Advanced JAVA Placement Readiness

SYLLABUS

UNIT I

Applets: Applet Fundamentals – Applet Class – Applet Life Cycle – Steps for developing an Applet Program – Passing values through Parameters – Graphics in an Applet – Event handling.

GUI Applications: Graphical User Interface – Creating Windows – Dialog Boxes – Layout Managers – AWT Component classes – Swing Component classes – Event handling – Other AWT Components – AWT graphics classes – Other Swing controls.

UNIT II

Networking: Basics – Networking in Java – Socket Programming using TCP/IP – Socket Programming using UDP – URL and InetAddress Classes.

Java Database Connectivity: Types of drivers – JDBC Architecture – JDBC Classes and Interfaces – Basic steps in developing JDBC application – Creating a new database and table with JDBC – Working with Database metadata.

UNIT III

Servlets: Basics – Advantages – Servlet alternatives – strengths – Architectures – Servlet Life Cycle – Generic Servlet – HTTP Servlet – Passing parameters – Retrieving parameters – server side include – Cookies – Filters.

UNIT IV

Java Server Pages: Overview – JSP and HTTP – JSP Engines – Working of JSP – Anatomy of JSP – JSP Syntax – Creating simple JSP page – Components of JSP – Implicit Objects.

UNIT V

Web Programming – Client Side Programming: Client Side Programming technologies – form design with HTML and CSS – Client side Validation using JavaScript – Content Structuring using XML – Adding interactivity with AJAX.

Web Programming – Server Side Programming: Web Servers – Handling Request and Response – Database Access – Session Management.

Text Book

Java Programming for Core and Advanced Learners – Sagayaraj, Denis, Karthik and Gajalakshmi, University Press, 2018.

UNIT I

APPLETS

Java Applet

Applet is a special type of program that is embedded in the webpage to generate the dynamic content. It runs inside the browser and works at client side.

Advantage of Applet

There are many advantages of applet. They are as follows:

- It works at client side so less response time.
- Secured
- It can be executed by browsers running under many platforms, including Linux, Windows, Mac Os etc.

Drawback of Applet

- Plug-in is required at client browser to execute applet.

Lifecycle of Java Applet

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

Lifecycle methods for Applet:

The java.applet.Applet class 4 life cycle methods and java.awt.Component class provides 1 life cycle methods for an Applet.

java.applet.Applet class

For creating any applet java.applet.Applet class must be inherited. It provides 4 life cycle methods of applet.

1. **public void init():** is used to initialized the Applet. It is invoked only once.
2. **public void start():** is invoked after the init() method or browser is maximized. It is used to start the Applet.
3. **public void stop():** is used to stop the Applet. It is invoked when Applet is stop or browser is minimized.
4. **public void destroy():** is used to destroy the Applet. It is invoked only once.

java.awt.Component class

The Component class provides 1 life cycle method of applet.

1. **public void paint(Graphics g):** is used to paint the Applet. It provides Graphics class object that can be used for drawing oval, rectangle, arc etc.

How to run an Applet?

There are two ways to run an applet

1. By html file.
2. By appletViewer tool (for testing purpose).

Simple example of Applet by html file:

To execute the applet by html file, create an applet and compile it. After that create an html file and place the applet code in html file. Now click the html file.

```
//First.java
import java.applet.Applet;
import java.awt.Graphics;
public class First extends Applet
{
public void paint(Graphics g)
{
g.drawString("welcome",150,150);
}
}
```

myapplet.html

```
<html>
<body>
<applet code="First.class" width="300" height="300">
</applet>
</body>
</html>
```

Simple example of Applet by appletviewer tool:

To execute the applet by appletviewer tool, create an applet that contains applet tag in comment and compile it. After that run it by: appletviewer First.java. Now Html file is not required but it is for testing purpose only.

```
//First.java
import java.applet.Applet;
import java.awt.Graphics;
public class First extends Applet
{
public void paint(Graphics g){
g.drawString("welcome to applet",150,150);
}
}
/*
<applet code="First.class" width="300" height="300">
</applet>
*/
```

To execute the applet by appletviewer tool, write in command prompt:

```
c:\>javac First.java
c:\>appletviewer First.java
```

Displaying Graphics in Applet

java.awt.Graphics class provides many methods for graphics programming.

Commonly used methods of Graphics class:

1. **public abstract void drawString(String str, int x, int y):** is used to draw the specified string.
2. **public void drawRect(int x, int y, int width, int height):** draws a rectangle with the specified width and height.
3. **public abstract void fillRect(int x, int y, int width, int height):** is used to fill rectangle with the default color and specified width and height.
4. **public abstract void drawOval(int x, int y, int width, int height):** is used to draw oval with the specified width and height.
5. **public abstract void fillOval(int x, int y, int width, int height):** is used to fill oval with the default color and specified width and height.
6. **public abstract void drawLine(int x1, int y1, int x2, int y2):** is used to draw line between the points(x1, y1) and (x2, y2).
7. **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** is used draw the specified image.
8. **public abstract void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used draw a circular or elliptical arc.
9. **public abstract void fillArc(int x, int y, int width, int height, int startAngle, int arcAngle):** is used to fill a circular or elliptical arc.
10. **public abstract void setColor(Color c):** is used to set the graphics current color to the specified color.
11. **public abstract void setFont(Font font):** is used to set the graphics current font to the specified font.

Example of Graphics in applet:

```
import java.applet.Applet;
import java.awt.*;
public class GraphicsDemo extends Applet
{
public void paint(Graphics g)
{
g.setColor(Color.red);
g.drawString("Welcome",50, 50);
g.drawLine(20,30,20,300);
g.drawRect(70,100,30,30);
g.fillRect(170,100,30,30);
g.drawOval(70,200,30,30);
```

```
g.setColor(Color.pink);
g.fillOval(170,200,30,30);
g.drawArc(90,150,30,30,30,270);
g.fillArc(270,150,30,30,0,180);
}
}
```

myapplet.html

```
<html>
<body>
<applet code="GraphicsDemo.class" width="300" height="300">
</applet>
</body>
</html>
```

Displaying Image in Applet

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

Syntax of drawImage() method:

1. **public abstract boolean drawImage(Image img, int x, int y, ImageObserver observer):** is used draw the specified image.

How to get the object of Image:

The java.applet.Applet class provides getImage() method that returns the object of Image.

Syntax:

1. **public** Image getImage(URL u, String image){}

Other required methods of Applet class to display image:

1. **public URL getDocumentBase():** is used to return the URL of the document in which applet is embedded.
2. **public URL getCodeBase():** is used to return the base URL.

Example of displaying image in applet:

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

  public void paint(Graphics g)
```

```
{
   g.drawImage(picture, 30,30, this);
  }
  }
```

In this example, drawImage() method of Graphics class is used to display the image. The 4th argument of drawImage() method of is ImageObserver object. The Component class implements ImageObserver interface. So current class object would also be treated as ImageObserver because Applet class indirectly extends the Component class.

myapplet.html

```
<html>
<body>
<applet code="DisplayImage.class" width="300" height="300">
</applet>
</body>
</html>
```

Animation in Applet

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

Example of animation in applet:

```
import java.awt.*;
import java.applet.*;
public class AnimationExample extends Applet
{
  Image picture;
  public void init()
{
    picture =getImage(getDocumentBase(),"bike_1.gif");
  }
  public void paint(Graphics g)
{
    for(int i=0;i<500;i++){
      g.drawImage(picture, i,30, this);
      try{Thread.sleep(100);}catch(Exception e)
{}
    }
  }
}
```

In this example, drawImage() method of Graphics class is used to display the image. The 4th argument of drawImage() method of is ImageObserver object. The Component class implements ImageObserver interface. So current class object would also be treated as ImageObserver

because Applet class indirectly extends the Component class.

myapplet.html

```
<html>
<body>
<applet code="DisplayImage.class" width="300" height="300">
</applet>
</body>
</html>
```

EventHandling in Applet

As we perform event handling in AWT or Swing, we can perform it in applet also. Let's see the simple example of event handling in applet that prints a message by click on the button.

Example of EventHandling in applet:

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

In this example, we have created all the controls in init() method because it is invoked only once.

myapplet.html

```
<html>
<body>
<applet code="EventApplet.class" width="300" height="300">
</applet>
```

```
</body>
</html>
```

JApplet class in Applet

As we prefer Swing to AWT. Now we can use JApplet that can have all the controls of swing. The JApplet class extends the Applet class.

Example of EventHandling in JApplet:

```
import java.applet.*;
import javax.swing.*;
import java.awt.event.*;
public class EventJApplet extends JApplet implements ActionListener
{
JButton b;
JTextField tf;
public void init()
{
tf=new JTextField();
tf.setBounds(30,40,150,20);
b=new JButton("Click");
b.setBounds(80,150,70,40);
add(b);add(tf);
b.addActionListener(this);
setLayout(null);
}
public void actionPerformed(ActionEvent e)
{
tf.setText("Welcome");
}
}
```

In this example, we have created all the controls in init() method because it is invoked only once.

myapplet.html

```
<html>
<body>
<applet code="EventJApplet.class" width="300" height="300">
</applet>
</body>
</html>
```

Painting in Applet

We can perform painting operation in applet by the mouseDragged() method of MouseMotionListener.

Example of Painting in Applet:

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
public class MouseDrag extends Applet implements MouseMotionListener
{
public void init()
{
addMouseMotionListener(this);
setBackground(Color.red);
}
public void mouseDragged(MouseEvent me)
{
Graphics g=getGraphics();
g.setColor(Color.white);
g.fillOval(me.getX(),me.getY(),5,5);
}
public void mouseMoved(MouseEvent me)
{}
}
```

In this example, getX() and getY() method of MouseEvent is used to get the current x-axis and y-axis. The getGraphics() method of Component class returns the object of Graphics.

myapplet.html

```
<html>
<body>
<applet code="MouseDrag.class" width="300" height="300">
</applet>
</body>
</html>
```

Parameter in Applet

We can get any information from the HTML file as a parameter. For this purpose, Applet class provides a method named getParameter().

Syntax:

1. **public** String getParameter(String parameterName)

Example of using parameter in Applet:

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

```
{
String str=getParameter("msg");
g.drawString(str,50, 50);
}
}
```

myapplet.html

```
<html>
<body>
<applet code="UseParam.class" width="300" height="300">
<param name="msg" value="Welcome to applet">
</applet>
</body>
</html>
```

GUI APPLICATIONS

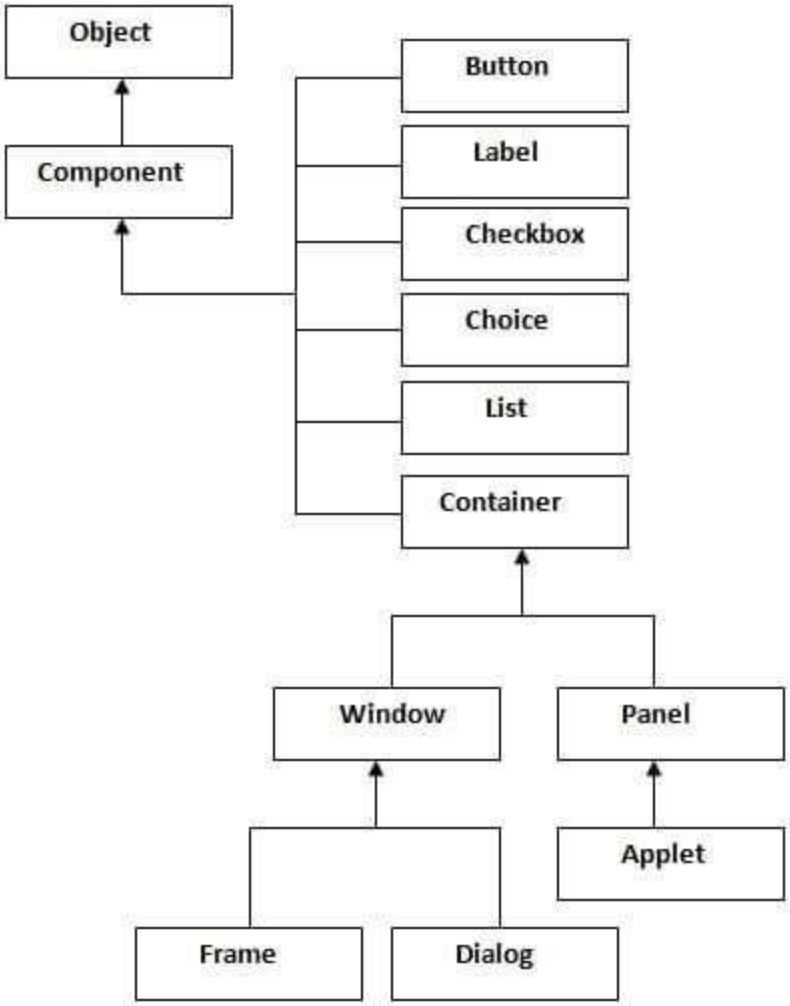
Java AWT

Java AWT (Abstract Window Toolkit) is *an API to develop GUI or window-based applications* in java. Java AWT components are platform-dependent i.e. components are displayed according to the view of operating system. AWT is heavyweight i.e. its components are using the resources of OS.

The java.awt package provides classes for AWT API such as TextField, Label, TextArea, RadioButton, CheckBox, Choice, List etc.

Java AWT Hierarchy

The hierarchy of Java AWT classes are given,



Container

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

Window

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

Panel

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

Frame

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

Useful Methods of Component class

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

Java AWT Example

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

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

AWT Example by Inheritance

A simple example of AWT where we are inheriting Frame class. Here, we are showing Button component on the Frame.

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

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

AWT Example by Association

A simple example of AWT where we are creating instance of Frame class. Here, we are showing Button component on the Frame.

```
import java.awt.*;
class First2
{
First2()
{
Frame f=new Frame();
Button b=new Button("click me");
b.setBounds(30,50,80,30);
f.add(b);
f.setSize(300,300);
f.setLayout(null);
f.setVisible(true);
}
public static void main(String args[])
{
First2 f=new First2();
}
}
```

Java AWT Panel

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

AWT Panel class declaration

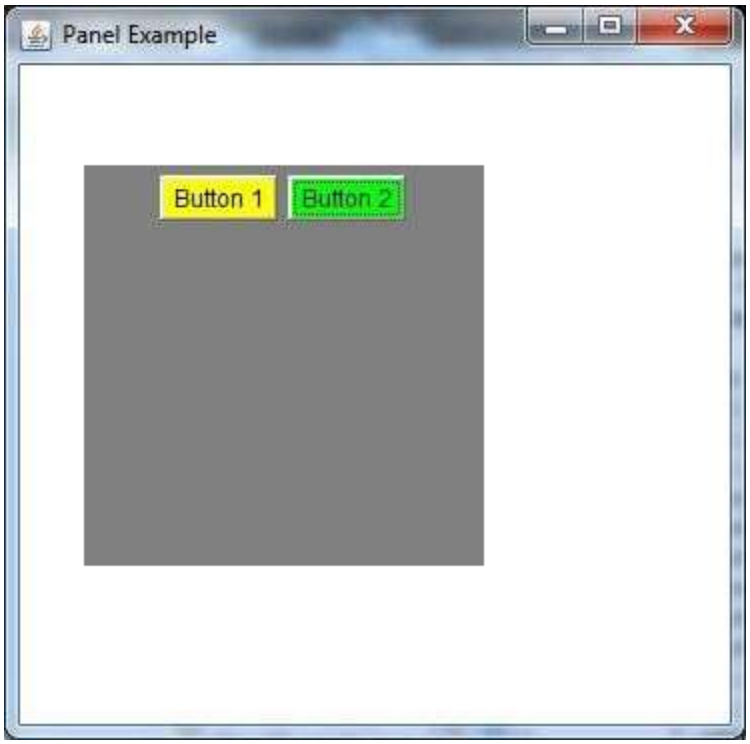
1. **public class** Panel **extends** Container **implements** Accessible

Java AWT Panel Example

```
import java.awt.*;
public class PanelExample
{
PanelExample()
 {
 Frame f= new Frame("Panel Example");
 Panel panel=new Panel();
 panel.setBounds(40,80,200,200);
 panel.setBackground(Color.gray);
 Button b1=new Button("Button 1");
 b1.setBounds(50,100,80,30);
 b1.setBackground(Color.yellow);
 Button b2=new Button("Button 2");
 b2.setBounds(100,100,80,30);
 b2.setBackground(Color.green);
 panel.add(b1); panel.add(b2);
```

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

Output:



Java AWT Dialog

The Dialog control represents a top level window with a border and a title used to take some form of input from the user. It inherits the Window class. Unlike Frame, it doesn't have maximize and minimize buttons.

Frame vs Dialog

Frame and Dialog both inherits Window class. Frame has maximize and minimize buttons but Dialog doesn't have.

AWT Dialog class declaration

1. **public class** Dialog **extends** Window

Java AWT Dialog Example

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

```
import java.awt.event.*;
public class DialogExample
{
   private static Dialog d;
   DialogExample()
{
      Frame f= new Frame();
      d = new Dialog(f , "Dialog Example", true);
      d.setLayout( new FlowLayout() );
      Button b = new Button ("OK");
      b.addActionListener ( new ActionListener()
      {
         public void actionPerformed( ActionEvent e )
         {
            DialogExample.d.setVisible(false);
         }
      });
      d.add( new Label ("Click button to continue."));
      d.add(b);
      d.setSize(300,300);
      d.setVisible(true);
   }
   public static void main(String args[])
   {
      new DialogExample();
   }
}
```

Output:

Java AWT Button

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

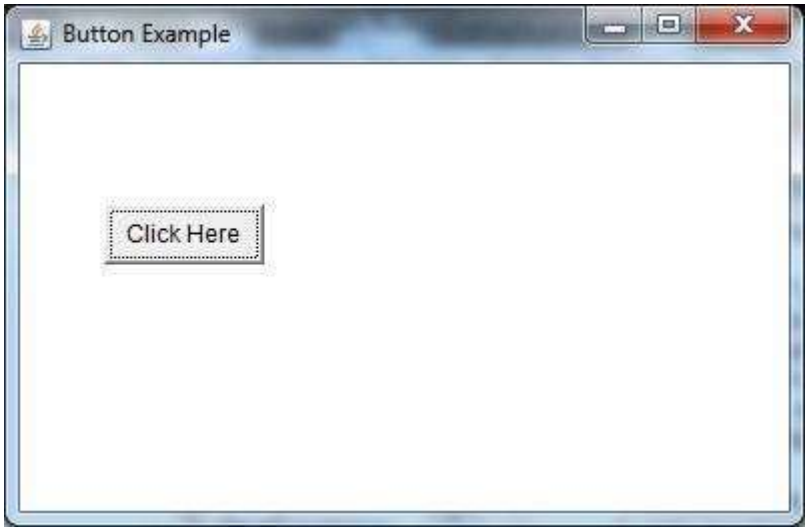
AWT Button Class declaration

1. **public class** Button **extends** Component **implements** Accessible

Java AWT Button Example

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

Output:

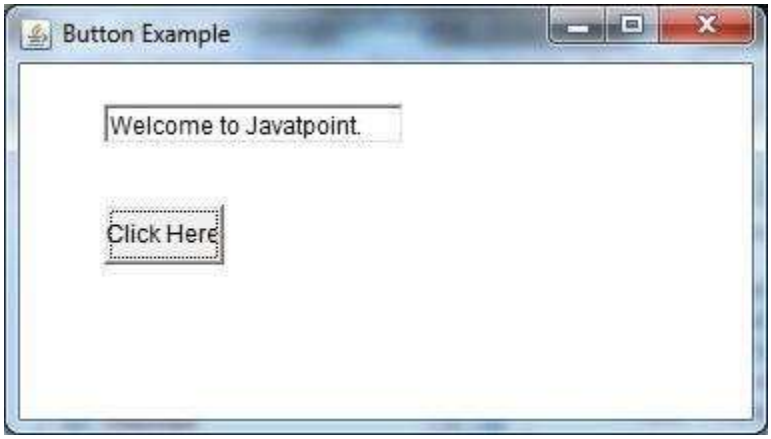


Java AWT Button Example with ActionListener

```
import java.awt.*;
import java.awt.event.*;
public class ButtonExample
{
public static void main(String[] args)
```

```
{
   Frame f=new Frame("Button Example");
   final TextField tf=new TextField();
   tf.setBounds(50,50, 150,20);
   Button b=new Button("Click Here");
   b.setBounds(50,100,60,30);
   b.addActionListener(new ActionListener()
{
   public void actionPerformed(ActionEvent e)
{
         tf.setText("Welcome to Javatpoint.");
      }
   });
   f.add(b);f.add(tf);
   f.setSize(400,400);
   f.setLayout(null);
   f.setVisible(true);
}
}
```

Output:



Java AWT Label

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

AWT Label Class Declaration

1. **public class** Label **extends** Component **implements** Accessible

Java Label Example

```
import java.awt.*;
class LabelExample
```

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

Output:

Java AWT Label Example with ActionListener

```
import java.awt.*;
import java.awt.event.*;
public class LabelExample extends Frame implements ActionListener
{
   TextField tf; Label l; Button b;
   LabelExample()
{
      tf=new TextField();
      tf.setBounds(50,50, 150,20);
      l=new Label();
      l.setBounds(50,100, 250,20);
```

```
        b=new Button("Find IP");
        b.setBounds(50,150,60,30);
        b.addActionListener(this);
        add(b);add(tf);add(l);
        setSize(400,400);
        setLayout(null);
        setVisible(true);
    }
    public void actionPerformed(ActionEvent e)
{
        Try
{
        String host=tf.getText();
        String ip=java.net.InetAddress.getByName(host).getHostAddress();
        l.setText("IP of "+host+" is: "+ip);
        }
catch(Exception ex){System.out.println(ex);
}
    }
    public static void main(String[] args)
{
        new LabelExample();
    }
}
```

Output:

Java AWT TextField

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

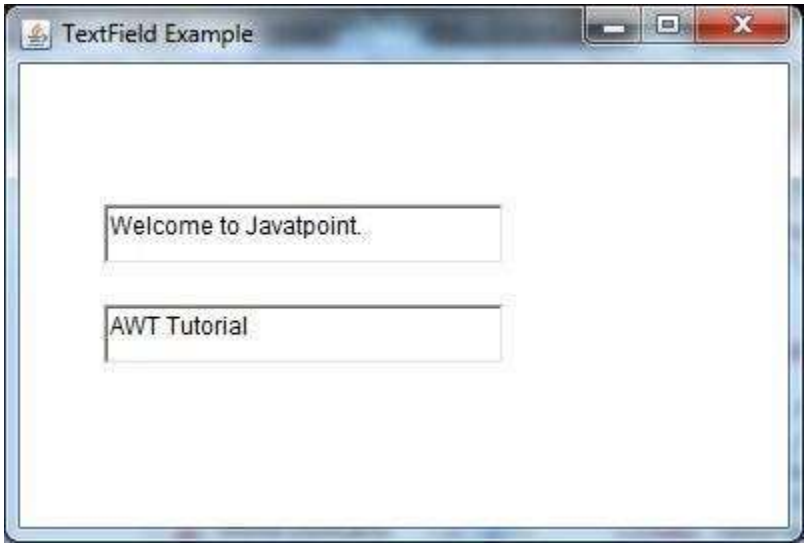
AWT TextField Class Declaration

1. **public class** TextField **extends** TextComponent

Java AWT TextField Example

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

Output:



Java AWT TextArea

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

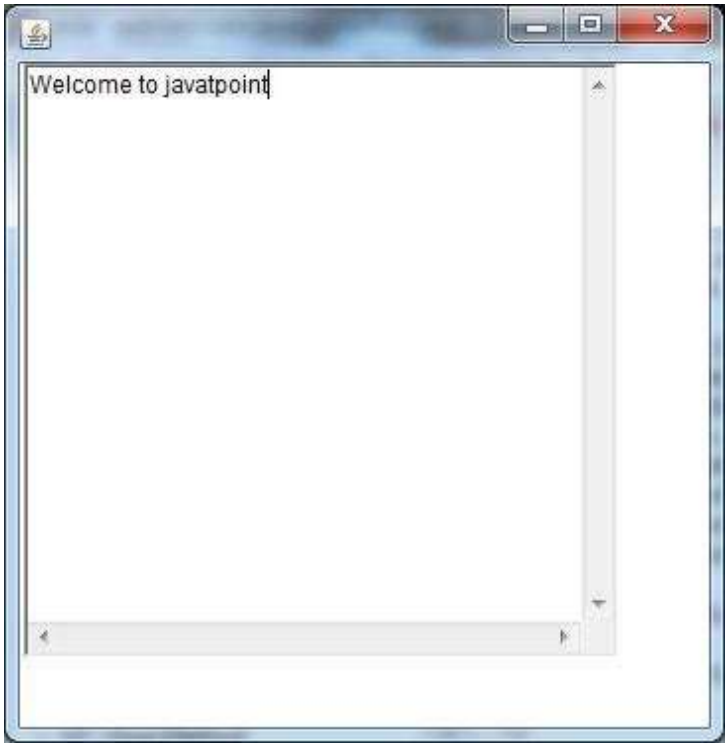
AWT TextArea Class Declaration

1. **public class** TextArea **extends** TextComponent

Java AWT TextArea Example

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

Output:



Java AWT Checkbox

The Checkbox class is used to create a checkbox. It is used to turn an option on (true) or off (false). Clicking on a Checkbox changes its state from "on" to "off" or from "off" to "on".

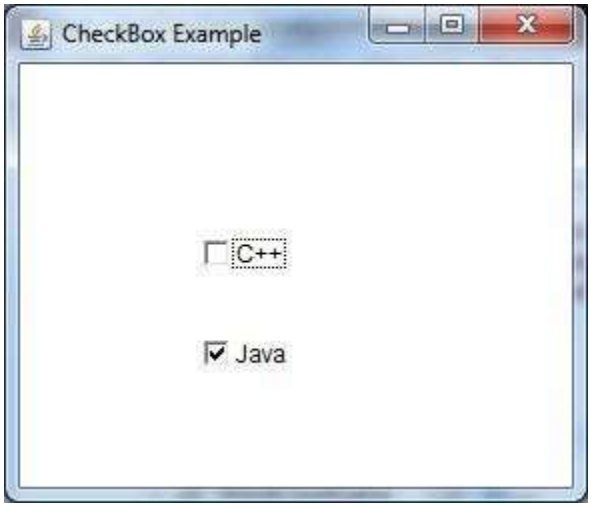
AWT Checkbox Class Declaration

1. **public class** Checkbox **extends** Component **implements** ItemSelectable, Accessible

Java AWT Checkbox Example

```
import java.awt.*;
public class CheckboxExample
{
    CheckboxExample()
{
      Frame f= new Frame("Checkbox Example");
       Checkbox checkbox1 = new Checkbox("C++");
       checkbox1.setBounds(100,100, 50,50);
       Checkbox checkbox2 = new Checkbox("Java", true);
       checkbox2.setBounds(100,150, 50,50);
       f.add(checkbox1);
       f.add(checkbox2);
       f.setSize(400,400);
       f.setLayout(null);
       f.setVisible(true);
    }
public static void main(String args[])
{
   new CheckboxExample();
}
}
```

Output:



Java AWT Choice

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

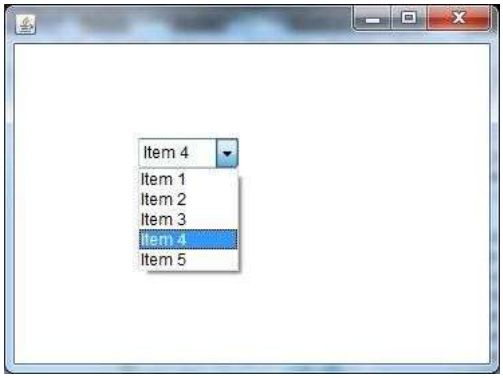
AWT Choice Class Declaration

1. **public class** Choice **extends** Component **implements** ItemSelectable, Accessible

Java AWT Choice Example

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

Output:



Java AWT List

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

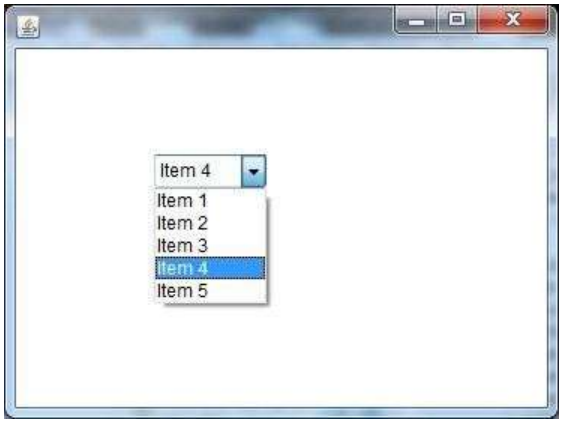
AWT List class Declaration

1. **public class** List **extends** Component **implements** ItemSelectable, Accessible

Java AWT List Example

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

Output:



Java AWT Scrollbar

The object of Scrollbar class is used to add horizontal and vertical scrollbar. Scrollbar is a GUI component allows us to see invisible number of rows and columns.

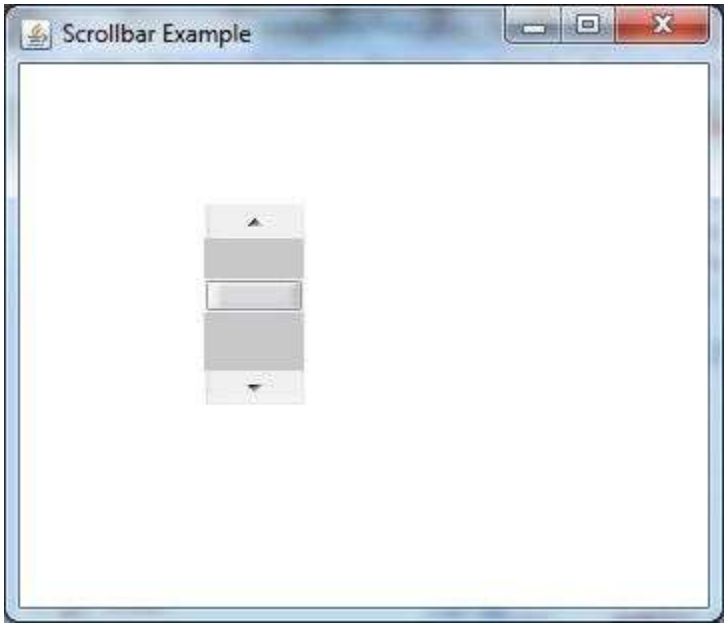
AWT Scrollbar class declaration

1. **public class** Scrollbar **extends** Component **implements** Adjustable, Accessible

Java AWT Scrollbar Example

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

Output:



Java AWT MenuItem and Menu

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

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

AWT MenuItem class declaration

1. **public class** MenuItem **extends** MenuComponent **implements** Accessible

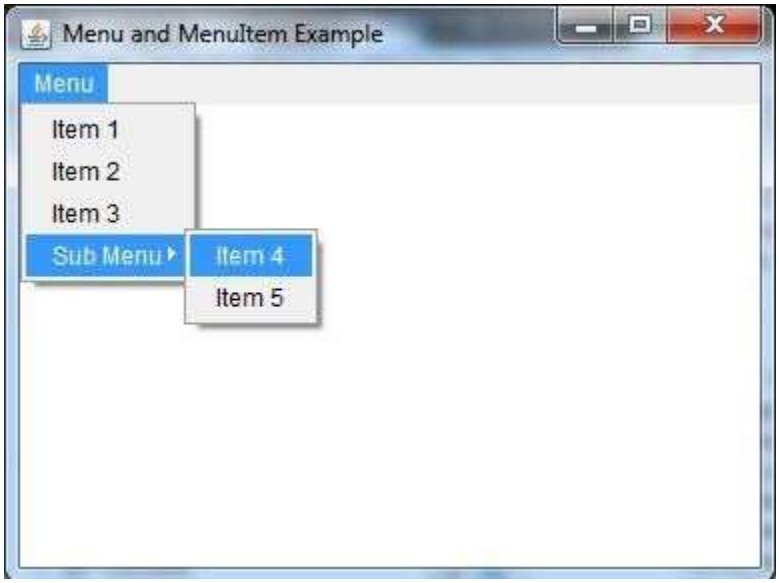
AWT Menu class declaration

1. **public class** Menu **extends** MenuItem **implements** MenuContainer, Accessible

Java AWT MenuItem and Menu Example

```
import java.awt.*;
class MenuExample
{
   MenuExample()
{
      Frame f= new Frame("Menu and MenuItem Example");
      MenuBar mb=new MenuBar();
      Menu menu=new Menu("Menu");
      Menu submenu=new Menu("Sub Menu");
      MenuItem i1=new MenuItem("Item 1");
      MenuItem i2=new MenuItem("Item 2");
      MenuItem i3=new MenuItem("Item 3");
      MenuItem i4=new MenuItem("Item 4");
      MenuItem i5=new MenuItem("Item 5");
      menu.add(i1);
      menu.add(i2);
      menu.add(i3);
      submenu.add(i4);
      submenu.add(i5);
      menu.add(submenu);
      mb.add(menu);
      f.setMenuBar(mb);
      f.setSize(400,400);
      f.setLayout(null);
      f.setVisible(true);
}
public static void main(String args[])
{
new MenuExample();
}
}
```

Output:



Java Swing Tutorial

Java Swing tutorial is a part of Java Foundation Classes (JFC) that is *used to create window-based applications*. It is built on the top of AWT (Abstract Windowing Toolkit) API and entirely written in java. Unlike AWT, Java Swing provides platform-independent and lightweight components.

The javax.swing package provides classes for java swing API such as JButton, JTextField, JTextArea, JRadioButton, JCheckbox, JMenu, JColorChooser etc.

Difference between AWT and Swing

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

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

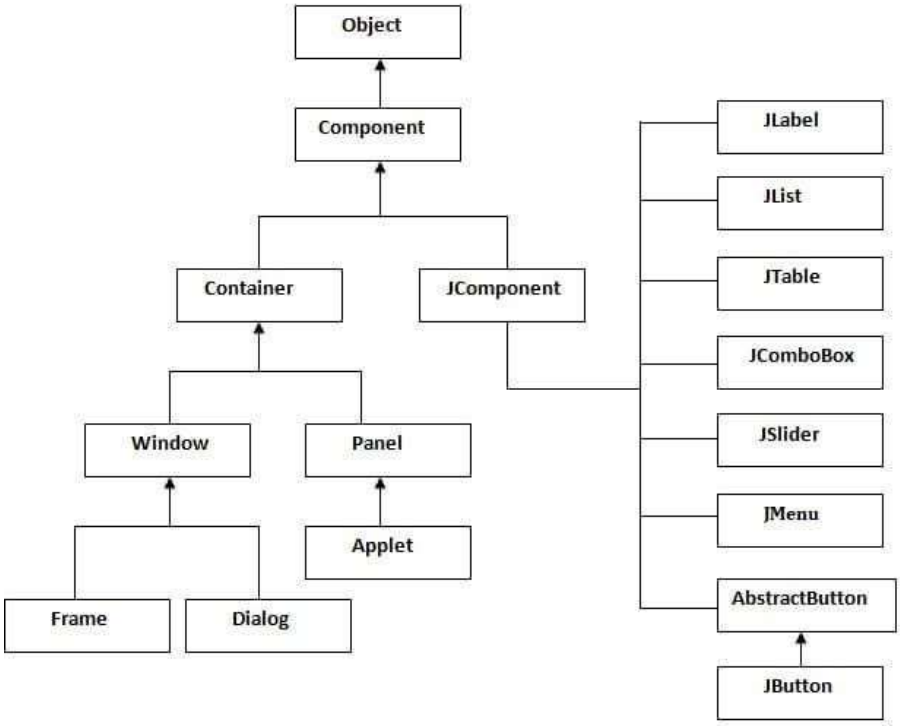
	an interface between model and view.	

What is JFC

The Java Foundation Classes (JFC) are a set of GUI components which simplify the development of desktop applications.

Hierarchy of Java Swing classes

The hierarchy of java swing API is given below.



Commonly used Methods of Component class

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

Method	Description
public void add(Component c)	add a component on another component.
public void setSize(int width,int height)	sets size of the component.
public void setLayout(LayoutManager m)	sets the layout manager for the component.
public void setVisible(boolean b)	sets the visibility of the component. It is by default false.

Java Swing Examples

There are two ways to create a frame:

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

We can write the code of swing inside the main(), constructor or any other method.

Simple Java Swing Example

Simple swing example where we are creating one button and adding it on the JFrame object inside the main() method.

FirstSwingExample.java

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

Example of Swing by Association inside constructor

We can also write all the codes of creating JFrame, JButton and method call inside the java constructor.

Simple.java

```
import javax.swing.*;
public class Simple
{
JFrame f;
```

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

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

Simple example of Swing by inheritance

We can also inherit the JFrame class, so there is no need to create the instance of JFrame class explicitly.

Simple2.java

```
import javax.swing.*;
public class Simple2 extends JFrame
{//inheriting JFrame
JFrame f;
Simple2()
{
JButton b=new JButton("click");//create button
b.setBounds(130,100,100, 40);
add(b);//adding button on frame
setSize(400,500);
setLayout(null);
setVisible(true);
}
public static void main(String[] args)
{
new Simple2();
}}
```

Java JPanel

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

JPanel class declaration

1. **public class** JPanel **extends** JComponent **implements** Accessible

Commonly used Constructors:

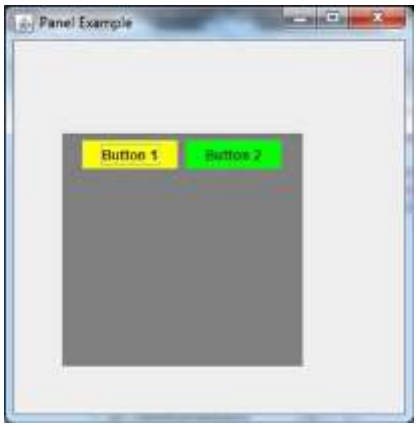
Constructor	Description
JPanel()	It is used to create a new JPanel with a double buffer and a flow layout.
JPanel(boolean isDoubleBuffered)	It is used to create a new JPanel with FlowLayout and the specified buffering strategy.
JPanel(LayoutManager layout)	It is used to create a new JPanel with the specified layout manager.

Java JPanel Example

```
import java.awt.*;
import javax.swing.*;
public class PanelExample
{
    PanelExample()
      {
      JFrame f= new JFrame("Panel Example");
      JPanel panel=new JPanel();
      panel.setBounds(40,80,200,200);
      panel.setBackground(Color.gray);
      JButton b1=new JButton("Button 1");
      b1.setBounds(50,100,80,30);
      b1.setBackground(Color.yellow);
      JButton b2=new JButton("Button 2");
      b2.setBounds(100,100,80,30);
      b2.setBackground(Color.green);
      panel.add(b1); panel.add(b2);
      f.add(panel);
            f.setSize(400,400);
            f.setLayout(null);
            f.setVisible(true);
      }
      public static void main(String args[])
      {
      new PanelExample();
```

```
    }
  }
```

Output:



Java JFrame

The javax.swing.JFrame class is a type of container which inherits the java.awt.Frame class. JFrame works like the main window where components like labels, buttons, textfields are added to create a GUI. Unlike Frame, JFrame has the option to hide or close the window with the help of setDefaultCloseOperation(int) method.

Nested Class

Modifier and Type	Class	Description
protected class	JFrame.AccessibleJFrame	This class implements accessibility support for the JFrame class.

Fields

Modifier and Type	Field	Description
protected AccessibleContext	accessibleContext	The accessible context property.
static int	EXIT_ON_CLOSE	The exit application default window close operation.
protected JRootPane	rootPane	The JRootPane instance that manages the contentPane and optional menuBar for this frame, as well as the glassPane.
protected boolean	rootPaneCheckingEnabled	If true then calls to add and setLayout will be forwarded to the contentPane.

Constructors

Constructor	Description
JFrame()	It constructs a new frame that is initially invisible.
JFrame(GraphicsConfiguration gc)	It creates a Frame in the specified GraphicsConfiguration of a screen device and a blank title.
JFrame(String title)	It creates a new, initially invisible Frame with the specified title.
JFrame(String title, GraphicsConfiguration gc)	It creates a JFrame with the specified title and the specified GraphicsConfiguration of a screen device.

Useful Methods

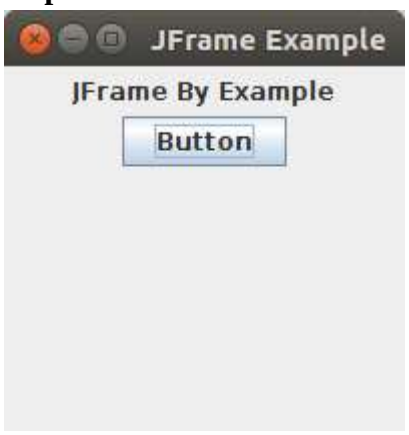
Modifier and Type	Method	Description
protected void	addImpl(Component comp, Object constraints, int index)	Adds the specified child Component.
protected JRootPane	createRootPane()	Called by the constructor methods to create the default rootPane.
protected void	frameInit()	Called by the constructors to init the JFrame properly.
void	setContentPane(Containe contentPane)	It sets the contentPane property
static void	setDefaultLookAndFeelDecorated(boolean defaultLookAndFeelDecorated)	Provides a hint as to whether or not newly created JFrames should have their Window decorations (such as borders, widgets to close the window, title...) provided by the current look and feel.
void	setIconImage(Image image)	It sets the image to be displayed as the icon for this window.
void	setJMenuBar(JMenuBar menubar)	It sets the menubar for this frame.
void	setLayeredPane(JLayeredPane layeredPane)	It sets the layeredPane property.
JRootPane	getRootPane()	It returns the rootPane object for this frame.
TransferHandler	getTransferHandler()	It gets the transferHandler property.

JFrame Example

import java.awt.FlowLayout;

```
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.Jpanel;
public class JFrameExample {
   public static void main(String s[])
{
      JFrame frame = new JFrame("JFrame Example");
      JPanel panel = new JPanel();
      panel.setLayout(new FlowLayout());
      JLabel label = new JLabel("JFrame By Example");
      JButton button = new JButton();
      button.setText("Button");
      panel.add(label);
      panel.add(button);
      frame.add(panel);
      frame.setSize(200, 300);
      frame.setLocationRelativeTo(null);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setVisible(true);
   }
}
```

Output



Java JPasswordField

The object of a JPasswordField class is a text component specialized for password entry. It allows the editing of a single line of text. It inherits JTextField class.

JPasswordField class declaration

The declaration for javax.swing.JPasswordField class.

1. **public class** JPasswordField **extends** JTextField

Commonly used Constructors:

Constructor	Description
JPasswordField()	Constructs a new JPasswordField, with a default document, null starting text string, and 0 column width.
JPasswordField(int columns)	Constructs a new empty JPasswordField with the specified number of columns.
JPasswordField(String text)	Constructs a new JPasswordField initialized with the specified text.
JPasswordField(String text, int columns)	Construct a new JPasswordField initialized with the specified text and columns.

Java JPasswordField Example

```
import javax.swing.*;
public class PasswordFieldExample
{
   public static void main(String[] args)
{
   JFrame f=new JFrame("Password Field Example");
    JPasswordField value = new JPasswordField();
    JLabel l1=new JLabel("Password:");
      l1.setBounds(20,100, 80,30);
       value.setBounds(100,100,100,30);
          f.add(value); f.add(l1);
          f.setSize(300,300);
          f.setLayout(null);
          f.setVisible(true);
}
}
```

Output:

Java JTable

The JTable class is used to display data in tabular form. It is composed of rows and columns.

JTable class declaration

The declaration for javax.swing.JTable class.

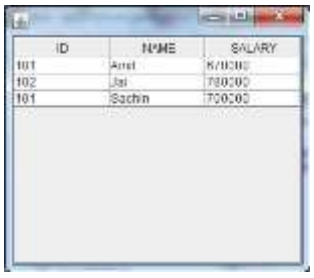
Commonly used Constructors:

Constructor	Description
JTable()	Creates a table with empty cells.
JTable(Object[][] rows, Object[] columns)	Creates a table with the specified data.

Java JTable Example

```
import javax.swing.*;
public class TableExample
{
   JFrame f;
   TableExample(){
   f=new JFrame();
   String data[][]={ {"101","Amit","670000"},
                     {"102","Jai","780000"},
                     {"101","Sachin","700000"}};
   String column[]={"ID","NAME","SALARY"};
   JTable jt=new JTable(data,column);
   jt.setBounds(30,40,200,300);
   JScrollPane sp=new JScrollPane(jt);
   f.add(sp);
   f.setSize(300,400);
   f.setVisible(true);
}
public static void main(String[] args)
{
   new TableExample();
}
}
```

Output:



Java JOptionPane

The JOptionPane class is used to provide standard dialog boxes such as message dialog box, confirm dialog box and input dialog box. These dialog boxes are used to display information or get input from the user. The JOptionPane class inherits JComponent class.

JOptionPane class declaration

1. **public class** JOptionPane **extends** JComponent **implements** Accessible

Common Constructors of JOptionPane class

Constructor	Description
JOptionPane()	It is used to create a JOptionPane with a test message.
JOptionPane(Object message)	It is used to create an instance of JOptionPane to display a message.
JOptionPane(Object message, int messageType	It is used to create an instance of JOptionPane to display a message with specified message type and default options.

Common Methods of JOptionPane class

Methods	Description
JDialog createDialog(String title)	It is used to create and return a new parentless JDialog with the specified title.
static void showMessageDialog(Component parentComponent, Object message)	It is used to create an information-message dialog titled "Message".
static void showMessageDialog(Component parentComponent, Object message, String title, int messageType)	It is used to create a message dialog with given title and messageType.
static int showConfirmDialog(Component parentComponent, Object message)	It is used to create a dialog with the options Yes, No and Cancel; with the title, Select an Option.
static String showInputDialog(Component parentComponent, Object message)	It is used to show a question-message dialog requesting input from the user parented to parentComponent.
void setInputValue(Object newValue)	It is used to set the input value that was selected or input by the user.

Java JOptionPane Example: showMessageDialog()

```
import javax.swing.*;
public class OptionPaneExample
{
JFrame f;
OptionPaneExample()
{
```

```
    f=new JFrame();
    JOptionPane.showMessageDialog(f,"Hello, Welcome to Javatpoint.");
}
public static void main(String[] args)
{
    new OptionPaneExample();
}
}
```

Output:

Java JTabbedPane

The JTabbedPane class is used to switch between a group of components by clicking on a tab with a given title or icon. It inherits JComponent class.

JTabbedPane class declaration

The declaration for javax.swing.JTabbedPane class.

1. **public class** JTabbedPane **extends** JComponent **implements** Serializable, Accessible, SwingCon stants

Commonly used Constructors:

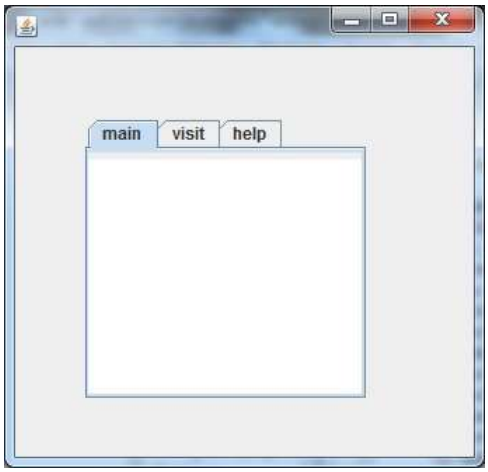
Constructor	Description
JTabbedPane()	Creates an empty TabbedPane with a default tab placement of JTabbedPane.Top.
JTabbedPane(int tabPlacement)	Creates an empty TabbedPane with a specified tab placement.
JTabbedPane(int tabPlacement, int tabLayoutPolicy)	Creates an empty TabbedPane with a specified tab placement and tab layout policy.

Java JTabbedPane Example

```
import javax.swing.*;
public class TabbedPaneExample
{
JFrame f;
TabbedPaneExample()
{
    f=new JFrame();
```

```
    JTextArea ta=new JTextArea(200,200);
    JPanel p1=new JPanel();
    p1.add(ta);
    JPanel p2=new JPanel();
    JPanel p3=new JPanel();
    JTabbedPane tp=new JTabbedPane();
    tp.setBounds(50,50,200,200);
    tp.add("main",p1);
    tp.add("visit",p2);
    tp.add("help",p3);
    f.add(tp);
    f.setSize(400,400);
    f.setLayout(null);
    f.setVisible(true);
}
public static void main(String[] args)
{
    new TabbedPaneExample();
}}
```

Output:



Java JTree

The JTree class is used to display the tree structured data or hierarchical data. JTree is a complex component. It has a 'root node' at the top most which is a parent for all nodes in the tree. It inherits JComponent class.

JTree class declaration

The declaration for javax.swing.JTree class.

1. **public class** JTree **extends** JComponent **implements** Scrollable, Accessible

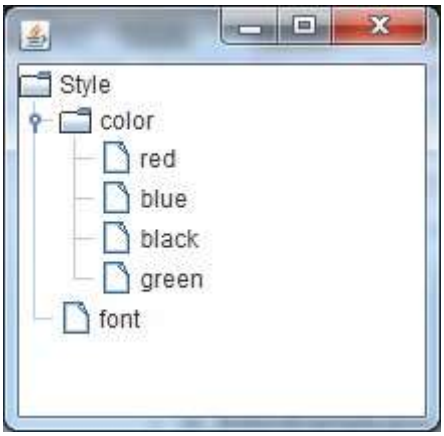
Commonly used Constructors:

Constructor	Description
JTree()	Creates a JTree with a sample model.
JTree(Object[] value)	Creates a JTree with every element of the specified array as the child of a new root node.
JTree(TreeNode root)	Creates a JTree with the specified TreeNode as its root, which displays the root node.

Java JTree Example

```
import javax.swing.*;
import javax.swing.tree.DefaultMutableTreeNode;
public class TreeExample
{
JFrame f;
TreeExample()
{
   f=new JFrame();
   DefaultMutableTreeNode style=new DefaultMutableTreeNode("Style");
   DefaultMutableTreeNode color=new DefaultMutableTreeNode("color");
   DefaultMutableTreeNode font=new DefaultMutableTreeNode("font");
   style.add(color);
   style.add(font);
   DefaultMutableTreeNode red=new DefaultMutableTreeNode("red");
   DefaultMutableTreeNode blue=new DefaultMutableTreeNode("blue");
   DefaultMutableTreeNode black=new DefaultMutableTreeNode("black");
   DefaultMutableTreeNode green=new DefaultMutableTreeNode("green");
   color.add(red); color.add(blue); color.add(black); color.add(green);
   JTree jt=new JTree(style);
   f.add(jt);
   f.setSize(200,200);
   f.setVisible(true);
}
public static void main(String[] args)
{
   new TreeExample();
}}
```

Output:



Java JProgressBar

The JProgressBar class is used to display the progress of the task. It inherits JComponent class.

JProgressBar class declaration

The declaration for javax.swing.JProgressBar class.

1. **public class** JProgressBar **extends** JComponent **implements** SwingConstants, Accessible

Commonly used Constructors:

Constructor	Description
JProgressBar()	It is used to create a horizontal progress bar but no string text.
JProgressBar(int min, int max)	It is used to create a horizontal progress bar with the specified minimum and maximum value.
JProgressBar(int orient)	It is used to create a progress bar with the specified orientation, it can be either Vertical or Horizontal by using SwingConstants.VERTICAL and SwingConstants.HORIZONTAL constants.
JProgressBar(int orient, int min, int max)	It is used to create a progress bar with the specified orientation, minimum and maximum value.

Commonly used Methods:

Method	Description
void setStringPainted(boolean b)	It is used to determine whether string should be displayed.
void setString(String s)	It is used to set value to the progress string.
void setOrientation(int orientation)	It is used to set the orientation, it may be either vertical or horizontal by using SwingConstants.VERTICAL and SwingConstants.HORIZONTAL constants.

void setValue(int value)	It is used to set the current value on the progress bar.

Java JProgressBar Example

```
import javax.swing.*;
public class ProgressBarExample extends JFrame
{
JProgressBar jb;
int i=0,num=0;
ProgressBarExample(){
jb=new JProgressBar(0,2000);
jb.setBounds(40,40,160,30);
jb.setValue(0);
jb.setStringPainted(true);
add(jb);
setSize(250,150);
setLayout(null);
}
public void iterate()
{
while(i<=2000)
{
  jb.setValue(i);
  i=i+20;
  try{Thread.sleep(150);}catch(Exception  e){}
}
}
public static void main(String[] args)
{
    ProgressBarExample m=new ProgressBarExample();
    m.setVisible(true);
    m.iterate();
}
}
```

Output:

Java LayoutManagers

The LayoutManagers are used to arrange components in a particular manner. LayoutManager is an interface that is implemented by all the classes of layout managers. There are following classes that represents the layout managers:

1. java.awt.BorderLayout
2. java.awt.FlowLayout
3. java.awt.GridLayout
4. java.awt.CardLayout
5. java.awt.GridBagLayout
6. javax.swing.BoxLayout
7. javax.swing.GroupLayout
8. javax.swing.ScrollPaneLayout
9. javax.swing.SpringLayout etc.

Java BorderLayout

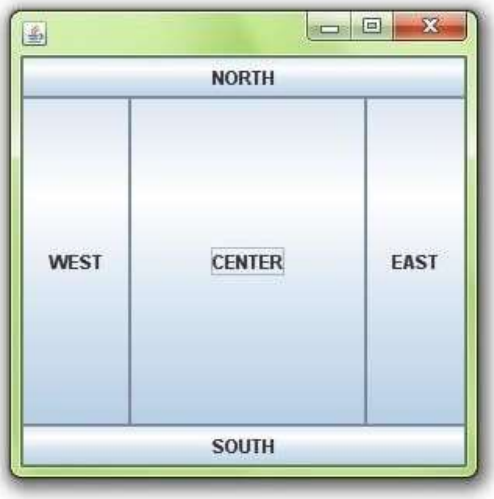
The BorderLayout is used to arrange the components in five regions: north, south, east, west and center. Each region (area) may contain one component only. It is the default layout of frame or window. The BorderLayout provides five constants for each region:

1. **public static final int NORTH**
2. **public static final int SOUTH**
3. **public static final int EAST**
4. **public static final int WEST**
5. **public static final int CENTER**

Constructors of BorderLayout class:

- **BorderLayout():** creates a border layout but with no gaps between the components.
- **JBorderLayout(int hgap, int vgap):** creates a border layout with the given horizontal and vertical gaps between the components.

Example of BorderLayout class:



import java.awt.*;

```
import javax.swing.*;
public class Border
{
JFrame f;
Border()
{
   f=new JFrame();
   JButton b1=new JButton("NORTH");;
   JButton b2=new JButton("SOUTH");;
   JButton b3=new JButton("EAST");;
   JButton b4=new JButton("WEST");;
   JButton b5=new JButton("CENTER");;
   f.add(b1,BorderLayout.NORTH);
   f.add(b2,BorderLayout.SOUTH);
   f.add(b3,BorderLayout.EAST);
   f.add(b4,BorderLayout.WEST);
   f.add(b5,BorderLayout.CENTER);
   f.setSize(300,300);
   f.setVisible(true);
}
public static void main(String[] args)
{
   new Border();
}
}
```

Java GridLayout

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

Constructors of GridLayout class

1. **GridLayout():** creates a grid layout with one column per component in a row.
2. **GridLayout(int rows, int columns):** creates a grid layout with the given rows and columns but no gaps between the components.
3. **GridLayout(int rows, int columns, int hgap, int vgap):** creates a grid layout with the given rows and columns alongwith given horizontal and vertical gaps.

Example of GridLayout class


```
import java.awt.*;
import javax.swing.*;
public class MyGridLayout
{
JFrame f;
MyGridLayout()
{
   f=new JFrame();
   JButton b1=new JButton("1");
   JButton b2=new JButton("2");
   JButton b3=new JButton("3");
   JButton b4=new JButton("4");
   JButton b5=new JButton("5");
      JButton b6=new JButton("6");
      JButton b7=new JButton("7");
   JButton b8=new JButton("8");
      JButton b9=new JButton("9");
   f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);
   f.add(b6);f.add(b7);f.add(b8);f.add(b9);
   f.setLayout(new GridLayout(3,3));
   //setting grid layout of 3 rows and 3 columns
   f.setSize(300,300);
   f.setVisible(true);
}
public static void main(String[] args)
{
   new MyGridLayout();
}
}
```

Java FlowLayout

The FlowLayout is used to arrange the components in a line, one after another (in a flow). It is the default layout of applet or panel.

Fields of FlowLayout class

1. **public static final int LEFT**
2. **public static final int RIGHT**
3. **public static final int CENTER**
4. **public static final int LEADING**
5. **public static final int TRAILING**

Constructors of FlowLayout class

1. **FlowLayout():** creates a flow layout with centered alignment and a default 5 unit horizontal and vertical gap.
2. **FlowLayout(int align):** creates a flow layout with the given alignment and a default 5 unit horizontal and vertical gap.
3. **FlowLayout(int align, int hgap, int vgap):** creates a flow layout with the given alignment and the given horizontal and vertical gap.

Example of FlowLayout class


```
import java.awt.*;
import javax.swing.*;
public class MyFlowLayout
{
JFrame f;
MyFlowLayout()
{
   f=new JFrame();
   JButton b1=new JButton("1");
   JButton b2=new JButton("2");
   JButton b3=new JButton("3");
   JButton b4=new JButton("4");
   JButton b5=new JButton("5");
```

```
    f.add(b1);f.add(b2);f.add(b3);f.add(b4);f.add(b5);
    f.setLayout(new FlowLayout(FlowLayout.RIGHT));
    //setting flow layout of right alignment
    f.setSize(300,300);
    f.setVisible(true);
}
public static void main(String[] args)
{
    new MyFlowLayout();
}
}
```

Java CardLayout

The CardLayout class manages the components in such a manner that only one component is visible at a time. It treats each component as a card that is why it is known as CardLayout.

Constructors of CardLayout class

1. **CardLayout():** creates a card layout with zero horizontal and vertical gap.
2. **CardLayout(int hgap, int vgap):** creates a card layout with the given horizontal and vertical gap.

Commonly used methods of CardLayout class

- **public void next(Container parent):** is used to flip to the next card of the given container.
- **public void previous(Container parent):** is used to flip to the previous card of the given container.
- **public void first(Container parent):** is used to flip to the first card of the given container.
- **public void last(Container parent):** is used to flip to the last card of the given container.
- **public void show(Container parent, String name):** is used to flip to the specified card with the given name.

Example of CardLayout class

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

```
import java.awt.event.*;
import javax.swing.*;
public class CardLayoutExample extends JFrame implements ActionListener
{
CardLayout card;
JButton b1,b2,b3;
Container c;
   CardLayoutExample()
{
      c=getContentPane();
      card=new CardLayout(40,30);
//create CardLayout object with 40 hor space and 30 ver space
      c.setLayout(card);
      b1=new JButton("Apple");
      b2=new JButton("Boy");
      b3=new JButton("Cat");
      b1.addActionListener(this);
      b2.addActionListener(this);
      b3.addActionListener(this);
      c.add("a",b1);c.add("b",b2);c.add("c",b3);
   }
   public void actionPerformed(ActionEvent e)
{
   card.next(c);
   }
   public static void main(String[] args)
{
      CardLayoutExample cl=new CardLayoutExample();
      cl.setSize(400,400);
      cl.setVisible(true);
      cl.setDefaultCloseOperation(EXIT_ON_CLOSE);
   }
}
```

UNIT II

NETWORKING

Java Networking

Java Networking is a concept of connecting two or more computing devices together so that we can share resources.

Java socket programming provides facility to share data between different computing devices.

Advantage of Java Networking

1. sharing resources
2. centralize software management

Java Networking Terminology

The widely used java networking terminologies,

1. IP Address
2. Protocol
3. Port Number
4. MAC Address
5. Connection-oriented and connection-less protocol
6. Socket

IP Address

IP address is a unique number assigned to a node of a network e.g. 192.168.0.1. It is composed of octets that range from 0 to 255. It is a logical address that can be changed.

Protocol

A protocol is a set of rules basically that is followed for communication. For example:

- TCP
- FTP
- Telnet
- SMTP
- POP

Port Number

The port number is used to uniquely identify different applications. It acts as a communication endpoint between applications. The port number is associated with the IP address for communication between two applications.

MAC Address

MAC (Media Access Control) Address is a unique identifier of NIC (Network Interface Controller). A network node can have multiple NIC but each with unique MAC.

Connection-oriented and connection-less protocol

In connection-oriented protocol, acknowledgement is sent by the receiver. So it is reliable but slow. The example of connection-oriented protocol is TCP. But, In connection-less protocol, acknowledgement is not sent by the receiver. So it is not reliable but fast. The example of connection-less protocol is UDP.

Socket

A socket is an endpoint between two way communications.

Java Socket Programming

Java Socket programming is used for communication between the applications running on different JRE. Java Socket programming can be connection-oriented or connection-less. Socket and ServerSocket classes are used for connection-oriented socket programming and DatagramSocket and DatagramPacket classes are used for connection-less socket programming. The client in socket programming must know two information:

1. IP Address of Server, and
2. Port number.

Socket class

A socket is simply an endpoint for communications between the machines. The Socket class can be used to create a socket.

Important methods

Method	Description
1) public InputStream getInputStream()	returns the InputStream attached with this socket.
2) public OutputStream getOutputStream()	returns the OutputStream attached with this socket.
3) public synchronized void close()	closes this socket

ServerSocket class

The ServerSocket class can be used to create a server socket. This object is used to establish communication with the clients.

Important methods

Method	Description
1) public Socket accept()	returns the socket and establish a connection between server and client.
2) public synchronized void close()	closes the server socket.

Example of Java Socket Programming

MyServer.java

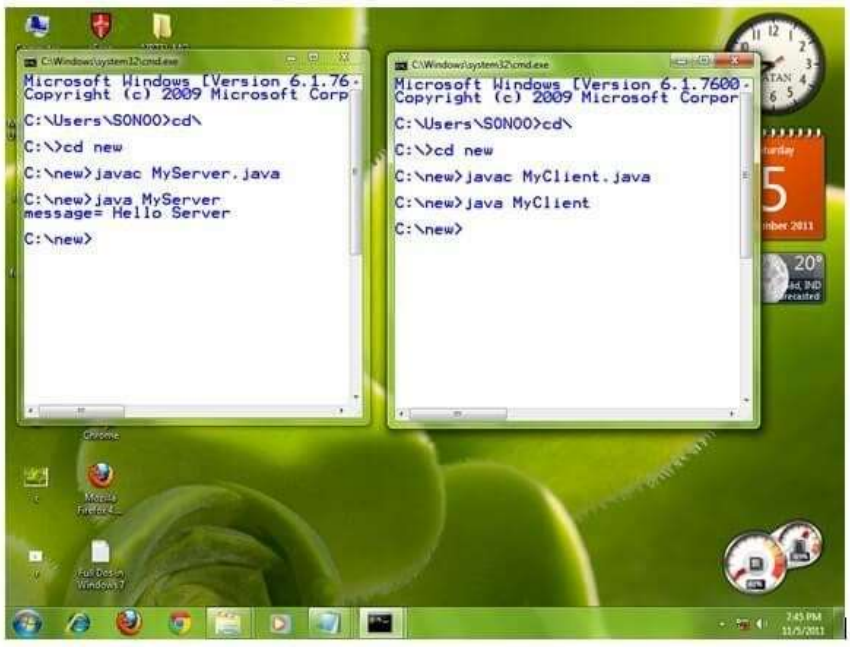
```
import java.io.*;
import java.net.*;
public class MyServer {
public static void main(String[] args){
try{
ServerSocket ss=new ServerSocket(6666);
Socket s=ss.accept();//establishes connection
DataInputStream dis=new DataInputStream(s.getInputStream());
String  str=(String)dis.readUTF();
System.out.println("message= "+str);
```

```
ss.close();
}catch(Exception e){System.out.println(e);}
}
}
```

MyClient.java

```
import java.io.*;
import java.net.*;
public class MyClient {
public static void main(String[] args) {
try{
Socket s=new Socket("localhost",6666);
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
dout.writeUTF("Hello Server");
dout.flush();
dout.close();
s.close();
}catch(Exception e){System.out.println(e);}
}
}
```

To execute this program open two command prompts and execute each program at each command prompt as displayed in the below figure. After running the client application, a message will be displayed on the server console.



Example of Java Socket Programming (Read-Write both side)

In this example, client will write first to the server then server will receive and print the text. Then server will write to the client and client will receive and print the text. The step goes on.

MyServer.java

```
import java.net.*;
import java.io.*;
class MyServer{
public static void main(String args[])throws Exception{
ServerSocket ss=new ServerSocket(3333);
Socket s=ss.accept();
DataInputStream din=new DataInputStream(s.getInputStream());
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
String str="",str2="";
while(!str.equals("stop")){
str=din.readUTF();
System.out.println("client says: "+str);
str2=br.readLine();
dout.writeUTF(str2);
dout.flush();
}
din.close();
s.close();
ss.close();
}}
```

MyClient.java

```
import java.net.*;
import java.io.*;
class MyClient{
public static void main(String args[])throws Exception{
Socket s=new Socket("localhost",3333);
DataInputStream din=new DataInputStream(s.getInputStream());
DataOutputStream dout=new DataOutputStream(s.getOutputStream());
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
String str="",str2="";
while(!str.equals("stop")){
str=br.readLine();
dout.writeUTF(str);
dout.flush();
str2=din.readUTF();
```

```
System.out.println("Server says: "+str2);
}
dout.close();
s.close();
}}
```

Java URL

The Java URL class represents an URL. URL is an acronym for Uniform Resource Locator. It points to a resource on the World Wide Web. For example: http://www.google.com. A URL contains many information

1. Protocol: In this case, http is the protocol.
2. Server name or IP Address: In this case, www.javatpoint.com is the server name.
3. Port Number: It is an optional attribute. If we write http//ww.javatpoint.com:80/sonoojaiswal/ , 80 is the port number. If port number is not mentioned in the URL, it returns -1.
4. File Name or directory name: In this case, index.jsp is the file name.

Methods of Java URL class

The java.net.URL class provides many methods. The important methods of URL class.

Method	Description
public String getProtocol()	it returns the protocol of the URL.
public String getHost()	it returns the host name of the URL.
public String getPort()	it returns the Port Number of the URL.
public String getFile()	it returns the file name of the URL.
public URLConnection openConnection()	it returns the instance of URLConnection i.e. associated with this URL.

Example of Java URL class

```
//URLDemo.java
import   java.io.*;
import java.net.*;
public class URLDemo{
public static void main(String[] args){
try{
URL url=new URL("http://www.javatpoint.com/java-tutorial");
System.out.println("Protocol: "+url.getProtocol());
System.out.println("Host Name: "+url.getHost());
System.out.println("Port Number: "+url.getPort());
System.out.println("File Name: "+url.getFile());
}catch(Exception e){System.out.println(e);}
}
}
```

Output:

Protocol: http
Host Name: www.javatpoint.com
Port Number: -1
File Name: /java-tutorial

Java URLConnection class

The Java URLConnection class represents a communication link between the URL and the application. This class can be used to read and write data to the specified resource referred by the URL.

Get the object of URLConnection class

The openConnection() method of URL class returns the object of URLConnection class.

Syntax:

```
public URLConnection openConnection()throws IOException{}
```

Displaying source code of a webpage by URLConnecton class

The URLConnection class provides many methods, we can display all the data of a webpage by using the getInputStream() method. The getInputStream() method returns all the data of the specified URL in the stream that can be read and displayed.

Example of Java URLConnection class

```
import java.io.*;
import java.net.*;
public class URLConnectionExample {
public static void main(String[] args){
try{
URL url=new URL("http://www.javatpoint.com/java-tutorial");
URLConnection urlcon=url.openConnection();
InputStream stream=urlcon.getInputStream();
int i;
while((i=stream.read())!=-1)
{
System.out.print((char)i);
}
}catch(Exception e){System.out.println(e);}
}
}
```

Java HttpURLConnection class

The Java HttpURLConnection class is http specific URLConnection. It works for HTTP protocol only. By the help of HttpURLConnection class, you can information of any HTTP URL such as header information, status code, response code etc. The java.net.HttpURLConnection is subclass of URLConnection class.

Get the object of HttpURLConnection class

The openConnection() method of URL class returns the object of URLConnection class.

Syntax:

```
public URLConnection openConnection()throws IOException{}
    URL url=new URL("http://www.javatpoint.com/java-tutorial");
    HttpURLConnection huc=(HttpURLConnection)url.openConnection();
```

Java HttpURLConnecton Example

```
import java.io.*;
import java.net.*;
public class HttpURLConnectionDemo{
public static void main(String[] args){
try{
URL url=new URL("http://www.javatpoint.com/java-tutorial");
HttpURLConnection huc=(HttpURLConnection)url.openConnection();
for(int i=1;i<=8;i++){
System.out.println(huc.getHeaderFieldKey(i)+" = "+huc.getHeaderField(i));
}
huc.disconnect();
}catch(Exception e){System.out.println(e);}
}
}
```

Output:

```
Date = Wed, 10 Dec 2014 19:31:14 GMT
Set-Cookie = JSESSIONID=D70B87DBB832820CACA5998C90939D48; Path=/
Content-Type = text/html
Cache-Control = max-age=2592000
Expires = Fri, 09 Jan 2015 19:31:14 GMT
Vary = Accept-Encoding,User-Agent
Connection = close
Transfer-Encoding = chunked
```

Java InetAddress class

Java InetAddress class represents an IP address. The java.net.InetAddress class provides methods to get the IP of any host name for example www.javatpoint.com, www.google.com, www.facebook.com.

Methods of InetAddress class

Method	Description
public static InetAddress getByName(String host) throws UnknownHostException	it returns the instance of InetAddress containing LocalHost IP and name.
public static InetAddress getLocalHost() throws UnknownHostException	it returns the instance of InetAdddress containing local host name and address.

public String getHostName()	it returns the host name of the IP address.
public String getHostAddress()	it returns the IP address in string format.

Example of Java InetAddress class

```
import java.io.*;
import java.net.*;
public class InetDemo{
public static void main(String[] args){
try{
InetAddress ip=InetAddress.getByName("www.javatpoint.com");
System.out.println("Host Name: "+ip.getHostName());
System.out.println("IP Address: "+ip.getHostAddress());
}catch(Exception e){System.out.println(e);}
}
}
```

Output:

```
Host Name: www.javatpoint.com
IP Address: 206.51.231.148
```

Java DatagramSocket and DatagramPacket

Java DatagramSocket and DatagramPacket classes are used for connection-less socket programming.

Java DatagramSocket class

Java DatagramSocket class represents a connection-less socket for sending and receiving datagram packets A datagram is basically an information but there is no guarantee of its content, arrival or arrival time.

Constructors of DatagramSocket class

- DatagramSocket() throws SocketEeption: it creates a datagram socket and binds it with the available Port Number on the localhost machine.
- DatagramSocket(int port) throws SocketEeption: it creates a datagram socket and binds it with the given Port Number.
- DatagramSocket(int port, InetAddress address) throws SocketEeption: it creates a datagram socket and binds it with the specified port number and host address.

Java DatagramPacket class

Java DatagramPacket is a message that can be sent or received. If you send multiple packet, it may arrive in any order. Additionally, packet delivery is not guaranteed.

Constructors of DatagramPacket class

- DatagramPacket(byte[] barr, int length): it creates a datagram packet. This constructor is used to receive the packets.
- DatagramPacket(byte[] barr, int length, InetAddress address, int port): it creates a datagram packet. This constructor is used to send the packets.

Example of Sending DatagramPacket by DatagramSocket

```
//DSender.java
import java.net.*;
public class DSender{
  public static void main(String[] args) throws Exception {
    DatagramSocket ds = new DatagramSocket();
    String str = "Welcome java";
    InetAddress ip = InetAddress.getByName("127.0.0.1");
    DatagramPacket dp = new DatagramPacket(str.getBytes(), str.length(), ip, 3000);
    ds.send(dp);
    ds.close();
  }
}
```

Example of Receiving DatagramPacket by DatagramSocket

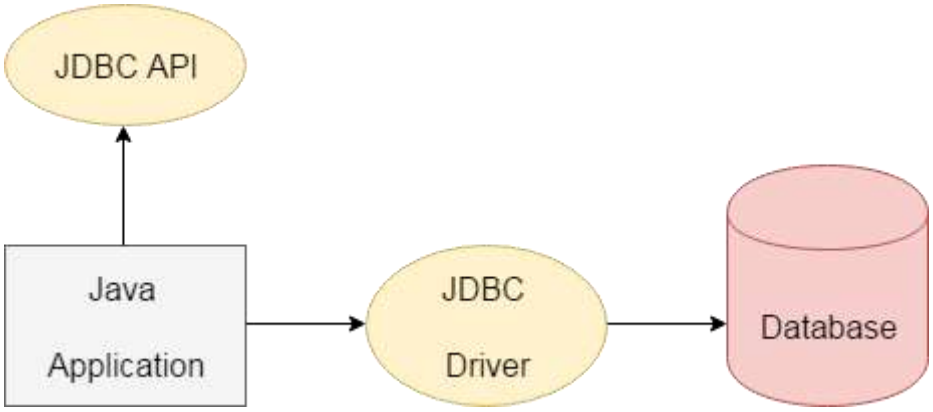
```
//DReceiver.java
import java.net.*;
public class DReceiver{
  public static void main(String[] args) throws Exception {
    DatagramSocket ds = new DatagramSocket(3000);
    byte[] buf = new byte[1024];
    DatagramPacket dp = new DatagramPacket(buf, 1024);
    ds.receive(dp);
    String str = new String(dp.getData(), 0, dp.getLength());
    System.out.println(str);
    ds.close();
  }
}
```

Java Java Database Connectivity

JDBC stands for Java Database Connectivity. JDBC is a Java API to connect and execute the query with the database. It is a part of JavaSE (Java Standard Edition). JDBC API uses JDBC drivers to connect with the database. There are four types of JDBC drivers:

- JDBC-ODBC Bridge Driver,
- Native Driver,
- Network Protocol Driver, and
- Thin Driver

We can use JDBC API to access tabular data stored in any relational database. By the help of JDBC API, we can save, update, delete and fetch data from the database. It is like Open Database Connectivity (ODBC) provided by Microsoft.



The current version of JDBC is 4.3. It is the stable release since 21st September, 2017. It is based on the X/Open SQL Call Level Interface. The java.sql package contains classes and interfaces for JDBC API. A list of popular interfaces of JDBC API.

- Driver interface
- Connection interface
- Statement interface
- PreparedStatement interface
- CallableStatement interface
- ResultSet interface
- ResultSetMetaData interface
- DatabaseMetaData interface
- RowSet interface

A list of popular classes of JDBC API

- DriverManager class
- Blob class
- Clob class
- Types class

Why Use JDBC

Before JDBC, ODBC API was the database API to connect and execute the query with the database. But, ODBC API uses ODBC driver which is written in C language (i.e. platform dependent and unsecured). That is why Java has defined its own API (JDBC API) that uses JDBC drivers (written in Java language).

We can use JDBC API to handle database using Java program and can perform the following activities:

1. Connect to the database
2. Execute queries and update statements to the database
3. Retrieve the result received from the database.

What is API

API (Application programming interface) is a document that contains a description of all the features of a product or software. It represents classes and interfaces that software programs can follow to communicate with each other. An API can be created for applications, libraries, operating systems, etc.

JDBC Driver

1. JDBC Drivers
2. JDBC-ODBC bridge driver
3. Native-API driver
4. Network Protocol driver
5. Thin driver

JDBC Driver is a software component that enables java application to interact with the database. There are 4 types of JDBC drivers:

1. JDBC-ODBC bridge driver
2. Native-API driver (partially java driver)
3. Network Protocol driver (fully java driver)
4. Thin driver (fully java driver)

JDBC-ODBC bridge driver

The JDBC-ODBC bridge driver uses ODBC driver to connect to the database. The JDBC-ODBC bridge driver converts JDBC method calls into the ODBC function calls. This is now discouraged because of thin driver.

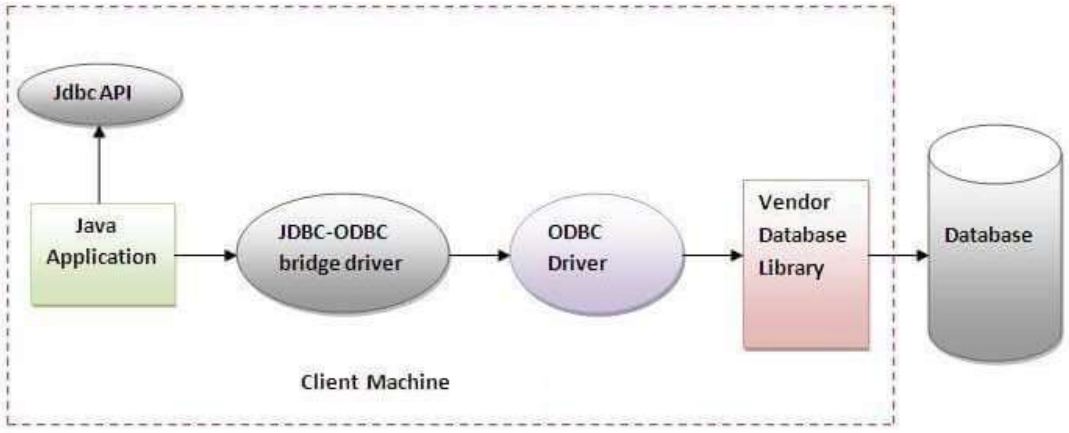


Figure- JDBC-ODBC Bridge Driver

Oracle does not support the JDBC-ODBC Bridge. Oracle recommends that you use JDBC drivers provided by the vendor of your database instead of the JDBC-ODBC Bridge.

Advantages:

- easy to use.
- can be easily connected to any database.

Disadvantages:

- Performance degraded because JDBC method call is converted into the ODBC function calls.
- The ODBC driver needs to be installed on the client machine.

Native-API driver

The Native API driver uses the client-side libraries of the database. The driver converts JDBC method calls into native calls of the database API. It is not written entirely in java.

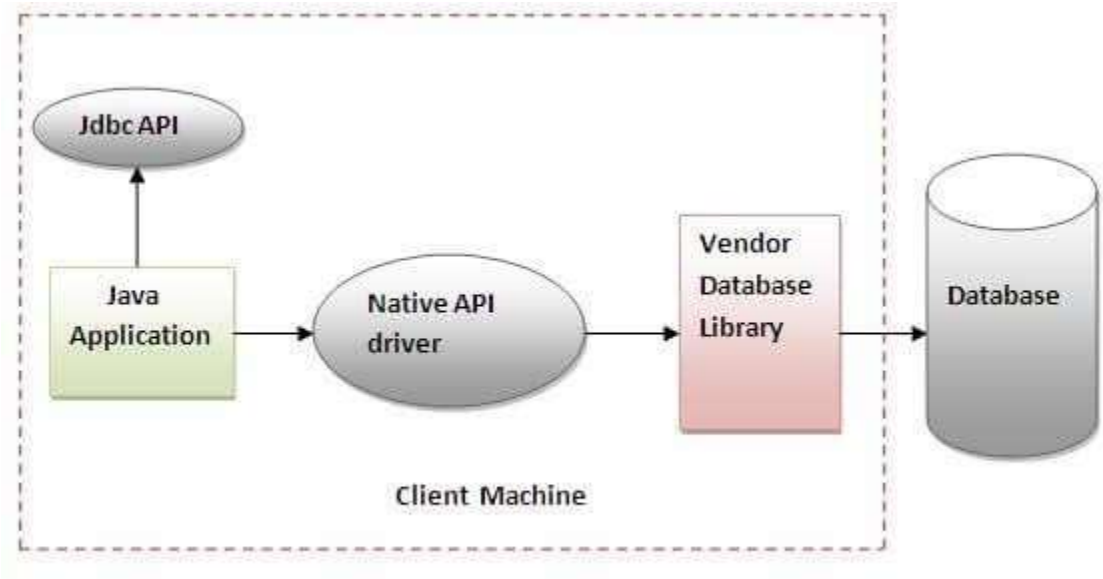


Figure- Native API Driver

Advantage:

- Performance upgraded than JDBC-ODBC bridge driver.

Disadvantage:

- The Native driver needs to be installed on the each client machine.
- The Vendor client library needs to be installed on client machine.

Network Protocol driver

The Network Protocol driver uses middleware (application server) that converts JDBC calls directly or indirectly into the vendor-specific database protocol. It is fully written in java.

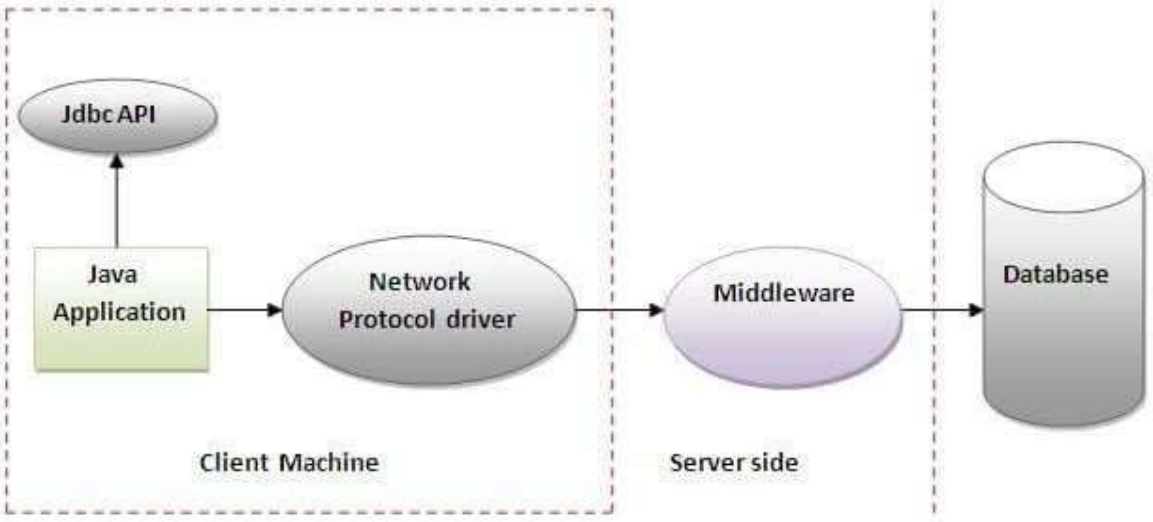


Figure- Network Protocol Driver

Advantage:

- No client side library is required because of application server that can perform many tasks like auditing, load balancing, logging etc.

Disadvantages:

- Network support is required on client machine.
- Requires database-specific coding to be done in the middle tier.
- Maintenance of Network Protocol driver becomes costly because it requires database-specific coding to be done in the middle tier.

Thin driver

The thin driver converts JDBC calls directly into the vendor-specific database protocol. That is why it is known as thin driver. It is fully written in Java language.

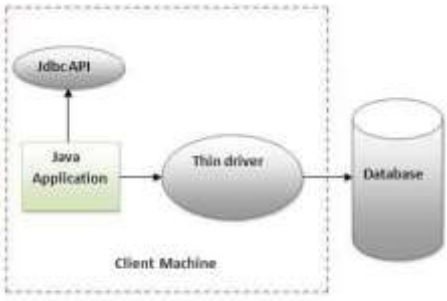


Figure- Thin Driver

Advantage:

- Better performance than all other drivers.
- No software is required at client side or server side.

Disadvantage:

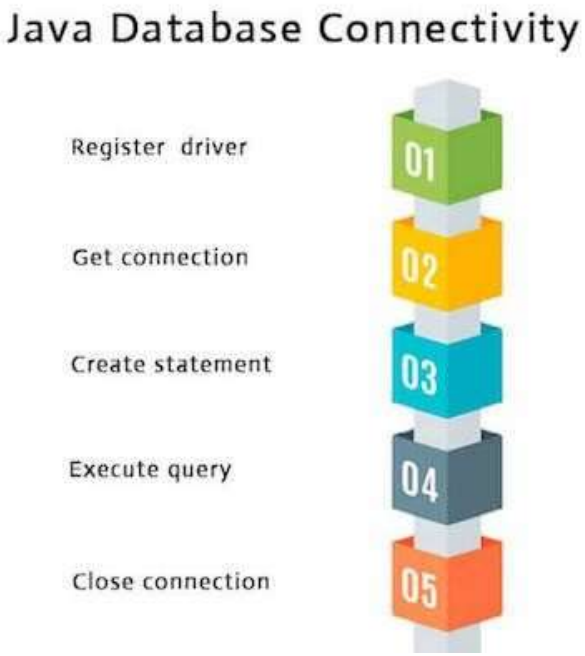
- Drivers depend on the Database.

Java Database Connectivity

1. 5 Steps to connect to the database in java
1. Register the driver class
2. Create the connection object
3. Create the Statement object
4. Execute the query
5. Close the connection object

There are 5 steps to connect any java application with the database using JDBC. These steps are as follows:

- Register the Driver class
- Create connection
- Create statement
- Execute queries
- Close connection



Register the driver class

The forName() method of Class class is used to register the driver class. This method is used to dynamically load the driver class.

Syntax of forName() method

public static void forName(String className)throws ClassNotFoundException

OracleDriver class

Here, Java program is loading oracle driver to establish database connection.

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

Create the connection object

The getConnection() method of DriverManager class is used to establish connection with the database.

Syntax of getConnection() method

1. public static Connection getConnection(String url)throws SQLException
2. public static Connection getConnection(String url,String name,String password)throws SQLException

Connection with the Oracle database

1. Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe", "system","password");

Create the Statement object

The createStatement() method of Connection interface is used to create statement. The object of statement is responsible to execute queries with the database.

Syntax of createStatement() method

```
public Statement createStatement()throws SQLException
```

Example to create the statement object

```
Statement stmt=con.createStatement();
```

Execute the query

The executeQuery() method of Statement interface is used to execute queries to the database. This method returns the object of ResultSet that can be used to get all the records of a table.

Syntax of executeQuery() method

```
public ResultSet executeQuery(String sql)throws SQLException
```

Example to execute query

1. ResultSet rs=stmt.executeQuery("select * from emp");
2. while(rs.next()){
3. System.out.println(rs.getInt(1)+" "+rs.getString(2));
4. }

Close the connection object

By closing connection object statement and ResultSet will be closed automatically. The close() method of Connection interface is used to close the connection.

Syntax of close() method

```
public void close()throws SQLException
```

Example to close connection

```
con.close();  - It avoids explicit connection closing step.
```

Java Database Connectivity with Oracle

To connect java application with the oracle database, we need to follow 5 following steps. In

this example, we are using Oracle 10g as the database. So we need to know following information for the oracle database:

1. **Driver class:** The driver class for the oracle database is **oracle.jdbc.driver.OracleDriver**.
2. **Connection URL:** The connection URL for the oracle10G database is **jdbc:oracle:thin:@localhost:1521:xe** where jdbc is the API, oracle is the database, thin is the driver, localhost is the server name on which oracle is running, we may also use IP address, 1521 is the port number and XE is the Oracle service name. You may get all these information from the tnsnames.ora file.
3. **Username:** The default username for the oracle database is **system**.
4. **Password:** It is the password given by the user at the time of installing the oracle database.

Create a Table

Before establishing connection, let's first create a table in oracle database. Following is the SQL query to create a table.

```
create table emp(id number(10),name varchar2(40),age number(3));
```

Example to Connect Java Application with Oracle database

In this example, we are connecting to an Oracle database and getting data from **emp** table. Here, **system** and **oracle** are the username and password of the Oracle database.

```
import java.sql.*;
class OracleCon{
public static void main(String args[]){
try{
//step1 load the driver class
Class.forName("oracle.jdbc.driver.OracleDriver");
//step2 create  the connection object
Connection con=DriverManager.getConnection(
"jdbc:oracle:thin:@localhost:1521:xe","system","oracle");
//step3 create the statement object
Statement stmt=con.createStatement();
//step4 execute query
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next())
System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
//step5 close the connection object
con.close();
}
```

```
catch(Exception e){ System.out.println(e);}
}
}
```

To connect java application with the Oracle database ojdbc14.jar file is required to be loaded.

Two ways to load the jar file:

1. paste the ojdbc14.jar file in jre/lib/ext folder
2. set classpath

1) paste the ojdbc14.jar file in JRE/lib/ext folder:

Firstly, search the ojdbc14.jar file then go to JRE/lib/ext folder and paste the jar file here.

2) set classpath:

There are two ways to set the classpath:

- Temporary
- Permanent

Temporary classpath:

Firstly, search the ojdbc14.jar file then open command prompt and write:

```
C:>set classpath=c:\folder\ojdbc14.jar;.;
```

Permanent classpath:

Go to environment variable then click on new tab. In variable name write **classpath** and in variable value paste the path to ojdbc14.jar by appending ojdbc14.jar;.; as C:\oraclexe\app\oracle\product\10.2.0\server\jdbc\lib\ojdbc14.jar;.;

Java Database Connectivity with MySQL

To connect Java application with the MySQL database, we need to follow 5 following steps. In this example we are using MySql as the database.

1. **Driver class:** The driver class for the mysql database is **com.mysql.jdbc.Driver**.
2. **Connection URL:** The connection URL for the mysql database is **jdbc:mysql://localhost:3306/sonoo**where jdbc is the API, mysql is the database, localhost is the server name on which mysql is running, we may also use IP address, 3306 is the port number and sonoo is the database name. We may use any database, in such case, we need to replace the sonoo with our database name.
3. **Username:** The default username for the mysql database is **root**.
4. **Password:** It is the password given by the user at the time of installing the mysql database. In this example, we are going to use root as the password.

First create a table in the mysql database, but before creating table, we need to create database first.

1. create database sonoo;
2. use sonoo;
3. create table emp(id **int**(10),name varchar(40),age **int**(3));

Example to Connect Java Application with mysql database

In this example, sonoo is the database name, root is the username and password both.

```
import java.sql.*;
class MysqlCon{
public static void main(String args[]){
try{
Class.forName("com.mysql.jdbc.Driver");
Connection con=DriverManager.getConnection(
"jdbc:mysql://localhost:3306/sonoo","root","root");
//here sonoo is database name, root is username and password
Statement stmt=con.createStatement();
ResultSet rs=stmt.executeQuery("select * from emp");
while(rs.next())
System.out.println(rs.getInt(1)+"  "+rs.getString(2)+"  "+rs.getString(3));
con.close();
}catch(Exception e){ System.out.println(e);}
}
}
```

To connect java application with the mysql database, **mysqlconnector.jar** file is required to be loaded.

Two ways to load the jar file:

1. Paste the mysqlconnector.jar file in jre/lib/ext folder
2. Set classpath

1) Paste the mysqlconnector.jar file in JRE/lib/ext folder:

Download the mysqlconnector.jar file. Go to jre/lib/ext folder and paste the jar file here.

2) Set classpath:

There are two ways to set the classpath:

- Temporary
- Permanent

Temporary classpath

open command prompt and write:

C:>set classpath=c:\folder\mysql-connector-java-5.0.8-bin.jar;.;

Permanent classpath

Go to environment variable then click on new tab. In variable name write **classpath** and in variable value paste the path to the mysqlconnector.jar file by appending mysqlconnector.jar;.; as C:\folder\mysql-connector-java-5.0.8-bin.jar;.;

Connectivity with Access without DSN

There are two ways to connect java application with the access database.

1. Without DSN (Data Source Name)
2. With DSN

Java is mostly used with Oracle, mysql, or DB2 database. So you can learn this topic only for knowledge.

Example to Connect Java Application with access without DSN

In this example, we are going to connect the java program with the access database. In such case, we have created the login table in the access database. There is only one column in the table named name.

```
import java.sql.*;
class Test{
public static void main(String ar[]){
 try{
   String database="student.mdb";//Here database exists in the current directory
   String url="jdbc:odbc:Driver={Microsoft Access Driver (*.mdb)};
              DBQ=" + database + ";DriverID=22;READONLY=true";
  Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
   Connection c=DriverManager.getConnection(url);
   Statement st=c.createStatement();
   ResultSet rs=st.executeQuery("select * from login");
    while(rs.next()){
    System.out.println(rs.getString(1));
   }
  }catch(Exception ee){System.out.println(ee);}
  }}
```

Example to Connect Java Application with access with DSN

Connectivity with type1 driver is not considered good. To connect java application with type1 driver, create DSN first, here we are assuming your dsn name is mydsn.

```
import java.sql.*;
class Test{
public static void main(String ar[]){
 try{
   String url="jdbc:odbc:mydsn";
   Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
   Connection c=DriverManager.getConnection(url);
   Statement st=c.createStatement();
   ResultSet rs=st.executeQuery("select * from login");
   while(rs.next()){
    System.out.println(rs.getString(1));
   }
}catch(Exception ee){System.out.println(ee);}
}}
```

DriverManager class

The DriverManager class acts as an interface between user and drivers. It keeps track of the drivers that are available and handles establishing a connection between a database and the appropriate driver. The DriverManager class maintains a list of Driver classes that have registered themselves by calling the method DriverManager.registerDriver().

Methods of DriverManager class

Method	Description
1) public static void registerDriver(Driver driver):	is used to register the given driver with DriverManager.
2) public static void deregisterDriver(Driver driver):	is used to deregister the given driver (drop the driver from the list) with DriverManager.
3) public static Connection getConnection(String url):	is used to establish the connection with the specified url.
4) public static Connection getConnection(String url,String userName,String password):	is used to establish the connection with the specified url, username and password.

Connection interface

A Connection is the session between java application and database. The Connection interface is a factory of Statement, PreparedStatement, and DatabaseMetaData i.e. object of Connection can be used to get the object of Statement and DatabaseMetaData. The Connection interface provide many methods for transaction management like commit(), rollback() etc.

Methods of Connection interface:

1) public Statement createStatement(): creates a statement object that can be used to execute SQL queries.

2) public Statement createStatement(int resultSetType,int resultSetConcurrency): Creates a Statement object that will generate ResultSet objects with the given type and concurrency.

3) public void setAutoCommit(boolean status): is used to set the commit status.By default it is true.

4) public void commit(): saves the changes made since the previous commit/rollback permanent.

5) public void rollback(): Drops all changes made since the previous commit/rollback.

6) public void close(): closes the connection and Releases a JDBC resources immediately.

Statement interface

The **Statement interface** provides methods to execute queries with the database. The statement interface is a factory of ResultSet i.e. it provides factory method to get the object of ResultSet.

Methods of Statement interface:

The important methods of Statement interface,

1) public ResultSet executeQuery(String sql): is used to execute SELECT query. It returns the object of ResultSet.

2) public int executeUpdate(String sql): is used to execute specified query, it may be create, drop, insert, update, delete etc.

3) public boolean execute(String sql): is used to execute queries that may return multiple results.

4) public int[] executeBatch(): is used to execute batch of commands.

Example of Statement interface

Statement interface to insert, update and delete the record.

```
import java.sql.*;
class FetchRecord{
public static void main(String args[])throws Exception{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system",
"oracle");
Statement stmt=con.createStatement();
//stmt.executeUpdate("insert into emp765 values(33,'Irfan',50000)");
//int result=stmt.executeUpdate("update emp765 set name='Vimal',salary=10000 where id=33");
int result=stmt.executeUpdate("delete from emp765 where id=33");
System.out.println(result+" records affected");
con.close();
}}
```

ResultSet interface

The object of ResultSet maintains a cursor pointing to a row of a table. Initially, cursor points to before the first row.

But we can make this object to move forward and backward direction by passing either TYPE_SCROLL_INSENSITIVE or TYPE_SCROLL_SENSITIVE in createStatement(int,int) method.

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

Methods of ResultSet interface

1) public boolean next():	is used to move the cursor to the one row next from the current position.
2) public boolean previous():	is used to move the cursor to the one row previous from the current position.
3) public boolean first():	is used to move the cursor to the first row in result set object.
4) public boolean last():	is used to move the cursor to the last row in

	result set object.
5) public boolean absolute(int row):	is used to move the cursor to the specified row number in the ResultSet object.
6) public boolean relative(int row):	is used to move the cursor to the relative row number in the ResultSet object, it may be positive or negative.
7) public int getInt(int columnIndex):	is used to return the data of specified column index of the current row as int.
8) public int getInt(String columnName):	is used to return the data of specified column name of the current row as int.
9) public String getString(int columnIndex):	is used to return the data of specified column index of the current row as String.
10) public String getString(String columnName):	is used to return the data of specified column name of the current row as String.

Example of Scrollable ResultSet

```
import java.sql.*;
class FetchRecord{
public static void main(String args[])throws Exception{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system",
"oracle");
Statement stmt=con.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE,ResultSet.CONC
UR_UPDATABLE);
ResultSet rs=stmt.executeQuery("select * from emp765");
//getting the record of 3rd row
rs.absolute(3);
System.out.println(rs.getString(1)+" "+rs.getString(2)+" "+rs.getString(3));
con.close();
}}
```

PreparedStatement interface

The PreparedStatement interface is a subinterface of Statement. It is used to execute parameterized query.

String sql="insert into emp values(?,?,?)";

We are passing parameter (?) for the values. Its value will be set by calling the setter methods of PreparedStatement.

Why PreparedStatement?

Improves performance: The performance of the application will be faster if you use PreparedStatement interface because query is compiled only once.

Get the instance of PreparedStatement

The prepareStatement() method of Connection interface is used to return the object of PreparedStatement.

Syntax:

public PreparedStatement prepareStatement(String query)throws SQLException{}

Methods of PreparedStatement interface

Method	Description
public void setInt(int paramIndex, int value)	sets the integer value to the given parameter index.
public void setString(int paramIndex, String value)	sets the String value to the given parameter index.
public void setFloat(int paramIndex, float value)	sets the float value to the given parameter index.
public void setDouble(int paramIndex, double value)	sets the double value to the given parameter index.
public int executeUpdate()	executes the query. It is used for create, drop, insert, update, delete etc.
public ResultSet executeQuery()	executes the select query. It returns an instance of ResultSet.

Example of PreparedStatement interface that inserts the record

First of all create table,

create table emp(id number(10),name varchar2(50));

Now insert records in this table,

```
import java.sql.*;
class InsertPrepared{
public static void main(String args[]){
try{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system",
"oracle");
PreparedStatement stmt=con.prepareStatement("insert into Emp values(?,?)");
stmt.setInt(1,101);//1 specifies the first parameter in the query
stmt.setString(2,"Ratan");
int i=stmt.executeUpdate();
System.out.println(i+" records inserted");
con.close();
}catch(Exception e){ System.out.println(e);}
}
}
```

Example of PreparedStatement interface that updates the record

```
PreparedStatement stmt=con.prepareStatement("update emp set name=? where id=?");
stmt.setString(1,"Sonoo");//1 specifies the first parameter in the query i.e. name
stmt.setInt(2,101);
int i=stmt.executeUpdate();
System.out.println(i+" records updated");
```

Example of PreparedStatement interface that deletes the record

```
PreparedStatement stmt=con.prepareStatement("delete from emp where id=?");
stmt.setInt(1,101);
int i=stmt.executeUpdate();
System.out.println(i+" records deleted");
```

Example of PreparedStatement interface that retrieve the records of a table

```
PreparedStatement stmt=con.prepareStatement("select * from emp");
ResultSet rs=stmt.executeQuery();
while(rs.next()){
System.out.println(rs.getInt(1)+" "+rs.getString(2));
}
```

Example of PreparedStatement to insert records until user press n

```
import java.sql.*;
import java.io.*;
class RS{
public static void main(String args[])throws Exception{
Class.forName("oracle.jdbc.driver.OracleDriver");
Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:xe","system",
"oracle");
PreparedStatement ps=con.prepareStatement("insert into emp130 values(?,?,?)");
BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
do{
System.out.println("enter id:");
int id=Integer.parseInt(br.readLine());
System.out.println("enter name:");
String name=br.readLine();
System.out.println("enter salary:");
float salary=Float.parseFloat(br.readLine());
ps.setInt(1,id);
ps.setString(2,name);
ps.setFloat(3,salary);
int i=ps.executeUpdate();
System.out.println(i+" records affected");
System.out.println("Do you want to continue: y/n");
String s=br.readLine();
```

```
if(s.startsWith("n")){
break;
}
}while(true);
con.close();
}}
```

UNIT III
Servlets

Servlet Introduction

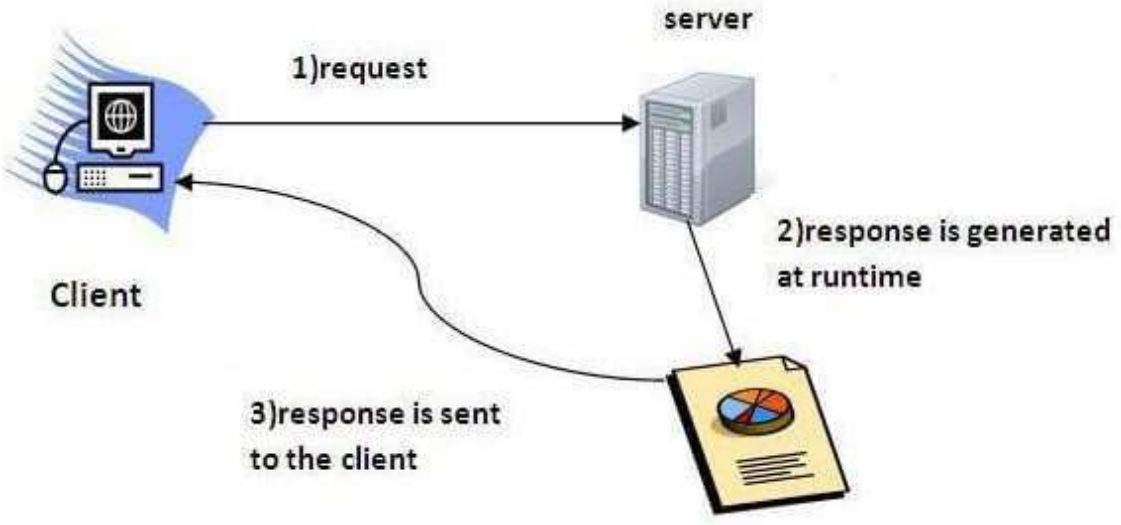
Servlet technology is used to create a web application (resides at server side and generates a dynamic web page). Servlet technology is robust and scalable because of java language. Before Servlet, CGI (Common Gateway Interface) scripting language was common as a server-side programming language. However, there were many disadvantages to this technology.

There are many interfaces and classes in the Servlet API such as Servlet, GenericServlet, HttpServlet, ServletRequest, ServletResponse, etc.

Servlet

Servlet can be described in many ways, depending on the context.

- Servlet is a technology which is used to create a web application.
- Servlet is an API that provides many interfaces and classes including documentation.
- Servlet is an interface that must be implemented for creating any Servlet.
- Servlet is a class that extends the capabilities of the servers and responds to the incoming requests. It can respond to any requests.
- Servlet is a web component that is deployed on the server to create a dynamic web page.

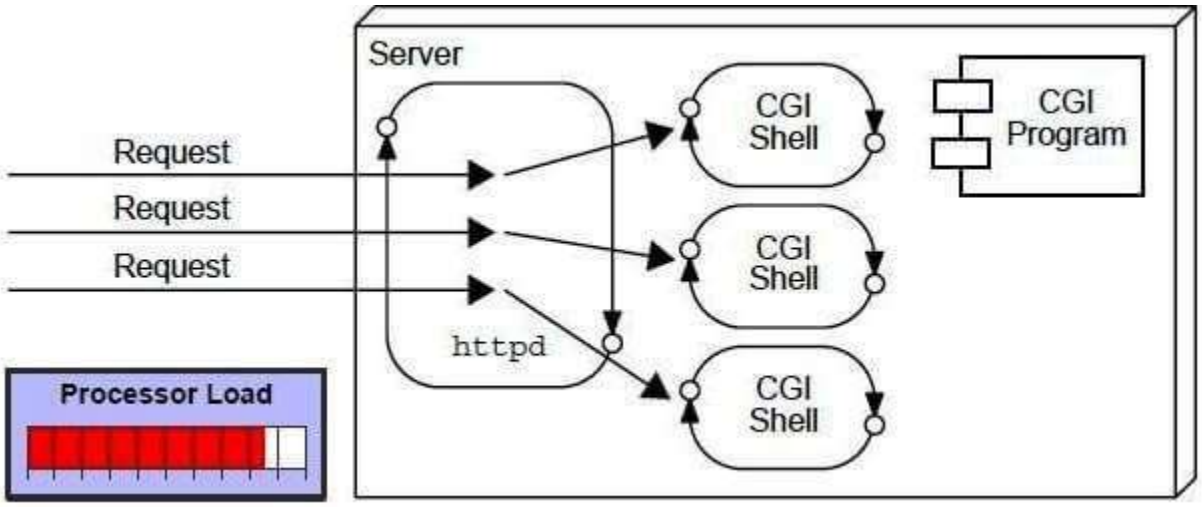


Web application

A web application is an application accessible from the web. A web application is composed of web components like Servlet, JSP, Filter, etc. and other elements such as HTML, CSS, and JavaScript. The web components typically execute in Web Server and respond to the HTTP request.

CGI (Common Gateway Interface)

CGI technology enables the web server to call an external program and pass HTTP request information to the external program to process the request. For each request, it starts a new process.

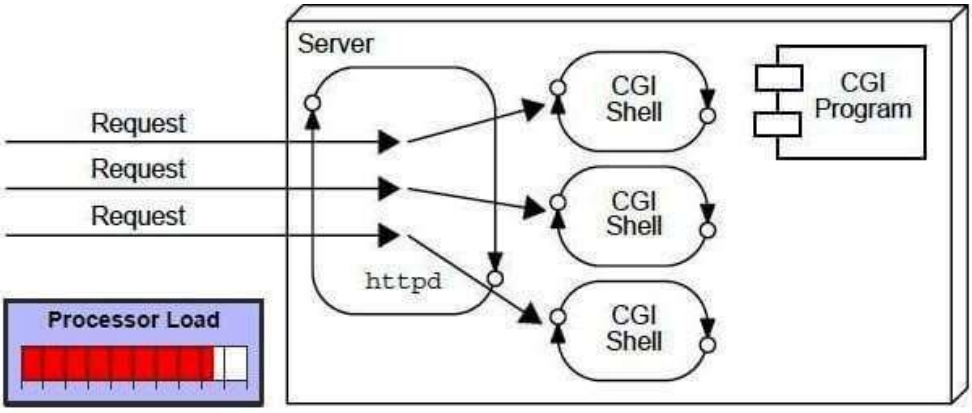


Disadvantages of CGI

There are many problems in CGI technology:

1. If the number of clients increases, it takes more time for sending the response.
2. For each request, it starts a process, and the web server is limited to start processes.
3. It uses platform dependent language e.g. C, C++, perl.

Advantages of Servlet



There are many advantages of Servlet over CGI. The web container creates threads for handling the multiple requests to the Servlet. Threads have many benefits over the Processes

such as they share a common memory area, lightweight, cost of communication between the threads are low. The advantages of Servlet are as follows:

1. Better performance: because it creates a thread for each request, not process.
2. Portability: because it uses Java language.
3. Robust: JVM manages Servlets, so we don't need to worry about the memory leak, garbage collection, etc.
4. Secure: because it uses java language.

Web Terminology

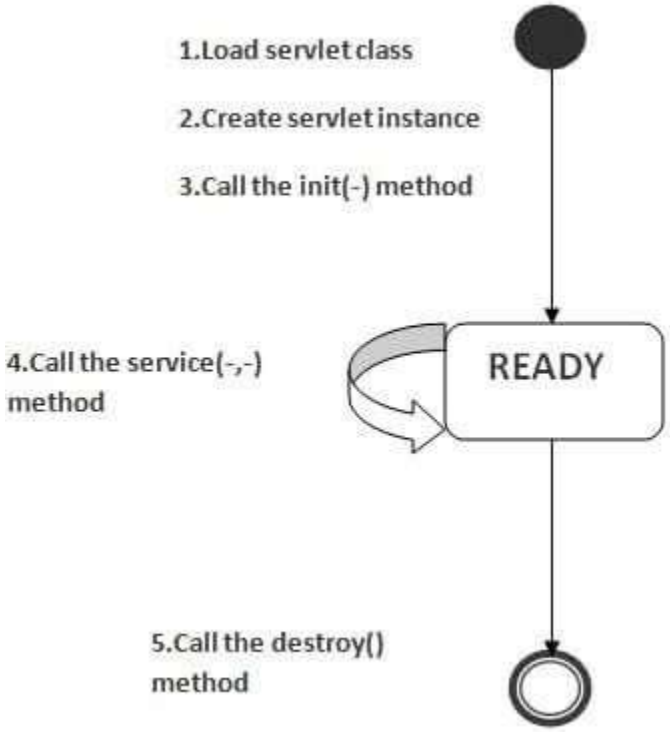
Servlet Terminology	Description
Website: static vs dynamic	It is a collection of related web pages that may contain text, images, audio and video.
HTTP	It is the data communication protocol used to establish communication between client and server.
HTTP Requests	It is the request send by the computer to a web server that contains all sorts of potentially interesting information.
Get vs Post	It gives the difference between GET and POST request.
Container	It is used in java for dynamically generating the web pages on the server side.
Server: Web vs Application	It is used to manage the network resources and for running the program or software that provides services.
Content Type	It is HTTP header that provides the description about what are you sending to the browser.

Life Cycle of a Servlet (Servlet Life Cycle)

1. Life Cycle of a Servlet
1. Servlet class is loaded
2. Servlet instance is created
3. init method is invoked
4. service method is invoked
5. destroy method is invoked

The web container maintains the life cycle of a servlet instance. Servlet class is loaded.

1. Servlet instance is created.
2. init method is invoked.
3. service method is invoked.
4. destroy method is invoked.



There are three states of a servlet: new, ready and end. The servlet is in new state if servlet instance is created. After invoking the init() method, Servlet comes in the ready state. In the ready state, servlet performs all the tasks. When the web container invokes the destroy() method, it shifts to the end state.

Servlet class is loaded

The classloader is responsible to load the servlet class. The servlet class is loaded when the first request for the servlet is received by the web container.

Servlet instance is created

The web container creates the instance of a servlet after loading the servlet class. The servlet instance is created only once in the servlet life cycle.

init method is invoked

The web container calls the init method only once after creating the servlet instance. The init method is used to initialize the servlet. It is the life cycle method of the javax.servlet.Servlet interface.

Syntax of the init method

```
public void init(ServletConfig config) throws ServletException
```

service method is invoked

The web container calls the service method each time when request for the servlet is received. If servlet is not initialized, it follows the first three steps as described above then calls the service method. If servlet is initialized, it calls the service method. That servlet is initialized only once.

Syntax of the service method

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

destroy method is invoked

The web container calls the destroy method before removing the servlet instance from the service. It gives the servlet an opportunity to clean up any resource for example memory, thread etc.

Syntax of the destroy method

public void destroy()

Servlet Interface

1. Servlet Interface
2. Methods of Servlet interface

Servlet interface provides common behavior to all the servlets. Servlet interface defines methods that all servlets must implement. Servlet interface needs to be implemented for creating any servlet (either directly or indirectly). It provides 3 life cycle methods that are used to initialize the servlet, to service the requests, and to destroy the servlet and 2 non-life cycle methods.

Methods of Servlet interface

There are 5 methods in Servlet interface. The init, service and destroy are the life cycle methods of servlet. These are invoked by the web container.

Method	Description
public void init(ServletConfig config)	initializes the servlet. It is the life cycle method of servlet and invoked by the web container only once.
public void service(ServletRequest request,ServletResponse response)	provides response for the incoming request. It is invoked at each request by the web container.
public void destroy()	is invoked only once and indicates that servlet is being destroyed.
public ServletConfig getServletConfig()	returns the object of ServletConfig.
public String getServletInfo()	returns information about servlet such as writer, copyright, version etc.

Servlet API

1. Servlet API
2. Interfaces in javax.servlet package
3. Classes in javax.servlet package
4. Interfaces in javax.servlet.http package
5. Classes in javax.servlet.http package

The javax.servlet and javax.servlet.http packages represent interfaces and classes for servlet api.

The javax.servlet package contains many interfaces and classes that are used by the servlet or web container. These are not specific to any protocol.

The javax.servlet.http package contains interfaces and classes that are responsible for http requests only.

GenericServlet class

1. GenericServlet class
2. Methods of GenericServlet class
3. Example of GenericServlet class

GenericServlet class implements Servlet, ServletConfig and Serializableinterfaces. It provides the implementation of all the methods of these interfaces except the service method. GenericServlet class can handle any type of request so it is protocol-independent.

We may create a generic servlet by inheriting the GenericServlet class and providing the implementation of the service method.

Methods of GenericServlet class

There are many methods in GenericServlet class.

1. public void init(ServletConfig config) is used to initialize the servlet.
2. public abstract void service(ServletRequest request, ServletResponse response) provides service for the incoming request. It is invoked at each time when user requests for a servlet.
3. public void destroy() is invoked only once throughout the life cycle and indicates that servlet is being destroyed.
4. public ServletConfig getServletConfig() returns the object of ServletConfig.
5. public String getServletInfo() returns information about servlet such as writer, copyright, version etc.
6. public void init() it is a convenient method for the servlet programmers, now there is no need to call super.init(config)
7. public ServletContext getServletContext() returns the object of ServletContext.
8. public String getInitParameter(String name) returns the parameter value for the given parameter name.
9. public Enumeration getInitParameterNames() returns all the parameters defined in the web.xml file.
10. public String getServletName() returns the name of the servlet object.

11. public void log(String msg) writes the given message in the servlet log file.
12. public void log(String msg,Throwable t) writes the explanatory message in the servlet log file and a stack trace.

Example of GenericServlet class

First.java

```
import java.io.*;
import javax.servlet.*;
public class First extends GenericServlet{
public void service(ServletRequest req,ServletResponse res)
throws IOException,ServletException
{
res.setContentType("text/html");
PrintWriter out=res.getWriter();
out.print("<html><body>");
out.print("<b>hello generic servlet</b>");
out.print("</body></html>");
}
}
```

HttpServlet class

1. HttpServlet class
2. Methods of HttpServlet class

The HttpServlet class extends the GenericServlet class and implements Serializable interface. It provides http specific methods such as doGet, doPost, doHead, doTrace etc.

Methods of HttpServlet class

There are many methods in HttpServlet class.

1. public void service(ServletRequest req,ServletResponse res) dispatches the request to the protected service method by converting the request and response object into http type.
2. protected void service(HttpServletRequest req, HttpServletResponse res) receives the request from the service method, and dispatches the request to the doXXX() method depending on the incoming http request type.
3. protected void doGet(HttpServletRequest req, HttpServletResponse res) handles the GET request. It is invoked by the web container.
4. protected void doPost(HttpServletRequest req, HttpServletResponse res) handles the POST request. It is invoked by the web container.
5. protected void doHead(HttpServletRequest req, HttpServletResponse res) handles the HEAD request. It is invoked by the web container.
6. protected void doOptions(HttpServletRequest req, HttpServletResponse res) handles the OPTIONS request. It is invoked by the web container.
7. protected void doPut(HttpServletRequest req, HttpServletResponse res) handles the PUT request. It is invoked by the web container.

8. protected void doTrace(HttpServletRequest req, HttpServletResponse res) handles the TRACE request. It is invoked by the web container.
9. protected void doDelete(HttpServletRequest req, HttpServletResponse res) handles the DELETE request. It is invoked by the web container.
10. protected long getLastModified(HttpServletRequest req) returns the time when HttpServletRequest was last modified since midnight January 1, 1970 GMT.

Steps to create a servlet example

1. Steps to create the servlet using Tomcat server
1. Create a directory structure
2. Create a Servlet
3. Compile the Servlet
4. Create a deployment descriptor
5. Start the server and deploy the application

There are given 6 steps to create a servlet example. These steps are required for all the servers.
The servlet example can be created by three ways:

1. By implementing Servlet interface,
2. By inheriting GenericServlet class, (or)
3. By inheriting HttpServlet class

The mostly used approach is by extending HttpServlet because it provides http request specific method such as doGet(), doPost(), doHead() etc.

We are going to use apache tomcat server in this example. The steps are as follows:

1. Create a directory structure
2. Create a Servlet
3. Compile the Servlet
4. Create a deployment descriptor
5. Start the server and deploy the project
6. Access the servlet

Create a directory structures

The directory structure defines that where to put the different types of files so that web container may get the information and respond to the client.

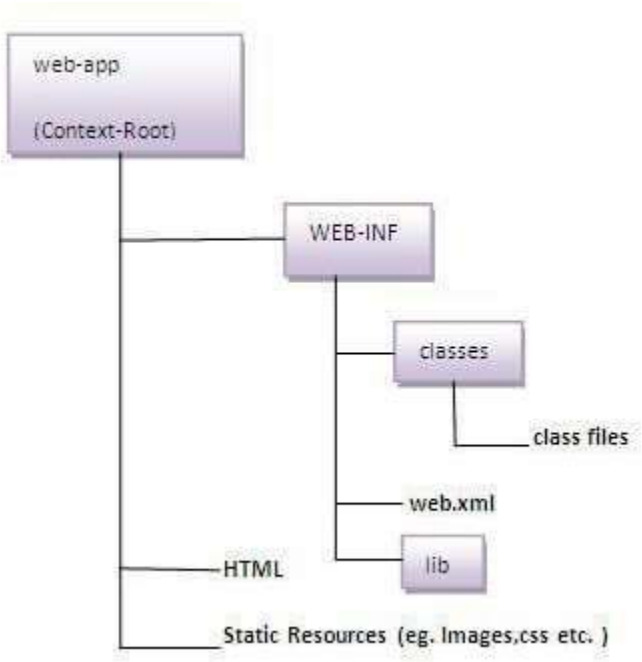
As you can see that the servlet class file must be in the classes folder. The web.xml file must be under the WEB-INF folder.

Create a Servlet

There are three ways to create the servlet.

1. By implementing the Servlet interface
2. By inheriting the GenericServlet class
3. By inheriting the HttpServlet class

The HttpServlet class is widely used to create the servlet because it provides methods to handle http requests such as doGet(), doPost, doHead() etc.



We are going to create a servlet that extends the HttpServlet class. In this example, we are inheriting the HttpServlet class and providing the implementation of the doGet() method. That get request is the default request.

DemoServlet.java

```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;
public class DemoServlet extends HttpServlet{
public void doGet(HttpServletRequest req,HttpServletResponse res)
throws ServletException,IOException
{
res.setContentType("text/html");//setting the content type
PrintWriter pw=res.getWriter();//get the stream to write the data
//writing html in the stream
pw.println("<html><body>");
pw.println("Welcome to servlet");
pw.println("</body></html>");
pw.close();//closing the stream
}}
```

Compile the servlet

For compiling the Servlet, jar file is required to be loaded. Different Servers provide different jar files:

Jar file	Server
1) servlet-api.jar	Apache Tomcat
2) weblogic.jar	Weblogic
3) javaee.jar	Glassfish
4) javaee.jar	JBoss

Two ways to load the jar file

1. set classpath
2. paste the jar file in JRE/lib/ext folder

Put the java file in any folder. After compiling the java file, paste the class file of servlet in WEB-INF/classes directory.

Create the deployment descriptor (web.xml file)

The deployment descriptor is an xml file, from which Web Container gets the information about the servet to be invoked.

The web container uses the Parser to get the information from the web.xml file. There are many xml parsers such as SAX, DOM and Pull.

There are many elements in the web.xml file. Here is given some necessary elements to run the simple servlet program.

web.xml file

```
<web-app>
<servlet>
<servlet-name>sonoojaiswal</servlet-name>
<servlet-class>DemoServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>sonoojaiswal</servlet-name>
<url-pattern>/welcome</url-pattern>
</servlet-mapping>
</web-app>
```

Description of the elements of web.xml file

There are too many elements in the web.xml file. Here is the illustration of some elements that is used in the above web.xml file.

<web-app> represents the whole application.

<servlet> is sub element of <web-app> and represents the servlet.

<servlet-name> is sub element of <servlet> represents the name of the servlet.

<servlet-class> is sub element of <servlet> represents the class of the servlet.

<servlet-mapping> is sub element of <web-app>. It is used to map the servlet.

<url-pattern> is sub element of <servlet-mapping>. This pattern is used at client side to invoke the servlet.

Start the Server and deploy the project

To start Apache Tomcat server, double click on the startup.bat file under apache-tomcat/bin directory.

Configuration for Apache Tomcat Server

Perform 2 tasks:

1. set JAVA_HOME or JRE_HOME in environment variable (It is required to start server).
2. Change the port number of tomcat (optional). It is required if another server is running on same port (8080).

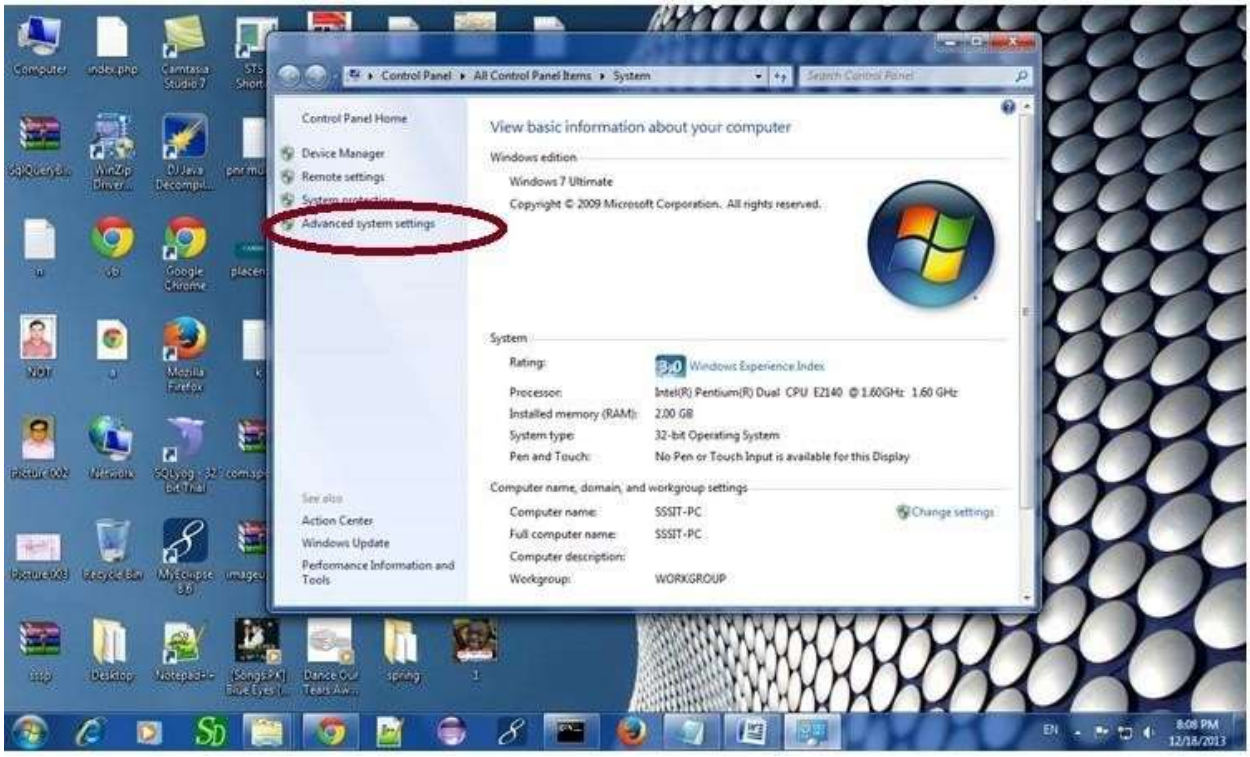
Set JAVA_HOME in environment variable

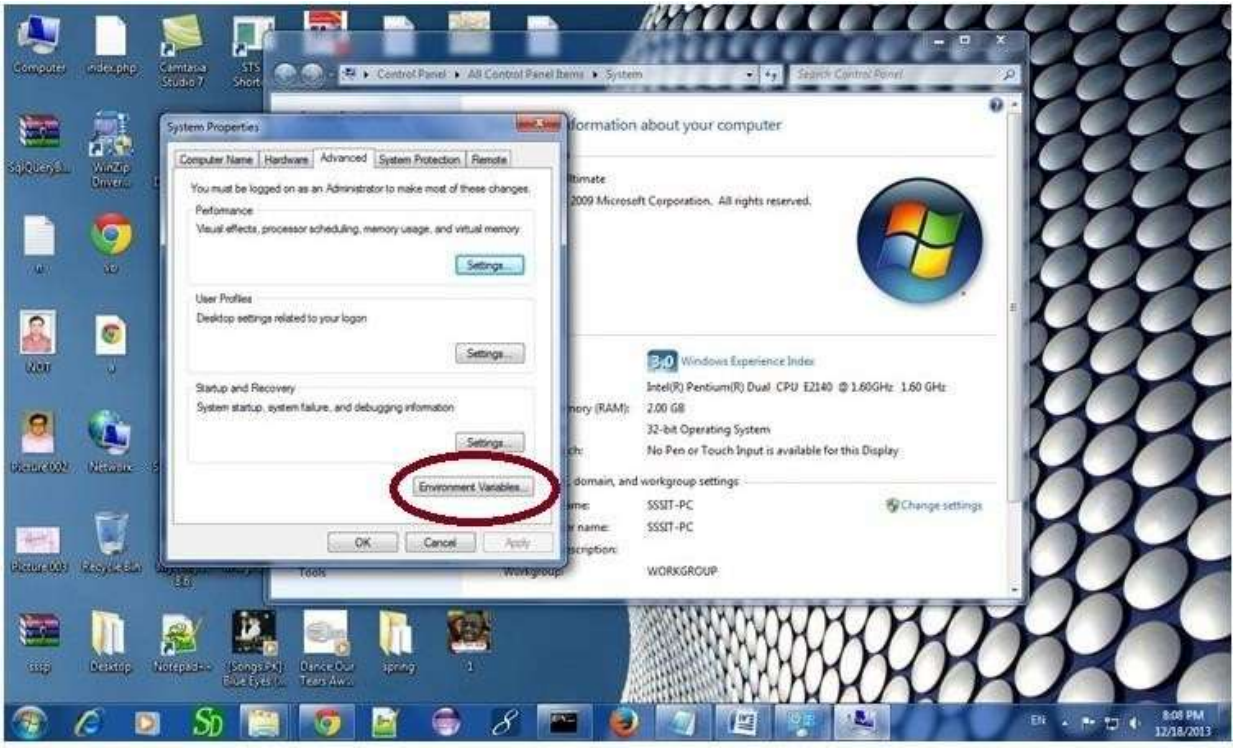
To start Apache Tomcat server JAVA_HOME and JRE_HOME must be set in Environment variables.

Go to My Computer properties -> Click on advanced tab then environment variables -> Click on the new tab of user variable -> Write JAVA_HOME in variable name and paste the path of jdk folder in variable value -> ok -> ok -> ok.

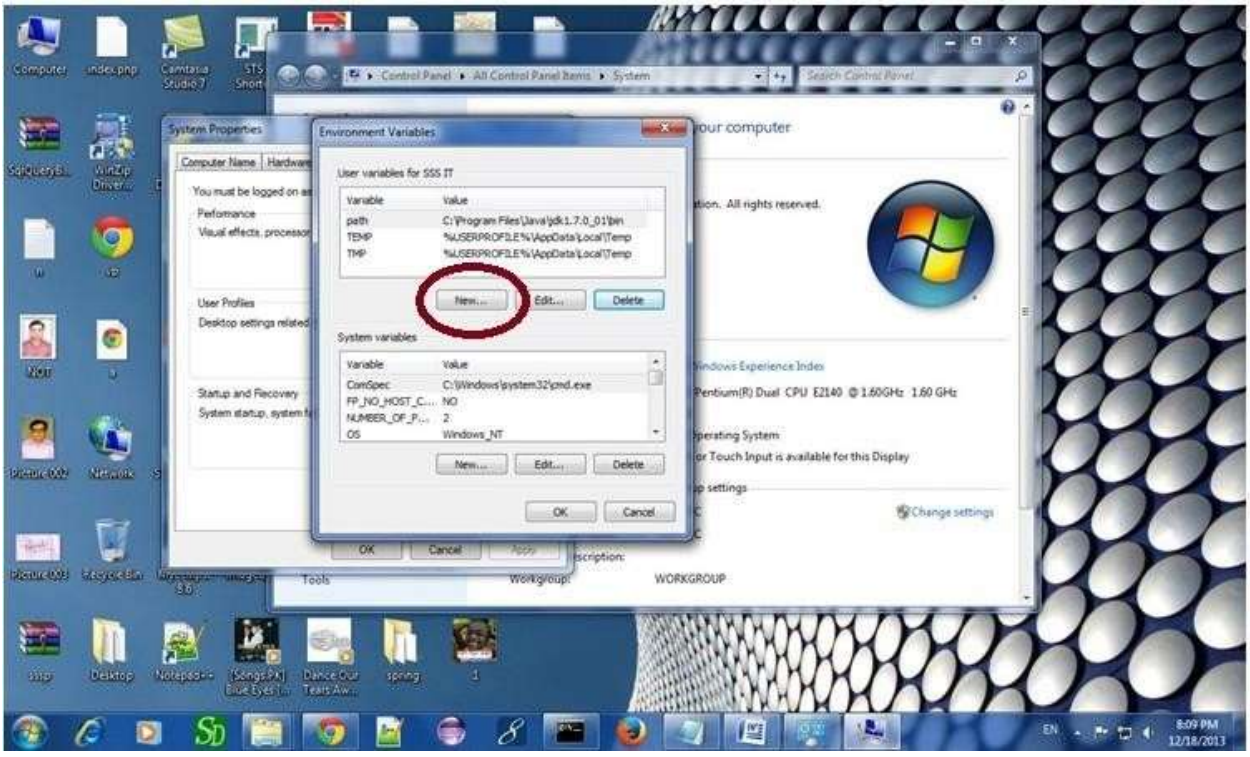
Go to My Computer properties

Click on advanced system settings tab then environment variables

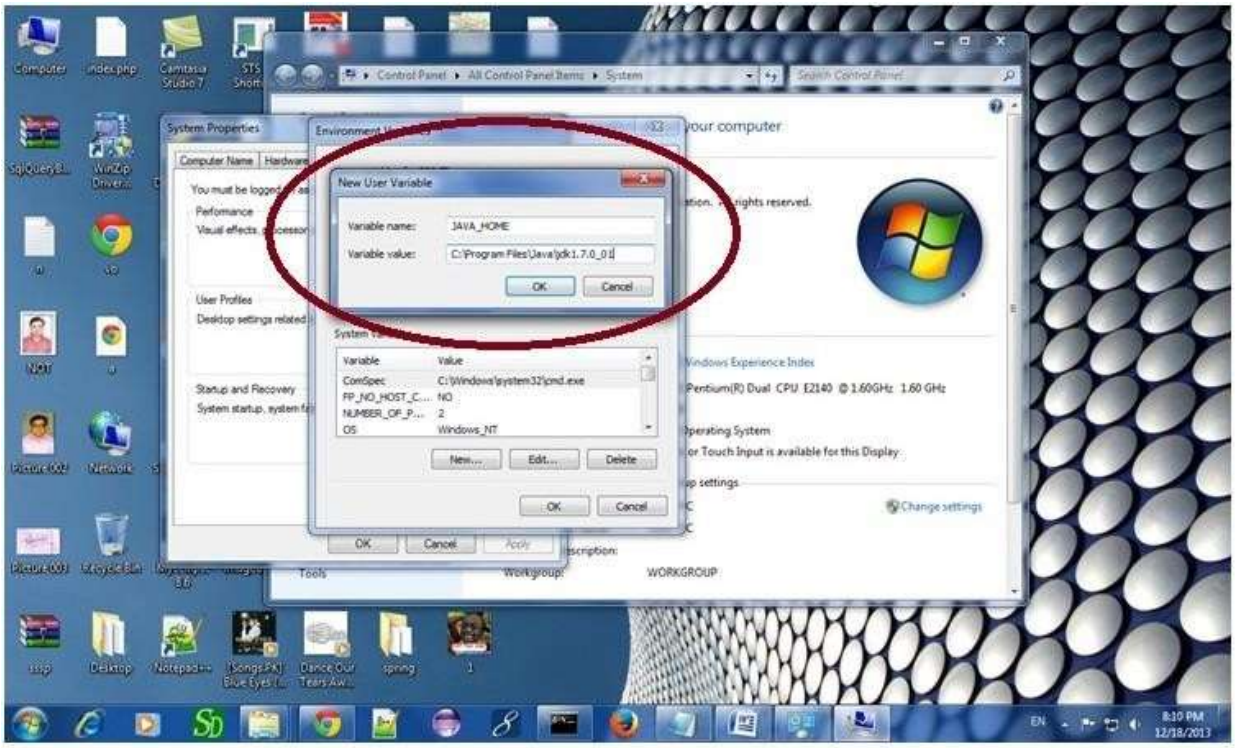




Click on the new tab of user variable or system variable



Write JAVA_HOME in variable name and paste the path of jdk folder in variable value



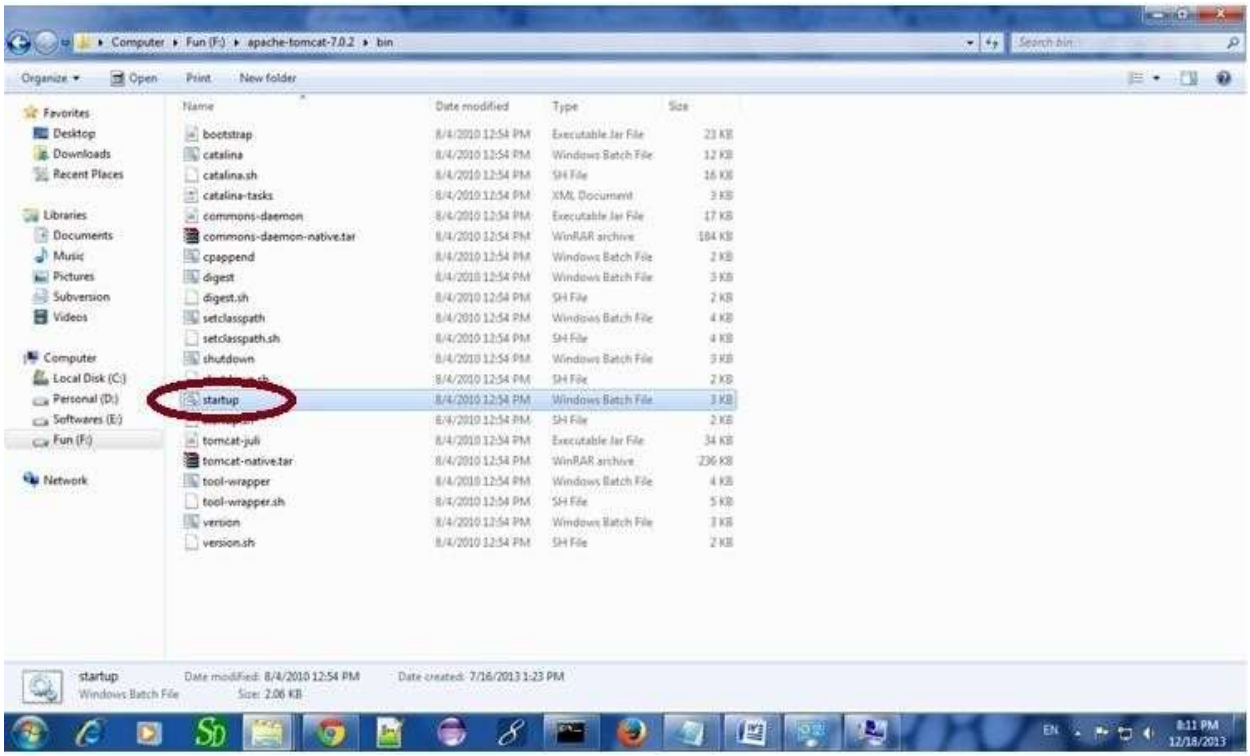
There must not be semicolon (;) at the end of the path.

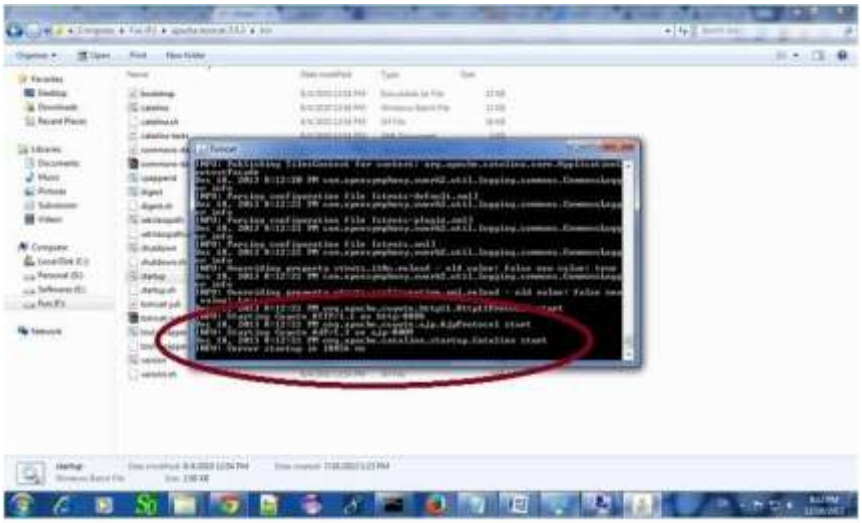
After setting the JAVA_HOME double click on the startup.bat file in apache tomcat/bin.

There are two types of tomcat available:

1. Apache tomcat that needs to extract only (no need to install)
2. Apache tomcat that needs to install

It is the example of apache tomcat that needs to extract only.





Now server is started successfully.

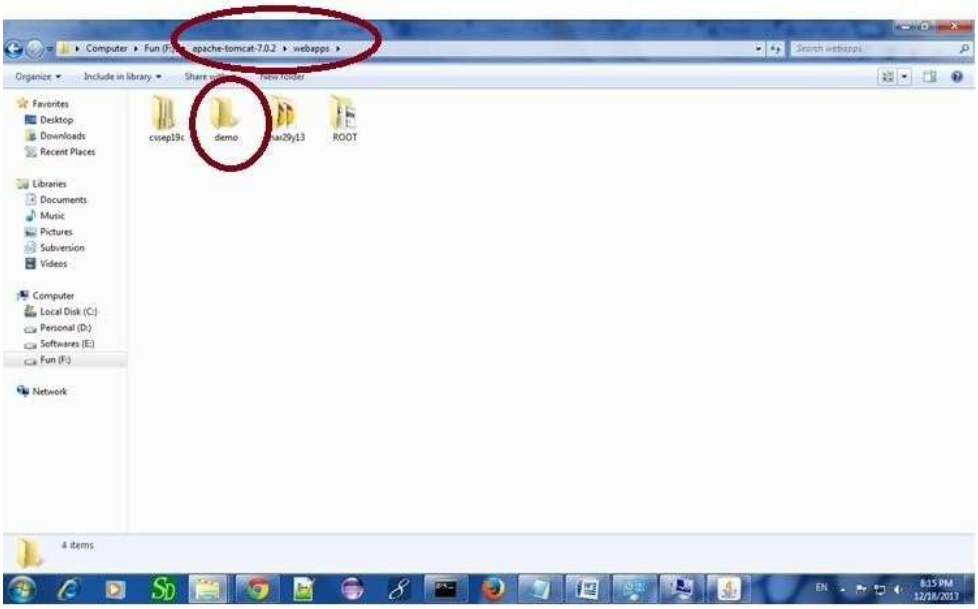
Change port number of apache tomcat

Changing the port number is required if there is another server running on the same system with same port number.Suppose you have installed oracle, you need to change the port number of apache tomcat because both have the default port number 8080.

Open server.xml file in notepad. It is located inside the apache-tomcat/conf directory . Change the Connector port = 8080 and replace 8080 by any four digit number instead of 8080. Let us replace it by 9999 and save this file.

Deploy the servlet project

Copy the project and paste it in the webapps folder under apache tomcat.



But there are several ways to deploy the project.

- By copying the context(project) folder into the webapps directory
- By copying the war folder into the webapps directory
- By selecting the folder path from the server
- By selecting the war file from the server

We are using the first approach.

You can also create war file, and paste it inside the webapps directory. To do so, you need to use jar tool to create the war file. Go inside the project directory (before the WEB-INF), then write:

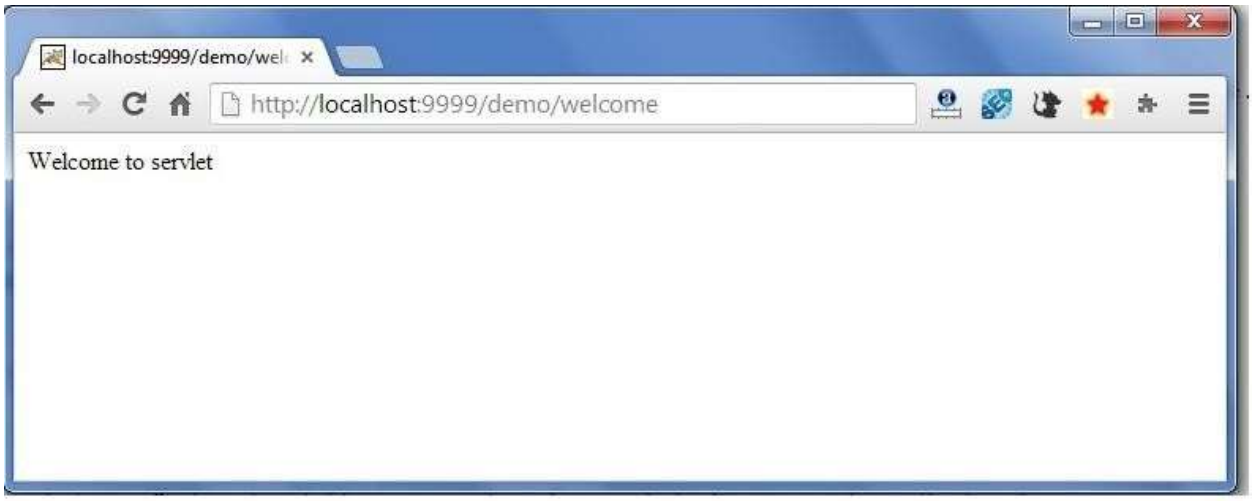
projectfolder> jar cvf myproject.war *

Creating war file has an advantage that moving the project from one location to another takes less time.

Access the servlet

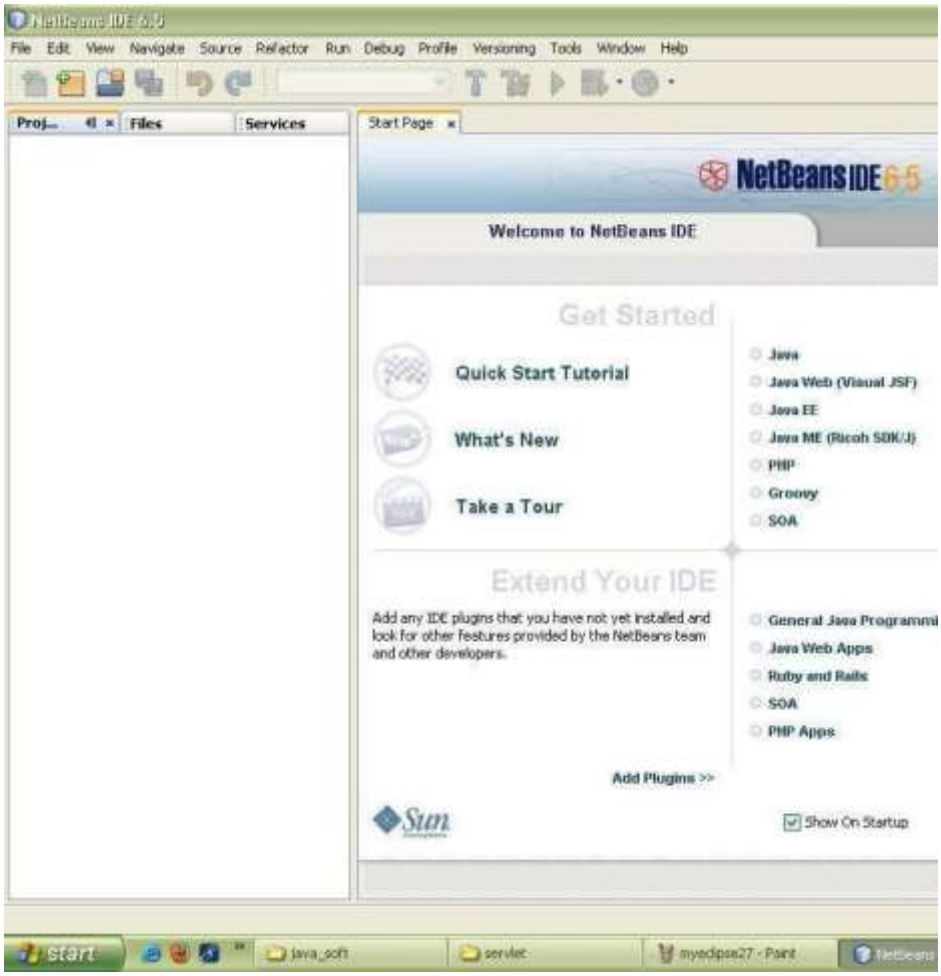
Open browser and write http://hostname:portno/contextroot/urlpatternofservlet. For example:

http://localhost:9999/demo/welcome



Creating a servlet in NetBeans IDE

1. Open NetBeans IDE

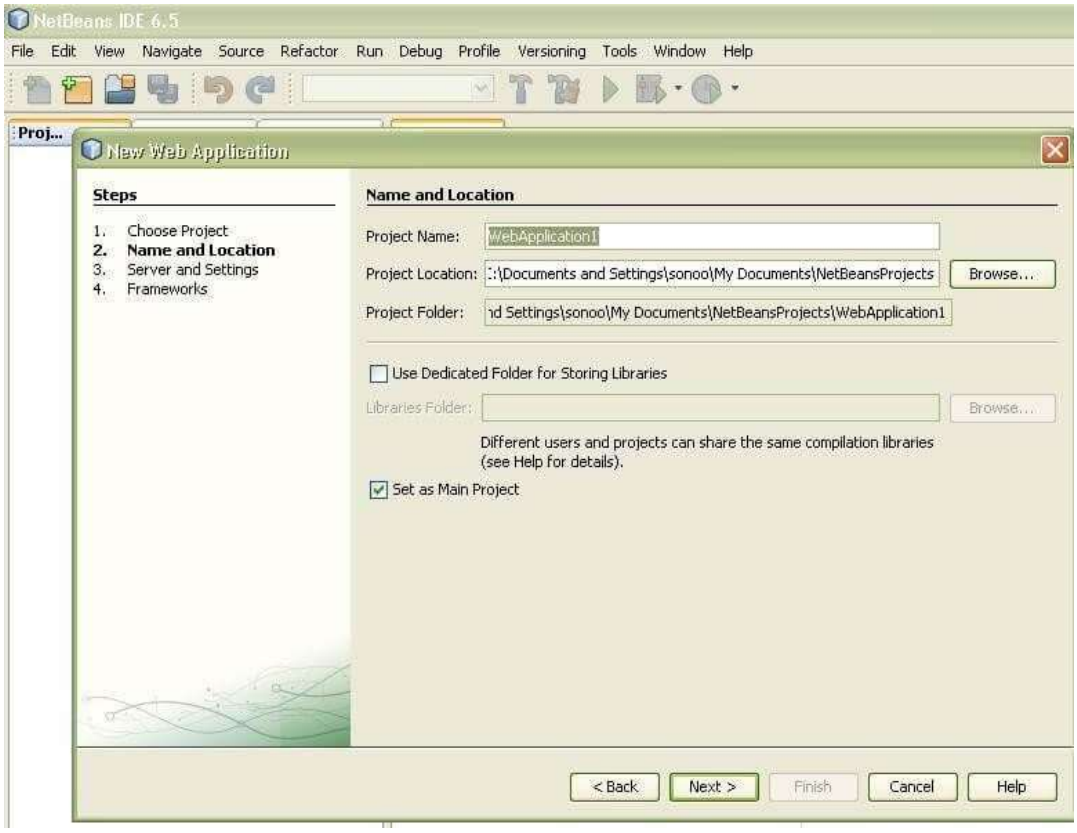


2. Select File > New Project

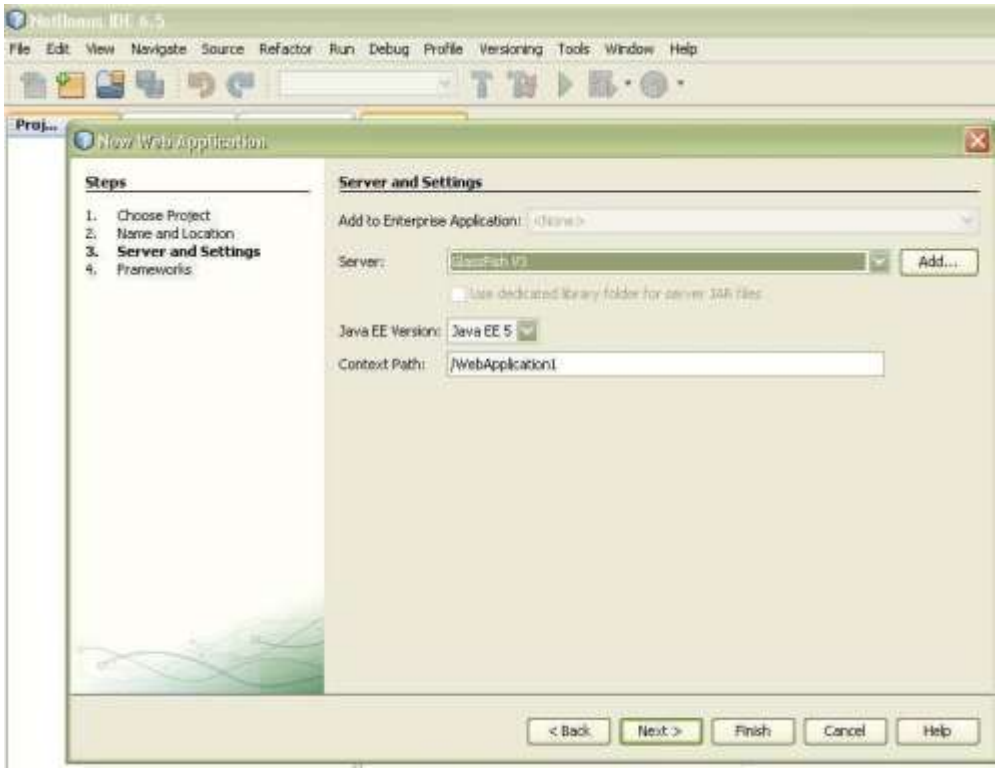
3. Select Java.

4. Select Web Application and then click on next.

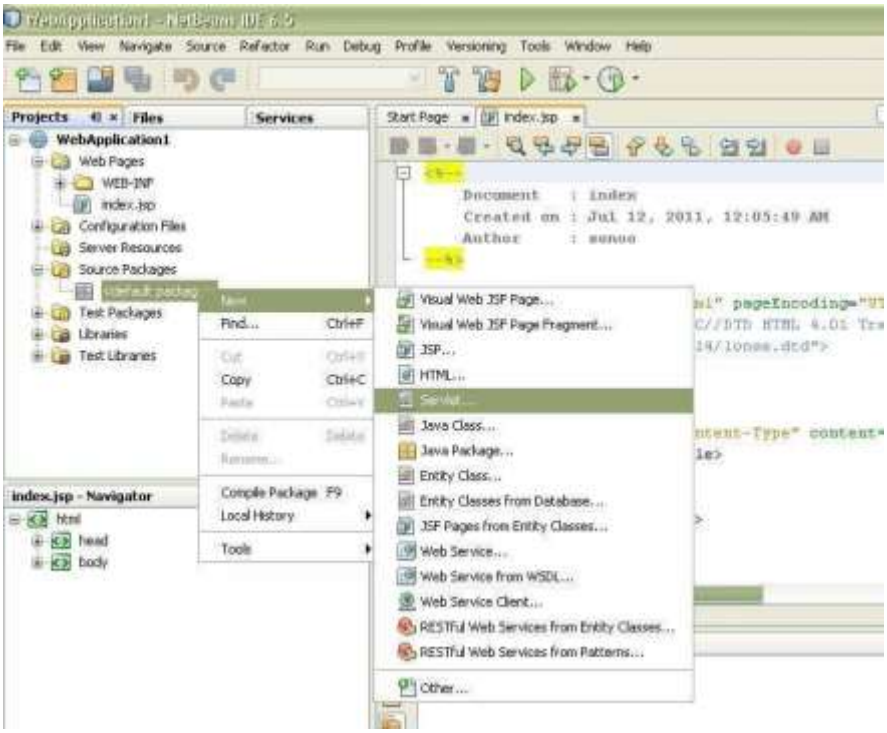
5. Give a name to the project and click on next.



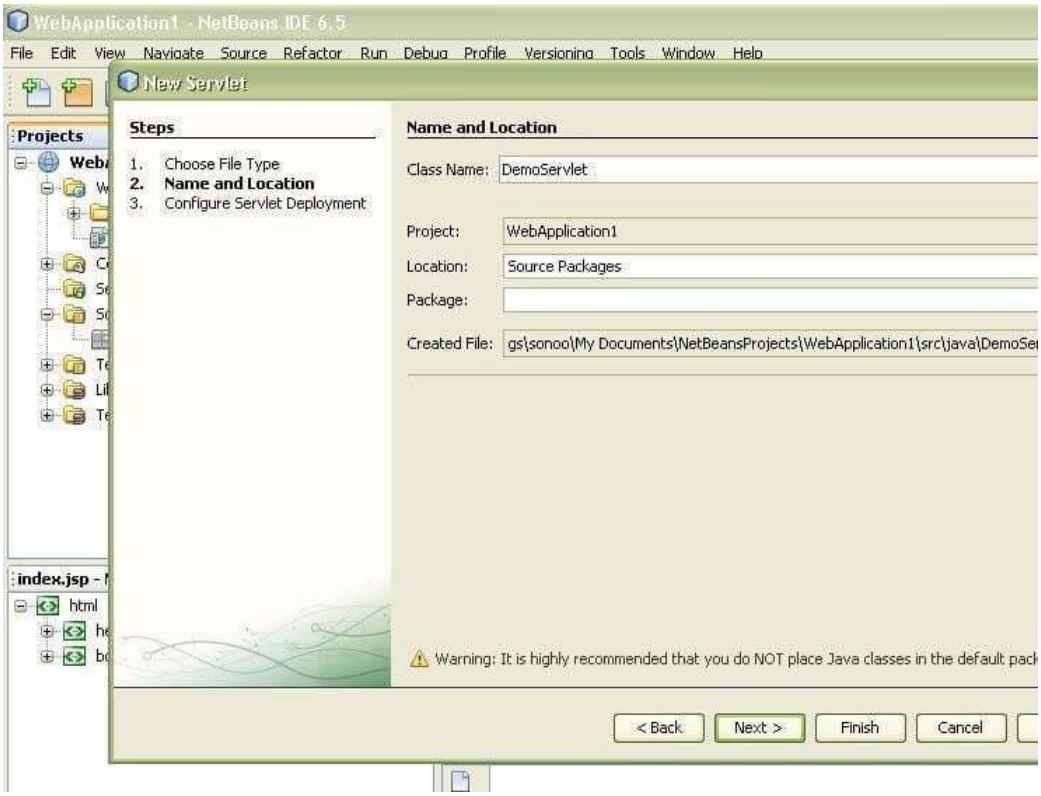
6. Select the server such as GlassFish or Tomcat and click Finish.



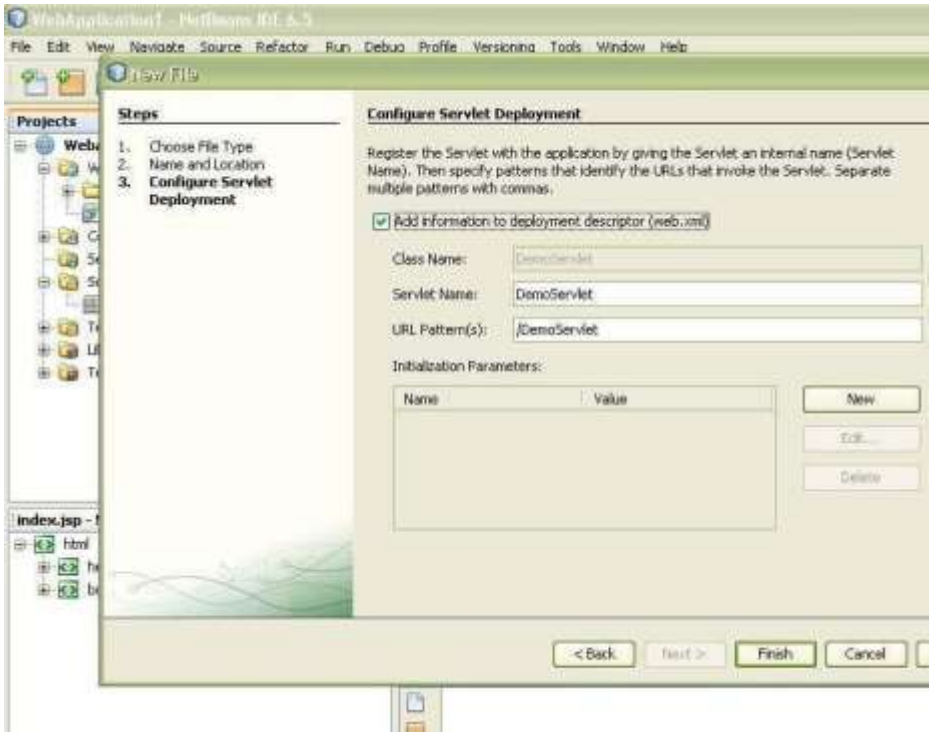
7. The complete directory structure required for the Servlet Application will be created automatically by the IDE. To create a servlet, open source package, right click on default package > New > Servlet.



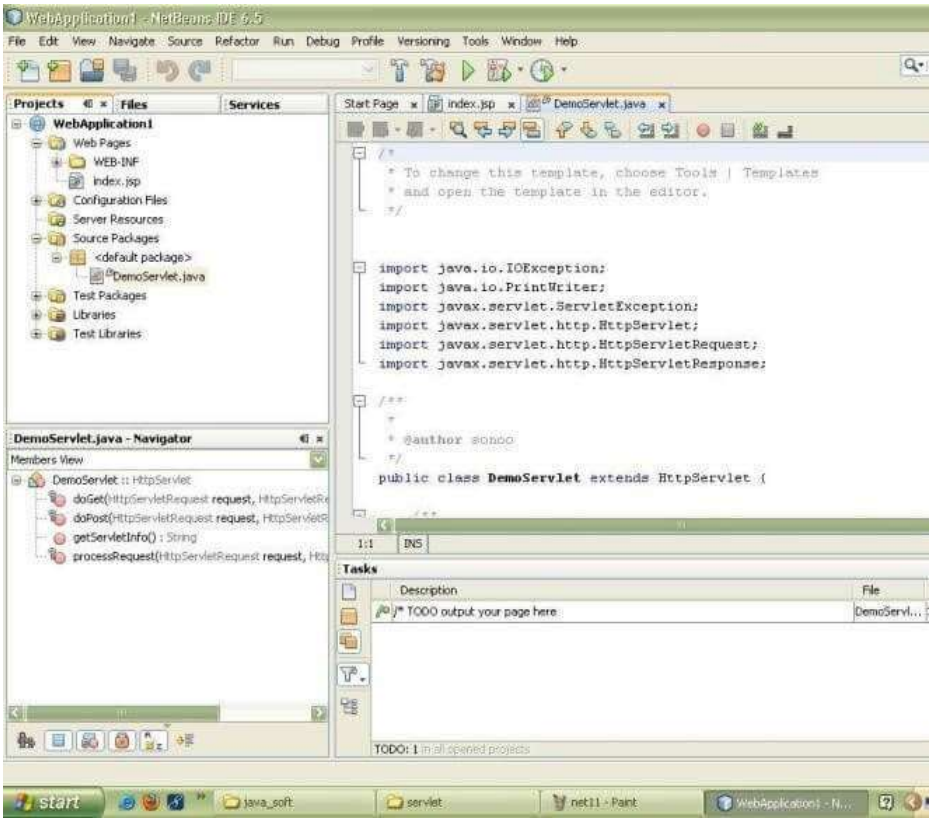
8. Provide a name to the Servlet class file and click next.



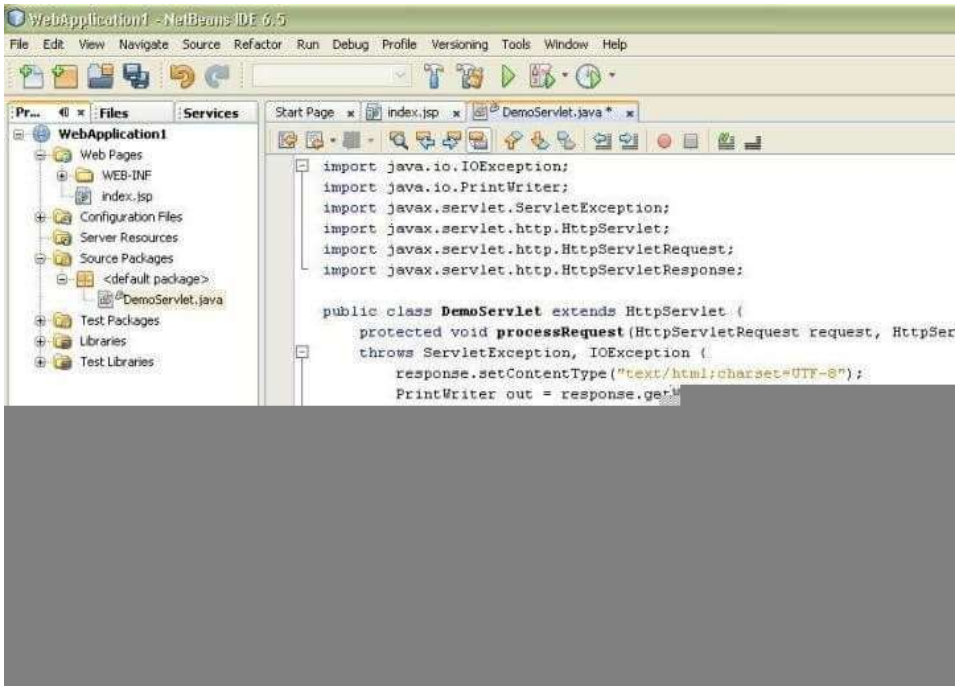
9. Provide Servlet name and URL pattern and click next.



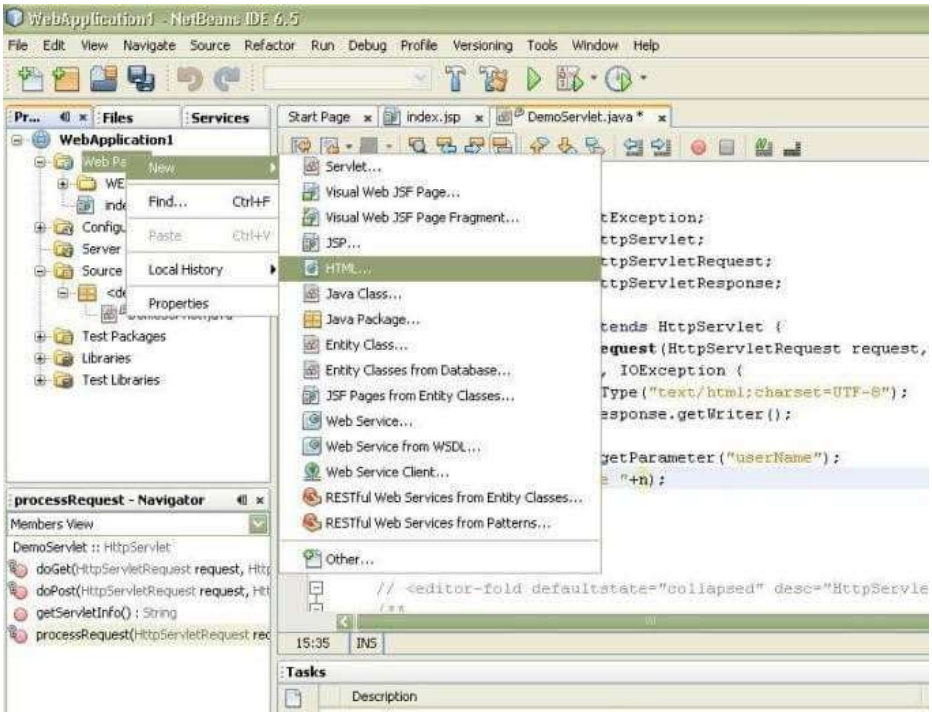
10. Servlet is ready and you just need to change the method definitions.



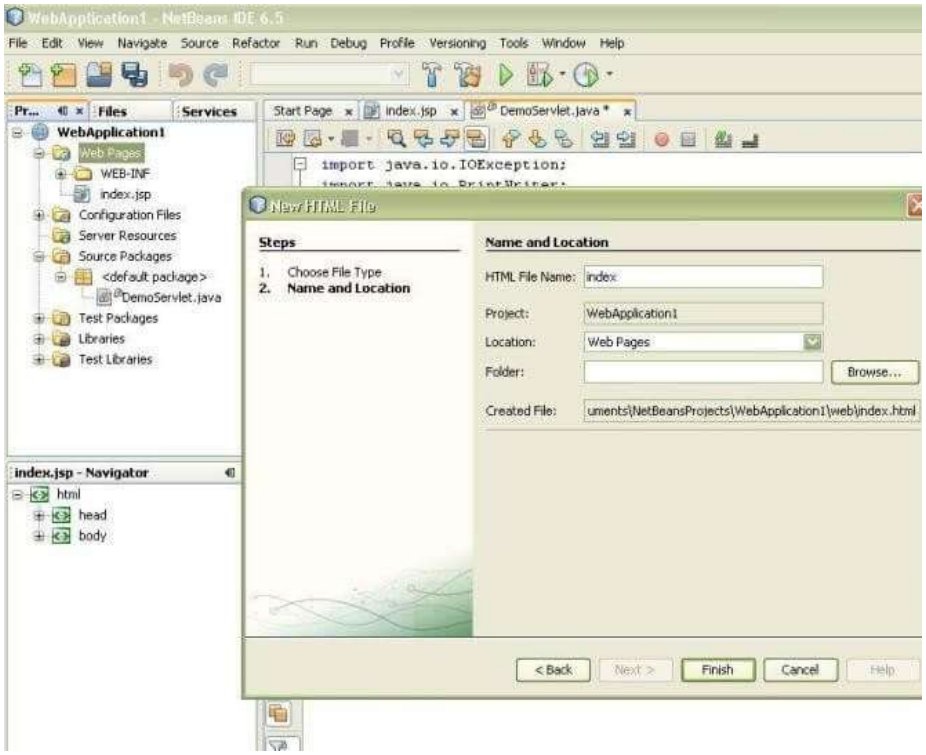
11. Write some code inside the Servlet class.



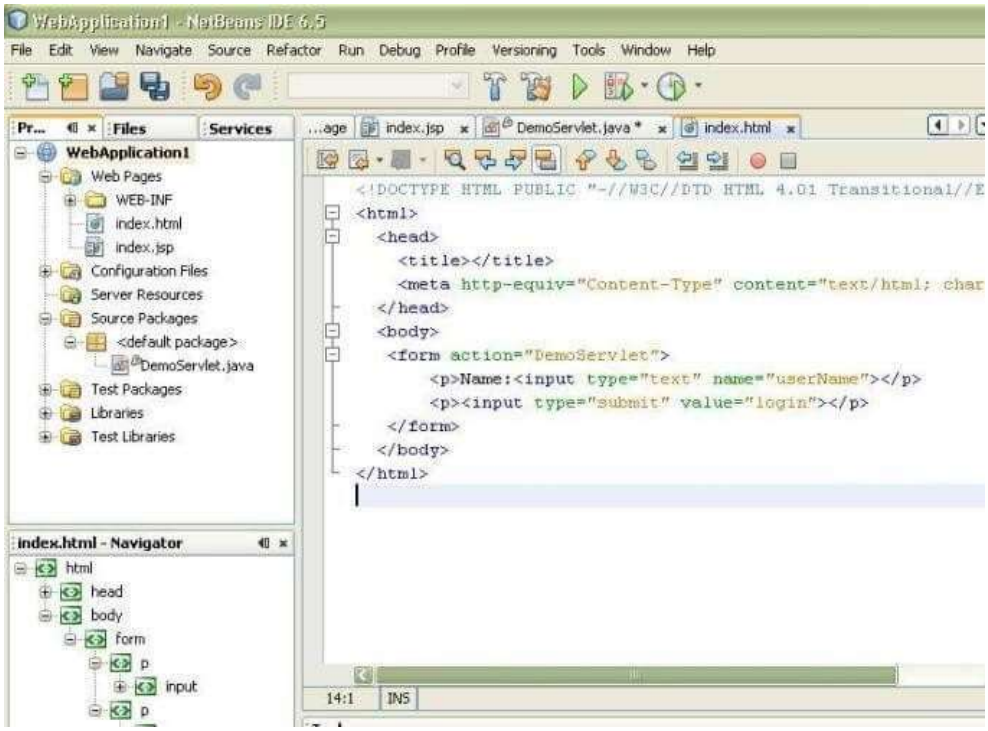
12. Create an HTML file, right click on web pages > new > next.

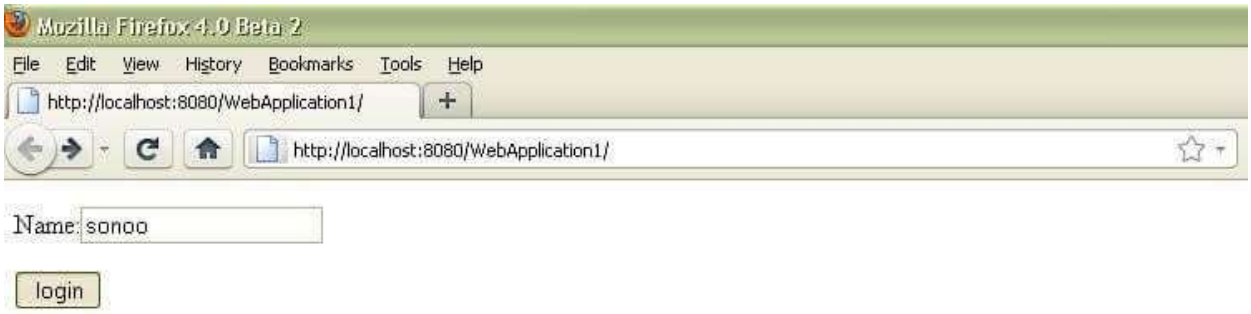


13. Provide a name for the HTML file. It is preferred to use index as the name, because the Bowser will always pick up the index.html file automatically from a directory.



14. Write some code inside the HTML file. A hyperlink is created to the servlet, in the HTML file. To run the application right click on project and select run.



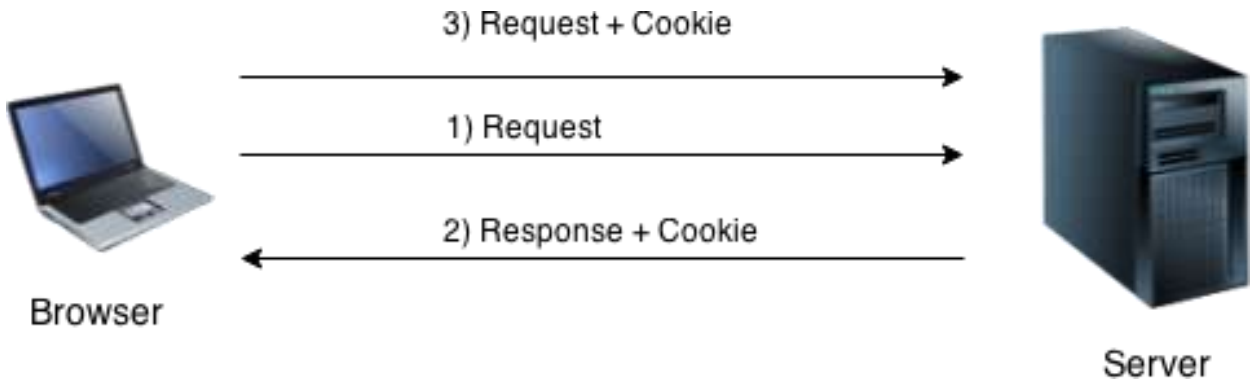


Cookies in Servlet

A cookie is a small piece of information that is persisted between the multiple client requests. A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.

Cookie works

By default, each request is considered as a new request. In cookies technique, we add cookie with response from the servlet. So cookie is stored in the cache of the browser. After that if request is sent by the user, cookie is added with request by default. Thus, we recognize the user as the old user.



Types of Cookie

There are 2 types of cookies in servlets.

1. Non-persistent cookie
2. Persistent cookie

Non-persistent cookie

It is valid for single session only. It is removed each time when user closes the browser.

Persistent cookie

It is valid for multiple session . It is not removed each time when user closes the browser. It is removed only if user logout or signout.

Advantage of Cookies

1. Simplest technique of maintaining the state.
2. Cookies are maintained at client side.

Disadvantage of Cookies

1. It will not work if cookie is disabled from the browser.
2. Only textual information can be set in Cookie object.

Cookie class

javax.servlet.http.Cookie class provides the functionality of using cookies. It provides a lot of useful methods for cookies.

Constructor of Cookie class

Constructor	Description
Cookie()	constructs a cookie.
Cookie(String name, String value)	constructs a cookie with a specified name and value.

Methods of Cookie class

Method	Description
public void setMaxAge(int expiry)	Sets the maximum age of the cookie in seconds.
public String getName()	Returns the name of the cookie. The name cannot be changed after creation.
public String getValue()	Returns the value of the cookie.
public void setName(String name)	changes the name of the cookie.
public void setValue(String value)	changes the value of the cookie.

Methods required for using Cookies

For adding cookie or getting the value from the cookie, we need some methods provided by other interfaces. They are:

1. public void addCookie(Cookie ck):method of HttpServletResponse interface is used to add cookie in response object.
2. public Cookie[] getCookies():method of HttpServletRequest interface is used to return all the cookies from the browser.

Create Cookie

```
Cookie ck=new Cookie("user","sonoo jaiswal");//creating cookie object
response.addCookie(ck);//adding cookie in the response
```

Delete Cookie

```
Cookie ck=new Cookie("user","");//deleting value of cookie
```

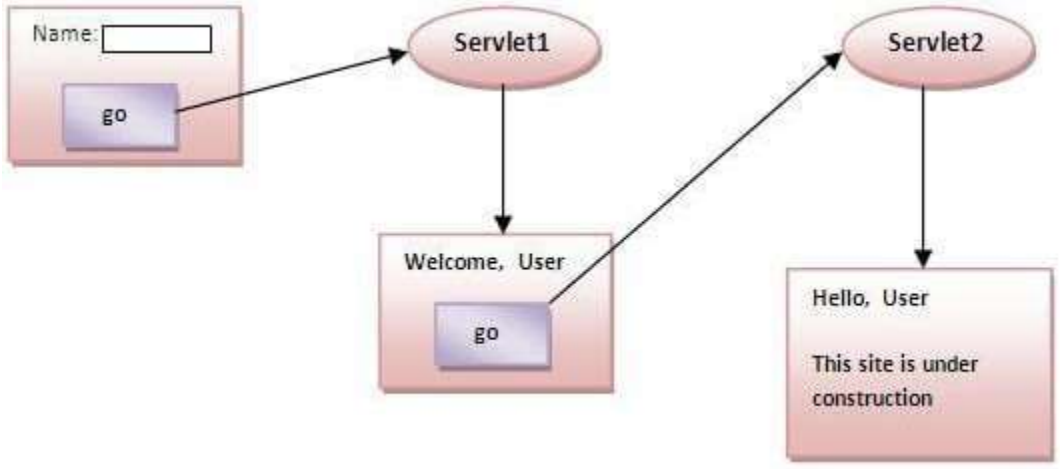
```
ck.setMaxAge(0);//changing the maximum age to 0 seconds
response.addCookie(ck);//adding cookie in the response
```

Get Cookies

```
Cookie ck[]=request.getCookies();
for(int i=0;i<ck.length;i++){
out.print("<br>"+ck[i].getName()+" "+ck[i].getValue());//printing name and value of cookie
}
```

Simple Servlet Cookies

We are storing the name of the user in the cookie object and accessing it in another servlet. As we know well that session corresponds to the particular user. So if you access it from too many browsers with different values, you will get the different value.



index.html

```
<form action="servlet1" method="post">
Name:<input type="text" name="userName"/><br/>
<input type="submit" value="go"/>
</form>
```

FirstServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
  public class FirstServlet extends HttpServlet {
  public void doPost(HttpServletRequest request, HttpServletResponse response){
    try{
   response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String n=request.getParameter("userName");
    out.print("Welcome "+n);
```

```
    Cookie ck=new Cookie("uname",n);//creating cookie object
    response.addCookie(ck);//adding cookie in the response
    //creating submit button
    out.print("<form action='servlet2'>");
    out.print("<input type='submit' value='go'>");
    out.print("</form>");
    out.close();
  }catch(Exception e){System.out.println(e);}
  }
}
```

SecondServlet.java

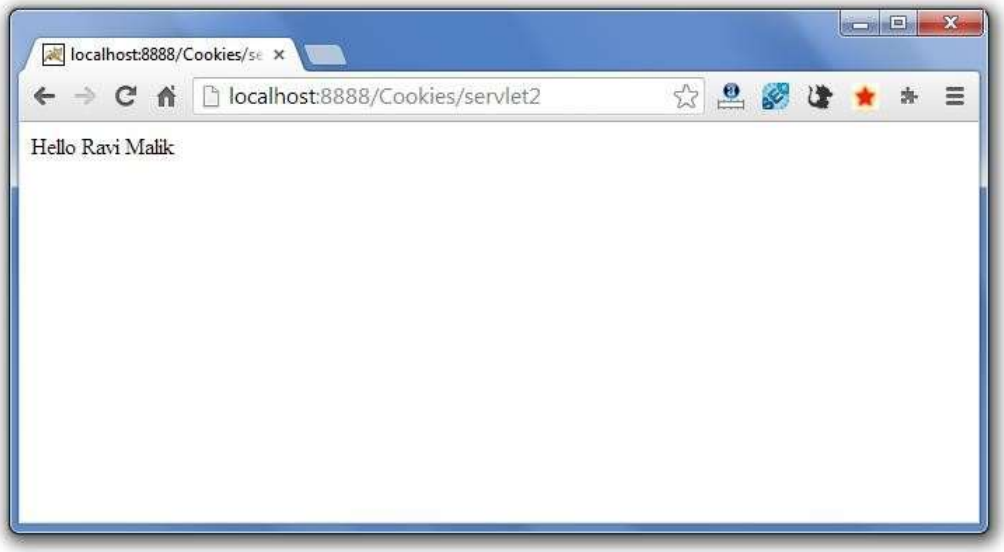
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
  public class SecondServlet extends HttpServlet {
  public void doPost(HttpServletRequest request, HttpServletResponse response){
    try{
    response.setContentType("text/html");
   PrintWriter out = response.getWriter();
   Cookie ck[]=request.getCookies();
   out.print("Hello "+ck[0].getValue());
   out.close();
  }catch(Exception e){System.out.println(e);}
    }
  }
```

web.xml

```
<web-app>
<servlet>
<servlet-name>s1</servlet-name>
<servlet-class>FirstServlet</servlet-class>
</servlet>
<servlet-mapping>
<servlet-name>s1</servlet-name>
<url-pattern>/servlet1</url-pattern>
</servlet-mapping>
<servlet>
<servlet-name>s2</servlet-name>
<servlet-class>SecondServlet</servlet-class>
</servlet>
<servlet-mapping>
```

<servlet-name>s2</servlet-name>
<url-pattern>/servlet2</url-pattern>
</servlet-mapping>
</web-app>

Output



UNIT IV
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.

Extension to Servlet

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

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.

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.

Less code than Servlet

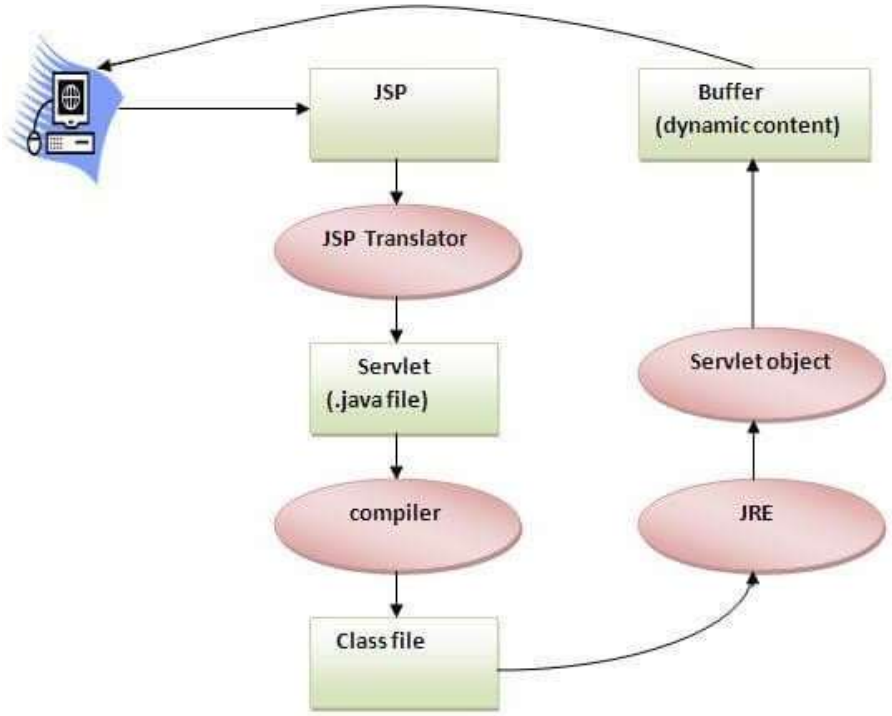
In JSP, we can use many tags such as action tags, JSTL, custom tags, etc. that reduces the code. We can use EL, implicit objects, etc.

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 jspInit() method).
- Request processing (the container invokes _jspService() method).
- Destroy (the container invokes jspDestroy() method).

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. 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

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

It will print 10 on the browser.

Run a simple JSP Page

Follow the following steps to execute this JSP page:

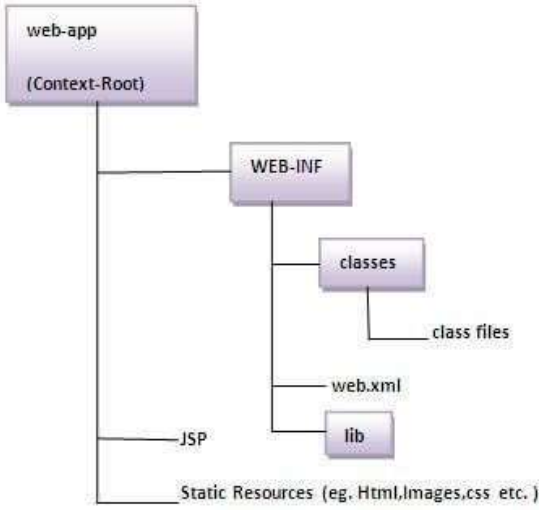
- Start the server
- Put the JSP file in a folder and deploy on the server
- Visit the browser by the URL http://localhost:portno/contextRoot/jspfile, for example, http://localhost:8888/myapplication/index.jsp

Directory structure

There is no need of directory structure if you don't have class files or TLD files. For example, put JSP files in a folder directly and deploy that folder. It will be running fine. However, if you are using Bean class, Servlet or TLD file, the directory structure is required.

Directory structure of JSP

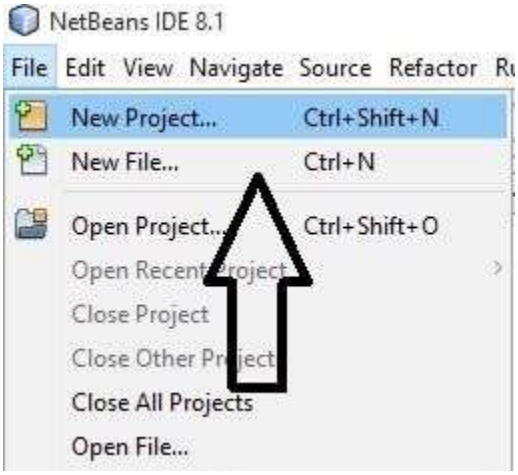
The directory structure of JSP page is same as Servlet. We contain the JSP page outside the WEB-INF folder or in any directory.



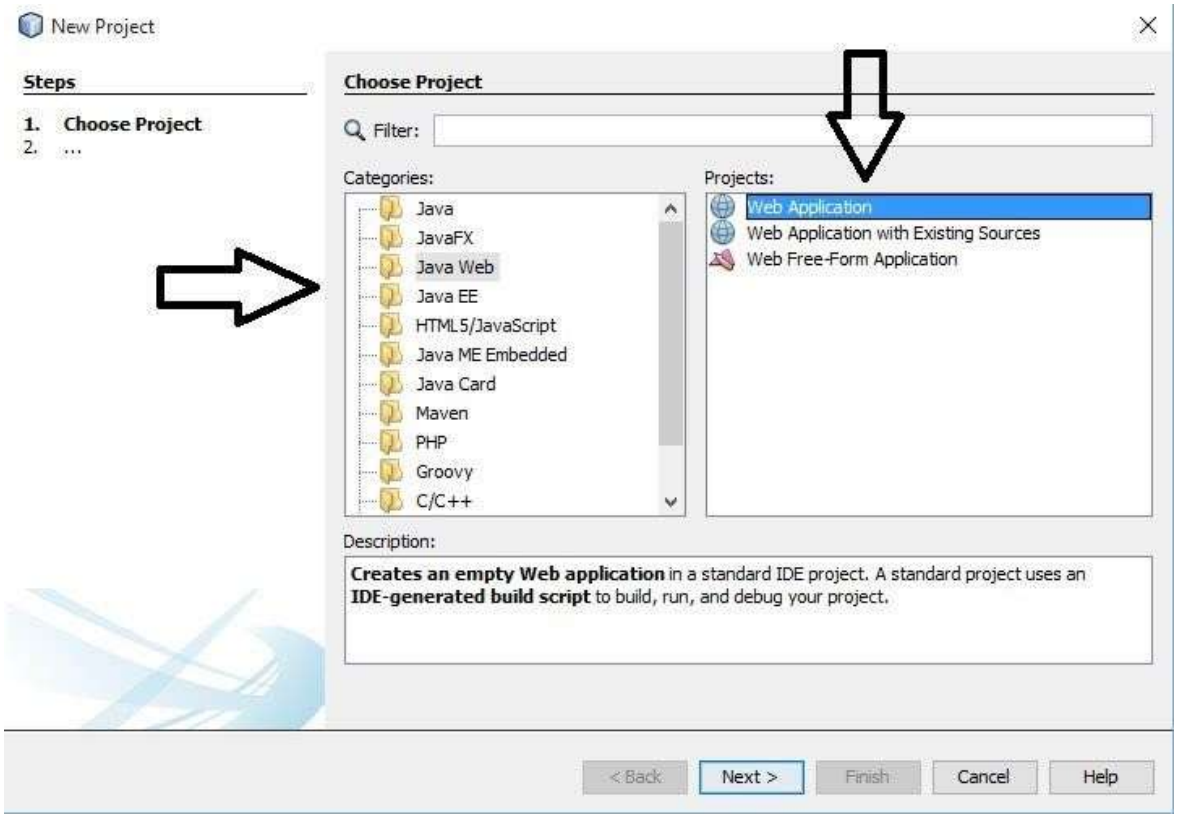
Implementing JSP on NetBeans IDE

Starting with implementing JSP on NetBeans IDE is easy.

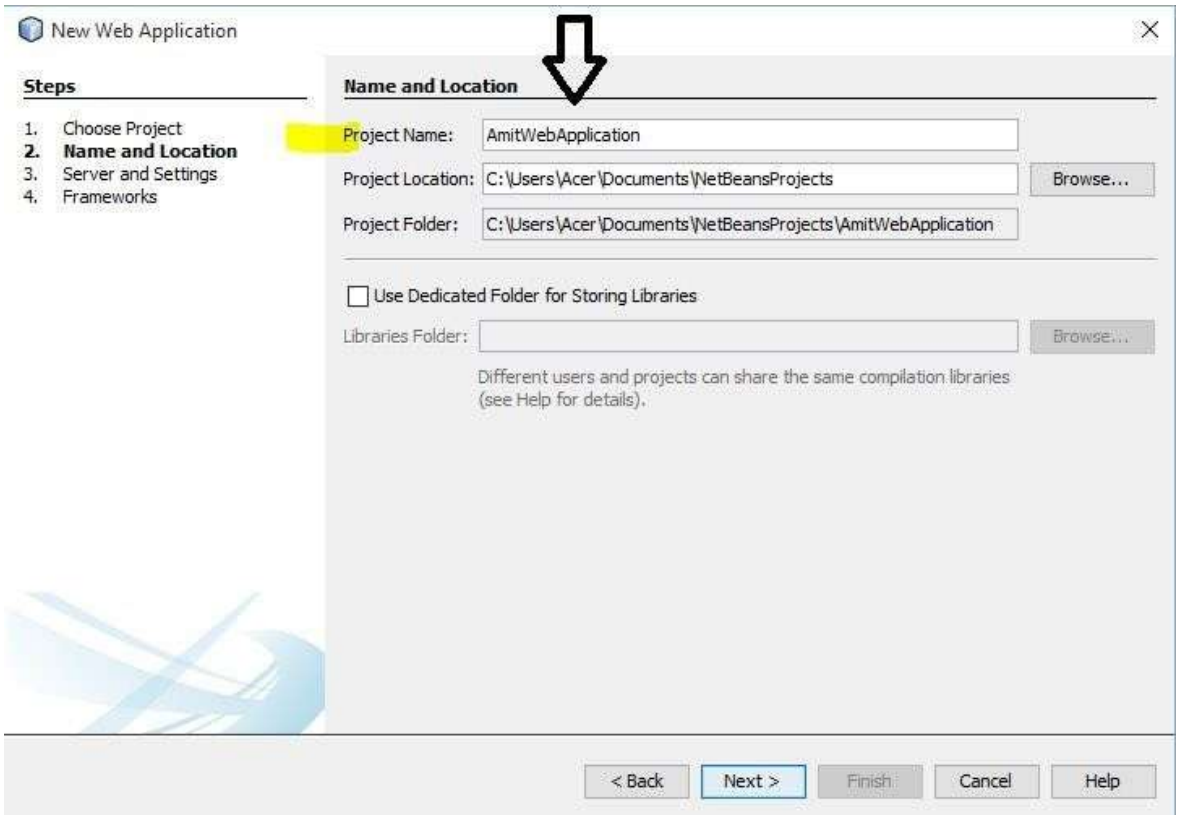
1. Open NetBeans IDE & click File menu and click New Project.



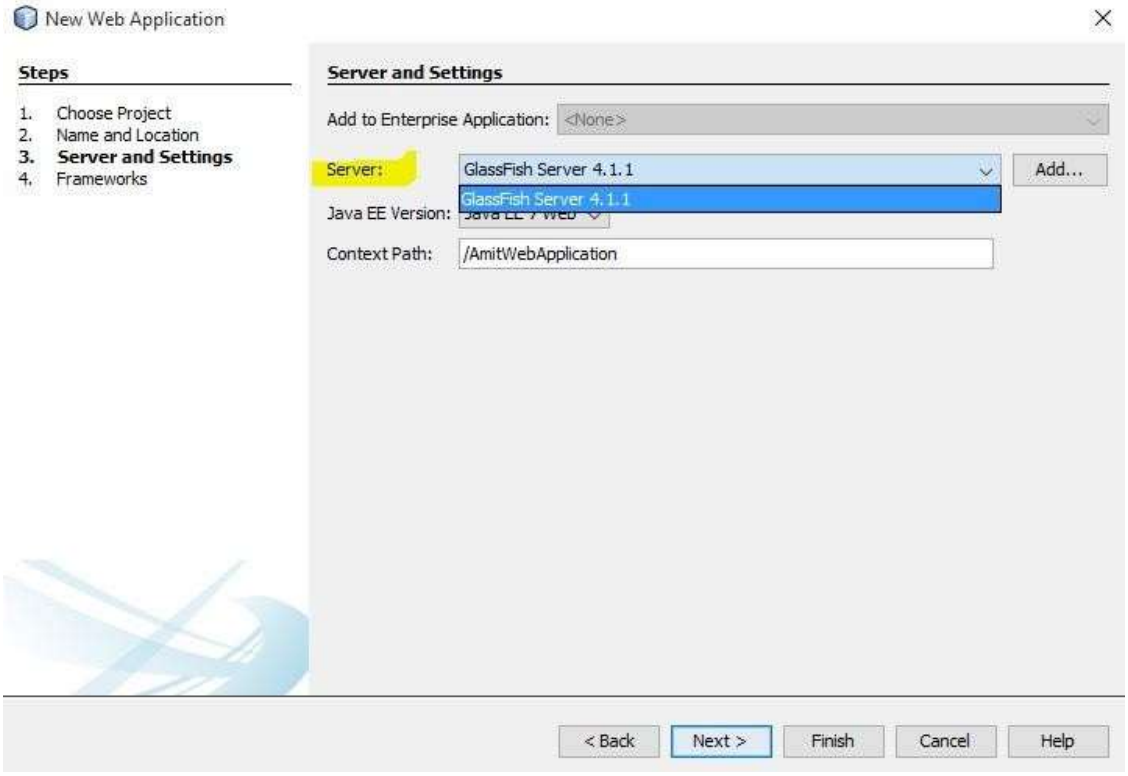
2. Select Java Web, after that Web Application and then click Next.



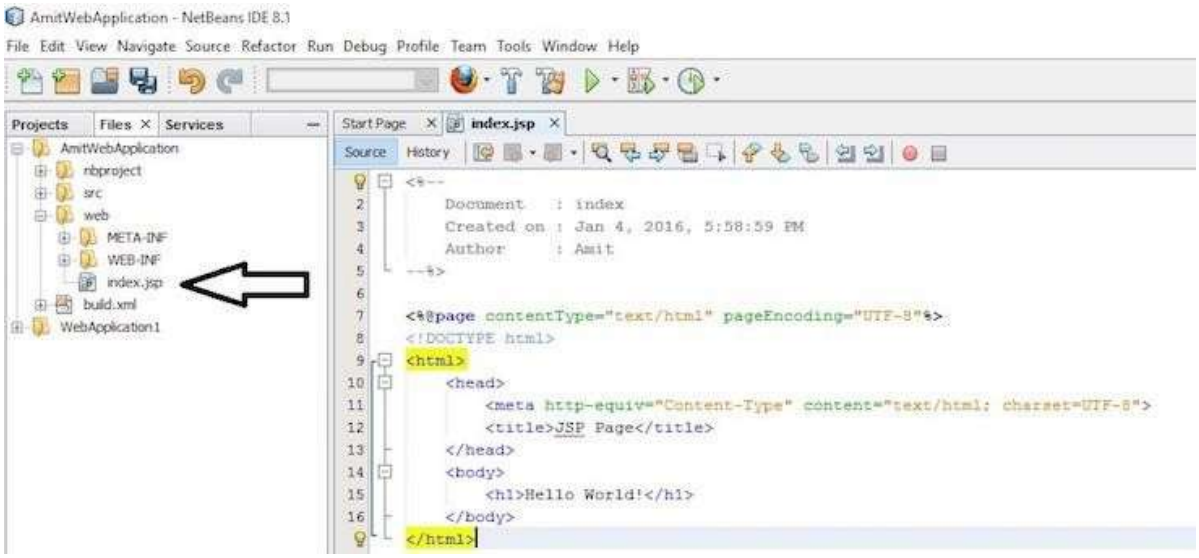
3. Give a name to the project and click on next.



4. Select the server Glassfish and click finish.



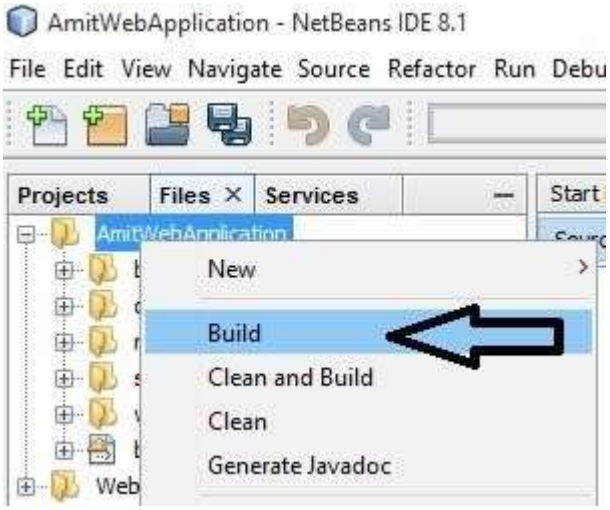
5. The complete directory structure required for JSP application will be created automatically by the IDE. A default index.jsp file is created under the folder web pages.



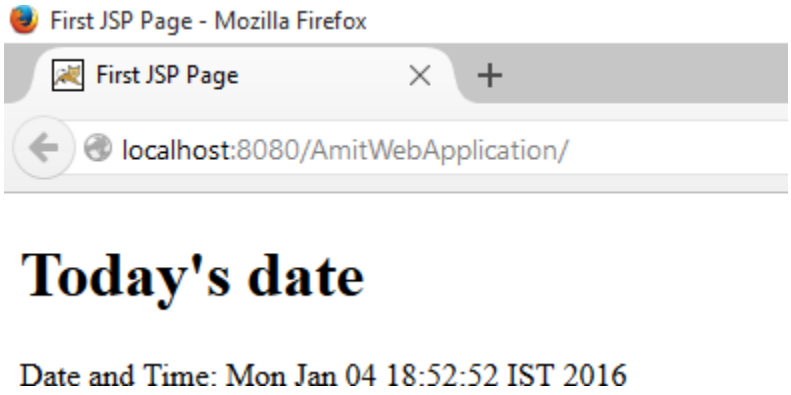
6. Write some code with JSP syntax in the created file.

```
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
  <title>First JSP Page</title>
</head>
<body>
  <h1>Today's date</h1>
  Date and Time: <%= (new java.util.Date().toString())%>
</body>
</html>
```

7. To run the index.jsp file, build the project. Right click on the project "AmitWebApplication" and click Build.



Output



JSP directives

The jsp directives are messages that tell the web container how to translate a JSP page into the corresponding servlet.

There are three types of directives:

1. page directive
2. include directive
3. taglib directive

Syntax of JSP Directive

<%@ directive attribute="value" %>

JSP page directive

The page directive defines attributes that apply to an entire JSP page.

Syntax of JSP page directive

<%@ page attribute="value" %>

import

The import attribute is used to import class,interface or all the members of a package.It is similar to import keyword in java class or interface.

<%@ page import="java.util.Date" %>

contentType

The contentType attribute defines the MIME(Multipurpose Internet Mail Extension) type of the HTTP response.

<%@ page contentType=application/msword %>

extends

The extends attribute defines the parent class that will be inherited by the generated servlet. It is rarely used.

info

This attribute simply sets the information of the JSP page which is retrieved later by using getServletInfo() method of Servlet interface.

<%@ page info="composed by Sonoo Jaiswal" %>

buffer

The buffer attribute sets the buffer size in kilobytes to handle output generated by the JSP page. The default size of the buffer is 8Kb.

<%@ page buffer="16kb" %>

language

The language attribute specifies the scripting language used in the JSP page. The default value is "java".

isThreadSafe

Servlet and JSP both are multithreaded. If you want to control this behaviour of JSP page, you can use isThreadSafe attribute of page directive. The value of isThreadSafe value is true. If you

make it false, the web container will serialize the multiple requests, i.e. it will wait until the JSP finishes responding to a request before passing another request to it. If you make the value of isThreadSafe attribute like:

<%@ page isThreadSafe="false" %>

errorPage

The errorPage attribute is used to define the error page, if exception occurs in the current page, it will be redirected to the error page.

<%@ page errorPage="myerrorpage.jsp" %>

isErrorPage

The isErrorPage attribute is used to declare that the current page is the error page.

<%@ page isErrorPage="true" %>

Jsp Include Directive

The include directive is used to include the contents of any resource it may be jsp file, html file or text file. The include directive includes the original content of the included resource at page translation time (the jsp page is translated only once so it will be better to include static resource).

Advantage of Include directive

Syntax of include directive

<%@ include file="resourceName" %>

Example of include directive

```
<html>
<body>
<%@ include file="header.html" %>
Today is: <%= java.util.Calendar.getInstance().getTime() %>
</body>
</html>
```

JSP Taglib directive

The JSP taglib directive is used to define a tag library that defines many tags. We use the TLD (Tag Library Descriptor) file to define the tags.

Syntax JSP Taglib directive

<%@ taglib uri="urioftheteglibrary" prefix="prefixoftaglibrary" %>

Example of JSP Taglib directive

```
<html>
<body>
<%@ taglib uri="http://www.javatpoint.com/tags" prefix="mytag" %>
<mytag:currentDate/>
</body>
</html>
```

JSP Scripting elements

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

1. scriptlet tag
2. expression tag
3. declaration tag

JSP scriptlet tag

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

<% java source code %>

Example of JSP scriptlet tag

```
<html>
<body>
<% out.print("welcome to jsp"); %>
</body>
</html>
```

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.

index.html

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

welcome.jsp

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

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 of JSP expression tag

<%= statement %>

Example of JSP expression tag

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

```
<html>
<body>
<%= "welcome to jsp" %>
</body>
</html>
```

Example of JSP expression tag that prints current time

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>
```

Example of JSP expression tag that prints the user name

In this example, we are printing the username using the expression tag. The index.html file gets the username and sends the request to the welcome.jsp file, which displays the username.

File: index.jsp

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

File: welcome.jsp

```
<html>
<body>
<%= "Welcome "+request.getParameter("uname") %>
```

```
</body>
</html>
```

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 does not get memory at each request.

Syntax of JSP declaration tag

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

Difference between JSP Scriptlet tag and Declaration tag

Jsp Scriptlet Tag	Jsp Declaration Tag
The jsp scriptlet tag can only declare variables not methods.	The jsp declaration tag can declare variables as well as methods.
The declaration of scriptlet tag is placed inside the _jspService() method.	The declaration of jsp declaration tag is placed outside the _jspService() method.

Example of JSP declaration tag

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>
```

Example of JSP declaration tag

In this example of JSP declaration tag, we are defining the method which returns the cube of given number and calling this method from the jsp expression tag. But we can also use jsp scriptlet tag to call the declared method.

index.jsp

```
<html>
<body>
<%!
int cube(int n){
return n*n*n*;
}
%>
<%= "Cube of 3 is:"+cube(3) %>
</body>
</html>
```

JSP Action Tags

There are many JSP action tags or elements. Each JSP action tag is used to perform some specific tasks. The action tags are used to control the flow between pages and to use Java Bean.

JSP Action Tags	Description
jsp:forward	forwards the request and response to another resource.
jsp:include	includes another resource.
jsp:useBean	creates or locates bean object.
jsp:setProperty	sets the value of property in bean object.
jsp:getProperty	prints the value of property of the bean.
jsp:plugin	embeds another components such as applet.
jsp:param	sets the parameter value. It is used in forward and include mostly.
jsp:fallback	can be used to print the message if plugin is working. It is used in jsp:plugin.

jsp:forward action tag

The jsp:forward action tag is used to forward the request to another resource it may be jsp, html or another resource.

Syntax of jsp:forward action tag without parameter

```
<jsp:forward page="relativeURL | <%= expression %>" />
```

Syntax of jsp:forward action tag with parameter

```
<jsp:forward page="relativeURL | <%= expression %>">
<jsp:param name="parametername" value="parametervalue | <%=expression%>" />
</jsp:forward>
```

Example of jsp:forward action tag without parameter

In this example, we are simply forwarding the request to the printdate.jsp file.

index.jsp

```
<html>
<body>
<h2>this is index page</h2>
<jsp:forward page="printdate.jsp" />
</body>
</html>
```

printdate.jsp

```
<html>
<body>
<% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
</body>
</html>
```

Example of jsp:forward action tag with parameter

In this example, we are forwarding the request to the printdate.jsp file with parameter and printdate.jsp file prints the parameter value with date and time.

index.jsp

```
<html>
<body>
<h2>this is index page</h2>
<jsp:forward page="printdate.jsp" >
<jsp:param name="name" value="javatpoint.com" />
</jsp:forward>
</body>
</html>
```

printdate.jsp

```
<html>
<body>
<% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
<%= request.getParameter("name") %>
</body>
</html>
```

jsp:include action tag

The jsp:include action tag is used to include the content of another resource it may be jsp, html or servlet. The jsp include action tag includes the resource at request time so it is better for dynamic pages because there might be changes in future. The jsp:include tag can be used to include static as well as dynamic pages.

Advantage of jsp:include action tag

We can use a page many times such as including header and footer pages in all pages. So it saves a lot of time.

Difference between jsp include directive and include action

JSP include directive	JSP include action
includes resource at translation time.	includes resource at request time.
better for static pages.	better for dynamic pages.
includes the original content in the generated servlet.	calls the include method.

Syntax of jsp:include action tag without parameter

```
<jsp:include page="relativeURL | <%= expression %>" />
```

Syntax of jsp:include action tag with parameter

```
<jsp:include page="relativeURL | <%= expression %>">
<jsp:param name="parametername" value="parametervalue | <%=expression%>" />
</jsp:include>
```

Example of jsp:include action tag without parameter

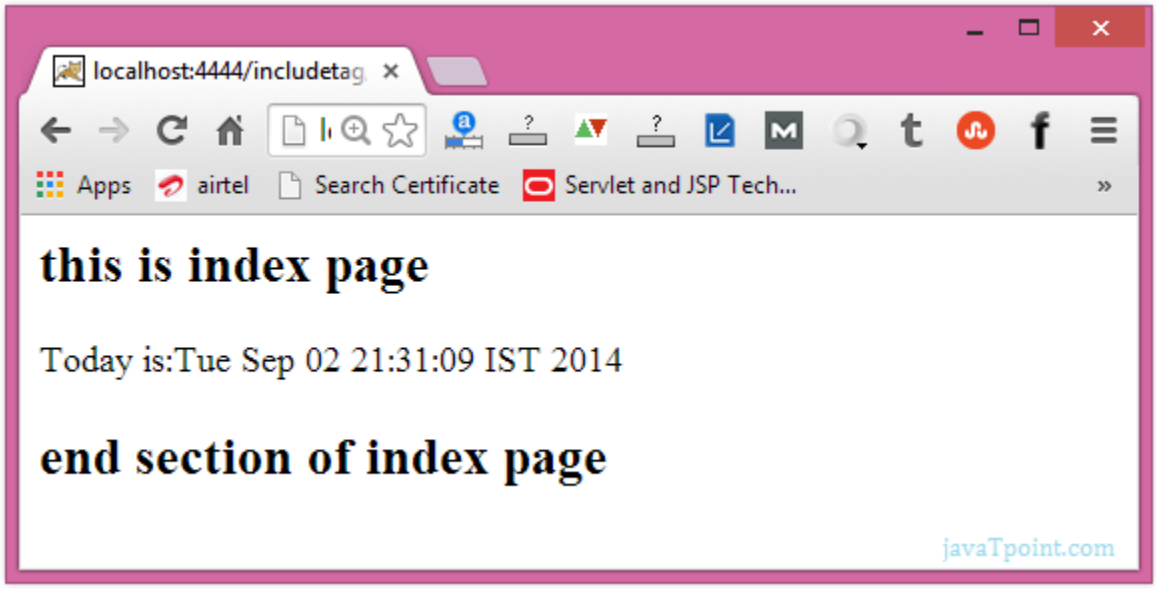
In this example, index.jsp file includes the content of the printdate.jsp file.

index.jsp

```
<h2>this is index page</h2>
<jsp:include page="printdate.jsp" />
<h2>end section of index page</h2>
```

printdate.jsp

```
<% out.print("Today is:"+java.util.Calendar.getInstance().getTime()); %>
```



jsp:useBean action tag

The jsp:useBean action tag is used to locate or instantiate a bean class. If bean object of the Bean class is already created, it doesn't create the bean depending on the scope. But if object of bean is not created, it instantiates the bean.

Syntax of jsp:useBean action tag

```
<jsp:useBean id= "instanceName" scope= "page | request | session | application"
class= "packageName.className" type= "packageName.className"
beanName="packageName.className | <%= expression >" >
</jsp:useBean>
```

Attributes and Usage of jsp:useBean action tag

1. id: is used to identify the bean in the specified scope.
2. scope: represents the scope of the bean. It may be page, request, session or application. The default scope is page.
    - page: specifies that you can use this bean within the JSP page. The default scope is page.
    - request: specifies that you can use this bean from any JSP page that processes the same request. It has wider scope than page.

- session: specifies that you can use this bean from any JSP page in the same session whether processes the same request or not. It has wider scope than request.
- application: specifies that you can use this bean from any JSP page in the same application. It has wider scope than session.

3. class: instantiates the specified bean class (i.e. creates an object of the bean class) but it must have no-arg or no constructor and must not be abstract.
4. type: provides the bean a data type if the bean already exists in the scope. It is mainly used with class or beanName attribute. If you use it without class or beanName, no bean is instantiated.
5. beanName: instantiates the bean using the java.beans.Beans.instantiate() method.

Example of jsp:useBean action tag

In this example, we are simply invoking the method of the Bean class.

Calculator.java (a simple Bean class)

```
package com.javatpoint;
public class Calculator
{
public int cube(int n)
{
return n*n*n;
}
}
```

index.jsp file

```
<jsp:useBean id="obj" class="com.javatpoint.Calculator"/>
<%  int m=obj.cube(5);
out.print("cube of 5 is "+m);
%>
```

jsp:setProperty and jsp:getProperty action tags

The setProperty and getProperty action tags are used for developing web application with Java Bean. In web devlopment, bean class is mostly used because it is a reusable software component that represents data.The jsp:setProperty action tag sets a property value or values in a bean using the setter method.

Syntax of jsp:setProperty action tag

```
<jsp:setProperty name="instanceOfBean" property= "*"  |
property="propertyName" param="parameterName" |
property="propertyName" value="{ string | <%= expression %>}"
/>
```

Example of jsp:setProperty action tag

```
<jsp:setProperty name="bean" property="*" />
```

jsp:getProperty action tag

The jsp:getProperty action tag returns the value of the property.

Syntax of jsp:getProperty action tag

```
<jsp:getProperty name="instanceOfBean" property="propertyName" />
```

Example of jsp:getProperty action tag

```
<jsp:getProperty name="obj" property="name" />
```

JSP Implicit Objects

There are 9 jsp implicit objects. These objects are created by the web container that are available to all the jsp pages. The available implicit objects are out, request, config, session, application etc.

List of the 9 implicit objects

Object	Type
Out	JspWriter
Request	HttpServletRequest
response	HttpServletResponse
Config	ServletConfig
application	ServletContext
Session	HttpSession
pageContext	PageContext
Page	Object
exception	Throwable

JSP out implicit object

For writing any data to the buffer, JSP provides an implicit object named out. It is the object of JspWriter.

```
PrintWriter out=response.getWriter();
```

JSP request implicit object

The JSP request is an implicit object of type HttpServletRequest i.e. created for each jsp request by the web container. It can be used to get request information such as parameter, header information, remote address, server name, server port, content type, character encoding etc. It can also be used to set, get and remove attributes from the jsp request scope.

JSP response implicit object

In JSP, response is an implicit object of type HttpServletResponse. The instance of HttpServletResponse is created by the web container for each jsp request. It can be used to add or manipulate response such as redirect response to another resource, send error etc.

pageContext implicit object

In JSP, pageContext is an implicit object of type PageContext class.The pageContext object can be used to set,get or remove attribute from one of the following scopes are page, request, session, application.

In JSP, page scope is the default scope.

session implicit object

In JSP, session is an implicit object of type HttpSession.The Java developer can use this object to set,get or remove attribute or to get session information.

JSP application implicit object

In JSP, application is an implicit object of type ServletContext. The instance of ServletContext is created only once by the web container when application or project is deployed on the server. This object can be used to get initialization parameter from configuaration file (web.xml). It can also be used to get, set or remove attribute from the application scope. This initialization parameter can be used by all jsp pages.

JSP config implicit object

In JSP, config is an implicit object of type ServletConfig. This object can be used to get initialization parameter for a particular JSP page. The config object is created by the web container for each jsp page. Generally, it is used to get initialization parameter from the web.xml file.

page implicit object:

In JSP, page is an implicit object of type Object class.This object is assigned to the reference of auto generated servlet class. It is written as:

Object page=this;

For using this object it must be cast to Servlet type. For example:

<% (HttpServlet)page.log("message"); %>

Since, it is of type Object it is less used because you can use this object directly in jsp. For example:

<% this.log("message"); %>

exception implicit object

In JSP, exception is an implicit object of type java.lang.Throwable class. This object can be

used to print the exception. But it can only be used in error pages.It is better to learn it after page directive.

UNIT V
Web Programming

Client Side Programming

Client-side programming is the name for all of the programs which are run on the Client. It has mostly to do with the user interface with which the user interacts. In web development, it is basically the browser in the user's machine that runs the code, and it is mainly done in JavaScript, Flash, etc.

- Make interactive webpages.
- Make stuff happen dynamically on the web page
- Interact with temporary storage as well as local storage (cookies etc.)
- Send requests to the server, and retrieve data from it, provide a remote service for client-side applications such as software registration, content delivery, remote multi-player gaming, etc.

Example languages

- JavaScript (primarily)
- HTML, CSS

Any language running on a client device that interacts with a remote service is a client-side language. However, HTML and CSS are not really programming languages. They are markup syntaxes by which the Client renders the page for a User.

Some of the main tasks of Client-side programming include,

- Validating input :Validation must be done in the server. A redundant validation in the client could be used to avoid server calls when speed is very critical.
- Animation & graphics
- Manipulating UI elements
- Applying styles, etc.

Form Design Using HTML

HTML is an acronym which stands for Hyper Text Markup Language.

Hyper Text

Hyper Text simply means "Text within Text". A text has a link within it, is a hypertext. Every time when you click on a word which brings you to a new webpage, you have clicked on a hypertext.

Markup language

A markup language is a programming language that is used make text more interactive and dynamic. It can turn a text into images, tables, links etc. An HTML document is made of many HTML tags and each HTML tag contains different content.

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

<body>
<h1>Write Your First Heading</h1>
Write Your First Paragraph.
</body>
</html>

Description of HTML

DOCTYPE: It defines the document type.

html : Text between html tag describes the web document.

body : Text between body tag describes the body content of the page that is visible to the end user.

h1 : Text between h1 tag describes the heading of the webpage.

p : Text between p tag describes the paragraph of the webpage.

Features of HTML

1) It is a very easy and simple language. It can be easily understood and modified.
2) It is very easy to make effective presentation with HTML because it has a lot of formatting tags.
3) It is a markup language so it provides a flexible way to design web pages along with the text.
4) It facilitates programmers to add link on the web pages (by html anchor tag) , so it enhances the interest of browsing of the user.
5) It is platform-independent because it can be displayed on any platform like Windows, Linux and Macintosh etc.
6) It facilitates the programmer to add Graphics, Videos, and Sound to the web pages which makes it more attractive and interactive

HTML Tags

HTML tags contain three main parts: opening tag, content and closing tag. But some HTML tags are unclosed tags. When a web browser reads an HTML document, browser reads it from top to bottom and left to right. HTML tags are used to create HTML documents and render their properties. Each HTML tags have different properties.

Syntax

<tag> content </tag>

HTML Tag

HTML Tags are always written in lowercase letters. The basic HTML tags are given below:

 Paragraph Tag
<h2> Heading Tag </h2>
 Bold Tag
 Italic Tag
<u> Underline Tag</u>

Unclosed HTML Tags

Some HTML tags are not closed, for example br and hr.

 Tag: br stands for break line, it breaks the line of the code.

<hr> Tag: hr stands for Horizontal Rule. This tag is used to put a line across the webpage.

HTML Meta Tags

DOCTYPE, title, link, meta and style

HTML Text Tags

<p>, <h1>, <h2>, <h3>, <h4>, <h5>, <h6>, , , <abbr>, <acronym>, <address>, <bdo>, <blockquote>, <cite>, <q>, <code>, <ins>, <del>, <dfn>, <kbd>, <pre>, <samp>, <var> and

HTML Link Tags

<a> and <base>

HTML Image and Object Tags

<img>, <area>, <map>, <param> and <object>

HTML List Tags

, <ol>, <li>, <dl>, <dt> and <dd>

HTML Table Tags

table, tr, td, th, tbody, thead, tfoot, col, colgroup and caption

HTML Form Tags

form, input, textarea, select, option, optgroup, button, label, fieldset and legend

HTML Scripting Tags

script and noscript

HTML Form

An HTML form is a section of a document which contains controls such as text fields, password fields, checkboxes, radio buttons, submit button, menus etc. An HTML form facilitates the user to enter data that is to be sent to the server for processing.

HTML Form Syntax

```
<form action="server url" method="get|post">
  //input controls e.g. textfield, textarea, radiobutton, button
</form>
```

HTML Form Tags

Tag	Description
<form>	It defines an HTML form to enter inputs by the used side.
<input>	It defines an input control.
<textarea>	It defines a multi-line input control.
<label>	It defines a label for an input element.
<fieldset>	It groups the related element in a form.
<legend>	It defines a caption for a <fieldset> element.
<select>	It defines a drop-down list.

<optgroup>	It defines a group of related options in a drop-down list.
<option>	It defines an option in a drop-down list.
<button>	It defines a clickable button.

JavaScript Tutorial

JavaScript is designed for beginners and professionals both. JavaScript is used to create client-side dynamic pages. JavaScript is an object-based scripting language which is lightweight and cross-platform.

JavaScript is not a compiled language, but it is a translated language. The JavaScript Translator (embedded in the browser) is responsible for translating the JavaScript code for the web browser.

Application of JavaScript

JavaScript is used to create interactive websites. It is mainly used for:

- Client-side validation.
- Dynamic drop-down menus.
- Displaying date and time.
- Displaying pop-up windows and dialog boxes (like an alert dialog box, confirm dialog box and prompt dialog box).
- Displaying clocks etc.

JavaScript Example

```
<script>
document.write("Hello JavaScript by JavaScript");
</script>
```

JavaScript

1. JavaScript Example
2. Within body tag
3. Within head tag

Javascript example is easy to code. JavaScript provides 3 places to put the JavaScript code within body tag, within head tag and external JavaScript file.

```
<script type="text/javascript">
document.write("JavaScript is a simple language for javatpoint learners");
</script>
```

Script Tags

The text/javascript is the content type that provides information to the browser about the data. The document.write() function is used to display dynamic content through JavaScript.

3 Places to put JavaScript code

1. Between the body tag of html
2. Between the head tag of html
3. In .js file (external javaScript)

JavaScript Example : code between the body tag

In the above example, we have displayed the dynamic content using JavaScript. Let's see the simple example of JavaScript that displays alert dialog box.

```
<script type="text/javascript">
 alert("Hello Javatpoint");
</script>
```

JavaScript Example : code between the head tag

The same example of displaying alert dialog box of JavaScript that is contained inside the head tag. In this example, we are creating a function msg(). To create function in JavaScript, you need to write function with function_name. To call function, you need to work on event. Here we are using onclick event to call msg() function.

```
<html>
<head>
<script type="text/javascript">
function msg(){
 alert("Hello Javatpoint");
}
</script>
</head>
<body>
<p>Welcome to JavaScript</p>
<form>
<input type="button" value="click" onclick="msg()"/>
</form>
</body>
</html>
```

External JavaScript

We can create external JavaScript file and embed it in many html page. It provides code re usability because single JavaScript file can be used in several html pages. An external JavaScript file must be saved by .js extension. It is recommended to embed all JavaScript files into a single file. It increases the speed of the webpage.

To create an external JavaScript file that prints Hello Javatpoint in a alert dialog box.

message.js

```
function msg(){
 alert("Hello Javatpoint");
}
```

To include the JavaScript file into html page. It calls the JavaScript function on button click.

index.html

```
<html>
```

```
<head>
<script type="text/javascript" src="message.js"></script>
</head>
<body>
<p>Welcome to JavaScript</p>
<form>
<input type="button" value="click" onclick="msg()"/>
</form>
</body>
</html>
```

XML

XML tutorial is designed for beginners and professionals. Our XML tutorial provides a detailed knowledge of XML technology like what is xml, features of xml, xml example, xml related technologies, creating xml structure by DTD, creating xml structure by schema (XSD), difference between DTD and schema.

XML Validation

XML file can be validated by 2 ways:

1. against DTD
2. against XSD

DTD (Document Type Definition) and XSD (XML Schema Definition) are used to define XML structure.

XML DTD

You will learn about DTD file, creating xml with DTD, using CSS file, CDATA vs PCDATA and difference between DTD and XML schema.

employee.xml

```
<?xml version="1.0"?>
<!DOCTYPE employee SYSTEM "employee.dtd">
<employee>
  <firstname>vimal</firstname>
  <lastname>jaiswal</lastname>
  <email>vimal@javatpoint.com</email>
</employee>
```

XML Schema

We will provide a detail description of schema file, XML schema validation, XML schema data types and XML parsers.

```
<?xml version="1.0"?>
<employee
xmlns="http://www.javatpoint.com"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.javatpoint.com employee.xsd">
```

```
  <firstname>vimal</firstname>
  <lastname>jaiswal</lastname>
  <email>vimal@javatpoint.com</email>
</employee>
```

Ajax

AJAX tutorial covers concepts and examples of AJAX technology for beginners and professionals. AJAX is an acronym for Asynchronous JavaScript and XML. It is a group of inter-related technologies like JavaScript, DOM, XML, HTML/XHTML, CSS, XMLHttpRequest etc. AJAX allows you to send and receive data asynchronously without reloading the web page. So it is fast. AJAX allows you to send only important information to the server not the entire page. So only valuable data from the client side is routed to the server side. It makes your application interactive and faster.

Uses

There are too many web applications running on the web that are using ajax technology like gmail, facebook, twitter, google map, youtube etc.

AJAX Technologies

Ajax is not a technology but group of inter-related technologies. AJAX technologies includes,

- HTML/XHTML and CSS
- DOM
- XML or JSON
- XMLHttpRequest
- JavaScript

HTML/XHTML and CSS

These technologies are used for displaying content and style. It is mainly used for presentation.

DOM

It is used for dynamic display and interaction with data.

XML or JSON

For carrying data to and from server. JSON (Javascript Object Notation) is like XML but short and faster than XML.

XMLHttpRequest

For asynchronous communication between client and server.

JavaScript

It is used to bring above technologies together. Independently, it is used mainly for client-side validation.

Ajax Example

To create ajax example, you need to use any server-side language e.g. Servlet, JSP, PHP, ASP.Net etc. Here we are using JSP for generating the server-side code. In this example, we are simply printing the table of the given number.

Steps to create ajax example with jsp

You need to follow following steps:

1. load the org.json.jar file
2. create input page to receive any text or number
3. create server side page to process the request
4. provide entry in web.xml file

Load the org.json.jar file

We have included the org.json.jar file inside the WEB-INF/lib directory.

Create input page to receive any text or number

We have created a form that gets input from the user. When user clicks on the showTable button, sendInfo() function is called. We have written all the ajax code inside this function.
We have called the getInfo() function whenever ready state changes. It writes the returned data in the web page dynamically by the help of innerHTML property.

table1.html

```
<html>
<head>
<script>
var request;
function sendInfo()
{
var v=document.vinform.t1.value;
var url="index.jsp?val="+v;
if(window.XMLHttpRequest){
request=new XMLHttpRequest();
}
else if(window.ActiveXObject){
request=new ActiveXObject("Microsoft.XMLHTTP");
}
  try
{
request.onreadystatechange=getInfo;
request.open("GET",url,true);
request.send();
}
catch(e)
{
alert("Unable to connect to server");
}
}
  function getInfo(){
```

```
if(request.readyState==4){
var val=request.responseText;
document.getElementById('amit').innerHTML=val;
}
}
</script>
</head>
<body>
   <marquee><h1>This is an example of ajax</h1></marquee>
<form name="vinform">
<input type="text" name="t1">
<input type="button" value="ShowTable" onClick="sendInfo()">
</form>
<span id="amit"> </span>
</body>
</html>
```

Create server side page to process the request

We printing the table of given number.

index.jsp

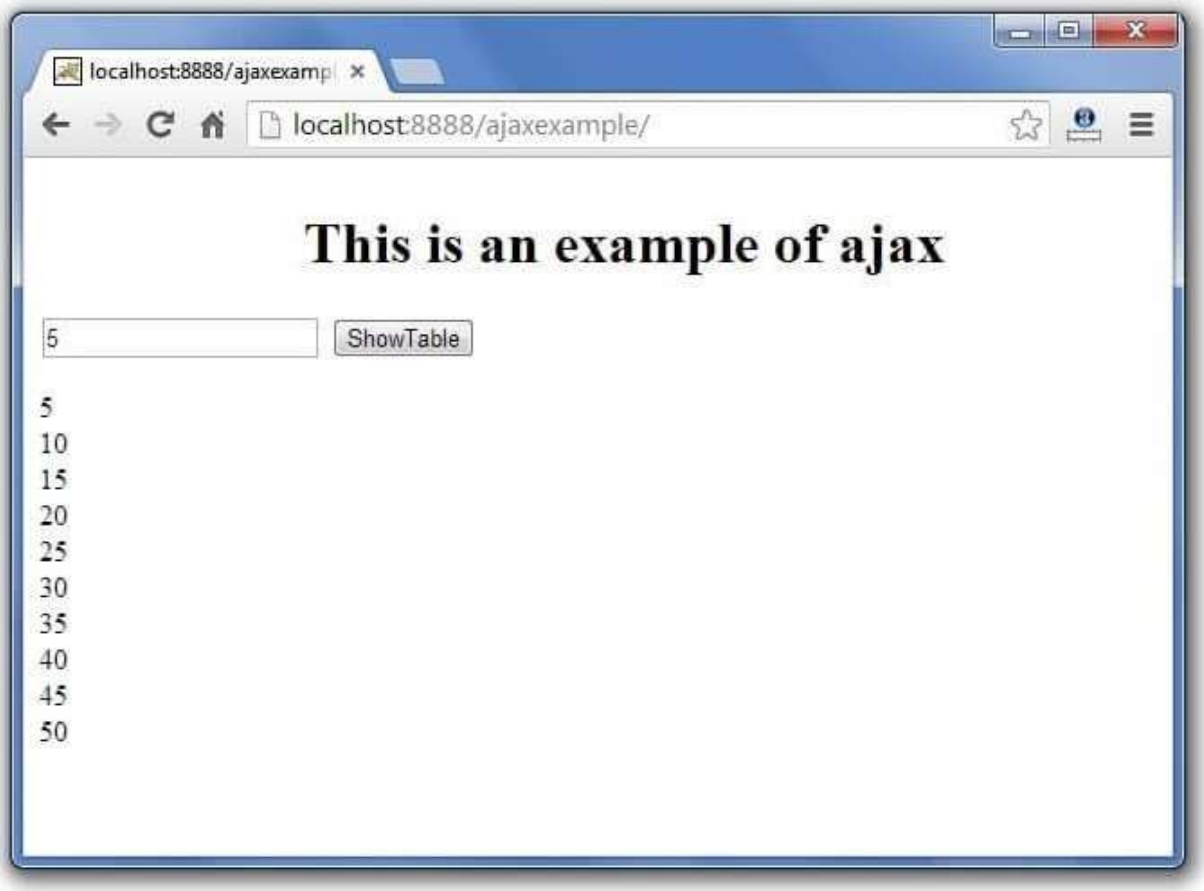
```
<%
int n=Integer.parseInt(request.getParameter("val"));
for(int i=1;i<=10;i++)
out.print(i*n+"<br>");
%>
```

web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app version="2.5" 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_2_5.xsd">

   <session-config>
      <session-timeout>
         30
      </session-timeout>
   </session-config>
   <welcome-file-list>
      <welcome-file>table1.html</welcome-file>
      </welcome-file-list>
   </web-app>
```

Output



Server Side Programming

Server: Web vs. Application

Server is a device or a computer program that accepts and responds to the request made by other program, known as client. It is used to manage the network resources and for running the program or software that provides services.

There are two types of servers:

1. Web Server
2. Application Server

Web Server

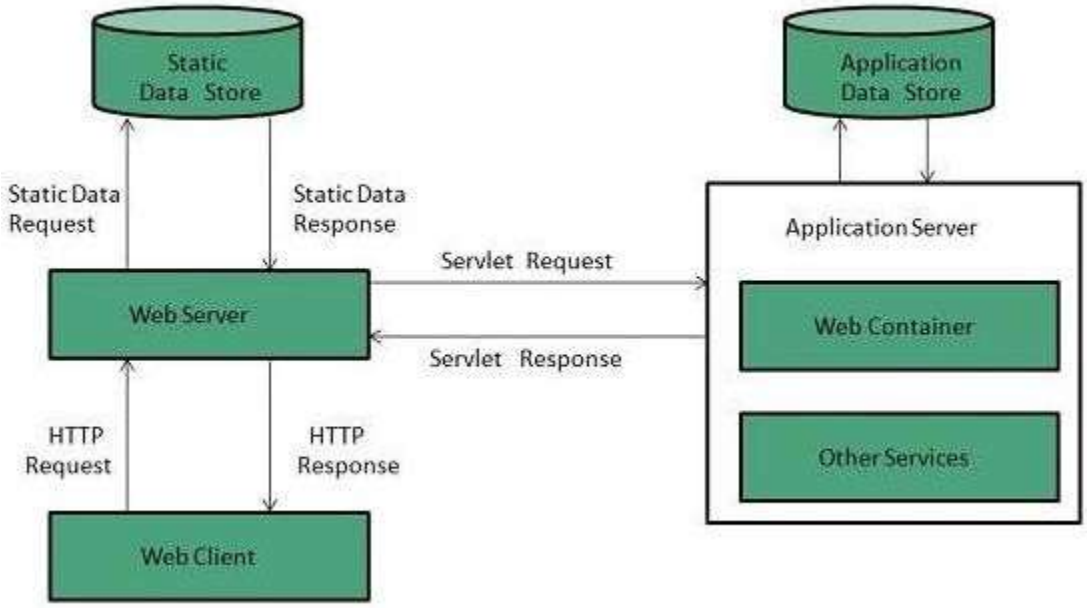
Web server contains only web or servlet container. It can be used for servlet, jsp, struts, jsf etc. It can't be used for EJB. It is a computer where the web content can be stored. In general web server can be used to host the web sites but there also used some other web servers also such as FTP, email, storage, gaming etc. Examples of Web Servers are: Apache Tomcat and Resin.

Web Server Working

It can respond to the client request in either of the following two possible ways:

- Generating response by using the script and communicating with database.
- Sending file to the client associated with the requested URL.

The block diagram representation of Web Server is shown below:



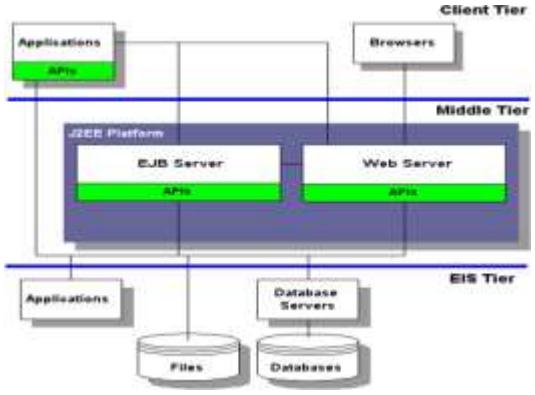
Important of Web Server

- If the requested web page at the client side is not found, then web server will sends the HTTP response: Error 404 Not found.
- When the web server searching the requested page if requested page is found then it will send to the client with an HTTP response.
- If the client requests some other resources then web server will contact to application server and data is store for constructing the HTTP response.

Application Server

Application server contains Web and EJB containers. It can be used for servlet, jsp, struts, jsf, ejb etc. It is a component based product that lies in the middle-tier of a server centric architecture. It provides the middleware services for state maintenance and security, along with persistence and data access. It is a type of server designed to install, operate and host associated services and applications for the IT services, end users and organizations.

The block diagram representation of Application Server is shown below:



The Example of Application Servers are:

1. JBoss: Open-source server from JBoss community.
2. Glassfish: Provided by Sun Microsystem. Now acquired by Oracle.
3. Weblogic: Provided by Oracle. It more secured.
4. Websphere: Provided by IBM.

***** THE END *****