**Spring Core & Maven Assignment**

Name: Divyansh Tiwari

Superset ID: 6394658

**Exercise 1: Configuring a Basic Spring Application**

**Objective:**

Set up a basic Spring application using Maven, define service and repository beans, and configure them via XML.

Steps**:**

1. **Setting Up a Spring Project:**

Created Maven Project: Named LibraryManagement.

Added Dependencies in pom.xml:

*<dependencies>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>5.3.27</version>  
 </dependency>  
</dependencies>*

1. **Configuring the Application Context:**

Created an XML configuration file named **applicationContext.xml** in the **src/main/resources** directory.

CODE:

*<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>*

Defined beans for **BookService** and **BookRepository** in the XML file.

1. **Define Service and Repository Classes:**

Created a package **com.library.service** and added a class **BookService**.

In BookService Class:

*package com.library.service;  
import com.library.repository.BookRepository;*

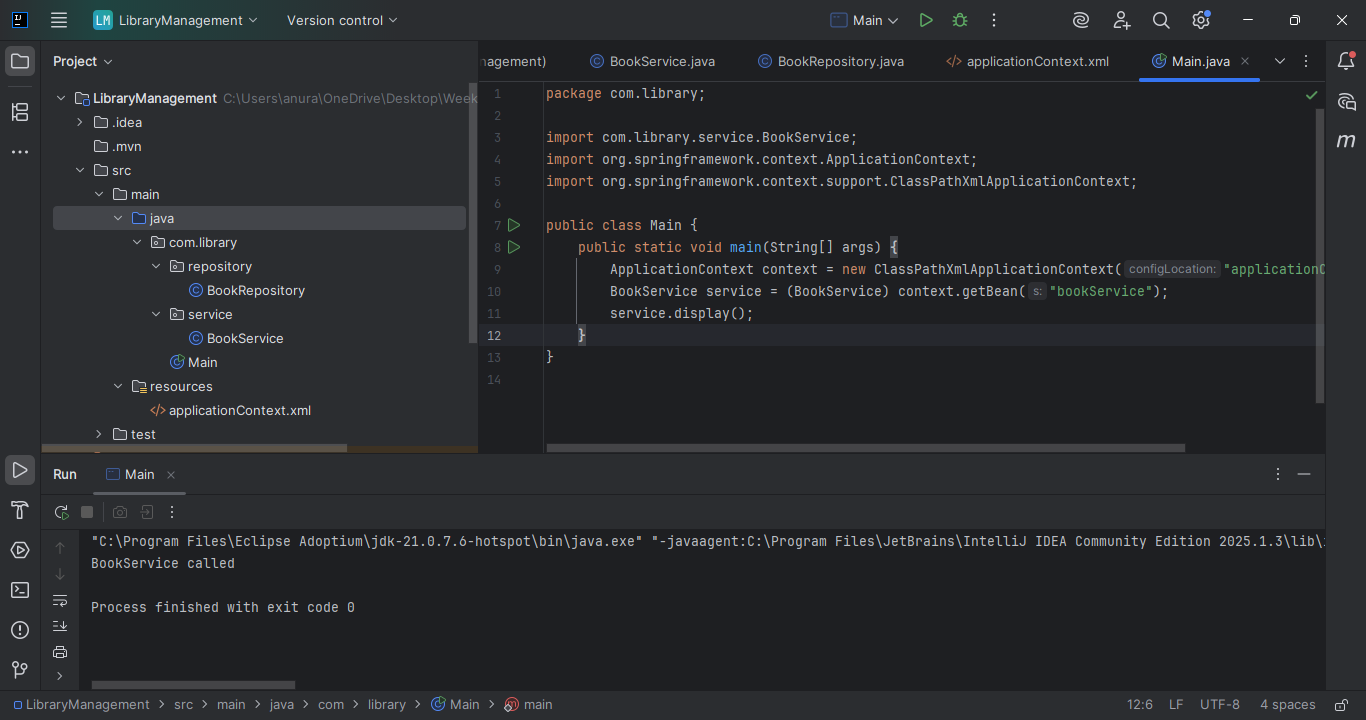
*public class BookService {  
 private BookRepository bookRepository;  
 public void setBookRepository(BookRepository bookRepository) {  
 this.bookRepository = bookRepository;  
 }  
 public void addBook() {  
 bookRepository.save();  
 }  
}*

Created a package **com.library.repository** and add a class **BookRepository**.

In BookRepository Class:

*package com.library.repository;  
  
public class BookRepository {  
 public void save() {  
 System.out.println("Book saved!");  
 }  
}*

1. **Run the Application: OUTPUT:**



**Exercise 2: Implementing Dependency Injection**

**Objective:**

Wire beans using Spring’s Dependency Injection (DI) with XML configuration.

Steps**:**

1. **Modifying the XML Configuration:**

Updated **applicationContext.xml** to wire **BookRepository** into **BookService**. This was done as follows in the previous exercise.

*<bean id="bookRepository" class="com.library.repository.BookRepository"/>  
 <bean id="bookService" class="com.library.service.BookService">  
 <property name="bookRepository" ref="bookRepository"/>  
 </bean>  
</beans>*

1. **Updating the BookService Class:**

Ensure that **BookService** class has a setter method for **BookRepository**.

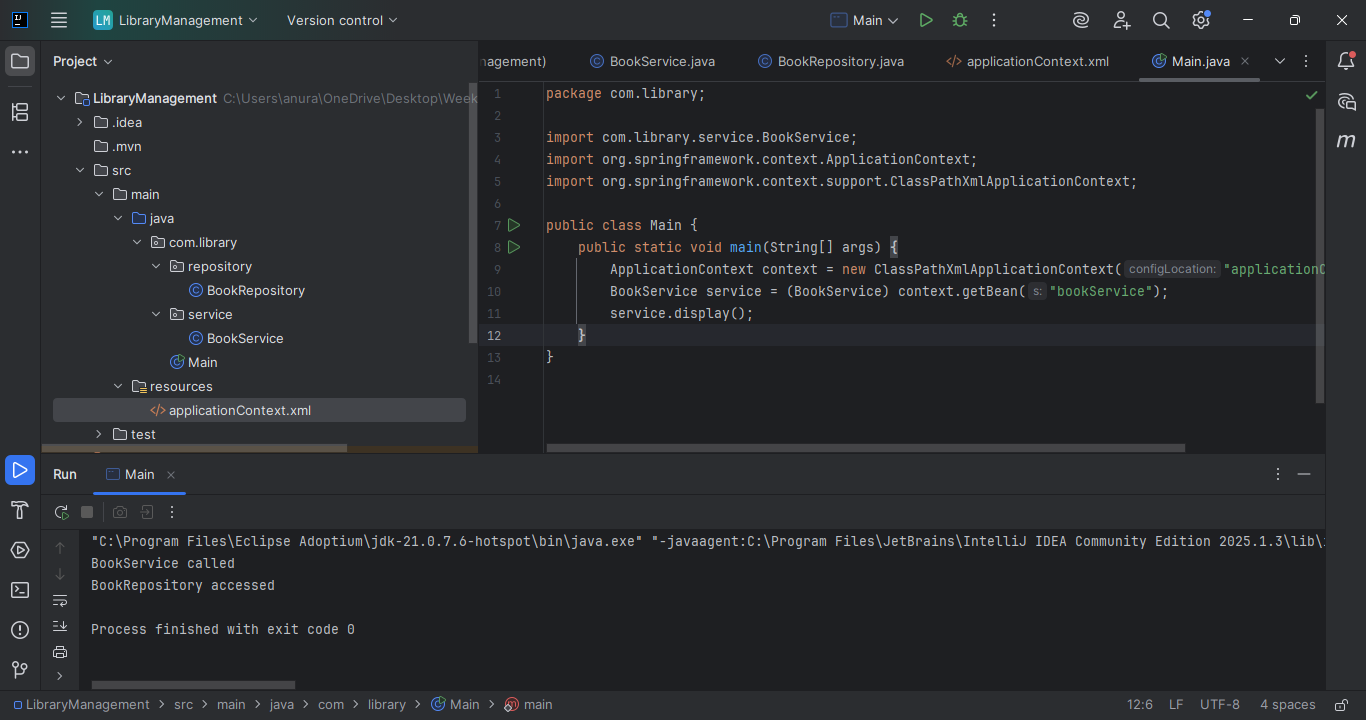
In BookService Class:

*public void setBookRepository(BookRepository bookRepository) {  
 this.bookRepository = bookRepository;  
 }*

1. **Testing the Configuration:**

Run the **LibraryManagementApplication** main class to verify the dependency injection.

OUTPUT:



**Exercise 3: Implementing Logging with Spring AOP**

**Objective:**

Add logging to track method execution using Spring AOP.

**Steps:**

1. **Added Spring AOP Dependency:**

Updated **pom.xml** to include Spring AOP dependency.

*<dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-aop</artifactId>  
 <version>5.3.27</version>  
</dependency>*

1. **Created an Aspect for Logging:**

Create a package **com.library.aspect** and add a class **LoggingAspect** with a method to log execution times.

CODE:

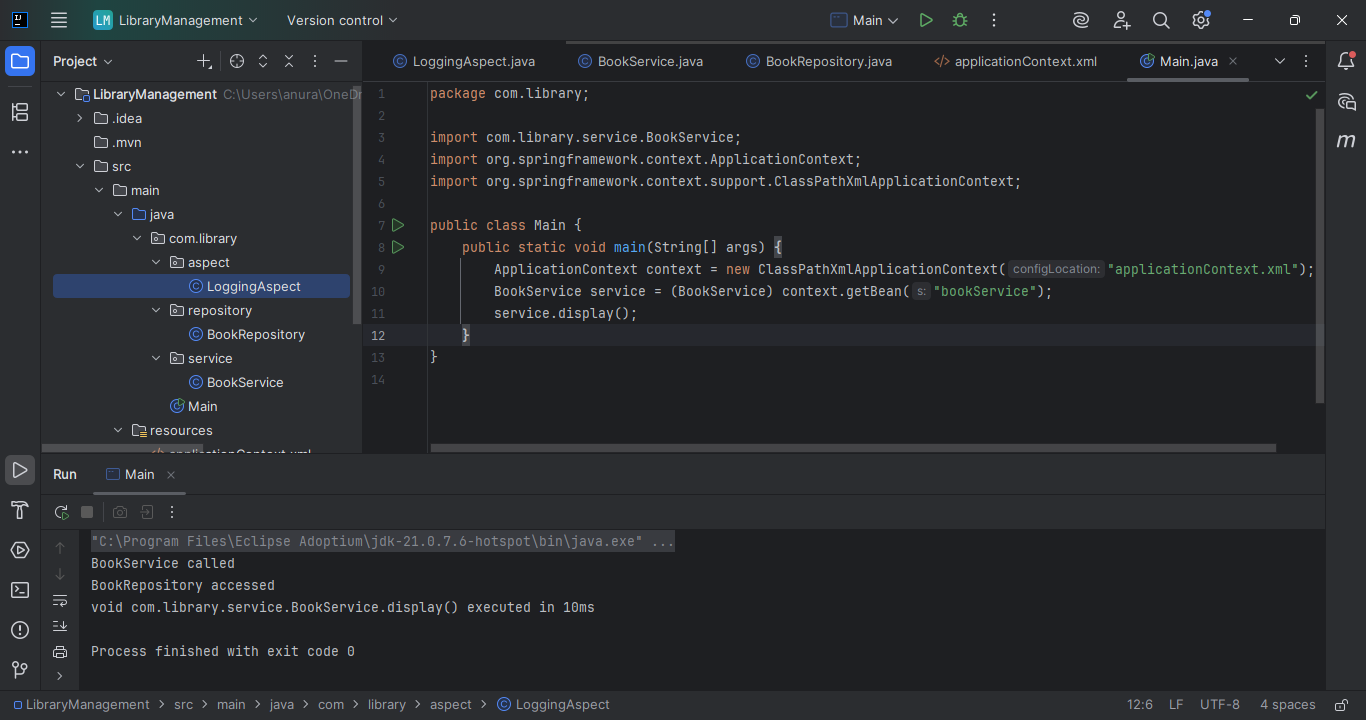
*package com.library.aspect;  
import org.aspectj.lang.annotation.Aspect;  
import org.aspectj.lang.annotation.Before;  
import org.aspectj.lang.annotation.After;  
  
@Aspect  
public class LoggingAspect {  
 @Before("execution(\* com.library.service.\*.\*(..))")  
 public void logBefore() {  
 System.out.println("Method execution started.");  
 }  
  
 @After("execution(\* com.library.service.\*.\*(..))")  
 public void logAfter() {  
 System.out.println("Method execution finished.");  
 }  
}*

1. **Enabled AspectJ Support:**

Update **applicationContext.xml** to enable **AspectJ** support and register the aspect.

*<beans xmlns:aop="http://www.springframework.org/schema/aop"  
 xsi:schemaLocation="... http://www.springframework.org/schema/aop/spring-aop.xsd">  
  
 <aop:aspectj-autoproxy/>  
 <bean id="loggingAspect" class="com.library.aspect.LoggingAspect"/>  
</beans>*

1. **Testing the Aspect:**



**Exercise 4: Creating and Configuring a Maven Project**

**Objective:**

Set up a fresh Maven project with Spring dependencies and plugins.

The steps in this exercise are already been performed previously in the exercises done before.

**Steps:**

1. **Created a New Maven Project:**

Created a new Maven project named **LibraryManagement** in IntelliJ IDEA.

1. **Added Spring Dependencies in pom.xml and configured Maven dependencies:**

Included dependencies for Spring Context, Spring AOP, and Spring WebMVC and configured the Maven Compiler Plugin for Java version 1.8 in the pom.xml file.

*<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>*

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

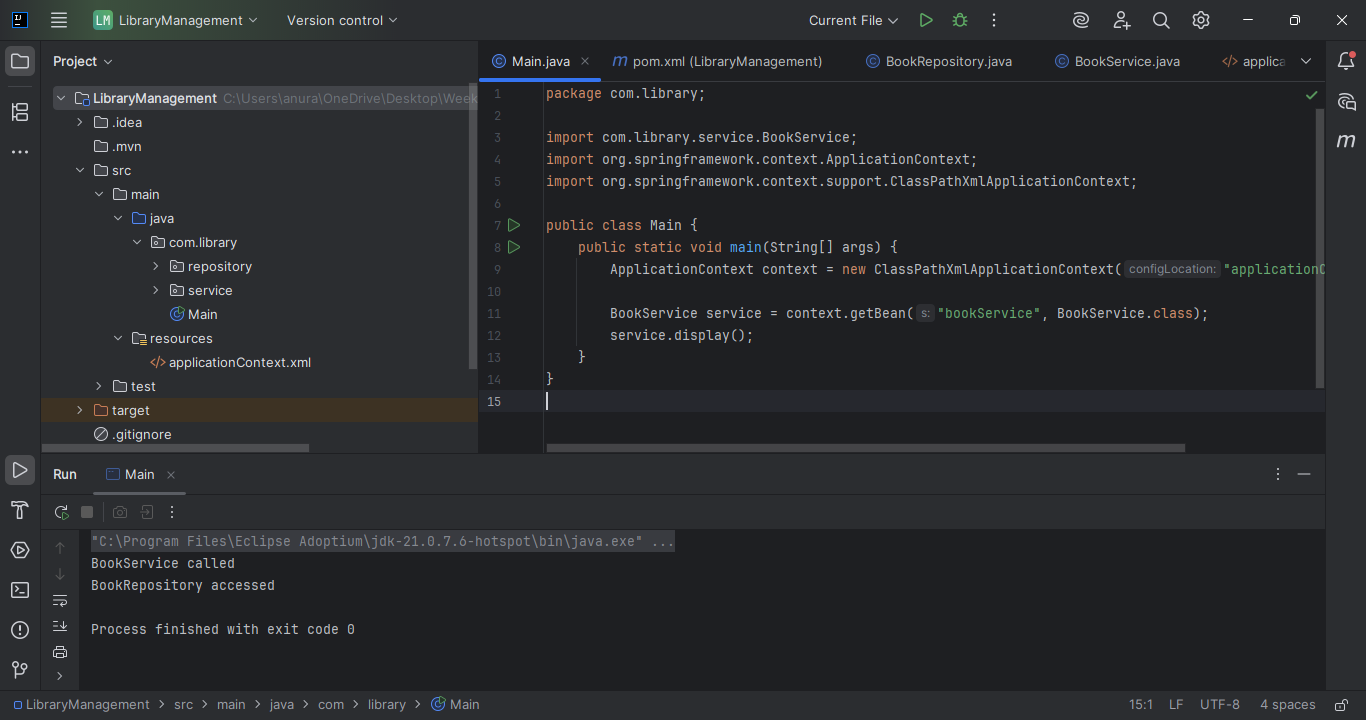
**Objective:**

Centralize bean configuration using Spring’s IoC container.

**Steps:**

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

OUTPUT:



**Exercise 6: Configuring Beans with Annotations**

**Objective:**

You need to simplify the configuration of beans in the library management application using annotations.

**Steps:**

1. **Enabled Component Scanning:**

Updated **applicationContext.xml** to include component scanning for the **com.library** package.

*<context:component-scan base-package="com.library"/>*

1. **Annotated Classes:**

Used **@Service** annotation for the **BookService** class.

*@Service*

*public class BookService {*

*...*

*}*

Used **@Repository** annotation for the **BookRepository** class.

*@Repository*

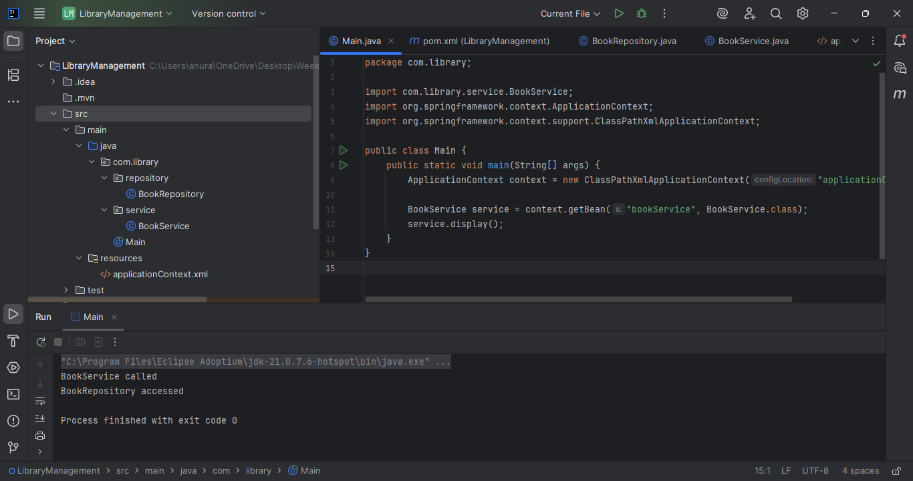
*public class BookRepository {*

*...*

*}*

1. **Testing the Configuration:**

OUTPUT:



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

**Objective:**

Use both constructor and setter injection.

**Steps:**

1. **Configuring Constructor Injection:**

Updated applicationContext.**xml** to configure constructor injection for **BookService**. This was done previously as follows:

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

*<constructor-arg ref="bookRepository"/>*

*</bean>*

1. **Configuring Setter Injection:**

**BookService** class has a setter method for **BookRepository** and configured in **applicationContext.xml**.

*public void setBookRepository(BookRepository bookRepository) {*

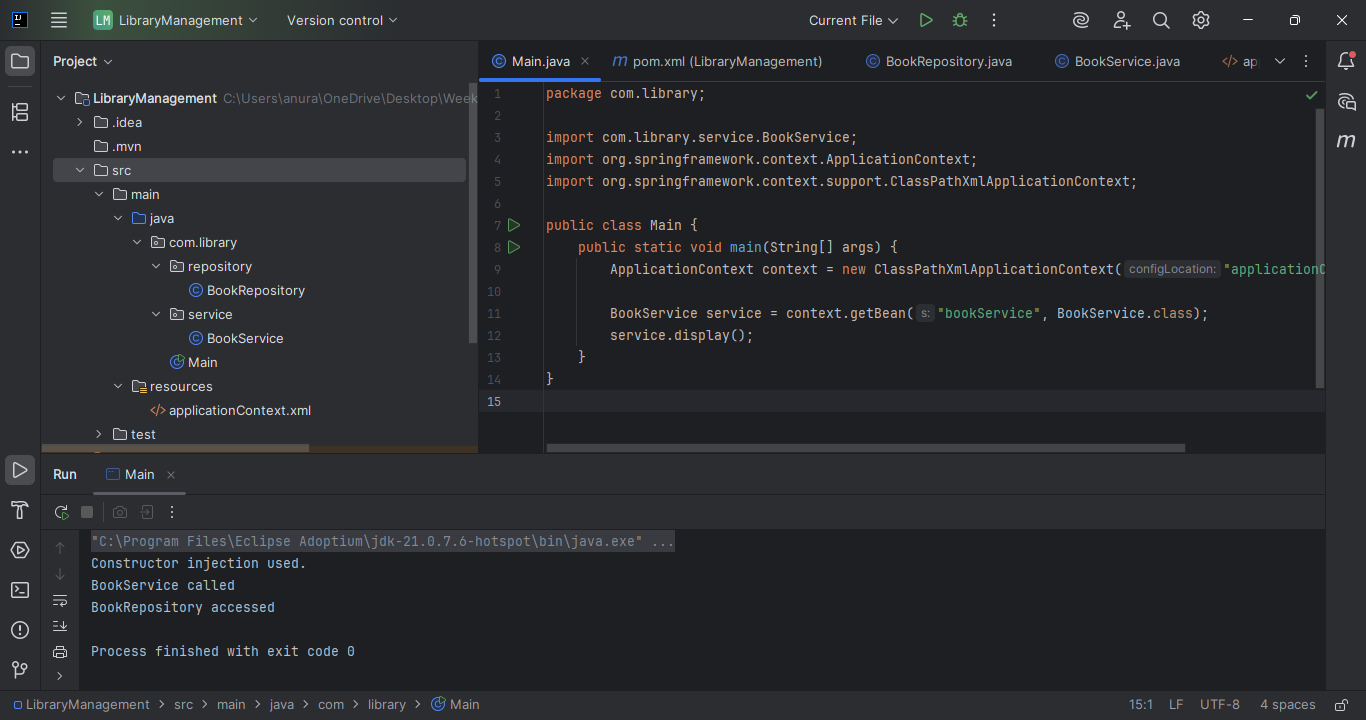
*this.bookRepository = bookRepository;*

*}*

1. **Testing the Injection:**

Run the **LibraryManagementApplication** main class to verify both constructor and setter injection.

OUTPUT:



**Exercise 8: Implementing Basic AOP with Spring**

**Objective:**

Use AOP to separate cross-cutting concerns.

**Steps:**

1. **Created an Aspect class:**

Created a package **com.library.aspect** and added a class **LoggingAspect**. This was already done in exercise 3.



1. **Created Advice Methods:**

Defined advice methods in **LoggingAspect** for logging before and after method execution.

1. **Configure the Aspect:**

Update **applicationContext.xml** to register the aspect and enable **AspectJ** auto-proxying.

CODE:

*<!--Enable AOP -->*

*<aop:aspectj-autoproxy/>*

*<!-- Aspect Bean -->*

*<bean id="loggingAspect" class="com.library.aspect.LoggingAspect"/>*

*<!-- AOP Configuration -->*

*<aop:config>*

*<aop:aspect ref="loggingAspect">*

*<aop:pointcut id="serviceMethods" expression="execution(\* com.library.service.\*.\*(..))"/>*

*<aop:before method="beforeAdvice" pointcut-ref="serviceMethods"/>*

*<aop:after method="afterAdvice" pointcut-ref="serviceMethods"/>*

*</aop:aspect>*

*</aop:config>*

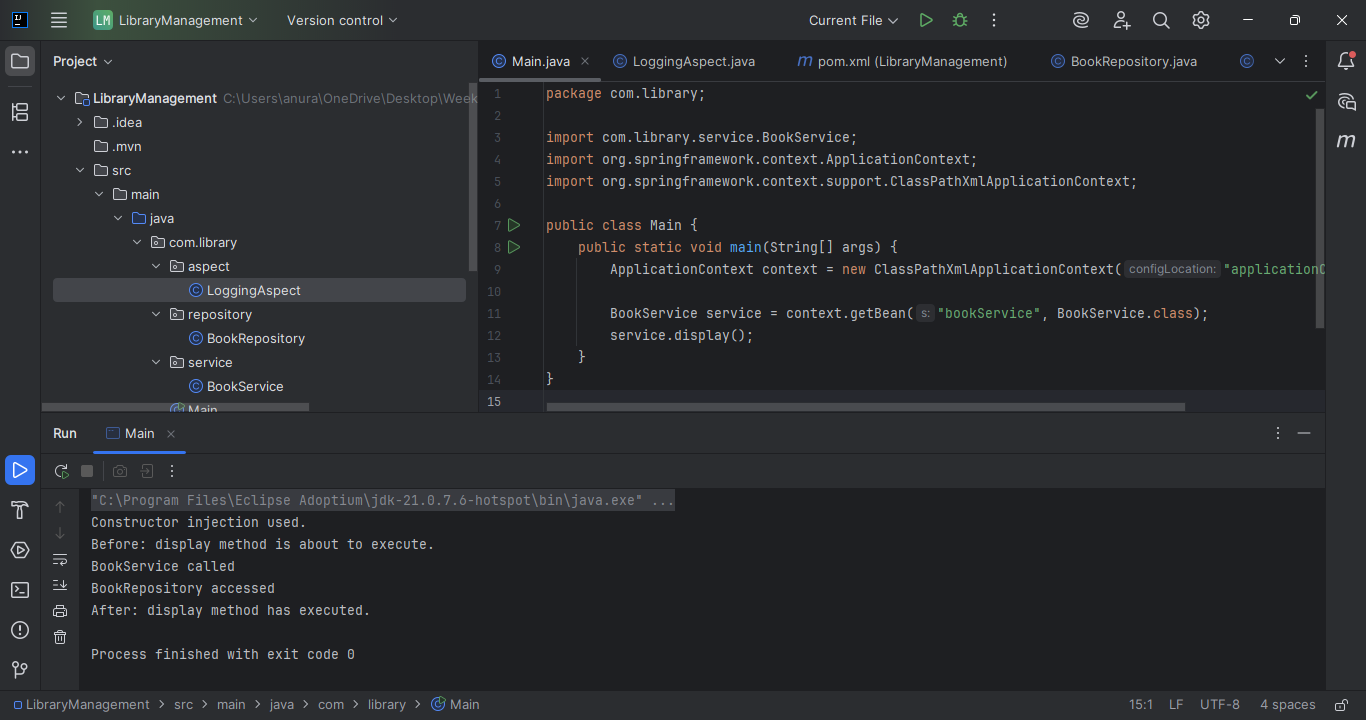
*<!-- Application Beans -->*

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

1. **Testing the Aspect:**

Run the **LibraryManagementApplication** main class to verify the AOP functionality.

OUTPUT:



**Exercise 9: Creating a Spring Boot Application**

**Objective:**

You need to create a Spring Boot application for the library management system to simplify configuration and deployment.

**Steps:**

1. **Created a Spring Boot Project:**

Used **Spring Initializr** to create a new Spring Boot project named **LibraryManagement**.

Project: Maven

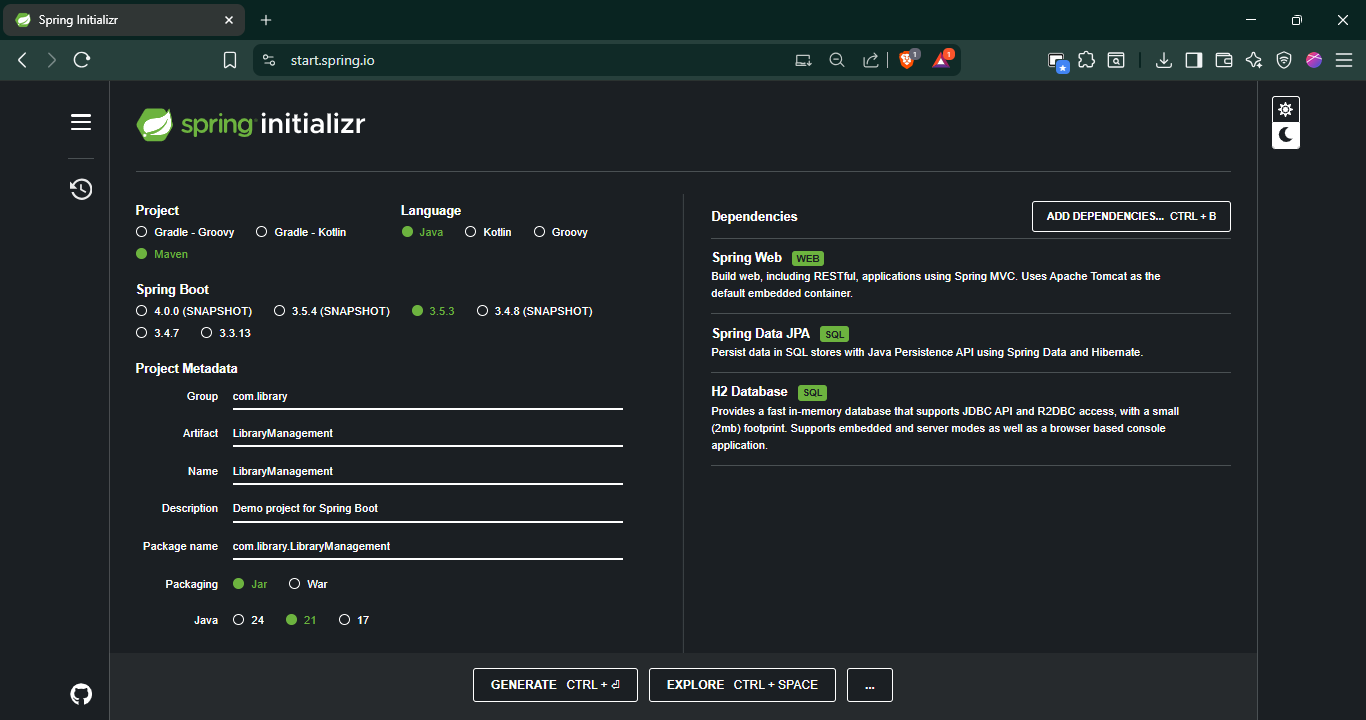
Language: Java

Spring Boot Version: 2.7.x or 3.x (latest stable)

Group ID: com.library

Artifact ID: LibraryManagement

Dependencies: Spring Web, Spring Data JPA, H2 Database



1. **Added Dependencies:**

Included dependencies for **Spring Web, Spring Data JPA, and H2 Database**.

1. **Created Application Properties:**

Configuring database connection properties in **application.properties**.

*spring.datasource.url=jdbc:h2:mem:librarydb*

*spring.datasource.driverClassName=org.h2.Driver*

*spring.datasource.username=sa*

*spring.datasource.password=*

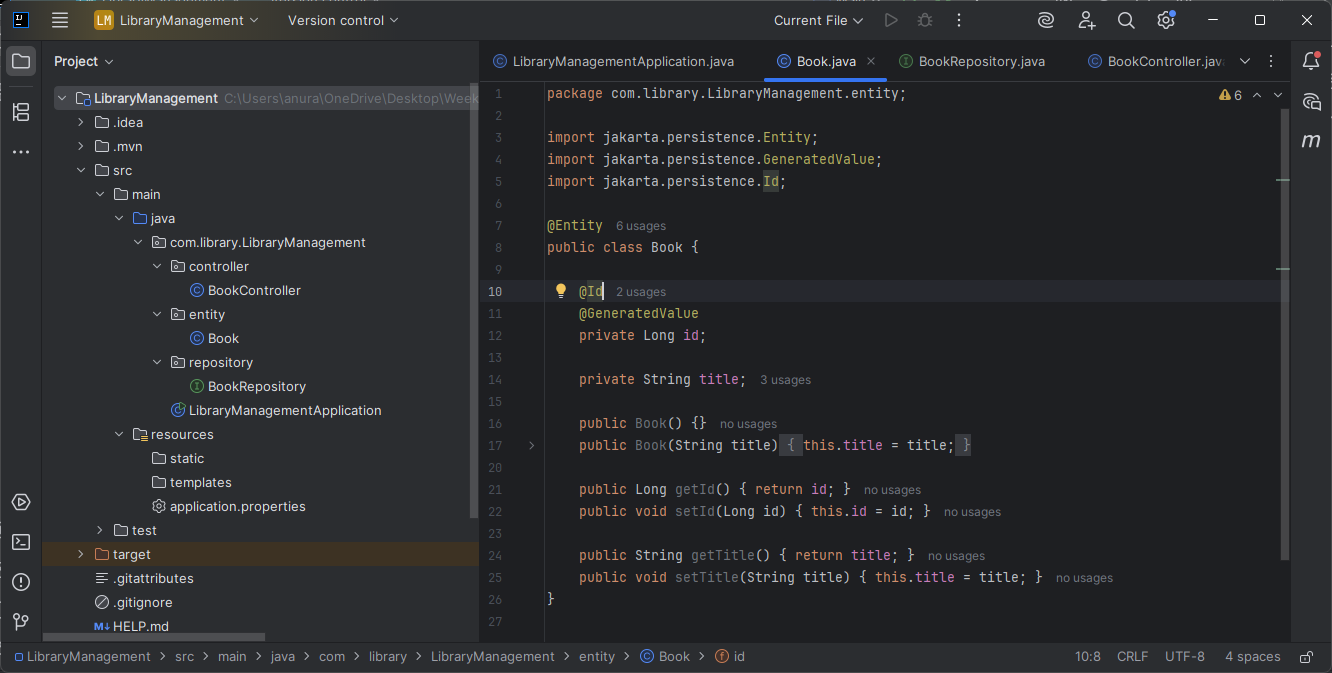
*spring.jpa.database-platform=org.hibernate.dialect.H2Dialect*

*spring.h2.console.enabled=true*

*spring.jpa.hibernate.ddl-auto=update*

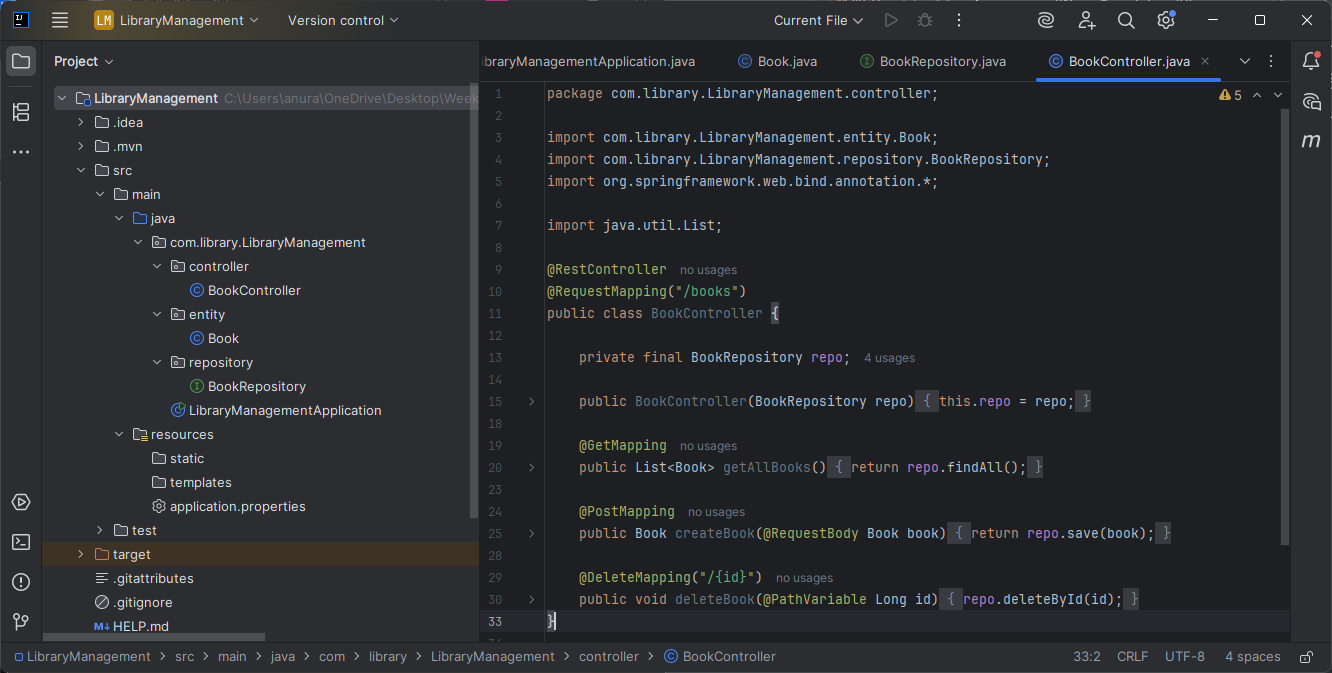
1. **Defined Entities and Repositories:**

Create **Book** entity and **BookRepository** interface.



1. **Created a REST Controller:**

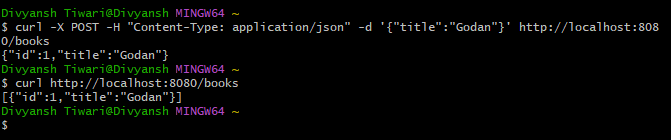
Created a **BookController** class to handle CRUD operations.



1. **Run the Application:**

Run the Spring Boot application and test the REST endpoints.

Test Output:



POST a book into the H2-in-memory-database and GET all books.