**Week3\_Spring Data JPA - Quick Example**

**Configure Dependencies in pom.xml**

*<?*xml version="1.0" encoding="UTF-8"*?>*<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>  
  
 *<!-- Spring Boot Parent provides smart dependency management -->* <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>2.7.5</version>  
 <relativePath/> *<!-- lookup parent from repository -->* </parent>  
  
 <groupId>com.example</groupId>  
 <artifactId>jpa-hibernate-showdown</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <properties>  
 <java.version>17</java.version>  
 </properties>  
  
 <dependencies>  
 *<!-- 1. DEPENDENCY FOR SPRING DATA JPA & SPRING BOOT -->  
 <!-- This starter pulls in Spring Data JPA, Hibernate as the provider, and Spring Core -->* <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-jpa</artifactId>  
 </dependency>  
  
 *<!-- 2. DEPENDENCY FOR OUR DATABASE -->  
 <!-- H2 is a lightweight, in-memory database. Perfect for demos, no installation needed! -->* <dependency>  
 <groupId>com.h2database</groupId>  
 <artifactId>h2</artifactId>  
 <scope>runtime</scope>  
 </dependency>  
 *<!-- 3. DEPENDENCY FOR PURE HIBERNATE (Not included in starter) -->  
 <!-- We add this explicitly to demonstrate the 'pure' Hibernate way -->* <dependency>  
 <groupId>org.hibernate</groupId>  
 <artifactId>hibernate-core</artifactId>  
 <version>${hibernate.version}</version>  
 </dependency>  
 </dependencies>  
  
 <build>  
 <plugins>  
 *<!-- Spring Boot Maven plugin to package the application -->* <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 </plugins>  
 </build>  
</project>  
  
 *<!-- This starter pulls in Spring Data JPA, Hibernate as the provider, and Spring Core -->*

**Create the Common Employee Entity. Create the com.example.entity package and add the Employee.java class.**

package com.example.entity;  
import javax.persistence.\*;  
*// @Entity: This is the most important annotation. It marks this Java class as  
// something that can be persisted to a database. Hibernate, JPA, and Spring Data JPA  
// all understand this annotation.*@Entity  
*// @Table: This annotation is optional but recommended. It explicitly tells the  
// persistence provider what to name the table in the database.*@Table(name = "employees")  
public class Employee {  
 *// @Id: Marks this field as the primary key for the database table.* @Id  
 *// @GeneratedValue: Tells the persistence provider how the ID should be generated.  
 // GenerationType.IDENTITY means we are delegating ID generation to the database itself,  
 // which will auto-increment the value. This is a common and robust strategy.* @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Integer id;  
 @Column(name = "name") *// Maps this field to a column named 'name'.* private String name;  
  
 *// A no-argument constructor is required by JPA.* public Employee() {}  
 public Employee(String name) {  
 this.name = name;  
 }  
 *// --- Getters, Setters, and toString() ---  
 // These are standard Java methods needed for accessing the object's data.* public Integer getId() {  
 return id;  
 }  
 public void setId(Integer id) {  
 this.id = id;  
 }  
 public String getName() {  
 return name;  
 }  
 public void setName(String name) {  
 this.name = name;  
 }  
 @Override  
 public String toString() {  
 return "Employee{" +  
 "id=" + id +", name='" + name + '\'' +'}';  
 }  
}

**Create the Hibernate Data Access Object (DAO). Configure Hibernate (hibernate.cfg.xml)**

<!DOCTYPE hibernate-configuration PUBLIC  
 "-//Hibernate/Hibernate Configuration DTD 3.0//EN"  
 "http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd"*>*<hibernate-configuration>  
 <session-factory>  
 *<!-- Database connection settings for our H2 in-memory database -->* <property name="connection.driver\_class">org.h2.Driver</property>  
 *<!-- The 'mem:hibernatedb' part creates a temporary in-memory DB named 'hibernatedb' -->* <property name="connection.url">jdbc:h2:mem:hibernatedb</property>  
 <property name="connection.username">sa</property>  
 <property name="connection.password"></property>  
 *<!-- The SQL dialect helps Hibernate generate syntax specific to our database (H2) -->* <property name="dialect">org.hibernate.dialect.H2Dialect</property>  
  
 *<!-- 'hbm2ddl.auto' = 'create-drop': Automatically creates the database schema when the  
 SessionFactory is created and drops it when it's closed. Perfect for testing. -->* <property name="hbm2ddl.auto">create-drop</property>  
  
 *<!-- 'show\_sql' = 'true': Prints all the generated SQL statements to the console.  
 Extremely useful for debugging! -->* <property name="show\_sql">true</property>  
  
 *<!-- Tell Hibernate which @Entity classes to look for -->* <mapping class="com.example.entity.Employee"/>  
 </session-factory>  
</hibernate-configuration>

**4.1. Configure Spring Boot (application.properties)**

package com.example.dao.hibernate;  
public class x {  
}  
# Configure the H2 database connection for Spring Boot  
spring.datasource.url=jdbc:h2:mem:springdb  
spring.datasource.driverClassName=org.h2.Driver  
spring.datasource.username=sa  
spring.datasource.password=  
 # We don't need ddl-auto here, Spring Boot is smart enough to handle it.  
 # Show the generated SQL in the console  
spring.jpa.show-sql=true

**Create the Repository Interface(EmployeeRepository.java)**

package com.example.repository;  
import com.example.entity.Employee;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.stereotype.Repository;  
  
@Repository *// Tells Spring this is a repository bean*public interface EmployeeRepository extends JpaRepository<Employee, Integer> {  
 *// THAT'S IT!  
 // All common methods like save(), findById(), findAll(), delete() are  
 // automatically available. No implementation code is needed.*}

**Create the Service Layer(EmployeeService.java)**

package com.example.service;  
import com.example.entity.Employee;  
import com.example.repository.EmployeeRepository;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
import org.springframework.transaction.annotation.Transactional;  
  
@Service *// Tells Spring this is a service bean*public class EmployeeService {  
  
 @Autowired *// Spring automatically injects the EmployeeRepository bean here* private EmployeeRepository employeeRepository;  
  
 @Transactional *// Spring handles the transaction automatically!* public void addEmployee(Employee employee) {  
 *// No try-catch, no begin/commit/rollback, no closing sessions.* employeeRepository.save(employee);  
 System.*out*.println("Spring Data JPA: Successfully saved employee " + employee.getName());  
 }  
}

**To run create the Main Class(HibernateMain.java)**

package com.example;  
import com.example.dao.hibernate.EmployeeHibernateDao;  
import com.example.entity.Employee;  
  
public class HibernateMain {  
 public static void main(String[] args) {  
 System.*out*.println("--- RUNNING PURE HIBERNATE EXAMPLE ---");  
 EmployeeHibernateDao hibernateDao = new EmployeeHibernateDao();  
 Employee employeeToSave = new Employee("Alice (from Hibernate)");  
 hibernateDao.addEmployee(employeeToSave);  
 System.*out*.println("--- HIBERNATE EXAMPLE FINISHED ---\n");  
 }  
}

**Output**

