TPCT's
College of Engineering, Osmanabad

Laboratory Manual

ADVANCED JAVA

For

Third Year Students

Manual Prepared by

Mr.A.A.Nikam

Author COE, Osmanabad

TPCT's

College of Engineering
Solapur Road, Osmanabad
Department of Computer Science & Enggineering

Vision of the Department:

To achieve and evolve as a center of academic excellence and research center in the field of Computer Science and Engineering. 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 Department

The department strives to continuously engage in providing the students with in-depth understanding of fundamentals and practical training related to professional skills and their applications through effective Teaching- Learning Process and state of the art laboratories pertaining to CSE and inter disciplinary areas.Preparing students in developing research, design, entrepreneurial skills and employability capabilities.

College of Engineering

Technical Document

This technical document is a series of Laboratory manuals of Department of Computer Science & Engineering and is a certified document of College of Engineering, Osmanabad. The care has been taken to make the document error-free. But still if any error is found. Kindly bring it to the notice of subject teacher and HOD.

Recommended by,

HOD

Approved by,

Principal

Copies:

1. Departmental Library
2. Laboratory
3. HOD
4. Principal

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 implementing advanced java concepts.

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.

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.

H.O.D.

LABORATORY MANUAL CONTENTS

This manual is intended for the third year students of engineering branches in the subject of Advanced Java. This manual typically contains practical/Lab Sessions related Signals and Systems covering various aspects related to the subject to enhance understanding.

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.

Mr. A.A.Nikam

SUBJECT INDEX

1. DOs and DON' Ts in Laboratory:

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.

Instruction for Laboratory Teachers::

- 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.
- Students should be taught for taking the printouts under the observation of lab teacher.
- The promptness of submission should be encouraged by way of marking and evaluation patterns that will benefit the sincere students.

.

2.Pre-Lab

Introduction to Advanced Java

Questions:

1] What do you mean by OOP's concepts using Java?
2]What is client server architecture?How it works?
3]Describe Enterprise architecture using java?

1. Program to demonstrate database handling.

Aim : Write Program to demonstrate database handling.

Tools Required:-jdk1.6,netbeans,oracle/mysql DB.

Objective :-Students will able to perform databse operation.

Theory:
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.

Load 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!" );
```

Create a oracle 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

Create 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.

Execute 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().

A lgorithm :

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.

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

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:

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

2. Servlet Program using HTTPServlet.

Aim : Write a program to demonstrate HttpServlet.

Objective :-Students will able to demonstrate HTTPServlet.

Tools Required:-jdk1.6,netbeans

Theory:

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 required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Extend HttpServlet class
public class HelloWorld extends HttpServlet {

  private String message;

  public void init() throws ServletException {
    // Do required initialization
    message = "Hello World";
  }

  public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {

    // Set response content type
    response.setContentType("text/html");

    // Actual logic goes here.
    PrintWriter out = response.getWriter();
```

```
    out.println("<h1>" + message + "</h1>");
  }

  public void destroy() {
    // do nothing.
  }
}
```

Web.xml file:-

```
<servlet>
  <servlet-name>HelloWorld</servlet-name>
  <servlet-class>HelloWorld</servlet-class>
</servlet>

<servlet-mapping>
  <servlet-name>HelloWorld</servlet-name>
  <url-pattern>/HelloWorld</url-pattern>
</servlet-mapping>
```

Conclus ion:Hence we studied how to create a sample servlet using HttpServlet

3.Basic Jsp program

Aim: Write a program to demonstrate basic jsp example.

Objective :-Students will able to implement jsp program.

Tools Required:-jdk1.6,netbeans

Theory :

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 ().

Program :

```
<%@ 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>
```

Output:.

Conclusion:

Hence we studied how create the JSP page .

4. Program for Database Operation in jsp

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

Objective :-Students will able to perform database operation.

Tools Required:-jdk1.6,netbeans,mysql/oracle

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>

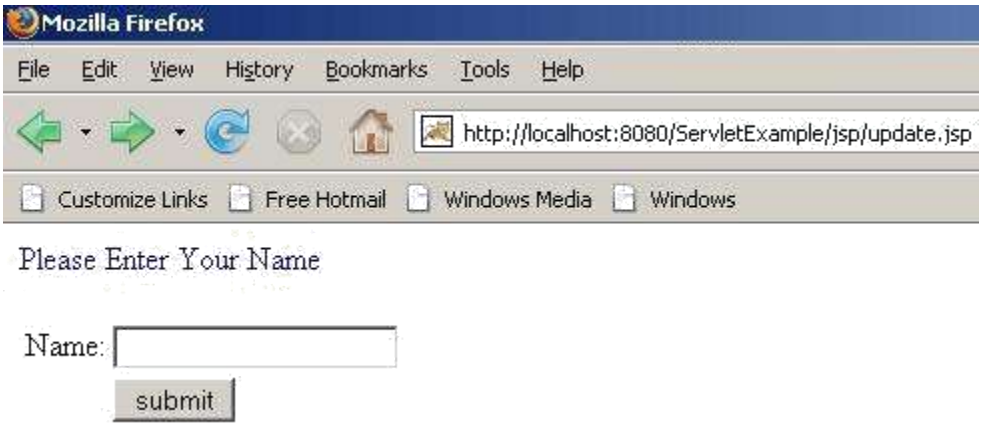
<TITLE>insert data using prepared statement </TITLE> </HEAD>

```
<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:



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

5.Program to implement usebean tag.

Aim:-To implement usebean tag in jsp.

Objective :-Students will able to implement usebean tag.

Tools:jdk1.6,netbeans

Theory:

- create a Java Bean. The Java Bean is a specially constructed Java class that provides a default, no-argument constructor, implements the Serializable interface and it has getter and setter methods for its properties.
- Create a jsp page, using the <%code fragment%> scriptlet. It can contain any number of JAVA language statements, variable or method declarations, or expressions that are valid in the page scripting language.
- Use the useBean action to declare the JavaBean for use in the JSP page. Once declared, the bean becomes a scripting variable that can be accessed by both scripting elements and other custom tags used in the JSP.
- Use the getProperty action to access get methods and setProperty action to access set methods of the bean.

```
01package com.javacodegeeks.snippets.enterprise;
02
03 import java.util.Date;
04
05 public class SampleBean {
06
07    private String param1;
08    private Date param2 = new Date();
09
10    public String getParam1() {
11        return param1;
12    }
13    public void setParam1(String param1) {
14        this.param1 = param1;
15    }
16
17    public Date getParam2() {
18        return param2;
```

```
19     }
20     public void setParam2(Date param2) {
21         this.param2 = param2;
22     }
23 
24     @Override
25     public String toString() {
26         return "SampleBean [param1=" + param1 + ", param2=" + param2 + "]";
27     }
28 
29 }
```

UseBean.jsp

```
01 <%@ page language="java" contentType="text/html;charset=UTF-8" %>
02 <%@ page import="com.javacodegeeks.snippets.enterprise.SampleBean"%>
03 
04 <html>
05 
06 <head>
07     <title>Java Code Geeks Snippets - Use a Bean in JSP Page</title>
08 </head>
09 
10 <body>
11 
1
2     <jsp:useBean
  id="sampleBean" class="com.javacodegeeks.snippets.enterprise.SampleBean"scope="session"
  >
13         <%-- intialize bean properties --%>
14         <jsp:setProperty name="sampleBean" property="param1" value="value1" />
15     </jsp:useBean>
16 
17     Sample Bean: <%= sampleBean %>
18 
19     param1: <jsp:getProperty name="sampleBean" property="param1" />
20     param2: <jsp:getProperty name="sampleBean" property="param2" />
21 
22 </body>
```

URL:

```
http://myhost:8080/jcgsnippets/UseBean.jsp
```

Output:

```
Sample Bean: SampleBean [param1=value1, param2=Thu Nov 17 21:28:03 EET 2011]

param1: value1 param2: Thu Nov 17 21:28:03 EET 2011
```

Conclusion:Hence we implemted usebean using jsp.

6.Program to implement session management in jsp.

Aim:-To implement session management in jsp.

Objective :-Students will able to perform session management in jsp.

Tools:jdk1.6,netbeans

Theory:

Maintaining Session Between Web Client And Server

Let us now discuss a few options to maintain the session between the Web Client and the Web Server −

Cookies

A webserver can assign a unique session ID as a cookie to each web client and for subsequent requests from the client they can be recognized using the received cookie.

This may not be an effective way as the browser at times does not support a cookie. It is not recommended to use this procedure to maintain the sessions.

Hidden Form Fields

A web server can send a hidden HTML form field along with a unique session ID as follows −

<input type = "hidden" name = "sessionid" value = "12345">

This entry means that, when the form is submitted, the specified name and value are automatically included in the GET or the POST data. Each time the web browser sends the request back, the session_id value can be used to keep the track of different web browsers.

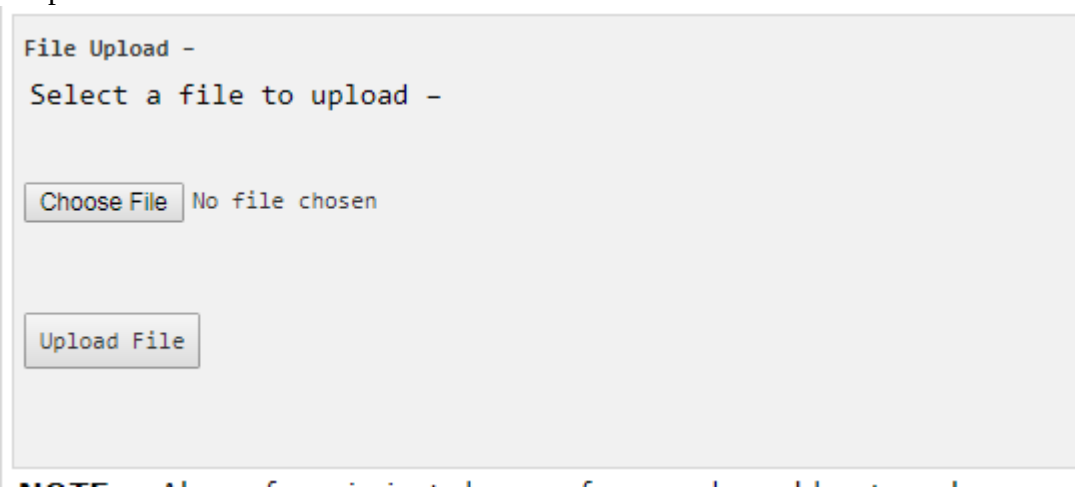
Program:

```
<%@ page import = "java.io.*,java.util.*" %>
<%
  // Get session creation time.
  Date createTime = new Date(session.getCreationTime());

  // Get last access time of this Webpage.
  Date lastAccessTime = new Date(session.getLastAccessedTime());

  String title = "Welcome Back to my website";
  Integer visitCount = new Integer(0);
  String visitCountKey = new String("visitCount");
  String userIDKey = new String("userID");
  String userID = new String("ABCD");

  // Check if this is new comer on your Webpage.
  if (session.isNew() ){
    title = "Welcome to my website";
```

```
    session.setAttribute(userIDKey, userID);
    session.setAttribute(visitCountKey,  visitCount);
   }
   visitCount = (Integer)session.getAttribute(visitCountKey);
   visitCount = visitCount + 1;
   userID = (String)session.getAttribute(userIDKey);
   session.setAttribute(visitCountKey,  visitCount);
%>

<html>
   <head>
      <title>Session Tracking</title>
   </head>

   <body>
      <center>
         <h1>Session Tracking</h1>
      </center>

      <table border = "1" align = "center">
         <tr bgcolor = "#949494">
            <th>Session info</th>
            <th>Value</th>
         </tr>
         <tr>
            <td>id</td>
            <td><% out.print( session.getId()); %></td>
         </tr>
         <tr>
            <td>Creation Time</td>
            <td><% out.print(createTime); %></td>
         </tr>
         <tr>
            <td>Time of Last Access</td>
            <td><% out.print(lastAccessTime); %></td>
         </tr>
         <tr>
            <td>User ID</td>
            <td><% out.print(userID); %></td>
         </tr>
         <tr>
            <td>Number of visits</td>
            <td><% out.print(visitCount); %></td>
         </tr>
      </table>
```

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

Output:



Conclusion: Hence we implemented session management in jsp.

7.Program to implement Custum tags jsp.

Aim:-To implement custum tags in jsp.

Objective :-Students will able to implement custum tags in jsp.

Tools:jdk1.6,netbeans

Theory:

To create the Tag Handler, we are inheriting the **TagSupport class** and overriding its method **doStartTag()**.To write data for the jsp, we need to use the **JspWriter class**.

The **PageContext** class provides **getOut()** method that returns the instance of JspWriter class. TagSupport class provides instance of pageContext bydefault.

Programs:-

```
package com.javatpoint.sonoo;
import java.util.Calendar;
import javax.servlet.jsp.JspException;
import javax.servlet.jsp.JspWriter;
import javax.servlet.jsp.tagext.TagSupport;
public class MyTagHandler extends TagSupport{

public int doStartTag() throws JspException {
   JspWriter out=pageContext.getOut();//returns the instance of JspWriter
   try{
    out.print(Calendar.getInstance().getTime());//printing date and time using JspWriter
   }catch(Exception e){System.out.println(e);}
   return SKIP_BODY;//will not evaluate the body content of the tag
}
}
```

2) Create the TLD file

Tag Library Descriptor (TLD) file contains information of tag and Tag Hander classes. It must be contained inside the WEB-INF directory.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE taglib
      PUBLIC "-//Sun Microsystems, Inc.//DTD JSP Tag Library 1.2//EN"
   "http://java.sun.com/j2ee/dtd/web-jsptaglibrary_1_2.dtd">

<taglib>

  <tlib-version>1.0</tlib-version>
  <jsp-version>1.2</jsp-version>
  <short-name>simple</short-name>
  <uri>http://tomcat.apache.org/example-taglib</uri>

<tag>
```

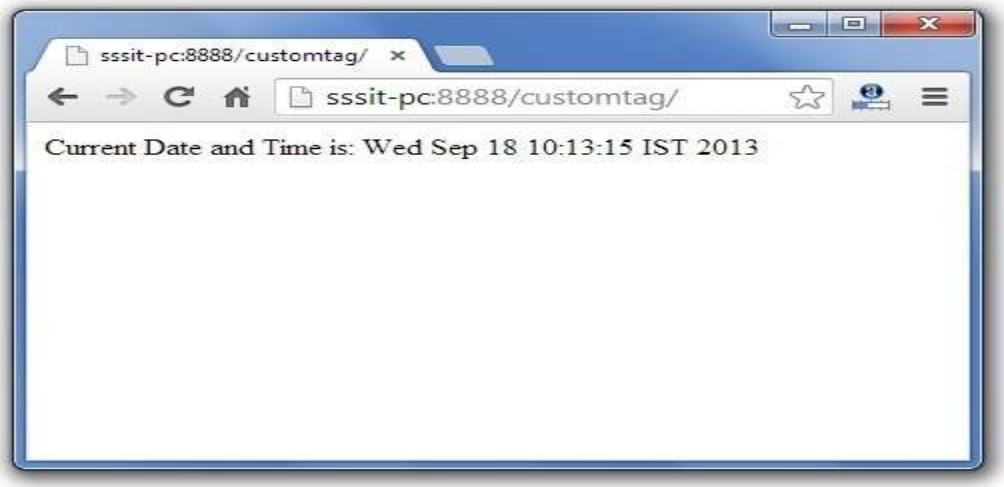
<name>today</name>
<tag-class>com.javatpoint.sonoo.MyTagHandler</tag-class>
</tag>
</taglib>

3) Create the JSP file
Let's use the tag in our jsp file. Here, we are specifying the path of tld file directly. But it is recommended to use the uri name instead of full path of tld file. We will learn about uri later.

It uses taglib directive to use the tags defined in the tld file.

File: index.jsp
<%@ taglib uri="WEB-INF/mytags.tld" prefix="m" %>
Current Date and Time is: <m:today/>

Output:-



Conclusion:-Hence we implemented custum tags in jsp.

8.Program to implement basic hibernate program.

Aim:-To implement simple hibernate program.

Objective :-Students will able to implement hibernate program.

Tools:jdk1.6,netbeans

Theory: Here, we are going to create a simple example of hibernate application using netbeans IDE. For creating the first hibernate application in Eclipse IDE, we need to follow following steps:

1. Create the java project
2. Add jar files for hibernate
3. Create the Persistent class
4. Create the mapping file for Persistent class
5. Create the Configuration file
6. Create the class that retrieves or stores the persistent object
7. Run the application

```
public class Employee {
  private int id;
  private String firstName;
  private String lastName;
  private int salary;

  public Employee() {}
  public Employee(String fname, String lname, int salary) {
    this.firstName = fname;
    this.lastName = lname;
    this.salary = salary;
  }

  public int getId() {
    return id;
  }

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

  public String getFirstName() {
    return firstName;
  }
```

```
  public void setFirstName( String first_name ) {
    this.firstName = first_name;
  }

  public String getLastName() {
    return lastName;
  }

  public void setLastName( String last_name ) {
    this.lastName = last_name;
  }

  public int getSalary() {
    return salary;
  }

  public void setSalary( int salary ) {
    this.salary = salary;
```

Create database table

```
create table EMPLOYEE (

  id INT NOT NULL auto_increment,

  first_name VARCHAR(20) default NULL,

  last_name  VARCHAR(20) default NULL,

  salary     INT  default NULL,

  PRIMARY KEY (id)

);
```

Create Mapping Configuration File

This step is to create a mapping file that instructs Hibernate how to map the defined class or classes to the database tables.

```
<?xml version = "1.0" encoding = "utf-8"?>

<!DOCTYPE hibernate-mapping PUBLIC

"-//Hibernate/Hibernate Mapping DTD//EN"
```

```
"http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">


<hibernate-mapping>
  <class name = "Employee" table = "EMPLOYEE">


    <meta attribute = "class-description">
      This class contains the employee detail.
    </meta>


    <id name = "id" type = "int" column = "id">
      <generator class="native"/>
    </id>


    <property name = "firstName" column = "first_name" type = "string"/>
    <property name = "lastName" column = "last_name" type = "string"/>
    <property name = "salary" column = "salary" type = "int"/>


  </class>
</hibernate-mapping>
```

create Application Class

Finally, we will create our application class with the main() method to run the application. We will use this application to save few Employee's records and then we will apply CRUD operations on those records.

```
import java.util.List;

import java.util.Date;

import java.util.Iterator;
```

```
import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.Transaction;
import org.hibernate.SessionFactory;
import org.hibernate.cfg.Configuration;

public class ManageEmployee {
   private static SessionFactory factory;
   public static void main(String[] args) {

      try {
         factory = new Configuration().configure().buildSessionFactory();
      } catch (Throwable ex) {
         System.err.println("Failed to create sessionFactory object." + ex);
         throw new ExceptionInInitializerError(ex);
      }

      ManageEmployee ME = new ManageEmployee();

      /* Add few employee records in database */
      Integer empID1 = ME.addEmployee("Zara", "Ali", 1000);
      Integer empID2 = ME.addEmployee("Daisy", "Das", 5000);
      Integer empID3 = ME.addEmployee("John", "Paul", 10000);

      /* List down all the employees */
```

```
   ME.listEmployees();

   /* Update employee's records */
   ME.updateEmployee(empID1, 5000);

   /* Delete an employee from the database */
   ME.deleteEmployee(empID2);

   /* List down new list of the employees */
   ME.listEmployees();
}

/* Method to CREATE an employee in the database */
public Integer addEmployee(String fname, String lname, int salary){
   Session session = factory.openSession();
   Transaction tx = null;
   Integer employeeID = null;

   try {
      tx = session.beginTransaction();
      Employee employee = new Employee(fname, lname, salary);
      employeeID = (Integer) session.save(employee);
      tx.commit();
   } catch (HibernateException e) {
      if (tx!=null) tx.rollback();
      e.printStackTrace();
```

```
   } finally {
      session.close();
   }
   return employeeID;
}

/* Method to  READ all the employees */
public void listEmployees( ){
   Session session = factory.openSession();
   Transaction tx = null;

   try {
      tx = session.beginTransaction();
      List employees = session.createQuery("FROM Employee").list();
      for (Iterator iterator = employees.iterator(); iterator.hasNext();){
         Employee employee = (Employee) iterator.next();
         System.out.print("First Name: " + employee.getFirstName());
         System.out.print("  Last Name: " + employee.getLastName());
         System.out.println("  Salary: " + employee.getSalary());
      }
      tx.commit();
   } catch (HibernateException e) {
      if (tx!=null) tx.rollback();
      e.printStackTrace();
   } finally {
      session.close();
```

```
   }
}

/* Method to UPDATE salary for an employee */
public void updateEmployee(Integer EmployeeID, int salary ){
   Session session = factory.openSession();
   Transaction tx = null;

   try {
      tx = session.beginTransaction();
      Employee employee = (Employee)session.get(Employee.class, EmployeeID);
      employee.setSalary( salary );
                        session.update(employee);
      tx.commit();
   } catch (HibernateException e) {
      if (tx!=null) tx.rollback();
      e.printStackTrace();
   } finally {
      session.close();
   }
}

/* Method to DELETE an employee from the records */
public void deleteEmployee(Integer EmployeeID){
   Session session = factory.openSession();
   Transaction tx = null;
```

```
    try {
       tx = session.beginTransaction();
       Employee employee = (Employee)session.get(Employee.class, EmployeeID);
       session.delete(employee);
       tx.commit();
    } catch (HibernateException e) {
       if (tx!=null) tx.rollback();
       e.printStackTrace();
    } finally {
       session.close();
    }
  }
}
```

create Application Class

Finally, we will create our application class with the main() method to run the application. We will use this application to save few Employee's records and then we will apply CRUD operations on those records.

```
import java.util.List;
import java.util.Date;
import java.util.Iterator;

import org.hibernate.HibernateException;
import org.hibernate.Session;
import org.hibernate.Transaction;
import org.hibernate.SessionFactory;
```

```
import org.hibernate.cfg.Configuration;

public class ManageEmployee {
  private static SessionFactory factory;
  public static void main(String[] args) {

    try {
      factory = new Configuration().configure().buildSessionFactory();
    } catch (Throwable ex) {
      System.err.println("Failed to create sessionFactory object." + ex);
      throw new ExceptionInInitializerError(ex);
    }

    ManageEmployee ME = new ManageEmployee();

    /* Add few employee records in database */
    Integer empID1 = ME.addEmployee("Zara", "Ali", 1000);
    Integer empID2 = ME.addEmployee("Daisy", "Das", 5000);
    Integer empID3 = ME.addEmployee("John", "Paul", 10000);

    /* List down all the employees */
    ME.listEmployees();

    /* Update employee's records */
    ME.updateEmployee(empID1, 5000);
```

```
   /* Delete an employee from the database */
   ME.deleteEmployee(empID2);

   /* List down new list of the employees */
   ME.listEmployees();
}

/* Method to CREATE an employee in the database */
public Integer addEmployee(String fname, String lname, int salary){
   Session session = factory.openSession();
   Transaction tx = null;
   Integer employeeID = null;

   try {
      tx = session.beginTransaction();
      Employee employee = new Employee(fname, lname, salary);
      employeeID = (Integer) session.save(employee);
      tx.commit();
   } catch (HibernateException e) {
      if (tx!=null) tx.rollback();
      e.printStackTrace();
   } finally {
      session.close();
   }
   return employeeID;
}
```

```
/* Method to  READ all the employees */
public void listEmployees( ){
   Session session = factory.openSession();
   Transaction tx = null;

   try {
      tx = session.beginTransaction();
      List employees = session.createQuery("FROM Employee").list();
      for (Iterator iterator = employees.iterator(); iterator.hasNext();){
         Employee employee = (Employee) iterator.next();
         System.out.print("First Name: " + employee.getFirstName());
         System.out.print("  Last Name: " + employee.getLastName());
         System.out.println("  Salary: " + employee.getSalary());
      }
      tx.commit();
   } catch (HibernateException e) {
      if (tx!=null) tx.rollback();
      e.printStackTrace();
   } finally {
      session.close();
   }
}

/* Method to UPDATE salary for an employee */
public void updateEmployee(Integer EmployeeID, int salary ){
```

```
   Session session = factory.openSession();
   Transaction tx = null;

   try {
      tx = session.beginTransaction();
      Employee employee = (Employee)session.get(Employee.class, EmployeeID);
      employee.setSalary( salary );
                        session.update(employee);
      tx.commit();
   } catch (HibernateException e) {
      if (tx!=null) tx.rollback();
      e.printStackTrace();
   } finally {
      session.close();
   }
}

/* Method to DELETE an employee from the records */
public void deleteEmployee(Integer EmployeeID){
   Session session = factory.openSession();
   Transaction tx = null;

   try {
      tx = session.beginTransaction();
      Employee employee = (Employee)session.get(Employee.class, EmployeeID);
      session.delete(employee);
```

```
        tx.commit();
      } catch (HibernateException e) {
        if (tx!=null) tx.rollback();
        e.printStackTrace();
      } finally {
        session.close();
      }
    }
}
```

Conclusion :Hence we implemented hibernate program.

9.**Datbase operations using hibernate.**
Aim :To implement database operation using hibernate.

Objective :-Students will able to implement db operation in hibernate

Tools required:jdk1.6.netbaeans

Theory: 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.
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:

Program:-

```
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();
```

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();
```

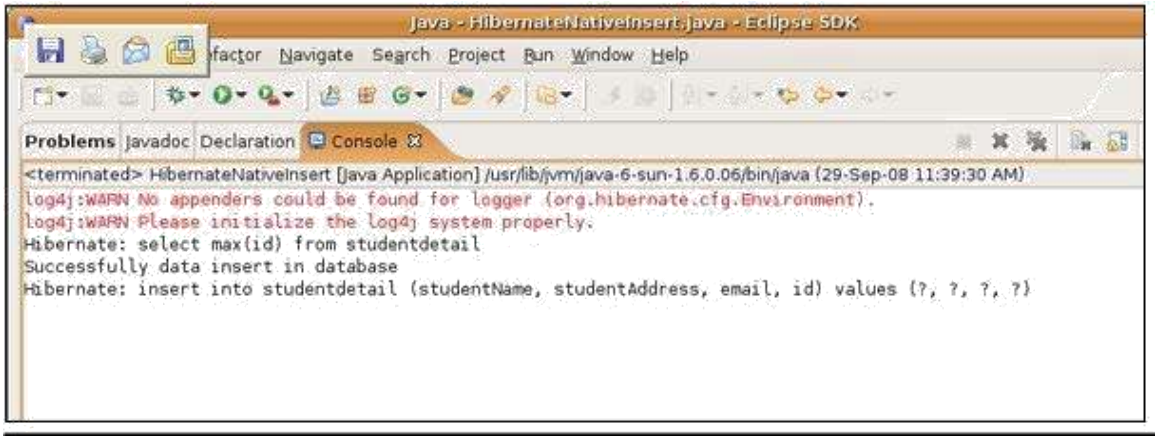
Hibernate native uses the Plain Old Java Objects (POJOs) classes to map to the database table. We can configure the variables to map to the database column

"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;
}

}
```

Output:



Conclusion:Hence we implemented database operation in hibernate.

10. Web Services

Aim : Program to create simple web service.

Objective :-Students will able to implement web services.

Tools Required:jdk1.6,netbeans

Theory:

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.

WebServiceEndpointInterface

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;

}
```

StatelessSessionBeanImplementationClass

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) {}

}
```

Building Hello ServiceBe 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

Creating the Application

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

PackagingtheEnterpriseBean

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.

Deploying theEnterpriseApplication

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.

Building t he Web Service Client

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

Running the WebServiceClient

To run the client, go to the

<INSTALL>/j2eetutorial14/examples/jaxrpc/staticstub/ directory and enter

asant run

The client should display the following line:

Output:

Hello Duke! (from HelloServiceBean)

Conclusion: Thus, we learnt how to create simple web service.

11.RMI Application.

Aim-To implement Remote method invocation

Objective :-Students will able to implement RMI application.

Tools required:JDK1.6,netbeans

Theory:-

RMI

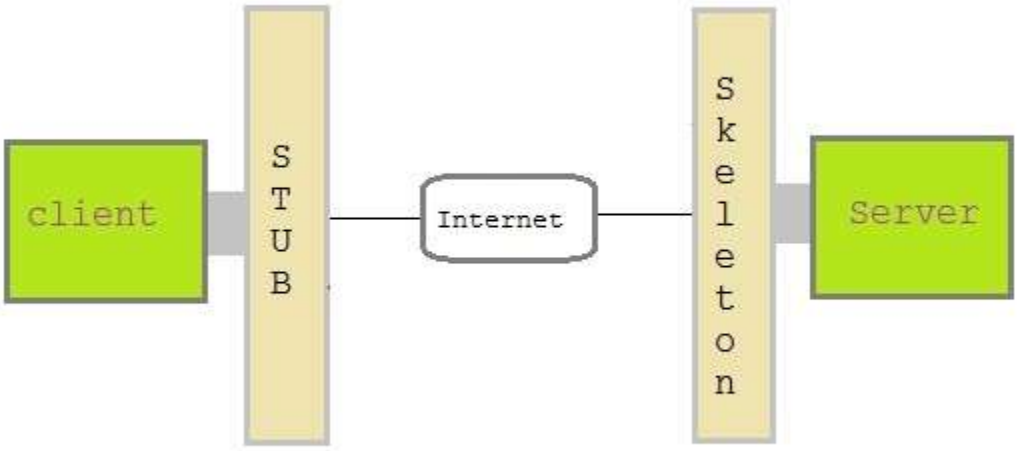
Remote method invocation(RMI) allow a java object to invoke method on an object running on another machine. RMI provide remote communication between java program. RMI is used for building distributed application.

Concept of RMI application

A RMI application can be divided into two part,**Client** program and **Server** program.
A **Server** program creates some remote object, make their references available for the client to invoke method on it. A **Client** program make request for remote objects on server and invoke method on them. **Stub** and **Skeleton** are two important object used for communication with remote object.

Stub and Skeleton

Stub act as a gateway for Client program. It resides on Client side and communicate with **Skeleton**object. It establish the connection between remote object and transmit request to it.



Skeleton object resides on server program. It is responsible for passing request from **Stub** to remote object.

Creating a Simple RMI application involves following steps

- Define a remote interface.
- Implementing remote interface.
- create and start remote application
- create and start client application

Define a remote interface

A remote interface specifies the methods that can be invoked remotely by a client. Clients program communicate to remote interfaces, not to classes implementing it. To be a remote interface, a interface must extend the **Remote** interface of **java.rmi** package.

```
import java.rmi.*;
public interface AddServerInterface extends Remote
{
public int sum(int a,int b);
}
```

Implementation of remote interface

For implementation of remote interface, a class must either extend **UnicastRemoteObject** or use exportObject() method of **UnicastRemoteObject** class.

```
import java.rmi.*;
import java.rmi.server.*;
public class Adder extends UnicastRemoteObject implements AddServerInterface
{
Adder()throws RemoteException{
super();
}
public int sum(int a,int b)
{
return a+b;
}
}
```

Create AddServer and host rmi service

You need to create a server application and host rmi service **Adder** in it. This is done using rebind()method of **java.rmi.Naming** class. rebind() method take two arguments, first represent the name of the object reference and second argument is reference to instance of **Adder**

```
import java.rmi.*;
import java.rmi.registry.*;
public class AddServer{
public static void main(String args[]){
try{
AddServerInterface addService=new Adder();
Naming.rebind("AddService",addService);
//addService object is hosted with name AddService.

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

```
}
```

Create client application

Client application contains a java program that invokes the lookup() method of the **Naming** class. This method accepts one argument, the **rmi** URL and returns a reference to an object of type **AddServerInterface**. All remote method invocation is done on this object.

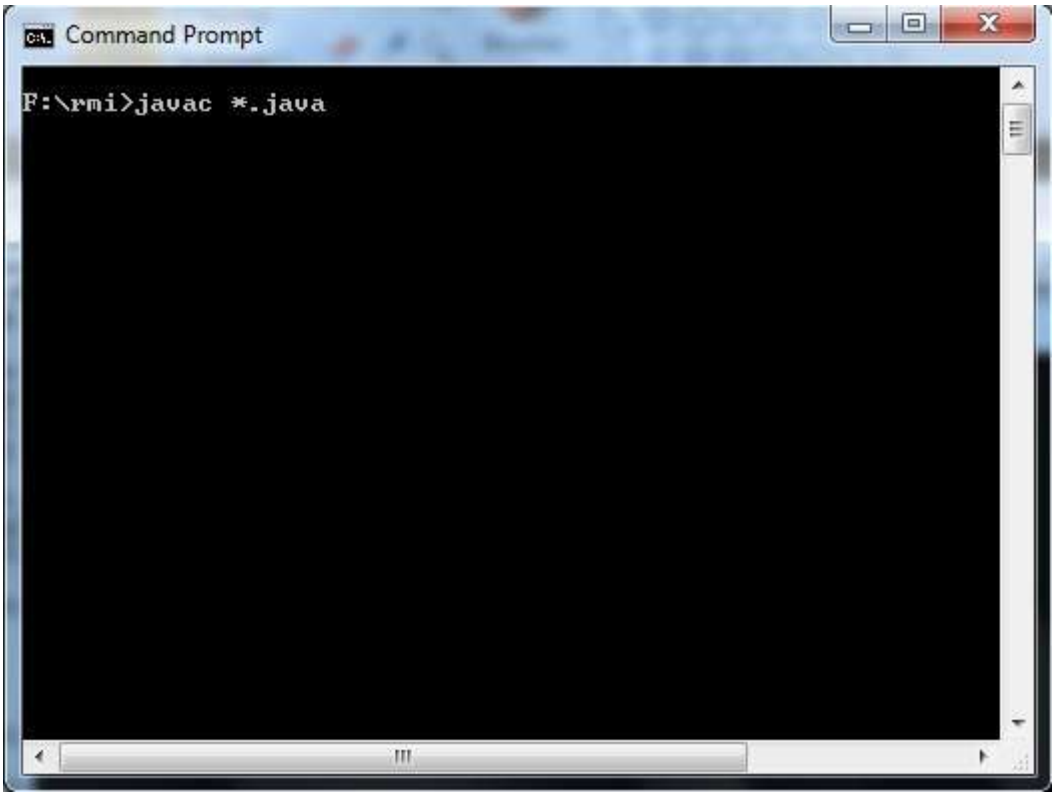
```
import java.rmi.*;
public class Client{
public static void main(String args[]){
try{
AddServerInterface st=(AddServerInterface)Naming.lookup("rmi://"+args[0]+"/AddService");
System.out.println(st.sum(25,8));
}catch(Exception e){System.out.println(e);}
}
}
```

Output(Steps to run this RMI application)

Save all the above java file into a directory and name it as "rmi"

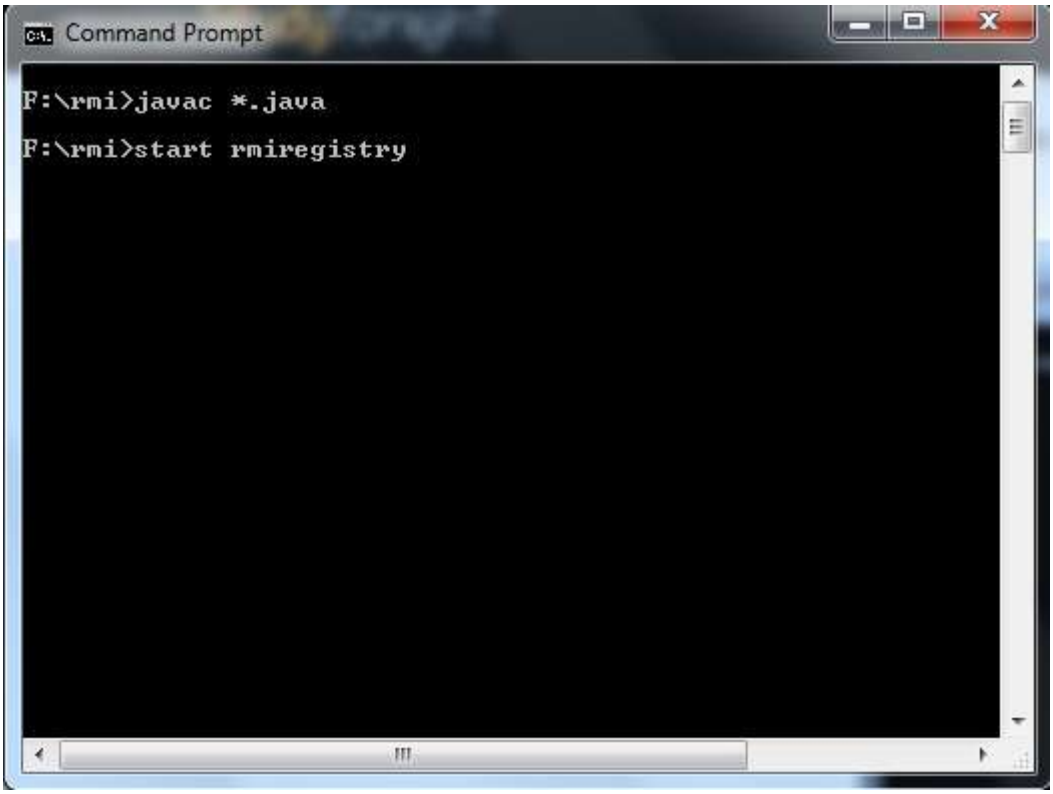
- compile all the java files

```
javac *.java
```



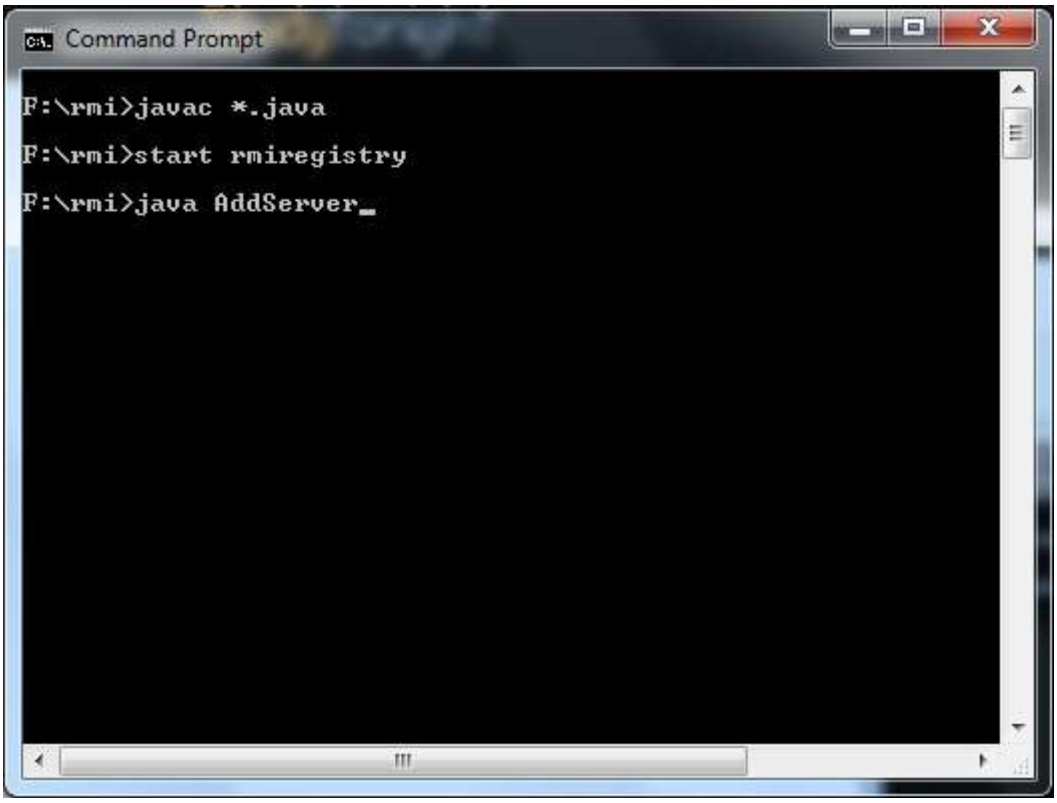
- Start RMI registry

```
start rmiregistry
```



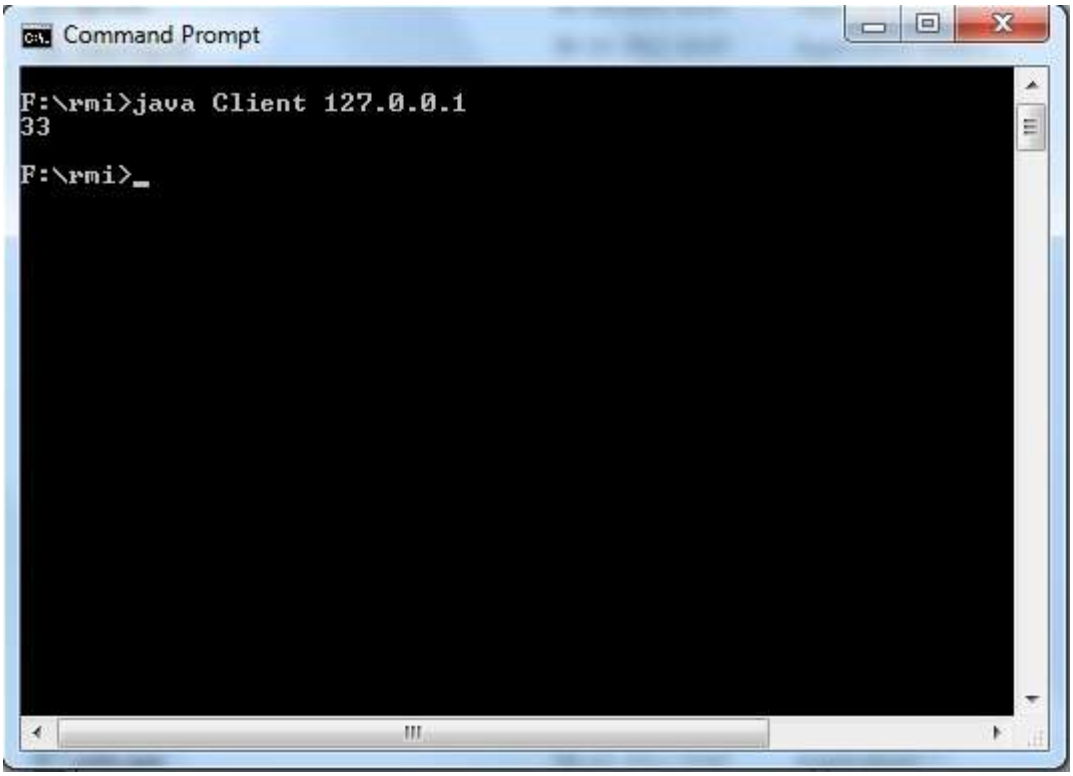
- Run Server file

```
java AddServer
```



- Run Client file in another command prompt abd pass local host port number at run time

```
java Client 127.0.0.1
```



Conclusion:Hence we implemented Remote method invocation.

11.Java Mailing System.

Aim:To implement Java mailing system.

Objective :-Students will able to implement java mailing system.

Tools required:-JDK1.6,netbeans.

Theory:- There are various ways to send email using JavaMail API. For this purpose, you must have SMTP server that is responsible to send mails.

You can use one of the following techniques to get the SMTP server:

- Install and use any SMTP server such as Postcast server, Apache James server, cmail server etc. (or)
- Use the SMTP server provided by the host provider e.g. my SMTP server is mail.javatpoint.com (or)
- Use the SMTP Server provided by other companies e.g. gmail etc.

Steps to send email using JavaMail API

There are following three steps to send email using JavaMail. They are as follows:

1. **Get the session object** that stores all the information of host like host name, username, password etc.
2. **compose the message**
3. **send the message**

Simple example of sending email in Java

In this example, we are going to learn how to send email by SMTP server installed on the machine e.g. Postcast server, Apache James server, Cmail server etc. If you want to send email by using your SMTP server provided by the host provider, see the example after this one.

For sending the email using JavaMail API, you need to load the two jar files:

mail.jar
activation.jar

download these jar files or go to the Oracle site to download the latest version.

Program:-

```
import java.util.*;
import javax.mail.*;
import javax.mail.internet.*;
import javax.activation.*;

public class SendEmail
```

```
{
 public static void main(String [] args){
     String to = "sonoojaiswal1988@gmail.com";//change accordingly
     String from = "sonoojaiswal1987@gmail.com";change accordingly
     String host = "localhost";//or IP address

    //Get the session object
     Properties properties = System.getProperties();
     properties.setProperty("mail.smtp.host", host);
     Session session = Session.getDefaultInstance(properties);

    //compose the message
     try{
       MimeMessage message = new MimeMessage(session);
       message.setFrom(new InternetAddress(from));
       message.addRecipient(Message.RecipientType.TO,new InternetAddress(to));
       message.setSubject("Ping");
       message.setText("Hello, this is example of sending email  ");

       // Send message
       Transport.send(message);
       System.out.println("message sent successfully....");

     }catch (MessagingException mex) {mex.printStackTrace();}
   }
}
```

Output:-

Load the jar file c:\> **set classpath=mail.jar;activation.jar;.;**

compile the source file c:\> **javac SendEmail.java**

run by c:\> **java SendEmail**

Conclusion: Hence we implemented Java mailing system.

13.Struts application.

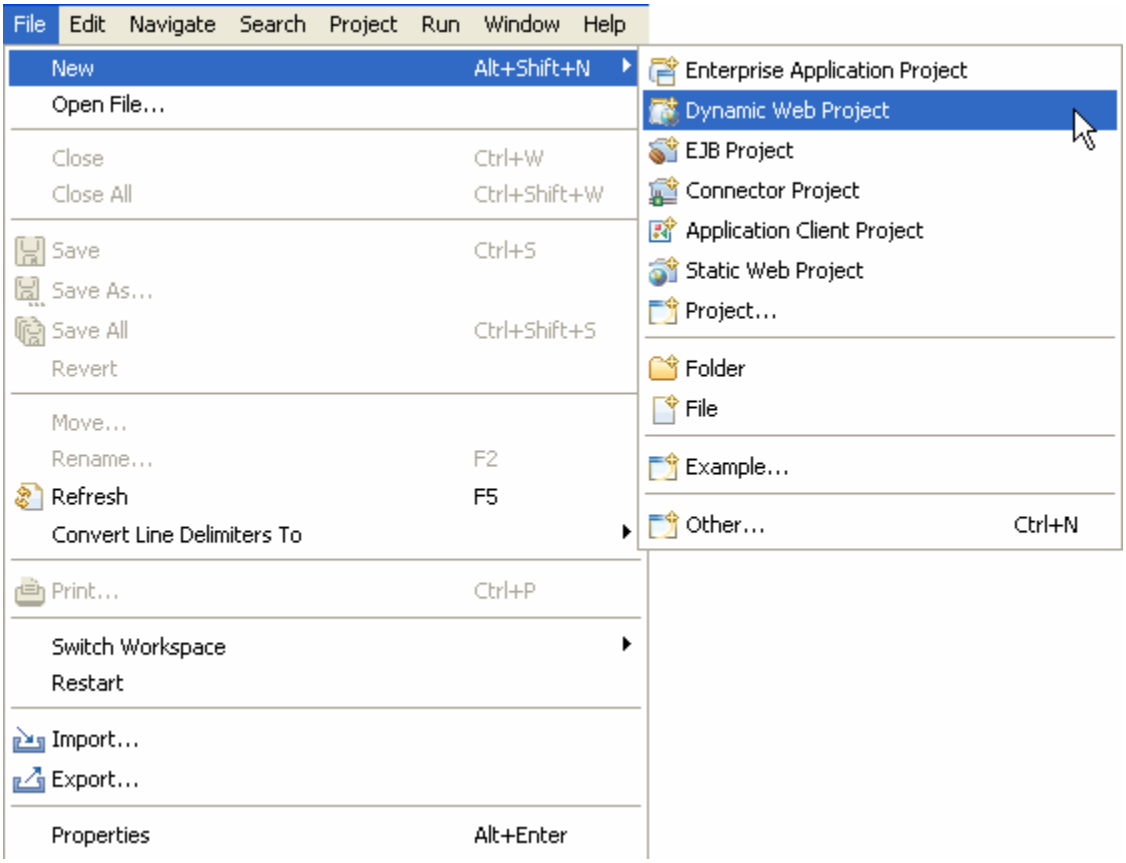
Aim: To implement struts application.

Objective :-Students will able to implement struts application.

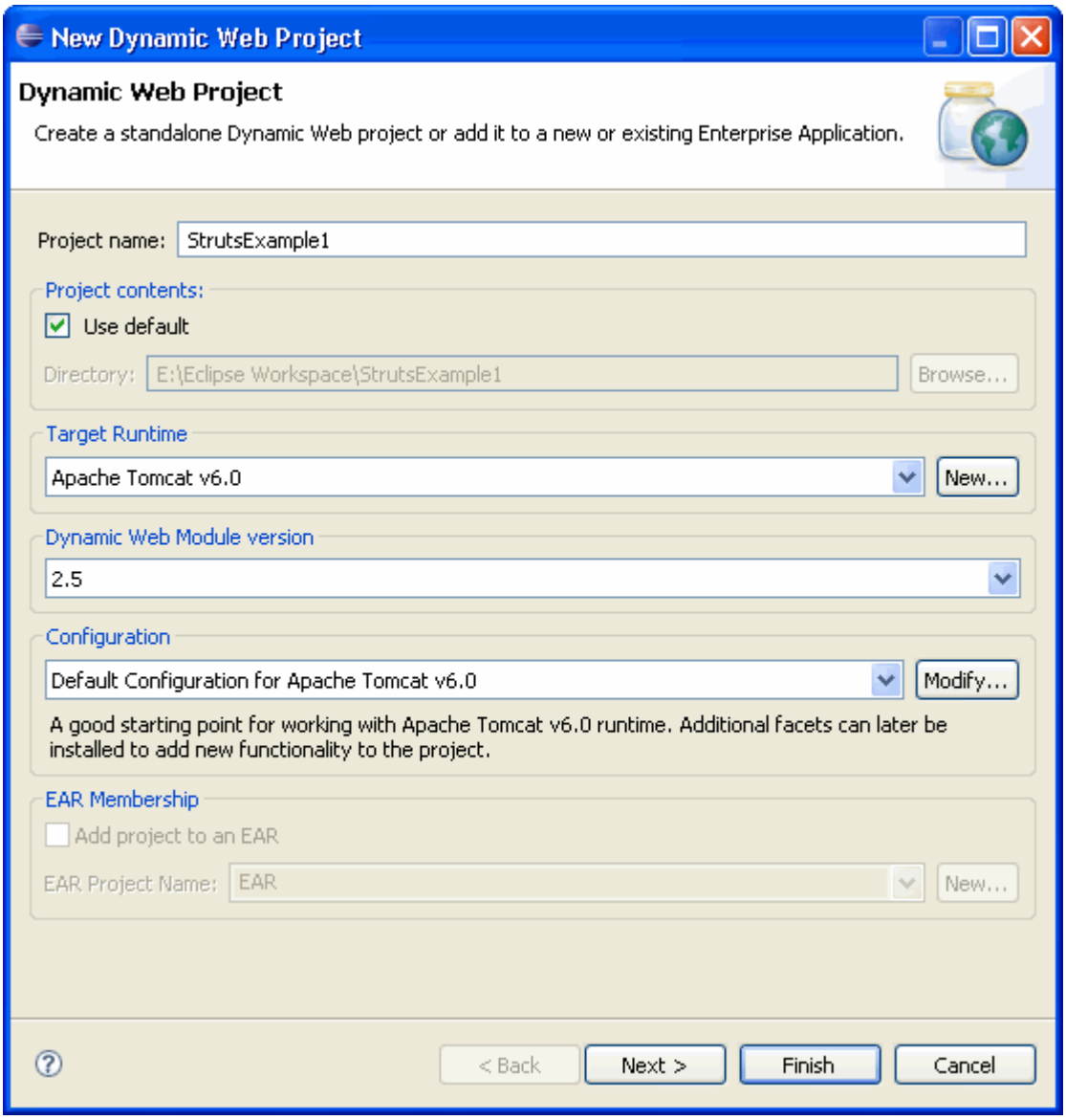
Tools required:netbeans,JDK 1.6

Theory:-

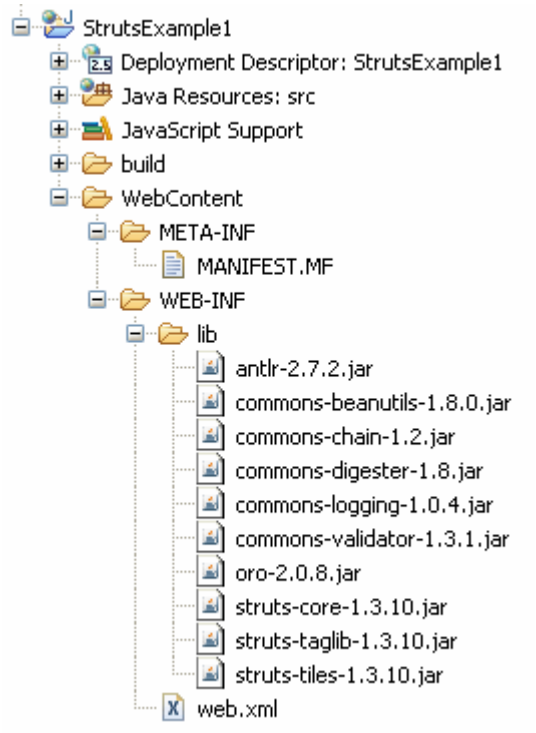
First create a new project, go to File->New and select DynamicWebProject.



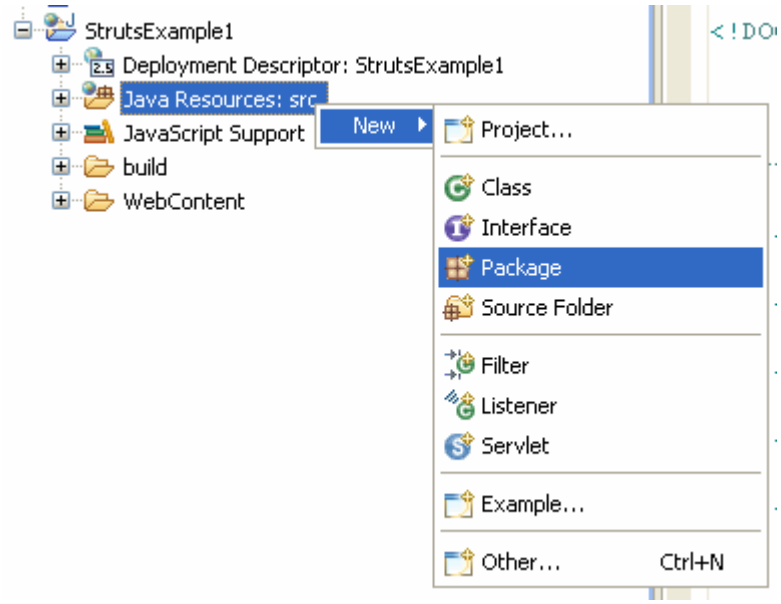
Enter the project name and click the Finish button.



Add the following jar files to the WEB-INF\lib directory.

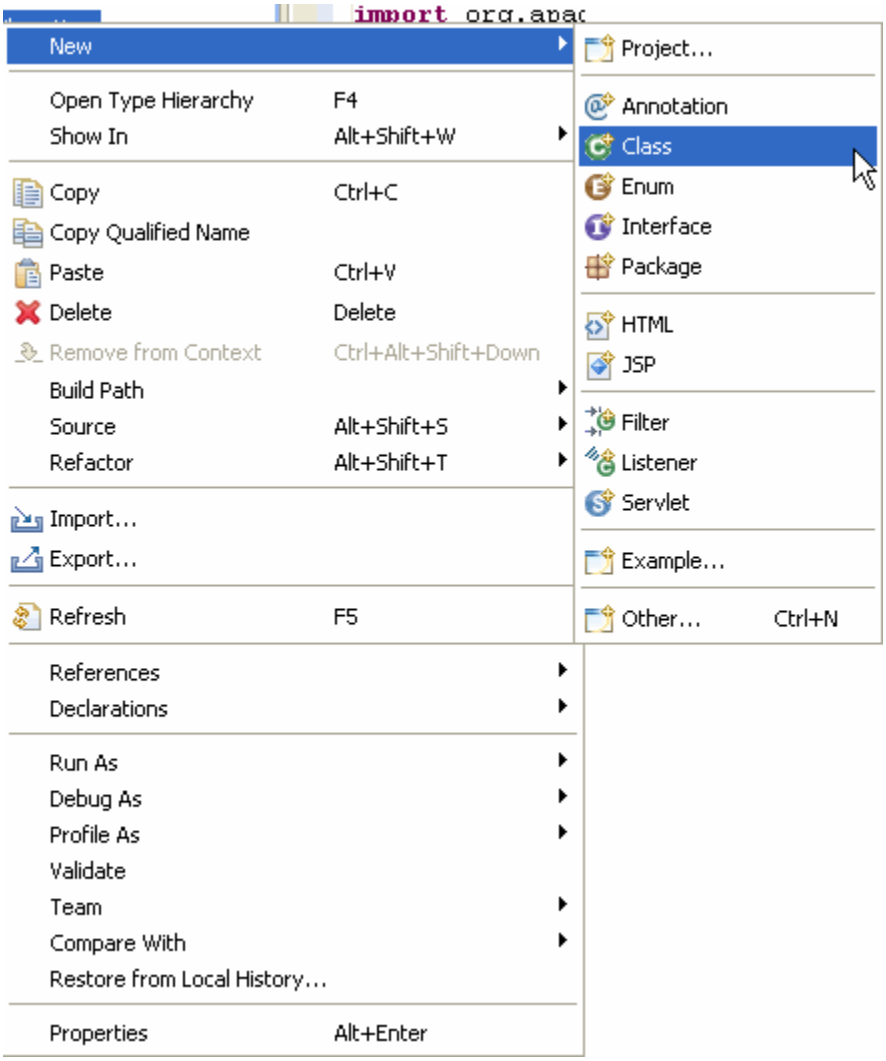


Right click the src folder and select New->Package.

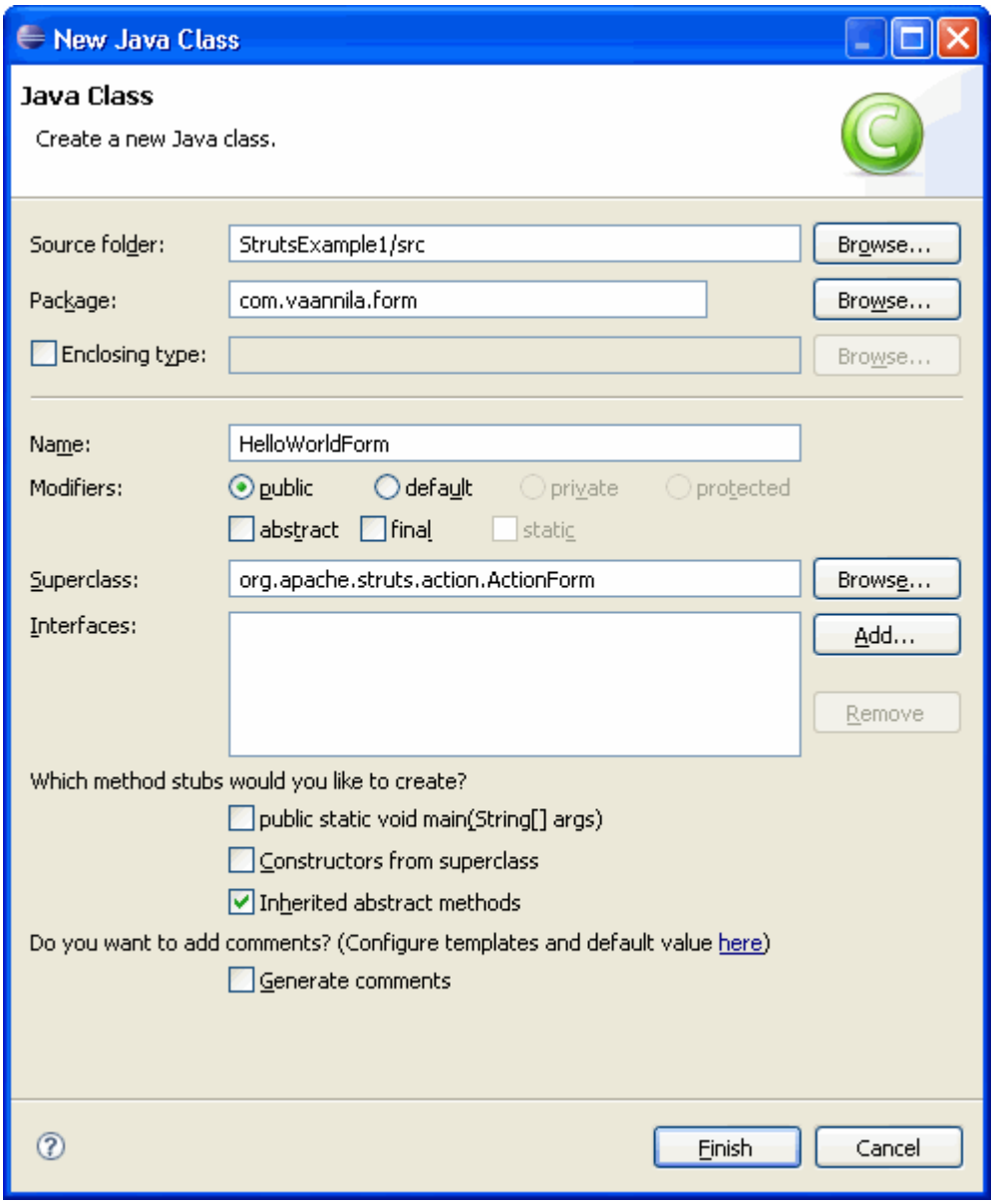


Enter the package name as *com.vaannila.form* and click Finish.

Now right click the newly created package and select New->Class.



Enter the class name as *HelloWorldForm* and the superclass name as *org.apache.struts.action.ActionForm* and click Finish.



In the HelloWorldForm class add the following code.

```
package com.vaannila.form;

import org.apache.struts.action.ActionForm;

public class HelloWorldForm extends ActionForm {

    private static final long serialVersionUID = -4735625968524520 21L;

    private String message;

    public String getMessage() {
```

```
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }
}
```

In the same way create a new package *com.vaannila.action* and create a *HelloWorldAction* class extending *org.apache.struts.action.Action*. Add the following code to the action class and save it.

```
package com.vaannila.action;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

import com.vaannila.form.HelloWorldForm;

public class HelloWorldAction extends Action {

@Override
public ActionForward execute(ActionMapping mapping, ActionForm form, HttpServletRequest
request, HttpServletResponse response) throws Exception {
    HelloWorldForm hwForm = (HelloWorldForm) form;
    hwForm.setMessage("Hello World");
    return mapping.findForward("success");
    }
}
```

Here we typecast the ActionForm to *HelloWorldForm* and set the message value.

Add the following entries in the *struts-config.xml* file.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>

<!DOCTYPE struts-config PUBLIC
        "-//Apache Software Foundation//DTD Struts Configuration 1.3//EN"
        "http://struts.apache.org/dtds/struts-config_1_3.dtd">

<struts-config>
```

```
    <form-beans>
            <form-bean name="helloWorldForm" type="com.vaannila.form.HelloWorldForm"/>
    </form-beans>

            <global-forwards>
        <forward name="helloWorld" path="/helloWorld.do"/>
    </global-forwards>

    <action-mappings>
        <action path="/helloWorld" type="com.vaannila.action.HelloWorldAction"
name="helloWorldForm">
            <forward name="success" path="/helloWorld.jsp" />
        </action>
    </action-mappings>

</struts-config>
```

Now configure the deployment descriptor. Add the following configuration information in the *web.xml* file.

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee" xmlns:web="http://java.sun.com/xml/ns/javaee/web-
app_2_5.xsd" xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd" id="WebApp_ID" version="2.5">
  <display-name>StrutsExample1</display-name>

  <servlet>
    <servlet-name>action</servlet-name>
    <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
    <init-param>
      <param-name>config</param-name>
      <param-value>/WEB-INF/struts-config.xml</param-value>
    </init-param>
    <load-on-startup>2</load-on-startup>
 </servlet>

  <servlet-mapping>
    <servlet-name>action</servlet-name>
    <url-pattern>*.do</url-pattern>
  </servlet-mapping>

  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>
  </welcome-file-list>
</web-app>
```

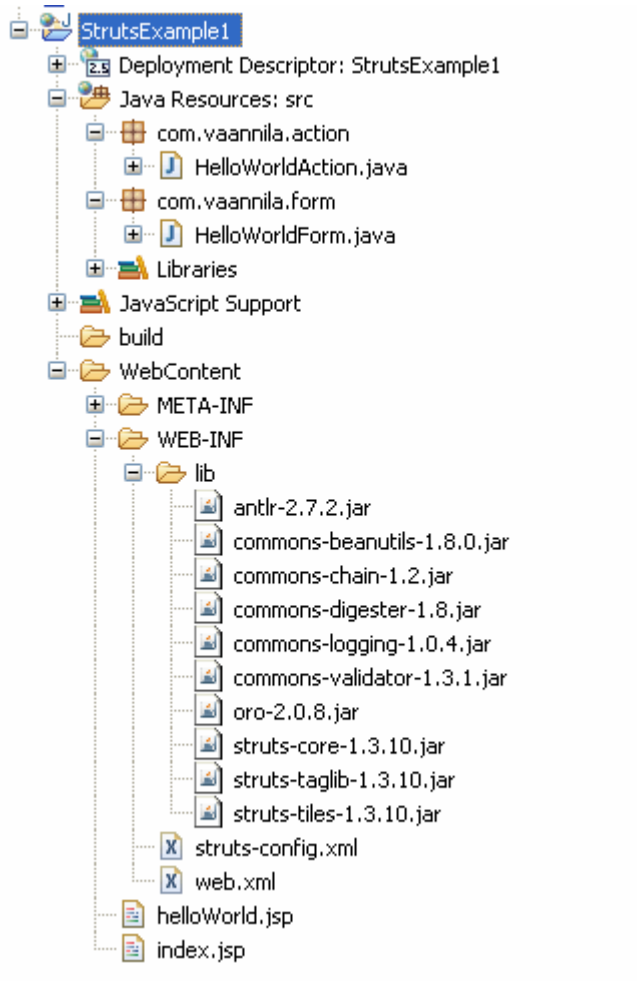
When we run the application the index.jsp page will be executed first. In the *index.jsp* page we redirect the request to the *helloWorld.do* URI, which inturn invokes the *HelloWorldAction*.

```
<%@ taglib uri="http://struts.apache.org/tags-logic" prefix="logic" %>
<logic:redirect forward="helloWorld"/>
```

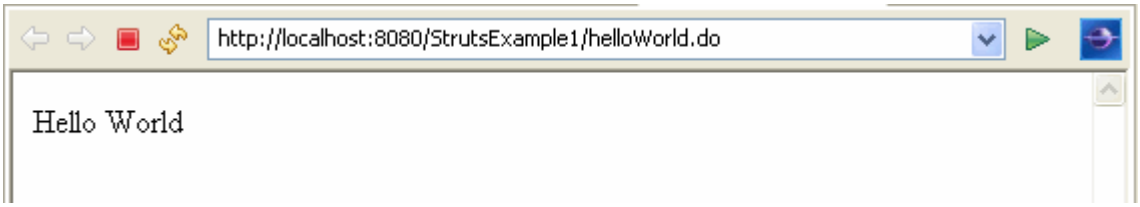
In the action class we return the ActionForward "*success*" which is mapped to the *helloWorld.jsp* page. In the *helloWorld.jsp* page we display the "Hello World" message.

```
<%@taglib uri="http://struts.apache.org/tags-bean" prefix="bean" %>
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Hello World</title>
</head>
<body>
<bean:write name="helloWorldForm" property="message"/>
</body>
</html>
```

After creating all the files the directory structure of the application looks like this.



On executing the application the "Hello World" message gets displayed to the user.



Conclusion :Hence we implemented struts application.

4. Quiz on subject:

- What is different between web server and application server?
- Which HTTP method is non-idempotent?
- What is the difference between GET and POST method?
- What is MIME Type?
- What is a web application and what is it's directory structure?
- What is a servlet?
- What are the advantages of Servlet over CGI?
- What are common tasks performed by Servlet Container?
- What is ServletConfig object?
- What is ServletContext object?
- What is difference between ServletConfig and ServletContext?
- What is Request Dispatcher?
- What is difference between PrintWriter and ServletOutputStream?
- Can we get PrintWriter and ServletOutputStream both in a servlet?
- How can we create deadlock situation in servlet?
- What is the use of servlet wrapper classes?
- What is SingleThreadModel interface?
- Do we need to override service() method?
- Is it good idea to create servlet constructor?
- What is difference between GenericServlet and HttpServlet?
- What is the inter-servlet communication?
- Are Servlets Thread Safe? How to achieve thread safety in servlets?
- What is servlet attributes and their scope?
- How do we call one servlet from another servlet?
- How can we invoke another servlet in a different application?
- What is difference between ServletResponse sendRedirect() and RequestDispatcher forward() method?
- Why HttpServlet class is declared abstract?
- What are the phases of servlet life cycle?
- What are life cycle methods of a servlet?
- why we should override only no-agrs init() method.
- What is URL Encoding?
- What are different methods of session management in servlets?
- What is URL Rewriting?
- How does Cookies work in Servlets?

- How to notify an object in session when session is invalidated or timed-out?
- What is the difference between encodeRedirectUrl and encodeURL?
- Why do we have servlet filters?
- What is the effective way to make sure all the servlets are accessible only when user has a valid session?
- Why do we have servlet listeners?
- How to handle exceptions thrown by application with another servlet?
- What is a deployment descriptor?
- How to make sure a servlet is loaded at the application startup?
- How to get the actual path of servlet in server?
- How to get the server information in a servlet?
- Write a servlet to upload file on server.
- How do we go with database connection and log4j integration in servlet?
- How to get the IP address of client in servlet?
- What are important features of Servlet 3?
- What are different ways for servlet authentication?
- How can we achieve transport layer security for our web application?
- What is JSP and why do we need it?
- What are the JSP lifecycle phases?
- What are JSP lifecycle methods?
- Which JSP lifecycle methods can be overridden?
- How can we avoid direct access of JSP pages from client browser?
- What are different types of comments in JSP?
- What is Scriptlet, Expression and Declaration in JSP?
- What are JSP implicit objects?
- Can we use JSP implicit objects in a method defined in JSP Declaration?
- Which implicit object is not available in normal JSP pages?
- What are the benefits of PageContext implicit object?
- How do we configure init params for JSP?
- Why use of scripting elements in JSP is discouraged?
- Can we define a class in a JSP Page?
- How can we disable java code or scripting in JSP page?
- Explain JSP Action Elements or Action Tags?
- What is difference between include directive and jsp:include action?
- What is JSP Expression Language and what are it’s benefits?
- What are JSP EL implicit objects and how it’s different from JSP implicit Objects?
- How to use JSP EL to get HTTP method name?
- What is JSP Standard Tag Library, provide some example usage?
- What are the types of JSTL tags?
- What is JSP Custom Tag and what are it’s components?
- Give an example where you need JSP Custom Tag?

- Why don’t we need to configure JSP standard tags in web.xml?
- How can we handle exceptions thrown by JSP service method?
- How do we catch exception and process it using JSTL?
- How do we print “
 creates a new line in HTML” in JSP?
- What is jsp-config in deployment descriptor?
- How to ignore the EL expression evaluation in a JSP?
- When will Container initialize multiple JSP/Servlet Objects?
- Can we use JavaScript with JSP Pages?
- How can we prevent implicit session creation in JSP?
- What is difference between JspWriter and Servlet PrintWriter?
- How can we extend JSP technology?
- Provide some JSP Best Practices?