**SPRING-DATA-JPA-HANDSON**

**Hands on 1**

**Spring Data JPA - Quick Example**

**application.properties**

# Logging

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

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  
  
**Country.java**  
package com.cognizant.orm\_learn.model;

import jakarta.persistence.\*;

@Entity

@Table(name = "country")

public class Country {

@Id

@Column(name = "code")

private String code;

@Column(name = "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.orm\_learn.repository;

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

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

import org.springframework.stereotype.Repository;

@Repository

public interface CountryRepository extends JpaRepository<Country, String> {

}  
  
**CountryService.java**  
package com.cognizant.orm\_learn.service;

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

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

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

import org.springframework.stereotype.Service;

import jakarta.transaction.Transactional;

import java.util.List;

@Service

public class CountryService {

@Autowired

private CountryRepository countryRepository;

@Transactional

public List<Country> getAllCountries() {

return countryRepository.findAll();

}

}

**OrmLearnApplication.java**

package com.cognizant.orm\_learn;

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

import com.cognizant.orm\_learn.service.CountryService;

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 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) {

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();

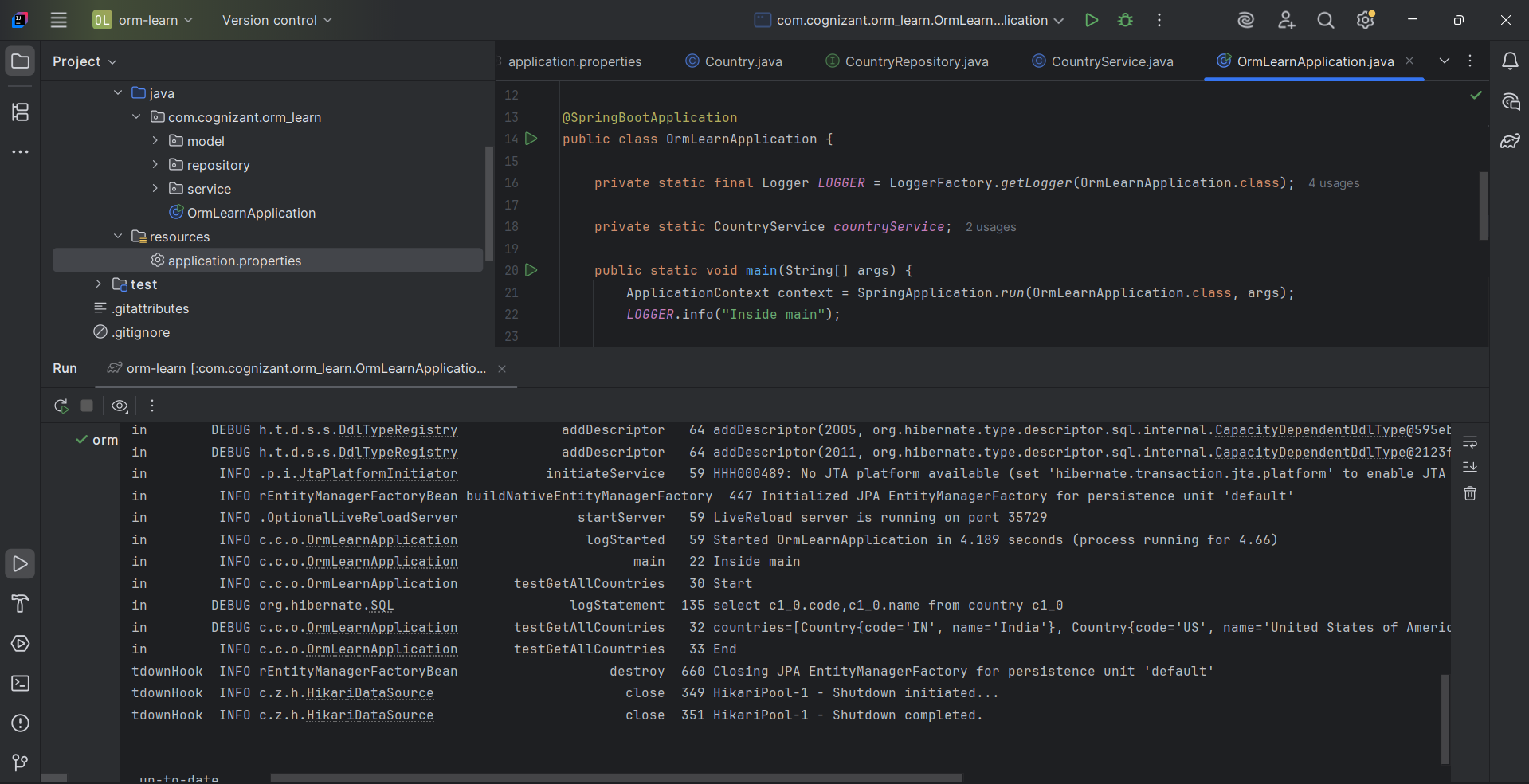
LOGGER.debug("countries={}", countries);

LOGGER.info("End");

}

}

**Output**

****

**Hands on 4**

**Difference between JPA, Hibernate and Spring Data JPA   
  
1. What is JPA?**

• JPA (Java Persistence API) is a specification for persisting Java objects to relational databases.  
• It provides annotations and APIs to map Java classes to database tables.  
• JPA is not an implementation. It requires a provider like Hibernate to work.  
•Example Providers: Hibernate, EclipseLink, OpenJPA

**2. What is Hibernate?**

• Hibernate is an ORM (Object Relational Mapping) tool and the most popular JPA implementation.  
• It provides its own APIs and also fully implements the JPA specification.  
• Helps in managing database transactions, sessions, and mappings using Java classes.

**3. What is Spring Data JPA?**

• Spring Data JPA is not a JPA provider.  
• It is a Spring abstraction over JPA/Hibernate that removes boilerplate code.  
• Helps create JPA repositories without writing actual queries.  
• Comes with built-in methods like save(), findAll(), deleteById(), etc.  
• Works with Hibernate underneath (or other JPA providers).

**Comparison Table**

|  |  |  |  |
| --- | --- | --- | --- |
| Feature | JPA | Hibernate | Spring Data JPA |
| Type | Specification (API only) | JPA Provider & ORM tool | Abstraction over JPA |
| Implementation | No | Yes (implements JPA) | No |
| Reduces Boilerplate? | Some | Moderate | Yes (very minimal code) |
| Configuration Needed | Yes | Yes | Minimal (uses Spring Boot magic) |
| Querying | JPQL, Criteria | JPQL, HQL | Derived querymethods, JPQL |
| Transaction Mgmt | Manual or via framework | Manual or via Spring | Handled by Spring (via annotations) |

**Hibernate vs Spring Data JPA — Code Snippet Comparison**

**Hibernate (Manual Way)**

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 (Simplified)**

@Repository  
public interface EmployeeRepository extends JpaRepository<Employee, Integer> {}  
  
@Service  
public class EmployeeService {  
  
 @Autowired  
 private EmployeeRepository employeeRepository;  
  
 @Transactional  
 public void addEmployee(Employee employee) {  
 employeeRepository.save(employee);  
 }  
}