**Hands on 1**

**Spring Data JPA - Quick Example**

**application.properties.xml**

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

# Database connection

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=toor

# Hibernate config

spring.jpa.hibernate.ddl-auto=validate

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

**Country.java**

package com.cognizant.ormlearn.model;

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

import jakarta.persistence.Table;

@Entity

@Table(name = "country")

public class Country {

@Id

private String code;

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 + "]";

}

}

**CountryRepository.java**

package com.cognizant.ormlearn.repository;

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

import com.cognizant.ormlearn.model.Country;

public interface CountryRepository extends JpaRepository<Country, String> {

}

**CountryService.java**

package com.cognizant.ormlearn.service;

import java.util.List;

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;

public List<Country> getAllCountries() {

return countryRepository.findAll();

}

}

**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);

countryService = context.getBean(CountryService.class);

testGetAllCountries();

}

private static void testGetAllCountries() {

LOGGER.info("Start");

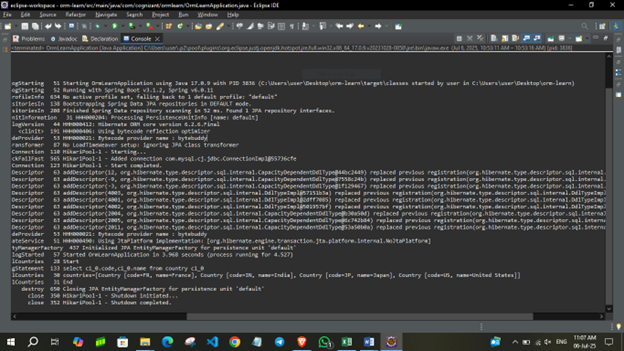
List<Country> countries = countryService.getAllCountries();

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

LOGGER.info("End");

}

}

**OUTPUT:** 

**Hands on 4**

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

**Hibernate**

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;

   }

**Spring Data JPA**  
**EmployeeRespository.java**

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

**EmployeeService.java**

@Autowire

  private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

  employeeRepository.save(employee);

  }

**Difference :**

* Spring Data JPA handles session and transaction management automatically, unlike Hibernate which requires manual coding for these operations.
* CRUD operations are simplified in Spring Data JPA through built-in repository interfaces, reducing the need for repetitive boilerplate code seen in Hibernate.
* Development with Spring Data JPA is faster and cleaner due to the use of annotations and minimal configuration, whereas Hibernate needs more explicit setup.
* Custom queries in Spring Data JPA can be created using method naming conventions, while Hibernate typically requires writing HQL or SQL manually.
* Spring Data JPA integrates tightly with Spring features like dependency injection and REST controllers, making full-stack development smoother than with standalone Hibernate.
* Transactions in Spring Data JPA are managed declaratively, whereas Hibernate demands explicit handling of transaction boundaries.
* Pagination and sorting are built into Spring Data JPA using Pageable and Sort, but with Hibernate, developers must write custom code for these features.
* Testing is more straightforward with Spring Data JPA due to Spring Boot’s testing tools and in-memory database support, unlike the more complex setup needed with Hibernate.