**Spring Core Maven Exercises**

# Exercise 1: Configuring a Basic Spring Application

To build a simple backend system for a library using **Spring Framework (Core, XML-based)**. It demonstrates how to define and wire service and repository layers using **Spring's Dependency Injection (DI)** mechanism.

## 1. Set Up a Spring Project

Create a Maven project named LibraryManagement and add the following Spring Core dependencies in pom.xml:

<dependencies>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>5.3.30</version>  
 </dependency>  
</dependencies>

## 2. Configure the Application Context

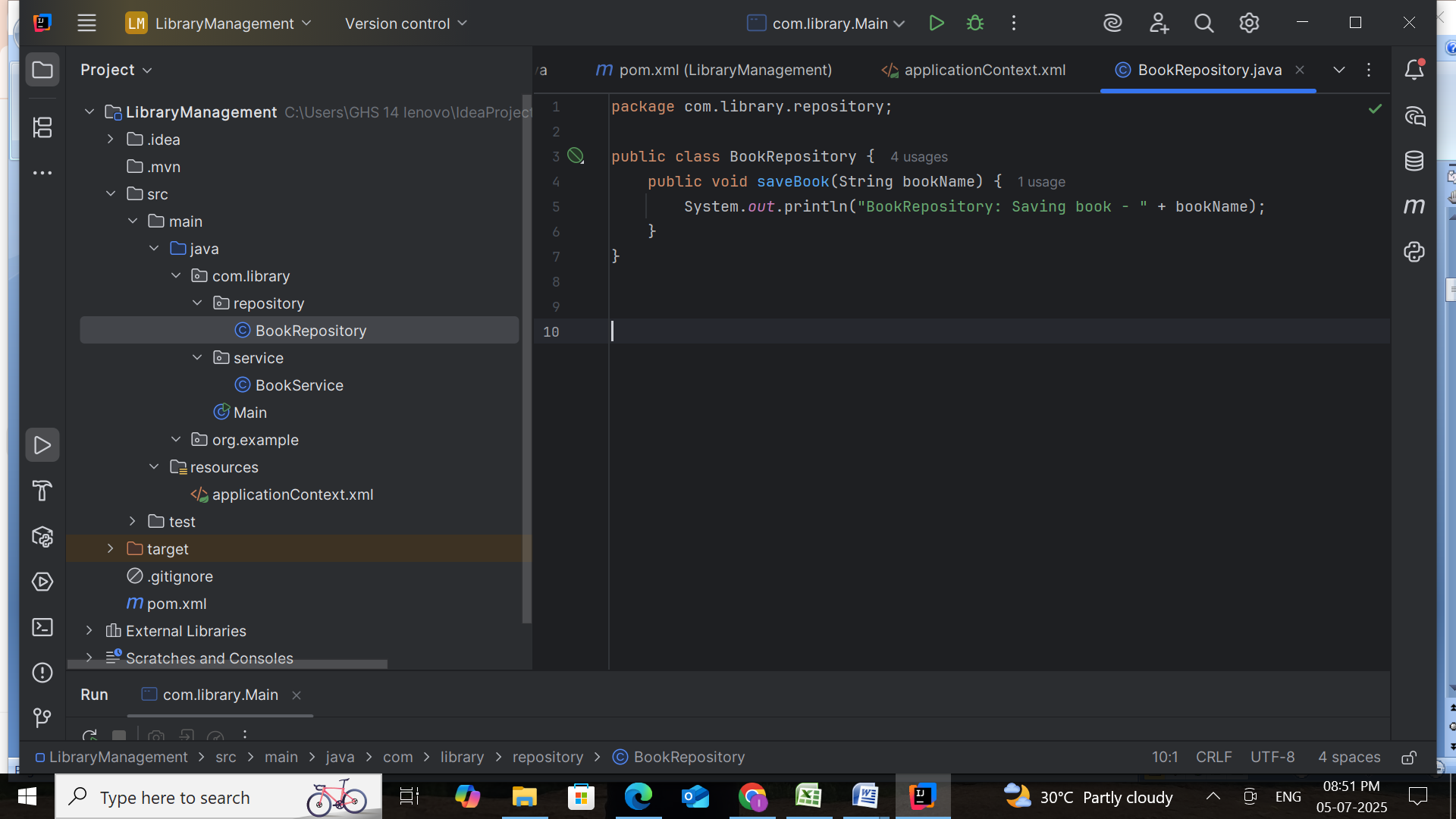
Create an XML configuration file named applicationContext.xml under src/main/resources with the following:

<?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>

## 3. Define Service and Repository Classes

**BookRepository.java**

package com.library.repository;  
  
public class BookRepository {  
 public void saveBook(String bookName) {  
 System.*out*.println("BookRepository: Saving book - " + bookName);  
 }  
}



**BookService.java**

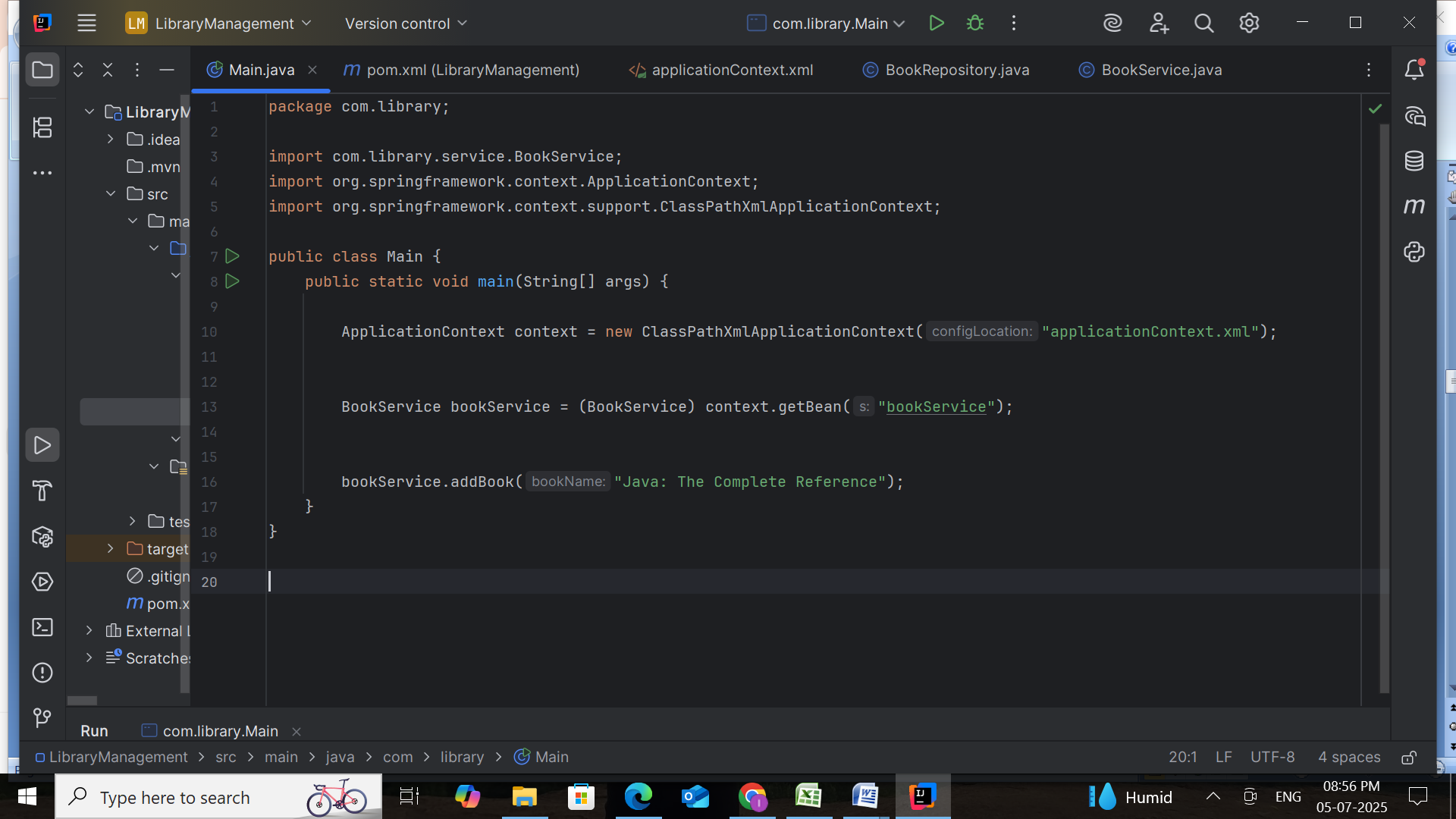
package com.library.service;  
  
import com.library.repository.BookRepository;  
  
public class BookService {  
 private BookRepository bookRepository;  
  
   
 public void setBookRepository(BookRepository bookRepository) {  
 this.bookRepository = bookRepository;  
 }  
  
 public void addBook(String bookName) {  
 System.*out*.println("BookService: Adding book - " + bookName);  
 bookRepository.saveBook(bookName);  
 }  
}

## 

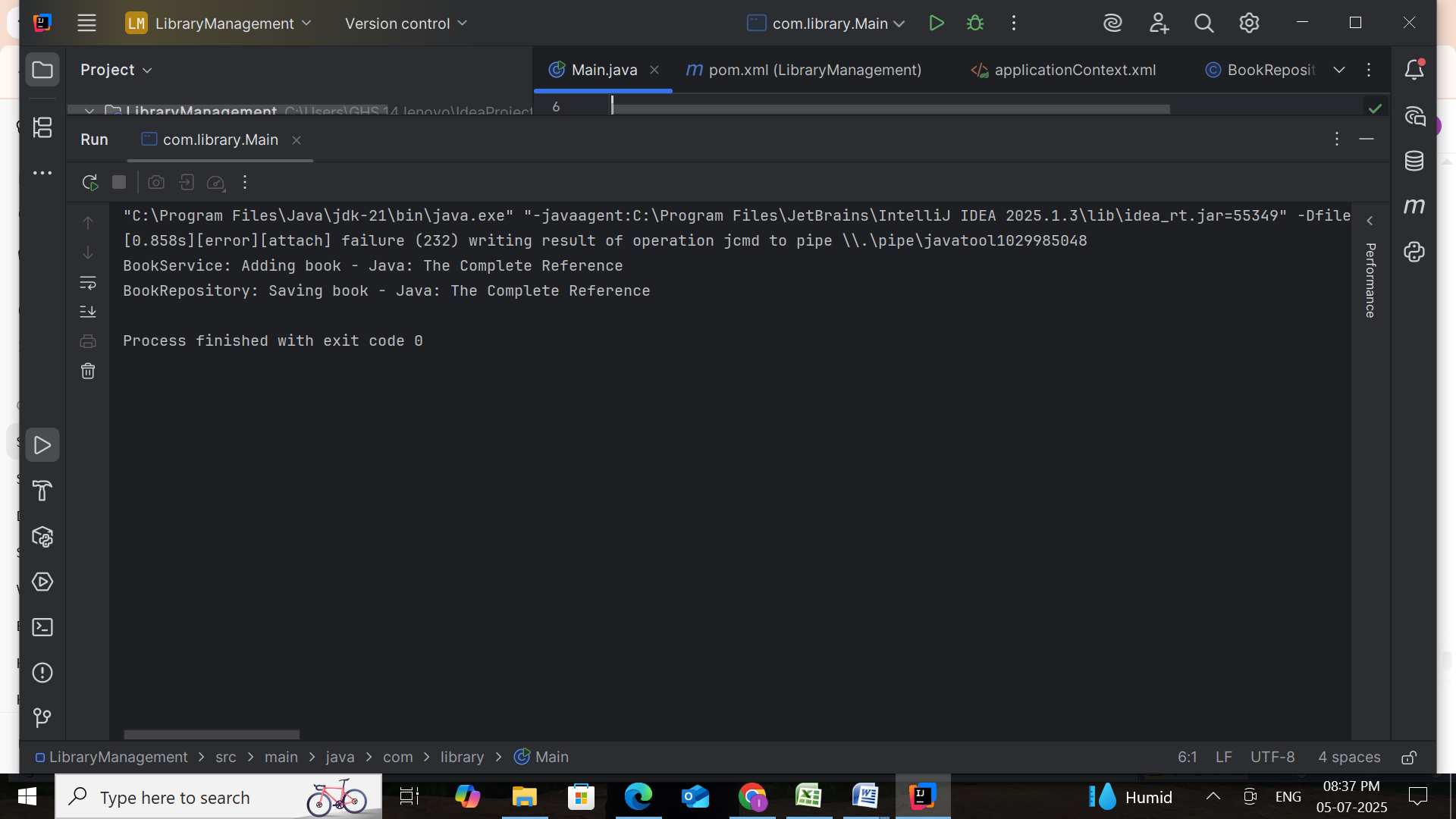
## 4. Run the Application

**Main.java**

package com.library;  
  
import com.library.service.BookService;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
  
public class Main {  
 public static void main(String[] args) {  
   
 ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");  
  
   
 BookService bookService = (BookService) context.getBean("bookService");  
  
   
 bookService.addBook("Java: The Complete Reference");  
 }  
}



## 5. Output:



# Exercise 2: Implementing Dependency Injection

In a library management application, classes like BookService depend on others like BookRepository. Instead of manually creating objects, we use Spring's Dependency Injection to automatically manage and inject these dependencies, improving modularity and maintainability.

## 1. Modify the XML Configuration

Update applicationContext.xml to inject BookRepository into BookService using setter injection:

<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>

## 2. Update the BookService Class

To allow Spring to inject the BookRepository dependency, the BookService class must provide a setter method named setBookRepository. Spring will use this method to inject the dependency during runtime.

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 bookName) {  
 System.*out*.println("BookService: Adding book - " + bookName);  
 bookRepository.saveBook(bookName);  
 }  
}

## 3. Test the Configuration

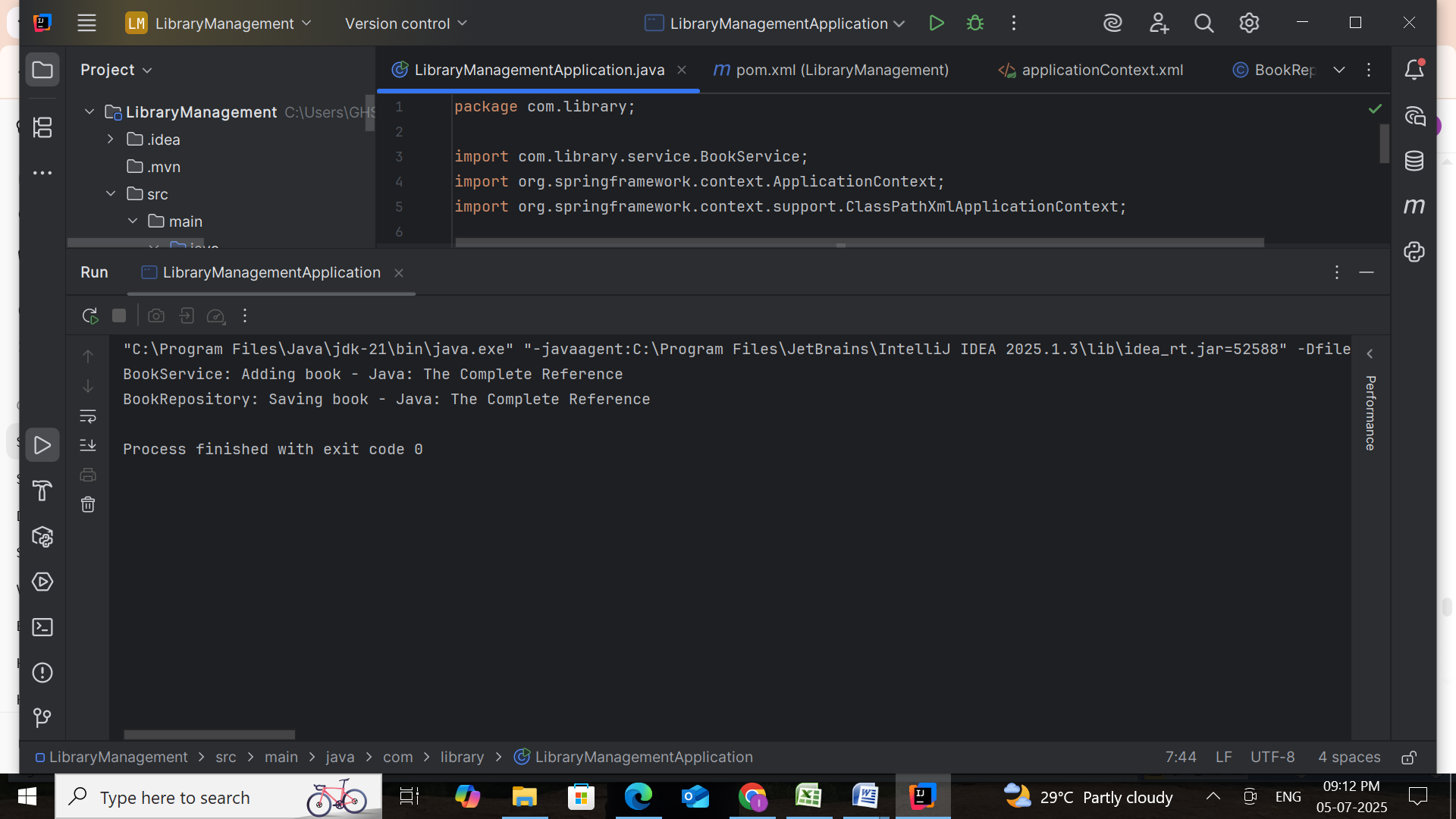
Create or update the main class to load the Spring context and verify that the dependency injection works correctly.

package com.library;  
  
import com.library.service.BookService;  
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 = (BookService) context.getBean("bookService");

bookService.addBook("Java: The Complete Reference");  
 }  
}

**4.Output:**



Exercise 4: Creating and Configuring a Maven Project

To build and manage the library management application's backend efficiently, Maven is used as the build automation and dependency management tool. This exercise involves creating a new Maven project named LibraryManagement and integrating key Spring Framework modules. Additionally, the project will be configured to compile with Java version 1.8 using the Maven Compiler Plugin.

## 1. Create a Maven Project

* Open your IDE (e.g., IntelliJ IDEA).
* Create a new **Java project**.
* Under **Build System**, select **Maven**.
* Fill in the project details:
  + **Project Name**: LibraryManagement
  + **Group ID**: com.library
  + **Artifact ID**: LibraryManagement
* Finish the project setup. This creates a basic Maven structure including the pom.xml file.

## 2. Add Spring Dependencies in pom.xml

Open the pom.xml file and add the following Spring dependencies inside the <dependencies> section:

<dependencies>  
 <!-- Spring Context (for core DI functionality) -->  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>5.3.20</version>  
 </dependency>  
  
 <!-- Spring AOP (for aspect-oriented programming support) -->  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-aop</artifactId>  
 <version>5.3.20</version>  
 </dependency>  
  
 <!-- Spring WebMVC (for web and controller layer support) -->  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-webmvc</artifactId>  
 <version>5.3.20</version>  
 </dependency>  
</dependencies>

## 3. Configure Maven Plugins

To ensure your project uses Java 1.8, configure the Maven Compiler Plugin by adding this after the <dependencies> section in pom.xml:

<build>  
 <plugins>  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <version>3.8.1</version>  
 <configuration>  
 <source>1.8</source>  
 <target>1.8</target>  
 </configuration>  
 </plugin>  
 </plugins>  
</build>

