**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.

**Pom.xml:**

<dependencies>

<!-- Spring Core -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.32</version>

</dependency>

</dependencies>

1. **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.

***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 for BookRepository -->

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

<!-- Bean for BookService -->

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

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

</bean>

</beans>

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

1. Create a package **com.library.service** and add a class **BookService**.
2. Create a package **com.library.repository** and add a class **BookRepository**.

BookRepository.java

package com.library.repository;

public class BookRepository {

public void save(String bookName) {

System.***out***.println("Book '" + bookName + "' saved to the repository.");

}

}

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.save(bookName);

}

}

**4.Run the Application:**

Create a main class to load the Spring context and test the configuration.

MainApp.java

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");

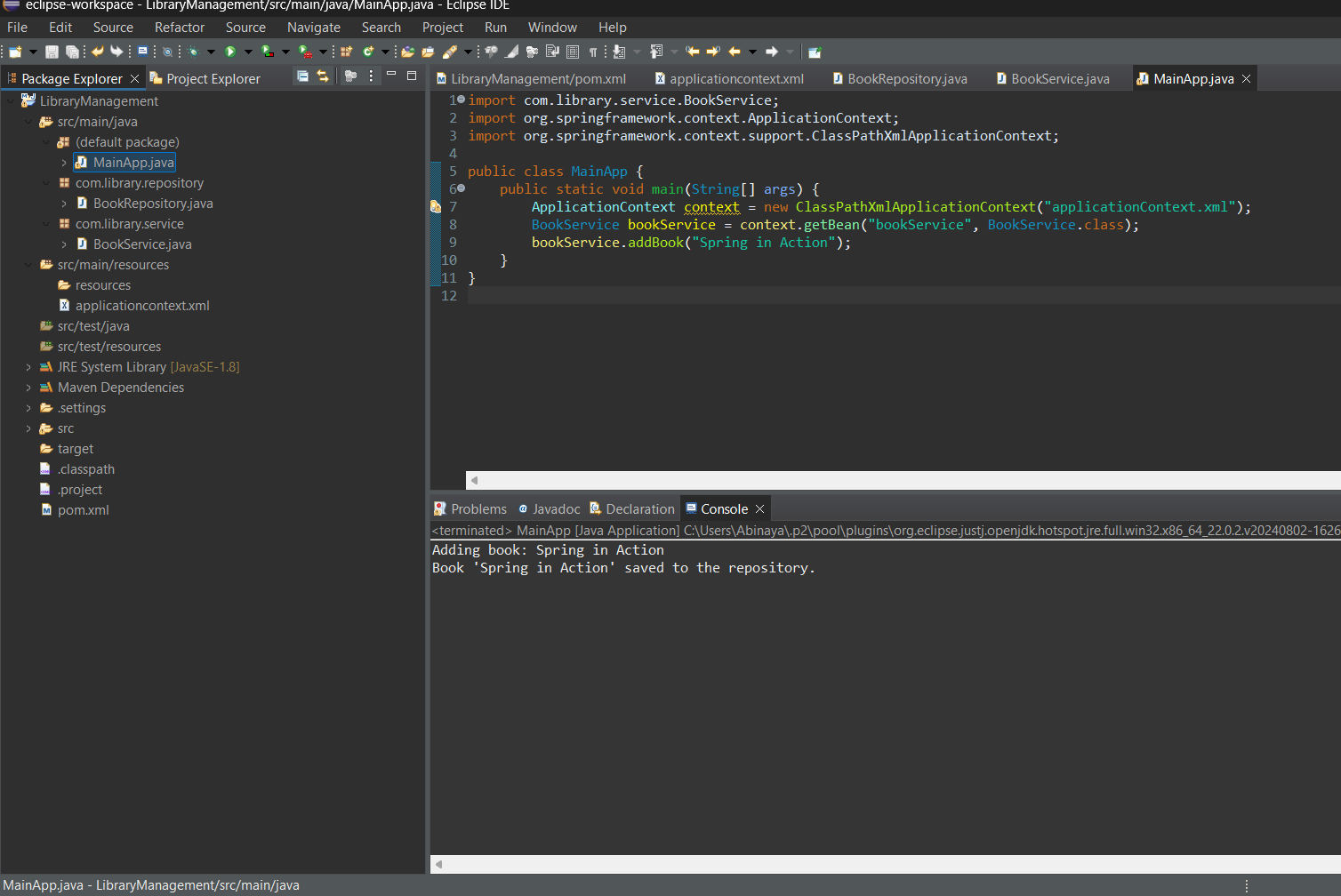
BookService bookService = context.getBean("bookService", BookService.class);

bookService.addBook("Spring in Action");

}

}

**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**.

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">

<!-- Step 1: Define the BookRepository Bean -->

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

<!-- Step 1: Define the BookService Bean and inject BookRepository -->

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

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

</bean>

</beans>

1. **Update the BookService Class:**
   * Ensure that **BookService** class has a setter method for **BookRepository**.

BookRepository.java

package com.example.repository;

public class BookRepository {

public void save() {

System.out.println("Saving book into the repository...");

}}

BookService .java

package com.example.service;

import com.example.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

// Setter for DI

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void performService() {

System.***out***.println("BookService is working...");

bookRepository.save();

}

}

1. **Test the Configuration:**
   * Run the **LibraryManagementApplication** main class to verify the dependency injection.

LibraryManagementApplication.java

package com.example.librarymanagement;

import com.example.librarymanagement.service.BookService;

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

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class LibraryManagementApplication implements CommandLineRunner {

@Autowired

private BookService bookService;

public static void main(String[] args) {

SpringApplication.run(LibraryManagementApplication.class, args);

}

@Override

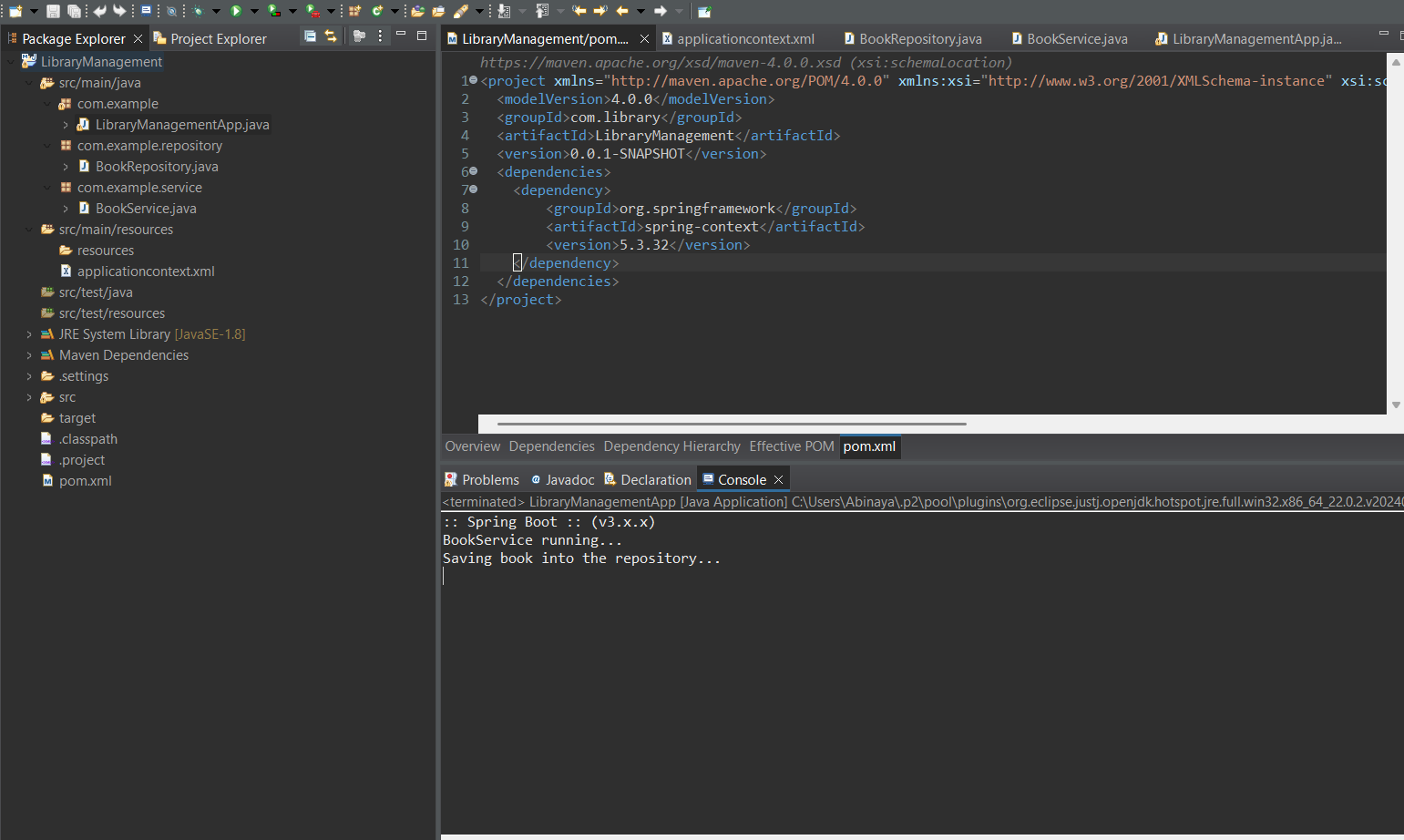
public void run(String... args) {

bookService.performService();

}

}

**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.

**1.MainApp.java**

package com.library;

public class MainApp {

public static void main(String[] args) {

System.*out*.println("Library Management Application Started");

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

} }

**Output:**

