**Week 3- Spring-data-jpa hands-on**

**Name- Adi Jain**

**Superset Id- 6390400**

**Hands-on 1**

**Spring Data JPA - Quick Example**  
Create an Eclipse Project using Spring Initializr.

**Code**

**File Name :** **application.properties**

spring.application.name=orm-learn

# Spring Framework and application log

logging.level.org.springframework=info

logging.level.com.cognizant=debug

# Hibernate logs for displaying executed SQL, input and output

logging.level.org.hibernate.SQL=trace

logging.level.org.hibernate.type.descriptor.sql=trace

# Log pattern

logging.pattern.console=%d{dd-MM-yy} %d{HH:mm:ss.SSS} %-20.20thread %5p %-25.25logger**{25}** %25M %4L %m%n

# Database configuration

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.datasource.url=jdbc:mysql://localhost:3306/ormlearn

spring.datasource.username=root

spring.datasource.password=123456

# Hibernate configuration

spring.jpa.hibernate.ddl-auto=validate

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect

**========================================================**

**File Name :** **OrmLearnApplication.java**

**package** com.cognizant.orm\_learn;

**import** java.util.List;

**import** com.cognizant.orm\_learn.model.Country;

**import** com.cognizant.orm\_learn.service.CountryService;

**import** org.slf4j.Logger;

**import** org.slf4j.LoggerFactory;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

**import** org.springframework.context.ApplicationContext;

@SpringBootApplication

**public** **class** OrmLearnApplication {

**private** **static** com.cognizant.orm\_learn.service.CountryService *countryService*;

**private** **static** **final** Logger ***LOGGER*** = LoggerFactory.*getLogger*(OrmLearnApplication.**class**);

**public** **static** **void** main(String[] args) {

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

// 👇 Retrieve the CountryService bean from the context

CountryService countryService = context.getBean(CountryService.**class**);

// 👇 Call your test method

*testGetAllCountries*(countryService);

}

**private** **static** **void** testGetAllCountries(CountryService countryService) {

***LOGGER***.info("Start");

List<Country> countries = countryService.getAllCountries();

***LOGGER***.debug("countries={}", countries);

***LOGGER***.info("End");

}

}

**================================================================**

**File Name: Country.java**

**package** com.cognizant.orm\_learn.model;

**import** jakarta.persistence.Entity;

**import** jakarta.persistence.Id;

**import** jakarta.persistence.Column;

**import** jakarta.persistence.Table;

@Entity

@Table(name="country")

**public** **class** Country {

@Id

@Column(name="co\_code")

**private** String co\_code;

@Column(name="co\_name")

**private** String co\_name;

// getters and setters

// toString()

@Override

**public** String toString() {

**return** "Country [code=" + co\_code + ", name=" + co\_name + "]";

}

}

**========================================================**

**File Name: CountryRepository.java**

**package** com.cognizant.orm\_learn.repository;

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

**import** org.springframework.stereotype.Repository;

**import** com.cognizant.orm\_learn.model.Country;

@Repository

**public** **interface** CountryRepository **extends** JpaRepository<Country, String> {

}

**========================================================**

**File Name:CountryService.java**

**package** com.cognizant.orm\_learn.service;

**import** java.util.List;

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

**import** org.springframework.stereotype.Service;

**import** org.springframework.transaction.annotation.Transactional;

**import** com.cognizant.orm\_learn.model.Country;

**import** com.cognizant.orm\_learn.repository.CountryRepository;

@Service

**public** **class** CountryService {

@Autowired

**private** CountryRepository countryRepository;

@Transactional

**public** List<Country> getAllCountries() {

**return** countryRepository.findAll();

}

}

**========================================================**

**File Name: OrmLearnApplicationTests.java**

**package** com.cognizant.orm\_learn;

**import** org.junit.jupiter.api.Test;

**import** org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest

**class** OrmLearnApplicationTests {

@Test

**void** contextLoads() {

}

}

**========================================================**

**File Name: 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 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

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

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

<version>3.5.3</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>com.cognizant</groupId>

<artifactId>orm-learn</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>orm-learn</name>

<description>Demo project for Spring Data JPA and Hibernate</description>

<url/>

<licenses>

<license/>

</licenses>

<developers>

<developer/>

</developers>

<scm>

<connection/>

<developerConnection/>

<tag/>

<url/>

</scm>

<properties>

<java.version>17</java.version>

</properties>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

<optional>true</optional>

</dependency>

<dependency>

<groupId>com.mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.hibernate.orm</groupId>

<artifactId>hibernate-core</artifactId>

<version>6.4.4.Final</version> <!-- or a version matching your Spring Boot version -->

</dependency>

<dependency>

<groupId>jakarta.persistence</groupId>

<artifactId>jakarta.persistence-api</artifactId>

<version>3.1.0</version> <!-- Or latest -->

</dependency>

<dependency>

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

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

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

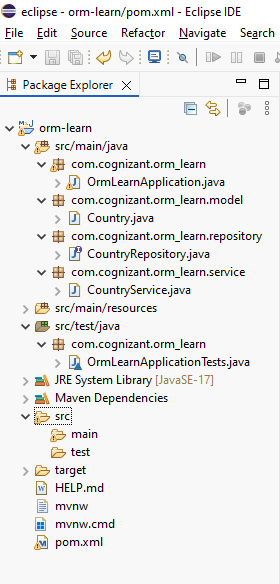
</build>

</project>

**========================================================**

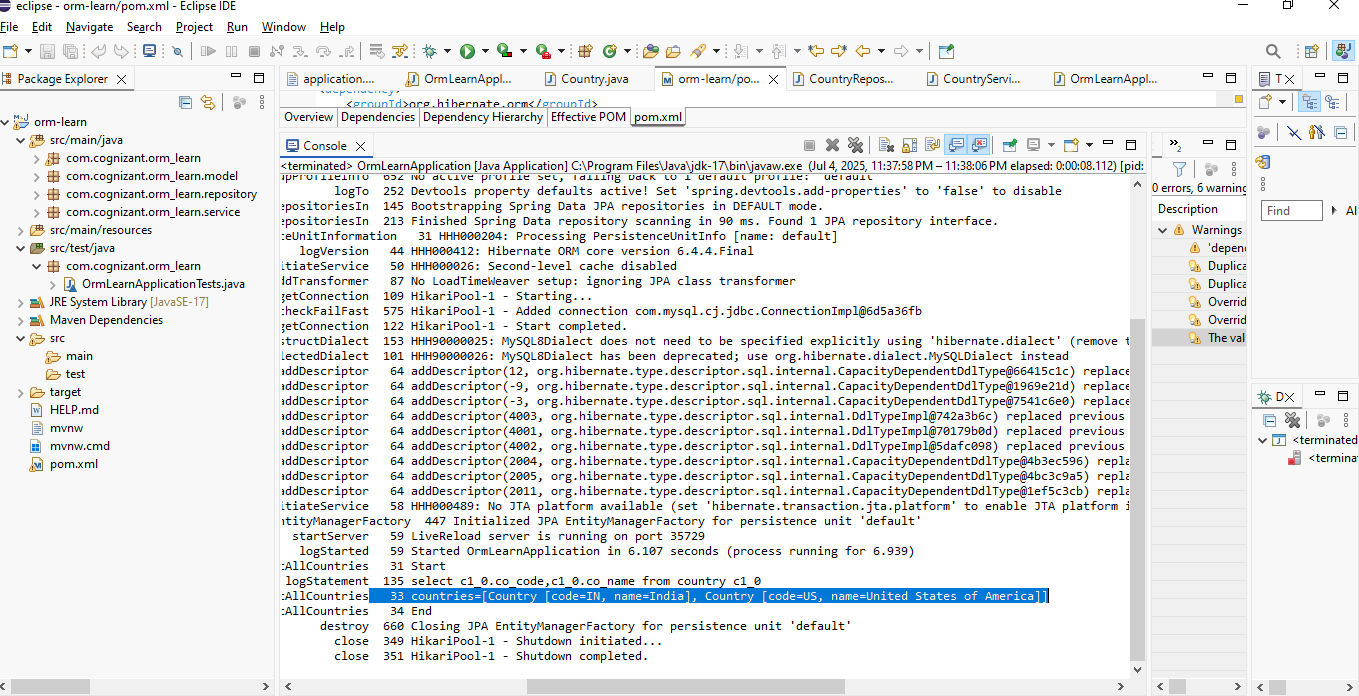
**Output**

**Folder Structure**



**========================================================**

**Final Output**



**========================================================**

**Hands on 4**

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

**Solution**

In Java enterprise applications, handling data persistence (saving and retrieving data from databases) is a common requirement. Over the years, several tools and specifications have evolved to simplify and standardize this process. Among them, **JPA (Java Persistence API)**, **Hibernate**, and **Spring Data JPA** are the most popular. Though often used together, they serve different roles and have distinct purposes. Understanding the differences and how they work together is crucial for building robust data-driven applications.

**1. Java Persistence API (JPA)**

### ****What is JPA?****

JPA stands for **Java Persistence API**. It is a **Java specification** (part of Java EE and Jakarta EE) that defines a standard way to manage **relational data in Java applications**. It provides a set of interfaces and annotations to map Java objects to database tables.

### ****Key Characteristics of JPA****

* **Specification, not an implementation**: JPA defines the contract for object-relational mapping (ORM), but does not provide the implementation. We need a provider (like Hibernate) to use it.
* **POJO-based entity modeling**: JPA allows us to use Plain Old Java Objects (POJOs) as entities that map to database tables.
* **Annotations and XML mapping**: JPA supports both annotation-based and XML-based mapping of classes to database tables.
* **EntityManager API**: JPA uses the EntityManager to manage database operations like CRUD, queries, and transactions.

### ****Main Features****

* Object-Relational Mapping (ORM)
* JPQL (Java Persistence Query Language)
* Entity lifecycle management
* Transaction management (in conjunction with JTA or Spring)
* Lazy and eager loading
* Caching (first-level)

### ****Common JPA Implementations****

* **Hibernate** (most widely used)
* EclipseLink
* OpenJPA
* DataNucleus

**2. Hibernate**

### ****What is Hibernate?****

Hibernate is an **Object-Relational Mapping (ORM) framework** for Java. It is the **most popular implementation** of the JPA specification, but it also provides features **beyond the JPA standard**.

### ****Key Characteristics of Hibernate****

* **Implements JPA**: Hibernate provides a JPA implementation, so when we use JPA with Hibernate as the provider, Hibernate handles the actual persistence logic.
* **Hibernate-native features**: It also offers additional features not covered by JPA, such as interceptors, batch processing, filters, multi-tenancy, and better caching mechanisms.
* **HQL (Hibernate Query Language)**: In addition to JPQL, Hibernate provides its own query language.
* **Schema generation and validation**: Hibernate can automatically generate or validate schema based on entity classes.

### ****Hibernate as a JPA Provider****

When using Hibernate as a JPA provider, our code can remain JPA-compliant, and Hibernate works behind the scenes to perform the actual operations.

### ****Hibernate-Only Features****

* Support for @Filter, @Formula, @Subselect
* Stateless sessions
* Native SQL queries
* Fine-grained caching (second-level cache, query cache)
* Enhanced fetching strategies

**3. Spring Data JPA**

### ****What is Spring Data JPA?****

**Spring Data JPA** is a part of the **Spring Data** project. It is a **Spring-based abstraction** over JPA that aims to **simplify data access** in Spring applications by reducing boilerplate code.

### ****Key Characteristics of Spring Data JPA****

* **Built on JPA**: Spring Data JPA sits on top of JPA and uses an implementation like Hibernate underneath.
* **Repository Abstraction**: It introduces the concept of repositories—interfaces that automatically generate query methods at runtime.
* **Derived query methods**: We can define query methods using method names (e.g., findByEmail).
* **Custom queries**: Supports JPQL and native SQL queries through annotations like @Query.
* **Integration with Spring ecosystem**: Works seamlessly with Spring Boot, Spring Security, and other modules.

### ****Why Use Spring Data JPA?****

* It removes the need to write boilerplate DAO (Data Access Object) implementations.
* It helps us focus on domain logic rather than database interaction code.
* It supports pagination, sorting, specification queries, and query DSLs.

### ****Core Interfaces****

* CrudRepository<T, ID>
* JpaRepository<T, ID>
* PagingAndSortingRepository<T, ID>

**Key Differences between JPA, Hibernate, and Spring Data JPA**

| **Feature** | **JPA** | **Hibernate** | **Spring Data JPA** |
| --- | --- | --- | --- |
| **Type** | Specification (API) | Framework (Implementation) | Framework (Abstraction) |
| **Purpose** | Standard for ORM in Java | Actual ORM implementation | Simplified data access for Spring |
| **Provides Implementation?** | No | Yes | Yes (uses Hibernate/JPA) |
| **Query Language** | JPQL | JPQL + HQL + native SQL | Derived queries + JPQL + native SQL |
| **Boilerplate Code** | Moderate | Moderate | Minimal |
| **Repository Pattern Support** | No (manual implementation needed) | No (manual or semi-automated) | Yes |
| **Advanced Features** | Only standard features | Many Hibernate-specific features | Relies on JPA implementation for features |
| **Ease of Use** | Medium | Medium | High |