# Cognizant Java FSE – (Deep Skilling) (WEEK-3)

### MODULE : Spring Core and Maven

#### Submitted by

Name: S Subhashri

Email: [22102152@rmd.ac.in](mailto:22102152@rmd.ac.in)

College: RMD ENGINEERING COLLEGE

Batch: Java FSE – 2026 Superset ID: 6397078

### 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.

### Project structure overview

LibraryManagement/

├── src/

│ └── main/

│ ├── java/

│ │ └── com/

│ │ └── library/

│ │ ├── LibraryMain.java

│ │ ├── service/

│ │ │ └── BookService.java

│ │ └── repository/

│ │ └── BookRepository.java

│ └── resources/

│ └── applicationContext.xml

└── pom.xml

### BookRepository.java

package com.library.repository; public class BookRepository {

public void saveBook(String bookName) { System.out.println("Book saved: " + 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("Adding book: " + bookName); bookRepository.saveBook(bookName);

}

}

### LibraryMain.java

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext; public class LibraryMain {

public static void main(String[] args) { ApplicationContext context =

new ClassPathXmlApplicationContext("applicationContext.xml");

BookService bookService = (BookService) context.getBean("bookService"); bookService.addBook("The Great Gatsby");

}

}

### 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"](http://www.springframework.org/schema/beans/spring-beans.xsd)>

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

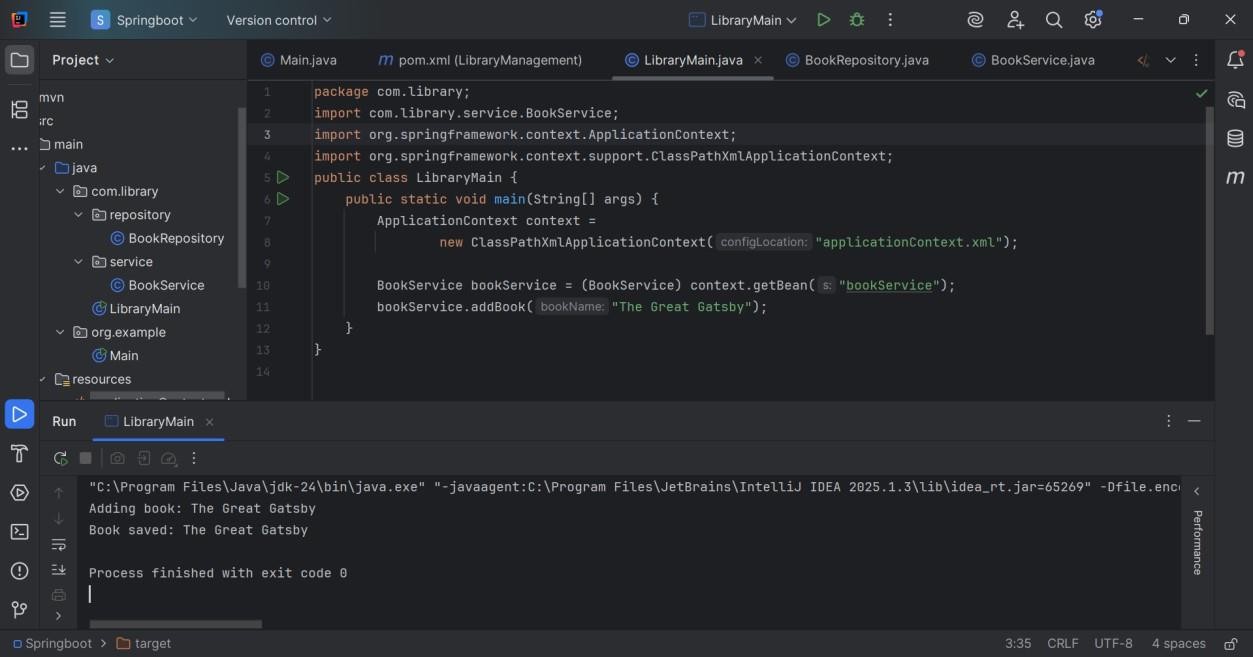
<bean id="bookService" class="com.library.service.BookService">

<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.

### Project structure overview

LibraryManagement/

├── src/

│ ├── main/

│ │ ├── java/

│ │ │ └── com/example/library/

│ │ │ ├── BookRepository.java

│ │ │ ├── BookService.java

│ │ │ └── LibraryManagementApplication.java

│ │ └── resources/

│ │ └── applicationContext.xml

└── pom.xml

**BookRepository.java** package com.example.library; public class BookRepository {

public void saveBook(String bookName) { System.out.println("BookRepository: saving book => " + bookName);

}

}

**BookService.java** package com.example.library; public class BookService {

private BookRepository bookRepository;

// setter method for Spring DI

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

}

public void addBook(String bookName) { System.out.println("BookService: adding book => " + bookName); bookRepository.saveBook(bookName);

}

}

### LibraryManagementApplication.java

package com.example.library;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext; public class LibraryManagementApplication {

public static void main(String[] args) {

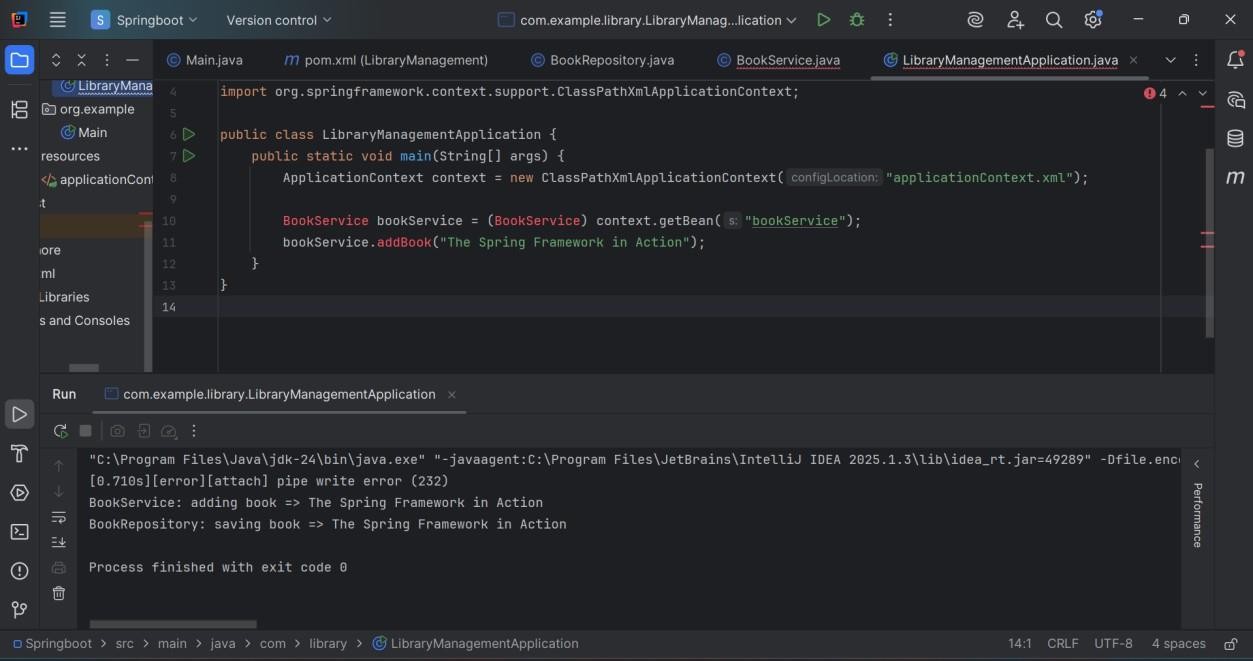
ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

BookService bookService = (BookService) context.getBean("bookService"); bookService.addBook("The Spring Framework in Action");

}

}

## 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.

LibraryManagement/

├── src/

│ ├── main/

│ │ └── java/

│ └── test/

├── pom.xml

### Add Spring Dependencies in pom.xml

Edit the pom.xml file inside the LibraryManagement folder. Add the following under <dependencies>:

<dependencies>

<!-- Spring Context -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.36</version>

</dependency>

<!-- Spring AOP -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.36</version>

</dependency>

<!-- Spring WebMVC -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.36</version>

</dependency>

<!-- Servlet API (provided by server like Tomcat) -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>4.0.1</version>

<scope>provided</scope>

</dependency>

</dependencies>

### Configure Maven Compiler Plugin for Java 1.8

Inside pom.xml, add the Maven Compiler Plugin inside <build> section:

<build>

<plugins>

<!-- Maven Compiler Plugin -->

<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>

#### Final pom.xml Example

<project xmlns="[http://maven.apache.org/POM/4.0.0"](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"](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>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.36</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.36</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.36</version>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>4.0.1</version>

<scope>provided</scope>

</dependency>

</dependencies>

<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>

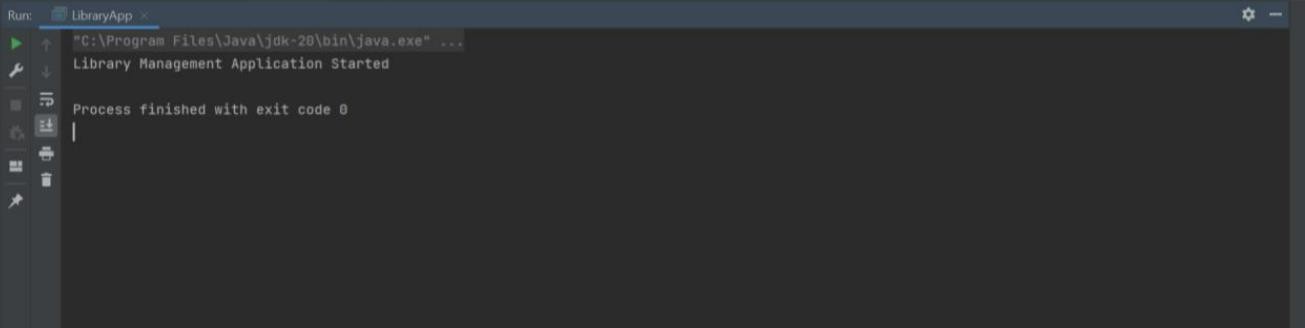
</plugin>

</plugins>

</build>

</project>

## OUTPUT

****

***Additional important hands-on***

### Exercise 5: Configuring the Spring IoC Container

#### Scenario:

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

### Project structure overview

LibraryManagement/

├── src/

│ ├── main/

│ │ ├── java/

│ │ │ └── com/example/library/

│ │ │ ├── LibraryApp.java

│ │ │ ├── service/BookService.java

│ │ │ └── repository/BookRepository.java

│ │ └── resources/

│ │ └── applicationContext.xml

└── pom.xml

### BookRepository.java

package com.example.library.repository; public class BookRepository {

public BookRepository() { System.out.println("BookRepository initialized");

}

public void findAllBooks() {

System.out.println("Fetching all books from the repository...");

}

}

### BookService.java

package com.example.library.service;

import com.example.library.repository.BookRepository; public class BookService {

private BookRepository bookRepository;

// Setter for injection

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

}

public void display() {

System.out.println("BookService is working with: " + bookRepository);

}

}

### LibraryApp.java

package com.example.library;

import com.example.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext; public class LibraryApp {

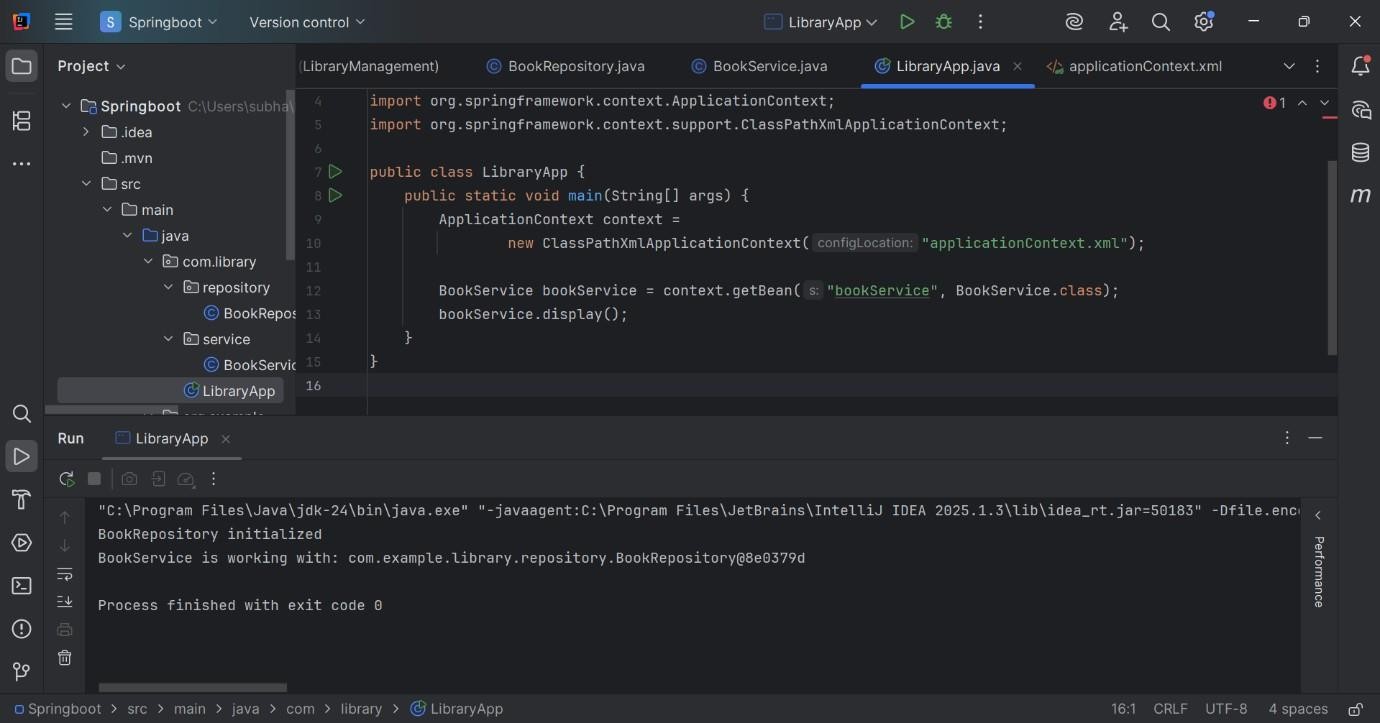
public static void main(String[] args) { ApplicationContext context =

new ClassPathXmlApplicationContext("applicationContext.xml"); BookService bookService = context.getBean("bookService", BookService.class); bookService.display();

}

}

## OUTPUT

****

### Exercise 7: Implementing Constructor and Setter Injection

#### Scenario:

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

### Implementation Details

1. Constructor Injection
   * Used in BookService to inject a serviceName via its constructor.
   * Configured in the applicationContext.xml using the <constructor-arg> tag.
2. Setter Injection
   * Used in BookService to inject the BookRepository dependency through a public setter method.

### ApplicationContext.xml Configuration

* + Declares a bookRepository bean of type BookRepository.
  + Declares a bookService bean of type BookService.
    - Provides a value to the serviceName parameter through constructor injection.
    - Injects the bookRepository through setter injection.

### Sample snippet:

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

<constructor-arg value="Library Book Service" />

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

</bean>

### Testing the Injection

* + The LibraryManagementApplication class retrieves the bookService bean from the Spring context.
  + It calls the displayServiceDetails() method, which prints:
    - The service name injected through constructor injection
    - Book repository output injected through setter injection

**BookRepository class** package com.example.library; public class BookRepository {

public void displayBooks() {

System.out.println("BookRepository: Displaying list of books.");

}

}

### BookService class

package com.example.library; public class BookService {

private BookRepository bookRepository; private String serviceName;

### // Constructor for constructor injection

public BookService(String serviceName) { this.serviceName = serviceName;

}

### // Setter for setter injection

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

}

public void displayServiceDetails() { System.out.println("BookService name: " + serviceName); bookRepository.displayBooks();

}

}

### LibraryManagementApplication main class

package com.example.library;

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.displayServiceDetails();

}

}

### 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.example.library.BookRepository" />

<!-- BookService bean with constructor and setter injection -->

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

<!-- constructor-arg for constructor injection -->

<constructor-arg value="Library Book Service" />

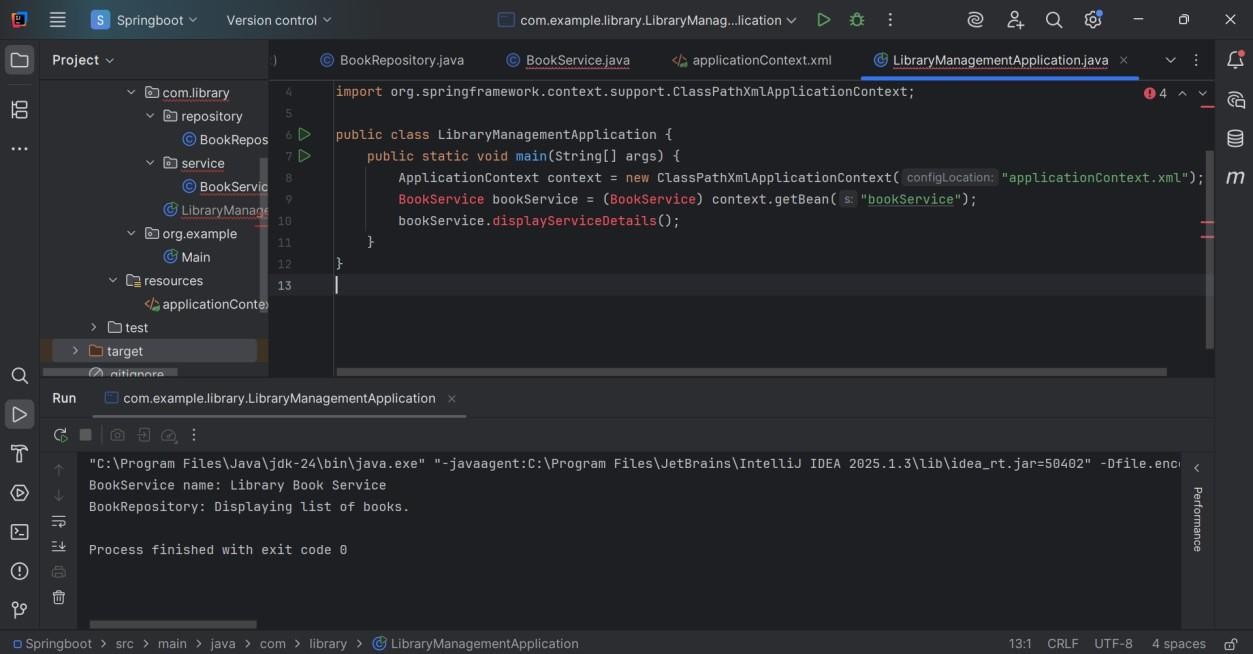
<!-- property for setter injection -->

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

</bean>

</beans>

**OUTPUT**

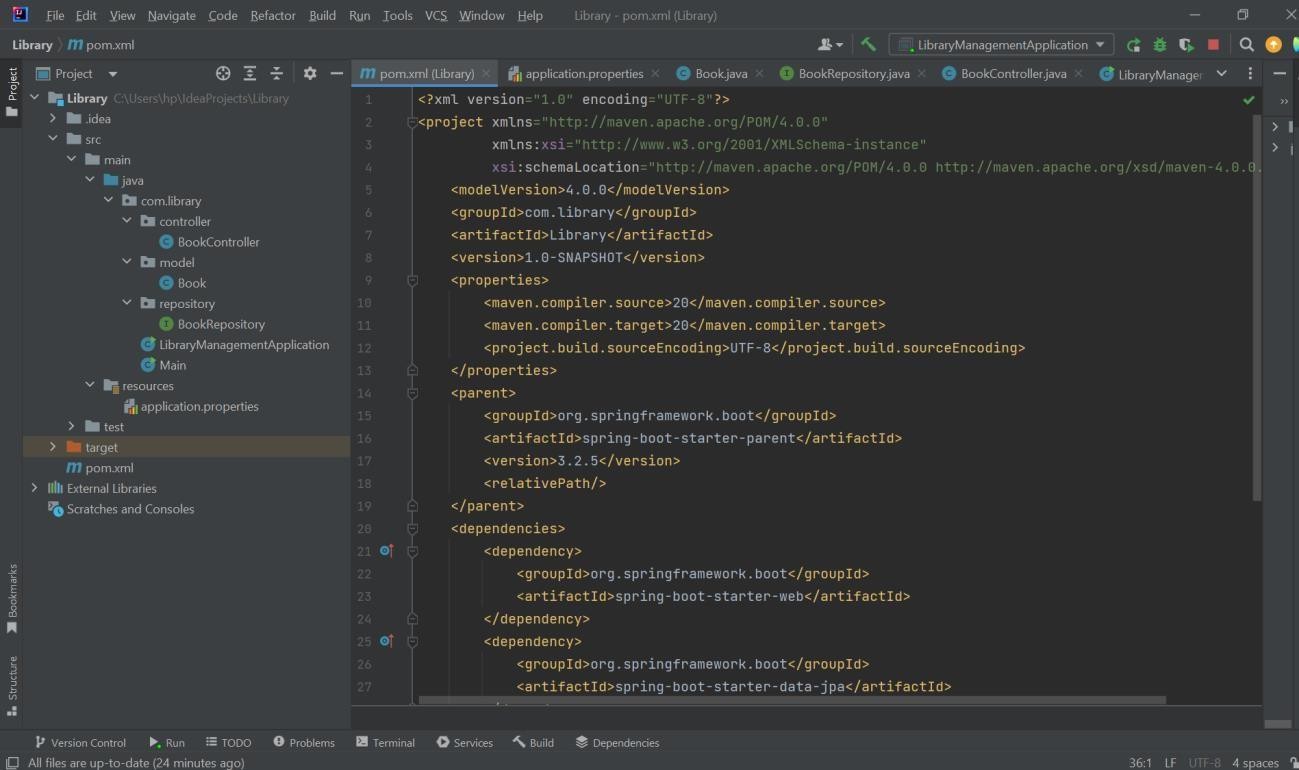
****

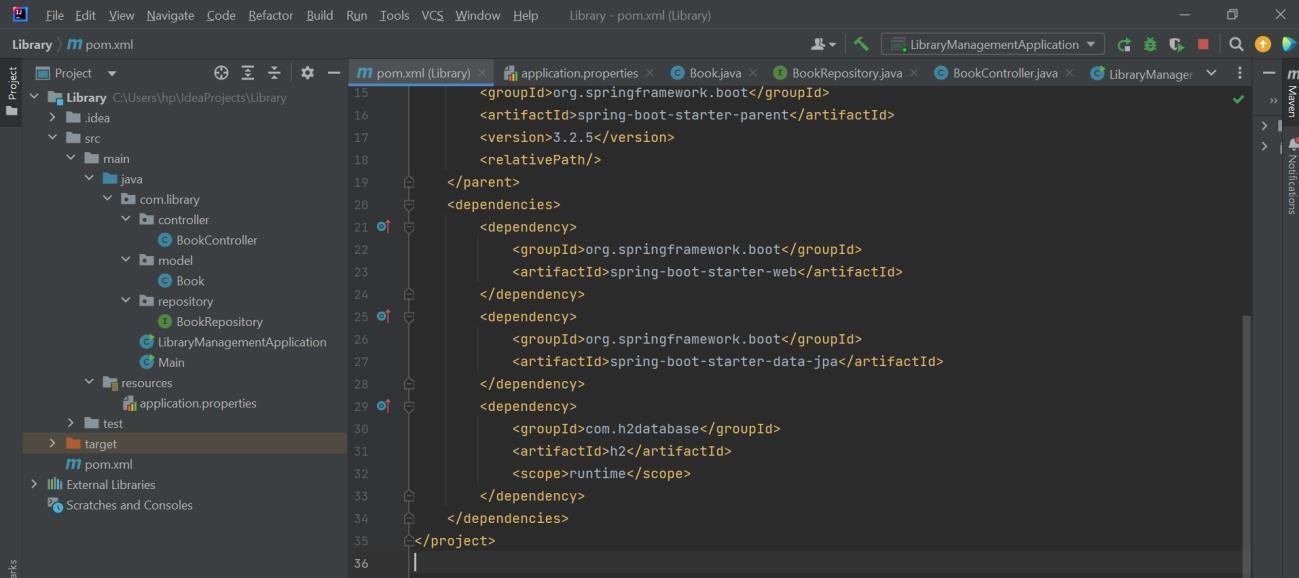
# Exercise 9: Creating a Spring Boot Application

### Scenario:

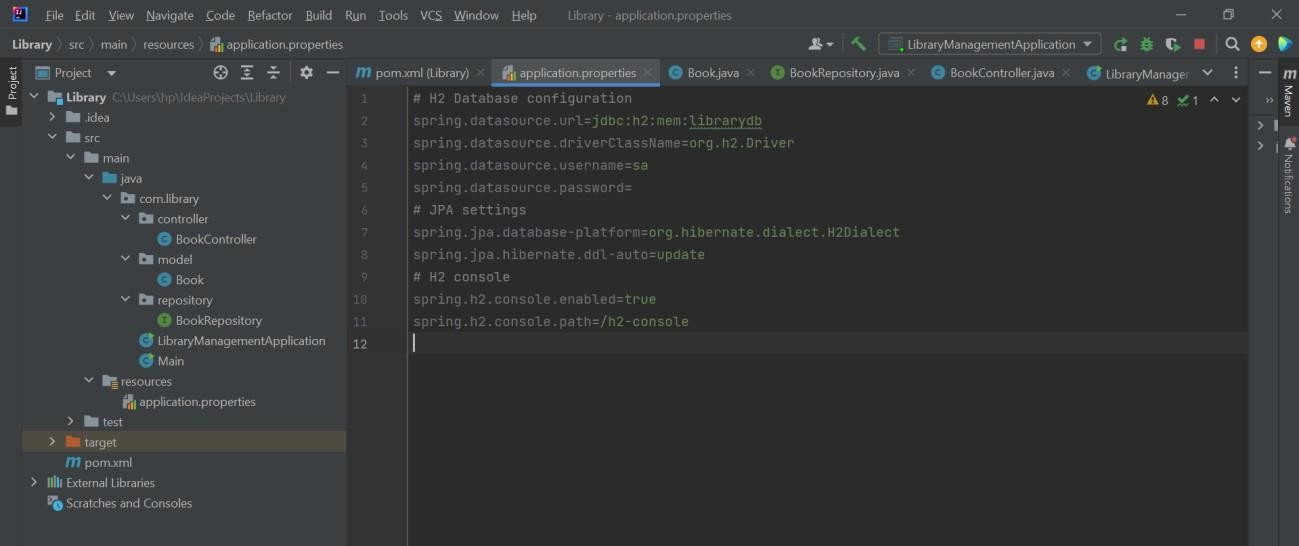
You need to create a Spring Boot application for the library management system to simplify configuration and deployment.

**Step 1:Add Spring dependencies to pom.xml**

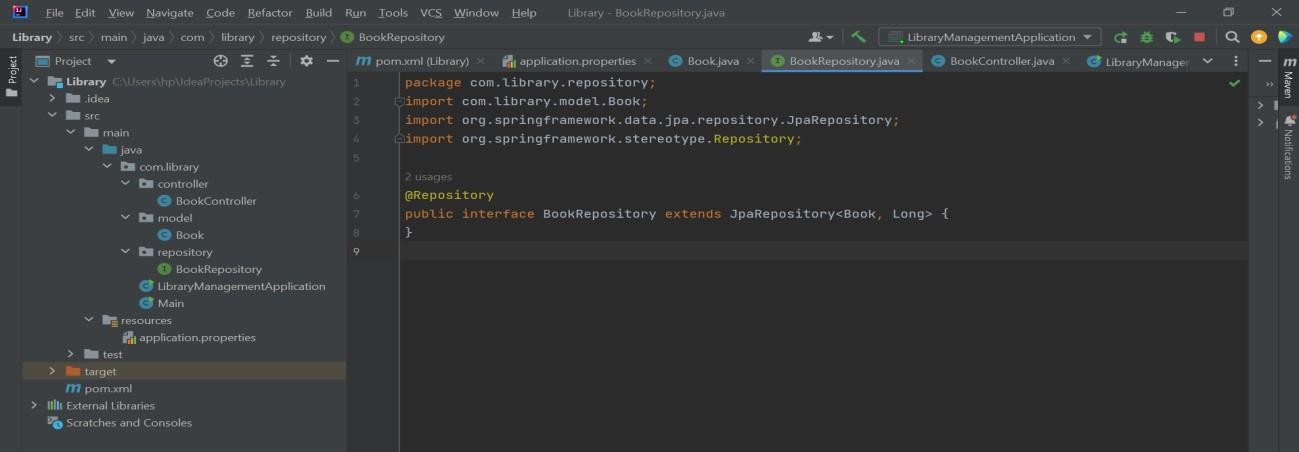
****

****

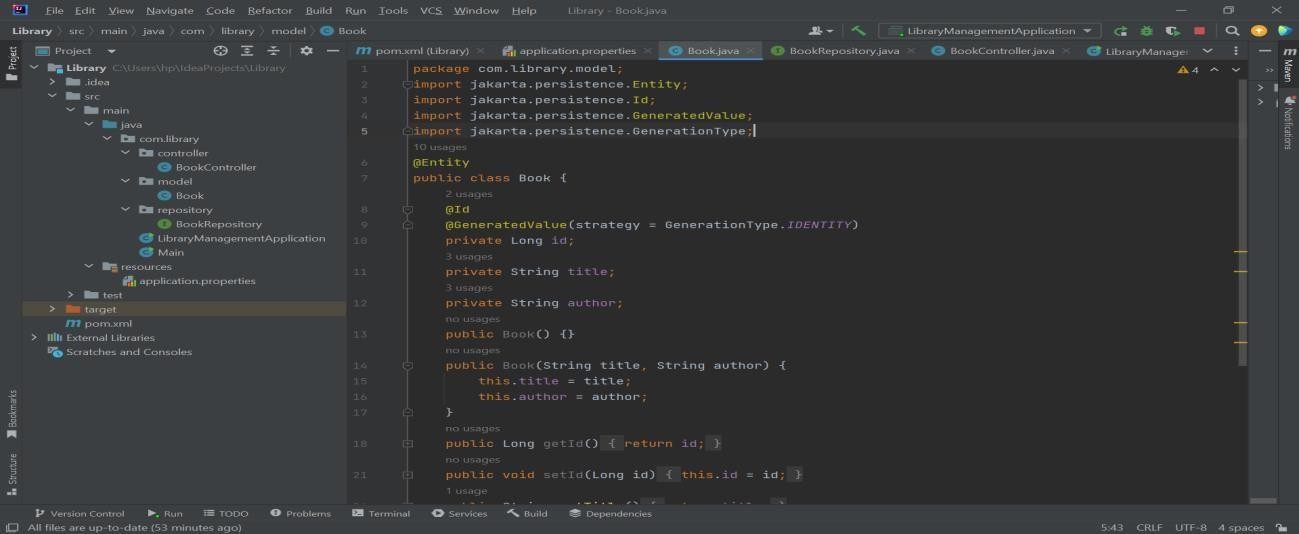
**Step 2:Create application.properties file**

****

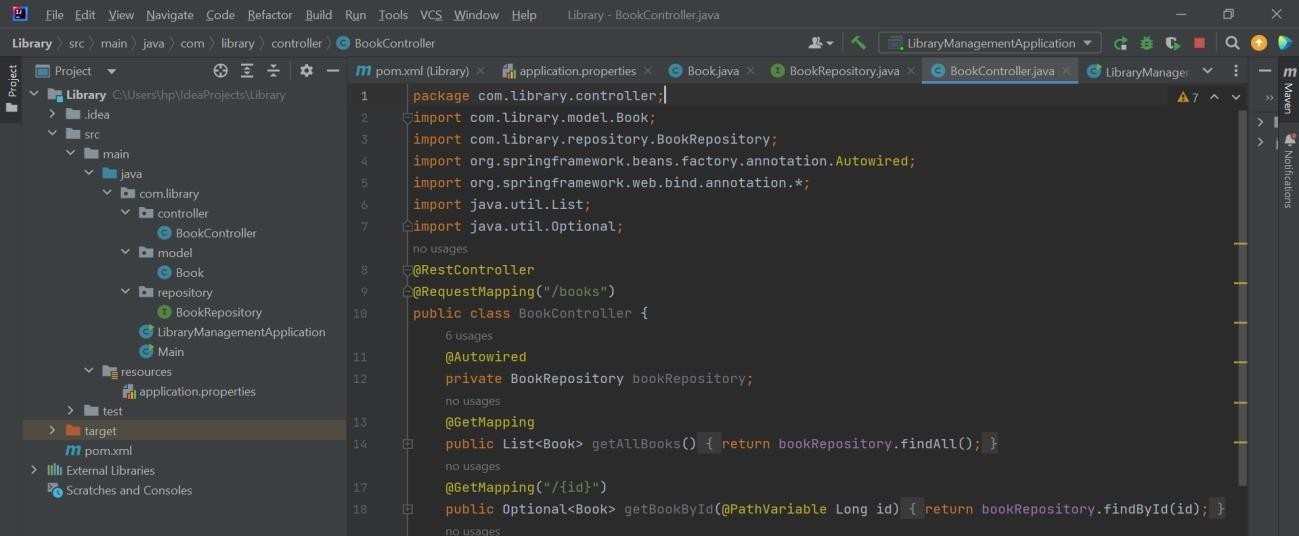
**Step 3:Create BookRepository.java and Book.java**

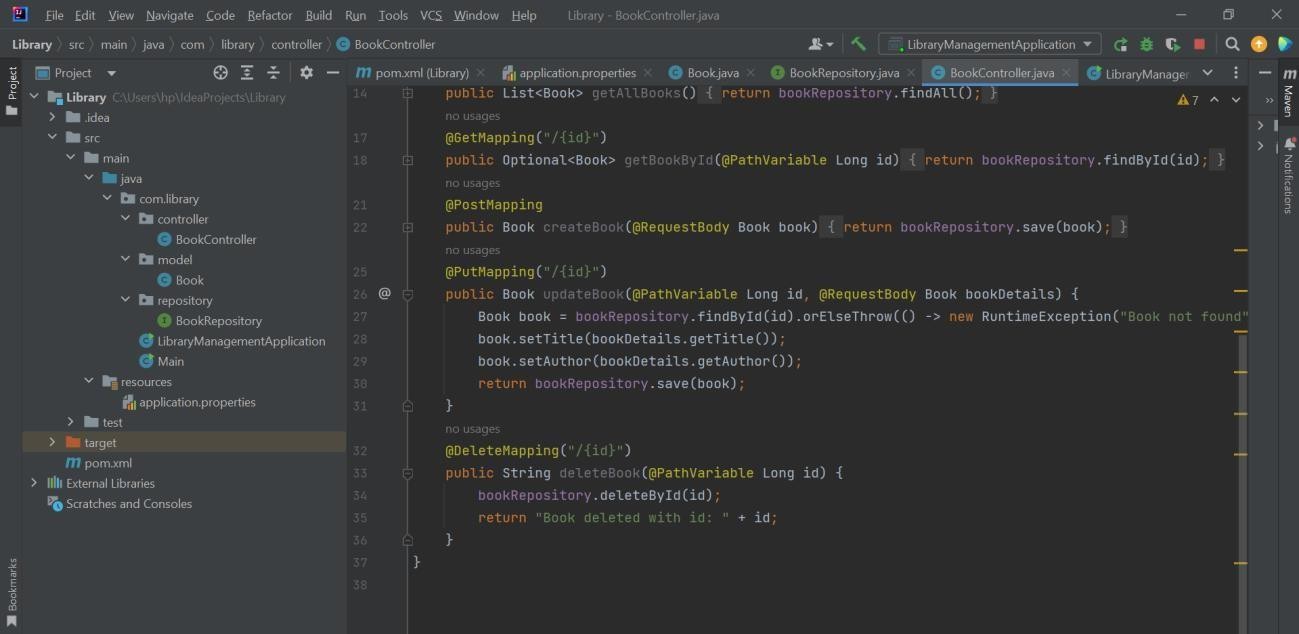
****

**Book.java**

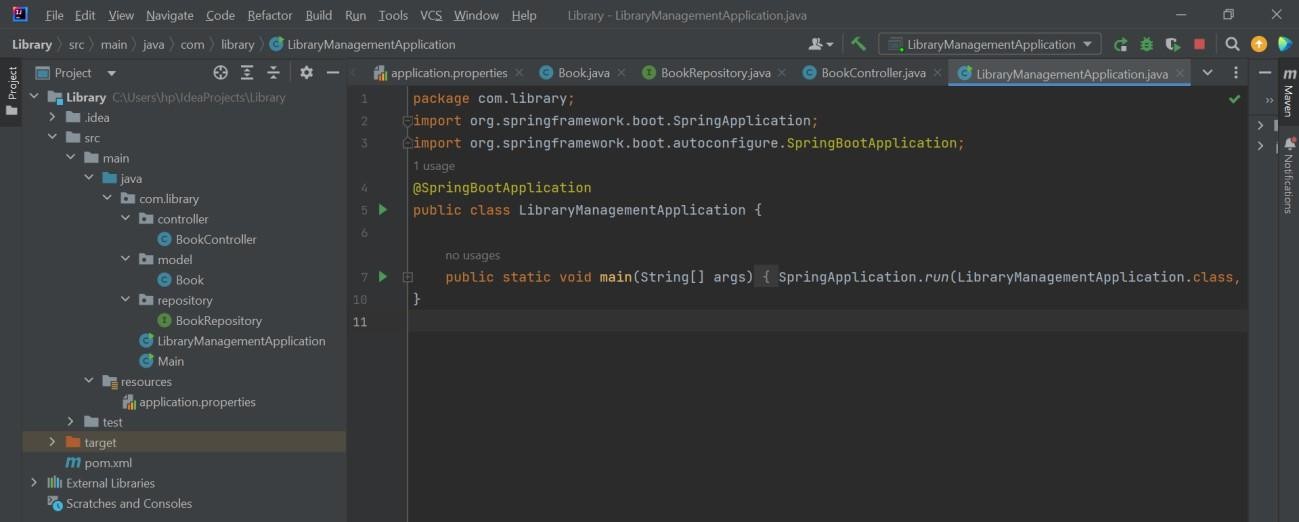
****

**Step 4:Create a controller class BookController.java**

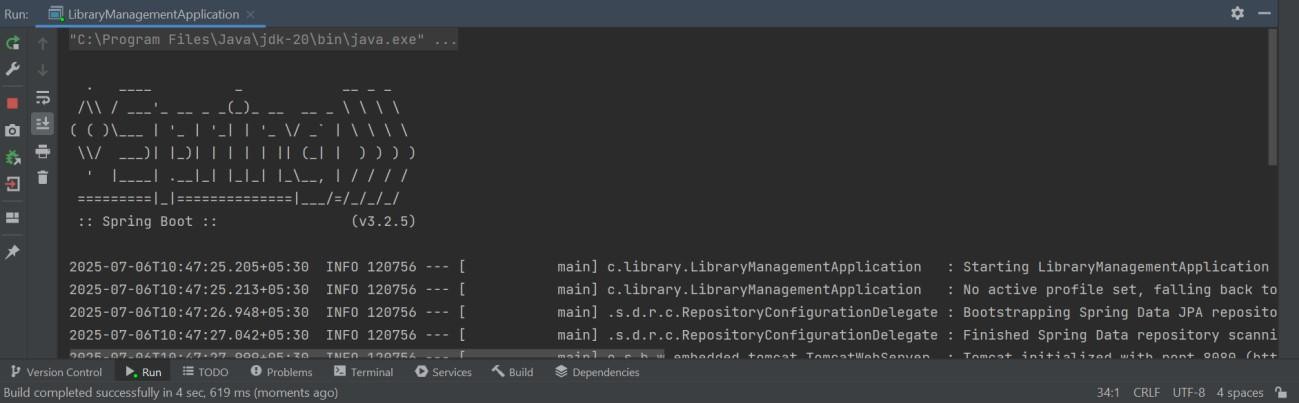
****



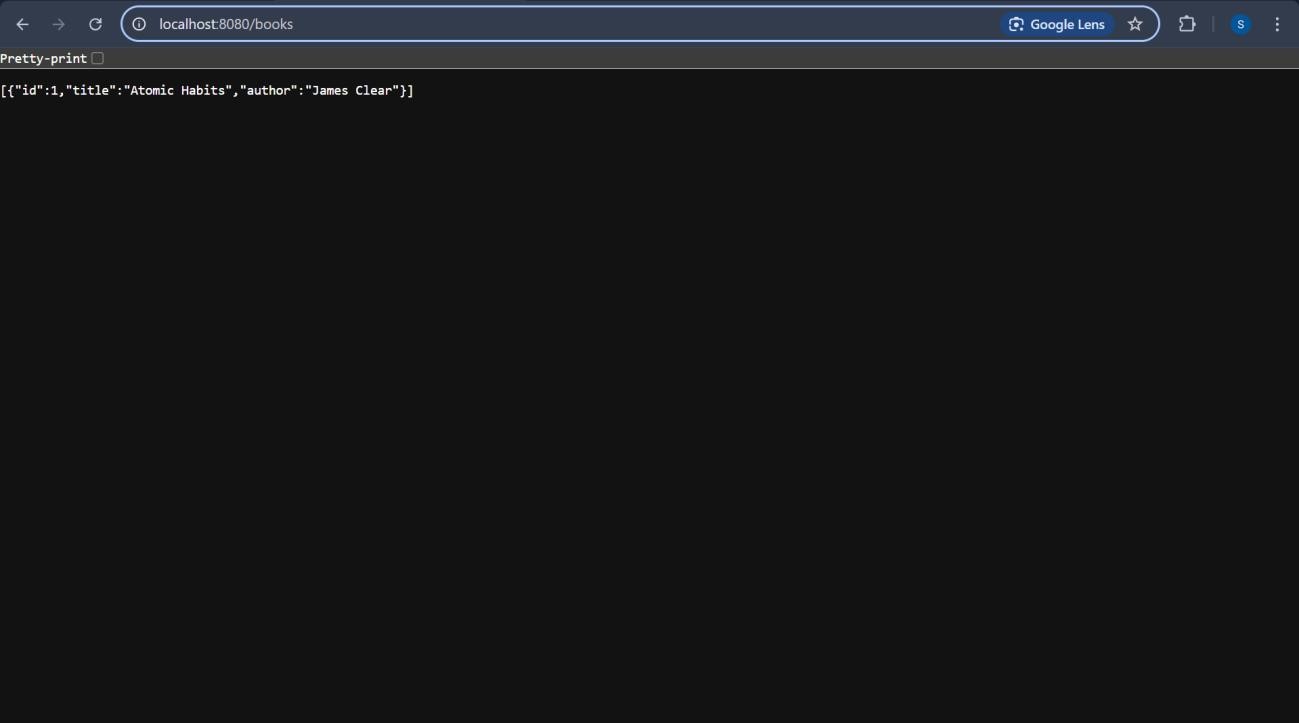
**Step 5:Create and run LibraryManagementApplication.java**

****

## OUTPUT:

****

**Step 6:Test the REST end points**

****