

## 06. Applets, AWT, Swing and Event Handling

### Applets:

- An applet is a special kind of Java program that runs in a Java enabled browser. This is the first Java program that can run over the network using the browser. Applet is typically embedded inside a web page and runs in the browser.
- In other words, we can say that Applets are small Java applications that can be accessed on an Internet server, transported over Internet, and can be automatically installed and run as apart of a web document.
- After a user receives an applet, the applet can produce a graphical user interface. It has limited access to resources so that it can run complex computations without introducing the risk of viruses or breaching data integrity.
- To create an applet, a class must extends from **Applet** class which was found under **java.applet** package.
- An Applet class does not have any main() method. But it is viewed using JVM by appletviewer.
- The JVM can use either a plug-in of the Web browser or a separate runtime environment to run an applet application.
- JVM creates an instance of the applet class and invokes `init()` method to initialize an Applet.

### Applet class:

- Applet class provides all necessary support for applet execution, such as initializing and destroying of applet. It also provide methods that load and display images and methods that load and play audio clips.

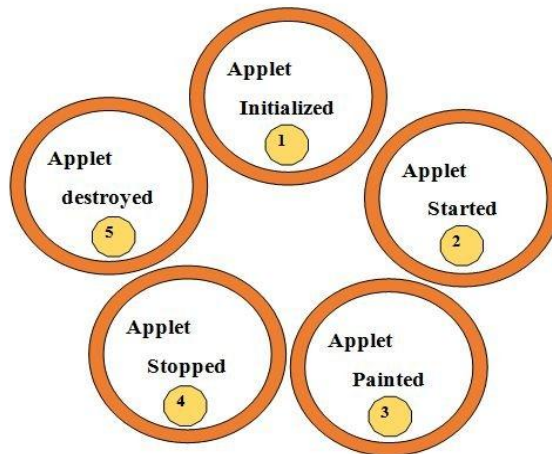
### Lifecycle of Java Applet:

- Applet lifecycle describes all states of applet from initialization to destroy.

Following are the stages shows Applet life cycle:

- 1) Applet is initialized.
- 2) Applet is started
- 3) Applet is painted.
- 4) Applet is stopped.
- 5) Applet is destroyed.

Following diagram shows Applet life cycle:



Most applets override these four methods. These four methods forms Applet lifecycle.

- 1) **init()** : `init()` is the first method to be called. This is where variable or componets of applet are initialized. This method is called only once during the runtime of applet.
- 2) **start()** : `start()` method is called after `init()`. This method is called to restart an applet after it has been stopped.
- 3) **paint(Graphics g)**: `paint()` is used to paint the Applet. It provides Graphics class object that can be used for drawing line, circle, oval, rectangle, arc etc. Graphics componets.
- 4) **stop()** : `stop()` method is used to suspend thread that does not need to run the applet.
- 5) **destroy()** : `destroy()` method is called when your applet needs to be removed completely from memory.

### Note:

- To use above **init()**, **start()**, **stop()** and **destroy()** methods, we have to import package **java.applet.Applet**;
- To use **paint()** method, we have to import package **java.awt.\***;
- The `stop()` method is always called before `destroy()` method.

### A simple Java applet Program:

```
import java.awt.*;
import java.applet.*;
public class Simple extends Applet
{
    String msg;
    public void init()
    {
        // set the foreground and background colors.
        setBackground(Color.cyan);
        setForeground(Color.red);
        msg = "Inside init( ) method";
    }
    // Initialize the string to be displayed.
    public void start()
    {
        msg =msg+ " Inside start( ) method";
    }
    // Display msg in applet window.
    public void paint(Graphics g)
    {
        msg =msg+" Inside paint( ).";
        g.drawString(msg, 10, 30);
    }
}
/*
<applet code="Simple.class" width=300 height=50>
</applet>
*/
```

### How to run an Applet?

There are two ways to run an applet

1. By creating separate html file.
2. By embedding html code to applet java source file.

Let's see ways in details-

#### 1. By creating separate html file:

Follow the steps:

- 1) Create new java source program (.java file) that contains applet source code. As follow:

```
import java.awt.*;
import java.applet.*;
public class First extends Applet
{
    public void init( )
    {
        setBackground(Color.white);
    }
    public void paint(Graphics g)
    {
        g.drawString("My First Applet", 20, 20);
    }
}
```

- 2) Save above code with First.java and then compile it as follow:

**d:\2201>javac First.java**

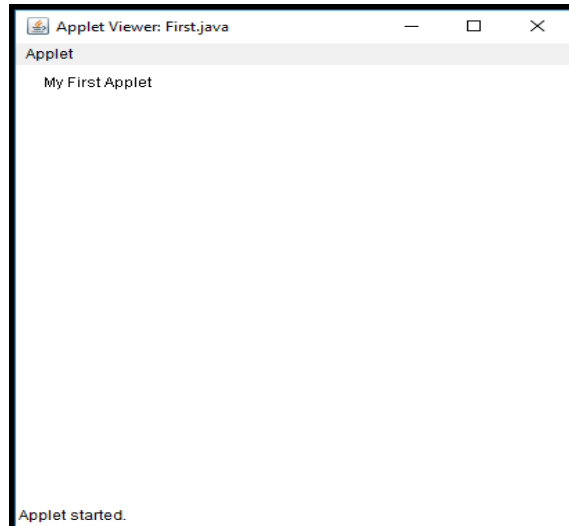
- 3) Create new separate html file that uses created byte code (First.class) As follow:

```
<html>
<body>
<applet code= "First.class" height=400 width=400></applet>
</body>
</html>
```

- 4) Save above html code with **.html extension**. Let I store it with name **First.html**. And then **interpret this html file with JDK tool appletviewer** as follow:

**d:\2201>appletviewer First.html**

After executing above applet we will get output as-



## 2. By embedding html code to applet java source file.

Step 1) Create new java source program (.java file) that contains applet source code and also embed html applet code using multiline comments as follow:

```
import java.awt.*;
import java.applet.*;
public class Second extends Applet
{
    public void init( )
    {
        setBackground(Color.white);
    }
    public void paint(Graphics g)
    {
        g.drawString("My Second Applet", 20, 20);
    }
}
/*
<applet code= "Second.class" height=400 width=400></applet>
*/
```

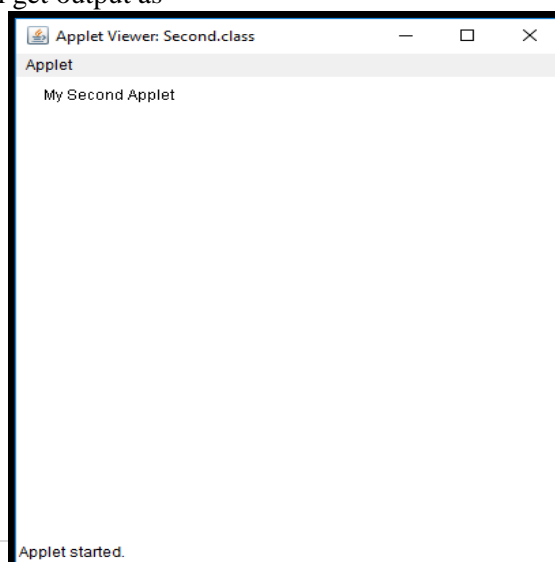
Step 2) Save above code with Second.java and then compile it as follow:

**d:\2201>javac Second.java**

Step 3) Then **interpret this java file with JDK tool appletviewer** as follow:

**d:\2201>appletviewer Second.java**

After executing above applet we will get output as-



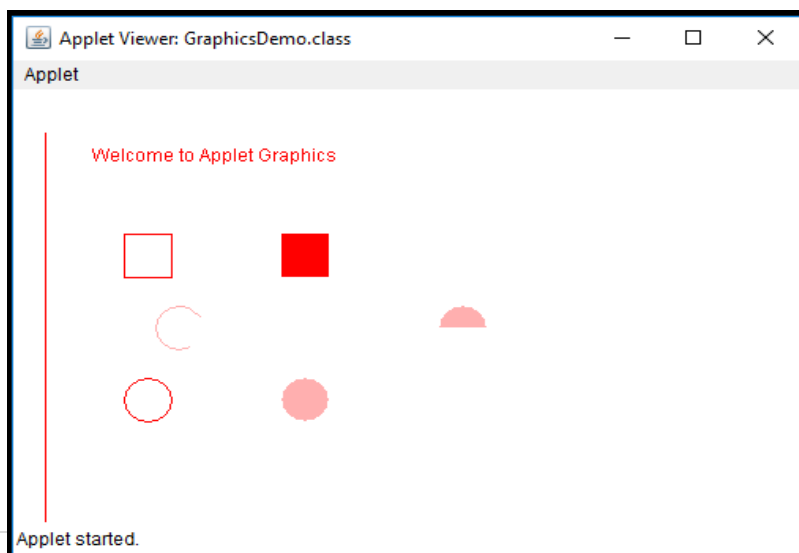
## Displaying Graphics in Applet

- **java.awt.Graphics** class provides many methods for graphics programming. Commonly used methods of Graphics class are discussed below-
  - 1) **drawString(String str, int x, int y)**: It is used to draw the specified string.
  - 2) **drawRect(int x, int y, int width, int height)**: It draws a rectangle with the specified width and height.
  - 3) **fillRect(int x, int y, int width, int height)**: It is used to fill rectangle with the default color Black and specified width and height.
  - 4) **drawOval(int x, int y, int width, int height)**: It is used to draw oval with the specified width and height.
  - 5) **fillOval(int x, int y, int width, int height)**: It is used to fill oval with the default color and specified width and height.
  - 6) **drawLine(int x1, int y1, int x2, int y2)**: It is used to draw line between the points(x1, y1) and (x2, y2).
  - 7) **drawImage(Image img, int x, int y, ImageObserver observer)**: It is used draw the specified image.
  - 8) **drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)**: It is used draw a circular or elliptical arc.
  - 9) **fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)**: It is used to fill a circular or elliptical arc.
  - 10) **setColor(Color c)**: It is used to set the graphics current color to the specified color.
  - 11) **setFont(Font font)**: It is used to set the graphics current font to the specified font.

Following program shows use of above mentioned graphics methods:

```
import java.applet.Applet;
import java.awt.*;
public class GraphicsDemo extends Applet
{
    public void init()
    {
        setBackground(Color.white);
    }
    public void paint(Graphics g)
    {
        g.setColor(Color.red);
        g.drawString("Welcome",50, 50);
        g.drawLine(20,30,20,300);
        g.drawRect(70,100,30,30);
        g.fillRect(170,100,30,30);
        g.drawOval(70,200,30,30);
        g.setColor(Color.pink);
        g.fillOval(170,200,30,30);
        g.drawArc(90,150,30,30,30,270);
        g.fillArc(270,150,30,30,0,180);
    }
}
/*<applet code="GraphicsDemo.class" width="300" height="300">
</applet>*/
```

**Output of above code:**



### //Display Smiley in Applet.

```
import java.awt.*;
import java.applet.*;
public class Smiley extends Applet
{
    Font f = new Font("Helvetica", Font.BOLD,20);
    public void init()
    {
        setBackground(Color.white);
        setFont(f);
    }

    public void paint(Graphics g)
    {
        g.drawString("Keep Smiling!!!", 50, 30);
        g.drawOval(60, 60, 200, 200);
        g.fillOval(90, 120, 50, 20);
        g.fillOval(190, 120, 50, 20);
        g.drawLine(165, 125, 165, 175);
        g.drawArc(110, 130, 95, 95, 0, -180);
    }
}
/*
<html>
<body>
<applet code = "Smiley.class" width= "500" height = "300"> </applet>
</body>
</html> */
```

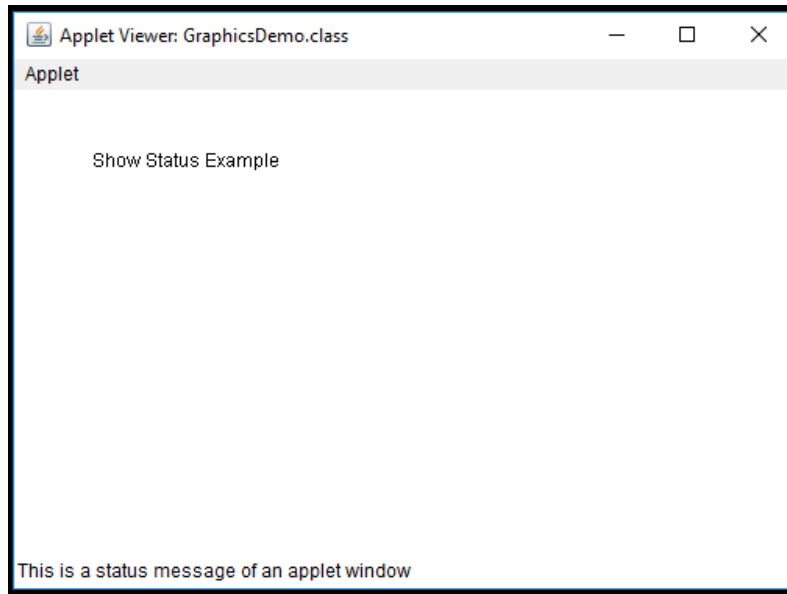
**Output of above code:**



### //Set Status Message in Applet Window Example

```
import java.applet.Applet;
import java.awt.Graphics;
public class StatusMessage extends Applet
{
    public void paint(Graphics g)
    {
        g.drawString("Show Status Example", 50, 50); //this will be displayed inside an applet
        showStatus("This is a status message of an applet window"); //this will be displayed in a status bar of an applet
    }
}
/*
<applet code= "StatusMessage.class" width=300 height=500></applet> */
```

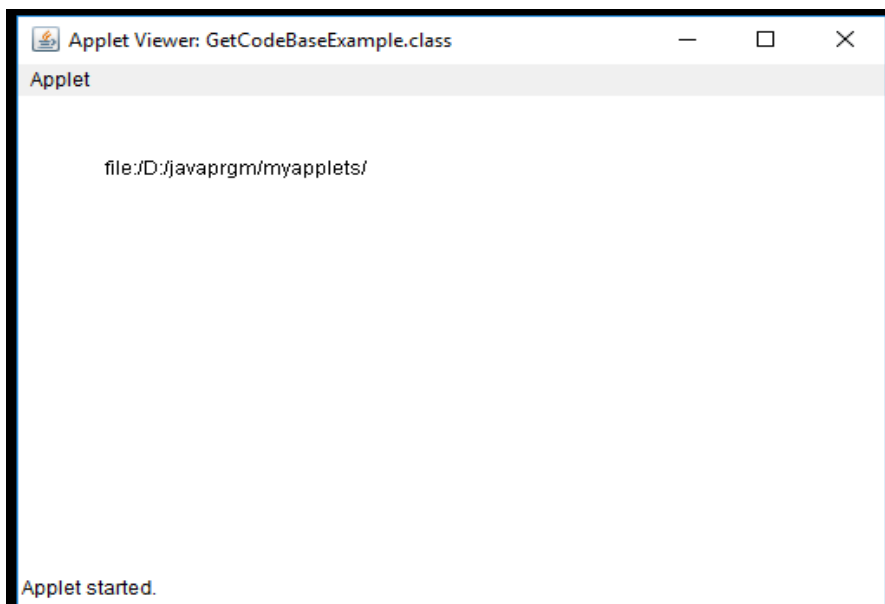
## OUTPUT:



### //Get Applet Directory URL (i.e path of applet source code) or Code Base Example

```
import java.applet.Applet;
import java.awt.Graphics;
import java.net.URL;
public class GetCodeBaseExample extends Applet
{
    public void paint(Graphics g)
    {
        URL appletDir = getCodeBase();
        g.drawString(appletDir.toString(), 50, 50);
    }
}
/*
<applet code="GetCodeBaseExample.class" width=200 height=200>
</applet>
*/
```

## OUTPUT:



## Displaying Image in Applet:

- Applet is mostly used in games and animation. For this purpose image is required to be displayed on applet.
- The java.awt.Graphics class provide a method drawImage() to display the image.
- Syntax of drawImage() method:

**drawImage(Image img, int x, int y, ImageObserver observer);**

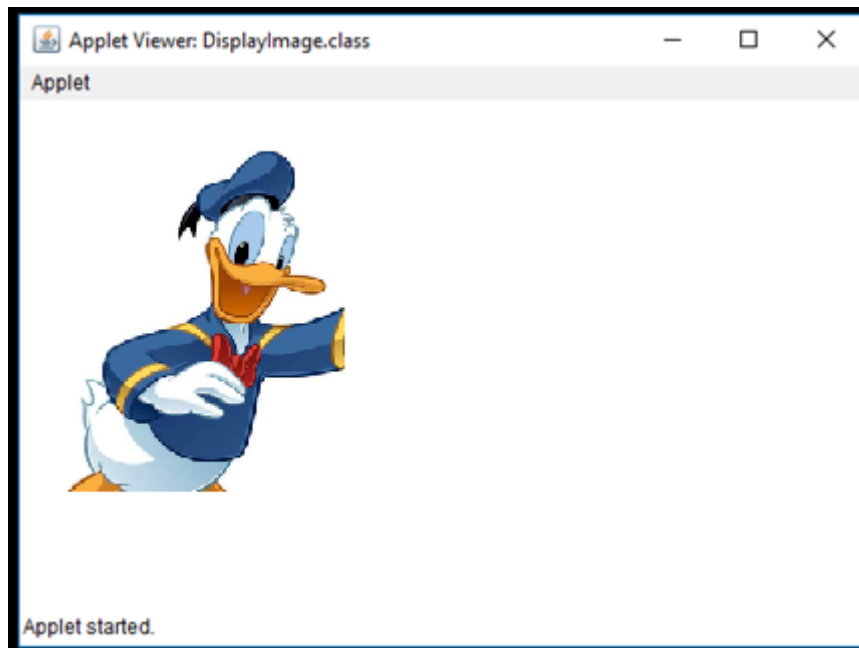
In above syntax-

- We get the object of Image by using getImage() method of Applet class that returns the object of Image.  
Syntax: **getImage(URL u, String imageName);**
- In above syntax we have to pass URL of image as first argument by calling getDocumentBase() method and second argument string which is image name.
- x, y is location of image where image will display
- The 4th is ImageObserver which is an interface. So current class reference would also be treated as ImageObserver which is **this**.

Following program shows displaying an image in Applet

```
import java.awt.*;
import java.applet.*;
public class DisplayImage extends Applet
{
    Image picture;
    public void init()
    {
        picture = getImage(getDocumentBase(),"duck.jpg");
    }
    public void paint(Graphics g)
    {
        g.drawImage(picture, 30,30, this);
    }
}
/*<applet code="DisplayImage.class" width="300" height="300">
</applet> */
```

**OUTPUT:**



## Animation in Applet:

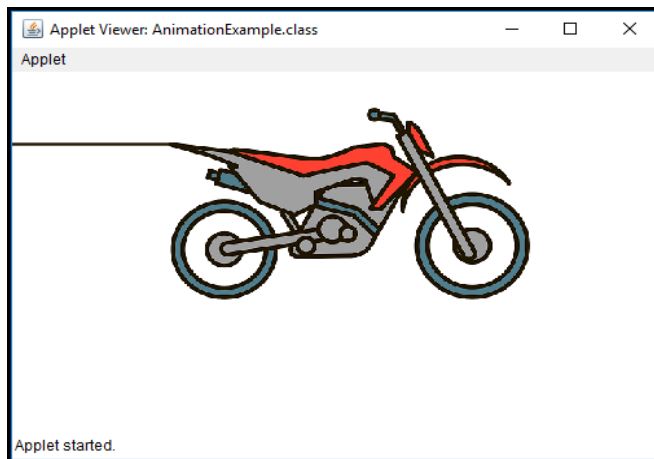
Applet is mostly used in games and animation. For this purpose image is required to be moved.

**Following program shows moving image in Applet (Animation using Applet)**

```
import java.awt.*;
import java.applet.*;
public class AnimationExample extends Applet
{
    Image picture;
    public void init()
    {
        picture =getImage(getDocumentBase(),"bike.png");
    }

    public void paint(Graphics g)
    {
        try
        {
            for(int i=10;i<500;i++)
            {
                g.drawImage(picture, i,30, this);
                Thread.sleep(10);
            }
        }
        catch(Exception e){}
    }
}
/*<applet code="AnimationExample.class" width="300" height="300">
</applet> */
```

**OUTPUT:**



## Java Event Handling (Event and Listener):

- An event can be defined as changing the state of an object or behavior by performing actions.
- Actions can be a button click, cursor movement, entering a character in Textbox, Clicking or dragging a mouse, keypress through keyboard or page scrolling, etc.
- For event handling in AWT, we have to import package **java.awt.event.\***;
- Java Event classes and Listener interfaces are shown in following table-

Event Classes	Listener Interfaces
ActionEvent	ActionListener
MouseEvent	MouseListener and MouseMotionListener
MouseWheelEvent	MouseWheelListener
KeyEvent	KeyListener



ItemEvent	ItemListener
TextEvent	TextListener
AdjustmentEvent	AdjustmentListener
WindowEvent	WindowListener
ComponentEvent	ComponentListener
ContainerEvent	ContainerListener
FocusEvent	FocusListener

## Steps to perform Event Handling:

Following steps are required to perform event handling:

**Step 1: Add the components on to the Applet or java GUI application.**

**Step 2: Register the component with the Listener.**

- For registering the component with the Listener, many classes provide the registration methods and these methods are –
- Following are the respective methods of components to register the component with listener:
- **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){ }`

## Step 3: Java Event Handling Code

We can put the event handling code into `actionPerformed(ActionEvent e)` method shown as follow:

```
public void actionPerformed(ActionEvent aa)
{
    if(aa.getSource()==b1)           // b1 is Button object.
        setBackground(Color.red);    // getSource( ) method returns object of clicked Button object
}
```

Ex-

**Following applet program shows Event handling demo that changes background color of applet by clicking on appropriate button:**

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;
public class apple extends Applet implements ActionListener
{
    Button    b1,b2,b3,b4,b5,b6,b7;
    public void init()
    {
        // creating objects for Buttons
        b1=new Button("RED");
        b2=new Button("Orange");
        b3=new Button("Yellow");
```

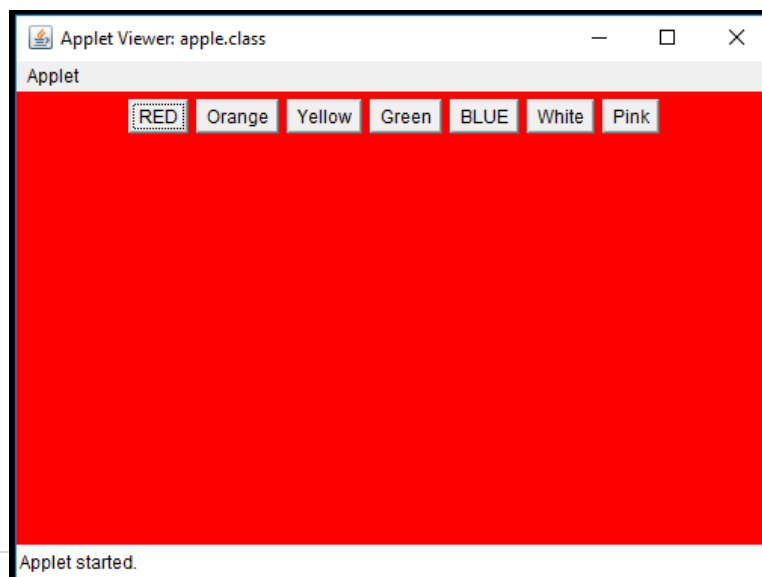
```

b4=new Button("Green");
b5=new Button("BLUE");
b6=new Button("White");
b7=new Button("Pink");
    //add buttons onto applet
add(b1);
add(b2);
add(b3);
add(b4);
add(b5);
add(b6);
add(b7);
    //register Listener for buttons
b1.addActionListener(this);
b2.addActionListener(this);
b3.addActionListener(this);
b4.addActionListener(this);
b5.addActionListener(this);
b6.addActionListener(this);
b7.addActionListener(this);
}

public void  actionPerformed(ActionEvent  aa)
{
    if(aa.getSource()==b1)
        setBackground(Color.red);
    else if(aa.getSource()==b2)
        setBackground(Color.orange);
    else if(aa.getSource()==b3)
        setBackground(Color.yellow);
    else if(aa.getSource()==b4)
        setBackground(Color.green);
    else if(aa.getSource()==b5)
        setBackground(Color.blue);
    else if(aa.getSource()==b6)
        setBackground(Color.white);
    else if(aa.getSource()==b7)
        setBackground(Color.pink);
}
}
/*
<applet      code      = "apple.class"          width  = "500"          height  = "300">
</applet>
*/

```

## OUTPUT:

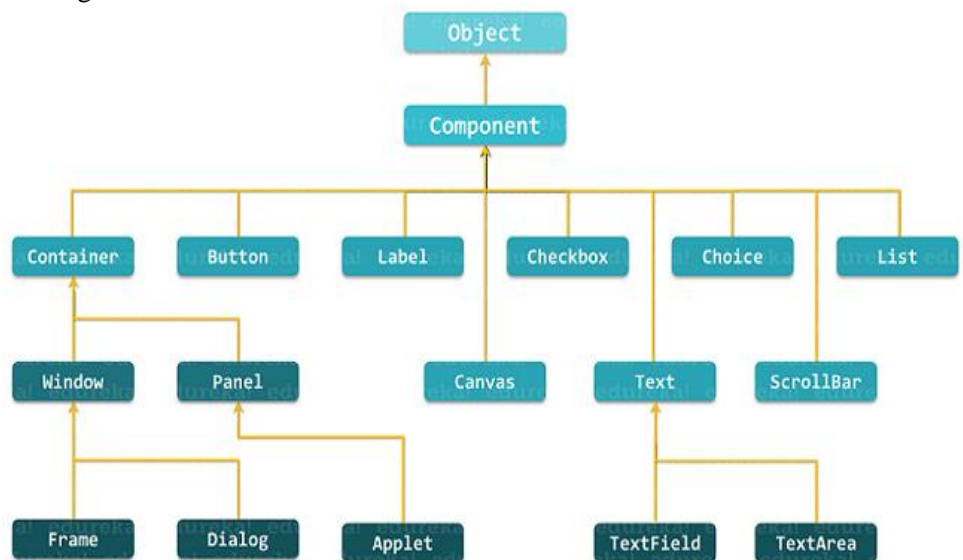


## Java AWT:

- 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.
- 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.
- 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 below.



- In the above diagram, Component is the superclass of all the GUI controls. It is an abstract class which encapsulates all the attributes of a visual component and represents an object with graphical representation. A component class instance is basically responsible for the look and feel of the current interface.

## 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.
- We 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.

## Useful Methods of Component class

Method	Description
public void add(Component c)	inserts a component on the Window,frame,Applet etc.
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.

## Creating Frame:

- To create simple awt example, we need a frame. A frame is basic component in AWT.
- The frame has to be created before any other component because all other components (button, textfield etc) can be displayed in frame.
- There are two ways to create a frame in AWT.
  - 1) By extending Frame class (inheritance)
  - 2) By creating the object of Frame class (association)

Let's see these ways in details-

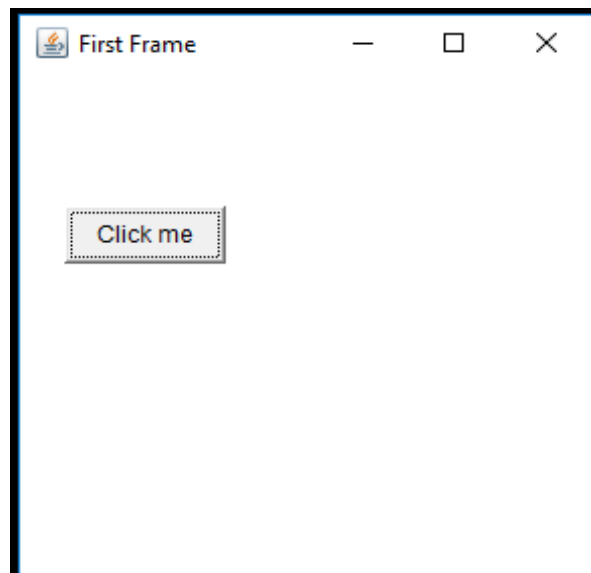
### 1) Creating Frame by extending Frame class (Using inheritance)

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 FrameDemo1 extends Frame
{
    public static void main(String args[])
    {
        FrameDemo1 f=new FrameDemo1();
        f.setTitle("First Frame"); //set frame title
        f.setSize(300,300); //frame size 300 width and 300 height
        f.setLayout(null); //no layout manager
        f.setVisible(true); //now frame will be visible, by default not visible
        Button b=new Button("Click me");
        b.setBounds(30,100,80,30); //setting button position
        f.add(b); //adding button on frame
    }
}
```

**OUTPUT:**



- In above example **setBounds (int x, int y, int width, int height)** method is used in to set the position of the awt button.

### 2) Creating Frame by object of Frame class (Association):

Let's see a simple example of AWT where we are creating object i.e. instance of Frame class.

Here, we are showing Button component on the Frame.

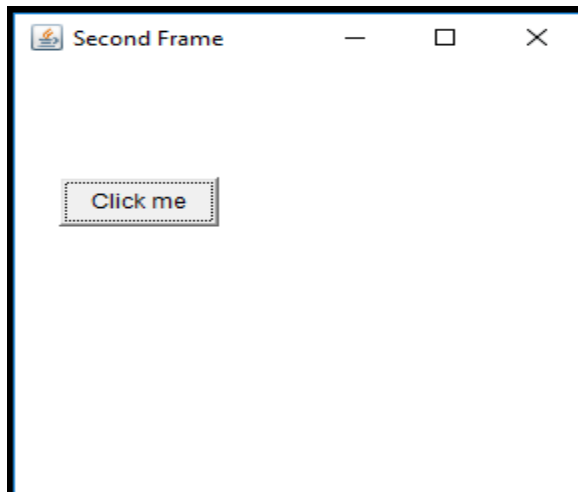
```
import java.awt.*;
class FrameDemo2
{
    public static void main(String args[])
    {
        Frame f=new Frame(); //create frame
        f.setTitle("Second Frame "); //set frame title
        f.setSize(300,300); //frame size 300 width and 300 height
        f.setLayout(null); //no layout manager
    }
}
```

```

f.setVisible(true);    //now frame will be visible, by default not visible
Button b=new Button("Click me");
b.setBounds(30,100,80,30); // setting button position
f.add(b);    //adding button into frame
}
}

```

**OUTPUT:**



### Closing the Frame:

- When we run application of Frame then it does not close by clicking on close button.
- If we have to close Frame application, then we have to **addWindowListener** containing **windowClosing ( ) method** that has code **System.exit(0)** it exit from entire awt application.
- **OR.** If we have to close current running Frame, then we have to **addWindowListener** containing **windowClosing ( ) method** that uses Frame class method **dispose ( )** it closes only current Frame.

Following program shows frame closing demo.

```

import java.awt.*;
import java.awt.event.*;
class FrameClose extends Frame
{
public static void main(String args[])
{
    FrameClose f=new FrameDemo2();
    f.setTitle("CLOSE DEMO"); //set frame title
    f.setSize(300,300);    //frame size 300 width and 300 height
    f.setLayout(null);    //no layout manager
    f.setVisible(true);    //now frame will be visible, by default not visible
    Button b=new Button("Click me");
    b.setBounds(30,100,80,30); // setting button position
    f.add(b);    //adding button into frame
    f.addWindowListener(new WindowAdapter ( )
    {
        public void windowClosing(WindowEvent e)
        {
            System.exit(0);
        }
    });
}
}

```

## Java awt Controls:

- Java AWT controls are the controls that are used to design graphical user interfaces for java applications.
- To make an effective GUI, Java provides **java.awt** package that supports various AWT controls like Label, Button, CheckBox, CheckBox Group, List, Text Field, Text Area, Choice, Canvas, Image, Scrollbar, Dialog, File Dialog, etc that creates or draw various components on web and GUI based application.
- To design and manipulate Java GUI application, java provides different awt controls which are discussed bellow (**Note that all these are built in classes of java.awt package**) –

### 1) Label

- A label is a GUI control which can be used to display static text. Label can be created using the Label class.
- Label class has constructors which are listed below:
  - Label()
  - Label(String str)
  - Label(String str, int how)
- The parameter *how* specifies the text alignment. And its valid values are Label.LEFT, Label.CENTER or Label.RIGHT

#### Methods available in the Label class are as follows:

- void setText(String str) – To set or assign text to the label.
- String getText() – To retrieve the text of a label.
- void setAlignment(int how) – To set the alignment of text in a label.
- int getAlignment() – To get the alignment of text in a label.
- Following program shows use of Label class:

```
import java.awt.*;
import java.awt.event.*;
public class MyFrame
{
    Label myLabel=new Label("This is a label!");
    MyFrame()
    {
        Frame f=new Frame();
        f.setSize(400, 200);
        f.setTitle("My Application");
        f.setLayout(new FlowLayout());
        f.setVisible(true);
        myLabel.setBounds(150,20,180,25);
        f.add(myLabel);
        f.addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent we)
            {
                System.exit(0);
            }
        });
    }
    public static void main(String[] args)
    {
        MyFrame mf = new MyFrame();
    }
}
```

OUTPUT:



## 2) Button

- A Button or push button is the frequently used GUI control.
- A push button or a button can be created by using the Button class.
- Button class has constructors which are given below:

Button()

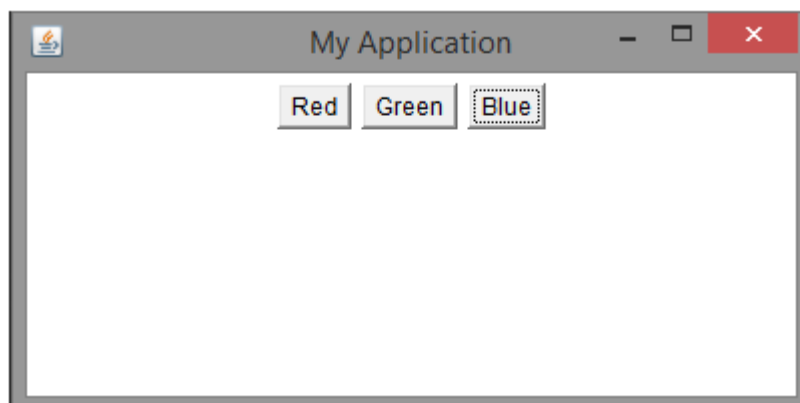
Button(String str)

### Methods available in the Button class are as follows:

- void setLabel(String str) – To set or assign the text to be displayed on the button.
- String getLabel() – To retrieve the text on the button.
- When a button is clicked, it generates an ActionEvent which can be handled using the ActionListener interface and the event handling method is actionPerformed(). If there are multiple buttons we can get the label of the button which was clicked by using the method getActionCommand().
- Following program shows use of Button class:

```
import java.awt.*;
import java.awt.event.*;
public class MyFrame
{
    Button b1 = new Button("Red");
    Button b2 = new Button("Green");
    Button b3 = new Button("Blue");
    MyFrame()
    {
        Frame f=new Frame();
        f.setSize(400, 200);
        f.setTitle("My Application");
        f.setLayout(null);
        f.setVisible(true);
        b1.setBounds(100,30,60,30);
        b2.setBounds(180,30,60,30);
        b3.setBounds(260,30,60,30);
        f.add(b1);
        f.add(b2);
        f.add(b3);
        f.addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent we)
            {
                System.exit(0);
            }
        });
    }
    public static void main(String[] args)
    {
        MyFrame mf = new MyFrame();
    }
}
```

### OUTPUT:



### 3) Checkbox

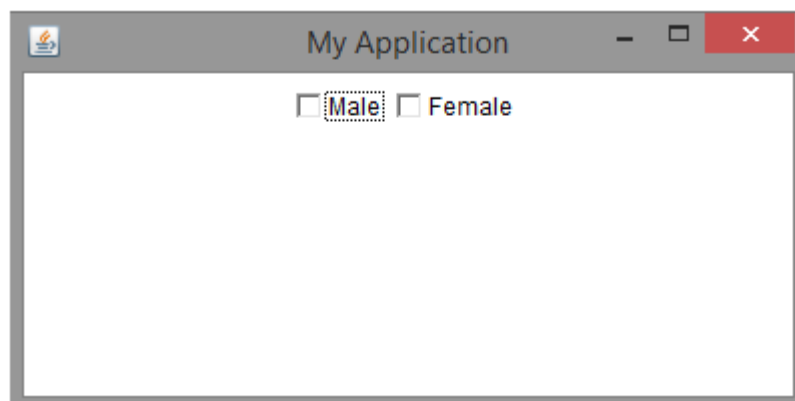
- A checkbox control can be created using the Checkbox class.
- Checkbox class has following constructors:  
Checkbox()  
Checkbox(String str)  
Checkbox(String str, boolean on)  
Checkbox(String str, boolean on, CheckboxGroup cbGroup)  
Checkbox(String str, CheckboxGroup cbGroup, boolean on)

#### Methods available in the Checkbox class:

- boolean getState() – To retrieve the state of a checkbox.
- void setState(boolean on) – To set the state of a checkbox.
- String getLabel() – To retrieve the text of a checkbox.
- void setLabel(String str) – To set the text of a checkbox.
- A checkbox when selected or deselected, generates an ItemEvent which can be handled using the ItemListener interface and the corresponding event handling method is itemStateChanged().
- Following program shows use of Checkbox class:

```
import java.awt.*;
import java.awt.event.*;
public class MyFrame
{
    Checkbox c1 = new Checkbox("Male");
    Checkbox c2 = new Checkbox("Female");
    MyFrame()
    {
        Frame f=new Frame();
        f.setSize(400, 200);
        f.setTitle("My Application");
        f.setLayout(null);
        f.setVisible(true);
        c1.setBounds(100,40,80,30);
        c2.setBounds(200,40,80,30);
        f.add(c1);
        f.add(c2);
        f.addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent we)
            {
                System.exit(0);
            }
        });
    }
    public static void main(String[] args)
    {
        MyFrame mf = new MyFrame();
    }
}
```

Output:





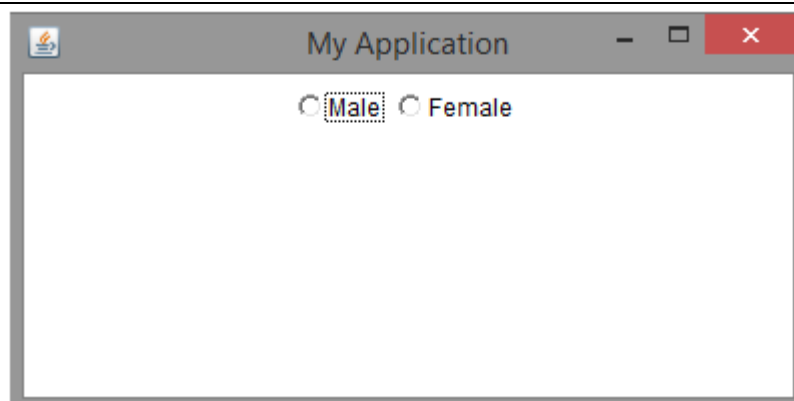
**Note:** In AWT, there is no separate class for creating radio buttons. The difference between a checkbox and radio button is, a user can select one or more checkboxes. Whereas, a user can select only one radio button in a group.

**Radio buttons can be create by using Checkbox class and CheckboxGroup class** as shown in the below code:

```
import java.awt.*;
import java.awt.event.*;
public class MyFrame
{
    CheckboxGroup cbg= new CheckboxGroup();
    Checkbox c1 = new Checkbox("Male", cbg, false);
    Checkbox c2 = new Checkbox("Female", cbg, false);

    MyFrame()
    {
        Frame f=new Frame();
        f.setSize(400, 200);
        f.setTitle("My Application");
        f.setLayout(null);
        f.setVisible(true);
        c1.setBounds(100,40,80,30);
        c2.setBounds(200,40,80,30);
        f.add(c1);
        f.add(c2);
        f.addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent we)
            {
                System.exit(0);
            }
        });
    }
    public static void main(String[] args)
    {
        MyFrame mf = new MyFrame();
    }
}
```

**Output:**



#### 4) Choice class (Dropdown list)

- A drop down box or a combo box contains a list of items (strings).
- When a user clicks on a drop down box, it pops up a list of items from which user can select a single item.
- A drop down list can be created using the **Choice class**.
- There is only one constructor in the choice class using which we can create an empty list.

##### Methods available in Choice class:

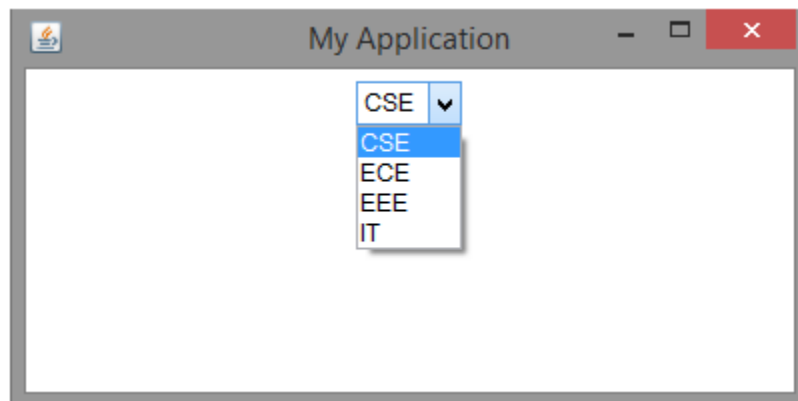
- void add(String name) – To add an item to the drop down list.
- String getSelectedItem() – To retrieve the item selected by the user.
- int getSelectedIndex() – To retrieve the index of the item selected by the user.
- int getItemCount() – To retrieve the number of items in the drop down list.
- void select(int index) – To select an item based on the given index.

- void select(String name) – To select an item based on the given item name.
- void getItem(int index) – To retrieve an item at the given index.
- Whenever an user selects an item from the drop down box, an ItemEvent is generated. It can be handled using the ItemListener interface and the event handling method is itemStateChanged().

Following code demonstrates working with drop down boxes:

```
import java.awt.*;
import java.awt.event.*;
public class MyFrame
{
    Choice myList = new Choice();
    MyFrame()
    {
        Frame f=new Frame();
        f.setSize(400, 200);
        f.setTitle("My Application");
        f.setLayout(null);
        f.setVisible(true);
        myList.setBounds(100,50,120,30);
        myList.add("CSE");
        myList.add("ECE");
        myList.add("EEE");
        myList.add("IT");
        f.add(myList);
        f.addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent we)
            {
                System.exit(0);
            }
        });
    }
    public static void main(String[] args)
    {
        MyFrame mf = new MyFrame();
    }
}
```

**Output:**



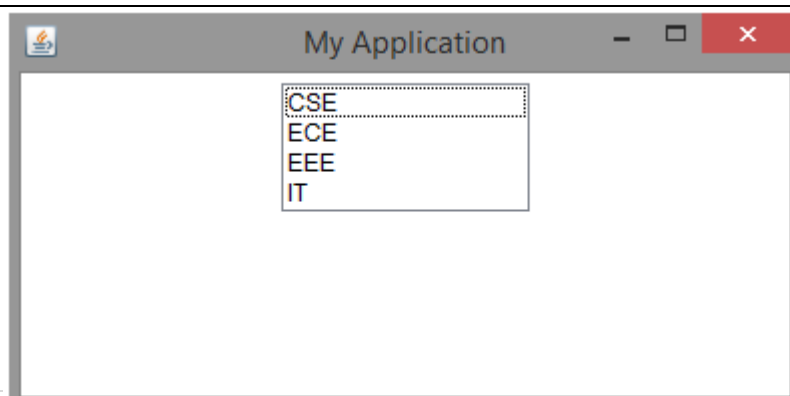
## 5) List Box

- A List box contains a list of items among which the user can select one or more items.
- More than one items in the list box are visible to the user.
- A list box can be created using the **List class**.
- List class has following constructors:
  - List()
  - List(int numRows)
  - List(int numRows, boolean multipleSelect)
- In the above constructors, numRows specifies the number of items to be visible to the user and multipleSelect specifies whether the user can select multiple items or not.

- When a list item is double clicked, `ActionEvent` is generated. It can be handled with `ActionListener` and the event handling method is `actionPerformed()`. We can get the name of the item using `getActionCommand()` method.
- When a list item is selected or deselected, `ItemEvent` is generated. It can be handled with `ItemListener` and the event handling method is `itemStateChanged()`. We can use `getItemSelectable()` method to obtain a reference to the object that raised this event.
- **Methods available in the List class:**
  - `void add(String name)` – To add an item to the list box.
  - `void add(String name, int index)` – To add an item at the specified index in the list box.
  - `String getSelectedItem()` – To get the item name which is selected by the user.
  - `int getSelectedIndex()` – To get the item index which is selected by the user.
  - `String[] getSelectedItems()` – To retrieve the selected item names by the user.
  - `int[] getSelectedIndexes()` – To retrieve the selected item indexes by the user.
  - `int getItemCount()` – To retrieve the number of items in the list box.
  - `void select(int index)` – To select an item based on the given index.
- Following code demonstrates working with list boxes:

```
import java.awt.*;
import java.awt.event.*;
public class MyFrame extends Frame
{
    List myList=new List();
    MyFrame()
    {
        Frame f=new Frame();
        f.setSize(400, 200);
        f.setTitle("My Application");
        f.setLayout(null);
        f.setVisible(true);
        myList.setBounds(150,50,100,90);
        myList = new List();
        myList.add("CSE");
        myList.add("ECE");
        myList.add("EEE");
        myList.add("IT");
        f.add(myList);
        f.addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent we)
            {
                System.exit(0);
            }
        });
    }
    public static void main(String[] args)
    {
        MyFrame mf = new MyFrame();
    }
}
```

**Output:**



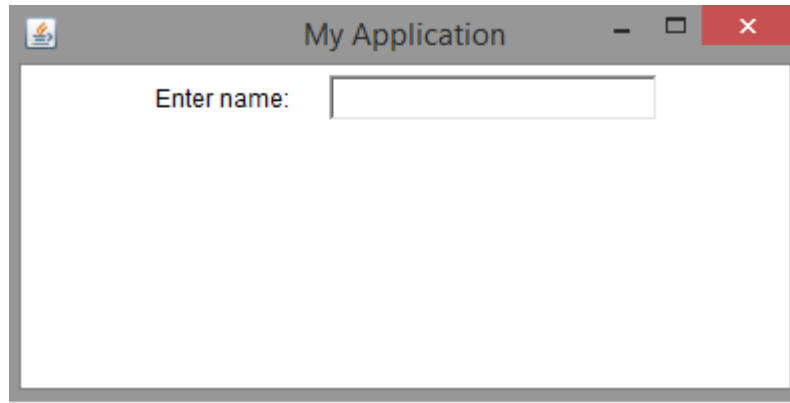
## 6) Text Fields

- A text field or text box is a single line text entry control which allows the user to enter a single line of text.
- A text field can be created using the **TextField class**
- TextFields has following constructors:
  - TextField()
  - TextField(int numChars)
  - TextField(String str)
  - TextField(String str, int numChars)
- In the above constructors numChars specifies the width of the text field, and str specifies the initial text in the text field.
- When an user hits 'Enter' key on the keyboard in a text field, an **ActionEvent** is generated. It can be handled using **ActionListener** and the event handling method is **actionPerformed()**.
- Whenever an user modifies the text in the text field, a **TextEvent** is generated which can be handled using **TextListener** and the event handling method is **textValueChanged()**.
- **Methods available in TextField class:**
  - String **getText()** – Retrieves the text in the text field.
  - void **setText(String str)** – Assigns or sets text in the text field.
  - String **getSelectedText()** – Retrieves the selected text in the text field.
  - void **select(int startindex, int endindex)** – To select the text in text field from startindex to endindex – 1.
  - boolean **isEditable()** – To check whether the text field is editable or not.
  - void **setEditable(boolean canEdit)** – To make a text field editable or non-editable.
  - void **setEchoChar(char ch)** – To set the echo character of a text field. This is generally used for password fields.
  - boolean **echoCharIsSet()** – To check whether the echo character for the text field is set or not.
  - char **getEchoChar()** – To retrieve the current echo character.

Following code demonstrates working with text fields:

```
import java.awt.*;
import java.awt.event.*;
public class MyFrame
{
    Label myLabel= new Label("Enter name: ");
    TextField tf= new TextField();
    MyFrame()
    {
        Frame f=new Frame();
        f.setSize(400, 200);
        f.setTitle("My Application");
        f.setLayout(null);
        f.setVisible(true);
        myLabel.setBounds(80,50,110,30);
        tf.setBounds(200,50,150,30);
        f.add(myLabel);
        f.add(tf);
        f.addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent we)
            {
                System.exit(0);
            }
        });
    }
    public static void main(String[] args)
    {
        MyFrame mf = new MyFrame();
    }
}
```

**Output:**



## 7) Text Areas

- A text area is a multi-line text entry control in which user can enter multiple lines of text.
- A text area can be created using the **TextArea** class.
- It has following constructors:
  - TextArea()
  - TextArea(int numLines, int numChars)
  - TextArea(String str)
  - TextArea(String str, int numLines, int numChars)
  - TextArea(String str, int numLines, int numChars, int sBars)
- In the above constructors, numLines specifies the height of the text area, numChars specifies the width of the text area, str specifies the initial text in the text area and sBars specifies the scroll bars. Valid values of sBars can be any one of the following:

SCROLLBARS\_BOTH

SCROLLBARS\_NONE

SCROLLBARS\_HORIZONTAL\_ONLY

SCROLLBARS\_VERTICAL\_ONLY

### Methods available in the TextArea class:

- String getText() – To retrieve the text in the text area.
- void setText(String str) – To assign or set the text in a text area.
- String getSelectedText() – To retrieve the selected text in a text area.
- void select(int startindex, int endindex) – To select the text in text field from startindex to endindex – 1.
- boolean isEditable() – To check whether the text field is editable or not.
- void setEditable(boolean canEdit) – To make a text field editable or non-editable.
- void append(String str) – To append the given string to the text in the text area.
- void insert(String str, int index) – To insert the given string at the specified index.
- void replaceRange(String str, int startIndex, int endIndex) – To replace the text from startIndex to endIndex – 1 with the given string.

Following code demonstrates use of Textarea class:

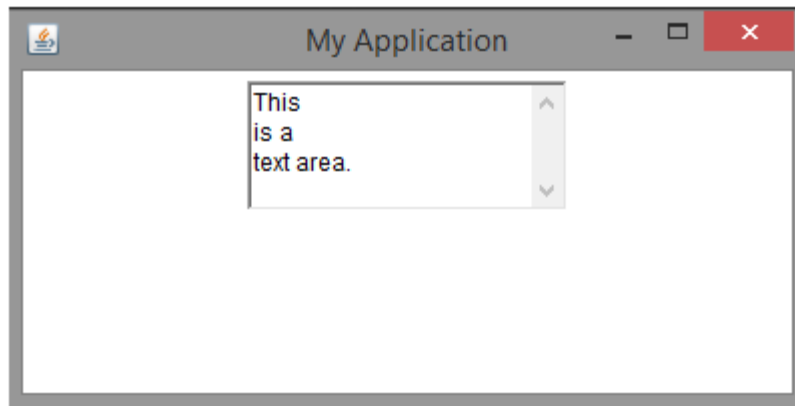
```
import java.awt.*;
import java.awt.event.*;
public class MyFrame
{
    TextArea ta= new TextArea();
    MyFrame()
    {
        Frame f=new Frame();
        f.setSize(400, 200);
        f.setTitle("My Application");
        f.setLayout(null);
        f.setVisible(true);
        ta.setBounds(150,70,120,100);
        f.add(ta);
        f.addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent we)
            {
                System.exit(0);
            }
        });
    }
}
```

```

    }
    });
}
public static void main(String[] args)
{
    MyFrame mf = new MyFrame();
}
}

```

**OUTPUT:**



**Following program demonstrate use of different awt controls.**

```

import java.awt.event.*;
import java.awt.*;
class college
{
    TextField    tx1=new TextField();
    TextArea     ta=new TextArea();

    Label        l1=new Label("STUDENT INFORMATION");
    Label        l2=new Label("Student Name");
    Label        l3=new Label("Select Class");
    Label        l4=new Label("Enter Address");
    Label        l5=new Label("Select Subjects");
    Label        l6=new Label("Staying at Hostel?");
    Label        l7=new Label("Select Hobby");

    CheckboxGroup cbg=new CheckboxGroup(); /*in java.awt, when we add check boxes on checkbox group
                                                then they became radiobox automatically*/

    Checkbox     chk1=new Checkbox("Div-B",cbg,false);
    Checkbox     chk2=new Checkbox("Div-C",cbg,false);

    Checkbox     chkSub1=new Checkbox("C");
    Checkbox     chkSub2=new Checkbox("C++");
    Checkbox     chkSub3=new Checkbox("JAVA");

    List         lb=new List(4,false);
    Choice       hoby=new Choice();

    Button       btn1=new Button("Submit");
    Button       btn2=new Button("Cancel");
    college()
    {
        Frame f=new Frame("AWT Controls Demo");
        //set size, layout and visibility of frame
        f.setSize(400,400);
        f.setLayout(null);
        f.setVisible(true);
        //add items into List control
        lb.add("Yes");
    }
}

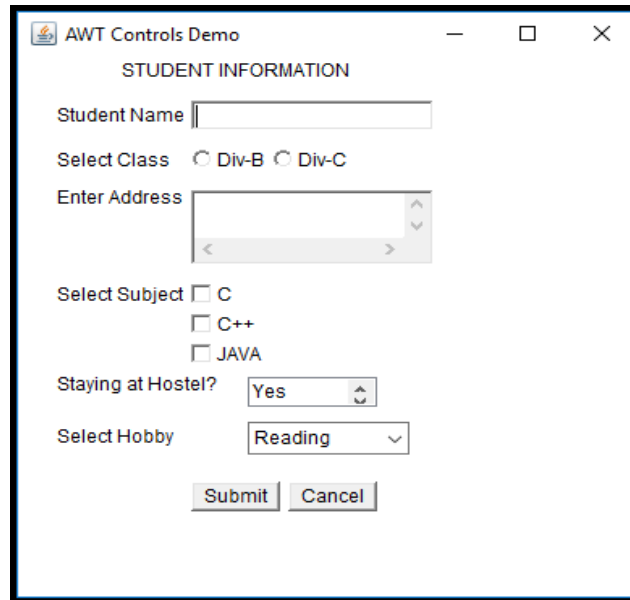
```

```

lb.add("No");
    //add items into Choice control
hoby.add("Reading");
hoby.add("Swimming");
hoby.add("Watching TV");
hoby.add("Internet Surfing");
    // set position of controls onto frame
l1.setBounds(70,30,150,20);
tx1.setBounds(115,60,150,20);
ta.setBounds(115,120,150,50);
l2.setBounds(30,60,80,20);
l3.setBounds(30,90,80,20);
l4.setBounds(30,115,80,20);
l5.setBounds(30,180,80,20);
l6.setBounds(30,240,120,20);
l7.setBounds(30,275,120,20);
chk1.setBounds(115,90,50,20);
chk2.setBounds(165,90,50,20);
chkSub1.setBounds(115,180,80,20);
chkSub2.setBounds(115,200,80,20);
chkSub3.setBounds(115,220,80,20);
lb.setBounds(150,245,80,20);
hoby.setBounds(150,275,100,20);
btn1.setBounds(115,315,55,20);
btn2.setBounds(175,315,55,20);
    //add controls onto frame
f.add(tx1);
f.add(l1);
f.add(l2);
f.add(l3);
f.add(chk1);
f.add(chk2);
f.add(ta);
f.add(l4);
f.add(l5);
f.add(chkSub1);
f.add(chkSub2);
f.add(chkSub3);
f.add(l6);
f.add(lb);
f.add(hoby);
f.add(l7);
f.add(btn1);
f.add(btn2);
    // to exit from running frame
f.addWindowListener(new WindowAdapter()
{
    public void windowClosing(WindowEvent e)
    {
        System.exit(1);
    }
});
}
public static void main(String arg[])
{
    college p=new college();
}
}

```

OUTPUT:



### Java GUI Event Handling: Event and Listener (Java 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.

- Note: For GUI event handling notes Refere above mentioned notes in Applet part on (Page No- 8 and Page No- 9)**

**Program-1) Following program shows GUI event handling that displayed message in TextField by clicking on Button.**

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

public class MouseClkDemo implements ActionListener
{
    TextField txt=new TextField();
    Button btn=new Button("ClickMe");
    public MouseClkDemo()
    {
        Frame fr=new Frame("MouseClicked Demo");
        //set size,layout and visibilty for frame
        fr.setSize(300,300);
        fr.setLayout(null);
        fr.setVisible(true);
        //create font
        Font fnt=new Font("Arial",Font.BOLD,20);
        //set display possition for controls
        txt.setBounds(60,50,150,30);
        btn.setBounds(80,90,100,30);
        //apply created font to controls
        btn.setFont(fnt);
        txt.setFont(fnt);
        //adding controls onto frame
        fr.add(txt);
        fr.add(btn);
        //register listner to button
        btn.addActionListener(this);
        // to exit from frame
        fr.addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent e)
            {
            }
        });
    }
}
```

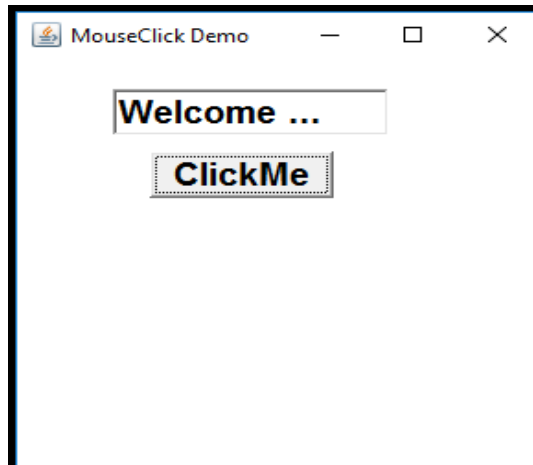


```

        System.exit(1);
    }
});
}
// code to execute after event occure
    public void actionPerformed(ActionEvent e)
    {
        txt.setText("Welcome ...");
    }
}
public static void main(String []arg)
{
    MouseClkDemo m=new MouseClkDemo();
}
}

```

**OUTPUT:**



**Program-2) Following program shows GUI event handling that displayed entered text of TextField to Label by clicking on Button.**

```

import java.awt.*;
import java.awt.event.*;
public class LabelAction implements ActionListener
{
    TextField txt=new TextField();
    Button btn=new Button("ClickMe");
    Label lb=new Label();
    public LabelAction()
    {
        Frame fr=new Frame("ButtonClick Demo");
        //set size,layout and visibility for frame
        fr.setSize(300,300);
        fr.setLayout(null);
        fr.setVisible(true);
        //create font
        Font fnt=new Font("Arial",Font.BOLD,18);
        //set display position for controls
        txt.setBounds(60,50,150,30);
        btn.setBounds(80,90,100,30);
        lb.setBounds(60,125,120,30);
        //apply created font to controls
        btn.setFont(fnt);
        txt.setFont(fnt);
        lb.setFont(fnt);
        //adding controls onto frame
        fr.add(txt);
        fr.add(btn);
        fr.add(lb);
        //register listener to button
    }
}

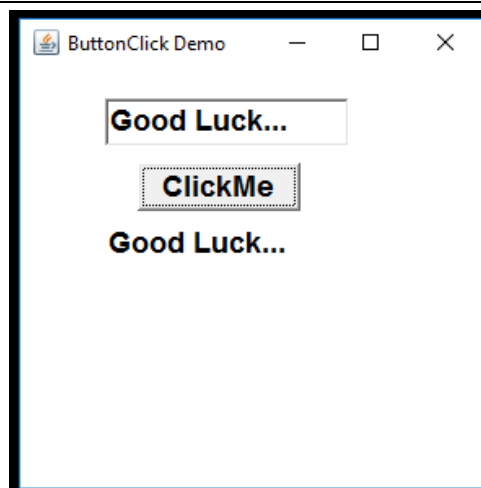
```

```

        btn.addActionListener(this);
        // to exit from frame
        fr.addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent e)
            {
                System.exit(1);
            }
        });
    }
    // code to execute after event occur
    public void actionPerformed(ActionEvent e)
    {
        String str=txt.getText(); //retrieve text of TextField
        lb.setText(str); // set text str to Label
    }
    public static void main(String []arg)
    {
        LabelAction m=new LabelAction();
    }
}

```

**OUTPUT:**



**Program-3) Program shows GUI application that finds addition, subtraction, multiplication and division.**

```

import java.awt.*;
import java.awt.event.*;
public class Calculation implements ActionListener
{
    TextField txt1=new TextField();
    TextField txt2=new TextField();
    TextField txt3=new TextField();
    Button btn1=new Button("ADD");
    Button btn2=new Button("SUB");
    Button btn3=new Button("MULTI");
    Button btn4=new Button("DIV");
    Label lb1=new Label("First");
    Label lb2=new Label("Second");
    Label lb3=new Label("Result");
    public Calculation()
    {
        Frame fr=new Frame("Simple Calculation Demo");
        //set size,layout and visibility for frame
        fr.setSize(400,300);
        fr.setLayout(null);
        fr.setVisible(true);
        //create font
    }
}

```

```

Font fnt=new Font("Arial",Font.BOLD,15);
    //set display position for controls
txt1.setBounds(130,50,150,30);
txt2.setBounds(130,90,150,30);
txt3.setBounds(130,130,150,30);
lb1.setBounds(50,50,150,30);
lb2.setBounds(50,90,150,30);
lb3.setBounds(50,130,150,30);

btn1.setBounds(50,180,50,30);
btn2.setBounds(110,180,50,30);
btn3.setBounds(170,180,55,30);
btn4.setBounds(230,180,50,30);

    //apply created font to controls
btn1.setFont(fnt);
btn2.setFont(fnt);
btn3.setFont(fnt);
btn4.setFont(fnt);
txt1.setFont(fnt);
txt2.setFont(fnt);
txt3.setFont(fnt);
lb1.setFont(fnt);
lb2.setFont(fnt);
lb3.setFont(fnt);

    //adding controls onto frame
fr.add(txt1);
fr.add(txt2);
fr.add(txt3);
fr.add(btn1);
fr.add(btn2);
fr.add(btn3);
fr.add(btn4);
fr.add(lb1);
fr.add(lb2);
fr.add(lb3);
    //register listener to buttons
btn1.addActionListener(this);
btn2.addActionListener(this);
btn3.addActionListener(this);
btn4.addActionListener(this);

    // to exit from frame
fr.addWindowListener(new WindowAdapter()
{
    public void windowClosing(WindowEvent e)
    {
        System.exit(1);
    }
});
}
// code to execute after event occur
public void actionPerformed(ActionEvent e)
{
    String s1=txt1.getText();
    String s2=txt2.getText();
    double d1=Double.parseDouble(s1);

```

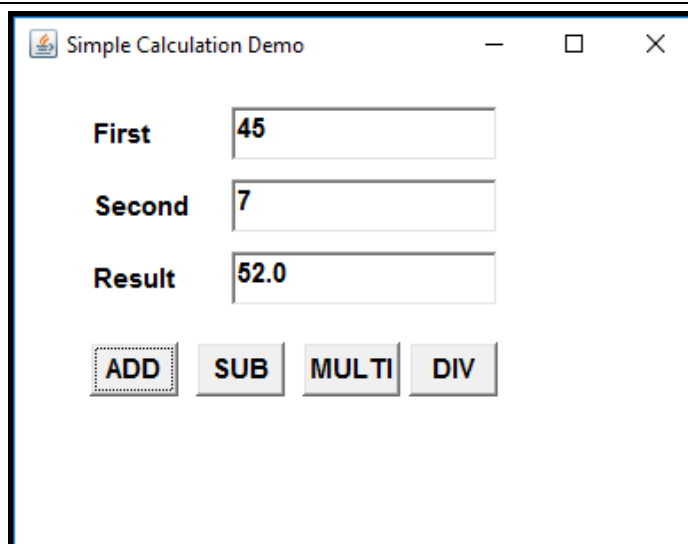
```

double d2=Double.parseDouble(s2);
double d3;
if(e.getSource()==btn1)
{
    d3=d1+d2;
    String res=Double.toString(d3);
    txt3.setText(res);
}
else if(e.getSource()==btn2)
{
    d3=d1-d2;
    String res=Double.toString(d3);
    txt3.setText(res);
}
else if(e.getSource()==btn3)
{
    d3=d1*d2;
    String res=Double.toString(d3);
    txt3.setText(res);
}
else if(e.getSource()==btn4)
{
    d3=d1/d2;
    String res=Double.toString(d3);
    txt3.setText(res);
}
}
}

public static void main(String []arg)
{
    Calculation m=new Calculation();
}
}

```

**OUTPUT:**



**Program-4) Program shows GUI application that display selected item of First List into Second List and vice versa by clicking on respective buttons.**

```

import java.awt.*;
import java.awt.event.*;
public class ListItemSelect implements ActionListener
{
    List lst_one=new List();
    List lst_two=new List();
    Button b1=new Button(">>");
}

```

```

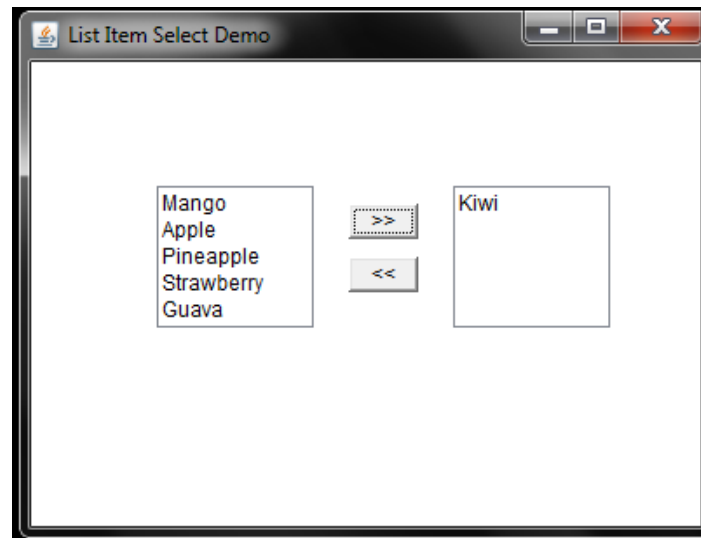
Button b2=new Button("<<");
ListItemSelect()
{
    Frame f=new Frame("List Item Select Demo");
    f.setSize(400,300);
    f.setVisible(true);
    f.setLayout(null);
    lst_one.setBounds(80,100,90,80);
    lst_two.setBounds(250,100,90,80);
    b1.setBounds(190,110,40,20);
    b2.setBounds(190,140,40,20);
    //add items in First List Box
    lst_one.add("Mango");
    lst_one.add("Apple");
    lst_one.add("Kiwi");
    lst_one.add("Pineapple");
    lst_one.add("Strawberry");
    lst_one.add("Guava");
    //add controls on to Frame
    f.add(lst_one);
    f.add(lst_two);
    f.add(b1);
    f.add(b2);
    // register listener to Buttons
    b1.addActionListener(this);
    b2.addActionListener(this);
    //to close frame
    f.addWindowListener(new WindowAdapter()
    {
        public void windowClosing(WindowEvent e)
        {
            System.exit(1);
        }
    });
}

public void actionPerformed(ActionEvent e)
{
    if(e.getSource()==b1)
    {
        String s=lst_one.getSelectedItemAt();
        lst_two.add(s);
        lst_one.remove(s);
    }
    else if(e.getSource()==b2)
    {
        String s=lst_two.getSelectedItemAt();
        lst_one.add(s);
        lst_two.remove(s);
    }
}

public static void main(String []ar)
{
    ListItemSelect p=new ListItemSelect();
}
}

```

## OUTPUT:



## 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.
- We know that Java AWT is platform dependent but Java Swing provides platform-independency and it has lightweight components.
- The java swing provides decent look for all component as compared to java awt components that's why now a days Java swing is widely used to develop GUI application.
- The **javax.swing** package provides classes for java swing programming such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

## Difference between AWT and Swing

There are many differences between java awt and swing that are given below.

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

## Main Features of Swing Toolkit:

- Platform Independent
- Customizable
- Extensible
- Configurable
- Lightweight
- Rich Controls
- Pluggable Look and Feel

## What is JFC?

- The Java Foundation Classes (JFC) are a set of GUI components which simplify the development of desktop applications.
- Features of JFC
  - Swing GUI components.
  - Look and Feel support.

## Swing and JFC

- JFC is an abbreviation for Java Foundation classes, which encompass a group of features for building Graphical User Interfaces (GUI) and adding rich graphical functionalities and interactivity to Java applications. Java Swing is a part of Java Foundation Classes (JFC).
- Commonly used Methods of swing Component class are-
- 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.

### Creating JFrame:

- To create simple swing example, we need a JFrame. A JFrame is basic component in AWT.
- The JFrame has to be created before any other component because all other componets (JButton, JTextfield etc) can be displayed on JFrame.
- There are two ways to create a JFrame in Swing-
  - 3) By extending JFrame class (inheritance)
  - 4) By creating the object of JFrame class (association)

Let's see these ways in details-

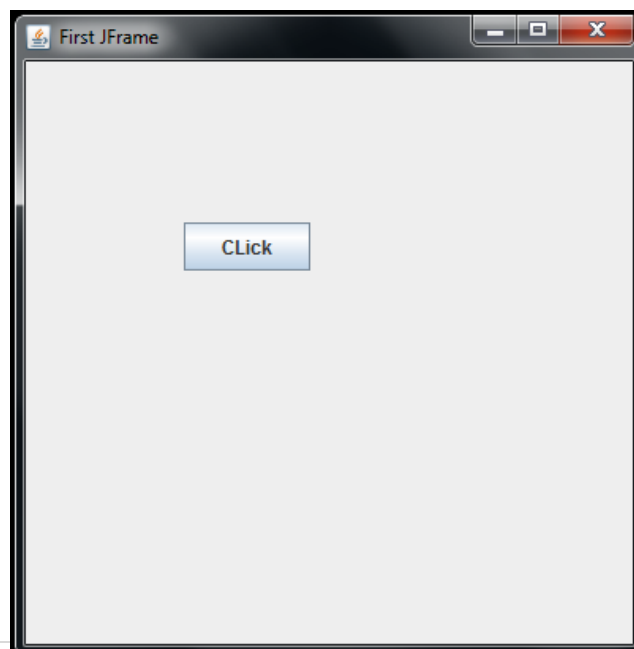
#### 1) Creating Frame by extending JFrame class (Using inheritance)

Let's see a simple example of Swing where we are inheriting JFrame class.

Here, we are showing JButton component on the JFrame.

```
import javax.swing.*;
public class NewFrameDemo extends JFrame
{
    public static void main(String []ar)
    {
        NewFrameDemo f=new NewFrameDemo();
        f.setSize(400,400);
        f.setVisible(true);
        f.setLayout(null);
        f.setTitle("First JFrame");
        JButton btn=new JButton("Click");
        btn.setBounds(100,100,80,30);
        f.add(btn);
    }
}
```

**OUTPUT:**



- In above example **setBounds (int x, int y, int width, int height)** method is used in to set the position of the swing JButton.

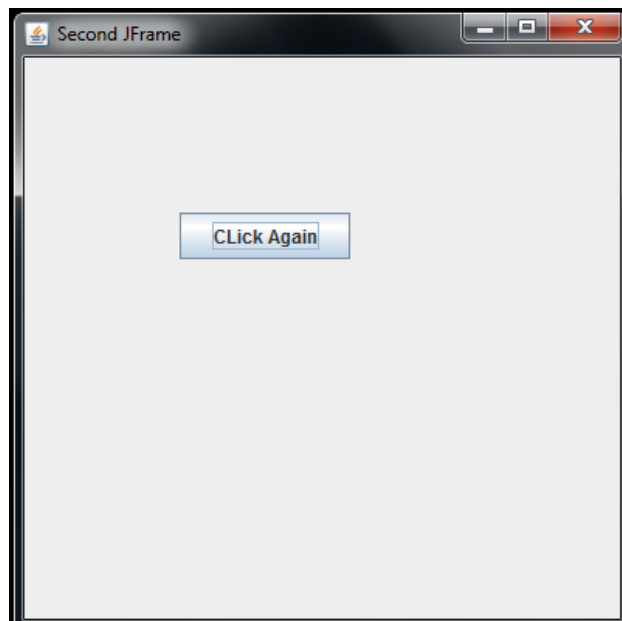
## 2) Creating JFrame by object of JFrame class (Association):

Let's see a simple example of swing where we are creating object i.e. instance of JFrame class.

Here, we are showing JButton component on the JFrame.

```
import javax.swing.*;
public class SecondJFrame
{
    public static void main(String []ar)
    {
        JFrame f=new JFrame();
        f.setSize(400,400);
        f.setVisible(true);
        f.setLayout(null);
        f.setTitle("Second JFrame");
        JButton btn=new JButton("CLick Again");
        btn.setBounds(100,100,110,30);
        f.add(btn);
    }
}
```

**OUTPUT:**



## Closing the JFrame:

- When we run application of JFrame then it closes by clicking on close button but still this JFrame application remains active in RAM.
- If we have to close JFrame application totally (from RAM also), then we have to use JFrame method **setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);** that closes entire swing application.
- **OR** If we have to close current running JFrame only (from RAM also), then we have to use JFrame method **setDefaultCloseOperation(JFrame.DISPOSE\_ON\_CLOSE);** that closes current JFrame only.
- Following program shows JFrame closing demo.

```
import java.awt.*;
import javax.swing.*;
class MySecFrame
{
    public static void main(String args[])
    { JFrame f=new JFrame("MY Second JFrame"); //create JFrame with title
      f.setSize(300,300); //frame size 300 width and 300 height
      f.setLayout(null); //no layout manager
      f.setVisible(true); //now JFrame will be visible, by default not visible
      JButton b=new JButton("Click...");
      b.setBounds(70,70,80,30); // setting button position
      f.add(b); //adding JButton on JFrame
    }
}
```



```
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
}  
}
```

## Java Swing Controls:

- Java swing controls are the controls that are used to design graphical user interfaces for java applications.
- To make an effective GUI, Java provides **javax.swing** package that supports various swing controls like JLabel, JButton, JCheckBox, JTextField, JList, JRadioButton, JComboBox etc that creates or draw various components on web and GUI based application.
- To design and manipulate Java GUI application, java provides different swing controls which are discussed bellow (**Note that all these are built in classes of javax.swing package**) –

### 1) JLabel

- The JLabel class is used to display a label i.e., static text onto GUI application.
- A JLabel class has following constructors:
  - JLabel(Icon icon)
  - JLabel(String str)
  - JLabel(String str, Icon icon, int align)
- In the above constructors icon is used to specify an image to be displayed as a label. Icon is a predefined interface which is implemented by the ImageIcon class. str is used to specify the text to be displayed in the label and align is used to specify the alignment of the text.
- **Methods of JLabel class are as follows:**
  - void setText(String str) – To set the text of the label
  - String getText() – To get the text of the label
  - void setIcon(Icon icon) – To display an image in the label
  - Icon getIcon() – To retrieve the image displayed in the label

### 2) JButton

- The JButton class is used to display a push button.
- A JButton has following constructors:
  - JButton(Icon icon)
  - JButton(String str)
  - JButton(String str, Icon icon)
- In the above constructors, icon specifies the image to be displayed as button and str specifies the text to be displayed on the button.
- JButton objects raise ActionEvent when the button is clicked. It can be handled by implementing ActionListener interface and the event handling method is actionPerformed().
- **The JButton class implements the abstract class AbstractButton which provides the following methods:**
  - void setDisableIcon(Icon di) – To set the icon to be displayed when button is disabled
  - void setPressedIcon(Icon pi) – To set the icon to be displayed when button is pressed
  - void setSelectedIcon(Icon si) – To set the icon to be displayed when button is selected
  - void setRolloverIcon(Icon ri) – To set the icon to be displayed when the button is rolled over
  - void setText(String str) – To set the text to be displayed on the button
  - String getText() – To retrieve the text displayed on the button

### 3) JCheckBox

- Check box can be created using the JCheckBox class which inherits the JToggleButton class.
- JCheckBox has following constructor:
  - JCheckBox() : creates a new checkbox with no text or icon
  - JCheckBox(Icon i) : creates a new checkbox with the icon specified
  - JCheckBox(Icon icon, boolean s) : creates a new checkbox with the icon specified and the boolean value specifies whether it is selected or not.
  - JCheckBox(String t) : creates a new checkbox with the string specified
  - JCheckBox(String text, boolean selected) : creates a new checkbox with the string specified and the boolean value specifies whether it is selected or not.
  - JCheckBox(String text, Icon icon) : creates a new checkbox with the string and the icon specified.

JCheckBox(String text, Icon icon, boolean selected): creates a new checkbox with the string and the icon specified and the boolean value specifies whether it is selected or not.

### Methods of JCheckBox-

- setIcon(Icon i) : sets the icon of the checkbox to the given icon
- setText(String s) : sets the text of the checkbox to the given text
- setSelected(boolean b) : sets the checkbox to selected if boolean value passed is true or vice versa
- getIcon() : returns the image of the checkbox
- getText() : returns the text of the checkbox
- updateUI() : resets the UI property with a value from the current look and feel.
- getUI() : returns the look and feel object that renders this component.
- paramString() : returns a string representation of this JCheckBox.
- getUIClassID() : returns the name of the Look and feel class that renders this component.
- getAccessibleContext() : gets the AccessibleContext associated with this JCheckBox.
- isBorderPaintedFlat() : gets the value of the borderPaintedFlat property.
- setBorderPaintedFlat(boolean b) : sets the borderPaintedFlat property,

## 4) JTextField

- A text field is a GUI control which allows the user to enter a single line of text.
- A text field can be created using the class JTextField which inherits the class JTextComponent.
- A JTextField class has following constructors:
  - JTextField(int cols)
  - JTextField(String str)
  - JTextField(String str, int cols)
- In the above constructors cols specifies the size of the text field and str specifies the default string to be displayed in the text field.

### Methods of JTextField:

- setColumns(int n) : set the number of columns of the text field.
- setFont(Font f) : set the font of text displayed in text field.
- addActionListener(ActionListener l) : set an ActionListener to the text field.
- int getColumns() : get the number of columns in the textfield.

## 5) JList

- JList is part of Java Swing package.
- JList is a component that displays a set of Objects and allows the user to select one or more items.
- JList inherits JComponent class. JList is a easy way to display an array of Vectors.
- Constructors of JList are :
  - JList(): creates an empty blank list
  - JList(E [ ] l) : creates a new list with the elements of the array.
  - JList(ListModel d): creates a new list with the specified List Model
  - JList(Vector l) : creates a new list with the elements of the vector

### Methods of JList are:

- getSelectedIndex() returns the index of selected item of the list
- getSelectedValue() returns the selected value of the element of the list
- setSelectedIndex(int i) sets the selected index of the list to i
- setSelectionBackground(Color c) sets the background Color of the list
- setSelectionForeground(Color c) Changes the foreground color of the list
- setListData(E [ ] l) Changes the elements of the list to the elements of l .
- setVisibleRowCount(int v) Changes the visibleRowCount property
- setSelectedValue(Object a, boolean s) selects the specified object from the list.
- setSelectedIndices(int[] i) changes the selection to be the set of indices specified by the given array.
- setListData(Vector l) constructs a read-only ListModel from a Vector specified.
- setLayoutOrientation(int l) defines the orientation of the list
- setFixedCellWidth(int w) Changes the cell width of list to the value passed as parameter.
- setFixedCellHeight(int h) Changes the cell height of the list to the value passed as parameter.
- isSelectedIndex(int i) returns true if the specified index is selected, else false.
- indexToLocation(int i) returns the origin of the specified item in the list's coordinate system.
- getToolTipText(MouseEvent e) returns the tooltip text to be used for the given event.

- `getSelectedValuesList()` returns a list of all the selected items.
- `getSelectedIndices()` returns an array of all of the selected indices, in increasing order
- `getMinSelectionIndex()` returns the smallest selected cell index, or -1 if the selection is empty.
- `getMaxSelectionIndex()` returns the largest selected cell index, or -1 if the selection is empty.
- `getListSelectionListeners()` returns the listeners of list
- `getLastVisibleIndex()` returns the largest list index that is currently visible.
- `getDragEnabled()` returns whether or not automatic drag handling is enable
- `addListSelectionListener(ListSelectionListener l)` adds a listSelectionlistener to the list

## 6) JComboBox

- JComboBox is a part of Java Swing package.
- JComboBox inherits JComponent class.
- JComboBox shows a popup menu that shows a list and the user can select a option from that specified list. JComboBox can be editable or read- only depending on the choice of the programmer.
- Constructors of the JComboBox are:  
`JComboBox()` : creates a new empty JComboBox .  
`JComboBox(ComboBoxModel M)` : creates a new JComboBox with items from specified ComboBoxModel  
`JComboBox(E [ ] i)` : creates a new JComboBox with items from specified array.  
`JComboBox(Vector items)` : creates a new JComboBox with items from the specified vector

### Methods of JComboBox are:

- `addItem(E item)` : adds the item to the JComboBox
- `addItemListener( ItemListener l)` : adds a ItemListener to JComboBox
- `getItemAt(int i)` : returns the item at index i
- `getItemCount()` : returns the number of items from the list
- `getSelectedItem()` : returns the item which is selected
- `removeItemAt(int i)` : removes the element at index i
- `setEditable(boolean b)` : the boolean b determines whether the combo box is editable or not .If true is passed then the combo box is editable or vice versa.
- `setSelectedIndex(int i)` : selects the element of JComboBox at index i.
- `showPopup()` : causes the combo box to display its popup window.
- `setUI(ComboBoxUI ui)` : sets the L&F object that renders this component.
- `setSelectedItem(Object a)` : sets the selected item in the combo box display area to the object in the argument.
- `setSelectedIndex(int a)` : selects the item at index anIndex.
- `setPopupVisible(boolean v)` : sets the visibility of the popup.
- `setModel(ComboBoxModel a)` : sets the data model that the JComboBox uses to obtain the list of items.
- `setMaximumRowCount(int count)` : sets the maximum number of rows the JComboBox displays.
- `setEnabled(boolean b)` : enables the combo box so that items can be selected.
- `removeItem(Object anObject)` : removes an item from the item list.
- `removeAllItems()` : removes all items from the item list.
- `removeActionListener(ActionListener l)` : removes an ActionListener.
- `isPopupVisible()` : determines the visibility of the popup.
- `addPopupMenuListener(PopupMenuListener l)` : adds a PopupMenu listener which will listen to notification messages from the popup portion of the combo box.
- `getActionCommand()` : returns the action command that is included in the event sent to action listeners.
- `getEditor()` : returns the editor used to paint and edit the selected item in the JComboBox field.
- `getItemCount()` : returns the number of items in the list.
- `getItemListeners()` : returns an array of all the ItemListeners added to this JComboBox with `addItemListener()`.
- `createDefaultKeySelectionManager()` : returns an instance of the default key-selection manager.
- `fireItemStateChanged(ItemEvent e)` : notifies all listeners that have registered interest for notification on this event type.
- `firePopupMenuCanceled()` : notifies PopupMenuListeners that the popup portion of the combo box has been canceled.
- `firePopupMenuWillBecomeInvisible()` : notifies PopupMenuListeners that the popup portion of the combo box has become invisible.
- `firePopupMenuWillBecomeVisible()` : notifies PopupMenuListeners that the popup portion of the combo box will become visible.

- `setEditor(ComboBoxEditor a)`: sets the editor used to paint and edit the selected item in the `JComboBox` field.
- `setActionCommand(String a)` : sets the action command that should be included in the event sent to `actionListeners`.
- `getUI()` : returns the look and feel object that renders this component.
- `paramString()` : returns a string representation of this `JComboBox`.
- `getUIClassID()` : returns the name of the Look and feel class that renders this component.
- `getAccessibleContext()` : gets the `AccessibleContext` associated with this `JComboBox`

## 7) JRadioButton

- We use the `JRadioButton` class to create a radio button.
- Radio button is use to select one option from multiple options.
- Basically it is used in filling forms, online objective papers and quiz.
- We add radio buttons in a `ButtonGroup` so that we can select only one radio button at a time. We use “`ButtonGroup`” class to create a `ButtonGroup` and add radio button in a group.

### Methods of JRadioButton:

- `JRadioButton()` : Creates a unselected `RadioButton` with no text.
- `JButton(String s)` : Creates a `JButton` with a specific text.
- `JLabel(String s)` : Creates a `JLabel` with a specific text.

### Steps to Group the radio buttons together:

- `ButtonGroup()` : Use to create a group, in which we can add `JRadioButton`. We can select only one `JRadioButton` in a `ButtonGroup`.
- Create a `ButtonGroup` instance by using “`ButtonGroup()`” Method.
- `ButtonGroup G = new ButtonGroup();`
- Now add buttons in a Group “`G`”, with the help of “`add()`” Method.
- Example:  
`G.add(Button1);`  
`G.add(Button2);`

## 8) JTextArea

- `JTextArea` is a part of java Swing package.It represents a multi line area that displays text. It is used to edit the text.
- `JTextArea` inherits `JComponent` class.
- The text in `JTextArea` can be set to different available fonts and can be appended to new text.
- A text area can be customized to the need of user.
- **Constructors of JTextArea are:**
  - `JTextArea()` : constructs a new blank text area .
  - `JTextArea(String s)` : constructs a new text area with a given initial text.
  - `JTextArea(int row, int column)` : constructs a new text area with a given number of rows and columns.
  - `JTextArea(String s, int row, int column)` : constructs a new text area with a given number of rows and columns and a given initial text.

### Methods of JTextArea:

- `append(String s)` : appends the given string to the text of the text area.
- `getLineCount()` : get number of lines in the text of text area.
- `setFont(Font f)` : sets the font of text area to the given font.
- `setColumns(int c)` : sets the number of columns of the text area to given integer.
- `setRows(int r)` : sets the number of rows of the text area to given integer.
- `getColumns()` : get the number of columns of text area.
- `getRows()` : get the number of rows of text area.

### Following program shows java swing application uses different java swing controls

```
import javax.swing.*;
import java.awt.*;
public class SwingControls
{
    JLabel headLbl=new JLabel("EMPLOYEE INFORMATION") ;
    JLabel nameLbl=new JLabel("Full Name") ;
    JLabel genLbl=new JLabel("Gender") ;
    JLabel addLbl=new JLabel("Address") ;
```

```

JLabel eduLbl=new JLabel("Education");
JLabel hobbyLbl=new JLabel("Your Hobbies");

JRadioButton genRadM=new JRadioButton("Male");
JRadioButton genRadF=new JRadioButton("FeMale");
ButtonGroup G = new ButtonGroup();

JTextField nameTxt=new JTextField();
JTextArea addTxtA=new JTextArea();

JCheckBox chk1=new JCheckBox("Painting");
JCheckBox chk2=new JCheckBox("Reading");
JCheckBox chk3=new JCheckBox("Writing");
JCheckBox chk4=new JCheckBox("Swimming");
JCheckBox chk5=new JCheckBox("Others..");

JButton subBtn=new JButton("SAVE");
JButton resBtn=new JButton("RESET");

String s1[] = {"Not Educated.", "Std 1 To 9", "SSC", "HSC", "Graduate", "Post Graduate"};
JComboBox eduCom=new JComboBox(s1);

public SwingControls()
{
    JFrame f=new JFrame("Swing Controls");
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
    //Add RadioButton to ButtonGroup
    G.add(genRadM);
    G.add(genRadF);
    //Add background color to lightgray for JTextArea
    addTxtA.setBackground(Color.lightGray);
    //set position of controls
    headLbl.setBounds(120,10,200,25);
    nameLbl.setBounds(20,40,200,25);
    genLbl.setBounds(20,70,200,25);
    addLbl.setBounds(20,95,200,25);
    eduLbl.setBounds(20,160,200,25);
    hobbyLbl.setBounds(20,190,200,25);
    nameTxt.setBounds(80,40,240,25);
    genRadM.setBounds(70,70,60,25);
    genRadF.setBounds(130,70,70,25);
    addTxtA.setBounds(80,100,240,50);
    eduCom.setBounds(80,160,120,25);
    chk1.setBounds(70,210,90,25);
    chk2.setBounds(70,240,90,25);
    chk3.setBounds(160,210,90,25);
    chk4.setBounds(160,240,90,25);
    chk5.setBounds(70,270,90,25);
    subBtn.setBounds(50,310,100,25);
    resBtn.setBounds(160,310,100,25);
    //add controls on JFrame
    f.add(headLbl);
    f.add(nameLbl);
    f.add(genLbl);
    f.add(addLbl);
    f.add(addLbl);
    f.add(eduLbl);
    f.add(hobbyLbl);

```

```

        f.add(nameTxt);
        f.add(genRadM);
        f.add(genRadF);
        f.add(addTxtA);
        f.add(eduCom);
        f.add(chk1);
        f.add(chk2);
        f.add(chk3);
        f.add(chk4);
        f.add(chk5);
        f.add(subBtn);
        f.add(resBtn);
        //to close JFrame
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
    public static void main(String []arg)
    {
        SwingControls  obj=new SwingControls();
    }
}

```

**OUTPUT:**

**Note:**

- Event handling in Java Swing is similary done as Java Applet or Java AWT. For that we have to import package **java.awt.event.\***;
- **Reffer Event Handling Notes on Page No. 8 and Page No. 9**

## Java Servlet Overview:

- Servlet technology which is used to create a web application using java language that resides at server side and generates a dynamic web page.
- Java Servlets are java programs that run on a Web server and act as a middle layer between a requests coming from a Web browser or other HTTP client and databases or applications on the HTTP server.
- There are many interfaces and classes in the Servlet API such as Servlet, GenericServlet, HttpServlet, ServletRequest, ServletResponse, etc.
- Using Servlets, we can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.

## What is servlet?

Servlet can be described in many ways, depending on the context some of these as follow-

- Servlet is a technology which is used to create a server side web application.
- Servlet is an API that provides many interfaces and classes to create web applications.

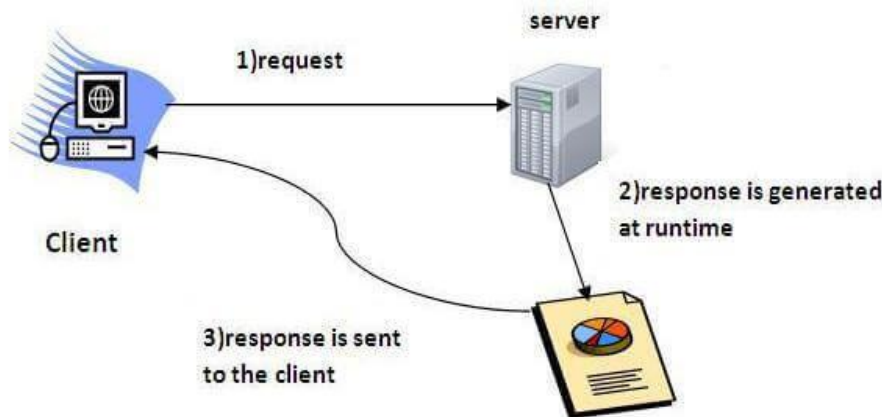
- Servlet is an interface that must be implemented for creating any Servlet.
- Servlet is a class that extends the capabilities of the servers and responds to the incoming requests. It can respond to any requests that come from client.
- Servlet is a web component that is deployed on the server to create a dynamic web page.

## Characteristics or Properties of Servlet:

- Servlet technology is robust and scalable because of java language.
- Servlets execute within the address space of a Web server. It is not necessary to create a separate process to handle each client request.
- Servlets are platform-independent because they are written in Java.
- Servlets are trusted: Java security manager on the server enforces a set of restrictions to protect the resources on a server machine.
- The full functionality of the Java class libraries is available to a servlet. Therefore servlet can communicate with applets, databases, or other software via the sockets and RMI mechanisms.
- Servlet performance is significantly better as compared to other server applications.

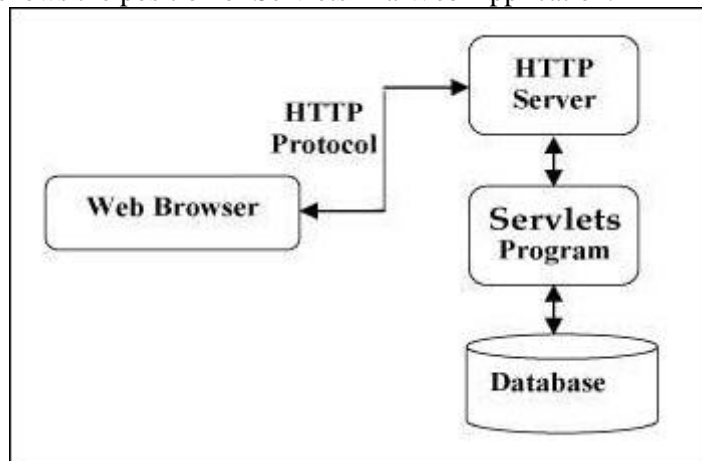
## Working of Servlet:

Following diagram shows working of servlet.



## Servlets Architecture:

The following diagram shows the position of Servlets in a Web Application.



## Servlets Tasks:

Servlets perform the following major tasks –

- Servlet can read the explicit data sent by the clients (browsers). This includes an HTML form on a Web page or it could also come from an applet or a custom HTTP client program.
- Servlet can read the implicit HTTP request data sent by the clients (browsers). This includes cookies, media types and compression schemes the browser understands.
- Servlet process the data and generate the results as per client request. This process may require access to a database, executing an RMI or CORBA call, invoking a Web service, or computing the response directly.
- Servlet also capable to send the explicit data (i.e., the document) to the clients (browsers). This document can be sent in a variety of formats, including text (HTML or XML), binary (GIF images), Excel, etc.

## **Theory Assignment No: 06**

- 1) What is Applet? Explain life cycle of Applet.
- 2) What are the steps to perform event handling?
- 3) What is Java AWT? Explain AWT hierarchy.
- 4) How to create Frame in Java awt?
- 5) Explain following Java awt controls-
  - 1) Label
  - 2) Button
  - 3) Checkbox
  - 4) Choice (Dropdown Boxes)
  - 5) List
  - 6) TextField
  - 7) TextArea
- 6) What is Java Swing?
- 7) Write the difference between Java awt and Javax Swing.
- 8) Explain following Java Swing controls-
  - 1) JLabel
  - 2) JButton
  - 3) JCheckbox
  - 4) JCheckBox
  - 5) JComboBox
  - 6) JTextField
  - 7) JTextArea
  - 8) JList
- 9) What is Java Servlet? List out characteristics or properties of servlet.
- 10) List out different tasks performed by servlet.

## **Practical Assignment No: 08**

- 1) Write a program to implement Applet containing different Graphics objects.
- 2) Write a program to implement Applet showing smiley face.
- 3) Write a program to implement Applet showing event handling that changes background color of applet
- 4) Write a program to implement Applet that display an image.
- 5) Write a program to implement Applet that moves image (Animation using Applet)
- 6) Write a program that add different AWT controls onto Applet.
- 7) Write a program that demonstrate use of different awt controls on Frame.
- 8) Write an awt GUI application that finds addition, subtraction, multiplication and division.
- 9) Write a program that shows awt GUI application that display selected item of First List into Second List and vice versa by clicking on respective buttons.
- 10) Write a program that shows java swing application with different java swing controls.
- 11) Write any program in java swing that shows use of multiple JFrames.