**Spring Data JPA - Quick Example**

src/main/resources/application.properties

# Spring and app log

logging.level.org.springframework=info

logging.level.com.cognizant=debug

# Hibernate SQL

logging.level.org.hibernate.SQL=trace

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

# Log pattern

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

# Database

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

spring.jpa.hibernate.ddl-auto=validate

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

src/main/java/com/cognizant/ormlearn/OrmLearnApplication.java

package com.cognizant.ormlearn;

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.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");

System.out.println(countryService.getAllCountries());

LOGGER.info("End");

}

}

src/main/java/com/cognizant/ormlearn/model/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 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 + "]";

}

}

src/main/java/com/cognizant/ormlearn/repository/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> {}

src/main/java/com/cognizant/ormlearn/service/CountryService.java

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;

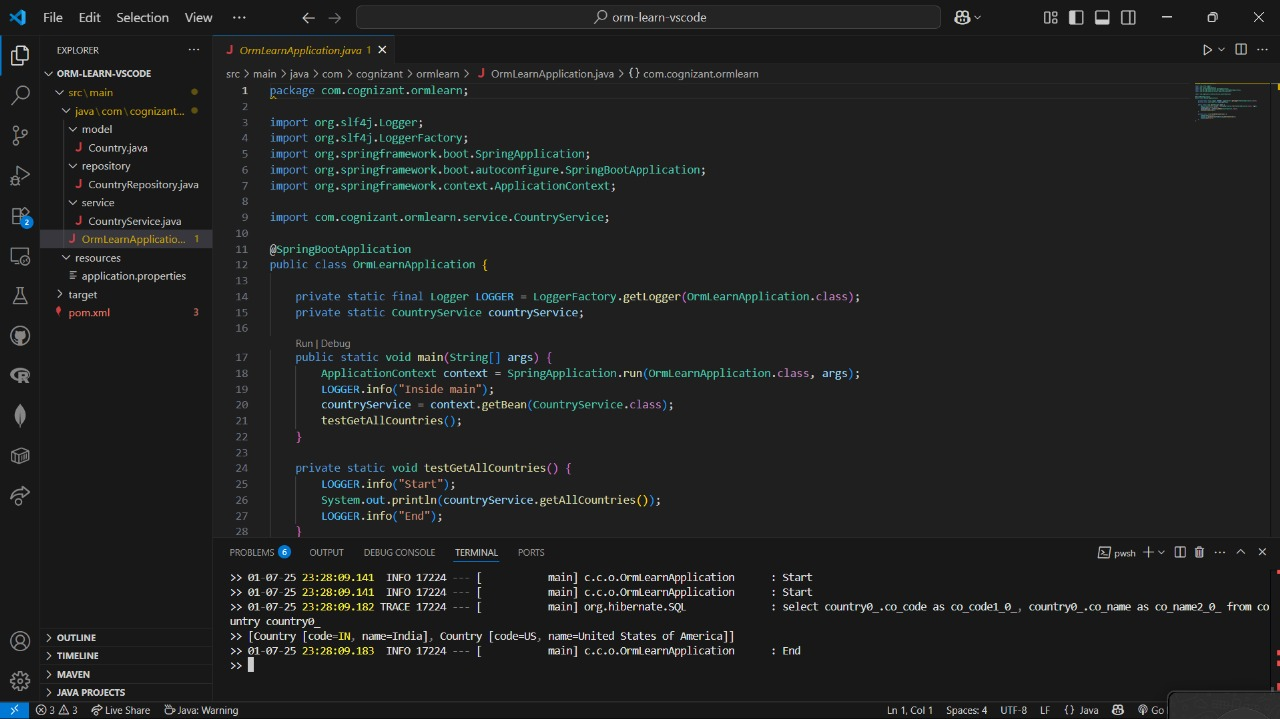
@Transactional

public List<Country> getAllCountries() {

return countryRepository.findAll();

}

}



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

JPA (Java Persistence API)

What is it?

JPA is a specification (JSR 338) for persisting Java objects to a relational database.

Key Points:

It’s just an interface/standard; it does not provide any implementation.

It defines annotations, configuration, and APIs for object-relational mapping (ORM).

Developers use JPA to write code that’s independent of the underlying ORM tool.

Example providers: Hibernate, EclipseLink, OpenJPA.

Hibernate

What is it?

Hibernate is a popular ORM tool and a concrete implementation of JPA.

Key Points:

It provides the actual logic and code to interact with the database, using JPA APIs or its own APIs.

Offers advanced features like lazy loading, caching, and custom queries.

When you use Hibernate, you can choose to use JPA annotations/APIs for portability or Hibernate-specific ones for extra features.

Spring Data JPA

What is it?

Spring Data JPA is a Spring project that builds on top of JPA implementations (like Hibernate) to make data access simpler.

Key Points:

It does not implement JPA itself; it relies on an implementation (typically Hibernate).

Reduces boilerplate code using powerful repository abstractions.

Offers CRUD methods, pagination, query derivation by method names, and more—out of the box.

Manages transactions and integrates seamlessly with the Spring ecosystem.

You mostly focus on interfaces and method signatures, not boilerplate code.

Code Comparison

Hibernate Approach (Manual):

java

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;

}

You manually manage sessions and transactions.

More verbose and error-prone.

Spring Data JPA Approach (Declarative):

java

public interface EmployeeRepository extends JpaRepository<Employee, Integer> { }

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

}

No session or transaction code required in business logic.

Spring manages everything; you just call repository methods.

Much less boilerplate, easier to test and maintain.