**JAVA FSE WEEK 3**

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

**Mandatory Hands-on Excercises**

**Hands on 1 : Spring Data JPA - Quick Example :**

**Step 1: MySQL Setup**

create schema ormlearn;

use ormlearn;

create table country (

code varchar(2) primary key,

name varchar(50)

);

insert into country values ('IN', 'India');

insert into country values ('US', 'United States of America');

**Step 2: application.properties**

application.properties:

# Logging Config

logging.level.org.springframework=info

logging.level.com.cognizant=debug

logging.level.org.hibernate.SQL=trace

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

# Console log format

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

# Database config

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 config

spring.jpa.hibernate.ddl-auto=validate

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

**Step 3: Entity Class – Country**

Country.java:

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 = "code")

private String code;

@Column(name = "name")

private String name;

// Getters and Setters

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 + "]";

}

}

**Step 4: Repository Interface – CountryRepository**

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> {

}

**Step 5: Service Class – CountryService**

CountryService.java:

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();

}

}

**Step 6: Main Application – OrmLearnApplication**

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 CountryService countryService;

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

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");

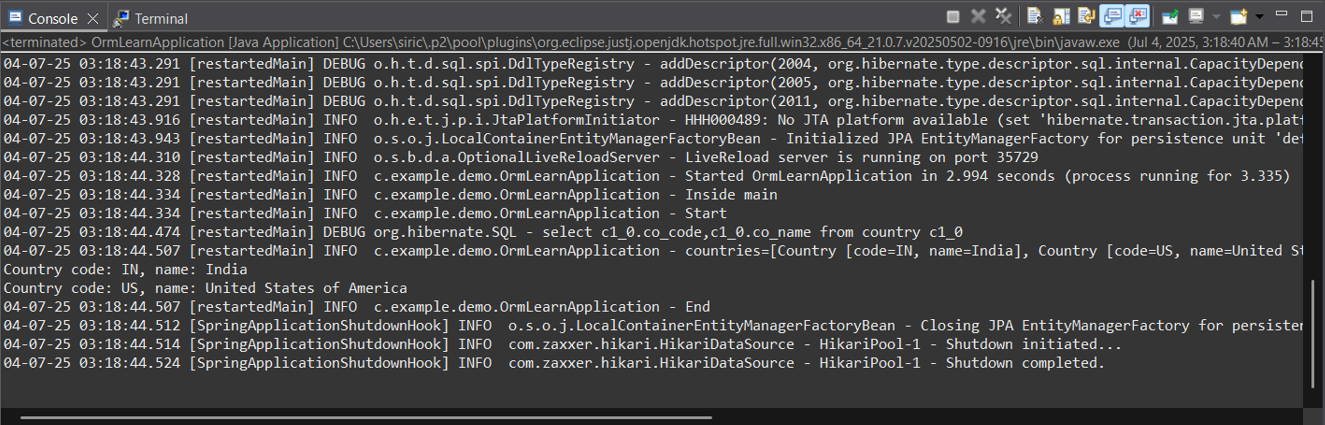
List<Country> countries = countryService.getAllCountries();

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

LOGGER.info("End");

}

}

**OUTPUT:**  


**Hands on 4: Difference between JPA, Hibernate and Spring Data JPA**

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

* JPA is a specification (JSR 338) for accessing, persisting, and managing data between Java objects and relational databases.
* It provides an abstraction layer over ORM implementations like Hibernate.
* JPA itself does not contain any implementation code**.**

**2. Hibernate**

* Hibernate is a popular ORM (Object Relational Mapping) tool and a concrete implementation of JPA.
* It handles CRUD operations and maps Java classes to database tables.

**Hibernate code example:**

// Method to CREATE an employee in the database

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;

}

**3. Spring Data JPA**

* Spring Data JPA is a part of the Spring ecosystem and provides an abstraction over JPA/Hibernate.
* It removes boilerplate code by allowing you to use repository interfaces.
* Automatically provides implementation at runtime, such as save(), findAll(), etc.

**Spring Data JPA Code Example**

**EmployeeRepository.java:**

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

**EmployeeService.java:**

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

**Summary**

* JPA is only a specification; it needs an implementation like Hibernate.
* Hibernate is a complete ORM framework that implements JPA and does the actual database operations.
* Spring Data JPA simplifies development by removing the need to write repository code. It builds on top of JPA and uses Hibernate under the hood.