

# JAVA DATABASE CONNECTIVITY (JDBC)

## Introduction

It is a pure JAVA API used to execute SQL Statements. This provides set of classes and interfaces which can be used for developing database applications.

## Java and Database

As java has gained favour as the language of choice for internet and intranet applications, has also gained importance for the Java database connectivity driver(JDBC). The JDBC uses a simple class hierarchy for database objects. The classes are contained in the `java.sql` package

## Models of Database Access

1. Two-tier Architecture
2. Three-Tier Architecture

### Two-Tier Architecture

Client-server computing involves two or more computers sharing tasks related to a complete application. Ideally, each computer is performing logic appropriate to its design and stated function.

The most widely used form of client-server implementation is a two-tier client-server. This involves a front-end client application communicating with a back-end database engine running on a separate computer. Client programs send SQL statements to the database server. The server returns the appropriate results, and the client is responsible for handling the data.

The basic two-tier client-server model is used for applications that can run with many popular databases including Oracle, Sybase, and Informix.

### Thick Client

A major performance penalty is paid in two-tier client-server. The client software ends up larger and more complex because most of the logic is

handled there. The use of server-side logic is limited to database operations. The client here is referred to as a thick client.

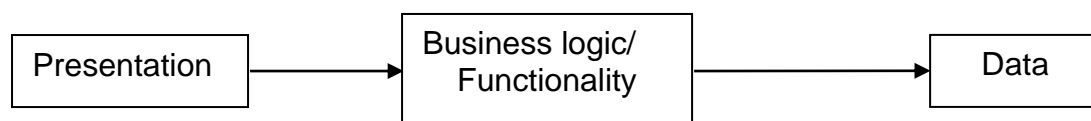
Thick clients tend to produce frequent network traffic for remote database access. This works well for intranet and local area networks (LAN)-based network topologies but produces a large footprint on the desktop in terms of disk and memory requirements. Also, not all back-end database servers are the same in terms of server logic offered, and all of them have their own API sets that programmers must use to optimize and scale performance.

Three-tier client-server, which is described next, takes care of scalability, performance, and logic partitioning in a more efficient manner.

### The Three-Tier Architecture

Three-tier is the most advanced type of client-server software architecture. A three-tier client-server demands a much steeper development curve initially, especially when you to support a number of different platforms and network environments. The payback comes in the form of reduced network traffic, excellent Internet and intranet performance, and more control over system expansion and growth.

### Three-Tier Client-Server Definition



### Front End

**The three components or tiers of a three-tier client-server environment are presentation, business logic or functionality, and data. They are separated such that the software for any one of the tiers can be replaced by a different implementation without affecting the other tiers.**

For example, if you wanted to replace a character-oriented screen (or screens) with a GUI (the presentation tier), you would write the GUI using an established API or interface to access the same functionality programs in the character-oriented screens.

## Middle-Ware

The business logic offers functionality in terms of defining all of the business rules through which the data can be manipulated. Changes to business policies can affect this layer without having any impact on the actual databases.

## Back-End

The third tier or data tier includes existing systems, applications, and data that has been encapsulated to take advantage of this architecture with minimal transitional programming effort.

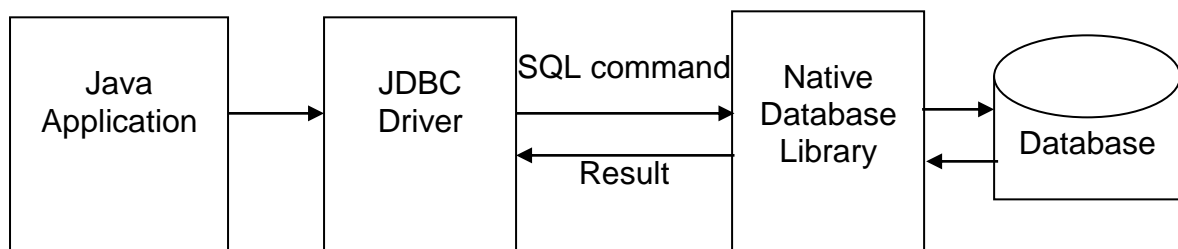
## JDBC Driver Types

There are four types in which the drivers are categorized

1. **Native-API partly-Java driver**
2. **JDBC- Net pure Java driver**
3. **Native-protocol pure Java driver**
4. **JDBC-ODBC bride plus ODBC driver**

### 1. Native-API partly-Java driver

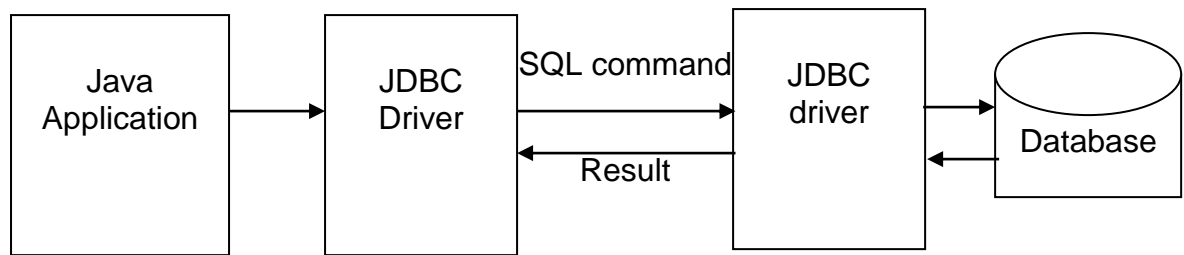
This kind of driver converts JDBC calls on the API for Oracle, SQL, Server or other RDBMS. Note that, it also some what the bridge driver, this style of driver also requires some driver to be loaded on each client where we will be using this code. In practice this type of driver is rarely used.



### 2. JDBC-Net pure java driver

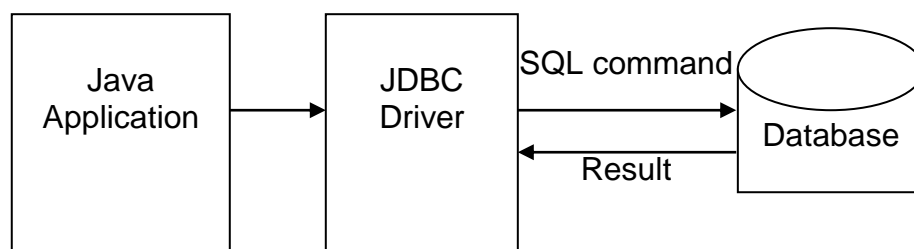
The driver translates JDBC call into a DBMS-independent net protocol, which is then translated, to a DBMS protocol by a server or the middle tier of the architecture normally connects its pure java clients to many different databases. In this situation there is more load on the

middle in terms of security and the processing so they must handle the additional requirements for security, access through firewalls.



### 3. Native-protocol pure java driver

This kind of driver converts JDBS calls into the Network protocol used by DBMS directly. This allows a direct call from the client machine to the DBMS server and is practical solution for



### 4. JDBC-ODBC Bridge plus ODBC driver

The java product provides JDBC access via ODBC drivers. In this particular type of drivers there are certain limitation. When this driver is used it is necessary to create ODBC entries on every machine where it is being used by this application. To be very precise there is a need to create the ODBC DSN (data source Name) on every machine. So this kind of driver is most appropriate on a corporate network where client is not a major problem, or even for application server code written in java in a three-tier architecture.

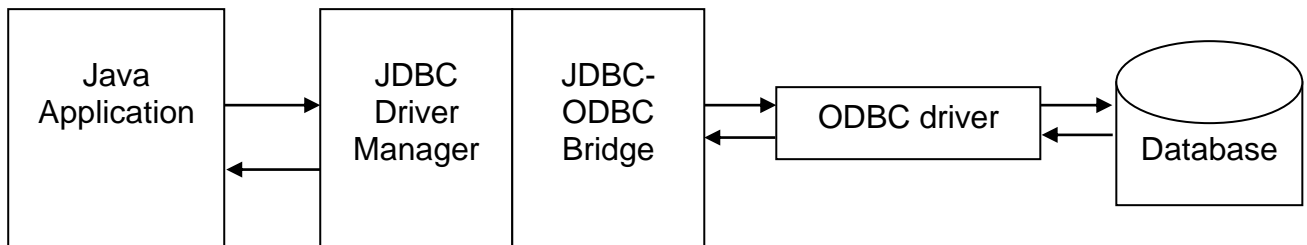
### JDBC API

JDBC API stands for Java Data Base Connectivity Application Programming Interface. As its name implies, the JDBC API is a set of specifications that defines how a program written in Java can communicate and interact with a database. More specially, the JDBC API defines how an application opens a connection, communicates with a database, execute SQL statements, and retrieves query result.

## ODBC API

ODBC API is a set of library routines that enable your programs to access a variety of databases. All you need to do is to install a DBMS - special ODBC driver and write your program to interact with the specified ODBC driver.

## JDBC Application Architecture



### JDBC Driver Manager

The JDBC driver manager is the backbone of JDBC architecture. The function of the JDBC driver manager is to connect a Java application to the appropriate driver specified in your program.

### JDBC-ODBC Bridge

As a part of JDBC, sun Microsystems provides a driver to access ODBC data sources from JDBC. This driver is called the JDBC-ODBC Bridge.

### Configuration of ODBC driver

1. Click on the Start button.
2. Select the Settings option from the start menu.
3. Select Control Panel option from the settings submenu.
4. Double click on the Administrative tools
5. Click on Data Source (ODBC) icon.
6. Click on the Add button.
7. Select the ODBC driver from the list, which is required.
8. Click on the finish button.
9. Type the Data Source Name in the Data Source Name text box.
10. Click on the Ok button.
11. Again click on the Ok button.

## The Seven basic steps to JDBC

There are seven basic steps to access a database. Each step has distinct purpose and is necessary to complete a database transaction.

### Step 1:

#### Import the java.sql package

The JDBC API is a set of classes and interfaces. The package name for these classes and interfaces is java.sql and is imported in the first line of the program.

```
import java.sql.*;
```

### Step 2:

#### Loading the Driver

The driver can be loaded in any one of the several different ways. The most common and easiest method to load the driver using the `Class.forName()` method. The `Class.forName()` method takes the complete package name of the driver as its argument.

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
```

### Step 3:

#### Establish the connection

Once a driver is loaded, the standard method of establishing a connection to the database is to call `DriverManager.getConnection()` method. The `getConnection()` method takes atleast three arguments. The first is a string representing the URL of the database followed by a set of login properties, such as the user name and password.

```
Connection conn=DriverManager.getConnection("jdbc:odbc:
                                         datasourcename","username","password"
                                         );
```

**Step 4:****Create a Statement**

Once you have established a connection to the database, you need to create a statement object from which to execute your query.

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

**Step 5:****Execute the SQL statement**

Finally, after all the preparations we are ready to actually execute an SQL statement. The method used to execute a simple query is `executeQuery()`.

```
ResultSet rs=stmt.executeQuery("SQL Query");
```

The `executeQuery()` method takes an SQL query string as an argument and returns the results of the query as a `ResultSet` object.

**Step 6:****Retrieve the record(s) from the table**

After you have executed your SQL statement, your next task is to retrieve the results(records). Results are stored in a `ResultSet` object. To retrieve the data out of the `ResultSet` and into the Java variable, you need to use one of the result set methods `getString()`. The `getString()` method retrieves the data from the `ResultSet` and converts it to a Java type. The `getString()` method takes the column name as an argument and returns the value found in the current row of that column. By placing the `getString()` method in a loop and incrementing the row pointer using the `next()` each time, you can step through the entire result set.

**Step 7:****Close the connection and Statements**

The final step in any database application should close the connection and any open statement.

```
stmt.close();  
conn.close();
```

## Executing SQL Statements using `executeUpdate()` Method

The `executeUpdate()` method used to create, alter and drop tables and also insert, update, delete records of the table

### Adding Records

#### Syntax

```
stmt.executeUpdate("insert into <table name> values( )");  
stmt.execute("commit");
```

#### Program

```
//program to add record into the table  
  
import java.sql.*;  
class insert  
{  
    public static void main(String args[])  
    {  
        try  
        {  
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");  
            Connection  
conn=DriverManager.getConnection("jdbc:odbc:link","scott"  
,"tiger");  
            Statement stmt=conn.createStatement();  
            stmt.executeUpdate("insert          into          student  
values(1,'anand',60,70,80,'12-feb-2010')");  
            System.out.println("Record has Inserted");  
  
            stmt.close();  
            conn.close();  
        }  
        catch(Exception e)  
        {  
            System.out.println("Sql Error "+e);  
        }  
    }  
}
```



**Modify Records****Syntax**

```
stmt.executeUpdate("update <table name> set <expr> where <condition>");
```

**Delete Records****Syntax**

```
stmt.executeUpdate("delete from <table name> where <condition>");
```

**PreparedStatement:****Insert or Select records using PreparedStatement Interface****The PreparedStatement Interface and prepareStatement() method**

The PreparedStatement object allows you to execute parameterized queries. PreparedStatement object is created using the prepareStatement() method of the Connection object. The prepareStatement() of the connection object takes an SQL statement as a parameter. The SQL statement can contain placeholders that can be replaced by INPUT parameters at runtime. The "?" symbol is a placeholder that can be replaced by the INPUT parameters at run time.

**Passing INPUT as parameter:**

Before executing a PreparedStatement object, you must set the value of each ? parameter. This is done by the calling of setXXX() method, where XXX is the data type of the parameter.

**Example**

```
stmt=con.prepareStatement("insert into student
                                values(?,?,?, ?, ?, ?)");
st.setInt(1,reg);
st.setString(2,name);
.....;
```

## Program

```
//program to insert records into the table

import java.sql.*;
import java.awt.event.*;
import javax.swing.*;

class InsertRecord extends JFrame implements
ActionListener
{
    JTextField t1,t2,t3,t4,t5,t6;
    Connection con;
    PreparedStatement st;

    InsertRecord ()
    {
        super("Insert Record ");
        try
        {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

            con =
            DriverManager.getConnection("jdbc:odbc:aravind","scott","
            tiger");
        }
        catch(Exception e)
        {
            System.out.println("Exception in Loading Driver .... ");
        }
        JPanel jp=new JPanel();
        JLabel l1= new JLabel("Register No ");
        JLabel l2= new JLabel("Name ");
        JLabel l3= new JLabel("Tamil ");
        JLabel l4= new JLabel("English ");
        JLabel l5= new JLabel("Maths ");
        JLabel l6= new JLabel("Date of Join ");
        t1=new JTextField(10);
        t2=new JTextField(10);
        t3=new JTextField(10);
        t4=new JTextField(10);
        t5=new JTextField(10);
        t6=new JTextField(10);
        JButton b = new JButton("Insert");
        JButton c = new JButton("Close");
        jp.add(l1);
        jp.add(t1);
        jp.add(l2);
        jp.add(t2);
        jp.add(l3);
        jp.add(t3);
```

```

jp.add(l4);
jp.add(t4);
jp.add(l5);
jp.add(t5);
jp.add(l6);
jp.add(t6);
jp.add(b);
jp.add(c);
b.addActionListener(this);
c.addActionListener(this);
getContentPane().add(jp);
}
public void actionPerformed(ActionEvent ae)
{
String s=ae.getActionCommand();
if(s.equals("Insert"))
{
String str,name,doj;
int reg, m1,m2,m3;
str=t1.getText();
reg=Integer.parseInt(str);
name=t2.getText();
str=t3.getText();
m1=Integer.parseInt(str);
str=t4.getText();
m2=Integer.parseInt(str);
str=t5.getText();
m3=Integer.parseInt(str);
doj=t6.getText();

try
{
st=con.prepareStatement("insert          into          stud
values(?,?,?, ?, ?, ?)");
st.setInt(1,reg);
st.setString(2,name);
st.setInt(3,m1);
st.setInt(4,m2);
st.setInt(5,m3);
st.setString(6,doj);
st.executeUpdate();
JOptionPane.showMessageDialog(null,"Record      has      been
Inserted","Insert Record",JOptionPane.PLAIN_MESSAGE);

st.close();
con.close();
t1.setText(" ");
t2.setText(" ");
t3.setText(" ");
t4.setText(" ");
t5.setText(" ");

```

```
t6.setText(" ");

t1.requestFocus();
}

catch(Exception e)
{
System.out.println("Exception on Record Insert .... ");
}
}

else if(s.equals("Close"))
{
System.exit(0);
}
}
}

class InsertDemo
{
public static void main(String args[])
{
InsertRecord ir = new InsertRecord ();
ir.setSize(250,300);
ir.setVisible(true);
ir.setLocation(200,200);
}
}
```

### Retrieve Records from the Table

The data can be retrieved from the table using the SELECT command. A query is made to the table using `executeQuery()` method. The method retrieved the data from the table and stored in a object of the `ResultSet` class.

#### **Example**

```
ResultSet rs =stmt.executeQuery("select * from student");
```

**Program**

```

import java.sql.*;
import java.awt.event.*;
import javax.swing.*;

class ViewRecord extends JFrame implements ActionListener
{
    JTextField t1,t2,t3,t4,t5,t6;
    ViewRecord ()
    {
        super("View Record ");
        JPanel jp=new JPanel();
        JLabel l1= new JLabel("Register No  ");
        JLabel l2= new JLabel("Name          ");
        JLabel l3= new JLabel("Tamil          ");
        JLabel l4= new JLabel("English       ");
        JLabel l5= new JLabel("Maths         ");
        JLabel l6= new JLabel("Date of Join ");
        t1=new JTextField(10);
        t2=new JTextField(10);
        t3=new JTextField(10);
        t4=new JTextField(10);
        t5=new JTextField(10);
        t6=new JTextField(10);
        JButton b = new JButton("View Record");
        jp.add(l1);
        jp.add(t1);
        jp.add(l2);
        jp.add(t2);
        jp.add(l3);
        jp.add(t3);
        jp.add(l4);
        jp.add(t4);
        jp.add(l5);
        jp.add(t5);
        jp.add(l6);
        jp.add(t6);
        jp.add(b);
        b.addActionListener(this);
        getContentPane().add(jp);
    }
    public void actionPerformed(ActionEvent ae)
    {
        String str,name,doj;
        int reg, m1,m2,m3;

        str=JOptionPane.showInputDialog("Enter your Register No.
        ");
        reg=Integer.parseInt(str);
    }
}

```

```
try
{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

Connection con =
DriverManager.getConnection("jdbc:odbc:anand","scott","ti
ger");
System.out.println("Preparing statement!...");
PreparedStatement st=con.prepareStatement("select * from
student where rno=?");
st.setInt(1,reg);
ResultSet rs=st.executeQuery();
if(rs.next())
{
t1.setText(rs.getString("rno")+" ");
t2.setText(rs.getString("name")+" ");
t3.setText(rs.getString("m1")+" ");
t4.setText(rs.getString("m2")+" ");
t5.setText(rs.getString("m3")+" ");
t6.setText(rs.getString("doj")+" ");
}
else
{
JOptionPane.showMessageDialog(null,"Invalid      Record
","Record not found!... ",JOptionPane.ERROR_MESSAGE);
}
st.close();
con.close();
}
catch(Exception e)
{
System.out.println("Error "+e);
}
}
}
class ViewDemo
{

public static void main(String args[])
{
ViewRecord vr = new ViewRecord ();
vr.setSize(200,200);
vr.setVisible(true);
}
}
```

### Scrollable result sets

To navigate JDBC result sets we are having the next() method, which enables us to move forward only. Another option is to create a

scrollable result set, so the cursor can navigate the result set both backward and forward. A two argument version of the createStatement() method exists. The first argument specifies the type of scrolling (TYPE\_FORWARD\_ONLY, TYPE\_SCROLL\_INSENSITIVE, or TYPE\_SCROLL\_SENSITIVE), and the second enables you to make the result set either read only or updatable (CONCUR\_READ\_ONLY or CONCUR\_UPDATABLE)

### Statement

```
stmt=conn.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_READ_ONLY);
```

We can use the methods like rs.next(), rs.previous(), rs.first(), and rs.last()

### Program

```
import java.sql.*;
import java.awt.event.*;
import javax.swing.*;

class RecordScroll extends JFrame implements
ActionListener
{
    JTextField t1,t2,t3,t4,t5;
    RecordScroll ()
    {
        super("Scrollable Dialog Box");
        JPanel jp=new JPanel();
        JLabel l1=new JLabel("Register No :");
        JLabel l2=new JLabel("Name :");
        JLabel l3=new JLabel("Tamil Mark :");
        JLabel l4=new JLabel("English Mark:");
        JLabel l5=new JLabel("Maths Mark :");
        t1=new JTextField(10);
        t2=new JTextField(10);
        t3=new JTextField(10);
        t4=new JTextField(10);
        t5=new JTextField(10);
        JButton b1=new JButton("First");
        JButton b2=new JButton("Last");
        JButton b3=new JButton("Next");
        JButton b4=new JButton("Previous");
        jp.add(l1);
        jp.add(t1);
        jp.add(l2);
        jp.add(t2);
```

```

        jp.add(l3);
        jp.add(t3);
        jp.add(l4);
        jp.add(t4);
        jp.add(l5);
        jp.add(t5);
        jp.add(b1);
        jp.add(b2);
        jp.add(b3);
        jp.add(b4);
        b1.addActionListener(this);
        b2.addActionListener(this);
        b3.addActionListener(this);
        b4.addActionListener(this);
        getContentPane().add(jp);
    }
    public void actionPerformed(ActionEvent ae)
    {
        String str;
        str=ae.getActionCommand();
        try
        {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection
conn=DriverManager.getConnection("jdbc:odbc:link");
            Statement stmt=conn.createStatement();
            ResultSet      rs=stmt.executeQuery("select      *      from
student");
            if(str.equals("First"))
                if(rs.first())
            else if(str.equals("Last"))
                if(rs.last())
            else if(str.equals("Next"))
                if(rs.next())
            else if(str.equals("Previous"))
                if(rs.previous())
            t1.setText(rs.getString("Regno")+" ");
            t2.setText(rs.getString("Name")+" ");
            t3.setText(rs.getString("Tamil")+" ");
            t4.setText(rs.getString("English")+" ");
            t5.setText(rs.getString("Maths")+" ");
            stmt.close();
            conn.close();
        }
        catch(Exception e)
        {
        }
    }
}
class RecordScrollDemo
{

```



```
public static void main(String args[])
{
    RecordScroll rs=new RecordScroll();
    rs.setSize(300,300);
    rs.setVisible(true);
}}
```

## The ResultSetMetaData class

JDBC enables you to process a result set even if the database table columns are not specified in the SQL query. Imagine that you need to write a program that will accept any SQL select statement and display the retrieved data. The `java.sql.ResultSetMetaData` class can dynamically find out the structure of the underlying database table - how many columns it contains, and the types and names of the columns.

```
ResultSet rs=stmt.executeQuery("select * from <table name>");  
ResultSetMetaData rsMeta=rs.getMetaData();
```

We can use the following methods `rsMeta.getColumnCount()`, `rsMeta.getColumnName()`.

## Performing Batch Updates

Sometimes several database modifications have to be processed as a batch; in this case, if one of the updates fails the whole transaction has to be rolled back. A well-known definition states that the transaction is a logical unit of work. In batch updates database operations must be explicitly committed in case of success or rolled back in case of failure.

### **Program**

```
import java.sql.*;  
class BatchDemo  
{  
    public static void main(String args[])  
    {  
        try  
        {  
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");  
            Connection  
conn=DriverManager.getConnection("jdbc:odbc:link");  
            Statement stmt=conn.createStatement();  
            stmt.addBatch("insert          into          student  
values(101,'Viswa',67,78,54)");  
            stmt.addBatch("insert          into          student  
values(102,'Anand',78,89,77)");  
            stmt.executeBatch();  
            conn.commit();  
            stmt.addBatch("insert          into          student  
values(103,'Moorthy',87,55,97)");  
            conn.rollback();  
            stmt.executeBatch();  
        }  
    }  
}
```

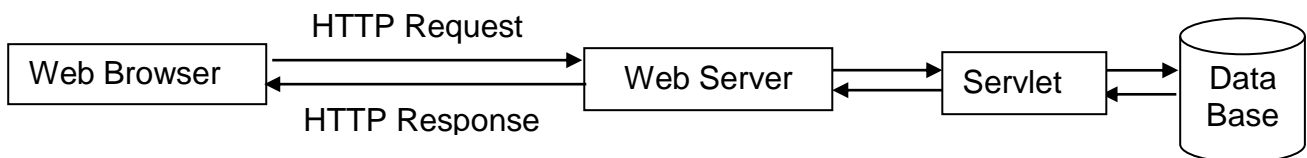
```
catch(Exception e)
{
    System.out.println("Error"+e);
}
}
```

# JAVA SERVLET

## Introduction

Servlets are server side programs. Applets function on the browser while servlets function on the server. Servlets are loaded and executed by a web server in the same manner that applets are loaded and executed by a web browser.

A servlet accepts a request from a client via the web server performs some task and returns the results.



### Steps illustrating the basic flow while using servlets:

1. The client (Web browser) makes a request via an HTTP.
2. The web server receives the request and sends it to the servlet.
3. The servlet will receive the HTTP request and perform task.
4. The servlet will return a response to the web server.
5. The web server will forward the response to the client.

### Example:

When you use an online store or bank, to search for some information on the Internet, your request is usually processed on the server side. If the user requests some information that should be retrieved programmatically from a database or any other resource, we will use Java Servlets that will accommodate the request and build an output HTML page dynamically, and pass that page over to the user with the help of the web server.

### Client Side

A `<FORM>` tag has important attributes such as `action` and `method`. The `action` attribute contains the Uniform Resource Locator (URL) to which the user's input should be sent. The `method` tells the browser how to send the data; this attribute is usually either **get** or **post**.

An Http URL contains the following parts:

`http://[host]:[port][request path]`

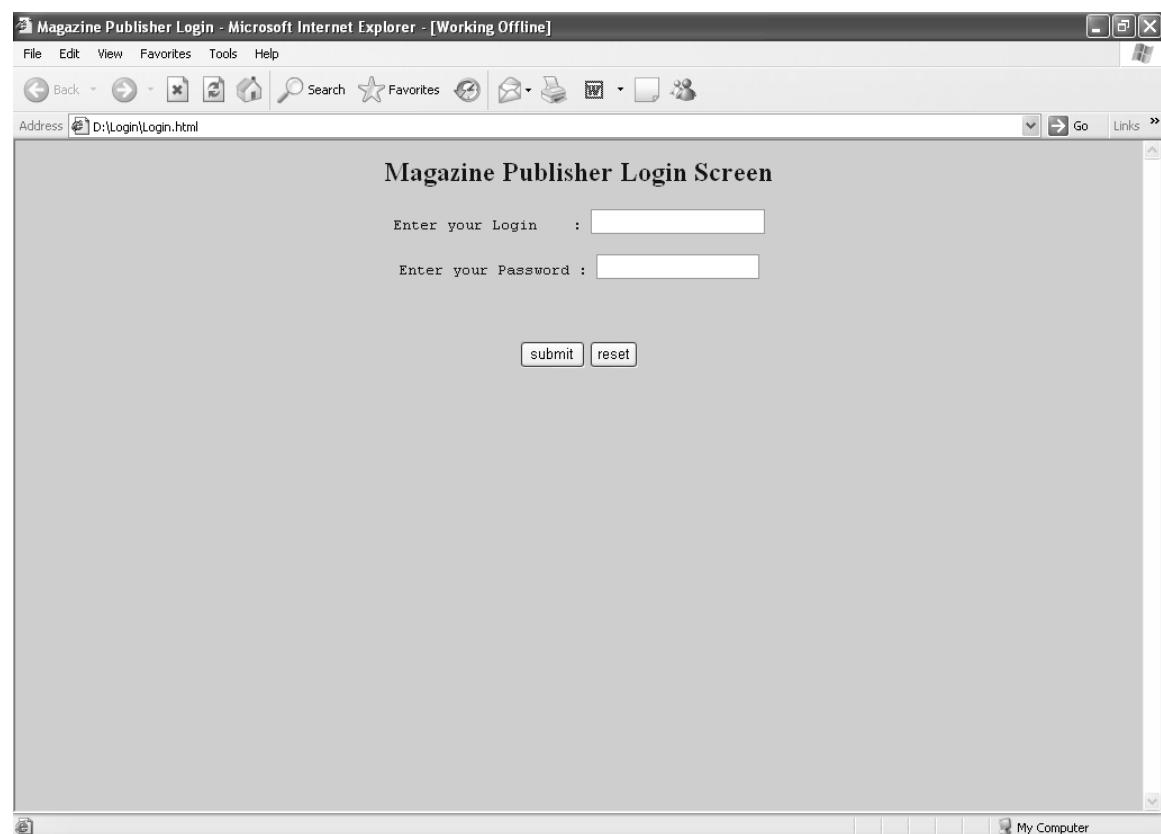
The request path is further composed of the following elements.

**Context path:** A concatenation of the forward slash (/) with the context root of the servlet's web application.

**Server path:** The path section that corresponds to the component alias that activated this request.

## Example

Create a simple HTML Login screen containing two text fields for the user ID and the password and two buttons Reset and Submit.



```
<html>
<head>
<title>Magazine Publisher Login</title>
</head>
<body bgcolor="pink" text="blue">
<center>
<h2>Magazine Publisher Login Screen</h2>
<form action="http://localhost:8080/servlet/LoginServlet"
method="post">
<pre>
Enter    your    Login           :    <input    type="text"
name="uid"><br><br>
```

```
Enter your Password : <input type="password"
name="pwd"><br><br>
</pre>
<br>
<input type="submit" value="submit"> <input
type="reset" value="reset">
</form>
</html>
```

Create a LoginServlet that will be invoked by the login.html. This servlet checks the user's ID and password. If user's ID and password are correct, greets the user, otherwise it displays an error message. To stay focused on the servlet - coding techniques we will just compare the ID and password with value "palar" and "admin".

#### Step1:

The `javax.servlet` and `javax.servlet.http` packages provide interfaces and classes for writing servlets. So first we have to import `javax.servlet` and `javax.servlet.http` packages.

#### Step2:

When implementing a servlet program, we can extend `HttpServlet` class. The `HttpServlet` class provides methods such as `doGet()` and `doPost()` for handling `Http` - specific services.

#### Step3:

Indicate the content type (for example `text/html`) being returned by the response with the `setContentType()` method.

```
response.setContentType("text/html");
```

#### Step 4:

Retrieve an `OutputStream` to send data to the client. To send the character data, use the `PrintWriter` returned by the response's `getWriter()` method.

#### Step 5:

The Servlet gets parameters supplied by the user's browser from the request object. It can do this with the help of `getParameter()` method.

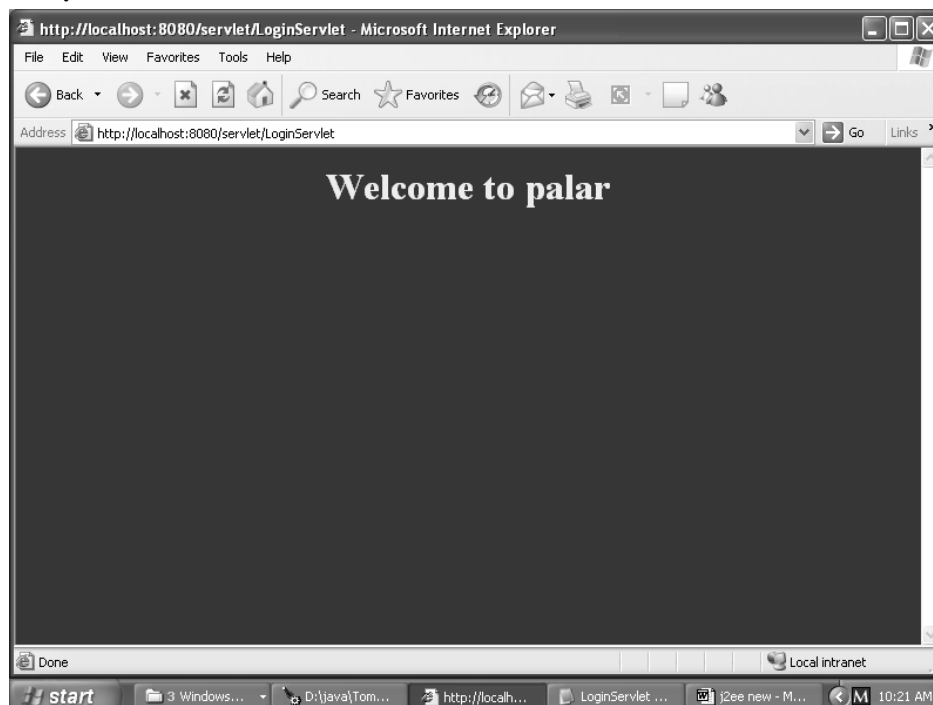
#### Step 6:

The servlet forms the content of the output page and sends it to the output stream `printWriter()`.

## Servlet Program

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class LoginServlet extends HttpServlet
{
    public void doPost (HttpServletRequest
req, HttpServletResponse res) throws
ServletException, IOException
    {
        res.setContentType("text/html");
        PrintWriter out=res.getWriter();
        String id=req.getParameter("uid");
        String pass=req.getParameter("pwd");
        out.println("<html><body bgcolor=red text=yellow>");
        if(id.equals("palar") && pass.equals("admin"))
        {
            out.println("<h2><center>Welcome to our
Magazine</center></h2>");
        }
        else
        {
            out.println("<h2><center>Invalid user id or
password</center></h2>");
        }
        out.println("</body></html>");
    }
}
```

## Output



## Life - Cycle of a Servlet

The Servlet engine invokes methods such as `init`, `service` and `destroy`.

### `init( )`

The servlet container calls the `init()` method exactly once after instantiating the servlet. This method is called when the servlet is initially loaded by the container.

### `service( )`

The `service( )` method is called on the servlet's ancestor every time the servlet receives a user's request. At this point the servlet container passes an instance of the class `ServletRequest`, which contains the client's data, to the servlet. The `service()` method creates the objects `HttpServletRequest` and `HttpServletResponse` and passes them as parameters to the `doGet()` or `doPost()` method of the descendent class.

### `destroy( )`

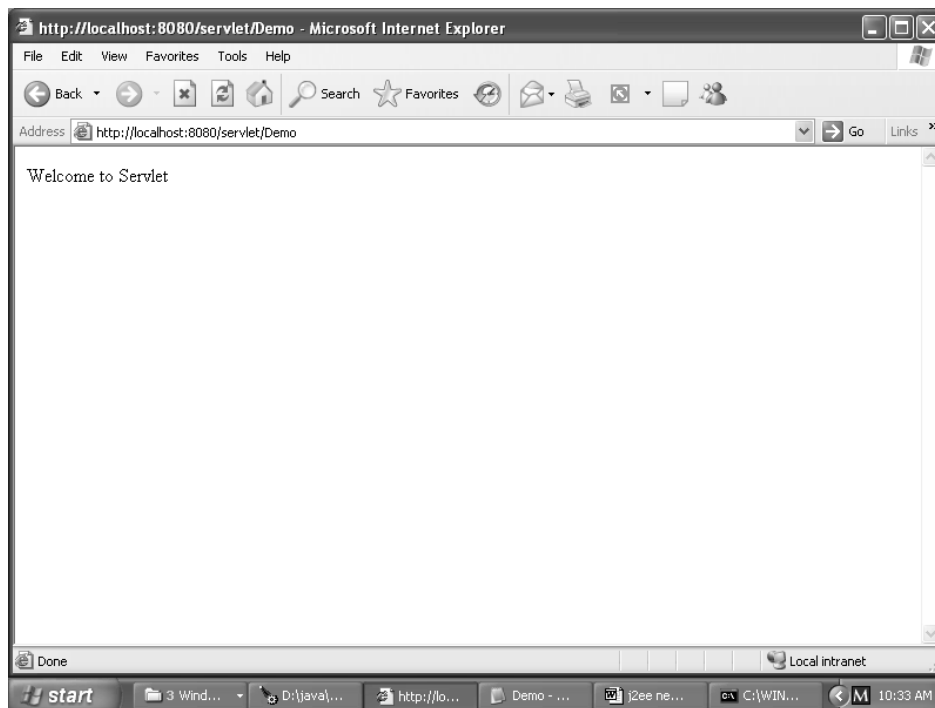
The method `destroy()` is called by the servlet container to notify the servlet that it's about to be removed from service. This method will be called only once, and developers should use it to program clean up of any resources that are being held.

## Example

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class Demo extends HttpServlet
{
    String str;
    public void init() throws ServletException
    {
        str="Welcome to Servlet";
    }
    public void service(ServletRequest req,ServletResponse
res) throws ServletException,IOException
    {
        PrintWriter out=res.getWriter();
        out.println(str);
    }
    public void destroy(){}
}
```



## Output



## Performing URL Redirection

Servlets often need to redirect the processing of their user's request to a different URL, servlet or a Java Server Page. We can do this by using either of the following methods: `HttpServletResponse.sendRedirect()` or `RequestDispatcher.forward()`.

### Example

#### Login.html

```
<html>
<head>
<title>Magazine Publisher Login</title>
<body bgcolor="pink" text="blue">
<center><h1>Magazine Publisher Login</h1></center>
<pre>
<form action="http://localhost:8080/servlet/LoginServlet"
method="post">
Enter Login ID           : <input type="text" name="uid">

Enter Password           : <input type="password"
name="pwd">

<input type="submit" value="login">           <input
type="reset">
</pre>
```

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

### LoginServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class LoginServlet extends HttpServlet
{
    public void doPost (HttpServletRequest
req, HttpServletResponse res) throws
ServletException, IOException
    {
        res.setContentType("text/html");
        PrintWriter out=res.getWriter();
        String id=req.getParameter("uid");
        String password=req.getParameter("pwd");
        out.println("<html><body bgcolor=red text=yellow>");
        if(id.equals("palar") && password.equals("admin"))
        {
            out.println("<center><h1>Welcome to
palar</center></h1>");
        }
        else
        {
            ServletContext context=getServletContext();
            RequestDispatcher
requestDisp=context.getRequestDispatcher("/servlet/LastPa
ssword");
            requestDisp.forward(req, res);
        }
        out.println("</body></html>");
    }
}
```

### Password.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class LastPassword extends HttpServlet
{
    public void doPost (HttpServletRequest
req, HttpServletResponse res) throws
ServletException, IOException
    {
        PrintWriter out=res.getWriter();
```

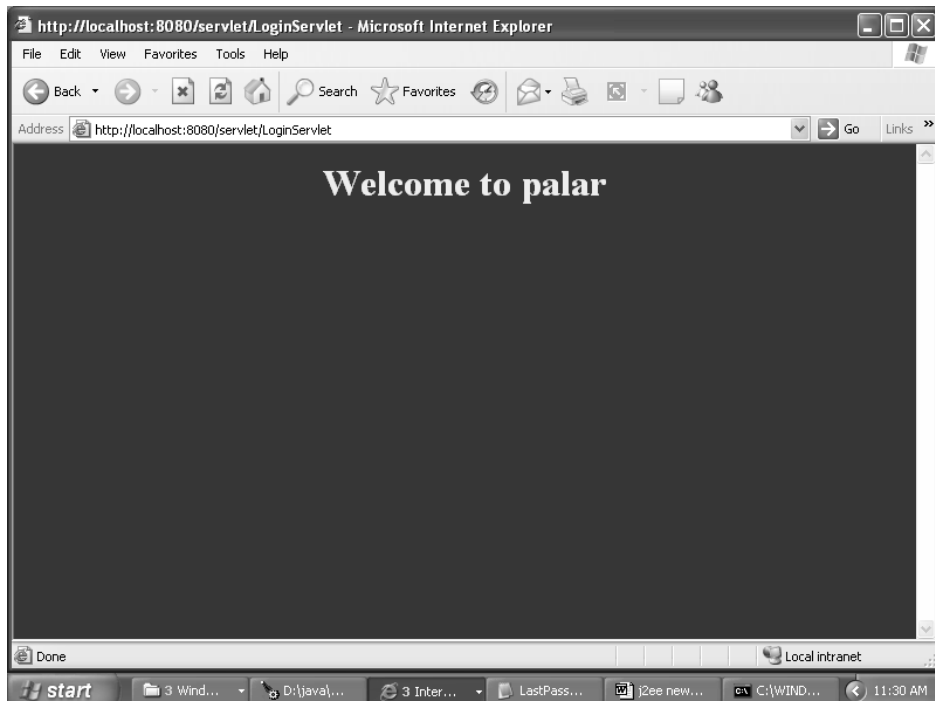
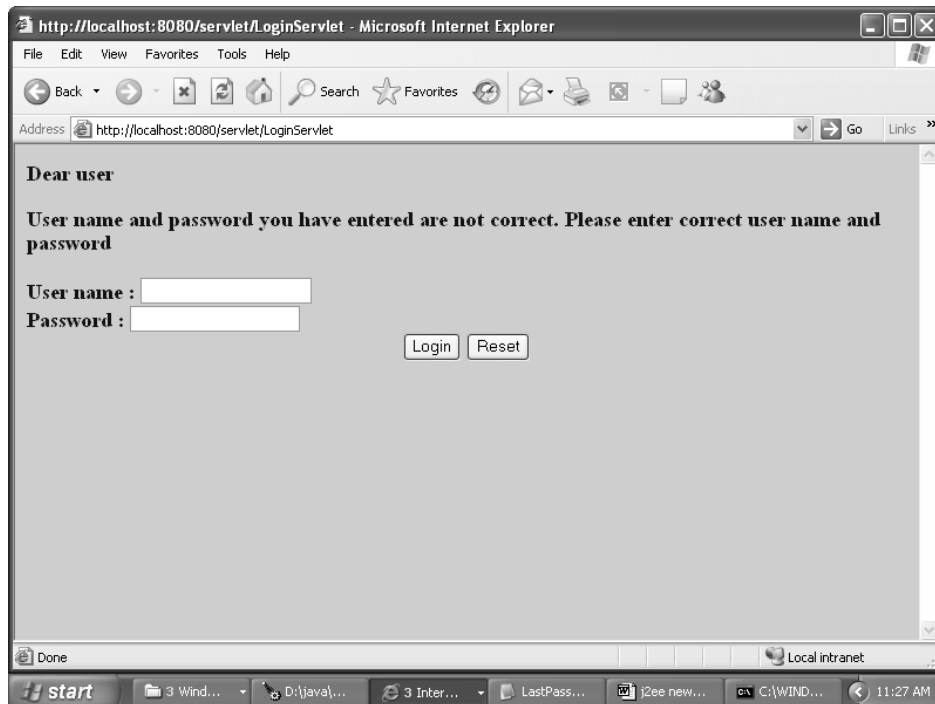
```

    out.println("<html><body bgcolor=pink text=blue>");
    out.println("<h3>Dear user<br>");
    out.println("<p>User name and password you have entered
are not correct. Please enter correct user name and
password<br>");
    out.println("<form          action=/servlet/LoginServlet
method=post>");
    out.println("User name          : <input type=text
name=uid><br>");
    out.println("Password          : <input type=password
name=pwd><br>");
    out.println("<center><input type=submit value=Login>
<input type=reset>");
    out.println("</form></body></html>");
}
}

```

## Output





## Java Servlet and JDBC

Create a university result form which is having following items

- ◆ Register number
- ◆ Submit button
- ◆ Reset button

Write a servlet program to get all marks corresponding to given register no.

### Result.html

```
<html>
<head>
<title>B.E Result page</title>
</head>
<body bgcolor="pink" text="blue">
<center><h1>ANNA UNIVERSITY</h1><br>
<h2>April 2006 Result</h2><center><br><br></center>
<pre>
<form
action="http://localhost:8080/servlet/ResultServlet"
method="post">
Enter your register no : <input type="text" name="reg">
<center> <input type="submit" value="submit">      <input
type="reset">
</form>
</body>
</html>
```

### ResultServlet.java

```
import java.io.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ResultServlet extends HttpServlet
{
    public void doPost (HttpServletRequest
req, HttpServletResponse res) throws
ServletException, IOException
    {
        res.setContentType("text/html");
        PrintWriter out=res.getWriter();
        String str=req.getParameter("reg");
        int reg=Integer.parseInt(str);
        out.println("<html><body bgcolor=red text=yellow>");
    }
}
```

```

        out.println("<center><h1>Anna
University</h1><br><h2>April
Result</h2></center><br><br>");
        try
        {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection
conn=DriverManager.getConnection("jdbc:odbc:link");
            PreparedStatement
stmt=conn.prepareStatement("select * from student where
Regno=?");
            stmt.setInt(1,reg);
            ResultSet rs=stmt.executeQuery();
            if(rs.next())
            {
                out.println("Register
: "+rs.getString("Regno")+"<br><br>");
                out.println("Name
: "+rs.getString("Name")+"<br><br><br><br>");
                out.println("<center>");
                out.println("<table border=2>");
                out.println("<tr><th>SUBJECT
CODE</th><th>Marks</th></tr>");

                out.println("<tr><td>Mark1</td><td>"+rs.getString("m1")+"
</td></tr>");

                out.println("<tr><td>Mark2</td><td>"+rs.getString("m2")+"
</td></tr>");

                out.println("<tr><td>Mark3</td><td>"+rs.getString("m3")+"
</td></tr>");

                out.println("<tr><td>Total</td><td>"+rs.getString("total"
)+"</td></tr>");
                out.println("<tr><td>Average</td><td>"+rs.getString("avg"
)+"</td></tr>");
                out.println("</table></center>");
            }
            else
            {
                out.println("<center><h2>Please      check      your
register no</h2></center>");
            }
            stmt.close();
            conn.close();
        }
        catch(Exception e)
        {
            out.println("Error"+e);
        }
    }
}

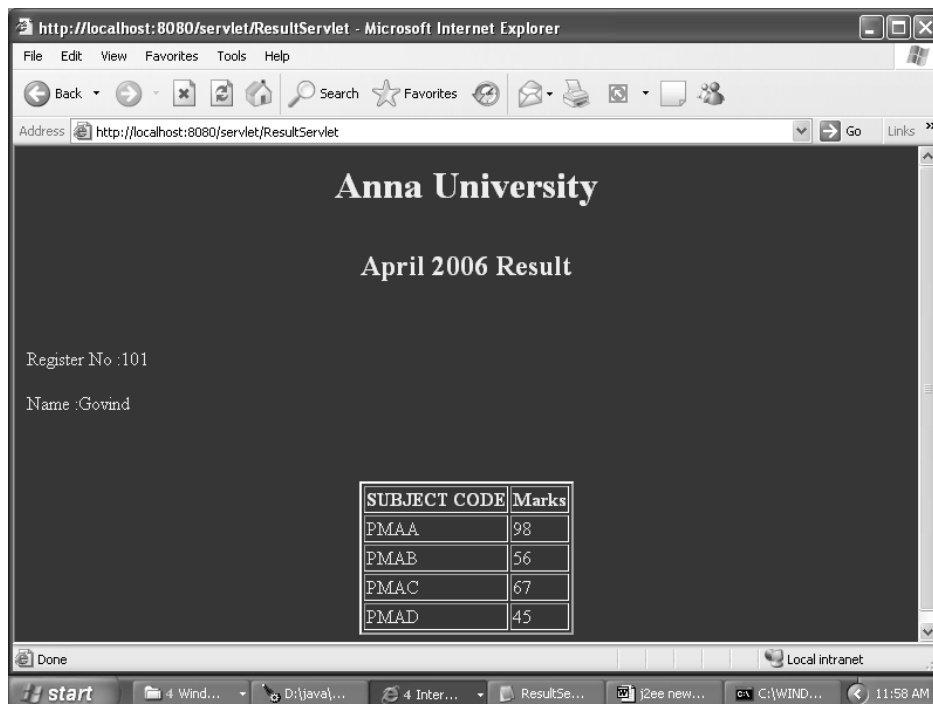
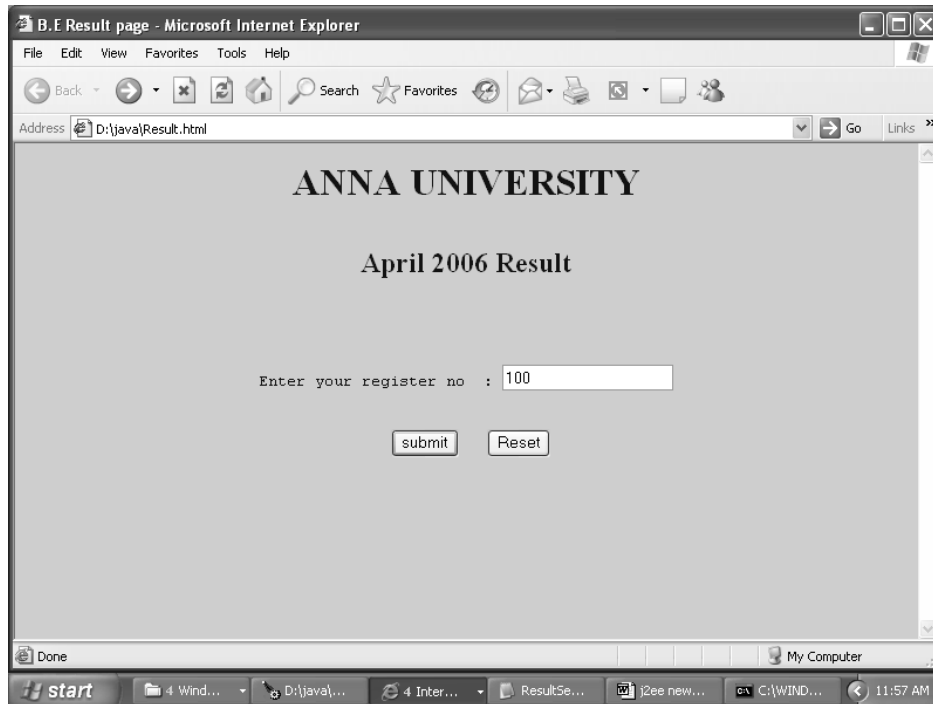
```

```

    out.println("</body></html>");
}
}

```

## Output



### Create a registration form which is having input items

- ◆ User Name
- ◆ Password
- ◆ Main ID
- ◆ Country (list)
- ◆ Submit button
- ◆ Reset button

Create a database for the above details and write a servlet to insert all information into the database.

### Registration.html

```
<html>
<head>
<title>Online Application Form</title>
</head>
<body bgcolor="pink" text="blue">
<center><h1>Online Registration Form</h1></center>
<pre>
<form
action="http://localhost:8080/servlet/RegistrationServlet
" method="post">
Enter User name                : <input type="text"
name="uid">

Enter Password                  : <input type="password"
name="pwd">

Enter your maid id              : <input type="text"
name="mid">

Select your country              : <select name="coun">
                                <option>India
                                <option>Malaysia
                                <option>Canada
                                <option>Asia
                                <option>Russia
                                </select>

<center>  <input type="submit" value="submit">      <input
type="reset"></center>
</form>
</pre>
</body>
</html>
```



**RegistrationServlet.java**

```

import java.io.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class RegistrationServlet extends HttpServlet
{
    public          void          doPost (HttpServletRequest
req,HttpServletResponse          res)          throws
ServletException,IOException
    {
        res.setContentType("text/html");
        PrintWriter out=res.getWriter();
        String uid=req.getParameter("uid");
        String pwd=req.getParameter("pwd");
        String mid=req.getParameter("mid");
        String coun=req.getParameter("coun");
        out.println("<html><body bgcolor=red text=yellow>");
        try
        {
            Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection
conn=DriverManager.getConnection("jdbc:odbc:link");
            PreparedStatement
stmt=conn.prepareStatement("insert          into          applicant
values(?,?,?,?)");
            stmt.setString(1,uid);
            stmt.setString(2,pwd);
            stmt.setString(3,mid);
            stmt.setString(4,coun);
            stmt.executeUpdate();
            out.println("<center><h1>Record          has          been
inserted</h1></center>");
            stmt.close();
            conn.close();
        }
        catch(Exception e)
        {
            out.println("Error"+e);
        }
        out.println("</body></html>");
    }
}

```

## Output

The screenshot shows a web browser window titled "Online Application Form - Microsoft Internet Explorer". The address bar displays "D:\java\Registration.html". The page content is titled "Online Registration Form". It contains four input fields: "Enter User name" with the value "Govind", "Enter Password" with masked characters "••••••••", "Enter your maid id" with the value "govindmsc\_82@yahoo.", and "Select your country" with a dropdown menu showing "India". Below the fields are two buttons: "submit" and "Reset". The taskbar at the bottom shows the "start" button and several open applications, including "4 Wi...", "D:\jav...", "5 In...", "Regist...", "j2ee n...", "C:\WI...", and "db1 : ...". The system clock shows "12:19 PM".

Online Application Form - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Forward Stop Home Search Favorites Print Mail New Tab New Window

Address D:\java\Registration.html Go Links

### Online Registration Form

Enter User name : Govind

Enter Password : ••••••••

Enter your maid id : govindmsc\_82@yahoo.

Select your country : India

submit Reset

Done My Computer

start 4 Wi... D:\jav... 5 In... Regist... j2ee n... C:\WI... db1 : ... 12:19 PM

The screenshot shows a web browser window titled "http://localhost:8080/servlet/RegistrationServlet - Microsoft Internet Explorer". The address bar displays "http://localhost:8080/servlet/RegistrationServlet". The page content is a large black rectangle with the text "Record has been inserted" in white. The taskbar at the bottom shows the "start" button and several open applications, including "4 Wi...", "D:\jav...", "5 In...", "Regist...", "j2ee n...", "C:\WI...", and "db1 : ...". The system clock shows "12:20 PM".

http://localhost:8080/servlet/RegistrationServlet - Microsoft Internet Explorer

File Edit View Favorites Tools Help

Back Forward Stop Home Search Favorites Print Mail New Tab New Window

Address http://localhost:8080/servlet/RegistrationServlet Go Links

### Record has been inserted

Done Local intranet

start 4 Wi... D:\jav... 5 In... Regist... j2ee n... C:\WI... db1 : ... 12:20 PM

# JSP - Java Server Page

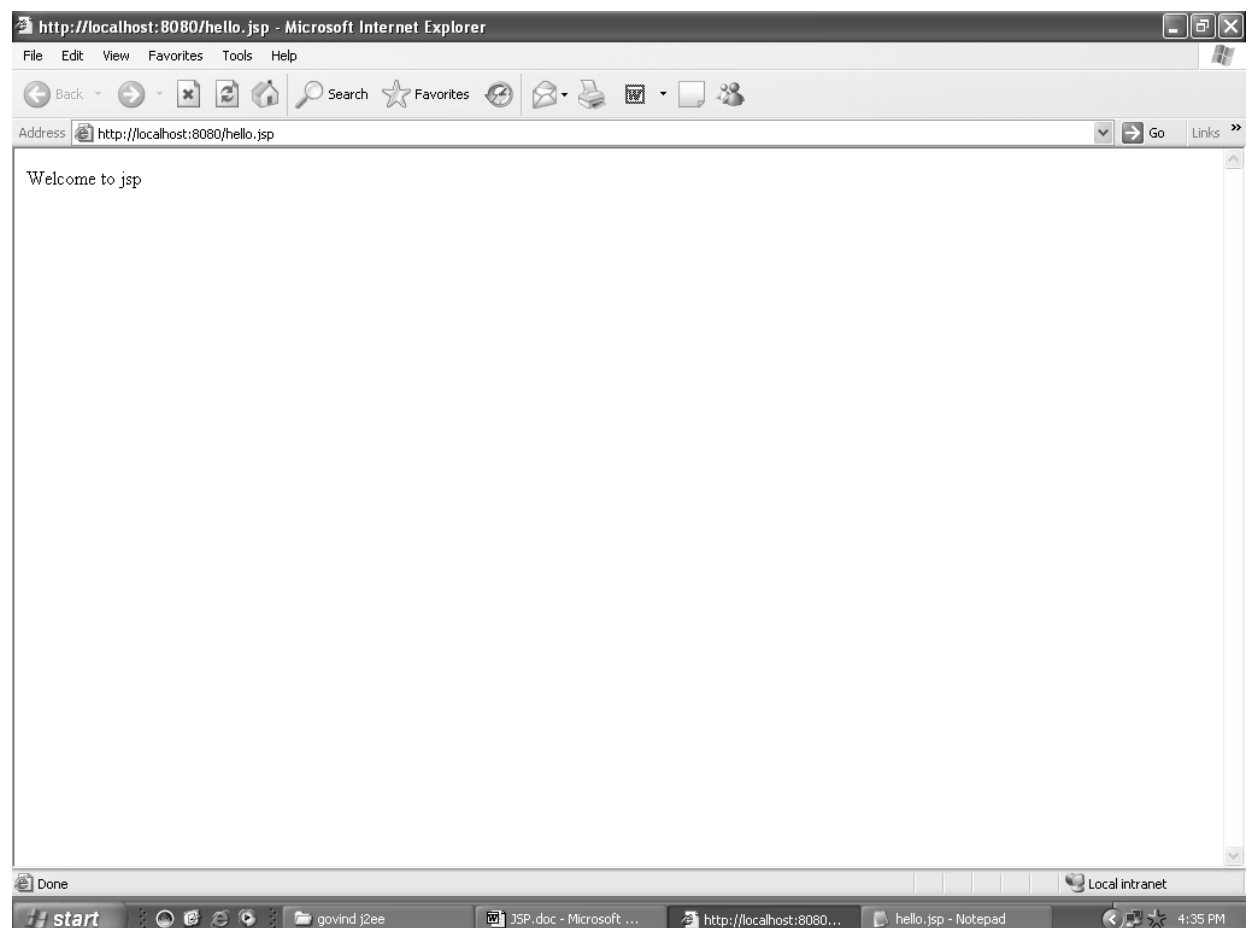
## Introduction

Java Server Pages are a simple but powerful technology used to generate dynamic HTML on the server side. The JSP specification defines JSP as a technology for building the applications for generating dynamic web content such as HTML, DHTML and XML.

## Example

```
<html>
<body>
<% out.println("Welcome to jsp"); %>
</body>
</html>
```

## Output



## JSP Scripting elements

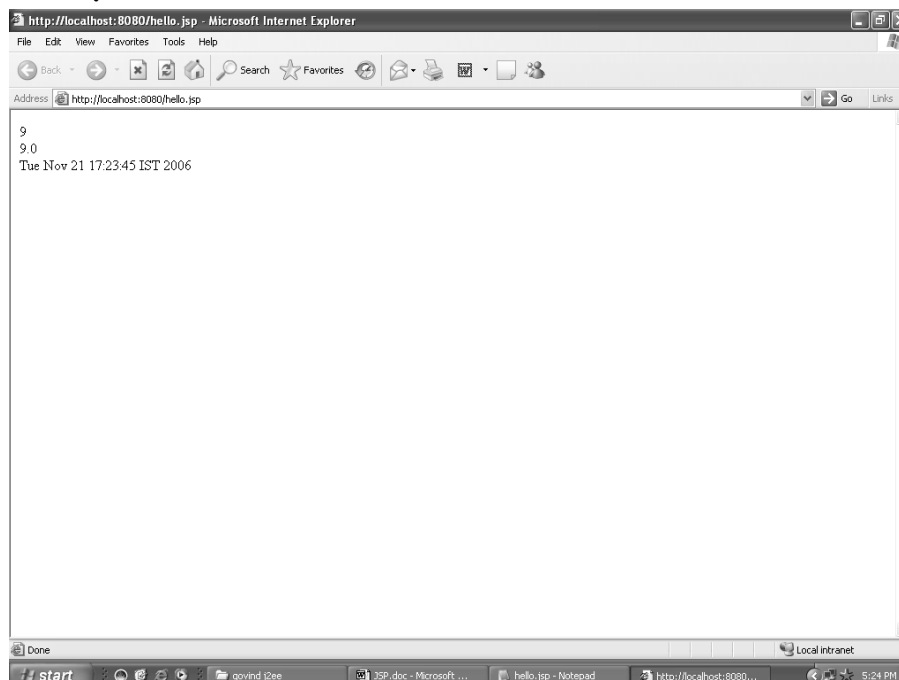
JSP elements (tags) can be grouped according to the functions they perform. The tags are surrounded by angle brackets (< >), and like HTML and XML tags can have attributes. Everything else that can be contained in the JSP page, that cannot be known to the JSP translators (plain text, html tags, and so on) is called template data.

### 1. Expressions

JSP expressions start with `<%=` and can contain any java expression, which will be evaluated and its result inserted into the HTML page right where the expression is located.

```
<html>
<body>
<%= 5+4 %> <br>
<%= Math.sqrt(81.0) %> <br>
<%= new java.util.Date() %>
</body>
</html>
```

### Output

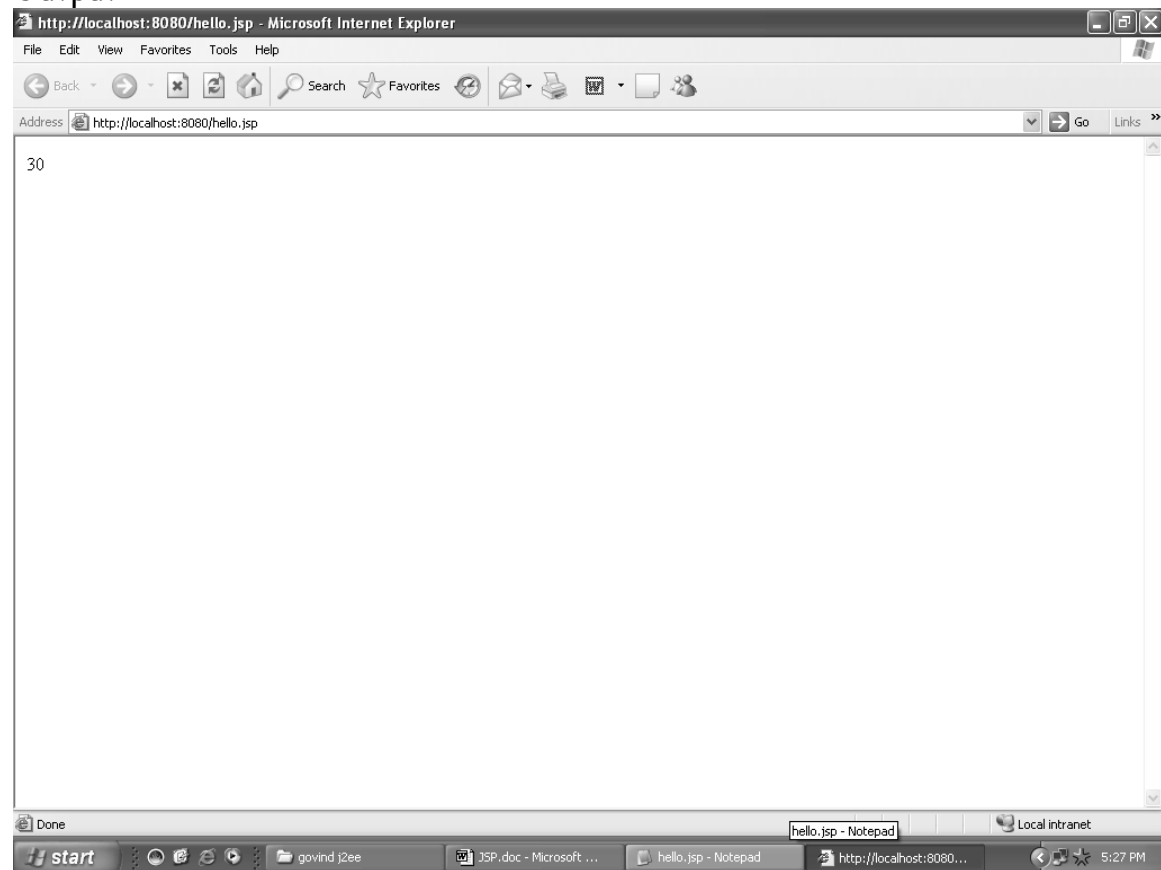


## 2. Declarations

Declarations do not generate the output to the user's screen. They are used for declaring class variables and methods and start with `<%!`

```
<html>
<body>
<%! int a=20; %>
<%! int b=10; %>
<%= a+b %>
</body>
</html>
```

### Output



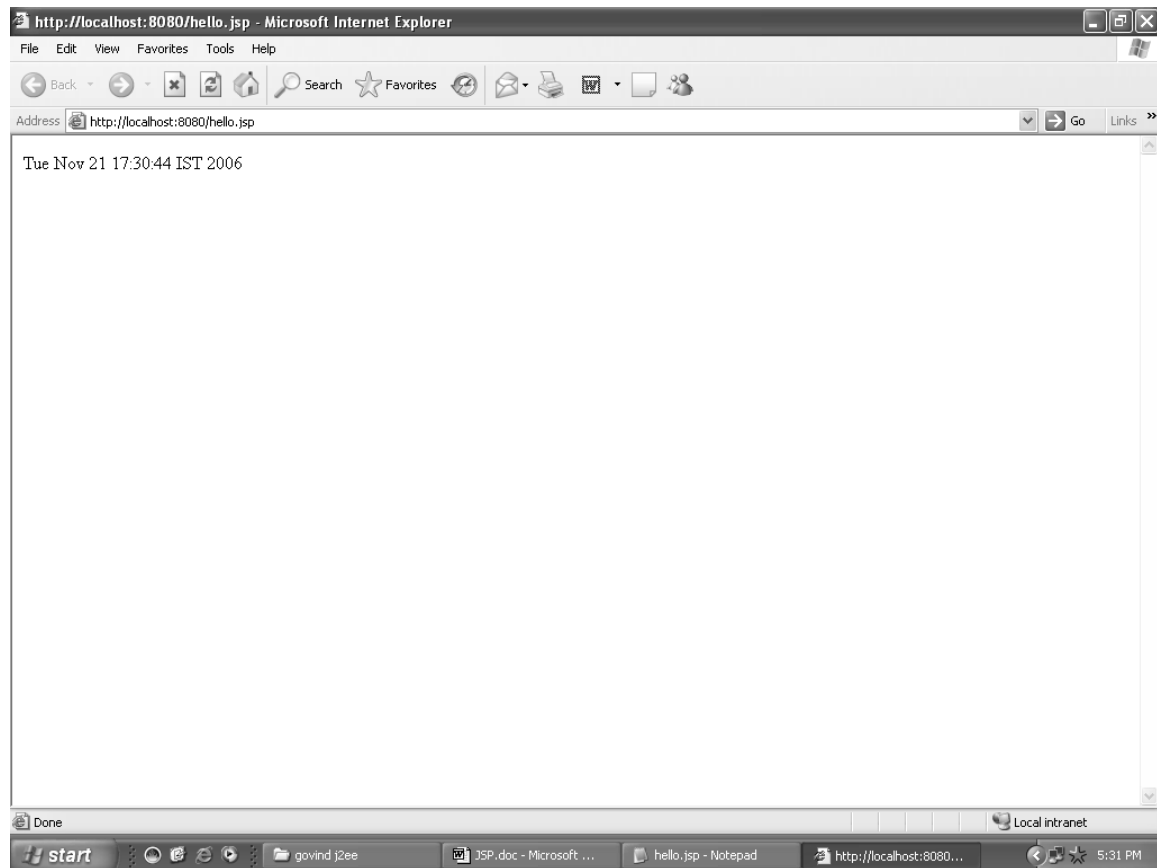
## 3. Directives

Directives do not generate screen output. They inform the JSP engine about the rules to be applied to the JSP. The page directive starts with `<%@` page.

## Example

```
<html>
<body>
<%@ page import="java.util.*" %>
<%! Date da=new Date(); %>
<%= da %>
</body>
</html>
```

## Output



## Note:

The include directive allows inclusion of any text file or code from another JSP, at the time when the page is compiled into a servlet.

## Example

```
<%@ include file="Hello.jsp" %>
```

## 4. Scriptlets

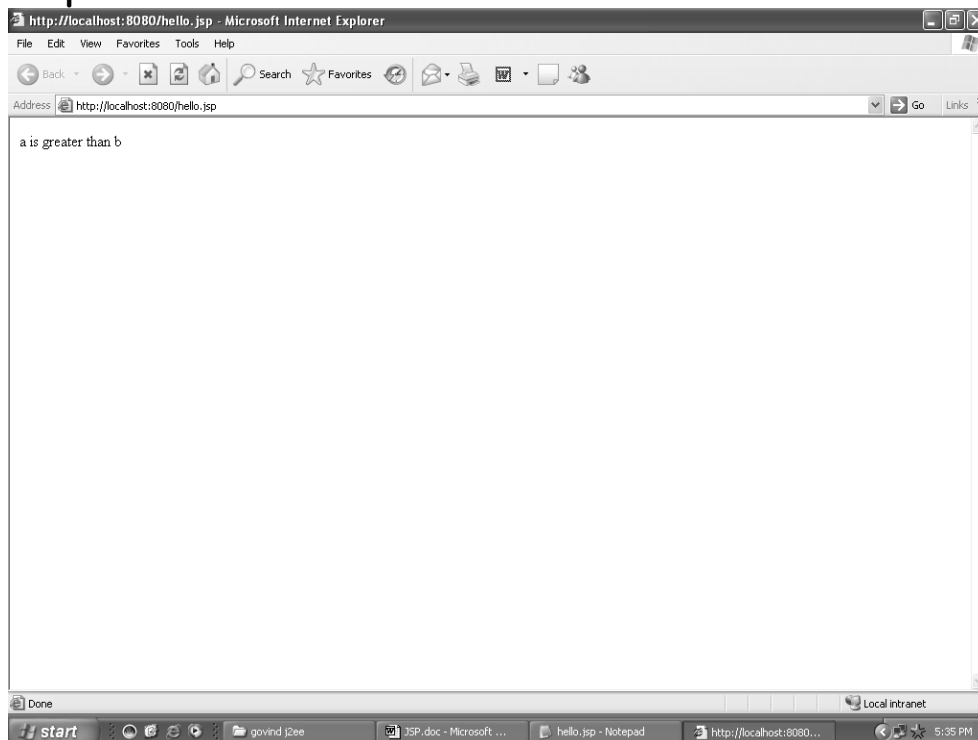
Scriptlets can contain any valid Java code. For example, within scriptlets you can also insert variable and method declaration, java expressions, and so on. Scriptlets starts with `<%`.

### Example

```
<html>
<body>
<%! int a=20; %>
<%! int b=10; %>
<%
    if(a>b)
    {
        out.println("a is greater than b");
    }
    else
    {
        out.println("b is greater than a");
    }
%>

</body>
</html>
```

### Output



## 5. Comments

JSP comments starts with `<%--` and ends with `--%>`, and are not included in the output web page.

### Note:

If you need to include comments in the source of the output page, use the HTML comments, which starts with `<!--` and ends with `-->`

## JSP and HTML forms

Creating an HTML form using a JSP is just like creating a form using any other tool. JSP just gives you the ability to create form elements dynamically.

### Example

Create a simple HTML login screen containing two text fields for the user ID and the password and two buttons Login and Reset

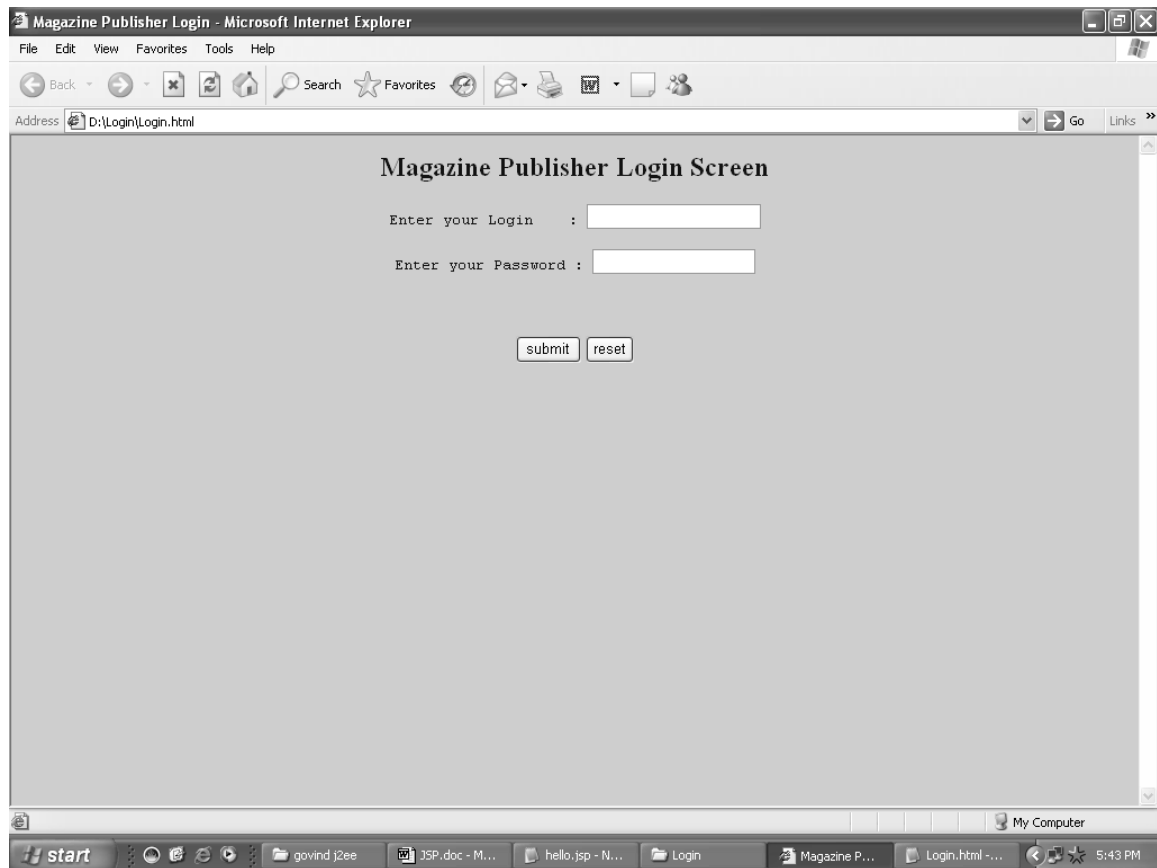
```
<html>
<head>
<title>Magazine Publisher Login</title>
</head>
<body bgcolor="pink" text="blue">
<center>
<h2>Magazine Publisher Login Screen</h2>
<form action="http://localhost:8080/login.jsp" method="post">

<pre>
Enter your Login : <input type="text"
name="uid"><br><br>
Enter your Password : <input type="password"
name="pwd"><br><br>
</pre>

<br>
<input type="submit" value="submit"> <input
type="reset" value="reset">
</form>
</html>
```



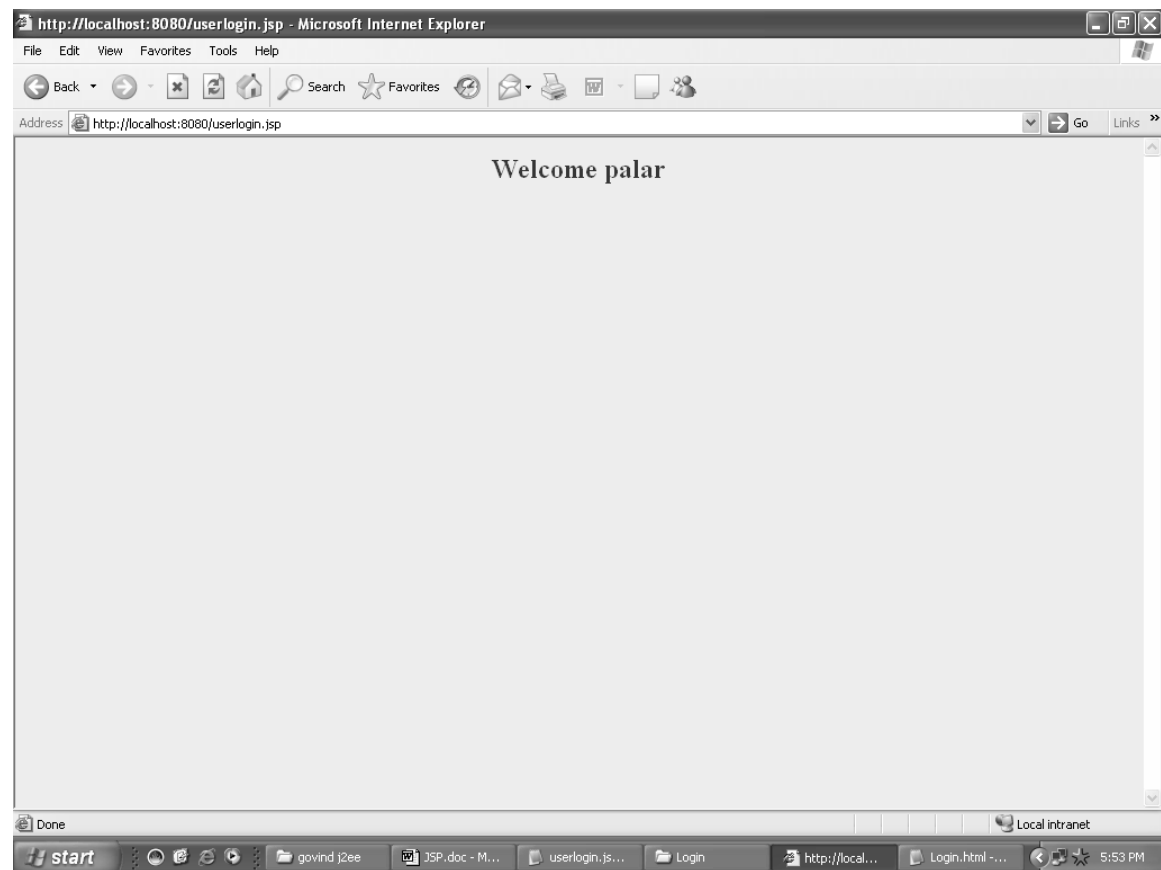
## Output



Create the userlogin.jsp that will be invoked by login.html. This program checks the user's ID and password. If it is correct, greets the user; otherwise it displays an error message.

```
<html>
<body bgcolor=yellow text=red>
<%! String uname; %>
<%! String pwd; %>
<% uname=new String(request.getParameter("uid"));
    pwd=new String(request.getParameter("pwd"));
    out.println("<h2><center>");
    if(uname.equals("palar") && pwd.equals("admin"))
    {
        out.println("Welcome "+uname);
    }
    else
    {
        out.println("You are not a registered user");
    }
    out.println("</h2><center>");
%>
</body>
</html>
```

## Output



# JSP and JDBC

## Example (Hello.html)

```
<html>
<head>
<title>Magazine Publisher Login</title>
</head>
<body bgcolor="pink" text="blue">
<center>
<h2>Magazine Publisher Login Screen</h2>
<form          action="http://localhost:8080/result.jsp"
method="post">
<pre>
Enter your register no : <input type="text" name="reg">
</pre>
<br>
<input  type="submit"  value="submit">          <input
type="reset" value="reset">
</form>
</html>
```

## Output



## Result.jsp

```

<html>
<body bgcolor=yellow text=red>
<%@ page import="java.sql.*" %>
<%! String str; %>
<%! int reg; %>
<%
    str=new String(request.getParameter("reg"));
    reg=Integer.parseInt(str);
    try
    {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection
conn=DriverManager.getConnection("jdbc:odbc:palar","scott
","tiger");
        PreparedStatement stmt=conn.prepareStatement("select
* from student where regno=?");
        stmt.setInt(1,reg);

        ResultSet rs=stmt.executeQuery();
        if(rs.next())
        {
            out.println("Register no
:"+rs.getString("regno")+"<br>");
            out.println("Name
:"+rs.getString("name")+"<br>");
            out.println("<table
border=2><tr><th>Subject</th><th>Mark</th></tr>");

            out.println("<tr><td>Tamil</td><td>"+rs.getString("tamil"
)+"</td></tr>");

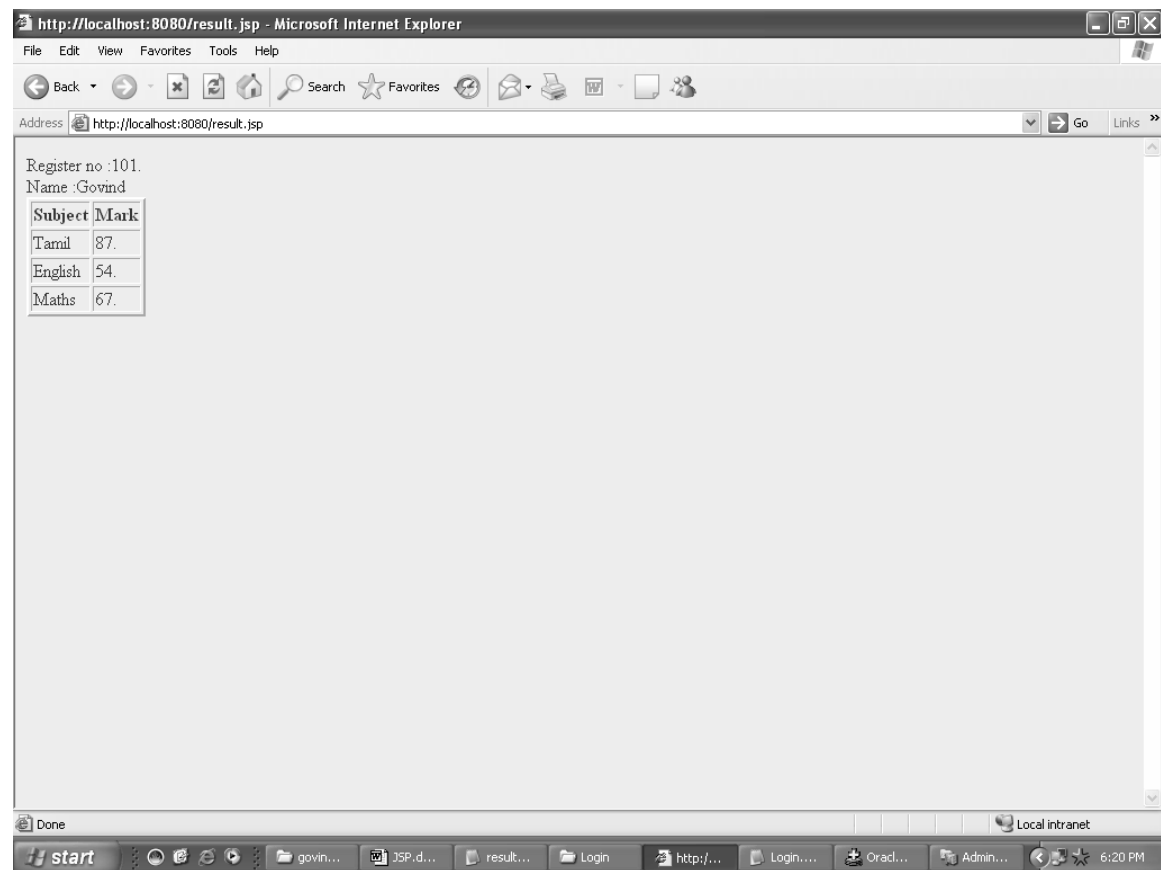
            out.println("<tr><td>English</td><td>"+rs.getString("engl
ish")+"</td></tr>");

            out.println("<tr><td>Maths</td><td>"+rs.getString("maths"
)+"</td></tr>");
            out.println("</table>");
        }
        else
        {
            out.println("<h2><center>Record Not
Found</center></h2>");
        }
    }
    catch(Exception e)
    {
        out.println("Error"+e);
    }
%>

```

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

## Output



## Handling JSP Errors

Errors can occur in a JSP in two different phases of its life. The first type of JSP error occurs when a Java Server Page is first requested and goes through the initial translation from a JSP source file into a corresponding servlet class file is known as a translation time error. The second type of JSP error occurs during request time. These errors are runtime errors that can occur in either the body of the JSP page or in some other object that is called from the body of the JSP page, are known as request time error.

### Creating a JSP error page

To create a JSP error page, you need to create a basic Java Server Page and then you need to tell the JSP engine that the page is an error page. You do this by setting its page attribute `isErrorPage` to `true`.

### Example

```
<html>
<body>
<%@ page isErrorPage="true" %>
Dear friend!
<p>
We are sorry to inform that there is a little problem
during the execution of this problem.
</body>
</html>
```

## Using a JSP Error page

It takes only one additional attribute, in your page directive, to make your JSP aware of an error page. You simply need to add the `errorPage` attribute and set its value equal to the location of your JSP error page.

```
<html>
<body>
<%@ page errorPage="DivisionError.jsp" %>
</body>
</html>
```

## Session

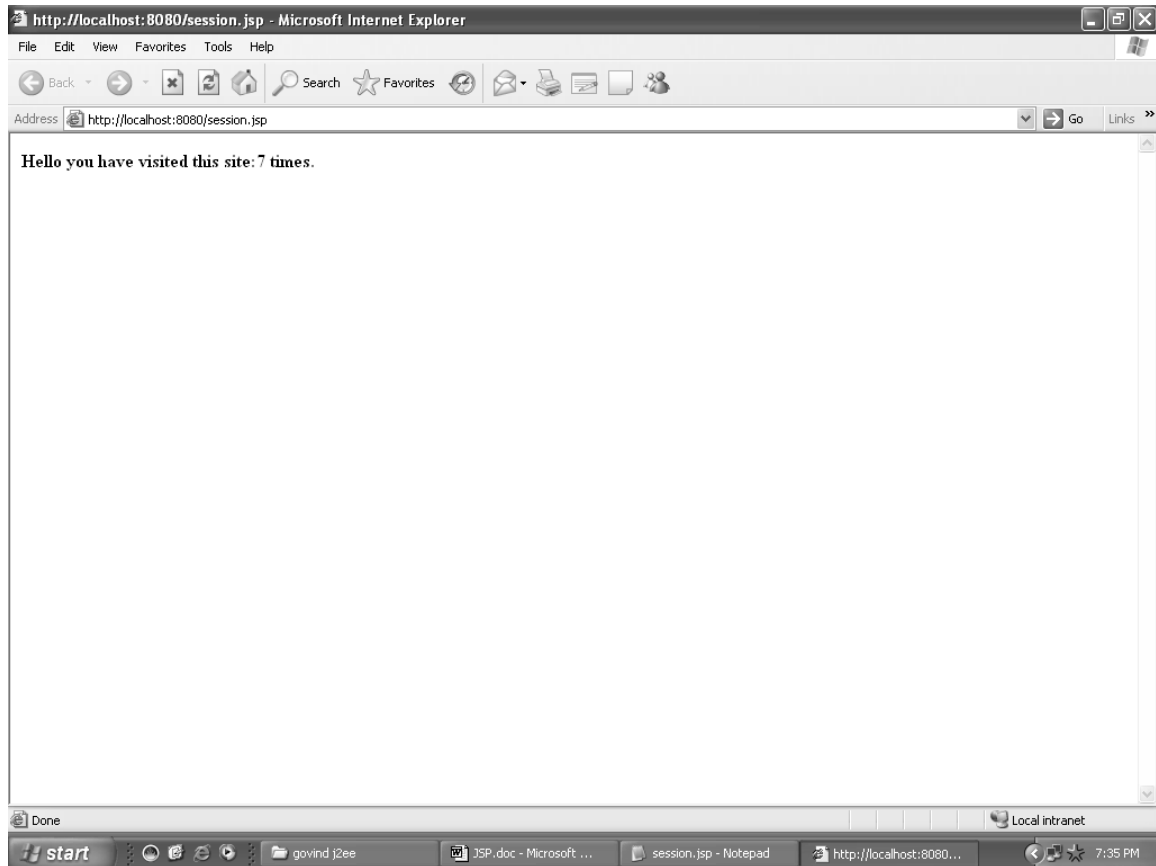
The implicit session object holds a reference to a `javax.servlet.http.HttpSession` object. The `HttpSession` object is used to store objects in between client requests.

### Example

```
<html>
<body>
<%
    Integer
count=(Integer) session.getAttribute("COUNT");
    if(count==null)
    {
        count=new Integer(1);
        session.setAttribute("COUNT",count);
    }
    else
    {
        count=new Integer(count.intValue()+1);
        session.setAttribute("COUNT",count);
    }
}
```

```
        out.println("<b>Hello    you    have    visited    this  
site:"+count+" times.</b>");  
    %>  
</body>  
</html>
```

## Output



# J2EE Applications, Architecture and Design

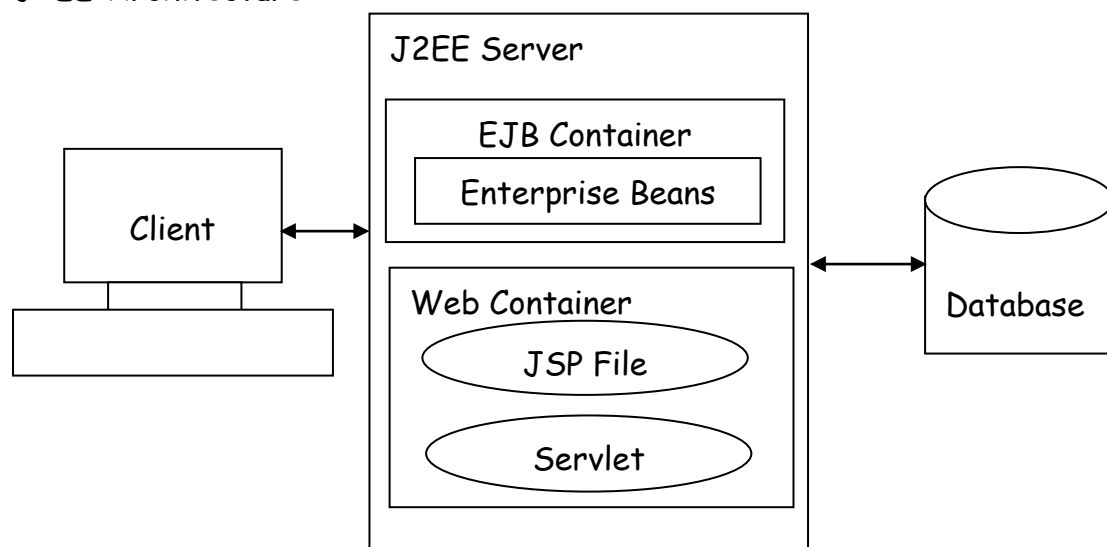
## History of Java, J2SE and J2EE

In 1995, sun released Java, a fully object - oriented programming language. While most of the concepts within Java were not new, it did meld many features, such as memory management and garbage collection from the syntax of C/C++, into a new easy - to - learn programming language. Java brought the concept of a virtual machine into the mainstream. Java is not compiled completely, but instead is compiled to an intermediary stage as java byte codes. At runtime, Java byte codes are executed within a virtual machine, which is a piece of software that interprets the byte codes in runtime into the native binary for the operating system.

Around 1998, sun updated the Java specification and introduced Java1.2 along with the accompanying libraries, making Java not only a language, but also a platform - Java 2 Standard Edition (J2SE). Prior to the J2SE the Java Development Kit (JDK) was the primary package that was installed and developers would choose which additional libraries they would want such as Java Database Connectivity (JDBC) or Swing. J2SE provided libraries for GUI support, networking, database access, and more. J2SE is also foundation for the J2EE.

J2EE introduced in 1998, defines a multi-tier architecture for enterprise information systems (EIS). By defining the way in which multi - tier applications should be developed, J2EE reduces the costs, in both time and money, of developing large - scale enterprise systems.

## J2EE Architecture





## **Application Components:**

Four application components are defined within the J2EE platform. They are as follows

- Application clients (Stand alone java clients)
- Applets (Java code which executes within a browser)
- Web components (JSP, Servlets)
- Server Component (EJB)

## **Application Clients:**

Clients are generally stand - alone applications written in Java. They run within a virtual machine and can use the J2EE standard services to access components located within another tier. The J2EE standard services are usually provided on the clients via an installation of J2SE, or along with the distribution of the application itself.

## **Applets**

Applets are similar to application clients, but executes within a web browser. Initially applets generated extensive attention, as they were seen as a means of making web pages more dynamic. Most web browsers have an embedded Java Virtual Machine (JVM); however, the Java plug-in can be used to force the browser to use a particular version of JVM.

## **Web components**

Although the term can be misleading web components do not execute on the client side. Web components are server side components, generally used to provide the presentation layer to be returned to a client. Two types of web components exist: Java Server Pages (JSP) and Java Servlets. Very basically, JSP are similar to regular HTML pages but contain embedded Java code while java servlets are Java classes that use Java's I/O application programming interfaces (API) to output HTML to the client. Both JSP and Servlet can be used to output other format types.

## **Server Components**

Server components come in the form of Enterprise Java Beans (EJB). EJB executes within a container that manage the runtime behavior of the EJB. EJBs are usually where the business logic for an enterprise system resides.

## **Application Servers**

An Application Server provides the infrastructure and services to run components/applications

### **Popular Application Servers are**

- BEA Web logic
- Borland Enterprise Server
- IBM Web sphere
- JBoss
- Oracle 9iAS
- Orion
- Sun ONE Application Server

### **Partial J2EE Application Servers**

- Apache Tomcat
- Resin
- Servlet Exec.

## **Packaging**

A J2EE application is delivered in an Enterprise Archive (EAR) file, a standard Java Archive (JAR) file with an .ear extension. Using EAR files and modules makes it possible to assemble a number of different J2EE applications using some of the same components. No extra coding is needed; it is only a matter of assembling various J2EE modules into J2EE EAR files.

An EAR file contains J2EE modules and deployment descriptors. A deployment descriptor is an XML document with an .xml extension that describes the deployment settings of an application, a module, or a component. Because deployment descriptor information is declarative, it can be changed without the need to modify the source code. At runtime, the J2EE server reads the deployment descriptor and acts upon the application, module, or component accordingly.

## **Deployment Roles**

Reusable modules make it possible to divide the application development and deployment process into distinct roles so that different people or companies can perform different parts of the process.

The first two roles involve purchasing and installing the J2EE product and tools. After software is purchased and installed, J2EE components can be developed by application component providers, assembled by application assemblers, and deployed by application deployers. In a large organization, each of these roles **might be executed by different individuals or teams.**

### **Starting the J2EE Server**

To start the J2EE server type the following command in DOS prompt.

```
j2ee -verbose
```

### **Shut downing the J2EE Server**

To stop the J2EE server type the following command in DOS prompt.

```
j2ee -stop
```

### **Starting the deploytool Utility**

To package J2EE applications, specify deployment descriptor elements, and deploy applications on the Sun Java System Application Server, you use the deploytool utility. To start deploytool, open a terminal window or command prompt and execute

```
deploytool
```

# **EJB**

## **Enterprise Java Bean**

### **Introduction**

Enterprise beans are server side components written in the Java programming language. Enterprise beans contain the business logic for the applications. The Enterprise Java Beans architecture is a component architecture for the development and deployment of component - based distributed business applications. Applications written using the Enterprise Java Beans architecture is scalable, transactional, and multi - user, secure. These applications may be written once and deployed on any server platform that supports the Enterprise Java Beans specification.

### **Benefits of Enterprise Beans**

For several reasons, enterprise beans simplify the development of large, distributed applications.

First, because the EJB container provides system level services to enterprise beans, the bean developer can concentrate on solving business problems. The EJB container and not the bean developer - is a responsible for system level services such as transaction management and security authorization.

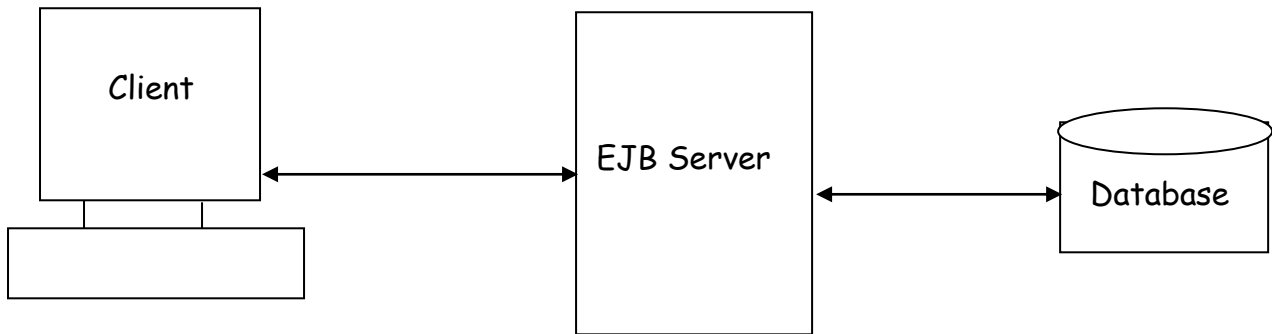
Second, because the beans - and not the clients contain the application's business logic. The client developer can focus on the presentation of the client. The client developer does not have to code the routines that implement business rules or access databases. As a result, the clients are thinner, a benefit that is particularly important for clients that run on small devices.

Third, because enterprise beans are portable components, the application assembler can build new applications from existing beans. These applications can run on any compliant J2EE server provided that they use the standard APIs.

### **EJB Architecture**

An EJB system is logically a three - tier system. Three tiers are as follows:

1. The Client
2. The EJB Server
3. The Database

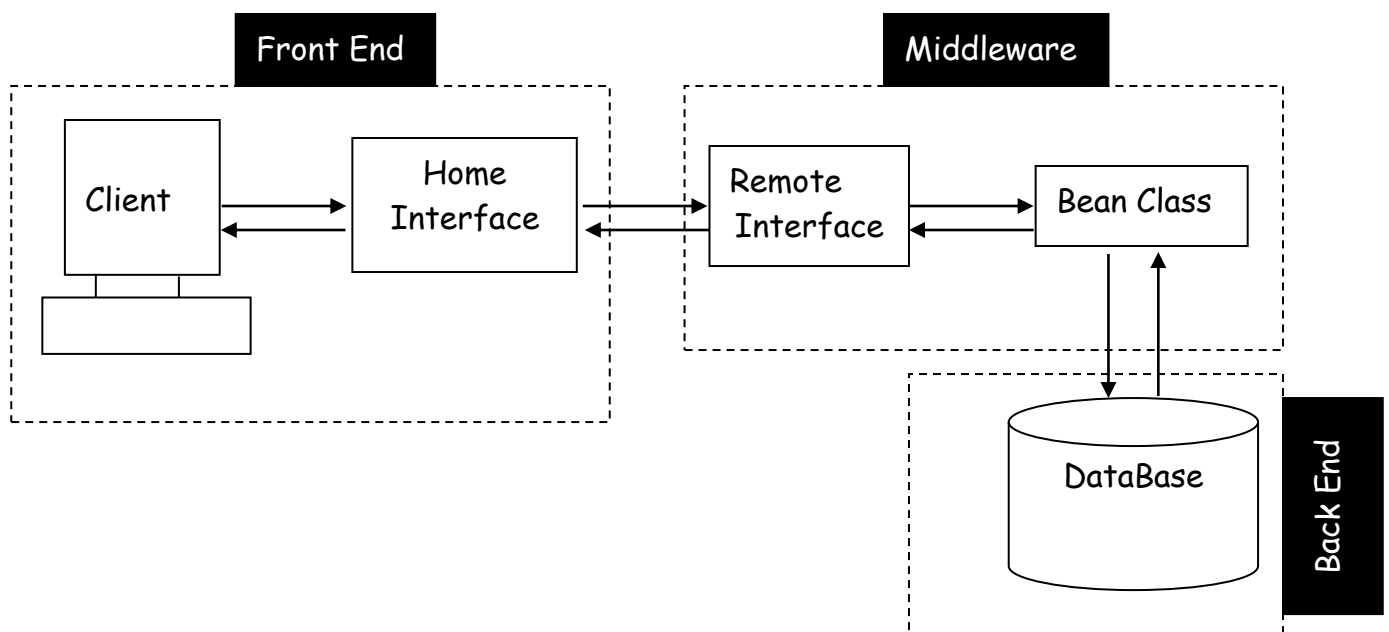


### EJB's Software Architecture

The Enterprise Java Beans Application should consist of the following programs.

1. The Remote Interface
2. The Home Interface
3. The Enterprise Java Bean
4. The Client Program

The client never communicates directly with EJB Server. It communicates to the bean through its home interface and its remote interface, both of which are provided by the container.



## **Types of EJB**

1. Session Bean
2. Entity Bean

### **Session Bean**

A Session Bean represents a single client inside the application server. To access an application that is deployed on the server, the client invokes the session bean's method.

There are two type of Session Bean

1. **Stateless**
2. **Stateful**

### **Stateless**

Stateless session beans are generally used to provide service. They do not store any conversation state between calls.

### **Case study**

Eco Scientific Labs is an organization involved in performing scientific researches. One of their projects requires the daily temperatures at different locations in the country. The organization has branches at different places in the country. The branches are equipped with instruments to measure the temperature in the Fahrenheit scale. But the research process requires the temperatures in the Celsius scale. Buying instruments to measure the temperature in Celsius scale is very expensive, since each branch should be given a separate instrument. Instead, they have opted to create a distributed application that will convert the temperature from Fahrenheit to Celsius scale.

The developer wants to implement this application using EJB.

## Creating Program

### The Remote Interface (Temp.java)

A remote interface defines the business methods that a client can call. The business methods are implemented in the enterprise bean code.

```
package Conn;
import java.rmi.*;
import javax.ejb.*;
public interface Temp extends EJBObject
{
    public double ftoc(double f) throws RemoteException;
}
```

### The Home Interface (TempHome.java)

The home interface defines the methods that allow a client to create, find, or remove an enterprise bean.

```
package Conn;
import java.io.*;
import java.rmi.*;
import javax.ejb.*;
public interface TempHome extends EJBHome
{
    Temp create() throws RemoteException,CreateException;
}
```

### The Enterprise Bean class (TempBean.java)

This bean implements a business method, that the remote interface defines

```
package Conn;
import java.rmi.*;
import javax.ejb.*;
public class TempBean implements SessionBean
{
    public TempBean() {}
    public void ejbCreate() {}
    public void ejbRemove() {}
    public void ejbActivate() {}
    public void ejbPassivate() {}
    public void setSessionContext(SessionContext sc) {}
    public double ftoc(double f)
    {
        double c;
        c=((f-32)*5)/9;
        return(c);
    } }
}
```

## Compiling the source files

Now we are ready to compile the remote interface (Temp.java), the home interface (TempHome.java), and the enterprise bean class (TempBean.class).

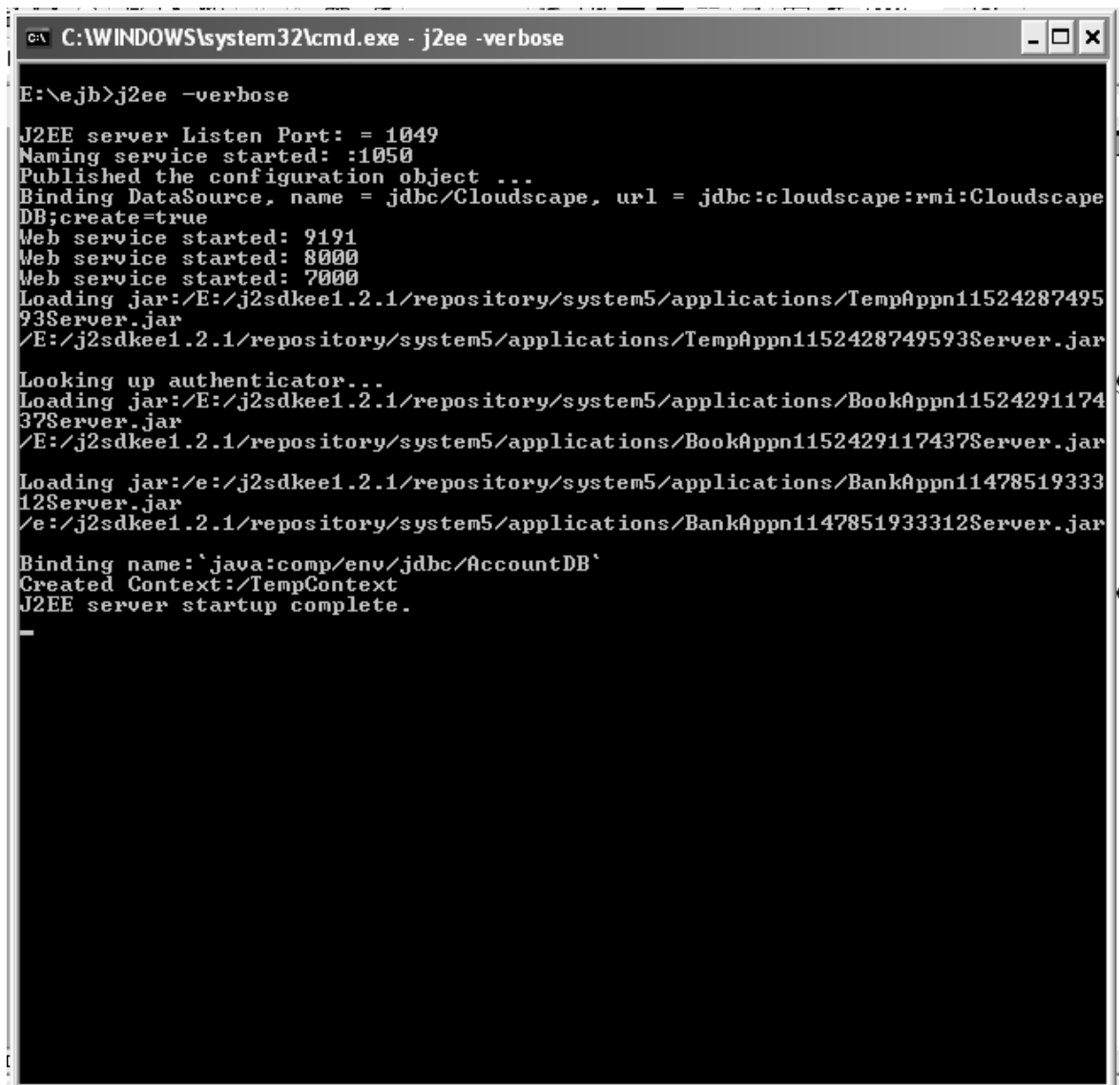
To do this type the following command.

```
javac *.java
```

## Deployment

1. Open Dos prompt and type the following command to start J2EE server.

```
j2ee -verbose
```



```
C:\WINDOWS\system32\cmd.exe - j2ee -verbose

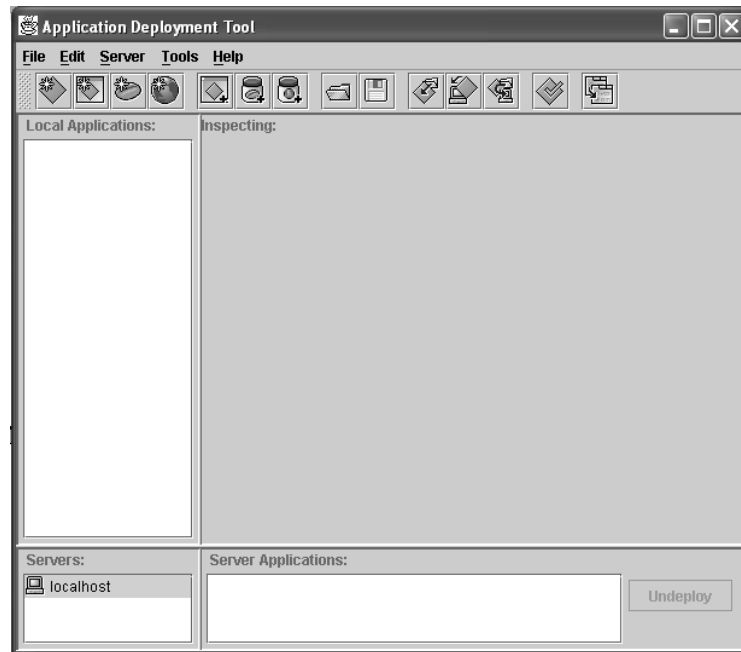
E:\ejb>j2ee -verbose
J2EE server Listen Port: = 1049
Naming service started: :1050
Published the configuration object ...
Binding DataSource, name = jdbc/Cloudscape, url = jdbc:cloudscape:rmi:Cloudscape
DB;create=true
Web service started: 9191
Web service started: 8000
Web service started: 7000
Loading jar:/E:/j2sdkee1.2.1/repository/system5/applications/TempAppn11524287495
93Server.jar
/E:/j2sdkee1.2.1/repository/system5/applications/TempAppn1152428749593Server.jar
Looking up authenticator...
Loading jar:/E:/j2sdkee1.2.1/repository/system5/applications/BookAppn11524291174
37Server.jar
/E:/j2sdkee1.2.1/repository/system5/applications/BookAppn1152429117437Server.jar
Loading jar:/e:/j2sdkee1.2.1/repository/system5/applications/BankAppn11478519333
12Server.jar
/e:/j2sdkee1.2.1/repository/system5/applications/BankAppn1147851933312Server.jar
Binding name:`java:comp/env/jdbc/AccountDB`
Created Context:/TempContext
J2EE server startup complete.
```



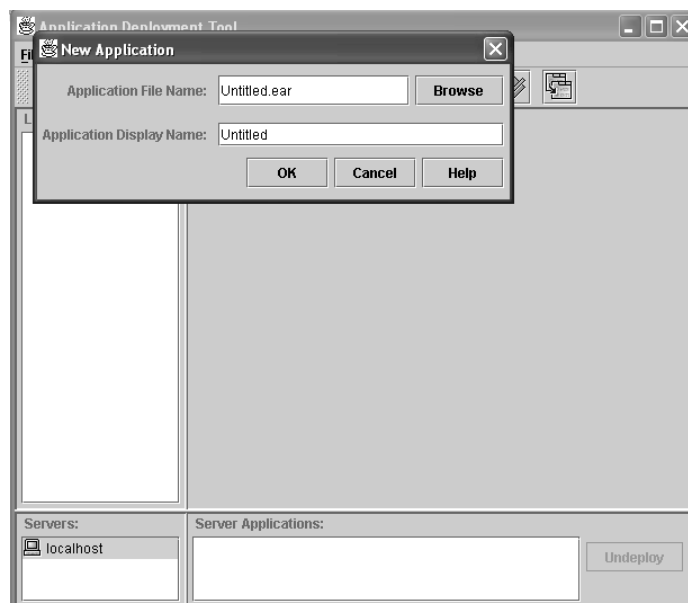
2. Type the following command to run deployment tool in the second command prompt.

```
deploytool
```

It will display Application Deployment Tool window

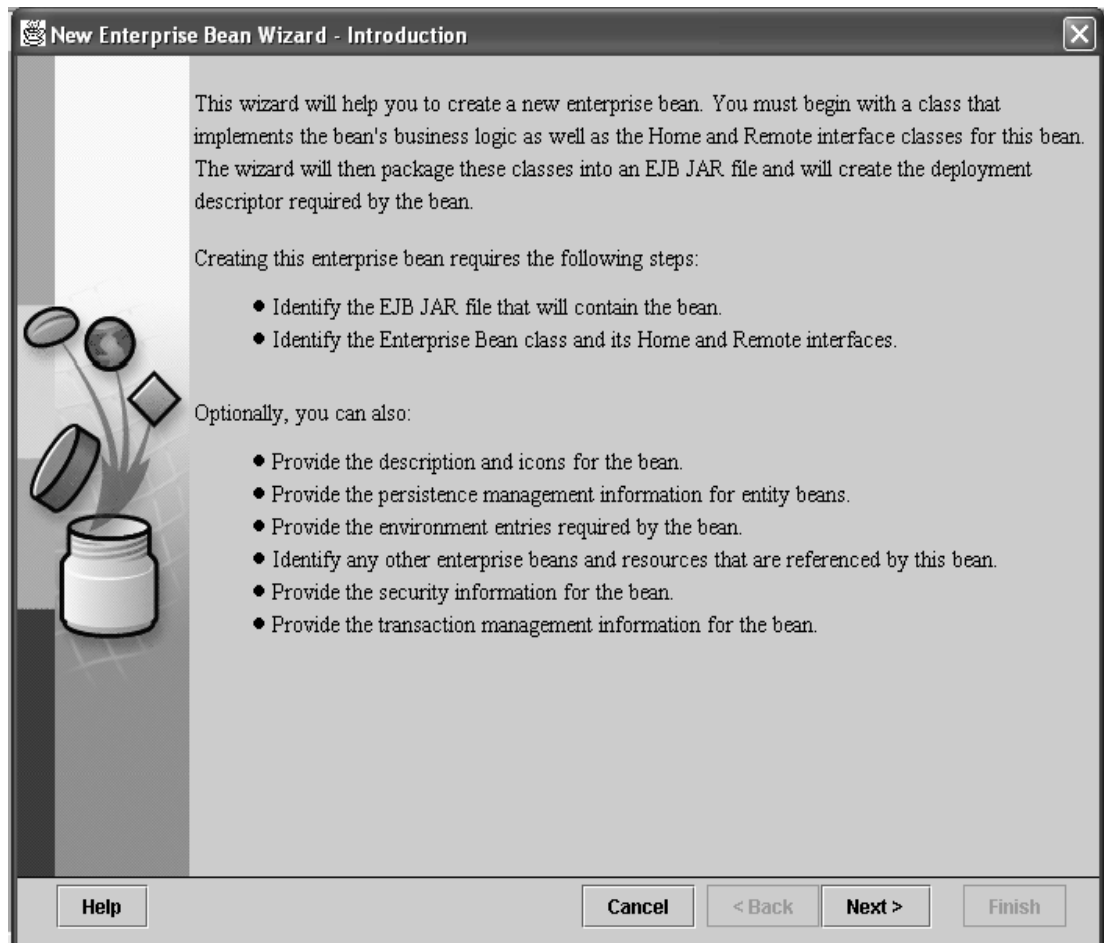


3. Select New Application option for the File menu. It will display New Application dialog box.

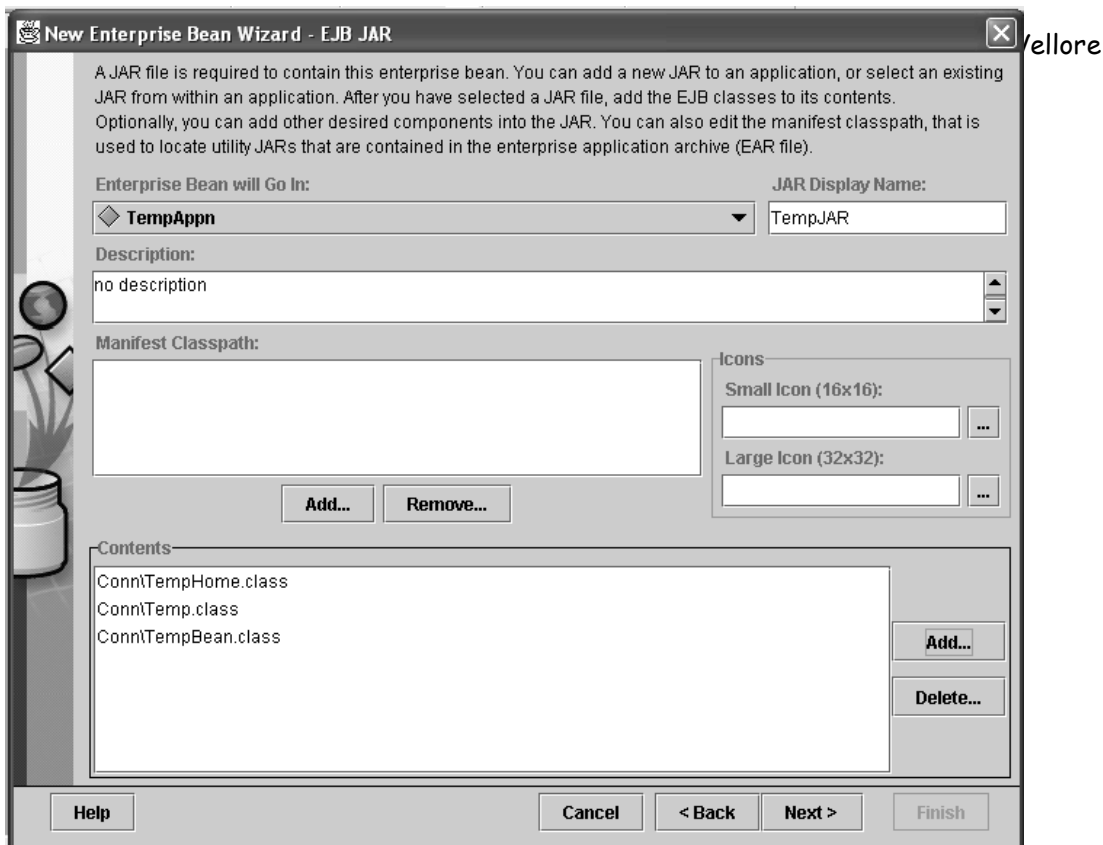


4. Click on Browse button. Select the location, where we want to create EJB Application.

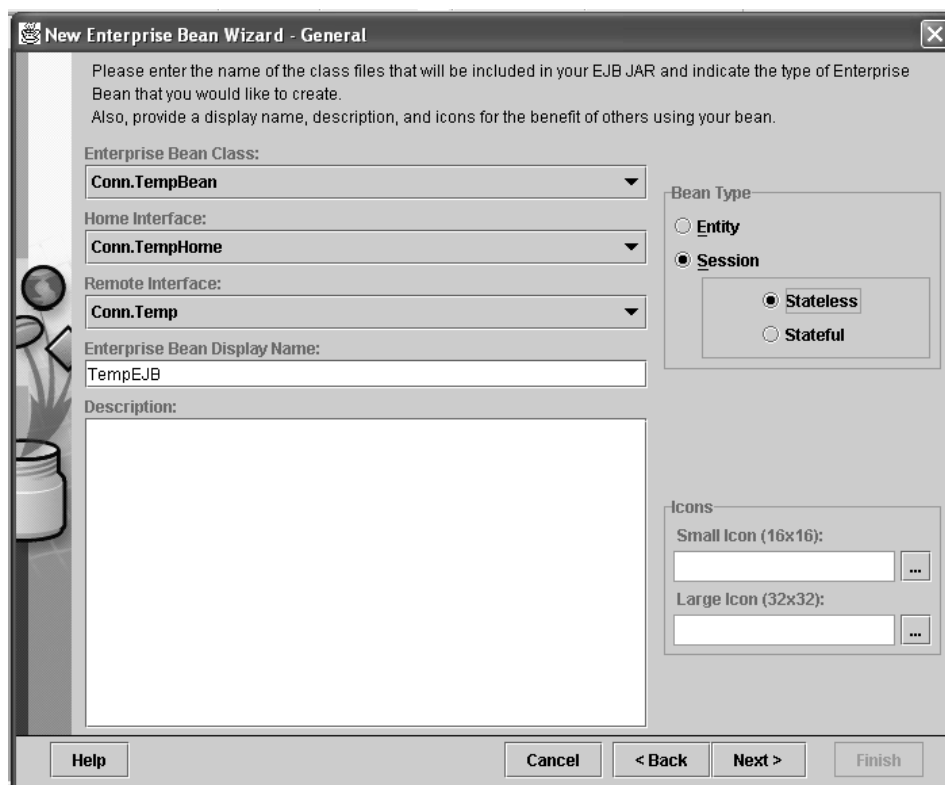
5. Type the application name (TempAppn) and click on New Application button.
6. Click on OK button.
7. Select New Enterprise Bean option from the File menu. It will display New Enterprise Bean wizard Dialog box.



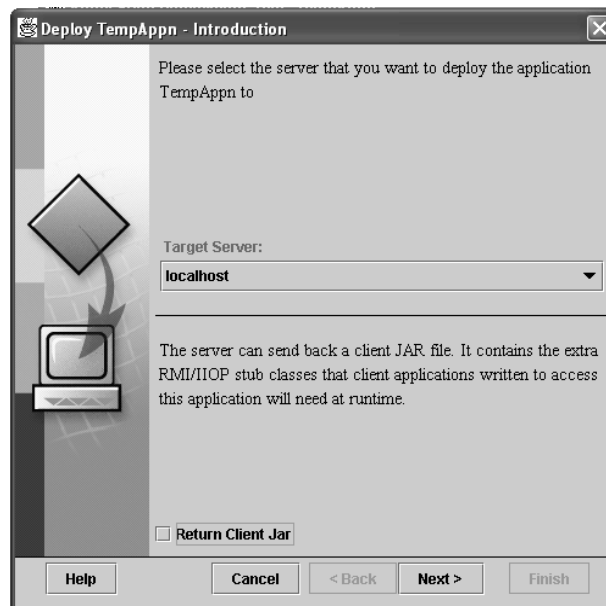
8. Click on Next button.
9. Type JAR name in the JAR display name text box (TempJAR)
10. In the context location click on Add button.
11. Add all class files and click on ok button.



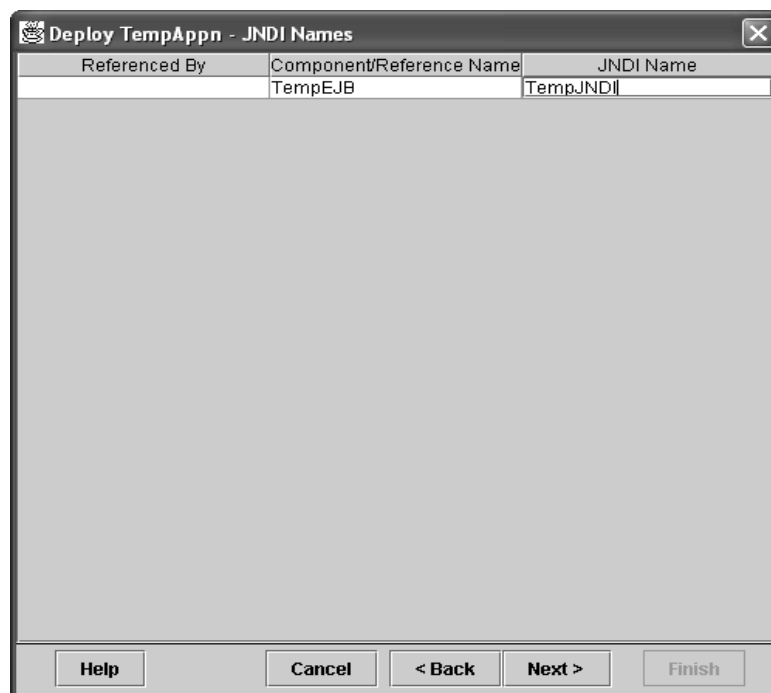
12. Click on Next button.
13. Under Bean type, select the stateless.
14. In the Enterprise Bean class combo box, select Conn.TempBean.
15. In the Home Interface combo box, select Conn.TempHome.
16. In the Remote Interface combo box, select Conn.Temp.
17. Type Bean display name in the Enterprise Bean display name text field.



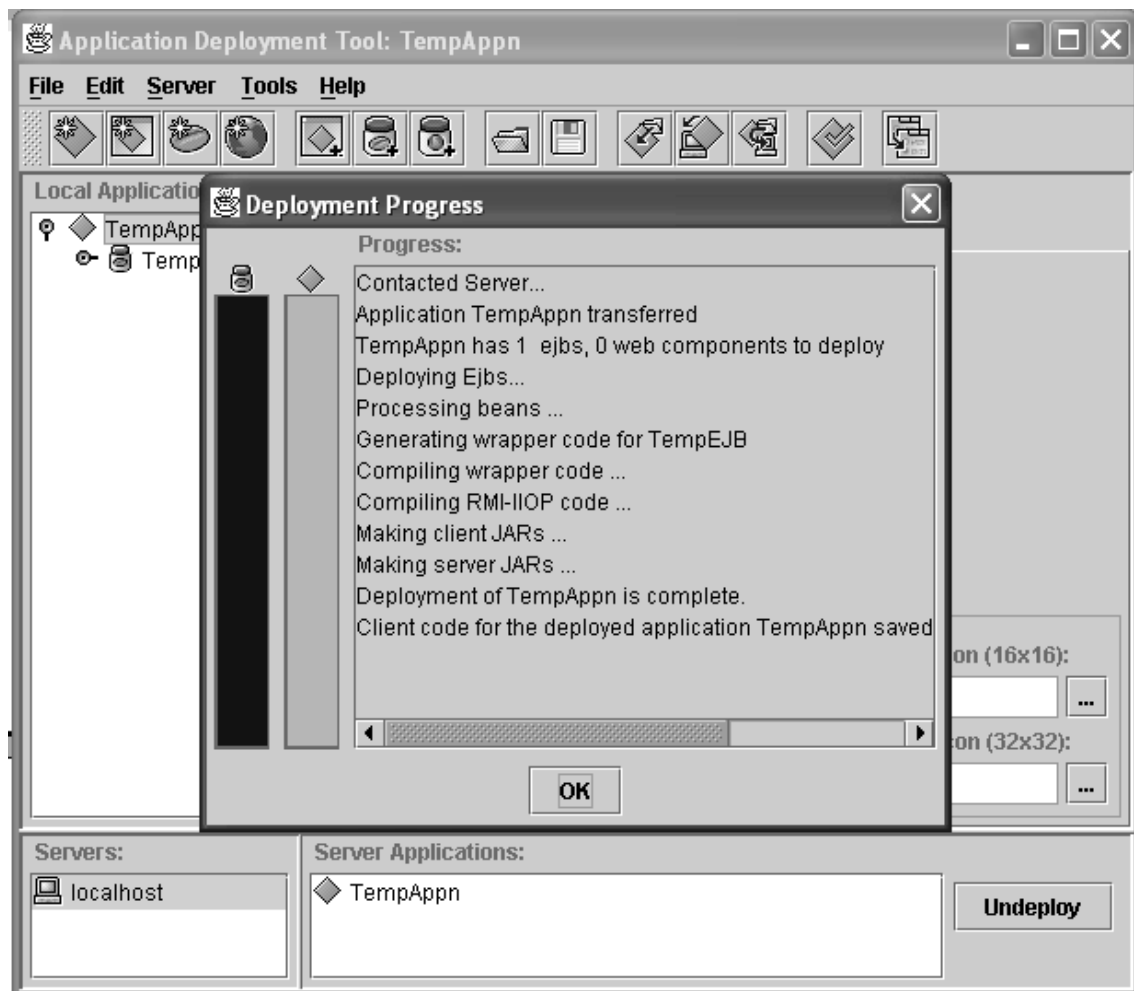
18. Click on Next button.
19. Click on Finish button.
20. Select Deploy Application option from the tools menu.



21. Select Return Client Jar check box.
22. Click on Next button.
23. Type JNDI name in the JNDI Name text box.



24. Click on Next Button.
25. Click on Finish Button. Application will be deployed.



## Creating the Client program

We can create two types of client programs

1. Application client
2. Web client.

### 1. Application client

```
import Conn.*;
import javax.swing.*;
import javax.swing.event.*;
import java.awt.*;
import java.awt.event.*;
import javax.naming.*;
import javax.rmi.*;
class Myframe extends JFrame implements ActionListener
{
    Temp conn;
    JTextField t1,t2;
    Myframe()
    {
        super("Fahrenheit to Celsius");
        JPanel jp=new JPanel();
        JLabel l1=new JLabel("Enter temperature in
fahrenheit");
        JLabel l2=new JLabel("Temperature in Celsius");
        t1=new JTextField(10);
        t2=new JTextField(10);
        JButton b=new JButton("Find");
        jp.add(l1);
        jp.add(t1);
        jp.add(l2);
        jp.add(t2);
        jp.add(b);
        getContentPane().add(jp);
        b.addActionListener(this);
        try
        {
            Context ic=new InitialContext();
            Object obj=ic.lookup("TempJNDI");
            TempHome
home=(TempHome) PortableRemoteObject.narrow(obj,TempHome
.class);
            conn=home.create();
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
    }
}
```

```

    }
    public void actionPerformed(ActionEvent ae)
    {
        String str;
        double f,c=0;
        str=t1.getText();
        f=Double.parseDouble(str);
        try
        {
            c=Math.round(conn.ftoc(f));
        }
        catch(Exception e)
        {
            System.out.println(e);
        }
        t2.setText(c+" ");
    }
}
class TempClient
{
    public static void main(String args[])
    {
        Myframe mf=new Myframe();
        mf.setSize(300,300);
        mf.setVisible(true);
    }
}

```

## Output



## Web client

The web client may be either servlet or JSP.

### Servlet as Web Client

#### Html Form

```
<html>
<head>
<title>Fahrenheit to Celcius</title>
</head>
<body bgcolor="pink" text="blue">
<form
action="http://localhost:8000/TempContext/TempAlias"
method="post">
<pre>
Enter temperature in Fahrenheit : <input type="text"
name="temp">

<center><input type="submit" value="Find"></center>
</pre>
</form>
</body>
</html>
```

#### Servlet Coding

```
import Conn.*;
import javax.servlet.*;
import javax.servlet.http.*;
import javax.naming.*;
import javax.rmi.*;
import java.io.*;
public class TempServlet extends HttpServlet
{
    public void doPost (HttpServletRequest
req,HttpServletResponse res) throws
ServletException,IOException
    {
        res.setContentType("text/html");
        PrintWriter out=res.getWriter();
        String str;
        double f,c=0;
        str=req.getParameter("temp");
        f=Double.parseDouble(str);
        try
        {
            Context ic=new InitialContext();
            Object obj=ic.lookup("TempJNDI");
```



```

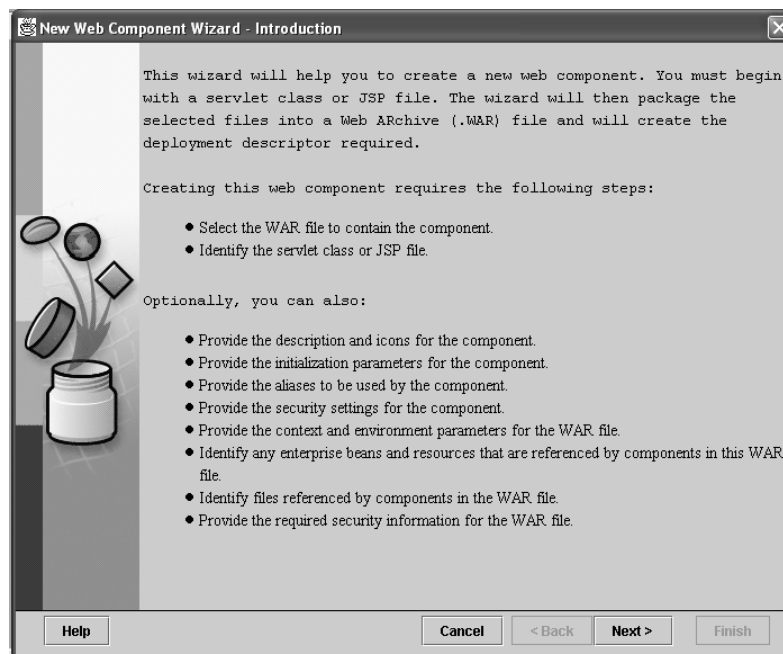
TempHome
home=(TempHome) PortableRemoteObject.narrow(obj,TempHome
.class);
Temp conn=home.create();
c=conn.ftoc(f);
}
catch(Exception e)
{
out.println(e);
}
out.println("<html><body bgcolor=lime>");
out.println("<center><h1>");
out.println("Celsius value is "+c);
out.println("</h1></center></body></html>");
}
}

```

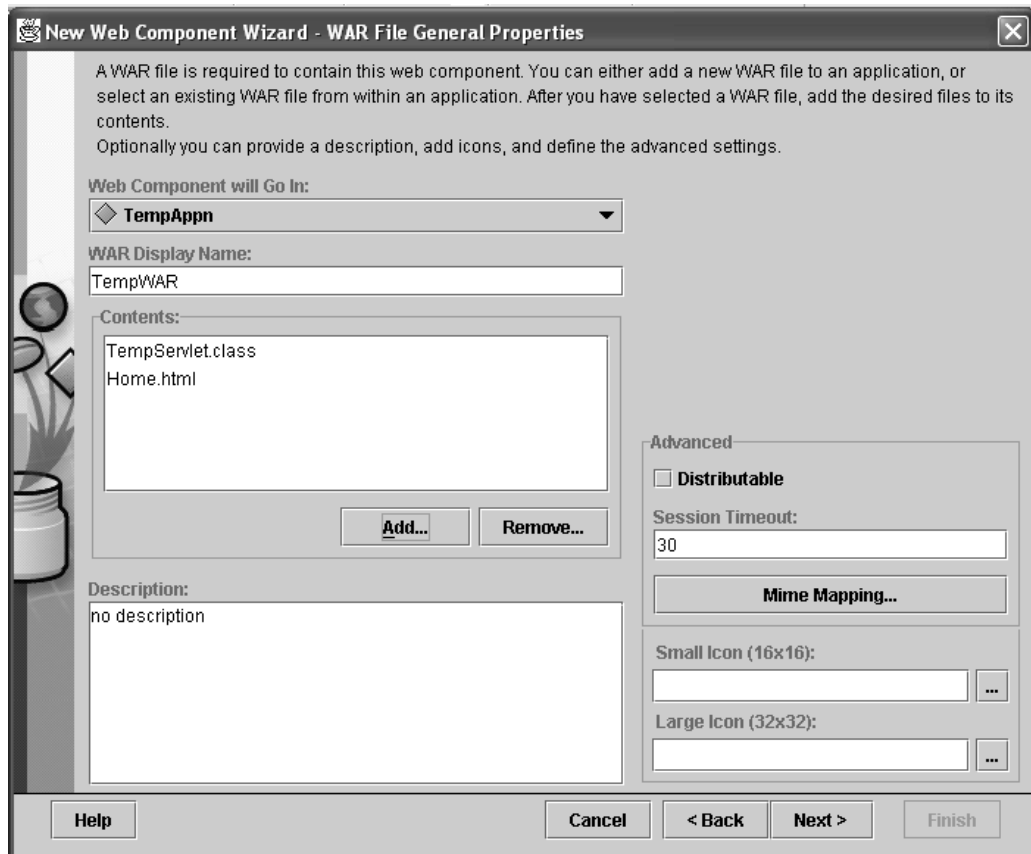
## Deployment

### Repeat step1 to step19

1. Select New Web Component option from the File menu. It will display one dialog box.

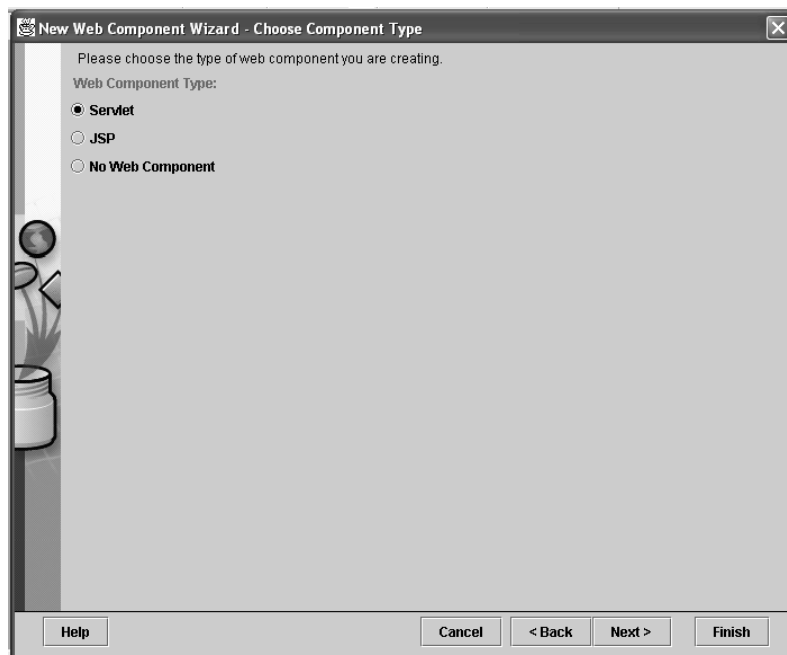


2. Click on Next button
3. Type WAR display name in the WAR display name text field.
4. Add HTML file and Servlet class file in the contents field.



5. Click on Next button.

6.



6. Select Servlet option button and click on Next button.
7. Select Servlet class name from the servlet class combo box.
8. Type the web component name in the Web Component Display Name text box and click on Next button.

**New Web Component Wizard - Component General Properties**

Please identify the JSP file or servlet class and provide its name, description, and icons. Optionally, you can define the relative position in which this component will be loaded on startup.

Servlet Class:

Web Component Display Name:

Startup Load Sequence Position:

Description:

Icons  
 Small Icon (16x16):  ...  
 Large Icon (32x32):  ...

Buttons: Help, Cancel, < Back, Next >, Finish

9. In the Alias dialog box click on Add button and type the alias name. and click on Next button.

**New Web Component Wizard - Component Aliases**

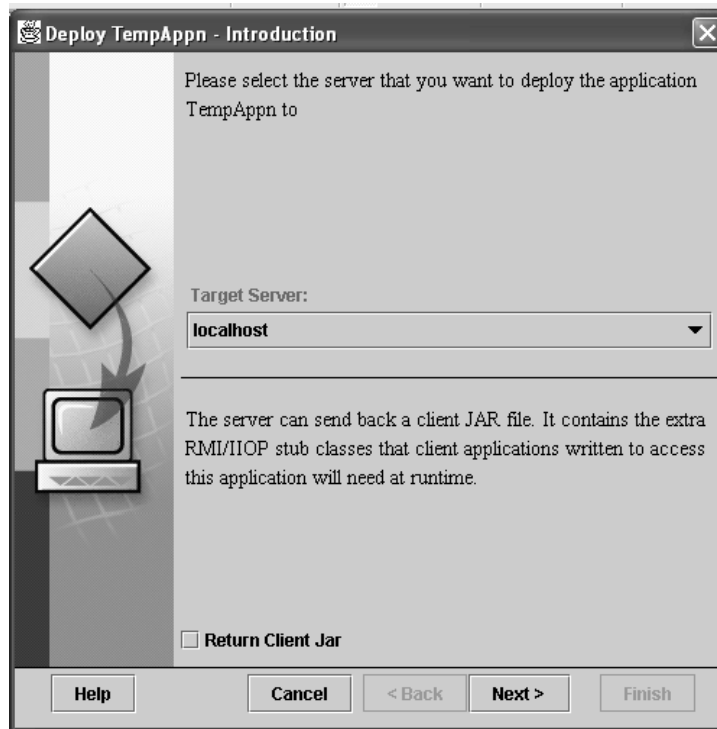
Please provide the aliases for this web component. Calls to these aliases will be redirected to an instance of the web component.

Aliases

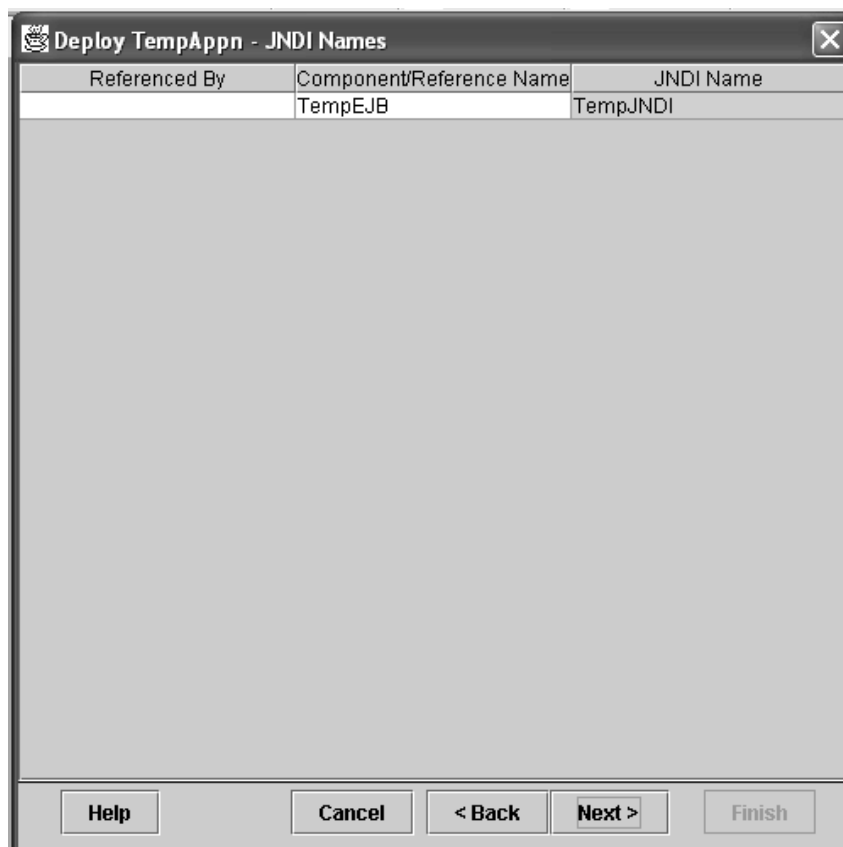
Buttons: Add, Delete...

Buttons: Help, Cancel, < Back, Next >, Finish

10. Click on Finish button.
11. Select Deploy Application option from the Tools menu. It will display one dialog box.



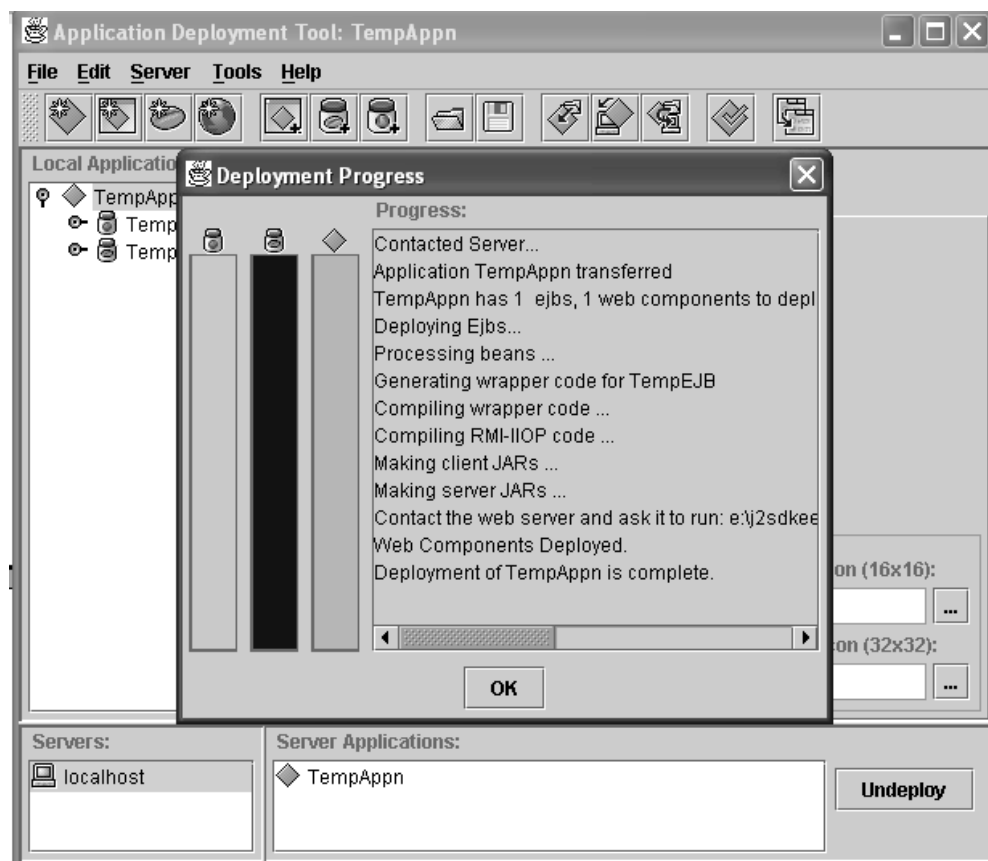
12. Click on Next button.
13. Type JNDI name in the JNDI name text field and click on Next button.



14. Type context name in the Context Root text field (TempContext) and click on Next button.



15. Click on Finish button.

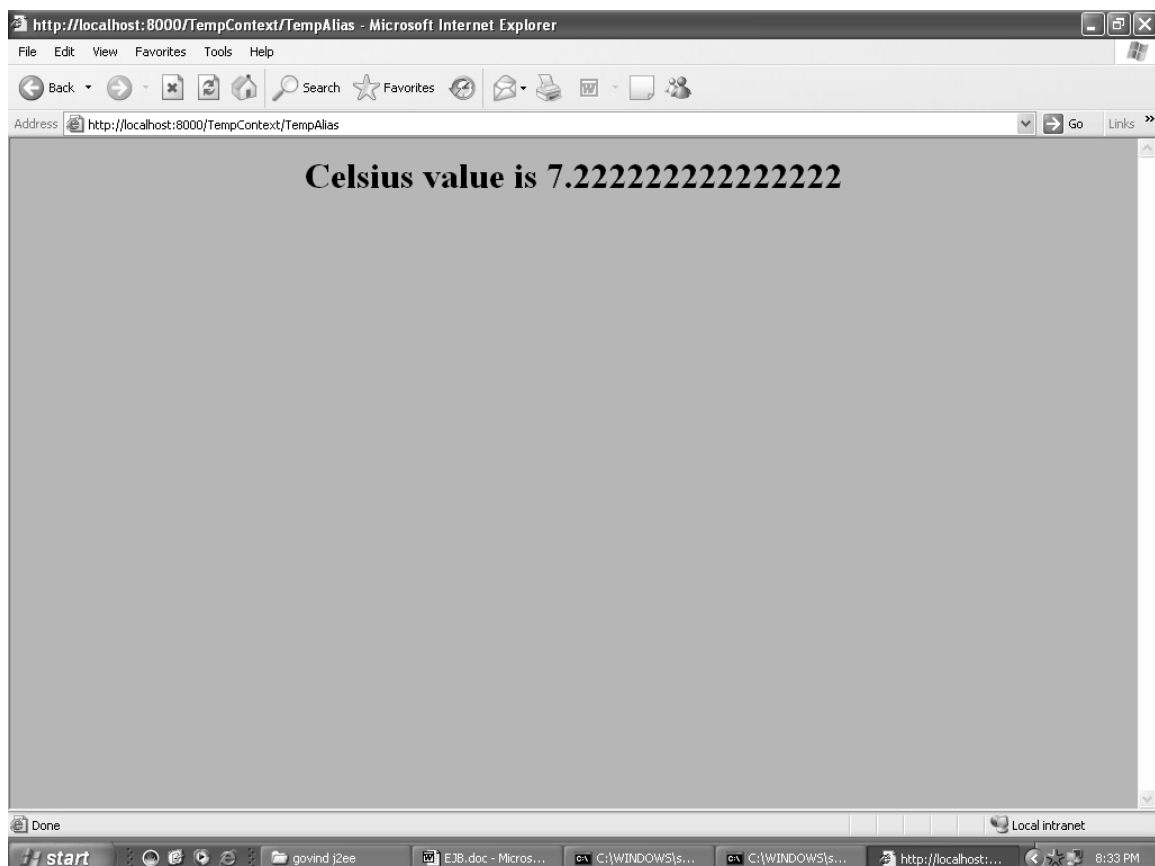
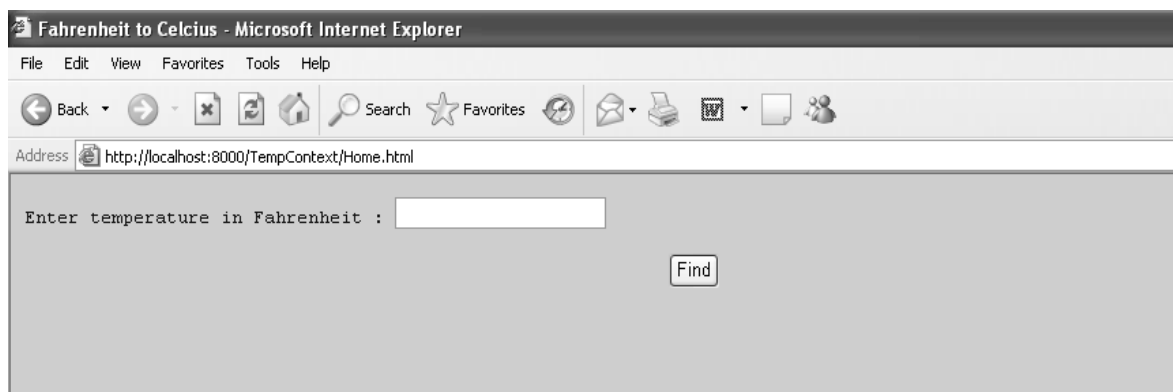


## Executing Web client

1. Open Internet Explorer
2. Type the following line in the address bar.

`http://localhost:8000/TempContext`

3. Click on Home.html
4. Type Fahrenheit value in the text field and click on Find button.



**JSP as Web client**

```

<%@ page import="Conn.*" %>
<%@ page import="javax.naming.*" %>
<%@ page import="javax.rmi.*" %>
<html>
<head>
<title>Fahrenheit to Celsius</title>
</head>
<body bgcolor="red" text="yellow">
<form method="get">
<pre>
Enter Temperature in Fahrenheit :<input type="text"
name="temp">

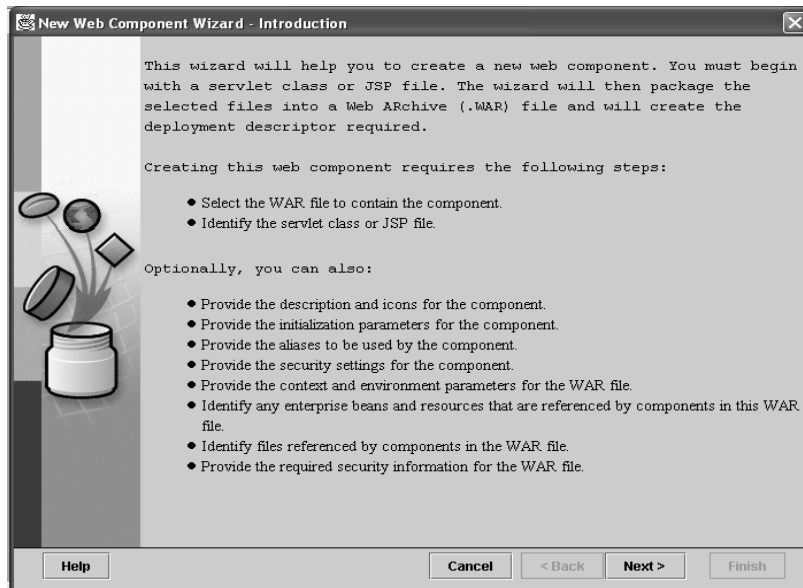
<center><input type="submit" value="Find"></center>
</pre>
</form>
<%! String str; %>
<%! double f,c=0; %>
<%
    try
    {
        Context ic=new InitialContext();
        Object obj=ic.lookup("TempJNDI");
        TempHome
home=(TempHome) PortableRemoteObject.narrow(obj,TempHome
.class);
        Temp conn=home.create();
        str=request.getParameter("temp");
        f=Double.parseDouble(str);
        c=conn.ftoc(f);
    }
    catch(Exception e)
    {
        out.println(e);
    }
    out.println("<center><h1>");
    out.println("Celsius value =" +c);
    out.println("</h1></center>");
%>
</body>
</html>

```

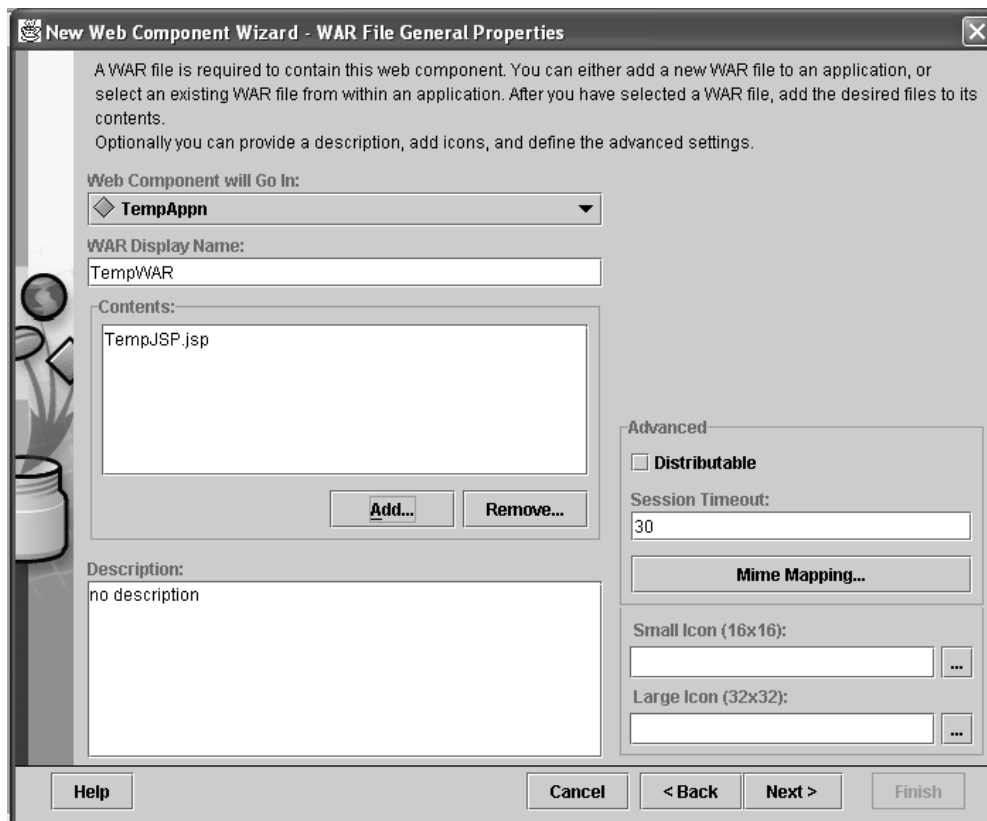
## Deployment

### Repeat step 1 to step 19

1. Select Web Component option from the File menu. It will display one dialog box.

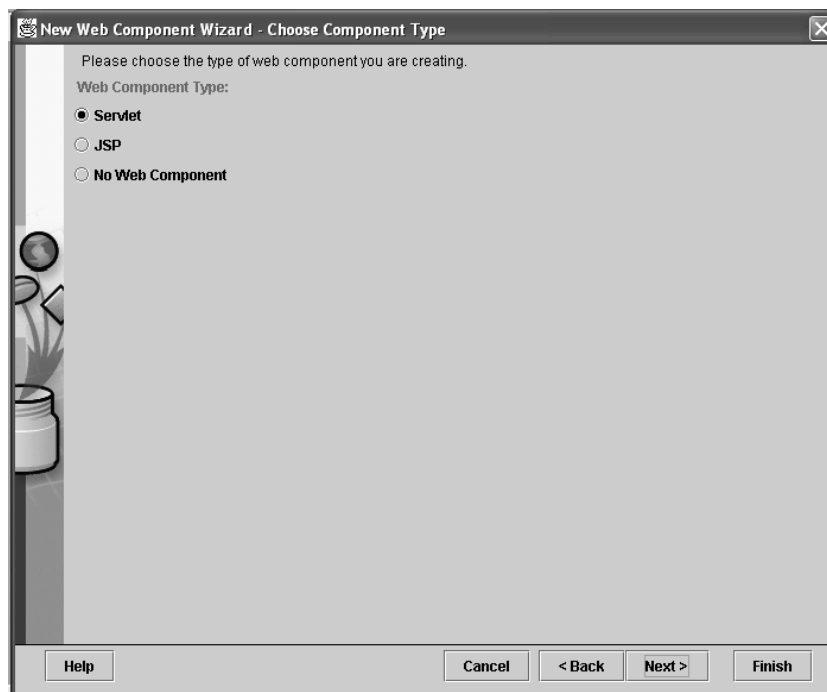


2. Click on Next button.
3. Select application name from the Web component combo box.
4. Type the War name in the WAR display name text field.
5. Add JSP file in the contents box.

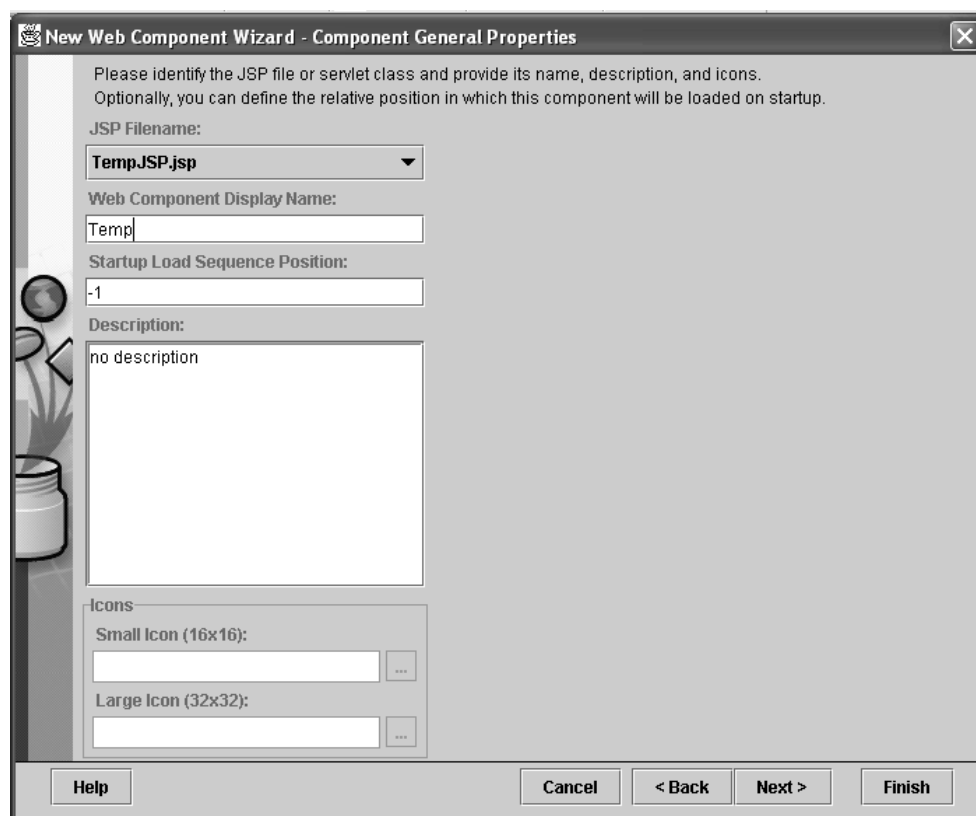




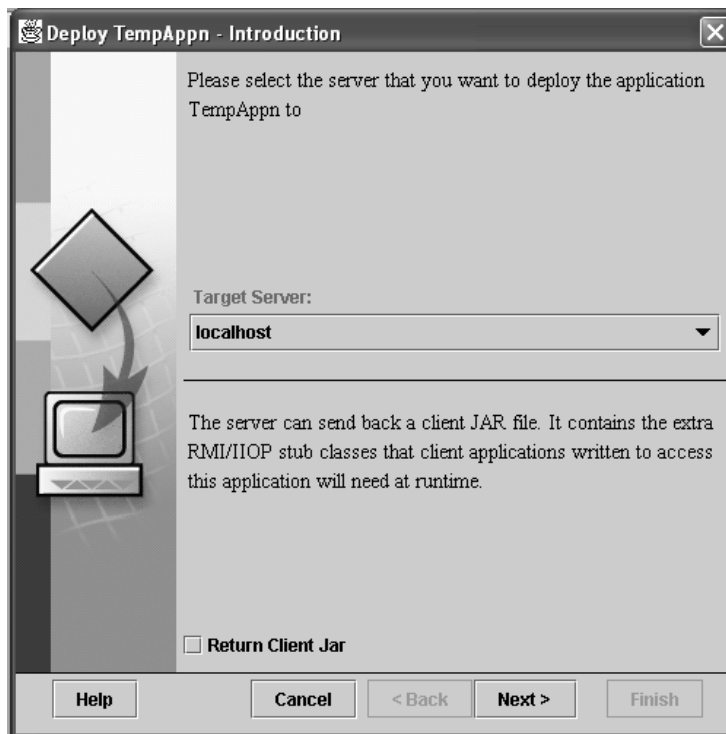
6. Click on Next button.
7. Select JSP option button and click on Next button.



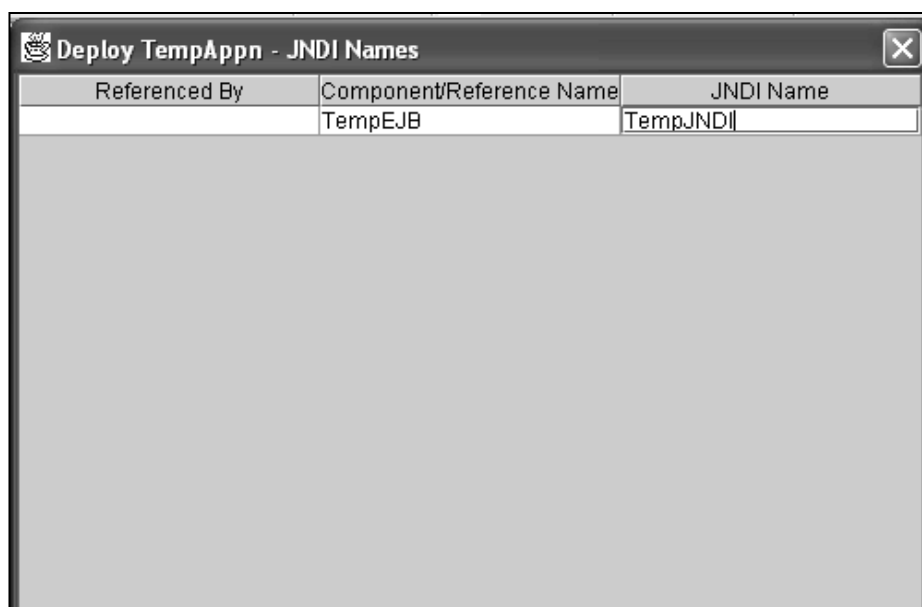
8. Select JSP file name in the JSP file name combo box.
9. Type the web component name in the Web Component Display name text field and click on Next button.



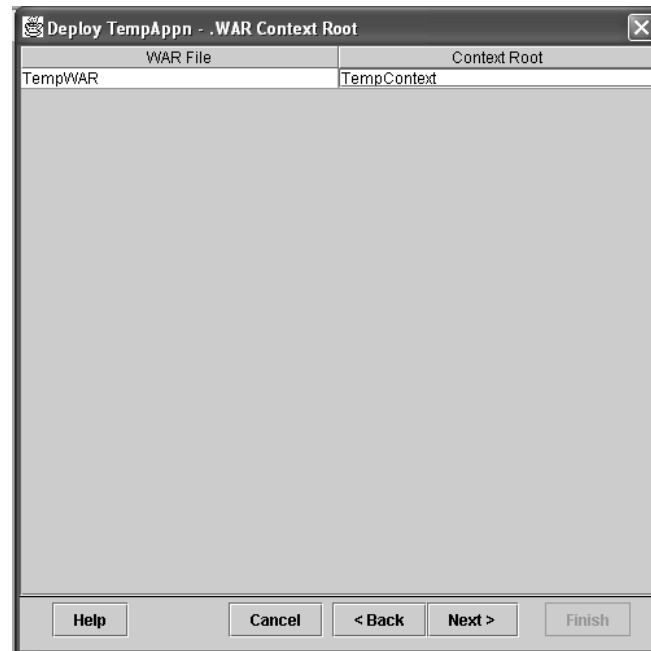
10. Click on Finish button.
11. Select Deploy Application option from the tools menu.



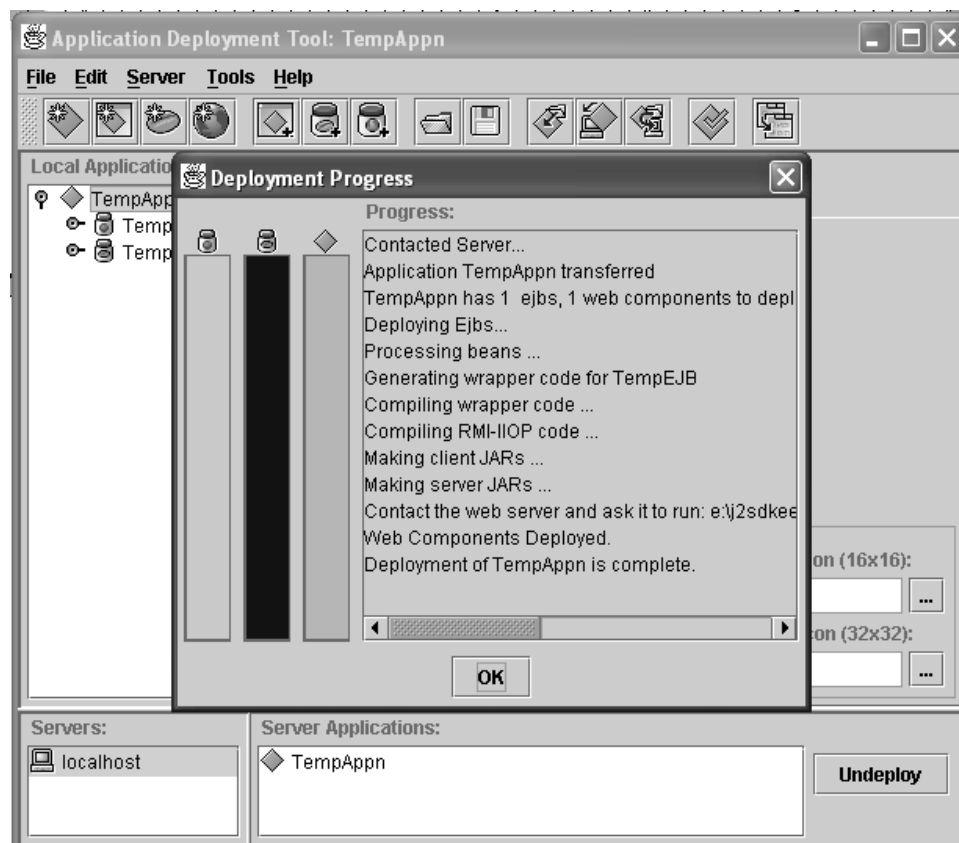
12. Click on Next button.
13. Type JNDI name in the JNDI name text field and click on Next button.



14. Type Context name in the Context Root text field and click on Next button.



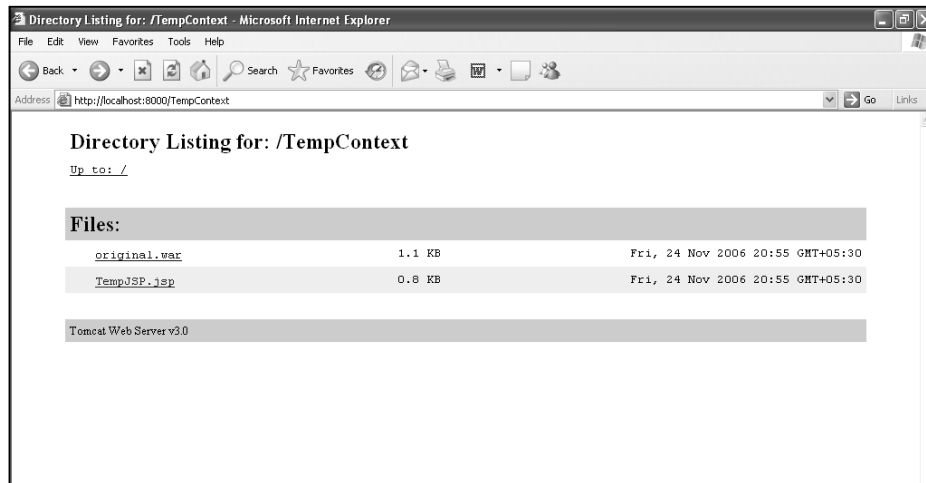
15. Click on Finish button.



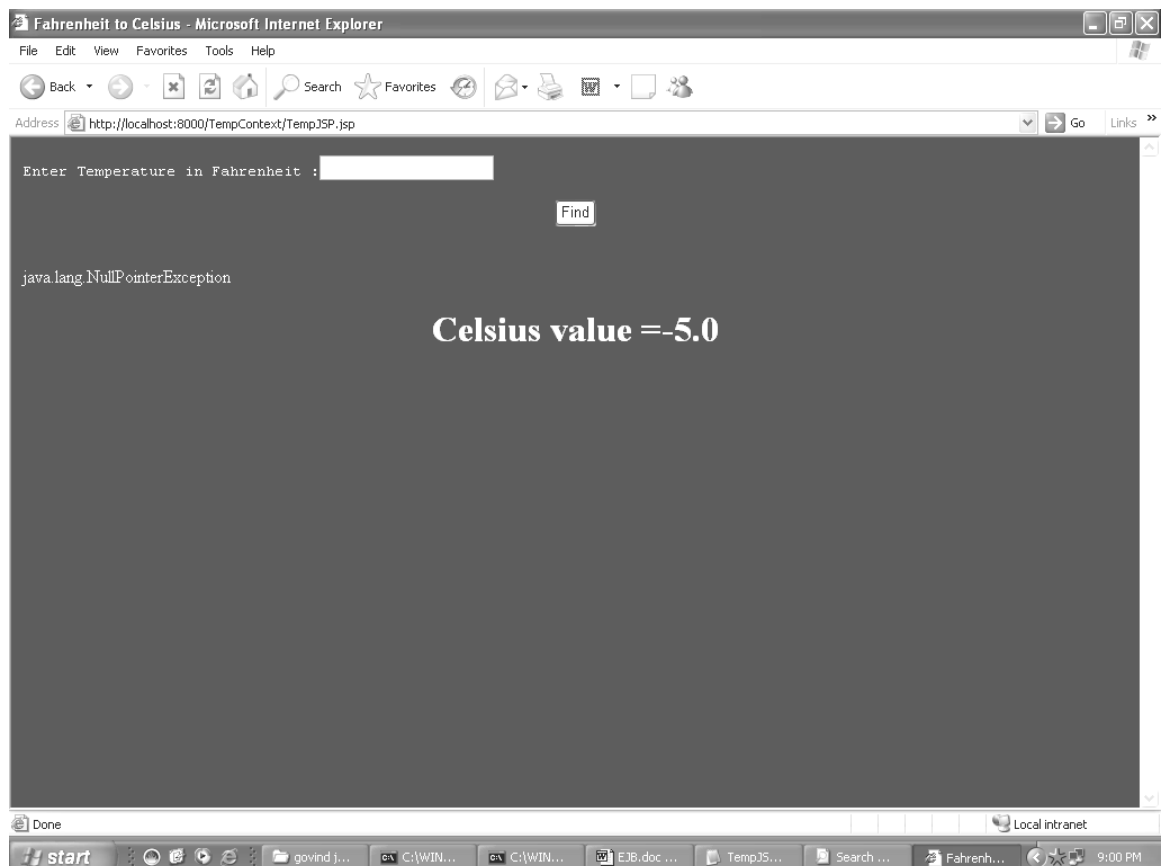
## Executing JSP web client

1. Open Internet Explorer.
2. Type the following line in the address tool bar.

`http://localhost:8000/TempContext`



3. Click on TempJSP.jsp



## Stateful Session Beans

The Stateful session bean is considered if any of the following conditions are true.

1. The bean's state must be initialized when it is created.
2. The bean needs to hold information about the client across method invocations.

### Case study

A. K. Brothers & Company is an organization involved in selling books online. The company has developed an e-commerce web site to sell the books. For this site, they need shopping cart to store the books selected by the customers. They want to develop the shopping cart as a EJB bean. This shopping cart bean has to be developed by using a session bean. The bean's client may add a book to the cart, remove a book, or retrieve the cart's contents.

### Creating programs

#### Remote Interface (Book.java)

```
package stateful;
import java.rmi.*;
import javax.ejb.*;
import java.util.*;
public interface Book extends EJBObject
{
    public void addBook(String title) throws
RemoteException;
    public void removeBook(String title) throws
RemoteException;
    public Vector contents() throws RemoteException;
}
```

#### Home Interface (BookHome.java)

```
package stateful;
import java.rmi.*;
import javax.ejb.*;
import java.io.Serializable;
public interface BookHome extends EJBHome
{
    Book create(String name) throws
RemoteException,CreateException;
}
```

## The Enterprise Bean class (BookBean.java)

```
package stateful;
import java.rmi.*;
import javax.ejb.*;
import java.util.*;
public class BookBean implements SessionBean
{
    String cname;
    Vector book;
    public void ejbCreate(String name) throws
CreateException
    {
        cname=name;
        book=new Vector();
    }
    public BookBean(){}
    public void ejbRemove(){}
    public void ejbActivate(){}
    public void ejbPassivate(){}
    public void setSessionContext(SessionContext sc){}
    public void addBook(String title)
    {
        book.addElement(title);
    }
    public void removeBook(String title)
    {
        book.removeElement(title);
    }
    public Vector contents()
    {
        return(book);
    }
}
```

**Client Program (BookClient.java)**

```

import stateful.*;
import javax.rmi.*;
import javax.naming.*;
import java.util.*;
import java.io.*;
class BookClient
{
    public static void main(String args[]) throws
IOException
    {
        try
        {
            DataInputStream inn=new DataInputStream(System.in);
            int choice;
            String str,title;
            Vector content;
            Context ic=new InitialContext();
            Object obj=ic.lookup("BookJNDI");
            BookHome
home=(BookHome) PortableRemoteObject.narrow(obj,BookHome
.class);
            System.out.print("Enter customer name:");
            String name=inn.readLine();
            Book conn=home.create(name);
            do
            {
                System.out.println();
                System.out.println("1. Add Book");
                System.out.println("2. Remove Book");
                System.out.println("3. Display");
                System.out.println("4. Exit");
                System.out.println("Enter your choice");
                str=inn.readLine();
                choice=Integer.parseInt(str);
                switch(choice)
                {
                    case 1:
                        System.out.println("Enter book title");
                        title=inn.readLine();
                        conn.addBook(title);
                        break;
                    case 2:
                        System.out.println("Enter book title to be
deleted:");
                        title=inn.readLine();
                        conn.removeBook(title);
                        break;
                    case 3:

```

```
        content=conn.contents();  
        System.out.println(content);  
        break;  
    }  
    }while(choice!=4);  
}  
catch(Exception e)  
{  
    System.out.println("Error"+e);  
}  
}  
}
```

## Entity Bean

An entity bean represents an entity kept in a persistent storage mechanism, usually a database. A business application might use a database to store business entity objects such as accounts, customers, orders, and products. Inside the J2EE server, this application would represent the business entity objects with entity beans. Entity beans are persistent, allow shared access, and have primary keys while session beans are non persistent, not sharable and do not have primary key.

There are two types of Entity Bean

1. **Bean Managed Entity Beans**
2. **Container Managed Entity Beans**