**Spring Core and Maven**

**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.

**Step 1: Project Setup**

* Create a new Maven project named 'LibraryManagement' in Eclipse.
* Update the 'pom.xml' file to include Spring Core dependency:

<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

https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.library</groupId>

<artifactId>LibraryManagement</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.33</version>

</dependency>

</dependencies>

</project>

**Step 2: Create Packages and Java Classes**

* Create Packages and Classes
* com.library
* com.library.repository
* com.library.service
* Create the following Java classes:
* BookRepository.java (in repository package)
* BookService.java (in service package)
* MainApp.java (in com.library package)

**BookRepository.java**

**CODE:**

**package** com.library.repository;

**import** org.springframework.stereotype.Repository;

@Repository

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

**public** **void** showAllBooks() {

System.***out***.println("Book list: Java, Spring, Hibernate, etc.");

}

}

**BookService.java**

**CODE:**

**package** com.library.service;

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

@Service("bookService")

**public class** BookService {

@Autowired

**private** BookRepository bookRepository;

**public void** displayBooks() {

System.***out***.println("Displaying books from the repository...");

bookRepository.showAllBooks();

}

}

**MainApp.java**

**CODE:**

**package** com.library;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**import** com.library.service.BookService;

**public** **class** MainApp {

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

**try** (ClassPathXmlApplicationContext context = **new** ClassPathXmlApplicationContext("applicationContext.xml")) {

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

bookService.displayBooks();

}

}

}

**Step 3: applicationContext.xml**

* Create 'applicationContext.xml' file in 'src/main/resources' directory and add the following:

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:context="http://www.springframework.org/schema/context"

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/context

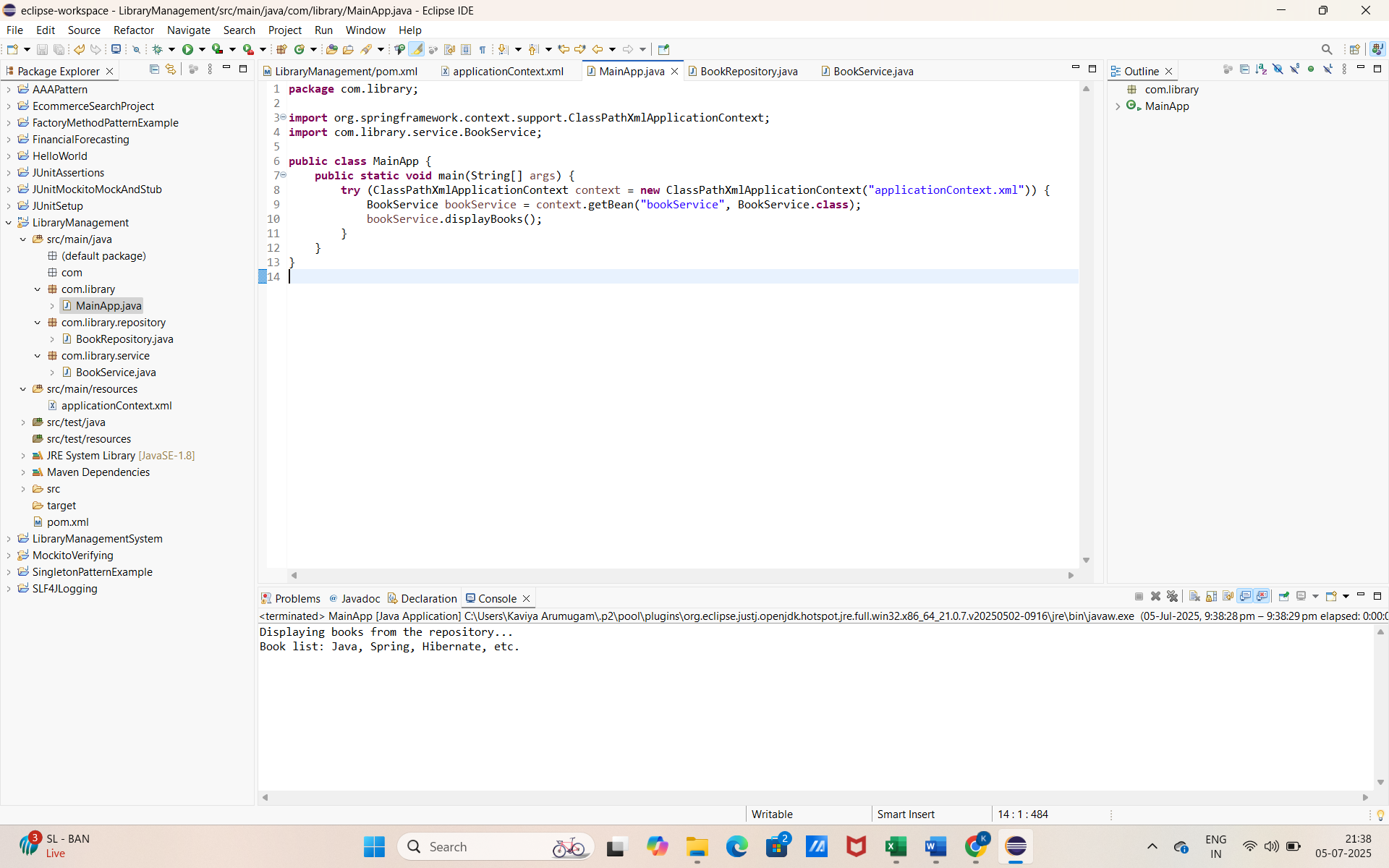
http://www.springframework.org/schema/context/spring-context.xsd">

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

</beans>

**Step 4: Running the Application**

**OUTPUT:**



**Exercise 2: Implementing Dependency Injection**

**Scenario:**

In the same library management application, you now need to manually wire the dependency between BookService and BookRepository using Spring's XML-based Dependency Injection.

**Step 1: Modify applicationContext.xml**

* Instead of using @Autowired and @Service, you'll configure the beans manually.
* Update your applicationContext.xml like this:

<?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">

<!-- Define Repository Bean -->

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

<!-- Define Service Bean and Inject Dependency -->

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

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

</bean>

</beans>

**Step 2: Update BookService.java**

* Remove annotations like @Autowired and @Service, and include a **setter method** for bookRepository.

**BookService.java**

**CODE:**

**package** com.library.service;

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

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

**private** BookRepository bookRepository;

// Setter method for DI

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

**this**.bookRepository = bookRepository;

}

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

System.***out***.println("Displaying books from the repository...");

bookRepository.showAllBooks();

}

}

**Step 3: Run the Application**

* Use the same MainApp.java to test the DI setup

**OUTPUT:**

A screenshot of a computer

AI-generated content may be incorrect.

**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.

**Step 1: Create a New Maven Project**

* This step was already done in Exercise 1. We are continuing in the same project: LibraryManagement.

**Step 2: Add Spring Dependencies**

* You need to **add two more Spring dependencies** to your existing <dependencies> block:

<!-- Spring AOP -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.33</version>

</dependency>

<!-- Spring WebMVC -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.33</version>

</dependency>

A screenshot of a computer

AI-generated content may be incorrect.

**Step 3: Add Maven Compiler Plugin**

* Scroll **below the <dependencies> section** and add the <build> block like this:

<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>

A computer screen shot of a computer screen

AI-generated content may be incorrect.

**Step 4: Update Maven Project**

* After saving the pom.xml, update your project in Eclipse:

1. Right-click the project → **Maven → Update Project**
2. Click **OK** in the popup