

# Applet

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 class extends `java.applet.Applet` class.

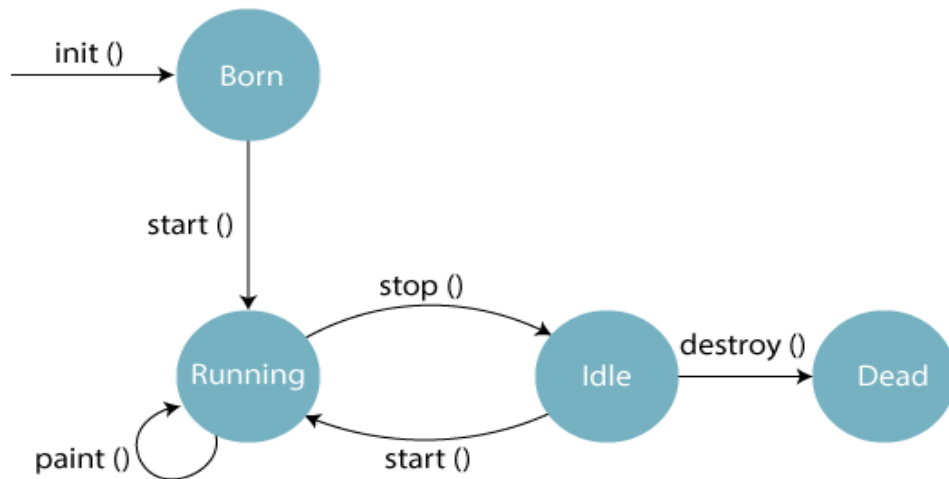
An Applet class does not have any `main()` method. It is viewed using JVM. The JVM can use either a plug-in of the Web browser or a separate runtime environment to run an applet application. JVM creates an instance of the applet class and invokes `init()` method to initialize an Applet.

**Note:** Java Applet is deprecated since Java 9. It means Applet API is no longer considered important.

## Lifecycle of Java Applet

Following are the stages in Applet:-

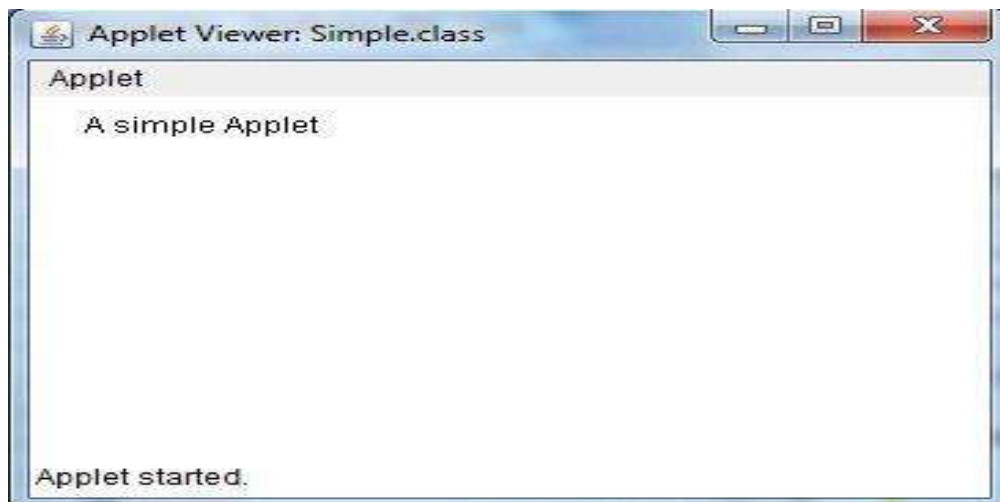
1. Applet is initialized.
2. Applet is started
3. Applet is painted.
4. Applet is stopped.
5. Applet is destroyed.



## A Simple Applet Program:-

```
import java.awt.*;  
import java.applet.*;  
public class Simple extends Applet  
{  
    public void paint(Graphics g)  
    {  
        g.drawString("A simple Applet", 20, 20);  
    }  
}
```

## OUTPUT:-



Every Applet application must import two packages - java.awt and java.applet.

java.awt.\* imports the Abstract Window Toolkit (AWT) classes. Applets interact with the user (either directly or indirectly) through the AWT. The AWT contains support for a window-based, graphical user interface.

java.applet.\* imports the applet package, which contains the class Applet. Every applet that you create must be a subclass of Applet class.

The class in the program must be declared as public, because it will be accessed by code that is outside the program. Every Applet application must declare a paint() method. This method is defined by AWT class and must be overridden by the applet. The paint() method is called each time when an applet needs to redisplay its output. Another important thing to notice about applet application is that, execution of an applet does not begin at main() method. In fact an applet application does not have any main() method.

## **Advantages of Applets:-**

1. It takes very less response time as it works on the client side.
2. It can be run on any browser which has JVM running in it.

## **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.

## **An Applet Skeleton:-**

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

**i). init()** : init() is the first method to be called. This is where variable are initialized. This method is called only once during the runtime of applet.

**ii).start()** : start() method is called after init(). This method is called to restart an applet after it has been stopped.

**iii).stop()** : stop() method is called to suspend thread that does not need to run when applet is not visible.

**iv).destroy()** : destroy() method is called when your applet needs to be removed completely from memory.

**Note:** The stop() method is always called before destroy() method.

## Example of an Applet Skeleton:-

```
import java.awt.*;
import java.applet.*;
public class AppletTest extends Applet
{
    public void init()
    {
        //initialization
    }
    public void start ()
    {
        //start or resume execution
    }
    public void stop()
    {
        //suspend execution
    }
    public void destroy()
    {
        //perform shutdown activity
    }
    public void paint (Graphics g)
    {
        //display the content of window
    }
}
```

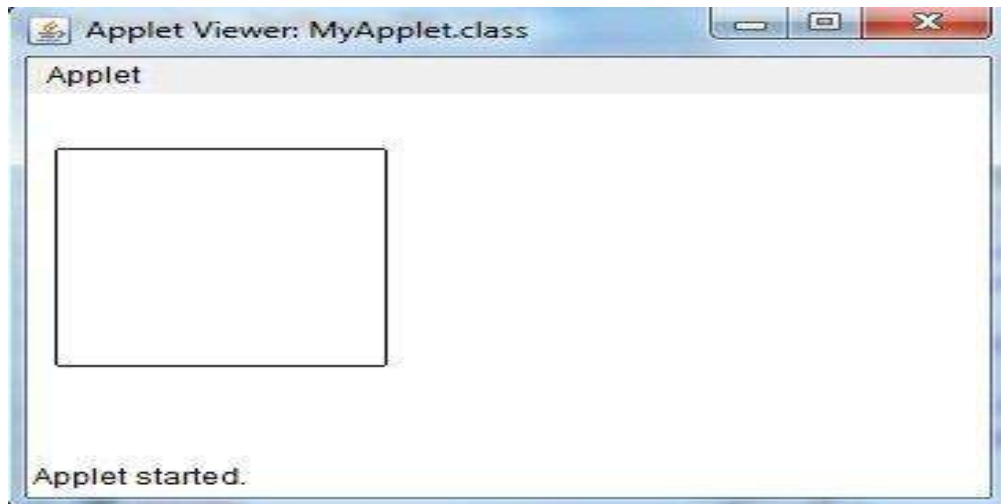
## Example of an Applet :-

```
import java.applet.*;
import java.awt.*;
public class MyApplet extends Applet
{
    int height, width;
    public void init()
    {
        height = getSize().height;
        width = getSize().width;
        setName("MyApplet");
    }
    public void paint(Graphics g)
    {

```

```
g.drawRoundRect(10, 30, 120, 120, 2, 3);  
}  
}
```

**OUTPUT :-**



## Parameter in Applet

User-defined Parameter can be applied in applet using tags. Each tag has a name and value attribute.

In an applet code, applet can refer to a parameter by its name and then find its value.

The two most important things to handle and set up the parameter is the tag in the HTML document and an applet code to parse this parameter.

init() method is used to get hold of the parameters which is defined in the tags. And getParameter() method is used for getting the parameters.

In Applet, Parameters are passed on applet when it is loaded.

## Example :

param.java

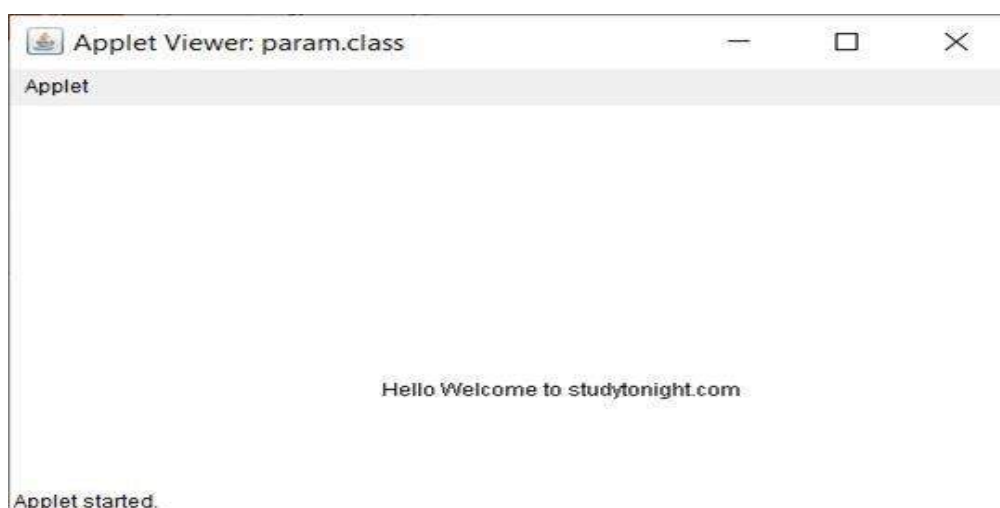
```
import java.applet.*;  
import java.awt.*;  
public class param extends Applet  
{
```

```
String str;  
public void init()  
{  
    str=getParameter("pname");  
    if (str == null)  
        str = "Welcome to studytonight.com";  
    str = "Hello " + str;  
}  
public void paint(Graphics g)  
{  
    g.drawString(str, 200, 200);  
}  
}
```

OUTPUT



```
C:\Windows\System32\cmd.exe - appletviewer param.html  
D:\Studytonight\Applet>javac param.java  
D:\Studytonight\Applet>appletviewer param.html
```



## How to run an Applet Program

An Applet program is compiled in the same way as you have been compiling your console programs. However there are two ways to run an applet.

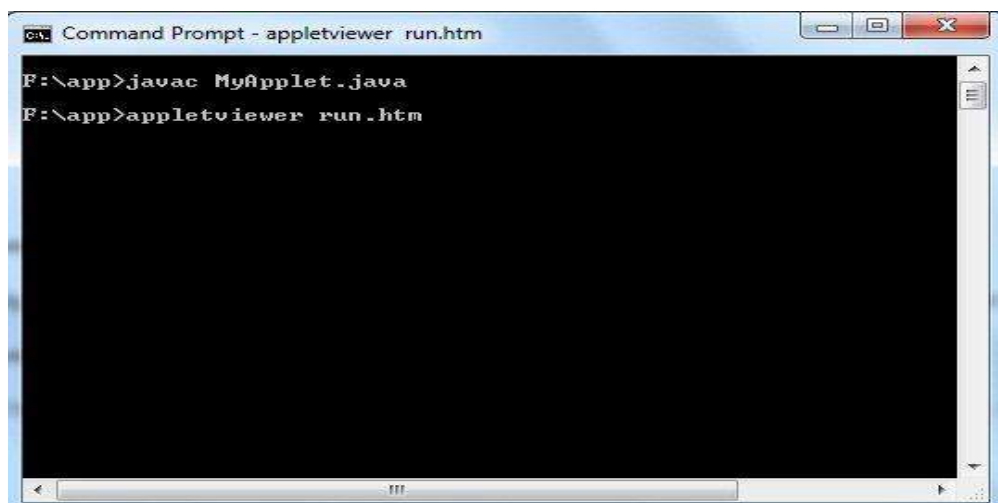
- Executing the Applet within Java-compatible web browser.
- Using an Applet viewer, such as the standard tool, applet viewer. An applet viewer executes your applet in a window

For executing an Applet in an web browser, create short HTML file in the same directory. Inside body tag of the file, include the following code. (applet tag loads the Applet class)

## Running Applet using Applet Viewer

To execute an Applet with an applet viewer, write short HTML file as discussed above. If you name it as run.htm, then the following command will run your applet program.

```
f:/>appletviewer run.htm
```



## Graphics in Applet

In Applet, java.awt.Graphicsclass provides methods for using graphics.

Below are the Methods of the Graphics class.

1. public abstract void drawString(String str, int x, int y) - Used to draw specified string.
2. public void drawRect(int x, int y, int width, int height) - Used to draw a rectangle of specified width and height.
3. public abstract void fillRect(int x, int y, int width, int height) - Used to draw a rectangle with a default colour of specified width and height.
4. public abstract void drawOval(int x, int y, int width, int height) - Used to draw oval of specified width and height.
5. public abstract void fillOval(int x, int y, int width, int height) - Used to draw oval with a default colour of specified width and height.
6. public abstract void drawLine(int x1, int y1, int x2, int y2) - Used for drawing lines between the point (x1, y1) and (x2, y2).
7. public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer) - Used for drawing a specified image.
8. public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle) - Used for drawing a circular arc.
9. public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle) - Used for filling circular arc.
10. public abstract void setColor(Color c) - Used to set a colour to the object.
11. public abstract void setFont(Font font) - Used to set font.

## EXAMPLE

```
GraphicsDemo1.java
import java.applet.Applet;
import java.awt.*;
public class GraphicsDemo1 extends Applet
{
    public void paint(Graphics g)
```



```

{
g.setColor(Color.black);
g.drawString("Welcome to studytonight",50, 50);
g.setColor(Color.blue);
g.fillOval(170,200,30,30);
g.drawArc(90,150,30,30,30,270);
g.fillArc(270,150,30,30,0,180);
g.drawLine(21,31,20,300);
g.drawRect(70,100,30,30);
g.fillRect(170,100,30,30);
g.drawOval(70,200,30,30);
}
}

```

OUTPUT

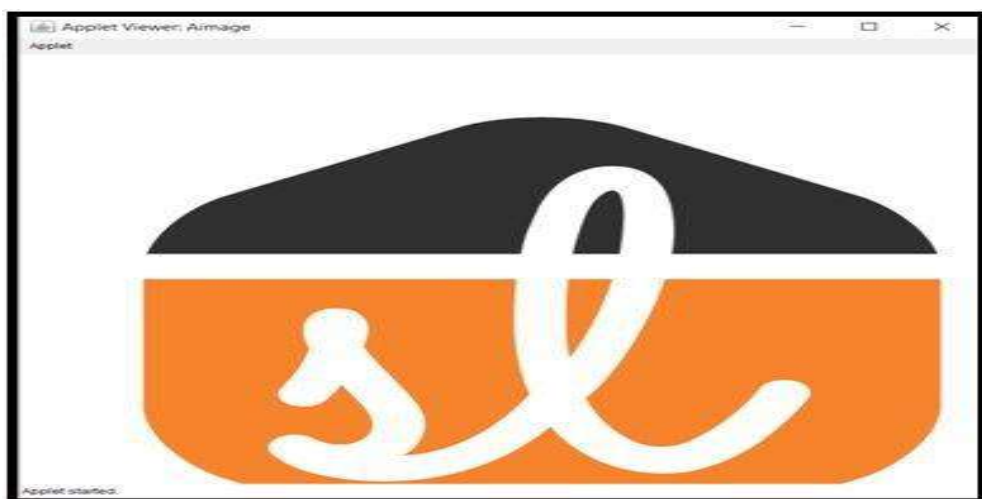
```

C:\Windows\System32\cmd.exe - appletviewer Aimage.html

D:\Studytonight\Applet>javac Aimage.java

D:\Studytonight\Applet>appletviewer Aimage.h

```



## EventHandling in Applet

In Applet we can also perform event handling.

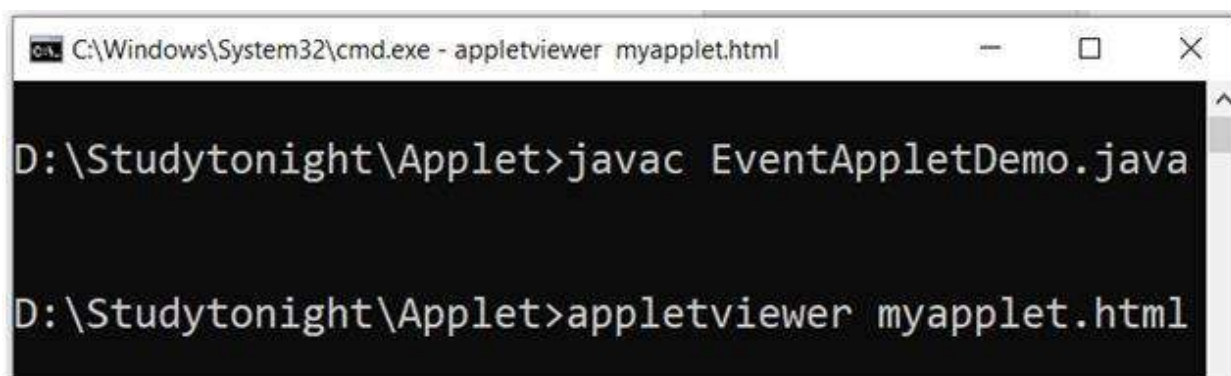
Below is an example of event handling which prints a message when clicked on the button.

### Example

EventAppletDemo.java

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class EventAppletDemo extends Applet implements ActionListener
{
    Button b1;
    TextField tf1;
    public void init()
    {
        tf1=new TextField();
        tf1.setBounds(30,40,200,20);
        b1=new Button("Click");
        b1.setBounds(80,150,60,50);
        add(b1);
        add(tf1);
        b1.addActionListener(this);
        setLayout(null);
    }
    public void actionPerformed(ActionEvent e)
    {
        tf1.setText("Welcome to studytonight");
    }
}
```

### OUTPUT



```
C:\Windows\System32\cmd.exe - appletviewer myapplet.html

D:\Studytonight\Applet>javac EventAppletDemo.java

D:\Studytonight\Applet>appletviewer myapplet.html
```



## Animation in Applet

In Applet, we can also create animations in a program using a gif image. Below is an example in which drawImage() method is used which is of Graphics class, it is used for displaying an image.

**Note:** Download a gif file for the below example

### Example:-

AnimationDemo.java

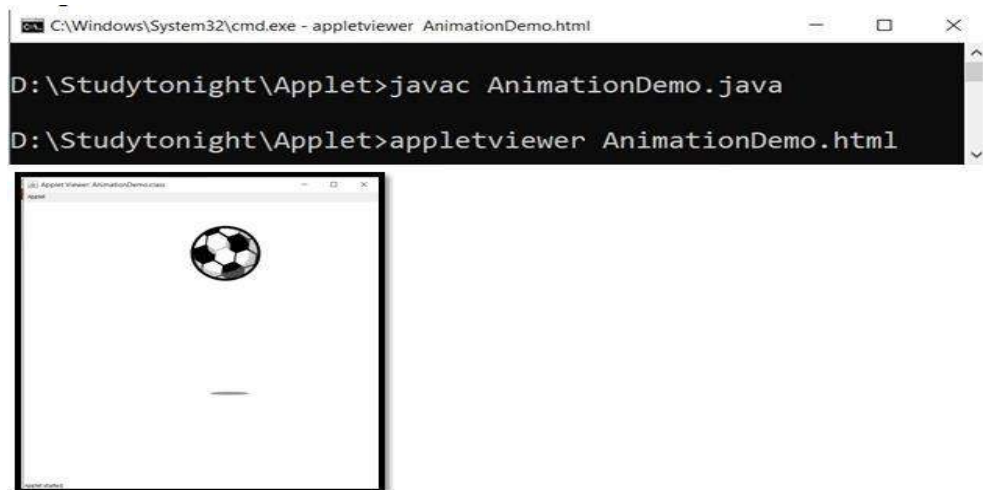
```
import java.awt.*;
import java.applet.*;
public class AnimationDemo extends Applet
{
    Image p;
    public void init()
    {
        p = getImage(getDocumentBase(),"ball.gif");
    }
    public void paint(Graphics g)
    {
        for(int i=0;i<500;i++)
        {
            g.drawImage(p, i,50, this);
            try
            {
```

```

Thread.sleep(100);
}
catch(Exception e)
{}
}
}
}
}

```

## OUTPUT



## JApplet class

In Java, JApplet is the public class of swing. JApplet extends the class in java.applet.Applet. JApplet generates a bytecode with the help of JVM or the Applet viewer. JApplet can be written in any programming language and then can be compiled for Java Byte code.

## EXAMPLE

JAppletDemo.java

```

import java.applet.*;
import javax.swing.*;
import java.awt.event.*;
public class JAppletDemo extends JApplet implements ActionListener
{
    JButton b;
    JTextField t;
    public void init()
    {
        t=new JTextField();
    }
}

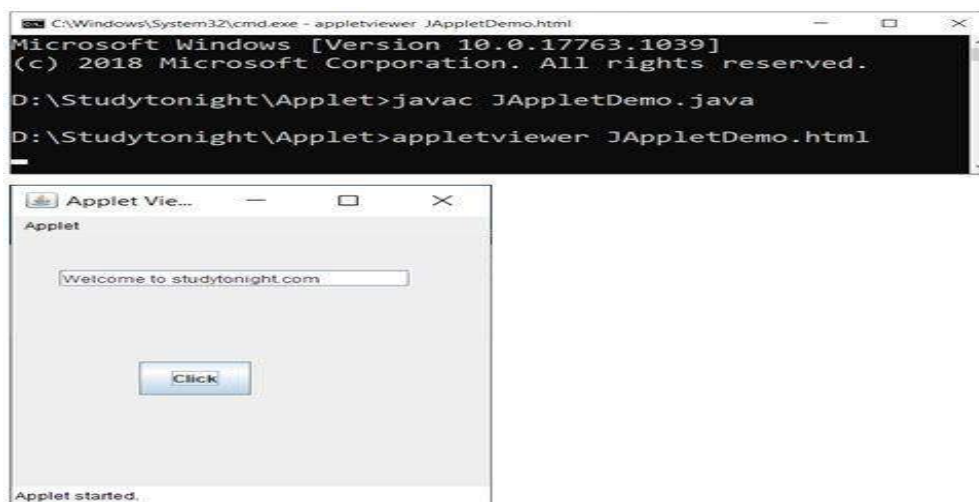
```

```

t.setBounds(30,40,220,20);
b=new JButton("Click");
b.setBounds(80,150,70,40);
add(b);
add(t);
b.addActionListener(this);
setLayout(null);
}
public void actionPerformed(ActionEvent e)
{
t.setText("Welcome to studytonight.com");
}
}

```

## OUTPUT



## Painting in Applet

Below is an example of Painting using mouseDragged() method of MouseMotionListener in Applet.

## EXAMPLE

PaintingDemo.java

```

import java.awt.*;
import java.awt.event.*;
import java.applet.*;
public class PaintingDemo extends Applet implements
MouseMotionListener
{

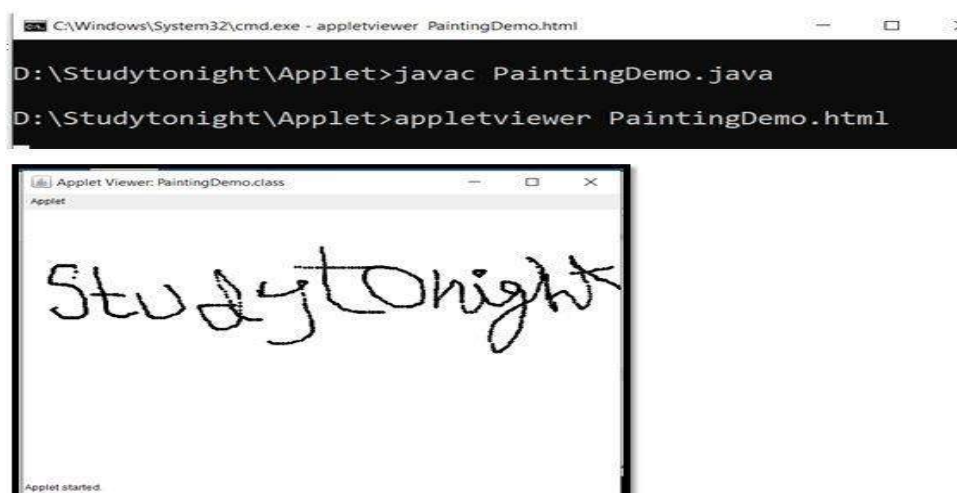
```

```

public void init()
{
    addMouseListener(this);
    setBackground(Color.white);
}
public void mouseDragged(MouseEvent me)
{
    Graphics g=getGraphics();
    g.setColor(Color.black);
    g.fillOval(me.getX(),me.getY(),5,5);
}
public void mouseMoved(MouseEvent me)
{}
}

```

## OUTPUT



## Analog Clock in Applet

In java, Applet can be used for creating an Analog Clock. For creating a program for the Analog clock, java.applet, java.awt, java.util, and java.text package are imported. Date and Time functions are also used. Math functions play an important role in creating an Analog Clock. below is a program for creating an Analog Clock

## EXAMPLE

AnalogDemo1.java

```

import java.applet.*;
import java.awt.*;

```

```

import java.util.*;
import java.text.*;
public class AnalogDemo1 extends Applet implements Runnable
{
    int width, height;
    Thread t = null;
    boolean threadSuspended;
    int hours=0, minutes=0, seconds=0;
    String timeString = "";
    public void init()
    {
        width = getSize().width;
        height = getSize().height;
        setBackground( Color.black );
    }
    public void start()
    {
        if ( t == null )
        {
            t = new Thread( this );
            t.setPriority( Thread.MIN_PRIORITY );
            threadSuspended = false;
            t.start();
        }
        else
        {
            if ( threadSuspended )
            {
                threadSuspended = false;
                synchronized( this )
                {
                    notify();
                }
            }
        }
    }
    public void stop()
    {
        threadSuspended = true;
    }
    public void run() {
        try {
            while (true) {
                Calendar cal = Calendar.getInstance();

```

```

hours = cal.get( Calendar.HOUR_OF_DAY );
if ( hours > 12 ) hours -= 12;
minutes = cal.get( Calendar.MINUTE );
seconds = cal.get( Calendar.SECOND );
SimpleDateFormat formatter = new SimpleDateFormat( "hh:mm:ss",
Locale.getDefault() );
Date date = cal.getTime();
timeString = formatter.format( date );
if ( threadSuspended ) {
synchronized( this ) {
while ( threadSuspended ) {
wait();
}
}
}
repaint();
t.sleep( 1000 );
}
}
catch (Exception e) { }
}

void drawHand( double angle, int radius, Graphics g ) {
angle -= 0.5 * Math.PI;
int x = (int)( radius*Math.cos(angle) );
int y = (int)( radius*Math.sin(angle) );
g.drawLine( width/2, height/2, width/2 + x, height/2 + y );
}

void drawWedge( double angle, int radius, Graphics g ) {
angle -= 0.5 * Math.PI;
int x = (int)( radius*Math.cos(angle) );
int y = (int)( radius*Math.sin(angle) );
angle += 2*Math.PI/3;
int x2 = (int)( 5*Math.cos(angle) );
int y2 = (int)( 5*Math.sin(angle) );
angle += 2*Math.PI/3;
int x3 = (int)( 5*Math.cos(angle) );
int y3 = (int)( 5*Math.sin(angle) );
g.drawLine( width/2+x2, height/2+y2, width/2 + x, height/2 + y );
g.drawLine( width/2+x3, height/2+y3, width/2 + x, height/2 + y );
g.drawLine( width/2+x2, height/2+y2, width/2 + x3, height/2 + y3 );
}

public void paint( Graphics g ) {
g.setColor( Color.pink );
drawWedge( 2*Math.PI * hours / 12, width/5, g );
}

```

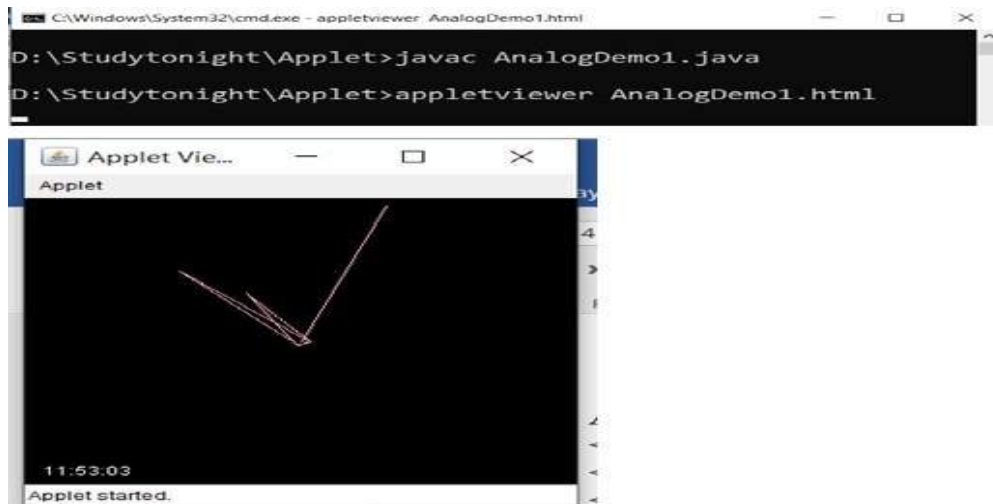


```

drawWedge( 2*Math.PI * minutes / 60, width/3, g );
drawHand( 2*Math.PI * seconds / 60, width/2, g );
g.setColor( Color.white );
g.drawString( timeString, 10, height-10 );
}
}

```

## OUTPUT



## Digital Clock in Applet

In java, Applet can be used for creating a Digital Clock. For creating a program for the digital clock, java.applet, java.awt, java.util, and java.text package are imported. Date and Time functions are also used. below is a program for creating a Digital Clock.

## EXAMPLE

DigitalDemo1.java

```

import java.applet.*;
import java.awt.*;
import java.util.*;
import java.text.*;
public class DigitalClockDemo1 extends Applet implements Runnable
{
    Thread t = null;
    int h=0, m=0, s=0;
    String timeString = "";
    public void init()
    {

```

```

setBackground( Color.black);
}
public void start()
{
t = new Thread( this );
t.start();
}
public void run()
{
try
{
while (true)
{
Calendar cal = Calendar.getInstance();
h = cal.get( Calendar.HOUR_OF_DAY );
if ( h> 12 ) h -= 12;
m = cal.get( Calendar.MINUTE );
s = cal.get( Calendar.SECOND );
SimpleDateFormat f = new SimpleDateFormat("hh:mm:ss");
Date date = cal.getTime();
timeString = f.format( date );
repaint();
t.sleep( 1000 );
}
}
catch (Exception e) { }
}
public void paint( Graphics g )
{
g.setColor( Color.white );
g.drawString( timeString, 50, 50 );
}
}

```

## OUTPUT

```
C:\Windows\System32\cmd.exe - appletviewer DigitalClockDemo1.html
D:\Studytonight\Applet>javac DigitalClockDemo1.java
D:\Studytonight\Applet>appletviewer DigitalClockDemo1.html
```



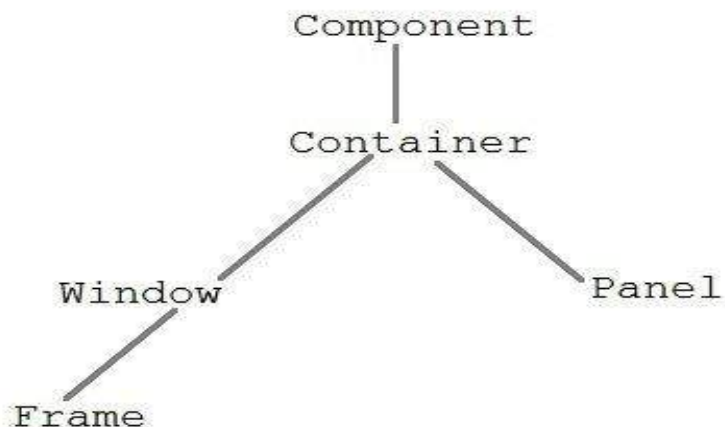
# Abstract Window Toolkit(AWT)

Java AWT is an API that contains large number of classes and methods to create and manage graphical user interface ( GUI ) applications. The AWT was designed to provide a common set of tools for GUI design that could work on a variety of platforms. The tools provided by the AWT are implemented using each platform's native GUI toolkit, hence preserving the look and feel of each platform. This is an advantage of using AWT. But the disadvantage of such an approach is that GUI designed on one platform may look different when displayed on another platform that means AWT component are platform dependent.

AWT is the foundation upon which Swing is made i.e Swing is a improved GUI API that extends the AWT. But now a days AWT is merely used because most GUI Java programs are implemented using Swing because of its rich implementation of GUI controls and light-weighted nature.

## Java AWT Hierarchy

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



## Component class

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

and the currently selected text font.

## **Container**

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

## **Panel**

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

## **Window class**

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

## **Frame**

Frame is a subclass of Window and have resizing canvas. It is a container that contain several different components like button, title bar, textfield, label etc. In Java, most of the AWT applications are created using Frame window. Frame class has two different constructors.

Frame() throws HeadlessException

Frame(String title) throws HeadlessException

## **Creating a Frame**

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

- 1).By Instantiating Frame class
- 2).By extending Frame class

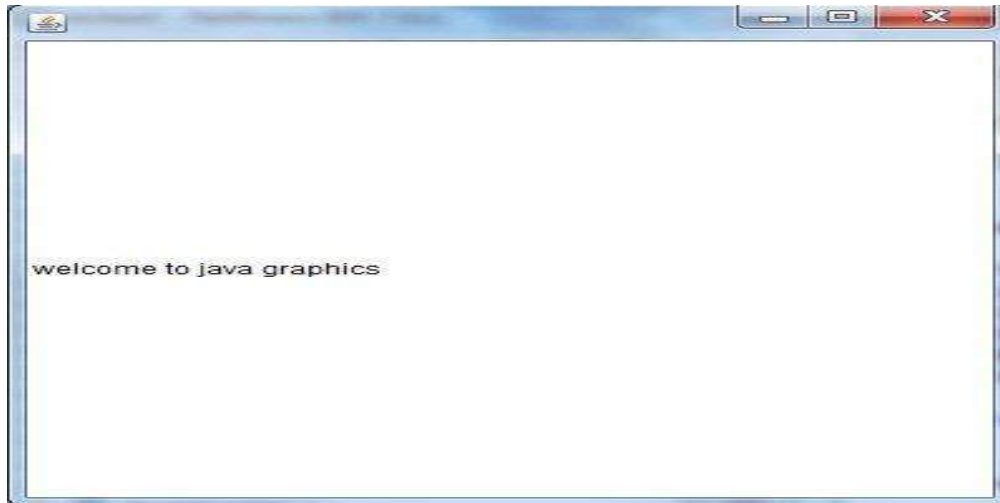
## **Creating Frame Window by Instantiating Frame class**

```
import java.awt.*;  
public class Testawt
```

```

{
Testawt()
{
Frame fm=new Frame(); //Creating a frame
Label lb = new Label("welcome to java graphics"); //Creating a label
fm.add(lb); //adding label to the frame
fm.setSize(300, 300); //setting frame size.
fm.setVisible(true); //set frame visibilty true
}
public static void main(String args[])
{
Testawt ta = new Testawt();
}
}

```



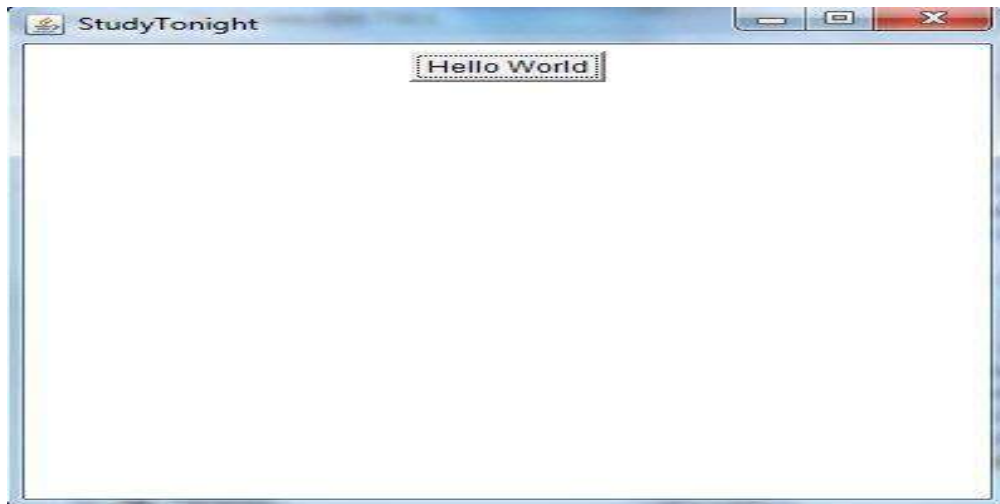
## Creating Frame window by extending Frame class

```

package testawt;
import java.awt.*;
import java.awt.event.*;
public class Testawt extends Frame
{
public Testawt()
{
Button btn=new Button("Hello World");
add(btn); //adding a new Button.
setSize(400, 500); //setting size.
setTitle("StudyTonight"); //setting title.
setLayout(new FlowLayout()); //set default layout for frame.
setVisible(true); //set frame visibilty true.
}
}

```

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



## Points to Remember:

1).While creating a frame (either by instantiating or extending Frame class), Following two attributes are must for visibility of the frame:

- setSize(int width, int height);
- setVisible(true);

2).When you create other components like Buttons, TextFields, etc. Then you need to add it to the frame by using the method - add(Component's Object);

3).You can add the following method also for resizing the frame -  
setResizable(true);

## AWT Button

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

## AWT Button Class Declaration

public class Button extends Component implements Accessible

## Example

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

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

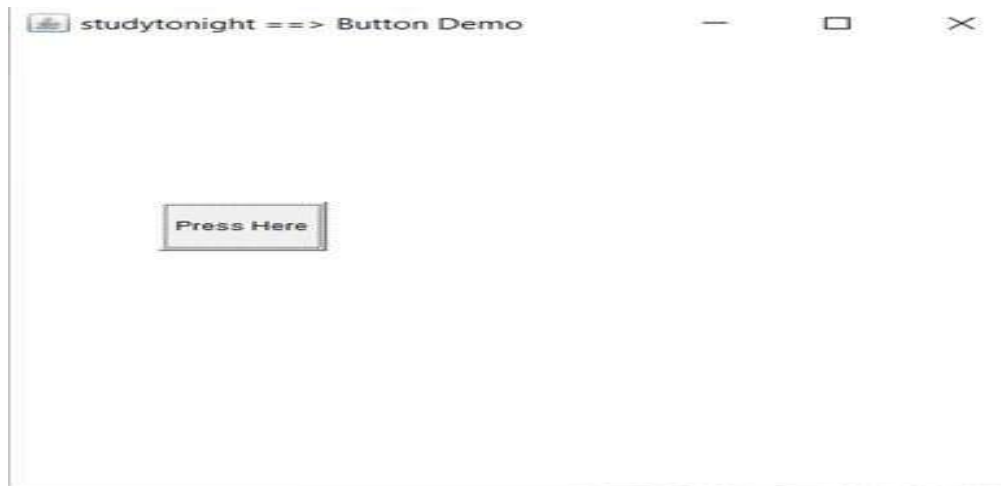


The screenshot shows a Windows command prompt window with the title bar "C:\Windows\System32\cmd.exe - java ButtonDe...". The command prompt displays the following commands and their output:

```
D:\Studytonight>javac ButtonDemo1.java
D:\Studytonight>java ButtonDemo1
```

A cursor is visible on the line following the second command.





## AWT Label

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

### Label Declaration:

```
public class Label extends Component implements Accessible
```

### Example

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

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

```
I_Frame.setLayout(null);  
I_Frame.setVisible(true);  
}  
}
```



```
C:\Windows\System32\cmd.exe - java LabelDemo1  
  
D:\Studytonight>javac LabelDemo1.java  
  
D:\Studytonight>java LabelDemo1
```



## AWT Textfield

In Java, AWT contains a `TextField` Class. It is used for displaying single line text.

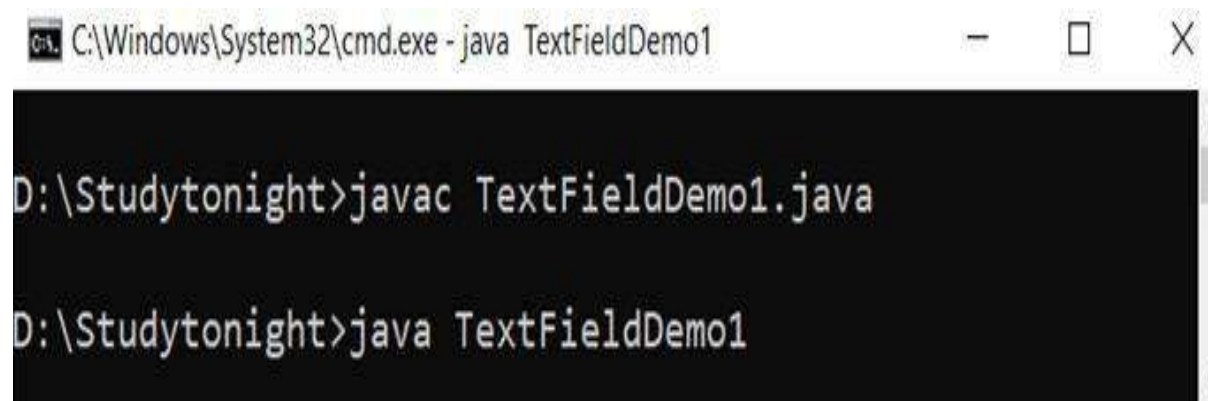
### TextField Declaration:

```
public class TextField extends TextComponent
```

### Example

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

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



The screenshot shows a Windows command prompt window titled "C:\Windows\System32\cmd.exe - java TextFieldDemo1". The prompt is at "D:\Studytonight>". The user has entered the command "javac TextFieldDemo1.java" and the next line shows "D:\Studytonight>java TextFieldDemo1".



## AWT TextArea

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

### TextArea Declaration:

```
public class TextArea extends TextComponent
```

### Example

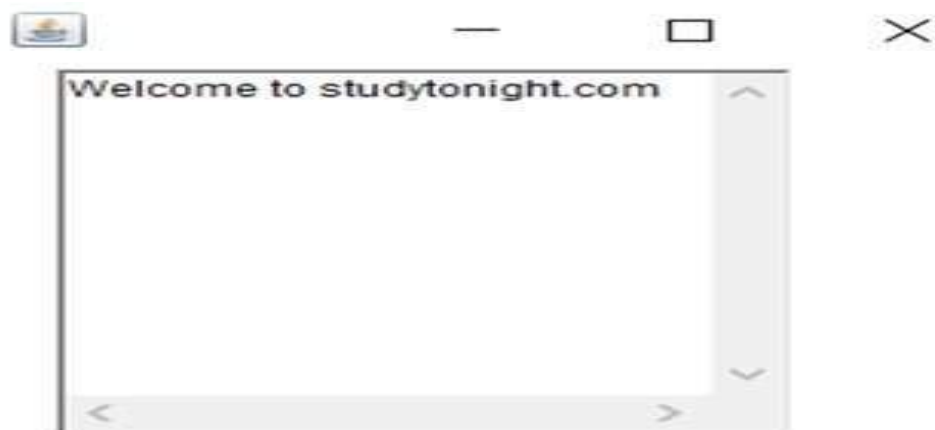
In this example, we are creating a `TextArea` that is used to display multiple-line text string and allows text editing as well.

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

```
{  
new TextAreaDemo1();  
}  
}
```



```
C:\Windows\System32\cmd.exe - java TextAreaDemo1  
D:\Studytonight>javac TextAreaDemo1.java  
D:\Studytonight>java TextAreaDemo1
```



## AWT Checkbox

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

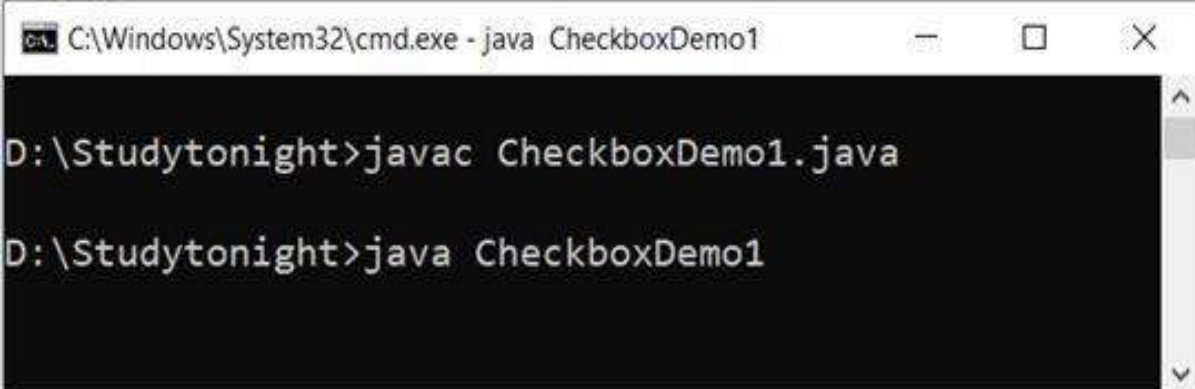
### Checkbox Syntax

```
public class Checkbox extends Component implements ItemSelectable,  
Accessible
```

## Example

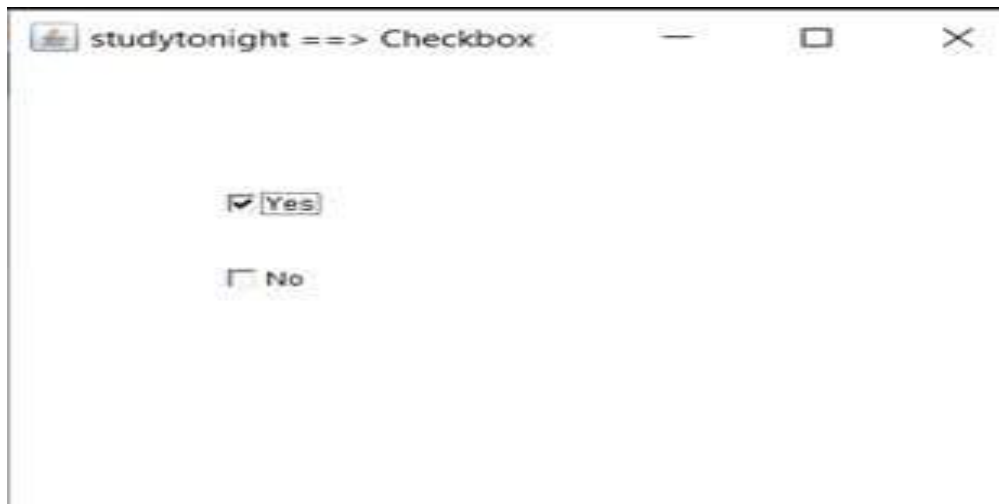
In this example, we are creating checkbox that are used to get user input. If checkbox is checked it returns true else returns false.

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



The screenshot shows a Windows command prompt window titled "C:\Windows\System32\cmd.exe - java CheckboxDemo1". The window has standard Windows window controls (minimize, maximize, close). The command prompt shows the following commands and output:

```
D:\Studytonight>javac CheckboxDemo1.java
D:\Studytonight>java CheckboxDemo1
```



## AWT CheckboxGroup

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

### CheckboxGroup Declaration:

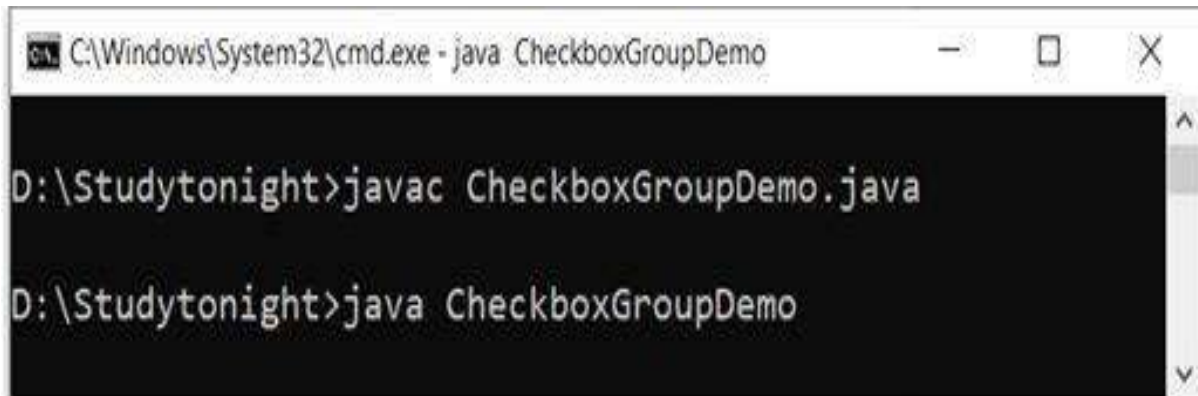
```
public class CheckboxGroup extends Object implements Serializable
```

### Example

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

```
import java.awt.*;
public class CheckboxGroupDemo
{
    CheckboxGroupDemo(){
        Frame ck_groupf= new Frame("studytonight ==>CheckboxGroup");
        CheckboxGroupobj = new CheckboxGroup();
        Checkbox ckBox1 = new Checkbox("Yes", obj, true);
        ckBox1.setBounds(100,100, 50,50);
        Checkbox ckBox2 = new Checkbox("No", obj, false);
        ckBox2.setBounds(100,150, 50,50);
        ck_groupf.add(ckBox1);
        ck_groupf.add(ckBox2);
    }
}
```

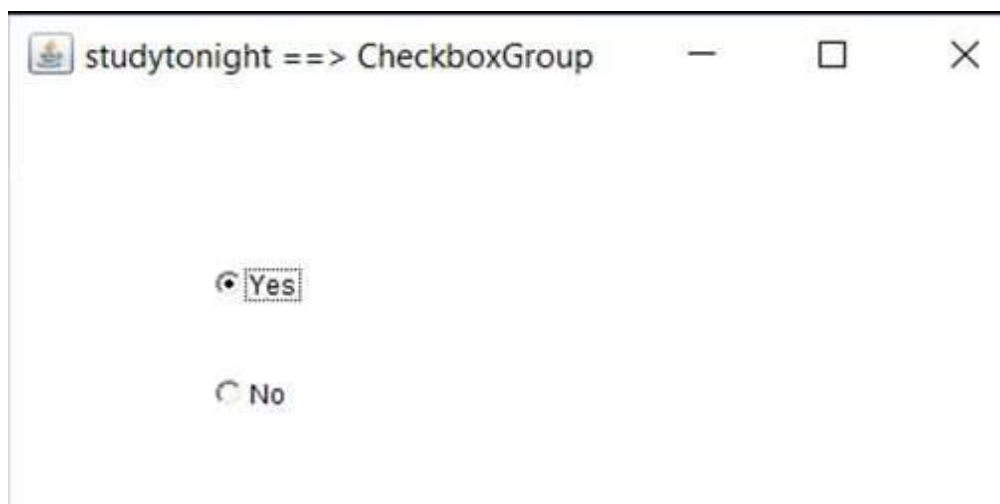
```
ck_groupf.setSize(400,400);
ck_groupf.setLayout(null);
ck_groupf.setVisible(true);
}
public static void main(String args[])
{
new CheckboxGroupDemo();
}
}
```



```
C:\Windows\System32\cmd.exe - java CheckboxGroupDemo

D:\Studytonight>javac CheckboxGroupDemo.java

D:\Studytonight>java CheckboxGroupDemo
```



## AWT Choice

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



## Choice Declaration:

public class Choice extends Component implements ItemSelectable,  
Accessible

## Example

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

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

```
C:\Windows\System32\cmd.exe - java ChoiceDemo

D:\Studytonight>javac ChoiceDemo.java

D:\Studytonight>java ChoiceDemo
```



## AWT List

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

### List Declaration:

```
public class List extends Component implements ItemSelectable,
Accessible
```

### Example

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

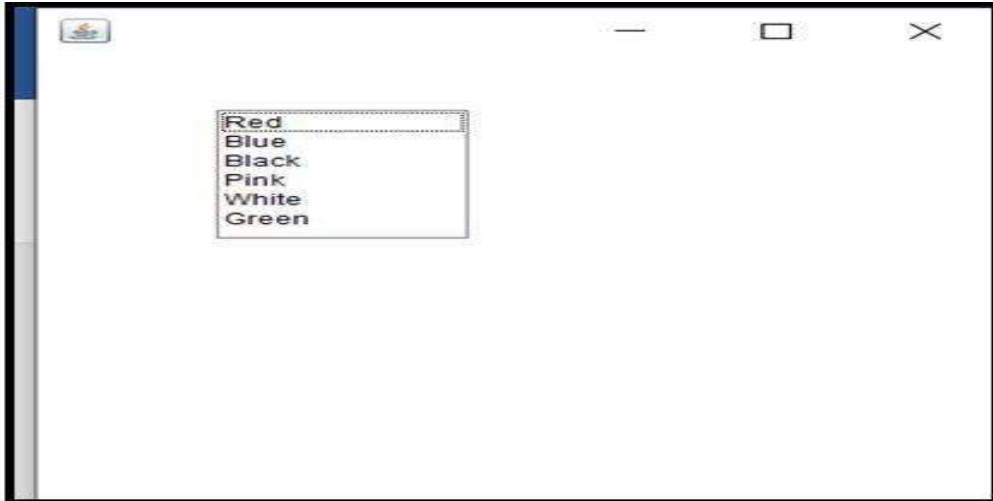
```
import java.awt.*;
public class ListDemo
{
    ListDemo()
    {
        Frame list_f= new Frame();
        List obj=new List(6);
        obj.setBounds(80,80, 100,100); obj.add("Red");
        obj.add("Blue");
        obj.add("Black");
        obj.add("Pink");
        obj.add("White");
        obj.add("Green");
        list_f.add(obj);
        list_f.setSize(400,400);
        list_f.setLayout(null);
        list_f.setVisible(true);
    }
    public static void main(String args[])
    {
        new ListDemo();
    }
}
```



The screenshot shows a Windows command prompt window titled "C:\Windows\System32\cmd.exe - java ListDemo". The window has standard Windows window controls (minimize, maximize, close) in the top right corner. The command prompt shows the following commands and their output:

```
D:\Studytonight>javac ListDemo.java

D:\Studytonight>java ListDemo
```



## AWT Canvas

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

### Declaration:

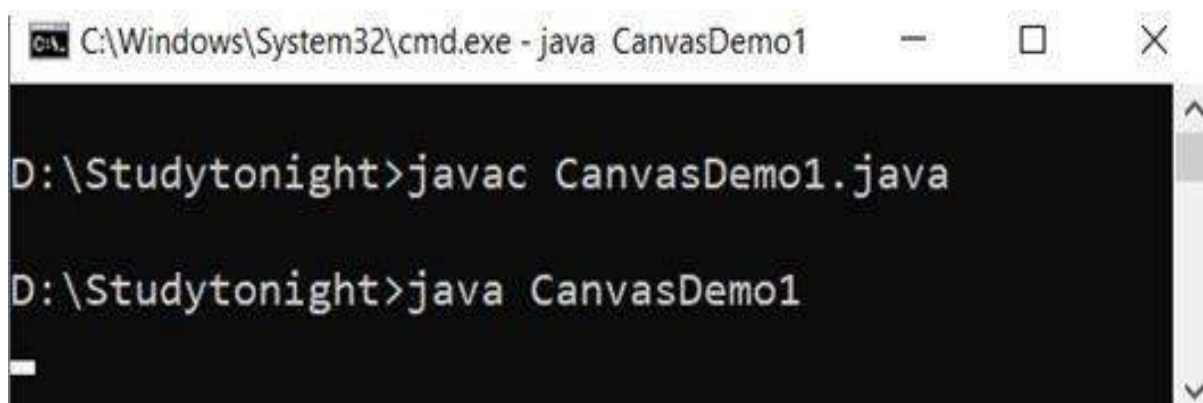
public class Canvas extends Component implements Accessible

### Example

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

```
import java.awt.*;
public class CanvasDemo1
{
    public CanvasDemo1()
    {
        Frame canvas_f= new Frame("studytonight ==> Canvas");
        canvas_f.add(new CanvasDemo());
        canvas_f.setLayout(null);
        canvas_f.setSize(500, 500);
        canvas_f.setVisible(true);
    }
    public static void main(String args[])
    {
        new CanvasDemo1();
    }
}
```

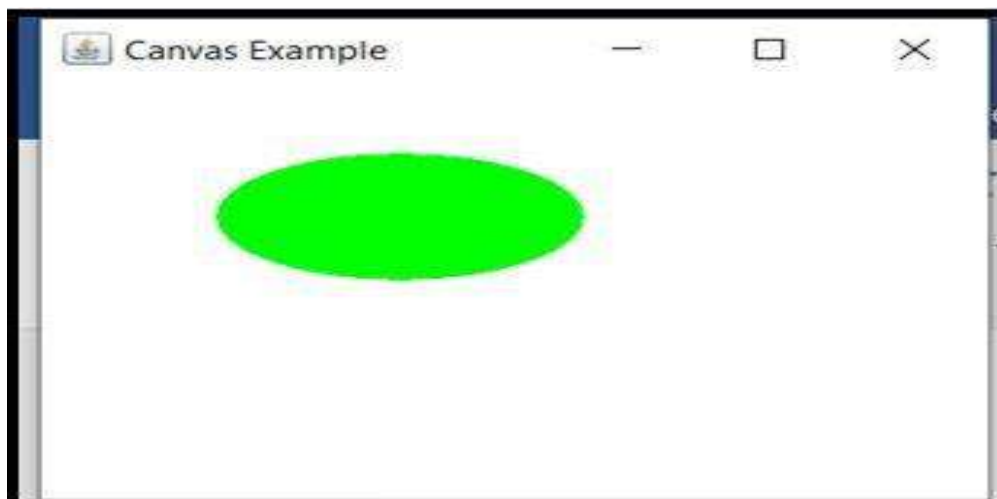
```
} class CanvasDemo extends Canvas
{
public CanvasDemo() {
setBackground (Color.WHITE);
setSize(300, 200);
}
public void paint(Graphics g)
{
g.setColor(Color.green);
g.fillOval(80, 80, 150, 75);
}
}
```



```
C:\Windows\System32\cmd.exe - java CanvasDemo1

D:\Studytonight>javac CanvasDemo1.java

D:\Studytonight>java CanvasDemo1
```



## AWT Canvas

In Java, AWT contains a MenuItem and Menu Class. MenuItem is used for

adding Lable in Menu. The menu is used to create a drop-down of menu components

## MenuItem declaration

public class MenuItem extends MenuComponent implements Accessible

## Menu declaration

public class Menu extends MenuItem implements MenuContainer,  
Accessible

## Example

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

```
import java.awt.*;
class MenuDemo1
{
    MenuDemo1()
    {
        Frame menu_f= new Frame("studytonight ==> Menu and MenuItem
        Demo");
        MenuBarmenu_bar=new MenuBar();
        Menu menu11=new Menu("Menu");
        Menu sub_menu1=new Menu("Sub Menu ==>");
        MenuItem a1=new MenuItem("Red");
        MenuItem a2=new MenuItem("Light Red");
        MenuItem a3=new MenuItem("Drak Red");
        MenuItem b1=new MenuItem("Green");
        MenuItem b2=new MenuItem("Light Green");
        MenuItem b3=new MenuItem("Dark Green");
        menu11.add(a1);
        sub_menu1.add(a2);
        sub_menu1.add(a3);
        menu11.add(b1);
        sub_menu1.add(b2);
        sub_menu1.add(b3);
        menu11.add(sub_menu1);
        menu_bar.add(menu11);
        menu_f.setMenuBar(menu_bar);
        menu_f.setSize(400,400);
    }
}
```

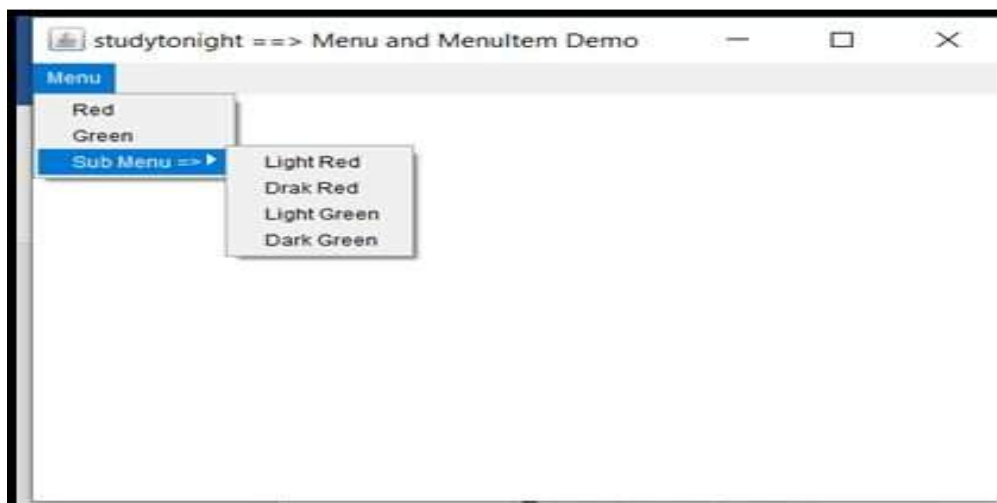
```
menu_f.setLayout(null);
menu_f.setVisible(true);
}
public static void main(String args[])
{
new MenuDemo1();
}
}
```



```
C:\Windows\System32\cmd.exe - java MenuDemo1

D:\Studytonight>javac MenuDemo1.java

D:\Studytonight>java MenuDemo1
```



## AWT PopupMenu

In Java, AWT contains a Popup Menu. It is a popup which is dynamic in nature.

### PopupMenu declaration

public class PopupMenu extends Menu implements MenuContainer,  
Accessible

## Example

```
import java.awt.*;
import java.awt.event.*;
class PopupMenuDemo1
{
    PopupMenuDemo1()
    {
        final Frame pop_menuf= new Frame("studytonight ==>PopupMenu
        Demo");
        final PopupMenu pop_menu = new PopupMenu("*Edit*");
        MenuItem pop_cut = new MenuItem("Cut");
        pop_cut.setActionCommand("Cut");
        MenuItem pop_copy = new MenuItem("Copy");
        pop_copy.setActionCommand("Copy");
        MenuItem pop_paste = new MenuItem("Paste");
        pop_paste.setActionCommand("Paste");
        pop_menu.add(pop_cut);
        pop_menu.add(pop_copy);
        pop_menu.add(pop_paste);
        pop_menuf.addMouseListener(new MouseAdapter()
        {
            public void mouseClicked(MouseEvent a)
            {
                pop_menu.show(pop_menuf , a.getX(), a.getY());
            }
        });
        pop_menuf.add(pop_menu);
        pop_menuf.setSize(400,400);
        pop_menuf.setLayout(null);
        pop_menuf.setVisible(true);
    }
    public static void main(String args[])
    {
        new PopupMenuDemo1();
    }
}
```

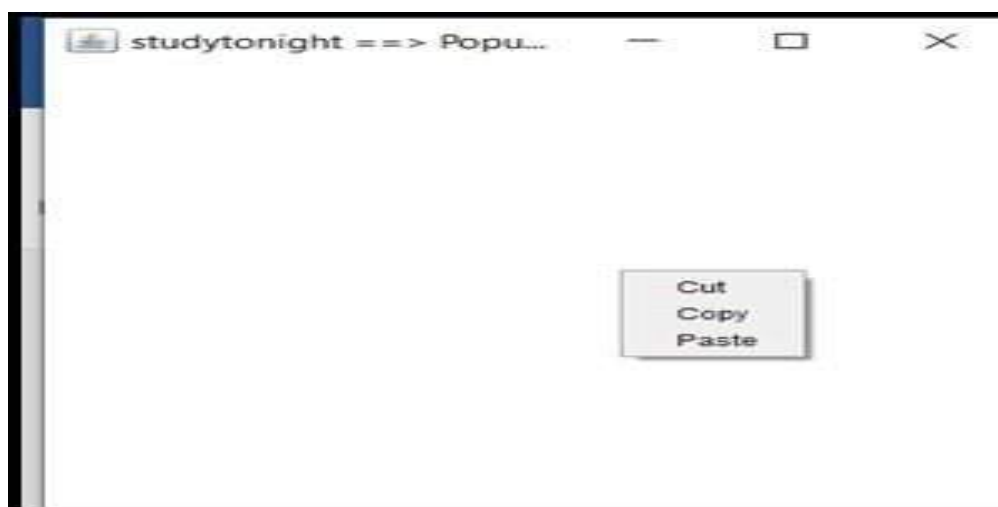




```
C:\Windows\System32\cmd.exe - java PopupMenuDemo1

D:\Studytonight>javac PopupMenuDemo1.java

D:\Studytonight>java PopupMenuDemo1
```



## AWT Panel

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

### Panel declaration

```
public class Panel extends Container implements Accessible
```

### Example

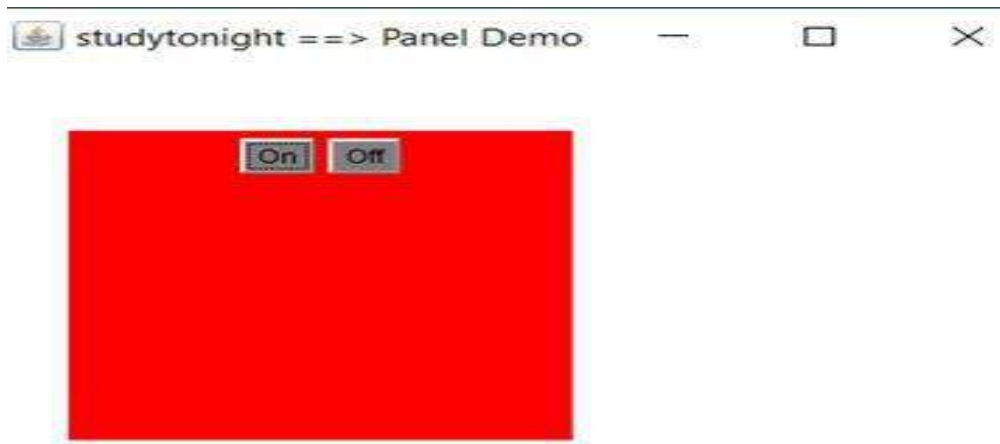
Lets create a panel to add components like: button, textbox etc. the panel provides a place to add awt components.

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



The screenshot shows a Windows command prompt window with the title bar "C:\Windows\System32\cmd.exe - java P...". The window has standard Windows window controls (minimize, maximize, close). The command prompt shows the following commands and output:

```
D:\Studytonight>javac PanelDemo1.java
D:\Studytonight>java PanelDemo1
```



## AWT Dialog

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

### Declaration

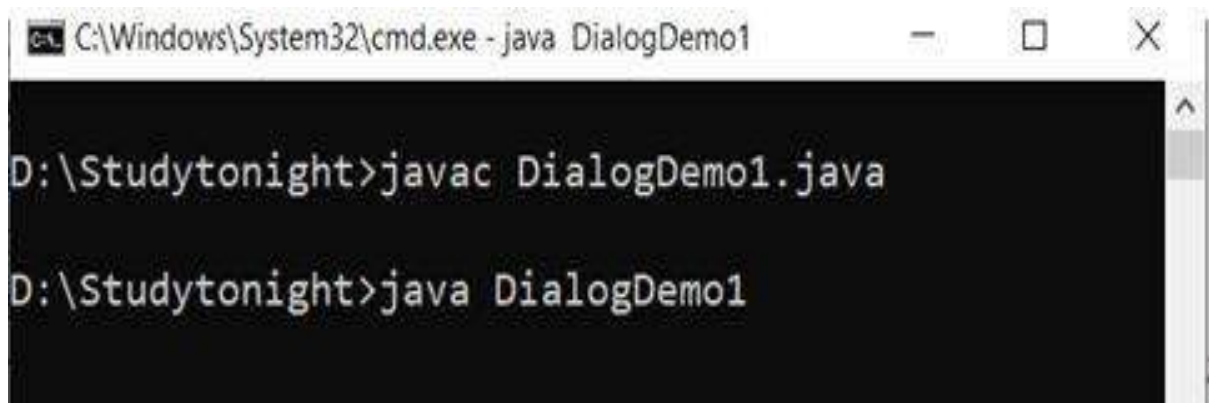
```
public class Dialog extends Window
```

### Example

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

```
import java.awt.*;
import java.awt.event.*;
public class DialogDemo1
{
    private static Dialog dialog_d;
    DialogDemo1()
    {
        Frame dialog_f= new Frame();
        dialog_d = new Dialog(dialog_f , "studytonight ==> Dialog Demo", true);
        dialog_d.setLayout( new FlowLayout() );
        Button dialog_b = new Button ("OK");
        dialog_b.addActionListener ( new ActionListener()
        {
            public void actionPerformed((ActionEvent e )
```

```
{  
DialogDemo1.dialog_d.setVisible(false);  
}  
});  
dialog_d.add( new Label ("Welcome to studytonight. Click on button to  
continue."));  
dialog_d.add(dialog_b);  
dialog_d.setSize(300,300);  
dialog_d.setVisible(true);  
}  
public static void main(String args[])  
{  
new DialogDemo1();  
}  
}
```



```
C:\Windows\System32\cmd.exe - java DialogDemo1  
  
D:\Studytonight>javac DialogDemo1.java  
  
D:\Studytonight>java DialogDemo1
```



# AWT Toolkit

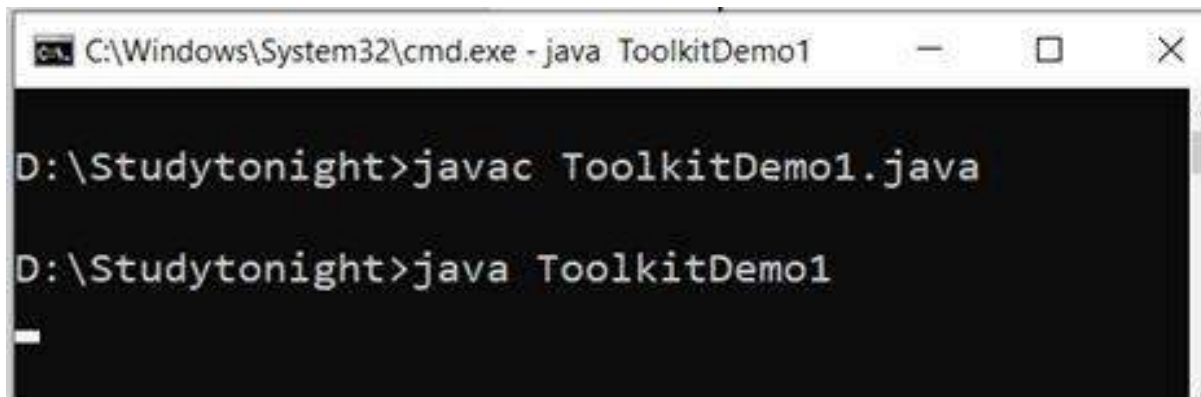
In Java, AWT contains a Toolkit. It is a superclass of Abstract Window Toolkit and can be implemented anywhere.

## Declaration

```
public abstract class Toolkit extends Object
```

## Example

```
import java.awt.*;
import java.awt.event.*;
public class ToolkitDemo1
{
    public static void main(String[] args)
    {
        Frame toolkit_f=new Frame("studytonight ==> Toolkit Demo");
        Button toolkit_b=new Button("beep");
        toolkit_b.setBounds(50,100,60,30);
        toolkit_f.add(toolkit_b);
        toolkit_f.setSize(300,300);
        toolkit_f.setLayout(null);
        toolkit_f.setVisible(true);
        toolkit_b.addActionListener(new ActionListener()
        {
            public void actionPerformed(ActionEvent a)
            {
                Toolkit.getDefaultToolkit().beep();
            }
        });
    }
}
```



```
C:\Windows\System32\cmd.exe - java ToolkitDemo1

D:\Studytonight>javac ToolkitDemo1.java

D:\Studytonight>java ToolkitDemo1
_
```



## ActionListener Interface

In java, ActionListener Interface is present under java.awt.event package. This interface is used when you want to notify click action on button or menu item. It has actionPerformed() method.

### Syntax

```
public abstract void actionPerformed(ActionEvent e)
```

### Following are the three steps to add ActionListener Interface

**Step 1:** Implement the ActionListener Interface in the class.

## Syntax

public class ActionListenerDemo Implements ActionListener

**Step 2:** Now Register all the components with the Listener.

## Syntax

component.addActionListener(instanceOfListenerclass);

**Step 3:** Aylast override the actionPerformed() method.

## Syntax

```
public void actionPerformed(ActionEvent e)
{
//statements
}
```

## Example:

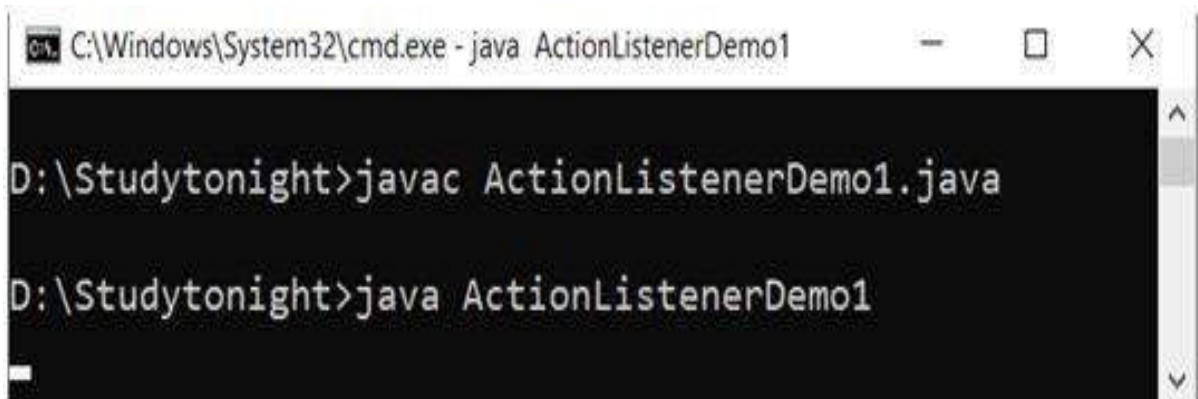
We can use action listener to implement event in awt component. Event can be anything like: mouse click, mouse dragged etc. in this example, we are implementing actionlistener.

```
import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JTextArea;
public class ActionListenerDemo1 implements ActionListener {
JButton aL_button;
JFrame aL_frame;
JTextArea aL_textArea;
public ActionListenerDemo1() {
aL_button = new JButton("Click Me");
aL_frame = new JFrame("studytonight ==>ActionListener Demo");
aL_textArea = new JTextArea(50, 50);
aL_button.addActionListener(this);
aL_textArea.setLineWrap(true);
aL_frame.setLayout(new BorderLayout());
```

```

aL_frame.add(aL_textArea, BorderLayout.NORTH);
aL_frame.add(aL_button, BorderLayout.SOUTH);
aL_frame.pack();
aL_frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
aL_frame.setVisible(true);
}
@Override
public void actionPerformed(ActionEvent e) {
aL_textArea.setText(aL_textArea.getText().concat("Welocme to
studytonight.com\n"));
}
public static void main(String args[]) {
ActionListenerDemo1 obj= new ActionListenerDemo1();
}
}

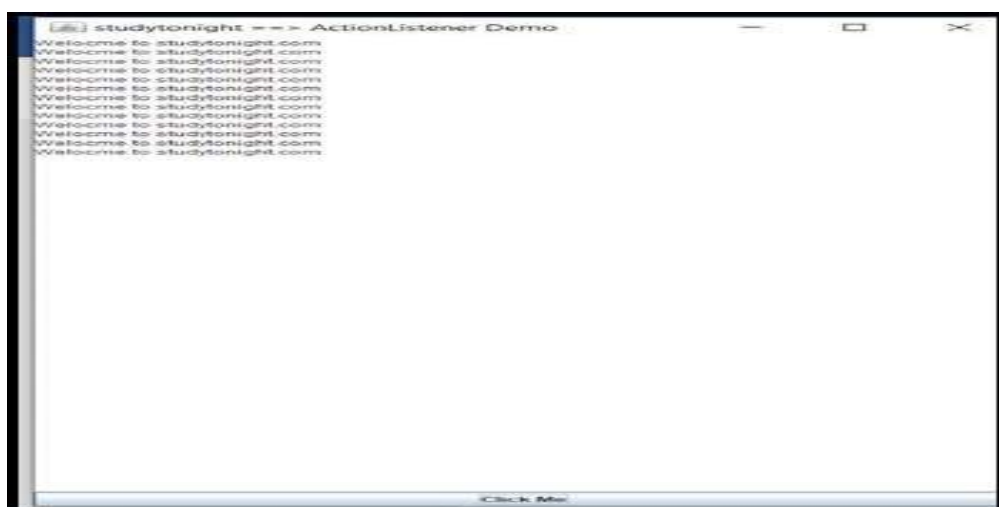
```



```

C:\Windows\System32\cmd.exe - java ActionListenerDemo1
D:\Studytonight>javac ActionListenerDemo1.java
D:\Studytonight>java ActionListenerDemo1

```





# MouseListener Interface


In Java, MouseListener Interface is under java.awt.event package. This interface is used when the state of the mouse is changed. It has 5 methods which are as following:

1. public abstract void mouseClicked(MouseEvent e);
2. public abstract void mouseEntered(MouseEvent e);
3. public abstract void mouseExited(MouseEvent e);
4. public abstract void mousePressed(MouseEvent e);
5. public abstract void mouseReleased(MouseEvent e);

## Example

```
import java.awt.*;
import java.awt.event.*;
public class MouseListenerDemo1 extends Frame implements
MouseListener{
    Label mL_I;
    MouseListenerDemo1(){
        addMouseListener(this);
        mL_I=new Label();
        mL_I.setBounds(10,20,500,100);
        add(mL_I);
        setSize(300,300);
        setLayout(null);
        setVisible(true);
    }
    public void mouseClicked(MouseEvent e) {
        mL_I.setText("studytonight ==> Mouse Clicked");
    }
    public void mouseEntered(MouseEvent e) {
        mL_I.setText("studytonight ==> Mouse Entered");
    }
    public void mouseExited(MouseEvent e) {
        mL_I.setText("studytonight ==> Mouse Exited");
    }
    public void mousePressed(MouseEvent e) {
        mL_I.setText("studytonight ==> Mouse Pressed"); }
    public void mouseReleased(MouseEvent e) {
        mL_I.setText("studytonight ==> Mouse Released");
    }
    public static void main(String[] args) {
```

```
new MouseListenerDemo1();  
}  
}
```



```
C:\Windows\System32\cmd.exe - java MouseListenerDem...  
D:\Studytonight>javac MouseListenerDemo1.java  
D:\Studytonight>java MouseListenerDemo1
```



## MouseMotionListener Interface

In Java, MouseMotionListener Interface is under java.awt.event package. This interface is used whenever the mouse is moved or dragged. It has 2 methods which are as following:

1. public abstract void mouseDragged(MouseEvent e);
2. public abstract void mouseMoved(MouseEvent e);

### Example

```

import java.awt.*;
import java.awt.event.*;
public class MouseMotionListenerDemo1 extends Frame implements
MouseMotionListener{
MouseMotionListenerDemo1(){
addMouseMotionListener(this);
setSize(500,500);
setLayout(null);
setVisible(true);
}
public void mouseDragged(MouseEvent a) {
Graphics mM_g=getGraphics();
mM_g.setColor(Color.ORANGE);
mM_g.fillOval(a.getX(),a.getY(),10,10);
}
public void mouseMoved(MouseEvent e) {}
public static void main(String[] args) {
new MouseMotionListenerDemo1();
}
}

```

## ItemListener Interface

In Java, ItemListener Interface is under java.awt.event package. This interface is used whenever the checkbox is clicked. It has itemStateChanged() method

### Syntax

```
public abstract void itemStateChanged(ItemEvent e)
```

### Example

This interface is used to handle item listener events like: item selected or checkbox checked. In this example, we are handling checkbox checked event.

```

import java.awt.*;
import java.awt.event.*;
public class ItemListenerDemo1 implements ItemListener{
Checkbox iT_checkBox1,iT_checkBox2;
Label iT_label;
ItemListenerDemo1(){

```

```

Frame iT_f= new Frame("studytonight ==>CheckBox Demo");
iT_label = new Label();
iT_label.setAlignment(Label.CENTER);
iT_label.setSize(400,100);
iT_checkBox1 = new Checkbox("Core Java");
iT_checkBox1.setBounds(100,100, 100,40);
iT_checkBox2 = new Checkbox("jsp");
iT_checkBox2.setBounds(100,150, 100,40);
iT_f.add(iT_checkBox1);
iT_f.add(iT_checkBox2);
iT_f.add(iT_label);
iT_checkBox1.addItemListener(this);
iT_checkBox2.addItemListener(this);
iT_f.setSize(400,400);
iT_f.setLayout(null);
iT_f.setVisible(true);
}
public void itemStateChanged(ItemEvent iT) {
if(iT.getSource()==iT_checkBox1)
iT_label.setText("Core Java Checkbox: "+
(iT.getStateChange()==1?"checked":"unchecked"));
if(iT.getSource()==iT_checkBox2)
iT_label.setText("jsp Checkbox: "+
(iT.getStateChange()==1?"checked":"unchecked"));
}
public static void main(String args[])
{
new ItemListenerDemo1();
}
}

```



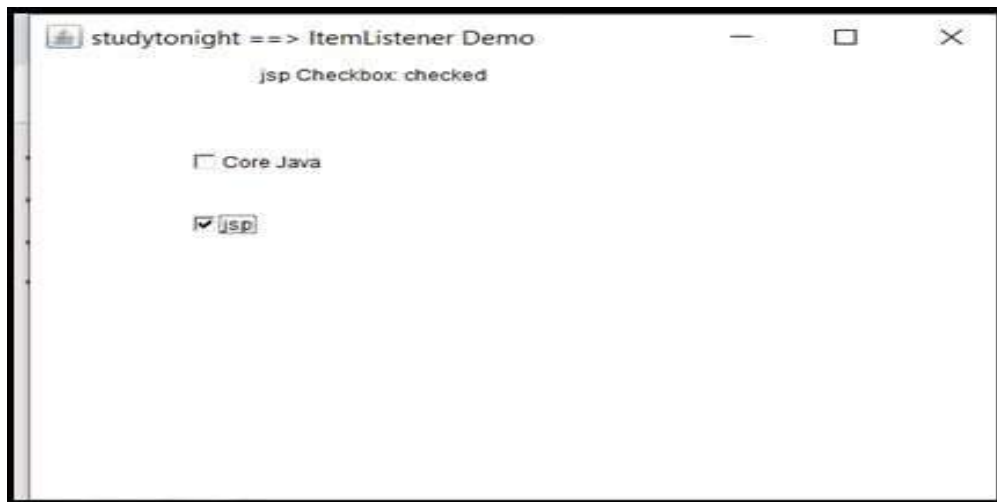
The screenshot shows a Windows command prompt window with the title bar "C:\Windows\System32\cmd.exe - java ItemListenerDemo1". The window has a black background with white text. The command prompt shows the following commands and their outputs:

```

D:\Studytonight>javac ItemListenerDemo1.java

D:\Studytonight>java ItemListenerDemo1

```



## KeyListener Interface

In Java, KeyListener Interface is under java.awt.event package. This interface is used when the state of the key is changed. It has 3 methods which are as following:

1. public abstract void keyPressed(KeyEvent e);
2. public abstract void keyReleased(KeyEvent e);
3. public abstract void keyTyped(KeyEvent e);

## Example

In this example, we are using keylistener interface to handle key events that can be key release, typed etc. see the below example.

```
import java.awt.*;
import java.awt.event.*;
public class KeyListenerDemo1 extends Frame implements KeyListener{
    Label kL_l;
    TextArea kL_area;
    KeyListenerDemo1(){
        kL_l=new Label();
        kL_l.setBounds(20,50,500,20);
        kL_area=new TextArea();
        kL_area.setBounds(20,80,300, 300);
        kL_area.addKeyListener(this);
        add(kL_l);
        add(kL_area);
    }
}
```

```

setSize(400,400);
setLayout(null);
setVisible(true);
}
public void keyPressed(KeyEvent e) {
kL_1.setText("studytonight ==> Key Pressed");
}
public void keyReleased(KeyEvent e) {
kL_1.setText("studytonight ==> Key Released");
}
public void keyTyped(KeyEvent e) {
kL_1.setText("studytonight ==> Key Typed");
}
public static void main(String[] args) {
new KeyListenerDemo1();
}
}

```

## WindowListener Interface

In Java, WindowListener Interface is under java.awt.event package. This interface is used when the state of the window is changed. It has 7 methods which are as following:

1. public abstract void windowActivated(WindowEvent e);
2. public abstract void windowClosed(WindowEvent e);
3. public abstract void windowClosing(WindowEvent e);
4. public abstract void windowDeactivated(WindowEvent e);
5. public abstract void windowDeiconified(WindowEvent e);
6. public abstract void windowIconified(WindowEvent e);
7. public abstract void windowOpened(WindowEvent e);

## Example

In this example, we are handling windows events like: window open, close etc.

```

import java.awt.*;
import java.awt.event.WindowEvent;
import java.awt.event.WindowListener;
public class WindowDemo1 extends Frame implements WindowListener
{
WindowDemo1()

```

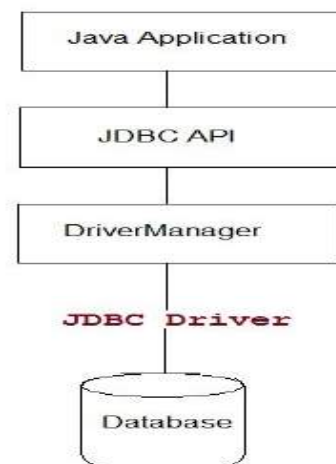
```
{
addWindowListener(this);
setSize(500,500);
setLayout(null);
setVisible(true);
}
public static void main(String[] args)
{
new WindowDemo1();
}
public void windowActivated(WindowEvent arg0)
{
System.out.println("studytonight ==> activated");
}
public void windowClosed(WindowEvent arg0)
{
System.out.println("studytonight ==> closed");
}
public void windowClosing(WindowEvent arg0)
{
System.out.println("studytonight ==> closing");
dispose();
}
public void windowDeactivated(WindowEvent arg0)
{
System.out.println("studytonight ==> deactivated");
}
public void windowDeiconified(WindowEvent arg0)
{
System.out.println("studytonight ==>deiconified");
}
public void windowIconified(WindowEvent arg0)
{
System.out.println("studytonight ==>iconified");
}
public void windowOpened(WindowEvent arg0)
{
System.out.println("studytonight ==> opened");
}
}
```

# Java Database Connectivity(JDBC)

Java Database Connectivity(JDBC) is an Application Programming Interface(API) used to connect Java application with Database. JDBC is used to interact with various type of Database such as Oracle, MS Access, My SQL and SQL Server. JDBC can also be defined as the platform-independent interface between a relational database and Java programming. It allows java program to execute SQL statement and retrieve result from database.

The JDBC API consists of classes and methods that are used to perform various operations like: connect, read, write and store data in the database. In this tutorial we will learn the JDBC with examples.

You can get idea of how JDBC connect Java Application to the database by following image.




## JDBC Driver

A JDBC driver is a JDBC API implementation used for connecting to a particular type of database. There are several types of JDBC drivers:

- 1).Type 1 : contains a mapping to another data access API; an example of this is the JDBC-ODBC driver
- 2). Type 2 : is an implementation that uses client-side libraries of the target database; also called a native-API driver
- 3).Type 3 : uses middleware to convert JDBC calls into database-specific calls; also known as a network protocol driver



4).Type 4 : connect directly to a database by converting JDBC calls into database-specific calls; known as database protocol drivers or thin drivers.

The most commonly used type is type 4, as it has the advantage of being platform-independent. Connecting directly to a database server provides better performance compared to other types. The downside of this type of driver is that it's database-specific  given each database has its own specific protocol.

# 1.Connecting to a Database:-

To connect to a database, we simply have to initialize the driver and open a database connection.

## 1.1.Registering the Driver:-

For our example, we will use a type 4 database protocol driver. Since we're using a MySQL database, we need the mysql-connector-java dependency:

Next, let's register the driver using the `Class.forName()` method, which dynamically loads the driver class:

```
Class.forName("com.mysql.cj.jdbc.Driver");
```

In older versions of JDBC, before obtaining a connection, we first had to initialize the JDBC driver by calling the `Class.forName` method. As of JDBC 4.0, all drivers that are found in the classpath are automatically loaded. Therefore, we won't need this `Class.forName` part in modern environments.

## 1.2.Creating the Connection:-

To open a connection, we can use the `getConnection()` method of `DriverManager` class. This method requires a connection URL String parameter:

```
try (Connection con = DriverManager
.getConnection("jdbc:mysql://localhost:3306/myDb", "user1", "pass")) {
// use con here
}
```

Since the `Connection` is an `AutoCloseable` resource, we should use it inside a `try-with-resources` block.

The syntax of the connection URL depends on the type of database used. Let's take a look at a few examples:

```
jdbc:mysql://localhost:3306/myDb?user=user1&password=pass
```

To connect to the specified myDb database, we will have to create the database and a user, and add grant necessary access:

```
CREATE DATABASE myDb;  
CREATE USER 'user1' IDENTIFIED BY 'pass';  
GRANT ALL on myDb.* TO 'user1';
```

## 2.Executing SQL Statements

To send SQL instructions to the database, we can use instances of type Statement, PreparedStatement, or CallableStatement, which we can obtain using the Connection object.

### 2.1.Statement:-

The Statement interface contains the essential functions for executing SQL commands.

First, let's create a Statement object:

```
try (Statement stmt = con.createStatement()) {  
    // use stmt here  
}
```

Again, we should work with Statements inside a try-with-resources block for automatic resource management.

Anyway, executing SQL instructions can be done through the use of three methods:

- 1). executeQuery() for SELECT instructions
- 2). executeUpdate() for updating the data or the database structure
- 3). execute() can be used for both cases above when the result is unknown

Let's use the execute() method to add a students table to our database:

```
String tableSql = "CREATE TABLE IF NOT EXISTS employees"  
+ "(emp_id int PRIMARY KEY AUTO_INCREMENT, name varchar(30),"  
+ "position varchar(30), salary double)";  
stmt.execute(tableSql);
```

When using the `execute()` method to update the data, then the `stmt.getUpdateCount()` method returns the number of rows affected.

If the result is 0 then either no rows were affected, or it was a database structure update command.

If the value is -1, then the command was a `SELECT` query; we can then obtain the result using `stmt.getResultSet()`.

Next, let's add a record to our table using the `executeUpdate()` method:

```
String insertSql = "INSERT INTO employees(name, position, salary)"
+ " VALUES('john', 'developer', 2000)";
stmt.executeUpdate(insertSql);
```

The method returns the number of affected rows for a command that updates rows or 0 for a command that updates the database structure.

We can retrieve the records from the table using the `executeQuery()` method which returns an object of type `ResultSet`:

```
String selectSql = "SELECT * FROM employees";
try (ResultSet resultSet = stmt.executeQuery(selectSql)) {
    // use resultSet here
}
```

We should make sure to close the `ResultSet` instances after use. Otherwise, we may keep the underlying cursor open for a much longer period than expected. To do that, it's recommended to use a try-with-resources block, as in our example above.

## **2.2. PreparedStatement:-**

`PreparedStatement` objects contain precompiled SQL sequences. They can have one or more parameters denoted by a question mark. Let's create a `PreparedStatement` which updates records in the `employees` table based on given parameters:

```
String updatePositionSql = "UPDATE employees SET position=? WHERE
emp_id=?";
try (PreparedStatement pstmt = con.prepareStatement(updatePositionSql))
{
    // use pstmt here
}
```

To add parameters to the PreparedStatement, we can use simple setters `setX()` where X is the type of the parameter, and the method arguments are the order and value of the parameter:

```
pstmt.setString(1, "lead developer");  
pstmt.setInt(2, 1);
```

The statement is executed with one of the same three methods described before: `executeQuery()`, `executeUpdate()`, `execute()` without the SQL String parameter:

```
int rowsAffected = pstmt.executeUpdate();
```

### 2.3. CallableStatement:-

The CallableStatement interface allows calling stored procedures.

To create a CallableStatement object, we can use the `prepareCall()` method of Connection:

```
String preparedSql = "{call insertEmployee(?,?,?,?)}";  
try (CallableStatement cstmt = con.prepareCall(preparedSql)) {  
    // use cstmt here  
}
```

Setting input parameter values for the stored procedure is done like in the PreparedStatement interface, using `setX()` methods:

```
cstmt.setString(2, "ana");  
cstmt.setString(3, "tester");  
cstmt.setDouble(4, 2000);
```

If the stored procedure has output parameters, we need to add them using the `registerOutParameter()` method:

```
cstmt.registerOutParameter(1, Types.INTEGER);
```

Then let's execute the statement and retrieve the returned value using a corresponding `getX()` method:

```
cstmt.execute();  
int new_id = cstmt.getInt(1);
```

For example to work, we need to create the stored procedure in our MySQL database:

```

delimiter //
CREATE PROCEDURE insertEmployee(OUT emp_id int,
IN emp_name varchar(30), IN position varchar(30), IN salary double)
BEGIN
INSERT INTO employees(name, position,salary) VALUES
(emp_name,position,salary);
SET emp_id = LAST_INSERT_ID();
END //
delimiter ;

```

The insertEmployee procedure above will insert a new record into the employees table using the given parameters and return the id of the new record in the emp\_id out parameter.

To be able to run a stored procedure from Java, the connection user needs to have access to the stored procedure's metadata. This can be achieved by granting rights to the user on all stored procedures in all databases:

```
GRANT ALL ON mysql.proc TO 'user1';
```

Alternatively, we can open the connection with the property noAccessToProcedureBodies set to true:

```

con = DriverManager.getConnection(
"jdbc:mysql://localhost:3306/myDb?noAccessToProcedureBodies=true",
"user1", "pass");

```

This will inform the JDBC API that the user does not have the rights to read the procedure metadata so that it will create all parameters as INOUT String parameters.

### 3. Parsing Query Results:-

After executing a query, the result is represented by a ResultSet object, which has a structure similar to a table, with lines and columns.

#### 3.1.ResultSet Interface:-

The ResultSet uses the next() method to move to the next line.

Let's first create an Employee class to store our retrieved records:

```

public class Employee {
private int id;
private String name;
private String position;

```

```
private double salary;  
// standard constructor, getters, setters  
}
```

Next, let's traverse the ResultSet and create an Employee object for each record:

```
String selectSql = "SELECT * FROM employees";  
try (ResultSet resultSet = stmt.executeQuery(selectSql)) {  
    List employees = new ArrayList<>();  
    while (resultSet.next()) {  
        Employee emp = new Employee();  
        emp.setId(resultSet.getInt("emp_id"));  
        emp.setName(resultSet.getString("name"));  
        emp.setPosition(resultSet.getString("position"));  
        emp.setSalary(resultSet.getDouble("salary"));  
        employees.add(emp);  
    }  
}
```

Retrieving the value for each table cell can be done using methods of type getX() where X represents the type of the cell data.

The getX() methods can be used with an int parameter representing the order of the cell, or a String parameter representing the name of the column. The latter option is preferable in case we change the order of the columns in the query.

### **3.2. Updatable ResultSet:-**

Implicitly, a ResultSet object can only be traversed forward and cannot be modified.

If we want to use the ResultSet to update data and traverse it in both directions, we need to create the Statement object with additional parameters:

```
stmt = con.createStatement(  
    ResultSet.TYPE_SCROLL_INSENSITIVE,  
    ResultSet.CONCUR_UPDATABLE  
);
```

To navigate this type of ResultSet, we can use one of the methods:

- i).first(), last(), beforeFirst(), beforeLast() - to move to the first or last line of a ResultSet or to the line before these
- ii).next(), previous() - to navigate forward and backward in the ResultSet
- iii).getRow() - to obtain the current row number
- iv).moveToInsertRow(), moveToCurrentRow() - to move to a new empty row to insert and back to the current one if on a new row
- v).absolute(int row) - to move to the specified row
- vi).relative(int nrRows) - to move the cursor the given number of rows

Updating the ResultSet can be done using methods with the format updateX() where X is the type of cell data. These methods only update the ResultSet object and not the database tables.

To persist the ResultSet changes to the database, we must further use one of the methods:

- i).updateRow() - to persist the changes to the current row to the database
- ii).insertRow(), deleteRow() - to add a new row or delete the current one from the database
- iii).refreshRow() - to refresh the ResultSet with any changes in the database
- iv).cancelRowUpdates() - to cancel changes made to the current row

Let's take a look at an example of using some of these methods by updating data in the employee's table:

```
try (Statement updatableStmt = con.createStatement(
    ResultSet.TYPE_SCROLL_INSENSITIVE,
    ResultSet.CONCUR_UPDATABLE)) {
    try (ResultSet updatableResultSet =
        updatableStmt.executeQuery(selectSql)) {
        updatableResultSet.moveToInsertRow();
        updatableResultSet.updateString("name", "mark");
        updatableResultSet.updateString("position", "analyst");
        updatableResultSet.updateDouble("salary", 2000);
        updatableResultSet.insertRow();
    }
}
```

## 4.Parsing Metadata

The JDBC API allows looking up information about the database, called metadata. **4.1.DatabaseMetadata** The DatabaseMetadata interface can be used to obtain general information about the database such as the tables,

stored procedures, or SQL dialect.

Let's have a quick look at how we can retrieve information on the database tables:

```
DatabaseMetaData dbmd = con.getMetaData();
ResultSet tablesResultSet = dbmd.getTables(null, null, "%", null);
while (tablesResultSet.next()) {
    LOG.info(tablesResultSet.getString("TABLE_NAME"));
}
```

## 4.2.ResultSetMetadata

This interface can be used to find information about a certain ResultSet, such as the number and name of its columns:

```
ResultSetMetaData rsmd = rs.getMetaData();
int nrColumns = rsmd.getColumnCount();
IntStream.range(1, nrColumns).forEach(i -> {
    try {
        LOG.info(rsmd.getColumnName(i));
    } catch (SQLException e) {
        e.printStackTrace();
    }
});
```

## 5.Closing the Resources

When we're no longer using it, we need to close the connection to release database resources.

We can do this using the close() API:

```
con.close();
```

However, if we're using the resource in a try-with-resources block, we don't need to call the close() method explicitly, as the try-with-resources block does that for us automatically.

The same is true for the Statements, PreparedStatements, CallableStatements, and ResultSets.



# Java Server Pages(JSP)

JSP technology is used to create web application just like Servlet technology. It can be thought of as an extension to Servlet because it provides more functionality than servlet such as expression language, JSTL, etc.

A JSP page consists of HTML tags and JSP tags. The JSP pages are easier to maintain than Servlet because we can separate designing and development. It provides some additional features such as Expression Language, Custom Tags, etc.

## Advantages of JSP over Servlet:-

There are many advantages of JSP over the Servlet. They are as follows:

### 1) Extension to Servlet:-

JSP technology is the extension to Servlet technology. We can use all the features of the Servlet in JSP. In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP, that makes JSP development easy.

### 2) Easy to maintain:-

JSP can be easily managed because we can easily separate our business logic with presentation logic. In Servlet technology, we mix our business logic with the presentation logic.

### 3) Fast Development: No need to recompile and redeploy:-

If JSP page is modified, we don't need to recompile and redeploy the project. The Servlet code needs to be updated and recompiled if we have to change the look and feel of the application.

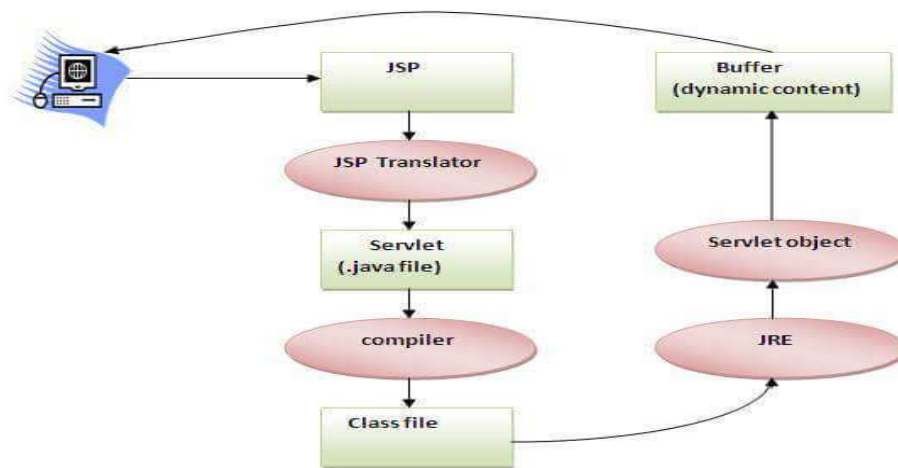
### 4) Less code than Servlet:-

In JSP, we can use many tags such as action tags, JSTL, custom tags, etc. that reduces the code. Moreover, we can use EL, implicit objects, etc.

## The Lifecycle of a JSP Page:-

The JSP pages follow these phases:

- Translation of JSP Page
- Compilation of JSP Page
- Classloading (the classloader loads class file)
- Instantiation (Object of the Generated Servlet is created).
- Initialization ( the container invokes jsplnit() method).
- Request processing ( the container invokes \_jspService() method).
- Destroy ( the container invokes jspDestroy() method).



As depicted in the above diagram, JSP page is translated into Servlet by the help of JSP translator. The JSP translator is a part of the web server which is responsible for translating the JSP page into Servlet. After that, Servlet page is compiled by the compiler and gets converted into the class file. Moreover, all the processes that happen in Servlet are performed on JSP later like initialization, committing response to the browser and destroy.

## Creating a simple JSP Page:-

To create the first JSP page, write some HTML code as given below, and save it by .jsp extension. We have saved this file as index.jsp. Put it in a folder and paste the folder in the web-apps directory in apache tomcat to run the JSP page.

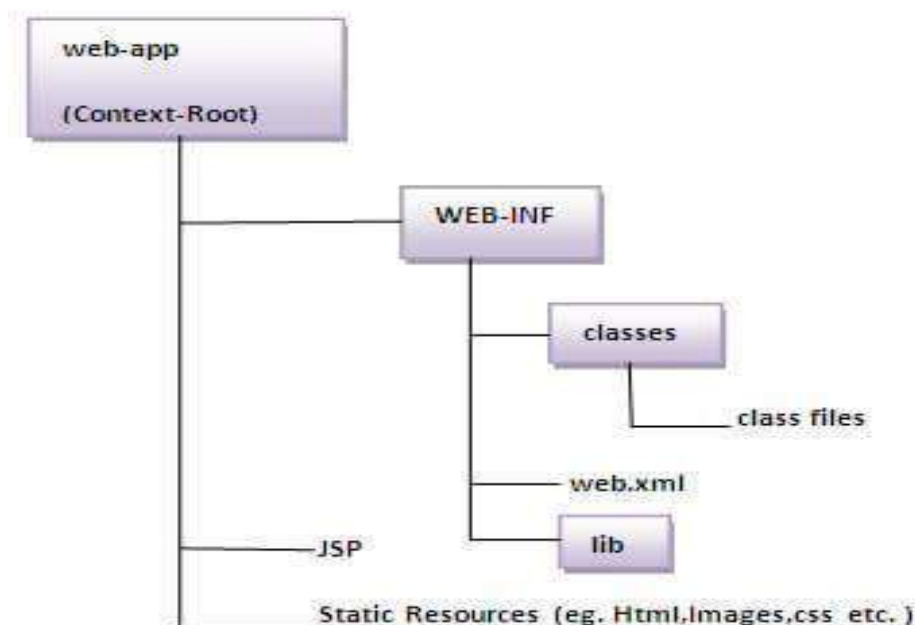
### index.jsp

Let's see the simple example of JSP where we are using the scriptlet tag to put Java code in the JSP page. We will learn scriptlet tag later.

```
<html>
<body>
<% out.print(2*5); %>
</body>
</html>
```

## The Directory structure of JSP:-

The directory structure of JSP page is same as Servlet. We contain the JSP page outside the WEB-INF folder or in any directory.



## JSP Scriptlet tag (Scripting elements):-

In JSP, java code can be written inside the jsp page using the scriptlet tag. Let's see what are the scripting elements first.

## JSP Scripting elements:-

The scripting elements provides the ability to insert java code inside the jsp. There are three types of scripting elements:

- scriptlet tag
- expression tag

- declaration tag

## 1.JSP scriptlet tag:-

A scriptlet tag is used to execute java source code in JSP. Syntax is as follows:

```
<% java source code %>
```

### Example:-

In this example, we have created two files index.html and welcome.jsp. The index.html file gets the username from the user and the welcome.jsp file prints the username with the welcome message.

File: index.html

```
< html >
< body>
< form action="welcome.jsp">
< input type="text" name="uname" >
< input type="submit" value="go">
</form>
</body >
</html >
```

File: welcome.jsp

```
< html >
< body>
<%
String name=request.getParameter("uname");
out.print("welcome "+name);
%>
</form>
</body >
</html >
```

## 2.JSP expression tag:-

The code placed within JSP expression tag is written to the output stream of the response. So you need not write out.print() to write data. It is mainly used to print the values of variable or method.

### Syntax

```
<%= statement %>
```

### **Example:-**

In this example of jsp expression tag, we are simply displaying a welcome message.

```
< html >
< body>
<%= "Welcome to jsp" %> </form>
</body >
</html >
```

### **Example:-**

To display the current time, we have used the getTime() method of Calendar class. The getTime() is an instance method of Calendar class, so we have called it after getting the instance of Calendar class by the getInstance() method.

index.jsp

```
< html >
< body>
Current Time: <%= java.util.Calendar.getInstance().getTime() %> </body >
</html >
```

### **3.JSP Declaration Tag-**

The JSP declaration tag is used to declare fields and methods. The code written inside the jsp declaration tag is placed outside the service() method of auto generated servlet. So it doesn't get memory at each request.

### **Syntax**

The syntax of the declaration tag is as follows:

```
<%! field or method declaration %>
```

### **Example**

In this example of JSP declaration tag, we are declaring the field and printing the value of the declared field using the jsp expression tag.

index.jsp

```
< html >
< body>
<%! int data=50; %> <%= "Value of the variable is:"+data %> </body >
</html >
```

# JSP-Directives

Directives supply directions and messages to a JSP container. The directives provide global information about the entire page of JSP. Hence, they are an essential part of the JSP code. These special instructions are used for translating JSP to servlet code. In this chapter, you will learn about the different components of directives in detail.

In JSP's life cycle phase, the code must be converted to a servlet that deals with the translation phase. They provide commands or directions to the container on how to deal with and manage certain JSP processing portions. Directives can contain several attributes that are separated by a comma and act as key-value pairs. In JSP, directives are described with a pair of `<%@ .... %>` tags.

The syntax of Directives looks like:

```
<%@ directive attribute="" %>
```

There are 3 types of directives:

- 1).Page directive
- 2).Include directive
- 3).Taglib directive

## 1).Page Directive

The page directive is used for defining attributes that can be applied to a complete JSP page. You may place your code for Page Directives anywhere within your JSP page. However, in general, page directives are implied at the top of your JSP page.

The basic syntax of the page directive is:`<%@ page attribute = "attribute_value" %>`

The XML equivalent for the above derivation is:`<jsp:directive.page attribute = "attribute_value" />`

The attributes used by the Page directives are:

**1). buffer:** Buffer attribute sets the buffer size in KB to control the JSP page's output.

**2).contentType:** The ContentType attribute defines the document's MIME (Multipurpose Internet Mail Extension) in the HTTP response header.

**3).autoFlush:** The autofill attribute controls the behavior of the servlet output buffer. It monitors the buffer output and specifies whether the filled buffer output should be flushed automatically or an exception should be raised to indicate buffer overflow.

**4).errorPage:** Defining the "ErrorPage" attribute is the correct way to handle JSP errors. If an exception occurs on the current page, it will be redirected to the error page.

**5). extends:** extends attribute used for specifying a superclass that tells whether the generated servlet has to extend or not.

**6). import:** The import attribute is used to specify a list of packages or classes used in JSP code, just as Java's import statement does in a Java code.

**7).isErrorPage:** This "isErrorPage" attribute of the Page directive is used to specify that the current page can be displayed as an error page.

**8).info:** This "info" attribute sets the JSP page information, which is later obtained using the `getServletInfo()` method of the servlet interface.

**9).isThreadSafe:** Both the servlet and JSP are multithreaded. If you want to control JSP page behavior and define the threading model, you can use the "isThreadSafe" attribute of the page directive.

**10).Language:** The language attribute specifies the programming language used in the JSP page. The default value of the language attribute is "Java".

**11).Session:** In JSP, the page directive session attribute specifies whether the current JSP page participates in the current HTTP session.

**12).isELIgnored:** This `isELIgnored` attribute is used to specify whether the expression language (EL) implied by the JSP page will be ignored.

**13).isScriptingEnabled:** This "isScriptingEnabled" attribute determines if the scripting elements are allowed for use or not.

## **2).Include Directive**

The JSP "include directive" is used to include one file in another JSP file.

This includes HTML, JSP, text, and other files. This directive is also used to create templates according to the developer's requirement and breaks the pages in the header, footer, and sidebar.

To use this Include Directive, you have to write it like: `<%@ include file = "relative url" >`

The XML equivalent of the above way of representation is: `<jsp:directive.include file = "relative url" />`

### Example

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
pageEncoding = "ISO-8859-1"%>
<%@ include file="directive_header_code.jsp" %>
```

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"
/>
<title>Include Directive Example</title>
</head>
<body>
<p>This file includes a header file named directive_header_code.jsp</p>
</body>
</html>
```

## 3).Taglib Directive

The JSP taglib directive is implemented to define a tag library with "taglib" as its prefix. Custom tag sections of JSP use taglib. JSP's taglibdirective is used as standard tag libraries.

To implement taglib, you have to write it like this: `<%@ taglib uri="uri" prefix="value"%>`

### Example

```
<%@ taglib uri = "http://www.example.com/custlib" prefix = "w3tag" %>
```

```
<!DOCTYPE html>
<html>
<body>
```



```
<mytag: hi/>
</body>
</html>
```

## Implicit Objects

The JSP engine produces these objects during the translation phase (i.e., at the time of translation from JSP to Servlet). They are being formed within the service method so that JSP developers can use them directly in Scriptlet without declaration and initialization.

There are a total of nine implicit objects supported by the JSP. These are:

- 1).out:** javax.servlet.jsp.JspWriter
- 2).request:** javax.servlet.http.HttpServletRequest
- 3).response:** javax.servlet.http.HttpServletResponse
- 4).session:** javax.servlet.http.HttpSession
- 5).application:** javax.servlet.ServletContext
- 6).exception:** javax.servlet.jsp.JspException
- 7).page:** java.lang.Object
- 8).pageContext:** javax.servlet.jsp.PageContext
- 9).config:** javax.servlet.ServletConfig

### 1).Out Implicit Objects

- An out object is an implicit object for writing data to the buffer and sending output as a response to the client's browser.
- The out implicit object is an instance of a javax.servlet.jsp.JspWriter class.

- You will learn more about the various concepts of the out Object in subsequent chapters.

### **Example(html file)**

```
<!DOCTYPE html>
<html>
<head>
<title>Please insert a User name and a password</title>
</head>
<body>
<% out.println("Today's date-time:
"+java.util.Calendar.getInstance().getTime()); %>
</body>
</html>
```

Output

Today's date-time: Nov 01 12:10:05 IST 2020

## **2).Request Implicit Objects**

- A request object is an implicit object that is used to request an implicit object, which is to receive data on a JSP page, which has been submitted by the user on the previous JSP/HTML page.
- The request implicit object used in Java is an instance of a javax.servlet.http.HttpServletRequest interface where a client requests a page every time the JSP engine has to create a new object for characterizing that request.
- The container creates it for every request.

- It is used to request information such as parameters, header information, server names, cookies, and HTTP methods.
- It uses the `getParameter()` method to access the request parameter.

Here is an example of a JSP request implicit object where a user submits login information, and another JSP page receives it for processing:

### **Example(html file)**

```
<!DOCTYPE html>
<html>
<head>
<title>Please insert a User name and a password</title>
</head>
<body>
<form action="login.jsp">
Please insert Username: <input type="text" name="u_name" />
Please insert Password: <input type="text" name="passwd" />
<input type="submit" value="Submit Details" />
</form>
</body>
</html>
```

### **Example(login.jsp):**

```
<%@ page import = " java.util.* " %>
<%
String username = request.getParameter("u_name");
String password = request.getParameter("passwd");
out.print("Name: "+username+" Password: " +passwd);
%>
```

## **3).Response Implicit Object:**

- A response object is an implicit object implemented to modify or deal with the reply sent to the client (i.e., browser) after processing the

request, such as redirect responding to another resource or an error sent to a client.

- The response implicit object is an instance of a `javax.servlet.http.HttpServletResponse` interface.
- The container creates it for every request.
- You will learn more about the various concepts of the request and response in subsequent chapters.

#### **4).Session Implicit Object:**

- A session object is the most commonly used implicit object implemented to store user data to make it available on other JSP pages until the user's session is active.
- The session implicit object is an instance of a `javax.servlet.http.HttpSession` interface.
- This session object has different session methods to manage data within the session scope.

- You will learn more about the use of the session in subsequent chapters.

## 5).Application Implicit Object:

An application object is another implicit object implemented to initialize application-wide parameters and maintain functional data throughout the JSP application.

## 6).Exception Implicit Object:

- An exception implicit object is implemented to handle exceptions to display error messages.
- The exception implicit object is an instance of the `java.lang.Throwable` class.
- It is only available for JSP pages, with the `isErrorPage` value set as "True". This means Exception objects can only be used in error pages.

### Example(html file)

```
<!DOCTYPE html>
<html>
<head>
<title>Enter two Integers for Division </title>
</head>
<body>
<form action="submit.jsp">
Insert first Integer: <input type="text" name="numone" />
Insert second Integer: <input type="text" name="numtwo" />
```

```
<input type="submit" value="Get Results" />
</form>
</body>
</html>
```

### **Example(submit.jsp):**

```
<%@ page errorPage="exception.jsp" %>

<%
String num1 = request.getParameter("numone");
String num2 = request.getParameter("numtwo");
int var1= Integer.parseInt(num1);
int var2= Integer.parseInt(num2);
int r= var1 / var2;
out.print("Output is: "+ r);
%>
```

### **Example(exception.jsp):**

```
<%@ page isErrorPage='true' %>

<%
out.print("Error Message : ");
out.print(exception.getMessage());
%>
```

## **7).Page Implicit Object:**

A page object is an implicit object that is referenced to the current instance of the servlet. You can use it instead. Covering it specifically is hardly ever used and not a valuable implicit object while building a JSP application.

```
<% String pageName = page.toString();
out.println("The current page is: " +pageName);%>
```

## **8).Config Implicit Object:**

A config object is a configuration object of a servlet that is mainly used to access and receive configuration information such as servlet context, servlet name, configuration parameters, etc. It uses various methods used to fetch configuration information.

You will learn more about how to set and use config objects in later

chapters.

## JSP Comments

JSP comment marks to text or statements that the JSP container should ignore. A JSP comment is useful when you want to hide or "comment out", a part of your JSP page.

Following is the syntax of the JSP comments :-`<%-- This is JSP comment - %>`

Following example shows the JSP Comments -

```
<html>
<head>
<title>A Comment Test</title>
</head>
<body>
<h2>A Test of Comments</h2>
<%-- This comment will not be visible in the page source --%>
</body>
</html>
```

The above code will generate the following result -

### A Test of Comments

There are a small number of special constructs you can use in various cases to insert comments or characters that would otherwise be treated specially. Here's a summary -

- 1.`<%-- comment --%>`-A JSP comment. Ignored by the JSP engine.
- 2.`<!-- comment -->`-An HTML comment. Ignored by the browser.
- 3.`<%`-Represents static `<%` literal.
- 4.`%>`-Represents static `%>` literal.
- 5.`'`-A single quote in an attribute that uses single quotes.
- 6.`"`-A double quote in an attribute that uses double quotes.





# SERVLETS

Java Servlets are programs that run on a Web or Application 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.

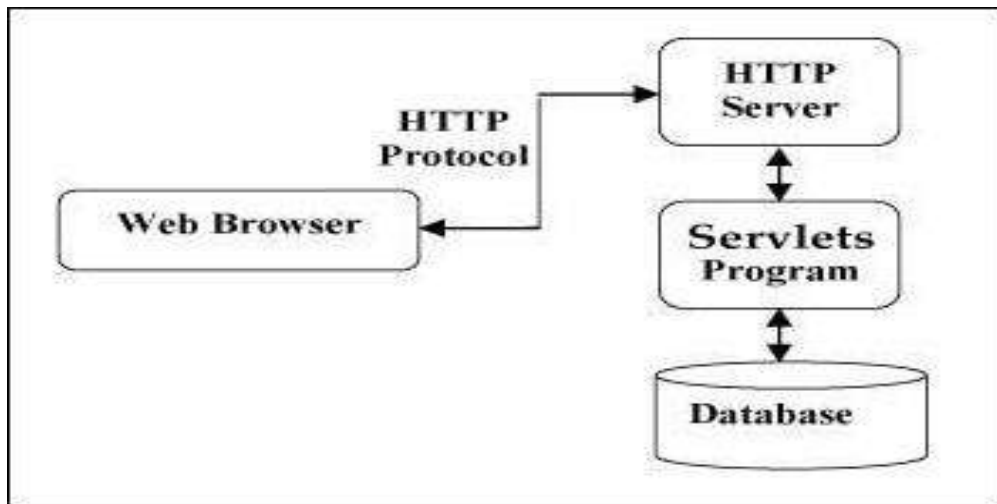
Using Servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.

Java Servlets often serve the same purpose as programs implemented using the Common Gateway Interface (CGI). But Servlets offer several advantages in comparison with the CGI.

- Performance is significantly better.
- 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.
- Java security manager on the server enforces a set of restrictions to protect the resources on a server machine. So servlets are trusted.
- The full functionality of the Java class libraries is available to a servlet. It can communicate with applets, databases, or other software via the sockets and RMI mechanisms that you have seen already.

## Servlets Architecture:-

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



## Servlets Tasks:-

Servlets perform the following major tasks -

- 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.
- Read the implicit HTTP request data sent by the clients (browsers). This includes cookies, media types and compression schemes the browser understands, and so forth.
- Process the data and generate the results. This process may require talking to a database, executing an RMI or CORBA call, invoking a Web service, or computing the response directly.
- 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.
- Send the implicit HTTP response to the clients (browsers). This includes telling the browsers or other clients what type of document is being returned (e.g., HTML), setting cookies and caching parameters, and other such tasks.

## Servlets Packages:-

Java Servlets are Java classes run by a web server that has an interpreter that supports the Java Servlet specification. Servlets can be created using the `javax.servlet` and `javax.servlet.http` packages, which are a standard part of the Java's enterprise edition, an expanded version of the Java class library that supports large-scale development projects.

These classes implement the Java Servlet and JSP specifications. At the

time of writing this tutorial, the versions are Java Servlet 2.5 and JSP 2.1.

Java servlets have been created and compiled just like any other Java class. After you install the servlet packages and add them to your computer's Classpath, you can compile servlets with the JDK's Java compiler or any other current compiler.

## **Life Cycle:-**

A servlet life cycle can be defined as the entire process from its creation till the destruction. The following are the paths followed by a servlet.

- The servlet is initialized by calling the `init()` method.
- The servlet calls `service()` method to process a client's request.
- The servlet is terminated by calling the `destroy()` method.
- Finally, servlet is garbage collected by the garbage collector of the JVM.

Now let us discuss the life cycle methods in detail.

### **The `init()` Method:-**

The `init` method is called only once. It is called only when the servlet is created, and not called for any user requests afterwards. So, it is used for one-time initializations, just as with the `init` method of applets.

The servlet is normally created when a user first invokes a URL corresponding to the servlet, but you can also specify that the servlet be loaded when the server is first started.

When a user invokes a servlet, a single instance of each servlet gets created, with each user request resulting in a new thread that is handed off to `doGet` or `doPost` as appropriate. The `init()` method simply creates or loads some data that will be used throughout the life of the servlet.

The `init` method definition looks like this -

```
public void init() throws ServletException {  
    // Initialization code...  
}
```

### **The `service()` Method :-**

The service() method is the main method to perform the actual task. The servlet container (i.e. web server) calls the service() method to handle requests coming from the client( browsers) and to write the formatted response back to the client.

Each time the server receives a request for a servlet, the server spawns a new thread and calls service. The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.

Here is the signature of this method -

```
public void service(ServletRequest request, ServletResponse response)
throws ServletException, IOException {
}
```

The service () method is called by the container and service method invokes doGet, doPost, doPut, doDelete, etc. methods as appropriate. So you have nothing to do with service() method but you override either doGet() or doPost() depending on what type of request you receive from the client.

The doGet() and doPost() are most frequently used methods with in each service request. Here is the signature of these two methods.

## **The doGet() Method:-**

A GET request results from a normal request for a URL or from an HTML form that has no METHOD specified and it should be handled by doGet() method.

```
public void doGet(HttpServletRequest request, HttpServletResponse
response)
throws ServletException, IOException {
// Servlet code
}
```

## **The doPost() Method:-**

A POST request results from an HTML form that specifically lists POST as the METHOD and it should be handled by doPost() method.

```
public void doPost(HttpServletRequest request, HttpServletResponse
response)
```

```
throws ServletException, IOException {  
// Servlet code  
}
```

## The destroy() Method:-

The destroy() method is called only once at the end of the life cycle of a servlet. This method gives your servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities.

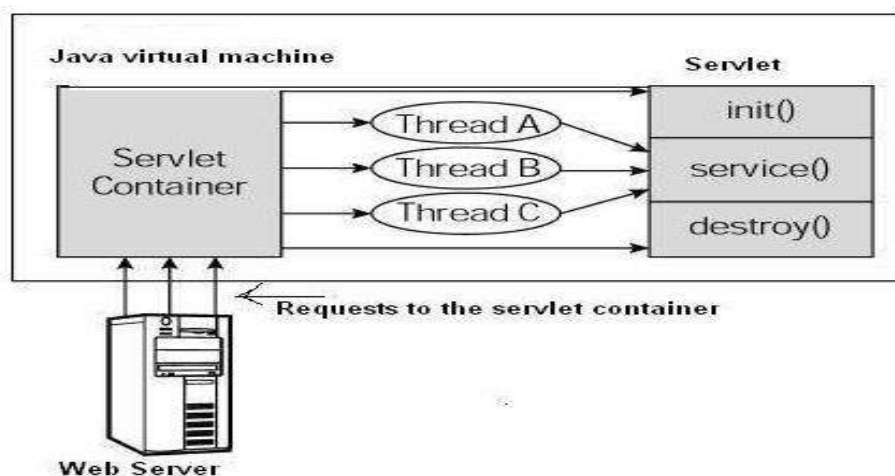
After the destroy() method is called, the servlet object is marked for garbage collection. The destroy method definition looks like this -

```
public void destroy() {  
// Finalization code...  
}
```

## Architecture Diagram:-

The following figure depicts a typical servlet life-cycle scenario.

- First the HTTP requests coming to the server are delegated to the servlet container.
- The servlet container loads the servlet before invoking the service() method.
- Then the servlet container handles multiple requests by spawning multiple threads, each thread executing the service() method of a single instance of the servlet.



## Steps to Create Servlet:-

To create a Servlet application you need to follow the below mentioned steps. These steps are common for all the Web server. In our example we are using Apache Tomcat server. Apache Tomcat is an open source web server for testing servlets and JSP technology. Download latest version of Tomcat Server and install it on your machine.

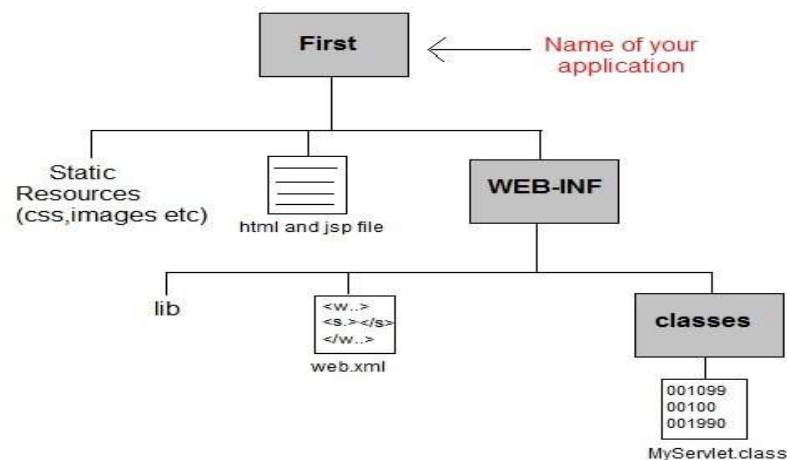
After installing Tomcat Server on your machine follow the below mentioned steps :

- 1.Create directory structure for your application.
- 2.Create a Servlet
- 3.Compile the Servlet
- 4.Create Deployment Descriptor for your application
- 5.Start the server and deploy the application

All these 5 steps are explained in details below, lets create our first Servlet Application.

## 1. Creating the Directory Structure:-

Sun Microsystem defines a unique directory structure that must be followed to create a servlet application.



Create the above directory structure inside Apache-Tomcat\webapps directory. All HTML, static files(images, css etc) are kept directly under Web application folder. While all the Servlet classes are kept inside classes folder.

The web.xml (deployment descriptor) file is kept under WEB-INF folder.

## 2. Creating a Servlet:-

There are three different ways to create a servlet.

- By implementing Servlet interface
- By extending GenericServlet class
- By extending HttpServlet class

But mostly a servlet is created by extending HttpServlet abstract class. As discussed earlier HttpServlet gives the definition of service() method of the Servlet interface. The servlet class that we will create should not override service() method. Our servlet class will override only doGet() or doPost() method.

When a request comes in for the servlet, the Web Container calls the servlet's service() method and depending on the type of request the service() method calls either the doGet() or doPost() method.

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
```

```
public MyServlet extends HttpServlet
{
    public void doGet(HttpServletRequest request,HttpServletResponse
    response)
    throws ServletException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("");
        out.println("

```

## Hello Readers

```
";
        out.println("");
    }
}
```

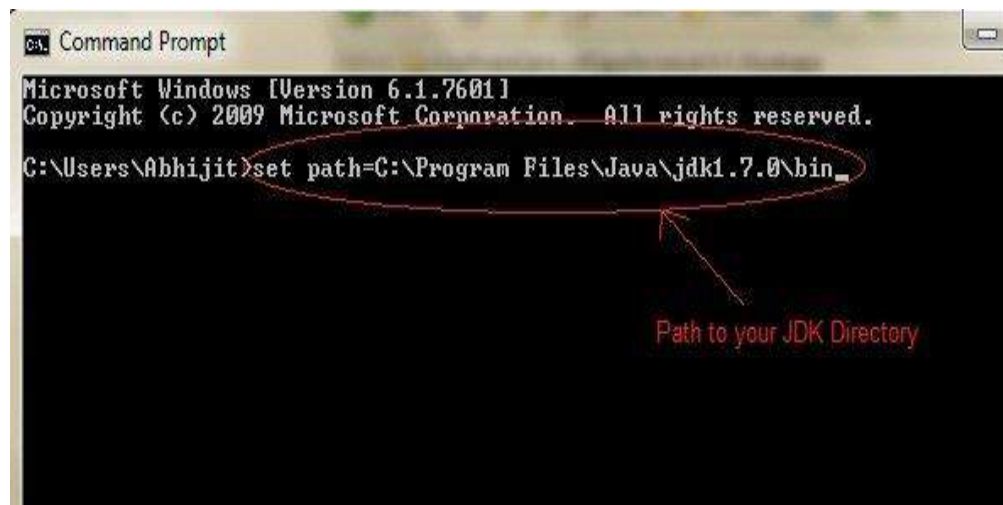
Write above code in a notepad file and save it as MyServlet.java anywhere on your PC. Compile it(explained in next step) from there and paste the class file into WEB-INF/classes/ directory that you have to create inside Tomcat/webapps directory.

### 3. Compiling a Servlet:-

To compile a Servlet a JAR file is required. Different servers require different JAR files. In Apache Tomcat server servlet-api.jar file is required to compile a servlet class.

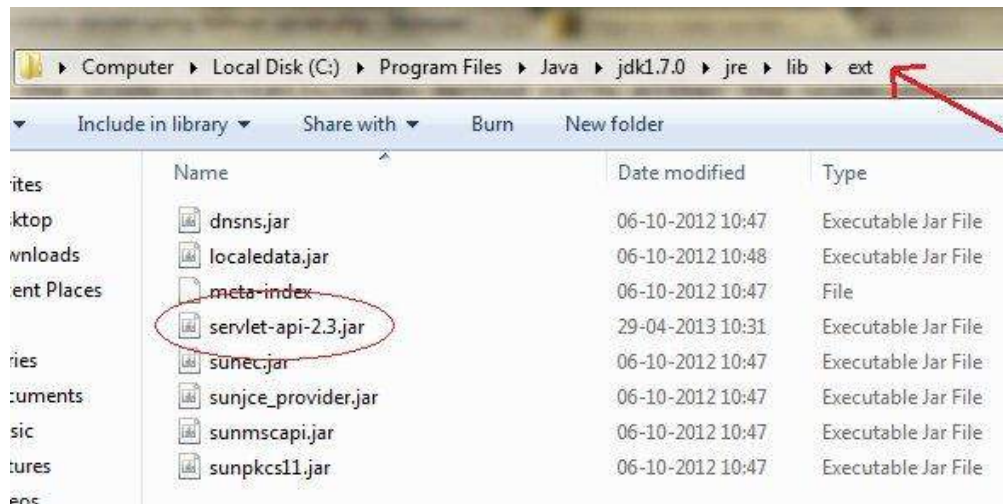
Steps to compile a Servlet:

- Set the Class Path.



- Download servlet-api.jar file.
- Paste the servlet-api.jar file inside Java\jdk\jre\lib\ext directory.





- Compile the Servlet class.



#### 4. Create Deployment Descriptor:-

Deployment Descriptor(DD) is an XML document that is used by Web Container to run Servlets and JSP pages. DD is used for several important purposes such as:

- Mapping URL to Servlet class.
- Initializing parameters.
- Defining Error page.
- Security roles.
- Declaring tag libraries.

We will discuss about all these in details later. Now we will see how to create a simple web.xml file for our web application.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="3.0"
  xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd">
  <servlet>
    <servlet-name>hello</servlet-name>
    <servlet-class>MyServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>hello</servlet-name>
    <url-pattern>/hello</url-pattern>
  </servlet-mapping>
</web-app>
```

First line of any xml document

root tag of web.xml file. All other tag come inside it

this tag maps internal name to fully qualified class name

Give a internal name to your servlet

this tag maps internal name to public URL name

servlet class that you have created

URL name. This is what the user will see to get to the servlet.

## 5. Start the Server:-

Double click on the startup.bat file to start your Apache Tomcat Server.

Or, execute the following command on your windows machine using RUN prompt

C:\apache-tomcat-7.0.14\bin\startup.bat