**EXERCISE 1**

**1.Project Structure**

spring-data-jpa-handson/

├── src/

│ ├── main/

│ ├── java/

│ │ └── com/example/demo/

│ │ ├── DemoApplication.java

│ │ ├── Student.java

│ │ └── StudentRepository.java

│ └── resources/

│ ├── application.properties

└── pom.xml

**2.pom.xml**

<project xmlns="http://maven.apache.org/POM/4.0.0"

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

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0

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

<modelVersion>4.0.0</modelVersion>

<groupId>com.example</groupId>

<artifactId>spring-data-jpa-handson</artifactId>

<version>1.0-SNAPSHOT</version>

<parent>

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

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

<version>2.7.18</version>

</parent>

<dependencies>

<!-- Spring Data JPA -->

<dependency>

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

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

</dependency>

<!-- H2 Database -->

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

<!-- Spring Boot Starter -->

<dependency>

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

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

</dependency>

</dependencies>

<properties>

<java.version>1.8</java.version>

</properties>

</project>

**3. src/main/resources/application.properties**

spring.datasource.url=jdbc:h2:mem:testdb

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

spring.h2.console.enabled=true

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=update

**4. src/main/java/com/example/demo/Student.java**

package com.example.demo;

import javax.persistence.Entity;

import javax.persistence.Id;

@Entity

public class Student {

@Id

private Long id;

private String name;

private String department;

public Student() {}

public Student(Long id, String name, String department) {

this.id = id;

this.name = name;

this.department = department;

}

// 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 getDepartment() { return department; }

public void setDepartment(String department) { this.department = department; }

}

**5. src/main/java/com/example/demo/StudentRepository.java**

package com.example.demo;

import org.springframework.data.jpa.repository.JpaRepository;

public interface StudentRepository extends JpaRepository<Student, Long> {

}

**6. src/main/java/com/example/demo/DemoApplication.java**

package com.example.demo;

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

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class DemoApplication implements CommandLineRunner {

@Autowired

private StudentRepository repository;

public static void main(String[] args) {

SpringApplication.run(DemoApplication.class, args);

}

@Override

public void run(String... args) {

// Saving data to the database

repository.save(new Student(1L, "Arun", "EEE"));

repository.save(new Student(2L, "Priya", "CSE"));

repository.save(new Student(3L, "Ravi", "ECE"));

System.out.println("All students in database:");

repository.findAll().forEach(student ->

System.out.println(student.getId() + " " + student.getName() + " " + student.getDepartment()));

}

}

**Build**

mvn clean install

**Run**

mvn spring-boot:run

**Output**

All students in database:

1 Arun EEE

2 Priya CSE

3 Ravi ECE

**EXERCISE 2**

| **Aspect** | **JPA** | **Hibernate** | **Spring Data JPA** |
| --- | --- | --- | --- |
| **What it is** | A **specification (interface)** for object-relational mapping in Java | An **implementation of JPA** (ORM framework) | A **Spring project** that simplifies data access using JPA |
| **Type** | API/Standard | Framework | Spring abstraction layer |
| **Role** | Provides guidelines for mapping Java objects to database tables | Provides the actual implementation and additional features beyond JPA | Simplifies JPA implementation with **ready-made CRUD, paging, sorting, and query methods** |
| **Implementation?** | Cannot be used directly, requires a provider like Hibernate | Implements JPA and can be used directly | Uses JPA provider (Hibernate) internally |
| **Entity management** | Defines @Entity, @Id, @OneToMany, etc. | Uses JPA annotations, supports advanced features like caching, interceptors | Uses JpaRepository interfaces for automatic method creation |
| **Boilerplate code** | Requires EntityManager handling and queries | Provides SessionFactory and advanced configuration | Reduces boilerplate by using Spring repositories |
| **Query handling** | Uses JPQL, Criteria API | Supports JPQL, Criteria API, HQL | Supports method name-based query creation, JPQL, and native queries easily |
| **Examples** | EntityManager.persist(), find(), merge() | session.save(), session.update(), session.createQuery() | findAll(), findById(), findByName(), paging, and sorting methods |