

## 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`.

#### Output:

```
http://www.springframework.org/schema/beans/spring-beans.xsd (xsi:schemaLocation)
1 <?xml version="1.0" encoding="UTF-8"?>
2 <beans xmlns="http://www.springframework.org/schema/beans"
3     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4     xsi:schemaLocation="http://www.springframework.org/schema/beans
5         http://www.springframework.org/schema/beans/spring-beans.xsd">
6
7     <!-- Repository Bean -->
8     <bean id="bookRepository" class="com.library.repository.BookRepository" />
9
10    <!-- Service Bean -->
11    <bean id="bookService" class="com.library.service.BookService">
12        <property name="bookRepository" ref="bookRepository"/>
13    </bean>
14
15 </beans>
```

#### 2. Update the BookService Class:

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

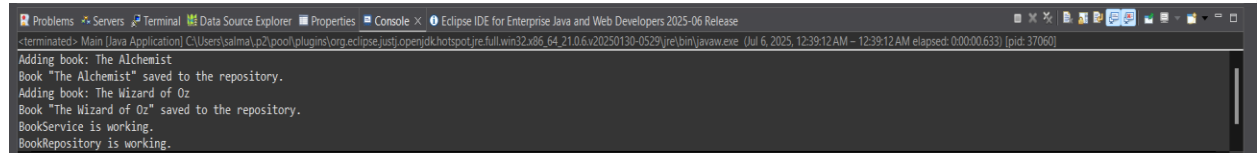
#### Output:

```
LibraryManagement/pom.xml  applicationContext.xml  applicationContext.xml  BookService.java
1 package com.library.service;
2 import com.library.repository.BookRepository;
3
4 public class BookService {
5     private BookRepository bookRepository;
6
7     public void setBookRepository(BookRepository bookRepository) {
8         this.bookRepository = bookRepository;
9     }
10
11     public void addBook(String bookName) {
12         System.out.println("Adding book: " + bookName);
13         bookRepository.saveBook(bookName);
14     }
15
16     public void performService() {
17         System.out.println("BookService is working.");
18         bookRepository.performRepositoryAction();
19     }
20 }
```

### 3. Test the Configuration:

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

#### Output:

A screenshot of the Eclipse IDE's console window. The window title is "Eclipse IDE for Enterprise Java and Web Developers 2025-06 Release". The console shows the following output: "<terminated> Main [Java Application] C:\Users\salma\AppData\Local\Temp\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86\_64.21.0.6\v20250130-0529\re\bin\javaw.exe (Jul 6, 2025, 12:39:12 AM - 12:39:12 AM elapsed: 0:00:00.633) [pid: 37060]". Below this, the application output is: "Adding book: The Alchemist", "Book 'The Alchemist' saved to the repository.", "Adding book: The Wizard of Oz", "Book 'The Wizard of Oz' saved to the repository.", "BookService is working.", and "BookRepository is working.".

```
<terminated> Main [Java Application] C:\Users\salma\AppData\Local\Temp\org.eclipse.justi.openjdk.hotspot.jre.full.win32.x86_64.21.0.6\v20250130-0529\re\bin\javaw.exe (Jul 6, 2025, 12:39:12 AM - 12:39:12 AM elapsed: 0:00:00.633) [pid: 37060]
Adding book: The Alchemist
Book 'The Alchemist' saved to the repository.
Adding book: The Wizard of Oz
Book 'The Wizard of Oz' saved to the repository.
BookService is working.
BookRepository is working.
```

- In this exercise, **Spring's Inversion of Control (IoC) container** manages object creation and dependency injection. The IoC container is configured **applicationContext.xml** where beans are defined and their relationships are established.
- The **BookService** class depends on **BookRepository** to perform backend data operations. Instead of manually instantiating the repository within the service class, Spring injects the dependency using **setter-based Dependency Injection**. This is achieved using the **<property>** tag in the **applicationContext.xml**