**Hands on 1**

**Spring Data JPA - Quick Example** 

spring.application.name=orm-learn

logging.level.org.springframework=info

logging.level.com.cognizant=debug

logging.level.org.hibernate.SQL=trace

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

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

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

spring.jpa.hibernate.ddl-auto=validate

spring.jpa.database-platform=org.hibernate.dialect.MySQL8Dialect

create table country(

co\_code varchar(2) primary key,

co\_name varchar(50)

);

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

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

package com.cognizant.orm\_learn.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 = "co\_code")

private String code;

@Column(name = "co\_name")

private String name;

// Getter and Setter for 'code'

public String getCode() {

return code;

}

public void setCode(String code) {

this.code = code;

}

// Getter and Setter for 'name'

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

// toString() method

@Override

public String toString() {

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

}

}

package com.cognizant.orm\_learn.repository;

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

import org.springframework.stereotype.Repository;

import com.cognizant.orm\_learn.model.Country;

@Repository

public interface CountryRepository extends JpaRepository<Country, String> {

}

package com.cognizant.orm\_learn.service;

import java.util.List;

import jakarta.transaction.Transactional;

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

import org.springframework.stereotype.Service;

import com.cognizant.orm\_learn.model.Country;

import com.cognizant.orm\_learn.repository.CountryRepository;

@Service

public class CountryService {

@Autowired

private CountryRepository countryRepository;

@Transactional

public List<Country> getAllCountries() {

return countryRepository.findAll();

}

}

package com.cognizant.orm\_learn;

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;

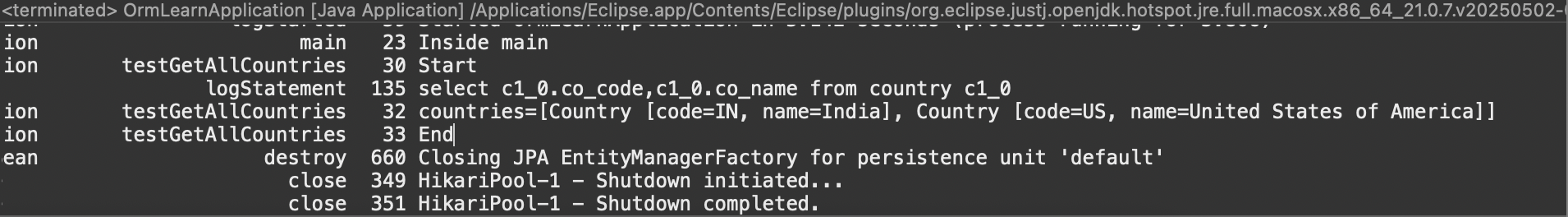
import com.cognizant.orm\_learn.model.Country;

import com.cognizant.orm\_learn.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");

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

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

LOGGER.info("End");

}

}

**Hands on 4**

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

1. Java Persistence API (JPA) — The Specification

* What it is:  
  JPA is just a set of rules (interfaces & annotations) for mapping Java objects to database tables (ORM: Object Relational Mapping).
* It doesn’t do anything on its own. It needs an implementation.

Example:

@Entity

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Integer id;

private String name;

private String department;

}

2. Hibernate — The Implementation

* What it is:  
  Hibernate is an actual ORM tool that implements the JPA specification (and adds extra features).
* Without Spring, you have to manually manage:
  + Session
  + Transactions
  + SessionFactory

Example with plain Hibernate (without Spring):

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 (Exception e) {

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

} finally {

session.close();

}

return employeeId;

}

Problems with this:

* A lot of boilerplate code (session, transaction, error handling).
* Hard to maintain.

3. Spring Data JPA — The Abstraction & Simplicity

* What it is:  
  Spring Data JPA is not an ORM tool itself.  
  It is a framework that makes JPA (and Hibernate) much easier to use by removing boilerplate code.
* It automatically manages:
  + Repositories
  + Sessions
  + Transactions

Steps:

Step 1 : Entity

@Entity

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Integer id;

private String name;

private String department;

}

Step 2 : Repository Interface

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

Step 3 : Service

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

}

Benefits:

* No manual session or transaction management.
* Auto-generated CRUD methods: save(), findById(), findAll(), delete(), etc.

|  |  |  |  |
| --- | --- | --- | --- |
| Feature | JPA | Hibernate | Spring Data JPA |
| What it is | Specification only (no code) | ORM Implementation of JPA | Abstraction layer over JPA/Hibernate |
| Manages transactions? | No | You do it manually | Yes (automated) |
| Handles sessions? | No | You do it manually | Yes (automated) |
| CRUD methods? | No | Write them yourself | Built-in via JpaRepository |