**Spring JDBC**

**1. Spring Boot Application Structure with JdbcTemplate**

**1.1Spring Boot Application Setup**

1. This application uses Spring Boot with JdbcTemplate to connect to an H2 database, perform SQL queries, and map results to Java objects.
2. The application consists of the following main parts:
   * + **Main Application Class**: SpringBootApp
     + **Model Class**: Student
     + **Repository Class**: StudentRepo
     + **Service Class**: StudentService

**1.1.1 SpringBootApp (Main Application Class)**

**Code:**

java

Copy code

package JDBCTemplate.app;

import java.util.List;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ApplicationContext;

import JDBCTemplate.app.Model.Student;

import JDBCTemplate.app.Service.StudentService;

@SpringBootApplication

public class SpringBootApp {

public static void main(String[] args) {

ApplicationContext context = SpringApplication.run(SpringBootApp.class, args);

// Creating a Student object

Student s = context.getBean(Student.class);

s.setName("Vikash Mishra");

s.setRollnum(95);

s.setMarks(100);

// Getting the StudentService bean to interact with the database

StudentService service = context.getBean(StudentService.class);

service.addStudents(s); // Saving a student to the database

// Retrieving all students from the database

List<Student> students = service.getStudents();

System.out.println(students);

}

}

**1.1.2 Explanation:**

* This is the main entry point for the Spring Boot application.
* It starts the application context, which loads all beans defined in the project.
* The StudentService bean is used to add a student to the database and retrieve all students.

**1.1.3 Student (Model Class)**

**Code:**

java

Copy code

package JDBCTemplate.app.Model;

import org.springframework.context.annotation.Scope;

import org.springframework.stereotype.Component;

@Component

@Scope("prototype")

public class Student {

private int rollnum;

private int marks;

private String name;

// Getters and Setters

public int getRollnum() { return rollnum; }

public void setRollnum(int rollnum) { this.rollnum = rollnum; }

public int getMarks() { return marks; }

public void setMarks(int marks) { this.marks = marks; }

public String getName() { return name; }

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

@Override

public String toString() {

return "Student [rollnum=" + rollnum + ", marks=" + marks + ", name=" + name + "]";

}

}

**1.1.6 Explanation:**

* The Student class represents a student entity in the application.
* It has three attributes: rollnum, marks, and name, with their respective getters and setters.
* The @Component annotation makes it a Spring Bean, and @Scope("prototype") ensures a new instance is created each time it's requested.

**1.1.7 StudentRepo (Repository Class)**

**Code:**

java

Copy code

package JDBCTemplate.app.DAO;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.jdbc.core.JdbcTemplate;

import org.springframework.jdbc.core.RowMapper;

import org.springframework.stereotype.Repository;

import JDBCTemplate.app.Model.Student;

@Repository

public class StudentRepo {

private final JdbcTemplate jdbc;

@Autowired

public StudentRepo(JdbcTemplate jdbc) {

this.jdbc = jdbc;

}

public void save(Student s) {

String sql = "insert into student(name,rollnum,marks) values(?, ?, ?)";

int rows = jdbc.update(sql, s.getName(), s.getRollnum(), s.getMarks());

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

}

public List<Student> findAll() {

String sql = "select \* from student";

RowMapper<Student> rowMapper = new RowMapper<Student>() {

@Override

public Student mapRow(java.sql.ResultSet rs, int rowNum) throws java.sql.SQLException {

Student student = new Student();

student.setName(rs.getString("name"));

student.setRollnum(rs.getInt("rollnum"));

student.setMarks(rs.getInt("marks"));

return student;

}

};

return jdbc.query(sql, rowMapper);

}

}

**1.1.8 Explanation:**

* StudentRepo is the data access layer. It directly interacts with the database using JdbcTemplate.
* **save(Student s)**:
  + This method inserts a new student record into the student table.
  + jdbc.update(sql, s.getName(), s.getRollnum(), s.getMarks()): Executes the SQL INSERT statement.
* **findAll()**:
  + This method retrieves all student records from the student table.
  + **RowMapper<Student>**: This is an anonymous inner class that maps rows of a ResultSet to Student objects. It iterates over each row and populates a Student object with data from each column (name, rollnum, marks).

**1.1.9 StudentService (Service Class)**

**Code:**

java

Copy code

package JDBCTemplate.app.Service;

import java.util.List;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import JDBCTemplate.app.DAO.StudentRepo;

import JDBCTemplate.app.Model.Student;

@Service

public class StudentService {

private final StudentRepo studentRepo;

@Autowired

public StudentService(StudentRepo studentRepo) {

this.studentRepo = studentRepo;

}

public void addStudents(Student s) {

studentRepo.save(s);

}

public List<Student> getStudents() {

return studentRepo.findAll();

}

}

**1.1.10 Explanation:**

* StudentService is the service layer that bridges the controller and the repository.
* **addStudents(Student s)**: Calls studentRepo.save(s) to add a student to the database.
* **getStudents()**: Calls studentRepo.findAll() to retrieve a list of all students.

**1.2 JdbcTemplate, update, and query Methods**

**1.2.1 JdbcTemplate**:

* It is a Spring-provided class used to simplify database interactions.
* JdbcTemplate handles resource management, such as opening/closing connections and exception handling.

**1.2.2update Method**:

* jdbc.update(sql, args...) is used to execute SQL INSERT, UPDATE, or DELETE statements.
* In save(Student s), update is used to insert data into the student table.

**1.2.3 query Method**:

* jdbc.query(sql, rowMapper) is used to execute a SQL SELECT query and map the results to a list of objects.
* In findAll(), query uses a RowMapper to map each row to a Student object.

**1.3 Explanation of RowMapper**

**1.3.1 RowMapper<Student>**:

* RowMapper is an interface in Spring that maps rows of a ResultSet to objects.
* The anonymous inner class implementation in findAll() creates a new Student object for each row in the ResultSet.
* mapRow(ResultSet rs, int rowNum):
  + - Takes each row of the result set and sets Student properties using data from columns.
  1. **Integration of schema.sql and data.sql**

**1.4.1 schema.sql**:

* This file contains SQL statements to define the database schema (e.g., creating tables).
* create table student(...) defines the structure of the student table.
  + 1. **data.sql**:
* This file contains SQL statements to populate the database with initial data.
* insert into student (...) values (...) adds sample student records.
  + 1. **Automatic Execution by Spring Boot**:
* Spring Boot automatically executes schema.sql and data.sql files located in src/main/resources when the application starts.
* This behavior allows the database to be initialized without manual calls in the code.

This structure, along with JdbcTemplate, creates a clean, maintainable application with separation of concerns:

* **Model Layer**: Student represents the data.
* **Repository Layer**: StudentRepo interacts with the database.
* **Service Layer**: StudentService provides business logic.
* **Main Application**: SpringBootApp drives the application.