Mahatma Gandhi Mission's

Jawaharlal Nehru Engineering College

Laboratory Manual

ADVANCED JAVA

For

Third Year CSE Students
Dept: Computer Science & Engineering (NBA Accredited)

FOREWORD

It is my great pleasure to present this laboratory manual for Third Year engineering students for the subject of Advanced Java keeping in view the vast coverage required for Programming with Java Language

As a student, many of you may be wondering with some of the questions in your mind regarding the subject and exactly what has been tried is to answer through this manual.

As you may be aware that MGM has already been awarded with ISO 9000 certification and it is our endure to technically equip our students taking the advantage of the procedural aspects of ISO 9000 Certification.

Faculty members are also advised that covering these aspects in initial stage itself, will greatly relived them in future as much of the load will be taken care by the enthusiasm energies of the students once they are conceptually clear.

Dr. S. D. Deshmukh
Principal

LABORATORY MANUAL CONTENTS

This manual is intended for the Third Year students of Computer Science and Engineering in the subject of "Advanced Java". This manual typically contains practical/Lab Sessions related Advanced Java covering various aspects related the subject to enhanced understanding.

As per the syllabus along with Study of Advanced Java, we have made the efforts to cover various aspects of Advanced Java covering different Techniques used to construct and understand concepts of Advanced Java Programming.

Students are advised to thoroughly go through this manual rather than only topics mentioned in the syllabus as practical aspects are the key to understanding and conceptual visualization of theoretical aspects covered in the books.

Good Luck for your Enjoyable Laboratory Sessions

Dr. V. B. Musande	Mr. P.S.Maniyar & Mr.C.M.Mane
HOD,	Asst. Professor,
Computer Science & Engg Dept.	Computer Science & Engg Dept.

MGM's

Jawaharlal Nehru Engineering College, Aurangabad

Department of Computer Science and Engineering

Vision of CSE Department:

To develop computer engineers with necessary analytical ability and human values who can creatively design, implement a wide spectrum of computer systems for welfare of the society.

Mission of the CSE Department:

I. Preparing graduates to work on multidisciplinary platforms associated with their professional position both independently and in a team environment.

II. Preparing graduates for higher education and research in computer science and engineering enabling them to develop systems for society development.

Programme Educational Objectives:

Graduates will be able to

I. To analyze, design and provide optimal solution for Computer Science & Engineering and multidisciplinary problems.

II. To pursue higher studies and research by applying knowledge of mathematics and fundamentals of computer science.

III. To exhibit professionalism, communication skills and adapt to current trends by engaging in lifelong learning.

Programme Outcomes (POs):

Engineering Graduates will be able to:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

DOsandDON'TsinLaboratory:

1. Make entry in the Log Book as soon as you enter the Laboratory.

2. All the students should sit according to their roll numbers starting from their left to right.

3. All the students are supposed to enter the terminal number in the log book.

4. Do not change the terminal on which you are working.

5. All the students are expected to get at least the algorithm of the program/concept to be implemented.

6. Strictly observe the instructions given by the teacher/Lab Instructor.

InstructionforLaboratoryTeachers::

1. Submission related to whatever lab work has been completed should be done during the next lab session. The immediate arrangements for printouts related to submission on the day of practical assignments.

2. Students should be taught for taking the printouts under the observation of lab teacher.

3. The promptness of submission should be encouraged by way of marking and evaluation patterns that will benefit the sincere students.

SUBJECT INDEX

HARDWARE REQUIRED

P-IV/CORE i3 PROCESSOR

HDD 40GB

RAM 512 OR ABOVE

SOFTWARE REQUIRED

Windows XP/7

Java Enterprise Edition 6

Web Server (Apache Tomcat)

Eclipse OR Net Beans IDE

1. Program to perform database operation

Aim : Write Program to perform database operation.

Theory:

JDBC

Call-level interfaces such as JDBC are programming interfaces allowing external access to SQL database manipulation and update commands. They allow the integration of SQL calls into a general programming environment by providing library routine which interface with the database. In particular, java based JDBC has a rich collection of routines which make such an interface extremely simple and intuitive.

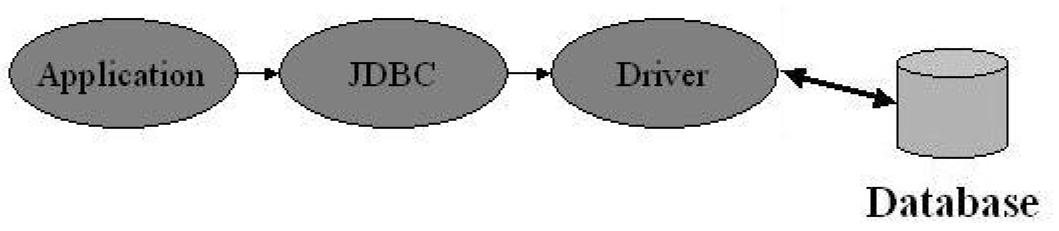
What happens in a call level interface: you are writing a normal java program .Somewhere in a program, you need to interact with a database. Using standard library routines, you open a connection with database. you then use JDBS to send your SQL code to the database, and process the result that are returned. When you are done, you close the connection.

A file system does not work well for data storage applications for e.g. Business applications so we required power of database. Java provide file system access with database connectivity called JDBC.

The JDBC (Java Database Connectivity) API defines interfaces and classes for writing database applications in Java by making database connections. Using JDBC, you can send SQL, PL/SQL statements to almost any relational database. JDBC is a Java API for executing SQL statements and supports basic SQL functionality. It provides RDBMS access by allowing you to embed SQL inside Java code. Because Java can run on a thin client, applets embedded in Web pages can contain downloadable JDBC code to enable remote database access.

We can also create a table, insert values into it, query the table, retrieve results, and update the table with the help of a JDBC Program example. Although JDBC was designed specifically to provide a Java interface to relational databases, we may find that you need to write Java code to access non-relational databases as well.

JDBC Architecture



Java application calls the JDBC library. JDBC loads a driver that talks to the database. We can change database engines without changing database code.

JDBC Basics - Java Database Connectivity Steps

Before you can create a java jdbc connection to the database, you must first import the java.sql package.

import java.sql.*; The star (*) indicates that all of the classes in the package java.sql are to be imported.

1. Loading a database driver,

In this step of the jdbc connection process, we load the driver class by calling Class.forName() with the Driver class name as an argument. Once loaded, the Driver class creates an instance of itself. A client can connect to Database Server through JDBC Driver. Since most of the Database servers support ODBC driver therefore JDBC-ODBC Bridge driver is commonly used.

The return type of the Class.forName (String ClassName) method is “Class”. java.lang package.

Syntax :

```
try  {
          Class.forName(”sun.jdbc.odbc.JdbcOdbcDriver”); //Or any other
driver
}
catch(Exception x){
          System.out.println( “Unable to load the driver class!” );
```

2.Creating aoracle jdbc Connection

The JDBC DriverManager class defines objects which can connect Java applications to a JDBC driver. DriverManager is considered the backbone of JDBC architecture. DriverManager class manages the JDBC drivers that are installed on the system. Its getConnection() method is used to establish a connection to a database. It uses a username, password, and a jdbc url to establish a connection to the database and returns a connection object. A jdbc Connection represents a session/connection with a specific database. Within the context of a Connection, SQL, PL/SQL statements are executed and results are returned. An application can have one or more connections with a single database, or it can have many connections with different databases. A Connection object provides metadata i.e. information about the database, tables, and fields. It also contains methods to deal with transactions.

3. Creating a jdbc Statement object,

Once a connection is obtained we can interact with the database. Connection interface defines methods for interacting with the database via the established connection. To execute SQL statements, you need to instantiate a Statement object from your connection object by using the createStatement() method.

Statement statement = dbConnection.createStatement();

A statement object is used to send and execute SQL statements to a database.

4. Executing a SQL statement with the Statement object, and returning a jdbc resultSet.

Statement interface defines methods that are used to interact with database via the execution of SQL statements. The Statement class has three methods executeQuery(), executeUpdate(), and execute().

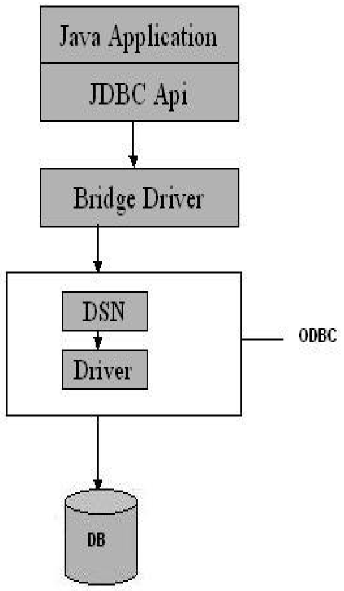
JDBC drivers Types:

JDBC drivers are divided into four types or levels. The **different types of jdbc drivers** are:

Type 1: JDBC-ODBC Bridge driver (Bridge) **Type 2:** Native-API/partly Java driver (Native) **Type 3:** All Java/Net-protocol driver (Middleware) **Type 4:** All Java/Native-protocol driver (Pure)

Type 1 JDBC Driver: **JDBC-O DBC Bridge driver**

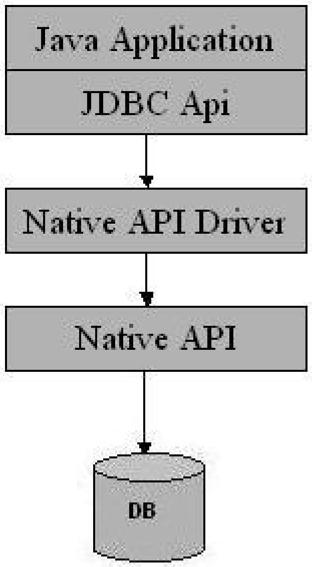
The Type 1 driver translates all JDBC calls into ODBC calls and sends them to the ODBC driver. ODBC is a generic API. The JDBC-ODBC Bridge



Type 1: JDBC-ODBC Bridge

Nativ e -AP I /p ar tly Jav a d ri ve r

The distinctive characteristic of type 2 jdbc drivers are that Type 2 drivers convert JDBC calls into database-specific calls i.e. this driver is specific to a particular database. Some distinctive characteristic of type 2 jdbc drivers are shown below. Example: Oracle will have oracle native api.



Type 2: Native api/ Partly Java Driver

E.g.:

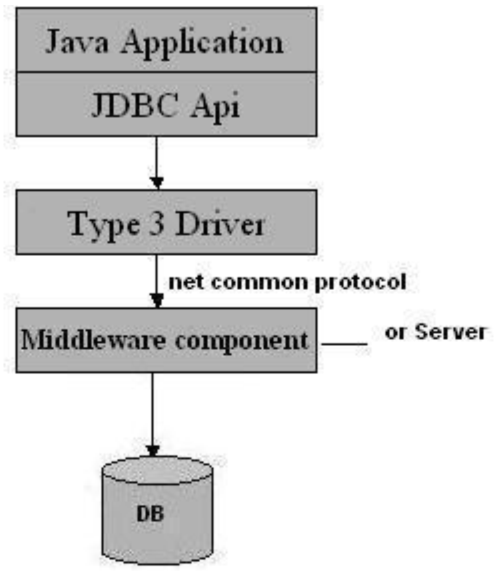
Orders Table:

```
CREATE TABLE Orders (
Prod_ID INTEGER,
ProductName VARCHAR(20),
Employee_ID INTEGER
);
```

Type3JDBCDriver

A ll Java/N et-protoc ol driver

Type 3 database requests are passed through the network to the middle-tier server. The middle-tier then translates the request to the database. If the middle-tier server can in turn use Type1, Type 2 or Type 4 drivers.

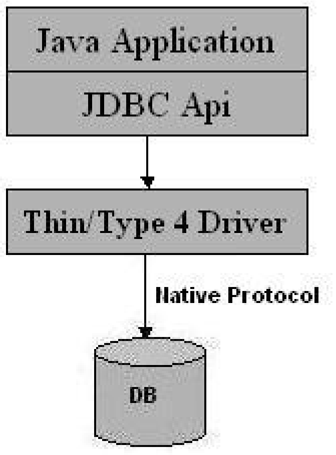


Type 3: All Java/ Net-Protocol Driver

Type4JDBCDriver

Native-protocol/all-Javadriver

The Type 4 uses java networking libraries to communicate directly with the database server.



Type 4: Native-protocol/all-Java driver

Advantage

The JDBC-ODBC Bridge allows access to almost any database, since the database's ODBC drivers are already available.

Disadvantages

1. Since the Bridge driver is not written fully in Java, Type 1 drivers are not portable.

2. A performance issue is seen as a JDBC call goes through the bridge to the ODBC driver, then to the database, and this applies even in the reverse process. They are the slowest of all driver types.

3. The client system requires the ODBC Installation to use the driver.

4. Not good for the Web.

A lgor ithm :

1)Cerate a Table in Ms Access,save it in data base.

2)write java program, Import java Packages in program.

3) Define class establish ODBC connection with the help of java code.

4) Insert record in to database with the help of SQL queries.

5) Execute program, see the Op on console prompt.

The method Class.forName(String) is used to load the JDBC driver class. The line below causes the JDBC driver from *some jdbc vendor* to be loaded into the application. (Some JVMs also require the class to be instantiated with .newInstance().)

```
Class.forName( "com.somejdbcvendor.TheirJdbcDriver" );
```

When a Driver class is loaded, it creates an instance of itself and registers it with the DriverManager. This can be done by including the needed code in the driver class's static block. e.g. DriverManager.registerDriver(Driver driver)

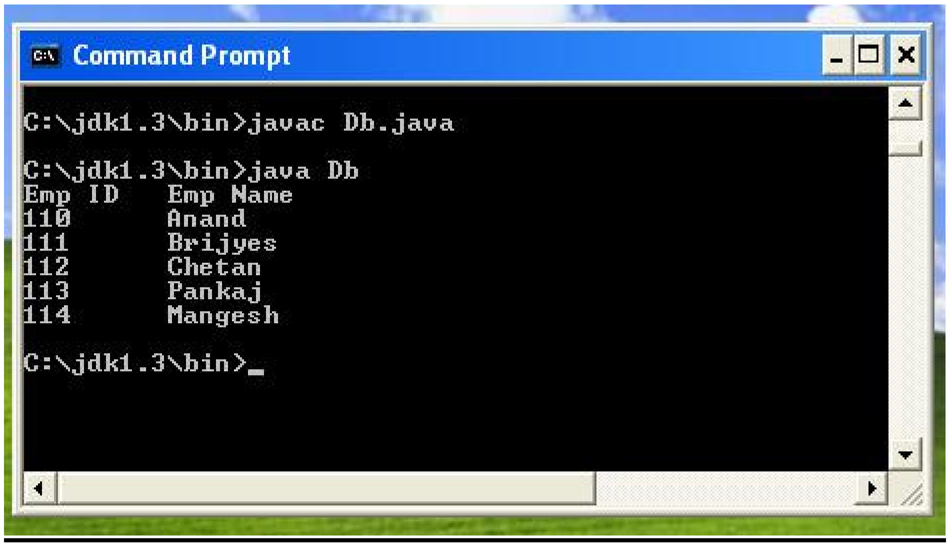
Now when a connection is needed, one of the DriverManager.getConnection() methods is used to create a JDBC connection.

```
Connection conn = DriverManager.getConnection(
    "jdbc:somejdbcvendor:other data needed by some jdbc vendor",
    "myLogin",
    "myPassword" );
```

The URL used is dependent upon the particular JDBC driver. It will always begin with the "jdbc:" protocol, but the rest is up to the particular vendor. Once a connection is established, a statement must be created.

```
Statement stmt = conn.createStatement();
try {
   stmt.executeUpdate( "INSERT INTO MyTable( name ) VALUES ( 'my
name' ) " );
} finally {
   //It's important to close the statement when you are done with it
   stmt.close();
}
```

Outpu t:



```
C:\jdk1.3\bin>javac Db.java

C:\jdk1.3\bin>java Db
Emp ID   Emp Name
110      Anand
111      Brijyes
112      Chetan
113      Pankaj
114      Mangesh

C:\jdk1.3\bin>_
```

Conclus ion: Hence, we learnt how to perform database operation.

2. Basic Servlet Program

Aim : Write a program to demonstrate Basic Servlet.

Theory:

As we know that the servlet extends the **HttpServlet** and overrides the *doGet ()* method which it inherits from the HttpServlet class. The server invokes doGet () method whenever web server receives the GET request from the servlet. The doGet() method takes two arguments first is HttpServletRequest object and the second one is HttpServletResponse object and this method throws the ServletException.

Whenever the user sends the request to the server then server generates two objects' first is **HttpServletRequest** object and the second one is **HttpServletResponse** object. **HttpServletRequest** object represents the client's request and the **HttpServletResponse** represents the servlet's response.

Inside the doGet() method our servlet has first used the **setContentType()** method of the response object which sets the content type of the response to **text/html** It is the standard MIME content type for the html pages. After that it has used the method **getWriter ()** of the response object to retrieve a **PrintWriter** object. To display the output on the browser we use the **println ()** method of the **PrintWriter** class.

Program

```
Import  java.io.IOException;
import  java.io.PrintWriter;
import   javax.servlet.ServletException;
import  javax.servlet.http.HttpServlet;
import  javax.servlet.http.HttpServletRequest;
import  javax.servlet.http.HttpServletResponse;

public  class hello extends HttpServlet {
        private static final long serialVersionUID = 1L;
        public void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {

        PrintWriter out=response.getWriter(); java.util.Date today=new
        java.util.Date(); out.println("<html>"+"<body>"+"<h1
        align=center>welcome to
exciting world of servlet</h1>"+"<br>"+today+"</body>"+"</html>");
}

}
```

<u>Step to Run the Program</u>

1) Remove all the errors in the program.

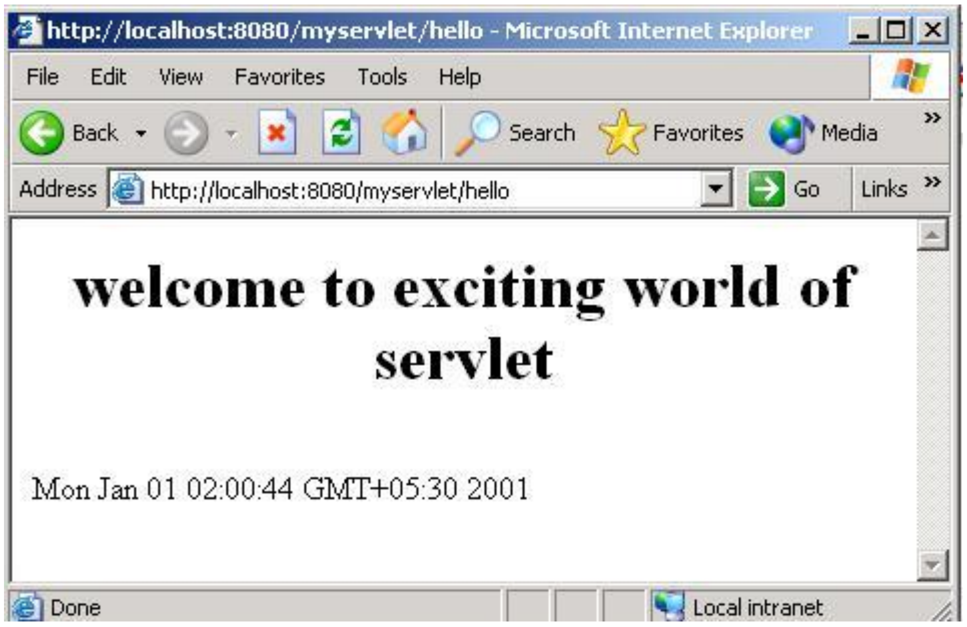
2) Start the web server.

3) Deploy your program to the web server

4) Open the web browser and execute the program as

http://localhost:8080/myservlet/hello

Output:


http://localhost:8080/myservlet/hello - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8080/myservlet/hello Go Links
Done
Local intranet

welcome to exciting world of servlet

Mon Jan 01 02:00:44 GMT+05:30 2001

Conclus ion:

Hence we studied how to create a sample servlet to display date and the time.

3. Basic JSP program

Aim: Write a program to demonstrate basic jsp example.

Theory :

JAVA SERVER PAGES

Java Server Pages (JSP) is a java technology that allows software developer to dynamically generate a HTML, XML or other types of the documents in response to a web client request. The technology allows java code and certain pre-defined actions to be embedded into static content.

The jsp syntax adds additional XML-like tags, called JSP action, to be used to invoke built in functionality dynamically generate .Additionally the technology allows us for the creation of the jsp tags libraries that acts as a extension to the standard HTML or XML tags. Tag libraries provide the platform independent way of extending the capabilities of a web server.

JSP are compiled into the java servlet by the java compiler. A jsp compiler may generate a servlet in java code that is then compiled by the java compiler.

JSP technology may enable the web developer and designers to rapidly develops and easily maintain, information rich, dynamic web pages that leverage existing business system.

This simple page contains plain HTML, except for couple of the JSP directives and tags. The first one in the HelloWorld.jsp is a **`page` directive** that defines the content type and character set of the entire page. Java method **`println`()** to print output. By this example we are going to learn that how can you write a jsp program on your browser..

This program also contains **HTML** (Hypertext Markup Language) code for designing the page and the contents. Following code of the program prints the string "Hello World!" by using **<%="Hello World!" %>** while you can also print the string by using **out.println ()**.

Pr ogr am :

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

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-
1">
<title>this is jsp1</title>
</head>
<body bgcolor="red">

<font size="10">
<%
String name="roseindia.net";
out.println("hello" + name +"!");
%>
</font>
</body>
</html>
```

Step to R un the Pr ogr am :

1) Remove all the errors in the program.

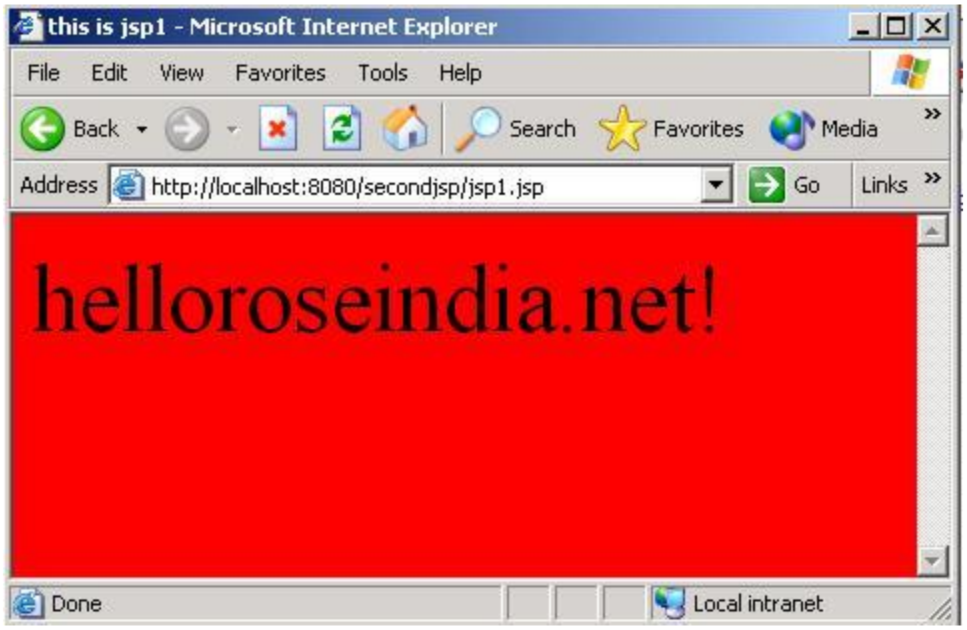
2) Start the web server.

3) Deploy your program to the web server

4) Open the web browser and execute the program as

http://localhost:8080/secondjsp/jsp1.jsp

Output:


this is jsp1 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8080/secondjsp/jsp1.jsp Go Links
Done
Local intranet

helloroseindia.net!

Conclusion:

Hence we studied how create the Servlet .

4. Program for Database Operation in jsp

Aim : Write a program to perform database operation in jsp.

Theory:

Create a database: First create a database named 'student' in mysql and table named "stu_info" in same database by sql query given below:

```
Create database student

  create table stu_info (
          ID int not null auto_increment,
          Name varchar(20),
          City varchar(20),
          Phone varchar(15),
          primary key(ID)
);
```

Create a new directory named "user" in the tomcat-6.0.16/webapps and WEB-INF directory in same directory. Before running this java code you need to paste a .jar file named mysql connector.jar in the Tomcat-6.0.16/webapps/user/WEB-INF/lib.

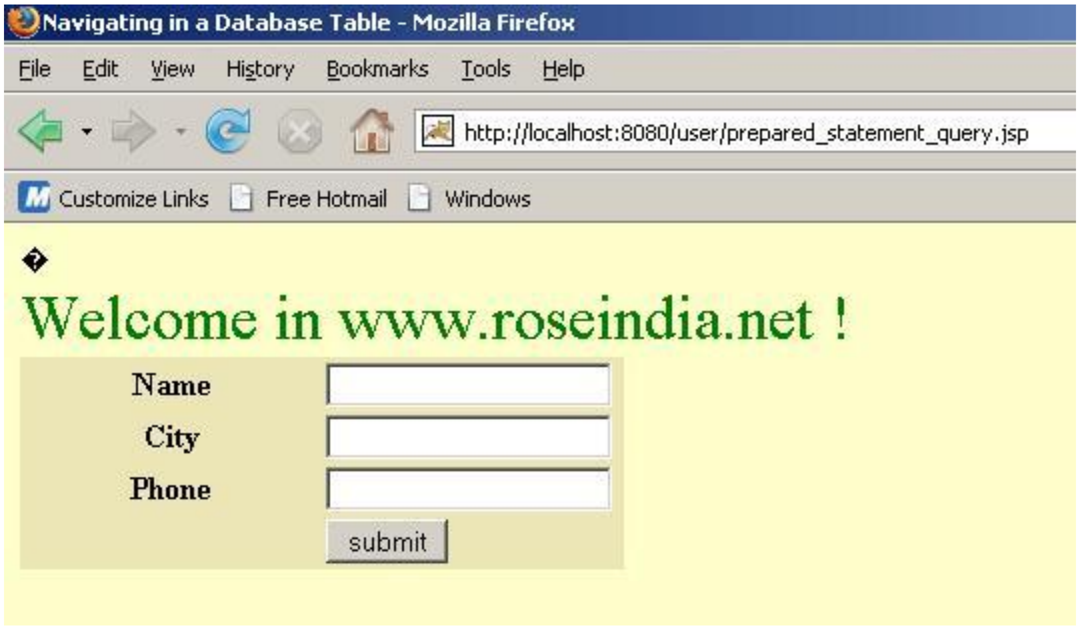
prepared_statement_query.jsp

Save this code as a .jsp file named "**prepared_statement_query.jsp**" in the directory Tomcat-6.0.16/webapps/user/ and you can run this jsp page with url **http://localhost:8080/user/prepared_statement_query.jsp** in address bar of the browser

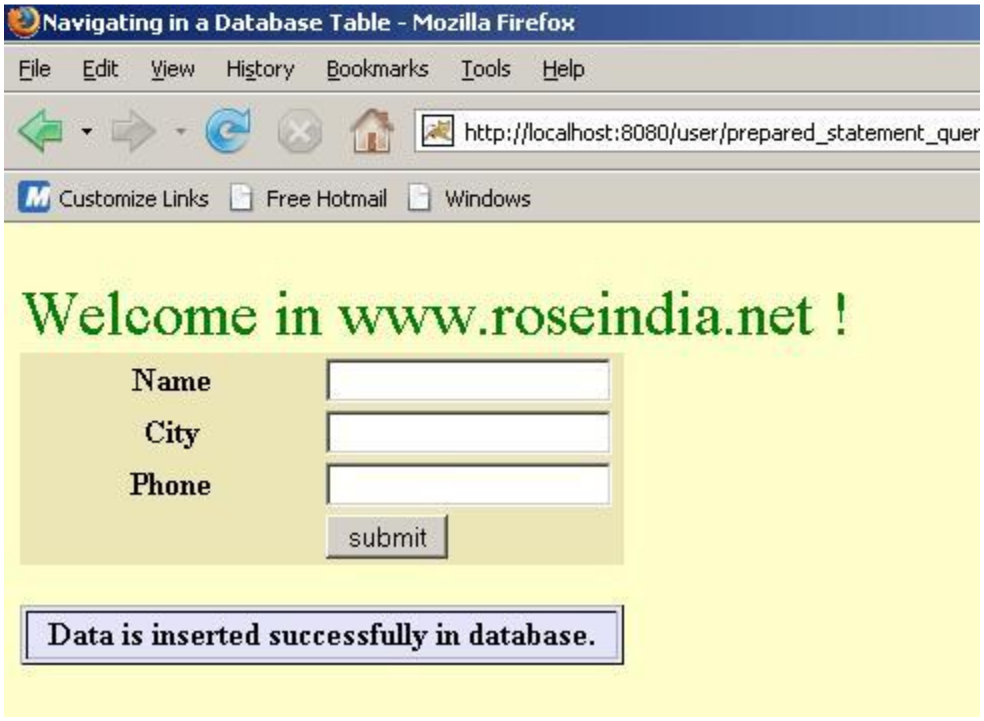
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd" >
<%@ page import="java.sql.*" %>
<%@ page import="java.io.*" %>
<HTML>
<HEAD>
<TITLE>insert data using prepared statement </TITLE>
</HEAD>
<BODY bgcolor="#ffffcc">
<font size="+3" color="green"><br>Welcome in www.roseindia.net
!</font>
<FORM action="prepared_statement_query.jsp" method="get">
<TABLE style="background-color: #ECE5B6;" WIDTH="30%" >

<TR>
<TH width="50%"> Name</TH>
<TD width="50%"><INPUT TYPE="text" NAME="name"></TD>
</tr>
<TR>
<TH width="50%">City</TH>
<TD width="50%"><INPUT TYPE="text" NAME="city"></TD>
</tr>
<TR>
<TH width="50%">Phone</TH>
<TD width="50%"><INPUT TYPE="text" NAME="phone"></TD>
</tr>

<TR>
<TH></TH>
<TD width="50%">< INPUT TYPE="submit" VALUE="submit"></TD>
</tr>
</TABLE>
<%
String name = request.getParameter ("name");
String city = request.getParameter ("city");
String phone = request.getParameter ("phone");

String connectionURL = "jdbc: mysql:
Connection  connection = null;

PreparedStatement pstatement = null;
```

```
Class.forName("com.mysql.jdbc.Driver").newInstance();
int updateQuery = 0;

if(name! =null && city!=null && phone!=null){
if(name!="" && city!="" && phone!="") {
try {

            connection = DriverManager.getConnection
             (connectionURL, "root", "root");

             String queryString = "INSERT INTO stu_info(Name,
             Address, Phone) VALUES (?, ?, ?)";

             pstatement = connection.prepareStatement(queryString);
             pstatement.setString(1, name);
                                     pstatement.setString(2, city);
                                     pstatement.setString(3, phone);
             updateQuery = pstatement.executeUpdate();
             if(updateQuery != 0) { %>
                     <br>
                     <TABLE style="background-color: #E3E4FA;"
                 WIDTH="30%" border="1">
                           <tr><th>Data is inserted successfully
                  in database.</th></tr>
                         </table>
             <%
             }
            }
           catch (Exception ex) {
           out.println("Unable to connect to database.");

           }
           finally {
              .
              pstatement.close();
              connection.close();
           }
              }
            }
%>
  </FORM>
 </body>
</html>
```

Output:


Navigating in a Database Table - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/user/prepared_statement_query.jsp
Customize Links Free Hotmail Windows
�
Welcome in www.roseindia.net !
Name
City
Phone
submit

Fill all the fields and click on submit button, that shows a response message. If any field is blank or only blank spaces are there page will remain same after clicking on submit button.


Navigating in a Database Table - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/user/prepared_statement_quer
Customize Links
Free Hotmail
Windows

Welcome in www.roseindia.net !

Name	
City	
Phone	
	submit

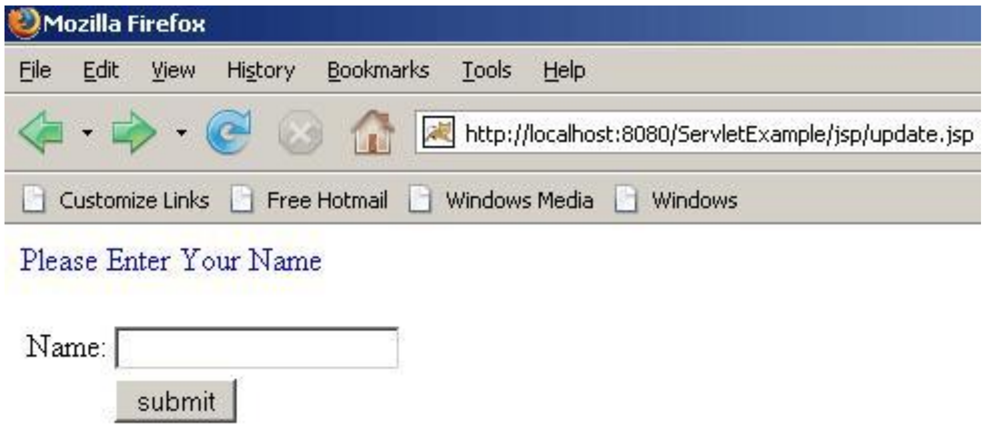
Data is inserted successfully in database.

Update Data:

This example shows how to update the existing record of mysql table using jdbc connectivity in the jsp page. In this example we have created two jsp pages **update.jsp** and **updatingDatabase.jsp.** In the **update.jsp** page, we are using a Text box where user can give his/her name and submit the page. After submitting the page, **updatingDatabase.jsp** will be called and the sql query **("update servlet set name = ? where id = ?")** is executed which will modify the table record.

```
<%@page language="java" session="true"
    contentType="text/html;charset= %>
<font color"blue" Please Enter Your Name </font><br><br>
<form name="frm" method="post" action="updatingDatabase.jsp">
<table border = "0">
  <tr align="left" valign="top">
    <td>Name:</td>
    <td><input type="text" name ="name" /></td>
  </tr>
  <tr align="left" valign="top">
    <td></td>
    <td><input type="submit" name="submit" value="submit"/></td>
  </tr>
</table>
</form>
```

Type the url: **http://localhost:8080/ServletExample/jsp/update.jsp** on your browser. Following output will be displayed:


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/ServletExample/jsp/update.jsp
Customize Links Free Hotmail Windows Media Windows

Please Enter Your Name

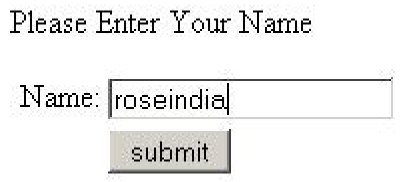
Name:

submit

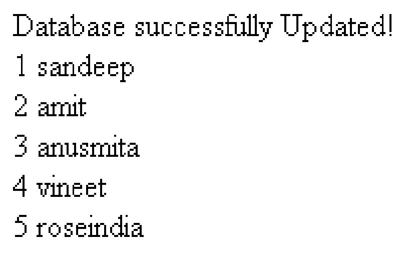
updatingDatabase.jsp

```
<%@ page language = "java" contentType =
        "text/html; charset = ISO-8859-1"
  import = "java.io.*"
  import = "java.sql.*"
  import = "java.util.*"
  import = "javax.sql.*"
  import = "java.sql.ResultSet"
  import = "java.sql.Statement"
  import = "java.sql.Connection"
  import = "java.sql.DriverManager"
  import = "java.sql.SQLException"
%>
<%
  Connection con = null;
  PreparedStatement ps = null;
  ResultSet rs = null;
  Statement stmt = null;
  String name = request.getParameter("name");
  Integer id = 5;
%>
<html>
<head>
  <title>Updating Database</title>
</head>
<body>
<%
  try {
    Class.forName("com.mysql.jdbc.Driver");
    con =DriverManager.getConnection
    ("jdbc:mysql://192.168.10.59:3306/example",
    "root", "root");
    ps = con.prepareStatement("update servlet set
    name = ? where id = ?");
    ps.setInt(2, id);
    ps.setString(1, name);
    ps.executeUpdate();
    %>
      Database successfully Updated!<br>
    <%
    if(ps.executeUpdate()>=1){
        stmt=con.createStatement();
        rs = stmt.executeQuery("SELECT * FROM servlet");
        while(rs.next()){
          %>
            <%=rs.getObject(1).toString()%>
            <%=("\t\t\t")%>
            <%=rs.getObject(2).toString()%>
            <%=("<br>")%>
          <%
        }
      }

    } catch (IOException e) {
      throw new IOException("Can not display records.", e);
    } catch (ClassNotFoundException e) {
      throw new SQLException("JDBC Driver not found.", e);
    } finally {
      try {
        if(stmt != null){
          stmt.close();
          stmt = null;
        }
        if(ps != null) {
          ps.close();
          ps = null;
        }
        if(con != null) {
          con.close();
          con = null;
        }
      } catch (SQLException e) {}
    }
%>
</body>
</html>
```

After submitting the name, it will be updated in the database and the records will be displayed on your browser.


Please Enter Your Name

Name: roseindia



```
Database successfully Updated!
1 sandeep
2 amit
3 anusmita
4 vineet
5 roseindia
```

Save this code as a .jsp file named "**prepared_statement_query.jsp**" in the directory Tomcat-6.0.16/webapps/user/ and you can run this jsp page with url **http://localhost:8080/user/prepared_statement_query.jsp** in address bar of the browser.

Conclusion:

Hence we studied how to perform different database operation in jsp.

5. Program for <jsp:useBean> Tag

A im : Program to use <jsp:useBean> Tag in JSP.

Theor y:

The <jsp:useBean> element locates or instantiates a JavaBeans component. <jsp:useBean> first attempts to locate an instance of the Bean. If the Bean does not exist, <jsp:useBean> instantiates it from a class or serialized template.

To locate or instantiate the Bean, <jsp:useBean> takes the following steps, in this order:

1. Attempts to locate a Bean with the scope and name you specify.
2. Defines an object reference variable with the name you specify.
3. If it finds the Bean, stores a reference to it in the variable. If you specified type, gives the Bean that type.
4. If it does not find the Bean, instantiates it from the class you specify, storing a reference to it in the new variable. If the class name represents a serialized template, the Bean is instantiated by java.beans.Beans.instantiate.
5. If <jsp:useBean> has *instantiated* the Bean, and if it has body tags or elements (between <jsp:useBean> and </jsp:useBean>), executes the body tags.

The body of a <jsp:useBean> element often contains a <jsp:setProperty> element that sets property values in the Bean. As described in Step 5, the body tags are only processed if <jsp:useBean> instantiates the Bean. If the Bean already exists and <jsp:useBean> locates it, the body tags have no effect.

JSPSyntax

```
<jsp:useBean
        id="beanInstanceName"
        scope="page | request | session | application"
        {
            class="package.class" |
            type="package.class" |
            class="package.class" type="package.class" |
            beanName="{package.class | <%= expression %>}"
type="package.class"
        }
        {
            /> |
            > other elements </jsp:useBean>
        }
```

Examples

```
<jsp:useBean id="cart" scope="session" class="session.Carts" />
<jsp:setProperty name="cart" property="*" />
<jsp:useBean id="checking" scope="session" class="bank.Checking" >
<jsp:setProperty name="checking" property="balance" value="0.0" />
</jsp:useBean>
```

AttributesandUsage

- id="*beanInstanceName*"

  A variable that identifies the Bean in the scope you specify. You can use the variable name in expressions or scriptlets in the JSP file.

  The name is case sensitive and must conform to the naming conventions of the scripting language used in the JSP page. If you use the Java programming language, the conventions in the *Java Language Specification*. If the Bean has already been created by another <jsp:useBean> element, the value of id must match the value of id used in the original <jsp:useBean> element.

- scope="**page** | request | session | application"

The scope in which the Bean exists and the variable named in id is available. The default value is page. The meanings of the different scopes are shown below:

  - page - You can use the Bean within the JSP page with the <jsp:useBean> element or any of the page's static include files, until the page sends a response back to the client or forwards a request to another file.
  - request - You can use the Bean from any JSP page processing the same request, until a JSP page sends a response to the client or forwards the request to another file. You can use the request object to access the Bean, for example, request.getAttribute(*beanInstanceName*).
  - session - You can use the Bean from any JSP page in the same session as the JSP page that created the Bean. The Bean exists across the entire session, and any page that participates in the session can use it. The page in which you create the Bean must have a <%@ page %> directive with session=true.
  - application - You can use the Bean from any JSP page in the same application as the JSP page that created the Bean. The Bean exists across an entire JSP application, and any page in the application can use the Bean.

- class="*package.class*"

  Instantiates a Bean from a class, using the new keyword and the class constructor. The class must not be abstract and must have a public, no-argument constructor. The package and class name are case sensitive.

- type="*package.class*"

  If the Bean already exists in the scope, gives the Bean a data type other than the class from which it was instantiated. If you use type without class or beanName, no Bean is instantiated. The package and class name are case sensitive.

- class="*package.class*" type="*package.class*"

  Instantiates a Bean from the class named in class and assigns the Bean the data type you specify in type. The value of type can be the same as class, a superclass of class, or an interface implemented by class.

The class you specify in class must not be abstract and must have a public, no-argument constructor. The package and class names you use with both class and type are case sensitive.

- beanName="{*package.class* | <%= *expression* %>}" type="*package.class*"

  Instantiates a Bean from either a class or a serialized template, using the java.beans.Beans.instantiate method, and gives the Bean the type specified in type. The Beans.instantiate method checks whether a name represents a class or a serialized template. If the Bean is serialized, Beans.instantiate reads the serialized form (with a name like *package.class.ser*) using a class loader. For more information, see the *JavaBeans API Specification*.

  The value of beanName is either a package and class name or an Expression that evaluates to a package and class name, and is passed to Beans.instantiate. The value of type can be the same as beanName, a superclass of beanName, or an interface implemented by beanName.

  The package and class names you use with both beanName and type are case sensitive

Pr ogr am :

The standard way of handling forms in JSP is to define a "bean".You just need to define a class that has a field corresponding to each field in the form. The class fields must have "setters" that match the names of the form fields.

getname.html

```
<HTML>
<BODY>
<FORM METHOD=POST ACTION="SaveName.jsp">
What's your name? <INPUT TYPE=TEXT NAME=username
SIZE=20><BR>
What's your e-mail address? <INPUT TYPE=TEXT NAME=email
SIZE=20><BR>
What's your age? <INPUT TYPE=TEXT NAME=age SIZE=4>
<P><INPUT TYPE=SUBMIT>
</FORM>
</BODY>
```

```
</HTML>
```

To collect this data, we define a Java class with fields **"username", "email"** and **"age"** and we provide setter methods **"setUsername", "setEmail"** and **"setAge",** as shown. A "setter" method is just a method that starts with **"set"** followed by the name of the field. The first character of the field name is upper-cased. So if the field is **"email",** its **"setter"** method will be **"setEmail".** Getter methods are defined similarly, with **"get"** instead of "set".

UserData.java

```
package user;

public class UserData {
    String username;
    String email;
    int age;

    public void setUsername( String value )
    {
        username = value;
    }

    public void setEmail( String value )
    {
        email = value;
    }

    public void setAge( int value )
    {
        age = value;
    }

    public String getUsername() { return username; }
    public String getEmail() { return email; }

    public int getAge() { return age; }
}
```

The method names must be exactly as shown. Once you have defined the class, compile it and make sure it is available in the web-server's classpath.

Now let us change "SaveName.jsp" to use a bean to collect the data.

savename.jsp

```
<jsp:useBean id="user" class="user.UserData" scope="session"/>
<jsp:setProperty name="user" property="*"/>
<HTML>
<BODY>
<A HREF="NextPage.jsp">Continue</A>
</BODY>
</HTML>
```

All we need to do now is to add the **jsp:useBean** tag and the **jsp:setProperty** tag! The useBean tag will look for an instance of the **"user.UserData"** in the session. If the instance is already there, it will update the old instance. Otherwise, it will create a new instance of **user.UserData** and put it in the session.

The setProperty tag will automatically collect the input data, match names against the bean method names, and place the data in the bean!

Let us modify NextPage.jsp to retrieve the data from bean..

nextpage.jsp

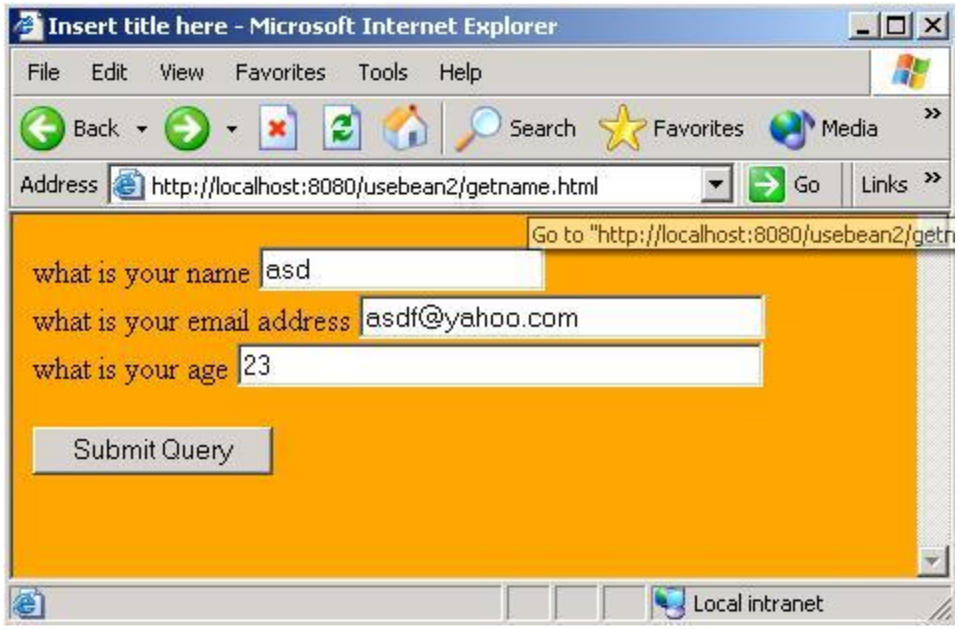
```
<jsp:useBean id="user" class="user.UserData" scope="session"/>
<HTML>
<BODY>
You entered<BR>
Name: <%= user.getUsername() %><BR>
Email: <%= user.getEmail() %><BR>
Age: <%= user.getAge() %><BR>
</BODY>
</HTML>
```

Step F or The E xec ution :

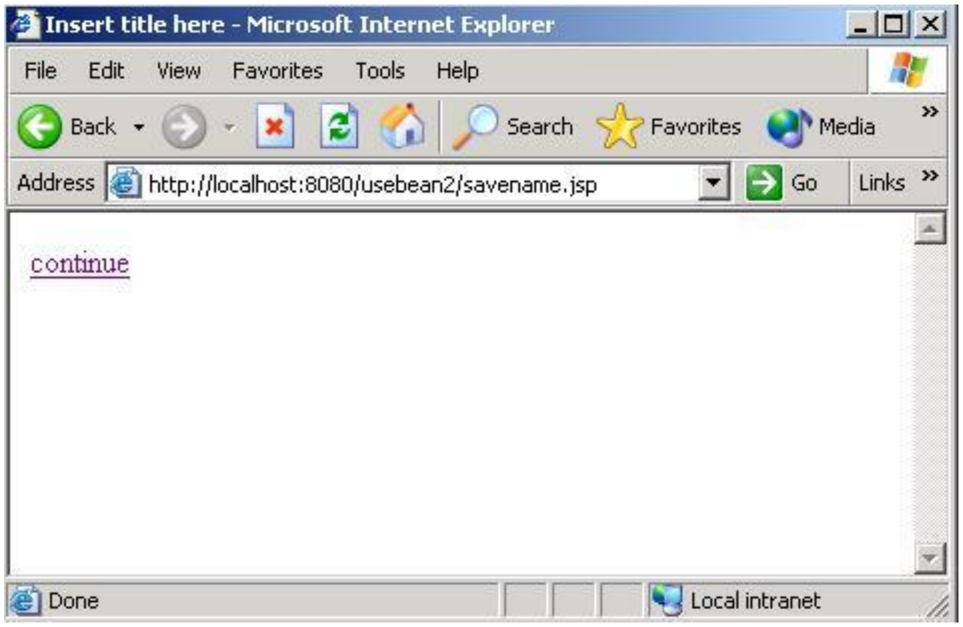
1) Remove all the errors in program
2) Start Your Web Server
3) Deployment
4) Open Your Web Browser and Type as Following

   http://localhost:8080/usebean2/getname.html

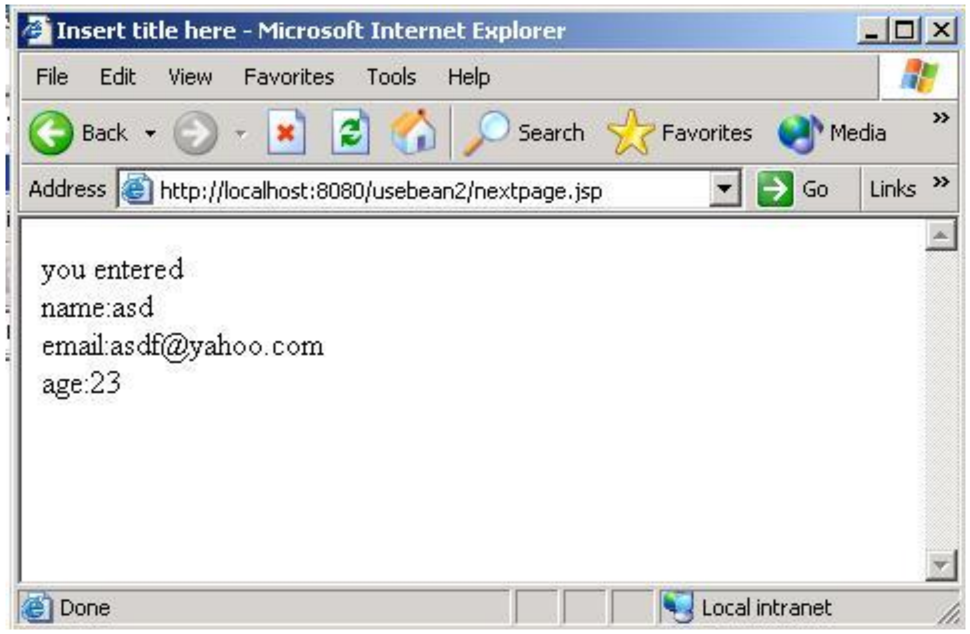
Outpu t:


Insert title here - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8080/usebean2/getname.html Go Links
Go to "http://localhost:8080/usebean2/getn
Submit Query
Local intranet

what is your name asd
what is your email address asdf@yahoo.com
what is your age 23


Insert title here - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8080/usebean2/savename.jsp Go Links
Done Local intranet

continue


Insert title here - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8080/usebean2/nextpage.jsp Go Links
Done Local intranet

```
you entered
name:asd
email:asdf@yahoo.com
age:23
```

C onclus io n: Hence we studied how to use <jsp:useBean> tag in jsp.

6. Program for session management

A im : Program to Create Session Management in jsp.

Theor y:

Session Management in JSP

In Java, the **Session** object is an instance of javax.servlet.http.HttpSession. The **Session** object provides a stateful context across multiple page requests from the same client during a conversation with the server. In other words, once a user has connected to the Web site, the **Session** object will be available to all of the servlets and JSPs that the user accesses until the session is closed due to timeout or error. You can access the HttpSession object for a servlet using the following code:

Listing 1. Accessing the HttpSession

```
//Inside doGet() or doPost()
HttpSession session = req.getSession();
```

In JSP, **Session** objects are used just as they are in servlets, except that you do not need the initialization code. For example, to store attributes in a **Session** object, you would use code like this:

Listing 2. Using the session variable in JSP

```
<%session.setAttribute("number", new Float(42.5));%>
```

This creates a key in the session variable named number, and assigns to it a value of "42.5." To retrieve information from a session, we simply use the **getAttribute** function, like this:

Listing 3. Retrieving information from a session

```
<td>Number: <%=
   session.getAttribute("number") %></td>
```

Attributes

The most commonly used feature of the **Session** object is the attribute **storage**. attributes are attached to several support objects and are the means for storing Web state in a particular scope. Session attributes are commonly used to store user values (such as name and authentication) and other information that needs to be shared between pages. For example, you could easily store an e-commerce shopping cart JavaBean in a session attribute. When the user selects the first product, a new shopping cart bean would be created and stored in the **Session** object. Any shopping page would have access to the cart, regardless of how the user navigates between pages.

Session management can be achieved by using the following thing.

1. **Cookies:** cookies are small bits of textual information that a web server sends to a browser and that browsers returns the cookie when it visits the same site again. In cookie the information is stored in the form of a name, value pair. By default the cookie is generated. If the user doesn't want to use cookies then it can disable them.

2. **URL rewriting:** In URL rewriting we append some extra information on the end of each URL that identifies the session. This URL rewriting can be used where a cookie is disabled. It is a good practice to use URL rewriting. In this session ID information is embedded in the URL, which is recieved by the application through Http GET requests when the client clicks on the links embedded with a page.

3. **Hidden form fields:** In hidden form fields the html entry will be like this : <input type ="hidden" name = "name" value="">. This means that when you submit the form, the specified name and value will be get included in get or post method. In this session ID information would be embedded within the form as a hidden field and submitted with the Http POST command.

In JSP we have been provided a implicit object session so we don't need to create a object of session explicitly as we do in Servlets. In Jsp the session is by default true. The session is defined inside the directive <%@ page session = "true/false" %>. If we don't declare it inside the jsp page then session will be available to the page, as it is default by true.

In this will create an session that takes the user name from the user and then saves into the user session. We will display the saved data to the user in another page.

savenameform.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-
1">
<title>name input from</title>
</head>
<body>
<form method="post" action="savenametosession.jsp">
<p><b>Enter Your Name: </b><input type="text" name="username"><br>
<input type="submit" value="Submit">

</form>

</body>
</html>
```

The above form prompts the user to enter his/her name. Once the user clicks on the submit button, savenametosession.jsp is called. The JSP savenametosession.jsp retrieves the user name from request attributes and saves into the user session using the function *session.setAttribute("username",username)*.

savenametosession.jsp:

```
<%@ page language="java" %>
<%
String username=request.getParameter("username");
if(username==null) username="";

session.setAttribute("username",username);
%>

<html>
<head>
<title>Name Saved</title>
</head>
```

```
<body>
<p><a href="showsessionvalue.jsp">Next Page to view the session
value</a><p>

</body>
```

The above JSP saves the user name into the session object and displays a link to next pages (showsessionvalue.jsp). When user clicks on the "Next Page to view session value" link, the JSP page showsessionvalue.jsp displays the user name to the user.

showsessionvalue.jsp:

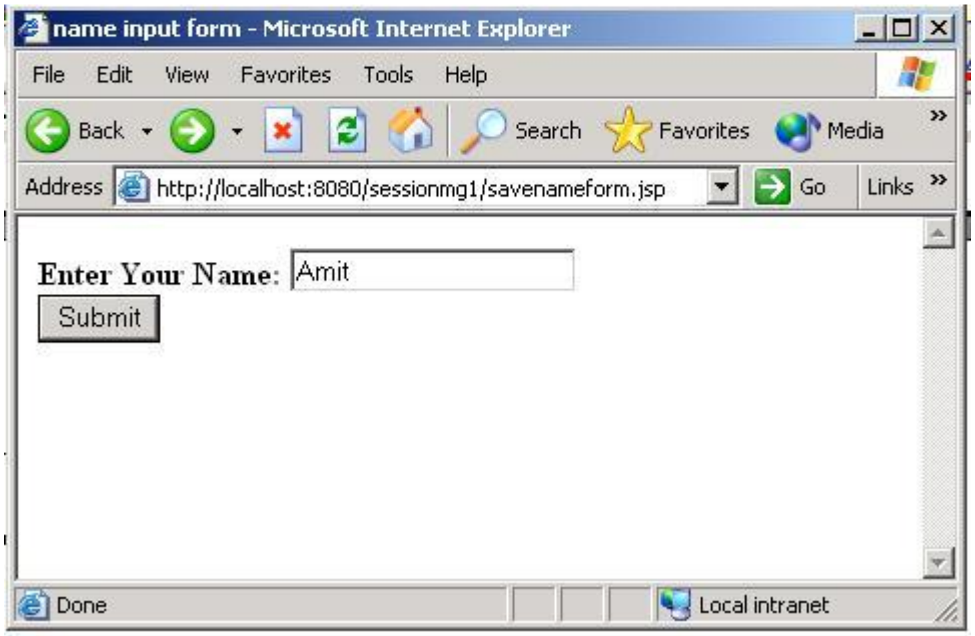
```
<%@ page language="java" %>
<%
String username=(String) session.getAttribute("username");
if(username==null) username="";
%>
<html>
<head>
<title>Show Saved Name</title>
</head>
<body>
<p>Welcome: <%=username%><p>

</body>
```

<u>Step F or The E xec ution :</u>

1) Remove all the errors in program
2) Start Your Web Server
3) Deployment
4) Open Your Web Browser and Type as Following

   http://localhost:8080/sessionmg1/savenameform.jsp

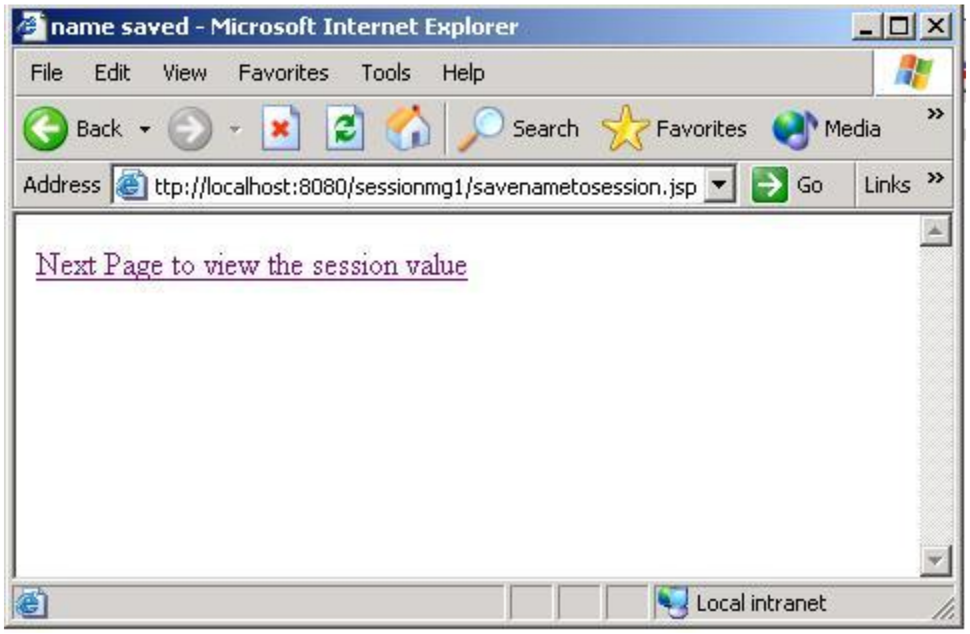
Outpu t:


name input form - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address Go Links
Done
Local intranet

http://localhost:8080/sessionmg1/savenameform.jsp

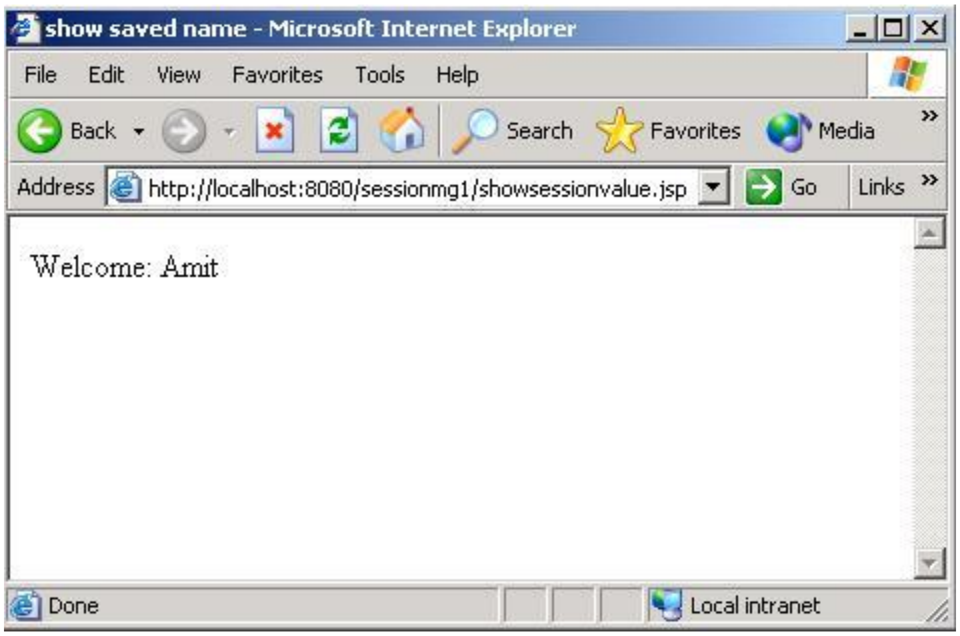
Enter Your Name: Amit

Submit


name saved - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address Go Links
Local intranet

ttp://localhost:8080/sessionmg1/savenametosession.jsp

Next Page to view the session value


show saved name - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8080/sessionmg1/showsessionvalue.jsp Go Links
Done Local intranet

Welcome: Amit

C onclus io n: Hence we studied how to create and manage the session in jsp.

7. Program For Custom JSP Tag

A im : Program To Create Custom JSP Tag.

Theor y:

JSP Custom Tags

JSP custom tags are merely Java classes that implement special interfaces. Once they are developed and deployed, their actions can be called from your HTML using XML syntax. They have a start tag and an end tag. They may or may not have a body. A bodyless tag can be expressed as:

```
<tagLibrary:tagName />
```

And, a tag with a body can be expressed as:

```
<tagLibrary:tagName>
   body
</tagLibrary:tagName>
```

Again, both types may have attributes that serve to customize the behavior of a tag. The following tag has an attribute called name, which accepts a String value obtained by evaluating the variable yourName:

```
<mylib:hello name="<%= yourName %>" />
```

Or, it can be written as a tag with a body as:

```
<mylib:hello>
  <%= yourName %>
</mylib:hello>
```

Benefits of Custom Tags

A very important thing to note about JSP custom tags is that they do not offer more functionality than scriptlets, they simply provide better packaging, by helping you improve the separation of business logic and presentation logic. Some of the benefits of custom tags are:

They can reduce or eliminate scriptlets in your JSP applications. Any necessary parameters to the tag can be passed as attributes or body content, and therefore no Java code is needed to initialize or set component properties.

They have simpler syntax. Scriptlets are written in Java, but custom tags can be used in an HTML-like syntax.

They can improve the productivity of nonprogrammer content developers, by allowing them to perform tasks that cannot be done with HTML.

They are reusable. They save development and testing time. Scritplets are not reusable, unless you call cut-and-paste reuse.

In short, you can use custom tags to accomplish complex tasks the same way you use HTML to create a presentation.

Defining a Tag

A tag is a Java class that implements a specialized interface. It is used to encapsulate the functionality from a JSP page. To define a simple bodyless tag, your class must implement the Tag interface. Developing tags with a body is discussed later. Sample 1 shows the source code for the Tag interface that you must implement:

Sample 1: Tag.java

```
public interface Tag {
  public final static int SKIP_BODY = 0;
  public final static int EVAL_BODY_INCLUDE = 1;
  public final static int SKIP_PAGE = 5;
  public final static int EVAL_PAGE = 6;

  void setPageContext(PageContext pageContext);
  void setParent(Tag parent);
  Tag getParent();
  int doStartTag() throws JspException;
  int doEndTag() throws JspException;
  void release();
}
```

All tags must implement the Tag interface (or one of its subinterfaces) as it defines all the methods the JSP runtime engine calls to execute a tag.

My First Tag
Now, let's look at a sample tag that when invoked prints a message to the client.

There are a few steps involved in developing a custom tag. These steps can be summarized as follows:

1. Develop the tag handler
2. Create a tag library descriptor
3. Test the tag

1. Develop the Tag Handler
A *tag handler* is an object invoked by the JSP runtime to evaluate a custom tag during the execution of a JSP page that references the tag. The methods of the tag handler are called by the implementation class at various points during the evaluation of the tag. Every tag handler must implement a specialized interface. In this example, the simple tag implements the Tag interface as shown in Sample 2.

Sample 2: HelloTag.java

```
package tags;

import java.io.*;
import javax.servlet.jsp.*;
import javax.servlet.jsp.tagext.*;

public class HelloTag implements Tag {
   private PageContext pageContext;
   private Tag parent;

   public HelloTag() {
      super();
   }

   public int doStartTag() throws JspException {
      try {
         pageContext.getOut().print(
         "This is my first tag!");
      } catch (IOException ioe) {
        throw new JspException("Error:
        IOException while writing to client"
        + ioe.getMessage());
      }
      return SKIP_BODY;
    }

    public int doEndTag() throws JspException {
      return SKIP_PAGE;
    }

    public void release() {
    }

    public void setPageContext(PageContext
    pageContext) {
      this.pageContext = pageContext;
    }

    public void setParent(Tag parent) {
      this.parent = parent;
    }

    public Tag getParent() {
      return parent;
    }
}
```

The two important methods to note in HelloTag are doStartTag and doEndTag. The doStartTag method is invoked when the start tag is encountered. In this example, this method returns SKIP_BODY because a simple tag has no body. The doEndTag method is invoked when the end tag is encountered. In this example, this method returns SKIP_PAGE because we do not want to evaluate the rest of the page; otherwise it should return EVAL_PAGE

To compile the HelloTag class, assuming that Tomcat is installed at: *c:\tomcat*:

Create a new subdirectory called *tags*, which is the name of the package containing the HelloTag class. This should be created at: *c:\tomcat\webapps\examples\web-inf\classes*.

Save *HelloTag.java* in the *tags* subdirectory.

Compile with the command:

```
        c:\tomcat\webapps\examples\web-inf\classes\tags>
        javac -classpath c:\tomcat\lib\servlet.jar
    HelloTag.java
```

2. Create a Tag Library Descriptor

The next step is to specify how the tag will be used by the JSP runtime that executes it. This can be done by creating a Tag Library Descriptor (TLD), which is an XML document. Sample 3 shows a sample TLD:

Sample 3: mytaglib.tld

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib
      PUBLIC "-//Sun Microsystems, Inc.//
      DTD JSP Tag Library 1.1//EN"
      "http://java.sun.com/j2ee/dtds/
      web-jsptaglibrary_1_1.dtd">

<!-- a tag library descriptor -->

<taglib>
   <tlibversion>1.0</tlibversion>
   <jspversion>1.1</jspversion>
   <shortname>first</shortname>
   <uri></uri>
   <info>A simple tab library for the
   examples</info>

  <tag>
    <name>hello</name>
    <tagclass>tags.HelloTag</tagclass>
    <bodycontent>empty</bodycontent>
    <info>Say Hi</info>
  </tag>
</taglib>
```

First we specify the tag library version and JSP version. The <shortname> tag specifies how we are going to reference the tag library from the JSP page. The <uri> tag can be used as a unique identifier for your tag library.

Save *mytaglib.tld* in the directory: *c:\tomcat\webapps\examples\web-inf\jsp*.

4. Test the Tag

The final step is to test the tag we have developed. In order to use the tag, we have to reference it, and this can be done in three ways:

1. Reference the tag library descriptor of an unpacked tag library. For example:
2. <@ taglib uri="/WEB-INF/jsp/mytaglib.tld"
3. prefix="first" %>
4. Reference a JAR file containing a tag library. For example:
5. <@ taglib uri="/WEB-INF/myJARfile.jar"
6. prefix='first" %>
7. Define a reference to the tag library descriptor from the web-application descriptor (web.xml) and define a short name to reference the tag library from the JSP. To do this, open the file: *c:\tomcat\webapps\examples\web-inf\web.xml* and add the following lines before the end line, which is <web-app>:

3. Test the Tag

The final step is to test the tag we have developed. In order to use the tag, we have to reference it, and this can be done in three ways:

1. Reference the tag library descriptor of an unpacked tag library. For example:
2. <@ taglib uri="/WEB-INF/jsp/mytaglib.tld"
3. prefix="first" %>
4. Reference a JAR file containing a tag library. For example:
5. <@ taglib uri="/WEB-INF/myJARfile.jar"
6. prefix='first" %>
7. Define a reference to the tag library descriptor from the web-application descriptor (web.xml) and define a short name to reference the tag library from the JSP. To do this, open the file: *c:\tomcat\webapps\examples\web-inf\web.xml* and add the following lines before the end line, which is <web-app>: \

```
        <taglib>
              <taglib-uri>mytags</taglib-uri>
              <taglib-location>/WEB-INF/jsp/
             mytaglib.tld</taglib-location>

           </taglib
```

Now, write a JSP and use the first syntax. Sample 4 shows an example:

Sample 4: Hello.jsp

```
<%@ taglib uri="/WEB-INF/jsp/mytaglib.tld"
 prefix="first" %>
<HTML>
<HEAD>
<TITLE>Hello Tag</TITLE>
</HEAD>

<BODY bgcolor="#ffffcc">

<B>My first tag prints</B>:

<first:hello/>

</BODY>
</HTML>
```

The taglib is used to tell the JSP runtime where to find the descriptor for our tag library, and the prefix specifies how we will refer to tags in this library. With this in place, the JSP runtime will recognize any usage of our tag throughout the JSP, as long as we precede our tag name with the prefix first as in <first:hello/>.

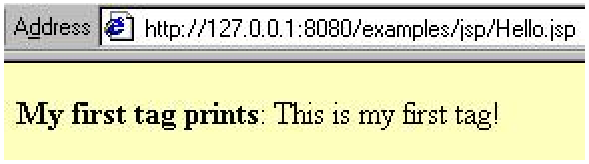
Alternatively, you can use the second reference option by creating a JAR file. Or, you can use the third reference option simply by replacing the first line in Sample 4 with the following line:

```
<%@ taglib uri="mytags" prefix="first" %>
```

Basically, we have used the mytags name, which was added to web.xml, to reference the tag library. For the rest of the examples in this article, this reference will be used.

Now, if you request Hello.jsp from a browser, you would see something similar to Figure below.

Outpu t:



Address http://127.0.0.1:8080/examples/jsp/Hello.jsp

```
My first tag prints: This is my first tag!
```

C onclus io n:

Thus ,we studied how to create custom tag in jsp

8. Basic Hibernate Program.

A im : Program to create simple hibernate

Theor y:

Hibernate is an object-relational mapping (ORM) library for the Java language, providing a framework for mapping an object-oriented domain model to a traditional relational database. Hibernate solves object-relational impedance mismatch problems by replacing direct persistence-related database accesses with high-level object handling functions.

Hibernate is free as open source software that is distributed under the GNU Lesser General Public License.

Hibernate's primary feature is mapping from Java classes to database tables (and from Java data types to SQL data types). Hibernate also provides data query and retrieval facilities. Hibernate generates the SQL calls and relieves the developer from manual result set handling and object conversion, keeping the application portable to all supported SQL databases, with database portability delivered at very little performance overhead

Hibernate applications define persistent classes that are "mapped" to database tables. Our "Hello World" example consists of one class and one mapping file. Let's see what a simple persistent class looks like, how the mapping is specified, and some of the things we can do with instances of the persistent class using Hibernate.

The objective of our sample application is to store messages in a database and to retrieve them for display. The application has a simple persistent class, Message, which represents

Listing 1. Message.java: A simple persistent class

```
package hello;
public class Message {
   private Long id;
   private String text;
   private Message nextMessage;
   private Message() {}
   public Message(String text) {
      this.text = text;
   }
   public Long getId() {
      return id;
   }
   private void setId(Long id) {
      this.id = id;
   }
   public String getText() {
      return text;
   }
   public void setText(String text) {
      this.text = text;
   }
   public Message getNextMessage() {
      return nextMessage;
   }
   public void setNextMessage(Message nextMessage) {
      this.nextMessage = nextMessage;
   }
}
```

Our Message class has three attributes: the identifier attribute, the text of the message, and a reference to another Message. The identifier attribute allows the application to access the database identity—the primary key value—of a persistent object. If two instances of Message have the same identifier value, they represent the same row in the database. We've chosen Long for the type of our identifier attribute, but this isn't a requirement. Hibernate allows virtually anything for the identifier type, as you'll see later.

You may have noticed that all attributes of the Message class have JavaBean-style property accessor methods. The class also has a constructor with no parameters. The persistent classes we use in our examples will almost always look something like this.
Instances of the Message class may be managed (made persistent) by Hibernate, but they don't *have* to be. Since the Message object doesn't implement any Hibernate-specific classes or interfaces, we can use it like any other Java class:

```
Message message = new Message("Hello World");
System.out.println( message.getText() );
```

This code fragment prints "Hello World" to the console. Our persistent class can be used in any execution save a new Message to the database:

```
Session session = getSessionFactory().openSession();
Transaction tx = session.beginTransaction();
Message message = new Message("Hello World");
session.save(message);
tx.commit();
session.close();
```

This code calls the Hibernate Session and Transaction interfaces. It results in the execution of something similar to the following SQL:

```
insert into MESSAGES (MESSAGE_ID, MESSAGE_TEXT,
NEXT_MESSAGE_ID)
values (1, 'Hello World', null)
```

Hold on—the MESSAGE_ID column is being initialized to a strange value. id property is *identifier property*—it holds a generated unique value. The value is assigned to the Message instance by Hibernate when save() is called.
For this example, we assume that the MESSAGES table already exists. Of course, we want our "Hello World" program to print the message to the console. Now that we have a message in the database, we're ready to demonstrate this. The next example retrieves all messages from the database, in alphabetical order, and prints them:

```
Session newSession = getSessionFactory().openSession();
Transaction newTransaction = newSession.beginTransaction();
List messages =
      newSession.find("from Message as m order by m.text asc");
System.out.println( messages.size() + " message(s) found:" );
for ( Iterator iter = messages.iterator(); iter.hasNext(); ) {
   Message message = (Message) iter.next();
   System.out.println( message.getText() );
}
newTransaction.commit();
newSession.close();
```

The literal string "from Message as m order by m.text asc" is a Hibernate query, expressed in Hibernate's own object-oriented Hibernate Query Language (HQL). This query is internally translated into the following SQL when find() is called:

```
select m.MESSAGE_ID, m.MESSAGE_TEXT, m.NEXT_MESSAGE_ID
from MESSAGES m
order by m.MESSAGE_TEXT asc
```

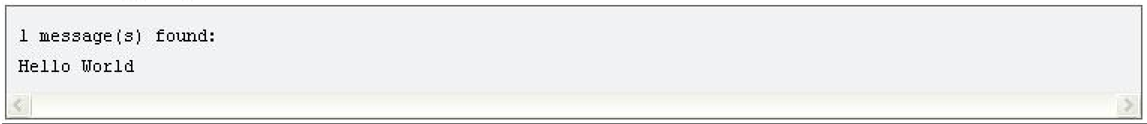
Hibernate needs more information about how the Message class should be made persistent. This information is usually provided in an *XML mapping document*. The mapping document defines, among other things, how properties of the Message class map to columns of the MESSAGES table. Let's look at the mapping document in Listing 2.

Listing 2. A simple Hibernate XML mapping

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
  "-//Hibernate/Hibernate Mapping DTD//EN"
  "http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd">
<hibernate-mapping>
  <class
    name="hello.Message"
    table="MESSAGES">
    <id name="id"
      column="MESSAGE_ID">
      <generator class="increment"/>
    </id>
    <property
      name="text"
      column="MESSAGE_TEXT"/>
    <many-to-one
      name="nextMessage"
      cascade="all"
      column="NEXT_MESSAGE_ID"/>
  </class>
</hibernate-mapping>
```

The mapping document tells Hibernate that the Message class is to be persisted to the MESSAGES table, that the identifier property maps to a column named MESSAGE_ID, that the text property maps to a column named MESSAGE_TEXT, and that the property named nextMessage is an association with *many-to-one multiplicity* that maps to a column named NEXT_MESSAGE_ID.
1
2
3

Outpu t:



```
1 message(s) found:
Hello World
```

C onclus io n:

Thus,we studied how to create simple hibernate application.

9. Database Operation Using Hibernate

A im : Program to perform different database operation using HQL.

Theor y:

Hibernate provides a powerful query language Hibernate Query Language that is expressed in a familiar SQL like syntax and includes full support for polymorphic queries. Hibernate also supports native SQL statements. It also selects an effective way to perform a database manipulation task for an application.

Step1: Create hibernate native sql for inserting data into database.

Hibernate Native uses only the Hibernate Core for all its functions. The code for a class that will be saved to the database is displayed below:

```
package hibernateexample;

import javax.transaction.*;
import org.hibernate.Transaction;
import org.hibernate.*;
import org.hibernate.criterion.*;
import org.hibernate.cfg.*;
import java.util.*;


public class HibernateNativeInsert {
  public static void main(String args[]){
  Session sess = null;
    try{
      sess = HibernateUtil.currentSession(); Transaction
      tx = sess.beginTransaction(); Studentdetail student =
      new Studentdetail();
      student.setStudentName("Amardeep Patel");
      student.setStudentAddress("rohini,sec-2, delhi-85");
      student.setEmail("amar@rediffmail.com");
      sess.save(student);
      System.out.println("Successfully data insert in database");
      tx.commit();
    }
    catch(Exception e){
      System.out.println(e.getMessage());
    } finally{
    sess.close();
    }
  }
}
```

Step 2: Create session factory 'HibernateUtil.java'.

code of session Factory:

```
package hibernateexample;

import java.sql.*;
import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;
import java.io.*;

public class HibernateUtil {
  public static final SessionFactory sessionFact;
  static {
    try {
      // Create the SessionFactory from hibernate.cfg.xml
      sessionFact = new Configuration().configure().buildSessionFactory();
    }
    catch(Throwable e) {
      System.out.println("SessionFactory creation failed." + e);
      throw new ExceptionInInitializerError(e);
    }
  }
  public static final ThreadLocal session = new ThreadLocal();
  public static Session currentSession() throws HibernateException {
    Session sess = (Session) session.get();
    // Open a new Session, if this thread has none yet
    if(sess == null){
      sess = sessionFact.openSession();
      // Store it in the ThreadLocal variable
      session.set(sess);
    }
    return sess;
  }
  public static void SessionClose() throws Exception {
    Session s = (Session) session.get();
    if (s != null)
    s.close();
    session.set(null);
  }
}
```

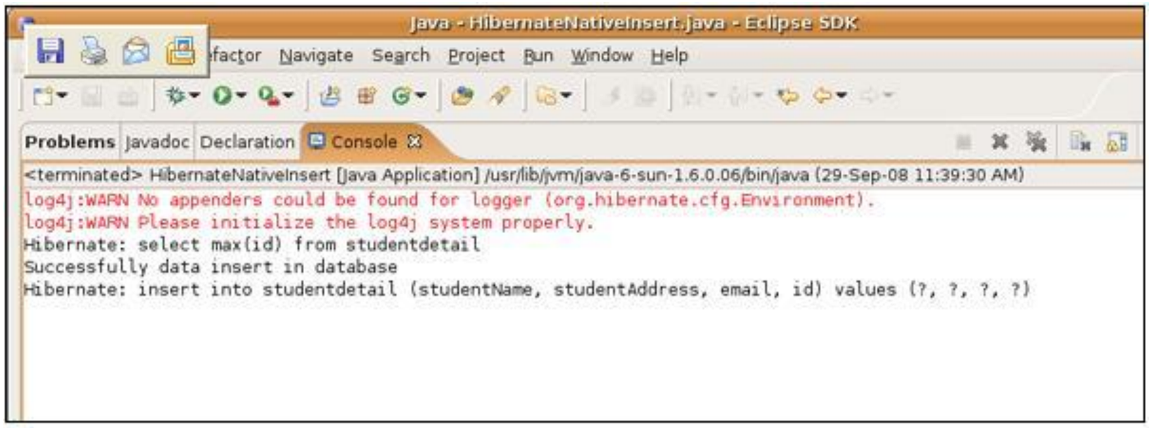
Step 3: Hibernate native uses the Plain Old Java Objects (POJOs) classes to map to the database table. We can configure the variables to map to the database column

"Studenetdetail.java":

```
package hibernateexample;

public class Studentdetail {
  private String studentName;
  private String studentAddress;
  private String email;
  private int id;

  public String getStudentName(){
    return studentName;
  }

  public void setStudentName(String studentName){
    this.studentName = studentName;
```

```
 }

 public String getStudentAddress(){
   return studentAddress;

 }

 public void setStudentAddress(String studentAddress){
   this.studentAddress = studentAddress;
 }

 public String getEmail(){
   return email;
 }

 public void setEmail(String email){
   this.email = email;
 }

 public int getId(){
   return id;
 }

 public void setId(int id){
   this.id = id;
 }
}
```

Outpu t:


Java - HibernateNativeInsert.java - Eclipse SDK
factor Navigate Search Project Run Window Help
Problems Javadoc Declaration Console

```
<terminated> HibernateNativeInsert [Java Application] /usr/lib/jvm/java-6-sun-1.6.0.06/bin/java (29-Sep-08 11:39:30 AM)
log4j:WARN No appenders could be found for logger (org.hibernate.cfg.Environment).
log4j:WARN Please initialize the log4j system properly.
Hibernate: select max(id) from studentdetail
Successfully data insert in database
Hibernate: insert into studentdetail (studentName, studentAddress, email, id) values (?, ?, ?, ?)
```

2) Delete Record From Database

To delete the record from the database write the query

“delete from Product where name=:name”;

3) Update Record From Database

To update the record from the database write the query

update "name of table"
set "column name"="new value" where column name=" "

C onclus io n: Thus, we studied how to perform different database operation using hibernate

10. Web Services

A im : Program to create simple web service.

Theor y:

Web Services:

Web services constitute a distributed computer architecture made up of many different computers trying to communicate over the network to form one system. They consist of a set of standards that allow developers to implement distributed applications - using radically different tools provided by many different vendors - to create applications that use a combination of software_modules called from systems in disparate departments or from other companies.

A Web service contains some number of classes, interfaces, enumerations and structures that provide black box functionality to remote clients. Web services typically define business objects that execute a unit of work (e.g., perform a calculation, read a data source, etc.) for the consumer and wait for the next request. Web service consumer does not necessarily need to be a browser-based client. Console-baed and Windows Forms-based clients can consume a Web service. In each case, the client indirectly interacts with the Web service through an intervening proxy. The proxy looks and feels like the real remote type and exposes the same set of methods. Under the hood, the proxy code really forwards the request to the Web service using standard **HTTP** or optionally **SOAP** messages.

A Web Service Example: HelloServiceBean

This example demonstrates a simple web service that generates a response based on information received from the client. HelloServiceBean is a stateless session bean that implements a single method, sayHello. This method matches the sayHello method invoked by the clients described in Static Stub Client.

<u>WebServiceEndpointInterface</u>

HelloService is the bean's web service endpoint interface. It provides the client's view of the web service, hiding the stateless session bean from the client. A web service endpoint interface must conform to the rules of a JAX-RPC service definition interface.

HelloService interface:

```
package helloservice;
import java.rmi.RemoteException;
import java.rmi.Remote;

public interface HelloService extends Remote {

   public String sayHello(String name) throws RemoteException;
}
```

<u>StatelessSessionBeanImplementationClass</u>

The HelloServiceBean class implements the sayHello method defined by the HelloService interface. The interface decouples the implementation class from the type of client access. For example, if you added remote and home interfaces to HelloServiceBean, the methods of the HelloServiceBean class could also be accessed by remote clients. No changes to the HelloServiceBean class would be necessary. The source code for the HelloServiceBean class follows:

```
package helloservice;
import java.rmi.RemoteException;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;

public class HelloServiceBean implements SessionBean {

   public String sayHello(String name) {

      return "Hello "+ name + " from HelloServiceBean";
   }

   public HelloServiceBean() {}
   public void ejbCreate() {}
```

```
  public void ejbRemove() {}
  public void ejbActivate() {}
  public void ejbPassivate() {}
  public void setSessionContext(SessionContext sc) {}
}
```

Build ing Hel lo Servic eBe an

In a terminal window, go to the *<INSTALL>*/j2eetutorial14/examples/ejb/helloservice/ directory. To build HelloServiceBean, type the following command:

```
asant build-service
```

This command performs the following tasks:

- Compiles the bean's source code files
- Creates the MyHelloService.wsdl file by running the following wscompile command:

  ```
  wscompile -define -d build/output -nd build -classpath build -mapping
  build/mapping.xml config-interface.xml
  ```

The wscompile tool writes the MyHelloService.wsdl file to the *<INSTALL>*/j2eetutorial14/examples/ejb/helloservice/build/ subdirectory. For more information about the wscompile tool, see Chapter 8.

Use deploytool to package and deploy this example

CreatingtheApplication

In this section, you'll create a J2EE application named HelloService, storing it in the file HelloService.ear.

1. In deploytool, select File→New→Application.
2. Click Browse.

3. In the file chooser, navigate to *<INSTALL>*/j2eetutorial14/examples/ejb/helloservice/.
4. In the File Name field, enter HelloServiceApp.
5. Click New Application.
6. Click OK.
7. Verify that the HelloServiceApp.ear file resides in <*INSTALL*>/j2eetutorial14/examples/ejb/helloservice

<u>**PackagingtheEnterpriseBean**</u>

Start the Edit Enterprise Bean wizard by selecting File→New→Enterprise Bean. The wizard displays the following dialog boxes.

1. Introduction dialog box
    a. Read the explanatory text for an overview of the wizard's features.
    b. Click Next.
2. EJB JAR dialog box
    a. Select the button labeled Create New JAR Module in Application.
    b. In the combo box below this button, select HelloService.
    c. In the JAR Display Name field, enter HelloServiceJAR.
    d. Click Edit Contents.
    e. In the tree under Available Files, locate the <*INSTALL*>/j2eetutorial14/examples/ejb/helloservice/build/ directory.
    f. In the Available Files tree select the helloservice directory and mapping.xml and MyHelloService.wsdl.
    g. Click Add.
    h. Click OK.
    i. Click Next.
3. General dialog box
    a. In the Enterprise Bean Class combo box, select helloservice.HelloServiceBean.
    b. Under Enterprise Bean Type, select Stateless Session.
    c. In the Enterprise Bean Name field, enter HelloServiceBean.
    d. Click Next.
4. In the Configuration Options dialog box, click Next. The wizard will automatically select the Yes button for Expose Bean as Web Service Endpoint.
5. In the Choose Service dialog box:

    a. Select META-INF/wsdl/MyHelloService.wsdl in the WSDL File combo box.
    b. Select mapping.xml from the Mapping File combo box.
    c. Make sure that MyHelloService is in the Service Name and Service Display Name edit boxes.
6. In the Web Service Endpoint dialog box:
    a. Select helloservice.HelloIF in the Service Endpoint Interface combo box.
    b. In the WSDL Port section, set the Namespace to urn:Foo, and the Local Part to HelloIFPort.
    c. In the Sun-specific Settings section, set the Endpoint Address to hello-ejb/hello.
    d. Click Next.
7. Click Finish.
8. Select File→Save.

DeployingtheEnterpriseApplication

Now that the J2EE application contains the enterprise bean, it is ready for deployment.

1. Select the HelloService application.
2. Select Tools→Deploy.
3. Under Connection Settings, enter the user name and password for the Application Server.
4. Click OK.
5. In the Distribute Module dialog box, click Close when the deployment completes.
6. Verify the deployment.
    a. In the tree, expand the Servers node and select the host that is running the Application Server.
    b. In the Deployed Objects table, make sure that HelloService is listed and that its status is Running.

Build ing t he Web Service Clie nt

To verify that HelloServiceBean has been deployed, click on the target Application Server in the Servers tree in deploytool. In the Deployed Objects tree you should see HelloServiceApp.

To build the static stub client, perform these steps:

1. In a terminal go to the *<INSTALL>*/j2eetutorial14/examples/jaxrpc/helloservice/ directory and type

   ```
   asant build
   ```

2. In a terminal go to the *<INSTALL>*/j2eetutorial14/examples/jaxrpc/staticstub/ directory.
3. Open config-wsdl.xml in a text editor and change the line that reads

   ```
   <wsdl location="http://localhost:8080/hello-jaxrpc/hello?WSDL"
   ```

   to

   ```
   <wsdl location="http://localhost:8080/hello-ejb/hello?WSDL"
   ```

4. Type

   ```
   asant build
   ```

5. Edit the build.properties file and change the endpoint.address property to

   ```
   http://localhost:8080/hello-ejb/hello
   ```

RunningtheWebServiceClient

To run the client, go to the *<INSTALL>*/j2eetutorial14/examples/jaxrpc/staticstub/ directory and enter

```
asant run
```

The client should display the following line:

Outpu t:

```
Hello Duke! (from HelloServiceBean)
```

C onclus io n:

Thus,we learnt how to create simple web service.