**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.

**LibraryManagement/**

**│**

**├── pom.xml**

**└── src/**

**└── main/**

**├── java/**

**│ └── com/**

**│ └── library/**

**│ ├── service/**

**│ │ └── BookService.java**

**│ └── repository/**

**│ └── BookRepository.java**

**└── resources/**

**└── applicationContext.xml**

**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>5.3.36</version> <!-- Use a stable version -->

</dependency>

</dependencies>

</project>

Create 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>

**BookRepository.java**

**package** com.library.repository;

**public** **class** BookRepository {

**public** **void** saveBook(String title) {

System.***out***.println("Book saved: " + title);

}

}

**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** addBook(String title) {

bookRepository.saveBook(title);

}

}

**Create Main Class to Load Spring Context**

**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) {

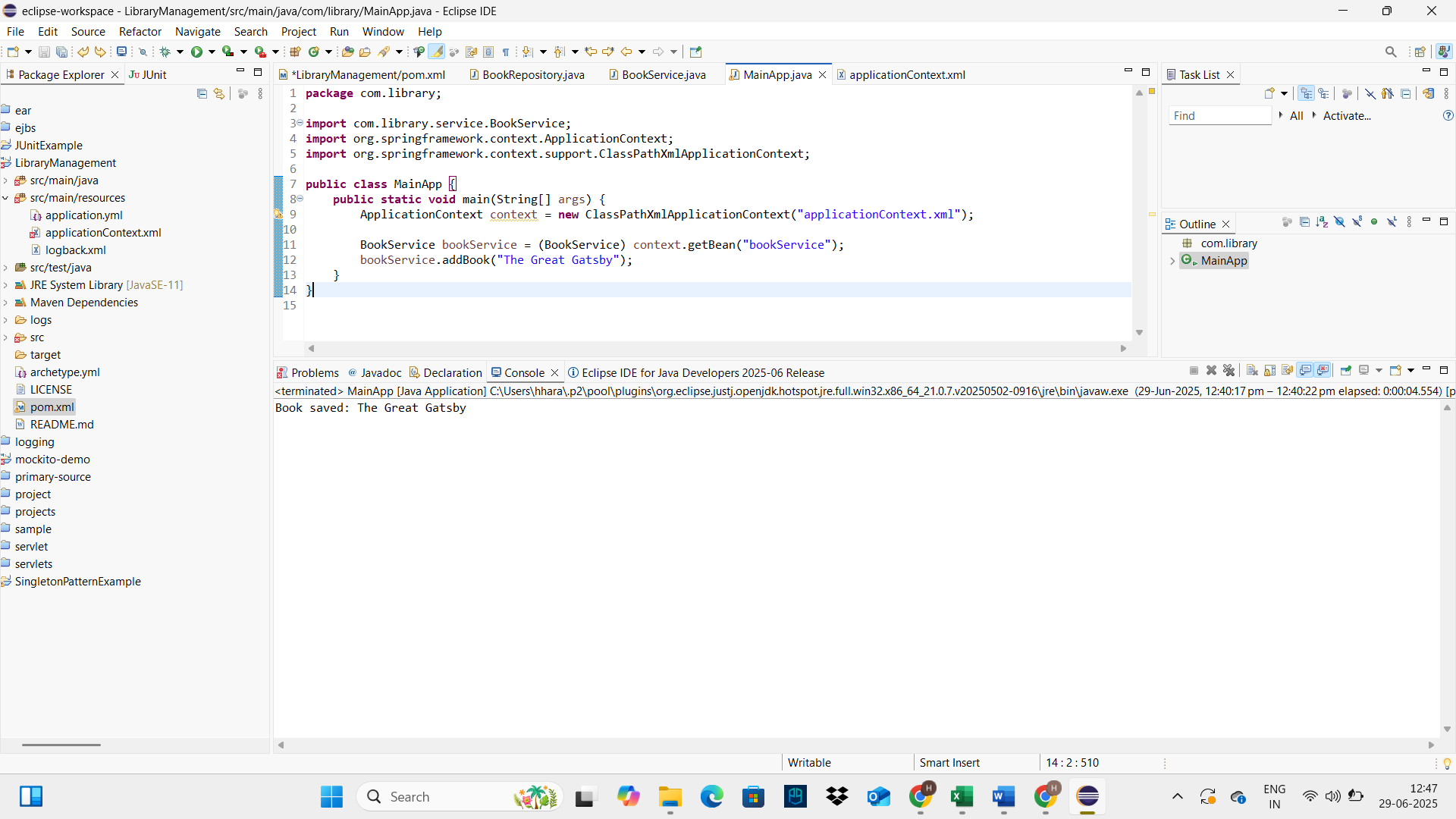
ApplicationContext context = **new** ClassPathXmlApplicationContext("applicationContext.xml");

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

bookService.addBook("The Great Gatsby");

}

}

**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.

**applicationContext.xml**

<!-- 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">

<!-- Define BookRepository bean -->

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

<!-- Define BookService bean and inject BookRepository using setter injection -->

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

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

</bean>

</beans>

**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** addBook(String title) {

bookRepository.saveBook(title);

}

**public** **void** displayBooks() {

bookRepository.findAllBooks().forEach(System.***out***::println);

}

}

**BookRepository.java**

**package** com.library.repository;

**import** java.util.Arrays;

**import** java.util.List;

**public** **class** BookRepository {

**public** **void** saveBook(String title) {

System.***out***.println("Book saved: " + title);

}

**public** List<String> findAllBooks() {

**return** Arrays.*asList*("Book A", "Book B", "Book C");

}

}

**LibraryManagementApplication.java**

**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");

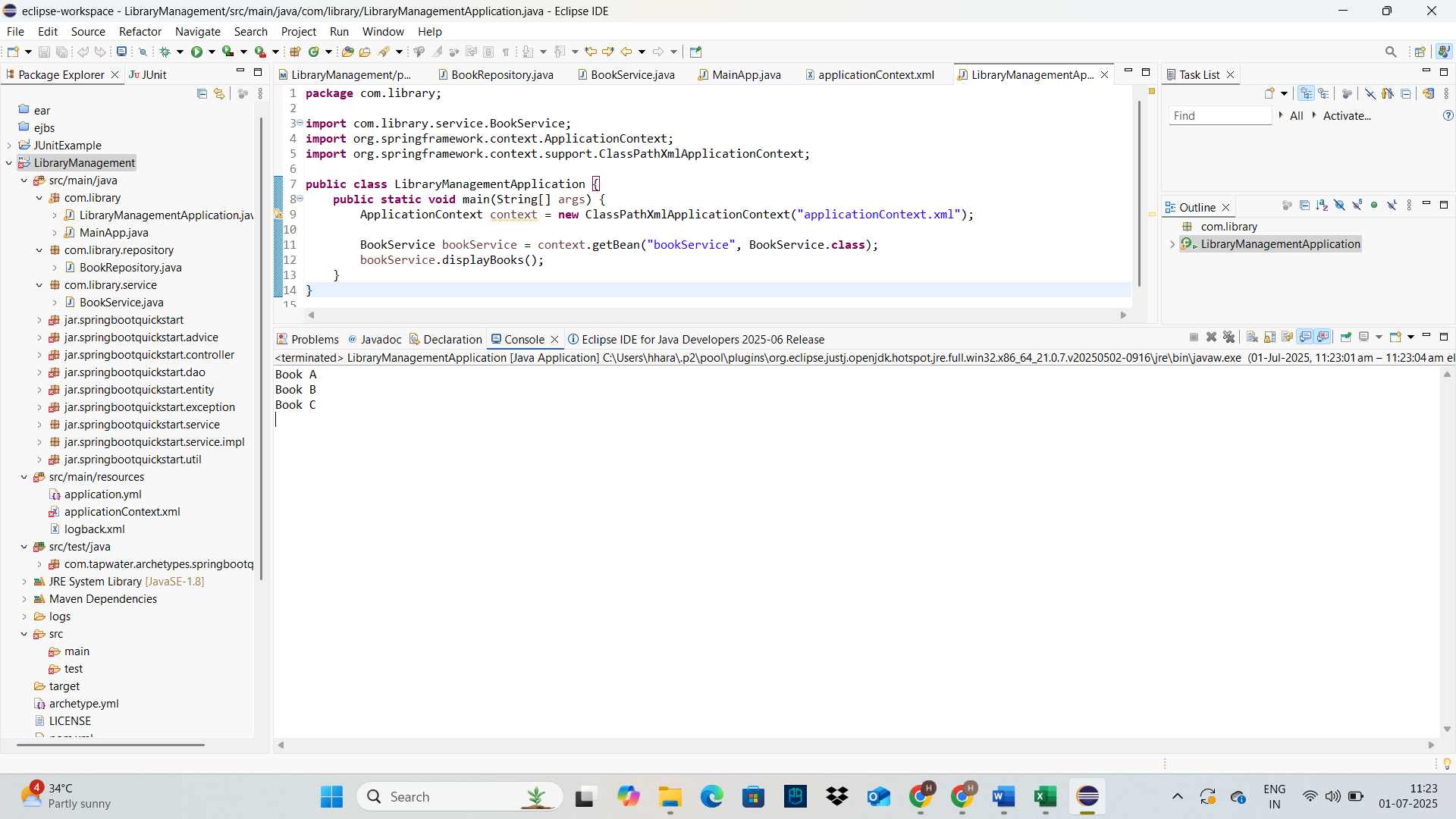
BookService bookService = context.getBean("bookService", BookService.**class**);

bookService.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.

**Open the pom.xml and add the following content inside <dependencies>:**

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.34</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.34</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.34</version>

</dependency>

<dependency>

<groupId>commons-logging</groupId>

<artifactId>commons-logging</artifactId>

<version>1.2</version>

</dependency>

</dependencies>

**Add the following inside the <build> section of your pom.xml:**

**<build>**

**<plugins>**

**<plugin>**

**<groupId>org.apache.maven.plugins</groupId>**

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

**<version>3.10.1</version>**

**<configuration>**

**<source>1.8</source>**

**<target>1.8</target>**

**</configuration>**

**</plugin>**

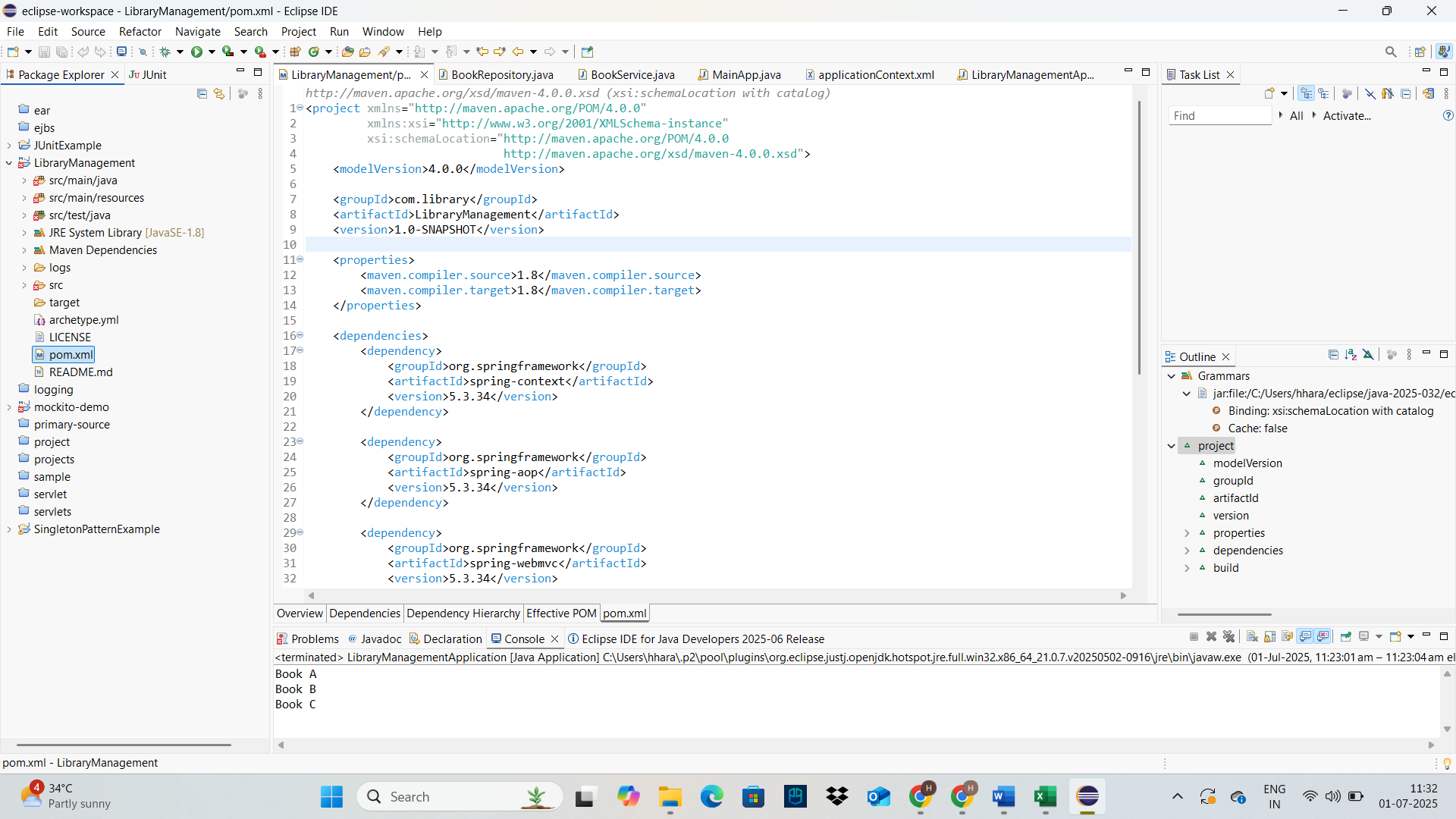
**</plugins>**

**</build>**

1. **Right-click the project** in Project Explorer
2. Go to Maven → Update Project
3. Check Force Update of Snapshots/Releases
4. Click OK

✅ Maven will now download Spring libraries automatically

**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.

**applicationContext.xml**

<!-- 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">

<!-- Define BookRepository bean -->

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

<!-- Define BookService bean and inject BookRepository using setter injection -->

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

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

</bean>

</beans>

**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** addBook(String title) {

bookRepository.saveBook(title);

}

**public** **void** displayBooks() {

bookRepository.findAllBooks().forEach(System.***out***::println);

}

}

**BookRepository.java**

**package** com.library.repository;

**import** java.util.Arrays;

**import** java.util.List;

**public** **class** BookRepository {

**public** **void** saveBook(String title) {

System.***out***.println("Book saved: " + title);

}

**public** List<String> findAllBooks() {

**return** Arrays.*asList*("Book A", "Book B", "Book C");

}

}

**LibraryManagementApplication.java**

**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");

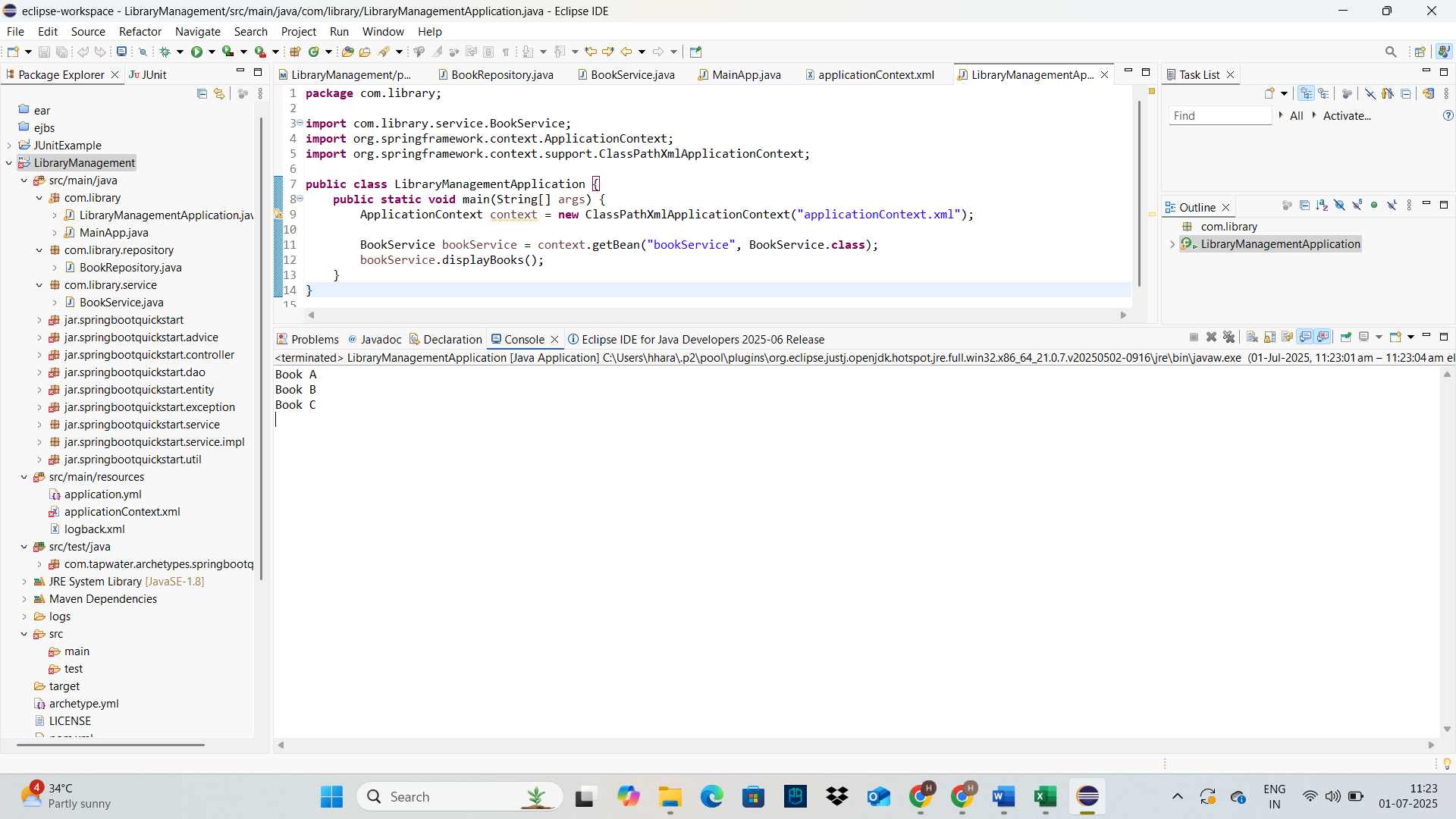
BookService bookService = context.getBean("bookService", BookService.**class**);

bookService.displayBooks();

}

}

**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.

**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.

**applicationContext.xml**

<!-- 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">

<!-- Define BookRepository bean -->

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

<!-- Define BookService bean and inject BookRepository using setter injection -->

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

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

</bean>

</beans>

**BookService.java**

**package** com.library.service;

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

**public** **class** BookService {

**private** BookRepository bookRepository;

**private** String serviceName;

// Constructor injection

**public** BookService(String serviceName) {

**this**.serviceName = serviceName;

}

// Setter injection

**public** **void** setBookRepository(BookRepository bookRepository) {

**this**.bookRepository = bookRepository;

}

**public** **void** displayBooks() {

System.***out***.println("Service Name: " + serviceName);

bookRepository.findAllBooks().forEach(System.***out***::println);

}

}

**BookRepository.java**

**package** com.library.repository;

**import** java.util.Arrays;

**import** java.util.List;

**public** **class** BookRepository {

**public** **void** saveBook(String title) {

System.***out***.println("Book saved: " + title);

}

**public** List<String> findAllBooks() {

**return** Arrays.*asList*("Book A", "Book B", "Book C");

}

}

**LibraryManagementApplication.java**

**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");

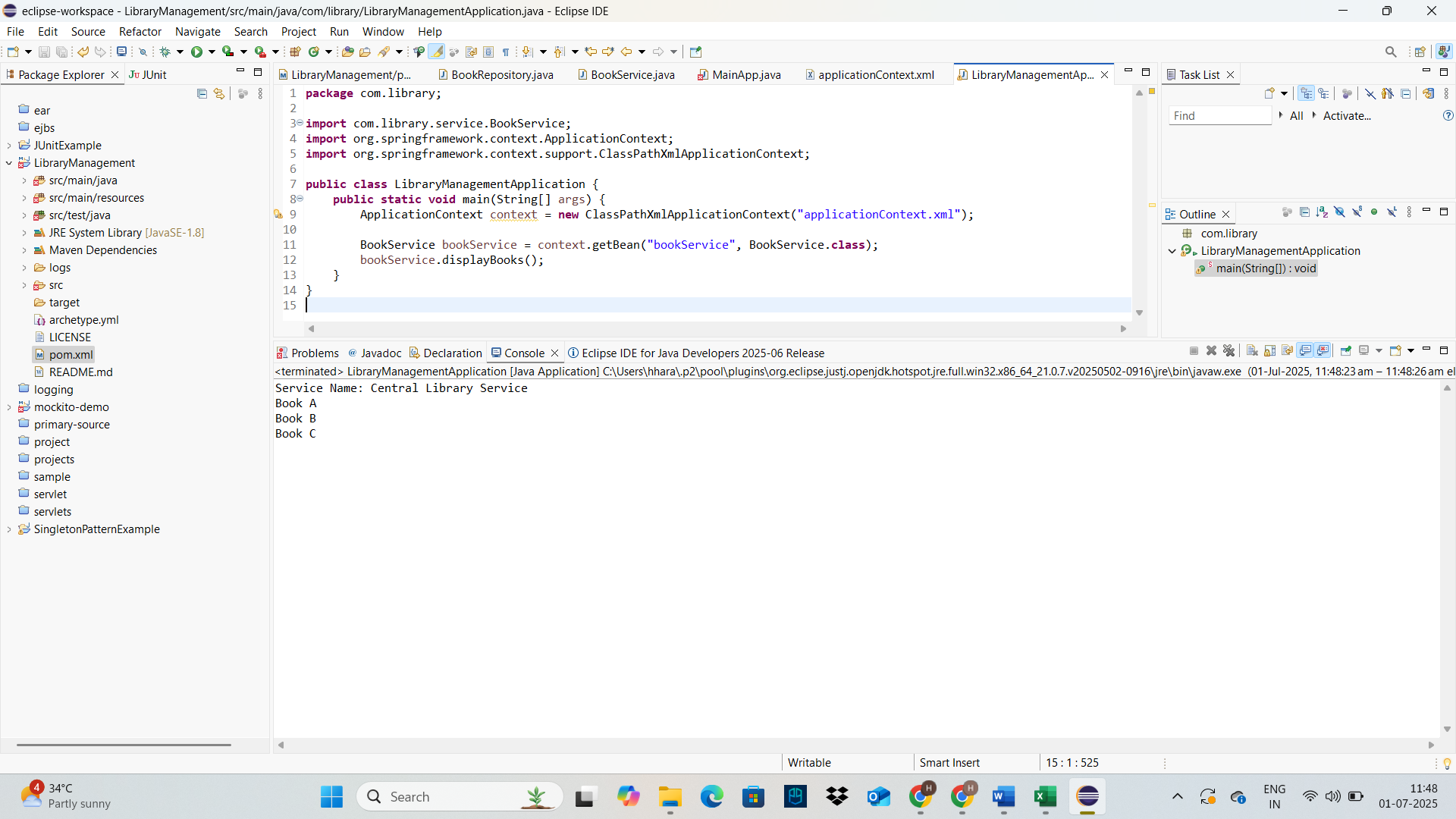
BookService bookService = context.getBean("bookService", BookService.**class**);

bookService.displayBooks();

}

}

**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.

**Steps:**

1. **Create a Spring Boot Project:**
   * Use **Spring Initializr** to create a new Spring Boot project named **LibraryManagement**.
2. **Add Dependencies:**
   * Include dependencies for **Spring Web, Spring Data JPA, and H2 Database**.
3. **Create Application Properties:**
   * Configure database connection properties in **application.properties**.
4. **Define Entities and Repositories:**
   * Create **Book** entity and **BookRepository** interface.
5. **Create a REST Controller:**
   * Create a **BookController** class to handle CRUD operations.
6. **Run the Application:**
   * Run the Spring Boot application and test the REST endpoints.

**Application.properties**

# H2 Database settings

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

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=

# JPA settings

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

spring.jpa.hibernate.ddl-auto=update

spring.h2.console.enabled=true

**Book.java**

**package** com.library.model;

**import** jakarta.persistence.\*;

@Entity

**public** **class** Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

**private** Long id;

**private** String title;

**private** String author;

// Constructors

**public** Book() {}

**public** Book(String title, String author) {

**this**.title = title;

**this**.author = author;

}

// Getters and Setters

**public** Long getId() { **return** id; }

**public** **void** setId(Long id) { **this**.id = id; }

**public** String getTitle() { **return** title; }

**public** **void** setTitle(String title) { **this**.title = title; }

**public** String getAuthor() { **return** author; }

**public** **void** setAuthor(String author) { **this**.author = author; }

}

**BookRepository.java**

**package** com.library.repository;

**import** com.library.model.Book;

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

**public** **interface** BookRepository **extends** JpaRepository<Book, Long> {

}

**BookController.java**

**package** com.library.controller;

**import** com.library.model.Book;

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

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

**import** org.springframework.web.bind.annotation.\*;

**import** java.util.List;

@RestController

@RequestMapping("/api/books")

**public** **class** BookController {

@Autowired

**private** BookRepository bookRepository;

@GetMapping

**public** List<Book> getAllBooks() {

**return** bookRepository.findAll();

}

@PostMapping

**public** Book createBook(@RequestBody Book book) {

**return** bookRepository.save(book);

}

@GetMapping("/{id}")

**public** Book getBookById(@PathVariable Long id) {

**return** bookRepository.findById(id).orElse(**null**);

}

@PutMapping("/{id}")

**public** Book updateBook(@PathVariable Long id, @RequestBody Book updatedBook) {

**return** bookRepository.findById(id).map(book -> {

book.setTitle(updatedBook.getTitle());

book.setAuthor(updatedBook.getAuthor());

**return** bookRepository.save(book);

}).orElse(**null**);

}

@DeleteMapping("/{id}")

**public** **void** deleteBook(@PathVariable Long id) {

bookRepository.deleteById(id);

}

}

**LibraryManagementApplication.java**

**package** com.library;

**import** org.springframework.boot.SpringApplication;

**import** org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

**public** **class** LibraryManagementApplication {

**public** **static** **void** main(String[] args) {

SpringApplication.run(LibraryManagementApplication.**class**, args);

}

}