**WEEK 3**

**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.

**MainApp.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) {

ApplicationContextcontext=newClassPathXmlApplicationContext("applicationContext.xml")

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

bookService.displayBook();

}

}

**BookRepository.java**

package com.library.repository;

public class BookRepository {

public String getBook() {

return "The Great Gatsby";

}

}

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 displayBook() {

System.*out*.println("Book: " + bookRepository.getBook());

}

}

**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">

<!-- BookRepository Bean -->

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

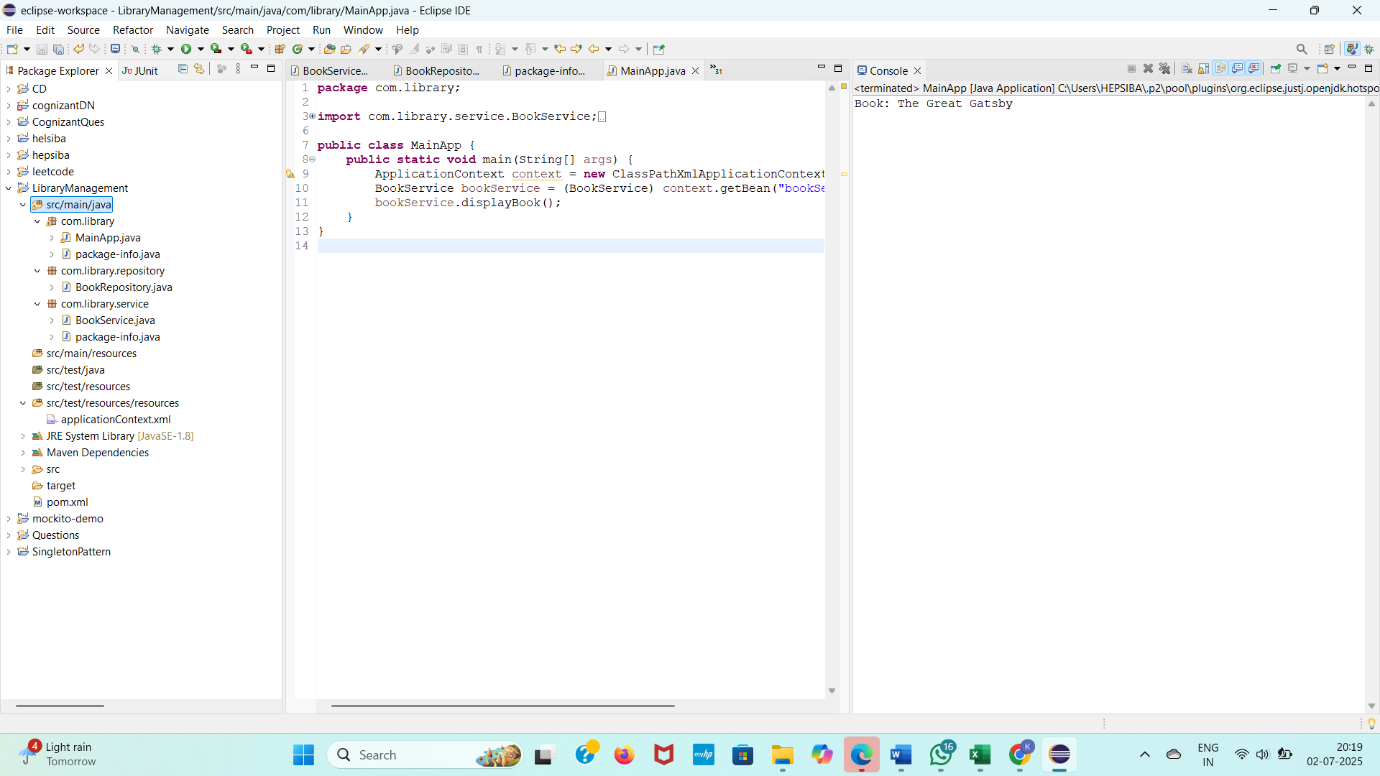
<!-- BookService Bean -->

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

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

</bean>

</beans>



**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.

**Application.xml**

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) {

ApplicationContextcontext=newClassPathXmlApplicationContext("applicationContext.xml");

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

bookService.displayBook();

}

}

**MainApp.java**

package com.library.repository;

public class AdvancedBookRepository extends BookRepository {

@Override

public String getBook() {

return "📘 Injected Book: 1984 by George Orwell";

}

}

**BookRepository**

package com.library.repository;

public class BookRepository {

public String getBook() {

return " Injected Book: To Kill a Mockingbird";

}

}

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

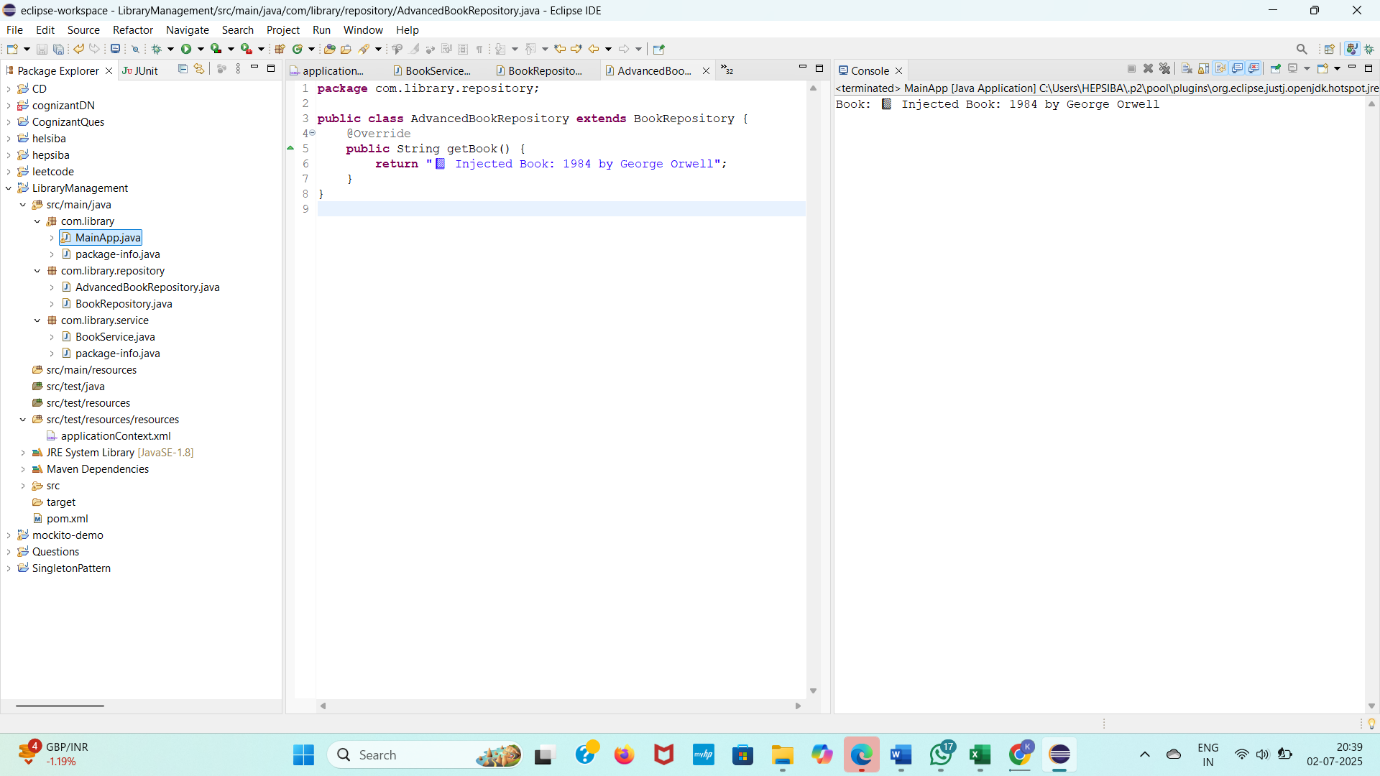
public void displayBook() {

System.*out*.println("Book: " + bookRepository.getBook());

}

}

**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.

**Book.java**

package com.library.model;

public class Book {

private String title;

private String author;

// Constructor

public Book(String title, String author) {

this.title = title;

this.author = author;

}

// Getters

public String getTitle() {

return title;

}

public String getAuthor() {

return author;

}

}

**BookRepository.java**

package com.library.repository;

import com.library.model.Book;

import java.util.ArrayList;

import java.util.List;

public class BookRepository {

public List<Book> getAllBooks() {

List<Book> books = new ArrayList<>();

books.add(new Book("1984", "George Orwell"));

books.add(new Book("To Kill a Mockingbird", "Harper Lee"));

books.add(new Book("Pride and Prejudice", "Jane Austen"));

books.add(new Book("The Hobbit", "J.R.R. Tolkien"));

return books;

}

}

**3. BookService.java**

package com.library.service;

import com.library.model.Book;

import com.library.repository.BookRepository;

import java.util.List;

public class BookService {

private BookRepository bookRepository;

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void displayAllBooks() {

List<Book> books = bookRepository.getAllBooks();

System.out.println(" Available Books in the Library:");

int i = 1;

for (Book book : books) {

System.out.println(i++ + ". " + book.getTitle() + " by " + book.getAuthor());

}

}

}

**4. MainApp.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) {

ApplicationContextcontext=newClassPathXmlApplicationContext("applicationContext.xml");

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

bookService.displayAllBooks();

}

}

**5. 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.library.repository.BookRepository" />

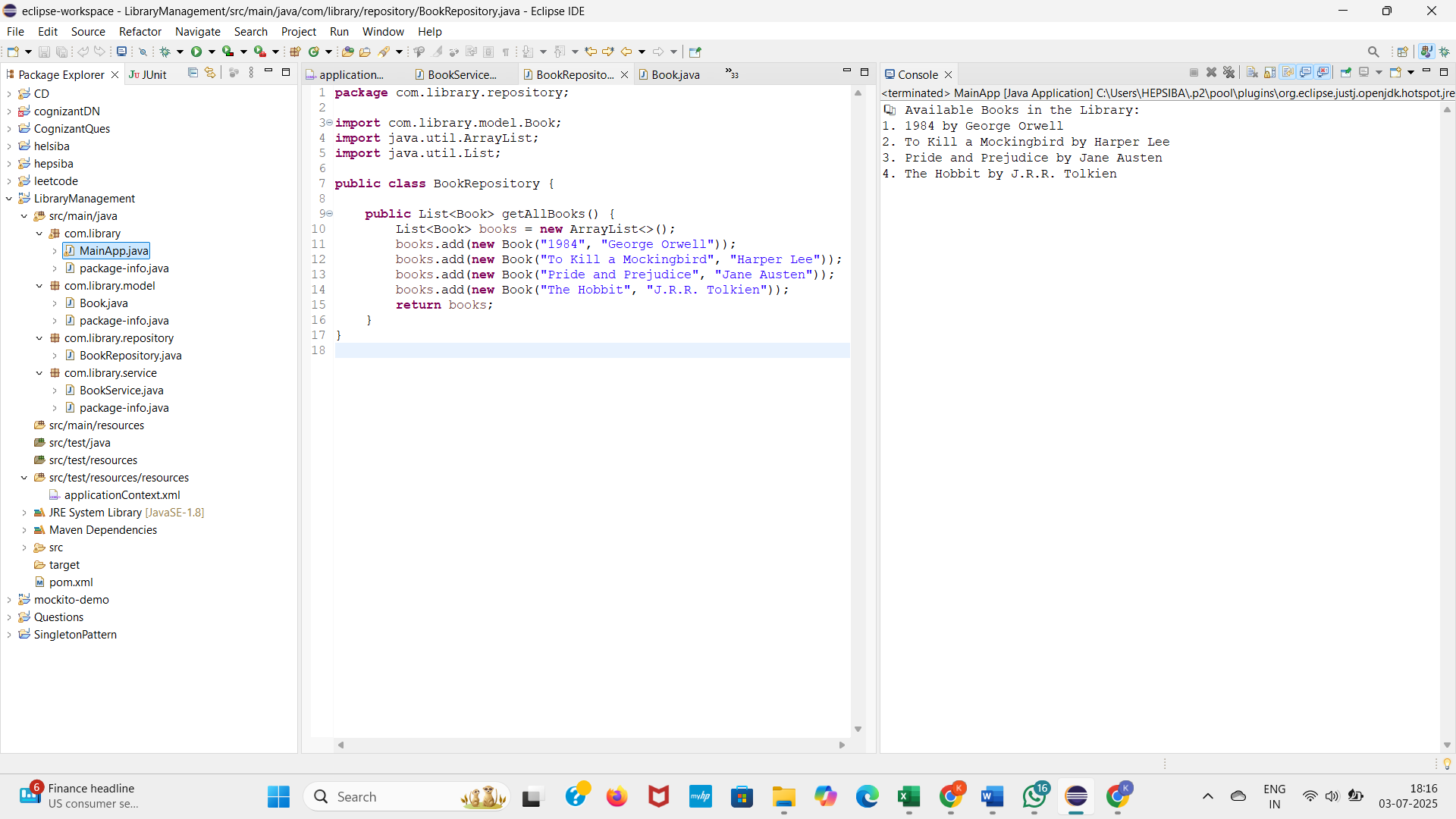
<bean id="bookService" class="com.library.service.BookService">

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

</bean>

</beans>

**OUTPUT**



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

JPA, Hibernate, and Spring Data JPA are all related to persistence in Java applications, but they operate at different levels of abstraction:

* JPA (Java Persistence API): This is a specification (a standard API) for managing relational data in Java applications. It defines how to map Java objects to database tables (Object-Relational Mapping - ORM) and how to perform CRUD (Create, Read, Update, Delete) operations.
* Hibernate: This is a concrete implementation of the JPA specification. It's a full-featured ORM framework that allows you to interact with databases using Java objects instead of raw SQL. While Hibernate fully implements JPA, it also offers additional features and capabilities that go beyond the standard JPA specification, such as various caching mechanisms, its own query language (HQL - Hibernate Query Language.
* Spring Data JPA: This is a Spring project that provides a higher-level abstraction and simplification layer on top of JPA. It significantly reduces the amount of boilerplate code you need to write for data access operations. Spring Data JPA achieves this by:
  + Repository Interfaces: It provides predefined interfaces like CrudRepository and JpaRepository that offer common CRUD operations out of the box. You just need to define your own interfaces extending these, and Spring Data JPA automatically provides the implementation.
  + Query Derivation: It can automatically generate queries based on method names. For example, a method findByUsernameAndEmail(String username, String email) will automatically be translated into the correct SQL query by Spring Data JPA.
  + Integration with Spring Framework: It seamlessly integrates with the Spring ecosystem, providing features like declarative transaction management (Transactional).

**In summary:**

* JPA is the standard.
* Hibernate is a popular implementation of that standard.
* Spring Data JPA is a further abstraction that makes using JPA (and its implementations like Hibernate) much easier and more productive, especially within a Spring application.

**Spring Data JPA - Quick Example**

package com.cognizant.ormlearn.model;

import javax.persistence.\*;

@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 + "]";

}

}

**CountryService.java**

package com.cognizant.ormlearn.repository;

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

import com.cognizant.ormlearn.model.Country;

public interface CountryRepository extends JpaRepository<Country, String> {

}

package com.cognizant.ormlearn.service;

import java.util.List;

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

import org.springframework.stereotype.Service;

import com.cognizant.ormlearn.model.Country;

import com.cognizant.ormlearn.repository.CountryRepository;

@Service

public class CountryService {

@Autowired

private CountryRepository countryRepository;

public List<Country> getAllCountries() {

return countryRepository.findAll();

}

}

**MainApp.java**

package com.cognizant.ormlearn;

import com.cognizant.ormlearn.service.CountryService;

import com.cognizant.ormlearn.model.Country;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ApplicationContext;

import java.util.List;

@SpringBootApplication

public class OrmLearnApplication {

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() {

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

countries.forEach(System.out::println);

}

}

**SQL**

CREATE DATABASE ormlearn;

USE ormlearn;

CREATE TABLE country (

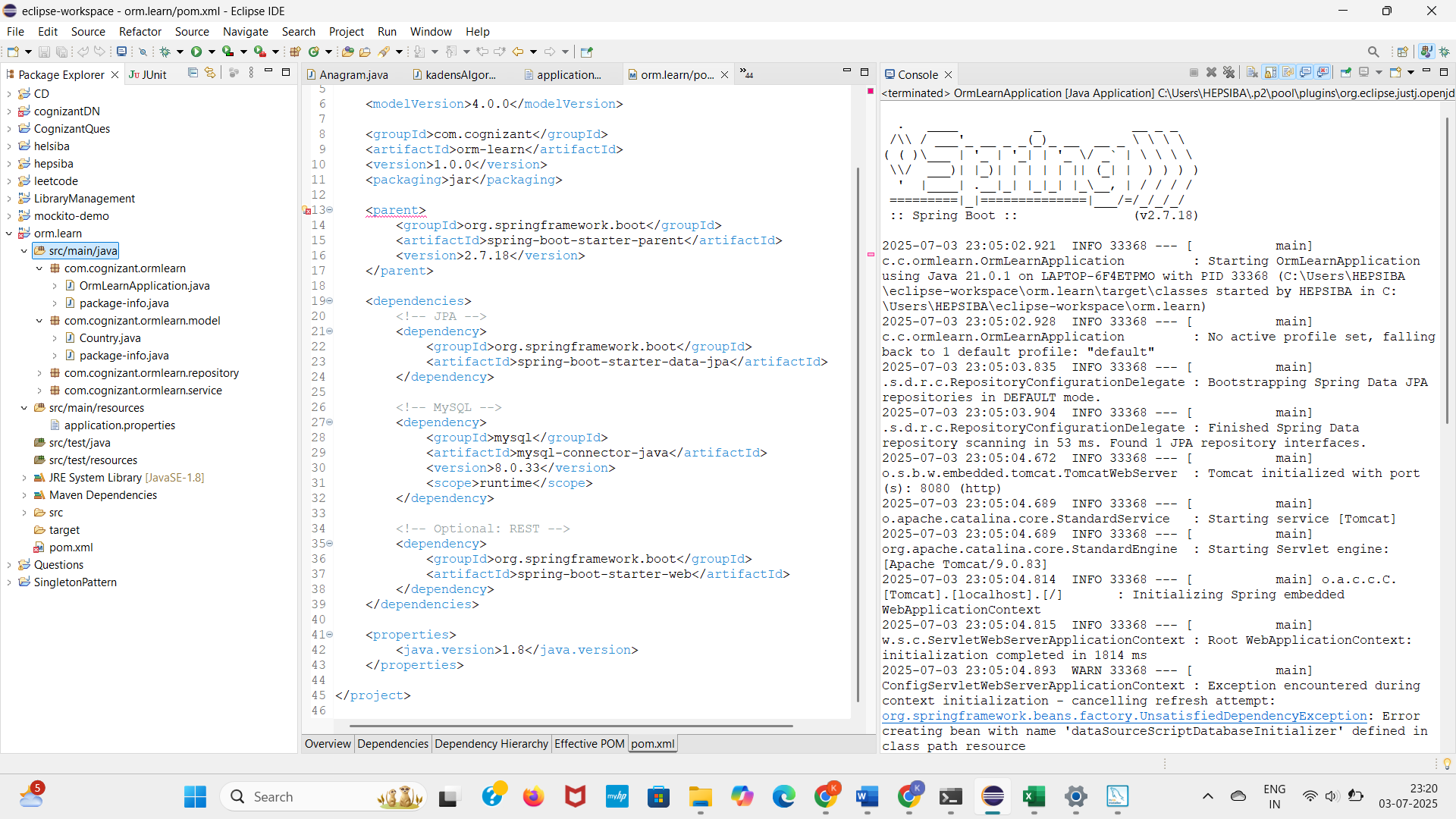
co\_code VARCHAR(2) PRIMARY KEY,

co\_name VARCHAR(50));

INSERT INTO country VALUES ('IN', 'India');

INSERT INTO country VALUES ('US', 'United States of America');

**OUTPUT**



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

**Scenario:**

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

**Application.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.library.repository.BookRepository" />

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

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

</bean>

</beans>

**BookRepository.java**

package com.library.repository;

import com.library.model.Book;

import java.util.\*;

public class BookRepository {

public List<Book> getAllBooks() {

List<Book> books = new ArrayList<>();

books.add(new Book("1984", "George Orwell"));

books.add(new Book("To Kill a Mockingbird", "Harper Lee"));

books.add(new Book("Pride and Prejudice", "Jane Austen"));

books.add(new Book("The Hobbit", "J.R.R. Tolkien"));

return books;

}

}

**BookService.java**

package com.library.service;

import com.library.model.Book;

import com.library.repository.BookRepository;

import java.util.List;

public class BookService {

private BookRepository bookRepository;

// Setter for DI

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public List<Book> getAllBooks() {

return bookRepository.getAllBooks();

}

}

**Book.java**

package com.library.model;

public class Book {

private String title;

private String author;

public Book(String title, String author) {

this.title = title;

this.author = author;

}

public String getTitle() { return title; }

public String getAuthor() { return author; }

@Override

public String toString() {

return title + " by " + author;

}

}

**OUTPUT**

