**Deep Skilling - Week 3 Mandatory Hands-on Exercises**

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

**Exercise - 1: Spring Data JPA Example**

**Code:**

**OrmLearnApplication.java**

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

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

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

**import org.slf4j.Logger;**

**import org.slf4j.LoggerFactory;**

**import java.util.List;**

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

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

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

***@SpringBootApplication***

**public class OrmLearnApplication {**

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

**private static CountryService *countryService*;**

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

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

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

***testGetAllCountries*();**

**// SpringApplication.run(OrmLearnApplication.class, args);**

**// LOGGER.info("Inside main");**

**}**

**private static void testGetAllCountries() {**

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

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

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

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

**}**

**}**

**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=Aditi5\*#**

**# Hibernate configuration**

**spring.jpa.hibernate.ddl-auto=validate**

**spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect**

**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="code")**

**private String code;**

***@Column*(name="name")**

**private String name;**

**// getters and setters**

**public void setCode(String code) {**

**this.code = code;**

**}**

**public void setName(String name) {**

**this.name = name;**

**}**

***@Column*(name="co\_name")**

**public String getName() {**

**return name;**

**}**

***@Column*(name="co\_code")**

**public String getCode() {**

**return code;**

**}**

**// toString()**

***@Override***

**public String toString() {**

**return "Country {name=" + name + ", code=" + code + "}";**

**}**

**}**

**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> {**

**}**

**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();**

**}**

**}**

**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-starter</artifactId>**

**</dependency>**

**<dependency>**

**<groupId>com.mysql</groupId>**

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

**<version>8.0.33</version>**

**</dependency>**

**<dependency>**

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

**<artifactId>hibernate-core</artifactId>**

**<version>6.4.4.Final</version> <!-- or use a version compatible with Spring Boot 3.2+ -->**

**</dependency>**

**<dependency>**

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

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

**<scope>runtime</scope>**

**<optional>true</optional>**

**</dependency>**

**<dependency>**

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

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

**<scope>test</scope>**

**</dependency>**

**</dependencies>**

**<build>**

**<plugins>**

**<plugin>**

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

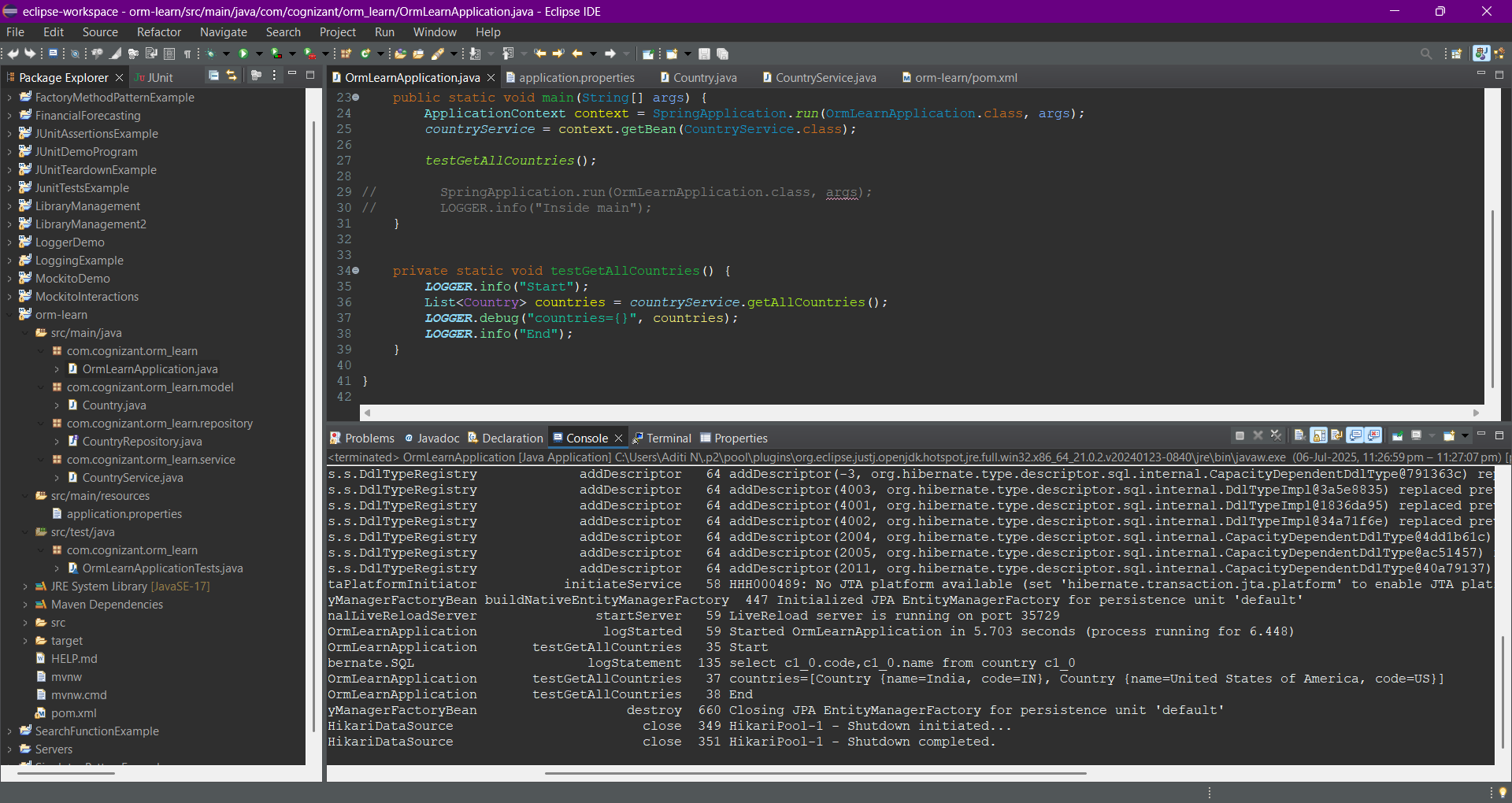
**<artifactId>spring-boot-maven-plugin</artifactId>**

**</plugin>**

**</plugins>**

**</build>**

**</project>**

**Output:  
**

**Exercise - 2: Difference between JPA, Hibernate and Spring Data JPA**

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

* What it is: JPA is a specification for managing relational data in Java applications. Think of it as a set of rules, interfaces, and annotations that define *how* you should interact with a database using Object-Relational Mapping (ORM) principles.
* Key Concept: It's an API (Application Programming Interface), not an implementation. This means JPA itself doesn't contain the actual code that performs database operations. Instead, it provides a standard way for developers to write persistence logic, regardless of the underlying ORM tool.
* Purpose: To provide a standard, portable, and simplified approach to persistence for Java applications. It allows developers to map Java objects to database tables without writing low-level JDBC code.
* Standardization: JPA is part of the Java EE (now Jakarta EE) platform and is defined by a Java Specification Request (JSR), specifically JSR 338 for JPA 2.2.
* Analogy: Imagine JPA as the blueprint for building a house. It defines where the walls go, how many rooms there are, and the general structure, but it doesn't actually build the house.

### 2. Hibernate

* What it is: Hibernate is a popular Object-Relational Mapping (ORM) tool and one of the most widely used implementations of the JPA specification.
* Key Concept: While JPA provides the "what," Hibernate provides the "how." It's the concrete library that takes your JPA-defined entities and operations and translates them into actual SQL queries that interact with your database.
* Features:
  + ORM Capabilities: Maps Java classes to database tables and Java data types to SQL data types.
  + Query Languages: Supports HQL (Hibernate Query Language), a powerful object-oriented query language, and native SQL.
  + Caching: Provides various levels of caching to improve performance.
  + Session Management: Manages the lifecycle of objects and their persistence state.
  + Database Agnostic: Works with a wide range of relational databases.
* Purpose: To handle the complex task of mapping objects to relational tables, managing transactions, and abstracting away the low-level JDBC API, making data persistence easier and more efficient.
* Analogy: If JPA is the blueprint, Hibernate is the construction company that follows the blueprint to build the house.

### 3. Spring Data JPA

* What it is: Spring Data JPA is part of the broader Spring Data project and provides an abstraction layer on top of JPA providers like Hibernate. It's not a new ORM tool itself, but rather a framework that simplifies the development of data access layers using JPA.
* Key Concept: Spring Data JPA significantly reduces the amount of boilerplate code you need to write for common data access operations. It achieves this by providing interfaces and conventions that automatically generate query implementations at runtime.
* How it Works:
  + You define repository interfaces by extending JpaRepository (or other Spring Data interfaces).
  + Spring Data JPA analyzes the method names in your interfaces (e.g., findByLastNameAndFirstName) and automatically generates the corresponding JPA queries (which are then executed by the underlying JPA provider like Hibernate).
  + It also provides ready-to-use methods for common operations like save(), findAll(), findById(), delete(), etc., without you having to implement them.
* Purpose:
  + Reduced Boilerplate Code: Eliminates the need to write repetitive CRUD (Create, Read, Update, Delete) repository implementations.
  + Simplified Query Methods: Allows you to define query methods simply by declaring method signatures in an interface.
  + Integration with Spring Ecosystem: Seamlessly integrates with other Spring features like transaction management, dependency injection, and aspect-oriented programming.
  + Abstraction over JPA Provider: Provides a consistent API regardless of the underlying JPA provider (Hibernate, EclipseLink, etc.).
* Analogy: If Hibernate is the construction company building the house, Spring Data JPA is like a pre-fabrication system that provides ready-made components (like pre-built walls or windows) that the construction company can quickly assemble, making the process faster and less labor-intensive.

Relationship Between Them:

The relationship between these three technologies is hierarchical:

* We write your persistence logic using the JPA specification.
* Hibernate is the ORM tool that implements the JPA specification, translating your JPA concepts into actual database operations.
* Spring Data JPA then sits on top of Hibernate (or any other JPA provider) to simplify our data access layer development by providing conventions and automatically generating common repository implementations, reducing the need for explicit Hibernate API calls for many common tasks.

In essence, we use Spring Data JPA to write less code, which in turn leverages Hibernate to perform the actual persistence operations according to the JPA standard.