**Spring Data JPA with Spring Boot, Hibernate**

**Spring Data JPA - Quick Example**

**Code:**

**Student.java**

package org.example;  
  
import jakarta.persistence.Entity;  
import jakarta.persistence.GeneratedValue;  
import jakarta.persistence.GenerationType;  
import jakarta.persistence.Id;  
  
@Entity  
public class Student {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
 private String name;  
 private String department;  
  
 public Student() {}  
  
 public Student(String name, String department) {  
 this.name = name;  
 this.department = department;  
 }  
  
 public String toString() {  
 return "Student[id=" + id + ", name=" + name + ", department=" + department + "]";  
 }  
}

**StudentRepository.java:**

package org.example;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
  
public interface StudentRepository extends JpaRepository<Student, Long> {  
}

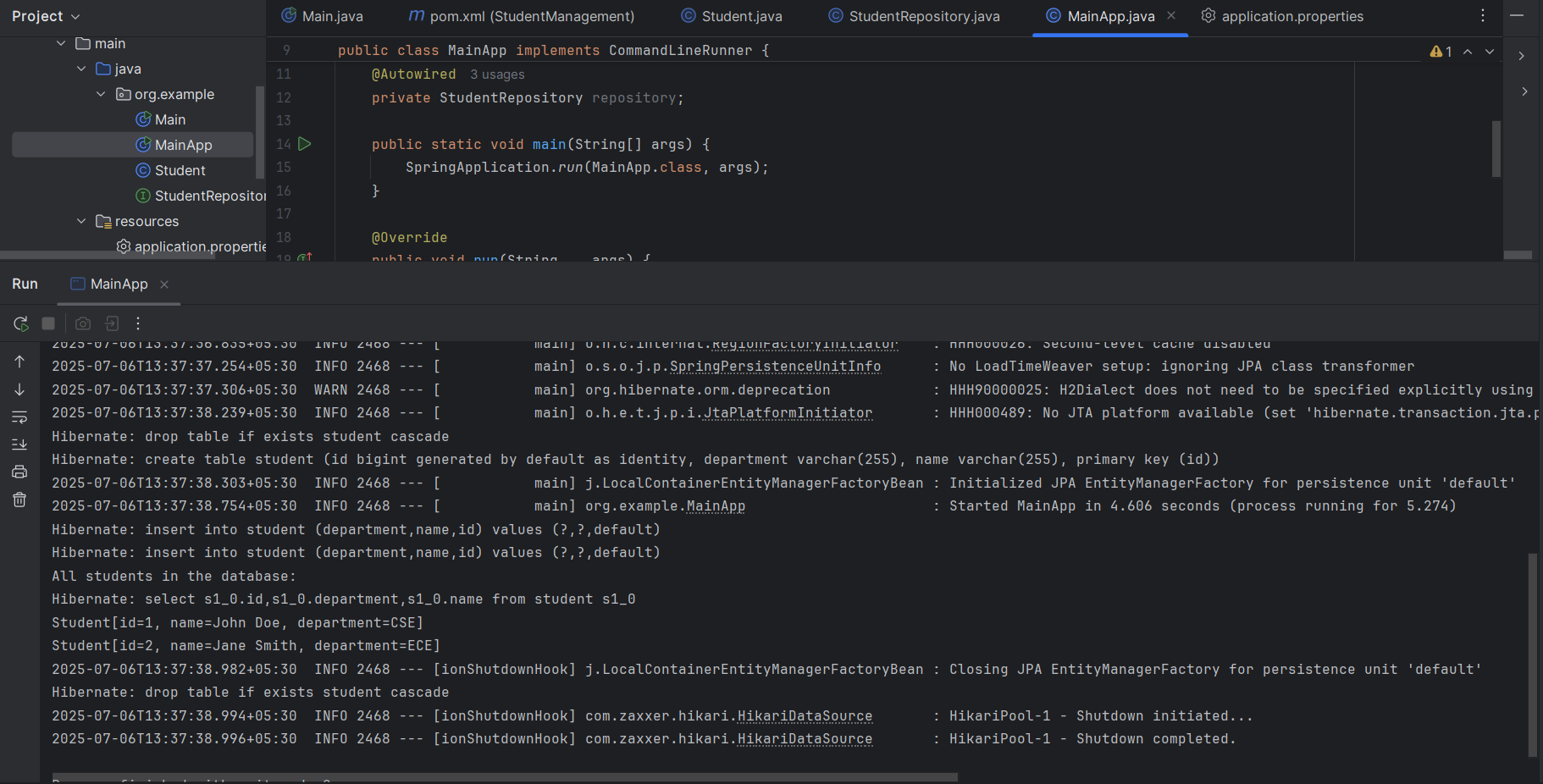
**application.properties:**

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.jpa.show-sql=true  
spring.h2.console.enabled=true

**MainApp.java:**

package org.example;  
  
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 MainApp implements CommandLineRunner {  
  
 @Autowired  
 private StudentRepository repository;  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(MainApp.class, args);  
 }  
  
 @Override  
 public void run(String... args) {  
 repository.save(new Student("John Doe", "CSE"));  
 repository.save(new Student("Jane Smith", "ECE"));  
  
 System.*out*.println("All students in the database:");  
 repository.findAll().forEach(System.*out*::println);  
 }  
}

**output:**

****

**Difference between JPA, Hibernate and Spring Data JPA**

**JPA (Java Persistence API)**

A Java specification that standardizes how Java objects are mapped to relational database tables using ORM (Object-Relational Mapping). It provides interfaces and annotations but does not implement them; requires a provider like Hibernate.

**Hibernate**

An ORM framework for Java that implements JPA and also provides extra features beyond JPA, such as caching, batch processing, and better lazy loading. It allows mapping Java objects to database tables and handles SQL generation automatically.

**Spring Data JPA**

A Spring project that simplifies JPA-based data access by providing repository abstractions, reducing boilerplate code, and enabling CRUD operations without writing implementation code. It internally uses JPA providers like Hibernate.

|  |  |  |  |
| --- | --- | --- | --- |
| **Aspect** | **JPA** | **Hibernate** | **Spring Data JPA** |
| Type | Specification (API) | Implementation of JPA and ORM framework | Abstraction over JPA with repository support |
| Provided By | Java (javax.persistence) | Red Hat | Spring Framework |
| Purpose | Standardize ORM in Java | Provides ORM with additional features | Simplifies data access and repository creation |
| Implementation | Needs provider (like Hibernate) | Provides JPA implementation and native features | Uses JPA providers internally |
| Query Language | JPQL | HQL + JPQL | Uses JPQL, method naming conventions, @Query support |
| Boilerplate Code | Requires writing EntityManager, DAO manually | Less boilerplate but still needs Session/DAO | Reduces boilerplate with CRUD repository interfaces |
| Ease of Use | More configuration and code needed | Easier than JPA alone | Easiest, focuses on productivity |
| Advanced Features | Only what JPA specification provides | Caching, batch processing, interceptors, etc. | Paging, sorting, dynamic queries with ease |
| Transaction Handling | Requires manual setup or Spring support | Integrated but needs management | Integrated with Spring’s transaction management |
| Usage Scenario | When vendor independence is required | When advanced ORM features are needed | When rapid development with minimal code is needed |