**Hands on 1**

**Spring Data JPA - Quick Example** 

**Country.java**  
  
package com.cognizant.springjpaormlearn.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;

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.springjpaormlearn.repository;

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

import org.springframework.stereotype.Repository;

import com.cognizant.springjpaormlearn.model.Country;

@Repository

public interface CountryRepository extends JpaRepository<Country, String> {

}

[**CountryService.java**](https://github.com/Deepa2794/Digital-Nurture-4.0---6416882/blob/main/Week%203/Week%203%20-%20Spring%20Data%20JPA%20with%20Spring%20Boot%2C%20Hibernate/Spring%20Data%20JPA%20-%20Quick%20Example/CountryService.java)

package com.cognizant.springjpaormlearn.service;

import java.util.List;

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

import org.springframework.stereotype.Service;

import jakarta.transaction.Transactional;

import com.cognizant.springjpaormlearn.model.Country;

import com.cognizant.springjpaormlearn.repository.CountryRepository;

@Service

public class CountryService {

@Autowired

private CountryRepository countryRepository;

@Transactional

public List<Country> getAllCountries() {

return countryRepository.findAll();

}

}

[**SpringjpaormlearnApplication.java**](https://github.com/Deepa2794/Digital-Nurture-4.0---6416882/blob/main/Week%203/Week%203%20-%20Spring%20Data%20JPA%20with%20Spring%20Boot%2C%20Hibernate/Spring%20Data%20JPA%20-%20Quick%20Example/SpringjpaormlearnApplication.java)

package com.cognizant.springjpaormlearn;

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.springjpaormlearn.model.Country;

import com.cognizant.springjpaormlearn.service.CountryService;

@SpringBootApplication

public class SpringjpaormlearnApplication {

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

private static CountryService countryService;

public static void main(String[] args) {

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

}

**}**

[**SpringjpaormlearnApplicationTests.java**](https://github.com/Deepa2794/Digital-Nurture-4.0---6416882/blob/main/Week%203/Week%203%20-%20Spring%20Data%20JPA%20with%20Spring%20Boot%2C%20Hibernate/Spring%20Data%20JPA%20-%20Quick%20Example/SpringjpaormlearnApplicationTests.java)

package com.cognizant.springjpaormlearn;

import org.junit.jupiter.api.Test;

import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest

class SpringjpaormlearnApplicationTests {

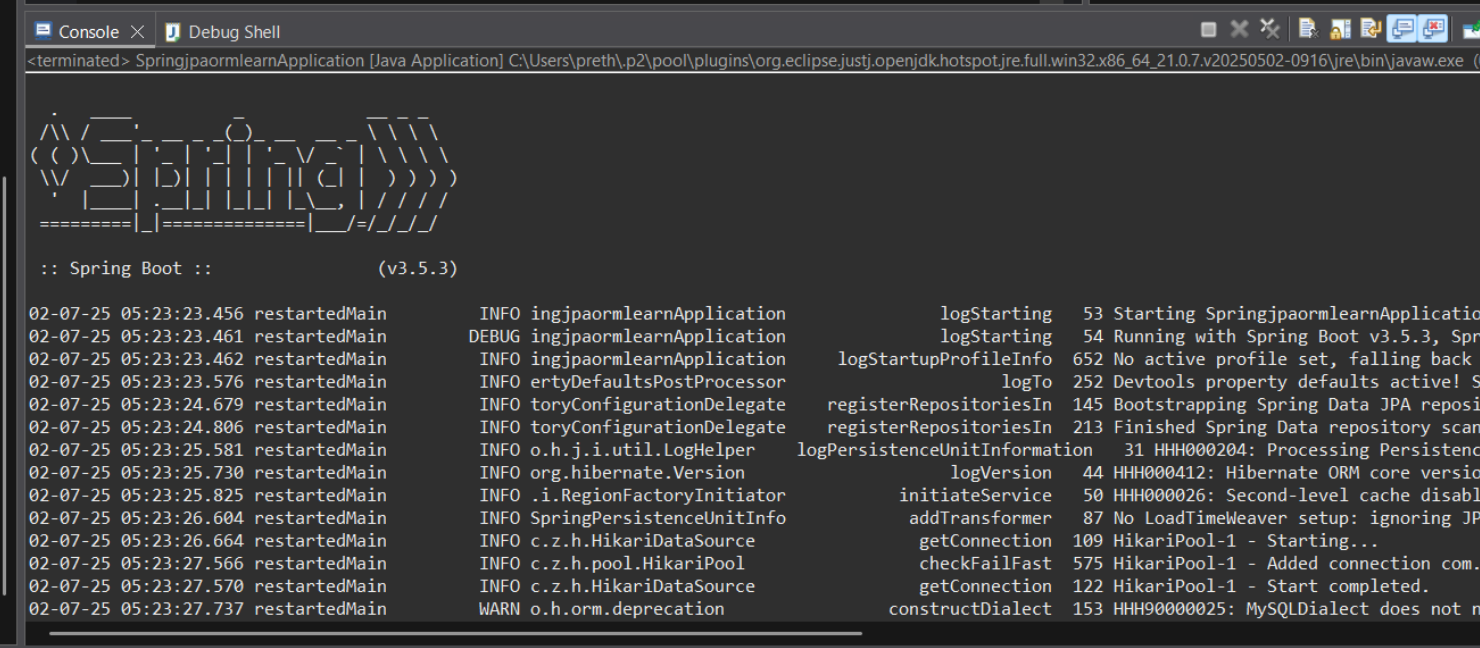
@Test

void contextLoads() {

}

}

**Output:**

****

**Hands on 4**

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

**Java Persistence API (JPA)**

* JPA is a specification (JSR 338) for persisting, reading, and managing data between Java objects and relational databases.
* It does not provide a concrete implementation.
* Requires a JPA provider like Hibernate or EclipseLink.
* Acts as a standard API for Object-Relational Mapping (ORM) in Java.

**Example Syntax:**

@Entity

@Table(name = "employee")

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int id;

@Column(name = "name")

private String name;

}

**Hibernate**

* Hibernate is a popular implementation of JPA.
* It is an ORM tool that maps Java objects to database tables.
* Handles the low-level JDBC operations like session and transaction management.
* Offers additional features like caching, lazy loading, and fetching strategies.

**Example Syntax:**

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**

* It is not a JPA provider, but a higher-level abstraction over JPA (usually with Hibernate underneath).
* Reduces boilerplate code with built-in repository interfaces.
* Uses Spring Framework’s features like dependency injection and transaction management.
* Helps developers focus on business logic by automatically generating CRUD methods.

**Example Syntax:**

EmployeeRepository.java

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

EmployeeService.java

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

**Comparison:**

| Feature | JPA | Hibernate | Spring Data JPA |
| --- | --- | --- | --- |
| Type | Specification (JSR 338) | JPA Implementation (ORM Tool) | Abstraction over JPA |
| Implementation | No | Yes | Uses existing JPA provider |
| Boilerplate Code | Moderate | High | Low |
| Transaction Handling | Manual or via frameworks | Manual | Automatically handled by Spring |
| CRUD Operations | Manually coded | Manually coded | Auto-generated via Repository |

**Analysis:**

From the comparison, it is evident that the three technologies serve different purposes within the Java persistence ecosystem. JPA offers a standard interface, Hibernate provides a powerful implementation, and Spring Data JPA enhances productivity by reducing boilerplate and streamlining data access. The progression from JPA to Spring Data JPA reflects a shift towards abstraction and simplicity in modern application development. Choosing the right approach depends on the project's complexity, need for customization, and the development team's familiarity with Spring and ORM concepts.