# Week 3 – Spring Core and Maven Hands-On

## 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.

Code:

## 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;  
  
 public void setBookRepository(BookRepository bookRepository) {  
 this.bookRepository = bookRepository;  
 }  
  
 public void displayBooks() {  
 bookRepository.display();  
 }  
}

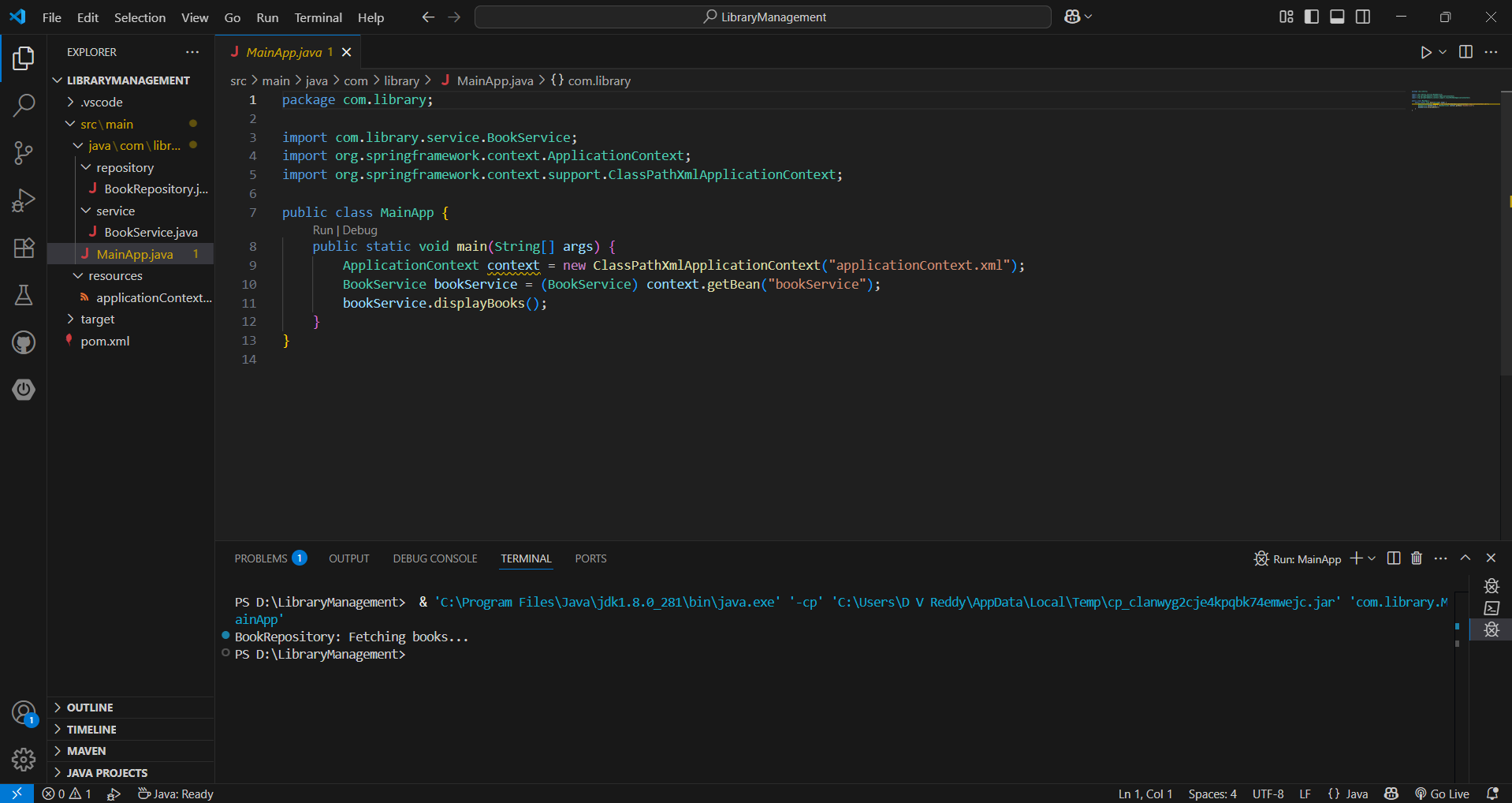
## MainApp.java

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

## applicationContext.xml

<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.

Dependency Injection is demonstrated in the BookService class via the setter method and configured in applicationContext.xml using the <property> tag.

Code:

**BookService.java**

java

CopyEdit

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

// Setter Injection

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void displayBooks() {

bookRepository.display();

}

}

**applicationContext.xml**

xml

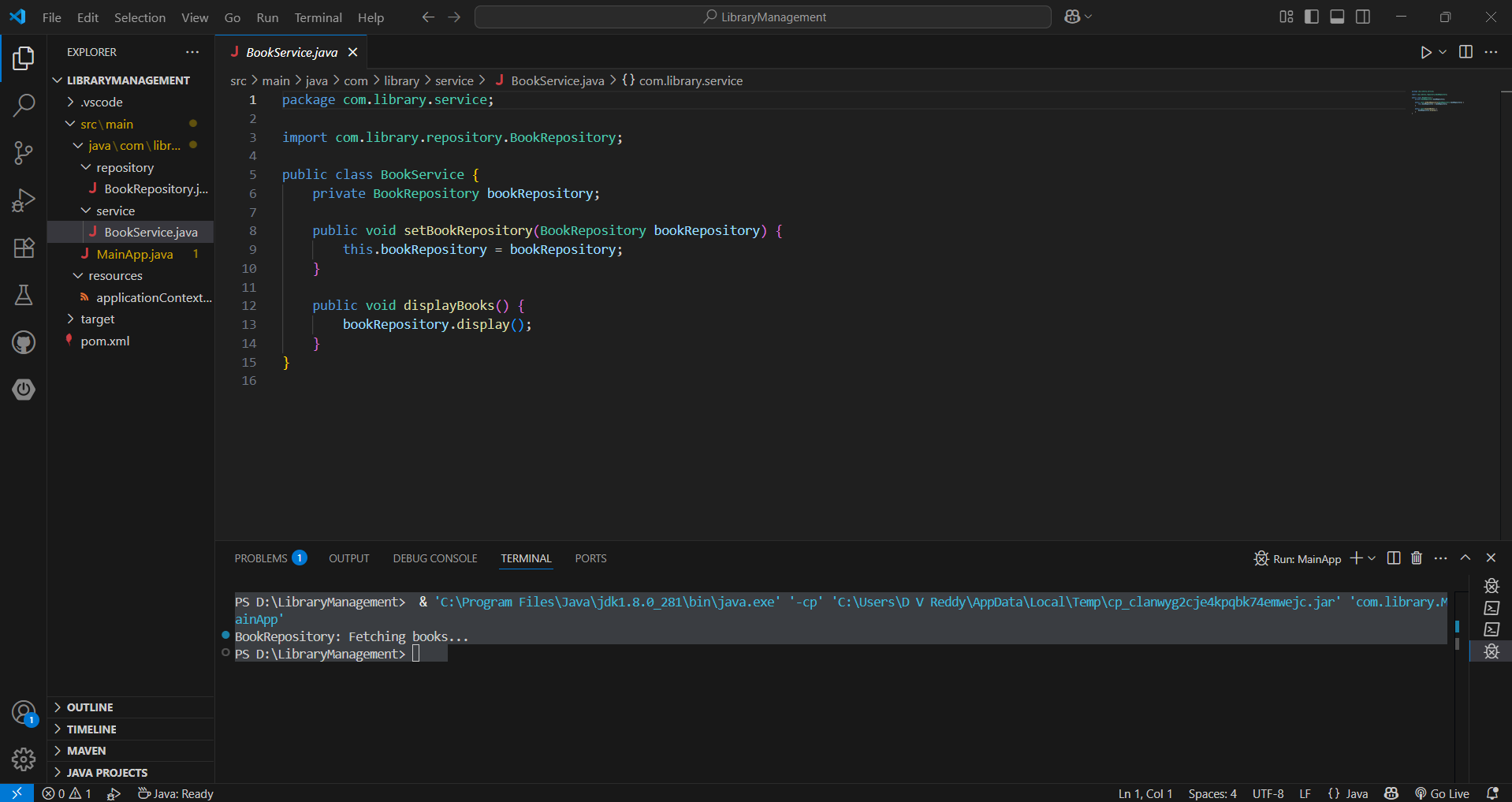
CopyEdit

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

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

</bean>

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.

## 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>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>5.3.25</version>  
 </dependency>  
 </dependencies>  
</project>

Output:

