**Cognizant Digital Nurture 4.0**

**Name: Siri Chandana Chittipolu**

**Email:** [**sirichittipolu11@gmail.com**](mailto:sirichittipolu11@gmail.com)

**Superset ID: 6386277**

**Mandatory Hands-On Exercises**

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

**Hands on 1 : Spring Data JPA - Quick Example :**

**Implementation Steps**

1. Project Setup
   * Created a new Spring Boot project via Spring Initializr
   * Added dependencies: *Spring Data JPA*, *MySQL Driver*, *Spring Boot DevTools*
   * Imported the project into Eclipse IDE
2. Database Setup
   * Created schema: ormlearn
   * Created table as per the given task
3. Configuration
   * Configured database connection in application.properties
   * Enabled Hibernate SQL logging for better visibility
4. Code Implementation
   * Created Country entity class with @Entity, @Table, and @Column annotations
   * Created CountryRepository interface extending repository
   * Created CountryService with getAllCountries() method annotated with @Transactional
   * In OrmLearnApplication, loaded CountryService bean, called testGetAllCountries() method to retrieve data and logged output

**application.properties:**

spring.application.name=orm-learn

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} [%thread] %-5level %logger**{36}** - %msg%n

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

spring.jpa.hibernate.ddl-auto=validate

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

**Country.java:**

package com.example.demo;

import jakarta.persistence.Entity;

import jakarta.persistence.Table;

import jakarta.persistence.Id;

import jakarta.persistence.Column;

*@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.example.demo.repository;

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

import com.example.demo.Country;

import org.springframework.stereotype.Repository;

*@Repository*

public interface CountryRepository extends JpaRepository<Country, String> {

}

**CountryService.java:**

package com.example.demo.service;

import java.util.List;

import org.springframework.transaction.annotation.Transactional;

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

import org.springframework.stereotype.Service;

import com.example.demo.Country;

import com.example.demo.repository.CountryRepository;

*@Service*

public class CountryService {

*@Autowired*

private CountryRepository countryRepository;

*@Transactional*

public List<Country> getAllCountries() {

return countryRepository.findAll();

}

}

**OrmLearnApplication.java:**

package com.example.demo;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ApplicationContext;

import com.example.demo.service.CountryService;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

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*.info("countries={}", countries);

for (Country country : countries) {

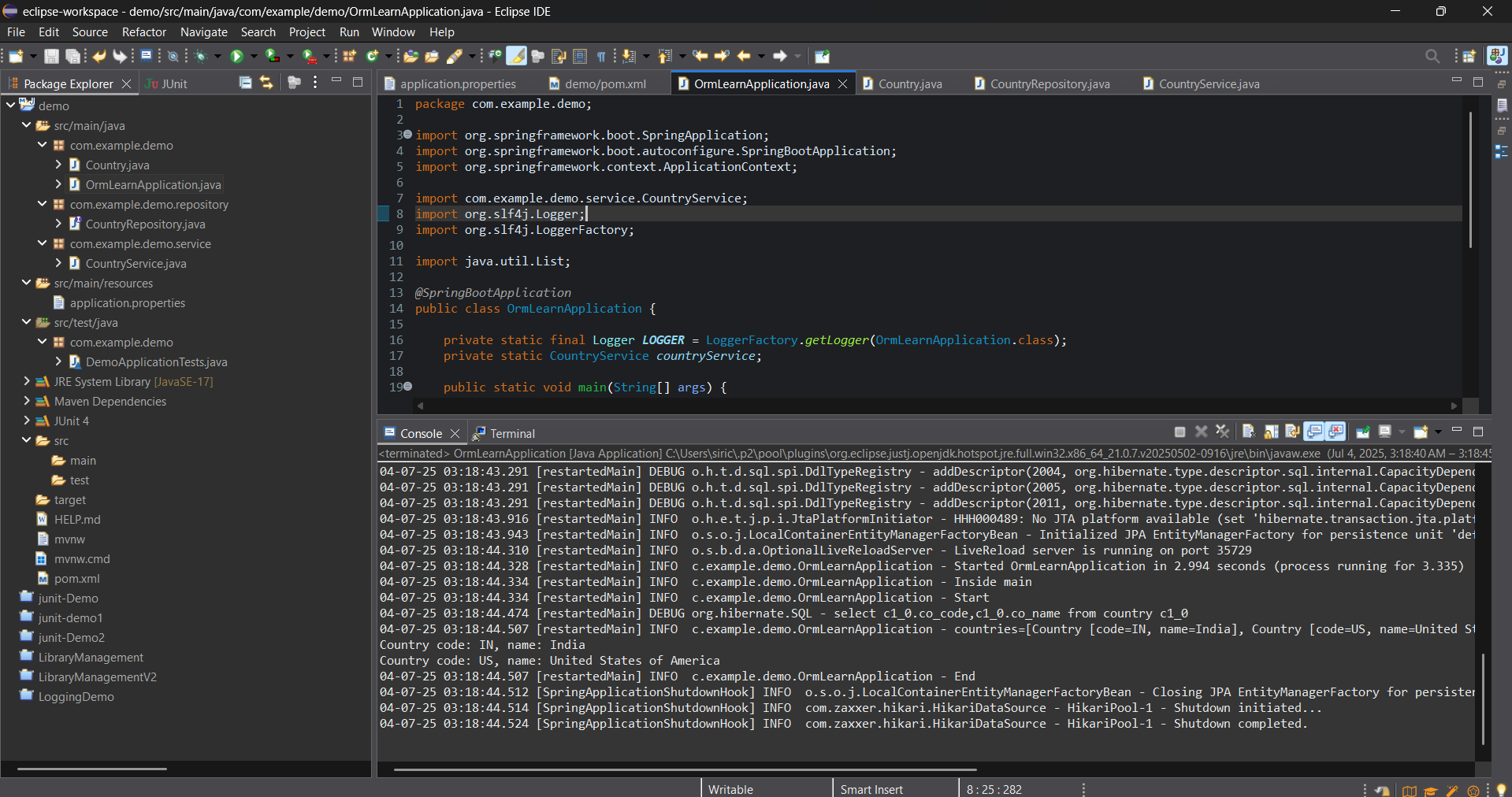
System.*out*.println("Country code: " + country.getCode() + ", name: " + country.getName()); }

*LOGGER*.info("End");

}

}

**Output:**



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

**Introduction:**

* In modern Java applications, we often work with databases to store and manage data.
* To make this easier, there are several tools and frameworks available.
* This document explains the difference between JPA, Hibernate, and Spring Data JPA with simple explanations and code examples.

**Java Persistence API (JPA):**

* JPA stands for Java Persistence API.
* It is a specification (JSR 338) that defines how to map Java objects (entities) to database tables.
* JPA itself does not have any implementation.
* We need an implementation like Hibernate, EclipseLink, etc., to actually use it.
* Basically JPA is like a set of rules or guidelines for ORM in Java.

**Hibernate:**

* Hibernate is an Object‑Relational Mapping (ORM) tool.
* It is the most popular implementation of JPA.
* With Hibernate, we can directly use its API (Session, Transaction, etc.) to interact with the database.
* However, this usually requires more boilerplate code like opening/closing sessions and manually managing transactions.

**Example code using 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, as we can notice we open the session, begin/commit transaction, and handle exceptions manually.

**Spring Data JPA:**

* Spring Data JPA is a module provided by Spring Framework.
* It does not implement JPA itself.
* Instead, it sits on top of a JPA implementation (like Hibernate) and provides another layer of abstraction.
* It helps to reduce boilerplate code by letting us define interfaces, and Spring generates the implementation at runtime.
* It also manages transactions automatically.

**Example code using Spring Data JPA:**

**EmployeeRepository.java:**

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

**EmployeeService.java:**

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

* Here, we don’t need to write any code to open sessions or manage transactions.  
  We just call save() on the repository, and Spring handles the rest.

**Conclusion:**

* JPA provides standard rules for ORM in Java.
* Hibernate is a popular tool that implements these rules.
* Spring Data JPA makes data access even easier by reducing code, managing transactions, and integrating with Spring.