**Spring Data JPA with Spring Boot, Hibernate**

**Spring Data JPA - Quick Example**

**//Country.java**

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;  
  
 // getters and setters   
  
 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;  
 }  
  
 // toString()  
 @Override  
 public String toString() {  
 return "Country{" + "code=" + code + ", name=" + name + '}';  
 }  
  
}

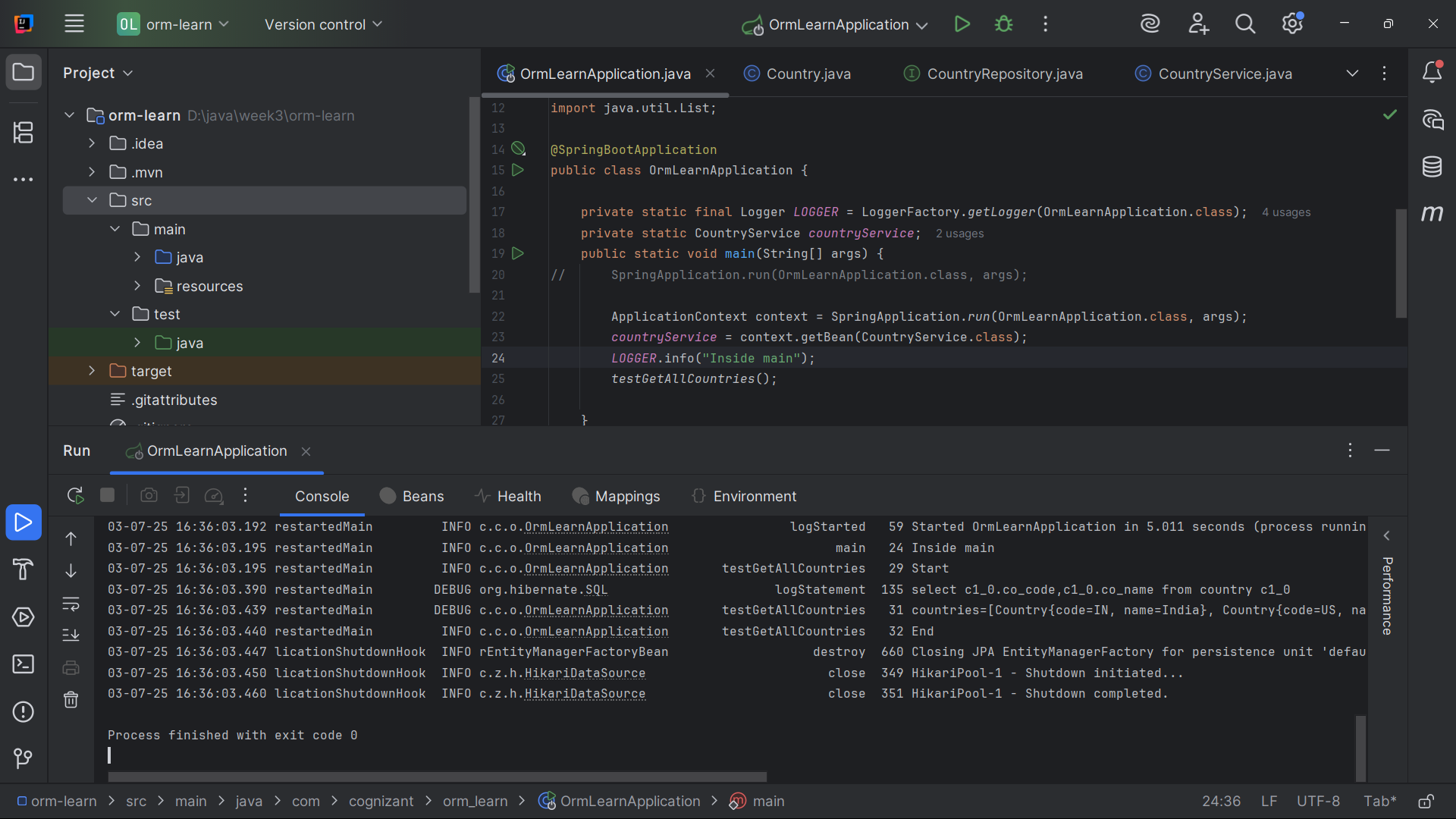
**//CountryService.java**

@Service  
public class CountryService {  
 @Autowired  
 private CountryRepository countryRepository;  
 @Transactional  
 public List<Country> getAllCountries(){  
 return countryRepository.findAll();  
 }  
}

**//OrmLearnApplication.java (Main class)**

package com.cognizant.orm\_learn;  
  
import com.cognizant.orm\_learn.model.Country;  
import com.cognizant.orm\_learn.service.CountryService;  
import com.cognizant.orm\_learn.service.exception.CountryNotFoundException;  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.context.ApplicationContext;  
  
import java.util.List;  
  
@SpringBootApplication  
public class OrmLearnApplication {  
  
 private static final Logger *LOGGER* = LoggerFactory.*getLogger*(OrmLearnApplication.class);  
 private static CountryService *countryService*;  
 public static void main(String[] args) {  
// SpringApplication.run(OrmLearnApplication.class, 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**



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

|  |  |  |
| --- | --- | --- |
| **JPA** | **HIBERNATE** | **SPRING DATA JPA** |
| JPA (Java Persistence API, now known as Jakarta Persistence API) is a specification that defines a set of interfaces and annotations for working with relational databases in Java applications. | Hibernate is an Object-Relational Mapping (ORM) framework that provides a high-level API for interacting with relational databases. | Spring Data JPA is a part of the Spring Framework that provides a higher-level, easier-to-use API for working with JPA. |
| JPA provides a common API for ORM frameworks like Hibernate, EclipseLink, and OpenJPA, making it easier to switch between different ORM frameworks without changing your code. | It allows you to map Java objects to database tables and perform database operations using a high-level API, instead of writing low-level SQL code. | It reduces the amount of boilerplate code required to interact with a database using JPA, and provides a repository abstraction that provides a number of methods out of the box for performing CRUD operations on JPA entities. |
| JPA is not an implementation. It is only a Java specification. | Hibernate is a JPA implementation. | Spring Data JPA is a JPA Data Access Abstraction. |
| JPA is described in javax.persistence package (For new version it is jakarta.persistence) | Hibernate is described in org.hibernate package. | Spring Data JPA is described in the org.springframework.data.jpa package |
| It is provided by Java EE / Jakarta EE | It is provided by Red Hat (Open Source) | It is provided by Spring Framework |
| CRUD OPERATIONS are written manually. | CRUD OPERATIONS are written manually. | Auto-generated by Spring from interface method names |
| Transaction Handling are done  Manually via EntityTransaction | Transaction Handling are done Manually or with Spring. | Transaction Handling are automatically handled using @Transactional |
| It is very flexible but verbose. | It is most flexible as it has full control over ORM features. | Limited to JPA abstraction; complex logic may still need custom implementation. |
| Entity Management is done Manually (via EntityManager). | Entity Management is done Manually (via Session) | Automatic with CrudRepository / JpaRepository |

**Refer following code to see difference**

**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;  
}

//here

//a transaction is a sequence of database operations treated as a single unit of work

//a Session represents a single unit of work with the database

**Spring Data JPA**

//EmployeeRespository.java

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

//EmployeeService.java

@Service

public class EmployeeService {  
@Autowired  
private EmployeeRepository employeeRepository;  
@Transactional  
public void addEmployee(Employee employee) {  
 employeeRepository.save(employee);  
}}

//here

//@Autowired is used to automatically inject dependencies into Spring-managed beans.

//@Transactional is used to manage database transactions