

EXPERIMENT NO: 1

DATE: //

TITLE: Introduction networking in Java.**OBJECTIVES:** On completion of this experiment student will able to...

- know the concept of Networking in java.
- Create simple programs using URL Class,URLConnection Class , InetAddress Class.

THEORY:

What is URL :URL is the acronym for Uniform Resource Locator. It is a reference (an address) to a resource on the Internet. You provide URLs to your favorite Web browser so that it can locate files on the Internet in the same way that you provide addresses on letters so that the post office can locate your correspondents.

Java programs that interact with the Internet also may use URLs to find the resources on the Internet they wish to access. Java programs can use a class called URL in the java.net package to represent a URL address. The URL class provides several methods that let you query URL objects. You can get the protocol, authority, host name, port number, path, query, filename, and reference from a URL using these accessor methods: The URL class contains many methods for accessing the various parts of the URL being represented. Some of the methods in the URL class include the following –

**Experiment 1: Demonstrate use of URClass, URL Connection And
InetAddress Chat Application Using TCP and UDP.**

a. URL Class

```
import java.net.*;

import java.io.*;

public class URL_class
{
    public static void main(String args[])
    {
        try{
            //declare the url object

            URL obj=new URL("http://www.javatpoint.com/java-tutorial");

            // methods of geting port,host,path,filename etc
```

```
        System.out.println("PROTOCOL:"+obj.getProtocol());

        System.out.println("PORT:"+obj.getPort());

        System.out.println("HOST:"+obj.getHost());

        System.out.println("FILE:"+obj.getFile());

        System.out.println("PATH:"+obj.getPath());

    }

    catch(Exception e)

    {

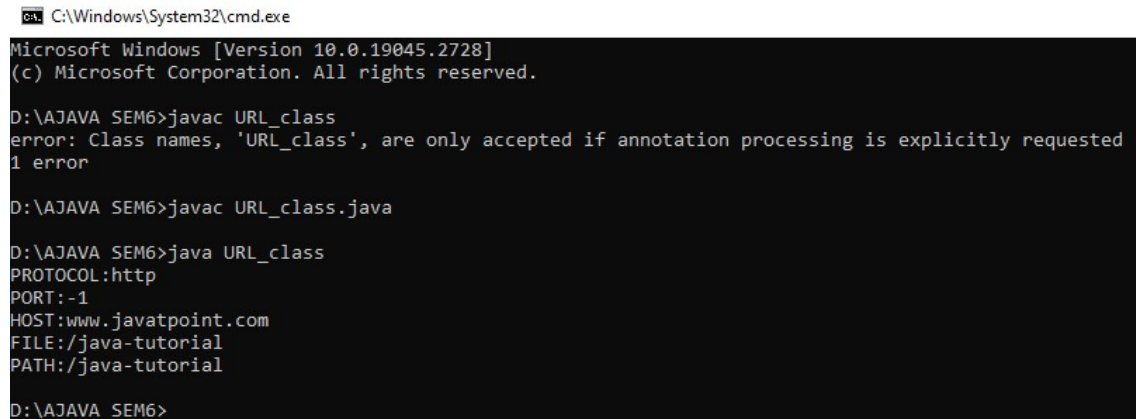
        System.out.println(e);

    }

}

}
```

Output:



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.2728]
(c) Microsoft Corporation. All rights reserved.

D:\AJAVA SEM6>javac URL_class
error: Class names, 'URL_class', are only accepted if annotation processing is explicitly requested
1 error

D:\AJAVA SEM6>javac URL_class.java

D:\AJAVA SEM6>java URL_class
PROTOCOL:http
PORT:-1
HOST:www.javatpoint.com
FILE:/java-tutorial
PATH:/java-tutorial

D:\AJAVA SEM6>
```

b. URL Connection:

```
import java.io.*;

import java.net.*;

public class url_connection

{

public static void main(String[] args)

{

try

{

//first url object are created

URL url=new URL("https://internship.aicte-

india.org/login_new.php?task=PlseLogin");

//here openConnection method are declared

URLConnection urlcon=url.openConnection();

//here getInputStream method is used to get data from the url (link) and read all content

InputStream stream=urlcon.getInputStream();

int i;

while((i=stream.read())!=-1)

{

System.out.print((char)i);

}

}

catch(Exception e)

{

}
```

```

        System.out.println(e);
    }
}
}
}

```

Output:

```

D:\AJAVA SEM6>javac url_connection.java
D:\AJAVA SEM6>java url_connection
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>AICTE Internship Enterprise Portal - Learning by doing. 1 Crore Internships by 2025</title>
  <meta name="description" content="" />
  <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
  <meta http-equiv="X-UA-Compatible" content="IE=9" />
  <link rel="shortcut icon" href="/images/favicons/favicon.ico">
  <link rel="icon" type="image/png" sizes="16x16" href="/images/favicons/favicon-16x16.png">
  <link rel="icon" type="image/png" sizes="32x32" href="/images/favicons/favicon-32x32.png">
  <link rel="manifest" href="/images/favicons/manifest.json">
  <meta name="mobile-web-app-capable" content="yes">
  <meta name="theme-color" content="#ffff">
  <meta name="application-name" content="AICTE">
  <link rel="apple-touch-icon" sizes="57x57" href="/images/favicons/apple-touch-icon-57x57.png">
  <link rel="apple-touch-icon" sizes="60x60" href="/images/favicons/apple-touch-icon-60x60.png">
  <link rel="apple-touch-icon" sizes="72x72" href="/images/favicons/apple-touch-icon-72x72.png">
  <link rel="apple-touch-icon" sizes="76x76" href="/images/favicons/apple-touch-icon-76x76.png">
  <link rel="apple-touch-icon" sizes="114x114" href="/images/favicons/apple-touch-icon-114x114.png">
  <link rel="apple-touch-icon" sizes="120x120" href="/images/favicons/apple-touch-icon-120x120.png">
  <link rel="apple-touch-icon" sizes="144x144" href="/images/favicons/apple-touch-icon-144x144.png">
  <link rel="apple-touch-icon" sizes="152x152" href="/images/favicons/apple-touch-icon-152x152.png">
  <link rel="apple-touch-icon" sizes="167x167" href="/images/favicons/apple-touch-icon-167x167.png">
  <link rel="apple-touch-icon" sizes="180x180" href="/images/favicons/apple-touch-icon-180x180.png">
  <link rel="apple-touch-icon" sizes="1024x1024" href="/images/favicons/apple-touch-icon-1024x1024.png">
  <meta name="apple-mobile-web-app-capable" content="yes">
  <meta name="apple-mobile-web-app-status-bar-style" content="black-translucent">
  <meta name="apple-mobile-web-app-title" content="AICTE">
  <meta name="viewport" content="width=device-width,initial-scale=1">
  <link href="style.1c48d001a3f8a2fd1816.css" rel="stylesheet">
  <link rel="stylesheet" href="style.css">
  <script src="/jquery-3.5.1.min.js"></script>
  <script src="/sweetalert.min.js"></script>
  <script src="/assets/js/validate.js"></script>
  <script src="/assets/js/cities.js"></script>
</head>
<script async src="https://www.googletagmanager.com/gtag/js?id=UA-162699883-2"></script>
<script>

```

c. InetAddress

```
// practical 1 program3
```

```
import java.io.*;
```

```
import java.net.*;
```

```
public class inetAddress
```

```
{
```

```
    public static void main(String args[])
```

```
    {
```

```
        try{
```

```
            // declared ip address object and gave name of url
```

```
InetAddress ip=InetAddress.getByName("www.pwskills.com");

System.out.println("Local host is:"+ip.getHostName());

System.out.println("IP ADDRESS is:"+ip.getHostAddress());

}

catch(Exception e)

{

    System.out.println(e);

}

}

}
```

Output:

```
D:\AJAVA SEM6>javac inetAddress.java

D:\AJAVA SEM6>java inetAddress
Local host is:www.pwskills.com
IP ADDRESS is:104.21.62.86

D:\AJAVA SEM6>_
```

EVALUATION:

Problem Analysis & Solution (3)	Understanding Level (3)	Timely Completion (2)	Mock (2)	Total (10)

EXPERIMENT NO: 2**DATE: //****TITLE:** Chat Application using TCP and UDP.**OBJECTIVES:** On completion of this experiment student will able to...

- know the concept of Socket class and Datagram class.
- Create simple programs using Socket , ServerSocketClass, DatagramPacket, DatagramSocket.

THEORY:

URLs and URLConnections provide a relatively high-level mechanism for accessing resources on the Internet. Sometimes your programs require lower-level network communication, for example, when you want to write a client-server application. In client-server applications, the server provides some service, such as processing database queries or sending out current stock prices. The client uses the service provided by the server, either displaying database query results to the user or making stock purchase recommendations to an investor. The communication that occurs between the client and the server must be reliable. That is, no data can be dropped and it must arrive on the client side in the same order in which the server sent it. TCP provides a reliable, point-to-point communication channel that client-server applications on the Internet use to communicate with each other. To communicate over TCP, a client program and a server program establish a connection to one another. Each program binds a socket to its end of the connection. To communicate, the client and the server each reads from and writes to the socket bound to the connection.

What is Socket and ServerSocket:

Normally, a server runs on a specific computer and has a socket that is bound to a specific port number. The server just waits, listening to the socket for a client to make a connection request. On the client-side: The client knows the hostname of the machine on which the server is running and the port number on which the server is listening. To make a connection request, the client tries to rendezvous with the server on the server's machine and port. The client also needs to identify itself to the server so it binds to a local port number that it will use during this connection. This is usually assigned by the system.



If everything goes well, the server accepts the connection. Upon acceptance, the server gets a new socket bound to the same local port and also has its remote endpoint set to the address and port of the client. It needs a new socket so that it can continue to listen to the original socket for connection requests while tending to the needs of the connected client.



On the client side, if the connection is accepted, a socket is successfully created and the client can use the socket to communicate with the server. The client and server can now communicate by writing to or reading from their sockets. Client Server Application using Socket

Experiment 2: Chat Application Using TCP and UDP (DatagramSocket, DatagramPac class, Socket, ServerSocket).

a. TCP Client Server Programs(Socket Programming)

//server side

```
package com.mycompany.serversocket;
import java.io.*;
import java.net.*;

public class ServerSocket {
    public static void main(String[] args) throws IOException {
        try (java.net.ServerSocket serverSocket = new java.net.ServerSocket(6666))
        {
            System.out.println("Server started. Listening for connections...");

            Socket clientSocket = serverSocket.accept();
            System.out.println("Client connected: " + clientSocket.getInetAddress());

            // set up streams for reading and writing data to the client
            InputStream inputStream = clientSocket.getInputStream();
            OutputStream outputStream = clientSocket.getOutputStream();
            BufferedReader reader = new BufferedReader(new
            InputStreamReader(inputStream));
            PrintWriter writer = new PrintWriter(outputStream, true);

            // create a separate thread to read messages from the client
            Thread clientThread = new Thread(() -> {
                String clientMessage;
```

```

        try {
            while ((clientMessage = reader.readLine()) != null) {
                System.out.println("Client: " + clientMessage);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    });
    clientThread.start();

    // read messages from the console and send them to the client
    BufferedReader consoleReader = new BufferedReader(new
InputStreamReader(System.in));
    String consoleMessage;
    while ((consoleMessage = consoleReader.readLine()) != null) {
        writer.println("Server: " + consoleMessage);
    }

    // clean up resources
    writer.close();
    reader.close();
    clientSocket.close();
}
}
}

```

//Client side

```

package com.mycompany.clientsocket;

import java.io.*;
import java.net.*;

public class ClientSocket {
    public static void main(String[] args) throws IOException {
        Socket clientSocket = new Socket("localhost", 6666);
        System.out.println("Connected to server.");

        // set up streams for reading and writing data to the server
        InputStream inputStream = clientSocket.getInputStream();
        OutputStream outputStream = clientSocket.getOutputStream();
        BufferedReader reader = new BufferedReader(new
InputStreamReader(inputStream));
        PrintWriter writer = new PrintWriter(outputStream, true);
    }
}

```



```

// create a separate thread to read messages from the server
Thread serverThread = new Thread() -> {
    String serverMessage;
    try {
        while ((serverMessage = reader.readLine()) != null) {
            System.out.println(serverMessage);
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
};
serverThread.start();

// read messages from the console and send them to the server
BufferedReader consoleReader = new BufferedReader(new
InputStreamReader(System.in));
String consoleMessage;
while ((consoleMessage = consoleReader.readLine()) != null) {
    writer.println("Client: " + consoleMessage);
}

// clean up resources
writer.close();
reader.close();
clientSocket.close();
}
}

```

Outputs:

```

Output
Run (ServerSocket) x Run (ClientSocket) x
cd C:\Users\Vandan Raval\Documents\NetBeansProjects\clientSocket; "J
Running NetBeans Compile On Save execution. Phase execution is skippe
Scanning for projects...
-----< com.mycompany:clientSocket >-----
Building clientSocket 1.0-SNAPSHOT
-----[ jar ]-----
--- exec-maven-plugin:3.0.0:exec (default-cli) @ clientSocket ---
Connected to server.
hi
Server: hello vandan how are you
i am fine gopal
|

```

```

Output
Run (ServerSocket) x Run (ClientSocket) x
cd C:\Users\Vandan Raval\Documents\NetBeansProjects\serverSocket; "JAVA
Running NetBeans Compile On Save execution. Phase execution is skipped
Scanning for projects...

-----< com.mycompany:serverSocket >-----
Building serverSocket 1.0-SNAPSHOT
-----[ jar ]-----

--- exec-maven-plugin:3.0.0:exec (default-cli) @ serverSocket ---
Server started. Listening for connections...
Client connected: /127.0.0.1
Client: Client: hi
hello vandan how are you
Client: Client: i am fine gopal

```

b. UDP Client Server Programs(Socket Programming)

UDP Server Program:

```
package com.mycompany.serversocket;
```

```
import java.io.*;
```

```
import java.net.*;
```

```
public class ServerSocket {
```

```
    public static void main(String[] args) throws IOException {
```

```
        DatagramSocket serverSocket = new DatagramSocket(6666);
```

```
        System.out.println("Server started. Listening for connections...");
```

```
        byte[] receiveData = new byte[1024];
```

```
        byte[] sendData = new byte[1024];
```

```
        while (true) {
```

```
            DatagramPacket receivePacket = new DatagramPacket(receiveData,
receiveData.length);
```

```
            serverSocket.receive(receivePacket);
```

```
            String clientMessage = new String(receivePacket.getData()).trim();
```

```
            System.out.println("Client: " + clientMessage);
```

```
            // read messages from the console and send them to the client
```

```
            BufferedReader consoleReader = new BufferedReader(new
InputStreamReader(System.in));
```

```
            String consoleMessage = consoleReader.readLine();
```

```
            InetAddress clientAddress = receivePacket.getAddress();
```

```
            int clientPort = receivePacket.getPort();
```

```

        sendData = ("Server: " + consoleMessage).getBytes();
        DatagramPacket sendPacket = new DatagramPacket(sendData,
sendData.length, clientAddress, clientPort);
        serverSocket.send(sendPacket);
    }
}
}

```

What is DatagramPacket and DatagramSocket:

Java DatagramSocket and DatagramPacket classes are used for connection-less socket programming. Java DatagramSocket class represents a connection-less socket for sending and receiving datagram packets. A datagram is basically an information but there is no guarantee of its content, arrival or arrival time. Commonly used Constructors of DatagramSocket class.

DatagramSocket() throws SocketEption: it creates a datagram socket and binds it with the available Port Number on the localhost machine. **DatagramSocket(int port) throws SocketEption:** it creates a datagram socket and binds it with the given Port Number. **DatagramSocket(int port, InetAddress address) throws SocketEption:** it creates a datagram socket and binds it with the specified port number and host address.

Java DatagramPacket : is a message that can be sent or received. If you send multiple packet, it may arrive in any order. Additionally, packet delivery is not guaranteed. Commonly used Constructors of DatagramPacket class **DatagramPacket(byte[] barr, int length):** it creates a datagram packet. This constructor is used to receive the packets.

DatagramPacket(byte[] barr, int length, InetAddress address, int port): it creates a datagram packet. This constructor is used to send the packets. Example of Sending DatagramPacket by DatagramSocket

UDP Client Program:

```

package com.mycompany.clientsocket;
import java.io.*;
import java.net.*;

public class ClientSocket {
    public static void main(String[] args) throws IOException {
        DatagramSocket clientSocket = new DatagramSocket();
        System.out.println("Connected to server.");

        byte[] receiveData = new byte[1024];
        byte[] sendData = new byte[1024];
    }
}

```

```
// create a separate thread to read messages from the server
Thread serverThread = new Thread() -> {
    while (true) {
        DatagramPacket receivePacket = new DatagramPacket(receiveData,
receiveData.length);
        try {
            clientSocket.receive(receivePacket);
            String serverMessage = new String(receivePacket.getData()).trim();
            System.out.println(serverMessage);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
});
serverThread.start();

// read messages from the console and send them to the server
BufferedReader consoleReader = new BufferedReader(new
InputStreamReader(System.in));
String consoleMessage;
InetAddress serverAddress = InetAddress.getByName("localhost");
int serverPort = 6666;
while ((consoleMessage = consoleReader.readLine()) != null) {
    sendData = ("Client: " + consoleMessage).getBytes();
    DatagramPacket sendPacket = new DatagramPacket(sendData,
sendData.length, serverAddress, serverPort);
    clientSocket.send(sendPacket);
}

// clean up resources
clientSocket.close();
}
}
```

Output:

Output

Run (ServerSocket) ×
Run (ClientSocket) ×

cd C:\Users\Vandan Raval\Documents\NetBeansProjects\clientSocket; "JAVA_HOME=C:\\Program Files\\
Running NetBeans Compile On Save execution. Phase execution is skipped and output directories of
Scanning for projects...
-----< com.mycompany:clientSocket >-----
Building clientSocket 1.0-SNAPSHOT
-----[jar]-----
--- exec-maven-plugin:3.0.0:exec (default-cli) @ clientSocket ---
Connected to server.
hello...how are you
Server: i am fine ..what about you?

Building serverSocket 1.0-SNAPSHOT
-----[jar]-----
--- exec-maven-plugin:3.0.0:exec (default-cli) @ serverSocket ---
Server started. Listening for connections...
Client: Client: hello...how are you
i am fine ..what about you?

EVALUATION:

Problem Analysis & Solution (3)	Understanding Level (3)	Timely Completion (2)	Mock (2)	Total (10)

EXPERIMENT NO: 3

DATE: / /

TITLE: JDBC Programming Concepts.

OBJECTIVES: On completion of this experiment student will able to...

- know the concept of JDBC Driver Types of Different Classes to connect to database and How to get data from database and CRUD Operation.
- Create simple programs using Statement class, PreparedStatement class, CallableStatement class, ResultSet class, ResultSetMetaData.

THEORY: JDBC is Java application programming interface that allows the Java programmers to access database management system from Java code. It was developed by JavaSoft, a subsidiary of Sun Microsystems.

Definition:

Java Database Connectivity in short called as JDBC. It is a java API which enables the java programs to execute SQL statements. It is an application programming interface that defines how a java programmer can access the database in tabular format from Java code using a set of standard interfaces and classes written in the Java programming language.

JDBC has been developed under the Java Community Process that allows multiple implementations to exist and be used by the same application. JDBC provides methods for querying and updating the data in Relational Database Management system such as SQL, Oracle etc. The primary function of the JDBC API is to provide a means for the developer to issue SQL statements and process the results in a consistent, database-independent manner. JDBC provides rich, object-oriented access to databases by defining classes and interfaces that represent objects such as:

1. Database connections
2. SQL statements
3. Result Set
4. Database metadata
5. Prepared statements
6. Binary Large Objects (BLOBs)
7. Character Large Objects (CLOBs)
8. Callable statements
9. Database drivers
10. Driver manager

Processing SQL Statements with JDBC

In general, to process any SQL statement with JDBC, you follow these steps:

1. Establishing a connection.
2. Create a statement.
3. Execute the query.
4. Process the ResultSet object.
5. Close the connection.

Statement interface The Statement interface provides methods to execute queries with the database. The statement interface is a factory of ResultSet i.e. it provides factory method to get the object of ResultSet.

Experiment 3: JDBC Programming Concepts(Statement ,PreparedStatement , CallableStatement , Resultset)

```
import java.sql.*;

public class createDB {

    // Connection parameters

    static final String url = "jdbc:mysql://localhost/";

    static final String username = "root";

    static final String password = "";

    public static void main(String[] args) {

        // Database name

        String dbName = "Manual_501";

        // Create a connection to the database server

        try (Connection conn = DriverManager.getConnection(url, username, password)) {

            // Create a statement object

            Statement stmt = conn.createStatement();

            // SQL query to create the database

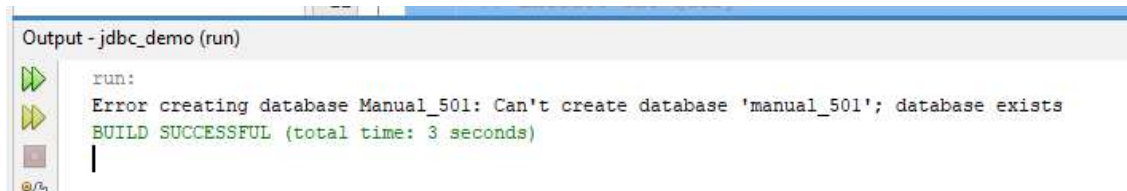
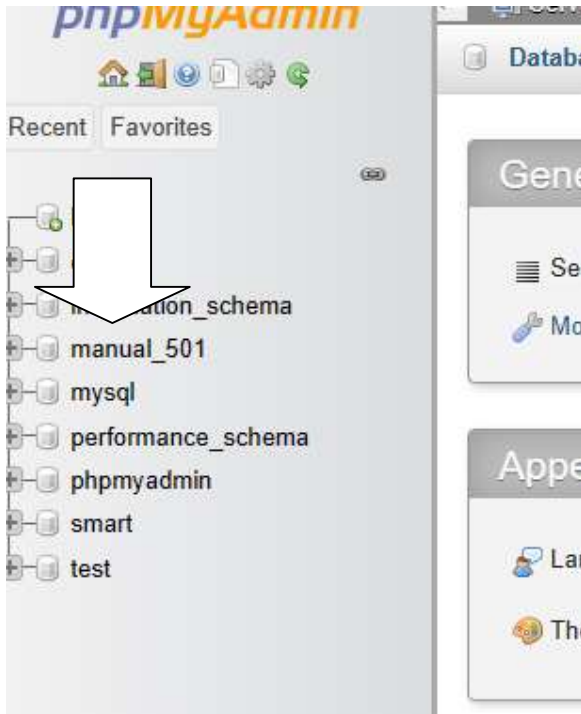
            String sql = "CREATE DATABASE " + dbName;

            // Execute the query

            stmt.executeUpdate(sql);
```

```
System.out.println("Database " + dbName + " created successfully!");  
} catch (SQLException e) {  
    System.out.println("Error creating database " + dbName + ": " + e.getMessage());  
}  
}  
}
```

Output:



CREATE TABLE:

```

import java.sql.*;

public class createTBL {

    // Connection parameters

    static final String url = "jdbc:mysql://localhost/manual_501";

    static final String username = "root";

    static final String password = "";

    public static void main(String[] args) {

// Table name and column names

        String tableName = "users";

// String column1 = "id INT NOT NULL AUTO_INCREMENT PRIMARY KEY";

        String column2 = "name VARCHAR(50) NOT NULL";

        String column3 = "email VARCHAR(50) NOT NULL";

// Create a connection to the database server

        try (Connection conn = DriverManager.getConnection(url, username, password)) {

            // Create a statement object

            Statement stmt = conn.createStatement();

            // SQL query to create the table

            String sql = "CREATE TABLE " + tableName + "(" + column1 + ", " + column2 + ", " +
column3 + ")";

            // Execute the query

            stmt.executeUpdate(sql);

            System.out.println("Table " + tableName + " created successfully!");

        } catch (SQLException e) {

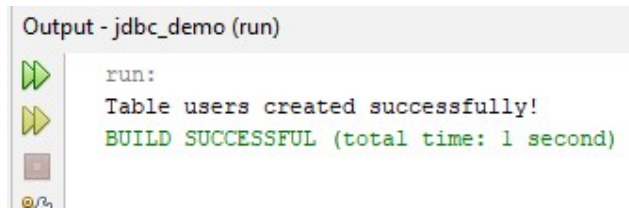
            System.out.println("Error creating table " + tableName + ": " + e.getMessage());

        }

    }

}

```

Output:


```
Output - jdbc_demo (run)
run:
Table users created successfully!
BUILD SUCCESSFUL (total time: 1 second)
```

PreparedStatement interface

The PreparedStatement interface is a subinterface of Statement. It is used to execute parameterized query. String sql="insert into emp values(?,?,?)"; As you can see, we are passing parameter (?) for the values. Its value will be set by calling the setter methods of PreparedStatement.

Why use PreparedStatement? Improves performance: The performance of the application will be faster if you use PreparedStatement interface because query is compiled only once.

Example of PreparedStatement interface that inserts the record

```
import java.sql.*;

public class InsertData {

    public static void main(String[] args) {

        // Connection parameters

        String url = "jdbc:mysql://localhost/mydatabase";

        String username = "root";

        String password = "mypassword";

        // User data to be inserted

        String name = "Vandan Raval";

        String email = "vandanraval2002@gmail.com";

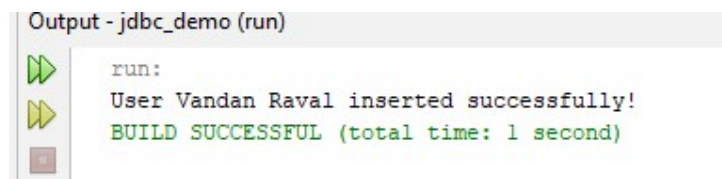
        // SQL query with prepared statement placeholders
```

```
String sql = "INSERT INTO users (name, email) VALUES (?, ?)";

try (Connection conn = DriverManager.getConnection(url, username, password);
    PreparedStatement stmt = conn.prepareStatement(sql)) {
    // Set the values for the prepared statement parameters
    stmt.setString(1, name);
    stmt.setString(2, email)
    // Execute the query
    int rowsAffected = stmt.executeUpdate();

    if (rowsAffected > 0) {
        System.out.println("User " + name + " inserted successfully!");
    } else {
        System.out.println("Error inserting user " + name + ".");
    }
} catch (SQLException e) {
    System.out.println("Error inserting user " + name + ": " + e.getMessage());
}
}
```

Output:



```
Output - jdbc_demo (run)
run:
User Vandan Raval inserted successfully!
BUILD SUCCESSFUL (total time: 1 second)
```

	id	name	email
<input type="checkbox"/> Edit Copy Delete	1	Vandan Raval	vandanraval2002@gmail.com

Java ResultSetMetaData Interface:

The metadata means data about data i.e. we can get further information from the data. If you have to get metadata of a table like total number of column, column name, column type etc. , ResultSetMetaData interface is useful because it provides methods to get metadata from the ResultSet object.

```
import java.sql.*;

public class resultsetmetadataex {

    public static void main(String[] args) {

        // Connection parameters

        String url = "jdbc:mysql://localhost/manual_501";

        String username = "root";

        String password = "";

        // SQL query to retrieve user data

        String sql = "SELECT * FROM users";

        try (Connection conn = DriverManager.getConnection(url, username, password);

            Statement stmt = conn.createStatement();

            ResultSet rs = stmt.executeQuery(sql)) {

            // Get the ResultSet metadata

            ResultSetMetaData rsmd = rs.getMetaData();

            // Print the number of columns in the ResultSet

            int numColumns = rsmd.getColumnCount();

            System.out.println("Number of columns: " + numColumns);

            // Print information about each column

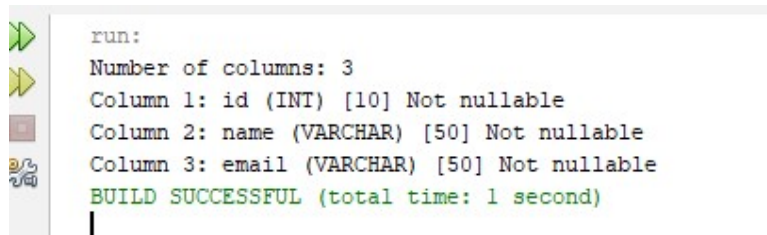
            for (int i = 1; i <= numColumns; i++) {

                String columnName = rsmd.getColumnName(i);
```

```
String columnType = rsmd.getColumnTypeName(i);
int columnSize = rsmd.getColumnDisplaySize(i);
boolean isNullable = (rsmd.isNullable(i) == ResultSetMetaData.columnNullable);

System.out.println("Column " + i + ": " + columnName + " (" + columnType + ") " +
    "[" + columnSize + "]" + (isNullable ? " Nullable" : " Not nullable"));
}
} catch (SQLException e) {
    System.out.println("Error getting ResultSet metadata: " + e.getMessage());
}
}
}
```

Output:



```
run:
Number of columns: 3
Column 1: id (INT) [10] Not nullable
Column 2: name (VARCHAR) [50] Not nullable
Column 3: email (VARCHAR) [50] Not nullable
BUILD SUCCESSFUL (total time: 1 second)
```

Java CallableStatement Interface

CallableStatement interface is used to call the stored procedures and functions. We can have business logic on the database by the use of stored procedures and functions that will make the performance better because these are precompiled. Suppose you need to get the age of the employee based on the date of birth, you may create a function that receives date as the input and returns age of the employee as the output.

```
import java.sql.*;

public class callablestmt {

    public static void main(String[] args) {

        // Connection parameters

        String url = "jdbc:mysql://localhost/manual_501";

        String username = "root";

        String password = "";

        try (Connection conn = DriverManager.getConnection(url, username, password);

            CallableStatement cstmt = conn.prepareCall("{CALL storedproc()}")) {

            int rows = cstmt.executeUpdate();

            System.out.println("Rows affected:"+rows);

        } catch (SQLException e) {

            System.out.println("Error executing stored procedure: " + e.getMessage());

        }

    }

}
```

```
CREATE PROCEDURE storedproc()

BEGIN

INSERT INTO `users`(`id`, `name`, `email`) VALUES
(3,'maulik','maulik7896@gmail.com');

END
```

OUTPUT:

Output - jdbc_demo (run)

```
run:
Rows affected:1
BUILD SUCCESSFUL (total time: 1 second)
```

				id	name	email
<input type="checkbox"/>		Edit		Copy		Delete
1	Vandan Raval	vandanraval2002@gmail.com				
<input type="checkbox"/>		Edit		Copy		Delete
2	gopal	gopal123@gmail.com				
<input type="checkbox"/>		Edit		Copy		Delete
3	maulik	maulik7896@gmail.com				

EVALUATION:

Problem Analysis & Solution (3)	Understanding Level (3)	Timely Completion (2)	Mock (2)	Total (10)

EXPERIMENT NO: 4

DATE: / /

TITLE: Introduction to Servlet .

OBJECTIVES: On completion of this experiment student will able to...

- know the concept of Servlet class.
- Create simple programs using HTTP methods,ServletContext and ServletConfig interface ,Request Dispatcher interface.

THEORY: 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.

Servlet API The javax.servlet and javax.servlet.http packages represent interfaces and classes for servlet api.The javax.servlet package contains many interfaces and classes that are used by the servlet or web container. These are not specific to any protocol. The javax.servlet.http package contains interfaces and classes that are responsible for http requests only.Let's see what are the interfaces of javax.servlet package. Interfaces in javax.servlet.http package There are many interfaces in javax.servlet.http package. They are as follows:

1. HttpServletRequest
2. HttpServletResponse
3. HttpSession
4. HttpSessionListener
5. HttpSessionAttributeListener
6. HttpSessionBindingListener
7. HttpSessionActivationListener
8. HttpSessionContext (deprecated now)

Task-1 (print msg in the browser)

```

package test;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class DisConfig extends HttpServlet
{
    String msg="";
    PrintWriter out;
    public void init() throws ServletException
    {
        msg="hello world: my first servlet program";
    }
    public void doGet(HttpServletRequest request, HttpServletResponse response) throws
        ServletException, IOException
    {
        response.setContentType("text/html");
        out=response.getWriter();
        out.println(msg);
    }
    public void destroy()
    {
        out.close();
    }
}

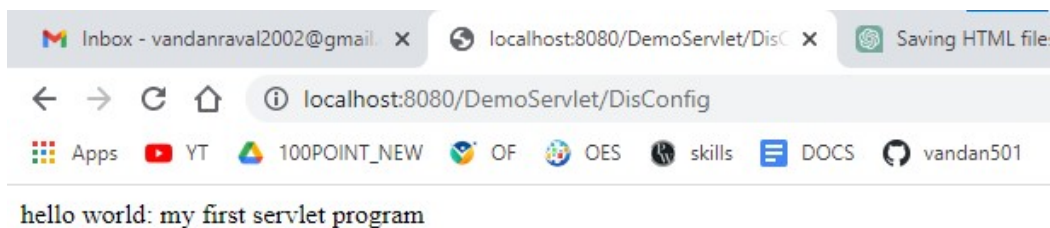
```

Web.xml

```

<web-app>
<servlet>
    <servlet-name>DisConfig</servlet-name>
    <servlet-class>test.DisConfig</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>DisConfig</servlet-name>
    <url-pattern>/DisConfig</url-pattern>
</servlet-mapping>
</web-app>

```



Methods of HttpServlet class

There are many methods in HttpServlet class. They are as follows:

1. **public void service(ServletRequest req, ServletResponse res)** dispatches the request to the protected service method by converting the request and response object into http type.
2. **protected void service(HttpServletRequest req, HttpServletResponse res)** receives the request from the service method, and dispatches the request to the doXXX() method depending on the incoming http request type.
3. **protected void doGet(HttpServletRequest req, HttpServletResponse res)** handles the GET request. It is invoked by the web container.
4. **protected void doPost(HttpServletRequest req, HttpServletResponse res)** handles the POST request. It is invoked by the web container.
5. **protected void doHead(HttpServletRequest req, HttpServletResponse res)** handles the HEAD request. It is invoked by the web container.
6. **protected void doOptions(HttpServletRequest req, HttpServletResponse res)** handles the OPTIONS request. It is invoked by the web container.
7. **protected void doPut(HttpServletRequest req, HttpServletResponse res)** handles the PUT request. It is invoked by the web container.
8. **protected void doTrace(HttpServletRequest req, HttpServletResponse res)** handles the TRACE request. It is invoked by the web container.
9. **protected void delete(HttpServletRequest req, HttpServletResponse res)** handles the DELETE request. It is invoked by the web container.
10. **protected long getLastModified(HttpServletRequest req)** returns the time when HttpServletRequest was last modified since midnight January 1, 1970 GMT.

Life Cycle of a Servlet (Servlet Life Cycle)

The web container maintains the life cycle of a servlet instance. Let's see the life cycle of the servlet:

1. Servlet class is loaded.
2. Servlet instance is created.
3. init method is invoked.
4. service method is invoked
5. destroy method is invoked.

Task-2 (using doGet() method display email using html_file)**Index.html**

```

<html>
  <head>
    <title> DoGetDemo </title>
  </head>
  <body>
    <form action="DisConfig">
      Enter Email:<input type="text" name="email">
      <p><input type="submit"></p>
    </form>
  </body>
</html>

```

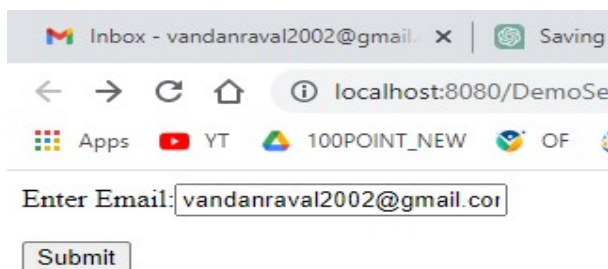
DisConfig.java

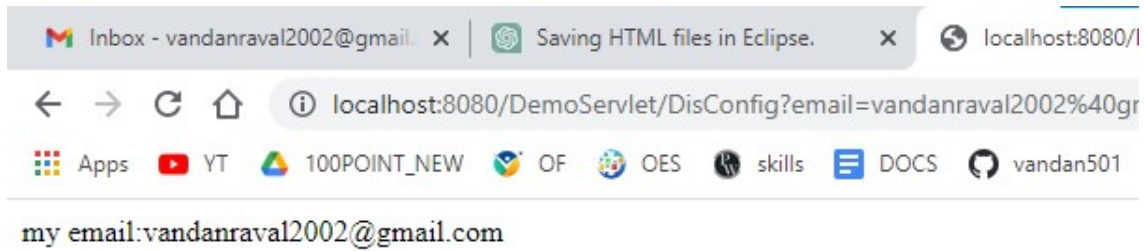
```
package test;
```

```

import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class DisConfig extends HttpServlet
{
    PrintWriter out;
    public void init(ServletConfig config)throws ServletException
    {
    }
    public void doGet(HttpServletRequest request,HttpServletResponse response)throws
        ServletException,IOException
    {
        String email=request.getParameter("email");
        response.setContentType("text/html");
        out =response.getWriter();
        out.println("my email:"+email);
    }
    public void destroy()
    {
        out.close();
    }
}

```





Task-3 (using doPost() method display maximum number from 2 numbers)

max.html

```
<!DOCTYPE html>
<html>
  <head>
    <title> Maximum number </title>
  </head>
  <body>
    <form action="DisConfig" method="POST" >
      <p>Enter No-1:<input type="text" name="no1"></p>
      <p>Enter No-2:<input type="text" name="no2"></p>
      <p><input type="submit"></p>
    </form>
  </body>
</html>
```

DisConfig.java

package test;

import java.io.*;

import javax.servlet.*;

import javax.servlet.http.*;

public class DisConfig **extends** HttpServlet
{

public void doPost(HttpServletRequest request, HttpServletResponse response)**throws**
 ServletException,IOException

```
{ int n1=0,n2=0;
  response.setContentType("text/html");
  PrintWriter out=response.getWriter();
  n1=Integer.parseInt(request.getParameter("no1"));
  n2=Integer.parseInt(request.getParameter("no2"));
  if(n1>n2)
    out.println("n1="+n1+"is max number");
  else if(n2>n1)
    out.println("n2="+n2+"is max number");
  else if(n1==n2)
    out.println("n1= "+n1+"and n2= "+n2+"are equal numbers");
```

```

    }
}

```

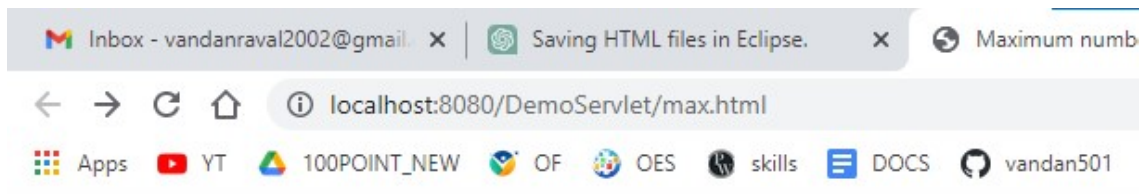
Web.xml

```

<web-app>
  <servlet>
    <servlet-name>DisConfig</servlet-name>
    <servlet-class>test.DisConfig</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>DisConfig</servlet-name>
    <url-pattern>/DisConfig</url-pattern>
  </servlet-mapping>
</web-app>

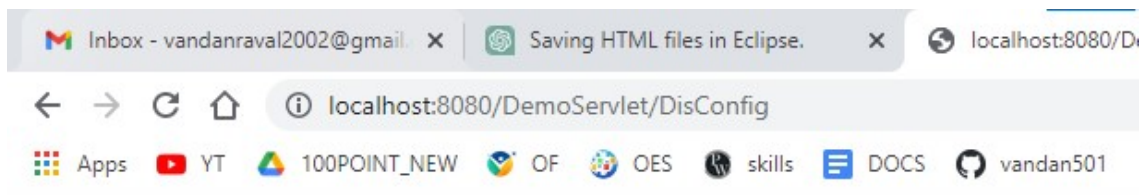
```

Output:



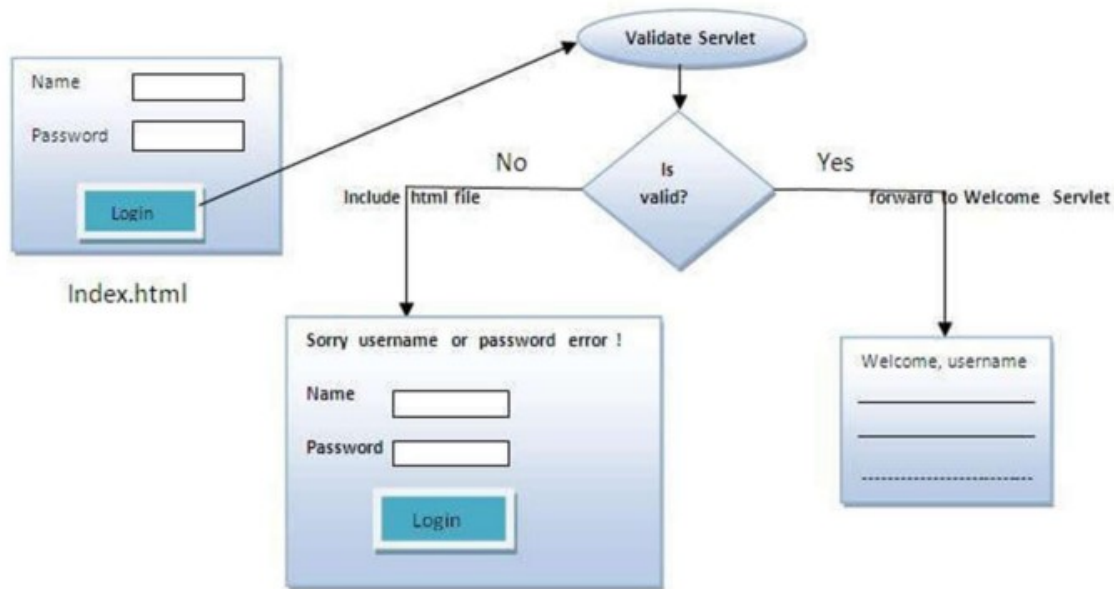
Enter No-1:

Enter No-2:



n2=28is max number

RequestDispatcher in Servlet The RequestDispatcher interface provides the facility of dispatching the request to another resource it may be html, servlet or jsp. This interface can also be used to include the content of another resource also. It is one of the way of servlet collaboration. There are two methods defined in the RequestDispatcher interface



Task 4: Create a login form using RequestDispatcher in Servlet

Index.html

```

<!DOCTYPE html>
<html>
<head>
  <title>Login Page</title>

<style>
  body {
    background-color: #f6f6f6;
    font-family: Arial, sans-serif;
    font-size: 16px;
    line-height: 1.5;
  }
  .container {
    max-width: 500px;
    margin: 0 auto;
    padding: 50px;
    background-color: #fff;
  }

```

```

    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
  }
  h1 {
    margin-top: 0;
    font-size: 32px;
    text-align: center;
  }
  form {
    display: flex;
    flex-direction: column;
  }
  label {
    margin-top: 20px;
  }
  input {
    padding: 10px;
    border-radius: 5px;
    border: none;
    box-shadow: 0 0 5px rgba(0, 0, 0, 0.1);
  }
  button {
    margin-top: 20px;
    padding: 10px;
    border-radius: 5px;
    border: none;
    background-color: #4caf50;
    color: #fff;
    font-weight: bold;
    text-transform: uppercase;
    cursor: pointer;
    transition: background-color 0.3s ease;
  }
  button:hover {
    background-color: white;
    color: green;
  }
</style>

```

```

</head>
<body>
  <div class="container">
    <h1>Login Page</h1>
    <form method="post" action="loginDone">
      <label for="username">Username:</label>
      <input type="text" id="username" name="login">
      <label for="password">Password:</label>
      <input type="password" id="password" name="pwd">
      <button type="submit">Login</button>
    </form>
  </div>
</body>
</html>

```

```

    </form>
</div>
</body>
</html>

```

loginDone.java

```

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

public class loginDone extends HttpServlet
{
    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
        response.setContentType("text/html");
        PrintWriter out=response.getWriter();
        RequestDispatcher rd;
        String login=request.getParameter("login");
        String pwd=request.getParameter("pwd");
        if(login != null && login.equals("admin") && pwd.equals("admin123"))
        {
            rd=request.getRequestDispatcher("homePage");
            rd.forward(request, response);
        }
        else
        {
            out.println("<p><h1>Incorrect Login Id/Password</h1></p>");
            rd=request.getRequestDispatcher("/index.html");
            rd.include(request, response);
        }
    }
}

```

homepage.java

```

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

public class homePage extends HttpServlet {
    public void doPost(HttpServletRequest request,HttpServletResponse response)
        throws ServletException,IOException
    {
        response.setContentType("text/html");
        PrintWriter out=response.getWriter();
        String username=request.getParameter("login");
        out.println("<h1>"+ "Welcome "+username+"</h1>");
    }
}

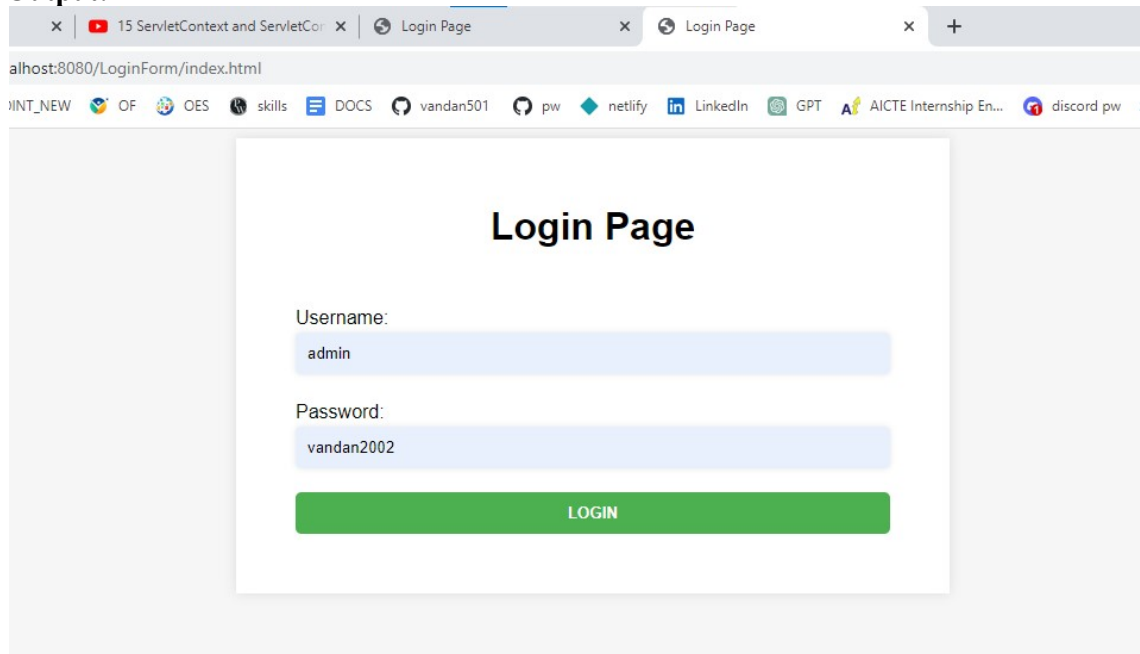
```

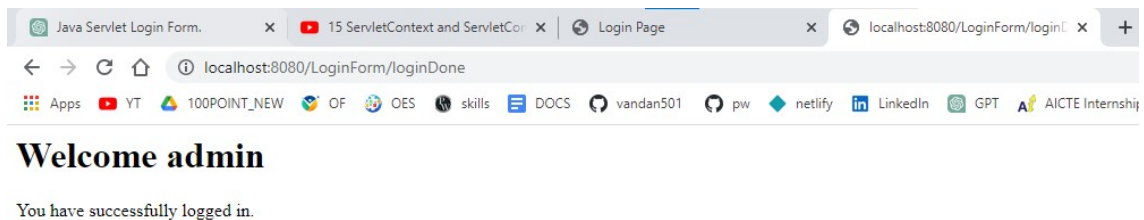
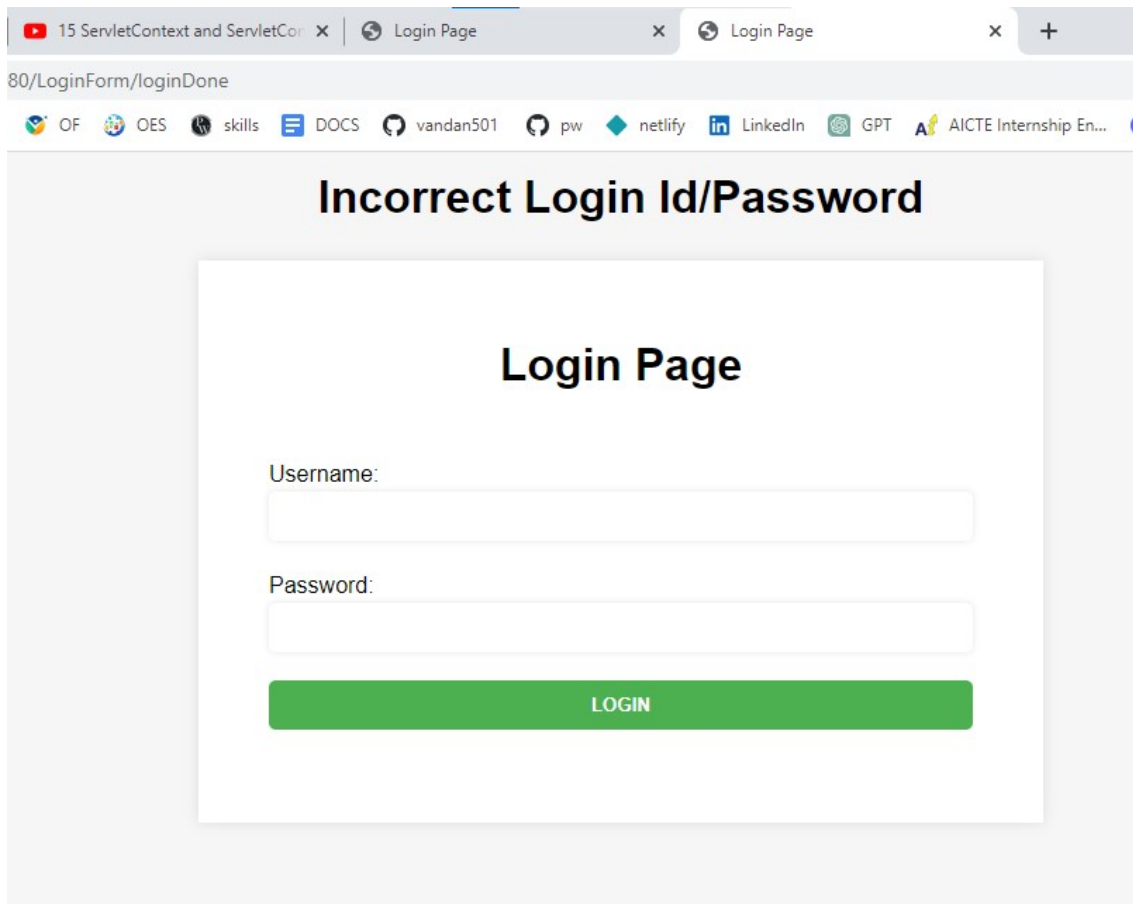


```
}
}
```

Web.xml

```
<web-app>
  <servlet>
    <servlet-name>homePage</servlet-name>
    <servlet-class>homePage</servlet-class>
  </servlet>
  <servlet>
    <servlet-name>loginDone</servlet-name>
    <servlet-class>loginDone</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>homePage</servlet-name>
    <url-pattern>/homePage</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>loginDone</servlet-name>
    <url-pattern>/loginDone</url-pattern>
  </servlet-mapping>
</web-app>
```

Output:



SendRedirect in servlet

The **sendRedirect()** method of **HttpServletResponse** interface can be used to redirect response to another resource, it may be servlet, jsp or html file. It accepts relative as well as absolute URL. It works at client side because it uses the url bar of the browser to make another request. So, it can work inside and outside the server.

Task 5. Create a login form using SendRedirect() in Servlet.

Index.html

```
<!DOCTYPE html>
<html>
  <head>
    <title>Login Page</title>

  <style>
    body {
      background-color: #f6f6f6;
      font-family: Arial, sans-serif;
      font-size: 16px;
      line-height: 1.5;
    }
    .container {
      max-width: 500px;
      margin: 0 auto;
      padding: 50px;
      background-color: #fff;
      box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    }
    h1 {
      margin-top: 0;
      font-size: 32px;
      text-align: center;
    }
    form {
      display: flex;
      flex-direction: column;
    }
    label {
      margin-top: 20px;
    }
    input {
      padding: 10px;
      border-radius: 5px;
      border: none;
      box-shadow: 0 0 5px rgba(0, 0, 0, 0.1);
    }
```

```

}
button {
  margin-top: 20px;
  padding: 10px;
  border-radius: 5px;
  border: none;
  background-color: #4caf50;
  color: #fff;
  font-weight: bold;
  text-transform: uppercase;
  cursor: pointer;
  transition: background-color 0.3s ease;
}
button:hover {
  background-color: white;
  color: green;
}
</style>

```

```

</head>
<body>
  <div class="container">
    <h1>Login Page</h1>
    <form method="get" action="sendRedirect">
      <label for="username">Username:</label>
      <input type="text" id="username" name="login">
      <label for="text">Password:</label>
      <input type="text" id="password" name="pwd">
      <button type="submit">Login</button>
    </form>
  </div>
</body>
</html>

```

sendRedirect.java

```

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 sendRedirect extends HttpServlet {

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

```

```

        response.setContentType("text/html");
        PrintWriter out=response.getWriter();
        String login=request.getParameter("login");
        String pwd=request.getParameter("pwd");

        if(login.equals("vandan") && pwd.equals("vandan2002"))
        {
            response.sendRedirect("success.html");
        } else {
            response.sendRedirect("error.html");
        }
    }
}

```

Success.html

```

<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Welcome</title>
</head>
<body>
<h1>Welcome in the homePage</h1>
</body>
</html>

```

error.html

```

<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>OOps</title>
</head>
<body>
<h2>Please try again!!</h2>
<h3>&u>Username or password are invalid</h3>
</body>
</html>

```

web.xml

```

<web-app>
  <servlet>
    <servlet-name>homePage</servlet-name>
    <servlet-class>homePage</servlet-class>
  </servlet>
  <servlet>
    <servlet-name>loginDone</servlet-name>
    <servlet-class>loginDone</servlet-class>
  </servlet>
  <servlet>
    <servlet-name>sendRedirect</servlet-name>
    <servlet-class>sendRedirect</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>homePage</servlet-name>
    <url-pattern>/homePage</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>loginDone</servlet-name>
    <url-pattern>/loginDone</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>sendRedirect</servlet-name>
    <url-pattern>/sendRedirect</url-pattern>
  </servlet-mapping>

  <servlet>
    <servlet-name>success</servlet-name>
    <jsp-file>/success.html</jsp-file>
  </servlet>

  <servlet-mapping>
    <servlet-name>success</servlet-name>
    <url-pattern>/success</url-pattern>
  </servlet-mapping>

  <servlet>
    <servlet-name>error</servlet-name>
    <jsp-file>/error.html</jsp-file>
  </servlet>

  <servlet-mapping>
    <servlet-name>error</servlet-name>
    <url-pattern>/error</url-pattern>
  </servlet-mapping>

</web-app>

```

Output:

index.html

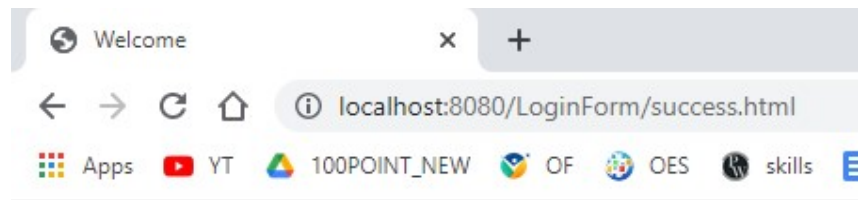
DES skills DOCS vandan501 pw netlify LinkedIn GPT AICTE Internshi

Login Page

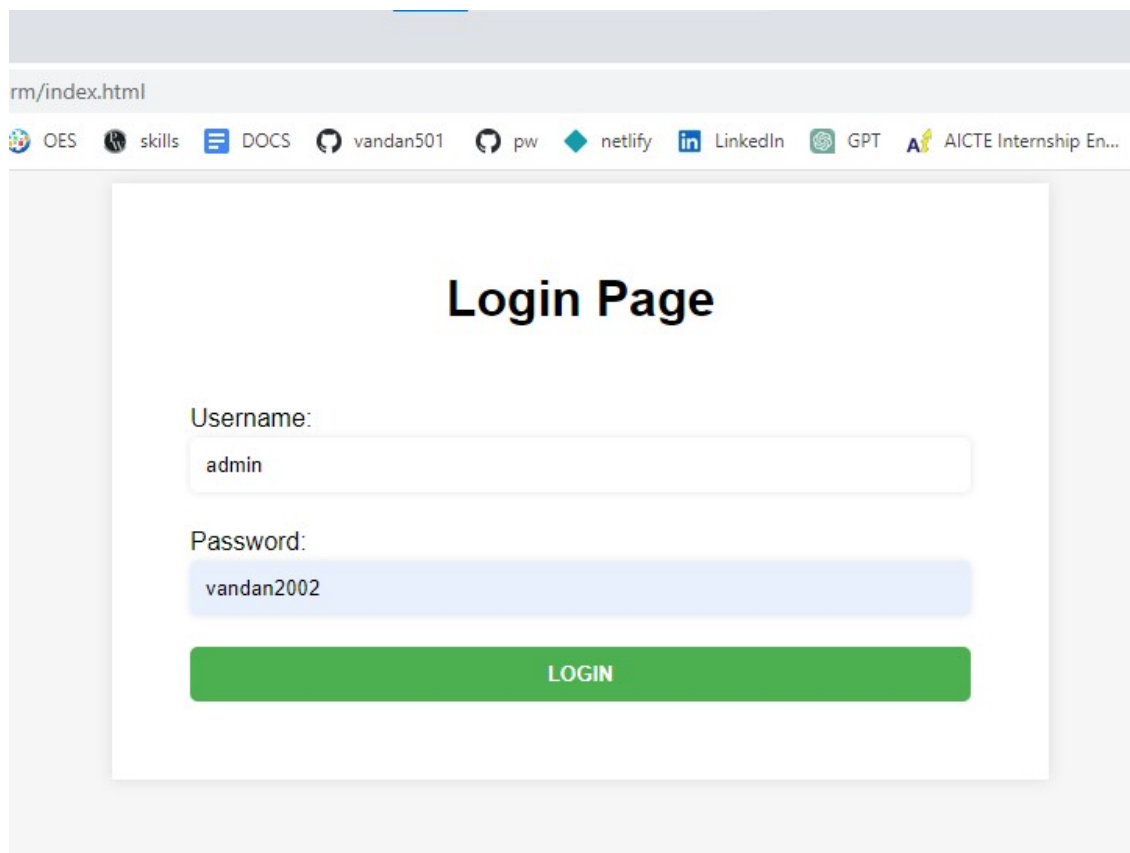
Username:
vandan

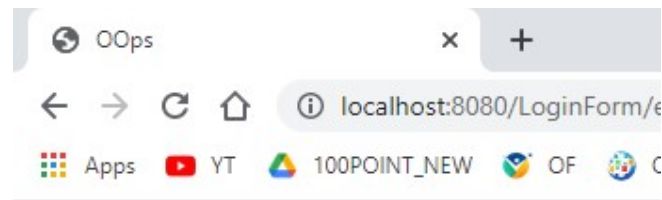
Password:
vandan2002

LOGIN



Welcome in the homePage





Please try again!!

Username or password are invalid

EVALUATION:

Problem Analysis & Solution (3)	Understanding Level (3)	Timely Completion (2)	Mock (2)	Total (10)

EXPERIMENT NO: 5

DATE: / /

TITLE: Session Tracking in Servlet and Filter .

OBJECTIVES: On completion of this experiment student will able to...

- know the concept of Session Tracking and Filter Class.
- Create simple programs using Cookies,UrlRewriting,Hidden form fields,HttpSession .

THEORY: Session simply means a particular interval of time. Session Tracking is a way to maintain state (data) of an user. It is also known as session management in servlet. Http protocol is a stateless so we need to maintain state using session tracking techniques. Each time user equests to the server, server treats the request as the new request. So we need to maintain the state of an user to recognize to particular user. HTTP is stateless that means each request is considered as the new request. It is shown in the figure given below:

Session Tracking Techniques

There are four techniques used in Session tracking:

1. Cookies
2. Hidden Form Field
3. URL Rewriting
4. HttpSession

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.

- **Task 1:Cookies**

Index.html

```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
<form method="get" action="CreateCookie">
<input type="text" name="username"> <br><br><br>
<input type="submit" value="SUBMIT">
</form>
</body>
</html>
```

CreateCookie.java

```
package cookieandsession;

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

public class CreateCookie extends HttpServlet
{
    public void doGet(HttpServletRequest req,HttpServletResponse res)throws
    IOException,ServletException
    {
        res.setContentType("text/html");
        PrintWriter out=res.getWriter();
        String uname=req.getParameter("username");
        Cookie c=new Cookie("username",uname);
        res.addCookie(c);
        out.println("Cookie Added");

    }
}
```

Web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee"
```

```

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
                    http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
version="3.0">

```

```

<servlet>
  <servlet-name>CreateCookieServlet</servlet-name>
  <servlet-class>cookieandsession.CreateCookie</servlet-class>
</servlet>

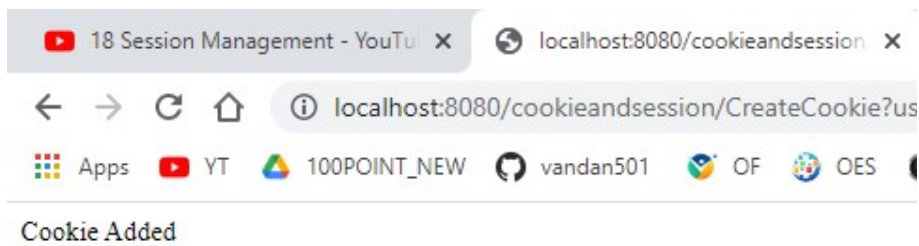
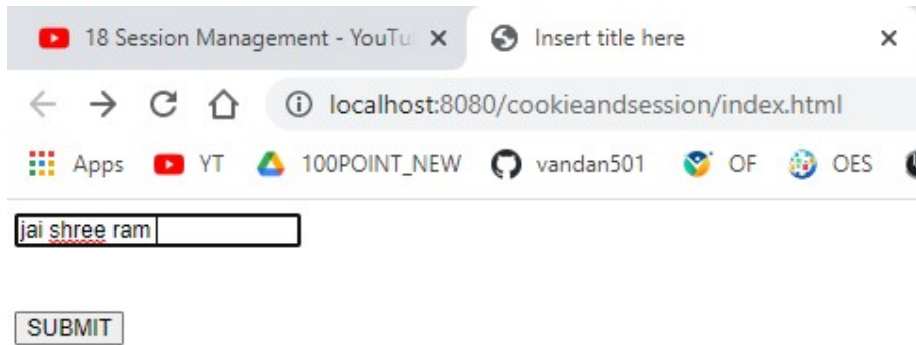
```

```

<servlet-mapping>
  <servlet-name>CreateCookieServlet</servlet-name>
  <url-pattern>/CreateCookie</url-pattern>
</servlet-mapping>

```

```
</web-app>
```



- **Task 2: hidden From Field**

Index.html

```
<html>
  <head>
    <title>login</title>
  </head>
  <body>
    <form action="Valid" method="POST">
      <p>Login ID:<input type="text" name="login"></p>
      <p>Password:<input type="text" name="pwd"></p>
      <p><input type="hidden" name="session_id" value="501"></p>
      <p><input type="submit" value="Sign In"></p>
    </form>
  </body>
</html>
```

Valid.java

```
package cookieandsession;

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

import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class Valid extends HttpServlet
{
    public void doPost(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException
    {
        response.setContentType("text/html");
        PrintWriter out=response.getWriter();
        RequestDispatcher rd;
        String login=request.getParameter("login");
        String pwd=request.getParameter("pwd");
        String session=request.getParameter("session_id");
        if(login.equals("vandan") && pwd.equals("vandan501"))
        {
            rd=request.getRequestDispatcher("welcome");
            rd.forward(request, response);
        }
        else
        {

```

```

        out.println("<p><h1>Incorrect LoginId/Password </h1></p>");
        rd=request.getRequestDispatcher("index.html");
        rd.include(request, response);
    }
}

```

Welcome.java

```

package cookieandsession;

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

public class welcome extends HttpServlet
{
    public void doPost(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
        response.setContentType("text/html");
        PrintWriter out=response.getWriter();
        String session=request.getParameter("session_id");
        String username=request.getParameter("login");
        out.println("<h1>"+ "id:"+session+"</h1>");
        out.println("<h3>"+ "Welcome "+username+"</h3>");
    }
}

```

Web.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://xmlns.jcp.org/xml/ns/javaee"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
        http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
    id="WebApp_ID" version="4.0">

    <display-name>cookieandsession</display-name>

    <servlet>
        <servlet-name>CreateCookie</servlet-name>
        <servlet-class>cookieandsession.CreateCookie</servlet-class>
    </servlet>

    <servlet>
        <servlet-name>Valid</servlet-name>
        <servlet-class>cookieandsession.Valid</servlet-class>
    </servlet>

    <servlet>
        <servlet-name>welcome</servlet-name>
        <servlet-class>cookieandsession.welcome</servlet-class>

```

```

</servlet>

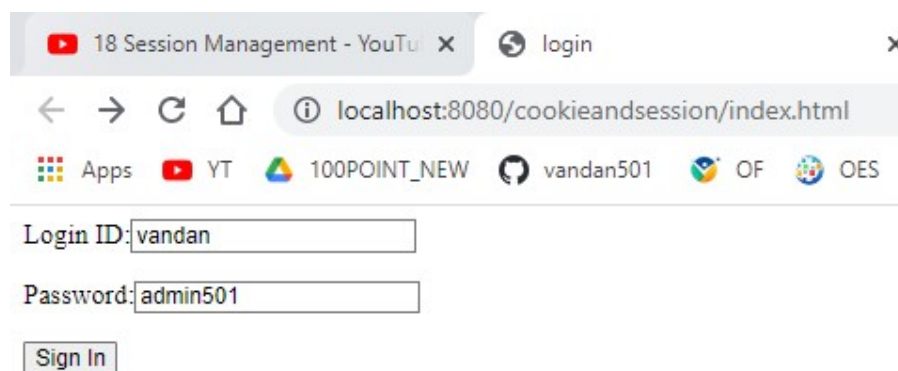
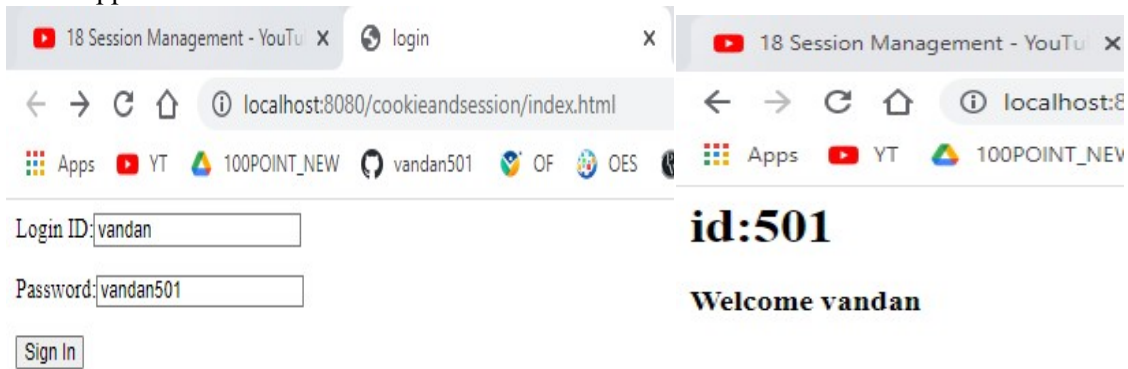
<servlet-mapping>
  <servlet-name>CreateCookie</servlet-name>
  <url-pattern>/CreateCookie</url-pattern>
</servlet-mapping>

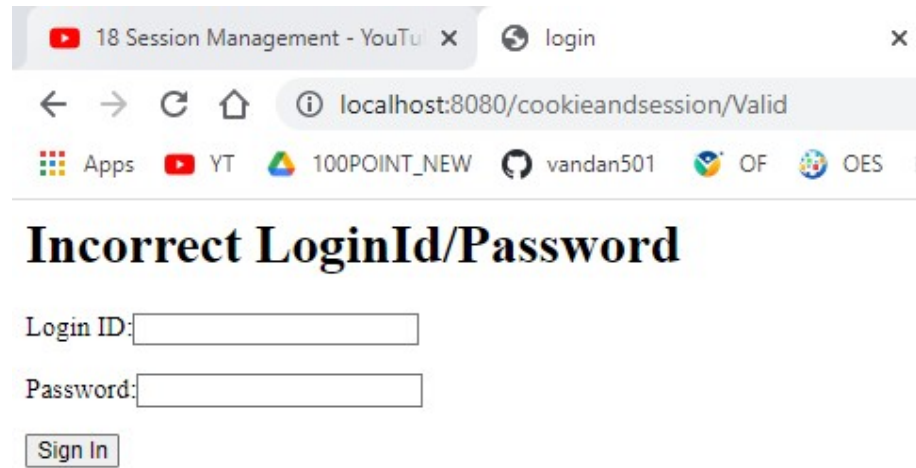
<servlet-mapping>
  <servlet-name>Valid</servlet-name>
  <url-pattern>/Valid</url-pattern>
</servlet-mapping>

<servlet-mapping>
  <servlet-name>welcome</servlet-name>
  <url-pattern>/welcome</url-pattern>
</servlet-mapping>

```

```
</web-app>
```





- URL REWRITING

url1.java

```
package cookieandsession;
```

```
import javax.servlet.*;
```

```
import javax.servlet.http.*;
```

```
import java.io.*;
```

```
public class url1 extends HttpServlet
```

```
{
```

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

```
    {
```

```
        String url;
```

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

```
        PrintWriter out=response.getWriter();
```

```
        //for URL rewriting
```

```
        url= "http://localhost:8080/cookieandsession/url2?s_id1=054&s_id2=055";
```

```
        out.println("<h1><a href="+url+">next page</a></h1>");
```

```
    }
```

```
}
```

url2.java

```
package cookieandsession;
```

```
import javax.servlet.*;
```

```
import javax.servlet.http.*;
```

```
import java.io.*;
```

```
public class url2 extends HttpServlet
```

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


throws ServletException,IOException

```

{ response.setContentType("text/html");
  PrintWriter out=response.getWriter();
  String session1=request.getParameter("s_id1");
  String session2=request.getParameter("s_id2");
  out.println("<h3>"+ "id:"+session1+"</h3>");
  out.println("<h3>"+ "id:"+session2+"</h3>");
}
}

```

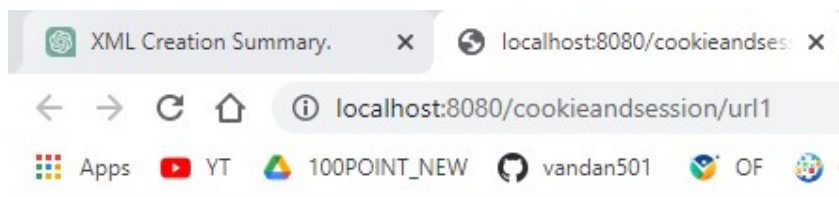
Web.xml

```

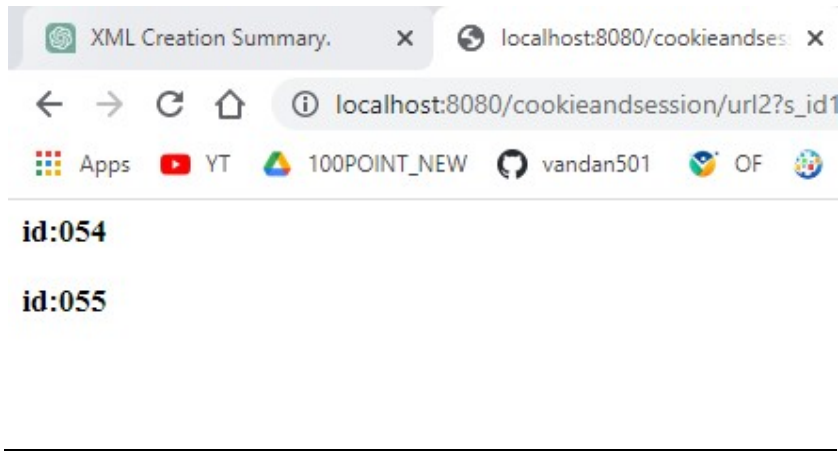
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
    http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
  version="4.0">

  <servlet>
    <servlet-name>url1</servlet-name>
    <servlet-class>cookieandsession.url1</servlet-class>
  </servlet>
  <servlet>
    <servlet-name>url2</servlet-name>
    <servlet-class>cookieandsession.url2</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>url1</servlet-name>
    <url-pattern>/url1</url-pattern>
  </servlet-mapping>
  <servlet-mapping>
    <servlet-name>url2</servlet-name>
    <url-pattern>/url2</url-pattern>
  </servlet-mapping>
</web-app>

```



[next page](#)



- **HttpSession**

Index.html

```
<html>
<head>
  <title>HttpSession</title>
</head>
<body>
  <form action="Home_session" method="post">
    <p>Login ID:<input type="text" name="login"></p>
    <p><input type="submit" value="Sign In"></p>
  </form>
</body>
</html>
```

Home_session.java

```
package cookieandsession;

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

public class Home_session extends HttpServlet {

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

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

```

PrintWriter out = response.getWriter();
RequestDispatcher rd;
String login = request.getParameter("login");

if (login.equals("shreeram")) {
    HttpSession hs = request.getSession();
    hs.setAttribute("s_id", login); // set HttpSession
    out.println("<h1>Session Created</h1>");
    out.print("<h2><a href='session2'>Homepage</a></h2>");
} else {
    out.println("<p><h1>Incorrect Login Id/Password</h1></p>");
    rd = request.getRequestDispatcher("/index.html");
    rd.include(request, response);
}
}
}

```

Session2.java

```

package cookieandsession;

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

public class session2 extends HttpServlet
{
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
        response.setContentType("text/html");
        PrintWriter out=response.getWriter();
        HttpSession hs=request.getSession(false);
        String n=(String)hs.getAttribute("s_id");
        out.print("Hello "+n);
        out.print("<p><a href='session3'>Logout</a></p>");
    }
}

```

Session3.java

```

package cookieandsession;

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

public class session3 extends HttpServlet

```

```

{   public void doGet(HttpServletRequest request, HttpServletResponse response)
                                   throws ServletException,IOException
    {
        response.setContentType("text/html");
        PrintWriter out=response.getWriter();
        HttpSession hs=request.getSession(false);
        hs.invalidate();// Session Invalidated
        try
        {
            String n=(String)hs.getAttribute("s_id");
        }
        catch(Exception ne)
        {
            out.println("Session Invalidated");
        }
        out.println("<form action='index.html'>");
        out.println("<p><input type='submit' value='Login'></p></form>");
    }
}

```

Web.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
    http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
  version="4.0">

  <servlet>
    <servlet-name>Home_session</servlet-name>
    <servlet-class>cookieandsession.Home_session</servlet-class>
  </servlet>

  <servlet>
    <servlet-name>session2</servlet-name>
    <servlet-class>cookieandsession.session2</servlet-class>
  </servlet>

  <servlet>
    <servlet-name>session3</servlet-name>
    <servlet-class>cookieandsession.session3</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>Home_session</servlet-name>
    <url-pattern>/Home_session</url-pattern>
  </servlet-mapping>

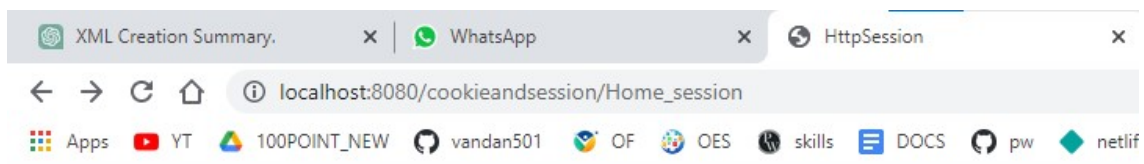
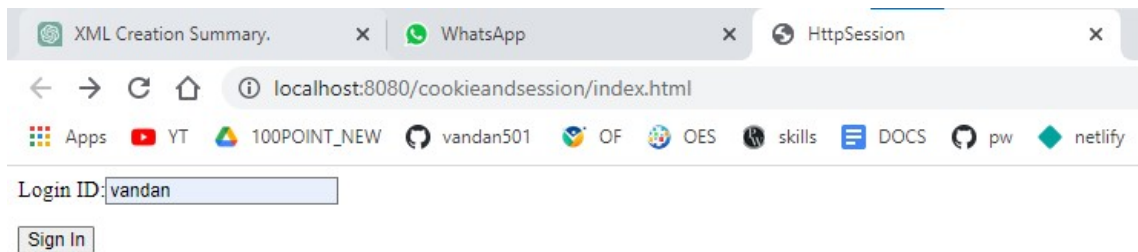
  <servlet-mapping>

```

```
<servlet-name>session2</servlet-name>
<url-pattern>/session2</url-pattern>
</servlet-mapping>
```

```
<servlet-mapping>
  <servlet-name>session3</servlet-name>
  <url-pattern>/session3</url-pattern>
</servlet-mapping>
```

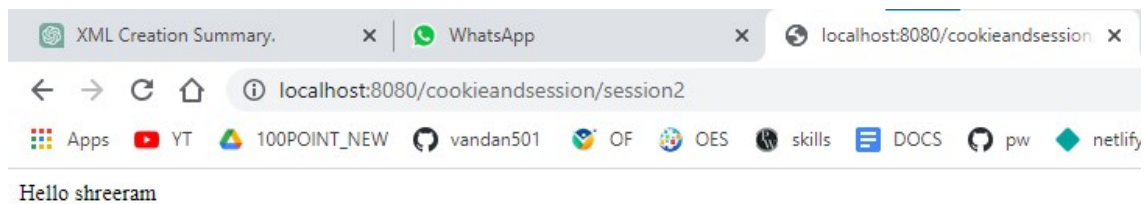
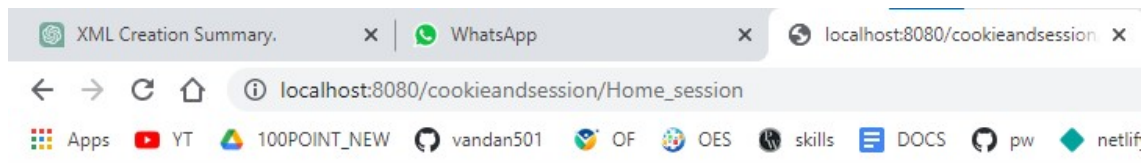
```
</web-app>
```



Incorrect Login Id/Password

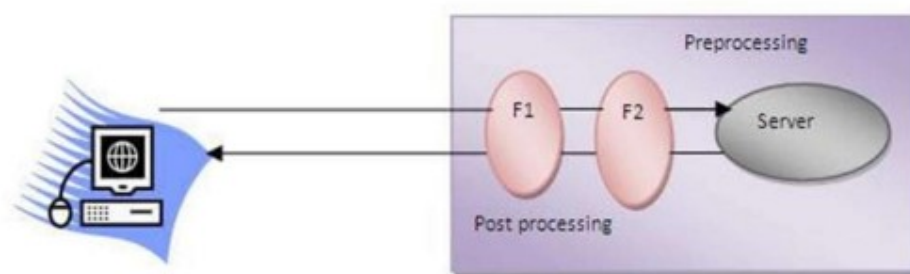
Login ID:

Login ID:



Servlet Filter

A filter is an object that is invoked at the preprocessing and postprocessing of a request. It is mainly used to perform filtering tasks such as conversion, logging, compression, encryption and decryption, input validation etc. The servlet filter is pluggable, i.e. its entry is defined in the web.xml file, if we remove the entry of filter from the web.xml file, filter will be removed automatically and we don't need to change the servlet. So maintenance cost will be less.



Usage of Filter

- recording all incoming requests
- logs the IP addresses of the computers from which the requests originate
- conversion

- data compression
- encryption and decryption
- input validation etc.

Advantage of Filter

1. Filter is pluggable.
2. One filter don't have dependency onto another resource.
3. Less Maintenance

Inxex.html

```
<html>
  <head>
    <title>Filter</title>
  </head>
  <body>
    <h1> <a href="FilteredServlet">click here</a></h1>
  </body>
</html>
```

FilteredServlet.java

```
package cookieandsession;

import java.io.IOException;

import java.io.PrintWriter;

import javax.servlet.*;

import javax.servlet.http.*;

public class FilteredServlet extends HttpServlet

{

public void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException

{

    response.setContentType("text/html");

    PrintWriter out = response.getWriter();
```

```

        out.println("<br>welcome to servlet<br>");
    }
}

```

Filter1.java

```

package cookieandsession;

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

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;

public class Filter1 implements Filter {

    @Override
    public void init(FilterConfig arg0) throws ServletException {
        // overridden init() method
    }

    @Override
    public void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain)
        throws IOException, ServletException {
        PrintWriter out = resp.getWriter();
        out.print("filter is invoked before");// exe. with request
        chain.doFilter(req, resp);// send request to next resource
        out.print("filter is invoked after");// exe. with response
    }

    @Override
    public void destroy() {
        // overridden destroy() method
    }
}

```


Web.xml

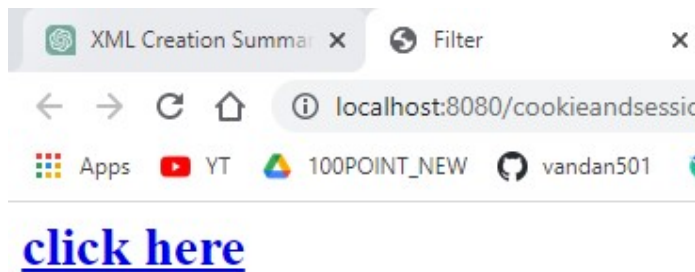
```

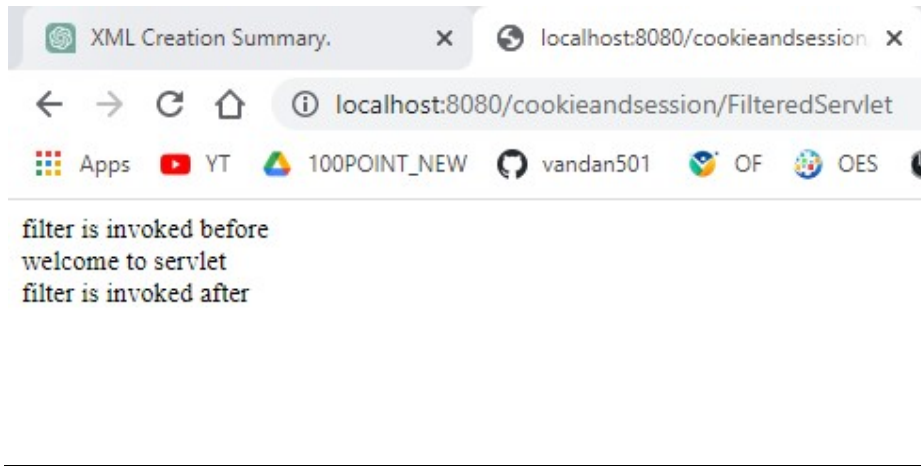
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
    http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
  version="4.0">
  <servlet>
    <servlet-name>FilteredServlet</servlet-name>
    <servlet-class>cookieandsession.FilteredServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>FilteredServlet</servlet-name>
    <url-pattern>/FilteredServlet</url-pattern>
  </servlet-mapping>

  <filter>
    <filter-name>f1</filter-name>
    <filter-class>cookieandsession.Filter1</filter-class>
  </filter>
  <filter-mapping>
    <filter-name>f1</filter-name>
    <servlet-name>FilteredServlet</servlet-name>
  </filter-mapping>

</web-app>

```

OUTPUT:



EVALUATION:

Problem Analysis & Solution (3)	Understanding Level (3)	Timely Completion (2)	Mock (2)	Total (10)

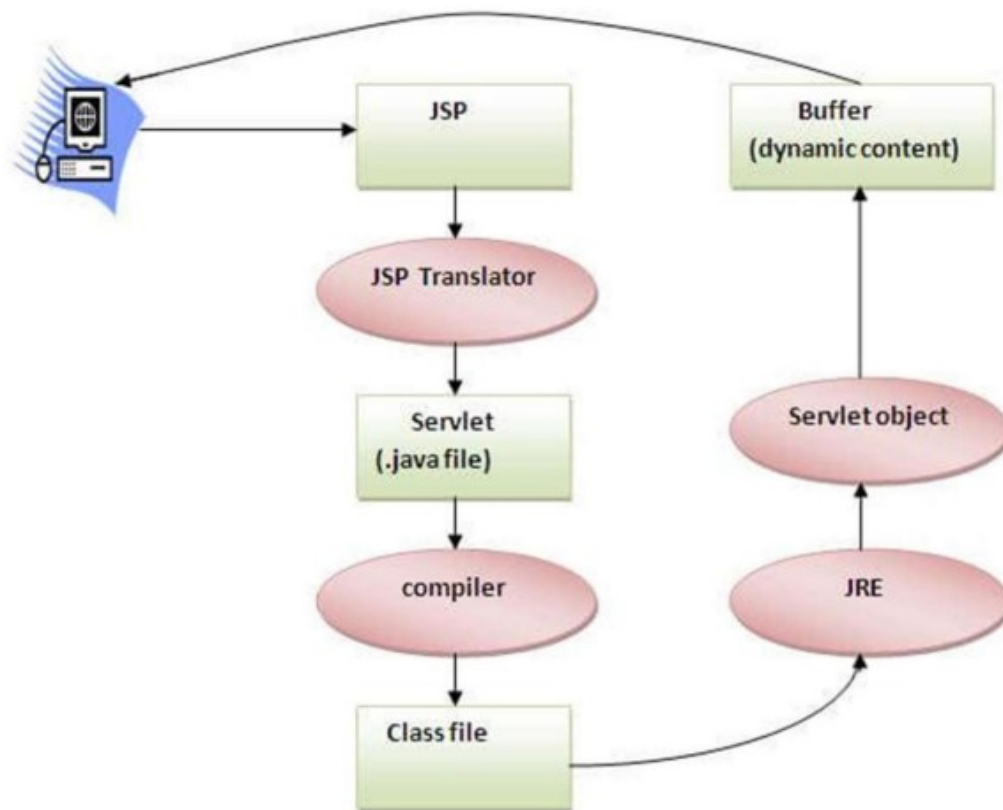
EXPERIMENT NO: 6

DATE: / /

TITLE: Introduction to JSP**OBJECTIVES:** On completion of this experiment student will able to...

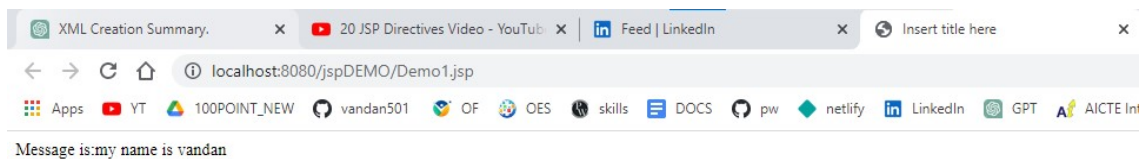
➤ know the concept of JSP.

➤ Create simple programs using JSP Directives, JSP Action, JSP Implicit Objects JSP Form Processing, JSP Session and Cookies Handling, JSP Session Tracking

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.**The Lifecycle of a JSP Page**

- **Basic Program to print name on browser**

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<%!
String msg="my name is vandan";
%>
<body>
Message is:<%out.println(msg); %>
</body>
</html>
```



- **Request**

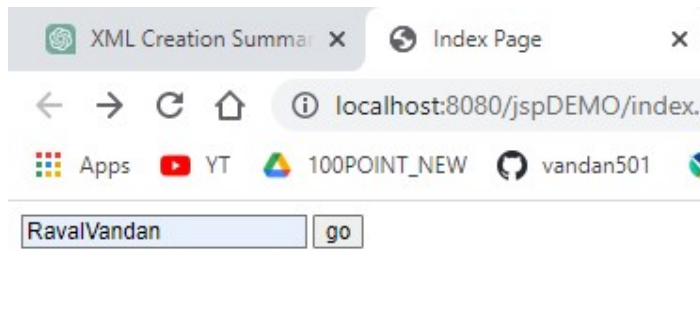
Index.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Index Page</title>
</head>
<body>
    <form action="welcome.jsp" method="get">
        <input type="text" name="uname">
        <input type="submit" value="go"><br/>
    </form>
</body>
</html>
```

Welcome.jsp

```
<%@ page language="java" %>
<html>
<head>
    <title>Welcome Page</title>
</head>
```

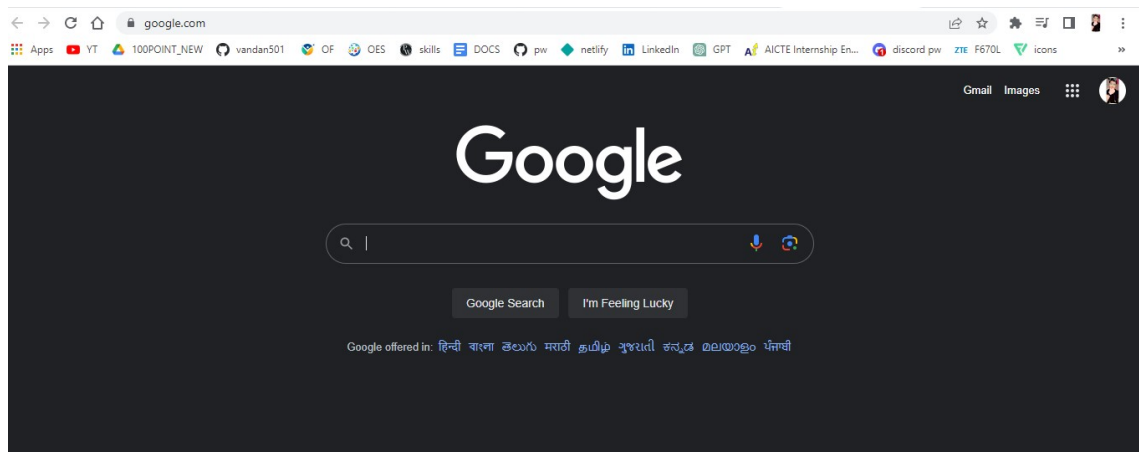
```
<body>
    <%
        String name=request.getParameter("uname");
        out.print("Welcome "+name);
    %>
</body>
</html>
```



- **Response**

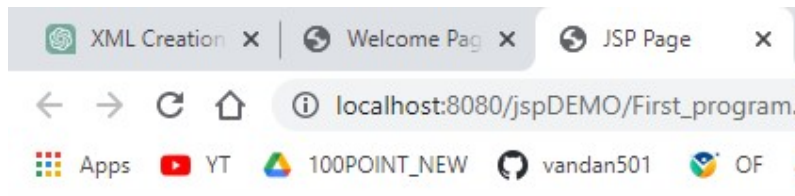
res.jsp

```
<%
response.sendRedirect("http://www.google.com");
%>
```



- **Current Date and Time**

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@page import="java.util.*" %>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>JSP Page</title>
  </head>
  <body>
    <p>Current Date and Time: <%= new java.util.Date() %></p>
    <%= out.println("This is my first jsp page");%>
  </body>
</html>
```



Current Date and Time: Sat May 13 14:49:13 IST 2023

This is my first jsp page

- **FORM PROCESSING USING JSP**

- Index.html**

```
<!DOCTYPE html>
<html>
<head>
  <title>JSP Form Processing Example</title>
</head>
<body>
  <h2>JSP Form Processing Example</h2>
  <form action="process.jsp" method="post">
    <label for="name">Name:</label>
    <input type="text" id="name" name="name" required><br><br>
    <label for="email">Email:</label>
    <input type="email" id="email" name="email" required><br><br>
    <label for="message">Message:</label>
    <textarea id="message" name="message" rows="5" cols="40"
required></textarea><br><br>
    <input type="submit" value="Submit">
```

```

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

```

Process.jsp

```

<%@ page language="java" %>
<%@ page import="java.io.*, java.util.*" %>
<%
    String name = request.getParameter("name");
    String email = request.getParameter("email");
    String message = request.getParameter("message");

    String confirmationMessage = "Thank you for your submission, " + name + "!";

    request.setAttribute("confirmationMessage", confirmationMessage);

    request.getRequestDispatcher("confirmation.jsp").forward(request, response);
%>

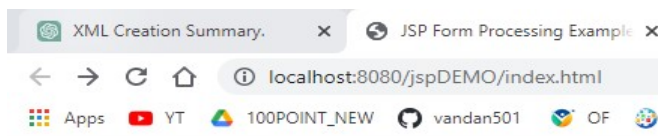
```

Confirmation.jsp

```

<!DOCTYPE html>
<html>
<head>
    <title>Confirmation Page</title>
</head>
<body>
    <h2>Confirmation Page</h2>
    <p><%= request.getAttribute("confirmationMessage") %></p>
</body>
</html>

```



JSP Form Processing Example

Name:

Email:

Message:

- **ACTION TAG**

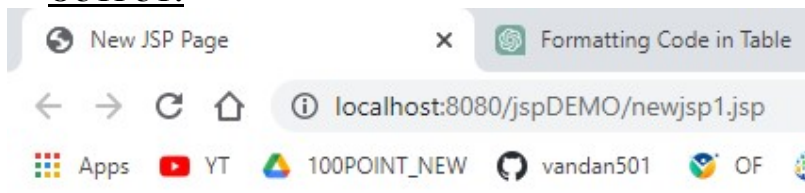
newjsp1.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Action tag</title>
</head>
<body>
<jsp:forward page="newjsp2.jsp"/>
</body>
</html>
```

Newjsp2.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<%@ page import="java.util.Date" %>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>New JSP Page</title>
</head>
<body>
<h1>
<%= new Date() %>
</h1>
</body>
</html>
```

OUTPUT:



Sat May 13 18:56:10 IST 2023

- **Implicit Object Example**

Index.html

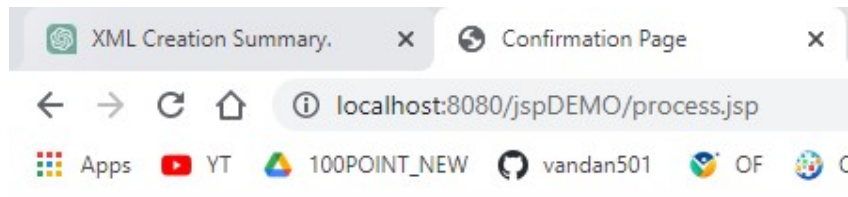
```
<!DOCTYPE html>
<html>
<head>
  <title>Implicit Object Example</title>
</head>
<body>
  <h2>Implicit Object Example</h2>
  <form action="implicitobjectDemo.jsp" method="post">
    <label for="name">Name:</label>
    <input type="text" id="name" name="name" required><br><br>
    <input type="submit" value="Submit">
  </form>
</body>
</html>
```

implicitobjectDemo.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
  pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
<%
out.println("Hello World");
out.println(request.getParameter("name"));
%>
</body>
</html>
```

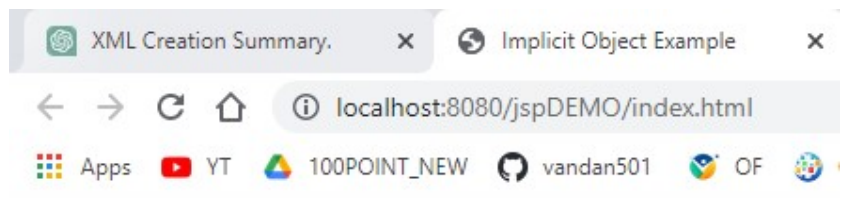
Confirmation.jsp

```
<!DOCTYPE html>
<html>
<head>
  <title>Confirmation Page</title>
</head>
<body>
  <h2>Confirmation Page</h2>
  <p><%= request.getAttribute("confirmationMessage") %></p>
</body>
</html>
```



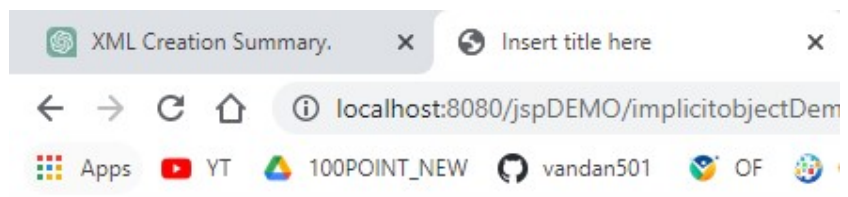
Confirmation Page

Thank you for your submission, Raval Vandan Dharmendrabhai!



Implicit Object Example

Name:



Hello World SAL education

- **OTHER IMPLICIT OBJECTS**

Index.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Implicit Object Example</title>
</head>
<body>
    <h2>Implicit Object Example</h2>
    <form action="implicitObjectDemo.jsp" method="post">
        <label for="username">Name:</label>
        <input type="text" id="name" name="username" required><br><br>
        <input type="submit" value="Submit">
    </form>
</body>
</html>
```

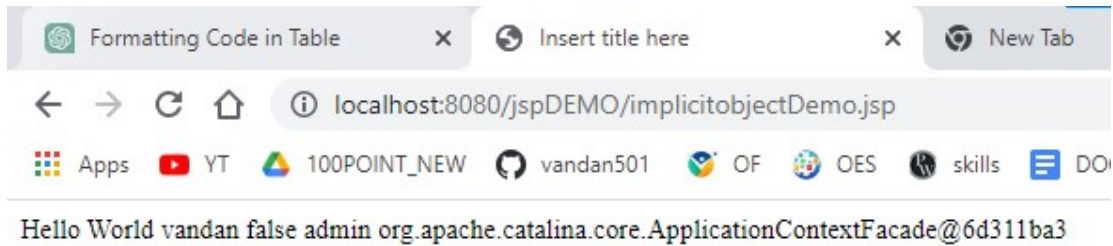
inmpliciObjectDemo.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8" %>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Insert title here</title>
</head>
<body>
    <%-- JSP scriptlet --%>
    <%
        out.println("Hello World");
        out.println(request.getParameter("username"));
        Cookie c = new Cookie("username","vandan");
        response.addCookie(c);

        out.println(session.isNew());

        application.setAttribute("username","admin");
        out.println(application.getAttribute("username"));

        out.println(config.getServletContext());
    %>
</body>
</html>
```



- **COOKIE**

Index.html

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Login Page</title>
</head>
<body>
  <h1>Login Page</h1>
  <form method="get" action="createcookie.jsp">
    <label for="username">Username:</label>
    <input type="text" id="username" name="username" required>
    <br>
    <label for="password">Password:</label>
    <input type="password" id="password" name="password" required>
    <br>
    <button type="submit">Login</button>
  </form>
</body>
</html>
```

Createcookie.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
  <meta charset="UTF-8">
  <title>Insert title here</title>
</head>
<body>
  <%
String username=request.getParameter("username");
```

```
String password=request.getParameter("password");
Cookie c=new Cookie("user",username);
response.addCookie(c);

%>
<a href="retrivecookie.jsp">To retrive Cookie Click here</a>
</body>
</html>
```

Retrivecookie.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8" %>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Cookie Example</title>
</head>
<body>
    <%-- Print the names and values of all cookies --%>
    <%
        Cookie[] cookies = request.getCookies();
        if (cookies != null) {
            for (Cookie cookie : cookies) {
                out.println("Name: " + cookie.getName() + "<br>");
                out.println("Value: " + cookie.getValue() + "<br>");
            }
        }
    %>

    <%-- Link to a page that deletes all cookies --%>
    <a href="deletcookie.jsp">Click here to delete all cookies</a>
</body>
</html>
```

Deletcookie.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Insert title here</title>
</head>
<body>
<%
```

```
Cookie c=new Cookie("user","");
c.setMaxAge(0);
response.addCookie(c);
```

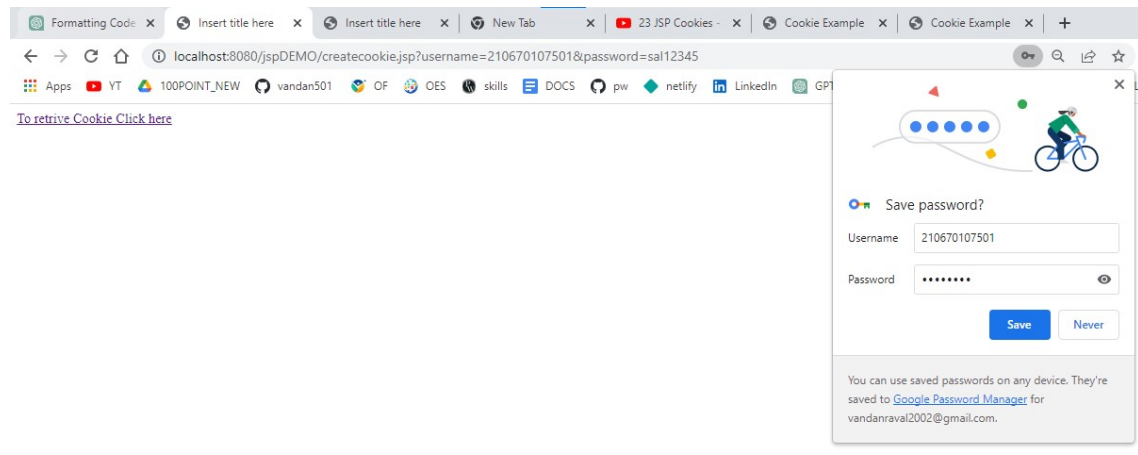
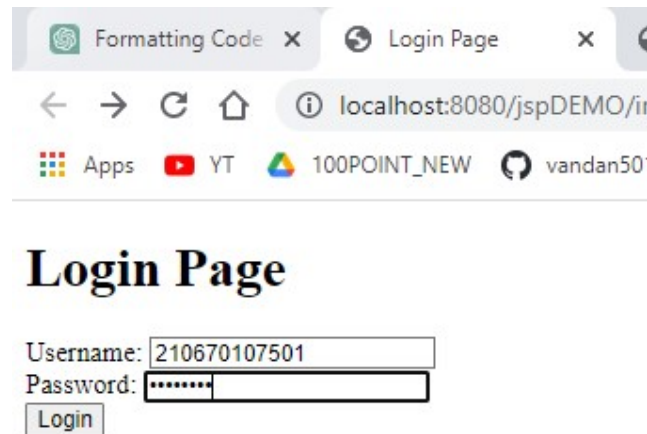
```
%>
```

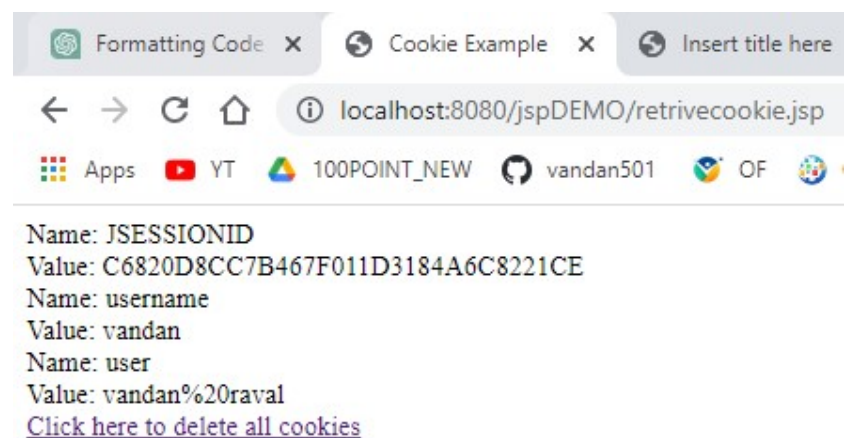
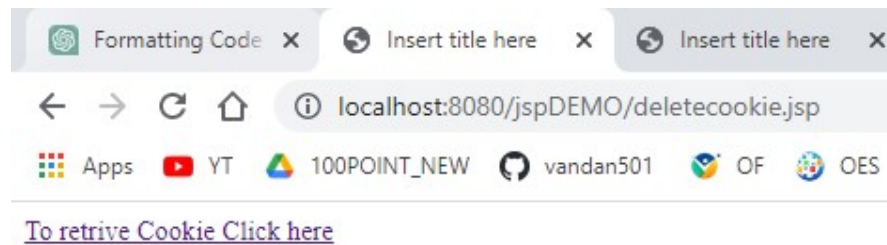
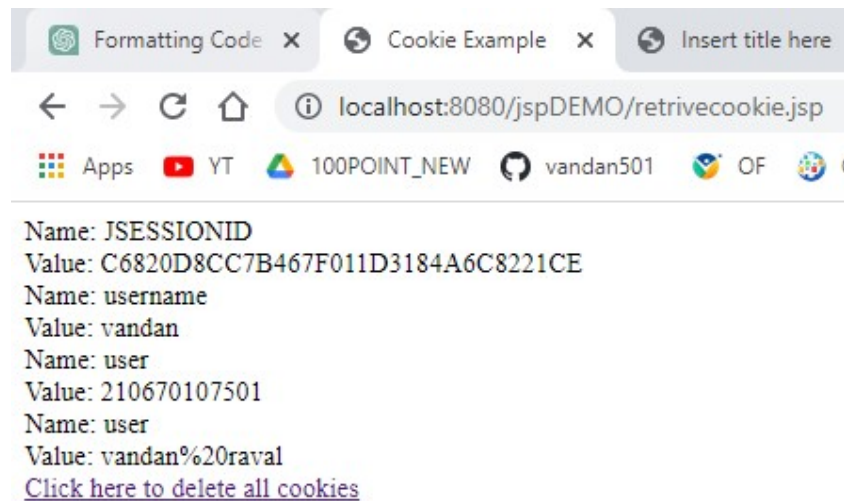
```
<a href="retrivecookie.jsp">To retrive Cookie Click here</a>
```

```
</body>
```

```
</html>
```

Output:





- **HttpSession example**

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>part1</title>
</head>
<body>
<h1>example of session in jsp</h1>
<%
out.println(session.isNew());

out.println(session.getId());
session.setAttribute("username", "Admin");
out.println(session.getAttribute("username"));
session.removeAttribute("username");
out.println(session.getAttribute("username"));

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

Example2 :session

sessionDemo2

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>HttpSession Demo</title>
</head>
<body>
    <h1>HttpSession Demo</h1>
    <%-- Check if this is the first visit to the site --%>
    <% if (session.isNew()) { %>
        <p>Welcome to the site!</p>
    <% } %>

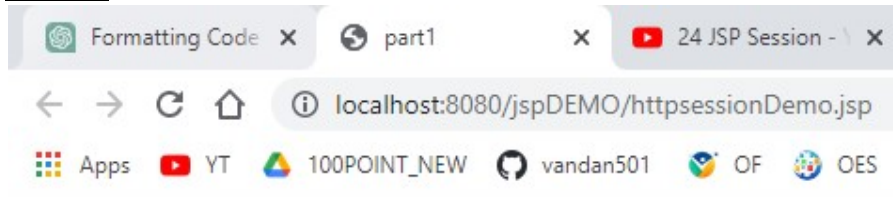
    <%-- Increase the count of site visits stored in the session --%>
    <% Integer count = (Integer) session.getAttribute("count");
    if (count == null) {
        count = 0;
    }
    count++;
```



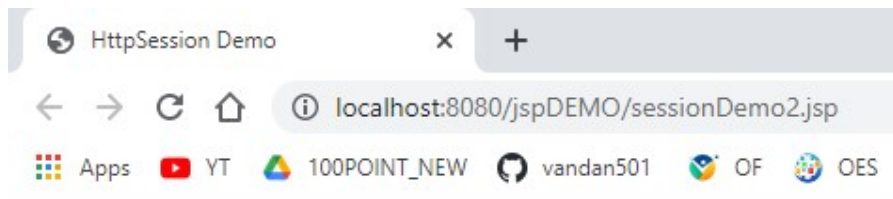
```

session.setAttribute("count", count);
%>
<p>You have visited this site <%=count%> times.</p>
</body>
</html>

```

Output:**example of session in jsp**

false C6820D8CC7B467F011D3184A6C8221CE Admin null

**HttpSession Demo**

You have visited this site 3 times.

EVALUATION:

Problem Analysis & Solution (3)	Understanding Level (3)	Timely Completion (2)	Mock (2)	Total (10)

EXPERIMENT NO: 7**DATE: / /****TITLE:** JSTL and JSP database connection.**OBJECTIVES:** On completion of this experiment student will able to...

➤ know the concept of JSTL core,function,formatting,xml,sql tags and JSP database CRUD operations.

THEORY: The JSP Standard Tag Library (JSTL) represents a set of tags to simplify the JSP development.

Advantage of JSTL

1. Fast Development JSTL provides many tags that simplify the JSP.
2. Code Reusability We can use the JSTL tags on various pages.
3. No need to use scriptlet tag It avoids the use of scriptlet tag.

Example:

```
<% @ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<title>Core Tag Example</title>
</head>
<body>
<c:set var="Income" scope="session" value="${4000*4}"/>
<c:out value="${Income}"/>
</body>
</html>
```

Ex:

```
<% @ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<title>Core Tag Example</title>
</head>
```

```

<body>
<c:set var="income" scope="session" value="\${4000*4}"/>
<p>Your income is : <c:out value="\${income}"/></p>
<c:choose>
<c:when test="\${income <= 1000}">
Income is not good.
</c:when>
<c:when test="\${income > 10000}">
Income is very good.
</c:when>
<c:otherwise>
Income is undetermined...
</c:otherwise>
</c:choose>
</body>
</html>

```

JSTL Function Tags

JSTL Function Tags List

JSTL Functions	Description
fn:contains()	It is used to test if an input string containing the specified substring in a program.
fn:containsIgnoreCase()	It is used to test if an input string contains the specified substring as a case insensitive way.
fn:endsWith()	It is used to test if an input string ends with the specified suffix.
fn:escapeXml()	It escapes the characters that would be interpreted as XML markup.
fn:indexOf()	It returns an index within a string of first occurrence of a specified substring.
fn:trim()	It removes the blank spaces from both the ends of a string.
fn:startsWith()	It is used for checking whether the given string is started with a particular string value.
fn:split()	It splits the string into an array of substrings.

fn:toLowerCase() It converts all the characters of a string to lower case.

fn:toUpperCase() It converts all the characters of a string to upper case.

fn:substring() It returns the subset of a string according to the given start and end position.

fn:substringAfter() It returns the subset of string after a specific substring.

fn:substringBefore() It returns the subset of string before a specific substring.

fn:length() It returns the number of characters inside a string, or the number of items in a collection.

fn:replace() It replaces all the occurrence of a string with another.

string sequence Ex

```
<% @ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
```

```
<% @ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
```

```
<html>
```

```
<head>
```

```
<title>Using JSTL Function </title>
```

```
</head>
```

```
<body>
```

```
<c:set var="site" value="javatpoint.com"/> <c:set var="author" value="Sonoo Jaiswal"/> Hi,  
This is ${fn:toUpperCase(site)} developed by ${fn:toUpperCase(author)}.
```

```
</body>
```

```
</html>
```

JSTL XML tags List

XML Tags	Descriptions
x:out	Similar to <%= ... > tag, but for XPath expressions.
x:parse	It is used for parse the XML data specified either in the tag body or an attribute.
x:set	It is used to sets a variable to the value of an XPath expression.
x:choose	It is a conditional tag that establish a context for mutually exclusive conditional operations. x:when It is a subtag of that will include its body if the condition evaluated be 'true'.
x:otherwise	It is subtag of that follows tags and runs only if all the prior conditions evaluated be 'false'.

x:if It is used for evaluating the test XPath expression and if it is true, it will processes its body content.

x:transform It is used in a XML document for providing the XSL(Extensible Stylesheet Language) transformation.

x:param It is used along with the transform tag for setting the parameter in the

XSLT style sheet. Ex:

```
<% @ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
<% @ taglib prefix="x" uri="http://java.sun.com/jsp/jstl/xml" %>
<html>
<head>
<title>x:choose Tag</title>
</head>
<body>
<h3>Books Information:</h3>
<c:set var="xmltext">
<books>
<book>
<name>Three mistakes of my life</name>
<author>Chetan Bhagat</author>
<price>200</price>
</book>
<book>
<name>Tomorrow land</name>
<author>Brad Bird</author>
<price>2000</price>
</book>
</books>
</c:set>
<x:parse xml="{xmltext}" var="output"/>
```

```

<x:choose>
<x:when select="$output//book/author = 'Chetan bhagat'">
Book is written by Chetan bhagat
</x:when>
<x:when select="$output//book/author = 'Brad Bird'">
Book is written by Brad Bird
</x:when>
x:otherwise>
The author is unknown...
</x:otherwise>
</x:choose>
</body>
</html>

```

JSTL SQL Tags

JSTL SQL Tags List

SQL Tags

Descriptions

sql:setDataSource It is used for creating a simple data source suitable only for prototyping.

sql:query It is used for executing the SQL query defined in its sql attribute or the body.

sql:update It is used for executing the SQL update defined in its sql attribute or in the tag body.

sql:param It is used for sets the parameter in an SQL statement to the specified value.

sql:dateParam It is used for sets the parameter in an SQL statement to a specified java.util.Date value.

sql:transaction It is used to provide the nested database action with a common connection.

EXAMPLE:

Sample.jsp

```

<% @ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<% @taglib uri="http://java.sun.com/jsp/jstl/sql" prefix="sql" %>

```

```

<sql:setDataSource var="db" driver="com.mysql.jdbc.Driver"
url="jdbc:mysql://localhost:3306/gec" user="root" password="root" />
<sql:query dataSource="${db}" var="rs">
SELECT * from student;
</sql:query>
<table border="1" width="100% ">
<tr>
<td> RollNo </td>
<td> Name </td>
<td> Branch </td>
<td> Subject 1 </td>
<td> Subject 2 </td>
<td> Subject 3 </td>
<td> Semester </td>
</tr>
<c:forEach var="table" items="${rs.rows}">
<tr>
<td> <c:out value="${table.RollNo}" /> </td>
<td> <c:out value="${table.Name}" /> </td>
<td> <c:out value="${table.Branch}" /> </td>
<td> <c:out value="${table.sub1_name}" /> </td>
<td> <c:out value="${table.sub2_name}" /> </td>
<td> <c:out value="${table.sub3_name}" /> </td>
<td> <c:out value="${table.semester}" /> </td>
</tr>
</c:forEach>
</table>

```

Output:

Student table will be created in mySQL Workbench as:

Running an application now...

RollNo	Name	Branch	Subject 1	Subject 2	Subject 3	Semester
1	Ansh	Mechanical				
2	Dhaval	Electrical				
3	Akhil	Computer				
101	Jaydeep	Computer	Data Structure	Database Systems	Programming in C	3
102	Hitesh	Computer	C++	Operating Systems	Computer Network	4
103	Ansh	Computer	Digital Electronics	Economics	Advanced Maths	3
104	Jignesh	Computer	Software Engineering	NET	Advanced JAVA	6
105	Jigar	Computer	Web Technology	Theory OF Computation	DE 2 B	6

EVALUATION:

Problem Analysis & Solution (3)	Understanding Level (3)	Timely Completion (2)	Mock(2)	Total (10)

Signature with date_____

EXPERIMENT NO: 8**DATE: / /****TITLE:** Introduction to JSF**OBJECTIVES:** On completion of this experiment student will able to...

➤ know the concept of JSF.

THEORY: It is a server side component based user interface framework. It is used to develop web applications. It provides a well-defined programming model and consists of rich API and tag libraries. The latest version JSF 2 uses Facelets as its default templating system. It is written in Java.

The JSF API provides components (inputText, commandButton etc) and helps to manage their states. It also provides server-side validation, data conversion, defining page navigation, provides extensibility, supports for internationalization, accessibility etc.

The JSF Tag libraries are used to add components on the web pages and connect components with objects on the server. It also contains tag handlers that implements the component tag.

With the help of these features and tools, you can easily and effortlessly create server-side user interface.

Example:**Index.xhtml**

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
Transitional//EN""http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:h="http://xmlns.jcp.org/jsf/html">
<h:head>
<title>User Form</title>
</h:head>
<h:body>
<h:form>
<h:outputLabel for="username">User Name</h:outputLabel>
<h:inputText id="username" value="#{user.name}" required="true"
requiredMessage="User Name is required" /><br/>
<h:commandButton id="submit-button" value="Submit" action="response.xhtml"/>
</h:form>
```

```
</h:body>
```

```
</html>
```

Response.xhtml

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

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0
```

```
Transitional//EN""http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml" xmlns:h="http://xmlns.jcp.org/jsf/html">
```

```
<h:head>
```

```
<title>Welcome Page</title>
```

```
</h:head>
```

```
<h:body>
```

```
<h2>Hello, <h:outputText value="#{user.name}"></h:outputText></h2>
```

```
</h:body>
```

```
</html>
```

User.java

```
package managedbean;
```

```
Facelet Title
```

```
import javax.faces.bean.ManagedBean;
```

```
import javax.faces.bean.RequestScoped;
```

```
@ManagedBean
```

```
@RequestScoped
```

```
public class User {
```

```
Advance Java Programming (3160707)
```

```
String name;
```

```
public String getName() {
```

```
return name;
```

```
}
```

```
public void setName(String name) {
```

```
this.name = name;
```

Enrollment no:210670107501

```

}
}

```

Web.xml

```

<?xml version="1.0" encoding="UTF-8"?>

<web-app version="3.1" xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">

  <context-param>

    <param-name>javax.faces.PROJECT_STAGE</param-name>

    <param-value>Development</param-value>

  </context-param>

  <servlet>

    <servlet-name>Faces Servlet</servlet-name>

    <servlet-class>javax.faces.webappFacelet Title.FacesServlet</servlet-class>

    <load-on-startup>1</load-on-startup>

  </servlet>

  <servlet-mapping>

    <servlet-name>Faces Servlet</servlet-name>

    <url-pattern>/faces/*</url-pattern>

  </servlet-mapping>

  <welcome-file-list>

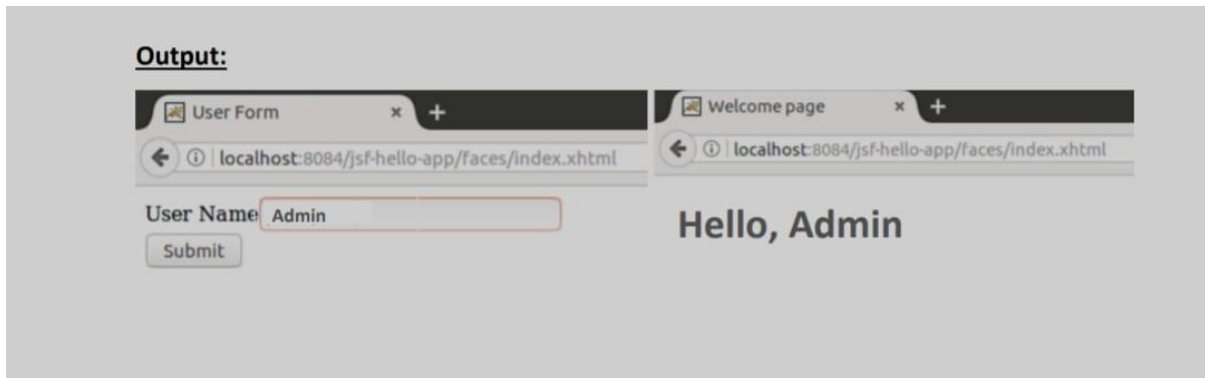
    <welcome-file>faces/index.xhtml</welcome-file>

  </welcome-file-list>

</web-app>

```

Output:



EVALUATION:

Problem Analysis & Solution (3)	Understanding Level (3)	Timely Completion (2)	Mock(2)	Total (10)

Signature with date_____

EXPERIMENT NO: 9**DATE: / /****TITLE:** Introduction to Hibernate.**OBJECTIVES:** On completion of this experiment student will able to...

➤ know the concept of Hibernate.

THEORY:

Hibernate is a Java framework that simplifies the development of Java application to interact with the database. It is an open source, lightweight, ORM (Object Relational Mapping) tool. Hibernate implements the specifications of JPA (Java Persistence API) for data persistence.

Index.jsp

```
<form action="register.jsp" method="post">
Name:<input type="text" name="name"/><br><br>
Password:<input type="password" name="password"/><br><br>
Email ID:<input type="text" name="email"/><br><br>
<input type="submit" value="register"/>
</form>
```

Register.jsp

```
<% @page import="com.javatpoint.mypack.UserDao"%>
<jsp:useBean id="obj" class="com.javatpoint.mypack.User">
</jsp:useBean>
<jsp:setProperty property="*" name="obj"/>
<%
int i=UserDao.register(obj);
if(i>0)
out.print("You are successfully registered");
%>
```

User.java

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

```
import org.hibernate.Transaction;
import org.hibernate.boot.Metadata;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

public class UserDao {

    public static int register(User u){
        int i=0;

        StandardServiceRegistry ssr = new
        StandardServiceRegistryBuilder().configure("hibernate.cfg.xml").build();
        Metadata meta = new MetadataSources(ssr).getMetadataBuilder().build();
        SessionFactory factory = meta.getSessionFactoryBuilder().build();
        Session session = factory.openSession();
        Transaction t = session.beginTransaction();
        i=(Integer)session.save(u);
        t.commit();
        session.close();
        return i;

    }
}
```

hibernate.cfg.xml

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
"-//Hibernate/Hibernate Configuration DTD 5.3//EN"
"http://hibernate.sourceforge.net/hibernate-configuration-5.3.dtd">
```

```

<hibernate-configuration>
  <session-factory>
    <property name="hbm2ddl.auto">create</property>
    <property name="dialect">org.hibernate.dialect.Oracle9Dialect</property>
    <property name="connection.url">jdbc:oracle:thin:@localhost:1521:xe</property>
    <property name="connection.username">system</property>
    <property name="connection.password">jtp</property>
    <property name="connection.driver_class">oracle.jdbc.driver.OracleDriver</property>

    <mapping resource="user.hbm.xml"/>
  </session-factory>
</hibernate-configuration>

```

user.hbm.xml

```

<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-mapping PUBLIC
"-//Hibernate/Hibernate Mapping DTD 5.3//EN"
"http://hibernate.sourceforge.net/hibernate-mapping-5.3.dtd">

```

```

<hibernate-mapping>
  <class name="com.java.mypack.User" table="u400">
    <id name="id">
      <generator class="increment"></generator>
    </id>
    <property name="name"></property>
    <property name="password"></property>
    <property name="email"></property>
  </class>
</hibernate-mapping>

```

Output:**Enrollment no:210670107501**

Output:

← → ↻ ⓘ localhost:8080/HBWebapp/ 🔑 ☆

Name:

Password:

Email ID:

You are successfully registered

EVALUATION:

Problem Analysis & Solution (3)	Understanding Level (3)	Timely Completion (2)	Mock(2)	Total (10)

Signature with date_____

EXPERIMENT NO: 10**DATE: //****TITLE:** Introduction to Spring**OBJECTIVES:** On completion of this experiment student will able to...

➤ know the concept of Spring framework.

THEORY:Spring is a lightweight framework. It can be thought of as a framework of frameworks because it provides support to various frameworks such as Struts, Hibernate, Tapestry, EJB, JSF etc. The framework, in broader sense, can be defined as a structure where we find solution of the various technical problems. The Spring framework comprises several modules such as IOC, AOP, DAO, Context, ORM, WEB MVC etc.Spring Framework is based on two design principles – Dependency Injection .

Aspect Oriented Programming.Java **Dependency** Injection design pattern allows us to remove the hard-coded dependencies and make our application loosely coupled, extendable and maintainable. We can implement dependency injection in java to move the dependency resolution from compile-time to runtime.

Spring AOP Overview : Most of the enterprise applications have some common crosscutting concerns that is applicable for different types of Objects and modules. Some of the common crosscutting concerns are logging, transaction management, data validation etc. In Object Oriented Programming, modularity of application is achieved by Classes whereas in Aspect Oriented Programming application modularity is achieved by Aspects and they are configured to cut across different classes. Spring AOP takes out the direct dependency of crosscutting tasks from classes that we can't achieve through normal object oriented programming model. For example, we can have a separate class for logging but again the functional classes will have to call these methods to achieve logging across the application.

Aspect Oriented Programming Core Concept Before we dive into implementation of Spring AOP implementation, we should understand the core concepts of AOP.

Aspect: An aspect is a class that implements enterprise application concerns that cut across multiple classes, such as transaction management. Aspects can be a normal class configured through Spring XML configuration or we can use Spring AspectJ integration to define a class as Aspect using @Aspect annotation.

Join Point: A join point is the specific point in the application such as method execution, exception handling, changing object variable values etc. In Spring AOP a join points is always the execution of a method.

Advice: Advices are actions taken for a particular join point. In terms of programming, they are method that gets executed when a certain join point with matching pointcut is reached in the application. You can think of Advices as Struts2 interceptors or Servlet Filters.

1. Pointcut: Pointcut are expressions that is matched with join points to determine whether advice needs to be executed or not. Pointcut uses different kinds of expressions that are matched with the join points and Spring framework uses the AspectJ pointcut expression language.

2. Target Object: They are the object on which advices are applied. Spring AOP is implemented using runtime proxies so this object is always a proxied object. What it means is that a subclass is created at runtime where the target method is overridden and advices are included based on their configuration.

3. AOP proxy: Spring AOP implementation uses JDK dynamic proxy to create the Proxy classes with target classes and advice invocations, these are called AOP proxy classes. We can also use CGLIB proxy by adding it as the dependency in the Spring AOP project.

4. Weaving: It is the process of linking aspects with other objects to create the advised proxy objects. This can be done at compile time, load time or at runtime. Spring AOP performs weaving at the runtime.

Example:

HelloSpring.java

```
package com.springEg;

public class HelloSpring {

    private String message;

    public void setMessage(String message){

        this.message = message;

    }

    public void getMessage(){

        System.out.println("Your Message : " + message);

    }

}
```

MainApp.java

```
package com.springEg;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

    public static void main(String[] args) {

        ApplicationContext context = new ClassPathXmlApplicationContext("Beans.xml");

        HelloSpring obj = (HelloSpring) context.getBean("helloSpring");

        obj.getMessage();

    }

}
```

```

}
}

```

Bean.xml

```

<?xml version = "1.0" encoding = "UTF-8"?>

<beans xmlns = "http://www.springframework.org/schema/beans"
xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation = "http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

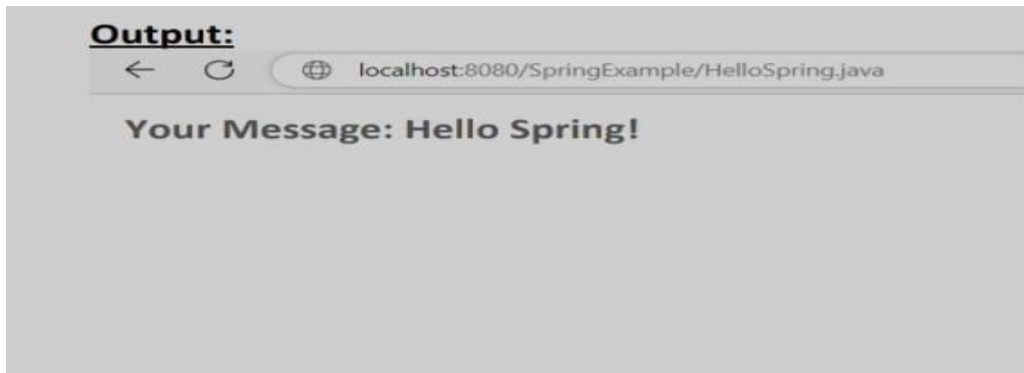
<bean id = "helloWorld" class = "com.springEg.HelloSpring">

<property name = "message" value = "Hello Spring!"/>

</bean>

</beans>

```

Output:**EVALUATION:**

Problem Analysis & Solution (3)	Understanding Level (3)	Timely Completion (2)	Mock(2)	Total (10)

Signature with date_____