JAVA AWT

By :

Subodh Sharma

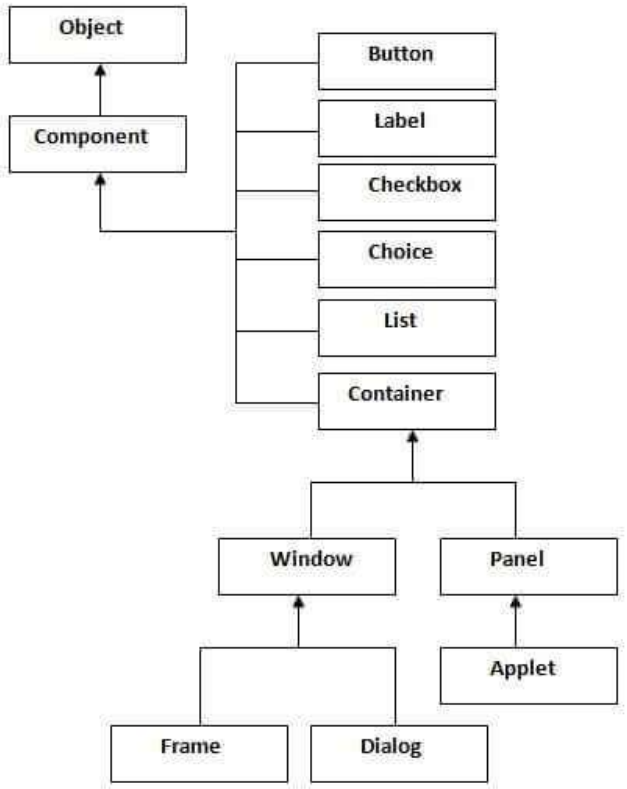
Java **application programming interface** (API) **is a list of all classes that are part of the** Java **development kit (JDK). It includes all** Java **packages, classes, and interfaces, along with their methods, fields, and constructors**

Java AWT

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.



Java AWT Hierarchy

The hierarchy of Java AWT classes are given.

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 (no contains ~~borders and menu bar~~)

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 (not contains ~~title bar, menu bar~~)

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

Frame (contains title bar, menu bar)

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

Useful Methods of Component class

Method	Description
public void add(Component c)	inserts a component on this component.
public void setSize(int width,int height)	sets the size (width and height) of the component.
public void setLayout(LayoutManager m)	defines the layout manager for the component.
public void setVisible(boolean status)	changes the visibility of the component, by default false.

Frame

Frame is a component that works as the main top-level window of the GUI application. It is created using the Frame class. For any GUI application, the first step is to create a frame. There are two methods to create a frame: by extending the Frame class or by creating an object of Frame class.

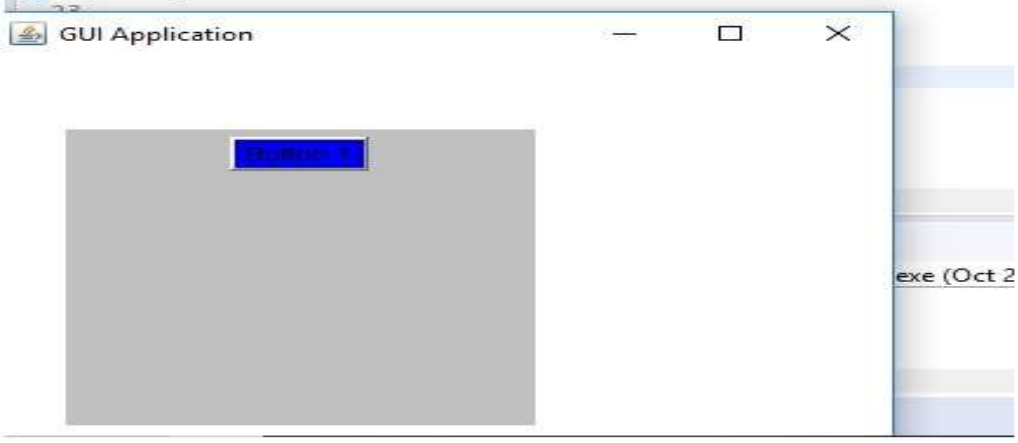
According to the given program (Figure), f is a Frame object. Other GUI components are added to it. Finally, the frame is displayed. The frame is a resizable and a movable window. It has the title bar. The default visibility of a Frame is hidden. The programmer has to make it visible by using setVisible method and providing the value “true” to it.

Panel

Panel is a component that allows placing multiple components on it. It is created using the Panel class. This class inherits the Container class. Refer the below program.

PanelTest.java

```
1  package ex1;
2  import java.awt.*;
3
4  public class PanelTest {
5
6      public static void main(String[] args) {
7
8              Frame f= new Frame("GUI Application");
9              Panel panel=new Panel();
10             panel.setBounds(40,80,200,200);
11             panel.setBackground(Color.LIGHT_GRAY);
12
13             Button b1=new Button("Button 1");
14             b1.setBounds(50,100,80,30);
15             b1.setBackground(Color.blue);
16
17             panel.add(b1);
18             f.add(panel);
19             f.setSize(400,400);
20             f.setLayout(null);
21             f.setVisible(true);
22     }
```



Difference Between Panel and Frame in Java

The main difference **between Panel and Frame in Java is that the** Panel is an internal region to a frame or another panel that helps to group multiple components together while a Frame is a resizable, movable independent window with a title bar which contains all other components.

Java AWT | Color Class

The Color class is a part of Java Abstract Window Toolkit(AWT) package. The Color class creates color by using the given RGBA values where **RGBA** stands for RED, GREEN, BLUE, ALPHA or using **HSB** value where HSB stands for HUE, SATURATION, BRIcomponents. The value for individual components RGBA ranges from 0 to 255 or 0.0 to 0.1. The value of alpha determines the opacity of the color, where 0 or 0.0 stands fully transparent and 255 or 1.0 stands opaque.

Constructors of Color Class

1. **Color(ColorSpace c, float[] co, float a)** : Creates a color in the specified ColorSpace with the color components specified in the float array and the specified alpha.
2. **Color(float r, float g, float b)** : creates a opaque color with specified RGB components(values are in range 0.0 – 0.1)
3. **Color(float r, float g, float b, float a)** : creates a color with specified RGBA components(values are in range 0.0 – 0.1)
4. **Color(int rgb)**: Creates an opaque RGB color with the specified combined RGB value consisting of the red component in bits 16-23, the green component in bits 8 – 15, and the blue component in bits 0-7.
5. **Color(int rgba, boolean b)**: Creates an sRGB color with the specified combined RGBA value consisting of the alpha component in bits 24-31, the red component in bits 16 – 23, the green component in bits 8 – 15, and the blue component in bits 0 – 7.
6. **Color(int r, int g, int b)** : Creates a opaque color with specified RGB components(values are in range 0 – 255)
7. **Color(int r, int g, int b, int a)** : Creates a color with specified RGBA components(values are in range 0 – 255)

Commonly Used Methods In Color Class

brighter()	creates a new Color that is a brighter version of this Color.
darker()	creates a new Color that is a darker version of this Color.
equals(Object obj)	determines whether another Color object is equal to this Color.
getAlpha()	returns the alpha component in the range 0-255.
getBlue()	returns the blue component in the range 0-255
getGreen()	returns the green component in the range 0-255 in the default sRGB space.
getRed()	returns the red component in the range 0-255 in the default sRGB space.
getRGB()	Returns the RGB value representing the color in the default sRGB ColorModel.
getHSBColor(float h, float s, float b)	Creates a Color object based on the specified

	values for the HSB color model.
getTransparency()	returns the transparency mode for this Color.
hashCode()	computes the hash code for this Color
HSBtoRGB(float h, float s, float b)	Converts the HSB value to RGB value
RGBtoHSB(int r, int g, int b, float[] hsbvals)	converts the RGB value to HSB value

Java AWT 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.
Label defines the following constructors

Label()	: blank label
Label(String str)	: lable with given string (Left align)
Label(String str, int how)	: lable with string, where how must be on of the three values Lable.LEFT, Label.Right, Label.CENTER.

Most common methods of Label :

void setText(String)
String getString()
void setAlignment(int how)
Int getAlignment();

Java AWT 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.
Button defines the following constructors :
Button()
Button(String str)

Most common methods of Buttons :
void setLabel(String str) --------→ where str is new label of button
String getLabel()

Event Handling with Button :
Interface : **ActionListener interface**
Methods of Interface : **actionPerformed()**
Event : **ActionEvent**

String getActionCommand() (of ActionEvent class) can get the label of Button in actionPerformed method of ActionListener interface.

Java AWT TextField

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

TextField()
TextField(int n) n-→ no of characters
TextField(String)
TextField(String, int no_of_chars)

Most common methods of Buttons :

String getText()
void setText(String);
String getSelectedText();
void select(int startIndex, int endIndex);
boolean isEditable();
void setEditable(boolean canEdit);

There may be times when we can disable the echoing the characters as they are typed. (password) :

void setEchoChar(char ch)
boolean echoCharIsSet();
char getEchoChar();

Java AWT 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".

Checkbox supports these constructors :
Checkbox();
Checkbox(String);
Checkbox(String str, boolean on);
Checkbox(String str, boolean on, CheckboxGroup cbGroup);
Checkbox(String str, boolean on, CheckboxGroup cbGroup);
Checkbox(String str, CheckboxGroup cbGroup, boolean on);

To retrieve the current state of check box,

boolean getState()
void setState(boolean on)
String getLabel()
void setLabel(String str);

Event Handling with Checkbox :
Interface : **ItemListener interface**
Methods of Interface : **itemStateChanged()**
Event : **ItemEvent**

Java AWT CheckboxGroup

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.

We can determine which checkbox in a group is currently selected by calling:

getSelectedCheckBox();
void setSelectedCheckbox(Checkbox which)

Example with event Handling :

```
import java.awt.*;
import java.awt.event.*;
class AwtCheck3 implements ItemListener
{
      Frame fr;
      Checkbox c1, c2, c3;
      AwtCheck3()
       {
     CheckboxGroup cbg = new CheckboxGroup();
       fr = new Frame("Checkbox example");
       c1 = new Checkbox("Red",cbg,true);
       c2 = new Checkbox("Blue",cbg, false);
       c3 = new Checkbox("Exit",cbg, false);
       c1.setBounds(100,30,100,30);
       c2.setBounds(100,100,100,30);
       c3.setBounds(100,170,100,30);
c1.addItemListener(this);
c2.addItemListener(this);
c3.addItemListener(this);
       fr.add(c1);
       fr.add(c2);
       fr.add(c3);
       fr.setSize(300,400);
       fr.setLayout(null);
       fr.setVisible(true);
       }
public void itemStateChanged(ItemEvent ie)
{
  String st1 = c1.getLabel();
  String st2 = c2.getLabel();
  String st3 = c3.getLabel();
     if(c1.getState())
         {   fr.setBackground(Color.red);
         }

     if(c2.getState())
         {   fr.setBackground(Color.blue);
         }

     if(c3.getState())
         {   fr.dispose();
         }
}

public static void main(String s[])
  {
  AwtCheck3 obj = new AwtCheck3();
  }}
```

Java AWT 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.

Constructor :

Choice ch = new Choice();

To add selection to the list, call add(), the general form is :

void add(String name);

To determine which item is currently selected :

String getSelectedItem();
int getSelectedIndex();

To obtain the number of items in the list :

int getItemCount()
void select(int index)
void select(String name)

Give an index, you can obtain the name associated with the item at that index by :

String getItem(int index);

```
import java.awt.*;
class AwtChoice
{
       AwtChoice()
       {         Frame f= new Frame();
           Choice c=new Choice();
          c.setBounds(100,100, 75,75);
          c.add("Pratik");
          c.add("Anurag");
          c.add("Rahul");
          c.add("Sparshi");
          c.add("Pooja");
          f.add(c);
          f.setSize(400,400);
          f.setLayout(null);
          f.setVisible(true);
      }
public static void main(String args[])
{
 new AwtChoice();
}
}
```

Example of Choice with event Handling :

```
import java.awt.*;
import java.awt.event.*;
class AwtChoice2 implements ActionListener, ItemListener
{
Frame f;
Choice c;
Button bexit;
TextField tf;
        AwtChoice2()
        {
Font fo = new Font("Times Roman",Font.BOLD,35);
        f= new Frame();
        bexit = new Button("Exit");
        tf = new TextField();
        tf.setFont(fo);
        f.add(bexit);  f.add(tf);
        tf.setBounds(100,200,200,50);
        bexit.setBounds(100,270,75,40);
        c=new Choice();
        c.setBounds(100,100, 200,75);
        c.add("Pratik");  c.add("Anurag");   c.add("Rahul");
c.add("Sparshi");
        c.add("Pooja");
c.select("Rahul");
        f.add(c);
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
bexit.addActionListener(this);
c.addItemListener(this);
       }

public void actionPerformed(ActionEvent ae)
{     String str= ae.getActionCommand();
     if(str.equals("Exit"))
         f.dispose();
}
public void itemStateChanged(ItemEvent ie)
{   tf.setText(c.getSelectedItem());
}

public static void main(String args[])
{   AwtChoice2 obj=new AwtChoice2();
}
}
```

Java AWT Panel

The Panel is a simplest container class. It provides space in which an application can attach any other component. It inherits the Container class. It doesn't have title bar.

```
import java.awt.*;
public class AwtPannl {
    AwtPannl()
       {
       Frame f= new Frame("Panel Example");
       Panel panel1=new Panel();
       Panel panel2=new Panel();
       panel1.setBounds(40,80,100,100);
       panel1.setBackground(Color.gray);

       panel2.setBounds(100,100,100,100);
       panel2.setBackground(Color.yellow);

       Button b1=new Button("Button 1");
       b1.setBounds(50,100,80,30);
       b1.setBackground(Color.yellow);
       Button b2=new Button("Button 2");
       b2.setBounds(100,100,80,30);
       b2.setBackground(Color.green);
       panel1.add(b1); panel1.add(b2);

panel2.add(b1);
       f.add(panel1);
       f.add(panel2);

       f.setSize(400,400);
       f.setLayout(null);
       f.setVisible(true);
       }
       public static void main(String args[])
       {
       new AwtPannl();
       }
}
```

Java AWT Dialog

The Dialog 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 Window class.

Unlike Frame, it doesn't have maximize and minimize buttons.

Java AWT MenuItem and Menu

The object of MenuItem class adds a simple labeled menu item on menu. The items used in a menu must belong to the MenuItem or any of its subclass.

The object of Menu class is a pull down menu component which is displayed on the menu bar. It inherits the MenuItem class.

```
import java.awt.*;
import java.awt.event.*;
class AwtM implements ActionListener
{
Frame fr;
AwtM()
  {
   fr = new Frame("My Example");
   MenuBar mb = new MenuBar();

   Menu me1= new Menu("File");
   MenuItem i11= new MenuItem("New");
   MenuItem i12= new MenuItem("Open");
   MenuItem i13= new MenuItem("Exit");
   me1.add(i11); me1.add(i12); me1.add(i13);
mb.add(me1);

   Menu me2= new Menu("Edit");

   MenuItem i21= new MenuItem("Cut");
   Menu subme= new Menu("Copy");
   MenuItem i23= new MenuItem("Paste");
   MenuItem i31= new MenuItem("Copy1");
   MenuItem i32= new MenuItem("Copy2");

   me2.add(i21); me2.add(subme); me2.add(i23);
mb.add(me2);
subme.add(i31);
subme.add(i32);

fr.setMenuBar(mb);
fr.setSize(400,400);
fr.setLayout(null);
fr.setVisible(true);
i13.addActionListener(this);
```

```
i12.addActionListener(this);
i11.addActionListener(this);
  }

public void actionPerformed(ActionEvent ae)
{
   String str = ae.getActionCommand();
   if(str.equals("Exit"))
     fr.dispose();
  if(str.equals("Open"))
     fr.setBackground(Color.yellow);
  if(str.equals("New"))
     fr.setBackground(Color.white);
}

public static void main(String s[])
  {
 AwtM ob= new AwtM();
  }
}
```

Java LayoutManagers :

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. Layout manager is set by the setLayout() method. If no call to setLayout() is made, then the default layout manger is used.

The setLayout() method has the following General Form :

void setLayout(LayoutManager layoutobj)

There are following classes that represents the layout managers in awt:

1. java.awt.BorderLayout
2. java.awt.FlowLayout
3. java.awt.GridLayout
4. java.awt.CardLayout
5. java.awt.GridBagLayout

Java FlowLayout

This is the default layout manager. It is a simple layout style, which is similar to how words flow in a text editor. The constructor for FlowLayout :

FlowLayout()	default layut which centers components and leaves 5 pixels of space between each component
FlowLayout(int how)	how ---- each line is aligned. Value of how : FlowLayout.LEFT FlowLayout.RIGHT FlowLayout.CENTER
FlowLayout(int how, int horz, int vert)	Here horz and vert are space left between components.

Example :

```
import java.awt.*;
import java.awt.event.*;
class AwtLFlow  implements ActionListener
{   Frame f;
    AwtLFlow()
    {
     f = new Frame("example");
    Button be= new Button("Click");
    Button b1= new Button("1");
    Button b2= new Button("2");
    Button b3= new Button("3");
    Button b4= new Button("4");

  f.setLayout(new FlowLayout(FlowLayout.CENTER,100,50));
  f.add(b1,FlowLayout.LEFT);
  f.add(b2,FlowLayout.LEFT);
  f.add(be,FlowLayout.CENTER);
  f.add(b3,FlowLayout.LEFT);
  f.add(b4,FlowLayout.LEFT);

  f.setSize(500,500);
```

```
   f.setVisible(true);
be.addActionListener(this);
  }
public void actionPerformed(ActionEvent ae)
{
   String cmd = ae.getActionCommand();
    if(cmd.equals("Click"))
      f.dispose();
}
public static void main(String s[])
  {
   AwtLFlow ob = new AwtLFlow();
  }
}
//--------------------------------------------------------------------------------------
```

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 (four narrow, fixed with components at the edges and one large are in the center).

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**

When ading components, we use the above constants as :

void add (Component Obj, Object region);

Constructors of BorderLayout class:

- **BorderLayout( ):** creates a border layout but with no gaps between the components.
- **BorderLayout(int horz, int vert) :** The horizontal and vertical space between components are defined in horz and vert arguments respectively.

Example :

```
import java.awt.*;
import java.awt.event.*;
class AwtLBorder  implements ActionListener
{
Frame f;
 AwtLBorder()
 {   f = new Frame("example");
     Button be= new Button("Click");
         Button b1= new Button("1");
         Button b2= new Button("2");
         Button b3= new Button("3");
         Button b4= new Button("4");

  f.setLayout(new BorderLayout(150,150));
         f.add(b1,BorderLayout.EAST);
         f.add(b2,BorderLayout.WEST);
         f.add(b3,BorderLayout.NORTH);
         f.add(b4,BorderLayout.SOUTH);
         f.add(be,BorderLayout.CENTER);
   f.setSize(500,500);    f.setVisible(true);
   be.addActionListener(this);
 }
public void actionPerformed(ActionEvent ae)
{   String cmd = ae.getActionCommand();
     if(cmd.equals("Click"))
      f.dispose();
}
public static void main(String s[])
 {  AwtLBorder ob = new AwtLBorder();
 }
```

Java GridLayout :

The GridLayout arrange components in a two dimensional grid. When we instantiate a GridLayout, we define the number of rows and columns. The constructors supported :

GridLayout()	Creates a single column grid layout
GridLayout(int numRow, int numCol)	Creates two dim grid with specified row and column
GridLayout(int numRow, int numCol, int horz, int vert)	Creates two dim grid with specified row & column and also specify horz and vertical space between componets.

Example :

```
import java.awt.*;
import java.awt.event.*;
class AwtLGrid  implements ActionListener
{    Frame f;
     AwtLGrid()   {
   f = new Frame("example");
   Button be= new Button("Click");
   Button b1= new Button("1");
   Button b2= new Button("2");
   Button b3= new Button("3");
   Button b4= new Button("4");
   Button b5= new Button("5");

   f.setLayout(new GridLayout(3,2));
   f.add(b1);    f.add(b2);    f.add(b3);    f.add(b4);    f.add(b5);
   f.add(be);

   f.setSize(500,500);
   f.setVisible(true);
be.addActionListener(this);
  }
public void actionPerformed(ActionEvent ae)
{
   String cmd = ae.getActionCommand();
    if(cmd.equals("Click"))
       f.dispose();
}
public static void main(String s[])
  {
   AwtLGrid ob = new AwtLGrid();
  }
```

Java CardLayout :

The CardLayout class is unique amount the other layout managers in that it stores several different layouts. Each layout can be thought of as being on a separate index card in a desk that can be shuffled so that any card is on top at a given time.
This can be useful for user interfaces with optional components that can be dynamically enabled and disabled upon user input.

The CardLayout privides two constructors :

CardLayout()
CardLayout(int horz, int vert)

Commonly Used Methods:

- **addLayoutComponent(Component cm, Object cn):** Used to add the specified component to this card layout's internal table of names.
- **toString():** Returns a string representation of the state of this card layout.
- **removeLayoutComponent(Component cm):** Used to remove the specified component from the layout.

Below programs illustrate the CardLayout class:

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

class AwtLCard implements ActionListener
{
    CardLayout card;
    Button b1, b2, b3;
    Frame fr;
  AwtLCard()
  {  fr = new Frame("Card");
     card = new CardLayout(100,100);
     fr.setLayout(card);
     b1 = new Button("Button1");
     b2 = new Button("Button2");
     b3 = new Button("Button3");
     b1.addActionListener(this);
     b2.addActionListener(this);
     b3.addActionListener(this);
     fr.add(b1);
     fr.add(b2);
     fr.add(b3);
fr.setSize(400,400);
fr.setVisible(true);
  }
 public void actionPerformed(ActionEvent e)
  {
     card.next(fr);
         String cmd= e.getActionCommand();
     if(cmd.equals("Button1"))
         fr.setBackground(Color.red);
```

```
        if(cmd.equals("Button2"))
            fr.setBackground(Color.green);
        if(cmd.equals("Button3"))
            fr.setBackground(Color.black);
    }
     public static void main(String[] args)
    {
        AwtLCard obj = new AwtLCard();
    }
}
```

Java Swing :

Java Swing tutorial 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.

Difference between AWT and Swing

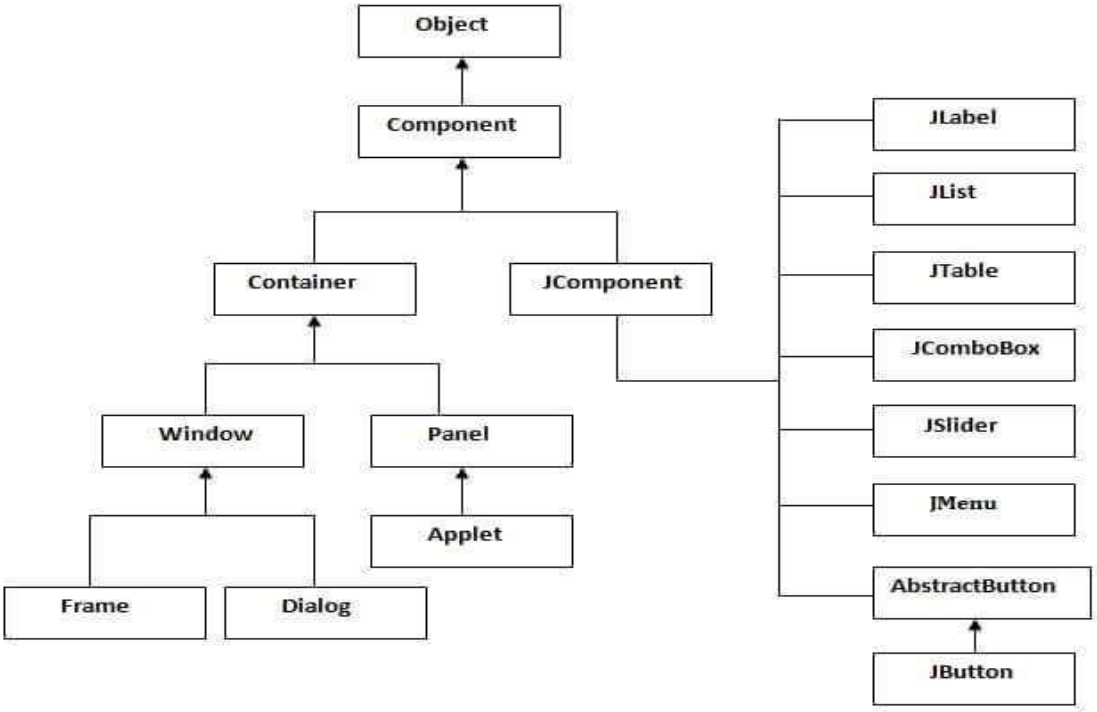
	Java AWT	Java Swing
1)	AWT components are **platform-dependent**.	Java swing components are **platform-independent**.
2)	AWT components are **heavyweight**.	Swing components are **lightweight**.
3)	AWT **doesn't support pluggable look and feel**.	Swing **supports pluggable look and feel**.
4)	AWT provides **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**.

Commonly used Methods of Component class

The methods of Component class are widely used in java swing that are given below.

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.

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.

Simple Java Swing Example **(By creating the object of Frame class)**

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

- Simple Java Swing Example (By extending Frame class (inheritance)

```
import javax.swing.*;
public class Simple2 extends JFrame          //inheriting JFrame
 {
JFrame f;
Simple2(){
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) {
new Simple2();
}
}
```

Swing JComponents :

- JButton class
- JRadioButton class
- JTextArea class
- JComboBox class
- JTable class
- JColorChooser class
- JProgressBar class
- JSlider class
- OpenDialog Box

Java JButton

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

JButton class declaration

Let's see the declaration for javax.swing.JButton class.

1. **public class** JButton **extends** AbstractButton **implements** Accessible

Commonly used Constructors:

Constructor	Description
JButton()	It creates a button with no text and icon.
JButton(String s)	It creates a button with the specified text.
JButton(Icon i)	It creates a button with the specified icon object.

Commonly used Methods of AbstractButton class:

Methods	Description
void setText(String s)	It is used to set specified text on button
String getText()	It is used to return the text of the button.
void setEnabled(boolean b)	It is used to enable or disable the button.
void setIcon(Icon b)	It is used to set the specified Icon on the button.
Icon getIcon()	It is used to get the Icon of the button.
void setMnemonic(int a)	It is used to set the mnemonic on the button.
void addActionListener(ActionListener a)	It is used to add the action listener to this object.

Java JButton Example with ActionListener

Java JButton Example

```
import javax.swing.*;
import java.awt.event.*;
public class Swin implements ActionListener
{
 JFrame f;
 JButton b;
  Swin()
   {
    f=new JFrame("Button Example");
    b=new JButton("Click Here");
   b.setBounds(50,100,95,30);
   f.add(b);
   f.setSize(400,400);
   f.setLayout(null);
   f.setVisible(true);
   b.addActionListener(this);
  }
public void actionPerformed(ActionEvent ae)
   {
  System.out.println("Hello");
   }

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