**MIT ART DESIGN &**

**TECHNOLOGY UNIVERSITY**

**MIT College of Management**

**(MITCOM), Pune**

**PROGRAMME: MASTER OF COMPUTER APPLICATION (MCA CC /DS)**

**ADVANCED JAVA**

# CERTIFICATE

This is to certify that, **Mr.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** has submitted a Practical Report on **Advanced Java** to MIT – ADT University, Pune for the partial fulfillment of Master in Computer Application

(Data Science/ Cloud Computing) submitted during the academic year 2024-25.

**PRN No.:- \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**. **MCA Year :- II. MCA Sem :- III**

Subject Incharge Dr.Alkawati Magadum Dr.Sangita Phunde

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| HOD MCA | | Principal | |  |
|  |  |
| **1.External Examiner** |  | **Sign of Examiners:** |  |  |
|  |  |  |  |  |
| **2.Internal Examiner** |  | **Sign of Examiners:** |  |  |
|  |  |  |  |  |

MIT ART DESIGN & TECHNOLOGY UNIVERSITY

**MIT College of Management**

**(MITCOM), Pune**

**Declaration**

I undersigned hereby declares that, the Journal of assignments solved by me and it is executed as per the course requirement of MCA program of MIT-ADT University, Pune. This report has not submitted by me or any other person to any other University or Institution for a degree or diploma course. This is my own and original work.

Place: MITCOM, Pune Sign of the student: ----------------------------

Date:

Name of the Student

**MIT ART DESIGN & TECHNOLOGY UNIVERSITY**

**MIT College of Management (MITCOM), Pune**

**Sub:- Advanced Java**

**Name:- \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** Div:- MCA (DS-B)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Sr No.** | **Name Of The Practical** | **Page** | **Date** | **Record Sign** |
| **1** | Write a Java program to connect to a specific database (e.g., MySQL, workbench etc.) using JDBC. Create a table in the database using JDBC and insert some sample data. and retrieve all data from a specific table and display it on the console |  |  |  |
| **2** | Implement a program to update a specific record in a table based on a given condition. and delete a record from a table based on a specific criteria. |  |  |  |
| **3** | Write a program to utilize transactions in JDBC, demonstrating both commit and rollback functionalities. |  |  |  |
| **4** | Implement a program to handle different types of JDBC exceptions effectively. Write JDBC Program to calculate Employee salary and print the salary statement in tabular form by selecting the details from database table  (Emp\_Sal) using Prepared Statement |  |  |  |
| **5** | Write a program to perform aggregation functions (e.g., COUNT, SUM,AVERAGE) on data retrieved from a database. |  |  |  |
| **6** | Write a program to create a simple Java application that interacts with a database to perform CRUD operations (Create, Read, Update, Delete) on a specific table. |  |  |  |
| **7** | Design a simple servlet that displays a welcome message with the user's name retrieved from request parameters. |  |  |  |
| **8** | Design a simple servlet that displays a welcome message with the user's name retrieved from request parameters. |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **9** | Create a servlet that utilizes session management to maintain a shopping cart for an online store |  |  |  |
| **10** | Write a servlet Program to calculate the addition of two numbers and print the result.(Eg:Addition of two numbers=50) |  |  |  |
| **11** | Write a Servlet Program to create a registration form using in html and CSS and print the message Registration is successful |  |  |  |
| **12** | Write a servlet Program for student information and display the information in tabular form by selecting the details from student database table |  |  |  |
| **13** | Write a Java Servlet program to read employee details including employee number (empno), name, designation, basic pay, deductions, and allowances, and then calculate and display the net salary. display the information in tabular form by selecting the details from Emp\_sal database table |  |  |  |
| **14** | Write a JSP program calculates factorial of an integer number, while the input is taken from an HTML form. |  |  |  |
| **15** | Write a JSP program to generate the Fibonacci series up to a particular term, while the input is taken from an HTML form |  |  |  |
| **16** | Write a JSP program to display the System date and time. |  |  |  |
| **17** | Write a JSP program to display a Sample shopping Order calculation Form and display output in tabular form. |  |  |  |
| **18** | Write a JSP program to perform Arithmetic operations such as Addition, Subtraction, Multiplication and Division. Design a HTML to accept two numbers in text box and radio buttons to display operations. On submit |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | display result as per the selected operation on next page using JSP |  |  |  |
| **19** | Define and illustrate the concept of entity mapping in JPA.Explain how JPA maps Java classes (entities) to database tables.Provide an example of an entity class with annotations and its corresponding database table schema |  |  |  |
| **20** | Describe the different types of relationships between entities (one-to-one, one-to-many, many-to-one, many-to- many).Explain how JPA represents these relationships using annotations.Provide code examples for each type of relationship |  |  |  |
| **21** | Create a JPA application to perform CRUD operations on a simple entity (e.g., Product).    Include methods for creating, retrieving, updating, and deleting Product entities.  Demonstrate the use of EntityManager for persistence operations. |  |  |  |
| **22** | Configure a Spring Boot application to connect to a specific MySQL database without explicitly defining beans for connection pool, DataSource, etc.  Use only the necessary dependencies and demonstrate how auto-configuration sets up the connection.  Explore using application.properties to customize connection details (URL, username, password). |  |  |  |
| **23** | Create a Spring Boot application that utilizes JPA repositories. Persist and retrieve data from an in-memory database (e.g., H2) without manual configuration.  Focus on the simplicity achieved through auto- configuration for JPA and repositories. Implement basic CRUD operations using JPA repositories.    Develop a Spring Boot application with a RESTful API that exposes an endpoint to retrieve a list of products. Utilize Spring MVC annotations like @RestController and  @GetMapping.  Implement a service layer to interact with a product repository (in-memory or database). |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Return the list of products in JSON format using @ResponseBody. |  |  |  |
| **25** | Write a Hibernate program to create the product table (product id,product name,product category,product price) and delete the specific product record.(using through the product id) |  |  |  |
| **26** | Write a Hibernate program to update the product price data from product table.(Using HQL) |  |  |  |
| **27** | Write a Hibernate Program for product information and display the information by selecting the details from product database table |  |  |  |

8

**Assignment 1: Programs On JAVA JDBC**

**1.1 Write a Java program to connect to a specific database (e.g., MySQL, workbench etc.) using JDBC. Create a table in the database using JDBC and insert some sample data.**

**and retrieve all data from a specific table and display it on the console.**

**Ans:-**

import java.sql.Connection; import java.sql.DriverManager; import java.sql.Statement; import java.sql.ResultSet;

public class Practical { public static void main(String[] args) {

// Database credentials

String url = "jdbc:mysql://localhost:3306/college"; // Replace with your database name

String username = "root"; // Replace with your MySQL username

String password = "1234567890"; // Replace with your MySQL password

// JDBC objects

Connection conn = null;

Statement stmt = null;

try {

// 1. Register JDBC Driver

Class.*forName*("com.mysql.cj.jdbc.Driver");

// 2. Open Connection conn = DriverManager.*getConnection*(url, username, password); System.*out*.println("Connection established successfully.");

// 3. Create Statement stmt = conn.createStatement();

// 4. Create Table

String createTableSQL = "CREATE TABLE IF NOT EXISTS students ("

+

"id INT AUTO\_INCREMENT PRIMARY KEY, " +

"name VARCHAR(50), " + "email VARCHAR(50), " +

"grade VARCHAR(10))"; stmt.executeUpdate(createTableSQL);

System.*out*.println("Table `students` created successfully.");

// 5. Insert Sample Data

String insertSQL = "INSERT INTO students (name, email, grade)

VALUES " +

"('Alice Johnson', 'alice@example.com', 'A'), " +

"('Bob Smith', 'bob@example.com', 'B'), " +

"('Charlie Brown', 'charlie@example.com', 'A+')"; stmt.executeUpdate(insertSQL);

System.*out*.println("Sample data inserted successfully into `students`.");

// 6. Retrieve and Display Data

String selectSQL = "SELECT \* FROM students";

ResultSet rs = stmt.executeQuery(selectSQL);

System.*out*.println("Data from `students` table:"); while (rs.next()) { int id = rs.getInt("id");

String name = rs.getString("name");

String email = rs.getString("email");

String grade = rs.getString("grade");

System.*out*.printf("ID: %d, Name: %s, Email: %s, Grade: %s%n", id,

name, email, grade);

}

// Close the ResultSet rs.close();

} catch (Exception e) {

e.printStackTrace();

} finally { try {

// Close the Statement and Connection

if (stmt != null) stmt.close(); if (conn != null) conn.close();

} catch (Exception ex) { ex.printStackTrace();

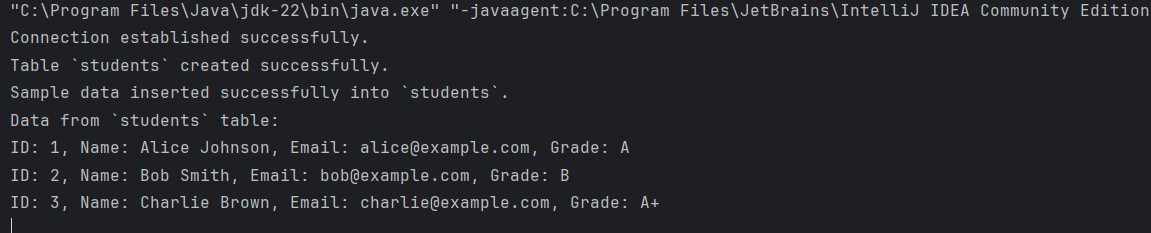
}

}

}

}

OUTPUT:



**1.2 Implement a program to update a specific record in a table based on a given condition and delete a record from a table based on a specific criteria**. **Ans:-**

import java.sql.Connection; import java.sql.DriverManager; import java.sql.PreparedStatement;

public class Practical2 { public static void main(String[] args) {

// Database credentials

String url = "jdbc:mysql://localhost:3306/college"; // Replace with your database name

String username = "root"; // Replace with your MySQL username

String password = "1234567890"; // Replace with your MySQL password

// JDBC objects

Connection conn = null;

try {

// 1. Register JDBC Driver

Class.*forName*("com.mysql.cj.jdbc.Driver");

// 2. Open Connection conn = DriverManager.*getConnection*(url, username, password);

System.*out*.println("Connection established successfully.");

// 3. Update Record

String updateSQL = "UPDATE students SET grade = ? WHERE id = ?"; PreparedStatement updateStmt = conn.prepareStatement(updateSQL); updateStmt.setString(1, "A+"); // New grade updateStmt.setInt(2, 2); // ID of the student to update int updateCount = updateStmt.executeUpdate(); if (updateCount > 0) {

System.*out*.println("Record updated successfully.");

} else {

System.*out*.println("No record found to update."); }

// 4. Delete Record

String deleteSQL = "DELETE FROM students WHERE id = ?"; PreparedStatement deleteStmt = conn.prepareStatement(deleteSQL);

deleteStmt.setInt(1, 3); // ID of the student to delete int deleteCount = deleteStmt.executeUpdate(); if (deleteCount > 0) {

System.*out*.println("Record deleted successfully.");

} else {

System.*out*.println("No record found to delete."); }

// Close statements updateStmt.close(); deleteStmt.close();

} catch (Exception e) {

e.printStackTrace();

} finally { try {

// Close Connection

if (conn != null) conn.close();

} catch (Exception ex) { ex.printStackTrace();

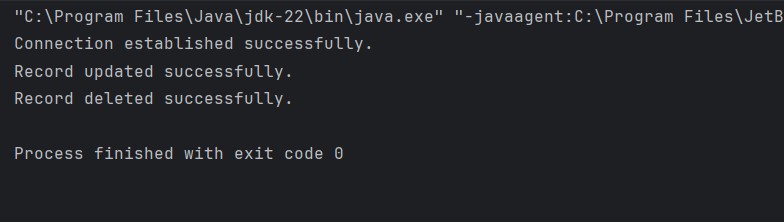
}

}

}

}

**Output:-**



**1.3 Write a program to utilize transactions in JDBC, demonstrating both commit and rollback functionalities**.

Ans:- import java.sql.Connection; import java.sql.DriverManager; import java.sql.PreparedStatement; import java.sql.SQLException;

public class Practical3\_TransactionExample { public static void main(String[] args) {

// Database credentials

String url = "jdbc:mysql://localhost:3306/college"; // Replace with your database name

String username = "root"; // Replace with your MySQL username

String password = "1234567890"; // Replace with your MySQL password Connection conn = null;

try {

// 1. Register JDBC Driver

Class.*forName*("com.mysql.cj.jdbc.Driver");

// 2. Open Connection conn = DriverManager.*getConnection*(url, username, password); System.*out*.println("Connection established successfully.");

// 3. Disable Auto-commit Mode conn.setAutoCommit(false);

// 4. Insert Record 1

String insertSQL1 = "INSERT INTO students (id, name, email, grade)

VALUES (?, ?, ?, ?)";

PreparedStatement stmt1 = conn.prepareStatement(insertSQL1);

stmt1.setString(2, "David Adams"); // Name stmt1.setString(3, "david@example.com"); // Email stmt1.setString(4, "B+"); // Grade stmt1.executeUpdate();

System.*out*.println("Inserted record 1.");

// 5. Insert Record 2

String insertSQL2 = "INSERT INTO students (id, name, email, grade)

VALUES (?, ?, ?, ?)";

PreparedStatement stmt2 = conn.prepareStatement(insertSQL2); stmt2.setInt(1, 5); // ID

stmt2.setString(2, "Eva Green"); // Name stmt2.setString(3, "eva@example.com"); // Email

stmt2.setString(4, "A"); // Grade stmt2.executeUpdate();

System.*out*.println("Inserted record 2.");

// Commit transaction if no error

conn.commit();

System.*out*.println("Transaction committed successfully.");

} catch (SQLException e) {

System.*err*.println("Error occurred, rolling back transaction."); e.printStackTrace(); try { if (conn != null) { conn.rollback(); // Rollback transaction

System.*out*.println("Transaction rolled back successfully."); }

} catch (SQLException rollbackEx) { rollbackEx.printStackTrace();

}

} catch (ClassNotFoundException e) {

e.printStackTrace();

} finally { try { if (conn != null) { conn.setAutoCommit(true); // Restore default auto-commit

behavior

conn.close();

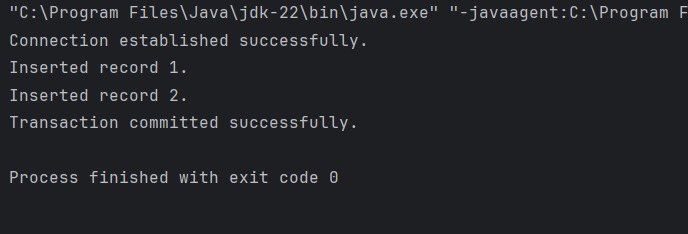
}

} catch (SQLException closeEx) { closeEx.printStackTrace(); }

}

}

OUTPUT :



**1.4 Implement a program to handle different types of JDBC exceptions effectively. Write JDBC Program to calculate Employee salary and print the salary statement in tabular form by selecting the details from database table (Emp\_Sal) using Prepared Statement Ans:-** import java.sql.\*; public class practical4 { public static void main(String[] args) {

String url = "jdbc:mysql://localhost:3306/company\_db"; // Replace with your database name

String username = "root"; // Replace with your MySQL username

String password = "1234567890"; // Replace with your MySQL password

Connection conn = null;

PreparedStatement pstmt = null;

ResultSet rs = null;

try {

// 1. Register JDBC Driver

Class.forName("com.mysql.cj.jdbc.Driver");

// 2. Open Connection conn = DriverManager.getConnection(url, username, password); System.out.println("Connection established successfully.");

// 3. Prepare SQL Query with PreparedStatement

String selectSQL = "SELECT emp\_id, emp\_name, base\_salary, bonus, deduction FROM Emp\_Sal"; pstmt = conn.prepareStatement(selectSQL);

// 4. Execute the query rs = pstmt.executeQuery();

// 5. Display salary statement in tabular form

System.out.println("Employee Salary Statement:");

System.out.printf("%-10s %-20s %-15s %-10s %-10s %-10s%n",

"Emp ID", "Name", "Base Salary", "Bonus", "Deduction", "Total Salary");

// 6. Process the ResultSet while (rs.next()) { int empId = rs.getInt("emp\_id");

String empName = rs.getString("emp\_name"); double baseSalary = rs.getDouble("base\_salary"); double bonus = rs.getDouble("bonus"); double deduction = rs.getDouble("deduction");

// Calculate the total salary

double totalSalary = baseSalary + bonus - deduction;

// Print the salary statement

System.out.printf("%-10d %-20s %-15.2f %-10.2f %-10.2f %-10.2f%n", empId, empName, baseSalary, bonus, deduction, totalSalary); }

} catch (SQLException e) {

// Handle SQL exceptions

System.err.println("SQL Error: " + e.getMessage()); e.printStackTrace();

} catch (ClassNotFoundException e) {

// Handle ClassNotFound exception (for JDBC Driver)

System.err.println("JDBC Driver not found: " + e.getMessage()); e.printStackTrace();

} catch (Exception e) {

// Catch any other exceptions

System.err.println("Unexpected error: " + e.getMessage()); e.printStackTrace();

} finally { try {

// 7. Close resources if (rs != null) rs.close(); if (pstmt != null) pstmt.close(); if (conn != null) conn.close();

} catch (SQLException e) {

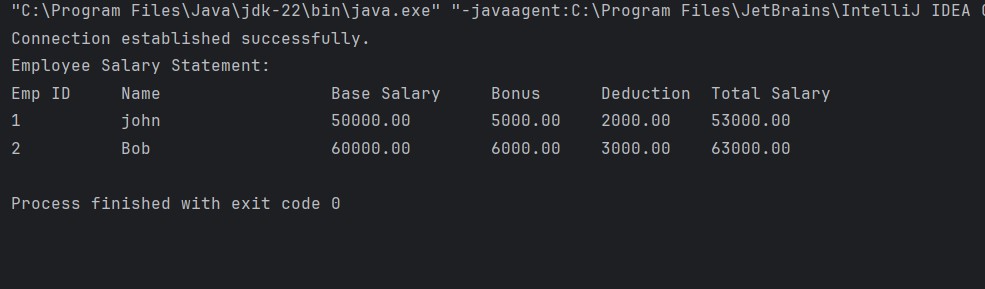
System.err.println("Error closing resources: " + e.getMessage()); e.printStackTrace();}

}

}

}

OUTPUT:



**1.5. Write a program to perform aggregation functions (e.g., COUNT, SUM,AVERAGE) on data retrieved from a database.**

**Ans:-** import java.sql.\*; public class practical5 {

public static void main(String[] args) {

String url = "jdbc:mysql://localhost:3306/company\_db"; // Replace with your database name

String username = "root"; // Replace with your MySQL username

String password = "1234567890"; // Replace with your MySQL password

Connection conn = null;

Statement stmt = null;

ResultSet rs = null;

try {

// 1. Register JDBC Driver

Class.forName("com.mysql.cj.jdbc.Driver");

// 2. Open Connection conn = DriverManager.getConnection(url, username, password); System.out.println("Connection established successfully.");

// 3. Create Statement stmt = conn.createStatement(); // 4. Perform Aggregation Queries

// Count the number of employees

String countSQL = "SELECT COUNT(\*) AS total\_employees FROM

Emp\_Sal"; rs = stmt.executeQuery(countSQL); if (rs.next()) { int totalEmployees = rs.getInt("total\_employees"); System.out.println("Total Employees: " + totalEmployees); }

// Sum of all salaries (Base salary + Bonus - Deduction)

String sumSQL = "SELECT SUM(base\_salary + bonus - deduction) AS total\_salary FROM Emp\_Sal"; rs = stmt.executeQuery(sumSQL);

if (rs.next()) { double totalSalary = rs.getDouble("total\_salary"); System.out.println("Total Salary Paid: " + totalSalary); }

// Average Salary (Base salary + Bonus - Deduction)

String avgSQL = "SELECT AVG(base\_salary + bonus - deduction) AS avg\_salary FROM Emp\_Sal"; rs = stmt.executeQuery(avgSQL); if (rs.next()) { double avgSalary = rs.getDouble("avg\_salary"); System.out.println("Average Salary: " + avgSalary); }

} catch (SQLException e) {

// Handle SQL exceptions

System.err.println("SQL Error: " + e.getMessage()); e.printStackTrace();

} catch (ClassNotFoundException e) {

// Handle ClassNotFoundException (for JDBC Driver)

System.err.println("JDBC Driver not found: " + e.getMessage()); e.printStackTrace();

} catch (Exception e) {

// Catch any other exceptions

System.err.println("Unexpected error: " + e.getMessage()); e.printStackTrace();

} finally { try {

// 5. Close resources if (rs != null) rs.close(); if (stmt != null) stmt.close(); if (conn != null) conn.close(); } catch (SQLException e) {

System.err.println("Error closing resources: " + e.getMessage());

e.printStackTrace();

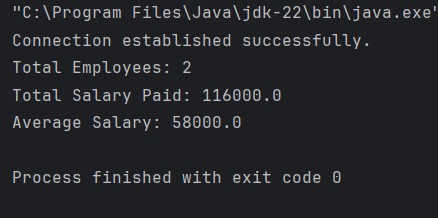
}

}

}

}

**OUTPUT:-**

` ` 

**1.6. Write a program to create a simple Java application that interacts with a database to perform CRUD operations (Create, Read, Update, Delete) on a specific table. Ans:-** import java.sql.\*; import java.util.Scanner;

public class practical6 {

// Database connection details

private static final String URL = "jdbc:mysql://localhost:3306/company\_db"; //

Replace with your database URL private static final String USER = "root"; // Replace with your MySQL

username private static final String PASSWORD = "1234567890"; // Replace with your

MySQL password

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

Connection conn = null;

PreparedStatement pstmt = null;

Statement stmt = null; ResultSet rs = null; try {

// Establish connection

conn = DriverManager.getConnection(URL, USER, PASSWORD); stmt = conn.createStatement();

// Show menu for CRUD operations

while (true) {

System.out.println("Choose an operation:");

System.out.println("1. Create (Insert) Employee"); System.out.println("2. Read (Select) Employees"); System.out.println("3. Update Employee Salary");

System.out.println("4. Delete Employee"); System.out.println("5. Exit"); int choice = scanner.nextInt();

switch (choice) { case 1: // Create (Insert) insertEmployee(conn, scanner); break;

case 2: // Read (Select) readEmployees(stmt);

break;

case 3: // Update updateEmployee(conn, scanner);

break;

case 4: // Delete deleteEmployee(conn, scanner);

break;

case 5: // Exit

System.out.println("Exiting..."); return;

default:

System.out.println("Invalid choice! Try again."); }

}

} catch (SQLException e) {

System.err.println("SQL Exception: " + e.getMessage());

} finally { try { if (rs != null) rs.close(); if (pstmt != null) pstmt.close(); if (stmt != null) stmt.close(); if (conn != null) conn.close();

} catch (SQLException e) {

System.err.println("Error closing resources: " + e.getMessage());

}

}

// Create (Insert)

private static void insertEmployee(Connection conn, Scanner scanner) { try {

System.out.print("Enter Employee Name: ");

String name = scanner.next();

System.out.print("Enter Base Salary: "); double baseSalary = scanner.nextDouble(); System.out.print("Enter Bonus: "); double bonus = scanner.nextDouble(); System.out.print("Enter Deduction: "); double deduction = scanner.nextDouble();

String insertSQL = "INSERT INTO Emp\_Sal (emp\_name, base\_salary, bonus, deduction) VALUES (?, ?, ?, ?)";

PreparedStatement pstmt = conn.prepareStatement(insertSQL); pstmt.setString(1, name); pstmt.setDouble(2, baseSalary); pstmt.setDouble(3, bonus); pstmt.setDouble(4, deduction); pstmt.executeUpdate();

System.out.println("Employee added successfully!");

} catch (SQLException e) {

System.err.println("Error while inserting: " + e.getMessage());

}

}

// Read (Select)

private static void readEmployees(Statement stmt) { try {

String selectSQL = "SELECT \* FROM Emp\_Sal"; ResultSet rs = stmt.executeQuery(selectSQL);

System.out.printf("%-10s %-20s %-15s %-10s %-10s%n", "Emp ID",

"Name", "Base Salary", "Bonus", "Deduction"); while (rs.next()) { int empId = rs.getInt("emp\_id"); String name = rs.getString("emp\_name"); double baseSalary = rs.getDouble("base\_salary"); double bonus = rs.getDouble("bonus"); double deduction = rs.getDouble("deduction");

System.out.printf("%-10d %-20s %-15.2f %-10.2f %-10.2f%n", empId,

name, baseSalary, bonus, deduction);

}

} catch (SQLException e) {

System.err.println("Error while reading: " + e.getMessage());

}

}

private static void updateEmployee(Connection conn, Scanner scanner) { try {

System.out.print("Enter Employee ID to Update: "); int empId = scanner.nextInt();

System.out.print("Enter New Base Salary: "); double baseSalary = scanner.nextDouble(); System.out.print("Enter New Bonus: "); double bonus = scanner.nextDouble(); System.out.print("Enter New Deduction: "); double deduction = scanner.nextDouble();

String updateSQL = "UPDATE Emp\_Sal SET base\_salary = ?, bonus = ?, deduction = ? WHERE emp\_id = ?";

PreparedStatement pstmt = conn.prepareStatement(updateSQL); pstmt.setDouble(1, baseSalary); pstmt.setDouble(2, bonus); pstmt.setDouble(3, deduction); pstmt.setInt(4, empId);

int rowsUpdated = pstmt.executeUpdate();

if (rowsUpdated > 0) {

System.out.println("Employee salary updated successfully!");

} else {

System.out.println("No employee found with that ID."); }

} catch (SQLException e) {

System.err.println("Error while updating: " + e.getMessage());

}

}

private static void deleteEmployee(Connection conn, Scanner scanner) { try {

System.out.print("Enter Employee ID to Delete: "); int empId = scanner.nextInt();

String deleteSQL = "DELETE FROM Emp\_Sal WHERE emp\_id = ?"; PreparedStatement pstmt = conn.prepareStatement(deleteSQL); pstmt.setInt(1, empId);

int rowsDeleted = pstmt.executeUpdate();

if (rowsDeleted > 0) {

System.out.println("Employee deleted successfully!");

} else {

System.out.println("No employee found with that ID."); }

} catch (SQLException e) {

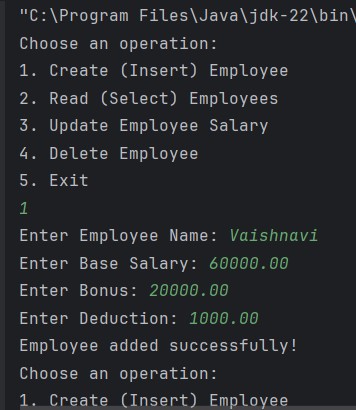
System.err.println("Error while deleting: " + e.getMessage());

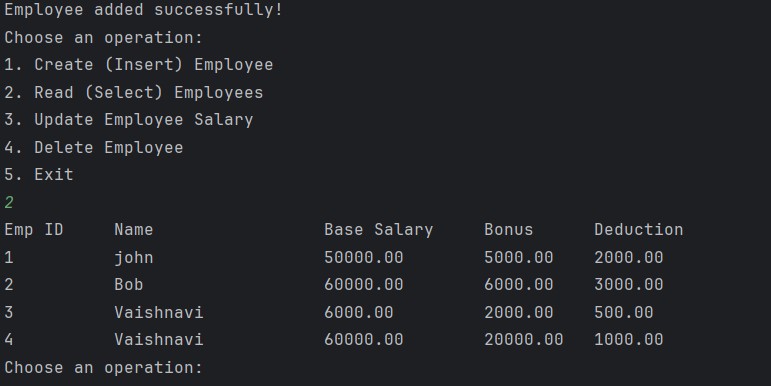
}

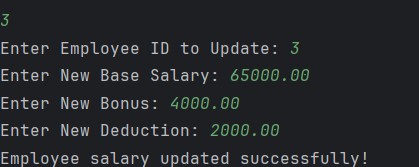
}

}

OUTPUT:-







**Assignment 2 : Programs On JAVA Servlet**

**1.Design a simple servlet that displays a welcome message with the user's name retrieved from request parameters Ans:-**

package com.example.servlet; import jakarta.servlet.ServletException; import jakarta.servlet.annotation.WebServlet; import jakarta.servlet.http.HttpServlet; import jakarta.servlet.http.HttpServletRequest; import jakarta.servlet.http.HttpServletResponse; import java.io.IOException;

/\*\*

* Servlet implementation class WelcomeServlet

\*/

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

/\*\*

* @see HttpServlet#HttpServlet()

\*/

public WelcomeServlet() { super();

}

/\*\*

* @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)

\*/

protected void doGet(HttpServletRequest request, HttpServletResponse

response) throws ServletException, IOException {

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

// Retrieve the user's name from request parameters String name = request.getParameter("name"); if (name == null || name.trim().isEmpty()) { name = "Guest"; // Default to "Guest" if no name is provided

}

// Generate a welcome message response.getWriter().append("<html><body>"); response.getWriter().append("<h1>Welcome, " + name + "!</h1>"); response.getWriter().append("</body></html>");

}

/\*\*

\* @see HttpServlet#doPost(HttpServletRequest request, HttpServletResponse response)

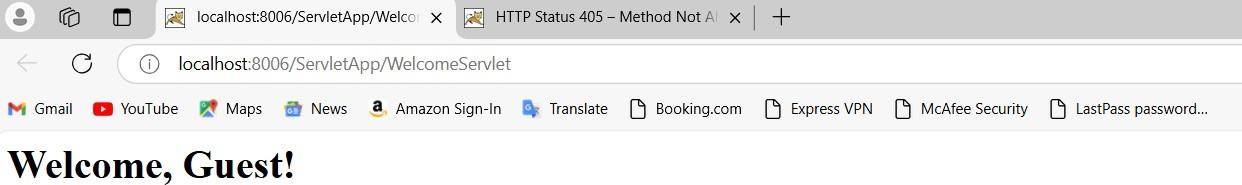
\*/

protected void doPost(HttpServletRequest request, HttpServletResponse

response) throws ServletException, IOException { doGet(request, response); // Reuse doGet for POST requests

}

}



**2 Implement a servlet that handles a login form and validates user credentials against a database.**

**Ans:-**

**LoginServlet.java package** com.loginapp;

**import** jakarta.servlet.ServletException; **import** jakarta.servlet.annotation.WebServlet; **import** jakarta.servlet.http.HttpServlet; **import** jakarta.servlet.http.HttpServletRequest; **import** jakarta.servlet.http.HttpServletResponse; **import** java.io.IOException; **import** java.io.PrintWriter; **import** java.sql.Connection; **import** java.sql.DriverManager; **import** java.sql.PreparedStatement; **import** java.sql.ResultSet;

//@WebServlet("/login") **public class** LoginServlet **extends** HttpServlet {

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

String username = request.getParameter("username");

String password = request.getParameter("password");

// Database credentials and connection details

String dbURL = "jdbc:mysql://localhost:3306/yourdb";

String dbUser = "yourusername";

String dbPass = "yourpassword";

// SQL query to check the credentials

String sql = "SELECT \* FROM users WHERE username = ? AND password = ?";

// Initialize response writer response.setContentType("text/html");

PrintWriter out = response.getWriter();

// Database connection and validation **try** {

// Connect to the database

Connection connection = DriverManager.*getConnection*(dbURL, dbUser, dbPass);

PreparedStatement stmt = connection.prepareStatement(sql); stmt.setString(1, username); stmt.setString(2, password);

// Execute query

ResultSet rs = stmt.executeQuery();

// Check if user exists **if** (rs.next()) {

// Successful login

out.println("<h2>Login Successful</h2>");

} **else** {

// Invalid credentials

out.println("<h2>Invalid Username or Password</h2>");

}

// Close the connection

rs.close(); stmt.close(); connection.close();

} **catch** (Exception e) { out.println("<h2>Error: " + e.getMessage() + "</h2>");

}

}

}

**Login.html**

<!DOCTYPE html>

<html lang=*"en"*>

<body>

<h2>Login</h2>

<form action=*"/LoginApp/login"* method=*"POST"*>

<label for=*"username"*>Username:</label><br>

<input type=*"text"* id=*"username"* name=*"username"* required><br><br>

<label for=*"password"*>Password:</label><br>

<input type=*"password"* id=*"password"* name=*"password"* required><br><br>

<input type=*"submit"* value=*"Login"*>

</form>

</body>

</html>

**Web.xml**

<element>

<web-app>

<servlet>

<servlet-name>LoginServlet</servlet-name>

<servlet-class>com.loginapp.LoginServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>LoginServlet</servlet-name>

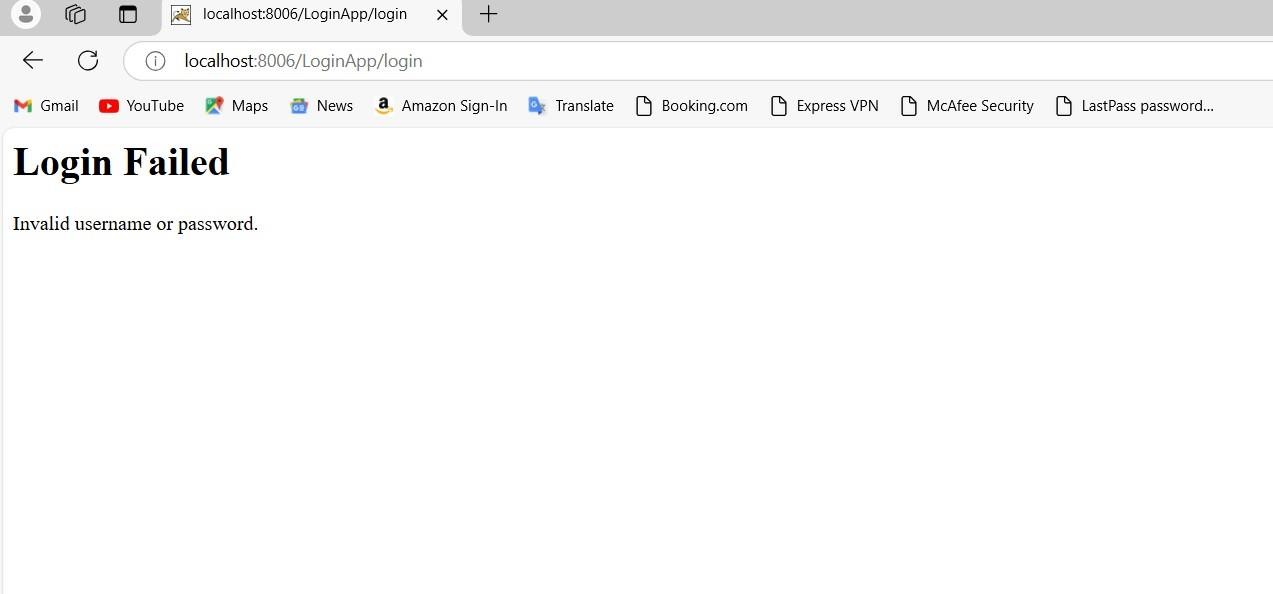
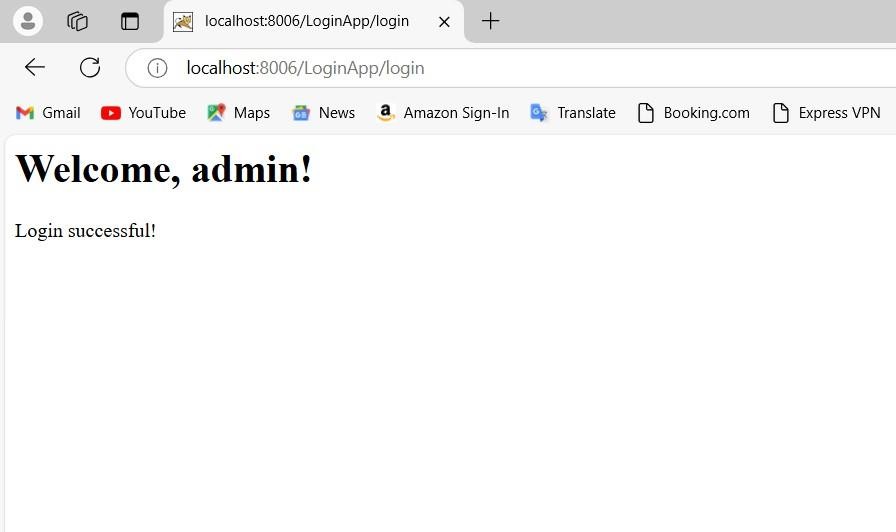
<url-pattern>/login</url-pattern>

</servlet-mapping>

</web-app>

</element>

**Outputs:-**



**3 Create a servlet that utilizes session management to maintain a shopping cart for an online store.**

**Ans:-**

**Loginservlet.java**

package com.shoppingcart;

import jakarta.servlet.\*; import jakarta.servlet.http.\*; import java.io.\*;

public class LoginServlet extends HttpServlet { protected void doPost(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

String username = request.getParameter("username");

String password = request.getParameter("password"); // Simple validation (use database for production)

if ("admin".equals(username) && "password123".equals(password)) {

HttpSession session = request.getSession(); session.setAttribute("user", username);

response.sendRedirect("cart"); // Redirect to the shopping cart page

} else { response.sendRedirect("login.html"); // Redirect to login page if

authentication fails

}

}

}

**Cartservleyt.java**

**package** com.shoppingcart; **import** jakarta.servlet.\*; **import** jakarta.servlet.http.\*; **import** java.io.\*; **import** java.util.\*;

**public class** CartServlet **extends** HttpServlet {

@SuppressWarnings("unchecked") **protected void** doGet(HttpServletRequest request, HttpServletResponse

response) **throws** ServletException, IOException { HttpSession session = request.getSession(**false**); **if** (session == **null** || session.getAttribute("user") == **null**) { response.sendRedirect("login.html"); **return**;

}

// Fetch the shopping cart from the session

List<String> cart = (List<String>) session.getAttribute("cart"); **if** (cart == **null**) { cart = **new** ArrayList<>(); session.setAttribute("cart", cart);

}

// Display the shopping cart response.setContentType("text/html"); PrintWriter out = response.getWriter(); out.println("<html><body>");

out.println("<h1>Welcome, " + session.getAttribute("user") + "</h1>"); out.println("<h3>Your Shopping Cart</h3>"); out.println("<table border='1'><tr><th>Product</th><th>Action</th></tr>"); **for** (String product : cart) { out.println("<tr><td>" + product + "</td><td><a href='cart?remove=" +

product + "'>Remove</a></td></tr>");

}

out.println("</table>");

out.println("<br><a href='index.html'>Continue Shopping</a>"); out.println("</body></html>");

}

@SuppressWarnings("unchecked") **protected void** doPost(HttpServletRequest request, HttpServletResponse

response) **throws** ServletException, IOException {

// Adding item to cart

String product = request.getParameter("product");

HttpSession session = request.getSession();

List<String> cart = (List<String>) session.getAttribute("cart"); **if** (cart == **null**) { cart = **new** ArrayList<>(); session.setAttribute("cart", cart); }

**if** (product != **null**) { cart.add(product); // Add selected product to the cart }

response.sendRedirect("cart"); // Redirect to the cart page to view updated cart

}

}

**Index.html**

<!DOCTYPE html>

<html lang=*"en"*>

<head>

<meta charset=*"UTF-8"*>

<meta name=*"viewport"* content=*"width=device-width, initial-scale=1.0"*>

<title>Welcome to the Online Store</title>

</head>

<body>

<h1>Welcome to the Online Store</h1>

<h3>Products</h3>

<ul>

<li>Product A - $10 <a href=*"cart?product=Product A"*>Add to

Cart</a></li>

<li>Product B - $20 <a href=*"cart?product=Product B"*>Add to

Cart</a></li>

<li>Product C - $30 <a href=*"cart?product=Product C"*>Add to

Cart</a></li>

</ul>

<a href=*"login.html"*>Login</a>

</body>

</html>

**Login.html**

<!DOCTYPE html>

<html lang=*"en"*>

<head>

<meta charset=*"UTF-8"*>

<meta name=*"viewport"* content=*"width=device-width, initial-scale=1.0"*> <title>Login</title>

</head>

<body>

<h1>Login to Your Account</h1>

<form action=*"login"* method=*"POST"*>

<label for=*"username"*>Username:</label>

<input type=*"text"* id=*"username"* name=*"username"* required><br><br>

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

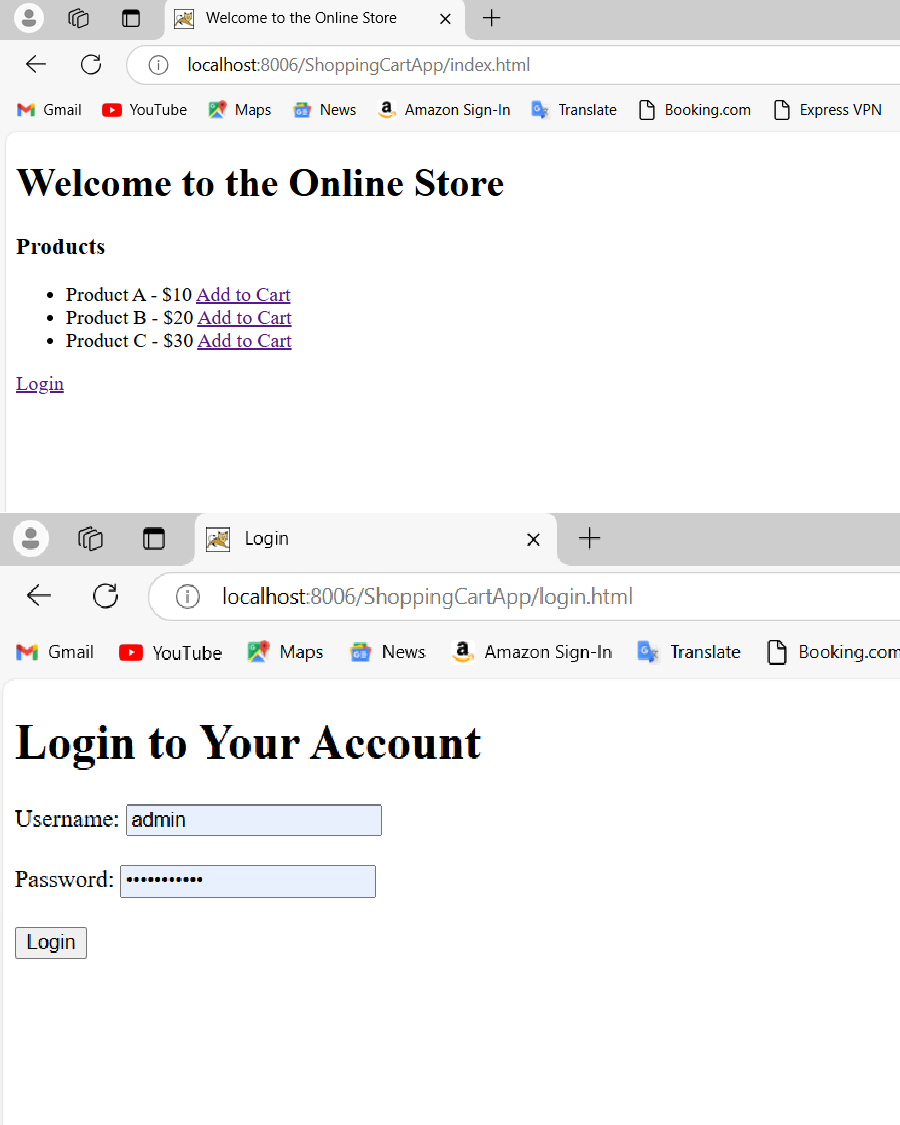
<input type=*"password"* id=*"password"* name=*"password"* required><br><br>

<button type=*"submit"*>Login</button>

</form>

</body>

</html>



**4 Write a servlet Program to calculate the addition of two numbers and print the result.(Eg:Addition of two numbers=50)**

**Ans:-**

**Additionservlet.java package** com.addition; **import** jakarta.servlet.\*;

**import** jakarta.servlet.annotation.WebServlet;

**import** jakarta.servlet.http.\*; **import** java.io.\*;

@WebServlet("/AdditionServlet") **public class** AdditionServlet **extends** HttpServlet {

@Override

**protected void** doPost(HttpServletRequest request, HttpServletResponse

response) **throws** ServletException, IOException {

String num1Str = request.getParameter("num1");

String num2Str = request.getParameter("num2");

**int** num1 = Integer.*parseInt*(num1Str); **int** num2 = Integer.*parseInt*(num2Str);

**int** sum = num1 + num2;

response.setContentType("text/html"); PrintWriter out = response.getWriter();

out.println("<html><body>"); out.println("<h2>Result</h2>");

out.println("Addition of " + num1 + " and " + num2 + " = " + sum); out.println("<br><br>");

out.println("<a href='addition.html'>Go back</a>"); out.println("</body></html>");

}

}

**Addition.html**

<!DOCTYPE html>

<html lang=*"en"*>

<head>

<meta charset=*"UTF-8"*>

<meta name=*"viewport"* content=*"width=device-width, initial-scale=1.0"*> <title>Addition of Two Numbers</title>

</head>

<body>

<h2>Enter Two Numbers to Add</h2>

<form action=*"AdditionServlet"* method=*"POST"*>

<label for=*"num1"*>Number 1:</label>

<input type=*"number"* id=*"num1"* name=*"num1"* required><br><br>

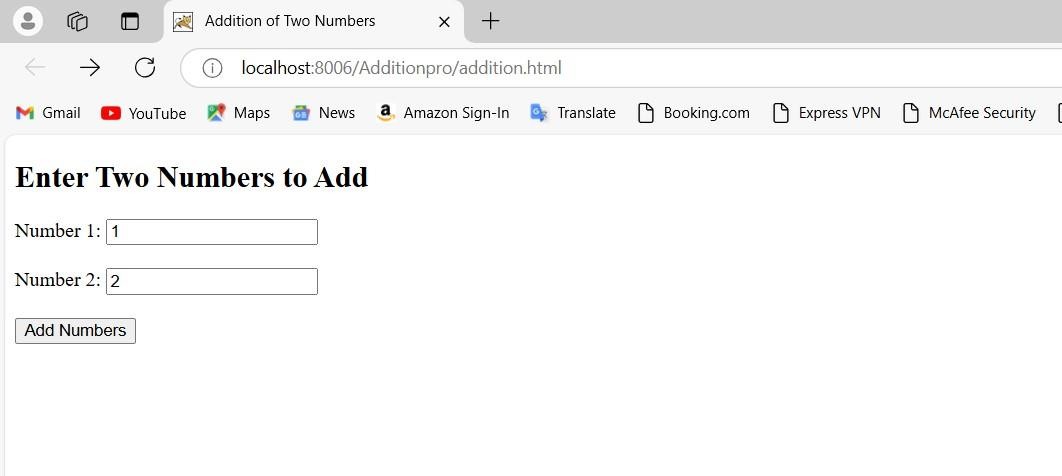
<label for=*"num2"*>Number 2:</label>

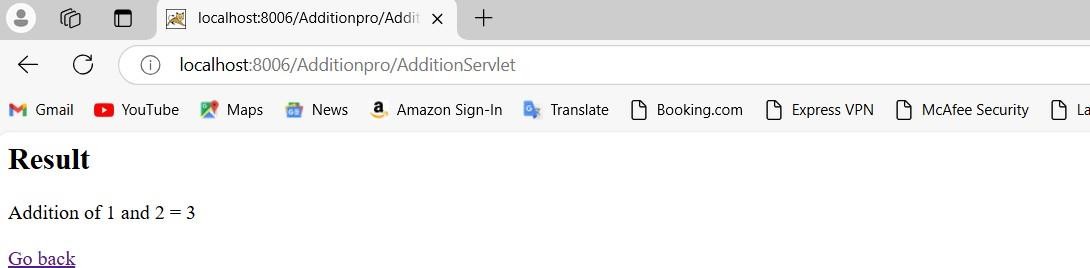
<input type=*"number"* id=*"num2"* name=*"num2"* required><br><br>

<input type=*"submit"* value=*"Add Numbers"*> </form>

</body>

</html>





**5. Write a Servlet Program to create a registration form using in html and CSS and print the message Registration is successful Ans:-**

**RegistrationServlet.java package** com.registration; **import** jakarta.servlet.\*;

**import** jakarta.servlet.annotation.WebServlet;

**import** jakarta.servlet.http.\*; **import** java.io.\*;

@WebServlet("/RegistrationServlet") **public class** RegistrationServlet **extends** HttpServlet {

@Override

**protected void** doPost(HttpServletRequest request, HttpServletResponse

response) **throws** ServletException, IOException {

// Get form data

String name = request.getParameter("name");

String email = request.getParameter("email");

String password = request.getParameter("password");

// Process registration (you can store it in a database or session, for now we just show success)

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

PrintWriter out = response.getWriter();

// Display success message out.println("<html><body>");

out.println("<h2>Registration Successful!</h2>");

out.println("<p>Thank you, " + name + "! Your registration was

successful.</p>"); out.println("<br><br>");

out.println("<a href='register.html'>Go back to Registration</a>"); out.println("</body></html>");

}

}

**Register.html**

<!DOCTYPE html>

<html lang=*"en"*>

<head>

<meta charset=*"UTF-8"*>

<meta name=*"viewport"* content=*"width=device-width, initial-scale=1.0"*> <title>User Registration</title>

<link rel=*"stylesheet"* href=*"style.css"*>

</head>

<body>

<div class=*"container"*>

<h2>User Registration</h2>

<form action=*"RegistrationServlet"* method=*"POST"*> <label for=*"name"*>Full Name:</label>

<input type=*"text"* id=*"name"* name=*"name"* required><br><br>

<label for=*"email"*>Email:</label>

<input type=*"email"* id=*"email"* name=*"email"* required><br><br>

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

<input type=*"password"* id=*"password"* name=*"password"* required><br><br>

<input type=*"submit"* value=*"Register"*>

</form>

</div>

</body>

</html>

**Style.css body** { font-family: *Arial, sans-serif*; background-color: *#f4f4f4*; display: *flex*; justify-content: *center*; align-items: *center*; height: *100vh*;

margin: *0*;

}

*.container* { background-color: *#fff*; padding: *20px*; border-radius: *8px*; box-shadow: *0 2px 10px rgba(0, 0, 0, 0.1)*; width: *300px*;

}

**h2** { text-align: *center*; margin-bottom: *20px*;

}

**label** { font-weight: *bold*;

}

**input**[type="text"]**, input**[type="email"]**, input**[type="password"] { width: *100%*; padding: *10px*; margin: *8px 0*;

border: *1px solid #ccc*;

border-radius: *4px*;

}

**input**[type="submit"] {

width: *100%*; padding: *10px*; background-color: *#4CAF50*;

color: *white*; border: *none*; border-radius: *4px*;

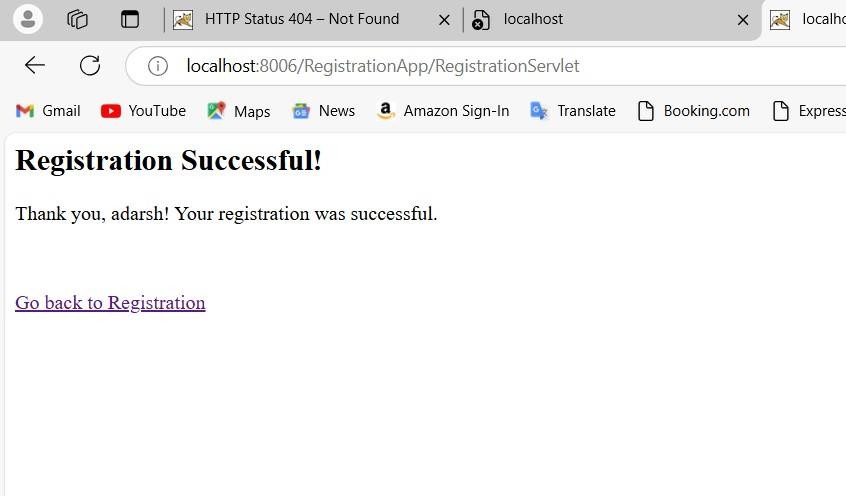
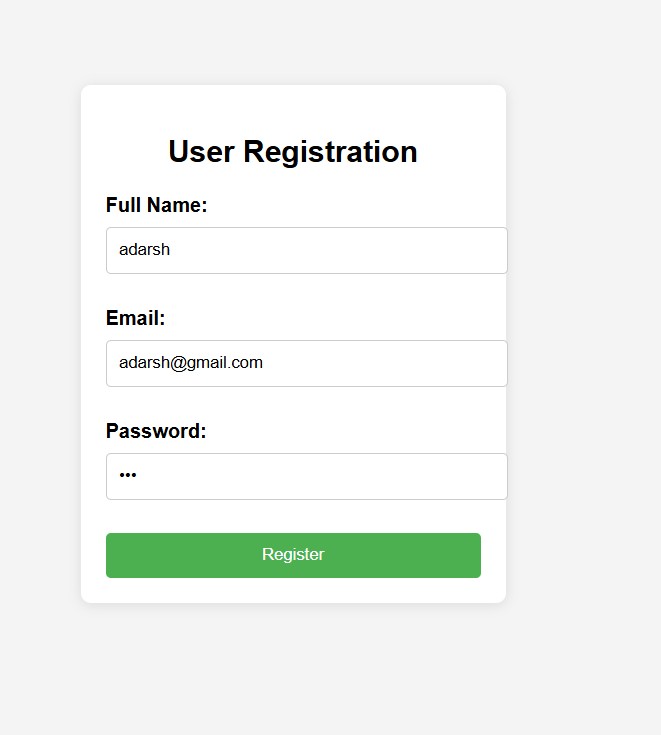
cursor: *pointer*;

}

**input**[type="submit"]*:hover* {

background-color: *#45a049*;

}



## Assignment 3: Java Server Pages(JSP)

**1. Write a JSP program calculates factorial of an integer number, while the input is taken from an HTML form**

**Ans:-**

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"%>

<%@ page import="java.math.BigInteger" %>

<!DOCTYPE html>

<html>

<head>

<title>Factorial Calculator</title>

</head>

<body>

<h2>Enter a number to calculate its factorial:</h2>

<form action="" method="POST">

<input type="number" name="number" placeholder="Enter a number" required>

<button type="submit">Calculate Factorial</button> </form>

<%

// Get the number from the request parameter

String numberStr = request.getParameter("number");

// Check if the number parameter is provided if (numberStr != null && !numberStr.isEmpty()) { try {

// Convert the input to an integer int number = Integer.parseInt(numberStr);

// Initialize the factorial result as 1

BigInteger factorial = BigInteger.ONE;

// Loop to calculate the factorial for (int i = 1; i <= number; i++) { factorial = factorial.multiply(BigInteger.valueOf(i)); }

// Display the result

out.println("<h3>Factorial of " + number + " is: " + factorial.toString() + "</h3>");

} catch (NumberFormatException e) { out.println("<h3>Please enter a valid integer.</h3>"); }

out.println("<h3>Please enter a number to calculate its factorial.</h3>"); }

%>

</body>

</html>

**OUTPUT:**



**2. Write a JSP program to generate the Fibonacci series up to a particular term, while the input is taken from an HTML form**.

**Ans:-**

<%@ page language="java" contentType="text/html; charset=ISO-8859-1"%>

<!DOCTYPE html>

<html>

<head>

<title>Fibonacci Series Generator</title>

</head>

<body>

<h2>Enter the number of terms for the Fibonacci Series:</h2>

<form action="" method="POST">

<input type="number" name="terms" placeholder="Enter number of terms" required>

<button type="submit">Generate Fibonacci Series</button> </form>

<%

// Get the number of terms from the request parameter String termsStr = request.getParameter("terms");

// Check if the terms parameter is provided if (termsStr != null && !termsStr.isEmpty()) { try {

// Convert the input to an integer int terms = Integer.parseInt(termsStr); // Initialize the first two Fibonacci numbers long first = 0, second = 1;

// Print the Fibonacci series

out.println("<h3>Fibonacci Series up to " + terms + " terms:</h3>"); out.println("<ul>"); for (int i = 1; i <= terms; i++) { out.println("<li>" + first + "</li>");

long next = first + second; // next number in the series first = second; second = next;

}

out.println("</ul>");

} catch (NumberFormatException e) { out.println("<h3>Please enter a valid integer.</h3>"); }

} else { out.println("<h3>Please enter a number to generate the Fibonacci

series.</h3>");

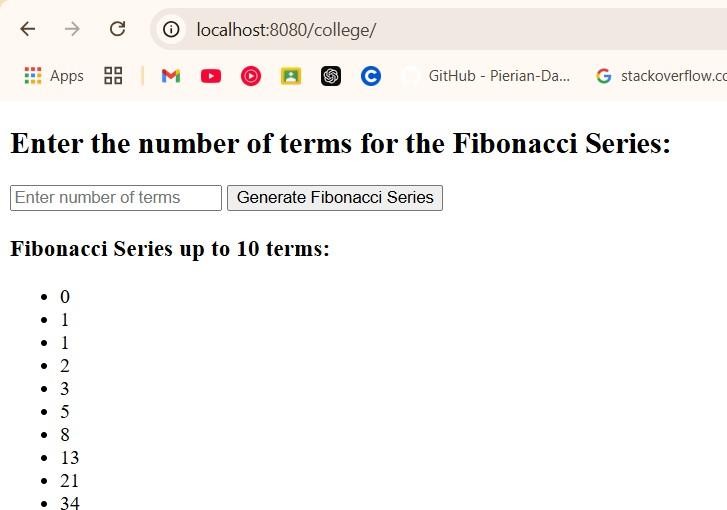
}

%>

</body>

</html>

**OUTPUT:**



**3. Write a JSP program to display the System date and time**.

**Ans:-**

<%@ page language="java" contentType="text/html; charset=ISO-8859-1" %>

<!DOCTYPE html>

<html>

<head>

<title>Current Date and Time</title>

</head>

<body>

<h2>Current Date and Time</h2>

<%

// Get the current system date and time java.util.Date date = new java.util.Date();

// Display the current date and time

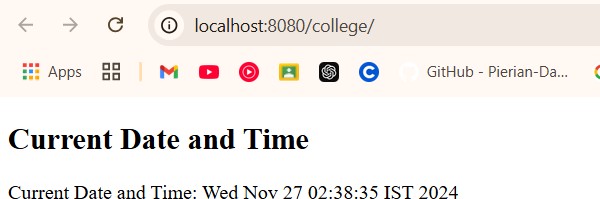
out.println("<p>Current Date and Time: " + date.toString() + "</p>");

%>

</body>

</html>

**OUTPUT:**



**4. Write a JSP program to display a**

**Sample shopping Order calculation Form and display output in tabular form.**

**Ans:-**

<%@ page language="java" contentType="text/html; charset=ISO-8859-1" %>

<!DOCTYPE html>

<html>

<head>

<title>Shopping Order Calculation</title>

<style> table { width: 60%; margin: 20px;

border-collapse: collapse;

}

table, th, td {

border: 1px solid black;

} th, td { padding: 10px; text-align: center;

}

</style>

</head>

<body>

<h2>Shopping Order Calculation</h2>

<!-- Shopping Form -->

<form action="order.jsp" method="post">

<table>

<tr>

<td>Item</td>

<td>Price</td>

<td>Quantity</td>

</tr>

<tr>

<td>Item 1 - Laptop</td>

<td>$500</td>

<td><input type="number" name="item1" value="0" min="0" /></td> </tr>

<tr>

<td>Item 2 - Headphones</td>

<td>$50</td>

<td><input type="number" name="item2" value="0" min="0" /></td> </tr>

<tr>

<td>Item 3 - Mouse</td>

<td>$20</td>

<td><input type="number" name="item3" value="0" min="0" /></td>

</tr>

<tr>

<td>Item 4 - Keyboard</td>

<td>$30</td>

<td><input type="number" name="item4" value="0" min="0" /></td>

</tr>

<tr>

<td colspan="3" style="text-align: center;">

<input type="submit" value="Calculate Order" /> </td>

</tr>

</table>

</form>

<%

// Retrieving form values and calculating order total

String item1Qty = request.getParameter("item1");

String item2Qty = request.getParameter("item2"); String item3Qty = request.getParameter("item3");

if (item1Qty != null && item2Qty != null && item3Qty != null && item4Qty

!= null) {

// Converting to integers int item1 = Integer.parseInt(item1Qty); int item2 = Integer.parseInt(item2Qty); int item3 = Integer.parseInt(item3Qty); int item4 = Integer.parseInt(item4Qty);

// Prices int priceItem1 = 500; int priceItem2 = 50; int priceItem3 = 20; int priceItem4 = 30;

// Calculating total cost for each item int totalItem1 = item1 \* priceItem1; int totalItem2 = item2 \* priceItem2; int totalItem3 = item3 \* priceItem3; int totalItem4 = item4 \* priceItem4; // Calculating final order total

int totalOrder = totalItem1 + totalItem2 + totalItem3 + totalItem4;

%>

<!-- Displaying the order summary in tabular form -->

<h3>Your Order Summary</h3>

<table>

<tr>

<th>Item</th>

<th>Quantity</th>

<th>Price</th>

<th>Total</th>

</tr>

<tr>

<td>Item 1 - Laptop</td>

<td><%= item1 %></td>

<td>$500</td>

<td>$<%= totalItem1 %></td>

</tr>

<tr>

<td>Item 2 - Headphones</td>

<td><%= item2 %></td>

<td>$50</td>

<td>$<%= totalItem2 %></td>

</tr>

<tr>

<td>Item 3 - Mouse</td> <td><%= item3 %></td>

<td>$20</td>

<td>$<%= totalItem3 %></td>

</tr>

<tr>

<td>Item 4 - Keyboard</td>

<td><%= item4 %></td>

<td>$30</td>

<td>$<%= totalItem4 %></td>

</tr>

<tr>

<td colspan="3"><strong>Total Order Cost</strong></td>

<td><strong>$<%= totalOrder %></strong></td> </tr>

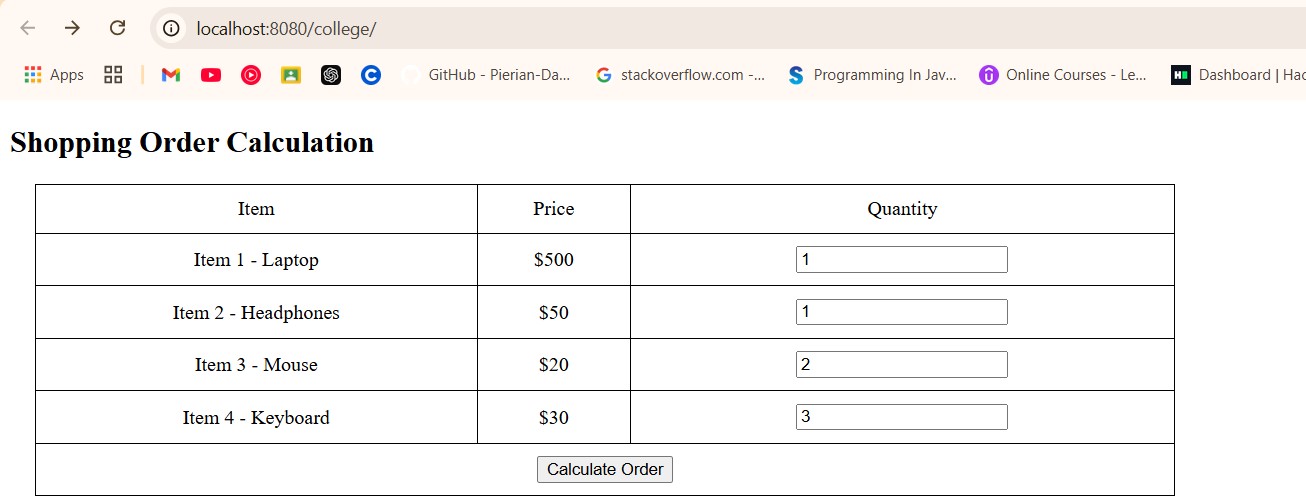
</table>

<% } %>

</body>

</html>

**OUTPUT:**



**5. Write a JSP program to perform Arithmetic operations such as Addition, Subtraction, Multiplication and Division. Design a HTML to accept two numbers in text box and radio buttons to display operations. On submit display**

**result as per the selected operation on next page using JSP**

**Ans:-**

<%@ page language="java" contentType="text/html; charset=ISO-8859-1" pageEncoding="ISO-8859-1"%>

<!DOCTYPE html>

<html>

<head>

<title>Arithmetic Operations</title>

</head>

<body>

<h2>Arithmetic Operations - JSP Program</h2>

<form method="post">

<!-- Input Fields -->

<label for="num1">Enter Number 1:</label>

<input type="number" name="num1" required><br><br>

<label for="num2">Enter Number 2:</label>

<input type="number" name="num2" required><br><br>

<!-- Radio Buttons for Operations -->

<label>Select Operation:</label><br>

<input type="radio" name="operation" value="addition" required>

Addition<br>

<input type="radio" name="operation" value="subtraction"> Subtraction<br>

<input type="radio" name="operation" value="multiplication">

Multiplication<br>

<input type="radio" name="operation" value="division"> Division<br><br>

<input type="submit" value="Calculate">

</form>

<%

// Only perform calculation if the form is submitted if (request.getMethod().equalsIgnoreCase("POST")) {

// Retrieve numbers and operation from the form

String num1Str = request.getParameter("num1");

String num2Str = request.getParameter("num2");

String operation = request.getParameter("operation");

// Convert input values to numbers double num1 = Double.parseDouble(num1Str); double num2 = Double.parseDouble(num2Str); double result = 0; String errorMessage = "";

// Perform arithmetic operation based on the selected radio button switch (operation) { case "addition":

result = num1 + num2;

break; case "subtraction": result = num1 - num2; break;

case "multiplication": result = num1 \* num2;

break;

case "division":

if (num2 != 0) { result = num1 / num2;

} else { errorMessage = "Error: Division by zero is not allowed!";

} break;

default:

errorMessage = "Invalid operation."; }

// Display the result or error message if (errorMessage.isEmpty()) {

%>

<h3>Result of <%= operation %>:</h3>

<p><%= num1 %> <%= (operation.equals("addition") ? "+" :

operation.equals("subtraction") ? "-" : operation.equals("multiplication") ? "\*" : "/")

%> <%= num2 %> = <%= result %></p>

<%

} else {

%>

<h3><%= errorMessage %></h3> <%

}

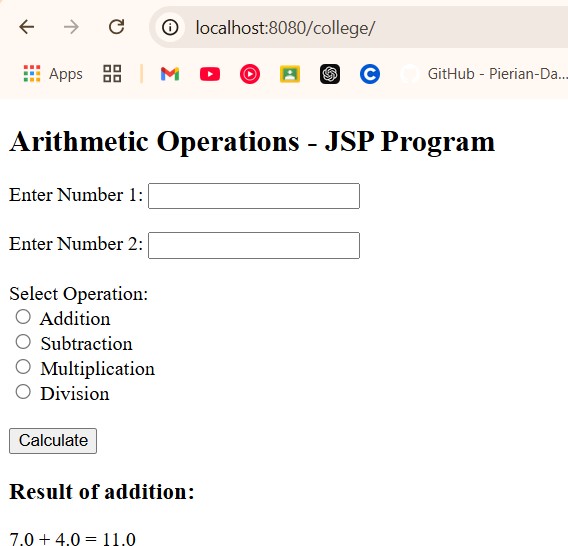
}

%>

</body>

</html>

**OUTPUT:**



**6. Write a servlet Program for student information and display the information in tabular form by selecting the details from student database table. Studt.java Ans:-** package com.example; import java.io.PrintWriter; import java.sql.Connection; import java.sql.DriverManager; import java.sql.ResultSet; import java.sql.Statement; import jakarta.servlet.ServletException; import jakarta.servlet.annotation.WebServlet; import jakarta.servlet.http.HttpServlet; import jakarta.servlet.http.HttpServletRequest; import jakarta.servlet.http.HttpServletResponse;

import java.io.IOException; import java.sql.SQLException;

/\*\*

\* Servlet implementation class StudentInfoServlet \*/

@WebServlet("/studentInfo")

public class StudentInfoServlet extends HttpServlet { private static final long serialVersionUID = 1L; @Override

protected void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException { response.setContentType("text/html"); PrintWriter out = response.getWriter();

// JDBC setup

String jdbcURL = "jdbc:mysql://localhost:3306/student\_db";

String jdbcUsername = "root";

String jdbcPassword = "Rohit@0801"; // Replace with your MySQL password

try {

// Establish connection

Class.forName("com.mysql.cj.jdbc.Driver");

Connection connection = DriverManager.getConnection(jdbcURL, jdbcUsername, jdbcPassword);

// Query student details

String sql = "SELECT \* FROM students";

Statement statement = connection.createStatement();

ResultSet resultSet = statement.executeQuery(sql);

// Display student details in a table out.println("<html><head><title>Student

Information</title></head><body>"); out.println("<h1>Student Information</h1>"); out.println("<table border='1' cellpadding='10'>");

out.println("<tr><th>ID</th><th>Name</th><th>Age</th><th>Grade</th><th>Em ail</th></tr>");

while (resultSet.next()) { int id = resultSet.getInt("id");

String name = resultSet.getString("name"); int age = resultSet.getInt("age");

String grade = resultSet.getString("grade"); String email = resultSet.getString("email"); out.println("<tr>"); out.println("<td>" + id + "</td>"); out.println("<td>" + name + "</td>"); out.println("<td>" + age + "</td>"); out.println("<td>" + grade + "</td>"); out.println("<td>" + email + "</td>"); out.println("</tr>");

}

out.println("</table>"); out.println("</body></html>");

resultSet.close(); statement.close(); connection.close();

} catch (Exception e) {

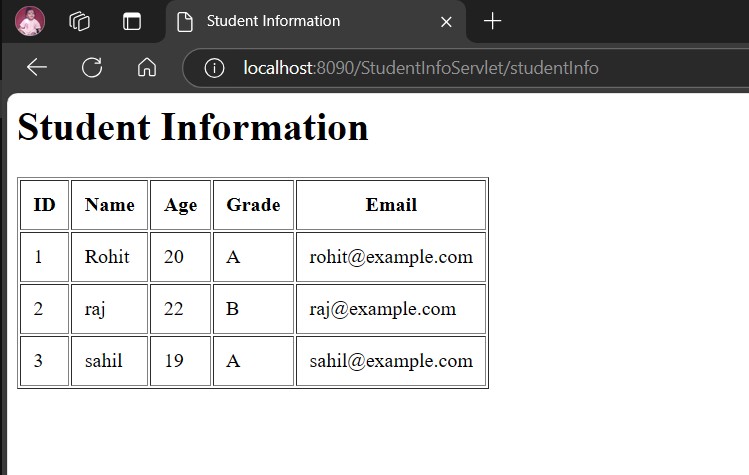
e.printStackTrace(); out.println("<p>Error: Unable to fetch data from the database.</p>"); out.println("<p>Details: " + e + "</p>");

}

}

}

**Output:-**



**7. Write a Java Servlet program to read employee details including employee number (empno), name, designation, basic pay, deductions, and allowances, and then calculate and display the net salary. display the information in tabular form by selecting the details from Emp\_sal database table.**

**Ans:- Emp.java** package com.example; import java.io.IOException; import java.io.PrintWriter; import java.sql.Connection; import java.sql.DriverManager; import java.sql.ResultSet; import java.sql.Statement; import jakarta.servlet.ServletException; import jakarta.servlet.annotation.WebServlet; import jakarta.servlet.http.HttpServlet; import jakarta.servlet.http.HttpServletRequest; import jakarta.servlet.http.HttpServletResponse; import java.io.IOException;

/\*\*

\* Servlet implementation class EmployeeServlet

\*/

@WebServlet("/employeeDetails")

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

@Override

protected void doGet(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException { response.setContentType("text/html"); PrintWriter out = response.getWriter();

// Database credentials

String jdbcURL = "jdbc:mysql://localhost:3306/employee\_db";

String jdbcUsername = "root";

String jdbcPassword = "Rohit@0801"; // Replace with your MySQL password

try {

// Load JDBC driver

Class.forName("com.mysql.cj.jdbc.Driver");

// Establish connection

Connection connection = DriverManager.getConnection(jdbcURL, jdbcUsername, jdbcPassword);

// Query the employee salary details

String sql = "SELECT \* FROM Emp\_sal";

Statement statement = connection.createStatement();

ResultSet resultSet = statement.executeQuery(sql);

// Display employee details in a table

out.println("<html><head><title>Employee Details</title></head><body>"); out.println("<h1>Employee Salary Details</h1>"); out.println("<table border='1' cellpadding='10'>"); out.println("<tr><th>Emp

No</th><th>Name</th><th>Designation</th><th>Basic

Pay</th><th>Deductions</th><th>Allowances</th><th>Net Salary</th></tr>"); while (resultSet.next()) { int empno = resultSet.getInt("empno");

String name = resultSet.getString("name");

String designation = resultSet.getString("designation"); double basicPay = resultSet.getDouble("basic\_pay"); double deductions = resultSet.getDouble("deductions"); double allowances = resultSet.getDouble("allowances");

// Calculate net salary

double netSalary = basicPay + allowances - deductions;

// Display employee data out.println("<tr>"); out.println("<td>" + empno + "</td>"); out.println("<td>" + name + "</td>"); out.println("<td>" + designation + "</td>"); out.println("<td>" + basicPay + "</td>"); out.println("<td>" + deductions + "</td>"); out.println("<td>" + allowances + "</td>"); out.println("<td>" + netSalary + "</td>"); out.println("</tr>");

}

out.println("</table>"); out.println("</body></html>");

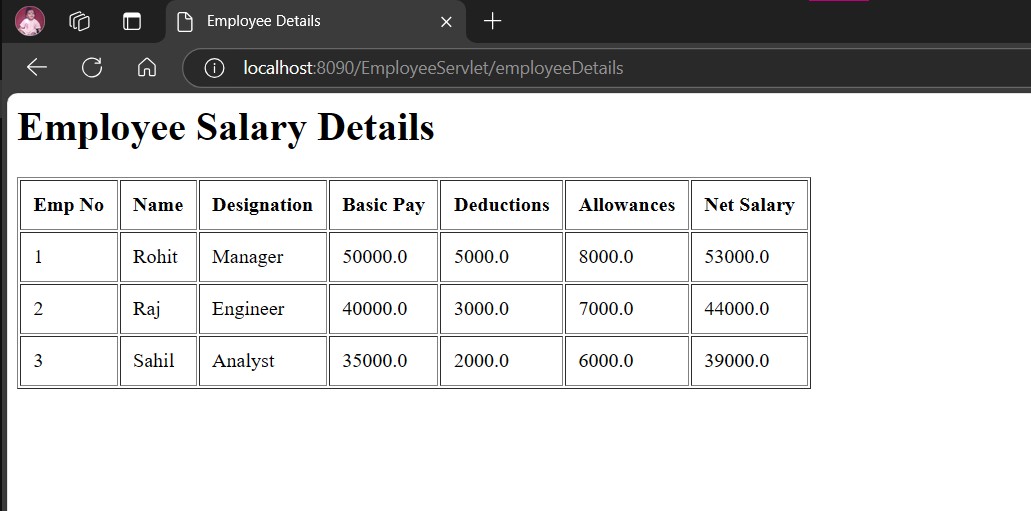
resultSet.close(); statement.close(); connection.close();

} catch (Exception e) {

e.printStackTrace(); out.println("<p>Error: Unable to fetch employee details.</p>"); out.println("<p>Details: " + e.getMessage() + "</p>");

}}}

**Output:-**



## Assignment 4: Java Persistence API

**4.1 Define and illustrate the concept of entity mapping in JPA.Explain how JPA maps Java classes (entities) to database tables.Provide an example of an entity class with annotations and its corresponding database table schema**

**ANS:**

**Entity Mapping in JPA (Java Persistence API)**

**Entity Mapping** in JPA refers to the process of linking a Java class (often called an **entity class**) to a database table. This mapping allows Java objects to be stored in and retrieved from a relational database. JPA provides a set of annotations to specify how the fields of the Java class correspond to the columns in the database table.

**How JPA Maps Java Classes to Database Tables**

1. **Entity Class**:

o An entity class in JPA is a Java class that is mapped to a database table. o Each instance of the class represents a row in the corresponding table. o The class must be annotated with the @Entity annotation to indicate that it is an entity.

2. **Primary Key**:

o Every entity class must have a primary key, which uniquely identifies each row. This is typically represented by a field annotated with @Id. o The @GeneratedValue annotation can be used to auto-generate the primary key values.

3. **Field to Column Mapping**:

o Fields in the Java class represent columns in the database table. By default, JPA assumes that the field names correspond to column names, but this can be customized using the @Column annotation.

4. **Table Mapping**:

o The @Table annotation allows you to specify the table name in the database if it differs from the class name.

5. **Relationships**:

oJPA also supports mapping relationships between entities, such as One-to- One, One-to-Many, Many-to-One, and Many-to-Many, using annotations like @OneToMany, @ManyToOne, etc.

**Example of an Entity Class with Annotations**

Let's consider an entity class called Customer, which is mapped to a customers table in the database.

**Java Class (Entity):**

import javax.persistence.\*;

@Entity

@Table(name = "customers") // Specifies the table name in the database public class Customer {

@Id // Marks this field as the primary key

@GeneratedValue(strategy = GenerationType.IDENTITY) // Auto-generate the primary key value

private Long id;

@Column(name = "first\_name", nullable = false) // Maps this field to the

'first\_name' column in the table

private String firstName;

@Column(name = "last\_name", nullable = false) // Maps this field to the

'last\_name' column in the table

private String lastName;

@Column(name = "email", unique = true) // Maps this field to the 'email' column in the table

private String email;

// Constructors, Getters, and Setters

public Customer() {}

public Customer(String firstName, String lastName, String email) { this.firstName = firstName; this.lastName = lastName; this.email = email;

}

public Long getId() { return id;

}

public void setId(Long id) { this.id = id;

}

public String getFirstName() { return firstName;

}

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

}

public String getLastName() { return lastName;

}

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

}

public String getEmail() { return email;

}

public void setEmail(String email) { this.email = email;

}

}

**Explanation of the Annotations:**

1. **@Entity**: Specifies that the class is an entity and should be mapped to a database table.
2. **@Table(name = "customers")**: Maps the Customer class to the customers table in the database.
3. **@Id**: Specifies the field id as the primary key of the entity.
4. **@GeneratedValue(strategy = GenerationType.IDENTITY)**: Configures the primary key to be generated automatically using an identity column (auto- increment).
5. **@Column**: Used to specify column details (e.g., nullable, unique).
   * name: Specifies the column name in the database.
   * nullable: Indicates whether the column can accept null values.
   * unique: Ensures that values in this column are unique.

**Corresponding Database Table Schema**

After mapping the Customer class to the customers table, the corresponding database table schema would look like this:

CREATE TABLE customers ( id BIGINT AUTO\_INCREMENT PRIMARY KEY, -- Maps to @Id and

@GeneratedValue first\_name VARCHAR(100) NOT NULL, -- Maps to @Column(name =

"first\_name")

last\_name VARCHAR(100) NOT NULL, -- Maps to @Column(name =

"last\_name")

email VARCHAR(100) UNIQUE -- Maps to @Column(name = "email")

);

**4.2 Describe the different types of relationships between entities (one-to- one, one-to- many, many-to-one, many-to-many).**

* **Explain how JPA represents these relationships using annotations.**
* **Provide code examples for each type of relationship.**

**ANS:**

**Different Types of Relationships between Entities in JPA**

In Java Persistence API (JPA), entities can be related to each other in different ways. These relationships help model real-world associations between objects and allow for complex data structures in relational databases. JPA provides annotations to define these relationships. The four main types of relationships between entities in JPA are:

1. One-to-One (1:1)
2. One-to-Many (1:M)
3. Many-to-One (M:1)
4. Many-to-Many (M:M)

Each of these relationships can be mapped using JPA annotations to represent the database schema.

**1. One-to-One Relationship (1:1)**

A **one-to-one relationship** means that one entity is associated with exactly one other entity. For example, a **Person** might have one **Passport**.

**JPA Representation:**

* **@OneToOne** annotation is used to represent a one-to-one relationship.
* **@JoinColumn** is used to specify the foreign key column.

**Example:**

import javax.persistence.\*;

@Entity public class Person { @Id

@GeneratedValue(strategy = GenerationType.IDENTITY) private Long id; private String name;

@OneToOne

@JoinColumn(name = "passport\_id") // Foreign key column in the 'person' table private Passport passport;

// Getters and Setters

}

@Entity

public class Passport {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY) private Long id; private String passportNumber;

// Getters and Setters

}

**Explanation:**

* The **Person** entity has a @OneToOne relationship with the **Passport** entity.

* **@JoinColumn** indicates that the foreign key (passport\_id) is present in the **Person** table.

* In the database, **person** will have a column passport\_id that references the **passport** table.

**2. One-to-Many Relationship (1:M)**

A **one-to-many relationship** means that one entity is associated with multiple other entities. For example, one **Department** can have many **Employees**.

**JPA Representation:**

* **@OneToMany** is used in the "one" side of the relationship.

* **@ManyToOne** is used in the "many" side of the relationship.

* **@JoinColumn** is used on the "many" side to specify the foreign key column.

**Example:**

import javax.persistence.\*; import java.util.List;

@Entity public class Department {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String departmentName;

@OneToMany(mappedBy = "department") // 'department' is the field in Employee class

private List<Employee> employees;

// Getters and Setters

}

@Entity

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY) private Long id; private String name;

@ManyToOne

@JoinColumn(name = "department\_id") // Foreign key in Employee table private Department department;

// Getters and Setters

}

**Explanation:**

* The **Department** entity has a @OneToMany relationship with the **Employee** entity.

* The **Employee** entity has a @ManyToOne relationship to **Department**.

* The foreign key department\_id is stored in the **Employee** table.

**3. Many-to-One Relationship (M:1)**

A **many-to-one relationship** means that multiple entities are associated with a single entity. For example, many **Employees** belong to one **Department**.

**JPA Representation:**

* **@ManyToOne** is used to map the relationship from the "many" side.

* **@OneToMany** is used from the "one" side (reverse side).

**Example:**

This relationship is essentially the reverse of the **One-to-Many** example:

@Entity

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY) private Long id;

private String name;

@ManyToOne

@JoinColumn(name = "department\_id") // Foreign key in Employee table private Department department;

// Getters and Setters

}

* **Employee** is mapped to **Department** using @ManyToOne.

* **Department** is mapped to **Employee** using @OneToMany, and the foreign key (department\_id) is stored in **Employee**.

**4. Many-to-Many Relationship (M:M)**

A **many-to-many relationship** means that many entities are associated with many other entities. For example, a **Student** can enroll in many **Courses**, and each **Course** can have many **Students**.

**JPA Representation:**

* **@ManyToMany** annotation is used on both sides of the relationship.

* **@JoinTable** is used to specify the intermediary table that stores the relationships (because many-to-many relationships require an association table).

**Example:**

import javax.persistence.\*;

import java.util.List;

@Entity public class Student {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY) private Long id; private String name;

@ManyToMany

@JoinTable( name = "student\_course", // Join table name joinColumns = @JoinColumn(name = "student\_id"), // Foreign key in join table

inverseJoinColumns = @JoinColumn(name = "course\_id") // Foreign key for Course

)

private List<Course> courses;

// Getters and Setters

}

@Entity

public class Course {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY) private Long id;

private String courseName;

@ManyToMany(mappedBy = "courses") // 'courses' is the field in the Student class

private List<Student> students;

// Getters and Setters

}

**Explanation:**

* The **Student** entity has a @ManyToMany relationship with the **Course** entity.

* The **@JoinTable** annotation specifies the join table student\_course, which will have two foreign keys: student\_id and course\_id.

* The **Course** entity has the reverse @ManyToMany annotation, with mappedBy specifying that the relationship is already mapped by the **Student** entity.

**4.3 Create a JPA application to perform CRUD operations on a simple entity (e.g., Product).**

* **Include methods for creating, retrieving, updating, and deleting Product entities.**
* **Demonstrate the use of EntityManager for persistence operations.** Ans:

**Product.java**

import javax.persistence.Entity; import javax.persistence.GeneratedValue; import javax.persistence.GenerationType; import javax.persistence.Id;

*@Entity*

public class Product {

*@Id*

*@GeneratedValue*(strategy = *GenerationType*.*IDENTITY*) private Long id; private String name; private double price; // Constructors public Product() {

public Product(String name, double price) { this.name = name; this.price = price;

}

// Getters and Setters public Long getId() { return id;

public void setId(Long id) { this.id = id;

}

public String getName() { return name;

}

public void setName(String name) { this.name = name;

}

public double getPrice() { return price;

}

public void setPrice(double price) { this.price = price;

}

*@Override*

public String toString() { return "Product{id=" + id + ", name='" + name + "', price=" + price + "}";

}

}

### ProductService.java

import javax.persistence.EntityManager; import javax.persistence.EntityManagerFactory; import javax.persistence.Persistence;

public class ProductService { private static EntityManagerFactory *emf* =

Persistence.*createEntityManagerFactory*("productPU");

private static EntityManager *em* = *emf*.createEntityManager();

// Create Product

public void createProduct(Product product) { *em*.getTransaction().begin(); *em*.persist(product); *em*.getTransaction().commit();

System.***out***.println("Product Created: " + product); }

// Retrieve Product by ID

public Product getProduct(Long id) {

Product product = *em*.find(Product.class, id); System.***out***.println("Product Retrieved: " + product); return product;

}

// Update Product

public void updateProduct(Long id, String newName, double newPrice) { *em*.getTransaction().begin();

Product product = *em*.find(Product.class, id);

if (product != null) {

product.setName(newName); product.setPrice(newPrice); *em*.getTransaction().commit();

System.***out***.println("Product Updated: " + product); }

}

// Delete Product

public void deleteProduct(Long id) { *em*.getTransaction().begin();

Product product = *em*.find(Product.class, id);

if (product != null) {

*em*.remove(product); *em*.getTransaction().commit();

System.***out***.println("Product Deleted: " + product); }

}

// Close EntityManager public void close() { *em*.close(); *emf*.close();

}

}

Main.java public class Main {

public static void main(String[] args) {

ProductService productService = new ProductService();

// Create products

Product product1 = new Product("Laptop", 1200.0); Product product2 = new Product("Smartphone", 800.0); productService.createProduct(product1); productService.createProduct(product2);

// Retrieve product by ID

Product retrievedProduct = productService.getProduct(1L);

// Update product

productService.updateProduct(1L, "Gaming Laptop", 1500.0);

// Delete product

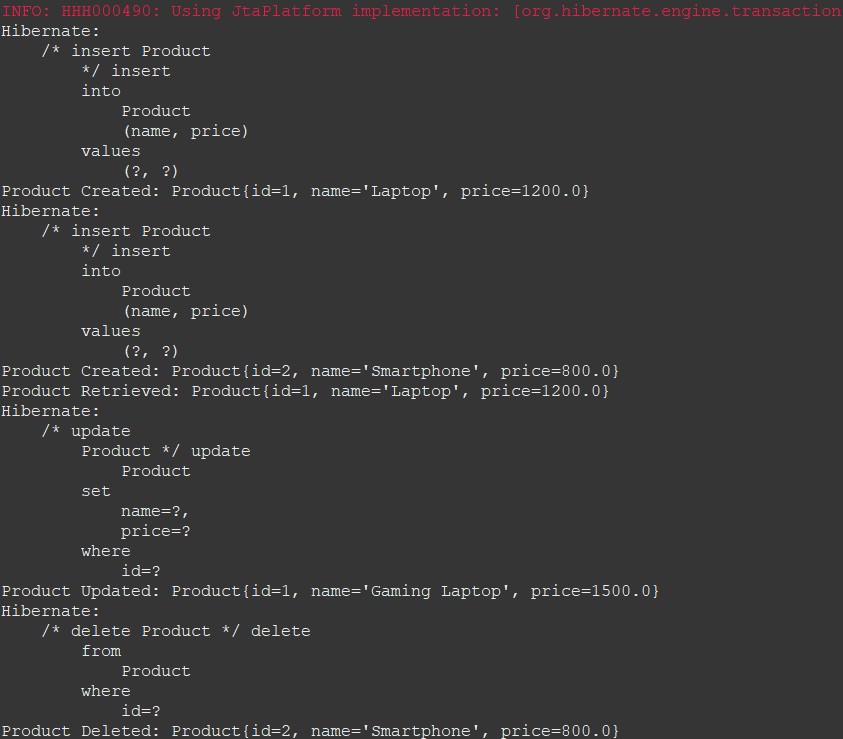
productService.deleteProduct(2L);

// Close resources productService.close();

}

}

**OUTPUT:**



### Assignment 5:Spring Boot

**1. Configure a Spring Boot application to connect to a specific MySQL database without explicitly defining beans for connection pool, DataSource, etc.**

○ **Use only the necessary dependencies and demonstrate how auto- configuration sets up the connection.**

○ **Explore using application.properties to customize connection details (URL, username, password).**

Main Application Class

**File Name**: SpringbootFirstApplication.java **Location**: src/main/java/com/java/springboot package com.java.springboot;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class SpringbootFirstApplication { public static void main(String[] args) {

SpringApplication.*run*(SpringbootFirstApplication.class, args);

}

}

**2. Entity Class**

**File Name**: User.java

**Location**: src/main/java/com/java/springboot/model package com.java.springboot.Model;

import jakarta.persistence.Entity; import jakarta.persistence.Id;

@Entity

public class User {

@Id

private Long id; private String name; private String email;

// Getters and Setters public Long getId() { return id;

}

public void setId(Long id) { this.id = id;

}

public String getName() { return name;

}

public void setName(String name) { this.name = name;

}

public String getEmail() { return email;

}

public void setEmail(String email) { this.email = email;

}

}

1. **Repository Interface**

**File Name**: UserRepository.java

**Location**: src/main/java/com/java/springboot/repository package com.java.springboot.repository;

import org.springframework.data.jpa.repository.JpaRepository; import com.java.springboot.Model.User;

public interface UserRepository extends JpaRepository<User, Long> { }

1. **Controller**

**File Name**: UserController.java

**Location**: src/main/java/com/java/springboot/controller package com.java.springboot.controller;

import com.java.springboot.Model.User; import com.java.springboot.repository.UserRepository; import org.springframework.web.bind.annotation.\*; import java.util.List;

@RestController

@RequestMapping("/api/users") public class UserController {

private final UserRepository userRepository;

public UserController(UserRepository userRepository) { this.userRepository = userRepository; }

@GetMapping

public List<User> getAllUsers() { return userRepository.findAll();

}

@PostMapping

public User createUser(@RequestBody User user) { return userRepository.save(user); }

}

**5. Application Properties**

**File Name**: application.properties

**Location**: src/main/resourc

# MySQL database connection spring.datasource.url=jdbc:mysql://localhost:3306/company\_db spring.datasource.username=root spring.datasource.password=1234567890

# JPA and Hibernate settings spring.jpa.hibernate.ddl-auto=update spring.jpa.show-sql=true

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDia lect

6) pom.xml

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

<project xmlns="[http://maven.apache.org/POM/4.0.0"](http://maven.apache.org/POM/4.0.0) xmlns:xsi="[http://www.w3.org/2001/XMLSchema-instance"](http://www.w3.org/2001/XMLSchema-instance) xsi:schemaLocation="<http://maven.apache.org/POM/4.0.0>

https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>3.4.0</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>com.java</groupId>

<artifactId>springboot-first</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>springboot-first</name>

<description>Demo project for Spring Boot</description>

<url/>

<licenses>

<license/>

</licenses>

<developers>

<developer/>

</developers>

<scm>

<connection/>

<developerConnection/>

<tag/>

<url/>

</scm>

<properties>

<java.version>17</java.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>com.mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

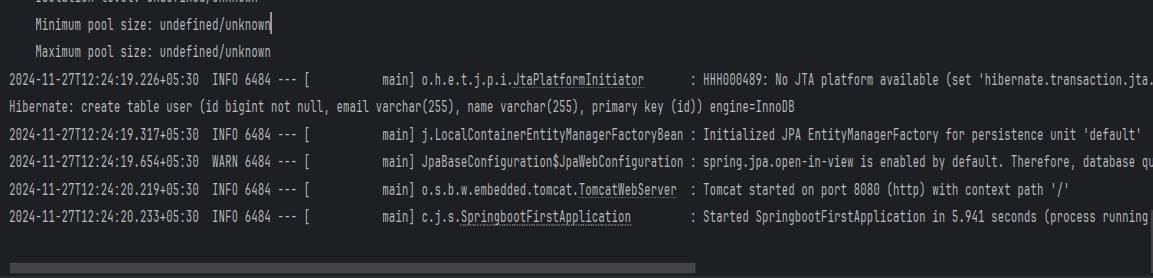
<artifactId>spring-boot-maven-plugin</artifactId> </plugin>

</plugins>

</build>

</project>

**OUTPUT**:



**2. Create a Spring Boot application that utilizes JPA repositories. Persist and retrieve data from an in-memory database (e.g., H2) without manual configuration.**

○ **Focus on the simplicity achieved through auto-configuration for JPA and repositories.**

○ **Implement basic CRUD operations using JPA repositorie**

ProductController.java package com.example.project2.controller;

import com.example.project2.model.Product; import com.example.project2.service.ProductService; import org.springframework.http.ResponseEntity; import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/products") public class ProductController { private final ProductService productService;

public ProductController(ProductService productService) { this.productService = productService; }

@GetMapping

public List<Product> getAllProducts() { return productService.getAllProducts(); }

@GetMapping("/{id}")

public ResponseEntity<Product> getProductById(@PathVariable Long id) { return productService.getProductById(id)

.map(ResponseEntity::*ok*)

.orElse(ResponseEntity.*notFound*().build());

}

@PostMapping

public Product addProduct(@RequestBody Product product) { return productService.addProduct(product); }

@PutMapping("/{id}")

public ResponseEntity<Product> updateProduct(@PathVariable Long id,

@RequestBody Product product) {

try { return ResponseEntity.*ok*(productService.updateProduct(id, product));

} catch (RuntimeException e) { return ResponseEntity.*notFound*().build(); }

}

@DeleteMapping("/{id}")

public ResponseEntity<Void> deleteProduct(@PathVariable Long id) { productService.deleteProduct(id); return ResponseEntity.*noContent*().build(); }

}

Product.java package com.example.project2.model;

import jakarta.persistence.Entity; import jakarta.persistence.GeneratedValue; import jakarta.persistence.GenerationType; import jakarta.persistence.Id;

@Entity public class Product {

@Id

@GeneratedValue(strategy = GenerationType.*IDENTITY*) private Long id;

private String name; private double price;

public Product() {}

public Product(String name, double price) {

this.name = name; this.price = price;

}

public Long getId() {

return id;

}

public void setId(Long id) { this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public double getPrice() {

return price;

}

public void setPrice(double price) {

this.price = price;

}

}

ProductRespository.java

package com.example.project2.repository;

import com.example.project2.model.Product; import org.springframework.data.jpa.repository.JpaRepository;

public interface ProductRepository extends JpaRepository<Product, Long> { }

ProductService.java package com.example.project2.service;

import com.example.project2.model.Product; import com.example.project2.repository.ProductRepository; import org.springframework.stereotype.Service;

import java.util.List; import java.util.Optional;

@Service public class ProductService {

private final ProductRepository productRepository;

public ProductService(ProductRepository productRepository) { this.productRepository = productRepository; }

public List<Product> getAllProducts() { return productRepository.findAll(); }

public Optional<Product> getProductById(Long id) { return productRepository.findById(id); }

public Product addProduct(Product product) { return productRepository.save(product);

}

public Product updateProduct(Long id, Product updatedProduct) { return productRepository.findById(id).map(product -> { product.setName(updatedProduct.getName()); product.setPrice(updatedProduct.getPrice()); return productRepository.save(product);

}).orElseThrow(() -> new RuntimeException("Product not found"));

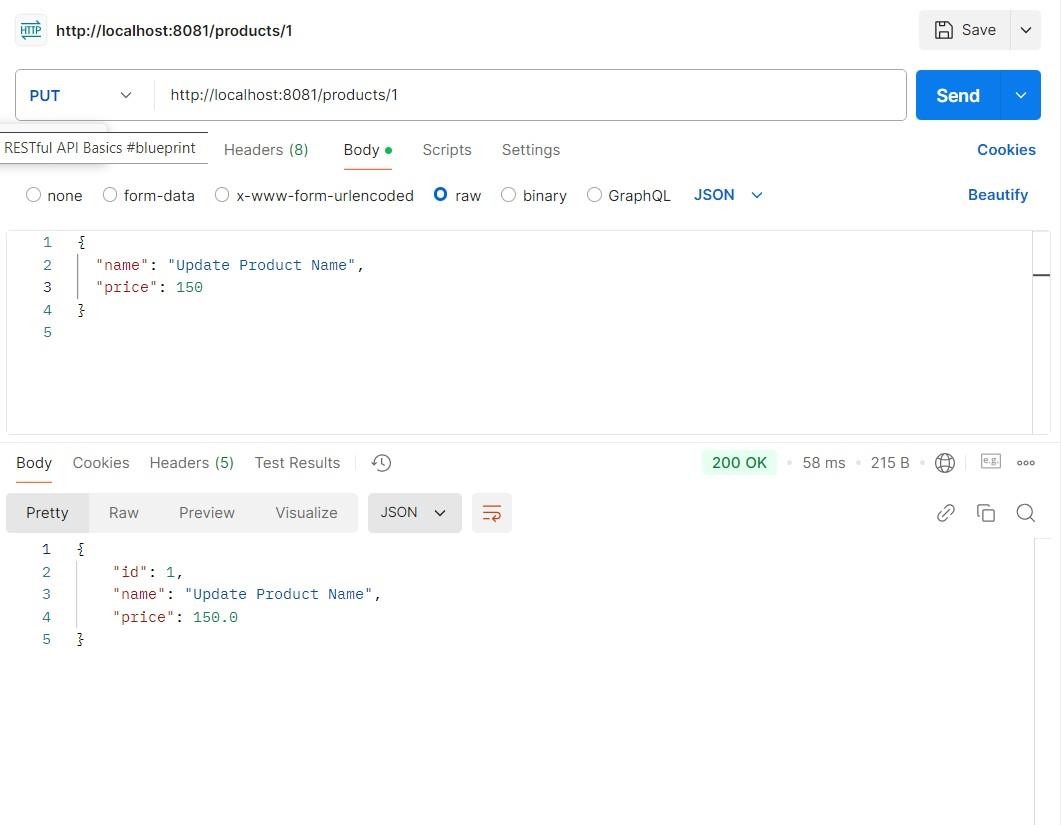
}

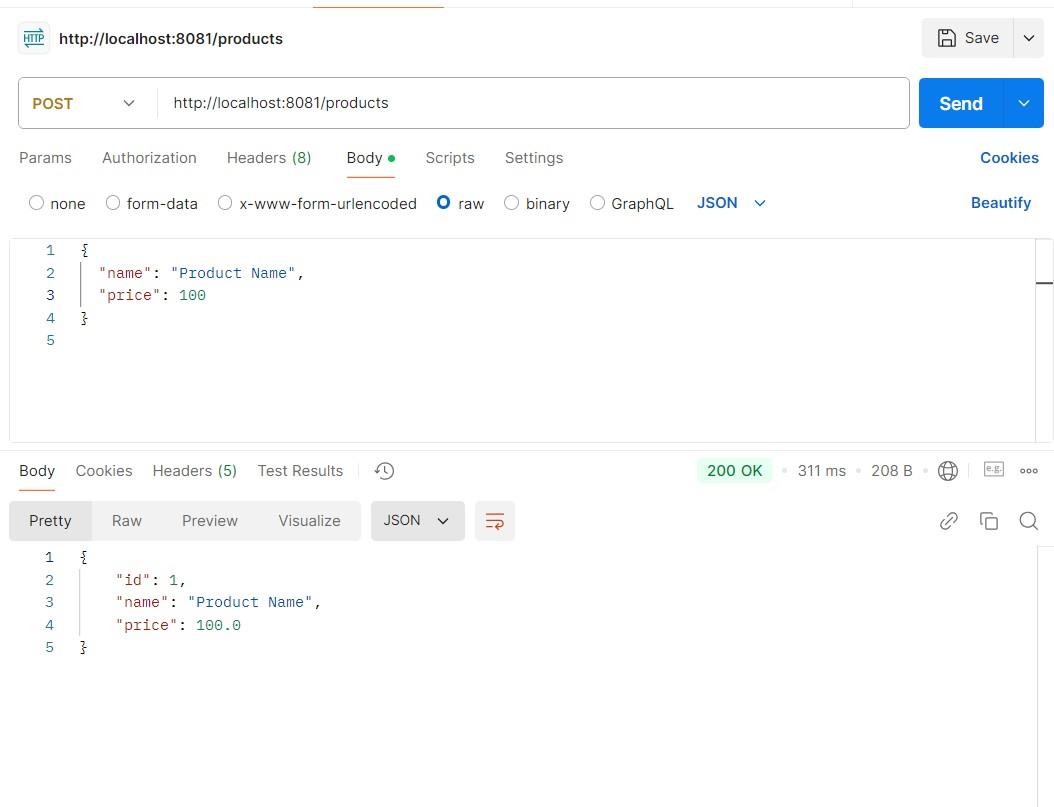
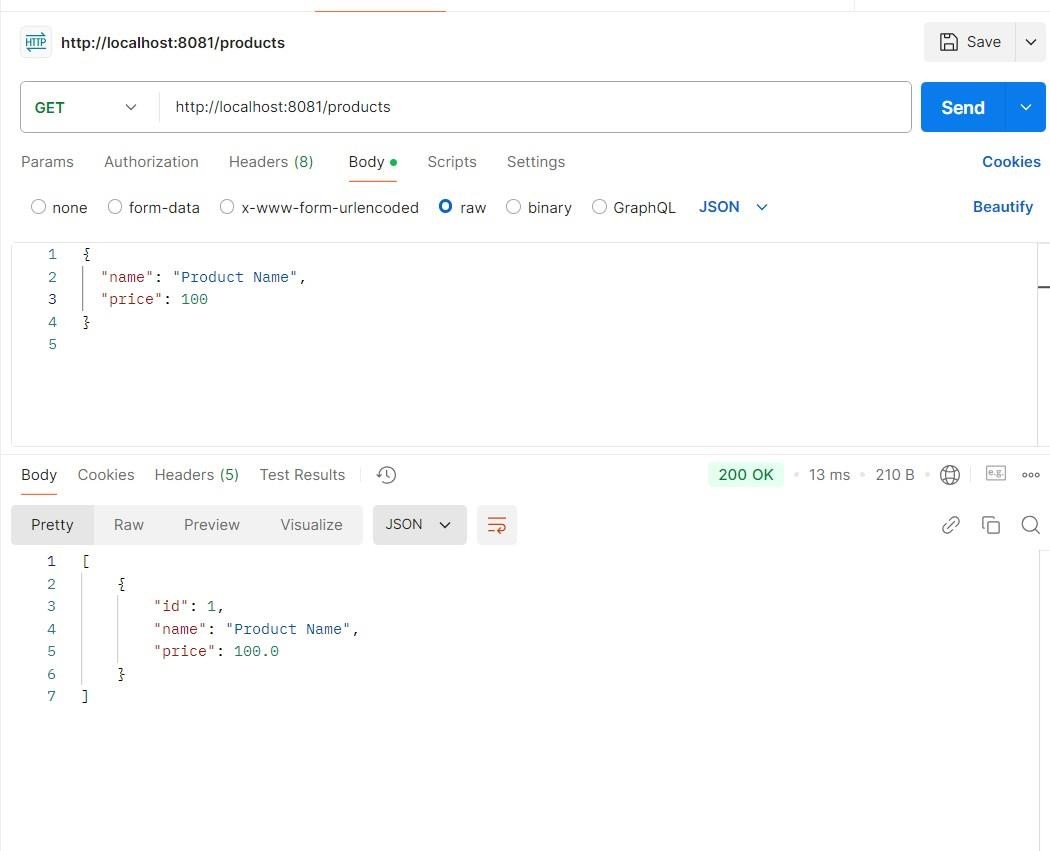
public void deleteProduct(Long id) { productRepository.deleteById(id);

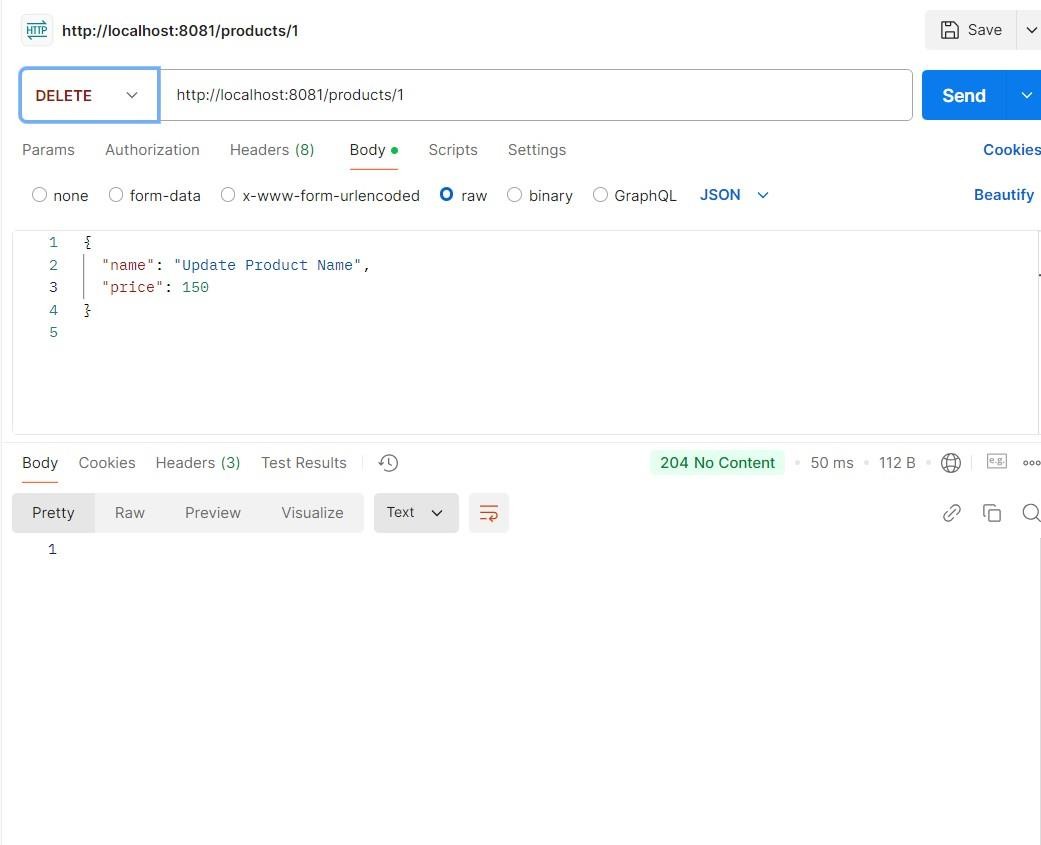
}

}

application.properties # H2 Database settings spring.datasource.url=jdbc:h2:mem:DEMO spring.datasource.driverClassName=org.h2.Driver spring.datasource.username=root spring.datasource.password=12345 spring.h2.console.enabled=true spring.jpa.show-sql=true spring.jpa.hibernate.ddl-auto=update server.port=8081







**3. Develop a Spring Boot application with a RESTful API that exposes an endpoint to retrieve a list of products.**

○ **Utilize Spring MVC annotations like @RestController and @GetMapping.**

○ **Implement a service layer to interact with a product repository (in-memory or database).**

○ **Return the list of products in JSON format using @ResponseBody**

ProductController.java package com.example.productapi.controller; import com.example.productapi.model.Product; import com.example.productapi.service.ProductService; import org.springframework.web.bind.annotation.GetMapping; import org.springframework.web.bind.annotation.RestController; import java.util.List; @RestController

public class ProductController { private final ProductService productService;

public ProductController(ProductService productService) { this.productService = productService; }

@GetMapping("/products") public List<Product> getProducts() { return productService.getProducts(); }

}

Product.java package com.example.productapi.model;

public class Product { private Long id; private String name; private double price; // Constructors

public Product(Long id, String name, double price) {

this.id = id; this.name = name; this.price = price;

}

// Getters and Setters public Long getId() {

return id;

}

public void setId(Long id) {

this.id = id; }

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public double getPrice() {

return price;

}

public void setPrice(double price) {

this.price = price;

}

}

ProductRepository.java package com.example.productapi.repository; import com.example.productapi.model.Product; import org.springframework.stereotype.Repository; import java.util.Arrays;

import java.util.List;

@Repository public class ProductRepository {

public List<Product> getAllProducts() {

return Arrays.*asList*( new Product(1L, "Laptop", 999.99),

new Product(2L, "Smartphone", 599.99),

new Product(3L, "Headphones", 199.99)

);

}

}

ProductService.java package com.example.productapi.service; import com.example.productapi.model.Product; import com.example.productapi.repository.ProductRepository; import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class ProductService { private final ProductRepository productRepository;

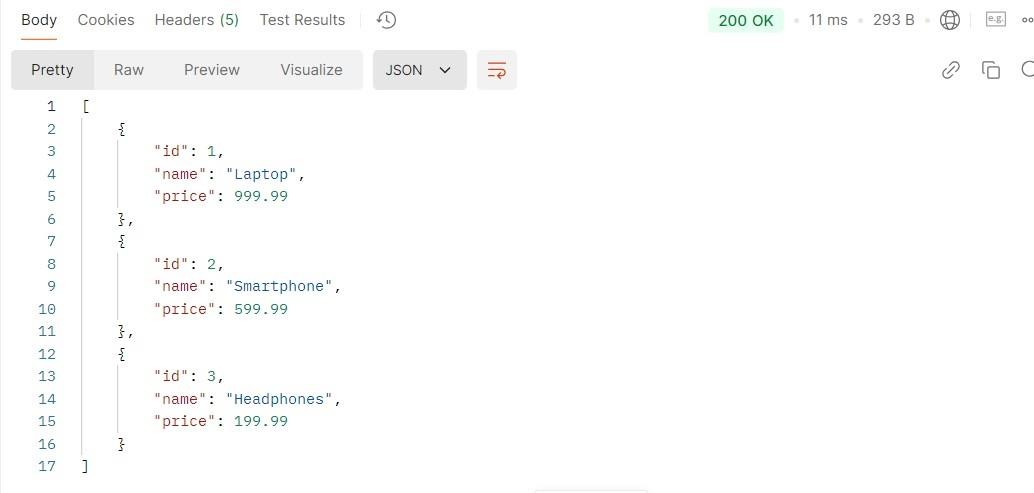
public ProductService(ProductRepository productRepository) { this.productRepository = productRepository; }

public List<Product> getProducts() { return productRepository.getAllProducts(); }

}

application.properties spring.h2. console.enabled=true spring.h2.console.path=/h2-console spring.datasource.url=jdbc:h2:mem:DEMO spring.datasource.driverClassName=org.h2.Driver spring.datasource.username=root spring.datasource.password=12345

**OUTPUT :**



**Assignment 6: Hibernate Framework**

**6.1 Write a Hibernate program to create the product table (product id,product name,product category,product price) and delete the specific product record.(using through the product id)**

#### Product.java

import javax.persistence.Entity; import javax.persistence.Id;

*@Entity*

public class Product {

*@Id*

private int id; private String name; private String category; private double price;

// Default constructor (required by JPA) public Product() {

}

// Constructor with parameters

public Product(int id, String name, String category, double price) {

this.id = id;

this.name = name; this.category = category;

this.price = price;

}

public int getId() { return id;

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

}

public String getName() { return name;

}

public void setName(String name) { this.name = name;

}

public String getCategory() { return category;

}

public void setCategory(String category) { this.category = category;

}

public double getPrice() { return price;

}

public void setPrice(double price) { this.price = price;

}

}

#### ProductService.java

import org.hibernate.Session;

import org.hibernate.SessionFactory; import org.hibernate.Transaction;

public class ProductService {

public void createProduct(Product product) {

// Get session factory

SessionFactory factory = HibernateUtil.*getSessionFactory*();

// Get session from the factory

Session session = factory.getCurrentSession();

// Begin transaction

Transaction transaction = session.beginTransaction();

try {

// Save the product session.save(product); // Commit transaction

transaction.commit();

} catch (Exception e) {

// Handle exception, roll back transaction

if (transaction != null) {

transaction.rollback();

}

e.printStackTrace();

} finally {

// Close the session (do not call closeSession here, just use session.close()) session.close();

}

}

}

#### Main.java

public class Main {

public static void main(String[] args) {

try {

// Create a new product

Product newProduct = new Product(2, "Laptop", "Electronics", 1200.00);

// Create ProductService instance

ProductService productService = new ProductService();

// Call method to create product productService.createProduct(newProduct);

} finally {

// Clean up resources by closing the SessionFactory

HibernateUtil.*closeSessionFactory*();

}

}

}

**InsertProduct.java**

import org.hibernate.Session;

import org.hibernate.SessionFactory; import org.hibernate.Transaction; public class InsertProduct {

public static void main(String[] args) {

// Create a new Product object

Product newProduct = new Product(2, "Laptop", "Electronics", 1200.00); // id changed to 2

// Get session factory

SessionFactory factory = HibernateUtil.*getSessionFactory*();

// Get session from factory

Session session = factory.getCurrentSession();

// Begin transaction

Transaction transaction = session.beginTransaction();

try {

// Save the Product object session.save(newProduct);

// Commit the transaction (this will persist the product in the database)

transaction.commit();

} catch (Exception e) {

// Handle exception (in case of any errors, roll back the transaction)

if (transaction != null) {

transaction.rollback();

}

e.printStackTrace();

} finally {

// Close the session session.close();

}

}

}

#### HibernateUtil.java

import org.hibernate.SessionFactory; import org.hibernate.cfg.Configuration;

public class HibernateUtil { private static SessionFactory *sessionFactory*;

static {

try {

// Initialize SessionFactory from Hibernate configuration file

*sessionFactory* = new

Configuration().configure("hibernate.cfg.xml").addAnnotatedClass(Product.class).bu ildSessionFactory();

} catch (Exception e) {

e.printStackTrace();

throw new ExceptionInInitializerError("SessionFactory initialization

failed.");

}

}

// Method to get SessionFactory

public static SessionFactory getSessionFactory() {

return *sessionFactory*;

}

// Method to close the SessionFactory public static void closeSessionFactory() { if (*sessionFactory* != null) {

*sessionFactory*.close();

}

}

}

#### Hibernate.cfg.xml

<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN" "<http://hibernate.sourceforge.net/hibernate->configuration-3.0.dtd">

##### <hibernate-configuration> <session-factory>

<!-- JDBC Database connection settings -->

<property name="hibernate.connection.driver\_class">com.mysql.cj.jdbc.Driver</property>

<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/products</property>

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

<property name="hibernate.connection.password">1234567890</property>

<!-- JDBC connection pool settings -->

<property name="hibernate.c3p0.min\_size">5</property>

<property name="hibernate.c3p0.max\_size">20</property>

<!-- Specify dialect -->

<property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>

<!-- Enable Hibernate's automatic session context management -->

<property name="hibernate.current\_session\_context\_class">thread</property>

<!-- Echo all executed queries -->

<property name="hibernate.show\_sql">true</property>

<!-- Drop and re-create the database schema on startup -->

<property name="hibernate.hbm2ddl.auto">update</property>

<!-- Disable the second-level cache -->

<property name="hibernate.cache.provider\_class">org.hibernate.cache.NoCacheProvider</pro perty>

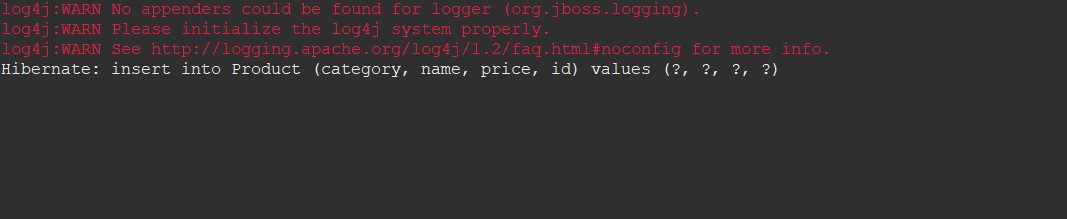
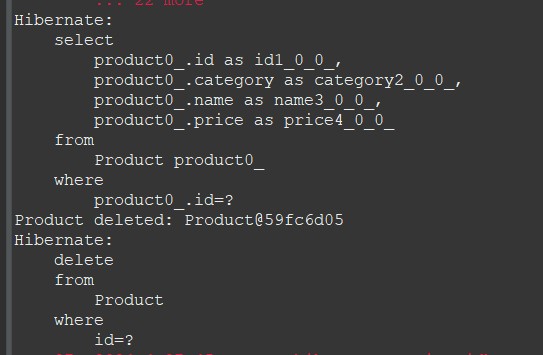
<!-- Drop and re-create the database schema on startup -->

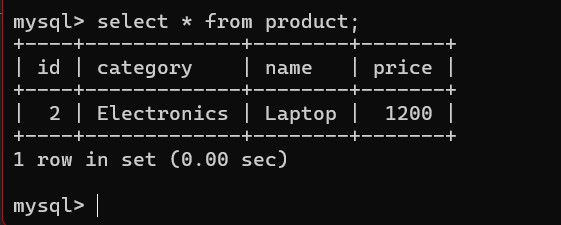
<property name="hibernate.hbm2ddl.auto">update</property>

</session-factory>

</hibernate-configuration>

**OUTPUT**





**6.2 Write a Hibernate program to update the product price data from product table.(Using HQL)**

#### Product.java

import javax.persistence.Entity; import javax.persistence.Id; import javax.persistence.Table;

*@Entity*

*@Table*(name = "product") // This maps the entity to the "product" table

public class Product {

*@Id* // Marks the "id" field as the primary key

private int id; private String name; private String category; private double price; // Default constructor public Product() {}

// Constructor with all fields

public Product(int id, String name, String category, double price) {

this.id = id;

this.name = name;

this.category = category; this.price = price;

}

// Getters and Setters public int getId() {

return id;

}

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

}

public String getName() { return name;

}

public void setName(String name) { this.name = name;

}

public String getCategory() { return category;

}

public void setCategory(String category) { this.category = category;

}

public double getPrice() { return price;

}

public void setPrice(double price) {

this.price = price;

}

*@Override*

public String toString() {

return "Product [id=" + id + ", name=" + name + ", category=" + category + ",

price=" + price + "]";

}

}

#### ProductService.java

import org.hibernate.Session; import org.hibernate.Transaction;

public class ProductService {

public void updateProductPrice(int productId, double newPrice) {

// Start session

Session session = HibernateUtil.getSessionFactory().openSession();

// Begin transaction

Transaction transaction = null;

try {

transaction = session.beginTransaction();

// HQL Query to update product price

String hql = "UPDATE Product p SET p.price = :price WHERE p.id = :productId";

// Create query and set parameters

int updatedEntities = session.createQuery(hql)

.setParameter("price", newPrice)

.setParameter("productId", productId)

.executeUpdate();

// Commit the transaction transaction.commit(); // Output success message if (updatedEntities > 0) {

System.***out***.println("Product price updated successfully!");

} else {

System.***out***.println("Product not found with id: " + productId);

}

} catch (Exception e) {

if (transaction != null) {

transaction.rollback(); // Rollback transaction on error

}

e.printStackTrace();

} finally {

session.close(); // Close session

}

}

}

#### HibernateUtil.java

import org.hibernate.SessionFactory; import org.hibernate.cfg.Configuration; public class HibernateUtil { private static SessionFactory *sessionFactory*;

// Static block to initialize sessionFactory

static {

try {

// Build the session factory using the configuration

*sessionFactory* = new Configuration().configure("hibernate.cfg.xml")

.addAnnotatedClass(Product.class) // Add the annotated entity class (Product)

.buildSessionFactory();

} catch (Exception e) {

e.printStackTrace();

throw new ExceptionInInitializerError(e);

}

}

// Method to get the sessionFactory

public static SessionFactory getSessionFactory() {

return *sessionFactory*;

}

// Method to close the sessionFactory

public static void closeSessionFactory() {

if (*sessionFactory* != null) {

*sessionFactory*.close();

}

}

}

#### Main.java

public class Main {

public static void main(String[] args) {

ProductService productService = new ProductService();

// Update product price where productId is 1 and new price is 899.99

productService.updateProductPrice(1, 899.99);

}

}

#### Hibernate.cfg.xml

<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN" "<http://hibernate.sourceforge.net/hibernate->configuration-3.0.dtd">

<hibernate-configuration>

<!-- JDBC Database connection settings -->

##### <session-factory>

<!-- JDBC driver -->

<property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>

<property name="hibernate.connection.driver\_class">com.mysql.cj.jdbc.Driver</property>

<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/products</property>

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

<property name="hibernate.connection.password">1234567890</property>

<!-- JDBC connection pool settings -->

<property name="hibernate.c3p0.min\_size">5</property>

<property name="hibernate.c3p0.max\_size">20</property>

<property name="hibernate.c3p0.timeout">300</property>

<property name="hibernate.c3p0.max\_statements">50</property>

<!-- Specify the JDBC transaction handling -->

<property name="hibernate.transaction.factory\_class">org.hibernate.transaction.JDBCTransact ionFactory</property>

<!-- Echo all executed SQL to stdout -->

<property name="hibernate.show\_sql">true</property>

<!-- Drop and re-create the database schema on startup -->

<property name="hibernate.hbm2ddl.auto">update</property>

<!-- Enable Hibernate's automatic session context management -->

<property name="hibernate.current\_session\_context\_class">thread</property>

<!-- Disable the second-level cache -->

<property name="hibernate.cache.provider\_class">org.hibernate.cache.NoCacheProvider</pro perty>

<!-- Echo all executed SQL to stdout -->

<property name="hibernate.format\_sql">true</property>

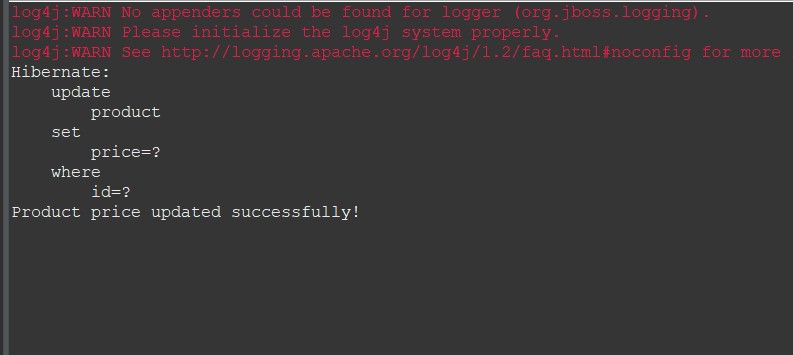
<!-- Specify annotated class for the entity -->

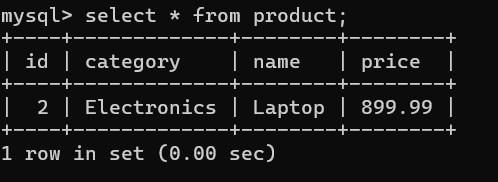
<mapping class="Product"/>

</session-factory>

</hibernate-configuration>

**OUTPUT:**





**6.3 Write a Hibernate Program for product information and display the information by selecting the details from product database table**

#### Product.java

import javax.persistence.Entity; import javax.persistence.Id; import javax.persistence.Table;

*@Entity*

*@Table*(name = "product") // Map to the 'product' table in the database public class Product {

*@Id*

private int id; private String name; private String category; private double price;

// Constructor, Getters, and Setters public Product() {}

public Product(int id, String name, String category, double price) { this.id = id; this.name = name; this.category = category; this.price = price;

}

public int getId() { return id;

}

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

}

public String getName() { return name;

}

public void setName(String name) { this.name = name;

}

public String getCategory() { return category;

}

public void setCategory(String category) { this.category = category;

}

public double getPrice() { return price;

}

public void setPrice(double price) { this.price = price;

}

*@Override*

public String toString() { return "Product [id=" + id + ", name=" + name + ", category=" + category + ", price=" + price + "]";

}

}

#### ProductService.java

import org.hibernate.Session; import org.hibernate.Transaction; import java.util.List; // Add this import statement public class ProductService {

public void displayProductInfo() {

// Get the session from the session factory

Session session = HibernateUtil.*getSessionFactory*().getCurrentSession();

// Begin a transaction

Transaction transaction = session.beginTransaction();

try {

// Retrieve product data using HQL (Hibernate Query Language)

String hql = "FROM Product"; // Get all product records

List<Product> products = session.createQuery(hql,

Product.class).getResultList(); // List is now recognized

// Display each product for (Product product : products) {

System.*out*.println(product);

}

// Commit the transaction transaction.commit();

} catch (Exception e) {

e.printStackTrace(); if (transaction != null) { transaction.rollback();

}

} finally {

HibernateUtil.*closeSessionFactory*(); }

}

}

#### Main.java

public class Main {

public static void main(String[] args) {

// Create an instance of ProductService

ProductService productService = new ProductService();

// Display product information from the database productService.displayProductInfo(); }

}

#### HibernateUtil.java

import org.hibernate.SessionFactory; import org.hibernate.cfg.Configuration;

public class HibernateUtil { private static SessionFactory *sessionFactory*;

static { try { *sessionFactory* = new Configuration().configure("hibernate.cfg.xml")

.addAnnotatedClass(Product.class) // Add Product class for mapping .buildSessionFactory();

} catch (Exception e) {

e.printStackTrace();

throw new ExceptionInInitializerError(e); }

}

public static SessionFactory getSessionFactory() { return *sessionFactory*;

}

public static void closeSessionFactory() { if (*sessionFactory* != null) { *sessionFactory*.close();

}

}

}

#### Hibernate.cfg.xml

<!DOCTYPE hibernate-configuration PUBLIC "-//Hibernate/Hibernate Configuration DTD 3.0//EN"

["http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd"](http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd)>

##### <hibernate-configuration> <session-factory>

<!-- JDBC Database connection settings -->

<property name="hibernate.connection.driver\_class">org.h2.Driver</property>

<property name="hibernate.connection.url">jdbc:h2:~/test;DB\_CLOSE\_ON\_EXI T=FALSE</property>

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

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

<!-- JDBC connection pool settings -->

<property name="hibernate.c3p0.min\_size">5</property>

<property name="hibernate.c3p0.max\_size">20</property>

<property name="hibernate.c3p0.timeout">300</property>

<property name="hibernate.c3p0.max\_statements">50</property>

<property name="hibernate.c3p0.idle\_test\_period">3000</property>

<!-- Specify dialect -->

<property name="hibernate.dialect">org.hibernate.dialect.H2Dialect</property>

<!-- Echo all executed queries -->

<property name="hibernate.show\_sql">true</property>

<!-- Drop and re-create the database schema on startup -->

<property name="hibernate.hbm2ddl.auto">update</property>

<!-- Enable Hibernate's automatic session context management -->

<property name="hibernate.current\_session\_context\_class">thread</property>

<!-- Disable the second-level cache -->

<property name="hibernate.cache.provider\_class">org.hibernate.cache.NoCacheP rovider</property>

</session-factory>

</hibernate-configuration>

**Output:-**

