**Cognizant Deep Skilling - Digital Nurture 4.0**

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

**Hands on 1**

**Spring Data JPA - Quick Example**

**application.properties**

# 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=root

# Hibernate configuration

spring.jpa.hibernate.ddl-auto=validate

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect

**OrmLearnApplication.java**

package com.cognizant.ormlearn;

import java.util.List;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.boot.autoconfigure.domain.EntityScan;

import org.springframework.context.ApplicationContext;

import com.cognizant.ormlearn.model.Country;

import com.cognizant.ormlearn.service.CountryService;

@SpringBootApplication

@EntityScan(basePackages = "com.cognizant.ormlearn.model")

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*();

}

private static void testGetAllCountries() {

*LOGGER*.info("Start");

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

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

*LOGGER*.info("End");

}

}

**Country.java**

package com.cognizant.ormlearn.model;

import jakarta.persistence.Column;

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

import jakarta.persistence.Table;

@Entity

@Table(name = "country")

public class Country {

@Id

@Column(name = "code")

private String code;

@Column(name = "name")

private String name;

public String getCode() { return code; }

public void setCode(String code) { this.code = code; }

public String getName() { return name; }

public void setName(String name) { this.name = name; }

@Override

public String toString() {

return "Country [code=" + code + ", name=" + name + "]";

}

}

**CountryRepository.java**

package com.cognizant.ormlearn.repository;

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

import org.springframework.stereotype.Repository;

import com.cognizant.ormlearn.model.Country;

@Repository

public interface CountryRepository extends JpaRepository<Country, String> {

}

**CountryService.java**

package com.cognizant.ormlearn.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.ormlearn.model.Country;

import com.cognizant.ormlearn.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>21</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>

</dependencies>

<build>

<plugins>

<plugin>

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

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

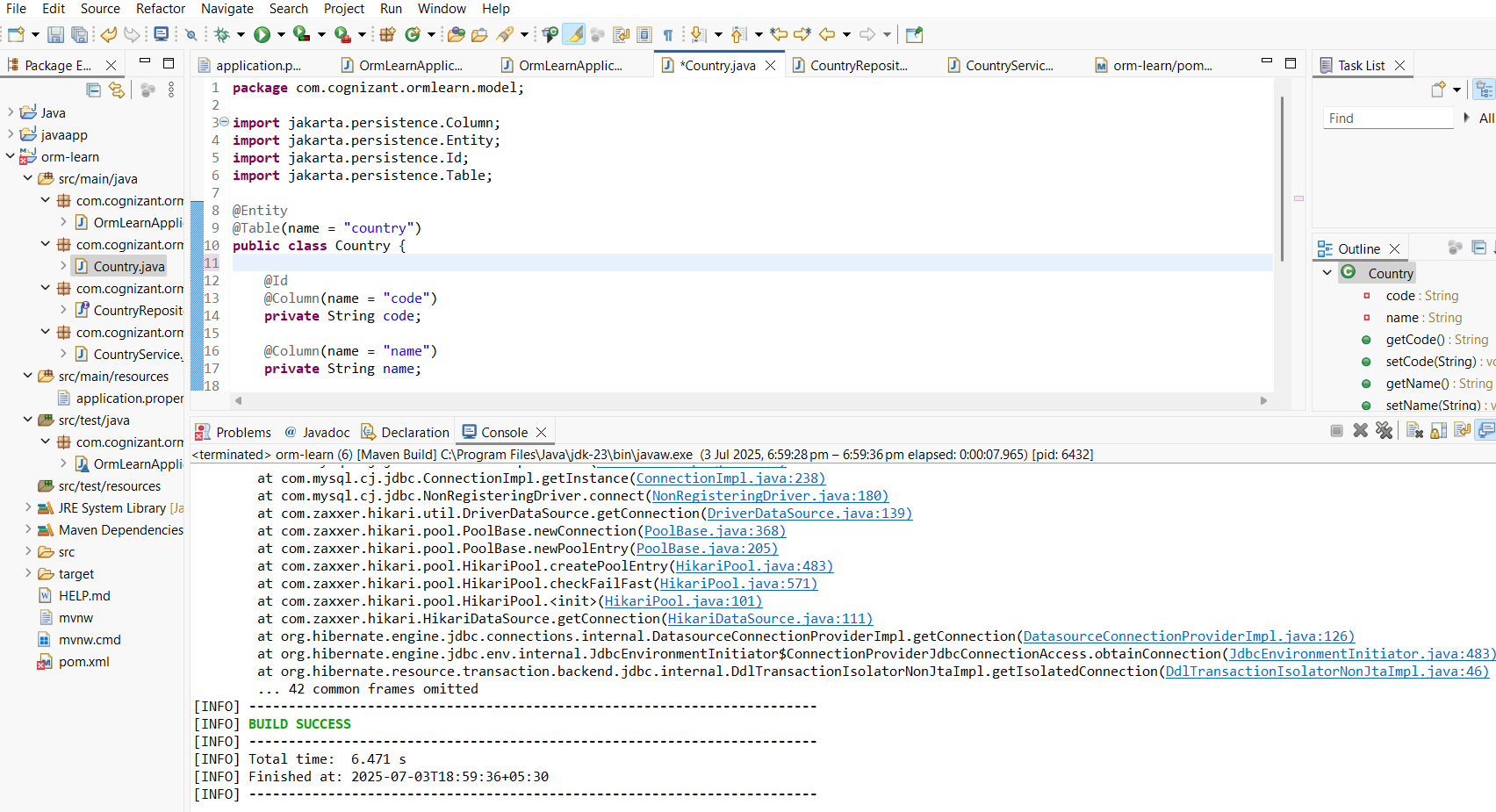
</plugin>

</plugins>

</build>

</project>

**Output:**

****

**Hands on 4**

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

**Java Persistence API (JPA)**

* **What it is:**  
  It is a **Java specification (JSR 338)** for **object-relational mapping (ORM)**.
* **Purpose:**  
  Defines a set of interfaces and annotations to manage relational data in Java applications.
* **Key Point:**  
  JPA itself does **not provide any implementation** — it only defines the standard.
* **Common Implementations:**  
  Hibernate, EclipseLink, OpenJPA, etc.
* **Role:**  
  Defines how to persist, read, update, and delete Java objects (entities) from relational databases.

**Hibernate**

* **What it is:**  
  An **ORM framework** and the **most popular JPA implementation**.
* **Purpose:**  
  Provides the actual implementation of JPA specifications, along with many additional features beyond JPA (e.g., caching, advanced mapping, etc.).
* **Key Point:**  
  It internally implements JPA interfaces and adds extra functionalities.
* **Role:**  
  Acts as a JPA provider (implementation) and can also be used as a standalone ORM.

**Spring Data JPA**

* **What it is:**  
  A **Spring framework module** that provides an additional layer of abstraction over JPA (and its providers like Hibernate).
* **Purpose:**  
  Reduces boilerplate code for data access layers by generating common CRUD operations automatically.
* **Key Features:**
  + Auto-implementation of repository interfaces.
  + Supports derived queries by method names.
  + Handles transactions.
* **Key Point:**  
  It **does not implement JPA** itself, but **requires a JPA provider** like Hibernate underneath.
* **Role:**  
  Simplifies data access in Spring-based apps.

**Summary Comparison Table:**

| **Feature** | **JPA (Java Persistence API)** | **Hibernate** | **Spring Data JPA** |
| --- | --- | --- | --- |
| **Type** | Specification (API) | Framework (JPA Implementation + ORM) | Spring Module (Abstraction over JPA) |
| **Implementation** | No (just APIs & annotations) | Yes (implements JPA + extra ORM features) | No (depends on JPA provider like Hibernate) |
| **Boilerplate Code** | Requires writing DAO layer manually | Less than plain JDBC but still requires DAO implementation | Minimal (Auto-generated repositories) |
| **Transactions** | Depends on Provider | Provided via Hibernate or JPA APIs | Manages automatically via Spring |
| **Advanced Features** | Not applicable | Many (Caching, advanced mappings) | Focused on CRUD + custom queries |
| **Dependency** | JPA provider needed | Can be standalone or used via JPA | Needs a JPA Provider (Hibernate) |

**Code Comparison:**

**Hibernate Example:**

public Integer addEmployee(Employee employee){

Session session = factory.openSession();

Transaction tx = null;

Integer employeeID = null;

try {

tx = session.beginTransaction();

employeeID = (Integer) session.save(employee);

tx.commit();

} catch (HibernateException e) {

if (tx != null) tx.rollback();

e.printStackTrace();

} finally {

session.close();

}

return employeeID;

}

**Spring Data JPA Example:**

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

}

**Key Takeaway:**

* **JPA** → Specification.
* **Hibernate** → Implementation of JPA (with more features).
* **Spring Data JPA** → Abstraction over JPA + Hibernate to **simplify data access** in Spring apps.