Jawaharlal Nehru Engineering College

Laboratory Manual

ADVANCED JAVA PROGRAMMING

for

S.Y (M.C.A) Students

26, June 2019 – Rev 00 – MCA – ISO 9001 Tech Document

FORWARD

It is my great pleasure to present this laboratory manual for S.Y (M.C.A) students for the subject of Adv.JAVA.

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 9001 certification and it is our endure to technically equip our students taking the advantage of the procedural aspects of ISO 9001 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.

Prof. Dr.H.H.SHINDE
Principal

LABORATORY MANUAL CONTENTS

This manual is intended for the students S.Y (M.C.A) engineering for the subject of Adv.JAVA. In the subject of this manual typically contains practical/Lab Sessions we have dealt with Adv.JAVA.

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

Good Luck for your Enjoyable Laboratory Sessions

Prof. V.S.Agrawal

Asst.Prof

Dr.S.S.Deshmukh

HOD MCA

DOs and DON'T DOs in Laboratory:

1. Do not handle any equipment before reading the instructions/Instruction manuals

2. Read carefully the power ratings of the equipment before it is switched on whether ratings 230 V/50Hz or 115V/60 Hz. For Indian equipments, the power ratings are normally 230V/50Hz. If you have equipment with 115/60 Hz ratings, do not insert power plug, as our normal supply is 230V/50 Hz, which will damage the equipment.

3. Observe type of sockets of equipment power to avoid mechanical damage

4. Do not forcefully place connectors to avoid the damage

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

Instruction for Laboratory Teachers::

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

(2 Hours):- 1Practical

1. Lab Exercises:

1 JDBC

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

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

We have discussed the above four drivers in the next chapter.

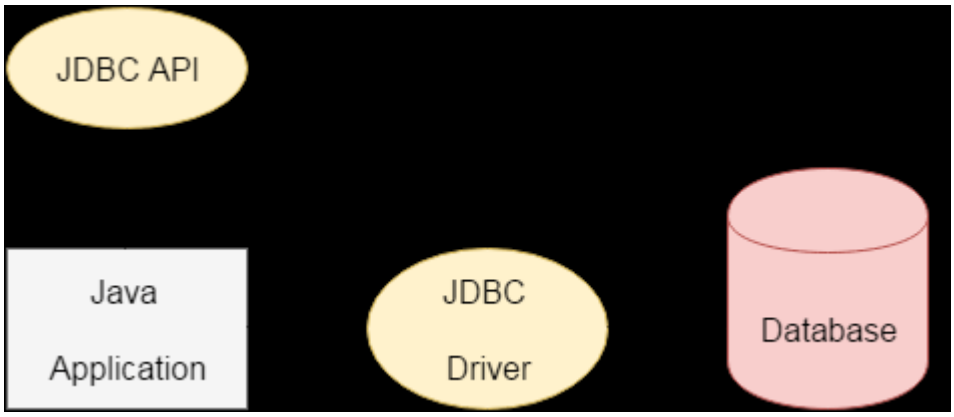
We can use JDBC API to access tabular data stored in any relational database. By the help of JDBC API, we can save, update, delete and fetch data from the database. It is like Open Database Connectivity (ODBC) provided by Microsoft.

Why Should We Use JDBC

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

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

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



Steps:-

1.Define the driver class to be used.

Class.forName(“sun.jdbc.odbc.JdbcOdbcDriver”);

2.Create the connection

Connection con=DriverManager.getConnection(“jdbc:odbc:mydsn”);

Here mydsn is the name of DSN which should be configured before to run this code.

3.Create the statement

Statement stmt=con.createStatement();

4.Define ResultSet class if required

ResultSet rs=stmt.executeQuery(“select * from emp”);

5. close the connection.

Note:- Handle ClassNotFoundException & SQLException

//source code

```
package Assignment7;
import java.sql.*;
import java.util.Scanner;

public class Db {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
try{Scanner s;
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        Connection con=DriverManager.getConnection("jdbc:odbc:my");
        Statement stmt=con.createStatement();


         s=new Scanner(System.in);
       while(true)
       {
            System.out.println("enter ur choice 1-select 2-insert 3-update 4-
delete 5:-exit");
           int ch=s.nextInt();
        switch (ch) {
          case 1:

              //System.out.println("enter rno");
           int rno;
              //int rno=s.nextInt();
           ResultSet rs=stmt.executeQuery("select count(*) from student where
name='abc'");
              while(rs.next())
              {
                  System.out.println(rs.getInt(1)+" "+rs.getString(2)+ "
"+rs.getString(3));
              }
              break;

          case 2:System.out.println("enter rno");
                rno=s.nextInt();
                System.out.println("enter name");
                String name=s.next();
                System.out.println("enteraddress");
                String address=s.next();
```

```
                stmt.executeUpdate("insert into student
values("+rno+",'"+name+"','"+address+"')");
                System.out.println("data inserted successfully...");
          break;
          case 3:System.out.println("enter rno");
            rno=s.nextInt();
            System.out.println("enter address to update");
             address=s.next();
            stmt.executeUpdate("update student set address='"+address+"'
where rno="+rno+"");
              System.out.println("data updated successfully...");
            break;
          case 4:System.out.println("enter rno to delete");
            rno=s.nextInt();
             stmt.executeUpdate("delete * from student where rno="+rno+"");
              System.out.println("data deleted successfully...");
            break;

          case 5:con.close();System.exit(0);
          default: System.out.println("wrong choice...");
                    break;
          }
       }
}catch(SQLException e){System.out.println("inavlid  sql query");}
catch(ClassNotFoundException e){System.out.println("invalid database
driver");}
catch(Exception e){System.out.println(e.getMessage());}

    }}
```

Input:- display

Output:- 1 a 2 b 3 c 4 d

Input:- updated 4 u

Input:- display

Output:- 1 a 2 b 3 c 4 u

(2 Hours):- 2Practical

2. Lab Exercises: -

A program for Servlet (Eclipse framework)

Servlet technology is used to create a web application (resides at server side and generates a dynamic web page).

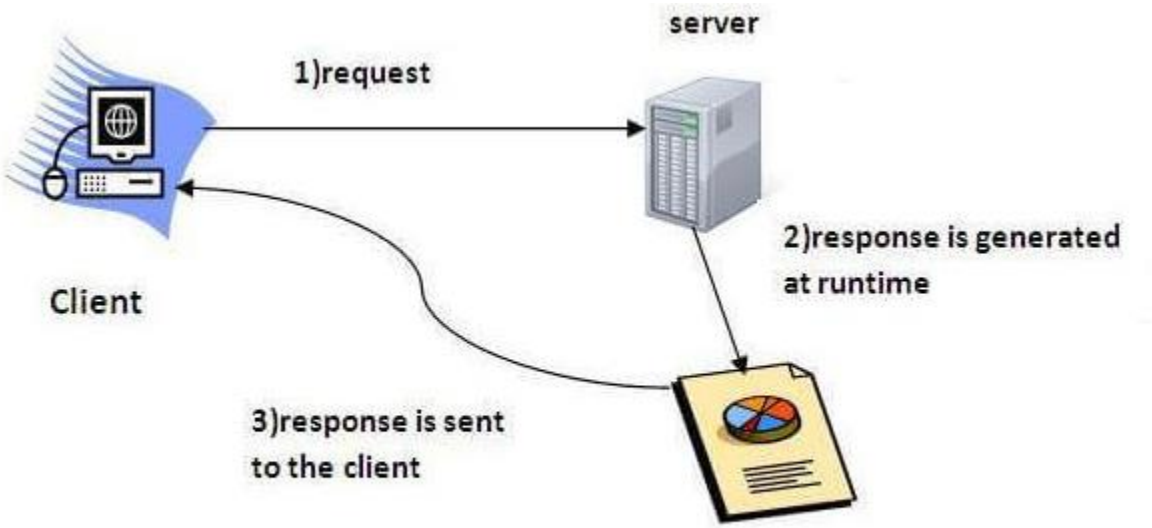
Servlet technology is robust and scalable because of java language. Before Servlet, CGI (Common Gateway Interface) scripting language was common as a server-side programming language. However, there were many disadvantages to this technology. We have discussed these disadvantages below.

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

What is a Servlet?

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

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



Steps:

1. Create New dynamic web Project.
   Right click on project. Go to build path->configure build path->libraies menu- >say add new external jar files.
2. Select servlet-api.jar file from c:\program files\apach tomcat\lib & say ok.
3. Right click on project and say new package .
4. Give the Name for package
5. Right click on package and say new Servlet.
6. Give the name to the servlet
7. Write the logic in doGet() or doPost() method.
8. Build the Project.

//Source Code

```
package Assign1;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
```

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

/**
 * Servlet implementation class Serv1
 */
@WebServlet(name = "Servlet1", urlPatterns = { "/Servlet1" })

public class Serv1 extends HttpServlet {
	private static final long serialVersionUID = 1L;

   /**
    * @see HttpServlet#HttpServlet()
    */
   public Serv1() {
      super();
      // TODO Auto-generated constructor stub
   }

	/**
	 * @see HttpServlet#doGet(HttpServletRequest request,
HttpServletResponse response)
	 */
	protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
		// TODO Auto-generated method stub
		PrintWriter pw=response.getWriter();
	pw.print("MGM's JNEC Aurangabad");
		}

	/**
	 * @see HttpServlet#doPost(HttpServletRequest request,
HttpServletResponse response)
	 */
	protected void doPost(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
		// TODO Auto-generated method stub
	}

}
```

Input:- to display hello world

Output:- hello world

(2 Hours):- 3Practical

Lab Exercises *3rd: -* To study State Management.

Steps:-

1 Create New dynamic web Project.
Right click on project. Go to build path->configure build path->libraies menu- >say add new external jar files.

2 Select servlet-api.jar file from c:\program files\apach tomcat\lib & say ok.

3 Right click on project and say new package .

4 Give the Name for package

5 Right click on package and say new Servlet.

6 Give the name to the servlet

7 Write the logic in doGet() or doPost() method.

8 Cookie class can be used to maintain the state
Cookie c1=new Cookie("user","abc");

9 add cookie class into the response object.
Response.addCookie(c1);

10 display all the cookies using request.getCookies();

Cookies in Servlet

A **cookie** is a small piece of information that is persisted between the multiple client requests.

A cookie has a name, a single value, and optional attributes such as a comment, path and domain qualifiers, a maximum age, and a version number.

How Cookie works

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

Types of Cookie

There are 2 types of cookies in servlets.

1. Non-persistent cookie
2. Persistent cookie

Non-persistent cookie

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

Persistent cookie

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

Advantage of Cookies

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

Disadvantage of Cookies

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

//Source Code

Serv4.java

```
package Assign1;
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
```

```
/**
 * Servlet implementation class Serv4
 */
@WebServlet("/Serv4")
public class Serv4 extends HttpServlet {
	private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#HttpServlet()
     */
    public Serv4() {
        super();
        // TODO Auto-generated constructor stub
    }

	/**
	 * @see HttpServlet#doGet(HttpServletRequest request,
HttpServletResponse response)
	 */
	protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
		// TODO Auto-generated method stub

	String s=request.getParameter("empid");
	Cookie c1=new Cookie("empid",s);
	PrintWriter pw=response.getWriter();
	response.addCookie(c1);
	pw.print("<a href=Serv5>serv5</a>");
	}

	/**
	 * @see HttpServlet#doPost(HttpServletRequest request,
HttpServletResponse response)
	 */
	protected void doPost(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
		// TODO Auto-generated method stub
	}

}
```

Serv5.java

```
package Assign1;
import java.io.IOException;
import java.io.PrintWriter;
```

```
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.Cookie;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
/**
 * Servlet implementation class Serv5
 */
@WebServlet("/Serv5")
public class Serv5 extends HttpServlet {
	private static final long serialVersionUID = 1L;
	 /**
    * @see HttpServlet#HttpServlet()
    */
   public Serv5() {
      super();
      // TODO Auto-generated constructor stub
   }
	 /**
	  * @see HttpServlet#doGet(HttpServletRequest request,
HttpServletResponse response)
	  */
	protected void doGet(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
		// TODO Auto-generated method stub
		PrintWriter pw=response.getWriter();
		Cookie c[]=request.getCookies();
		for(int i=0;i<c.length;i++)
		{	if(c[i].getName().equals("empid"))
			pw.print(c[i].getValue());
		}
		}
		/**
		 * @see HttpServlet#doPost(HttpServletRequest request,
HttpServletResponse response)
		 */
		protected void doPost(HttpServletRequest request,
HttpServletResponse response) throws ServletException, IOException {
			// TODO Auto-generated method stub
		}}
```

Input:- say empid 5 is set into cookie

Output:- your empid is 5

(2 Hours):- 4Practical

Lab Exercises 4*th:* - Servlet with database connectivity

In this session we will connect servlet with the database. Any relational database we can use to perform DDL, DML, DCL operations.

Steps:-

1) create new dynamic web project

2) Right click to source ad say new Servlet

3) give name to the servlet

4) write the code either in doGet() or doPost() method

5) maintain DSN in case of my access database.

```
//source code
// Loading required libraries
import java.io.*;
import java.util.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.sql.*;

public class DatabaseAccess extends HttpServlet{
```

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

    throws ServletException, IOException {


    // JDBC driver name and database URL

    static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";

    static final String DB_URL="jdbc:mysql://localhost/TEST";


    //  Database credentials

    static final String USER = "root";

    static final String PASS = "password";


    // Set response content type

    response.setContentType("text/html");

    PrintWriter out = response.getWriter();

    String title = "Database Result";


    String docType =

      "<!doctype html public \"-//w3c//dtd html 4.0 " + "transitional//en\">\n";


    out.println(docType +

      "<html>\n" +
```

```
        "<head><title>" + title + "</title></head>\n" +
        "<body bgcolor = \"#f0f0f0\">\n" +
        "<h1 align = \"center\">" + title + "</h1>\n");
      try {
        // Register JDBC driver
        Class.forName("com.mysql.jdbc.Driver");

        // Open a connection
        Connection conn = DriverManager.getConnection(DB_URL, USER,
PASS);

        // Execute SQL query
        Statement stmt = conn.createStatement();
        String sql;
        sql = "SELECT id, first, last, age FROM Employees";
        ResultSet rs = stmt.executeQuery(sql);

        // Extract data from result set
        while(rs.next()){
          //Retrieve by column name
          int id  = rs.getInt("id");
          int age = rs.getInt("age");
```

```
      String first = rs.getString("first");
      String last = rs.getString("last");

      //Display values
      out.println("ID: " + id + "<br>");
      out.println(", Age: " + age + "<br>");
      out.println(", First: " + first + "<br>");
      out.println(", Last: " + last + "<br>");
    }
    out.println("</body></html>");

    // Clean-up environment
    rs.close();
    stmt.close();
    conn.close();
  } catch(SQLException se) {
    //Handle errors for JDBC
    se.printStackTrace();
  } catch(Exception e) {
    //Handle errors for Class.forName
    e.printStackTrace();
  } finally {
```

```
        //finally block used to close resources
        try {
           if(stmt!=null)
              stmt.close();
        } catch(SQLException se2) {
        } // nothing we can do
        try {
           if(conn!=null)
           conn.close();
        } catch(SQLException se) {
           se.printStackTrace();
        } //end finally try
     } //end try
  }
}
```

Output:-

Id 01

Age 23

First abc

Last def

Lab Exercises 5 *Th:* - JSP

A JavaBean is a specially constructed Java class written in the Java and coded according to the JavaBeans API specifications.

Following are the unique characteristics that distinguish a JavaBean from other Java classes −

- It provides a default, no-argument constructor.
- It should be serializable and that which can implement the **Serializable** interface.
- It may have a number of properties which can be read or written.
- It may have a number of "**getter**" and "**setter**" methods for the properties.

JavaBeans Properties

A JavaBean property is a named attribute that can be accessed by the user of the object. The attribute can be of any Java data type, including the classes that you define.

A JavaBean property may be **read, write, read only**, or **write only**. JavaBean properties are accessed through two methods in the JavaBean's implementation class −

S.No.	Method & Description
1	get**PropertyName**() For example, if property name is *firstName*, your method name would be **getFirstName()** to read that property. This method is called accessor.
2	set**PropertyName**() For example, if property name is *firstName*, your method name would be **setFirstName()** to write that property. This method is called mutator.

A read-only attribute will have only a **getPropertyName()** method, and a write-only attribute will have only a **setPropertyName()** method.

Accessing JavaBeans

The **useBean** action declares a JavaBean for use in a JSP. Once declared, the bean becomes a scripting variable that can be accessed by both scripting elements and other custom tags used in the JSP. The full syntax for the useBean tag is as follows −

```
<jsp:useBean id = "bean's name" scope = "bean's scope" typeSpec/>
```

Here values for the scope attribute can be a **page, request, session** or **application based** on your requirement. The value of the **id** attribute may be any value as a long as it is a unique name among other **useBean declarations** in the same JSP.

Steps:- to display data in jsp using beans

1) Create input page index.jsp to input the data
2) Create bean which contains input data as a var & getters/setters
3) Create output.jsp page where we display the data from bean
4) Use <jsp: usebean id=”s” class=”package1.Emp”>
5) <jsp:setProperty > tag to set the data
6) <jsp:getProperty> tag to get the data

```
//index.jsp

<html>

<form action=”output.jsp”>

Empid<input type=”text” name=”eid”><br>

Ename< input type=”text” name=”ename”><br>

<input type=”submit”></form></html>
```

```
//output.jsp

<jsp:useBean id=”s” class=”package1.Emp”/>

<jsp:setproperty name=”s” property=”*”/>

Your id is<jsp:getProperty name=”s” property=”eid”/>

Your name is <jsp:getProperty name=”s” property=”ename”/>

//Emp.java  (bean)

Package package1;

Class EMP

{

Private String eid,ename;

p.v.setEid(String s)

{eid=s;}

p.v.setEname(String s)

{ename=s;}

p.String getEid()

{return id;}

p.String getEname()

{return id; }}
```

Lab Exercises *6 Th: -* JSP with MVC (2 Hours):-

The **Model-View-Controller (MVC)** is an architectural pattern that separates an application into three main logical components: the **model**, the view, and the controller. Each of these components are built to handle specific development aspects of an application. MVC is one of the most frequently used industry-standard web development framework to create scalable and extensible projects.

MVC Components

Following are the components of MVC −

Model

The Model component corresponds to all the data-related logic that the user works with. This can represent either the data that is being transferred between the View and Controller components or any other business logic-related data. For example, a Customer object will retrieve the customer information from the database, manipulate it and update it data back to the database or use it to render data.

View

The View component is used for all the UI logic of the application. For example, the Customer view will include all the UI components such as text boxes, dropdowns, etc. that the final user interacts with.

Controller

Controllers act as an interface between Model and View components to process all the business logic and incoming requests, manipulate data using the Model component and interact with the Views to render the final output. For example, the Customer controller will handle all the interactions and inputs from the Customer View and update the database using the Customer Model. The same controller will be used to view the Customer data.

//login.jsp

```
<html>
<form action="login">
username<input type="text" name="user"><br>
password<input type="text" name="pass"><br>
<input type="submit">
</form>
```

```
import javax.servlet.*;
impotr javax.servlet.http.*;
p class Login extends HttpServlet
{

 p v doGet(HttpServletRequest req,HttpServletResponse res)throws
ServletException,IOException
{
 String u=request.getParameter("user");

 String p=request.getParameter("pass");

 Emp e=new Emp();

 e.setId(u);e.setPass(p);
 boolean s=e.validate();
if(s)
{

  RequestDispatcher rd=new RequestDispatcher("success.jsp");
  rd.forward(req,res);
}

else
{

  RequestDispatcher rd=new RequestDispatcher("failure.jsp");
  rd.forward(req,res);
}


}

}
```

//success.jsp

```
<h1> login successful</h1>
```

//failure.jsp

```
<h1> login failed</h1>
```

```
package package1;
class Emp
{

private String user,pass;

p v setUser(String s)
{
user=s;
}
p v setPass(String s)
{
pass=s;
}

p boolean validate()
{

if(user.equals("jnec") && pass.equals("jnec"))
return true;
else
return false;

}

}
```

Input:- login id :- jnec password:- jnec

Output- login successful

Lab Exercises *7Th: -* To Study Struts application

The **model** contains the business logic and interact with the persistance storage to store, retrive and manipulate data.

The **view** is responsible for dispalying the results back to the user. In Struts the view layer is implemented using JSP.

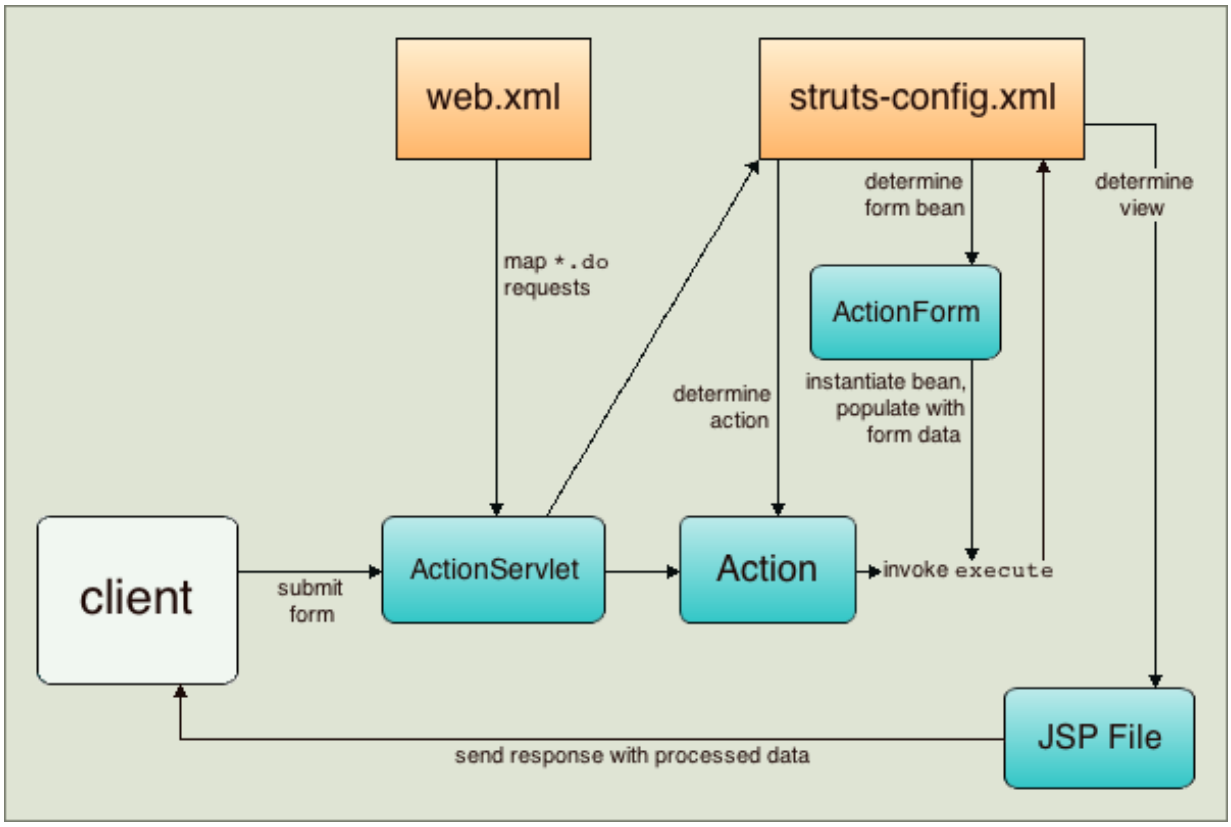
The **controller** handles all the request from the user and selects the appropriate view to return. In Sruts the controller's job is done by the *ActionServlet*.

The following events happen when the Client browser issues an HTTP request.

- The *ActionServlet* receives the request.
- The *struts-config.xml* file contains the details regarding the *Actions*, *ActionForms*, *ActionMappings*and *ActionForwards*.
- During the startup the *ActionServelet* reads the *struts-config.xml* file and creates a database of configuration objects. Later while processing the request the *ActionServlet* makes decision by refering to this object.

When the *ActionServlet* receives the request it does the following tasks.

- Bundles all the request values into a JavaBean class which extends Struts *ActionForm* class.
- Decides which action class to invoke to process the request.
- Validate the data entered by the user.
- The action class process the request with the help of the model component. The model interacts with the database and process the request.
- After completing the request processing the *Action* class returns an *ActionForward* to the controller.
- Based on the *ActionForward* the controller will invoke the appropriate view.
- The HTTP response is rendered back to the user by the view component.



Steps:

1) copy all jar files to apache tomcat\lib folder
2) create new dynamic web project
3) copy i/p and o/p pages in web content
4) copy configuration files in web\inf
5) package1 create formbean
6) package2 create action class
7) Run the application

//login.jsp

```
<html>
<form action="login">
username<input type="text" name="user"><br>
password<input type="text" name="pass"><br>
<input type="submit">
</form>
```

```
package package1;
import org.apache.strus.*;
class LoginForm extends ActionForm
```

```
{

private String user,pass;

p v setUser(String s)
{
user=s;
}
p v setPass(String s)
{
pass=s;
}
```

//success.jsp

```
<h1> login successful</h1>
```

//failure.jsp

```
<h1> login failed</h1>

P ActionForward LoginAction extends Action
{

P ActionMapping execute(ActionMapping mapping,ActionForm
for,HttpServlet req,HttpServlet res) throws Exception
{      Loginorm f=(LoginForm)form;
if(f.getUser().equals(“jnec”) && f.getPass().equals(“jnec”))
mapping.findForward(“success”);
else
mapping.findForward(“failure”);
}
}
```

Web.xml
Struts-config.xml

Input:- login id :- jnec password:- jnec

Output- login successful

(2 Hours):- 8Practical

Lab Exercises 8*Th:-* To Study struts insert operation

.

Steps:

1) copy all jar files to apache tomcat\lib folder
2) create new dynamic web project
3) copy i/p and o/p pages in web content
4) copy configuration files in web\inf
5) package1 create formbean
6) package2 create action class
7) Run the application

//login.jsp

```
<html>
<form action="login">
username<input type="text" name="user"><br>
password<input type="text" name="pass"><br>
<input type="submit">
</form>
```

```
package package1;
import org.apache.strus.*;
class LoginForm extends ActionForm
{

private String user,pass;

p v setUser(String s)
{
user=s;
}
p v setPass(String s)
{
pass=s;
}
```

//success.jsp

<h1> data inserted</h1>

//failure.jsp

<h1> error in data insertion</h1>

P ActionForward LoginAction extends Action
{

P ActionMapping execute(ActionMapping mapping,ActionForm for,HttpServlet req,HttpServlet res) throws Exception
{
Loginorm f=(LoginForm)form;
Try
{
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
Connection con=DriverManager.getConnection("jdbc:odbc:my");
Statement stmt=con.createStatement();
Int i=Stmt.executeUpdate("insert into emp values('"+f.getUser()+"','"+f.getPass()+"')");
Con.close();
if(i)
mapping.findForward("success");
else
mapping.findForward("failure");
}catch(Exception e){}

}}

Web.xml
Struts-config.xml

Output:- Data inserted successfully.

(2 Hours):- 9Practical

Lab Exercises *9Th: -* To Study Strut to display data

Steps:

1) copy all jar files to apache tomcat\lib folder
2) create new dynamic web project
3) copy i/p and o/p pages in web content
4) copy configuration files in web\inf
5) package1 create formbean
6) package2 create action class
7) Run the application

//login.jsp

```
<html>
<form action="login">
username<input type="text" name="user"><br>
password<input type="text" name="pass"><br>
<input type="submit">
</form>
```

```
package package1;
import org.apache.strus.*;
class LoginForm extends ActionForm
{

private String user,pass;

p v setUser(String s)
{
user=s;
}
p v setPass(String s)
{
pass=s;
}
```

//success.jsp

Your password is <bean:write name=”s” property=”pass”/>

//failure.jsp

<h1> data not found</h1>

P ActionForward LoginAction extends Action
{

P ActionMapping execute(ActionMapping mapping,ActionForm for,HttpServlet req,HttpServlet res) throws Exception
{
Try
{LoginForm f=(LoginForm)form;
Class.forName(“sun.jdbc.odbc.JdbcOdbcDriver”);
Connection con=DriverManager.getConnection(“jdbc:odbc:my”);
Statement stmt=con.createStatement();
Int i=Stmt.executeUpdate(“select * from emp where user”+f.getUser());
if(i)
{f.setPass(rs.getString(2));
mapping.findForward(“success”);
}
else
mapping.findForward(“failure”);
}catch(Exception e){}
}
}
}

Web.xml
Struts-config.xml

Input:- if id-005 entered which is unavailable

Output:- Data not found.

(2 Hours):- 10Practical

Lab Exercises *10Th: -* To Study Hibernate

Hibernate has a layered architecture which helps the user to operate without having to know the underlying APIs. Hibernate makes use of the database and configuration data to provide persistence services (and persistent objects) to the application.

Following is a very high level view of the Hibernate Application Architecture.

Hibernate uses various existing Java APIs, like JDBC, Java Transaction API(JTA), and Java Naming and Directory Interface (JNDI). JDBC provides a rudimentary level of abstraction of functionality common to relational databases, allowing almost any database with a JDBC driver to be supported by Hibernate. JNDI and JTA allow Hibernate to be integrated with J2EE application servers.

Following section gives brief description of each of the class objects involved in Hibernate Application Architecture.

Configuration Object

The Configuration object is the first Hibernate object you create in any Hibernate application. It is usually created only once during application initialization. It represents a configuration or properties file required by the Hibernate.

The Configuration object provides two keys components −

- **Database Connection** − This is handled through one or more configuration files supported by Hibernate. These files are **hibernate.properties** and **hibernate.cfg.xml**.
- **Class Mapping Setup** − This component creates the connection between the Java classes and database tables.

SessionFactory Object

Configuration object is used to create a SessionFactory object which in turn configures Hibernate for the application using the supplied configuration file and allows for a Session object to be instantiated. The SessionFactory is a thread safe object and used by all the threads of an application.

The SessionFactory is a heavyweight object; it is usually created during application start up and kept for later use. You would need one SessionFactory object per database using a separate configuration file. So, if you are using multiple databases, then you would have to create multiple SessionFactory objects.

Session Object

A Session is used to get a physical connection with a database. The Session object is lightweight and designed to be instantiated each time an interaction is needed with the database. Persistent objects are saved and retrieved through a Session object.

The session objects should not be kept open for a long time because they are not usually thread safe and they should be created and destroyed them as needed.

Transaction Object

A Transaction represents a unit of work with the database and most of the RDBMS supports transaction functionality. Transactions in Hibernate are handled by an underlying transaction manager and transaction (from JDBC or JTA).

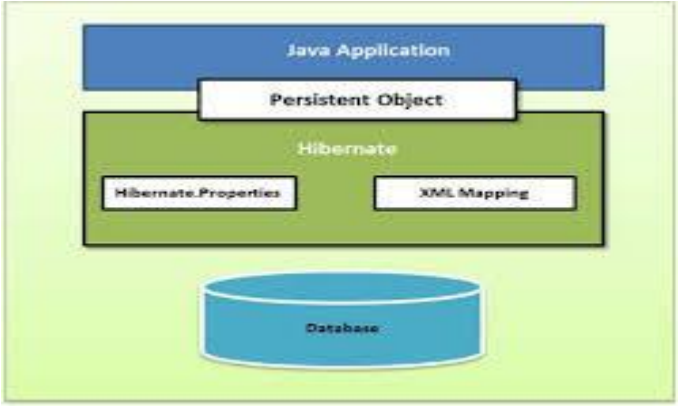
This is an optional object and Hibernate applications may choose not to use this interface, instead managing transactions in their own application code.

Query Object

Query objects use SQL or Hibernate Query Language (HQL) string to retrieve data from the database and create objects. A Query instance is used to bind query parameters, limit the number of results returned by the query, and finally to execute the query.

Criteria Object

Criteria objects are used to create and execute object oriented criteria queries to retrieve objects.



Steps:-

1)create hibernate.cfg.xaml file

2)create another Emp.hbm.xml file for class and database table mapping

3)create class for bean in package1

4)crate another class in package1 to write the logic.

//hibernate.cfg.xml

```
<?xml version='1.0' encoding='UTF-8'?>

<!DOCTYPE hibernate-configuration PUBLIC

        "-//Hibernate/Hibernate Configuration DTD 3.0//EN"

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



<hibernate-configuration>

<session-factory>

<property
name="hibernate.connection.driver_class">org.postgresql.Driver</property>

<property
name="hibernate.connection.url">jdbc:postgresql://localhost:5432/my</property
>

<property name="hibernate.connection.username">postgres</property>

<property name="hibernate.connection.password">admin</property>



<property name="dialect">org.hibernate.dialect.PostgreSQLDialect</property>
```

<property name="show_sql">true</property>

<mapping resource="Emp.hbm.xml" />

</session-factory>

</hibernate-configuration>

// Emp.hbm.xml

<?xml version="1.0"?>

<!DOCTYPE hibernate-mapping PUBLIC

"-//Hibernate/Hibernate Mapping DTD 3.0//EN"

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

<hibernate-mapping>

<class name="str.Emp" table="emp">

<id name="empid" column="empid" >

<generator class="assigned" />

</id>

<property name="ename" column="ename" length="10"/>

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

Class:- Emp

```
package str;
public class Emp{

    private int empid;
    private String ename;
    public int getEmpid() {
        return empid;
    }
    public void setEmpid(int empid) {
        this.empid = empid;
    }
    public String getEname() {
        return ename;
    }
    public void setEname(String ename) {
        this.ename = ename;
    }
```

```
}
```

//InsertSelect.java

```
package str;


import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.Transaction;

import org.hibernate.cfg.Configuration;


public class InsertSelect {

      public static void main(String[] args)throws Exception

        {


            Configuration cfg = new Configuration();

            cfg.configure("hibernate.cfg.xml");

             //insert a record

            SessionFactory factory = cfg.buildSessionFactory();

            Session session = factory.openSession();

            Emp p=new Emp();
```

```
        p.setEmpid(99);
        p.setEname("ABC");



        Transaction tx = session.beginTransaction();
        session.save(p);
        System.out.println("Object saved successfully.....!!");
        tx.commit();
        //select a record
        Object o=session.load(Emp.class,new Integer(99));
           Emp s=(Emp)o;
           // For loading Transaction scope is not necessary...
           System.out.println("Loaded object EMP name is___"+s.getEname());
           System.out.println("Loaded object EMP name is___"+s.getEmpid());


        session.close();
        factory.close();
    }
}
```

Output:- Data inserted successfully.

Quiz on the subject:

Quiz should be conducted on tips in the laboratory, recent trends and subject knowledge of the subject. The quiz questions should be formulated such that questions are normally are from the scope outside of the books. However twisted questions and self formulated questions by the faculty can be asked but correctness of it is necessarily to be thoroughly checked before the conduction of the quiz.

Conduction of Viva-Voce Examinations:

Teacher should oral exams of the students with full preparation. Normally, the objective questions with guess are to be avoided. To make it meaningful, the questions should be such that depth of the students in the subject is tested Oral examinations are to be conducted in co-cordial environment amongst the teachers taking the examination. Teachers taking such examinations should not have ill thoughts about each other and courtesies should be offered to each other in case of difference of opinion, which should be critically suppressed in front of the students.

Submission: Each student has to submit journal for this subject by completing all the assignments.

Evaluation and marking system:

Basic honesty in the evaluation and marking system is absolutely essential and in the process impartial nature of the evaluator is required in the examination system to become popular amongst the students. It is a wrong approach or concept to award the students by way of easy marking to get cheap popularity among the students to which they do not deserve. It is a primary responsibility of the teacher that right students who are really putting up lot of hard work with right kind of intelligence are correctly awarded.

The marking patterns should be justifiable to the students without any ambiguity and teacher should see that students are faced with unjust circumstances.