**Spring data Jpa**

**Hands On 1**

**Country.java**

package com.cognizant.orm\_learn.model; // ✅ Must match your folder structure exactly

import jakarta.persistence.Column;

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

import jakarta.persistence.Table;

*@Entity*

*@Table*(name = "country") // ✅ This must match your DB table name

public class Country {

*@Id*

*@Column*(name = "co\_code") // ✅ Match DB column name

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**

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

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

*@Service*

public class CountryService {

*@Autowired*

private CountryRepository countryRepository;

*@Transactional*

public List<Country> getAllCountries() {

return countryRepository.findAll();

}}

**3.CountryRepository.java**

package com.cognizant.orm\_learn.repository;

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

import org.springframework.stereotype.Repository;

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

@Repository

public interface CountryRepository extends JpaRepository<Country, String> {

}

**4.OrmLearnApplication.java**

package com.cognizant.orm\_learn;

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.orm\_learn.model.Country;

import com.cognizant.orm\_learn.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);

*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");

}

}

**5. application.properties**

spring.application.name=orm-learn

# Spring Framework and application log

logging.level.org.springframework=info

logging.level.com.cognizant=debug

# Hibernate logs for displaying executed SQL, input and output

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 configuration

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.datasource.url=jdbc:mysql://localhost:3306/ormlearn

spring.datasource.username=root

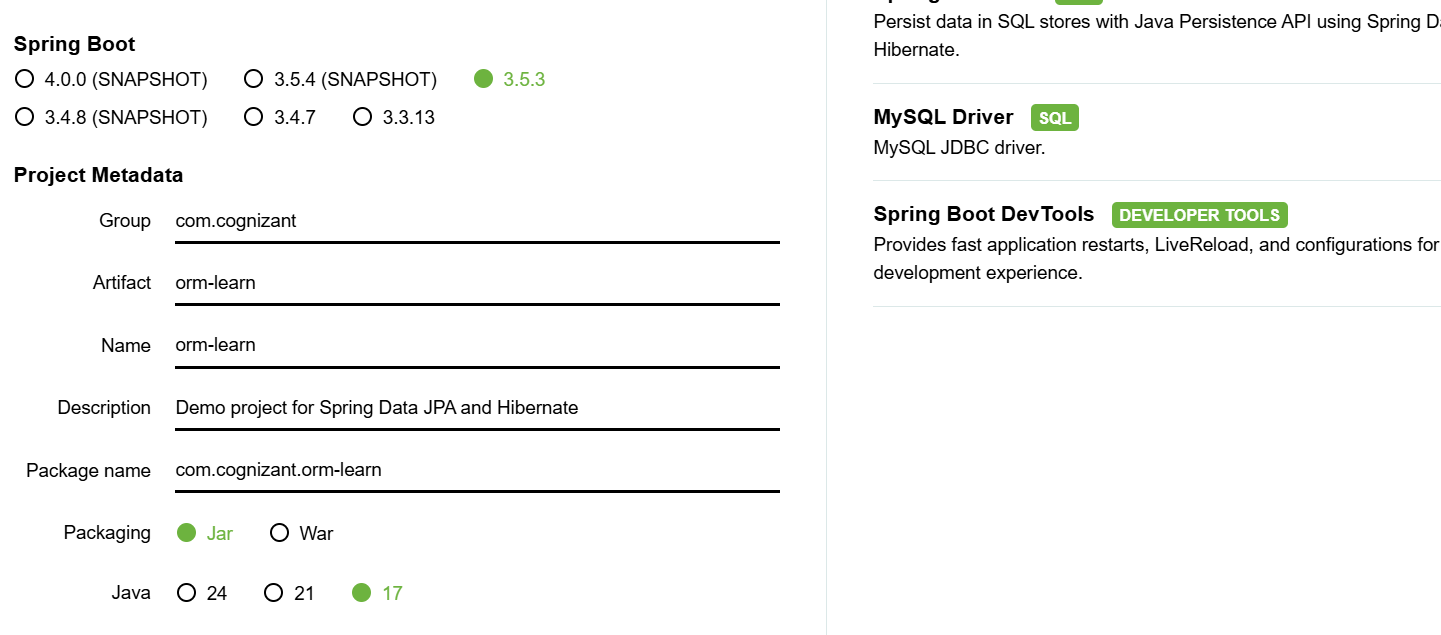
spring.datasource.password=root@123

# Hibernate configuration

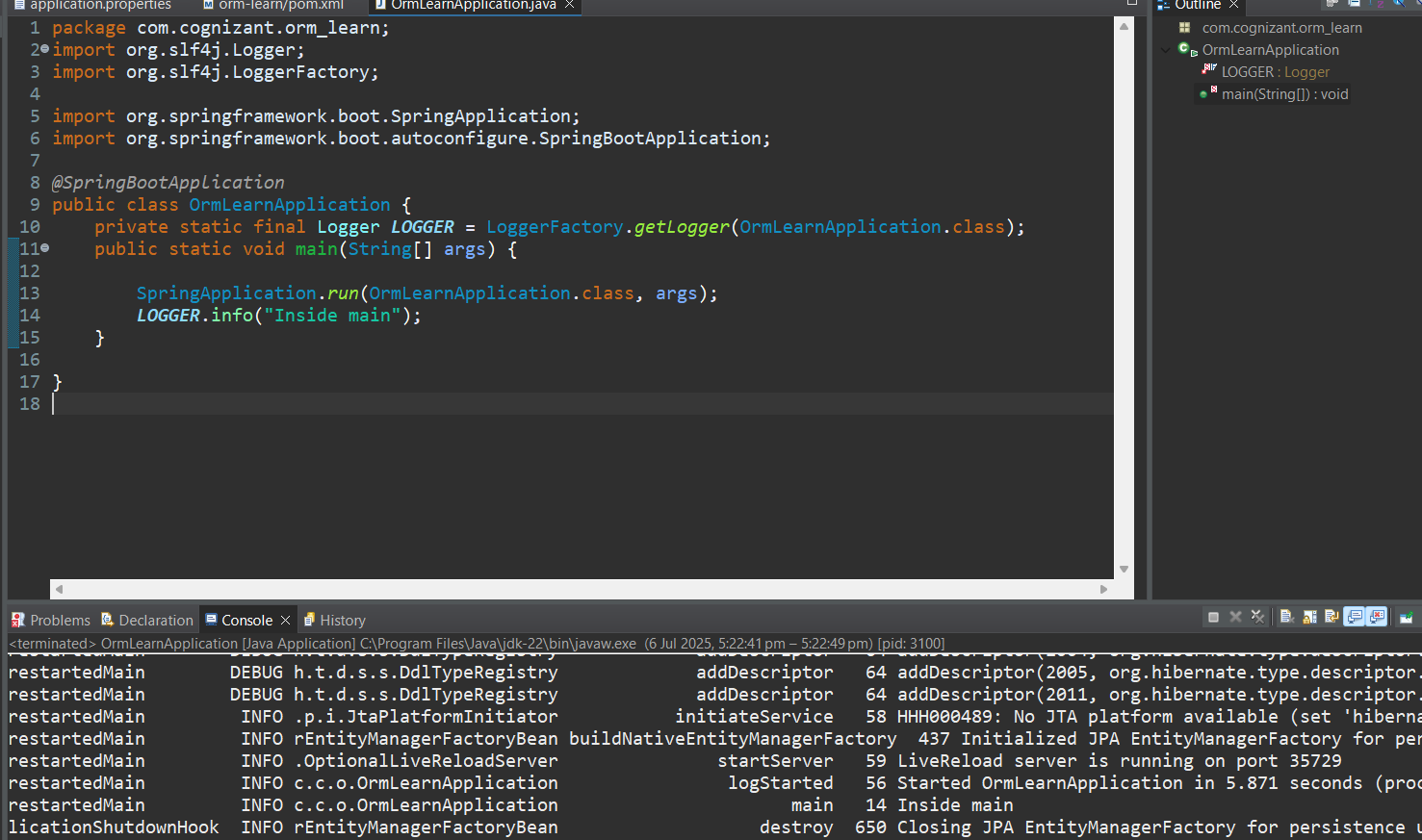
spring.jpa.hibernate.ddl-auto=validate

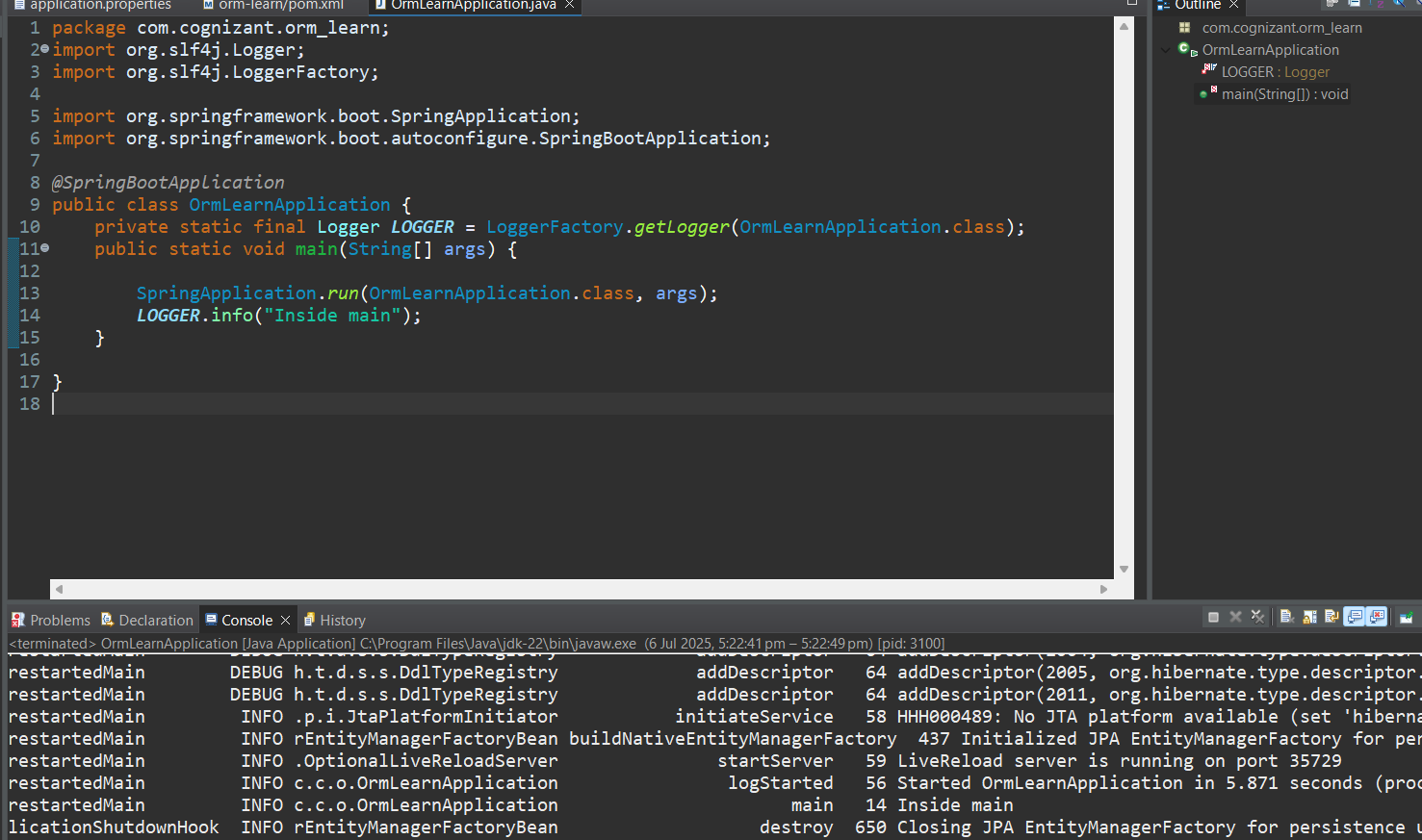
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL8Dialect

**1.SPRING INITIALIZR**

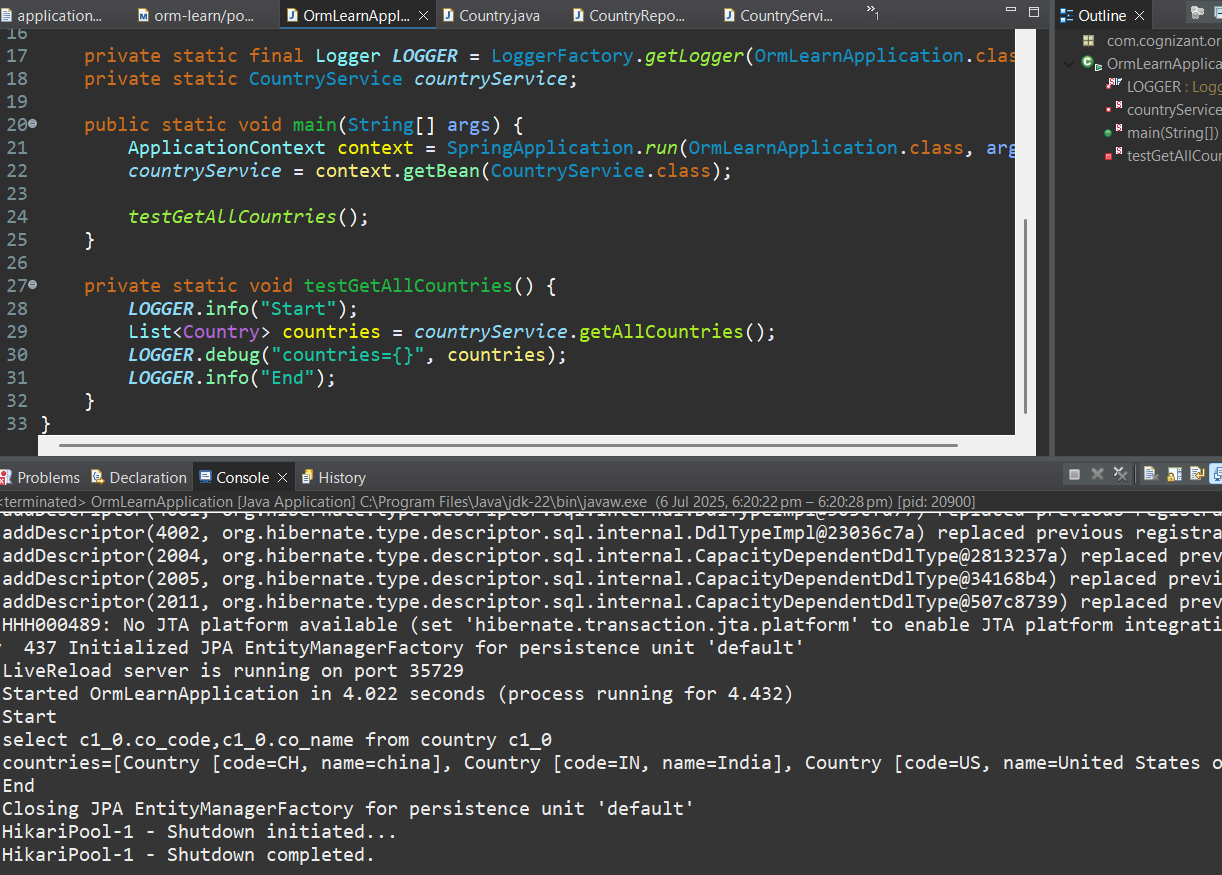


**2.MAIN METHOD()**



**3.SQL CONNECTION**

**4.COMMAND EXECUTED**



**Hands-on 4: Difference between JPA, Hibernate and Spring Data JPA**

1. Java Persistence API (JPA)

· JPA is a Java specification (JSR 338) that defines how to manage relational data using Java objects.

· It provides standard interfaces and annotations for object-relational mapping (ORM) but does not provide any implementation.

· It acts as a bridge between Java applications and the database, offering a standard way to perform CRUD operations.

· Examples of JPA annotations: @Entity, @Table, @Id, @Column.

· Popular JPA implementations include Hibernate, EclipseLink, and TopLink.

2. Hibernate

· Hibernate is a concrete ORM tool and the most widely used implementation of JPA.

· It provides additional features beyond JPA like caching, lazy loading, batch processing, and custom SQL execution.

· Hibernate allows developers to directly interact with the Session API, giving fine-grained control over transactions and database interactions.

· Requires more boilerplate code compared to Spring Data JPA.

3. Spring Data JPA

· Spring Data JPA is a Spring-based abstraction over JPA that reduces boilerplate code.

· It simplifies JPA-based data access layers by providing predefined repository interfaces like JpaRepository, CrudRepository, etc.

· It is not a JPA implementation itself, but it works on top of a JPA provider like Hibernate.

· Spring Data JPA handles the infrastructure and transaction management automatically using annotations like @Transactional and @Repository.

Code Comparison

1.Using Hibernate (Manual Transaction Management)

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;

}

2. Using Spring Data JPA (Simplified Approach)

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

}

---------------------------------------------------------------------------------------------------------------------------