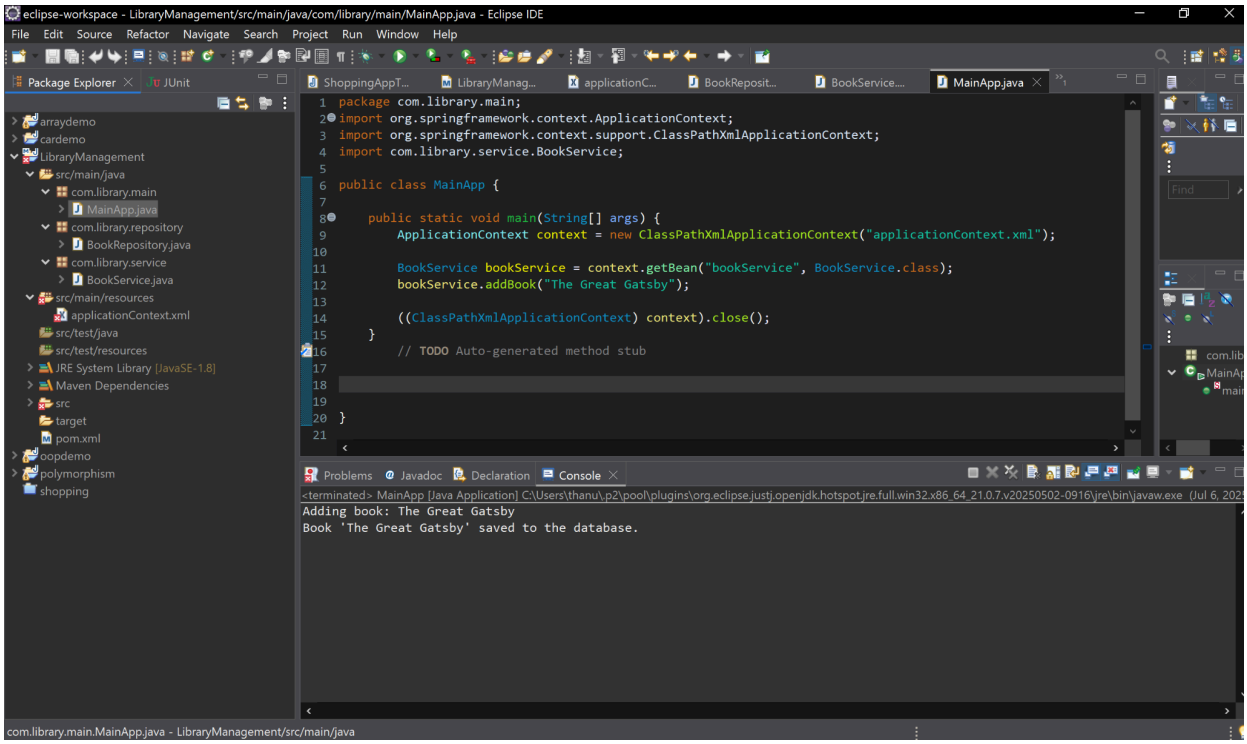
# Exercise 1: Configuring a Basic Spring Application

### Scenario:

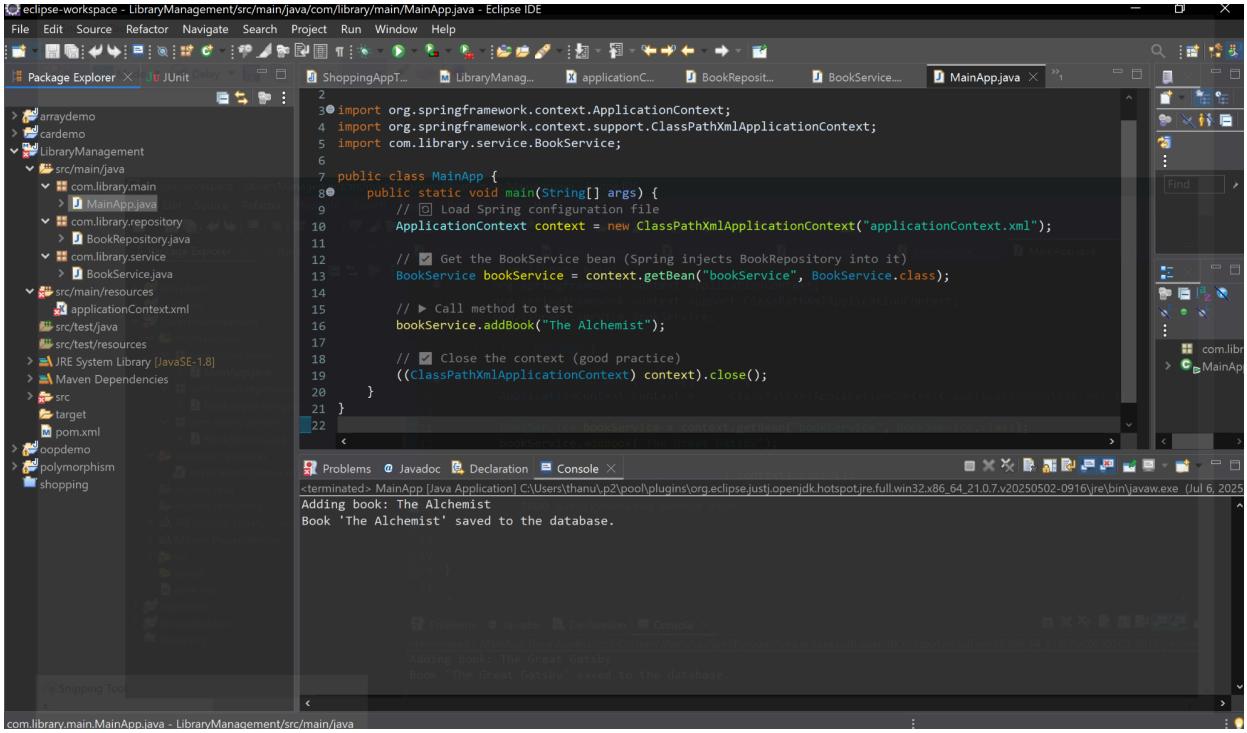
Your company is developing a web application for managing a library. You need to use the Spring Framework to handle the backend operations




# Exercise 2: Implementing Dependency Injection

### Scenario:

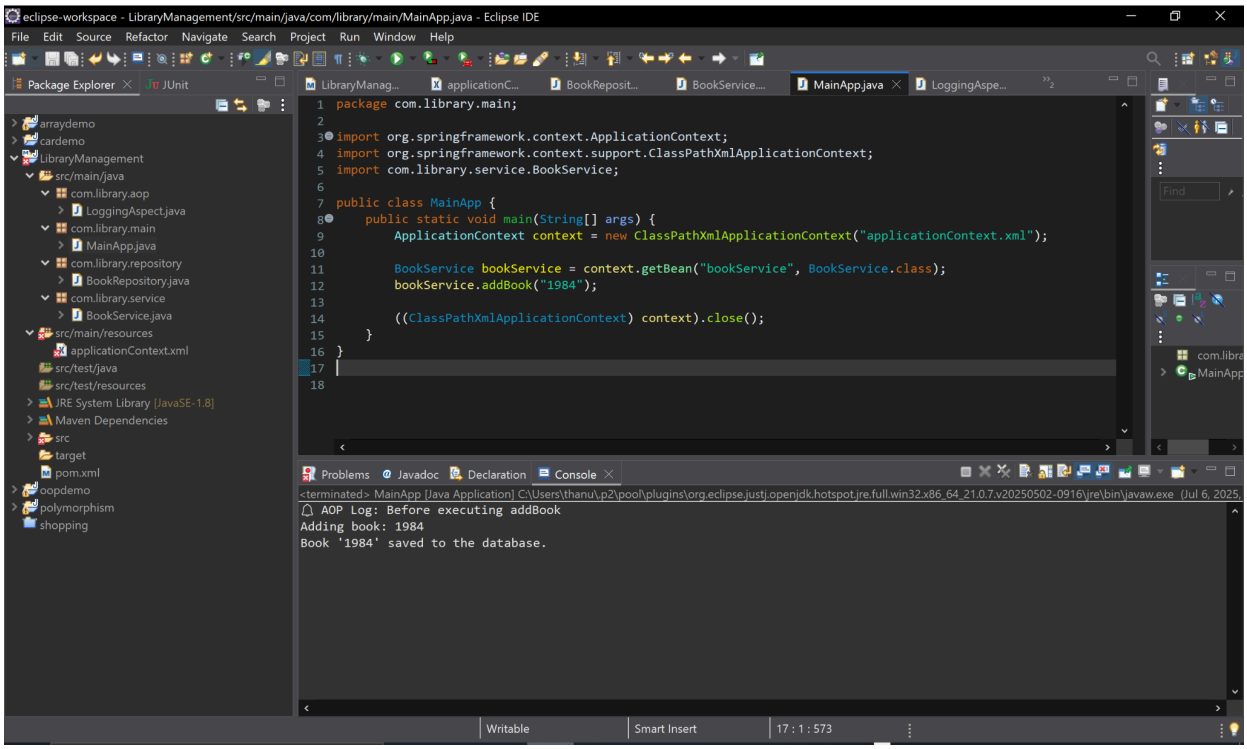
In the library management application, you need to manage the dependencies between the BookService and BookRepository classes using Spring's IoC and DI.




# 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.




## Explain the difference between Java Persistence API, Hibernate and Spring Data JPA

| Aspect | Java Persistence API (JPA) | Hibernate | Spring Data JPA |
|---|---|---|---|
| **1. Type** | It is a **specification** (interface) for ORM in Java. | It is a **framework** that implements JPA and adds extra features. | It is a **Spring-based abstraction** on top of JPA and Hibernate. |

| | | | |
|---|---|---|---|
| **2. Role** | Defines standard APIs for ORM (like `EntityManager`, `@Entity`). | Provides the actual implementation of JPA APIs (and more). | Simplifies data access using **repositories**, reducing boilerplate code. |
| **3. Boilerplate Code** | Requires manual creation of entity manager, queries, etc. | Reduces some boilerplate via APIs like `SessionFactory`. | Eliminates most boilerplate by using interfaces like `CrudRepository`. |
| **4. Vendor Neutrality** | Vendor-independent; can be used with any JPA provider (Hibernate, EclipseLink, etc.). | Specific to Hibernate; not vendor-neutral. | Depends on the underlying JPA provider (usually Hibernate). |
| **5. Use Case** | Used when you want **full control** and **vendor independence**. | Used when you need **advanced ORM features** beyond JPA spec. | Used when you want **rapid development** with minimal effort. |