**Mandatory Hands On**

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

**Code:**

pom.xml :-

<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

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.library</groupId>

<artifactId>LibraryManagement</artifactId>

<version>1.0-SNAPSHOT</version>

<dependencies>

<!-- Spring Core -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.34</version>

</dependency>

</dependencies>

</project>

applicationContent.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">

<!-- Define BookRepository bean -->

<bean id="bookRepository" class="com.library.repository.BookRepository"/>

<!-- Define BookService bean and inject BookRepository -->

<bean id="bookService" class="com.library.service.BookService">

<property name="bookRepository" ref="bookRepository"/>

</bean>

</beans>

BookRepository.java :-

package com.library.repository;

public class BookRepository {

public void saveBook(String bookName) {

System.out.println("Book saved: " + bookName);

}

}

BookService.java :-

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void addBook(String name) {

System.out.println("Adding book: " + name);

bookRepository.saveBook(name);

}

}

MainApp.java :-

package com.library.main;

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");

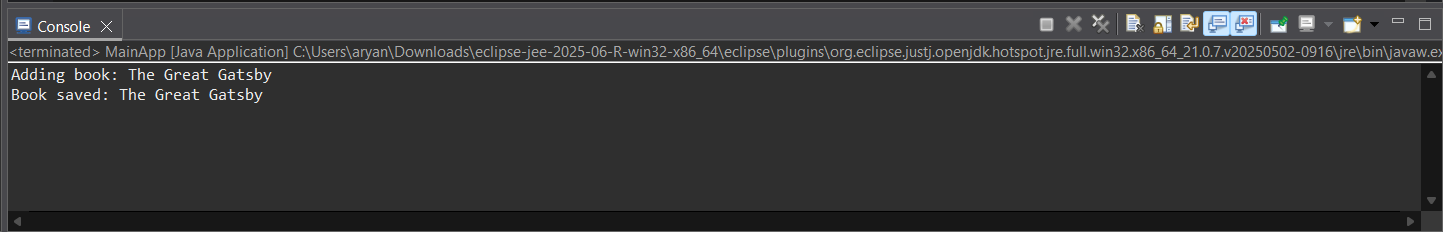
BookService bookService = context.getBean("bookService", BookService.class);

bookService.addBook("The Great Gatsby");

}

}

**Output:**



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

**Code:**

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">

<!-- Define BookRepository bean -->

<bean id="bookRepository" class="com.library.repository.BookRepository"/>

<!-- Define BookService bean and inject BookRepository -->

<bean id="bookService" class="com.library.service.BookService">

<property name="bookRepository" ref="bookRepository"/>

</bean>

</beans>

BookService.java :-

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

// Setter for Spring Dependency Injection

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void addBook(String bookName) {

bookRepository.saveBook(bookName);

}

}

MainApp.java :-

package com.library;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

public class MainApp {

public static void main(String[] args) {

ApplicationContext context =

new ClassPathXmlApplicationContext("applicationContext.xml");

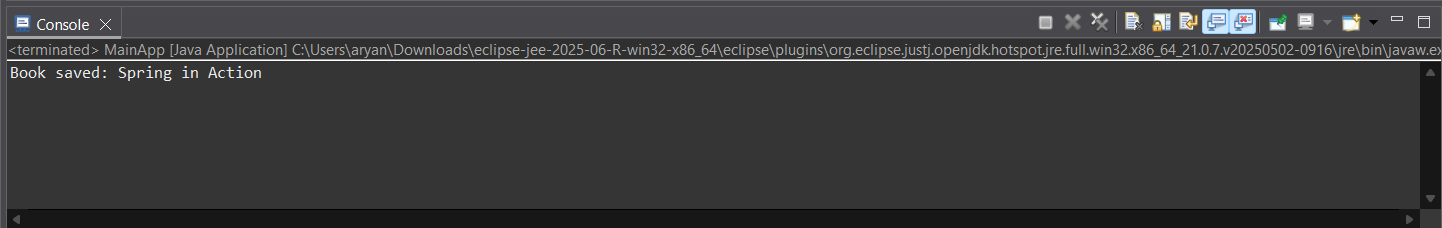
BookService bookService = context.getBean("bookService", BookService.class);

bookService.addBook("Spring in Action"); // This will invoke saveBook()

}

}

**Output:**



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

**Code:**

pom.xml :-

<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

http://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.library</groupId>

<artifactId>LibraryManagement</artifactId>

<version>1.0-SNAPSHOT</version>

<properties>

<maven.compiler.source>1.8</maven.compiler.source>

<maven.compiler.target>1.8</maven.compiler.target>

</properties>

<dependencies>

<!-- Spring Context -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.36</version>

</dependency>

<!-- Spring AOP -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.36</version>

</dependency>

<!-- Spring Web MVC -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.36</version>

</dependency>

</dependencies>

<build>

<plugins>

<!-- Maven Compiler Plugin for Java 1.8 -->

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.10.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

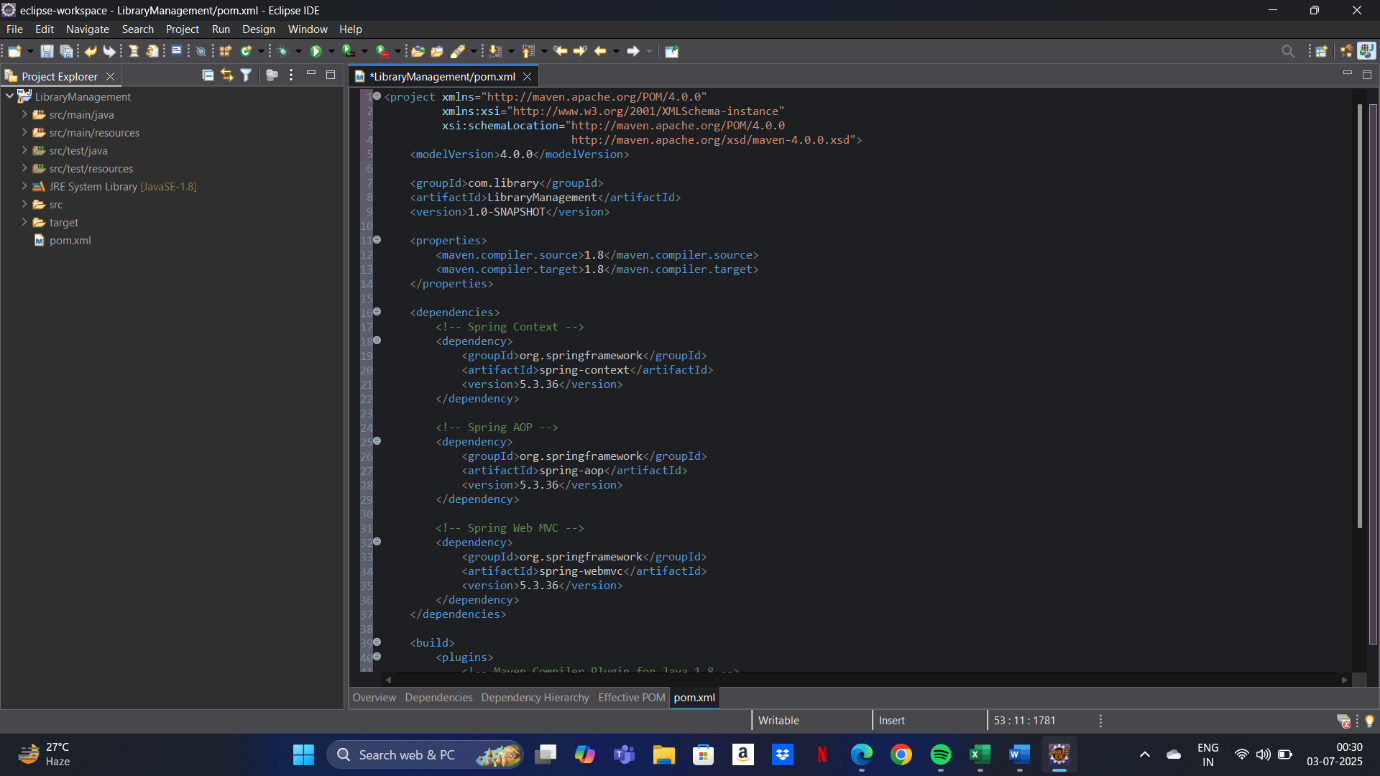
</plugin>

</plugins>

</build>

</project>

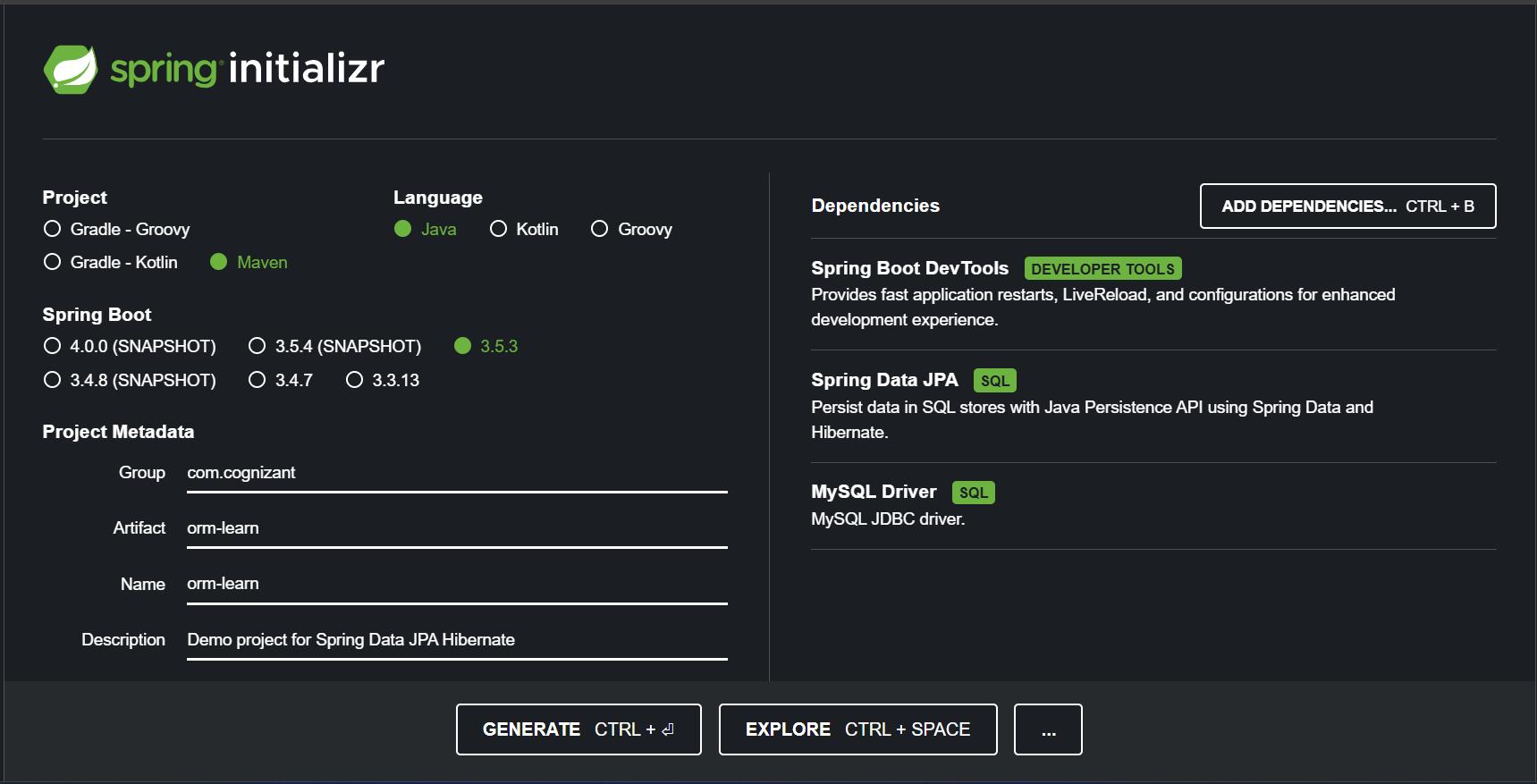
**Output:**

****

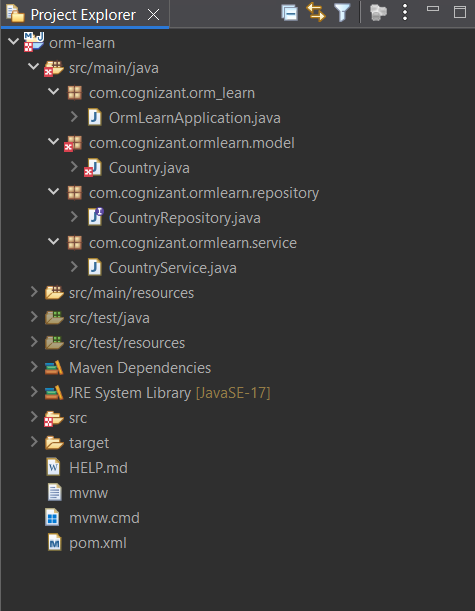
**Spring Data JPA with Spring Boot, Hibernate**

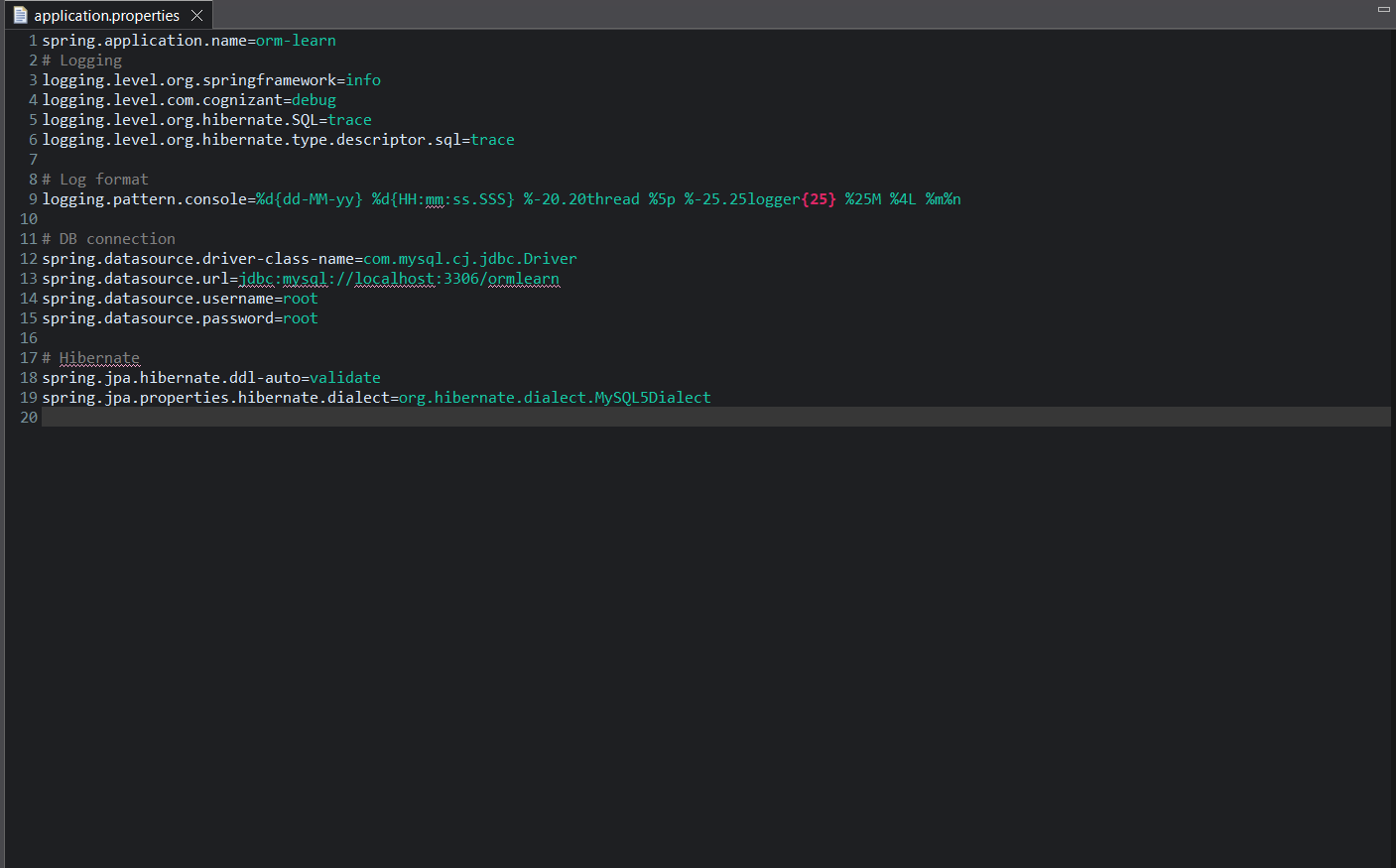
**Exercise 1: Spring Data JPA - Quick Example**

<https://start.spring.io/>

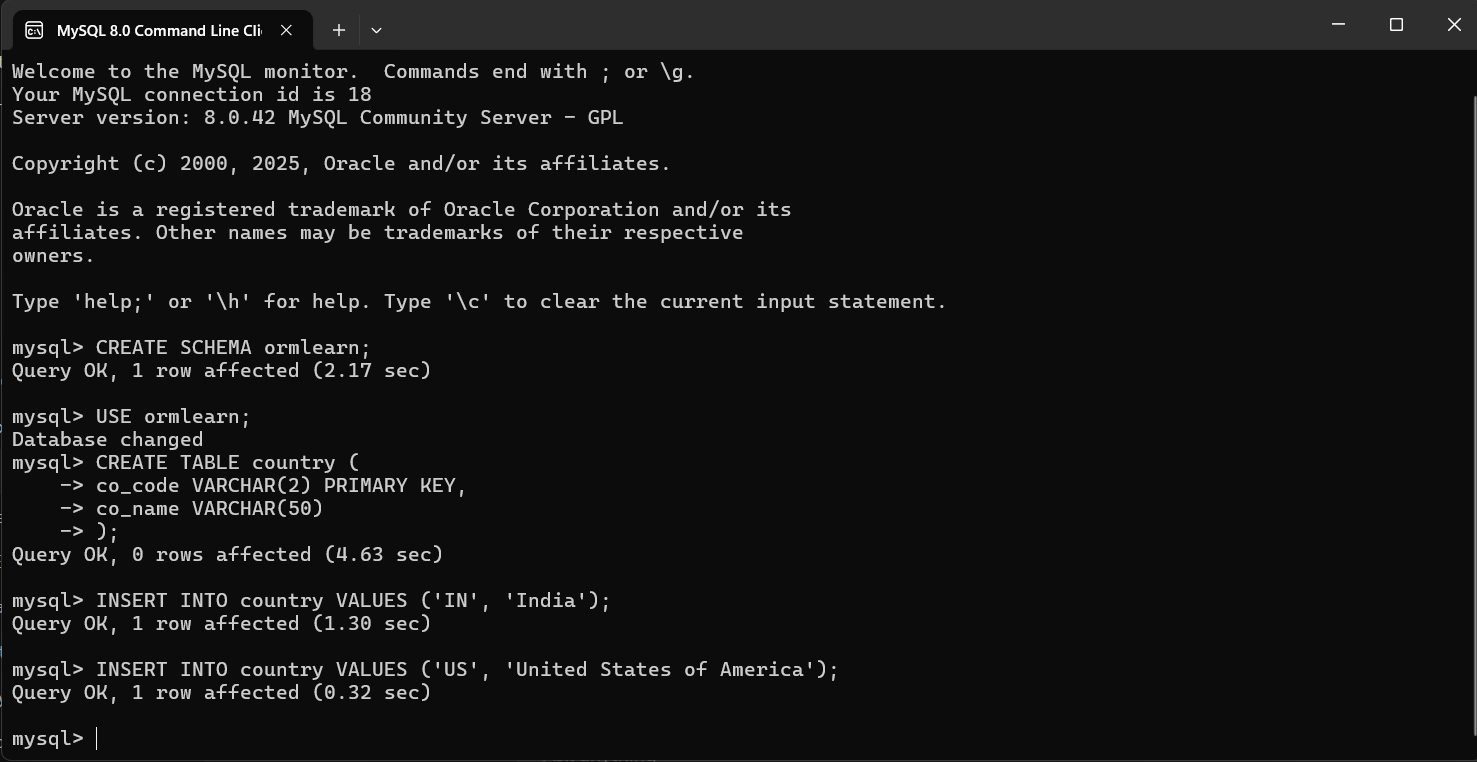


Eclipse IDE :-





MySQL CLI:



Java code portions:

OrmLearnApplication.java :-

package com.cognizant.ormlearn;

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.ormlearn.model.Country;

import com.cognizant.ormlearn.service.CountryService;

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");

}

}

Country.java :-

package com.cognizant.ormlearn.model;

import javax.persistence.Column;

import javax.persistence.Entity;

import javax.persistence.Id;

import javax.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 + "]";

}

}

Repository Interface :-

package com.cognizant.ormlearn.repository;

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

import org.springframework.stereotype.Repository;

import com.cognizant.ormlearn.model.Country;

@Repository

public interface CountryRepository extends JpaRepository<Country, String> {

}

CountryService.java :-

package com.cognizant.ormlearn.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.ormlearn.model.Country;

import com.cognizant.ormlearn.repository.CountryRepository;

@Service

public class CountryService {

@Autowired

private CountryRepository countryRepository;

@Transactional

public List<Country> getAllCountries() {

return countryRepository.findAll();

}

}

**Exercise 2:** **Difference between JPA, Hibernate and Spring Data JPA**

| **Feature / Aspect** | **JPA (Java Persistence API)** | **Hibernate** | **Spring Data JPA** |
| --- | --- | --- | --- |
| **Type** | Specification (API) | Implementation of JPA (ORM framework) | Framework abstraction over JPA/Hibernate |
| **Purpose** | Defines standard for ORM in Java | Provides actual ORM functionality | Simplifies JPA-based data access using Spring |
| **Who Provides It** | Java (part of Jakarta EE / Java EE) | Red Hat (as part of Hibernate ORM) | Spring Framework (by Pivotal/VMware) |
| **Functionality** | Only interfaces and annotations | Implements JPA and adds extra features (e.g., caching, HQL) | Provides repository support, query derivation, and custom queries |
| **Requires Provider?** | Yes (like Hibernate, EclipseLink, etc.) | No (it is a provider itself) | Yes (uses JPA provider internally like Hibernate) |
| **Ease of Use** | Requires manual configuration and coding of repositories | Easier than raw JDBC but still involves boilerplate | Easiest — reduces boilerplate with JpaRepository interfaces |
| **Query Language** | JPQL (Java Persistence Query Language) | HQL (Hibernate Query Language), supports JPQL too | JPQL, native SQL, derived queries (method names), and @Query |
| **Integration with Spring** | Indirect, needs more configuration | Direct support in Spring ORM module | Deeply integrated with Spring, uses Spring Boot and repositories |
| **Custom Features** | Limited to standard JPA behavior | Offers advanced features (e.g., lazy loading, batch fetching) | Offers pagination, auditing, query generation, and projections |
| **Boilerplate Code** | High — need to manage EntityManager manually | Moderate | Very low — CRUD and pagination handled automatically |

**Additional Important Hands On**

**Spring Core and Maven**

**Exercise 5: Configuring the Spring IoC Container**

**Scenario:**

**The library management application requires a central configuration for beans and dependencies.**

**Steps:**

1. **Create Spring Configuration File:**
   * **Create an XML configuration file named applicationContext.xml in the src/main/resources directory.**
   * **Define beans for BookService and BookRepository in the XML file.**
2. **Update the BookService Class:**
   * **Ensure that the BookService class has a setter method for BookRepository.**
3. **Run the Application:**
   * **Create a main class to load the Spring context and test the configuration.**

**Code:**

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">

<!-- BookRepository bean -->

<bean id="bookRepository" class="com.library.repository.BookRepository"/>

<!-- BookService bean with dependency injection -->

<bean id="bookService" class="com.library.service.BookService">

<property name="bookRepository" ref="bookRepository"/>

</bean>

</beans>

BookService.java :-

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

// Setter for Dependency Injection

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void addBook(String name) {

bookRepository.saveBook(name);

}

}

BookRepository.java :-

package com.library.repository;

public class BookRepository {

public void saveBook(String bookName) {

System.out.println("Book saved: " + bookName);

}

}

LibraryManagementApp(Main) :-

package com.library;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

public class LibraryManagementApp {

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

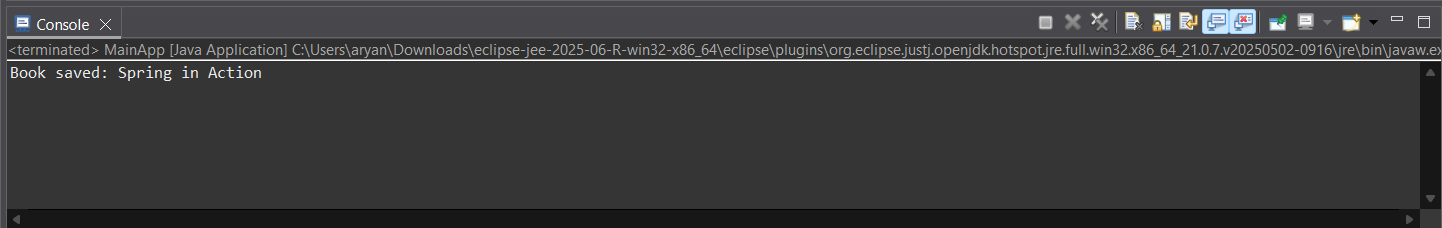
BookService bookService = (BookService) context.getBean("bookService");

bookService.addBook("Spring in Action");

}

}

**Output:**

****

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

**Exercise 6: Find a country based on country code**

CountryNotFoundException.java :-

package com.cognizant.spring-learn.service.exception;

public class CountryNotFoundException extends Exception {

public CountryNotFoundException(String message) {

super(message);

}

}

CountryService.java :-

package com.cognizant.spring-learn.service;

import com.cognizant.spring-learn.model.Country;

import com.cognizant.spring-learn.repository.CountryRepository;

import com.cognizant.spring-learn.service.exception.CountryNotFoundException;

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

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import java.util.Optional;

@Service

public class CountryService {

@Autowired

private CountryRepository countryRepository;

@Transactional

public Country findCountryByCode(String countryCode) throws CountryNotFoundException {

Optional<Country> result = countryRepository.findById(countryCode);

if (!result.isPresent()) {

throw new CountryNotFoundException("Country not found with code: " + countryCode);

}

return result.get();

}

}

Adding this method to OrmLearnApplication :-

private static final Logger LOGGER = LoggerFactory.getLogger(OrmLearnApplication.class);

@Autowired

private static CountryService countryService;

private static void getAllCountriesTest() {

LOGGER.info("Start");

try {

Country country = countryService.findCountryByCode("IN");

LOGGER.debug("Country:{}", country);

} catch (CountryNotFoundException e) {

LOGGER.error("Exception: {}", e.getMessage());

}

LOGGER.info("End");

}

Calling the test method in main :-

public static void main(String[] args) {

ApplicationContext context = SpringApplication.run(OrmLearnApplication.class, args);

countryService = context.getBean(CountryService.class);

getAllCountriesTest();

}

**Output:**

