**PRACTICAL NO. 05**

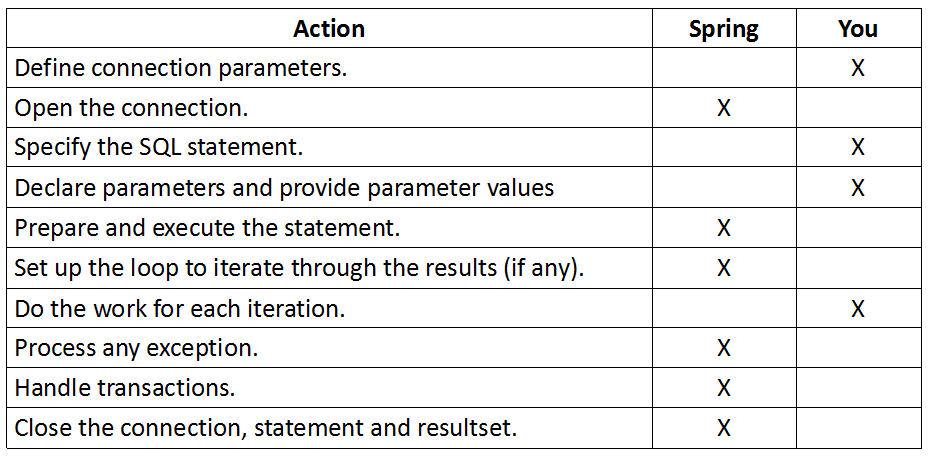
**JDBC Data Access with Spring using Oracle/MySQL database**

|  |  |
| --- | --- |
| LOB5 | Demonstrate JDBC Data Demonstrate Access with Spring using Oracle/MySQL database. |
| LO5 | Develop JDBC application with Spring using JdbcTemplate and Oracle/MySQL database. |

Spring framework has Data Access / Integration module which comprises of JDBC, ORM, OXM, JMS and Transaction modules. These modules basically provide support to interact with the database. Spring JDBC modules simplify the use of underlying JDBC API.

org.springframework.jdbc.core.JdbcOperations defines Spring way of handling database access using JDBC. This interface provides abstraction from the low level repetitive JDBC operations like opening the database connection, preparing the Jdbc statements, handling and processing exception, handling transactions, closing the connection etc.

The following table shows what actions Spring will take care of and which actions are the responsibility of the application developer.



Spring provides data access operations performed with JDBC using three main approaches:

1. Using Template Method pattern‐based utility classes, namely JdbcTemplate and NamedParameterJdbcTemplate, to perform JDBC operations more easily by removing repetitive data access code blocks in the application, properly handling resource cleanups, and so on.
2. Using database metadata to simplify queries using classes such as SimpleJdbcInsert and SimpleJdbcCall. That way you need to provide only a table or stored procedure name and a map of parameters corresponding to column names to perform an SQL operation.
3. Using MappingSqlQuery, SqlUpdate, and StoredProcedure classes to represent database operations as reusable Java objects so that you can use them over and over again by providing only different query parameters each time.

**Approaches for JDBC Database Access**

**JdbcTemplate –**

It is the classic Spring JDBC approach and the most popular. This "lowest level" approach and all others use a JdbcTemplate under the covers, and all are updated with Java 5 support such as generics and varargs.

**NamedParameterJdbcTemplate –**

It wraps a JdbcTemplate to provide named parameters instead of the traditional JDBC "?" placeholders. This approach provides better documentation and ease of use when you have multiple parameters for an SQL statement.

**SimpleJdbcTemplate –**

It combines the most frequently used operations of JdbcTemplate & NamedParameterJdbcTemplate.

**SimpleJdbcInsert and SimpleJdbcCall –**

It optimize database metadata to limit the amount of necessary configuration. This approach simplifies coding so that you only need to provide the name of the table or procedure and provide a map of parameters matching the column names.

This only works if the database provides adequate metadata. If the database doesn't provide this metadata, you will have to provide explicit configuration of the parameters.

**Exercise:**

|  |  |  |
| --- | --- | --- |
| Gr. No. | QN | Question |
| 01  (Roll No. 01 to 10) | Employee and Department Tables   * Employee Table: employee\_id, employee\_name, department\_id * Department Table: department\_id, department\_name | |
| 1 | Create a DAO (Data Access Object) classes that provides CRUD (Create, Read, Update, Delete) operations for the first table mentioned above. |
| 2 | Create a DAO (Data Access Object) classes that provides CRUD (Create, Read, Update, Delete) operations for the second table mentioned above. |
| 3 | Create a class to display all employees along with their respective department names using ResultSetExtractor Interface of Spring JDBC. |
| 4 | Create a method called "getEmployeeByName" in the DAO class that takes a name as input and returns the details of the employee with that name. The method should use PreparedStatement in Spring JdbcTemplate to execute a SQL query and retrieve the employee information using RowMapper. |
| 5 | Create a class for executing stored procedure from database that selects a particular record as per the user input. |

**Department.java**

package packP5Code1;

public class Department {

private int departmentId;

private String departmentName;

// Constructor

public Department(int departmentId, String departmentName) {

this.departmentId = departmentId;

this.departmentName = departmentName;

}

// Getters and Setters

public int getDepartmentId() {

return departmentId;

}

public void setDepartmentId(int departmentId) {

this.departmentId = departmentId;

}

public String getDepartmentName() {

return departmentName;

}

public void setDepartmentName(String departmentName) {

this.departmentName = departmentName;

}

@Override

public String toString() {

return "Department [departmentId=" + departmentId + ", departmentName=" + departmentName + "]";

}

}

**Employee.java**

package packP5Code1;

public class Employee {

private int employeeId;

private String employeeName;

private int departmentId;

// Constructor

public Employee(int employeeId, String employeeName, int departmentId) {

this.employeeId = employeeId;

this.employeeName = employeeName;

this.departmentId = departmentId;

}

// Default Constructor

public Employee() {}

// Getters and Setters

public int getEmployeeId() {

return employeeId;

}

public void setEmployeeId(int employeeId) {

this.employeeId = employeeId;

}

public String getEmployeeName() {

return employeeName;

}

public void setEmployeeName(String employeeName) {

this.employeeName = employeeName;

}

public int getDepartmentId() {

return departmentId;

}

public void setDepartmentId(int departmentId) {

this.departmentId = departmentId;

}

@Override

public String toString() {

return "Employee [employeeId=" + employeeId + ", employeeName=" + employeeName + ", departmentId=" + departmentId + "]\n";

}

}

**DepartmentDAO.java**

package packP5Code1;

import java.util.List;

import java.util.Map;

import org.springframework.jdbc.core.JdbcTemplate;

public class DepartmentDAO {

JdbcTemplate jdbcTemplate;

// Setter for JdbcTemplate

public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {

this.jdbcTemplate = jdbcTemplate;

}

// Add Department

public int addDepartment(Department dept) {

String sqlStr = "INSERT INTO Department (department\_id, department\_name) VALUES ("

+ dept.getDepartmentId() + ",'" + dept.getDepartmentName() + "')";

return jdbcTemplate.update(sqlStr);

}

// Get All Departments (Read)

public List<Map<String, Object>> getAllDepartments() {

String sqlStr = "SELECT \* FROM Department";

return jdbcTemplate.queryForList(sqlStr);

}

// Update Department (Update)

public int updateDepartment(int departmentId, String newDepartmentName) {

String sqlStr = "UPDATE Department SET department\_name = '" + newDepartmentName +

"' WHERE department\_id = " + departmentId;

return jdbcTemplate.update(sqlStr);

}

// Delete Department (Delete)

public int deleteDepartment(int departmentId) {

String sqlStr = "DELETE FROM Department WHERE department\_id = " + departmentId;

return jdbcTemplate.update(sqlStr);

}

}

**EmployeeDAO.java**

package packP5Code1;

import org.springframework.jdbc.core.JdbcTemplate;

import org.springframework.jdbc.core.RowMapper;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.List;

import java.util.Map;

public class EmployeeDAO {

JdbcTemplate jdbcTemplate;

// Setter for JdbcTemplate

public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {

this.jdbcTemplate = jdbcTemplate;

}

// Add Employee

public int addEmployee(Employee emp) {

String sqlStr = "INSERT INTO Employee VALUES ("

+ emp.getEmployeeId() + ",'" + emp.getEmployeeName() + "'," + emp.getDepartmentId() + ")";

return jdbcTemplate.update(sqlStr);

}

// List All Employees

public void listAllEmployees() {

String sqlStr = "SELECT \* FROM Employee";

System.out.println(jdbcTemplate.queryForList(sqlStr));

}

// Update Employee

public int updateEmployee(int employeeId, String employeeName, int departmentId) {

String sqlStr = "UPDATE Employee SET employee\_name='" + employeeName + "', department\_id=" + departmentId

+ " WHERE employee\_id=" + employeeId;

return jdbcTemplate.update(sqlStr);

}

// Delete Employee

public int deleteEmployee(int employeeId) {

String sqlStr = "DELETE FROM Employee WHERE employee\_id=" + employeeId;

return jdbcTemplate.update(sqlStr);

}

}

**EmployeeWithDepartmentDAO.java**

package packP5Code1;

import org.springframework.jdbc.core.JdbcTemplate;

import org.springframework.jdbc.core.ResultSetExtractor;

import org.springframework.jdbc.core.RowMapper;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.ArrayList;

import java.util.List;

public class EmployeeWithDepartmentDAO {

JdbcTemplate jdbcTemplate;

// Setter for JdbcTemplate

public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {

this.jdbcTemplate = jdbcTemplate;

}

// Method to fetch all employees with their department names

public List<String> getEmployeesWithDepartments() {

String sql = "SELECT e.employee\_id, e.employee\_name, d.department\_name " +

"FROM Employee e " +

"LEFT JOIN Department d ON e.department\_id = d.department\_id";

return jdbcTemplate.query(sql, new ResultSetExtractor<List<String>>() {

@Override

public List<String> extractData(ResultSet rs) throws SQLException {

List<String> result = new ArrayList<>();

while (rs.next()) {

int employeeId = rs.getInt("employee\_id");

String employeeName = rs.getString("employee\_name");

String departmentName = rs.getString("department\_name");

result.add("Employee ID: " + employeeId + ", Name: " + employeeName +

", Department: " + (departmentName != null ? departmentName : "No Department"));

}

return result;

}

});

}

// Method to retrieve employee details by name

@SuppressWarnings("deprecation")

public Employee getEmployeeByName(String name) {

String sql = "SELECT employee\_id, employee\_name, department\_id FROM Employee WHERE employee\_name = ?";

return jdbcTemplate.queryForObject(sql, new Object[]{name}, new RowMapper<Employee>() {

@Override

public Employee mapRow(ResultSet rs, int rowNum) throws SQLException {

Employee employee = new Employee();

employee.setEmployeeId(rs.getInt("employee\_id"));

employee.setEmployeeName(rs.getString("employee\_name"));

employee.setDepartmentId(rs.getInt("department\_id"));

return employee;

}

});

}

}

**StoredProcedureExecutor.java**

package packP5Code1;

import org.springframework.jdbc.core.JdbcTemplate;

import org.springframework.jdbc.core.RowMapper;

import org.springframework.jdbc.core.SqlOutParameter;

import org.springframework.jdbc.core.SqlParameter;

import org.springframework.jdbc.object.StoredProcedure;

import java.sql.ResultSet;

import java.sql.SQLException;

import java.util.Map;

public class StoredProcedureExecutor {

private JdbcTemplate jdbcTemplate;

// Setter for JdbcTemplate

public void setJdbcTemplate(JdbcTemplate jdbcTemplate) {

this.jdbcTemplate = jdbcTemplate;

}

// Execute the stored procedure

public Employee getEmployeeById(int employeeId) {

String procedureName = "GetEmployeeById";

// Call the stored procedure

return jdbcTemplate.queryForObject("{CALL " + procedureName + "(?)}", new Object[]{employeeId}, new RowMapper<Employee>() {

@Override

public Employee mapRow(ResultSet rs, int rowNum) throws SQLException {

Employee employee = new Employee();

employee.setEmployeeId(rs.getInt("employee\_id"));

employee.setEmployeeName(rs.getString("employee\_name"));

employee.setDepartmentId(rs.getInt("department\_id"));

return employee;

}

});

}

}

**code1Bean.xml**

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

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:context="http://www.springframework.org/schema/context"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context.xsd">

<!-- DataSource Configuration -->

<bean id="dataSource"

class="org.apache.commons.dbcp2.BasicDataSource">

<property name="driverClassName"

value="com.mysql.cj.jdbc.Driver" />

<property name="url"

value="jdbc:mysql://localhost:3306/javapractical5" />

<property name="username" value="root" />

<property name="password" value="" />

</bean>

<!-- JdbcTemplate Bean -->

<bean id="jdbcTemplate"

class="org.springframework.jdbc.core.JdbcTemplate">

<property name="dataSource" ref="dataSource" />

</bean>

<!-- DepartmentDAO Bean -->

<bean id="departmentDAO" class="packP5Code1.DepartmentDAO">

<property name="jdbcTemplate" ref="jdbcTemplate" />

</bean>

<!-- EmployeeDAO Bean -->

<bean id="employeeDAO" class="packP5Code1.EmployeeDAO">

<property name="jdbcTemplate" ref="jdbcTemplate" />

</bean>

<bean id="employeeWithDepartmentDAO"

class="packP5Code1.EmployeeWithDepartmentDAO">

<property name="jdbcTemplate" ref="jdbcTemplate" />

</bean>

<bean id="storedProcedureExecutor"

class="packP5Code1.StoredProcedureExecutor">

<property name="jdbcTemplate" ref="jdbcTemplate" />

</bean>

</beans>

**App.java**

package packP5Code1;

import java.util.List;

import java.util.Map;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class App {

public static void main(String[] args) {

// Load Spring Configuration

ClassPathXmlApplicationContext ctx = new ClassPathXmlApplicationContext("code1Bean.xml");

// Get EmployeeDAO Bean

EmployeeDAO employeeDAO = (EmployeeDAO) ctx.getBean("employeeDAO");

DepartmentDAO departmentDAO = (DepartmentDAO) ctx.getBean("departmentDAO");

// Insert Departments First

System.out.println("Adding Departments:");

departmentDAO.addDepartment(new Department(101, "HR"));

departmentDAO.addDepartment(new Department(102, "Finance"));

departmentDAO.addDepartment(new Department(103, "Food"));

// Add Employees with Valid department\_id

System.out.println("Adding Employees:");

employeeDAO.addEmployee(new Employee(1, "Tejas", 101)); // Alice belongs to HR

employeeDAO.addEmployee(new Employee(2, "Manjula", 102)); // Bob belongs to Finance

employeeDAO.addEmployee(new Employee(3, "Nikhil", 103));

// List All Employees

System.out.println("Listing All Employees:");

employeeDAO.listAllEmployees();

// Update Employee

System.out.println("Updating Employee:");

employeeDAO.updateEmployee(1, "Kashipra", 101);

// List All Employees After Update

System.out.println("Listing All Employees After Update:");

employeeDAO.listAllEmployees();

// Delete Employee

System.out.println("Deleting Employee:");

employeeDAO.deleteEmployee(2);

// List All Employees After Deletion

System.out.println("Listing All Employees After Deletion:");

employeeDAO.listAllEmployees();

/\*-------------------------------------------------------------------------------------\*/

// Read All Departments

System.out.println("Listing All Departments:");

List<Map<String, Object>> departments = departmentDAO.getAllDepartments();

for (Map<String, Object> dept : departments) {

System.out.println(dept);

}

// Update Department

System.out.println("Updating Department:");

departmentDAO.updateDepartment(101, "Human Resources");

// Read All Departments After Update

System.out.println("Listing All Departments After Update:");

departments = departmentDAO.getAllDepartments();

for (Map<String, Object> dept : departments) {

System.out.println(dept);

}

// Delete Department

System.out.println("Deleting Department:");

departmentDAO.deleteDepartment(102);

// Read All Departments After Deletion

System.out.println("Listing All Departments After Deletion:");

departments = departmentDAO.getAllDepartments();

for (Map<String, Object> dept : departments) {

System.out.println(dept);

}

// Get EmployeeWithDepartmentDAO Bean

EmployeeWithDepartmentDAO dao = (EmployeeWithDepartmentDAO) ctx.getBean("employeeWithDepartmentDAO");

// Display Employees with Department Names

System.out.println("Employees with Department Names:");

List<String> employeeDetails = dao.getEmployeesWithDepartments();

for (String detail : employeeDetails) {

System.out.println(detail);

}

// Fetch Employee by Name

String employeeName = "Nikhil";

Employee employee = dao.getEmployeeByName(employeeName);

if (employee != null) {

System.out.println("Employee Details: " + employee);

} else {

System.out.println("No employee found with the name: " + employeeName);

}

// Get StoredProcedureExecutor Bean

StoredProcedureExecutor executor = (StoredProcedureExecutor) ctx.getBean("storedProcedureExecutor");

// Fetch Employee by ID

int employeeId = 1; // Input Employee ID

Employee employee1 = executor.getEmployeeById(employeeId);

if (employee1 != null) {

System.out.println("Employee Details: " + employee1);

} else {

System.out.println("No employee found with ID: " + employeeId);

}

ctx.close();

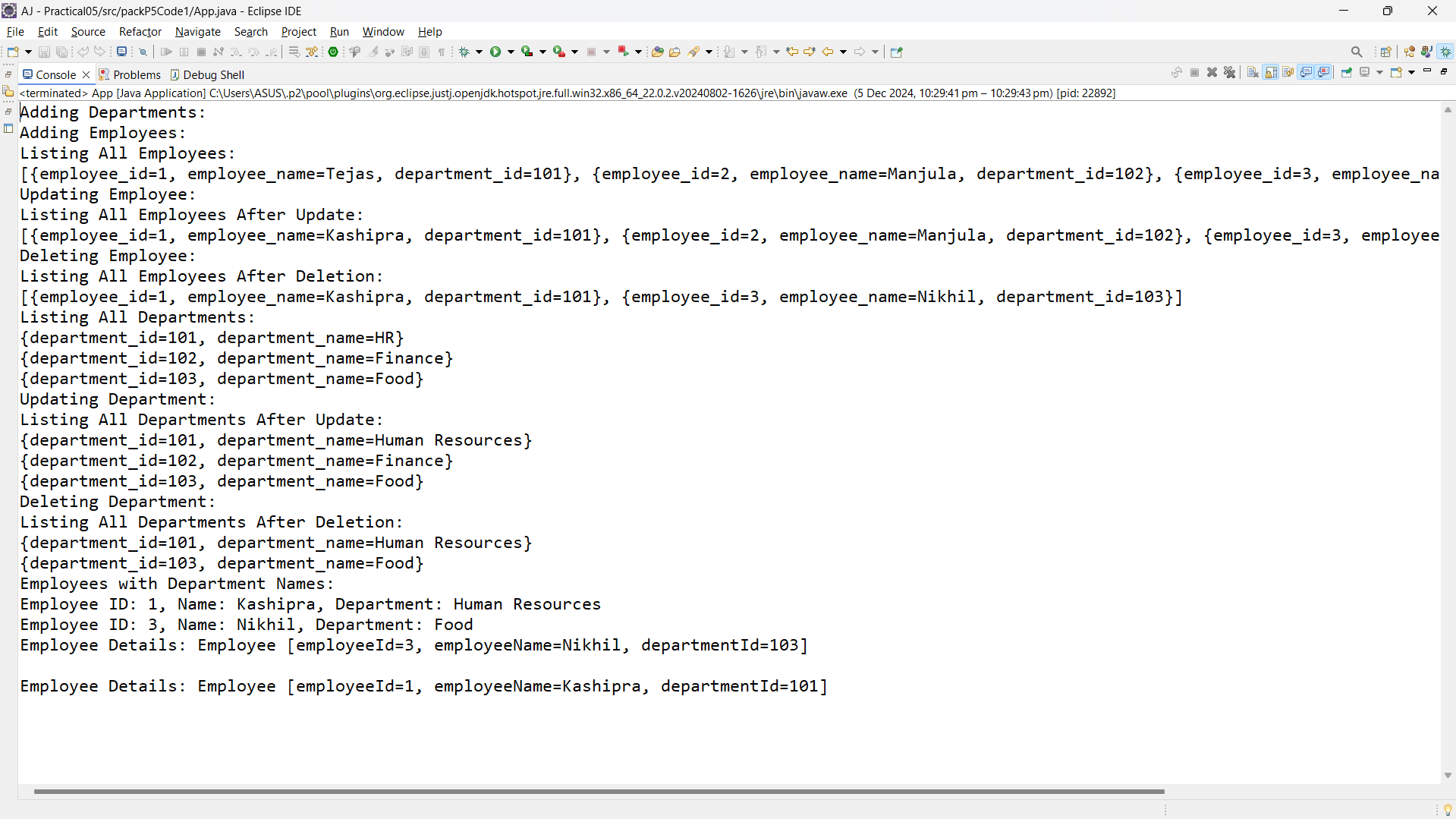
}

}

**Output :-**

****

****

****