**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:**

**POX.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>5.3.32</version>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>exec-maven-plugin</artifactId>

<version>3.1.0</version>

<configuration>

<mainClass>com.library.MainApp</mainClass>

</configuration>

</plugin>

</plugins>

</build>

</project>

**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) {

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

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

service.displayBooks();

}}

**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 displayBooks() {

System.out.println("BookService: Displaying books...");

bookRepository.printData();

}

}

**BookRepository.java:**

package com.library.repository;

public class BookRepository {

public void printData() {

System.out.println("Fetching book data from the repository...");

}

}

**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"/>

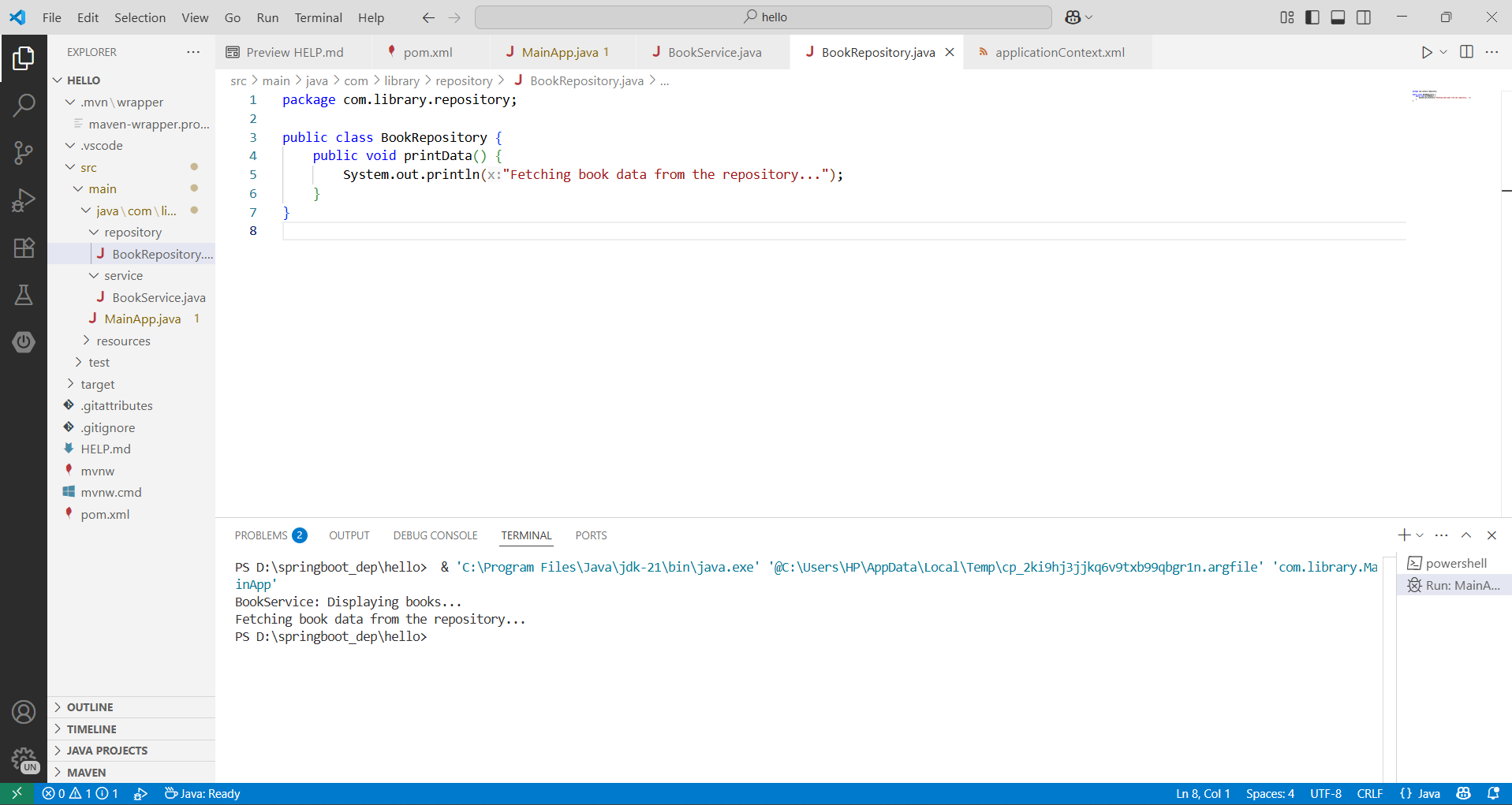
<bean id="bookService" class="com.library.service.BookService">

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

</bean>

</beans>

**OUTPUT:**



**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:**

**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>5.3.32</version>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>exec-maven-plugin</artifactId>

<version>3.1.0</version>

<configuration>

<mainClass>com.library.MainApp</mainClass>

</configuration>

</plugin>

</plugins>

</build>

</project>

**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 for BookRepository -->

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

<!-- Bean for BookService with DI -->

<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 printData() {

System.out.println("Fetching book data from the repository...");

}

}

**BookService.java:**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

// Setter for Spring to inject dependency

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void displayBooks() {

System.out.println("BookService: Displaying books...");

bookRepository.printData();

}

}

**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) {

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

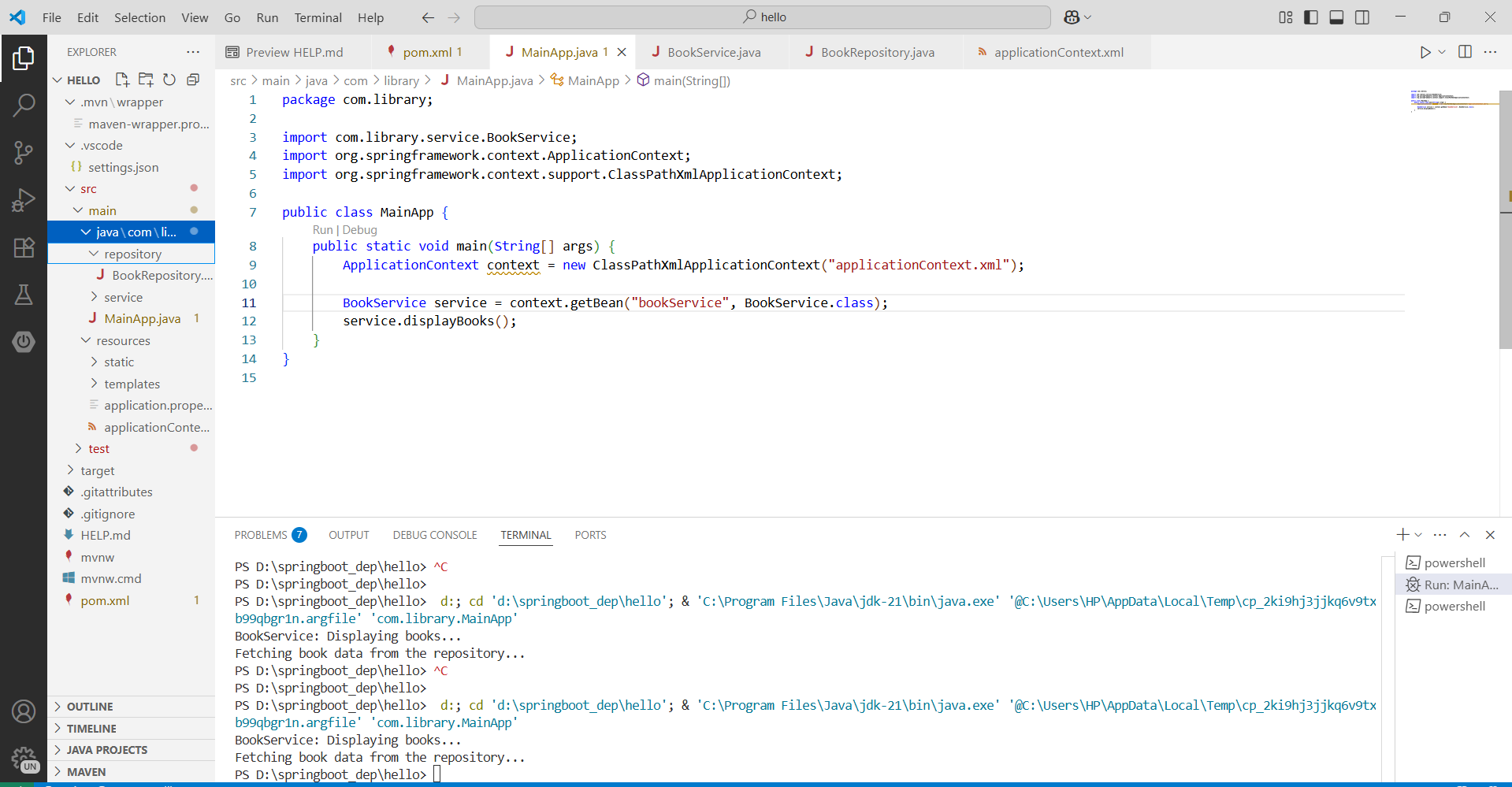
BookService service = context.getBean("bookService", BookService.class);

service.displayBooks();

}

}

**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>hello</artifactId>

<version>1.0.0</version>

<packaging>jar</packaging>

<parent>

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

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

<version>3.2.4</version>

<relativePath/>

</parent>

<dependencies>

<dependency>

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

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

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

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

</plugin>

</plugins>

</build>

</project>

MainApplication.java:

package com.library;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class MainApplication {

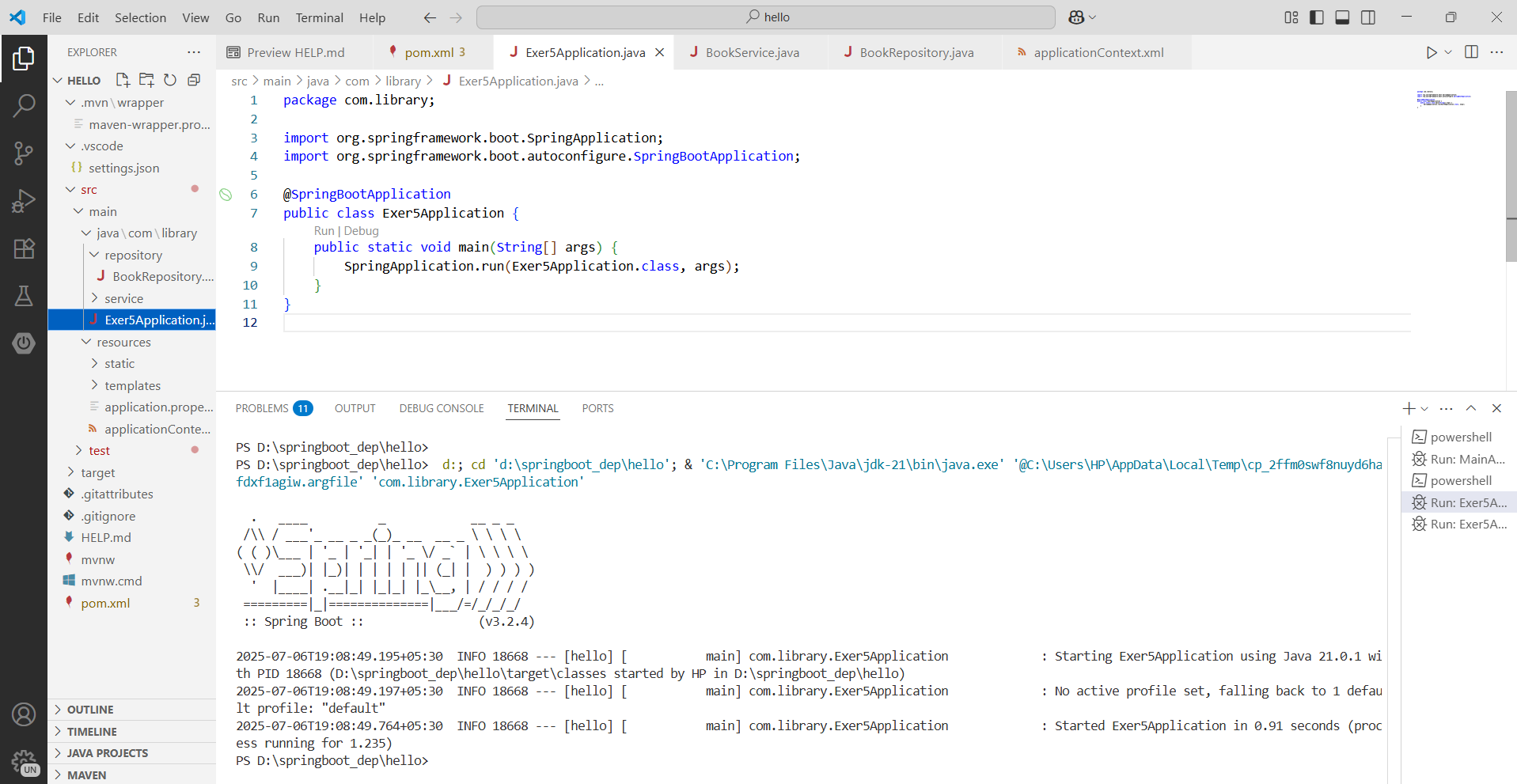
public static void main(String[] args) {

SpringApplication.run(Exer5Application.class, args);

}

}

**OUTPUT:**



**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:**

**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>1.0.0</version>

<dependencies>

<!-- Spring Context for core IoC -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.32</version>

</dependency>

</dependencies>

<build>

<plugins>

<!-- Java compiler plugin -->

<plugin>

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

<version>3.8.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

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>**

**BookRepository.java:**

package com.library.repository;

public class BookRepository {

public void display() {

System.out.println("BookRepository: Fetching books...");

}

}

**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 showBooks() {

System.out.println("BookService: Showing books.");

bookRepository.display();

}

}

**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) {

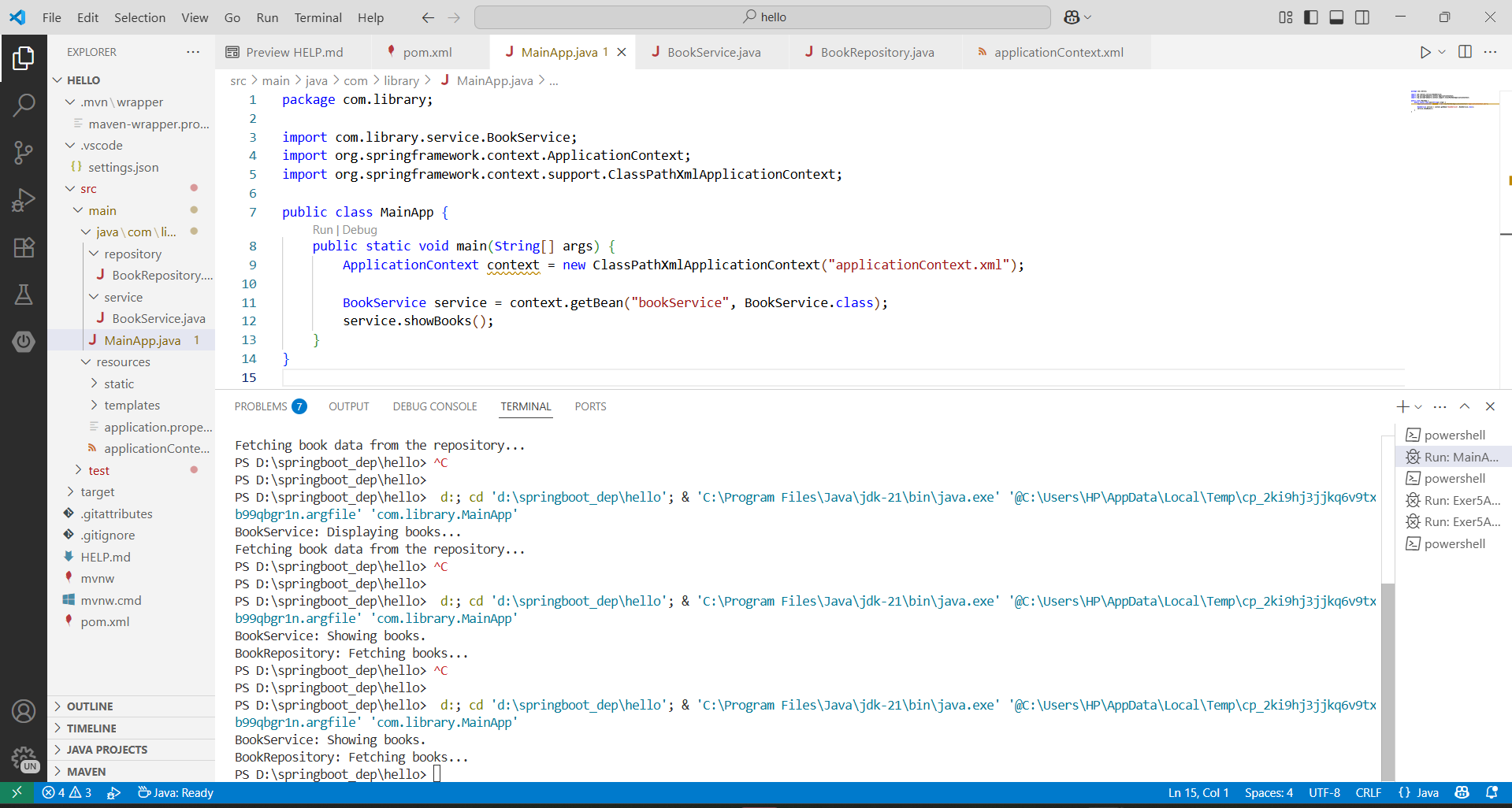
ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

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

service.showBooks();

}

}



**Exercise 7: Implementing Constructor and Setter Injection**

**Scenario:**

The library management application requires both constructor and setter injection for better control over bean initialization.

**Steps:**

1. **Configure Constructor Injection:**
   * Update applicationContext.**xml** to configure constructor injection for **BookService**.
2. **Configure Setter Injection:**
   * Ensure that the **BookService** class has a setter method for **BookRepository** and configure it in **applicationContext.xml**.
3. **Test the Injection:**
   * Run the **LibraryManagementApplication** main class to verify both constructor and setter injection.

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>1.0.0</version>

<dependencies>

<!-- Spring Context for core IoC -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.32</version>

</dependency>

</dependencies>

<build>

<plugins>

<!-- Java compiler plugin -->

<plugin>

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

<version>3.8.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

BookRepository.java:

package com.library.repository;

public class BookRepository {

public void displayBooks() {

System.out.println("BookRepository: Listing all books.");

}

}

**BookService.java:**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private String message;

private BookRepository bookRepository;

// Constructor for constructor injection

public BookService(String message) {

this.message = message;

}

// Setter for setter injection

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void showInfo() {

System.out.println("BookService Message: " + message);

bookRepository.displayBooks();

}

}

**MainApp.java:**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private String message;

private BookRepository bookRepository;

// Constructor for constructor injection

public BookService(String message) {

this.message = message;

}

// Setter for setter injection

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void showInfo() {

System.out.println("BookService Message: " + message);

bookRepository.displayBooks();

}

}

**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 Constructor + Setter Injection -->

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

<!-- Constructor injection -->

<constructor-arg value="Welcome to Library Service!" />

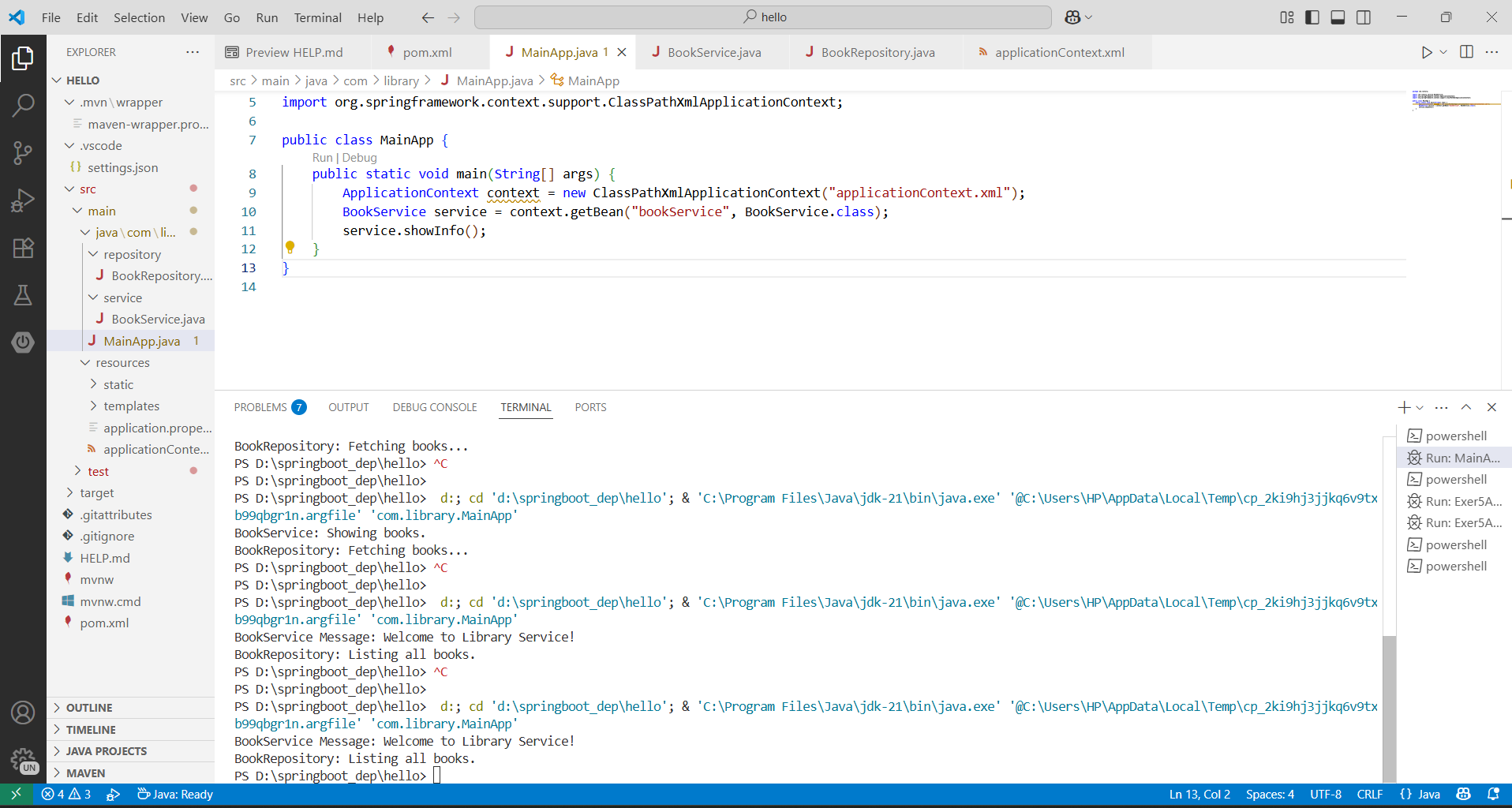
<!-- Setter injection -->

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

</bean>

</beans>

**OUTPUT:**



**SPRING DATA JPA AND HIBERATE**

**1.Spring Data JPA - Quick Example**

**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.cognizant</groupId>

<artifactId>orm-learn</artifactId>

<version>1.0.0</version>

<packaging>jar</packaging>

<name>ORM Learn</name>

<parent>

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

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

<version>3.2.5</version>

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

</parent>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

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

</plugin>

</plugins>

</build>

</project>

**Country.java:**

package com.cognizant.orm\_learn.model;

import jakarta.persistence.\*;

@Entity

@Table(name = "country")

public class Country {

@Id

@Column(name = "co\_code")

private String code;

@Column(name = "co\_name")

private String name;

// Getters and setters

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;

}

// toString

@Override

public String toString() {

return "Country{" + "code='" + code + '\'' + ", name='" + name + '\'' + '}';

}

}

**CountryRepository.java:**

package com.cognizant.orm\_learn.repository;

import com.cognizant.orm\_learn.model.Country;

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

public interface CountryRepository extends JpaRepository<Country, String> {

}

OrmLearnApplication.java:

package com.cognizant.orm\_learn;

import com.cognizant.orm\_learn.model.Country;

import com.cognizant.orm\_learn.repository.CountryRepository;

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

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import java.util.List;

@SpringBootApplication

public class OrmLearnApplication implements CommandLineRunner {

@Autowired

private CountryRepository countryRepository;

public static void main(String[] args) {

SpringApplication.run(OrmLearnApplication.class, args);

}

@Override

public void run(String... args) {

List<Country> countries = countryRepository.findAll();

System.out.println("Countries from DB:");

for (Country country : countries) {

System.out.println(country);

}

}

}

**application.properties:**

spring.datasource.url=jdbc:h2:mem:testdb

spring.datasource.driver-class-name=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=

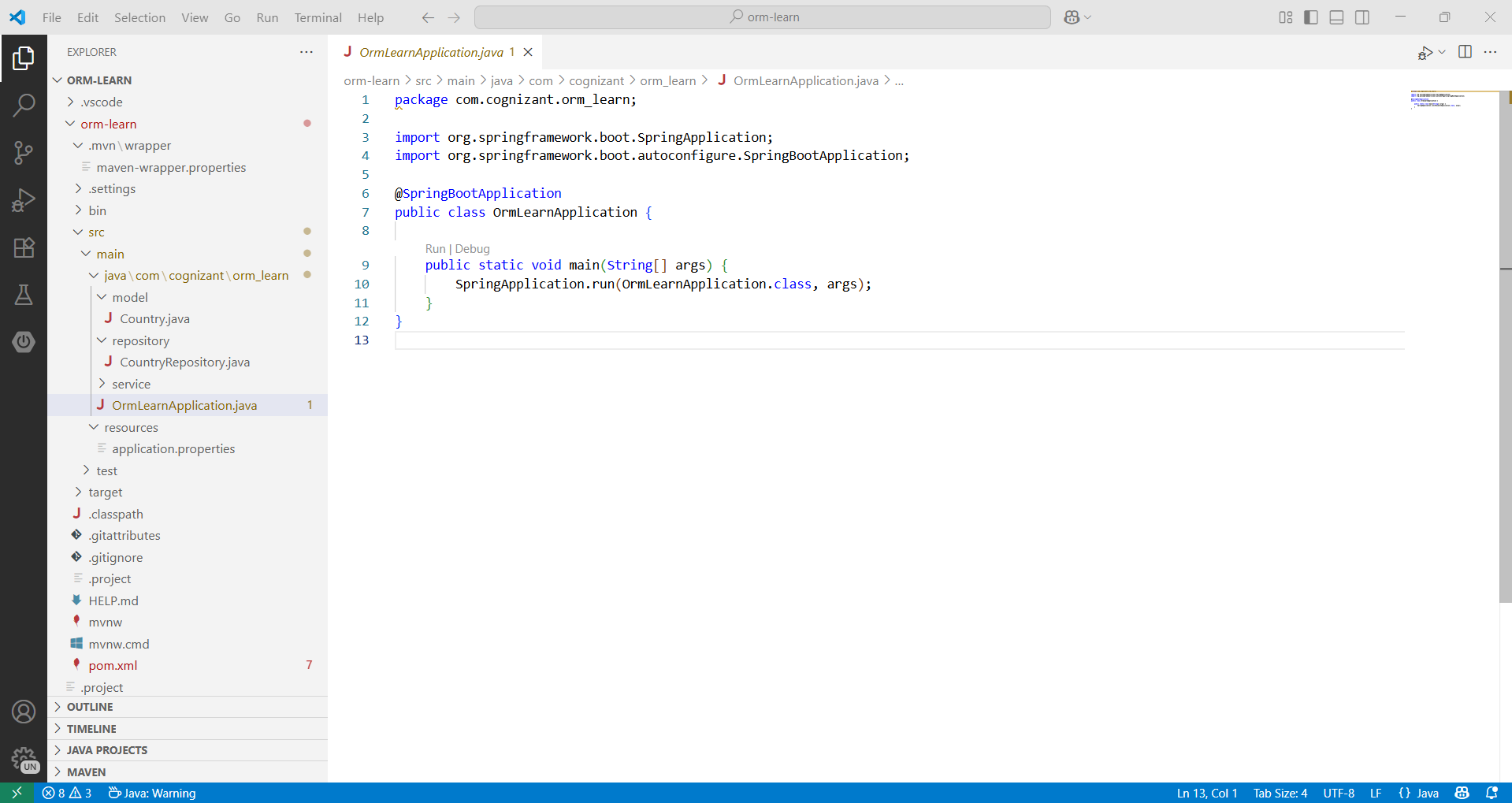
spring.jpa.hibernate.ddl-auto=create

spring.jpa.show-sql=true

# Optional: Initialize with sample data

spring.sql.init.mode=always

**OUTPUT:**



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

**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.example</groupId>

<artifactId>employee-demo</artifactId>

<version>1.0.0</version>

<packaging>jar</packaging>

<parent>

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

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

<version>3.2.5</version>

</parent>

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

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

</plugin>

</plugins>

</build>

</project>

**Employee.java:**

package com.example.employeedemo.model;

import jakarta.persistence.\*;

@Entity

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Integer id;

private String name;

private String department;

private double salary;

public Integer getId() {

return id;

}

public void setId(Integer id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getDepartment() {

return department;

}

public void setDepartment(String department) {

this.department = department;

}

public double getSalary() {

return salary;

}

public void setSalary(double salary) {

this.salary = salary;

}

@Override

public String toString() {

return "Employee{id=" + id + ", name='" + name + '\'' +

", department='" + department + '\'' + ", salary=" + salary + '}';

}

}

**EmployeeRepository.java:**

package com.example.employeedemo.repository;

import com.example.employeedemo.model.Employee;

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

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

**EmployeeService.java:**

package com.example.employeedemo.service;

import com.example.employeedemo.model.Employee;

import com.example.employeedemo.repository.EmployeeRepository;

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

import org.springframework.stereotype.Service;

import org.springframework.transaction.annotation.Transactional;

import java.util.List;

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

public List<Employee> getAllEmployees() {

return employeeRepository.findAll();

}

}

**EmployeeDemoApplication.java:**

package com.example.employeedemo;

import com.example.employeedemo.model.Employee;

import com.example.employeedemo.service.EmployeeService;

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

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class EmployeeDemoApplication implements CommandLineRunner {

@Autowired

private EmployeeService employeeService;

public static void main(String[] args) {

SpringApplication.run(EmployeeDemoApplication.class, args);

}

@Override

public void run(String... args) {

// Add sample employees

Employee emp1 = new Employee();

emp1.setName("John");

emp1.setDepartment("HR");

emp1.setSalary(50000);

Employee emp2 = new Employee();

emp2.setName("Jane");

emp2.setDepartment("IT");

emp2.setSalary(70000);

employeeService.addEmployee(emp1);

employeeService.addEmployee(emp2);

// Print all employees

System.out.println("All Employees:");

for (Employee emp : employeeService.getAllEmployees()) {

System.out.println(emp);

}

}

}

**application.properties:**

spring.datasource.url=jdbc:h2:mem:testdb

spring.datasource.driver-class-name=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=

spring.jpa.hibernate.ddl-auto=create

spring.jpa.show-sql=true