Candidate Name:- ROHITHGOWDA V

Superset ID:-6430364

Mail ID:- [rohith2005v@gmail.com](mailto:rohith2005v@gmail.com)

**WEEK – 3 HANDS ON EXERCISE (JAVA FSE DEEPSKILLING)**

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**SPRING CORE AND MAVEN**

**Exercise 1: Configuring a Basic Spring Application**

**Scenario:**

Your company is developing a web application for managing a library. You need to use the Spring Framework to handle the backend operations.

**Steps:**

1. **Set Up a Spring Project:**
   1. Create a Maven project named **LibraryManagement**.
   2. Add Spring Core dependencies in the **pom.xml** file.
2. **Configure the Application Context:**
   1. Create an XML configuration file named **applicationContext.xml** in the **src/main/resources** directory.
   2. Define beans for **BookService** and **BookRepository** in the XML file.
3. **Define Service and Repository Classes:**
   1. Create a package com.library.service and add a class BookService.
   2. Create a package **com.library.repository** and add a class **BookRepository**.
4. **Run the Application:**
   1. Create a main class to load the Spring context and test the configuration.

**Code for above question:**

**Java class:BookRepository.java**

package com.example.riohith;  
  
import java.util.Arrays;  
import java.util.List;  
  
public class BookRepository {  
 public List<String> findAllBooks() {  
 return Arrays.*asList*("Java Basics", "Spring in Action", "Clean Code");  
 }  
}

**Java class: BookService.java**

package com.example.riohith;  
  
import java.util.List;  
  
public class BookService {  
 private BookRepository bookRepository;  
  
 public void setBookRepository(BookRepository bookRepository) {  
 this.bookRepository = bookRepository;  
 }  
  
 public void displayBooks() {  
 List<String> books = bookRepository.findAllBooks();  
 for (String book : books) {  
 System.*out*.println("Book: " + book);  
 }  
 }  
}

**Java class: MainApp.java**

package com.example.riohith;  
  
public class MainApp {  
 public static void main(String[] args) {  
 BookRepository bookRepository = new BookRepository();  
 BookService bookService = new BookService();  
 bookService.setBookRepository(bookRepository);  
 bookService.displayBooks();  
 }  
}

**Output:**

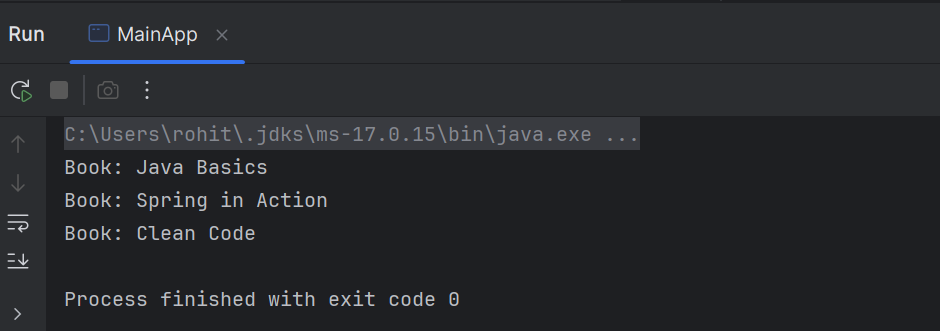
Book: Java Basics

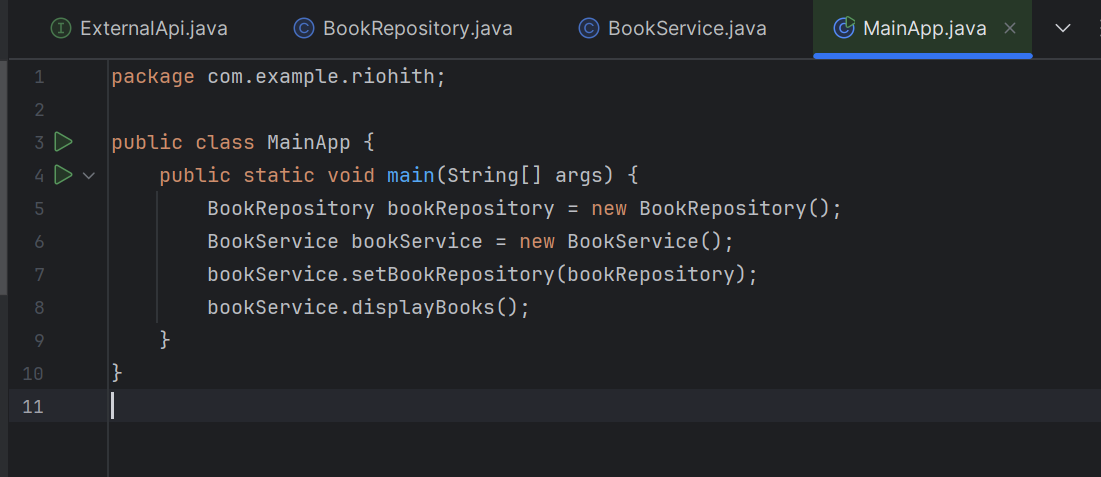
Book: Spring in Action

Book: Clean Code

Process finished with exit code 0

**Output Image:**

****

****

**Exercise 2: Implementing Dependency Injection**

**Scenario:**

In the library management application, you need to manage the dependencies between the BookService and BookRepository classes using Spring's IoC and DI.

**Steps:**

1. **Modify the XML Configuration:**
   1. Update **applicationContext.xml** to wire **BookRepository** into **BookService**.
2. **Update the BookService Class:**
   1. Ensure that **BookService** class has a setter method for **BookRepository**.
3. **Test the Configuration:**
   1. Run the **LibraryManagementApplication** main class to verify the dependency injection.

**Code for above question:**

**File: applicationContext.xml**

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd">

<bean id="bookRepository" class="com.example.riohith.BookRepository"/>  
 <bean id="bookService" class="com.example.riohith.BookService">  
 <property name="bookRepository" ref="bookRepository"/>  
 </bean>

</beans>

**Java class: BookRepository.java**

package com.example.riohith;  
import java.util.Arrays;  
import java.util.List;  
  
public class BookRepository {  
 public List<String> findAllBooks() {  
 return Arrays.*asList*("Spring Framework", "Java 17 Essentials", "Design Patterns");  
 }  
}

**Java class: BookService.java**

package com.example.riohith;  
import java.util.List;  
  
public class BookService {  
 private BookRepository bookRepository;  
  
 public void setBookRepository(BookRepository bookRepository) {  
 this.bookRepository = bookRepository;  
 }  
  
 public void displayBooks() {  
 List<String> books = bookRepository.findAllBooks();  
 for (String book : books) {  
 System.*out*.println("Book: " + book);  
 }  
 }  
}

**Java class: LibraryManagementApplication.java**

package com.example.riohith;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
  
public class LibraryManagementApplication {  
 public static void main(String[] args) {  
 ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");  
 BookService bookService = context.getBean("bookService", BookService.class);  
 bookService.displayBooks();  
 }  
}

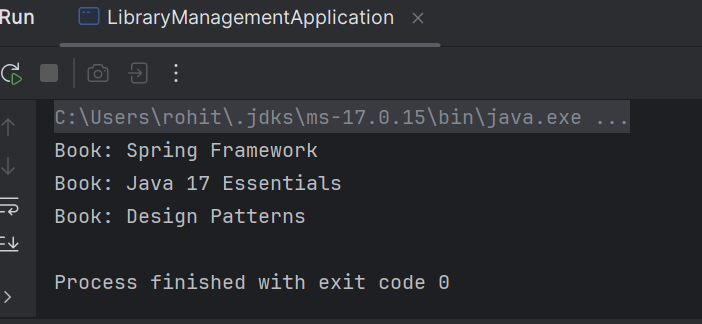
**Output:**

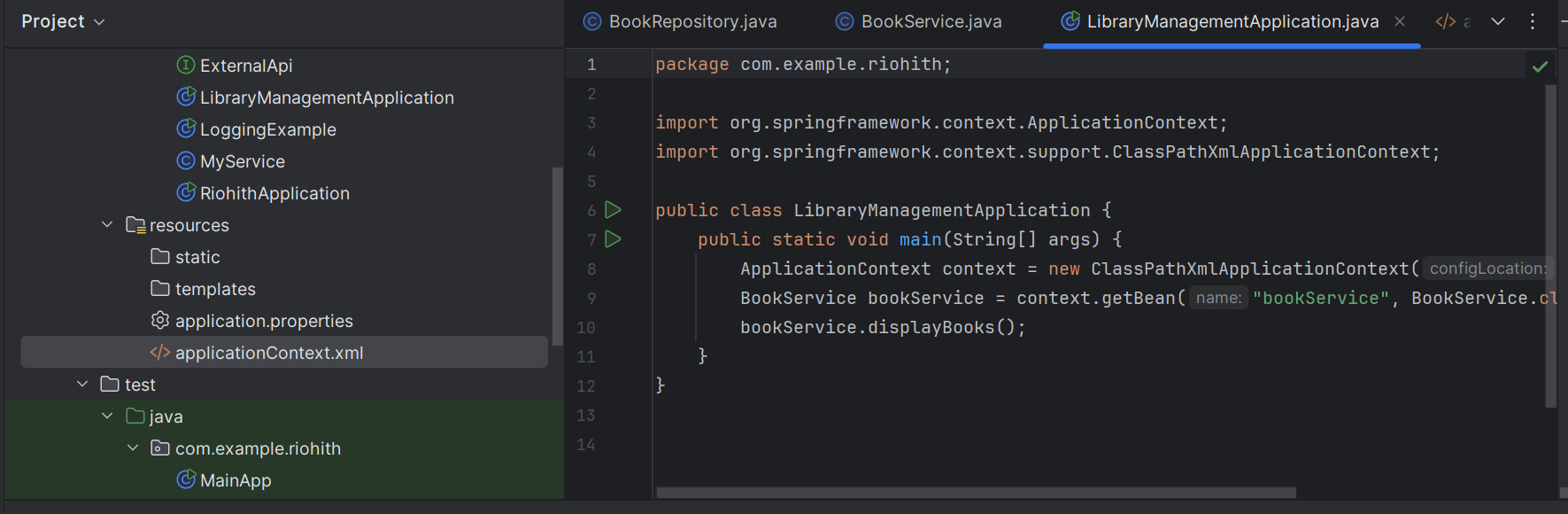
Book: Spring Framework

Book: Java 17 Essentials

Book: Design Patterns

**Output Image:**





**Exercise 4: Creating and Configuring a Maven Project**

**Scenario:**

You need to set up a new Maven project for the library management application and add Spring dependencies.

**Steps:**

1. **Create a New Maven Project:**
   1. Create a new Maven project named **LibraryManagement**.
2. **Add Spring Dependencies in pom.xml:**
   1. Include dependencies for Spring Context, Spring AOP, and Spring WebMVC.
3. **Configure Maven Plugins:**
   1. Configure the Maven Compiler Plugin for Java version 1.8 in the pom.xml file.

**Code for above question:**

**File: applicationContext.xml**

<?xml version="1.0" encoding="UTF-8"?>  
<beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="  
 http://www.springframework.org/schema/beans  
 http://www.springframework.org/schema/beans/spring-beans.xsd">  
  
 <bean id="bookRepository" class="com.example.riohith.BookRepository"/>  
 <bean id="bookService" class="com.example.riohith.BookService">  
 <property name="bookRepository" ref="bookRepository"/>  
 </bean>  
</beans>

**Java class: BookRepository.java**

package com.example.riohith;  
import java.util.Arrays;  
import java.util.List;  
  
public class BookRepository {  
 public List<String> findAllBooks() {  
 return Arrays.*asList*("Spring Framework", "Java Concurrency", "Clean Architecture");  
 }  
}

**Java class: BookService.java**

package com.example.riohith;  
  
import java.util.List;  
  
public class BookService {  
 private BookRepository bookRepository;  
  
 public void setBookRepository(BookRepository bookRepository) {  
 this.bookRepository = bookRepository;  
 }  
  
 public void displayBooks() {  
 List<String> books = bookRepository.findAllBooks();  
 for (String book : books) {  
 System.*out*.println("Book: " + book);  
 }  
 }  
}

**Java class: LibraryManagementApplication.java**

package com.example.riohith;  
  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
  
public class LibraryManagementApplication {  
 public static void main(String[] args) {  
 ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");  
 BookService service = context.getBean("bookService", BookService.class);  
 service.displayBooks();  
 }  
}

**Output:**

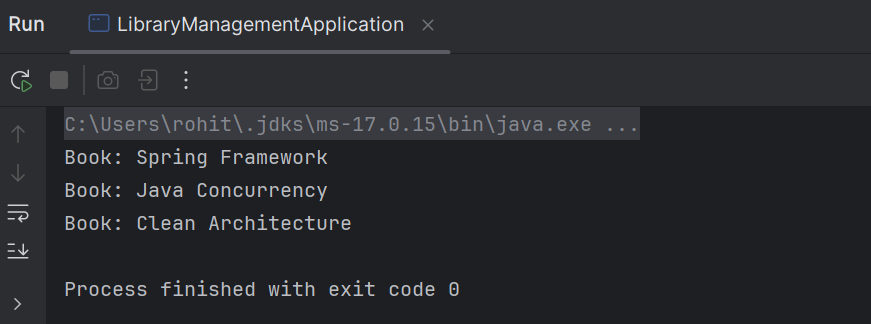
Book: Spring Framework

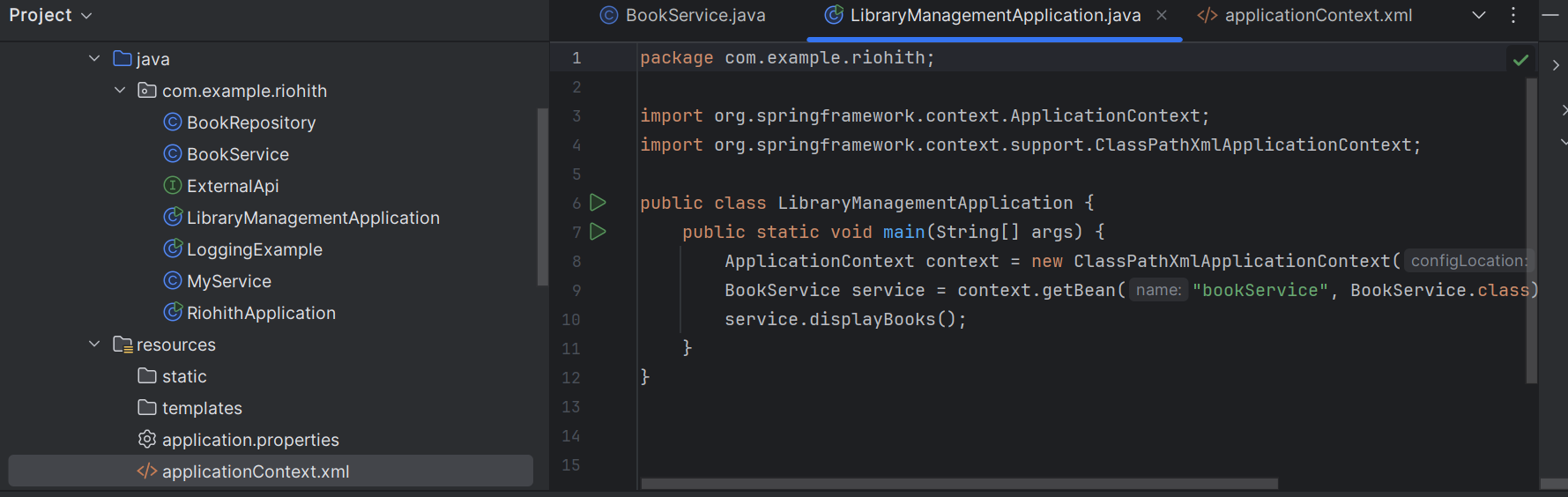
Book: Java Concurrency

Book: Clean Architecture

Process finished with exit code 0

**Output Image:**

****

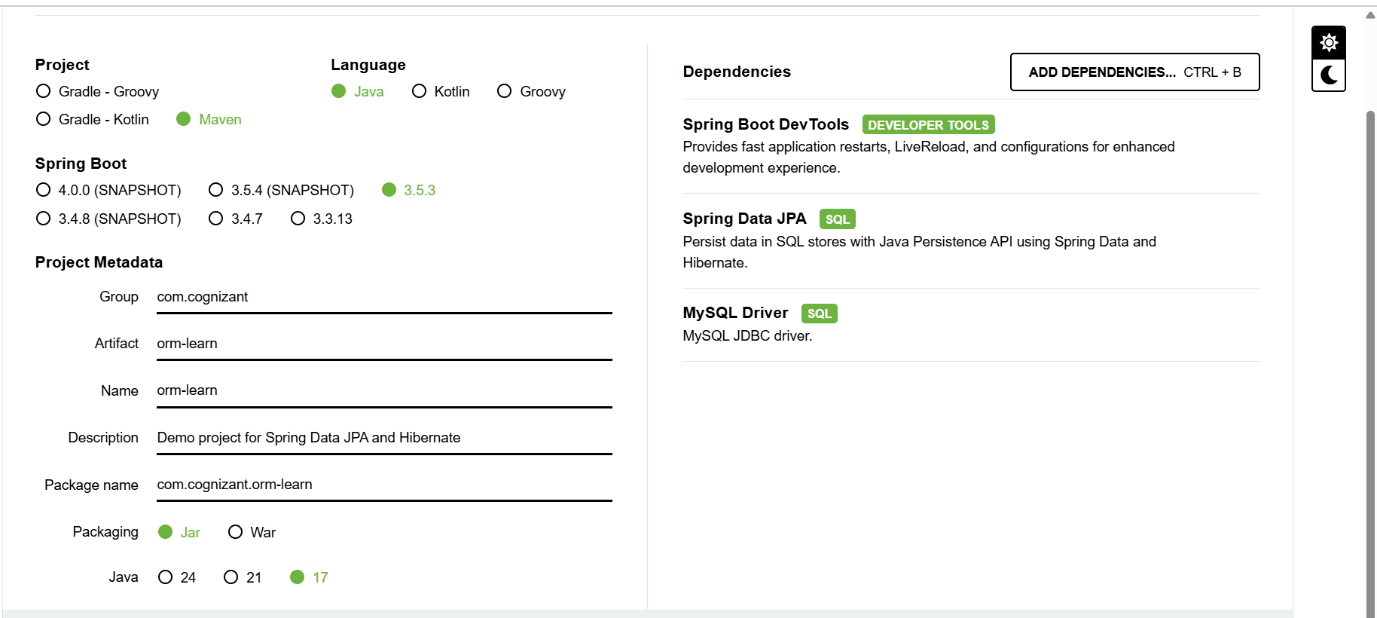
****

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

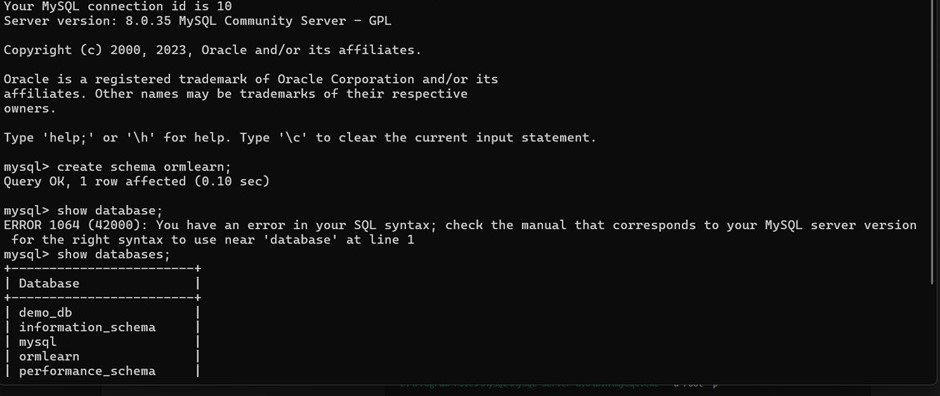
**Spring Data JPA - Quick Example**   
  
**Software Pre-requisites**

1. MySQL Server 8.0
2. MySQL Workbench 8
3. Eclipse IDE for Enterprise Java Developers 2019-03 R
4. Maven 3.6.2

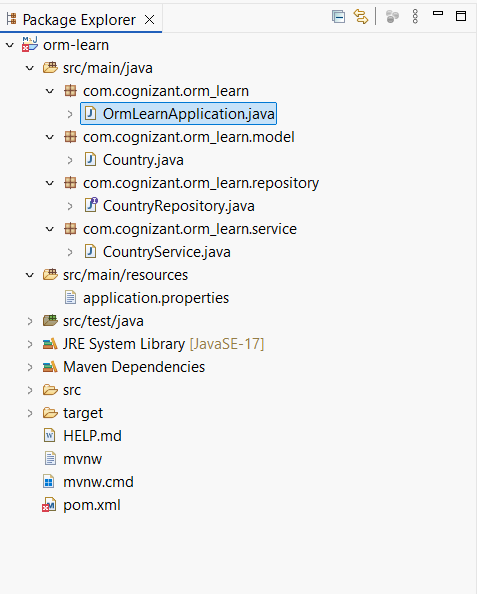
**Create a Eclipse Project using Spring Initializer**



Create a new schema "ormlearn" in MySQL database. Execute the following commands to open MySQL client and create schema.



**SNAPSHOT OF THE SCHEMA CREATED ACCORDING TO GIVEN INSTRUCTION**

****

**NOW THE CODE OF EACH FILE WILL BE GIVEN BELOW**

**pom.xml**

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>3.5.3</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>com.cognizant</groupId>

<artifactId>orm-learn</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>orm-learn</name>

<description>Demo project for Spring Data JPA and Hibernate</description>

<url/>

<licenses>

<license/>

</licenses>

<developers>

<developer/>

</developers>

<scm>

<connection/>

<developerConnection/>

<tag/>

<url/>

</scm>

<properties>

<java.version>17</java.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

<optional>true</optional>

</dependency>

<dependency>

<groupId>com.mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

**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=Rohith@12

# Hibernate configuration

spring.jpa.hibernate.ddl-auto=validate

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

**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);

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");

}

}

**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;

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 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> {

}

**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();

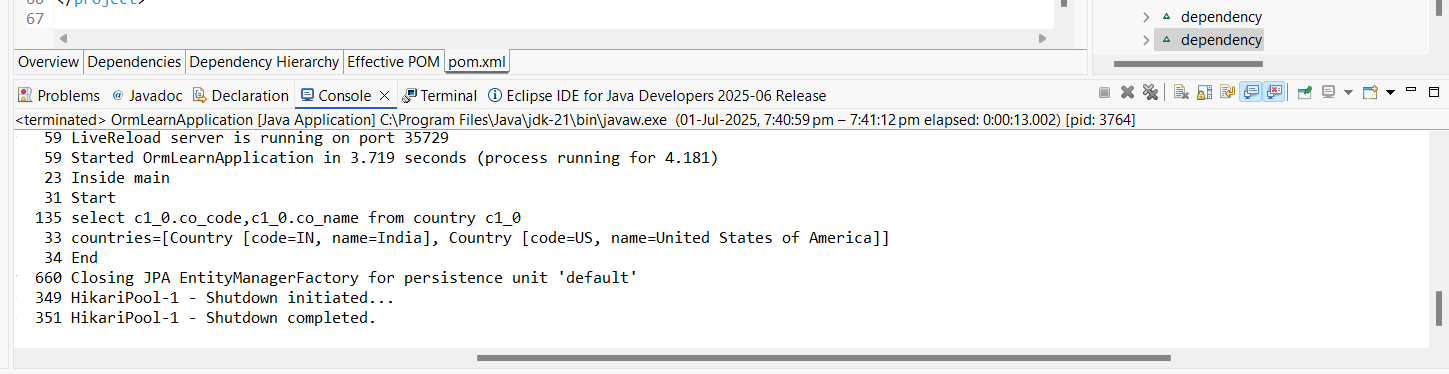
}

}

**Country table creation**



Output after running the main java code file where country being displayed above is the snapshot where the table is created and value is inserted.



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

**Answer:**

In Java-based applications that interact with relational databases, developers often use ORM (Object-Relational Mapping) tools to simplify data handling. Three commonly discussed terms in this area are JPA, Hibernate, and Spring Data JPA. While these are closely related, they serve different purposes in the architecture.

**JPA (Java Persistence API)**

What is JPA?

JPA is a Java specification that defines a standard way for managing relational data in Java applications using objects. It is a set of interfaces and annotations not an implementation.

Key Points:

* Part of Jakarta EE (formerly Java EE).
* Defines how Java objects (entities) are mapped to database tables.
* Requires a JPA provider (like Hibernate) to actually execute persistence logic.
* Provides common annotations like @Entity, @Id, @Table, @OneToMany, etc.

Example:

@Entity

public class Student {

@Id

private Long id;

private String name;

}

JPA by itself cannot persist data. It only specifies how data should be persisted. You need an implementation like Hibernate.

**Hibernate**

What is Hibernate?

Hibernate is an ORM tool and a JPA provider. It implements the JPA interfaces and also offers additional features beyond what JPA defines.

Key Points:

* It is a concrete framework, not just a specification.
* Translates Java class operations (like save, update) into SQL.
* Provides features like:
  + Caching (1st level and 2nd level)
  + Lazy loading
  + Custom HQL (Hibernate Query Language)
  + Automatic schema generation
* Can be used independently (without JPA).

Example using Hibernate Session API:

Session session = sessionFactory.openSession();

Student student = session.get(Student.class, 1L);

Hibernate is a powerful engine that works under the hood of most JPA-based frameworks.

**Spring Data JPA**

What is Spring Data JPA?

Spring Data JPA is a framework from Spring that builds on top of JPA and simplifies repository layer development. It automates boilerplate code like creating DAO classes and writing queries.

Key Points:

* Built into the Spring ecosystem (especially with Spring Boot).
* Uses JPA (typically backed by Hibernate) underneath.
* Provides ready-to-use interfaces like JpaRepository, CrudRepository.
* Supports automatic query generation using method names.
* Integrates easily with pagination, sorting, and custom queries.

Example:

public interface StudentRepository extends JpaRepository<Student, Long> {

List<Student> findByName(String name);

}

With Spring Data JPA, you often don’t need to write SQL or JPQL at all.

| **Feature** | **JPA** | **Hibernate** | **Spring Data JPA** |
| --- | --- | --- | --- |
| Type | Specification (API) | Implementation (Framework) | Abstraction Layer |
| Provides Implementation? | No | Yes | Built on JPA (uses Hibernate) |
| Maintained By | Jakarta EE Community | Red Hat | Spring (Pivotal) |
| Ease of Use | Moderate | Moderate | High |
| Query Language Support | JPQL | JPQL + HQL | JPQL, Native SQL, Method Name Queries |
| Reduces Boilerplate Code | No | Partially | Yes |
| Standalone Use | Yes | Yes | Requires Spring Context |
| Popular Use Case | Enterprise Java Standards | Advanced ORM Features | Spring Boot Applications |

In modern Spring Boot applications, Spring Data JPA with Hibernate as the JPA provider is a common and powerful combination that allows developers to build database-driven applications quickly and efficiently.