**Hands on 1**

**Spring Data JPA - Quick Example** 

**Code :-**

**//File : 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

# Log pattern

logging.pattern.console=%d{dd-MM-yy} %d{HH:mm:ss.SSS} %-20.20thread %5p %-25.25logger**{25}** %25M %4L %m%n

# DB Config

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=Vishu@1357

# Hibernate

spring.jpa.hibernate.ddl-auto=validate

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

**//File : 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 = "code")

private String code;

*@Column*(name = "name")

private String name;

// Getters, setters, toString

}

**//File : CountryService.java**

package com.cognizant.orm\_learn.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.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();

}

}

**//File : 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> {

}

**//File : OrmLearnApplication.java**

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

ApplicationContext context =

new ClassPathXmlApplicationContext("applicationContext.xml");

BookService bookService = context.getBean("bookService", BookService.class);

bookService.addBook("Spring in Action");

}

} 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);

***LOGGER***.info("Inside main");

*countryService* = context.getBean(CountryService.class);

*testGetAllCountries*();

}

private static void testGetAllCountries() {

***LOGGER***.info("Start");

List<Country> countries = *countryService*.getAllCountries();

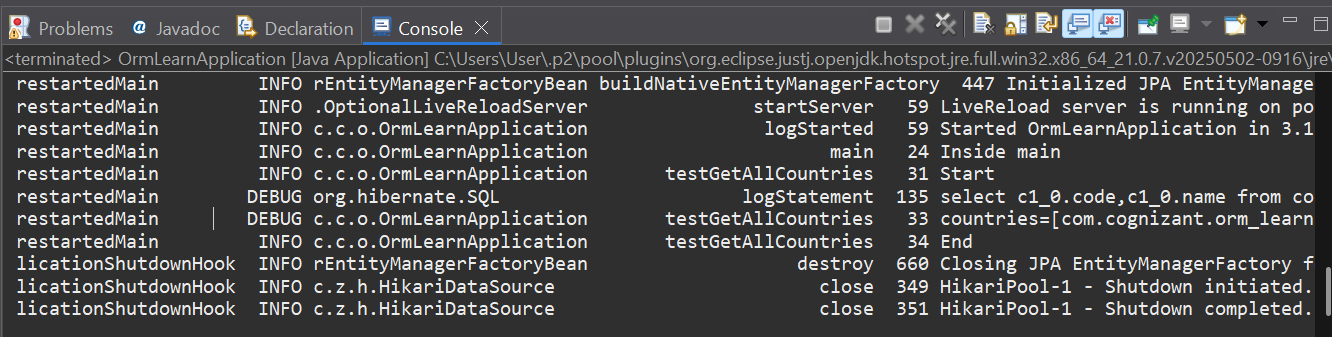
***LOGGER***.debug("countries={}", countries);

***LOGGER***.info("End");

}

}

**Output :-**

****

**Hands on 4**

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

In the Java ecosystem, interacting with databases efficiently and effectively is essential. This assignment discusses three popular technologies used for persistence:

* Java Persistence API (JPA)
* Hibernate
* Spring Data JPA

**Java Persistence API (JPA)**

* JPA is a Java specification (JSR 338) for accessing, persisting, and managing data between Java objects and relational databases.
* It is not an implementation — it's just a specification (like an interface).
* It requires a concrete implementation such as Hibernate, EclipseLink, or TopLink.

// Employee.java

@Entity

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Integer id;

private String name;

private String department;

// getters and setters

}

// persistence.xml (inside META-INF folder)

<persistence xmlns="http://xmlns.jcp.org/xml/ns/persistence"

version="2.2">

<persistence-unit name="EmployeePU">

<class>com.example.Employee</class>

<properties>

<property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3306/yourdb"/>

<property name="javax.persistence.jdbc.user" value="root"/>

<property name="javax.persistence.jdbc.password" value="password"/>

<property name="javax.persistence.jdbc.driver" value="com.mysql.cj.jdbc.Driver"/>

<property name="hibernate.dialect" value="org.hibernate.dialect.MySQL5Dialect"/>

<property name="hibernate.hbm2ddl.auto" value="update"/>

</properties>

</persistence-unit>

</persistence>

// EmployeeManager.java

public class EmployeeManager {

private static EntityManagerFactory emf = Persistence.createEntityManagerFactory("EmployeePU");

public void addEmployee(Employee employee) {

EntityManager em = emf.createEntityManager();

EntityTransaction tx = em.getTransaction();

try {

tx.begin();

em.persist(employee);

tx.commit();

} catch (Exception e) {

if (tx.isActive()) tx.rollback();

e.printStackTrace();

} finally {

em.close();

}

}

}

**Hibernate**

* Hibernate is an ORM (Object Relational Mapping) framework that implements the JPA specification.
* It provides features like caching, lazy loading, and HQL.
* Developers need to manually manage sessions and transactions.

// Employee.java (Entity)

@Entity

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Integer id;

private String name;

private String department;

// getters and setters

}

// HibernateUtil.java (Setup SessionFactory)

public class HibernateUtil {

private static SessionFactory factory;

static {

try {

factory = new Configuration().configure().buildSessionFactory();

} catch (Throwable ex) {

throw new ExceptionInInitializerError(ex);

}

}

public static SessionFactory getSessionFactory() {

return factory;

}

}

// EmployeeDAO.java

public class EmployeeDAO {

public Integer addEmployee(Employee employee) {

Session session = HibernateUtil.getSessionFactory().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**

* Spring Data JPA is part of the Spring Framework.
* It provides an abstraction over JPA and reduces boilerplate code.
* It uses repositories to handle common database operations.
* It automatically manages sessions and transactions using Spring.

// Employee.java (Entity)

@Entity

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Integer id;

private String name;

private String department;

// getters and setters

}

// EmployeeRepository.java

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);

}

}

| **Feature** | **Plain JPA** | **Hibernate** | **Spring Data JPA** |
| --- | --- | --- | --- |
| **Setup Needed** | **Yes (persistence.xml)** | **Yes (hibernate.cfg.xml)** | **Minimal (Spring Boot handles)** |
| **Code Amount** | **Medium** | **Medium** | **Very Low** |
| **Transaction Handling** | **Manual via EntityTransaction** | **Manual via Transaction** | **Auto with @Transactional** |
| **Ease of Use** | **Moderate** | **Moderate** | **Easy** |