MGM's

Jawaharlal Nehru Engineering College Aurangabad

MGM University, Aurangabad

Department of Computer Science & Engineering

LAB MANUAL

Program (UG/PG) :	UG
Year :	Third Year
Semester :	V
Course Code :	20UCS511L
Course Title :	Server Side Programming Lab
Prepared By :	Mr. S. N. Jaiswal
	Ms. C. G. Patil

Department of Computer Science & Engineering

2022-23

FOREWORD

It is our great pleasure to present this laboratory manual for Third Year engineering students for the subject of Server Side Programming Lab.
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 9001:2015,140001:2015 certification and it is our endure to technically equip our students taking the advantage of the procedural aspects of ISO Certification.

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

Dr. H. H. Shinde
Principal

LABORATORY MANUAL CONTENTS

This manual is intended for the Third year students of Computer Science and Engineering in the subject of Server Side Programming Lab. This manual typically contains practical/Lab Sessions related Java Programming covering various aspects relatedto the subject to enhanced 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.

Good Luck for your Enjoyable Laboratory Sessions

Prof. Vijaya B. Musande(HOD)

Subject Teacher

Mr. S. N. Jaiswal

Ms.C.G.Patil

MGM's

Jawaharlal Nehru Engineering College, Aurangabad

Department of Computer Science and Engineering

<u>Vision of CSE Department:</u>

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.

<u>Mission of the CSE Department:</u>

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

II. Preparing graduates for higher education and research in Computer Science and Engineering enabling them to develop systems for society development.

<u>Programme Educational Objectives:</u> Graduates will be able to

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

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

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

Programme Outcomes (POs): Engineering Graduates will be able to:

1. **Engineering knowledge**: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

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

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

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

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

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

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

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

10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

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

<u>LIST OF EXPERIMENTS</u>

Course Code : 20UCS511L

Course Title : Server Side Programming Lab

Sr. No.	Name of theExperiment
1	Write a Java program to perform following database operations (MySQL/Oracle DB) 1. DDL commands 2. DML commands
2	Design and develop standalone application / Desktop application to perform CRUD operations on MySQL/ Oracle Database
3	Design and develop a simple application program using RMI (any two) 1. Simple Mathematical Calculator 2. Displaying bank customer details from database 3. File transfer utility 4. Message transfer utility 5. Sorting Methods 6. Database operations
4	Write a program to create a simple servlet for 1. Demonstration of Servlet Life Cycle 2. Form processing (Student Information) 3. Printing request header information
5	Write a Servlet program to demonstrate different Session Management techniques
6	Design a simple application program using Servlet and Database 1. Simple login form 2. Customer Feedback Form 3. Admission Form 4. Student Mark Sheet
7	Design and develop JSP application to demonstrate 1. JSP Scripting elements 2. JSP Directives 3. JSP Implicit Objects 4. JSP Action tags
8	Write a JSP program for 1. Tag 2. Exception handling in JSP
9	Design and develop JSP ICT, practical application using JSP Custom oriented tags and JSTL tags
10	Design and implement SOAP based Web Service for 1. Mathematical Calculator 2. Currency Conversion 3. Temperature Conversion

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.
7. Do not disturb machine Hardware / Software Setup.

Instruction for Laboratory Teachers:

1. Submission related to whatever lab work has been completed should be doneduring the next lab session along with signing the index.
2. The promptness of submission should be encouraged by way of marking and evaluation patterns that will benefit the sincere students.
3. Continuous assessment in the prescribed format must be followed.

LABORATORY OUTCOMES

The practical/exercises in this section are psychomotor domain Learning Outcomes (i.e. subcomponents of the COs), to be developed and assessed to lead to the attainment of the competency.

LO1: Design and develop Java application to demonstrate the JDBC and Remote Procedure Call (RPC) in Java.(Apply/Create)

LO2: Demonstrate the concept of Servlet/JSP life cycle, Session tracking techniques, JSP Elements, Custom Tags and Exception handling in JSP (Demonstration)

LO3: Design and develop Dynamic Web/ Enterprise Application using Servlets, JSP, JSTL, EJB and MySQL Database to cater need of the Enterprise (Create)

LO4: Design and Develop SOAP/ RESTful Web Service to solve simple business problems (Create)

LO5: Design and Develop web application to use JavaMail API (Apply/Demonstrate)

LO6: Design and Develop MVC based Web/Enterprise Application using Java EE technologies and frameworks such as Spring, Hibernate (Apply/Create)

Practical : 1

Aim : Write a Java program to perform following database operations (MySQL/Oracle DB)

1. DDL commands

2. DML commands

Theory :

SQL is a database language designed for the retrieval and management of data in a relational database. SQL is the standard language for database management. All the RDBMS systems like MySQL, MS Access, Oracle, Sybase, Postgres, and SQL Server use SQL as their standard database language. SQL programming language uses various commands for different operations. We will learn about the like DCL, TCL, DQL, DDL and DML commands in SQL with examples.

Why Use SQL?

Here, are important reasons for using SQL

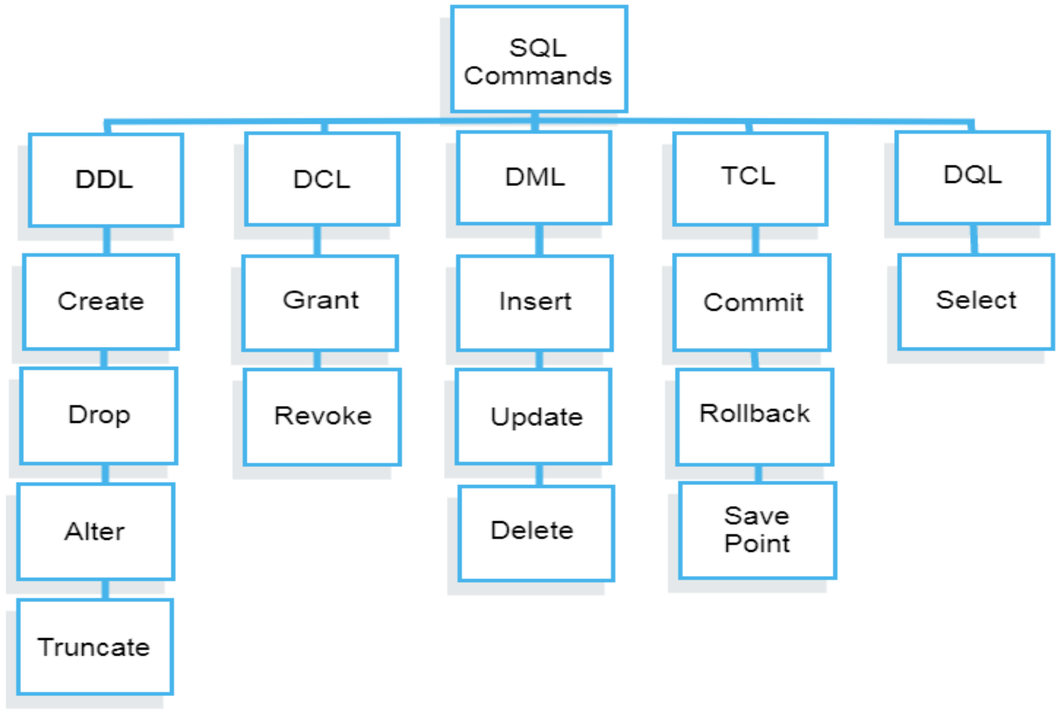
- It helps users to access data in the Database system.
- It helps you to describe the data.
- It allows you to define the data in a database and manipulate that specific data.
- With the help of SQL commands in java, you can create and drop databases and tables.
- SQL offers you to use the function in a database, create a view, and stored procedure.
- You can set permissions on tables, procedures, and views.

Types of SQL

Here are five types of widely used SQL queries.

- Data Definition Language (DDL)
- Data Manipulation Language (DML)
- Data Control Language(DCL)
- Transaction Control Language(TCL)

- Data Query Language (DQL)



What is DDL?

Data Definition Language helps you to define the database structure or schema. Let's learn about DDL commands with syntax.

Five types of DDL commands in SQL are:

CREATE

CREATE statements is used to define the database structure schema:

Syntax:

CREATE TABLE TABLE_NAME (COLUMN_NAME DATATYPES[,....]);

For example:

Create database university;

Create table students;

Create view for_students;

DROP

Drops commands remove tables and databases from RDBMS.

Syntax

DROP TABLE ;

For example:

Drop object_type object_name;

Drop database university;

Drop table student;

ALTER

Alters command allows you to alter the structure of the database.

Syntax:

To add a new column in the table

ALTER TABLE table_name ADD column_name COLUMN-definition;

To modify an existing column in the table:

ALTER TABLE MODIFY(COLUMN DEFINITION....);

For example:

Alter table guru99 add subject varchar;

TRUNCATE:

This command used to delete all the rows from the table and free the space containing the table.

Syntax:

TRUNCATE TABLE table_name;

Example:

TRUNCATE table students;

What is Data Manipulation Language?

Data Manipulation Language (DML) allows you to modify the database instance by inserting, modifying, and deleting its data. It is responsible for performing all types of data modification in a database.

There are three basic constructs which allow database program and user to enter data and information are:

Here are some important DML commands in SQL:

- INSERT
- UPDATE

- DELETE

INSERT:

This is a statement is a SQL query. This command is used to insert data into the row of a table.

Syntax:

INSERT INTO TABLE_NAME (col1, col2, col3,.... col N)

VALUES (value1, value2, value3, ... valueN);

Or

INSERT INTO TABLE_NAME

VALUES (value1, value2, value3, ... valueN);

For example:

INSERT INTO students (RollNo, FIrstName, LastName) VALUES ('60', 'Tom', Erichsen');

UPDATE:

This command is used to update or modify the value of a column in the table.

Syntax:

UPDATE table_name SET [column_name1= value1,...column_nameN = valueN] [WHERE CONDITION]

For example:

UPDATE students

SET FirstName = 'Jhon', LastName= 'Wick'

WHERE StudID = 3;

DELETE:

This command is used to remove one or more rows from a table.

Syntax:

DELETE FROM table_name [WHERE condition];

For example:

DELETE FROM students

WHERE FirstName = 'Jhon';

Practical : 2

Aim : Design and develop standalone application / Desktop application to perform CRUD operations on MySQL/ Oracle Database.

Theory :

Inserting Data in Database table using Statement

In this program we are going to insert the data in the database from our java program in the table stored in the database.

To accomplish our goal we first have to make a class named as **ServletInsertingData,** which must extends the abstract **HttpServlet** class, the name of the class should be such that other person can understand what this program is going to perform. The logic of the program will be written inside the **doGet()** method that takes two arguments, first is **HttpServletRequest** interface and the second one is the **HttpServletResponse** interface and this method can throw **ServletException.**

Inside this method call the **getWriter()** method of the **PrintWriter** class. We can insert the data in the database only and only if there is a connectivity between our database and the java program. To establish the connection between our database and the java program we first need to call the method **forName(),** which is static in nature of the class Class. It takes one argument which tells about the database driver we are going to use. Now use the static method **getConnection()** of the **DriverManager** class. This method takes three arguments and returns the Connection object. SQL statements are executed and results are returned within the context of a connection. Now your connection has been established. Now use the method **createStatement()** of the Connection object which will return the Statement object. This object is used for executing a static SQL statement and obtaining the results produced by it. We have to insert a values into the table so we need to write a query for inserting the values into the table. This query we will write inside the **executeUpdate()** method of the Statement object. This method returns int value.

If the record will get inserted in the table then output will show "record has been inserted" otherwise "sorry! Failure".

Program

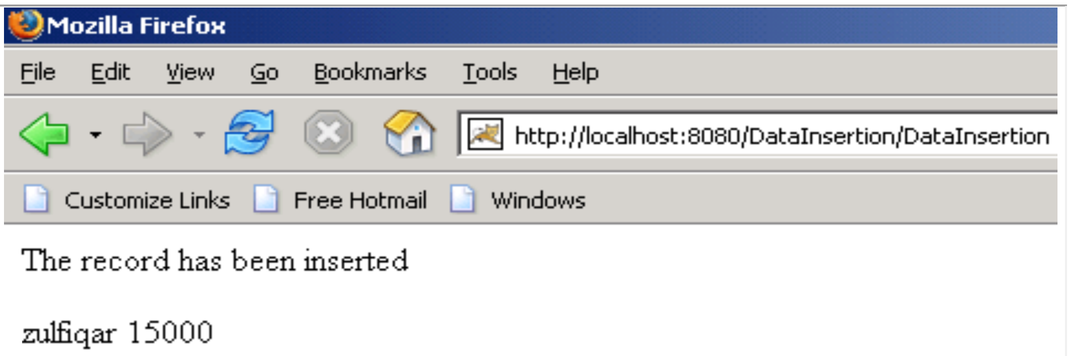
```
import java.io.*;
import java.sql.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class DataInsertion extends HttpServlet
{
    public void doGet(HttpServletRequest request, HttpServletResponse response)
                throws ServletException, IOException
 {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String url =            "jdbc:mysql://localhost/zulfiqar?user=root&password=admin";
    Connection conn;
    ResultSet rs;
    try
      {
         Class.forName("org.gjt.mm.mysql.Driver");
         conn = DriverManager.getConnection(url);
         Statement statement = conn.createStatement();
         String query = "insert into emp_sal values('Rajesh', 15000)";
         int i = statement.executeUpdate(query);

      if(i!=0)
      {
         out.println("The record has been inserted");
    }
    else
      {
```

```
                        out.println("Sorry! Failure");
                }
                rs = statement.executeQuery("select * from emp_sal");

                   while(rs.next())
{
        out.println("<p><table>" + rs.getString(1) + " "    + rs.getInt(2) + "</p></table>");
}
rs.close();
statement.close();
                   }
                catch (Exception e)
                   {
                        System.out.println(e);
                }
          }
        }
```

Table in the database before Insertion:

```
mysql> select * from
emp_sal;
Empty set (0.02 sec)
```

The output of the program is given below:


Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:8080/DataInsertion/DataInsertion
Customize Links Free Hotmail Windows

```
The record has been inserted

Rajesh zulfiqar 15000
```

Table in the database after Insertion:

```
mysql> select * from emp_sal;
+----------+--------+
| EmpName | salary |
+----------+--------+
| Rajesh     | 15000 |
+----------+--------+
1 row in set (0.02 sec)
```

<u>Conclusion:</u> Hence we have designed web application using My Sql.

Practical : 3

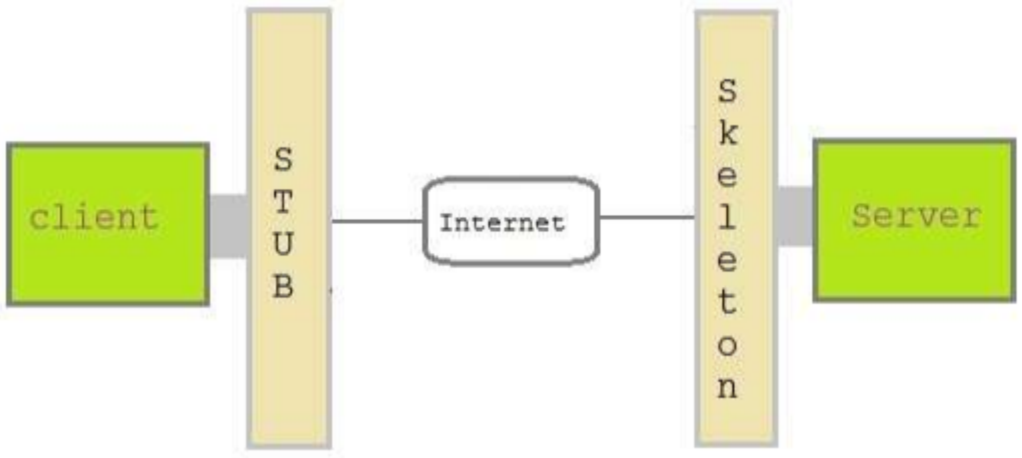
Aim : Design and develop a simple application program using RMI (any two) 1. Simple Mathematical Calculator 2. Displaying bank customer details from database 3. File transfer utility 4. Message transfer utility 5. Sorting Methods 6. Database operations

Theory/Description:

Introduction:

The Java Remote Method Invocation (Java RMI) is a Java API that performs the object-oriented equivalent of remote procedure calls (RPC), with support for direct transfer of serialized Java
objects and distributed garbage collection. The original implementation depends on Java Virtual Machine (JVM) class representation mechanisms and it thus only supports making calls from one JVM to another.
The protocol underlying this Java-only implementation is known as Java Remote Method Protocol (JRMP). In order to support code running in a non-JVM context, a CORBA version was later developed. Usage of the term RMI may denote solely the programming interface or may signify both the API and JRMP, whereas the term RMI-IIOP (read: RMI over IIOP) denotes the RMI interface delegating most of the functionality to the supporting CORBA implementation.

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 Skeletonobject. 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.

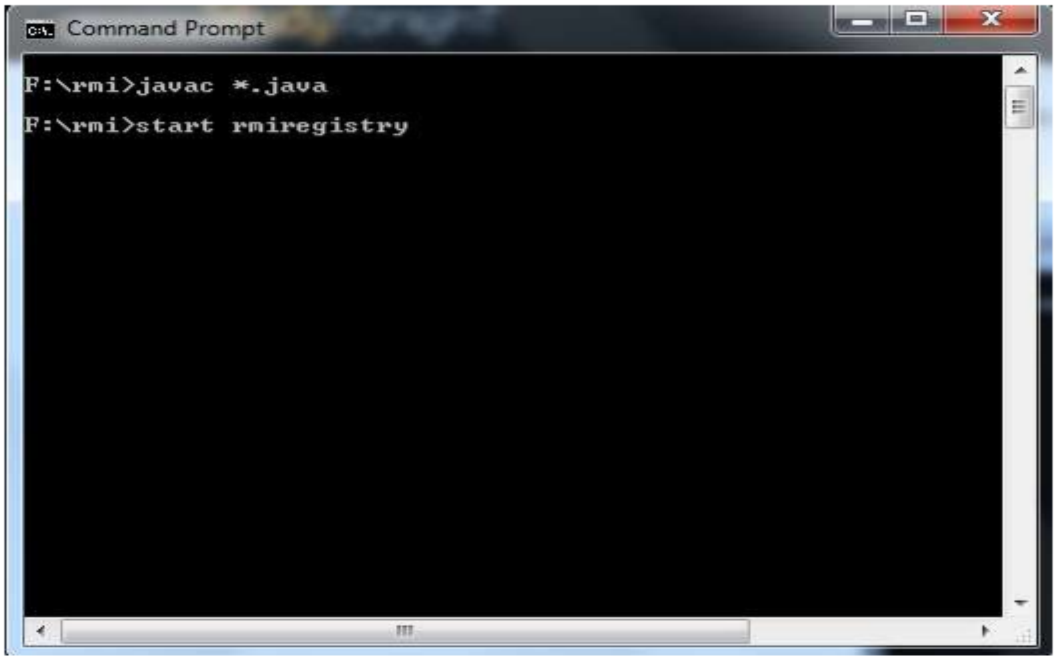
```
import java.rmi.*;
public class Client{ public static void main(String args[])
{
try
{
AddServerInterface        st=(AddServerInterface)Naming.lookup("rmi://"+args[0]+"/AddService");
System.out.println(st.sum(25,8));
 }
catch(Exception e)
{
System.out.println(e);
}
 }
 }
```

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

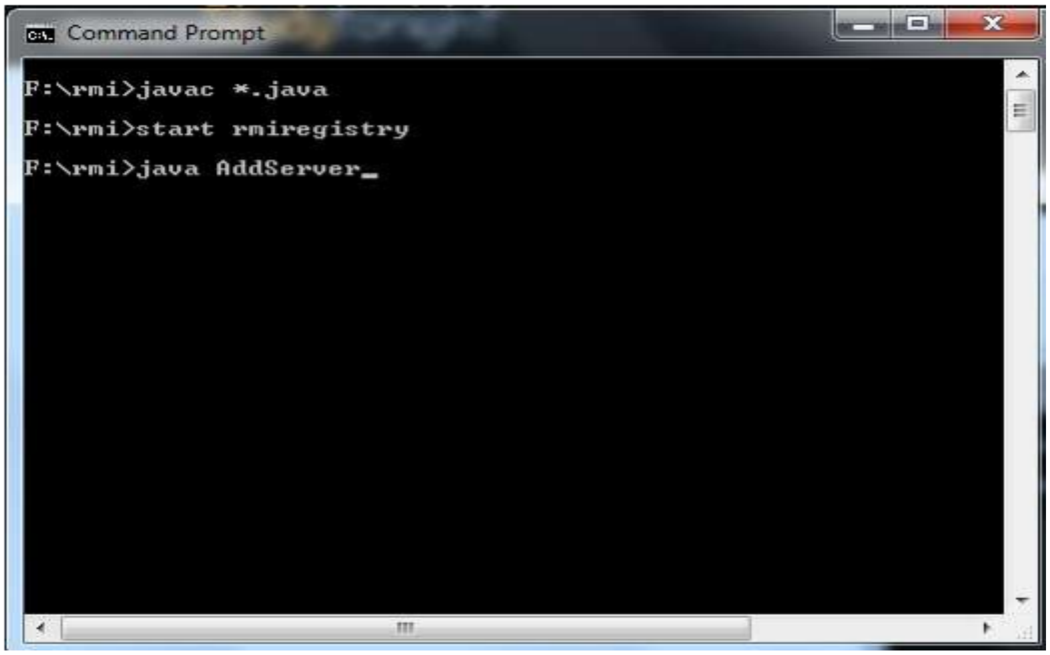
start rmiregistry



```
F:\rmi>javac *.java
F:\rmi>start rmiregistry
```

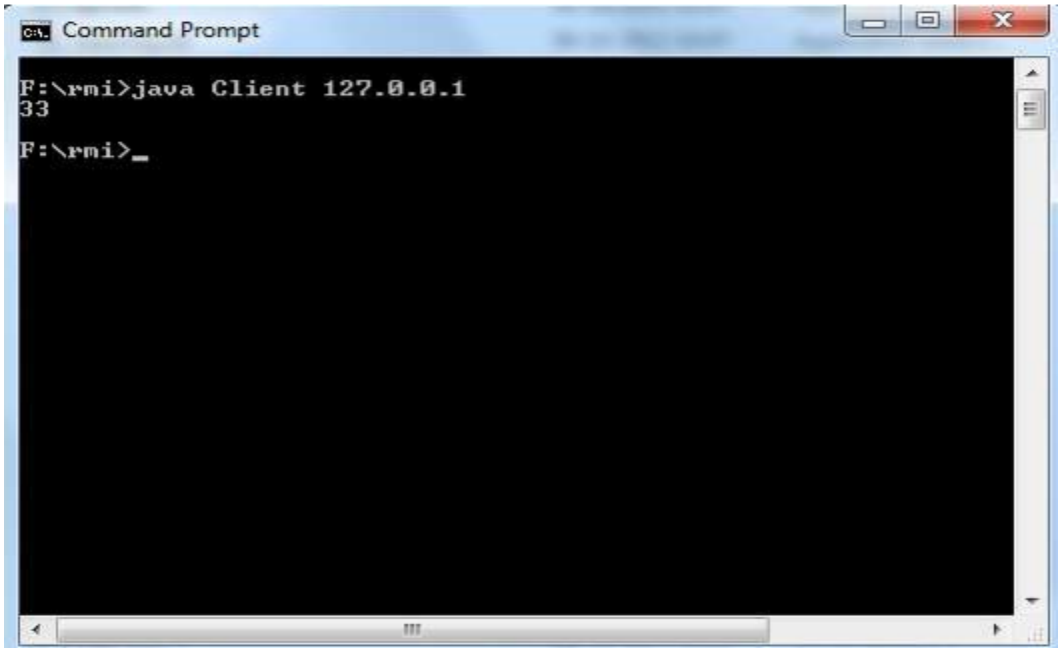
Run Server file

java AddServer



```
F:\rmi>javac *.java
F:\rmi>start rmiregistry
F:\rmi>java AddServer_
```

Run Client file in another command prompt abd pass local host port number at run time java Client 127.0.0.1



```
F:\rmi>java Client 127.0.0.1
33

F:\rmi>_
```

Conclusion: Hence we implemented Remote method invocation.

Practicale : 4

Aim : Write a program to create a simple servlet for 1. Demonstration of Servlet Life Cycle 2. Form processing (Student Information) 3. Printing request header information

Theory :

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

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

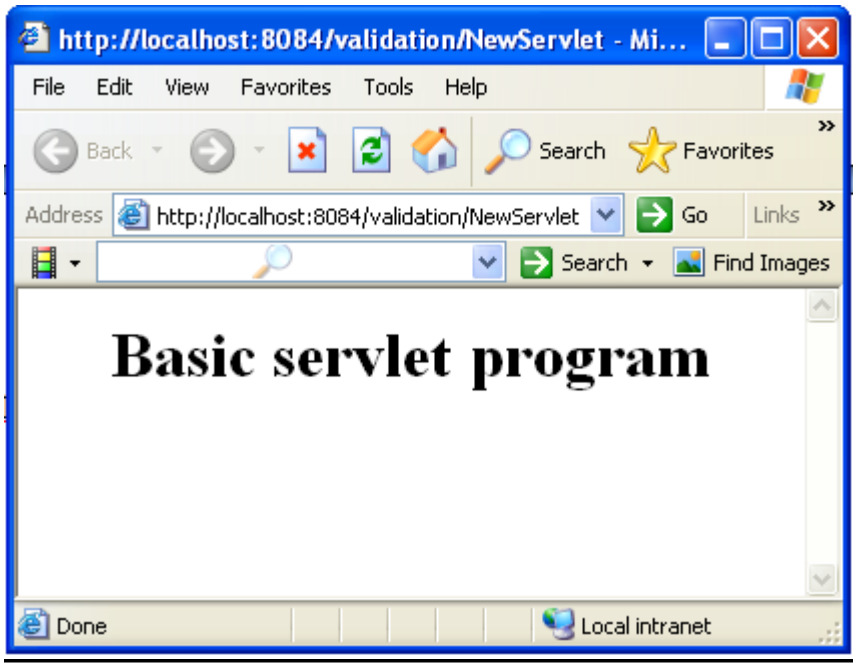
Inside the doGet() method our servlet has first used the setContentType() method of the response object which sets the content type of the response to text/html It is the standard MIME content type for the html pages.After that it has used the method getWriter () of the response object to retrieve a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

<u>Program</u>

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

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

          out.println("<html>");
          out.println("<body>");
          out.println("<h1 align=center>Basic servlet program</h1>");
                  out.println("</body>");
                  out.println("</html>");
          }
}
```

Output:


http://localhost:8084/validation/NewServlet - Mi...
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:8084/validation/NewServlet Go Links
Search Find Images
Done Local intranet

Basic servlet program

Journal Write-up:

- HTTP protocol & its methods
- Introduction to servlet

- Servlet configuration
- Life cycle of servlet
- Requests and Responses

Additional List of Programs :

1. WAP to display the username using servlet
2. WAP to design the login form using servlet.
3. Write a servlet program for customer registration.
4. WAP to design Tech Support Application which involves
   - Collecting a technical support request
   - Storing the support request in a database
   - Generating a confirmation page

Conclusion:

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

Practical : 5

Aim: Write a Servlet program to demonstrate different Session Management techniques

Theory :

1. **What is a Session?**

   HTTP protocol and Web Servers are stateless, what it means is that for web server every request is a new request to process and they can't identify if it's coming from client that has been sending request previously. But sometimes in web applications, we should know who the client is and process the request accordingly. For example, a shopping cart application should know who is sending the request to add an item and in which cart the item has to be added or who is sending checkout request so that it can charge the amount to correct client. **Session** is a conversional state between client and server and it can consists of multiple request and response between client and server. Since HTTP and Web Server both are stateless, the only way to maintain a session is when some unique information about the session (session id) is passed between server and client in every request and response. There are several ways through which we can provide unique identifier in request and response.

1. **User Authentication** - This is the very common way where we user can provide authentication credentials from the login page and then we can pass the authentication information between server and client to maintain the session. This is not very effective method because it wont work if the same user is logged in from different browsers.
2. **HTML Hidden Field** - We can create a unique hidden field in the HTML and when user starts navigating, we can set its value unique to the user and keep track of the session. This method can't be used with links because it needs the form to be submitted every time request is made from client to server with the hidden field. Also it's not secure because we can get the hidden field value from the HTML source and use it to hack the session.
3. **URL Rewriting** - We can append a session identifier parameter with every request and response to keep track of the session. This is very tedious because we need to keep track of this parameter in every response and make sure it's not clashing with other parameters.

4. **Cookies** - Cookies are small piece of information that is sent by web server in response header and gets stored in the browser cookies. When client make further request, it adds the cookie to the request header and we can utilize it to keep track of the session. We can maintain a session with cookies but if the client disables the cookies, then it won't work.
5. **Session Management API** - Session Management API is built on top of above methods for session tracking. Some of the major disadvantages of all the above methods are:

- Most of the time we don't want to only track the session, we have to store some data into the session that we can use in future requests. This will require a lot of effort if we try to implement this.
- All the above methods are not complete in themselves, all of them won't work in a particular scenario. So we need a solution that can utilize these methods of session tracking to provide session management in all cases.

That's why we need **Session Management API** and J2EE Servlet technology comes with session management API that we can use.

2. **Session Management in Java - Cookies**

Cookies are used a lot in web applications to personalize response based on your choice or to keep track of session. Before moving forward to the Servlet Session Management API, I would like to show how can we keep track of session with cookies through a small web application. We will create a dynamic web application **ServletCookieExample** with project structure

Welcome page of our application is login.html where we will get authentication details from user.

```
<!DOCTYPE html>
<html>
<head>
<meta charset="US-ASCII">
<title>Login Page</title>
</head>
<body>

<form action="LoginServlet" method="post">
```

```
Username: <input type="text" name="user">
<br>
Password: <input type="password" name="pwd">
<br>
<input type="submit" value="Login">
</form>
</body>
</html>
```

Here is the LoginServlet that takes care of the login request.

```
package com.journaldev.servlet.session;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.RequestDispatcher;
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 LoginServlet
 */
@WebServlet("/LoginServlet")
public class LoginServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    private final String userID = "Pankaj";
    private final String password = "journaldev";
	protected void doPost(HttpServletRequest request,
			HttpServletResponse response) throws ServletException, IOException {

		// get request parameters for userID and password
		String user = request.getParameter("user");
		String pwd = request.getParameter("pwd");

		if(userID.equals(user) && password.equals(pwd)){
			Cookie loginCookie = new Cookie("user",user);
			//setting cookie to expiry in 30 mins
			loginCookie.setMaxAge(30*60);
			response.addCookie(loginCookie);
			response.sendRedirect("LoginSuccess.jsp");
		}else{
			RequestDispatcher rd = getServletContext().getRequestDispatcher("/login.html");
			PrintWriter out= response.getWriter();
			out.println("<font color=red>Either user name or password is wrong.</font>");
			rd.include(request, response);
		}

	}

}
```

cookie that we are setting to the response and then forwarding it to LoginSuccess.jsp, this cookie will be used there to track the session. Also notice that cookie timeout is set to 30 minutes. Ideally there should be a complex logic to set the cookie value for session tracking so that it won't collide with any other request.

```
<%@ page language="java" contentType="text/html; charset=US-ASCII"
   pageEncoding="US-ASCII"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
 "https://www.w3.org/TR/html4/loose.dtd">
```

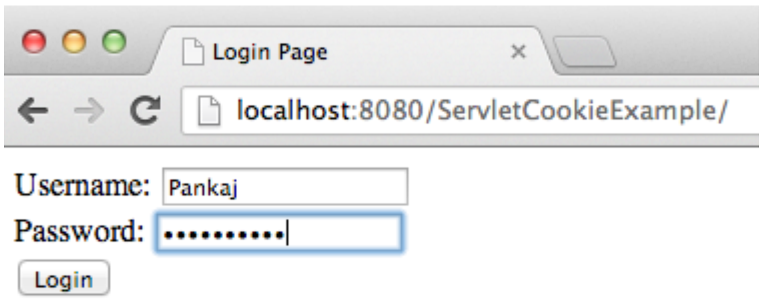
```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=US-ASCII">
<title>Login Success Page</title>
</head>
<body>
<%
String userName = null;
Cookie[] cookies = request.getCookies();
if(cookies !=null){
for(Cookie cookie : cookies){
    if(cookie.getName().equals("user")) userName = cookie.getValue();
}
}
if(userName == null) response.sendRedirect("login.html");
%>
<h3>Hi <%=userName %>, Login successful.</h3>
<br>
<form action="LogoutServlet" method="post">
<input type="submit" value="Logout" >
</form>
</body>
</html>
```

if we try to access the JSP directly, it will forward us to the login page. When we will click on Logout button, we should make sure that cookie is removed from client browser.

```
package com.journaldev.servlet.session;

import java.io.IOException;

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;
import javax.servlet.http.HttpSession;

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

  protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException {
    response.setContentType("text/html");
    Cookie loginCookie = null;
    Cookie[] cookies = request.getCookies();
    if(cookies != null){
    for(Cookie cookie : cookies){
        if(cookie.getName().equals("user")){
                loginCookie = cookie;
                break;
        }
    }
    }
    if(loginCookie != null){
        loginCookie.setMaxAge(0);
        response.addCookie(loginCookie);
    }
    response.sendRedirect("login.html");
  }
```

}

There is no method to remove the cookie but we can set the maximum age to 0 so that it will be deleted from client browser immediately. When we run above application, we get response like below images.



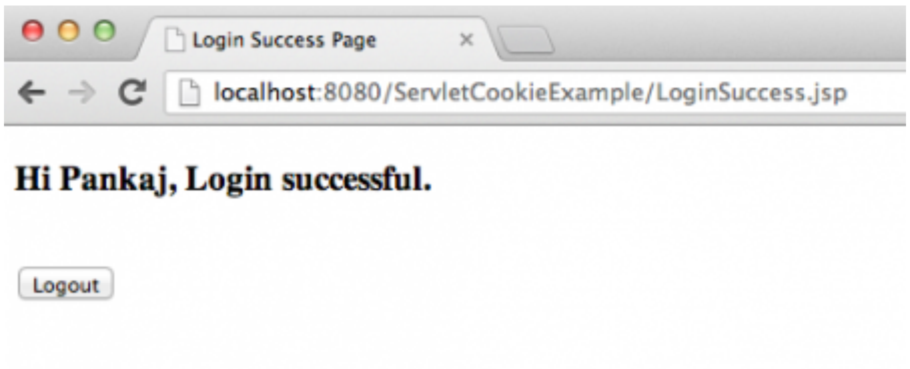
Login Page

localhost:8080/ServletCookieExample/

Username: Pankaj

Password: ••••••••••

Login



Login Success Page

localhost:8080/ServletCookieExample/LoginSuccess.jsp

Hi Pankaj, Login successful.

Logout

1. **Session in Java Servlet - HttpSession**

Servlet API provides Session management through HttpSession interface. We can get session from HttpServletRequest object using following methods. HttpSession allows us to set objects as attributes that can be retrieved in future requests.

1. **HttpSession getSession()** - This method always returns a HttpSession object. It returns the session object attached with the request, if the request has no session attached, then it creates a new session and return it.
2. **HttpSession getSession(boolean flag)** - This method returns HttpSession object if request has session else it returns null.

Some of the important methods of HttpSession are:

3. **String getId()** - Returns a string containing the unique identifier assigned to this session.
4. **Object getAttribute(String name)** - Returns the object bound with the specified name in this session, or null if no object is bound under the name. Some other methods to work with Session attributes are getAttributeNames(), removeAttribute(String

name) and setAttribute(String name, Object value).

5. **long getCreationTime()** - Returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT. We can get last accessed time with getLastAccessedTime() method.
6. setMaxInactiveInterval(int interval) - Specifies the time, in seconds, between client requests before the servlet container will invalidate this session. We can get session timeout value from getMaxInactiveInterval() method.
7. **ServletContext getServletContext()** - Returns ServletContext object for the application.
8. **boolean isNew()** - Returns true if the client does not yet know about the session or if the client chooses not to join the session.
9. **void invalidate()** - Invalidates this session then unbinds any objects bound to it.

Understanding JSESSIONID Cookie

When we use HttpServletRequest getSession() method and it creates a new request, it creates the new HttpSession object and also add a Cookie to the response object with name JSESSIONID and value as session id. This cookie is used to identify the HttpSession object in further requests from client. If the cookies are disabled at client side and we are using URL rewriting then this method uses the jsessionid value from the request URL to find the corresponding session. JSESSIONID cookie is used for session tracking, so we should not use it for our application purposes to avoid any session related issues. Let's see example of session management using HttpSession object. We will create a dynamic web project in Eclipse with servlet context as ServletHttpSessionExample. The project structure will look like below

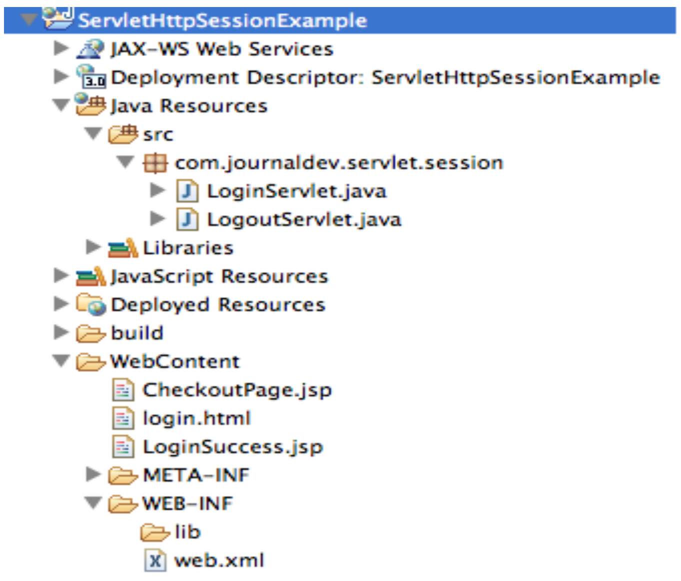

```
ServletHttpSessionExample
  JAX-WS Web Services
  Deployment Descriptor: ServletHttpSessionExample
  Java Resources
    src
      com.journaldev.servlet.session
        LoginServlet.java
        LogoutServlet.java
    Libraries
  JavaScript Resources
  Deployed Resources
  build
  WebContent
    CheckoutPage.jsp
    login.html
    LoginSuccess.jsp
    META-INF
    WEB-INF
      lib
      web.xml
```

image. login.html is same like earlier example and defined as welcome page for the application in web.xml LoginServlet servlet will create the session and set attributes that we can use in other resources or in future requests.

```
package com.journaldev.servlet.session;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.RequestDispatcher;
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;
import javax.servlet.http.HttpSession;

/**
 * Servlet implementation class LoginServlet
 */
@WebServlet("/LoginServlet")
public class LoginServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    private final String userID = "admin";
    private final String password = "password";

    protected void doPost(HttpServletRequest request,
                HttpServletResponse response) throws ServletException, IOException {

        // get request parameters for userID and password
        String user = request.getParameter("user");
        String pwd = request.getParameter("pwd");

        if(userID.equals(user) && password.equals(pwd)){
                HttpSession session = request.getSession();
                session.setAttribute("user", "Pankaj");
                //setting session to expiry in 30 mins
                session.setMaxInactiveInterval(30*60);
                Cookie userName = new Cookie("user", user);
                userName.setMaxAge(30*60);
                response.addCookie(userName);
                response.sendRedirect("LoginSuccess.jsp");
        }else{
                RequestDispatcher rd = getServletContext().getRequestDispatcher("/login.html");
                PrintWriter out= response.getWriter();
                out.println("<font color=red>Either user name or password is wrong.</font>");
                rd.include(request, response);
        }

    }
}
```

Our LoginSuccess.jsp code is given below.

```
<%@ page language="java" contentType="text/html; charset=US-ASCII"
   pageEncoding="US-ASCII"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
     "https://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=US-ASCII">
<title>Login Success Page</title>
</head>
<body>
<%
//allow access only if session exists
String user = null;
if(session.getAttribute("user") == null){
     response.sendRedirect("login.html");
}else user = (String) session.getAttribute("user");
String userName = null;
String sessionID = null;
Cookie[] cookies = request.getCookies();
if(cookies !=null){
for(Cookie cookie : cookies){
     if(cookie.getName().equals("user")) userName = cookie.getValue();
     if(cookie.getName().equals("JSESSIONID")) sessionID = cookie.getValue();
}
}
%>
<h3>Hi <%=userName %>, Login successful. Your Session ID=<%=sessionID %></h3>
<br>
User=<%=user %>
<br>
```

```
<a href="CheckoutPage.jsp">Checkout Page</a>
<form action="LogoutServlet" method="post">
<input type="submit" value="Logout" >
</form>
</body>
</html>
```

When a JSP resource is used, container automatically creates a session for it, so we can’t check if session is null to make sure if user has come through login page, so we are using session attribute to validate request. CheckoutPage.jsp is another page and it’s code is given below.

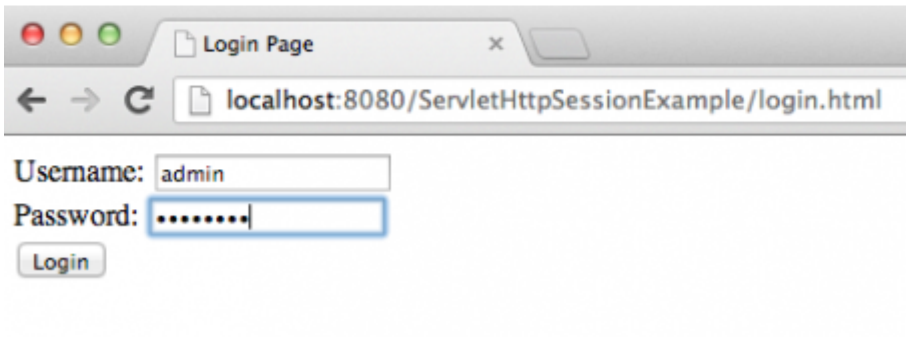
```
<%@ page language="java" contentType="text/html; charset=US-ASCII"
   pageEncoding="US-ASCII"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "https://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=US-ASCII">
<title>Login Success Page</title>
</head>
<body>
<%
//allow access only if session exists
if(session.getAttribute("user") == null){
    response.sendRedirect("login.html");
}
String userName = null;
String sessionID = null;
Cookie[] cookies = request.getCookies();
if(cookies !=null){
for(Cookie cookie : cookies){
    if(cookie.getName().equals("user")) userName = cookie.getValue();
}
}
```

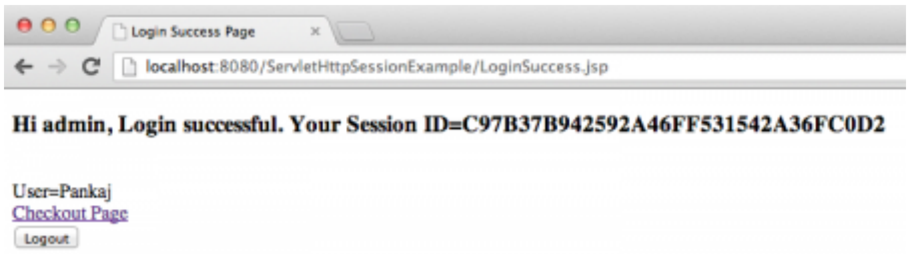
```
%>
<h3>Hi <%=userName %>, do the checkout.</h3>
<br>
<form action="LogoutServlet" method="post">
<input type="submit" value="Logout" >
</form>
</body>
</html>
```

Our LogoutServlet code is given below.

```
package com.journaldev.servlet.session;

import java.io.IOException;

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;
import javax.servlet.http.HttpSession;

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

   protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException {
    response.setContentType("text/html");
    Cookie[] cookies = request.getCookies();
    if(cookies != null){
    for(Cookie cookie : cookies){
        if(cookie.getName().equals("JSESSIONID")){
                System.out.println("JSESSIONID="+cookie.getValue());
                break;
        }
    }
    }
    //invalidate the session if exists
    HttpSession session = request.getSession(false);
    System.out.println("User="+session.getAttribute("user"));
    if(session != null){
        session.invalidate();
    }
    response.sendRedirect("login.html");
   }


}
```

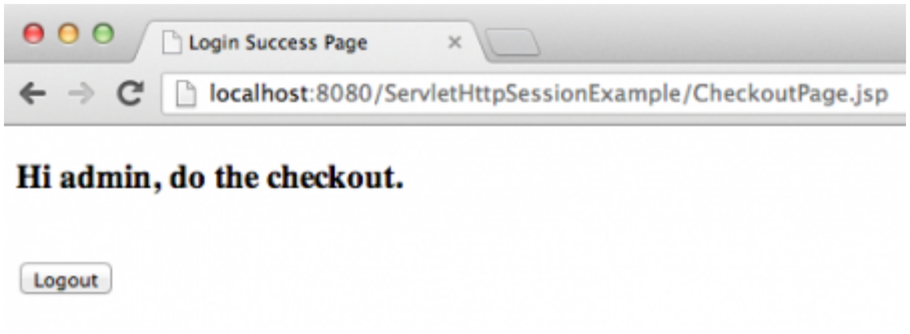
Notice that I am printing JSESSIONID cookie value in logs, you can check server log where it will be printing the same value as Session Id in LoginSuccess.jsp Below images shows the execution of our web application.





Hi admin, Login successful. Your Session ID=C97B37B942592A46FF531542A36FC0D2

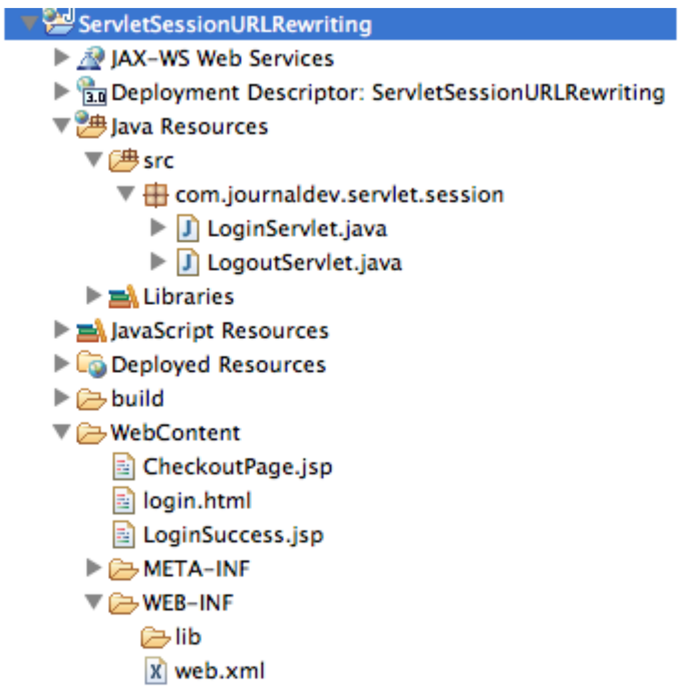
User=Pankaj
Checkout Page



Hi admin, do the checkout.

2. Session Management in Java Servlet - URL Rewriting

As we saw in last section that we can manage a session with HttpSession but if we disable the cookies in browser, it won't work because server will not receive the JSESSIONID cookie from client. Servlet API provides support for URL rewriting that we can use to manage session in this case. The best part is that from coding point of view, it's very easy to use and involves one step - encoding the URL. Another good thing with Servlet URL Encoding is that it's a fallback approach and it kicks in only if browser cookies are disabled. We can encode URL with HttpServletResponse encodeURL() method and if we have to redirect the request to another resource and we want to provide session information, we can use encodeRedirectURL() method. We will create a similar project like above except that we will use URL rewriting methods to make sure session management works fine even if cookies are disabled in browser. ServletSessionURLRewriting project structure in eclipse looks like



below image.

```
package com.journaldev.servlet.session;


import java.io.IOException;
import java.io.PrintWriter;


import javax.servlet.RequestDispatcher;
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;
import javax.servlet.http.HttpSession;


/**
 * Servlet implementation class LoginServlet
 */
@WebServlet("/LoginServlet")
public class LoginServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    private final String userID = "admin";
    private final String password = "password";

    protected void doPost(HttpServletRequest request,
                HttpServletResponse response) throws ServletException, IOException {

        // get request parameters for userID and password
        String user = request.getParameter("user");
        String pwd = request.getParameter("pwd");

        if(userID.equals(user) && password.equals(pwd)){
                HttpSession session = request.getSession();
                session.setAttribute("user", "Pankaj");
                //setting session to expiry in 30 mins
                session.setMaxInactiveInterval(30*60);
                Cookie userName = new Cookie("user", user);
                response.addCookie(userName);
                //Get the encoded URL string
                String encodedURL = response.encodeRedirectURL("LoginSuccess.jsp");
                response.sendRedirect(encodedURL);
        }else{
                RequestDispatcher rd = getServletContext().getRequestDispatcher("/login.html");
                PrintWriter out= response.getWriter();
                out.println("<font color=red>Either user name or password is wrong.</font>");
                rd.include(request, response);
        }

    }

}
<%@ page language="java" contentType="text/html; charset=US-ASCII"
```

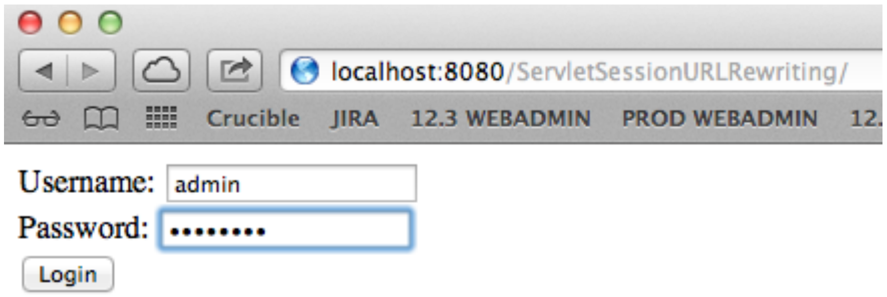
```
   pageEncoding="US-ASCII"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "https://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=US-ASCII">
<title>Login Success Page</title>
</head>
<body>
<%
//allow access only if session exists
String user = null;
if(session.getAttribute("user") == null){
    response.sendRedirect("login.html");
}else user = (String) session.getAttribute("user");
String userName = null;
String sessionID = null;
Cookie[] cookies = request.getCookies();
if(cookies !=null){
for(Cookie cookie : cookies){
    if(cookie.getName().equals("user")) userName = cookie.getValue();
    if(cookie.getName().equals("JSESSIONID")) sessionID = cookie.getValue();
}
}else{
    sessionID = session.getId();
}
%>
<h3>Hi <%=userName %>, Login successful. Your Session ID=<%=sessionID %></h3>
<br>
User=<%=user %>
<br>
<!-- need to encode all the URLs where we want session information to be passed -->
```

```
<a href="<%=response.encodeURL("CheckoutPage.jsp") %>">Checkout Page</a>
<form action="<%=response.encodeURL("LogoutServlet") %>" method="post">
<input type="submit" value="Logout" >
</form>
</body>
</html>
<%@ page language="java" contentType="text/html; charset=US-ASCII"
   pageEncoding="US-ASCII"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
     "https://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=US-ASCII">
<title>Login Success Page</title>
</head>
<body>
<%
String userName = null;
//allow access only if session exists
if(session.getAttribute("user") == null){
     response.sendRedirect("login.html");
}else userName = (String) session.getAttribute("user");
String sessionID = null;
Cookie[] cookies = request.getCookies();
if(cookies !=null){
for(Cookie cookie : cookies){
     if(cookie.getName().equals("user")) userName = cookie.getValue();
}
}
%>
<h3>Hi <%=userName %>, do the checkout.</h3>
<br>
```

```
<form action="<%=response.encodeURL("LogoutServlet") %>" method="post">
<input type="submit" value="Logout" >
</form>
</body>
</html>
package com.journaldev.servlet.session;

import java.io.IOException;

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;
import javax.servlet.http.HttpSession;

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

   protected void doPost(HttpServletRequest request, HttpServletResponse response) throws
    ServletException, IOException {
    response.setContentType("text/html");
    Cookie[] cookies = request.getCookies();
    if(cookies != null){
    for(Cookie cookie : cookies){
        if(cookie.getName().equals("JSESSIONID")){
                System.out.println("JSESSIONID="+cookie.getValue());
```

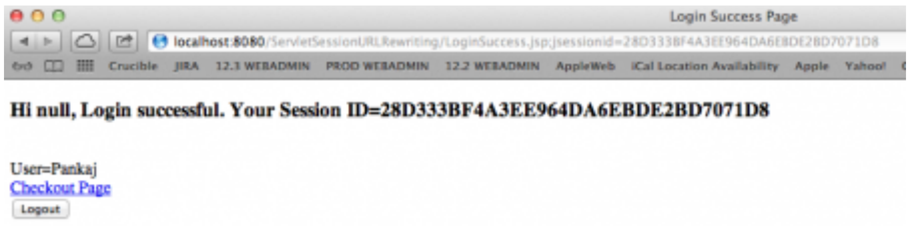
```
            }
            cookie.setMaxAge(0);
            response.addCookie(cookie);
        }
        }
        //invalidate the session if exists
        HttpSession session = request.getSession(false);
        System.out.println("User="+session.getAttribute("user"));
        if(session != null){
            session.invalidate();
        }
        //no encoding because we have invalidated the session
        response.sendRedirect("login.html");
    }


}
```

When we run this project keeping cookies disabled in the browser, below images shows the response pages, notice the jsessionid in URL of browser address bar. Also notice that on LoginSuccess page, user name is null because browser is not sending the cookie send in the last



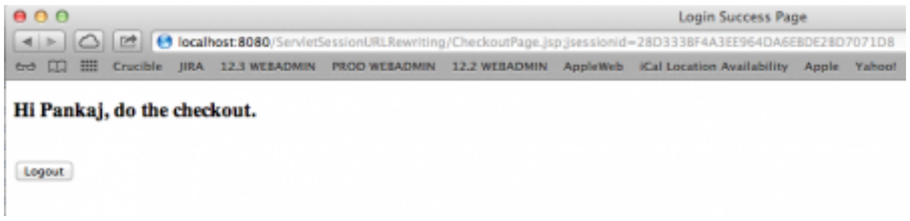
Username: admin
Password: ••••••••
Login

response.


Login Success Page
localhost:8080/ServletSessionURLRewriting/LoginSuccess.jsp;jsessionid=28D333BF4A3EE964DA6EBDE2BD7071D8
Crucible
JIRA
12.3 WEBADMIN
PROD WEBADMIN
12.2 WEBADMIN
AppleWeb
iCal Location Availability
Apple
Yahoo!

Hi null, Login successful. Your Session ID=28D333BF4A3EE964DA6EBDE2BD7071D8

User=Pankaj
Checkout Page
Logout


Login Success Page
localhost:8080/ServletSessionURLRewriting/CheckoutPage.jsp;jsessionid=28D333BF4A3EE964DA6EBDE28D7071D8
Crucible JIRA 12.3 WEBADMIN PROD WEBADMIN 12.2 WEBADMIN AppleWeb iCal Location Availability Apple Yahoo!
Logout

Hi Pankaj, do the checkout.

If cookies are not disabled, you won't see jsessionid in the URL because Servlet Session API will use cookies in that case.

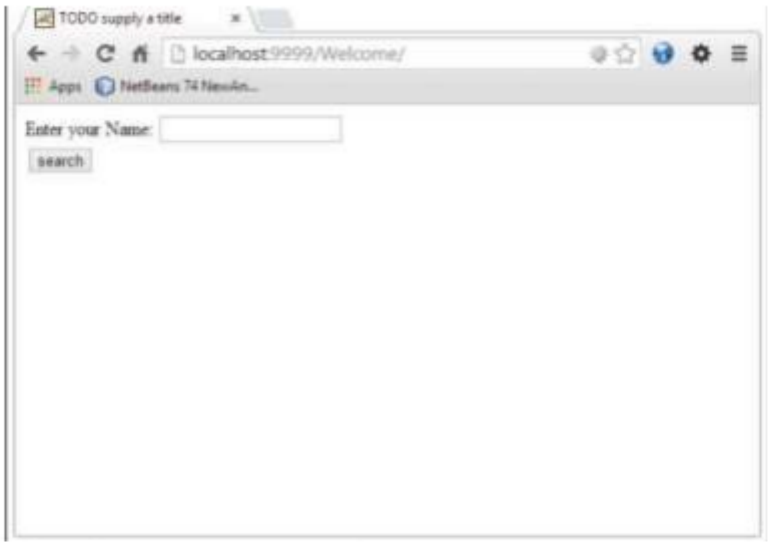
Conclusion: Hence we have performed the different Session Management techniques.

Program : 6

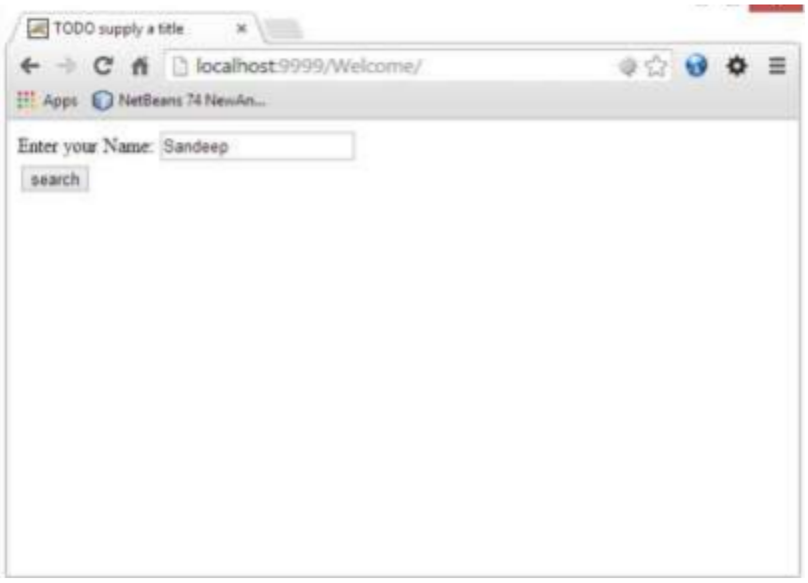
Aim : Design a simple application program using Servlet and Database 1. Simple login form 2. Customer Feedback Form 3. Admission Form 4. Student Mark Sheet.

Theory :

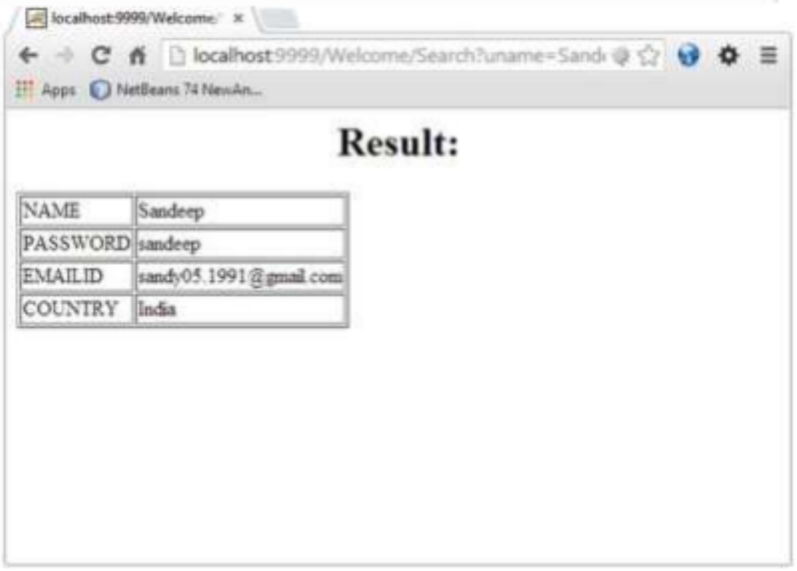
CREATE TABLE `users` (`user_name` VARCHAR(100) NOT NULL, `password` VARCHAR(100) NOT NULL, `phone_number` NUMBER(100) NOT NULL, `email_address` VARCHAR(50) NOT NULL, PRIMARY KEY (`user_name`)) COLLATE='latin1_swedish_ci' ENGINE=InnoDB



Enter your Name:



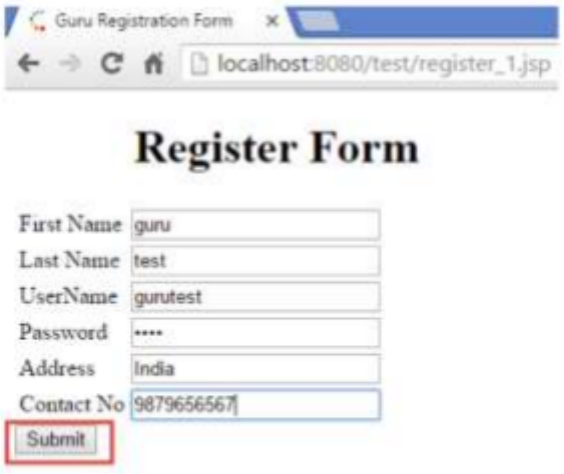
Enter your Name: Sandeep


localhost:9999/Welcome/Search?uname=Sand

Result:

NAME	Sandeep
PASSWORD	sandeep
EMAILID	sandy05.1991@gmail.com
COUNTRY	India

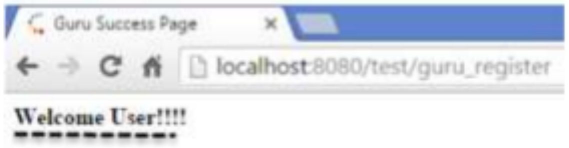
For creating registration form, you must have a table in the database. You can write the database logic in JSP file, but separating it from the JSP page is better approach. Here, we are going to use DAO, Factory Method, DTO and Singletion design patterns. There are many files: o index.jsp for getting the values from the user o User.java, a bean class that have properties and setter and getter methods. o process.jsp, a jsp file that processes the request and calls the methods o Provider.java, an interface that contains many constants like DRIVER_CLASS, CONNECTION_URL, USERNAME and PASSWORD o ConnectionProvider.java, a class that returns an object of Connection. It uses the Singleton and factory method design pattern. o RegisterDao.java, a DAO class that is responsible to get access to the database


Guru Registration Form

localhost:8080/test/register_1.jsp

Register Form

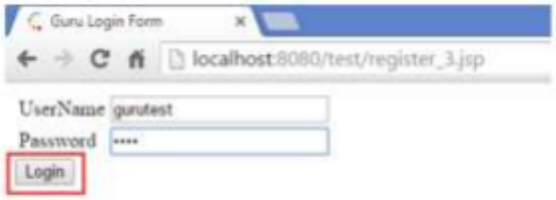
First Name	guru
Last Name	test
UserName	gurutest
Password	••••
Address	India
Contact No	9879656567


Guru Success Page

localhost:8080/test/guru_register

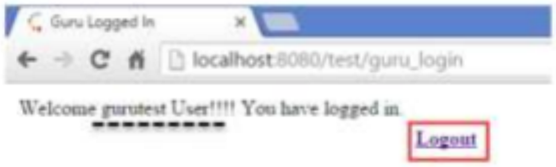
Welcome User!!!!

Authenticating the user when he submits the login form using the username and password.



UserName gurutest
Password ••••

After clicking on the Login button you get the below message with a button of Logout.



Welcome gurutest User!!!! You have logged in.

Practical : 7

Aim : Design and develop JSP application to demonstrate 1. JSP Scripting elements 2. JSP Directives 3. JSP Implicit Objects 4. JSP Action tags

Theory :

JSP technology is used to create web application just like Servlet technology. It can be thought of as an extension to Servlet because it provides more functionality than servlet such as expression language, JSTL, etc.

A JSP page consists of HTML tags and JSP tags. The JSP pages are easier to maintain than Servlet because we can separate designing and development. It provides some additional features such as Expression Language, Custom Tags, etc.

Advantages of JSP over Servlet

There are many advantages of JSP over the Servlet. They are as follows:

1) Extension to Servlet

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

2) Easy to maintain

JSP can be easily managed because we can easily separate our business logic with presentation logic. In Servlet technology, we mix our business logic with the presentation logic.

3) Fast Development: No need to recompile and redeploy

If JSP page is modified, we don't need to recompile and redeploy the project. The Servlet code needs to be updated and recompiled if we have to change the look and feel of the application.

4) Less code than Servlet

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

The Lifecycle of a JSP Page

The JSP pages follow these phases:

- Translation of JSP Page
- Compilation of JSP Page
- Classloading (the classloader loads class file)
- Instantiation (Object of the Generated Servlet is created).
- Initialization (the container invokes jspInit() method).
- Request processing (the container invokes _jspService() method).
- Destroy (the container invokes jspDestroy() method).

The JSP API consists of two packages:

1. javax.servlet.jsp
2. javax.servlet.jsp.tagext

javax.servlet.jsp package

The javax.servlet.jsp package has two interfaces and classes.The two interfaces are as follows:

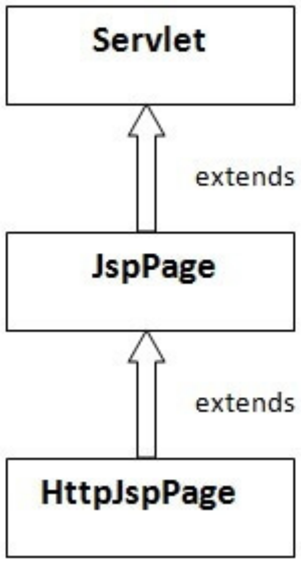
1. JspPage
2. HttpJspPage

The classes are as follows:

- JspWriter
- PageContext
- JspFactory
- JspEngineInfo
- JspException
- ▪ JspError

The JspPage interface

According to the JSP specification, all the generated servlet classes must implement the JspPage interface. It extends the Servlet interface. It provides two life cycle methods.



Methods of JspPage interface

1. **public void jspInit():** It is invoked only once during the life cycle of the JSP when JSP page is requested firstly. It is used to perform initialization. It is same as the init() method of Servlet interface.
2. **public void jspDestroy():** It is invoked only once during the life cycle of the JSP before the JSP page is destroyed. It can be used to perform some clean up operation.

The HttpJspPage interface

The HttpJspPage interface provides the one life cycle method of JSP. It extends the JspPage interface.

Method of HttpJspPage interface:

1. **public void _jspService():** It is invoked each time when request for the JSP page comes to the container. It is used to process the request. The underscore _ signifies that you cannot override this method.

Example of JSP request implicit object

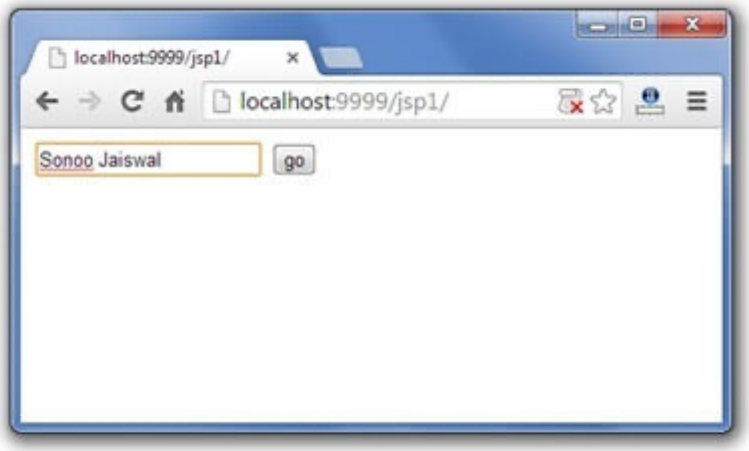
index.html

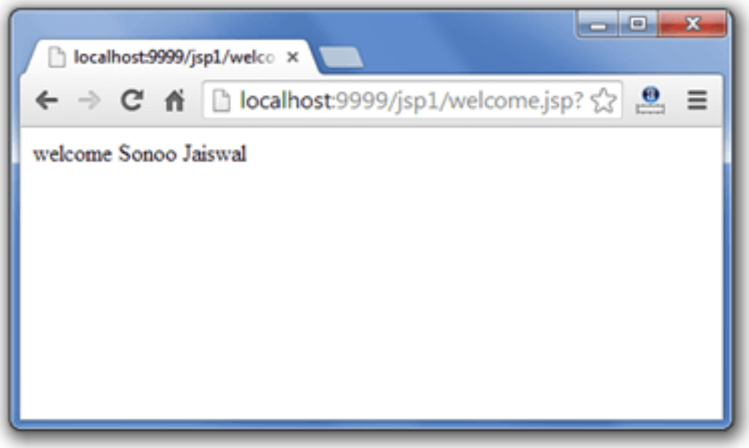
```
<form action="welcome.jsp">
```

```
<input type="text" name="uname">
<input type="submit" value="go"><br/>
</form>
```

welcome.jsp

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





welcome Sonoo Jaiswal

Practical : 8

Aim : Write a JSP program for 1. Tag 2. Exception handling in JSP.

Theory :

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

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

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

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

JSP Syntax :

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

Examples

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

Attributes and Usage

- id="beanInstanceName"

A variable that identifies the Bean in the scope you specify. You can use the variable name in expressions or scriptlets in the JSP file. The name is case sensitive and must conform to the naming conventions of the scripting language used in the JSP page. If you use the Java programming language, the conventions in the Java Language Specification. If the Bean has already been created by another <jsp:useBean> element, the value of id must match the value of id used in the original <jsp:useBean> element.

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

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

- page - You can use the Bean within the JSP page with the <jsp:useBean> element or any of the page's static include files, until the page sends a response back to the client or forwards a request to another file.
- request - You can use the Bean from any JSP page processing the same request, until a JSP page sends a response to the client or forwards the request to another file. You can use the

request object to access the Bean, for example, request.getAttribute(beanInstanceName).

- session - You can use the Bean from any JSP page in the same session as the JSP page that created the Bean. The Bean exists across the entire session, and any page that participates in the session can use it. The page in which you create the Bean must have a <%@ page %> directive with session=true.
- application - You can use the Bean from any JSP page in the same application as the JSP page that created the Bean. The Bean exists across an entire JSP application, and any page in the application can use the Bean.
- class="package.class" Instantiates a Bean from a class, using the new keyword and the class constructor. The class must not be abstract and must have a public, no-argument constructor. The package and class name are case sensitive.
- type="package.class" If the Bean already exists in the scope, gives the Bean a data type other than the class from which it was instantiated. If you use type without class or beanName, no Bean is instantiated. The package and class name are case sensitive.
- class="package.class" type="package.class" Instantiates a Bean from the class named in class and assigns the Bean the data type you specify in type. The value of type can be the same as class, a superclass of class, or an interface implemented by class.

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

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

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

```
<HTML>
<BODY>
<FORM METHOD=POST ACTION="SaveName.jsp">
What's your name? <INPUT TYPE=TEXT NAME=username
```

```
SIZE=20><BR>
What's your e-mail address? <INPUT TYPE=TEXT NAME=email
SIZE=20><BR>
What's your age? <INPUT TYPE=TEXT NAME=age SIZE=4>
<P><INPUT TYPE=SUBMIT>
</FORM>
</BODY>
</HTML>
```

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

UserData.java

```
package user;
public class UserData {
String username;
String email;
int age;
public void setUsername( String value )
{
username = value;
}
public void setEmail( String value )
{
email = value;
}
public void setAge( int value )
{
age = value;
}
public String getUsername() { return username; }
public String getEmail() { return email; }
public int getAge() { return age; }
}
```

The method names must be exactly as shown. Once you have defined the class, compile it and make sure it is available in the web-server's classpath. Now let us change "SaveName.jsp" to use a bean to collect the data.

savename.jsp

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

All we need to do now is to add the jsp:useBean tag and the jsp:setProperty tag! The useBean tag will look for an instance of the "user.UserData" in the session. If the instance is already there, it will update the old instance. Otherwise, it will create a new instance of user.UserData and put it in the session. The setProperty tag will automatically collect the input data, match names against the bean method names, and place the data in the bean!

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

nextpage.jsp

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

Step For The Exec ution :

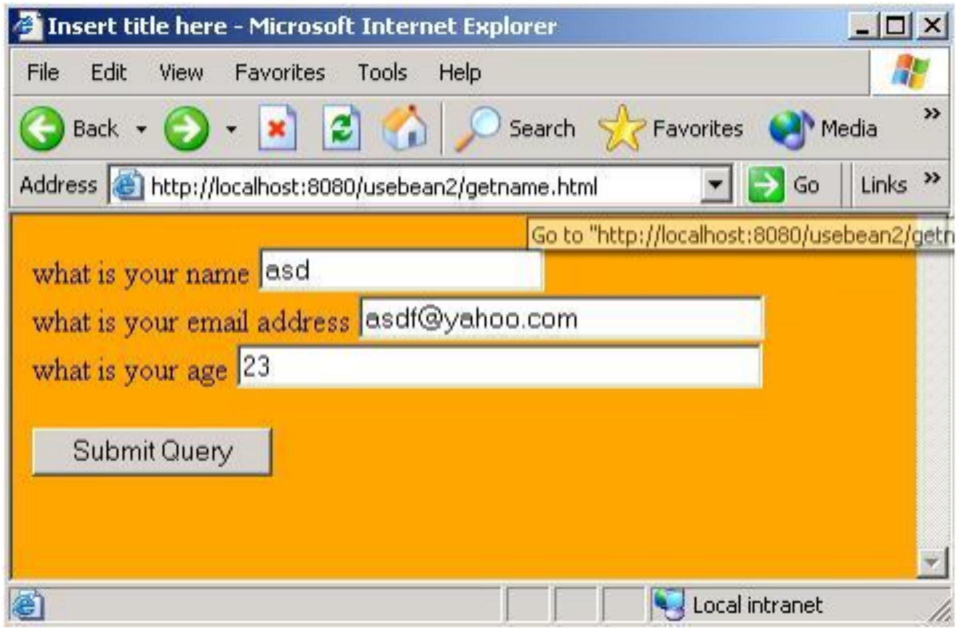
1) Remove all the errorsin program

2) Start Your Web Server

3) Deployment

4) Open Your Web Browser and Type as Following

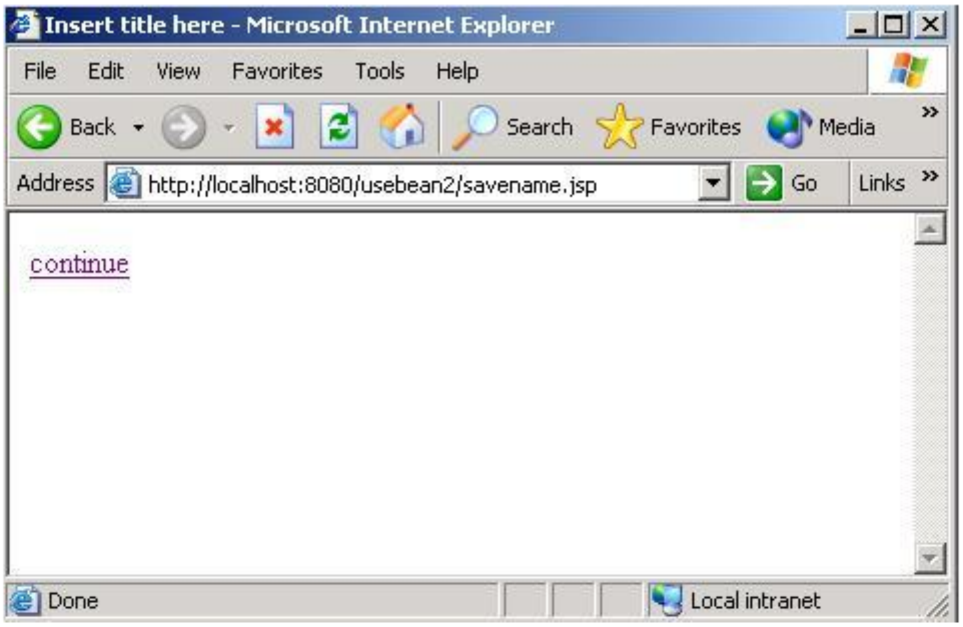
http://localhost:8080/usebean2/getname.html

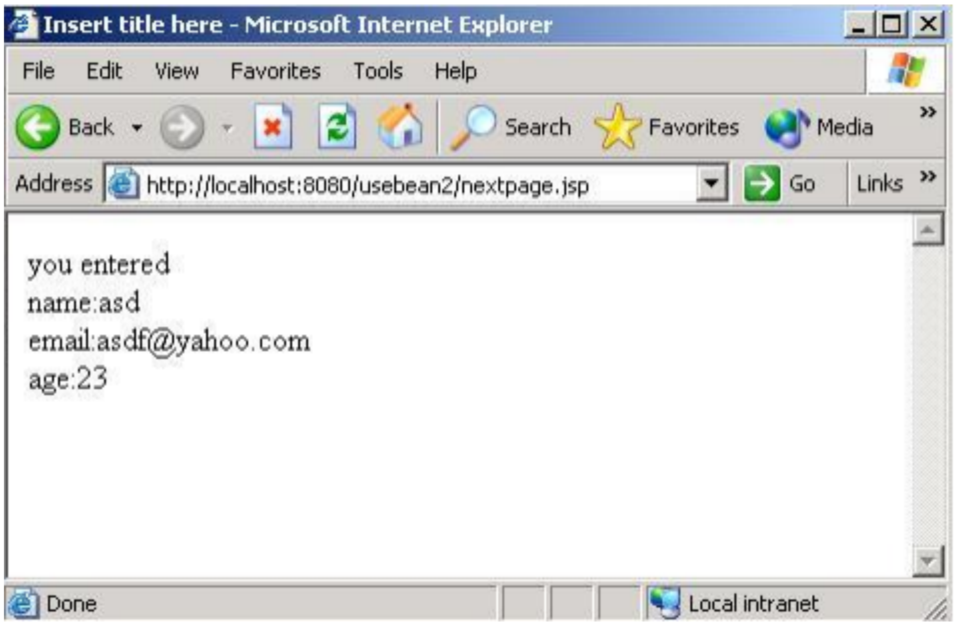

Insert title here - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8080/usebean2/getname.html Go Links
Go to "http://localhost:8080/usebean2/getn
Submit Query
Local intranet

what is your name asd

what is your email address asdf@yahoo.com

what is your age 23


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


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

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

Conclusion : Hence we studied how to use tag in jsp.

Practical : 9

Aim : Design and develop JSP practical application using JSP Custom oriented tags and JSTL tags.

Theory :

The JavaServer Pages Standard Tag Library (JSTL) is a collection of useful JSP tags which encapsulates the core functionality common to many JSP applications.

JSTL has support for common, structural tasks such as iteration and conditionals, tags for manipulating XML documents, internationalization tags, and SQL tags. It also provides a framework for integrating the existing custom tags with the JSTL tags.

Install JSTL Library

To begin working with JSP tages you need to first install the JSTL library. If you are using the Apache Tomcat container, then follow these two steps −

Step 1 − Download the binary distribution from Apache Standard Taglib and unpack the compressed file.

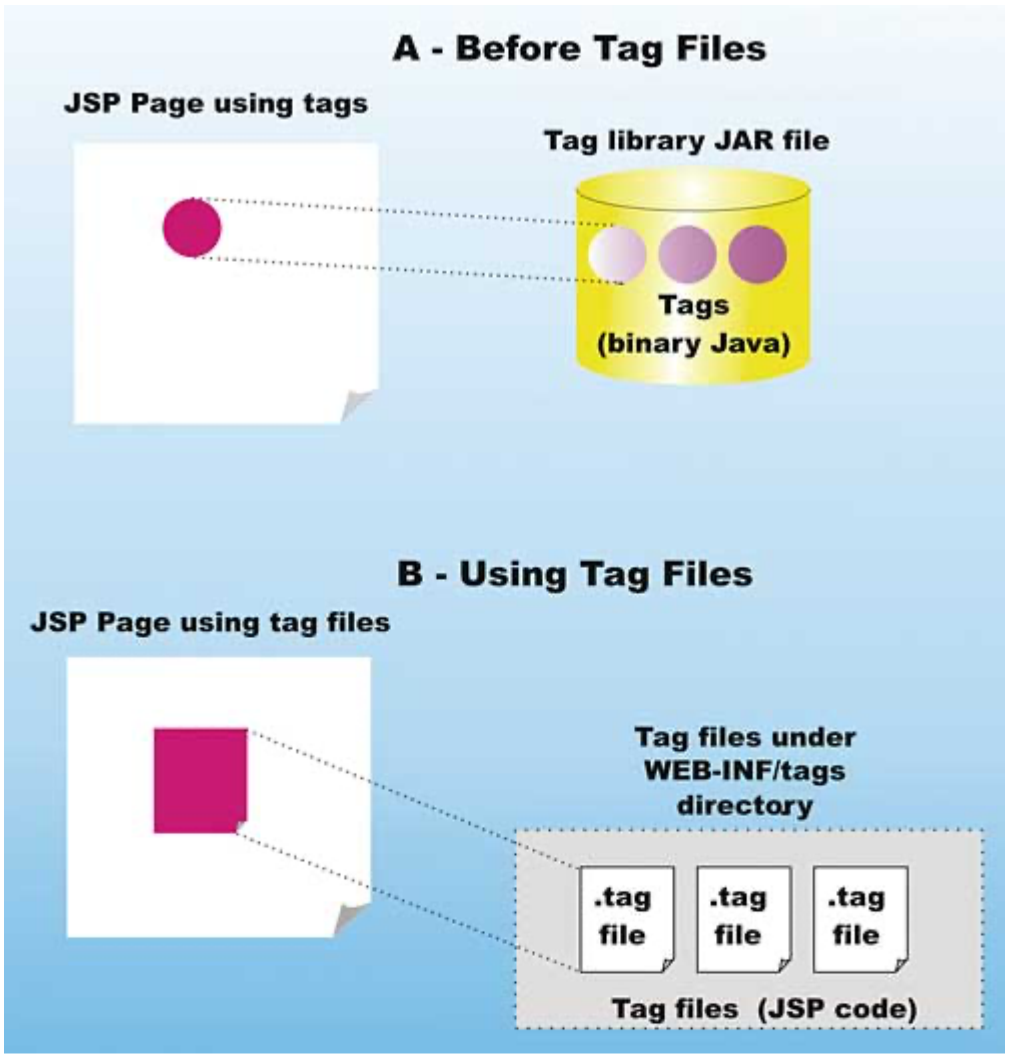
Step 2 − To use the Standard Taglib from its **Jakarta Taglibs distribution**, simply copy the JAR files in the distribution's 'lib' directory to your application's **webapps\ROOT\WEB-INF\lib** directory.

To use any of the libraries, you must include a <taglib> directive at the top of each JSP that uses the library.

Classification of The JSTL Tags

The JSTL tags can be classified, according to their functions, into the following JSTL tag library groups that can be used when creating a JSP page −

- Core Tags
- Formatting tags
- SQL tags
- XML tags
- JSTL Functions



File: MyTagHandler.java

```
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.

File: mytags.tld

```
<?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>
```

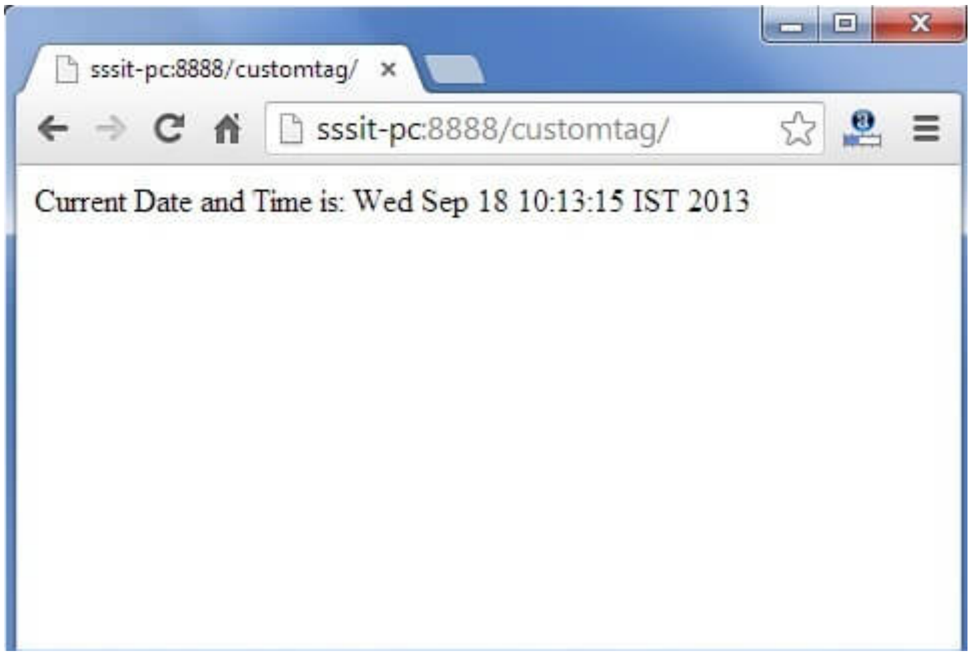
3) Create the JSP file

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/>
```



```
Current Date and Time is: Wed Sep 18 10:13:15 IST 2013
```

Conclusion : Hence we studied about different JSP tags in this experiment.

Practical : 10

Aim : Design and implement SOAP based Web Service for 1. Mathematical Calculator 2. Currency Conversion 3. Temperature Conversion.

Theory :

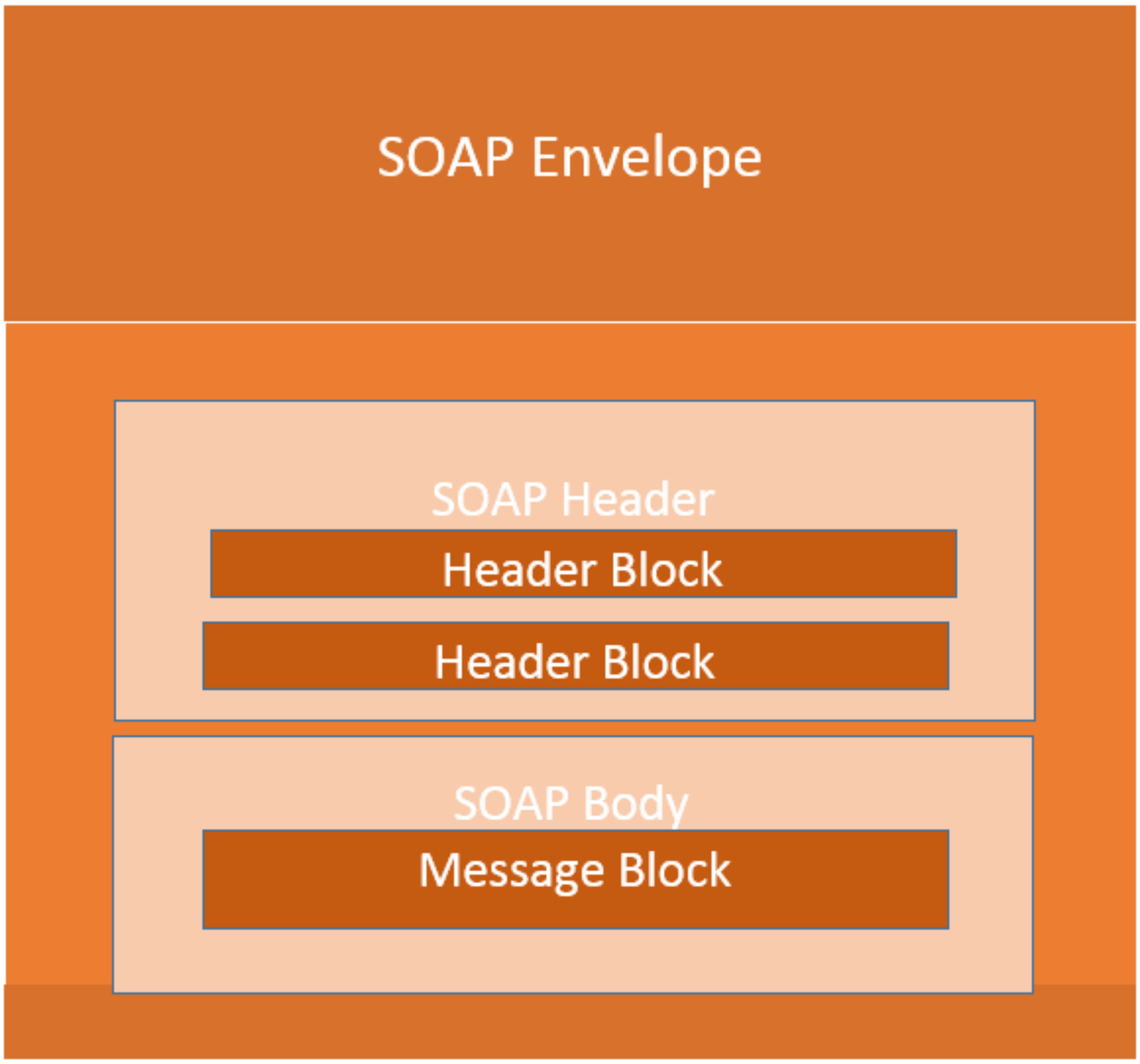
SOAP is an XML-based protocol for accessing web services over HTTP. It has some specification which could be used across all applications.

SOAP is known as the Simple Object Access Protocol, but in later times was just shortened to SOAP v1.2. SOAP is a protocol or in other words is a definition of how web services talk to each other or talk to client applications that invoke them.

Advantages of SOAP

SOAP is the protocol used for data interchange between applications. Below are some of the reasons as to why SOAP is used.

- When developing SOAP based Web services, you need to have some of language which can be used for web services to talk with client applications. SOAP is the perfect medium which was developed in order to achieve this purpose. This protocol is also recommended by the W3C consortium which is the governing body for all web standards.
- SOAP is a light-weight protocol that is used for data interchange between applications. Note the keyword '**light**.' Since SOAP programming is based on the XML language, which itself is a light weight data interchange language, hence SOAP as a protocol that also falls in the same category.
- SOAP is designed to be platform independent and is also designed to be operating system independent. So the SOAP protocol can work any programming language based applications on both Windows and Linux platform.
- It works on the HTTP protocol –SOAP works on the HTTP protocol, which is the default protocol used by all web applications. Hence, there is no sort of customization which is required to run the web services built on the SOAP protocol to work on the World Wide Web.



SOAP Message Building Blocks

The SOAP message is nothing but a mere XML document which has the below components.

- An Envelope element that identifies the XML document as a SOAP message – This is the containing part of the SOAP message and is used to encapsulate all the details in the SOAP message. This is the root element in the SOAP message.
- A Header element that contains header information – The header element can contain information such as authentication credentials which can be used by the calling application. It can also contain the definition of complex types which could be used in the SOAP message. By default, the SOAP message can contain parameters which could be of simple types such as strings and numbers, but can also be a complex object type.

A simple SOAP service example of a complex type is shown below.

Suppose we wanted to send a structured data type which had a combination of a "Tutorial Name" and a "Tutorial Description," then we would define the complex type as shown below.

The complex type is defined by the element tag <xsd:complexType>. All of the required elements of the structure along with their respective data types are then defined in the complex type collection.

```
<xsd:complexType>
 <xsd:sequence>
            <xsd:element name="Tutorial Name" type="string"/>
            <xsd:element name="Tutorial Description"  type="string"/>
  </xsd:sequence>
</xsd:complexType>
```

- A Body element that contains call and response information – This element is what contains the actual data which needs to be sent between the web service and the calling application. Below is an SOAP web service example of the SOAP body which actually works on the complex type defined in the header section. Here is the response of the Tutorial Name and Tutorial Description that is sent to the calling application which calls this web service.

```
<soap:Body>
   <GetTutorialInfo>
                        <TutorialName>Web Services</TutorialName>
                        <TutorialDescription>All about web services</TutorialDescription>
   </GetTutorialInfo>
</soap:Body>
```

SOAP Message Structure

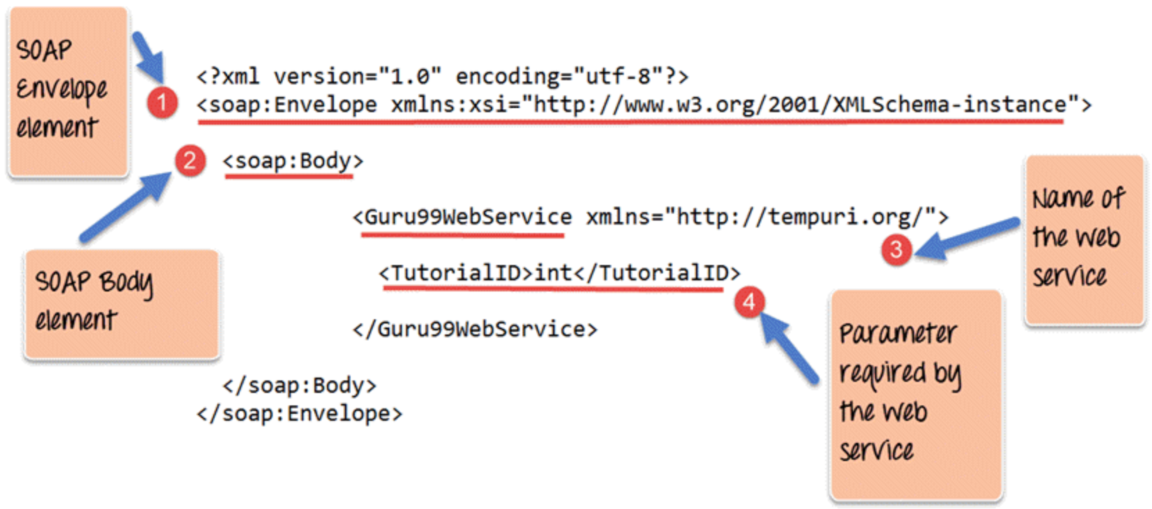
One thing to note is that SOAP messages are normally auto-generated by the web service when it is called.

Whenever a client application calls a method in the web service, the web service will automatically generate a SOAP message which will have the necessary details of the data which will be sent from the web service to the client application.

As discussed in the previous topic of this SOAP tutorial, a simple SOAP Message has the following elements –

- The Envelope element
- The header element and
- The body element
- The Fault element (Optional)

Let's look at an example below of a simple SOAP message and see what element actually does.



```
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
 <soap:Body>
     <Guru99WebService xmlns="http://tempuri.org/">
      <TutorialID>int</TutorialID>
     </Guru99WebService>
 </soap:Body>
</soap:Envelope>
```

SOAP Message Structure

1. As seen from the above SOAP message, the first part of the SOAP message is the envelope element which is used to encapsulate the entire SOAP message.
2. The next element is the SOAP body which contains the details of the actual message.
3. Our message contains a web service which has the name of "Guru99WebService".
4. The "Guru99Webservice" accepts a parameter of the type 'int' and has the name of TutorialID.

Now, the above SOAP message will be passed between the web service and the client application.

You can see how useful the above information is to the client application. The SOAP message tells the client application what is the name of the Web service, and also what parameters it expects and also what is the type of each parameter which is taken by the web service.

SOAP Envelope Element

The first bit of the building block is the SOAP Envelope.

The SOAP Envelope is used to encapsulate all of the necessary details of the SOAP messages, which are exchanged between the web service and the client application.

The SOAP envelope element is used to indicate the beginning and end of a SOAP message. This enables the client application which calls the web service to know when the SOAP message ends.

The following points can be noted on the SOAP envelope element.

- Every SOAP message needs to have a root Envelope element. It is absolutely mandatory for SOAP message to have an envelope element.
- Every Envelope element needs to have at least one soap body element.
- If an Envelope element contains a header element, it must contain no more than one, and it must appear as the first child of the Envelope, before the body element.
- The envelope changes when SOAP versions change.
- A v1.1-compliant SOAP processor generates a fault upon receiving a message containing the v1.2 envelope namespace.
- A v1.2-compliant SOAP processor generates a Version Mismatch fault if it receives a message that does not include the v1.2 envelope namespace.

Below is an SOAP API example of version 1.2 of the SOAP envelope element.

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope            xmlns:SOAP-ENV="http://www.w3.org/2001/12/soap-envelope"
SOAP-ENV:encodingStyle=" http://www.w3.org/2001/12/soap-encoding">
        <soap:Body>
      <Guru99WebService xmlns="http://tempuri.org/">
              <TutorialID>int</TutorialID>
            </Guru99WebService>
        </soap:Body>
</SOAP-ENV:Envelope>
```

Conclusion : In this experiment we studied about Simple Object Access Protocol.