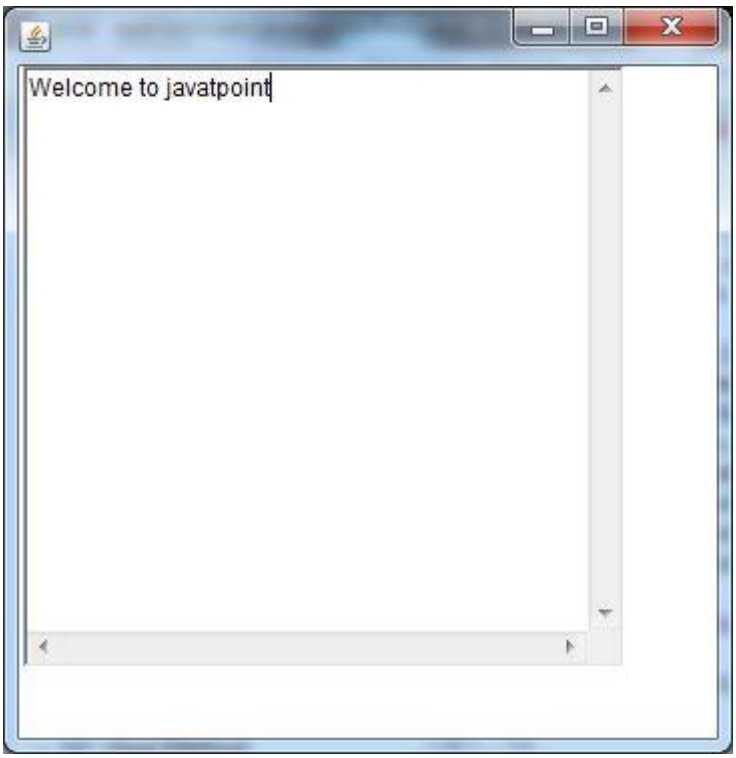


7. Program On TextArea

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

```
import java.awt.*;
public class TextAreaExample
{
    TextAreaExample(){
      Frame f= new Frame();
         TextArea area=new TextArea("Welcome to javatpoint");
      area.setBounds(10,30, 300,300);
      f.add(area);
      f.setSize(400,400);
      f.setLayout(null);
      f.setVisible(true);
    }
public static void main(String args[])
{
  new TextAreaExample();
}
}
```

Output :



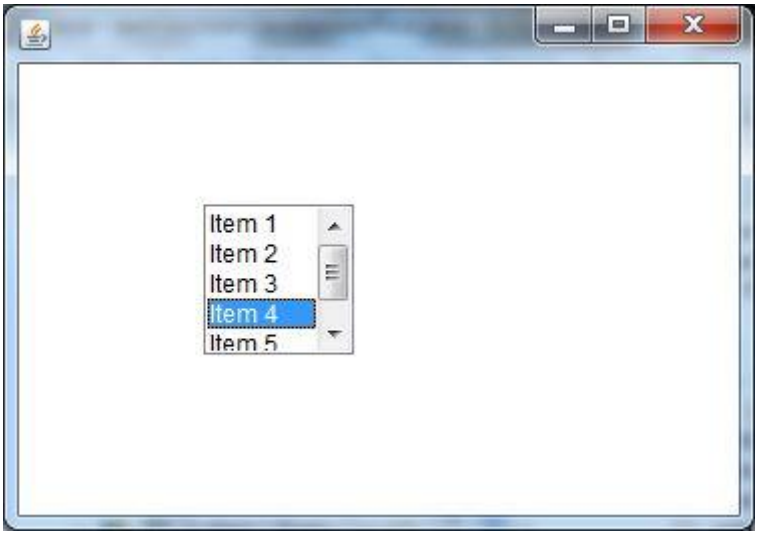
8. Program On List

The object of List class represents a list of text items. By the help of list, user can choose either one item or multiple items. It inherits Component class.

```
import java.awt.*;
public class ListExample
{
    ListExample(){
      Frame f= new Frame();
      List l1=new List(5);
      l1.setBounds(100,100, 75,75);
      l1.add("Item 1");
      l1.add("Item 2");
      l1.add("Item 3");
      l1.add("Item 4");
      l1.add("Item 5");
      f.add(l1);
      f.setSize(400,400);
      f.setLayout(null);
      f.setVisible(true);
    }
public static void main(String args[])
{
  new ListExample();
```

```
}
}
```

Output :



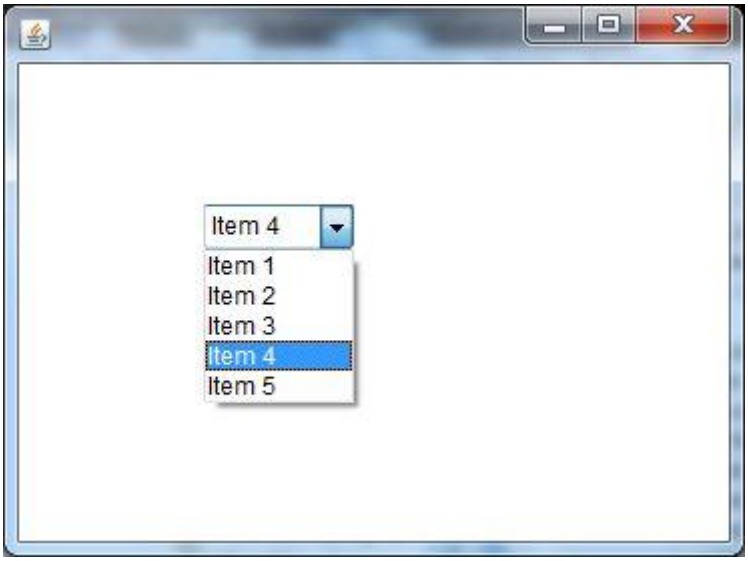
9. Program On Choice

The object of Choice class is used to show popup menu of choices. Choice selected by user is shown on the top of a menu. It inherits Component class.

```
import java.awt.*;
public class ChoiceExample
{
      ChoiceExample(){
      Frame f= new Frame();
      Choice c=new Choice();
      c.setBounds(100,100, 75,75);
      c.add("Item 1");
      c.add("Item 2");
      c.add("Item 3");
      c.add("Item 4");
      c.add("Item 5");
      f.add(c);
      f.setSize(400,400);
      f.setLayout(null);
      f.setVisible(true);
    }
public static void main(String args[])
{
  new ChoiceExample();
```

```
}
}
```

Output :



4. 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**

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 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();
  }
  }
```

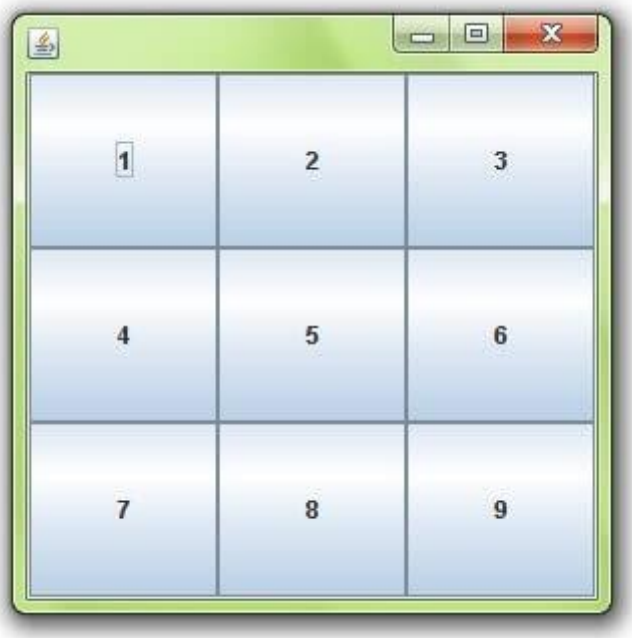
GridLayout

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

Constructors of GridLayout class

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 of GridLayout class



```
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);

   f.setLayout(new GridLayout(3,3));
   //setting grid layout of 3 rows and 3 columns

   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

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 of FlowLayout class


```
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();
}
}
```

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.

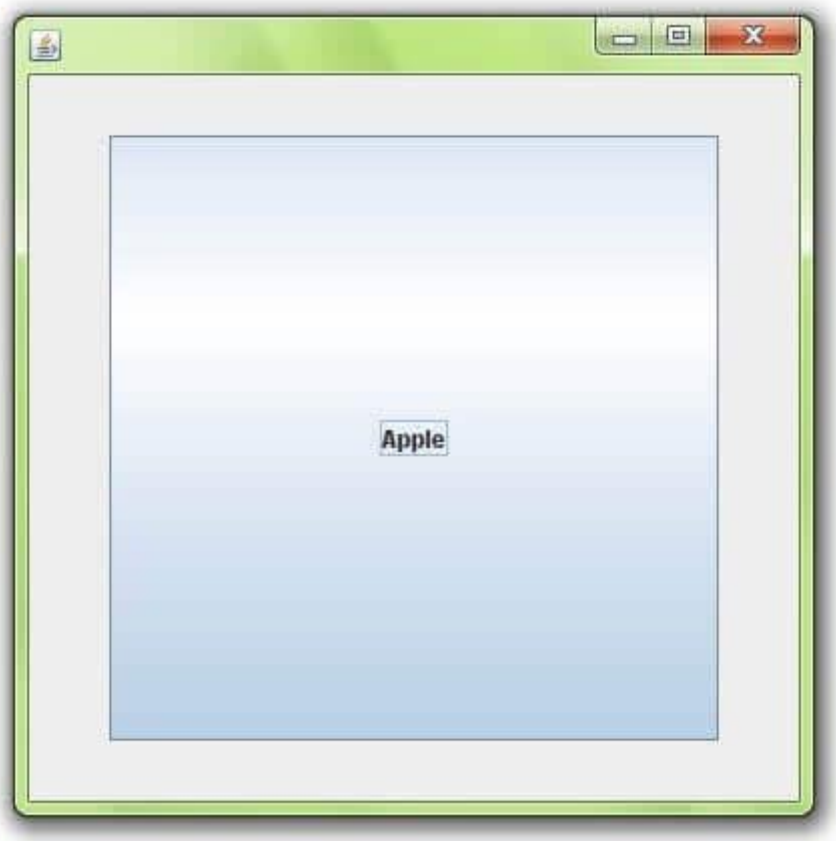
Constructors of CardLayout class

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 of CardLayout class




```
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);
    }
}
```

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.

```
public class GridBagLayoutDemo {
final static boolean shouldFill = true;
final static boolean shouldWeightX = true;
final static boolean RIGHT_TO_LEFT = false;

public static void addComponentsToPane(Container pane) {
if (RIGHT_TO_LEFT) {
pane.setComponentOrientation(ComponentOrientation.RIGHT_TO_LEFT);
}

JButton button;
pane.setLayout(new GridBagLayout());
GridBagConstraints c = new GridBagConstraints();
if (shouldFill) {
```

```
//natural height, maximum width
c.fill = GridBagConstraints.HORIZONTAL;
}

button = new JButton("Button 1");
if (shouldWeightX) {
c.weightx = 0.5;
}
c.fill = GridBagConstraints.HORIZONTAL;
c.gridx = 0;
c.gridy = 0;
pane.add(button, c);

button = new JButton("Button 2");
c.fill = GridBagConstraints.HORIZONTAL;
c.weightx = 0.5;
c.gridx = 1;
c.gridy = 0;
pane.add(button, c);

button = new JButton("Button 3");
c.fill = GridBagConstraints.HORIZONTAL;
c.weightx = 0.5;
c.gridx = 2;
c.gridy = 0;
pane.add(button, c);

button = new JButton("Long-Named Button 4");
c.fill = GridBagConstraints.HORIZONTAL;
c.ipady = 40;      //make this component tall
c.weightx = 0.0;
c.gridwidth = 3;
c.gridx = 0;
c.gridy = 1;
pane.add(button, c);

button = new JButton("5");
c.fill = GridBagConstraints.HORIZONTAL;
c.ipady = 0;       //reset to default
c.weighty = 1.0;   //request any extra vertical space
c.anchor = GridBagConstraints.PAGE_END; //bottom of space
c.insets = new Insets(10,0,0,0);  //top padding
c.gridx = 1;       //aligned with button 2
```

```
c.gridwidth = 2;   //2 columns wide
c.gridy = 2;       //third row
pane.add(button, c);
}


private static void createAndShowGUI() {
//Create and set up the window.
JFrame frame = new JFrame("GridBagLayoutDemo");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

//Set up the content pane.
addComponentsToPane(frame.getContentPane());

//Display the window.
frame.pack();
frame.setVisible(true);
}

public static void main(String[] args) {
javax.swing.SwingUtilities.invokeLater(new Runnable() {
public void run() {
createAndShowGUI();
}
});
}
}
```

Output :

FileDialog Class

FileDialog control represents a dialog window from which the user can select a file.

Class declaration

Following is the declaration for **java.awt.FileDialog** class:

```
public class FileDialog
  extends Dialog
```

Field

Following are the fields for **java.awt.Image** class:

- **static int LOAD** -- This constant value indicates that the purpose of the file dialog window is to locate a file from which to read.
- **static int SAVE** -- This constant value indicates that the purpose of the file dialog window is to locate a file to which to write.

Class constructors

S.N.	Constructor & Description
1	**FileDialog(Dialog parent)** Creates a file dialog for loading a file.
2	**FileDialog(Dialog parent, String title)** Creates a file dialog window with the specified title for loading a file.
3	**FileDialog(Dialog parent, String title, int mode)** Creates a file dialog window with the specified title for loading or saving a file.
4	**FileDialog(Frame parent)** Creates a file dialog for loading a file.
5	**FileDialog(Frame parent, String title)** Creates a file dialog window with the specified title for loading a file.
6	**FileDialog(Frame parent, String title, int mode)** Creates a file dialog window with the specified title for loading or saving a file.

Class methods

S.N.	Method & Description
1	**void addNotify()** Creates the file dialog's peer.

2	**String getDirectory()** Gets the directory of this file dialog.
3	**String getFile()** Gets the selected file of this file dialog.
4	**FilenameFilter getFilenameFilter()** Determines this file dialog's filename filter.
5	**int getMode()** Indicates whether this file dialog box is for loading from a file or for saving to a file.
6	**protected String paramString()** Returns a string representing the state of this FileDialog window.
7	**void setDirectory(String dir)** Sets the directory of this file dialog window to be the specified directory.
8	**void setFile(String file)** Sets the selected file for this file dialog window to be the specified file.
9	**void setFilenameFilter(FilenameFilter filter)** Sets the filename filter for this file dialog window to the specified filter.
10	**void setMode(int mode)** Sets the mode of the file dialog.

Methods inherited

This class inherits methods from the following classes:

- java.awt.Dialog
- java.awt.Window
- java.awt.Component
- java.lang.Object

FileDialog Example

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

public class AwtControlDemo {

  private Frame mainFrame;
  private Label headerLabel;
  private Label statusLabel;
  private Panel controlPanel;

  public AwtControlDemo(){
```

```
    prepareGUI();
  }

  public static void main(String[] args){
    AwtControlDemo  awtControlDemo = new AwtControlDemo();
    awtControlDemo.showFileDialogDemo();
  }

  private void prepareGUI(){
    mainFrame = new Frame("Java AWT Examples");
    mainFrame.setSize(400,400);
    mainFrame.setLayout(new GridLayout(3, 1));
    mainFrame.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent windowEvent){
        System.exit(0);
      }
    });
    headerLabel = new Label();
    headerLabel.setAlignment(Label.CENTER);
    statusLabel = new Label();
    statusLabel.setAlignment(Label.CENTER);
    statusLabel.setSize(350,100);

    controlPanel = new Panel();
    controlPanel.setLayout(new FlowLayout());

    mainFrame.add(headerLabel);
    mainFrame.add(controlPanel);
    mainFrame.add(statusLabel);
    mainFrame.setVisible(true);
  }

  private void showFileDialogDemo(){
    headerLabel.setText("Control in action: FileDialog");

    final FileDialog fileDialog = new FileDialog(mainFrame,"Select file");
    Button showFileDialogButton = new Button("Open File");
    showFileDialogButton.addActionListener(new ActionListener() {
      @Override
      public void actionPerformed(ActionEvent e) {
        fileDialog.setVisible(true);
        statusLabel.setText("File Selected :"
        + fileDialog.getDirectory() + fileDialog.getFile());
      }
    });
```

```
    controlPanel.add(showFileDialogButton);
    mainFrame.setVisible(true);
  }
}
```