

## 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.

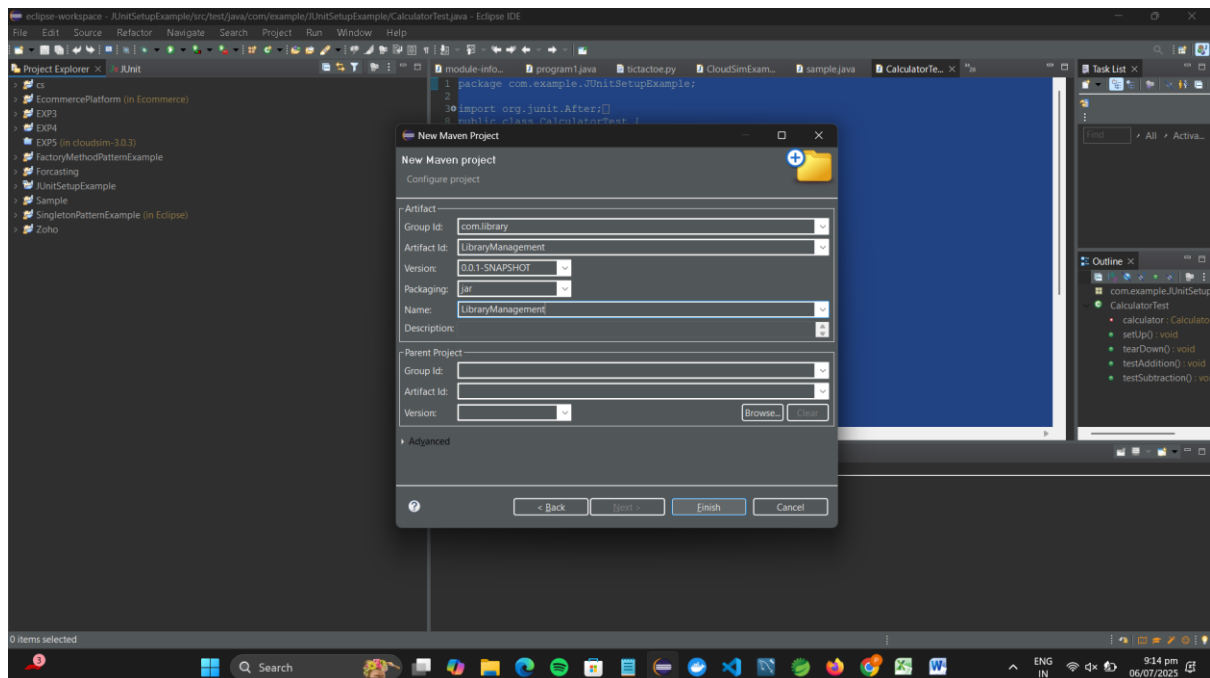
### Steps:

#### Set Up a Spring Project:

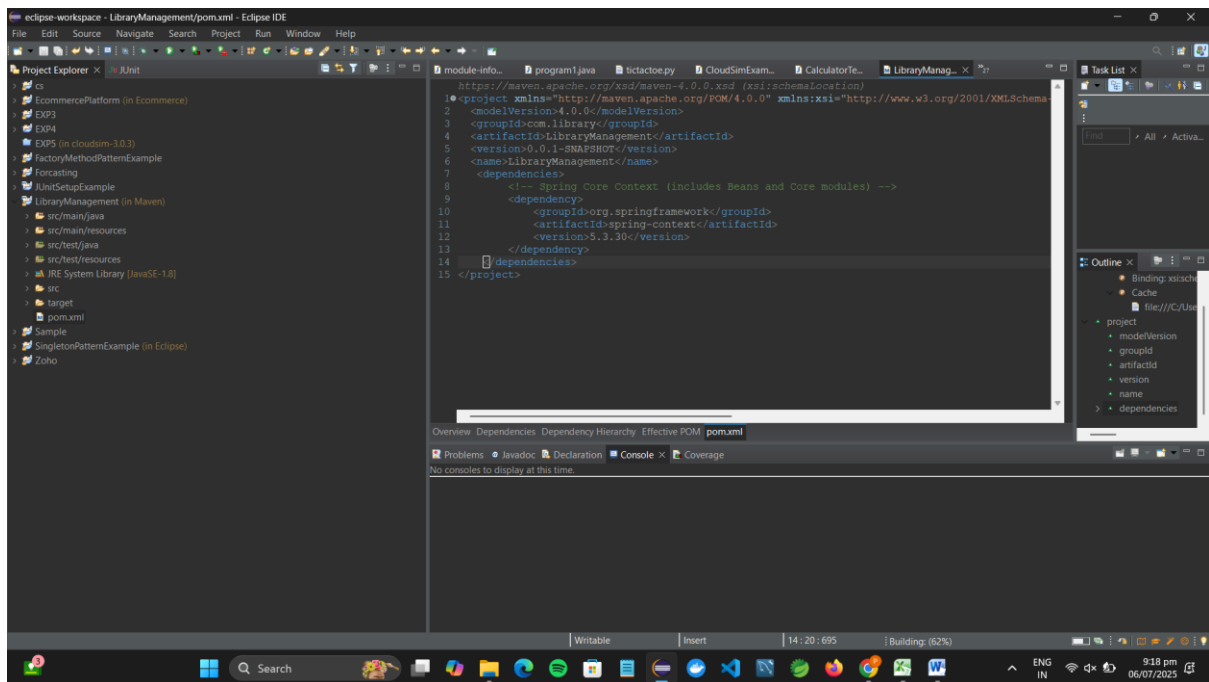
- Create a Maven project named **LibraryManagement**.

### Open Eclipse:

1. Go to File → New → Project
2. Select Maven → Maven Project → **Next**
3. Choose "**Create a simple project**" → **Next**
4. Fill:
  - **Group Id:** com.library
  - **Artifact Id:** LibraryManagement
  - **Version:** (default is fine)
  - **Packaging:** jar
  - Click **Finish**.

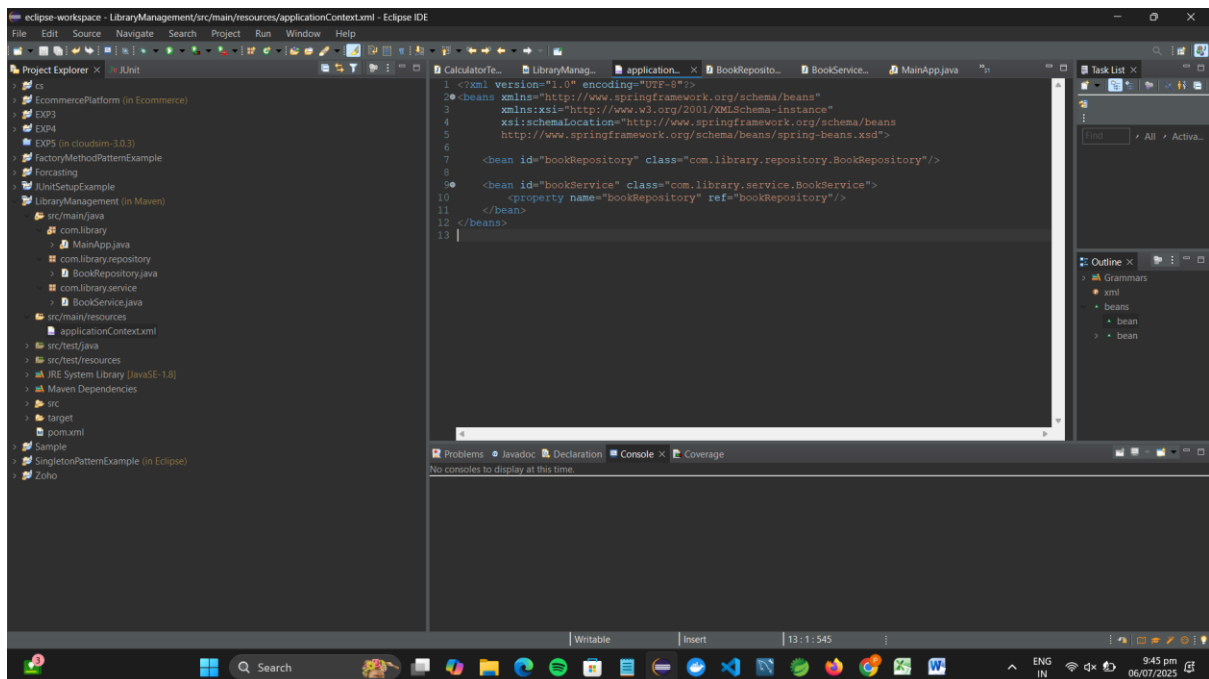


Add Spring Core dependencies in the **pom.xml** file.



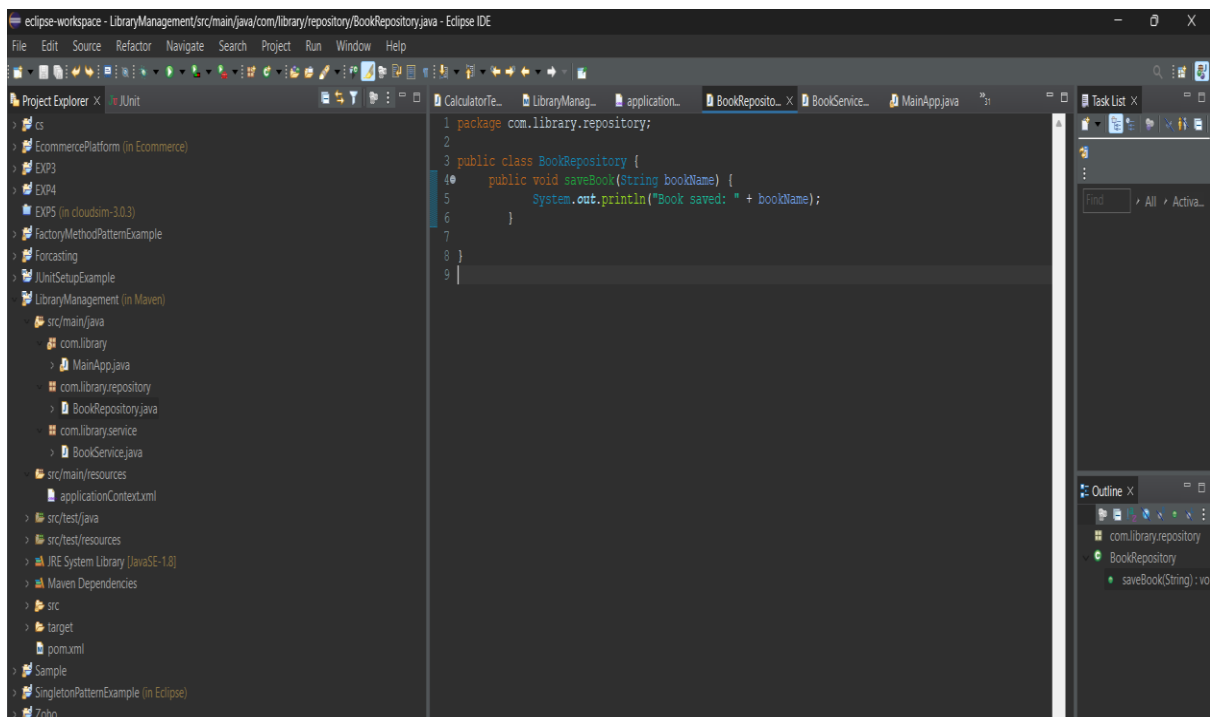
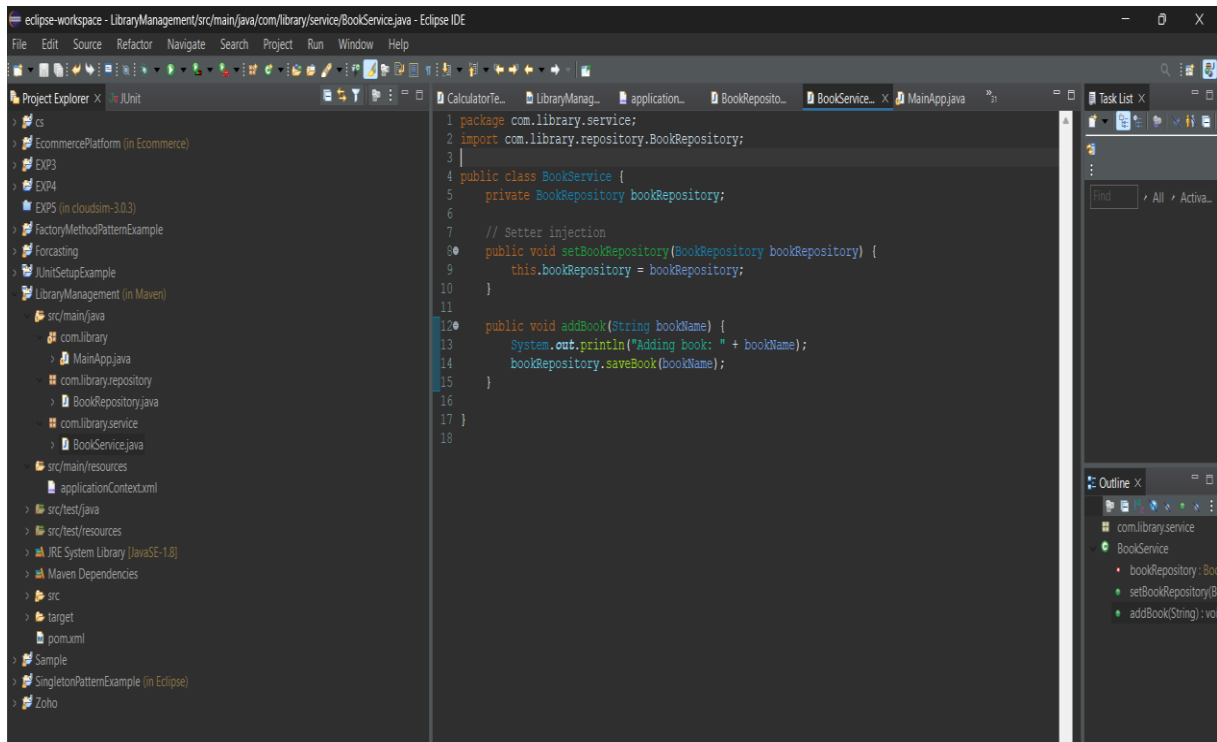
### Configure the Application Context:

- Create an XML configuration file named **applicationContext.xml** in the **src/main/resources** directory.
- Define beans for **BookService** and **BookRepository** in the XML file.



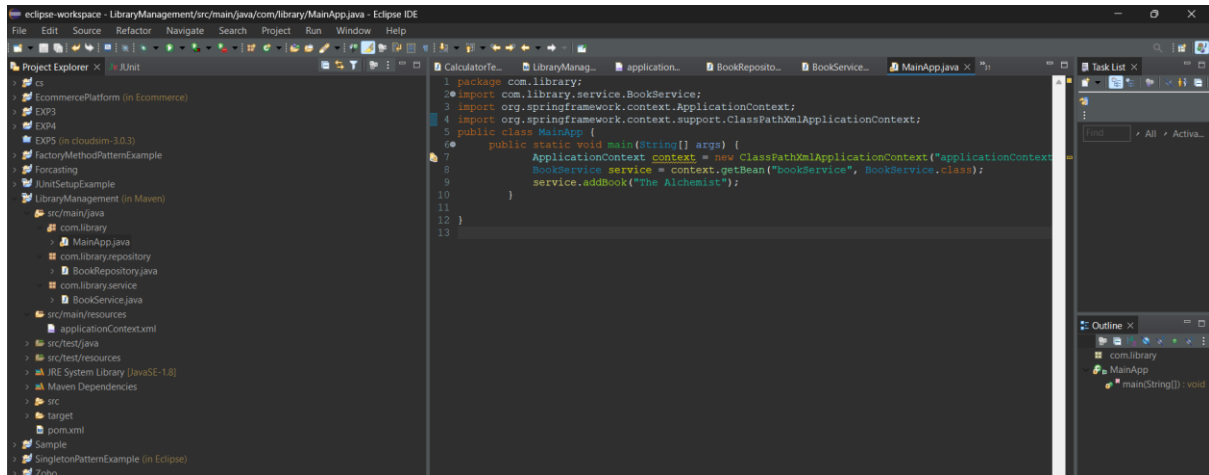
## Define Service and Repository Classes:

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



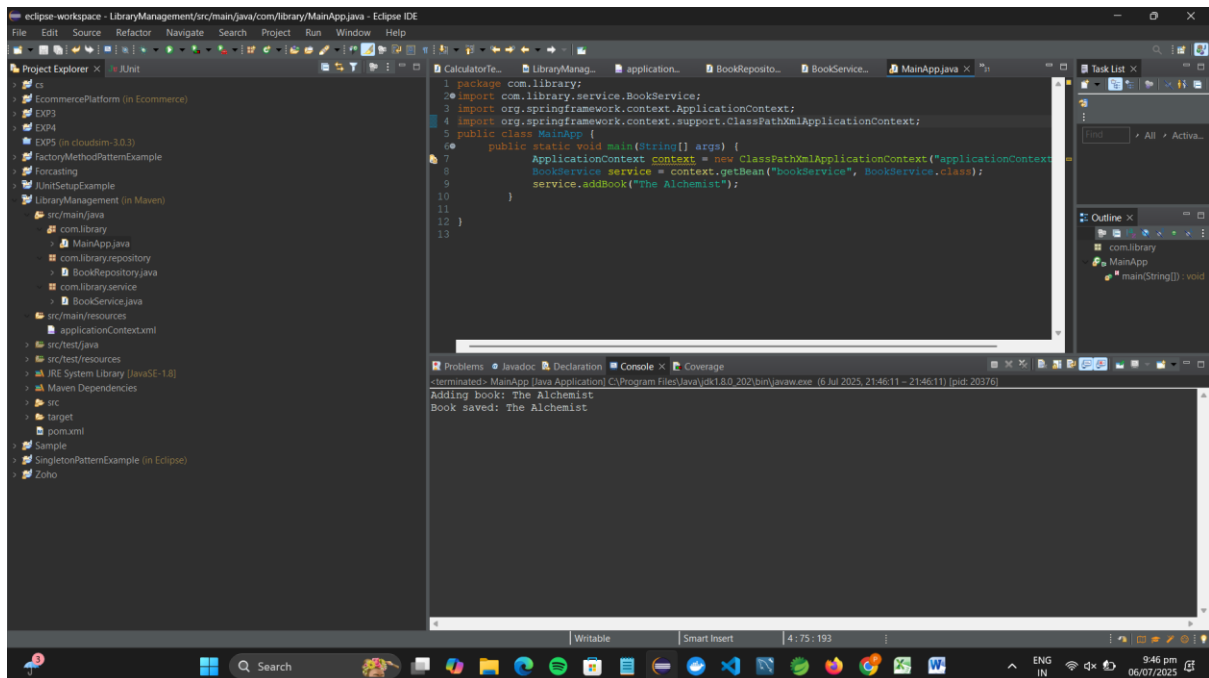
## Run the Application:

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



Right-click on MainApp.java → Run As → Java Application

Output Screenshot:



## Exercise 2: Implementing Dependency Injection

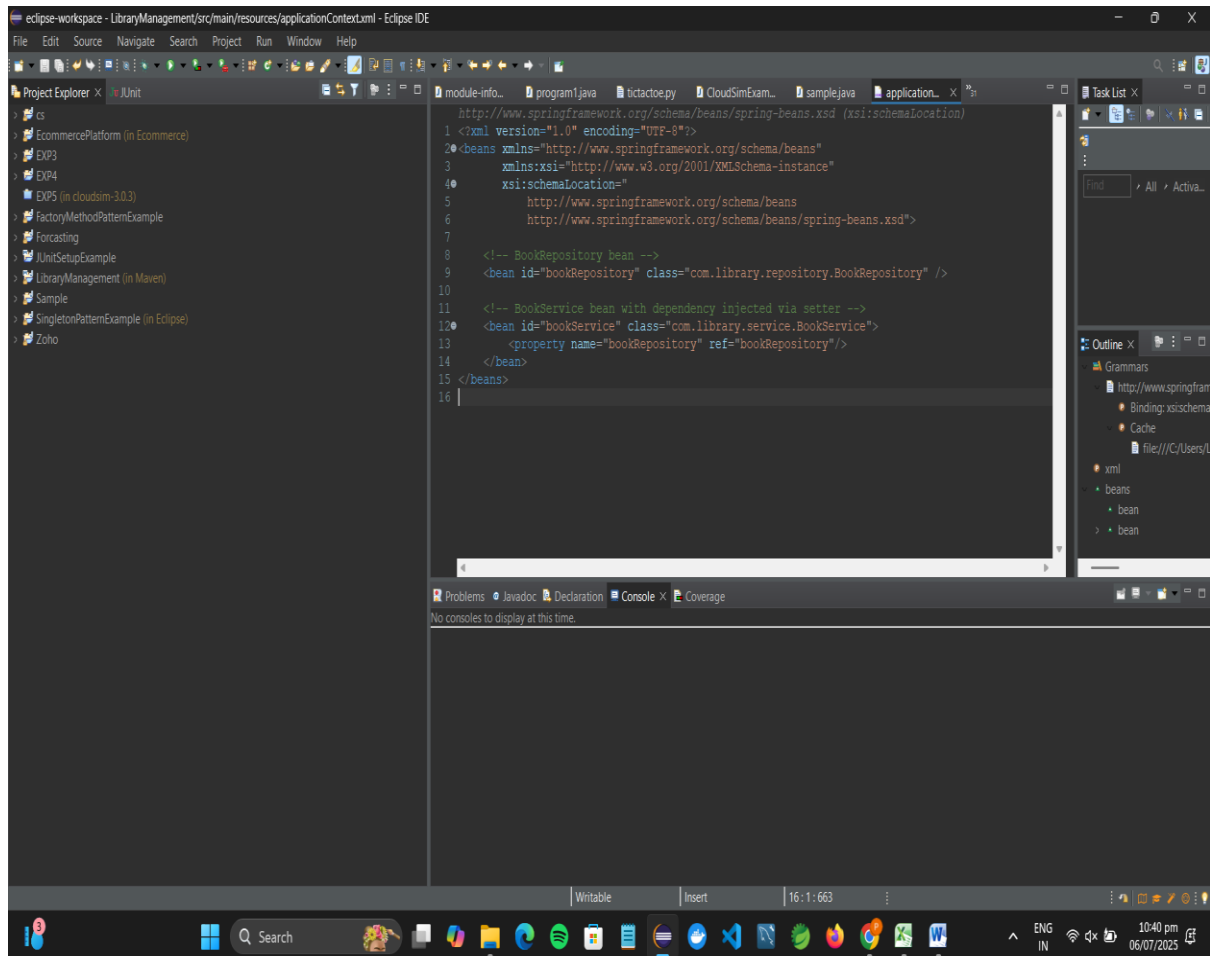
### Scenario:

In the library management application, you need to manage the dependencies between the `BookService` and `BookRepository` classes using Spring's IoC and DI.

### Steps:

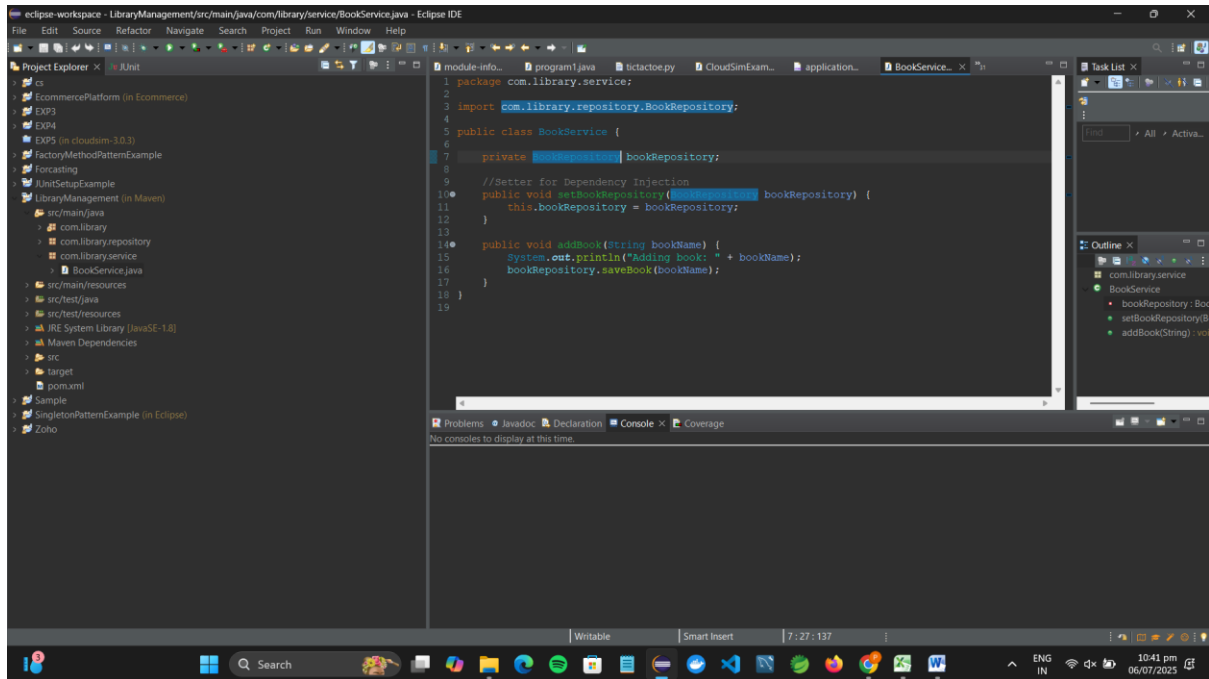
#### 1. Modify the XML Configuration:

- Update **applicationContext.xml** to wire **BookRepository** into **BookService**.



## 2. Update the BookService Class:

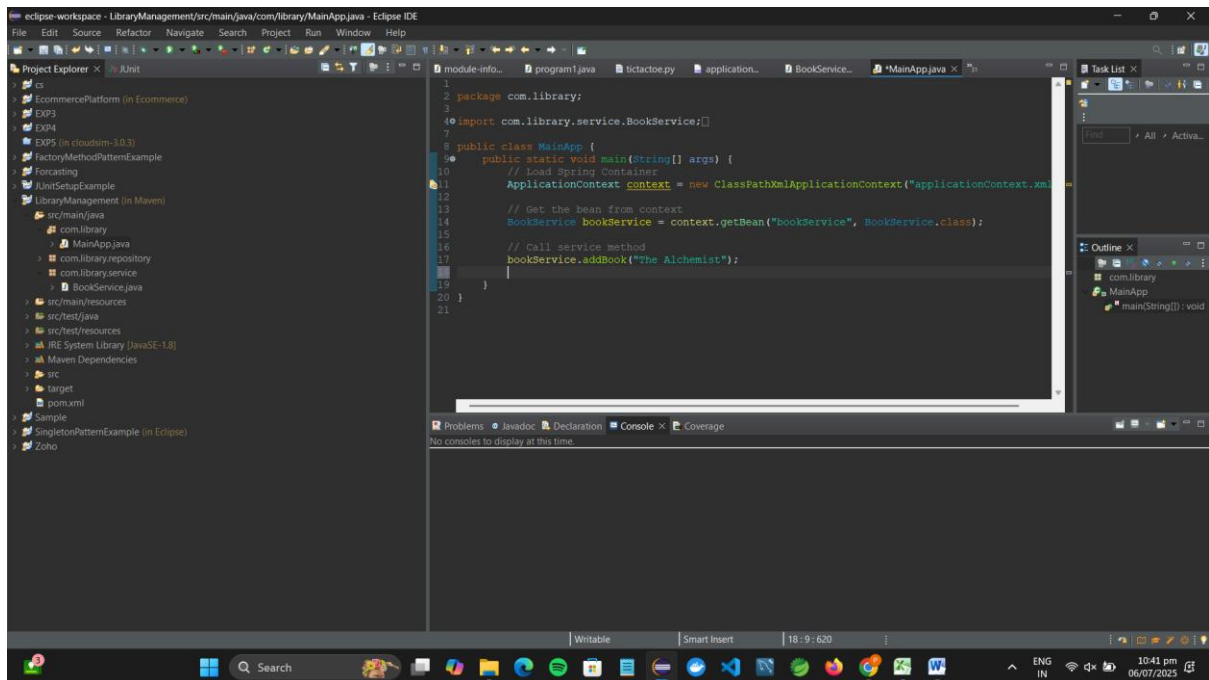
- Ensure that **BookService** class has a setter method for **BookRepository**.



```
1 package com.library.service;
2
3 import com.library.repository.BookRepository;
4
5 public class BookService {
6
7     private BookRepository bookRepository;
8
9     //Setter for Dependency Injection
10    public void setBookRepository(BookRepository bookRepository) {
11        this.bookRepository = bookRepository;
12    }
13
14    public void addBook(String bookName) {
15        System.out.println("Adding book: " + bookName);
16        bookRepository.saveBook(bookName);
17    }
18
19 }
```

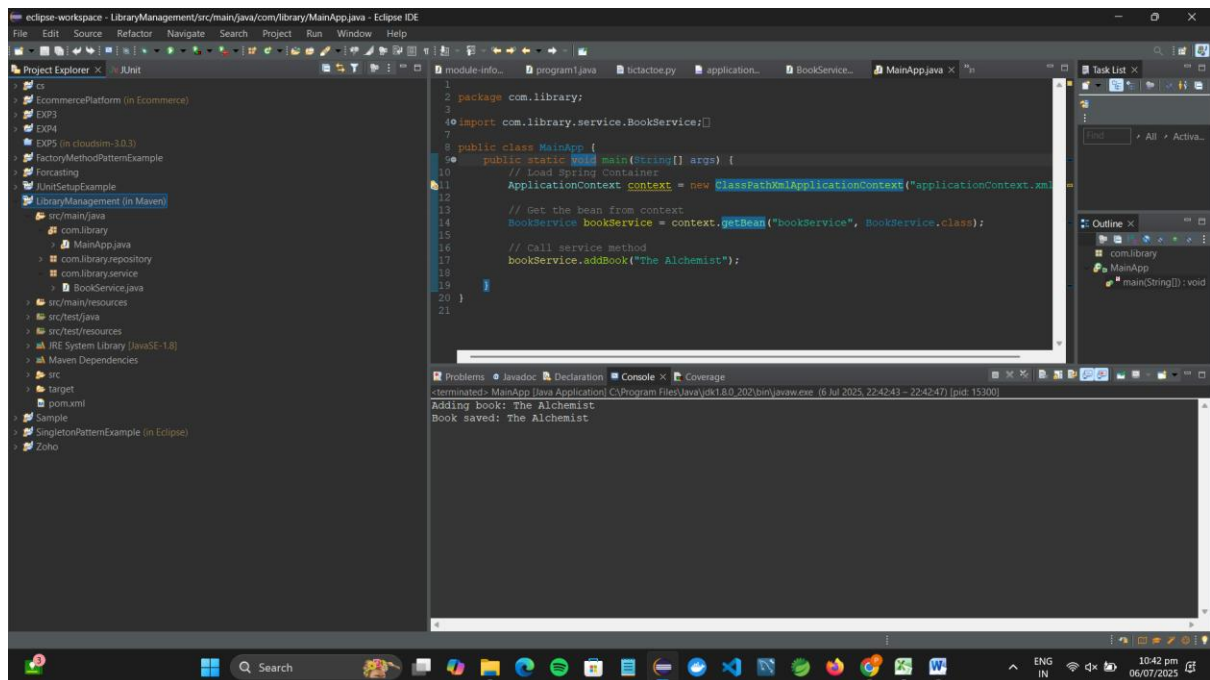
## 3. Test the Configuration:

- Run the **LibraryManagementApplication** main class to verify the dependency injection.



```
1 package com.library;
2
3 import com.library.service.BookService;
4
5 public class MainApp {
6
7     public static void main(String[] args) {
8         // Load Spring Container
9         ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");
10
11         // Get the bean from context
12         BookService bookService = context.getBean("bookService", BookService.class);
13
14         // Call service method
15         bookService.addBook("The Alchemist");
16     }
17
18 }
```

## Output Screenshot:



## Result:

Component	Purpose
applicationContext.xml	Wires bookRepository into bookService using DI
BookService.java	Has a setter method to accept the injected dependency
MainApp.java	Loads Spring context and verifies DI works

## 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.

Code:

```
<dependencies>
  <!-- Spring Core/Context -->
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>5.3.30</version>
  </dependency>

  <!-- Spring AOP (for Aspect-Oriented Programming) -->
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-aop</artifactId>
    <version>5.3.30</version>
  </dependency>

  <!-- Spring Web MVC -->
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.3.30</version>
  </dependency>
</dependencies>
```

#### 3. Configure Maven Plugins:

- Configure the Maven Compiler Plugin for Java version 1.8 in the pom.xml file.

```
<plugins>
  <!-- Maven Compiler Plugin for Java 1.8 -->
  <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>

```

Updated pom.xml:

The screenshot shows the Eclipse IDE with the 'pom.xml' file open in the editor. The Project Explorer on the left shows the project structure, including 'src/main/java' and 'src/main/resources'. The editor displays the following XML content:

```

<?xml version="1.0" encoding="UTF-8" ?>
<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>
  <packaging>jar</packaging>
  <name>LibraryManagement</name>
  <properties>
    <java.compiler.source>1.8</java.compiler.source>
    <java.compiler.target>1.8</java.compiler.target>
  </properties>
  <dependencies>
    <!-- Spring Core/Context -->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>5.3.30</version>
    </dependency>
    <!-- Spring AOP (for Aspect-Oriented Programming) -->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-aop</artifactId>
      <version>5.3.30</version>
    </dependency>
    <!-- Spring Web MVC -->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
      <version>5.3.30</version>
    </dependency>
  </dependencies>
</project>

```

The console output at the bottom shows the message: "Spring Context Loaded Successfully!".

MainApp.java:

The screenshot shows the Eclipse IDE with the 'MainApp.java' file open in the editor. The Project Explorer on the left shows the project structure, including 'src/main/java' and 'src/main/resources'. The editor displays the following Java code:

```

package com.library;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {
    public static void main(String[] args) {
        ClassPathXmlApplicationContext context =
            new ClassPathXmlApplicationContext("applicationContext.xml");
        System.out.println("Spring Context Loaded Successfully!");
        context.close();
    }
}

```

The console output at the bottom shows the message: "Spring Context Loaded Successfully!".

## Output Screenshot:

