**Exercise 2: Implementing Dependency Injection**

**Scenario:**

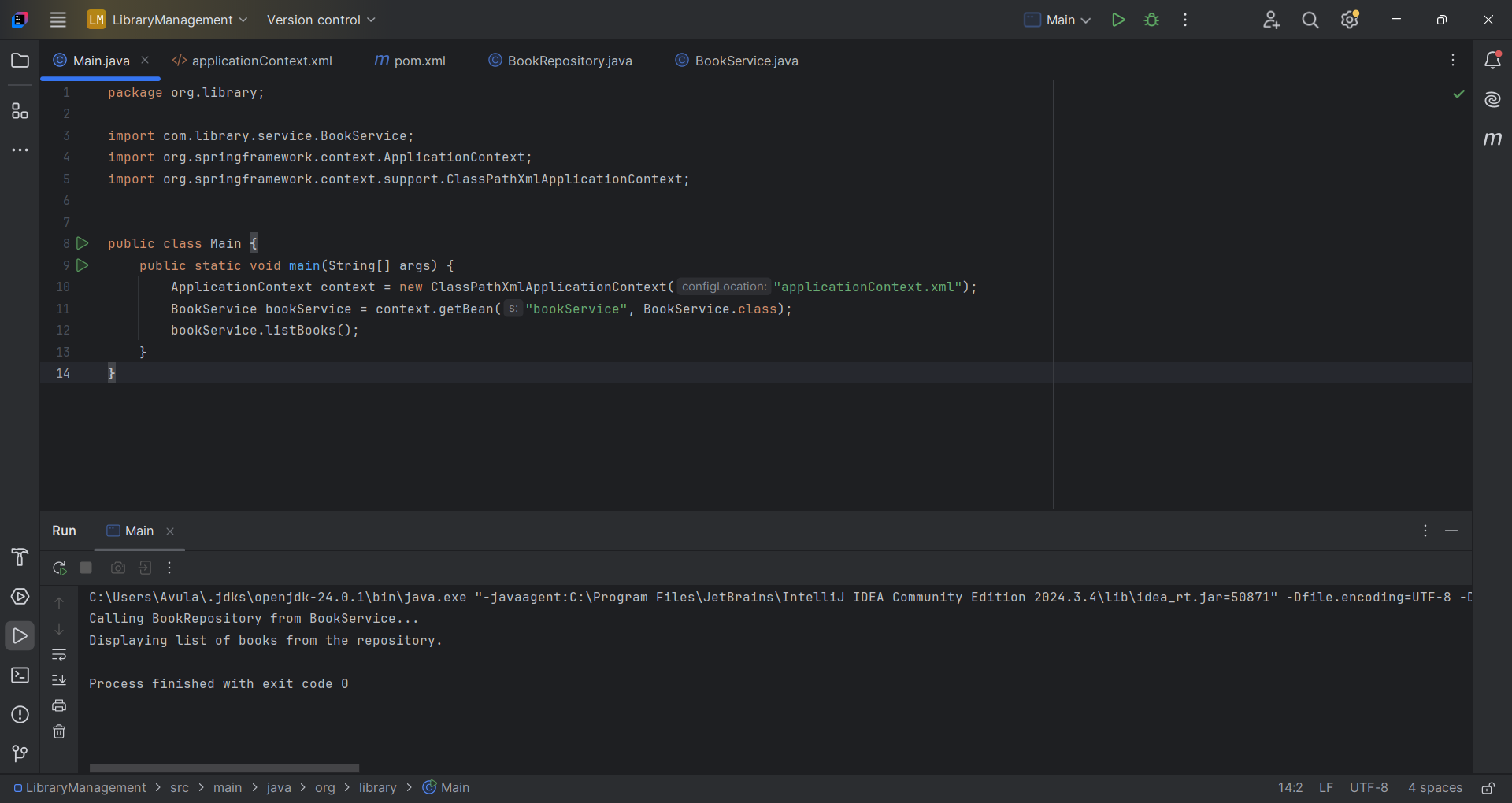
In the library management application, you need to manage the dependencies between the BookService and BookRepository classes using Spring's IoC and DI.

**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**.
3. **Test the Configuration:**
   * Run the **LibraryManagementApplication** main class to verify the dependency injection.

**Solution:**

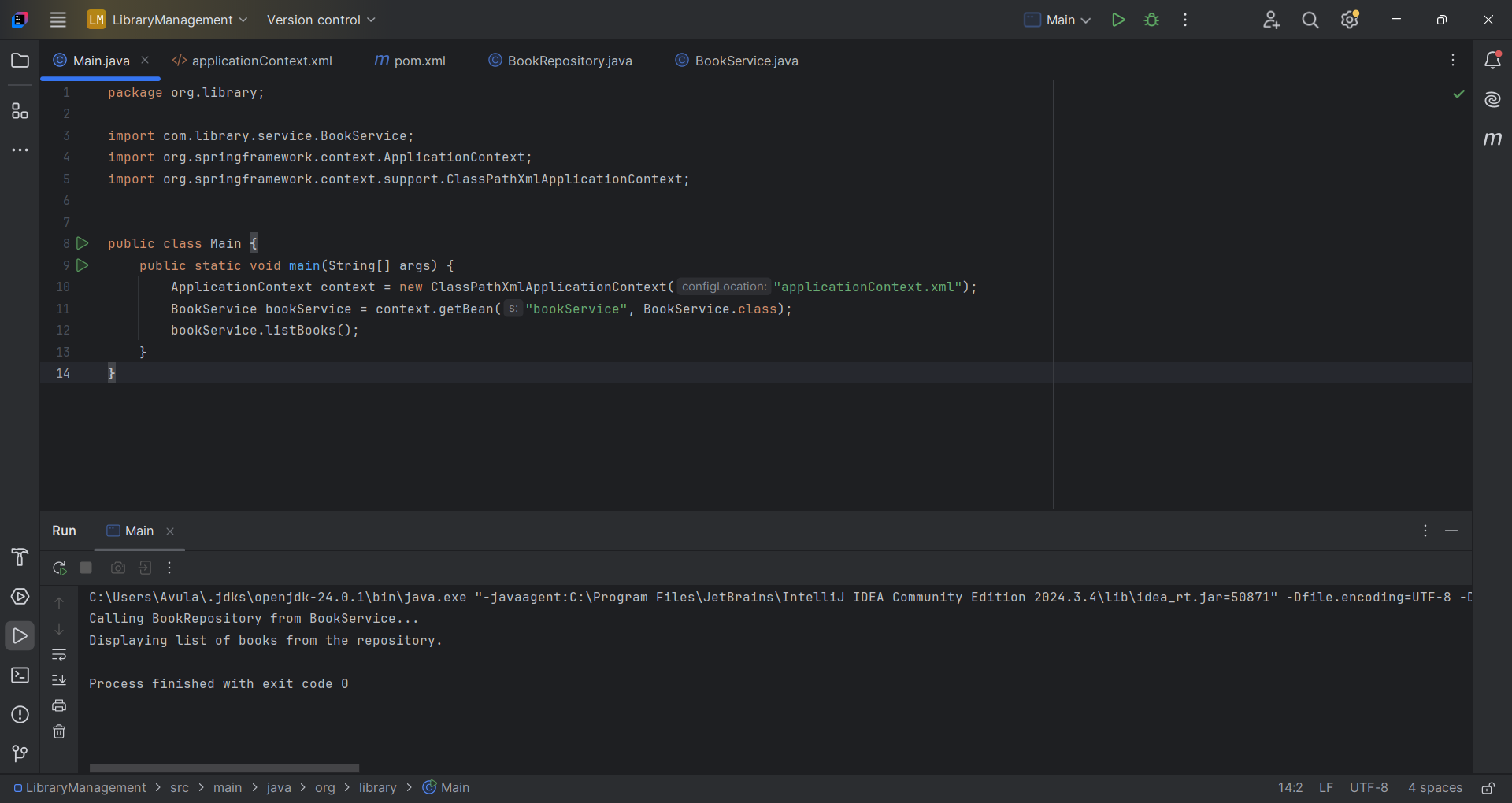
* Update **applicationContext.xml** to wire **BookRepository** into **BookService**.



* 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 listBooks() {  
 System.*out*.println("Calling BookRepository from BookService...");  
 bookRepository.displayBooks();  
 }  
}

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



**What is Dependency Injection (DI)?**

**Dependency Injection (DI)** is a design pattern used in Spring Framework (and other frameworks) to achieve **Inversion of Control (IoC)** — which means transferring the responsibility of managing dependencies from the object itself to the **Spring container**.

**Types of Dependency Injection in Spring:**

1. **Constructor Injection** – dependencies are provided through a class constructor.
2. **Setter Injection** – dependencies are set using JavaBean-style setters.
3. **Field Injection** (in annotation-based approach only).

We let **Spring inject BookRepository into BookService.** By using setter method (setBookRepository).