**Cognizant Digital Nurture 4.0-Week(3)**

**Name: Gandu Lasya Sri**

**Email: gandulasyasri@gmail.com**

**Superset ID:6428164**

**Mandatory Hands-On Exercises**

**Spring Core and Maven**

**Exercise 1: Configuring a Basic Spring Application**

**Step 1: Set Up a Spring Project**  
I created a Maven project and named it LibraryManagement. Then I added the Spring Core dependency in the pom.xml file to use the Spring features.

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.36</version>

</dependency>

I also added the exec plugin to run the main class easily:

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>exec-maven-plugin</artifactId>

<version>3.1.0</version>

<configuration>

<mainClass>com.library.MainApp</mainClass>

</configuration>

</plugin>

**Step 2: Configure the Application Context**  
I created a file called applicationContext.xml inside src/main/resources folder. In this file, I added beans for BookService and BookRepository.

<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>

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

I created a folder com.library.service and added a class BookService.java with the following code:

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 perform() {

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

bookRepository.display();

}

}

I created a folder com.library.repository and added a class

BookRepository.java with the following code:

package com.library.repository;

public class BookRepository {

public void display() {

System.out.println("BookRepository is working...");

}

}

**Step 4: Run the Application**

I created a MainApp.java class inside com.library package to load the Spring context and test the configuration.

package com.library;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

public class MainApp {

public static void main(String[] args) {

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

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

bookService.perform();

((ClassPathXmlApplicationContext) context).close();

}

}

Then I opened terminal and ran the application using:

mvn clean install

mvn exec:java

It worked successfully and printed:

BookService is working...

BookRepository is working…

OUTPUT:  
**Exercise 2: Implementing Dependency Injection**

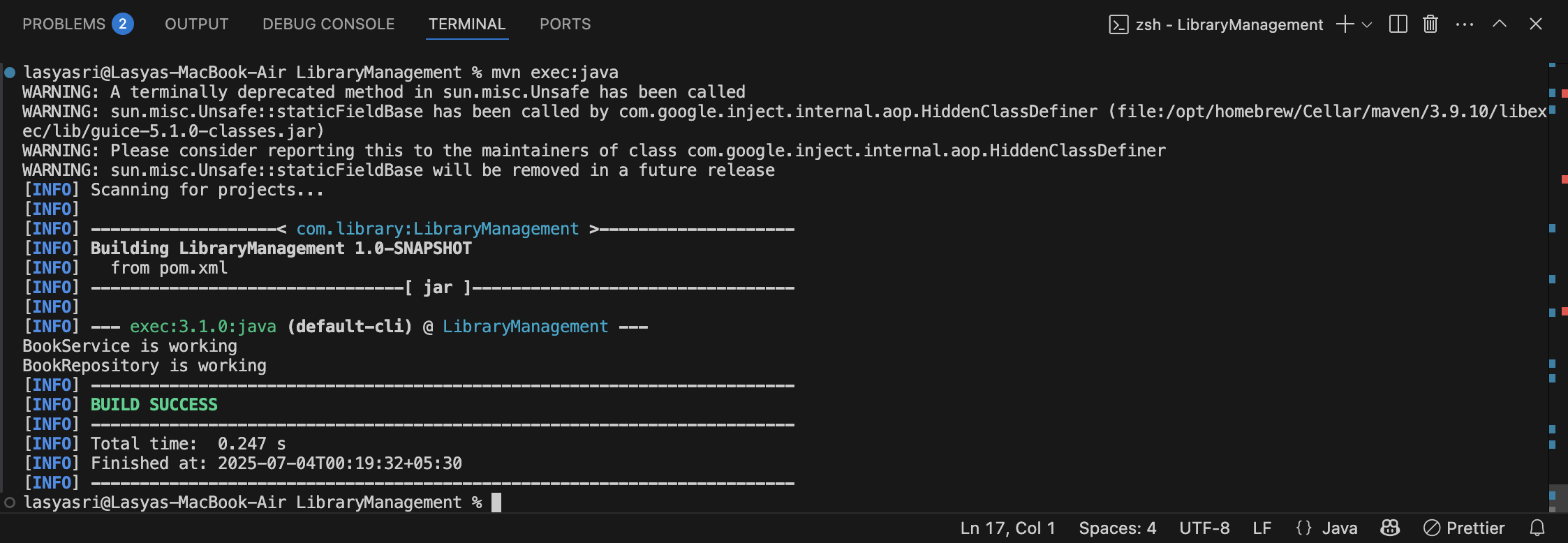
**Step 1: Modify the XML Configuration**  
In the applicationContext.xml, we add a property tag inside the bookService bean to inject the bookRepository bean.

<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>

**Step 2: Update the BookService Class**  
We create a setter method for bookRepository in BookService class so Spring can inject it.

src/main/java/com/library/service/BookService.java

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 perform() {

System.out.println("BookService: Calling repository...");

bookRepository.getBook();

}

}

**Step 3: BookRepository Class**  
We create a simple BookRepository class.

src/main/java/com/library/repository/BookRepository.java

package com.library.repository;

public class BookRepository {

public void getBook() {

System.out.println("BookRepository: Getting book data...");

}

}

**Step 4: Create the Main Class to Run**  
This class will load the Spring context and test DI.

src/main/java/com/library/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 = (BookService) context.getBean("bookService");

bookService.perform();

((ClassPathXmlApplicationContext) context).close();

}

}

**How I Ran This in VS Code (Step-by-step):**

1. I opened the LibraryManagement project in VS Code.
2. I created all the necessary Java classes inside src/main/java.
3. I added the applicationContext.xml file inside src/main/resources.
4. I made sure my pom.xml has Spring dependency and the exec plugin.
5. I opened the terminal and ran the command:

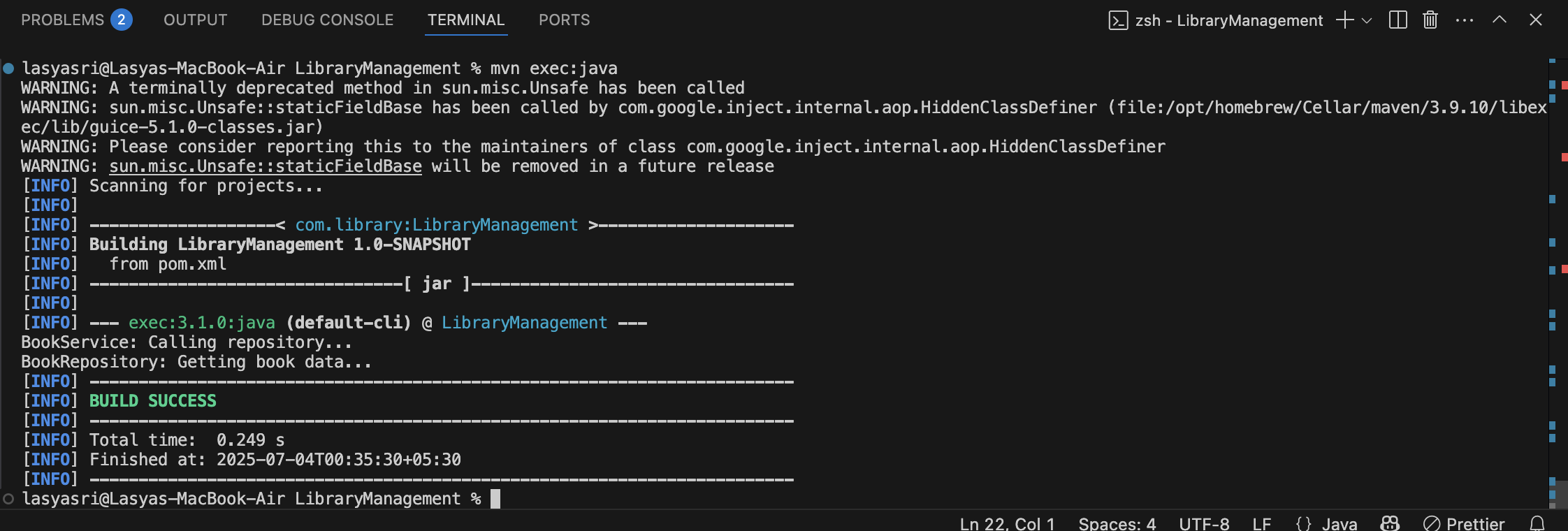
mvn clean compile exec:java

1. I saw the output:

BookService: Calling repository...

BookRepository: Getting book data...

This confirmed that dependency injection worked using Spring XML configuration.

OUTPUT:

**Exercise 4: Creating and Configuring a Maven Project**

**Scenario:**  
We are working on a library management application and now we need to set up the project using Maven and add Spring dependencies to manage our backend more effectively.

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

I already had a folder named LibraryManagement from previous exercises, so I used the same folder. I just made sure it has the proper Maven folder structure:

LibraryManagement/

├── pom.xml

├── src/

├── main/

│ ├── java/

│ └── resources/

└── test/

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

I opened the pom.xml file and added these dependencies inside the <dependencies> tag:

<!-- Spring Core Dependency -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.36</version>

</dependency>

<!-- Spring AOP Dependency -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.36</version>

</dependency>

<!-- Spring WebMVC Dependency -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.36</version>

</dependency>

<!-- JUnit for testing -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>3.8.1</version>

<scope>test</scope>

</dependency>

**Step 3: Configure Maven Plugins**

To make sure the project compiles properly using Java 1.8, I added this plugin inside the <build> tag:

<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>

**Conclusion:**  
I successfully configured my Maven project by setting up dependencies and plugins. Now my project is ready for running Spring-based Java applications smoothly.