SRM Institute of Science and Technology

Delhi – Meerut Road, Sikri Kalan, Ghaziabad, Uttar Pradesh – 201204

Department of Computer Applications

Circular – 2023-24

B.Sc. CS 4th Sem

Enterprise Java Programming (UCS23401J)- Lab Manual

This manual provides a structured guide for practical exercises in Enterprise Java Programming. Each laboratory exercise is detailed with its aim, step-by-step procedure, illustrative source code, sample input, and expected output.

Laboratory 1: Create distributed applications using RMI

Title

Distributed Applications using RMI

Aim

To understand and implement distributed applications using Java Remote Method Invocation (RMI) to allow objects running in one Java Virtual Machine (JVM) to invoke methods on an object running in another JVM.

Procedure

1. **Define the Remote Interface:** Create an interface that extends `java.rmi.Remote` and declares the methods that can be invoked remotely. Each method must declare `java.rmi.RemoteException` in its `throws` clause.
2. **Implement the Remote Interface:** Create a class that implements the remote interface. This class will provide the actual implementation of the remote methods. It must also extend `java.rmi.server.UnicastRemoteObject` (or use `java.rmi.activation.Activatable` for activatable objects) to make the object available for remote calls.
3. **Create the Server Application:**
    - Instantiate the remote object implementation.
    - Register the remote object with the RMI registry using `Naming.rebind()` or `Registry.rebind()`. This makes the object discoverable by clients.
4. **Create the Client Application:**
    - Look up the remote object in the RMI registry using `Naming.lookup()` or `Registry.lookup()`.
    - Cast the returned object to the remote interface type.
    - Invoke the remote methods as if they were local methods.
5. **Compile and Run:**
    - Compile all Java files (`.java` files) using `javac`.
    - Generate stubs and skeletons (for JDK 8 and earlier; not strictly necessary for JDK 9+ as they are generated dynamically) using `rmic` (e.g., `rmic RemoteInterfaceImpl`).

- Start the RMI registry (if not already running) using `rmiregistry` in a separate terminal.
- Run the server application.
- Run the client application.

Source Code

MyRemoteInterface.java (Remote Interface)

```
// MyRemoteInterface.java
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface MyRemoteInterface extends Remote {
    String sayHello() throws RemoteException;
    int add(int a, int b) throws RemoteException;
}
```

MyRemoteInterfaceImpl.java (Remote Object Implementation)

```
// MyRemoteInterfaceImpl.java
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class MyRemoteInterfaceImpl extends UnicastRemoteObject implements
MyRemoteInterface {

    // Constructor must throw RemoteException
    public MyRemoteInterfaceImpl() throws RemoteException {
        super(); // Calls the constructor of UnicastRemoteObject
    }

    @Override
    public String sayHello() throws RemoteException {
        return "Hello from the RMI Server!";
    }

    @Override
    public int add(int a, int b) throws RemoteException {
        System.out.println("Server received add request: " + a + " + " + b);
        return a + b;
    }

    public static void main(String[] args) {
        try {
            // Create an instance of the remote object implementation
            MyRemoteInterfaceImpl server = new MyRemoteInterfaceImpl();

            // Bind the remote object to a name in the RMI registry
            // The RMI registry should be running on the default port (1099)
            // You can specify a different port if needed:
Naming.rebind("//localhost:1099/MyRemoteService", server);
            java.rmi.Naming.rebind("MyRemoteService", server);

            System.out.println("RMI Server is running and waiting for
clients...");
        } catch (Exception e) {
            System.err.println("RMI Server exception: " + e.toString());
            e.printStackTrace();
        }
    }
}
```

RMIClient.java (Client Application)

```
// RMIClient.java
```

```
import java.rmi.Naming;
import java.rmi.RemoteException;

public class RMIClient {
    public static void main(String[] args) {
        try {
            // Look up the remote object from the RMI registry
            // The URL format is rmi://<host>:<port>/<serviceName>
            // If port is omitted, default RMI registry port (1099) is used.
            MyRemoteInterface service = (MyRemoteInterface)
Naming.lookup("rmi://localhost/MyRemoteService");

            // Invoke remote methods
            String message = service.sayHello();
            System.out.println("Message from server: " + message);

            int sum = service.add(10, 20);
            System.out.println("Sum from server: " + sum);

        } catch (RemoteException e) {
            System.err.println("Client RemoteException: " + e.getMessage());
            e.printStackTrace();
        } catch (Exception e) {
            System.err.println("Client exception: " + e.toString());
            e.printStackTrace();
        }
    }
}
```

Input

No direct user input for this basic example. The client invokes methods with hardcoded values.

Expected Output

Terminal 1 (rmiregistry): (No output until server binds)

Terminal 2 (Server - `java MyRemoteInterfaceImpl`):

```
RMI Server is running and waiting for clients...
Server received add request: 10 + 20
```

Terminal 3 (Client - `java RMIClient`):

```
Message from server: Hello from the RMI Server!
Sum from server: 30
```

Laboratory 2: Create applications which can demonstrate the use of JDBC for Database Connectivity.

Title

Database Connectivity using JDBC

Aim

To demonstrate the fundamental steps of connecting a Java application to a relational database using JDBC (Java Database Connectivity) API.

Procedure

1. **Set up Database:** Ensure you have a database (e.g., MySQL, PostgreSQL, H2) installed and running. Create a sample database and a table (e.g., `employees` table).
2. **Add JDBC Driver:** Download the appropriate JDBC driver JAR file for your database and add it to your project's classpath.
3. **Load the JDBC Driver:** Dynamically load the JDBC driver class using `Class.forName()`. This step registers the driver with the `DriverManager`.
4. **Establish Connection:** Obtain a database connection using `DriverManager.getConnection()`, providing the database URL, username, and password.
5. **Create Statement:** Create a `Statement` object from the `Connection` object. This object is used to execute SQL queries.
6. **Execute Query:** Execute an SQL query (e.g., `SELECT`, `INSERT`, `UPDATE`, `DELETE`) using methods like `executeQuery()` (for `SELECT`) or `executeUpdate()` (for `INSERT`, `UPDATE`, `DELETE`).
7. **Process ResultSet (for SELECT):** If executing a `SELECT` query, retrieve the results using a `ResultSet` object. Iterate through the `ResultSet` to access data from each row and column.
8. **Close Resources:** Close all JDBC resources (ResultSet, Statement, Connection) in the reverse order of their creation to release database resources and prevent memory leaks. Use a `finally` block to ensure resources are closed even if exceptions occur.

Source Code

JDBCExample.java

```
// JDBCExample.java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class JDBCExample {

    // Database credentials and URL
    static final String JDBC_DRIVER = "com.mysql.cj.jdbc.Driver"; // For
MySQL 8+
    // static final String JDBC_DRIVER = "org.postgresql.Driver"; // For
PostgreSQL
    static final String DB_URL = "jdbc:mysql://localhost:3306/mydatabase"; //
Change to your DB URL
    // static final String DB_URL =
"jdbc:postgresql://localhost:5432/mydatabase";
```

```
    static final String USER = "root"; // Your database username
    static final String PASS = "password"; // Your database password

    public static void main(String[] args) {
        Connection conn = null;
        Statement stmt = null;
        ResultSet rs = null;

        try {
            // STEP 1: Register JDBC driver (optional for modern JDBC, but
good practice)
            Class.forName(JDBC_DRIVER);

            System.out.println("Connecting to database...");
            // STEP 2: Open a connection
            conn = DriverManager.getConnection(DB_URL, USER, PASS);
            System.out.println("Connected successfully!");

            // STEP 3: Create a Statement object
            stmt = conn.createStatement();

            // STEP 4: Execute a SQL query (e.g., create a table if it
doesn't exist)
            String sqlCreateTable = "CREATE TABLE IF NOT EXISTS employees ("
+
                                    "id INT AUTO_INCREMENT PRIMARY KEY," +
                                    "name VARCHAR(255) NOT NULL," +
                                    "age INT," +
                                    "department VARCHAR(100))";
            stmt.executeUpdate(sqlCreateTable);
            System.out.println("Table 'employees' created or already
exists.");

            // STEP 5: Insert data
            String sqlInsert = "INSERT INTO employees (name, age, department)
VALUES " +
                               "('Alice', 30, 'HR'), " +
                               "('Bob', 24, 'IT'), " +
                               "('Charlie', 35, 'Finance')";
            int rowsAffected = stmt.executeUpdate(sqlInsert);
            System.out.println(rowsAffected + " rows inserted.");

            // STEP 6: Select data
            String sqlSelect = "SELECT id, name, age, department FROM
employees";
            rs = stmt.executeQuery(sqlSelect);

            // STEP 7: Process the ResultSet
            System.out.println("\n--- Employee Data ---");
            while (rs.next()) {
                int id = rs.getInt("id");
                String name = rs.getString("name");
                int age = rs.getInt("age");
                String department = rs.getString("department");

                System.out.println("ID: " + id + ", Name: " + name + ", Age:
" + age + ", Dept: " + department);
            }

        } catch (SQLException se) {
            // Handle errors for JDBC
            se.printStackTrace();
        } catch (Exception e) {
            // Handle errors for Class.forName
            e.printStackTrace();
```

```
        } finally {
            // STEP 8: Close resources in finally block
            try {
                if (rs != null) rs.close();
            } catch (SQLException se2) {
                // Do nothing
            }
            try {
                if (stmt != null) stmt.close();
            } catch (SQLException se2) {
                // Do nothing
            }
            try {
                if (conn != null) conn.close();
                System.out.println("Connection closed.");
            } catch (SQLException se) {
                se.printStackTrace();
            }
        }
    }
}
```

Input

No direct user input. The program interacts with a pre-configured database.

Expected Output

```
Connecting to database...
Connected successfully!
Table 'employees' created or already exists.
3 rows inserted.

--- Employee Data ---
ID: 1, Name: Alice, Age: 30, Dept: HR
ID: 2, Name: Bob, Age: 24, Dept: IT
ID: 3, Name: Charlie, Age: 35, Dept: Finance
Connection closed.
```

(Note: Actual IDs may vary based on database auto-increment behavior and prior runs.)

Laboratory 3: Create student applications using JDBC Database Connectivity

Title

Student Management Application using JDBC

Aim

To develop a simple command-line based student management application that performs basic CRUD (Create, Read, Update, Delete) operations on student records stored in a database using JDBC.

Procedure

1. **Database Setup:** Create a database (e.g., `student_db`) and a table (e.g., `students`) with columns like `id`, `name`, `roll_number`, `grade`.
2. **Student Model (POJO):** Create a `Student` class to represent student data (id, name, roll number, grade).
3. **DAO (Data Access Object) Class:** Create a `StudentDAO` class responsible for all database interactions. This class will contain methods for:
    - `addStudent(Student student)`: Inserts a new student record.
    - `getAllStudents()`: Retrieves all student records.
    - `getStudentById(int id)`: Retrieves a specific student record by ID.
    - `updateStudent(Student student)`: Updates an existing student record.
    - `deleteStudent(int id)`: Deletes a student record.
    - Each method will handle JDBC connection, statement creation, query execution, and resource closing.
4. **Main Application Class:** Create a `StudentApp` class with a `main` method to provide a menu-driven interface for the user to perform CRUD operations. It will interact with the `StudentDAO`.
5. **Error Handling:** Implement robust `try-catch-finally` blocks for JDBC operations to handle `SQLExceptions` and ensure resources are closed.

Source Code

`Student.java` (Model/POJO)

```
// Student.java
public class Student {
    private int id;
    private String name;
    private String rollNumber;
    private String grade;

    public Student(int id, String name, String rollNumber, String grade) {
        this.id = id;
        this.name = name;
        this.rollNumber = rollNumber;
        this.grade = grade;
    }

    public Student(String name, String rollNumber, String grade) {
        this.name = name;
        this.rollNumber = rollNumber;
        this.grade = grade;
    }
```

```
    // Getters
    public int getId() { return id; }
    public String getName() { return name; }
    public String getRollNumber() { return rollNumber; }
    public String getGrade() { return grade; }

    // Setters (optional, but useful for updates)
    public void setId(int id) { this.id = id; }
    public void setName(String name) { this.name = name; }
    public void setRollNumber(String rollNumber) { this.rollNumber =
rollNumber; }
    public void setGrade(String grade) { this.grade = grade; }

    @Override
    public String toString() {
        return "ID: " + id + ", Name: " + name + ", Roll No: " + rollNumber +
", Grade: " + grade;
    }
}
```

StudentDAO.java *(Data Access Object)*

```
// StudentDAO.java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

public class StudentDAO {

    private static final String JDBC_DRIVER = "com.mysql.cj.jdbc.Driver";
    private static final String DB_URL =
"jdbc:mysql://localhost:3306/student_db"; // Ensure 'student_db' exists
    private static final String USER = "root";
    private static final String PASS = "password";

    public StudentDAO() {
        try {
            Class.forName(JDBC_DRIVER);
            createTableIfNotExists();
        } catch (ClassNotFoundException e) {
            System.err.println("JDBC Driver not found: " + e.getMessage());
        }
    }

    private Connection getConnection() throws SQLException {
        return DriverManager.getConnection(DB_URL, USER, PASS);
    }

    private void createTableIfNotExists() {
        String sql = "CREATE TABLE IF NOT EXISTS students (" +
                     "id INT AUTO_INCREMENT PRIMARY KEY," +
                     "name VARCHAR(255) NOT NULL," +
                     "roll_number VARCHAR(50) UNIQUE NOT NULL," +
                     "grade VARCHAR(10))";
        try (Connection conn = getConnection();
             PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.executeUpdate();
            System.out.println("Table 'students' created or already
exists.");
        } catch (SQLException e) {
            System.err.println("Error creating table: " + e.getMessage());
```

```
        }
    }

    public void addStudent(Student student) {
        String sql = "INSERT INTO students (name, roll_number, grade) VALUES
(?, ?, ?)";
        try (Connection conn = getConnection();
             PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setString(1, student.getName());
            pstmt.setString(2, student.getRollNumber());
            pstmt.setString(3, student.getGrade());
            pstmt.executeUpdate();
            System.out.println("Student added successfully: " +
student.getName());
        } catch (SQLException e) {
            System.err.println("Error adding student: " + e.getMessage());
        }
    }

    public List<Student> getAllStudents() {
        List<Student> students = new ArrayList<>();
        String sql = "SELECT id, name, roll_number, grade FROM students";
        try (Connection conn = getConnection();
             PreparedStatement pstmt = conn.prepareStatement(sql);
             ResultSet rs = pstmt.executeQuery()) {
            while (rs.next()) {
                int id = rs.getInt("id");
                String name = rs.getString("name");
                String rollNumber = rs.getString("roll_number");
                String grade = rs.getString("grade");
                students.add(new Student(id, name, rollNumber, grade));
            }
        } catch (SQLException e) {
            System.err.println("Error retrieving students: " +
e.getMessage());
        }
        return students;
    }

    public Student getStudentById(int id) {
        String sql = "SELECT id, name, roll_number, grade FROM students WHERE
id = ?";
        try (Connection conn = getConnection();
             PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setInt(1, id);
            try (ResultSet rs = pstmt.executeQuery()) {
                if (rs.next()) {
                    String name = rs.getString("name");
                    String rollNumber = rs.getString("roll_number");
                    String grade = rs.getString("grade");
                    return new Student(id, name, rollNumber, grade);
                }
            }
        } catch (SQLException e) {
            System.err.println("Error retrieving student by ID: " +
e.getMessage());
        }
        return null;
    }

    public void updateStudent(Student student) {
        String sql = "UPDATE students SET name = ?, roll_number = ?, grade =
? WHERE id = ?";
        try (Connection conn = getConnection();
             PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setString(1, student.getName());
```

```
            pstmt.setString(2, student.getRollNumber());
            pstmt.setString(3, student.getGrade());
            pstmt.setInt(4, student.getId());
            int rowsAffected = pstmt.executeUpdate();
            if (rowsAffected > 0) {
                System.out.println("Student updated successfully: ID " +
student.getId());
            } else {
                System.out.println("Student with ID " + student.getId() + "
not found for update.");
            }
        } catch (SQLException e) {
            System.err.println("Error updating student: " + e.getMessage());
        }
    }

    public void deleteStudent(int id) {
        String sql = "DELETE FROM students WHERE id = ?";
        try (Connection conn = getConnection();
             PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setInt(1, id);
            int rowsAffected = pstmt.executeUpdate();
            if (rowsAffected > 0) {
                System.out.println("Student deleted successfully: ID " + id);
            } else {
                System.out.println("Student with ID " + id + " not found for
deletion.");
            }
        } catch (SQLException e) {
            System.err.println("Error deleting student: " + e.getMessage());
        }
    }
}
```

StudentApp.java **(Main Application)**

```
// StudentApp.java
import java.util.List;
import java.util.Scanner;

public class StudentApp {
    private StudentDAO studentDAO;
    private Scanner scanner;

    public StudentApp() {
        studentDAO = new StudentDAO();
        scanner = new Scanner(System.in);
    }

    public void start() {
        int choice;
        do {
            displayMenu();
            System.out.print("Enter your choice: ");
            choice = scanner.nextInt();
            scanner.nextLine(); // Consume newline

            switch (choice) {
                case 1:
                    addStudent();
                    break;
                case 2:
                    viewAllStudents();
                    break;
                case 3:
                    updateStudent();
```

```
                    break;
                case 4:
                    deleteStudent();
                    break;
                case 5:
                    System.out.println("Exiting application. Goodbye!");
                    break;
                default:
                    System.out.println("Invalid choice. Please try again.");
            }
            System.out.println("\nPress Enter to continue...");
            scanner.nextLine(); // Wait for user to press Enter
        } while (choice != 5);

        scanner.close();
    }

    private void displayMenu() {
        System.out.println("\n--- Student Management System ---");
        System.out.println("1. Add New Student");
        System.out.println("2. View All Students");
        System.out.println("3. Update Student");
        System.out.println("4. Delete Student");
        System.out.println("5. Exit");
    }

    private void addStudent() {
        System.out.println("\n--- Add New Student ---");
        System.out.print("Enter student name: ");
        String name = scanner.nextLine();
        System.out.print("Enter roll number: ");
        String rollNumber = scanner.nextLine();
        System.out.print("Enter grade: ");
        String grade = scanner.nextLine();

        Student newStudent = new Student(name, rollNumber, grade);
        studentDAO.addStudent(newStudent);
    }

    private void viewAllStudents() {
        System.out.println("\n--- All Students ---");
        List<Student> students = studentDAO.getAllStudents();
        if (students.isEmpty()) {
            System.out.println("No students found.");
        } else {
            for (Student student : students) {
                System.out.println(student);
            }
        }
    }

    private void updateStudent() {
        System.out.println("\n--- Update Student ---");
        System.out.print("Enter student ID to update: ");
        int id = scanner.nextInt();
        scanner.nextLine(); // Consume newline

        Student existingStudent = studentDAO.getStudentById(id);
        if (existingStudent == null) {
            System.out.println("Student with ID " + id + " not found.");
            return;
        }

        System.out.println("Current details: " + existingStudent);
        System.out.print("Enter new name (leave blank to keep current: " +
existingStudent.getName() + "): ");
```

```
        String name = scanner.nextLine();
        if (!name.isEmpty()) {
            existingStudent.setName(name);
        }

        System.out.print("Enter new roll number (leave blank to keep current:
" + existingStudent.getRollNumber() + "): ");
        String rollNumber = scanner.nextLine();
        if (!rollNumber.isEmpty()) {
            existingStudent.setRollNumber(rollNumber);
        }

        System.out.print("Enter new grade (leave blank to keep current: " +
existingStudent.getGrade() + "): ");
        String grade = scanner.nextLine();
        if (!grade.isEmpty()) {
            existingStudent.setGrade(grade);
        }

        studentDAO.updateStudent(existingStudent);
    }

    private void deleteStudent() {
        System.out.println("\n--- Delete Student ---");
        System.out.print("Enter student ID to delete: ");
        int id = scanner.nextInt();
        scanner.nextLine(); // Consume newline

        studentDAO.deleteStudent(id);
    }

    public static void main(String[] args) {
        StudentApp app = new StudentApp();
        app.start();
    }
}
```

Input

The user will interact with a command-line menu, providing choices (1-5) and details for adding, updating, or deleting students.

Example Input Sequence:

```
1
John Doe
S001
A
(Press Enter)
2
(Press Enter)
3
1
Jane Doe
S002
B
(Press Enter)
4
1
(Press Enter)
5
```

Expected Output

```
Table 'students' created or already exists.

--- Student Management System ---
1. Add New Student
2. View All Students
3. Update Student
4. Delete Student
5. Exit
Enter your choice: 1

--- Add New Student ---
Enter student name: John Doe
Enter roll number: S001
Enter grade: A
Student added successfully: John Doe

Press Enter to continue...

--- Student Management System ---
1. Add New Student
2. View All Students
3. Update Student
4. Delete Student
5. Exit
Enter your choice: 2

--- All Students ---
ID: 1, Name: John Doe, Roll No: S001, Grade: A

Press Enter to continue...

--- Student Management System ---
1. Add New Student
2. View All Students
3. Update Student
4. Delete Student
5. Exit
Enter your choice: 3

--- Update Student ---
Enter student ID to update: 1
Current details: ID: 1, Name: John Doe, Roll No: S001, Grade: A
Enter new name (leave blank to keep current: John Doe): Jane Doe
Enter new roll number (leave blank to keep current: S001): S002
Enter new grade (leave blank to keep current: A): B
Student updated successfully: ID 1

Press Enter to continue...

--- Student Management System ---
1. Add New Student
2. View All Students
3. Update Student
4. Delete Student
5. Exit
Enter your choice: 2

--- All Students ---
ID: 1, Name: Jane Doe, Roll No: S002, Grade: B

Press Enter to continue...

--- Student Management System ---
1. Add New Student
```

```
2. View All Students
3. Update Student
4. Delete Student
5. Exit
Enter your choice: 4

--- Delete Student ---
Enter student ID to delete: 1
Student deleted successfully: ID 1

Press Enter to continue...

--- Student Management System ---
1. Add New Student
2. View All Students
3. Update Student
4. Delete Student
5. Exit
Enter your choice: 2

--- All Students ---
No students found.

Press Enter to continue...

--- Student Management System ---
1. Add New Student
2. View All Students
3. Update Student
4. Delete Student
5. Exit
Enter your choice: 5
Exiting application. Goodbye!
```

Laboratory 4: Develop Web Applications Using Servlet

Title

Developing Web Applications Using Servlets

Aim

To understand the fundamentals of Java Servlets and develop a basic web application that processes HTTP requests and generates dynamic HTTP responses.

Procedure

1. **Setup Development Environment:** Install a Java Development Kit (JDK) and an IDE (e.g., Eclipse, IntelliJ IDEA) with Apache Tomcat or another servlet container configured.
2. **Create a Dynamic Web Project:** In your IDE, create a new Dynamic Web Project.
3. **Create a Servlet Class:**
    - Create a Java class that extends `javax.servlet.http.HttpServlet`.
    - Override the `doGet()` and/or `doPost()` methods to handle HTTP GET and POST requests, respectively.
    - Inside these methods, use `HttpServletRequest` to retrieve request parameters and `HttpServletResponse` to write the response back to the client (e.g., HTML content).
4. **Configure Servlet (Web.xml or Annotations):**
    - **Using Annotations (Recommended for modern Servlets):** Add `@WebServlet("/URL_Pattern")` annotation directly above your servlet class to map it to a URL pattern.
    - **Using `web.xml` (Deployment Descriptor):**
        - Define the servlet in `web.xml` using `<servlet>` tags (specifying `<servlet-name>` and `<servlet-class>`).
        - Map the servlet to a URL pattern using `<servlet-mapping>` tags (specifying `<servlet-name>` and `<url-pattern>`).
5. **Create an HTML Form (Optional but Recommended):** Create an `index.html` or `form.html` file with an HTML form that submits data to your servlet.
6. **Deploy and Run:** Deploy the web application to your servlet container (e.g., Tomcat) and access it through a web browser.

Source Code

`index.html` (Front-end HTML Form)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Simple Servlet App</title>
    <style>
        body { font-family: 'Inter', sans-serif; display: flex; justify-
content: center; align-items: center; min-height: 100vh; background-color:
#f0f2f5; margin: 0; }
        .container { background-color: #ffffff; padding: 30px; border-radius:
12px; box-shadow: 0 4px 15px rgba(0, 0, 0, 0.1); text-align: center; max-
width: 400px; width: 90%; }
        h1 { color: #333; margin-bottom: 25px; }
        form { display: flex; flex-direction: column; gap: 15px; }
```

```
        label { text-align: left; font-weight: bold; color: #555; }
        input[type="text"] { padding: 12px; border: 1px solid #ddd; border-
radius: 8px; font-size: 16px; width: calc(100% - 24px); }
        input[type="submit"] { background-color: #007bff; color: white;
padding: 12px 20px; border: none; border-radius: 8px; cursor: pointer; font-
size: 18px; font-weight: bold; transition: background-color 0.3s ease; }
        input[type="submit"]:hover { background-color: #0056b3; }
    </style>
</head>
<body>
    <div class="container">
        <h1>Welcome to My Servlet App</h1>
        <form action="helloServlet" method="get">
            <label for="userName">Enter your name:</label>
            <input type="text" id="userName" name="userName"
placeholder="e.g., John Doe" required>
            <input type="submit" value="Say Hello">
        </form>
    </div>
</body>
</html>
```

HelloServlet.java **(Servlet Class)**

```
// src/main/java/com/example/HelloServlet.java
package com.example;

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

// Annotation to map this servlet to the URL pattern /helloServlet
@WebServlet("/helloServlet")
public class HelloServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * Handles HTTP GET requests.
     * Retrieves the 'userName' parameter from the request and sends a
personalized greeting.
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {

        // Set content type of the response
        response.setContentType("text/html");
        response.setCharacterEncoding("UTF-8");

        // Get the 'userName' parameter from the request
        String userName = request.getParameter("userName");

        // If userName is null or empty, provide a default
        if (userName == null || userName.trim().isEmpty()) {
            userName = "Guest";
        }

        // Get a PrintWriter to send HTML response back to the client
        PrintWriter out = response.getWriter();

        // Generate HTML response
        out.println("<!DOCTYPE html>");
```

```
        out.println("<html lang='en'>");
        out.println("<head>");
        out.println("    <meta charset='UTF-8'>");
        out.println("    <meta name='viewport' content='width=device-width,
initial-scale=1.0'>");
        out.println("    <title>Greeting</title>");
        out.println("    <style>");
        out.println("        body { font-family: 'Inter', sans-serif;
display: flex; justify-content: center; align-items: center; min-height:
100vh; background-color: #f0f2f5; margin: 0; }");
        out.println("        .container { background-color: #ffffff; padding:
30px; border-radius: 12px; box-shadow: 0 4px 15px rgba(0, 0, 0, 0.1); text-
align: center; max-width: 400px; width: 90%; }");
        out.println("        h1 { color: #333; margin-bottom: 20px; }");
        out.println("        p { color: #555; font-size: 1.1em; }");
        out.println("        a { display: inline-block; margin-top: 25px;
padding: 10px 20px; background-color: #007bff; color: white; text-decoration:
none; border-radius: 8px; transition: background-color 0.3s ease; }");
        out.println("        a:hover { background-color: #0056b3; }");
        out.println("    </style>");
        out.println("</head>");
        out.println("<body>");
        out.println("    <div class='container'>");
        out.println("        <h1>Hello, " + userName + "!</h1>");
        out.println("        <p>This is a simple greeting from your first
Servlet.</p>");
        out.println("        <a href='index.html'>Go Back</a>");
        out.println("    </div>");
        out.println("</body>");
        out.println("</html>");

        // Close the PrintWriter
        out.close();
    }

    /**
     * Handles HTTP POST requests.
     * For this example, it simply calls doGet to process the request.
     * In a real application, you would differentiate between GET and POST
logic.
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        doGet(request, response);
    }
}
```

Input

Access `index.html` in a web browser, enter a name in the text field, and click "Say Hello".

Example Input: Text field value: `Alice`

Expected Output

Browser displaying `index.html`: A web page with a heading "Welcome to My Servlet App" and a text input field labeled "Enter your name:" and a "Say Hello" button.

After submitting "Alice": A new web page displaying:

```
<!DOCTYPE html>
```

```
<html lang='en'>
<head>
    <meta charset='UTF-8'>
    <meta name='viewport' content='width=device-width, initial-scale=1.0'>
    <title>Greeting</title>
    <style>
        /* ... (CSS styles as in source code) ... */
    </style>
</head>
<body>
    <div class='container'>
        <h1>Hello, Alice!</h1>
        <p>This is a simple greeting from your first Servlet.</p>
        <a href='index.html'>Go Back</a>
    </div>
</body>
</html>
```

Visually, it would show "Hello, Alice!" as a prominent heading, followed by a descriptive paragraph and a "Go Back" link.

Laboratory 5: Develop Web Applications Using ServletRequest, ServletResponse

Title

Developing Web Applications Using `ServletRequest` and `ServletResponse`

Aim

To gain a deeper understanding of `HttpServletRequest` and `HttpServletResponse` objects in Servlets, demonstrating how to retrieve various request parameters (headers, method, context path) and how to control the response (setting headers, status codes, and writing different content types).

Procedure

1. **Setup Dynamic Web Project:** Create a new Dynamic Web Project in your IDE.
2. **Create a Servlet:**
    - Extend `HttpServlet`.
    - Override `doGet()` and `doPost()` methods.
3. **Utilize `HttpServletRequest`:**
    - Demonstrate retrieving request parameters using `request.getParameter()`.
    - Show how to get request headers using `request.getHeader()`.
    - Access other request information like `request.getMethod()`, `request.getRequestURI()`, `request.getContextPath()`, etc.
4. **Utilize `HttpServletResponse`:**
    - Set the content type of the response using `response.setContentType()`.
    - Write HTML content to the response using `response.getWriter()`.
    - Demonstrate setting response headers using `response.setHeader()`.
    - Show how to send redirects using `response.sendRedirect()`.
    - Illustrate setting HTTP status codes using `response.setStatus()`.
5. **HTML Form:** Create an `index.html` with a form to submit data (both GET and POST) to the servlet.
6. **Deployment:** Deploy the application to a servlet container.

Source Code

`index.html` (Front-end HTML Form)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Request/Response Demo</title>
    <style>
        body { font-family: 'Inter', sans-serif; display: flex; flex-
direction: column; justify-content: center; align-items: center; min-height:
100vh; background-color: #f0f2f5; margin: 0; padding: 20px; box-sizing:
border-box; }
        .container { background-color: #ffffff; padding: 30px; border-radius:
12px; box-shadow: 0 4px 15px rgba(0, 0, 0, 0.1); text-align: center; max-
width: 600px; width: 100%; margin-bottom: 20px; }
        h1 { color: #333; margin-bottom: 25px; }
        .form-section { display: flex; flex-direction: column; gap: 20px;
margin-bottom: 30px; }
```

```
        form { display: flex; flex-direction: column; gap: 15px; border: 1px
solid #eee; padding: 20px; border-radius: 10px; background-color: #f9f9f9; }
        label { text-align: left; font-weight: bold; color: #555; }
        input[type="text"] { padding: 12px; border: 1px solid #ddd; border-
radius: 8px; font-size: 16px; width: calc(100% - 24px); }
        input[type="submit"] { background-color: #28a745; color: white;
padding: 12px 20px; border: none; border-radius: 8px; cursor: pointer; font-
size: 18px; font-weight: bold; transition: background-color 0.3s ease; }
        input[type="submit"].post-button { background-color: #007bff; }
        input[type="submit"]:hover { background-color: #218838; }
        input[type="submit"].post-button:hover { background-color: #0056b3; }
        .redirect-link { display: inline-block; margin-top: 20px; padding:
12px 25px; background-color: #ffc107; color: #333; text-decoration: none;
border-radius: 8px; font-weight: bold; transition: background-color 0.3s
ease; }
        .redirect-link:hover { background-color: #e0a800; }
    </style>
</head>
<body>
    <div class="container">
        <h1>Servlet Request/Response Demo</h1>

        <div class="form-section">
            <h2>GET Request Example</h2>
            <form action="requestResponseServlet" method="get">
                <label for="param1Get">Parameter 1:</label>
                <input type="text" id="param1Get" name="param1"
value="Value1_GET">

                <label for="param2Get">Parameter 2:</label>
                <input type="text" id="param2Get" name="param2"
value="Value2_GET">

                <input type="submit" value="Submit GET Request">
            </form>
        </div>

        <div class="form-section">
            <h2>POST Request Example</h2>
            <form action="requestResponseServlet" method="post">
                <label for="param1Post">Parameter 1:</label>
                <input type="text" id="param1Post" name="param1"
value="Value1_POST">

                <label for="param2Post">Parameter 2:</label>
                <input type="text" id="param2Post" name="param2"
value="Value2_POST">

                <input type="submit" value="Submit POST Request" class="post-
button">
            </form>
        </div>

        <a href="redirectServlet" class="redirect-link">Trigger Redirect
Example</a>
    </div>
</body>
</html>
```

RequestResponseServlet.java *(Main Servlet)*

```
// src/main/java/com/example/RequestResponseServlet.java
package com.example;

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

```
import java.util.Enumeration;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/requestResponseServlet")
public class RequestResponseServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        processRequest(request, response, "GET");
    }

    protected void doPost(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        processRequest(request, response, "POST");
    }

    private void processRequest(HttpServletRequest request,
HttpServletResponse response, String method)
            throws ServletException, IOException {

        // Set content type and character encoding
        response.setContentType("text/html");
        response.setCharacterEncoding("UTF-8");

        // Get PrintWriter to write response
        PrintWriter out = response.getWriter();

        // Start HTML response
        out.println("<!DOCTYPE html>");
        out.println("<html lang='en'>");
        out.println("<head>");
        out.println("    <meta charset='UTF-8'>");
        out.println("    <meta name='viewport' content='width=device-width,
initial-scale=1.0'>");
        out.println("    <title>Request/Response Details</title>");
        out.println("    <style>");
        out.println("        body { font-family: 'Inter', sans-serif;
background-color: #f0f2f5; margin: 0; padding: 20px; box-sizing: border-box;
}");
        out.println("        .container { background-color: #ffffff; padding:
30px; border-radius: 12px; box-shadow: 0 4px 15px rgba(0, 0, 0, 0.1); max-
width: 800px; margin: 30px auto; }");
        out.println("        h1, h2 { color: #333; text-align: center;
margin-bottom: 25px; }");
        out.println("        table { width: 100%; border-collapse: collapse;
margin-bottom: 20px; }");
        out.println("        th, td { border: 1px solid #ddd; padding: 10px;
text-align: left; }");
        out.println("        th { background-color: #f2f2f2; color: #555;
}");
        out.println("        .back-link { display: block; width: fit-content;
margin: 20px auto 0; padding: 10px 20px; background-color: #007bff; color:
white; text-decoration: none; border-radius: 8px; transition: background-
color 0.3s ease; }");
        out.println("        .back-link:hover { background-color: #0056b3;
}");
        out.println("    </style>");
        out.println("</head>");
        out.println("<body>");
```

```
        out.println("    <div class='container'>");
        out.println("        <h1>Request and Response Details</h1>");
        out.println("        <h2>Request Method: " + method + "</h2>");

        // Display Request Parameters
        out.println("        <h2>Request Parameters:</h2>");
        out.println("        <table>");
        out.println("            <tr><th>Parameter Name</th><th>Parameter
Value</th></tr>");
        Enumeration<String> paramNames = request.getParameterNames();
        if (!paramNames.hasMoreElements()) {
            out.println("            <tr><td colspan='2'>No parameters
found.</td></tr>");
        } else {
            while (paramNames.hasMoreElements()) {
                String paramName = paramNames.nextElement();
                String paramValue = request.getParameter(paramName);
                out.println("            <tr><td>" + paramName + "</td><td>"
+ paramValue + "</td></tr>");
            }
        }
        out.println("        </table>");

        // Display Request Headers
        out.println("        <h2>Request Headers:</h2>");
        out.println("        <table>");
        out.println("            <tr><th>Header Name</th><th>Header
Value</th></tr>");
        Enumeration<String> headerNames = request.getHeaderNames();
        if (!headerNames.hasMoreElements()) {
            out.println("            <tr><td colspan='2'>No headers
found.</td></tr>");
        } else {
            while (headerNames.hasMoreElements()) {
                String headerName = headerNames.nextElement();
                String headerValue = request.getHeader(headerName);
                out.println("            <tr><td>" + headerName + "</td><td>"
+ headerValue + "</td></tr>");
            }
        }
        out.println("        </table>");

        // Display Other Request Information
        out.println("        <h2>Other Request Information:</h2>");
        out.println("        <table>");
        out.println("            <tr><th>Property</th><th>Value</th></tr>");
        out.println("            <tr><td>Request URI</td><td>" +
request.getRequestURI() + "</td></tr>");
        out.println("            <tr><td>Context Path</td><td>" +
request.getContextPath() + "</td></tr>");
        out.println("            <tr><td>Servlet Path</td><td>" +
request.getServletPath() + "</td></tr>");
        out.println("            <tr><td>Query String</td><td>" +
request.getQueryString() + "</td></tr>");
        out.println("            <tr><td>Remote Address</td><td>" +
request.getRemoteAddr() + "</td></tr>");
        out.println("            <tr><td>Server Name</td><td>" +
request.getServerName() + "</td></tr>");
        out.println("            <tr><td>Server Port</td><td>" +
request.getServerPort() + "</td></tr>");
        out.println("        </table>");

        // Demonstrate setting a response header and status code
        response.setHeader("Custom-Header", "This is a custom header from
Servlet");
```

```
        response.setStatus(HttpServletResponse.SC_OK); // Explicitly set 200
OK

        out.println("        <p>Response status code set to: " +
response.getStatus() + " (OK)</p>");
        out.println("        <p>A custom response header 'Custom-Header' has
been set.</p>");

        out.println("        <a href='index.html' class='back-link'>Go Back
to Form</a>");
        out.println("    </div>");
        out.println("</body>");
        out.println("</html>");

        out.close();
    }
}
```

RedirectServlet.java (Servlet for Redirection Example)

```
// src/main/java/com/example/RedirectServlet.java
package com.example;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/redirectServlet")
public class RedirectServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        System.out.println("RedirectServlet: Redirecting to index.html...");
        // Redirect to index.html
        response.sendRedirect("index.html?message=redirected");
    }
}
```

Input

1. Access `index.html` in a web browser.
2. Fill in the parameters for the GET form and click "Submit GET Request".
3. Fill in the parameters for the POST form and click "Submit POST Request".
4. Click the "Trigger Redirect Example" link.

Example Input (GET Form): Parameter 1: `MyValue1` Parameter 2: `MyValue2`

Example Input (POST Form): Parameter 1: `PostDataA` Parameter 2: `PostDataB`

Expected Output

1. After submitting GET request (e.g., with "MyValue1", "MyValue2"): A web page displaying tables with:

- **Request Method:** `GET`
- **Request Parameters:**

    - `param1`: `MyValue1`
    - `param2`: `MyValue2`
- **Request Headers:** (A list of various HTTP headers sent by the browser, e.g., `Host`, `User-Agent`, `Accept`, etc.)
- **Other Request Information:** (Details like `Request URI`, `Context Path`, `Query String` (e.g., `param1=MyValue1&param2=MyValue2`), `Remote Address`, etc.)
- A message indicating: "Response status code set to: 200 (OK)" and "A custom response header 'Custom-Header' has been set."
- A "Go Back to Form" link.

2. After submitting POST request (e.g., with "PostDataA", "PostDataB"): A web page displaying tables with similar information as the GET request, but with:

- **Request Method:** `POST`
- **Request Parameters:**
    - `param1`: `PostDataA`
    - `param2`: `PostDataB`
- `Query String` will likely be `null` or empty for POST requests as parameters are in the request body.

3. After clicking "Trigger Redirect Example": The browser will redirect back to `index.html`. The URL might change to `http://localhost:8080/YourWebApp/index.html?message=redirected`. The console of your servlet container (Tomcat) will show:

```
RedirectServlet: Redirecting to index.html...
```

Laboratory 6: Program that demonstrates the use of session management in Servlet

Title

Session Management in Servlets

Aim

To understand and implement session management in Servlets using `HttpSession` to maintain user-specific data across multiple requests within a single session.

Procedure

1. **Setup Dynamic Web Project:** Create a new Dynamic Web Project.
2. **Create Login Servlet (`LoginServlet`):**
    - This servlet will handle user login.
    - If login is successful, retrieve the `HttpSession` object using `request.getSession()`.
    - Store user-specific data (e.g., username, user ID) as attributes in the `HttpSession` object using `session.setAttribute()`.
    - Redirect the user to a welcome page or another servlet that displays session data.
3. **Create Welcome Servlet (`WelcomeServlet`):**
    - This servlet will be accessed after login.
    - Retrieve the `HttpSession` object using `request.getSession(false)` (to avoid creating a new session if one doesn't exist).
    - Retrieve session attributes using `session.getAttribute()`.
    - If no session or required attributes are found, redirect the user back to the login page.
4. **Create Logout Servlet (`LogoutServlet`):**
    - This servlet will handle user logout.
    - Retrieve the `HttpSession` object.
    - Invalidate the session using `session.invalidate()` to remove all session attributes and destroy the session.
    - Redirect the user back to the login page with a logout message.
5. **Create HTML Pages:**
    - `login.html`: Contains a form for username and password.
    - `welcome.html`: A simple page that links to the `WelcomeServlet`.
6. **Deployment:** Deploy the application to a servlet container.

Source Code

`login.html` (Login Form)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Login Page</title>
    <style>
        body { font-family: 'Inter', sans-serif; display: flex; justify-
content: center; align-items: center; min-height: 100vh; background-color:
#f0f2f5; margin: 0; }
```

```
        .container { background-color: #ffffff; padding: 30px; border-radius:
12px; box-shadow: 0 4px 15px rgba(0, 0, 0, 0.1); text-align: center; max-
width: 400px; width: 90%; }
        h1 { color: #333; margin-bottom: 25px; }
        .message { color: red; margin-bottom: 15px; font-weight: bold; }
        form { display: flex; flex-direction: column; gap: 15px; }
        label { text-align: left; font-weight: bold; color: #555; }
        input[type="text"], input[type="password"] { padding: 12px; border:
1px solid #ddd; border-radius: 8px; font-size: 16px; width: calc(100% -
24px); }
        input[type="submit"] { background-color: #007bff; color: white;
padding: 12px 20px; border: none; border-radius: 8px; cursor: pointer; font-
size: 18px; font-weight: bold; transition: background-color 0.3s ease; }
        input[type="submit"]:hover { background-color: #0056b3; }
    </style>
</head>
<body>
    <div class="container">
        <h1>User Login</h1>
        <%
            // Display message if redirected from LogoutServlet or
WelcomeServlet
            String message = request.getParameter("message");
            if (message != null && !message.isEmpty()) {
                out.println("<p class='message'>" + message + "</p>");
            }
        %>
        <form action="loginServlet" method="post">
            <label for="username">Username:</label>
            <input type="text" id="username" name="username" required>

            <label for="password">Password:</label>
            <input type="password" id="password" name="password" required>

            <input type="submit" value="Login">
        </form>
    </div>
</body>
</html>
```

welcome.html *(Link to Welcome Servlet)*

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Welcome Page</title>
    <style>
        body { font-family: 'Inter', sans-serif; display: flex; justify-
content: center; align-items: center; min-height: 100vh; background-color:
#f0f2f5; margin: 0; }
        .container { background-color: #ffffff; padding: 30px; border-radius:
12px; box-shadow: 0 4px 15px rgba(0, 0, 0, 0.1); text-align: center; max-
width: 400px; width: 90%; }
        h1 { color: #333; margin-bottom: 25px; }
        p { color: #555; font-size: 1.1em; margin-bottom: 20px; }
        a { display: inline-block; margin: 10px; padding: 10px 20px;
background-color: #007bff; color: white; text-decoration: none; border-
radius: 8px; transition: background-color 0.3s ease; }
        a.logout-button { background-color: #dc3545; }
        a:hover { background-color: #0056b3; }
        a.logout-button:hover { background-color: #c82333; }
    </style>
</head>
<body>
```

```
    <div class="container">
        <h1>Welcome!</h1>
        <p>Click below to view your personalized welcome message.</p>
        <a href="welcomeServlet">Go to Welcome Page</a>
    </div>
</body>
</html>
```

LoginServlet.java

```
// src/main/java/com/example/LoginServlet.java
package com.example;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

@WebServlet("/loginServlet")
public class LoginServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    protected void doPost(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        String username = request.getParameter("username");
        String password = request.getParameter("password");

        // Simple authentication logic (for demonstration purposes only)
        if ("user".equals(username) && "pass".equals(password)) {
            // Get or create a session
            HttpSession session = request.getSession(); // true by default,
creates new if not exists
            session.setAttribute("loggedInUser", username);
            session.setMaxInactiveInterval(30 * 60); // Session will expire
in 30 minutes (optional)

            System.out.println("User '" + username + "' logged in. Session
ID: " + session.getId());

            // Redirect to a welcome page or servlet
            response.sendRedirect("welcomeServlet");
        } else {
            // Invalid credentials, redirect back to login page with an error
message
            response.sendRedirect("login.html?message=Invalid Username or
Password!");
        }
    }
}
```

WelcomeServlet.java

```
// src/main/java/com/example/WelcomeServlet.java
package com.example;

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

```
@WebServlet("/welcomeServlet")
public class WelcomeServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

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

        response.setContentType("text/html");
        response.setCharacterEncoding("UTF-8");
        PrintWriter out = response.getWriter();

        // Get the existing session (do not create a new one if it doesn't
exist)
        HttpSession session = request.getSession(false);

        String username = null;
        if (session != null) {
            username = (String) session.getAttribute("loggedInUser");
        }

        if (username != null) {
            // User is logged in, display welcome message
            out.println("<!DOCTYPE html>");
            out.println("<html lang='en'>");
            out.println("<head>");
            out.println("    <meta charset='UTF-8'>");
            out.println("    <meta name='viewport' content='width=device-
width, initial-scale=1.0'>");
            out.println("    <title>Welcome</title>");
            out.println("    <style>");
            out.println("        body { font-family: 'Inter', sans-serif;
display: flex; justify-content: center; align-items: center; min-height:
100vh; background-color: #f0f2f5; margin: 0; }");
            out.println("        .container { background-color: #ffffff;
padding: 30px; border-radius: 12px; box-shadow: 0 4px 15px rgba(0, 0, 0,
0.1); text-align: center; max-width: 400px; width: 90%; }");
            out.println("        h1 { color: #333; margin-bottom: 25px; }");
            out.println("        p { color: #555; font-size: 1.1em; margin-
bottom: 20px; }");
            out.println("        a { display: inline-block; margin: 10px;
padding: 10px 20px; background-color: #dc3545; color: white; text-decoration:
none; border-radius: 8px; transition: background-color 0.3s ease; }");
            out.println("        a:hover { background-color: #c82333; }");
            out.println("    </style>");
            out.println("</head>");
            out.println("<body>");
            out.println("    <div class='container'>");
            out.println("        <h1>Welcome, " + username + "!</h1>");
            out.println("        <p>You are successfully logged in.</p>");
            out.println("        <a href='logoutServlet'>Logout</a>");
            out.println("    </div>");
            out.println("</body>");
            out.println("</html>");
        } else {
            // Session expired or not logged in, redirect to login page
            response.sendRedirect("login.html?message=Please log in to access
this page.");
        }
        out.close();
    }
}
```

LogoutServlet.java

```
// src/main/java/com/example/LogoutServlet.java
```

```
package com.example;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;

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

    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        HttpSession session = request.getSession(false); // Get existing
session, don't create new

        if (session != null) {
            String username = (String) session.getAttribute("loggedInUser");
            session.invalidate(); // Invalidate the session
            System.out.println("User '" + username + "' logged out. Session
invalidated.");
        }

        // Redirect to login page with a logout message
        response.sendRedirect("login.html?message=You have been successfully
logged out.");
    }
}
```

Input

1. Navigate to `login.html`.
2. Enter `username: user` and `password: pass`.
3. Click "Login".
4. On the `welcome.html` page, click "Go to Welcome Page".
5. On the personalized welcome page, click "Logout".
6. Try to access `welcomeServlet` directly after logout.

Expected Output

1. Initial `login.html`: A login form with "Username" and "Password" fields and a "Login" button.

2. After successful login (user: `user`, pass: `pass`): The browser redirects to `welcomeServlet`, which displays:

```
<!DOCTYPE html>
<html lang='en'>
<head>
    </head>
<body>
    <div class='container'>
        <h1>Welcome, user!</h1>
        <p>You are successfully logged in.</p>
        <a href='logoutServlet'>Logout</a>
    </div>
</body>
```

```
</html>
```

And in the server console:

```
User 'user' logged in. Session ID: [some_session_id]
```

3. After clicking "Logout": The browser redirects back to `login.html`, displaying:

```
<body>
    <div class="container">
        <h1>User Login</h1>
        <p class='message'>You have been successfully logged out.</p>
        <form action="loginServlet" method="post">
            </form>
    </div>
</body>
```

And in the server console:

```
User 'user' logged out. Session invalidated.
```

4. Attempting to access `welcomeServlet` directly after logout: The browser redirects back to `login.html`, displaying:

```
<body>
    <div class="container">
        <h1>User Login</h1>
        <p class='message'>Please log in to access this page.</p>
        <form action="loginServlet" method="post">
            </form>
    </div>
</body>
```

Laboratory 7: Web Applications using JSP

Title

Developing Web Applications Using JSP (JavaServer Pages)

Aim

To understand the fundamentals of JavaServer Pages (JSP) and develop a basic web application that combines HTML with Java code to generate dynamic content.

Procedure

1. **Setup Dynamic Web Project:** Create a new Dynamic Web Project in your IDE.
2. **Create a JSP File:**
    - Create a `.jsp` file (e.g., `index.jsp` or `hello.jsp`) in the `WebContent` folder.
    - Include HTML markup.
    - Use JSP elements:
        - **Directives (`<%@ ... %>`):** For page settings (e.g., `page`, `taglib`, `include`).
        - **Scriptlets (`<% ... %>`):** To embed Java code.
        - **Expressions (`<%= ... %>`):** To output Java variable values or method calls directly into the HTML.
        - **Declarations (`<%! ... %>`):** To declare instance variables or methods in the generated servlet.
        - **Comments (`<%-- ... --%>`):** JSP comments, ignored by the client.
3. **Accessing Request Parameters:** Demonstrate how to retrieve request parameters within JSP using `request.getParameter()`.
4. **Deployment:** Deploy the web application to your servlet container (e.g., Tomcat) and access it through a web browser.

Source Code

index.jsp

```
<%-- WebContent/index.jsp --%>
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>JSP Hello World</title>
    <style>
        body { font-family: 'Inter', sans-serif; display: flex; justify-
content: center; align-items: center; min-height: 100vh; background-color:
#f0f2f5; margin: 0; }
        .container { background-color: #ffffff; padding: 30px; border-radius:
12px; box-shadow: 0 4px 15px rgba(0, 0, 0, 0.1); text-align: center; max-
width: 500px; width: 90%; }
        h1 { color: #333; margin-bottom: 25px; }
        p { color: #555; font-size: 1.1em; margin-bottom: 20px; }
        form { display: flex; flex-direction: column; gap: 15px; margin-top:
20px; }
        label { text-align: left; font-weight: bold; color: #555; }
        input[type="text"] { padding: 12px; border: 1px solid #ddd; border-
radius: 8px; font-size: 16px; width: calc(100% - 24px); }
```

```
        input[type="submit"] { background-color: #007bff; color: white;
padding: 12px 20px; border: none; border-radius: 8px; cursor: pointer; font-
size: 18px; font-weight: bold; transition: background-color 0.3s ease; }
        input[type="submit"]:hover { background-color: #0056b3; }
        .current-time { font-style: italic; color: #777; margin-top: 15px; }
    </style>
</head>
<body>
    <div class="container">
        <h1>Welcome to JSP World!</h1>

        <%-- Scriptlet to get current time --%>
        <%
            java.util.Date now = new java.util.Date();
            String currentTime = now.toString();
        %>
        <p class="current-time">Current Server Time: <%= currentTime %></p>

        <%-- Expression to display a simple message --%>
        <p>This page demonstrates basic JSP functionality.</p>

        <%-- Form to submit a name to this same JSP --%>
        <form action="index.jsp" method="get">
            <label for="userName">Enter your name:</label>
            <input type="text" id="userName" name="userName"
placeholder="e.g., Jane Doe" required>
            <input type="submit" value="Greet Me">
        </form>

        <%-- Scriptlet to process form submission --%>
        <%
            String userName = request.getParameter("userName");
            if (userName != null && !userName.trim().isEmpty()) {
        %>
                <p style="margin-top: 25px; font-size: 1.2em; color:
#28a745;">Hello, <%= userName %>! Nice to see you.</p>
        <%
            }
        %>

        <%-- JSP Comment: This comment will not appear in the client's
browser source. --%>
    </div>
</body>
</html>
```

Input

1. Access `index.jsp` in a web browser.
2. Enter a name in the text field and click "Greet Me".

Example Input: Text field value: `Alice`

Expected Output

1. Initial access to `index.jsp`: A web page displaying:

- A heading "Welcome to JSP World!"
- "Current Server Time: [current date and time]"
- "This page demonstrates basic JSP functionality."
- A form with a text input for "Enter your name:" and a "Greet Me" button.

2. After submitting "Alice": The same web page reloads, but now with an additional greeting message below the form:

```
<form action="index.jsp" method="get">
            </form>

        <p style="margin-top: 25px; font-size: 1.2em; color: #28a745;">Hello,
Alice! Nice to see you.</p>
```

Visually, it would show "Hello, Alice! Nice to see you." in green text below the form.

Laboratory 8: Include Directive JSP: include Action

Title

JSP Include Mechanisms: Directive and Action

Aim

To understand and differentiate between the JSP `include` directive (`<%@ include file="..." %>`) and the JSP `include` action (`<jsp:include page="..." />`), and to demonstrate their usage for code reusability in web applications.

Procedure

1. **Setup Dynamic Web Project:** Create a new Dynamic Web Project.
2. **Create Reusable JSP Fragments:**
    - `header.jspf` (or `header.jsp`): Contains common header HTML (e.g., doctype, head, start of body, navigation).
    - `footer.jspf` (or `footer.jsp`): Contains common footer HTML (e.g., copyright, closing body/html tags).
    - `menu.jsp`: Contains a simple navigation menu.
3. **Create Main JSP Pages:**
    - `directiveInclude.jsp`: Use the `<%@ include file="..." %>` directive to include `header.jspf` and `footer.jspf`. Demonstrate that the included content is merged at translation time.
    - `actionInclude.jsp`: Use the `<jsp:include page="..." />` action to include `menu.jsp`. Demonstrate that the included content is processed at request time. Show how to pass parameters to the included page using `<jsp:param>`.
4. **Deployment:** Deploy the application and access the main JSP pages. View the page source to observe the difference between directive and action includes.

Source Code

`header.jspf` (Fragment for Directive Include)

```
<%-- WebContent/WEB-INF/jspf/header.jspf --%>
<%-- It's good practice to put fragments in WEB-INF to prevent direct access
--%>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>JSP Include Demo</title>
    <style>
        body { font-family: 'Inter', sans-serif; margin: 0; padding: 0;
background-color: #f4f7f6; color: #333; }
        .main-header { background-color: #2c3e50; color: white; padding: 20px
0; text-align: center; }
        .main-header h1 { margin: 0; }
        .content { padding: 30px; max-width: 960px; margin: 20px auto;
background-color: #ffffff; border-radius: 8px; box-shadow: 0 2px 10px rgba(0,
0, 0, 0.05); }
        .footer { background-color: #34495e; color: white; text-align:
center; padding: 15px 0; position: fixed; bottom: 0; width: 100%; }
        .menu-nav { background-color: #ecf0f1; padding: 10px 0; text-align:
center; border-bottom: 1px solid #bdc3c7; margin-bottom: 20px; border-radius:
5px; }
```

```
        .menu-nav ul { list-style: none; padding: 0; margin: 0; display:
flex; justify-content: center; gap: 20px; }
        .menu-nav ul li a { text-decoration: none; color: #34495e; font-
weight: bold; padding: 8px 15px; border-radius: 5px; transition: background-
color 0.3s ease; }
        .menu-nav ul li a:hover { background-color: #bdc3c7; }
    </style>
</head>
<body>
    <header class="main-header">
        <h1>JSP Include Demo</h1>
    </header>
```

footer.jspf **(Fragment for Directive Include)**

```
<%-- WebContent/WEB-INF/jspf/footer.jspf --%>
    <footer class="footer">
        <p>&copy; <%= new java.util.Date().getYear() + 1900 %> JSP Include
Demo. All rights reserved.</p>
    </footer>
</body>
</html>
```

menu.jsp **(Fragment for Action Include)**

```
<%-- WebContent/menu.jsp --%>
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<nav class="menu-nav">
    <ul>
        <li><a href="directiveInclude.jsp">Directive Include Page</a></li>
        <li><a href="actionInclude.jsp">Action Include Page</a></li>
        <li><a href="#">About Us</a></li>
        <li><a href="#">Contact</a></li>
    </ul>
    <%
        // Demonstrate receiving parameter from jsp:include action
        String currentPage = request.getParameter("currentPage");
        if (currentPage != null) {
            out.println("<p style='font-size: 0.9em; color: #7f8c8d;'>Menu
included by: " + currentPage + "</p>");
        }
    %>
</nav>
```

directiveInclude.jsp **(Using Directive Include)**

```
<%-- WebContent/directiveInclude.jsp --%>
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%-- Include the header at translation time --%>
<%@ include file="/WEB-INF/jspf/header.jspf" %>

<div class="content">
    <h2>Page using JSP Include Directive</h2>
    <p>This content is part of <code>directiveInclude.jsp</code>.</p>
    <p>The header and footer are included using the <code>&lt;%@ include
file="..." %&gt;</code> directive.</p>
    <p>This means their content is merged into this JSP file *before* it is
compiled into a servlet. If you view the page source in your browser, you
will see the combined HTML.</p>

    <%-- Include menu using action --%>
    <jsp:include page="menu.jsp">
        <jsp:param name="currentPage" value="DirectiveIncludePage"/>
    </jsp:include>
</div>
```

```
<%-- Include the footer at translation time --%>
<%@ include file="/WEB-INF/jspf/footer.jspf" %>
```

actionInclude.jsp (Using Action Include)

```
<%-- WebContent/actionInclude.jsp --%>
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%-- Include the header at translation time --%>
<%@ include file="/WEB-INF/jspf/header.jspf" %>

<div class="content">
    <h2>Page using JSP Include Action</h2>
    <p>This content is part of <code>actionInclude.jsp</code>.</p>
    <p>The menu below is included using the <code>&lt;jsp:include page="..."
/&gt;</code> action.</p>
    <p>This means the menu content is included *at request time* (when the
page is served to the client). The included page is executed as a separate
servlet and its output is merged into this page. You can also pass parameters
to the included page.</p>

    <%-- Include menu using action, passing a parameter --%>
    <jsp:include page="menu.jsp">
        <jsp:param name="currentPage" value="ActionIncludePage"/>
    </jsp:include>

    <p>Notice how the menu might display which page included it,
demonstrating parameter passing.</p>
</div>

<%-- Include the footer at translation time --%>
<%@ include file="/WEB-INF/jspf/footer.jspf" %>
```

Input

1. Access `directiveInclude.jsp` in a web browser.
2. Access `actionInclude.jsp` in a web browser.
3. View the page source for both pages.

Expected Output

1. When accessing `directiveInclude.jsp`: A complete HTML page will be rendered, including the content from `header.jspf`, the specific content of `directiveInclude.jsp`, the menu from `menu.jsp`, and the content from `footer.jspf`.

- **Visually:** A header ("JSP Include Demo"), the main content for the directive page, a navigation menu, and a footer (e.g., copyright). The menu will show "Menu included by: DirectiveIncludePage".
- **Viewing Page Source (important for understanding):** The browser's "View Page Source" will show *all* the HTML, including the content of `header.jspf` and `footer.jspf` directly embedded, as if they were part of the original `directiveInclude.jsp` file. The `menu.jsp` content will also be present, as it's included via action.

2. When accessing `actionInclude.jsp`: A complete HTML page will be rendered, similar to the above.

- **Visually:** A header, the main content for the action page, a navigation menu, and a footer. The menu will show "Menu included by: ActionIncludePage".
- **Viewing Page Source (important for understanding):** The browser's "View Page Source" will show *all* the HTML, including the content of `header.jspf` and `footer.jspf` directly embedded. The content from `menu.jsp` will also be present, but conceptually, it was generated at request time.

Key Difference to Observe in Source: While both will show the full HTML, the conceptual difference is how and when the content is merged. The directive is a compile-time inclusion, making the included code part of the main JSP's servlet. The action is a runtime inclusion, where the included page is executed separately and its output is inserted.

Laboratory 9: Create a JSP based Web application which allows the user to edit his/her database Information

Title

JSP-based Database Information Editor

Aim

To develop a JSP-based web application that enables users to view, add, update, and delete their information stored in a database, combining JSP for presentation, Servlets for control, and JDBC for database interaction (MVC pattern).

Procedure

1. **Database Setup:** Create a database and a table (e.g., `users` with `id`, `username`, `email`, `password` columns).
2. **Model (POJO):** Create a `User` Java class to represent user data.
3. **DAO (Data Access Object):** Create a `UserDAO` Java class to encapsulate all JDBC operations (add, retrieve, update, delete users).
4. **Controller (Servlet):** Create a `UserServlet` that acts as the controller. This servlet will:
    - Receive requests from JSP pages (e.g., for listing, adding, editing, deleting).
    - Call appropriate methods in the `UserDAO` to interact with the database.
    - Set data as request or session attributes.
    - Forward requests to the appropriate JSP views using `RequestDispatcher`.
5. **Views (JSP Pages):**
    - `listUsers.jsp`: Displays a table of all users with links/buttons for edit and delete.
    - `addUser.jsp`: Contains a form to add a new user.
    - `editUser.jsp`: Contains a form pre-populated with existing user data for editing.
    - `error.jsp`: Displays error messages.
6. **Deployment:** Deploy the web application to a servlet container.

Source Code

`User.java` (Model)

```
// src/main/java/com/example/model/User.java
package com.example.model;

public class User {
    private int id;
    private String username;
    private String email;
    private String password; // In a real app, store hashed passwords

    public User(int id, String username, String email, String password) {
        this.id = id;
        this.username = username;
        this.email = email;
        this.password = password;
    }

    public User(String username, String email, String password) {
        this.username = username;
        this.email = email;
        this.password = password;
```

```
    }

    // Getters
    public int getId() { return id; }
    public String getUsername() { return username; }
    public String getEmail() { return email; }
    public String getPassword() { return password; }

    // Setters
    public void setId(int id) { this.id = id; }
    public void setUsername(String username) { this.username = username; }
    public void setEmail(String email) { this.email = email; }
    public void setPassword(String password) { this.password = password; } //
For demonstration

    @Override
    public String toString() {
        return "User [id=" + id + ", username=" + username + ", email=" +
email + "]";
    }
}
```

UserDAO.java (Data Access Object)

```
// src/main/java/com/example/dao/UserDAO.java
package com.example.dao;

import com.example.model.User;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.ArrayList;
import java.util.List;

public class UserDAO {

    private static final String JDBC_DRIVER = "com.mysql.cj.jdbc.Driver";
    private static final String DB_URL =
"jdbc:mysql://localhost:3306/user_db"; // Ensure 'user_db' exists
    private static final String USER = "root";
    private static final String PASS = "password";

    public UserDAO() {
        try {
            Class.forName(JDBC_DRIVER);
            createTableIfNotExists();
        } catch (ClassNotFoundException e) {
            System.err.println("JDBC Driver not found: " + e.getMessage());
        }
    }

    private Connection getConnection() throws SQLException {
        return DriverManager.getConnection(DB_URL, USER, PASS);
    }

    private void createTableIfNotExists() {
        String sql = "CREATE TABLE IF NOT EXISTS users (" +
                     "id INT AUTO_INCREMENT PRIMARY KEY," +
                     "username VARCHAR(255) UNIQUE NOT NULL," +
                     "email VARCHAR(255) NOT NULL," +
                     "password VARCHAR(255) NOT NULL)"; // In real app, store
hashed passwords
        try (Connection conn = getConnection();
             PreparedStatement pstmt = conn.prepareStatement(sql)) {
```

```
            pstmt.executeUpdate();
            System.out.println("Table 'users' created or already exists.");
        } catch (SQLException e) {
            System.err.println("Error creating users table: " +
e.getMessage());
        }
    }

    public void addUser(User user) throws SQLException {
        String sql = "INSERT INTO users (username, email, password) VALUES
(?, ?, ?)";
        try (Connection conn = getConnection();
             PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setString(1, user.getUsername());
            pstmt.setString(2, user.getEmail());
            pstmt.setString(3, user.getPassword());
            pstmt.executeUpdate();
        }
    }

    public List<User> getAllUsers() throws SQLException {
        List<User> users = new ArrayList<>();
        String sql = "SELECT id, username, email, password FROM users";
        try (Connection conn = getConnection();
             PreparedStatement pstmt = conn.prepareStatement(sql);
             ResultSet rs = pstmt.executeQuery()) {
            while (rs.next()) {
                int id = rs.getInt("id");
                String username = rs.getString("username");
                String email = rs.getString("email");
                String password = rs.getString("password");
                users.add(new User(id, username, email, password));
            }
        }
        return users;
    }

    public User getUserById(int id) throws SQLException {
        String sql = "SELECT id, username, email, password FROM users WHERE
id = ?";
        try (Connection conn = getConnection();
             PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setInt(1, id);
            try (ResultSet rs = pstmt.executeQuery()) {
                if (rs.next()) {
                    String username = rs.getString("username");
                    String email = rs.getString("email");
                    String password = rs.getString("password");
                    return new User(id, username, email, password);
                }
            }
        }
        return null;
    }

    public void updateUser(User user) throws SQLException {
        String sql = "UPDATE users SET username = ?, email = ?, password = ?
WHERE id = ?";
        try (Connection conn = getConnection();
             PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setString(1, user.getUsername());
            pstmt.setString(2, user.getEmail());
            pstmt.setString(3, user.getPassword());
            pstmt.setInt(4, user.getId());
            pstmt.executeUpdate();
        }
```

```
    }

    public void deleteUser(int id) throws SQLException {
        String sql = "DELETE FROM users WHERE id = ?";
        try (Connection conn = getConnection();
             PreparedStatement pstmt = conn.prepareStatement(sql)) {
            pstmt.setInt(1, id);
            pstmt.executeUpdate();
        }
    }
}
```

UserServlet.java **(Controller)**

```
// src/main/java/com/example/controller/UserServlet.java
package com.example.controller;

import com.example.dao.UserDAO;
import com.example.model.User;
import java.io.IOException;
import java.sql.SQLException;
import java.util.List;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/UserServlet")
public class UserServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;
    private UserDAO userDAO;

    public void init() {
        userDAO = new UserDAO();
    }

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

    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        String action = request.getParameter("action");

        if (action == null) {
            action = "list"; // Default action
        }

        try {
            switch (action) {
                case "new":
                    showNewForm(request, response);
                    break;
                case "insert":
                    insertUser(request, response);
                    break;
                case "delete":
                    deleteUser(request, response);
                    break;
                case "edit":
                    showEditForm(request, response);
```

```
                    break;
                case "update":
                    updateUser(request, response);
                    break;
                case "list":
                default:
                    listUsers(request, response);
                    break;
            }
        } catch (SQLException ex) {
            throw new ServletException(ex);
        }
    }

    private void listUsers(HttpServletRequest request, HttpServletResponse
response)
            throws SQLException, IOException, ServletException {
        List<User> listUser = userDAO.getAllUsers();
        request.setAttribute("listUser", listUser);
        RequestDispatcher dispatcher =
request.getRequestDispatcher("listUsers.jsp");
        dispatcher.forward(request, response);
    }

    private void showNewForm(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        RequestDispatcher dispatcher =
request.getRequestDispatcher("addUser.jsp");
        dispatcher.forward(request, response);
    }

    private void insertUser(HttpServletRequest request, HttpServletResponse
response)
            throws SQLException, IOException {
        String username = request.getParameter("username");
        String email = request.getParameter("email");
        String password = request.getParameter("password");
        User newUser = new User(username, email, password);
        userDAO.addUser(newUser);
        response.sendRedirect("UserServlet?action=list");
    }

    private void showEditForm(HttpServletRequest request, HttpServletResponse
response)
            throws SQLException, ServletException, IOException {
        int id = Integer.parseInt(request.getParameter("id"));
        User existingUser = userDAO.getUserById(id);
        RequestDispatcher dispatcher =
request.getRequestDispatcher("editUser.jsp");
        request.setAttribute("user", existingUser);
        dispatcher.forward(request, response);
    }

    private void updateUser(HttpServletRequest request, HttpServletResponse
response)
            throws SQLException, IOException {
        int id = Integer.parseInt(request.getParameter("id"));
        String username = request.getParameter("username");
        String email = request.getParameter("email");
        String password = request.getParameter("password"); // In real app,
handle password change carefully

        User user = new User(id, username, email, password);
        userDAO.updateUser(user);
        response.sendRedirect("UserServlet?action=list");
```

```
    }

    private void deleteUser(HttpServletRequest request, HttpServletResponse
response)
            throws SQLException, IOException {
        int id = Integer.parseInt(request.getParameter("id"));
        userDAO.deleteUser(id);
        response.sendRedirect("UserServlet?action=list");
    }
}
```

listUsers.jsp **(View: List Users)**

```
<%-- WebContent/listUsers.jsp --%>
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>User List</title>
    <style>
        body { font-family: 'Inter', sans-serif; background-color: #f0f2f5;
margin: 0; padding: 20px; box-sizing: border-box; }
        .container { background-color: #ffffff; padding: 30px; border-radius:
12px; box-shadow: 0 4px 15px rgba(0, 0, 0, 0.1); max-width: 900px; margin:
30px auto; }
        h1 { color: #333; text-align: center; margin-bottom: 30px; }
        table { width: 100%; border-collapse: collapse; margin-top: 20px; }
        th, td { border: 1px solid #ddd; padding: 12px; text-align: left; }
        th { background-color: #007bff; color: white; }
        tr:nth-child(even) { background-color: #f9f9f9; }
        .action-links a { text-decoration: none; padding: 6px 12px; border-
radius: 5px; margin-right: 5px; }
        .action-links a.edit { background-color: #ffc107; color: #333; }
        .action-links a.delete { background-color: #dc3545; color: white; }
        .action-links a.edit:hover { background-color: #e0a800; }
        .action-links a.delete:hover { background-color: #c82333; }
        .add-button { display: inline-block; margin-bottom: 20px; padding:
10px 20px; background-color: #28a745; color: white; text-decoration: none;
border-radius: 8px; transition: background-color 0.3s ease; }
        .add-button:hover { background-color: #218838; }
        .no-users { text-align: center; color: #777; font-style: italic; }
    </style>
</head>
<body>
    <div class="container">
        <h1>User Management</h1>
        <a href="UserServlet?action=new" class="add-button">Add New User</a>

        <c:if test="${empty listUser}">
            <p class="no-users">No users found in the database.</p>
        </c:if>
        <c:if test="${not empty listUser}">
            <table>
                <thead>
                    <tr>
                        <th>ID</th>
                        <th>Username</th>
                        <th>Email</th>
                        <th>Actions</th>
                    </tr>
                </thead>
                <tbody>
```

```
                    <c:forEach var="user" items="${listUser}">
                        <tr>
                            <td><c:out value="${user.id}" /></td>
                            <td><c:out value="${user.username}" /></td>
                            <td><c:out value="${user.email}" /></td>
                            <td class="action-links">
                                <a href="UserServlet?action=edit&id=<c:out
value='${user.id}' />" class="edit">Edit</a>
                                <a href="UserServlet?action=delete&id=<c:out
value='${user.id}' />" class="delete" onclick="return confirm('Are you sure
you want to delete this user?');">Delete</a>
                            </td>
                        </tr>
                    </c:forEach>
                </tbody>
            </table>
        </c:if>
    </div>
</body>
</html>
```

addUser.jsp *(View: Add User Form)*

```
<%-- WebContent/addUser.jsp --%>
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Add New User</title>
    <style>
        body { font-family: 'Inter', sans-serif; display: flex; justify-
content: center; align-items: center; min-height: 100vh; background-color:
#f0f2f5; margin: 0; }
        .container { background-color: #ffffff; padding: 30px; border-radius:
12px; box-shadow: 0 4px 15px rgba(0, 0, 0, 0.1); text-align: center; max-
width: 500px; width: 90%; }
        h1 { color: #333; margin-bottom: 25px; }
        form { display: flex; flex-direction: column; gap: 15px; }
        label { text-align: left; font-weight: bold; color: #555; }
        input[type="text"], input[type="email"], input[type="password"] {
padding: 12px; border: 1px solid #ddd; border-radius: 8px; font-size: 16px;
width: calc(100% - 24px); }
        input[type="submit"] { background-color: #28a745; color: white;
padding: 12px 20px; border: none; border-radius: 8px; cursor: pointer; font-
size: 18px; font-weight: bold; transition: background-color 0.3s ease; }
        input[type="submit"]:hover { background-color: #218838; }
        .back-link { display: inline-block; margin-top: 20px; padding: 10px
20px; background-color: #6c757d; color: white; text-decoration: none; border-
radius: 8px; transition: background-color 0.3s ease; }
        .back-link:hover { background-color: #5a6268; }
    </style>
</head>
<body>
    <div class="container">
        <h1>Add New User</h1>
        <form action="UserServlet?action=insert" method="post">
            <label for="username">Username:</label>
            <input type="text" id="username" name="username" required>

            <label for="email">Email:</label>
            <input type="email" id="email" name="email" required>

            <label for="password">Password:</label>
```

```
            <input type="password" id="password" name="password" required>

            <input type="submit" value="Add User">
        </form>
        <a href="UserServlet?action=list" class="back-link">Back to List</a>
    </div>
</body>
</html>
```

editUser.jsp *(View: Edit User Form)*

```
<%-- WebContent/editUser.jsp --%>
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Edit User</title>
    <style>
        body { font-family: 'Inter', sans-serif; display: flex; justify-
content: center; align-items: center; min-height: 100vh; background-color:
#f0f2f5; margin: 0; }
        .container { background-color: #ffffff; padding: 30px; border-radius:
12px; box-shadow: 0 4px 15px rgba(0, 0, 0, 0.1); text-align: center; max-
width: 500px; width: 90%; }
        h1 { color: #333; margin-bottom: 25px; }
        form { display: flex; flex-direction: column; gap: 15px; }
        label { text-align: left; font-weight: bold; color: #555; }
        input[type="text"], input[type="email"], input[type="password"] {
padding: 12px; border: 1px solid #ddd; border-radius: 8px; font-size: 16px;
width: calc(100% - 24px); }
        input[type="submit"] { background-color: #ffc107; color: #333;
padding: 12px 20px; border: none; border-radius: 8px; cursor: pointer; font-
size: 18px; font-weight: bold; transition: background-color 0.3s ease; }
        input[type="submit"]:hover { background-color: #e0a800; }
        .back-link { display: inline-block; margin-top: 20px; padding: 10px
20px; background-color: #6c757d; color: white; text-decoration: none; border-
radius: 8px; transition: background-color 0.3s ease; }
        .back-link:hover { background-color: #5a6268; }
    </style>
</head>
<body>
    <div class="container">
        <h1>Edit User</h1>
        <c:if test="${user == null}">
            <p style="color: red;">User not found!</p>
            <a href="UserServlet?action=list" class="back-link">Back to
List</a>
        </c:if>
        <c:if test="${user != null}">
            <form action="UserServlet?action=update" method="post">
                <input type="hidden" name="id" value="<c:out
value='${user.id}' />">

                <label for="username">Username:</label>
                <input type="text" id="username" name="username"
value="<c:out value='${user.username}' />" required>

                <label for="email">Email:</label>
                <input type="email" id="email" name="email" value="<c:out
value='${user.email}' />" required>
```

```
                <label for="password">Password (leave blank to keep
current):</label>
                <input type="password" id="password" name="password"
value="<c:out value='${user.password}' />">

                <input type="submit" value="Update User">
            </form>
            <a href="UserServlet?action=list" class="back-link">Back to
List</a>
        </c:if>
    </div>
</body>
</html>
```

Input

1. Access `UserServlet` (or `UserServlet?action=list`) in a web browser.
2. Click "Add New User", fill the form, and submit.
3. Observe the user in the list.
4. Click "Edit" for a user, modify details, and submit.
5. Click "Delete" for a user and confirm.

Example Input Sequence:

- **Add User:**
  - Username: `testuser`
  - Email: `test@example.com`
  - Password: `password123`
- **Edit User (for `testuser`):**
  - Change Username to: `updated_user`
  - Change Email to: `updated@example.com`
  - (Keep password same or change)
- **Delete User:** Click delete for `updated_user`.

Expected Output

1. Initial `listUsers.jsp`: A table displaying "No users found in the database." and an "Add New User" button.

2. After adding `testuser`: The `listUsers.jsp` page reloads, now showing a table with:

- ID: `1` (or next available ID)
- Username: `testuser`
- Email: `test@example.com`
- Actions: "Edit" and "Delete" links.

3. After editing `testuser` to `updated_user`: The `listUsers.jsp` page reloads, showing:

- ID: `1`
- Username: `updated_user`
- Email: `updated@example.com`
- Actions: "Edit" and "Delete" links.

4. After deleting `updated_user`: The `listUsers.jsp` page reloads, showing "No users found in the database." again.

Laboratory 10: An EJB application that demonstrates Session Bean- Stateless Bean

Title

EJB Session Bean: Stateless Bean Demonstration

Aim

To understand and implement a Stateless Session Bean in Enterprise JavaBeans (EJB) to provide business logic that does not maintain conversational state with the client.

Procedure

1. **Setup Environment:** Install a Java EE application server (e.g., WildFly, GlassFish, Open Liberty) and an IDE (e.g., Eclipse with JBoss Tools, IntelliJ IDEA Ultimate).
2. **Create an EJB Project:** Create a new EJB Project in your IDE.
3. **Define Business Interface:** Create a remote or local interface for the Stateless Session Bean. This interface declares the business methods.
4. **Implement Stateless Session Bean:**
    - Create a Java class that implements the business interface.
    - Annotate the class with `@Stateless`.
    - Implement the business methods.
5. **Create a Client Application:**
    - Create a separate Java application (can be a simple Java SE application or a web application).
    - Use JNDI (Java Naming and Directory Interface) lookup to obtain a reference to the Stateless Session Bean.
    - Invoke the business methods on the bean.
6. **Package and Deploy:** Package the EJB into a `.jar` file (EJB module) and the client (if a web app, into a `.war` file) and deploy them to the application server.

Source Code

`Calculator.java` (Business Interface)

```
// src/main/java/com/example/ejb/Calculator.java
package com.example.ejb;

import javax.ejb.Remote; // Or javax.ejb.Local for local interface

// @Remote annotation makes this interface accessible from remote clients
// If using @Local, it's only accessible within the same application (JVM)
@Remote
public interface Calculator {
    int add(int a, int b);
    int subtract(int a, int b);
}
```

`CalculatorBean.java` (Stateless Session Bean Implementation)

```
// src/main/java/com/example/ejb/CalculatorBean.java
package com.example.ejb;

import javax.ejb.Stateless;

// @Stateless annotation marks this as a Stateless Session Bean
@Stateless
```

```
public class CalculatorBean implements Calculator {

    public CalculatorBean() {
        // Constructor for the bean
        System.out.println("CalculatorBean instance created (Stateless).");
    }

    @Override
    public int add(int a, int b) {
        System.out.println("CalculatorBean: Performing addition of " + a + "
and " + b);
        return a + b;
    }

    @Override
    public int subtract(int a, int b) {
        System.out.println("CalculatorBean: Performing subtraction of " + a +
" and " + b);
        return a - b;
    }
}
```

EJBClient.java *(Client Application - Java SE)*

```
// src/main/java/com/example/client/EJBClient.java
package com.example.client;

import com.example.ejb.Calculator;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import java.util.Properties;

public class EJBClient {

    public static void main(String[] args) {
        Context initialContext = null;
        try {
            // Setup JNDI properties for connecting to the application server
            // These properties depend on your application server (e.g.,
WildFly, GlassFish)
            Properties jndiProperties = new Properties();
            jndiProperties.put(Context.INITIAL_CONTEXT_FACTORY,
"org.jboss.naming.remote.client.InitialContextFactory"); // For WildFly
            jndiProperties.put(Context.PROVIDER_URL, "http-
remoting://localhost:8080"); // For WildFly, change port if needed
            // jndiProperties.put(Context.SECURITY_PRINCIPAL,
"your_username"); // If security is enabled
            // jndiProperties.put(Context.SECURITY_CREDENTIALS,
"your_password"); // If security is enabled

            initialContext = new InitialContext(jndiProperties);

            // JNDI lookup name for a remote EJB
            // The format is typically:
            // ejb:<app-name>/<module-name>/<bean-name>!<fully-qualified-
remote-interface-name>
            // For a simple standalone EJB module, app-name is usually empty
string ""
            // module-name is the name of your EJB JAR file (e.g., "ejb-demo-
1.0-SNAPSHOT")
            // bean-name is the simple class name of your bean (e.g.,
"CalculatorBean")
            // interface-name is the fully qualified name of your remote
interface (e.g., "com.example.ejb.Calculator")
```

```
            String lookupName = "ejb:/ejb-stateless-
demo/CalculatorBean!com.example.ejb.Calculator";
            // Adjust "ejb-stateless-demo" to your actual EJB module name
(e.g., the name of your EJB JAR)

            System.out.println("Looking up EJB: " + lookupName);
            Calculator calculator = (Calculator)
initialContext.lookup(lookupName);

            System.out.println("EJB lookup successful. Invoking methods...");

            // Invoke business methods
            int sum = calculator.add(100, 50);
            System.out.println("100 + 50 = " + sum);

            int difference = calculator.subtract(200, 75);
            System.out.println("200 - 75 = " + difference);

            // Stateless beans are pooled, so the same instance might be
reused,
            // but no conversational state is maintained.
            System.out.println("\nInvoking again to show stateless nature (no
state carried over):");
            sum = calculator.add(5, 3);
            System.out.println("5 + 3 = " + sum);

        } catch (NamingException e) {
            System.err.println("JNDI NamingException: " + e.getMessage());
            e.printStackTrace();
        } finally {
            if (initialContext != null) {
                try {
                    initialContext.close();
                } catch (NamingException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

Input

No direct user input. The client application invokes methods with hardcoded values.

Expected Output

Application Server Console (when CalculatorBean instances are created/used):

```
CalculatorBean instance created (Stateless).
CalculatorBean: Performing addition of 100 and 50
CalculatorBean: Performing subtraction of 200 and 75
CalculatorBean: Performing addition of 5 and 3
```

(Note: The "CalculatorBean instance created" message might appear multiple times or only once depending on the server's pooling strategy, but it signifies the creation of a bean instance.)

Client Application Console (`java EJBClient`):

```
Looking up EJB: ejb:/ejb-stateless-
demo/CalculatorBean!com.example.ejb.Calculator
```

```
EJB lookup successful. Invoking methods...
100 + 50 = 150
200 - 75 = 125

Invoking again to show stateless nature (no state carried over):
5 + 3 = 8
```

Laboratory 11: An EJB application that demonstrates Session Bean- Stateful Bean

Title

EJB Session Bean: Stateful Bean Demonstration

Aim

To understand and implement a Stateful Session Bean in Enterprise JavaBeans (EJB) to provide business logic that maintains conversational state with a specific client across multiple method invocations.

Procedure

1. **Setup Environment:** Use a Java EE application server (e.g., WildFly, GlassFish, Open Liberty).
2. **Create an EJB Project:** Create a new EJB Project.
3. **Define Business Interface:** Create a remote or local interface for the Stateful Session Bean.
4. **Implement Stateful Session Bean:**
    - Create a Java class that implements the business interface.
    - Annotate the class with `@Stateful`.
    - Declare instance variables to hold the conversational state.
    - Implement business methods that modify or use this state.
    - Optionally, use `@PostConstruct`, `@PreDestroy`, `@Remove` annotations for lifecycle management.
5. **Create a Client Application:**
    - Obtain a reference to the Stateful Session Bean using JNDI lookup.
    - Invoke methods on the bean multiple times to demonstrate state persistence within the session.
    - Call a method annotated with `@Remove` (if applicable) or simply let the session time out to destroy the bean instance.
6. **Package and Deploy:** Package the EJB into a `.jar` and the client (if a web app) and deploy them to the application server.

Source Code

`ShoppingCart.java` (Business Interface)

```
// src/main/java/com/example/ejb/ShoppingCart.java
package com.example.ejb;

import javax.ejb.Remote; // Or javax.ejb.Local

@Remote
public interface ShoppingCart {
    void addItem(String item);
    void removeItem(String item);
    java.util.List<String> getItems();
    void checkout(); // Method to remove the bean instance
}
```

`ShoppingCartBean.java` (Stateful Session Bean Implementation)

```
// src/main/java/com/example/ejb/ShoppingCartBean.java
package com.example.ejb;
```

```
import javax.annotation.PostConstruct;
import javax.annotation.PreDestroy;
import javax.ejb.Remove;
import javax.ejb.Stateful;
import java.util.ArrayList;
import java.util.List;

// @Stateful annotation marks this as a Stateful Session Bean
@Stateful
public class ShoppingCartBean implements ShoppingCart {

    // Instance variable to hold the conversational state
    private List<String> items;

    @PostConstruct
    public void initialize() {
        items = new ArrayList<>();
        System.out.println("ShoppingCartBean instance created and
initialized.");
    }

    @Override
    public void addItem(String item) {
        items.add(item);
        System.out.println("ShoppingCartBean: Added item - " + item + ".
Current items: " + items);
    }

    @Override
    public void removeItem(String item) {
        if (items.remove(item)) {
            System.out.println("ShoppingCartBean: Removed item - " + item +
". Current items: " + items);
        } else {
            System.out.println("ShoppingCartBean: Item not found for removal
- " + item);
        }
    }

    @Override
    public List<String> getItems() {
        System.out.println("ShoppingCartBean: Retrieving items - " + items);
        return new ArrayList<>(items); // Return a copy to prevent external
modification
    }

    @Remove
    @Override
    public void checkout() {
        System.out.println("ShoppingCartBean: Checking out and removing bean
instance. Final items: " + items);
        items.clear(); // Clear items as part of checkout
    }

    @PreDestroy
    public void cleanup() {
        System.out.println("ShoppingCartBean instance is being destroyed.");
    }
}
```

StatefulEJBClient.java *(Client Application - Java SE)*

```
// src/main/java/com/example/client/StatefulEJBClient.java
package com.example.client;
```

```
import com.example.ejb.ShoppingCart;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import java.util.Properties;
import java.util.List;

public class StatefulEJBClient {

    public static void main(String[] args) {
        Context initialContext = null;
        try {
            // Setup JNDI properties for connecting to the application server
            Properties jndiProperties = new Properties();
            jndiProperties.put(Context.INITIAL_CONTEXT_FACTORY,
"org.jboss.naming.remote.client.InitialContextFactory"); // For WildFly
            jndiProperties.put(Context.PROVIDER_URL, "http-
remoting://localhost:8080"); // For WildFly

            initialContext = new InitialContext(jndiProperties);

            // JNDI lookup name for a remote EJB
            // ejb:<app-name>/<module-name>/<bean-name>!<fully-qualified-
remote-interface-name>
            String lookupName = "ejb:/ejb-stateful-
demo/ShoppingCartBean!com.example.ejb.ShoppingCart";
            // Adjust "ejb-stateful-demo" to your actual EJB module name

            System.out.println("Looking up Stateful EJB: " + lookupName);
            ShoppingCart cart = (ShoppingCart)
initialContext.lookup(lookupName);

            System.out.println("EJB lookup successful. Invoking methods to
demonstrate state...");

            // Add items to the cart
            cart.addItem("Laptop");
            cart.addItem("Mouse");
            cart.addItem("Keyboard");

            // Get current items
            List<String> currentItems = cart.getItems();
            System.out.println("Client: Items in cart after adding: " +
currentItems);

            // Remove an item
            cart.removeItem("Mouse");

            // Get items again to confirm removal and state persistence
            currentItems = cart.getItems();
            System.out.println("Client: Items in cart after removing Mouse: "
+ currentItems);

            // Add another item
            cart.addItem("Monitor");

            currentItems = cart.getItems();
            System.out.println("Client: Items in cart after adding Monitor: "
+ currentItems);

            // Checkout and remove the bean instance
            cart.checkout();
            System.out.println("Client: Checkout complete. Bean instance
should be removed.");
```

```
            // Attempting to call methods after @Remove will result in an
exception
            // System.out.println("Attempting to get items after checkout: "
+ cart.getItems()); // This will throw an exception

        } catch (NamingException e) {
            System.err.println("JNDI NamingException: " + e.getMessage());
            e.printStackTrace();
        } finally {
            if (initialContext != null) {
                try {
                    initialContext.close();
                } catch (NamingException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

Input

No direct user input. The client application invokes methods with hardcoded values.

Expected Output

Application Server Console (when ShoppingCartBean instances are created/used and destroyed):

```
ShoppingCartBean instance created and initialized.
ShoppingCartBean: Added item - Laptop. Current items: [Laptop]
ShoppingCartBean: Added item - Mouse. Current items: [Laptop, Mouse]
ShoppingCartBean: Added item - Keyboard. Current items: [Laptop, Mouse,
Keyboard]
ShoppingCartBean: Retrieving items - [Laptop, Mouse, Keyboard]
ShoppingCartBean: Removed item - Mouse. Current items: [Laptop, Keyboard]
ShoppingCartBean: Retrieving items - [Laptop, Keyboard]
ShoppingCartBean: Added item - Monitor. Current items: [Laptop, Keyboard,
Monitor]
ShoppingCartBean: Retrieving items - [Laptop, Keyboard, Monitor]
ShoppingCartBean: Checking out and removing bean instance. Final items:
[Laptop, Keyboard, Monitor]
ShoppingCartBean instance is being destroyed.
```

Client Application Console (`java StatefulEJBClient`):

```
Looking up Stateful EJB: ejb:/ejb-stateful-
demo/ShoppingCartBean!com.example.ejb.ShoppingCart
EJB lookup successful. Invoking methods to demonstrate state...
Client: Items in cart after adding: [Laptop, Mouse, Keyboard]
Client: Items in cart after removing Mouse: [Laptop, Keyboard]
Client: Items in cart after adding Monitor: [Laptop, Keyboard, Monitor]
Client: Checkout complete. Bean instance should be removed.
```

Laboratory 12: An EJB application that demonstrates Entity Bean.

Title

EJB Entity Bean (JPA/Hibernate) Demonstration

Aim

To understand the concept of Entity Beans, which are used to represent persistent data in a database. In modern Java EE, Entity Beans are typically implemented using the Java Persistence API (JPA) with an ORM (Object-Relational Mapping) framework like Hibernate.

Procedure

1. **Setup Environment:** Install a Java EE application server and configure a database (e.g., MySQL, H2). Ensure your project has JPA and Hibernate (or other ORM) dependencies.
2. **Define Entity Class:**
    - Create a simple Java class (POJO) that represents a database table.
    - Annotate the class with `@Entity` and define its primary key with `@Id`.
    - Use other JPA annotations like `@Table`, `@Column` as needed.
3. **Create Persistence Unit:** Configure `persistence.xml` in `META-INF` to define the persistence unit, including data source, entity classes, and JPA provider properties.
4. **Create DAO/Service Layer (Session Bean):**
    - Create a Stateless Session Bean (e.g., `UserDaoBean`) to manage CRUD operations for the `User` entity.
    - Inject `EntityManager` using `@PersistenceContext`.
    - Implement methods for `persist` (save), `find` (retrieve), `merge` (update), and `remove` (delete) operations.
5. **Create a Client Application:**
    - Use JNDI lookup to get a reference to the Stateless Session Bean.
    - Invoke the CRUD methods on the bean to interact with the database.
6. **Package and Deploy:** Package the EJB module and deploy it to the application server.

Source Code

`UserEntity.java` (Entity Class)

```
// src/main/java/com/example/entity/UserEntity.java
package com.example.entity;

import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;
import javax.persistence.Column;
import java.io.Serializable;

@Entity // Marks this class as a JPA entity
@Table(name = "users_entity") // Maps to a database table named
"users_entity"
public class UserEntity implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id // Marks this field as the primary key
    @GeneratedValue(strategy = GenerationType.IDENTITY) // Auto-increment for
primary key
    private int id;
```

```
    @Column(name = "username", unique = true, nullable = false)
    private String username;

    @Column(name = "email", nullable = false)
    private String email;

    @Column(name = "password", nullable = false) // In real app, store hashed
passwords
    private String password;

    public UserEntity() {
        // Default constructor required by JPA
    }

    public UserEntity(String username, String email, String password) {
        this.username = username;
        this.email = email;
        this.password = password;
    }

    // Getters and Setters
    public int getId() { return id; }
    public void setId(int id) { this.id = id; }
    public String getUsername() { return username; }
    public void setUsername(String username) { this.username = username; }
    public String getEmail() { return email; }
    public void setEmail(String email) { this.email = email; }
    public String getPassword() { return password; }
    public void setPassword(String password) { this.password = password; }

    @Override
    public String toString() {
        return "UserEntity [id=" + id + ", username=" + username + ", email="
+ email + "]";
    }
}
```

persistence.xml **(JPA Configuration)**

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1"
             xmlns="http://xmlns.jcp.org/xml/ns/persistence"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">

    <persistence-unit name="UserPU" transaction-type="JTA">
        <jta-data-source>java:jboss/datasources/ExampleDS</jta-data-source>
        <class>com.example.entity.UserEntity</class>

        <properties>
            <property name="hibernate.hbm2ddl.auto" value="update"/>
            <property name="hibernate.show_sql" value="true"/>
            <property name="hibernate.format_sql" value="true"/>
            <property name="hibernate.dialect"
value="org.hibernate.dialect.MySQL8Dialect"/>
            </properties>
    </persistence-unit>
</persistence>
```

UserDaoBean.java **(Stateless Session Bean for Entity Management)**

```
// src/main/java/com/example/ejb/UserDaoBean.java
package com.example.ejb;

import com.example.entity.UserEntity;
```

```
import java.util.List;
import javax.ejb.Stateless;
import javax.persistence.EntityManager;
import javax.persistence.PersistenceContext;
import javax.persistence.TypedQuery;

@Stateless // Marks this as a Stateless Session Bean
public class UserDaoBean {

    // Injects the EntityManager for the "UserPU" persistence unit
    @PersistenceContext(unitName = "UserPU")
    private EntityManager em;

    public void createUser(UserEntity user) {
        System.out.println("UserDaoBean: Persisting user: " +
user.getUsername());
        em.persist(user); // Saves the user entity to the database
    }

    public UserEntity getUserById(int id) {
        System.out.println("UserDaoBean: Finding user by ID: " + id);
        return em.find(UserEntity.class, id); // Finds an entity by its
primary key
    }

    public List<UserEntity> getAllUsers() {
        System.out.println("UserDaoBean: Retrieving all users.");
        // JPQL (Java Persistence Query Language) query
        TypedQuery<UserEntity> query = em.createQuery("SELECT u FROM
UserEntity u", UserEntity.class);
        return query.getResultList(); // Returns a list of all UserEntity
objects
    }

    public UserEntity updateUser(UserEntity user) {
        System.out.println("UserDaoBean: Merging user: " +
user.getUsername());
        return em.merge(user); // Updates an existing entity or makes a
detached entity managed
    }

    public void deleteUser(int id) {
        System.out.println("UserDaoBean: Deleting user with ID: " + id);
        UserEntity user = em.find(UserEntity.class, id);
        if (user != null) {
            em.remove(user); // Removes the entity from the database
        }
    }
}
```

EntityClient.java (Client Application - Java SE)

```
// src/main/java/com/example/client/EntityClient.java
package com.example.client;

import com.example.ejb.UserDaoBean;
import com.example.entity.UserEntity;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
import java.util.List;
import java.util.Properties;

public class EntityClient {

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

```
        Context initialContext = null;
        try {
            // Setup JNDI properties for connecting to the application server
            Properties jndiProperties = new Properties();
            jndiProperties.put(Context.INITIAL_CONTEXT_FACTORY,
"org.jboss.naming.remote.client.InitialContextFactory"); // For WildFly
            jndiProperties.put(Context.PROVIDER_URL, "http-
remoting://localhost:8080"); // For WildFly

            initialContext = new InitialContext(jndiProperties);

            // JNDI lookup name for the Stateless Session Bean managing
entities
            String lookupName = "ejb:/ejb-entity-
demo/UserDaoBean!com.example.ejb.UserDaoBean";
            // Adjust "ejb-entity-demo" to your actual EJB module name

            System.out.println("Looking up EJB: " + lookupName);
            UserDaoBean userDao = (UserDaoBean)
initialContext.lookup(lookupName);

            System.out.println("EJB lookup successful. Performing CRUD
operations...");

            // --- CREATE ---
            System.out.println("\n--- Creating a new user ---");
            UserEntity newUser = new UserEntity("john.doe",
"john.doe@example.com", "securepass");
            userDao.createUser(newUser);
            System.out.println("Created user: " + newUser.getUsername());

            // --- READ ALL ---
            System.out.println("\n--- Listing all users ---");
            List<UserEntity> users = userDao.getAllUsers();
            if (users.isEmpty()) {
                System.out.println("No users found.");
            } else {
                for (UserEntity user : users) {
                    System.out.println("Found: " + user);
                }
            }

            // --- READ BY ID ---
            System.out.println("\n--- Finding user by ID (assuming ID 1
exists) ---");
            // Note: The ID is assigned by the database after persist.
            // In a real app, you might refresh the object or query by
username.
            // For this demo, let's assume the first created user has ID 1.
            UserEntity foundUser = userDao.getUserById(1);
            if (foundUser != null) {
                System.out.println("Found user by ID 1: " + foundUser);
            } else {
                System.out.println("User with ID 1 not found.");
            }

            // --- UPDATE ---
            if (foundUser != null) {
                System.out.println("\n--- Updating user with ID " +
foundUser.getId() + " ---");
                foundUser.setEmail("john.doe.updated@example.com");
                foundUser.setPassword("new_secure_pass");
                UserEntity updatedUser = userDao.updateUser(foundUser);
                System.out.println("Updated user: " + updatedUser);
            }
```

```
            // --- READ ALL AGAIN (to confirm update) ---
            System.out.println("\n--- Listing all users after update ---");
            users = userDao.getAllUsers();
            for (UserEntity user : users) {
                System.out.println("Found: " + user);
            }

            // --- DELETE ---
            if (foundUser != null) {
                System.out.println("\n--- Deleting user with ID " +
foundUser.getId() + " ---");
                userDao.deleteUser(foundUser.getId());
                System.out.println("User with ID " + foundUser.getId() + "
deleted.");
            }

            // --- READ ALL AFTER DELETE ---
            System.out.println("\n--- Listing all users after delete ---");
            users = userDao.getAllUsers();
            if (users.isEmpty()) {
                System.out.println("No users found after deletion.");
            } else {
                for (UserEntity user : users) {
                    System.out.println("Found: " + user);
                }
            }

        } catch (NamingException e) {
            System.err.println("JNDI NamingException: " + e.getMessage());
            e.printStackTrace();
        } finally {
            if (initialContext != null) {
                try {
                    initialContext.close();
                } catch (NamingException e) {
                    e.printStackTrace();
                }
            }
        }
    }
}
```

Input

No direct user input. The client application performs hardcoded CRUD operations.

Expected Output

Application Server Console (showing JPA/Hibernate SQL logs):

```
Table 'users_entity' created or already exists.
UserDaoBean: Persisting user: john.doe
Hibernate: insert into users_entity (email, password, username) values (?, ?,
?)
UserDaoBean: Retrieving all users.
Hibernate: select userentity0_.id as id1_0_, userentity0_.email as email2_0_,
userentity0_.password as password3_0_, userentity0_.username as username4_0_
from users_entity userentity0_
UserDaoBean: Finding user by ID: 1
Hibernate: select userentity0_.id as id1_0_0_, userentity0_.email as
email2_0_0_, userentity0_.password as password3_0_0_, userentity0_.username
as username4_0_0_ from users_entity userentity0_ where userentity0_.id=?
UserDaoBean: Merging user: john.doe
```

```
Hibernate: update users_entity set email=?, password=?, username=? where id=?
UserDaoBean: Retrieving all users.
Hibernate: select userentity0_.id as id1_0_, userentity0_.email as email2_0_,
userentity0_.password as password3_0_, userentity0_.username as username4_0_
from users_entity userentity0_
UserDaoBean: Deleting user with ID: 1
Hibernate: select userentity0_.id as id1_0_0_, userentity0_.email as
email2_0_0_, userentity0_.password as password3_0_0_, userentity0_.username
as username4_0_0_ from users_entity userentity0_ where userentity0_.id=?
Hibernate: delete from users_entity where id=?
UserDaoBean: Retrieving all users.
Hibernate: select userentity0_.id as id1_0_, userentity0_.email as email2_0_,
userentity0_.password as password3_0_, userentity0_.username as username4_0_
from users_entity userentity0_
```

(Note: Actual SQL statements and logging might vary slightly based on JPA provider and configuration.)

Client Application Console (`java EntityClient`):

```
Looking up EJB: ejb:/ejb-entity-demo/UserDaoBean!com.example.ejb.UserDaoBean
EJB lookup successful. Performing CRUD operations...

--- Creating a new user ---
Created user: john.doe

--- Listing all users ---
Found: User [id=1, username=john.doe, email=john.doe@example.com]

--- Finding user by ID (assuming ID 1 exists) ---
Found user by ID 1: User [id=1, username=john.doe,
email=john.doe@example.com]

--- Updating user with ID 1 ---
Updated user: User [id=1, username=john.doe,
email=john.doe.updated@example.com]

--- Listing all users after update ---
Found: User [id=1, username=john.doe, email=john.doe.updated@example.com]

--- Deleting user with ID 1 ---
User with ID 1 deleted.

--- Listing all users after delete ---
No users found after deletion.
```

Laboratory 13: MVC Architecture(i) Implementing MVC with Request Dispatcher(ii) Data Sharing Approaches

Title

MVC Architecture in Web Applications: Request Dispatcher and Data Sharing

Aim

To understand and implement the Model-View-Controller (MVC) architectural pattern in a Java web application, specifically focusing on using `RequestDispatcher` for forwarding requests and various approaches for sharing data between components (Controller to View).

Procedure

1. **Setup Dynamic Web Project:** Create a new Dynamic Web Project.
2. **Model:** Create a simple Java POJO (e.g., `Message`) to represent data.
3. **Controller (Servlet):**
    - Create a `ControllerServlet`.
    - This servlet will receive initial requests.
    - Process the request (e.g., retrieve data from a "model" or perform business logic).
    - Store data in `request` scope using `request.setAttribute()`.
    - Forward the request to a JSP (View) using `request.getRequestDispatcher().forward()`.
4. **View (JSP):**
    - Create a `displayMessage.jsp`.
    - Retrieve data from `request` scope using `request.getAttribute()`.
    - Display the data.
5. **Data Sharing Approaches:**
    - **Request Scope:** Data available only for the current request and the forwarded page. (Primary focus for this lab).
    - **Session Scope:** Data available across multiple requests within the same user session (`session.setAttribute()`, `session.getAttribute()`).
    - **Application Scope:** Data available across the entire application (`application.setAttribute()`, `application.getAttribute()`).
6. **HTML Form:** Create an `index.html` to initiate requests to the controller.
7. **Deployment:** Deploy the application.

Source Code

`Message.java` (Model)

```
// src/main/java/com/example/model/Message.java
package com.example.model;

import java.io.Serializable;

public class Message implements Serializable {
    private static final long serialVersionUID = 1L;
    private String header;
    private String content;
    private String sender;

    public Message(String header, String content, String sender) {
        this.header = header;
        this.content = content;
```

```
        this.sender = sender;
    }

    // Getters
    public String getHeader() { return header; }
    public String getContent() { return content; }
    public String getSender() { return sender; }

    // Setters (optional, if message is mutable)
    public void setHeader(String header) { this.header = header; }
    public void setContent(String content) { this.content = content; }
    public void setSender(String sender) { this.sender = sender; }

    @Override
    public String toString() {
        return "Message [header=" + header + ", content=" + content + ",
sender=" + sender + "]";
    }
}
```

ControllerServlet.java *(Controller)*

```
// src/main/java/com/example/controller/ControllerServlet.java
package com.example.controller;

import com.example.model.Message;
import java.io.IOException;
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
import javax.servlet.ServletContext; // For application scope

@WebServlet("/controller")
public class ControllerServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

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

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

    private void processRequest(HttpServletRequest request,
HttpServletResponse response)
            throws ServletException, IOException {

        String action = request.getParameter("action");
        if (action == null) {
            action = "default";
        }

        String viewPage = "/displayMessage.jsp"; // Default view

        try {
            switch (action) {
                case "showMessage":
```

```
                    // 1. Model Logic: Create a Message object
                    Message msg = new Message("Important Notice", "This is a
message from the Controller!", "Admin");

                    // 2. Data Sharing (Request Scope): Put data into request
attribute
                    request.setAttribute("messageObject", msg);
                    request.setAttribute("greeting", "Hello from Request
Scope!");

                    // 3. Data Sharing (Session Scope - for demonstration)
                    HttpSession session = request.getSession();
                    session.setAttribute("sessionMessage", "This data lives
in your session!");

                    // 4. Data Sharing (Application Scope - for
demonstration)
                    ServletContext application = getServletContext();
                    application.setAttribute("appCounter",
(Integer)application.getAttribute("appCounter") == null ? 1 :
((Integer)application.getAttribute("appCounter") + 1));
                    application.setAttribute("applicationInfo", "This is
global application info.");

                    System.out.println("ControllerServlet: Data prepared and
forwarded to " + viewPage);
                    break;
                case "showError":
                    request.setAttribute("errorMessage", "An error occurred
during processing!");
                    viewPage = "/error.jsp";
                    System.out.println("ControllerServlet: Forwarding to
error page " + viewPage);
                    break;
                default:
                    request.setAttribute("messageObject", new
Message("Default Message", "No specific action requested.", "System"));
                    System.out.println("ControllerServlet: Default action,
forwarding to " + viewPage);
                    break;
            }

            // Forward the request to the JSP view
            RequestDispatcher dispatcher =
request.getRequestDispatcher(viewPage);
            dispatcher.forward(request, response);

        } catch (Exception e) {
            request.setAttribute("errorMessage", "An unexpected error
occurred: " + e.getMessage());
            RequestDispatcher dispatcher =
request.getRequestDispatcher("/error.jsp");
            dispatcher.forward(request, response);
            e.printStackTrace();
        }
    }
}
```

index.html ***(Entry Point)***

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>MVC Demo Home</title>
```

```
    <style>
        body { font-family: 'Inter', sans-serif; display: flex; justify-
content: center; align-items: center; min-height: 100vh; background-color:
#f0f2f5; margin: 0; }
        .container { background-color: #ffffff; padding: 30px; border-radius:
12px; box-shadow: 0 4px 15px rgba(0, 0, 0, 0.1); text-align: center; max-
width: 400px; width: 90%; }
        h1 { color: #333; margin-bottom: 25px; }
        .button-group { display: flex; flex-direction: column; gap: 15px; }
        .button-group a { display: inline-block; padding: 12px 25px;
background-color: #007bff; color: white; text-decoration: none; border-
radius: 8px; font-weight: bold; transition: background-color 0.3s ease; }
        .button-group a.error-button { background-color: #dc3545; }
        .button-group a:hover { background-color: #0056b3; }
        .button-group a.error-button:hover { background-color: #c82333; }
    </style>
</head>
<body>
    <div class="container">
        <h1>MVC Architecture Demo</h1>
        <p>Click a button to trigger a request to the Controller Servlet:</p>
        <div class="button-group">
            <a href="controller?action=showMessage">Show Message (MVC
Flow)</a>
            <a href="controller?action=showError" class="error-
button">Trigger Error Page</a>
            <a href="controller">Default Action</a>
        </div>
    </div>
</body>
</html>
```

displayMessage.jsp *(View: Display Message)*

```
<%-- WebContent/displayMessage.jsp --%>
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Display Message</title>
    <style>
        body { font-family: 'Inter', sans-serif; display: flex; justify-
content: center; align-items: center; min-height: 100vh; background-color:
#f0f2f5; margin: 0; }
        .container { background-color: #ffffff; padding: 30px; border-radius:
12px; box-shadow: 0 4px 15px rgba(0, 0, 0, 0.1); text-align: center; max-
width: 600px; width: 90%; }
        h1 { color: #333; margin-bottom: 25px; }
        h2 { color: #007bff; margin-top: 20px; }
        p { color: #555; font-size: 1.1em; margin-bottom: 10px; }
        .data-section { background-color: #e9ecef; padding: 15px; border-
radius: 8px; margin-top: 20px; text-align: left; }
        .data-section p { margin: 5px 0; }
        .back-link { display: inline-block; margin-top: 30px; padding: 10px
20px; background-color: #6c757d; color: white; text-decoration: none; border-
radius: 8px; transition: background-color 0.3s ease; }
        .back-link:hover { background-color: #5a6268; }
    </style>
</head>
<body>
    <div class="container">
        <h1>Message Display Page</h1>
```

```
        <%-- Retrieve data from Request Scope --%>
        <c:set var="message" value="${requestScope.messageObject}" />
        <c:set var="greeting" value="${requestScope.greeting}" />

        <c:if test="${not empty message}">
            <h2>${message.header}</h2>
            <p>${message.content}</p>
            <p><em>Sent by: ${message.sender}</em></p>
        </c:if>

        <div class="data-section">
            <h3>Data from Scopes:</h3>
            <p><strong>Request Scope:</strong> <c:out value="${greeting}"
/></p>
            <p><strong>Session Scope:</strong> <c:out
value="${sessionScope.sessionMessage}" /></p>
            <p><strong>Application Scope:</strong> <c:out
value="${applicationScope.applicationInfo}" /> (Visited: <c:out
value="${applicationScope.appCounter}" /> times)</p>
        </div>

        <a href="index.html" class="back-link">Go Back to Home</a>
    </div>
</body>
</html>
```

error.jsp **(View: Error Page)**

```
<%-- WebContent/error.jsp --%>
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Error</title>
    <style>
        body { font-family: 'Inter', sans-serif; display: flex; justify-
content: center; align-items: center; min-height: 100vh; background-color:
#f0f2f5; margin: 0; }
        .container { background-color: #ffffff; padding: 30px; border-radius:
12px; box-shadow: 0 4px 15px rgba(0, 0, 0, 0.1); text-align: center; max-
width: 500px; width: 90%; border-left: 5px solid #dc3545; }
        h1 { color: #dc3545; margin-bottom: 25px; }
        p { color: #555; font-size: 1.1em; margin-bottom: 20px; }
        .error-details { background-color: #f8d7da; border: 1px solid
#f5c6cb; color: #721c24; padding: 15px; border-radius: 8px; text-align: left;
font-family: monospace; white-space: pre-wrap; word-wrap: break-word; }
        .back-link { display: inline-block; margin-top: 30px; padding: 10px
20px; background-color: #6c757d; color: white; text-decoration: none; border-
radius: 8px; transition: background-color 0.3s ease; }
        .back-link:hover { background-color: #5a6268; }
    </style>
</head>
<body>
    <div class="container">
        <h1>Error Occurred!</h1>
        <p>We're sorry, but something went wrong.</p>
        <c:if test="${not empty requestScope.errorMessage}">
            <div class="error-details">
                <p><strong>Details:</strong></p>
                <p><c:out value="${requestScope.errorMessage}" /></p>
            </div>
```

```
        </c:if>
        <a href="index.html" class="back-link">Go Back to Home</a>
    </div>
</body>
</html>
```

Input

1. Access `index.html`.
2. Click "Show Message (MVC Flow)".
3. Click "Trigger Error Page".
4. Click "Default Action".

Expected Output

1. After clicking "Show Message (MVC Flow)": A web page titled "Display Message" will appear, showing:

- A heading "Message Display Page".
- "Important Notice" as a sub-heading.
- "This is a message from the Controller!" and "Sent by: Admin".
- A section "Data from Scopes:" with:
    - "Request Scope: Hello from Request Scope!"
    - "Session Scope: This data lives in your session!"
    - "Application Scope: This is global application info. (Visited: 1 times)" (The counter will increment with subsequent visits to `showMessage` or `default` action).
- A "Go Back to Home" link.

2. After clicking "Trigger Error Page": A web page titled "Error" will appear, showing:

- A heading "Error Occurred!".
- "We're sorry, but something went wrong."
- "Details: An error occurred during processing!"
- A "Go Back to Home" link.

3. After clicking "Default Action": A web page identical to the "Show Message (MVC Flow)" output, but the message details will be:

- "Default Message" as a sub-heading.
- "No specific action requested." and "Sent by: System".
- The application scope counter will increment (e.g., "Visited: 2 times" if it was 1 before).

Laboratory 14: Build a web application that collects the user's name and displays " Hello World " followed by the user name .

Title

Hello World Web Application with User Name

Aim

To create a simple web application that takes a user's name as input and displays a personalized "Hello World" greeting using a Servlet and an HTML form. This is a fundamental exercise combining HTML forms with server-side processing.

Procedure

1. **Setup Dynamic Web Project:** Create a new Dynamic Web Project in your IDE.
2. **Create an HTML Form:**
    - Create an `index.html` file.
    - Include an HTML form with a text input field for the user's name.
    - Set the form's `action` attribute to point to your servlet's URL pattern and `method` to `GET` or `POST`.
3. **Create a Servlet:**
    - Create a Java class that extends `javax.servlet.http.HttpServlet`.
    - Annotate it with `@WebServlet("/hello")` (or configure in `web.xml`).
    - Override the `doGet()` or `doPost()` method (matching your form's method).
    - Inside the method, retrieve the user's name from the `HttpServletRequest` object using `request.getParameter("name")`.
    - Construct the greeting message "Hello World, [UserName]!".
    - Use `HttpServletResponse` to write the greeting back to the client as HTML.
4. **Deployment:** Deploy the web application to your servlet container (e.g., Tomcat) and access `index.html` through a web browser.

Source Code

`index.html` (HTML Form)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Hello World App</title>
    <style>
        body { font-family: 'Inter', sans-serif; display: flex; justify-
content: center; align-items: center; min-height: 100vh; background-color:
#f0f2f5; margin: 0; }
        .container { background-color: #ffffff; padding: 30px; border-radius:
12px; box-shadow: 0 4px 15px rgba(0, 0, 0, 0.1); text-align: center; max-
width: 400px; width: 90%; }
        h1 { color: #333; margin-bottom: 25px; }
        form { display: flex; flex-direction: column; gap: 15px; }
        label { text-align: left; font-weight: bold; color: #555; }
        input[type="text"] { padding: 12px; border: 1px solid #ddd; border-
radius: 8px; font-size: 16px; width: calc(100% - 24px); }
        input[type="submit"] { background-color: #007bff; color: white;
padding: 12px 20px; border: none; border-radius: 8px; cursor: pointer; font-
size: 18px; font-weight: bold; transition: background-color 0.3s ease; }
```

```
        input[type="submit"]:hover { background-color: #0056b3; }
    </style>
</head>
<body>
    <div class="container">
        <h1>Enter Your Name</h1>
        <form action="hello" method="get">
            <label for="userName">Name:</label>
            <input type="text" id="userName" name="userName"
placeholder="e.g., Alice" required>
            <input type="submit" value="Submit">
        </form>
    </div>
</body>
</html>
```

HelloWorldServlet.java *(Servlet)*

```
// src/main/java/com/example/HelloWorldServlet.java
package com.example;

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

@WebServlet("/hello") // Maps this servlet to the /hello URL pattern
public class HelloWorldServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * Handles HTTP GET requests.
     * Retrieves the 'userName' parameter and sends a personalized greeting.
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {

        // Set content type of the response to HTML
        response.setContentType("text/html");
        response.setCharacterEncoding("UTF-8");

        // Get the 'userName' parameter from the request
        String userName = request.getParameter("userName");

        // If userName is null or empty, provide a default
        if (userName == null || userName.trim().isEmpty()) {
            userName = "Guest";
        }

        // Get a PrintWriter to send HTML response back to the client
        PrintWriter out = response.getWriter();

        // Generate HTML response
        out.println("<!DOCTYPE html>");
        out.println("<html lang='en'>");
        out.println("<head>");
        out.println("    <meta charset='UTF-8'>");
        out.println("    <meta name='viewport' content='width=device-width,
initial-scale=1.0'>");
        out.println("    <title>Greeting</title>");
        out.println("    <style>");
```

```
        out.println("        body { font-family: 'Inter', sans-serif;
display: flex; justify-content: center; align-items: center; min-height:
100vh; background-color: #f0f2f5; margin: 0; }");
        out.println("        .container { background-color: #ffffff; padding:
30px; border-radius: 12px; box-shadow: 0 4px 15px rgba(0, 0, 0, 0.1); text-
align: center; max-width: 400px; width: 90%; }");
        out.println("        h1 { color: #333; margin-bottom: 20px; }");
        out.println("        a { display: inline-block; margin-top: 25px;
padding: 10px 20px; background-color: #007bff; color: white; text-decoration:
none; border-radius: 8px; transition: background-color 0.3s ease; }");
        out.println("        a:hover { background-color: #0056b3; }");
        out.println("    </style>");
        out.println("</head>");
        out.println("<body>");
        out.println("    <div class='container'>");
        out.println("        <h1>Hello World, " + userName + "!</h1>");
        out.println("        <a href='index.html'>Go Back</a>");
        out.println("    </div>");
        out.println("</body>");
        out.println("</html>");

        // Close the PrintWriter
        out.close();
    }

    /**
     * Handles HTTP POST requests. For this simple example,
     * it just calls doGet to process the request.
     */
    protected void doPost(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {
        doGet(request, response);
    }
}
```

Input

1. Access `index.html` in a web browser.
2. Enter a name in the text field (e.g., `Alice`).
3. Click "Submit".

Expected Output

1. Initial `index.html`: A web page with a heading "Enter Your Name" and a text input field labeled "Name:" and a "Submit" button.

2. After submitting "Alice": A new web page displaying:

```
<!DOCTYPE html>
<html lang='en'>
<head>
    </head>
<body>
    <div class='container'>
        <h1>Hello World, Alice!</h1>
        <a href='index.html'>Go Back</a>
    </div>
</body>
</html>
```

Visually, it would show "Hello World, Alice!" as a prominent heading and a "Go Back" link.

Laboratory 15: creating our view which will be required to browse and upload a selected file .

Title

File Upload Web Application

Aim

To develop a web application that allows users to select a file from their local system and upload it to the server. This involves handling multipart form data in a Servlet.

Procedure

1. **Setup Dynamic Web Project:** Create a new Dynamic Web Project.
2. **Add Dependencies:** Ensure your project has `javax.servlet-api` and `commons-fileupload` (or `javax.servlet.annotation.MultipartConfig` for Servlet 3.0+) dependencies. For Servlet 3.0+, `commons-fileupload` is not strictly necessary for basic functionality, but it's often used for more robust handling. We'll use Servlet 3.0+ annotations.
3. **Create an HTML Form:**
    - Create an `index.html` file.
    - Include an HTML form with `enctype="multipart/form-data"` and an `input type="file"`.
    - Set the form's `action` to your upload servlet's URL pattern and `method` to `POST`.
4. **Create an Upload Servlet:**
    - Create a Java class extending `HttpServlet`.
    - Annotate the servlet with `@WebServlet("/upload")` and `@MultipartConfig` to enable multipart form data handling.
    - Override the `doPost()` method.
    - Inside `doPost()`, retrieve the uploaded `Part` using `request.getPart("fileInputName")`.
    - Get the filename from the `Part`.
    - Specify a directory on the server to save the uploaded files.
    - Write the input stream of the `Part` to a `FileOutputStream` to save the file.
    - Provide feedback to the user about the upload status.
5. **Create an Upload Directory:** Create a directory on your server (e.g., `uploads` within your web application's root or a separate path) where uploaded files will be stored. Ensure your servlet container has write permissions to this directory.
6. **Deployment:** Deploy the application to a servlet container.

Source Code

`index.html` (File Upload Form)

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>File Upload Demo</title>
    <style>
        body { font-family: 'Inter', sans-serif; display: flex; justify-
content: center; align-items: center; min-height: 100vh; background-color:
#f0f2f5; margin: 0; }
```

```
        .container { background-color: #ffffff; padding: 30px; border-radius:
12px; box-shadow: 0 4px 15px rgba(0, 0, 0, 0.1); text-align: center; max-
width: 500px; width: 90%; }
        h1 { color: #333; margin-bottom: 25px; }
        form { display: flex; flex-direction: column; gap: 20px; }
        label { text-align: left; font-weight: bold; color: #555; }
        input[type="file"] { padding: 10px; border: 1px solid #ddd; border-
radius: 8px; font-size: 16px; width: calc(100% - 20px); }
        input[type="submit"] { background-color: #28a745; color: white;
padding: 12px 20px; border: none; border-radius: 8px; cursor: pointer; font-
size: 18px; font-weight: bold; transition: background-color 0.3s ease; }
        input[type="submit"]:hover { background-color: #218838; }
        .message { margin-top: 20px; font-weight: bold; }
        .success { color: green; }
        .error { color: red; }
    </style>
</head>
<body>
    <div class="container">
        <h1>Upload Your File</h1>
        <form action="upload" method="post" enctype="multipart/form-data">
            <label for="fileToUpload">Select a file:</label>
            <input type="file" id="fileToUpload" name="fileToUpload"
required>
            <input type="submit" value="Upload File">
        </form>

        <%
            // Display upload status message if available
            String status = request.getParameter("status");
            String fileName = request.getParameter("fileName");
            if ("success".equals(status)) {
                out.println("<p class='message success'>File <strong>" +
fileName + "</strong> uploaded successfully!</p>");
            } else if ("error".equals(status)) {
                String errorMessage = request.getParameter("errorMessage");
                out.println("<p class='message error'>Error uploading file: "
+ (errorMessage != null ? errorMessage : "Unknown error") + "</p>");
            }
        %>
    </div>
</body>
</html>
```

FileUploadServlet.java **(Upload Servlet)**

```
// src/main/java/com/example/FileUploadServlet.java
package com.example;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import javax.servlet.ServletException;
import javax.servlet.annotation.MultipartConfig;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.Part;

// @WebServlet annotation maps this servlet to the /upload URL pattern
@WebServlet("/upload")
// @MultipartConfig enables handling of multipart/form-data requests
// fileSizeThreshold: The size in bytes at which the file is written to disk
// maxFileSize: The maximum size allowed for uploaded files (10MB)
```

```
// maxRequestSize: The maximum size allowed for a multipart/form-data request
(20MB)
@MultipartConfig(fileSizeThreshold = 1024 * 1024 * 2, // 2MB
                 maxFileSize = 1024 * 1024 * 10,    // 10MB
                 maxRequestSize = 1024 * 1024 * 20) // 20MB
public class FileUploadServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    // Define the directory where uploaded files will be saved
    // IMPORTANT: Ensure this path is writable by your application server!
    // For simplicity, we'll use a relative path within the web app's
deployed directory.
    // In production, use an absolute path outside the web app for
persistence.
    private static final String UPLOAD_DIRECTORY = "uploads";

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

        String uploadPath = getServletContext().getRealPath("") +
File.separator + UPLOAD_DIRECTORY;
        File uploadDir = new File(uploadPath);
        if (!uploadDir.exists()) {
            uploadDir.mkdir(); // Create the directory if it doesn't exist
            System.out.println("Created upload directory: " + uploadPath);
        }

        String fileName = null;
        try {
            // Get the Part corresponding to the file input field with name
"fileToUpload"
            Part filePart = request.getPart("fileToUpload");

            // Extract the filename from the Part header
            fileName = getFileName(filePart);

            if (fileName == null || fileName.isEmpty()) {
                throw new IllegalArgumentException("No file selected or
invalid filename.");
            }

            // Construct the full path where the file will be saved
            String filePath = uploadPath + File.separator + fileName;

            // Save the file to the server
            try (InputStream fileContent = filePart.getInputStream();
                 FileOutputStream fos = new FileOutputStream(filePath)) {

                byte[] buffer = new byte[1024];
                int bytesRead;
                while ((bytesRead = fileContent.read(buffer)) != -1) {
                    fos.write(buffer, 0, bytesRead);
                }
            }

            System.out.println("File uploaded successfully: " + filePath);
            // Redirect back to index.html with success status
            response.sendRedirect("index.html?status=success&fileName=" +
fileName);

        } catch (Exception ex) {
            System.err.println("File upload error: " + ex.getMessage());
            ex.printStackTrace();
            // Redirect back to index.html with error status
```

```
            response.sendRedirect("index.html?status=error&errorMessage=" +
ex.getMessage());
        }
    }

    /**
     * Extracts the filename from the Content-Disposition header of the Part.
     */
    private String getFileName(Part part) {
        for (String content : part.getHeader("content-
disposition").split(";")) {
            if (content.trim().startsWith("filename")) {
                return content.substring(content.indexOf('=') +
1).trim().replace("\"", "");
            }
        }
        return null;
    }
}
```

Input

1. Access `index.html` in a web browser.
2. Click "Choose File" and select any file from your local computer.
3. Click "Upload File".

Expected Output

1. Initial `index.html`: A web page with a heading "Upload Your File", a "Choose File" button, and an "Upload File" button.

2. After successful file upload: The `index.html` page reloads, and a success message appears below the form:

```
<p class='message success'>File <strong>[YourFileName.ext]</strong> uploaded
successfully!</p>
```

And in the server console:

```
Created upload directory: [path_to_your_webapp]/uploads (if it didn't exist)
File uploaded successfully: [path_to_your_webapp]/uploads/[YourFileName.ext]
```

The uploaded file will be present in the `uploads` directory within your deployed web application.

3. After a failed file upload (e.g., file too large, permissions issue): The `index.html` page reloads, and an error message appears below the form:

```
<p class='message error'>Error uploading file: [Specific error message, e.g.,
"Size exceeds maxFileSize"]</p>
```

And in the server console, an error message and stack trace will be printed.