**WEEK -3**

**SUPERSET ID: 6384042**

**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.

## **1. Set Up a Spring Project**

- Open Eclipse → File → New → Maven Project.  
- Choose archetype: maven-archetype-quickstart.  
- Set:  
 Group Id: com.library  
 Artifact Id: LibraryManagement  
- Click Finish to generate the project.  
- Open pom.xml and add Spring Core dependency:

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.34</version>

</dependency>

## **2. Configure the Application Context**

- Create the file applicationContext.xml under src/main/resources.  
- Define the Spring beans for BookService and BookRepository as follows:

<?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>

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

### **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;

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void addBook(String title) {

System.out.println("Adding book...");

bookRepository.saveBook(title);

}}

## **4. Run the Application**

- Create a main class named MainApp in a package com.library.  
- Load the Spring context and test the configuration using the following code:

### **MainApp.java**

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");

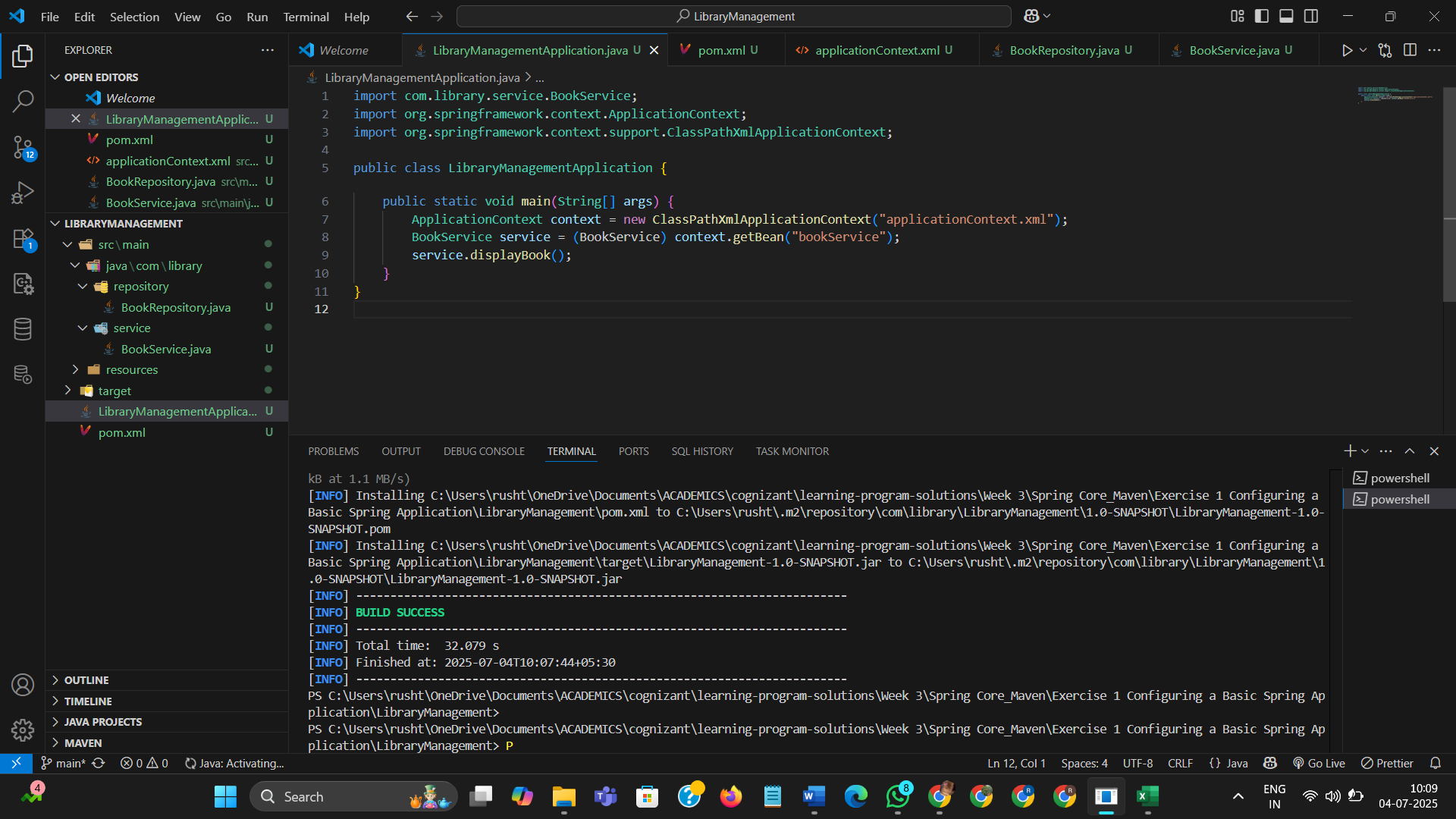
BookService bookService = (BookService) context.getBean("bookService");

bookService.addBook("The Alchemist");

}

}

**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.

# **1. Modify the XML Configuration**

# - Update applicationContext.xml to wire BookRepository into BookService as shown below:

<?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>

**2. Update the BookService Class**

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

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 title) {

System.out.println("Adding book...");

bookRepository.saveBook(title);

}

}

**3. Test the Configuration**

- Run the LibraryManagementApplication main class (MainApp.java) to verify the dependency injection.

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");

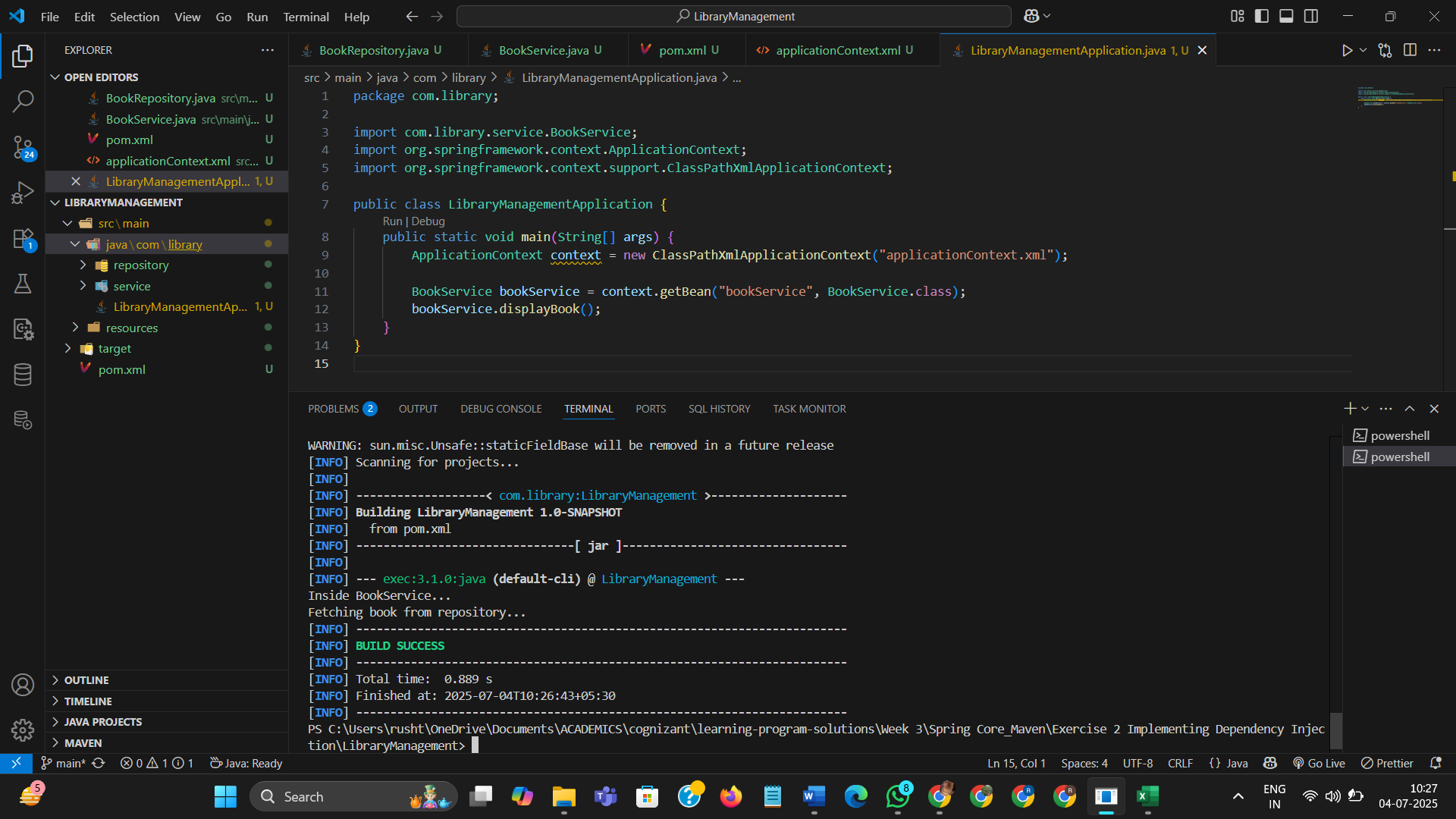
BookService bookService = (BookService) context.getBean("bookService");

bookService.addBook("The Alchemist");

}

}

**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.

**1. Create a New Maven Project**

- Open Eclipse → File → New → Maven Project.

- Choose archetype: maven-archetype-quickstart.

- Set:

Group Id: com.library

Artifact Id: LibraryManagement

- Click Finish to generate the project.

**2. Add Spring Dependencies in pom.xml**

- Open the pom.xml file and include the following dependencies for Spring Context, AOP, and WebMVC:

<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>

</dependencies>

**3. Configure Maven Plugins**

- Add the Maven Compiler Plugin to the pom.xml file to specify Java version 1.8:

<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>

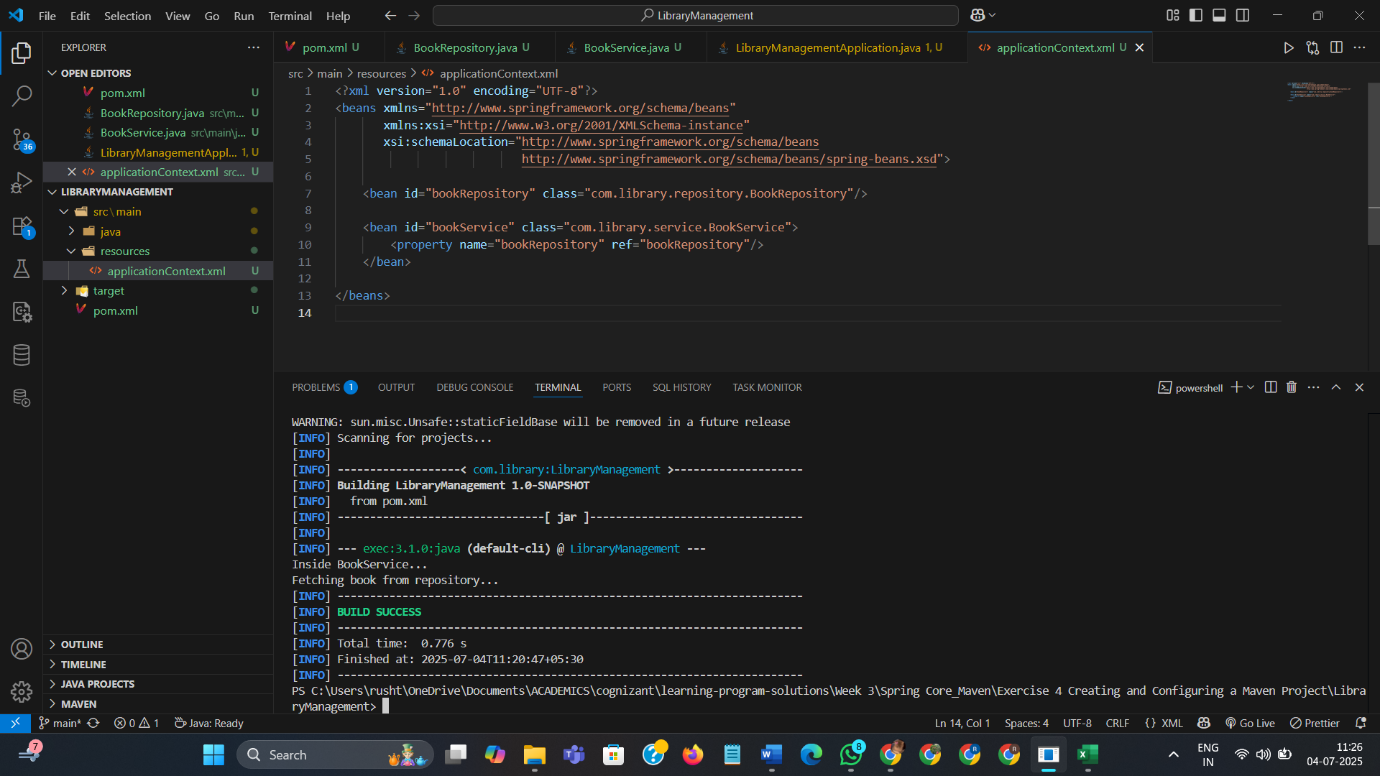
</configuration>

</plugin>

</plugins>

</build>

**Output:**

****