WEEK 3: Spring Data JPA with Spring Boot, Hibernate

1. **Spring Data JPA - Quick Example**

**com.cognizant.orm-learn.model.Country(Persistence Class)**

package com.cognizant.ormlearn.model;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.Table;

@Entity

@Table(name = "country")

public class Country {

@Id

@Column(name = "co\_code")

private String code;

@Column(name = "co\_name")

private String name;

public Country() {

}

public Country(String code, String name) {

this.code = code;

this.name = 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 + "'}";

}

}

**com.cognizant.orm-learn.CountryRepository(Repository Class)**

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> {

// Inherits findAll(), findById(), save(), delete() etc.

}

**com.cognizant.orm-learn.service.CountryService (Service Class)**

package com.cognizant.ormlearn.service;

import java.util.List;

import javax.transaction.Transactional;

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

import org.springframework.stereotype.Service;

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();

}

}

**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.context.ApplicationContext;

import com.cognizant.ormlearn.model.Country;

import com.cognizant.ormlearn.service.CountryService;

@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);

LOGGER.info("Inside main");

countryService = context.getBean(CountryService.class);

testGetAllCountries();

}

private static void testGetAllCountries() {

LOGGER.info("Start testGetAllCountries");

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

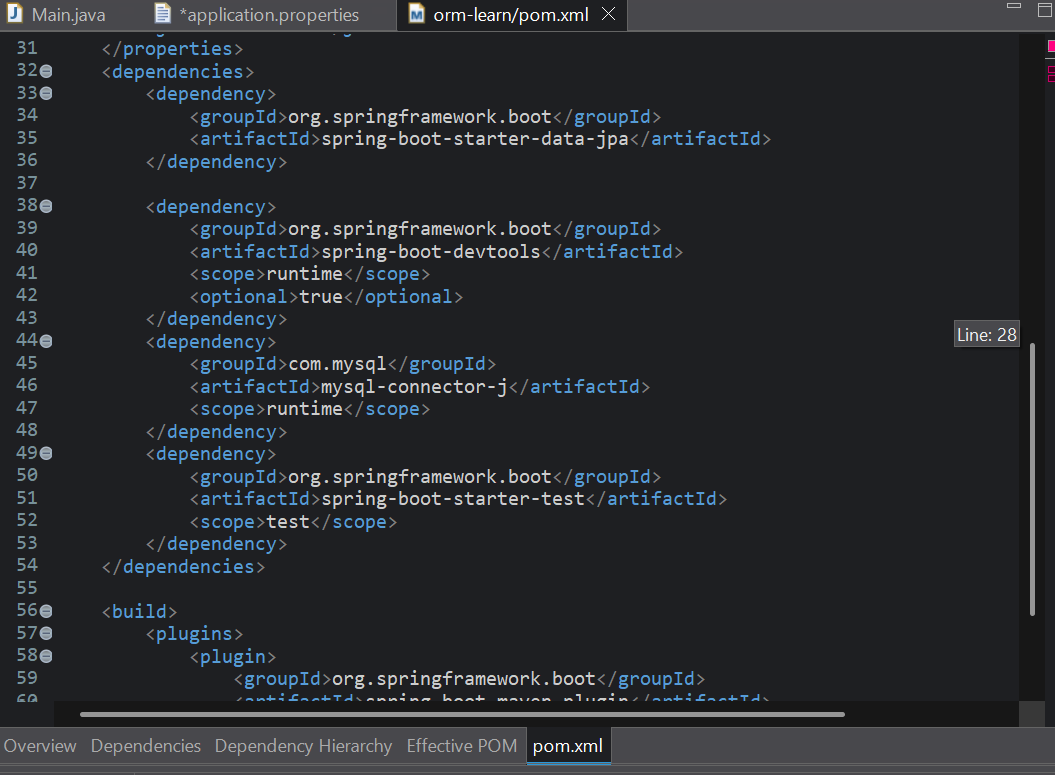
LOGGER.debug("countries={}", countries);

LOGGER.info("End testGetAllCountries");

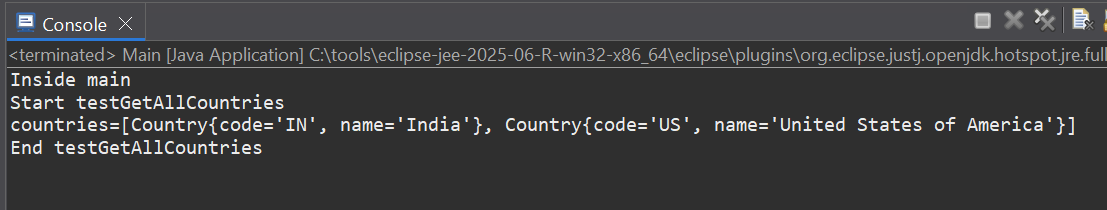
}

}

pom.xml



**OUTPUT :**



1. Difference between JPA, Hibernate and Spring Data JPA

**Java Persistence API (JPA)**

The Java Persistence API provides a specification for persisting, reading, and managing data from your Java object to relational tables in the database. JPA provides a unified approach to object-relational mapping (ORM), making it easier to handle database interactions within Java applications.

* Type: Specification (JSR 338)
* Purpose: Defines a standard for persisting Java objects to relational databases.
* Does not contain implementation.
* Requires a provider (like Hibernate) to work.
* Uses annotations like @Entity, @Id, @Table, etc.

**Hibernate**

Hibernate is an object-relational mapping solution for Java environments. Object-relational mapping or ORM is the programming technique to map application domain model objects to the relational database tables.

It provides a reference implementation of the Java Persistence API helping in loose coupling

* Type: JPA Implementation and Standalone ORM Tool
* Implements JPA, but also offers extra features beyond JPA.
* Requires manual handling of:
  + Sessions
  + Transactions
  + Exception handling
* One of the most popular JPA providers.

**Spring Data JPA**

It reduces the amount of boilerplate code required to implement data access layers for various persistence stores.It is a library/framework that adds an extra layer of abstraction on the top of our JPA provider.

* Type: Spring abstraction over JPA
* Does not implement JPA, but sits on top of JPA implementations (like Hibernate).
* It Handles:
  + Session management
  + Transaction management
  + Query method generation
  + Integration with Spring Boot