Cognizant - DN 4.0 Deep Skilling Java FSE

Week 03 – Spring data JPA

Superset ID: 6383725

Name: G Ashritha

**Exercise 1: Spring Data JPA – Quick Example**

//MODEL

package com.example.country.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 + "]";

}

}

//REPOSITORY  
  
package com.example.country.repository;

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

import org.springframework.stereotype.Repository;

import com.example.country.model.country;

@Repository

public interface countryRepo extends JpaRepository<country, String> {

}

//SERVICE

package com.example.country.service;

import java.util.List;

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

import org.springframework.stereotype.Service;

import com.example.country.model.country;

import com.example.country.repository.countryRepo;

import jakarta.transaction.Transactional;

@Service

public class countryService {

@Autowired

private countryRepo countryRepository;

@Transactional

public List<country> getAllCountries() {

return countryRepository.findAll();

}

}

//MAIN CLASS:  
  
package com.example.country;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import com.example.country.model.country;

import com.example.country.service.countryService;

import java.util.List;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.context.ApplicationContext;

@SpringBootApplication

public class CountryApplication {

private static countryService countryService;

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

public static void main(String[] args) {

ApplicationContext context = SpringApplication.run(CountryApplication.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");

}

}

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

**Java Persistence API (JPA)**

| **Aspect** | **Description** |
| --- | --- |
| 🔸 **What it is** | A **specification** (JSR 338) for managing relational data in Java applications. |
| 🔸 **Type** | Only **defines interfaces and rules** – no actual code or implementation. |
| 🔸 **Key Features** | Annotations (@Entity, @Id, @OneToMany, etc.), EntityManager, JPQL (Java Persistence Query Language). |
| 🔸 **Example Providers** | Hibernate, EclipseLink, OpenJPA, etc. implement the JPA specification. |

**🔷 2. Hibernate**

| **Aspect** | **Description** |
| --- | --- |
| 🔸 **What it is** | A **concrete implementation** of the JPA specification. |
| 🔸 **Type** | ORM **framework and JPA provider**. |
| 🔸 **Key Features** | Supports both JPA and its own native APIs (Session, Query, HQL). |
| 🔸 **Extra Features** | Lazy loading, caching, custom dialects, batch processing, etc. |

**🔷 3. Spring Data JPA**

| **Aspect** | **Description** |
| --- | --- |
| 🔸 **What it is** | A part of **Spring Data** that provides **abstraction** over JPA (e.g., Hibernate). |
| 🔸 **Type** | **Helper library** that uses JPA provider (like Hibernate) underneath. |
| 🔸 **Key Benefits** |  |

* Removes boilerplate code
* Auto-generates queries (findByName, etc.)
* Integrates seamlessly with Spring Boot
* Supports CrudRepository, JpaRepository, and more  
  | 🔸 **Transaction Management** | Spring handles transactions behind the scenes with @Transactional |