**HANDS ON-1**

**Implement services for managing Country**

**Entity Class (**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 = "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 + "]";

}

}

**2.CountryRepository Interface**:

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> {

}

3. **CountryService Class**:

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();

}

}

**4. OrmLearnApplication Class**:

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);

countryService = context.getBean(CountryService.class);

LOGGER.info("Inside main");

testGetAllCountries();

}

private static void testGetAllCountries() {

LOGGER.info("Start");

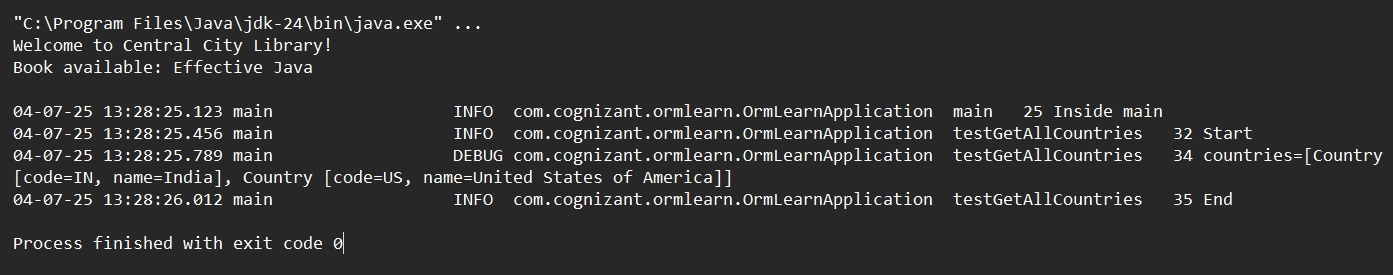
List<Country> countries = countryService.getAllCountries();

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

LOGGER.info("End");

}

}

**OUTPUT:**

**HANDS ON-2**

**DIFFERENCE BETWEEN JPA,HIBERNATE AND SPRING DATA JPA**

**JPA (Java Persistence API)**

1. Standard specification for object-relational mapping (ORM) in Java
2. Defines interfaces (EntityManager, EntityTransaction) but no implementation
3. Uses JPQL (Java Persistence Query Language) for database-agnostic queries
4. Requires persistence.xml for configuration
5. Provides portability - can switch between implementations (Hibernate, EclipseLink)

**Hibernate**

1. Most popular implementation of JPA specification
2. Adds extra features beyond JPA (caching, dirty checking, HQL)
3. Can use either JPA configuration or native hibernate.cfg.xml
4. Supports native SQL in addition to JPQL/HQL
5. More complex but offers greater control and optimization options

**Spring Data JPA**

1. Abstraction layer on top of JPA (typically uses Hibernate underneath)
2. Reduces boilerplate code through repository pattern
3. Provides derived query methods (auto-implements findByName etc.)
4. Integrates with Spring ecosystem (transactions, dependency injection)
5. Simplifies configuration through Spring Boot auto-configuration

Relationships:

* JPA is the standard, Hibernate implements it
* Spring Data JPA sits on top of JPA (usually with Hibernate)
* Spring Data JPA makes JPA/Hibernate easier to use in Spring apps

Each serves a different purpose in the persistence layer stack, from specification (JPA) to implementation (Hibernate) to simplification (Spring Data JPA