# WEEK 3 – SPRING CORE MAVEN: EXERCISES

**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:**

1. **Set Up a Spring Project:**
   * Create a Maven project named **LibraryManagement**.
   * Add Spring Core dependencies in the **pom.xml** file.
2. **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.
3. **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**.
4. **Run the Application:**
   * Create a main class to load the Spring context and test the configuration.

BookRepository.java:

**package** com.library.repository;

**public** **class** BookRepository {

**public** **void** saveBook(String bookName) {

System.***out***.println("Book '" + bookName + "' saved to the repository.");

}

}

BookService.java:

**package** com.library.service;

**import** com.library.repository.BookRepository;

**public** **class** BookService {

**private** BookRepository bookRepository;

**public** **void** setBookRepository(BookRepository bookRepository) {

**this**.bookRepository = bookRepository;

}

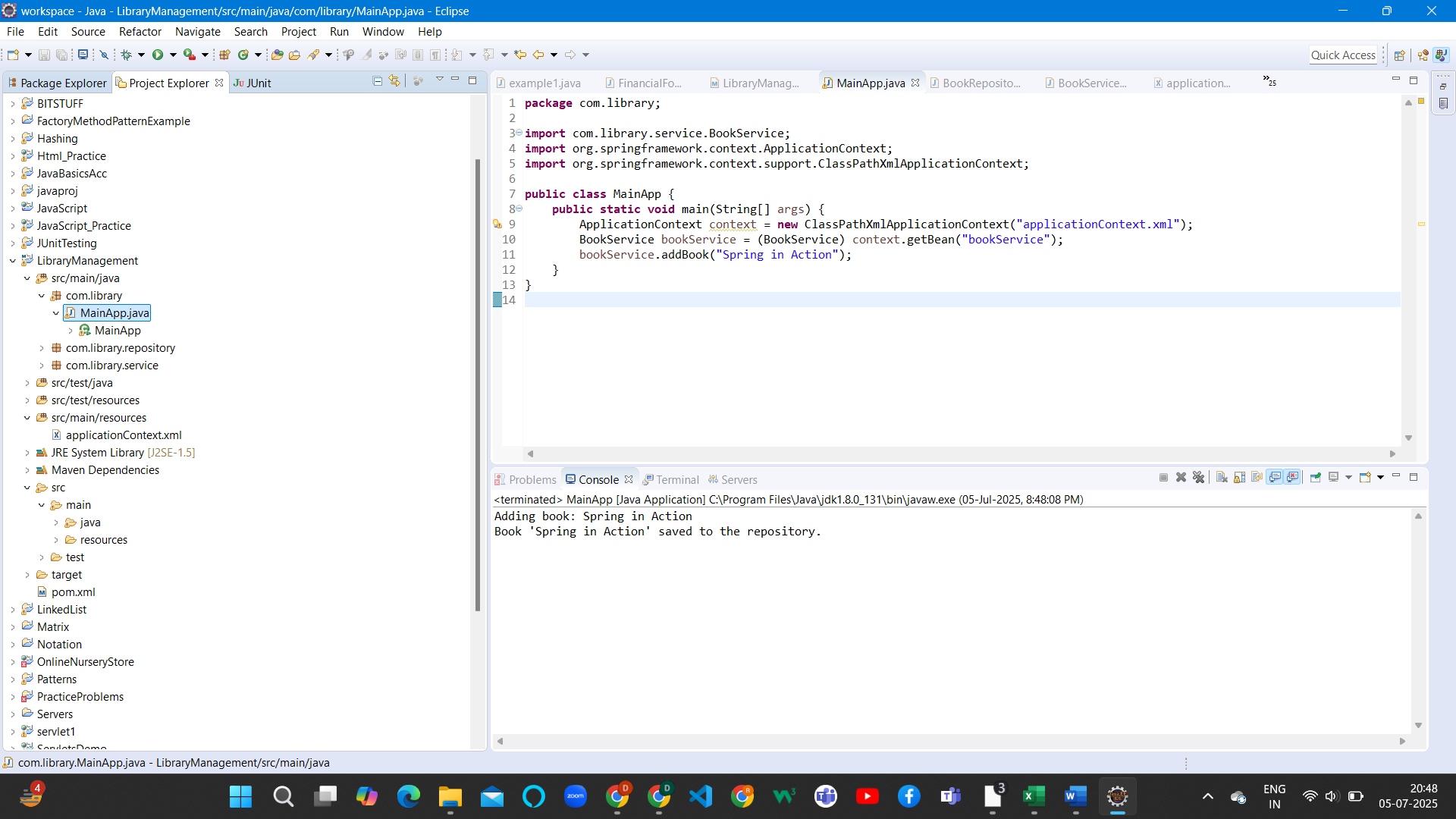
**public** **void** addBook(String bookName) {

System.***out***.println("Adding book: " + bookName);

bookRepository.saveBook(bookName);

}

}



**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.

applicationContext.xml:

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<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*

*https://www.springframework.org/schema/beans/spring-beans.xsd"*>

<!-- Define BookRepository bean -->

<bean id=*"bookRepository"* class=*"com.library.repository.BookRepository"* />

<!-- Define BookService bean and inject dependency -->

<bean id=*"bookService"* class=*"com.library.service.BookService"*>

<property name=*"bookRepository"* ref=*"bookRepository"* />

</bean>

</beans>

MainApp.java:

**package** com.library;

**import** com.library.service.BookService;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**public** **class** MainApp {

**public** **static** **void** main(String[] args) {

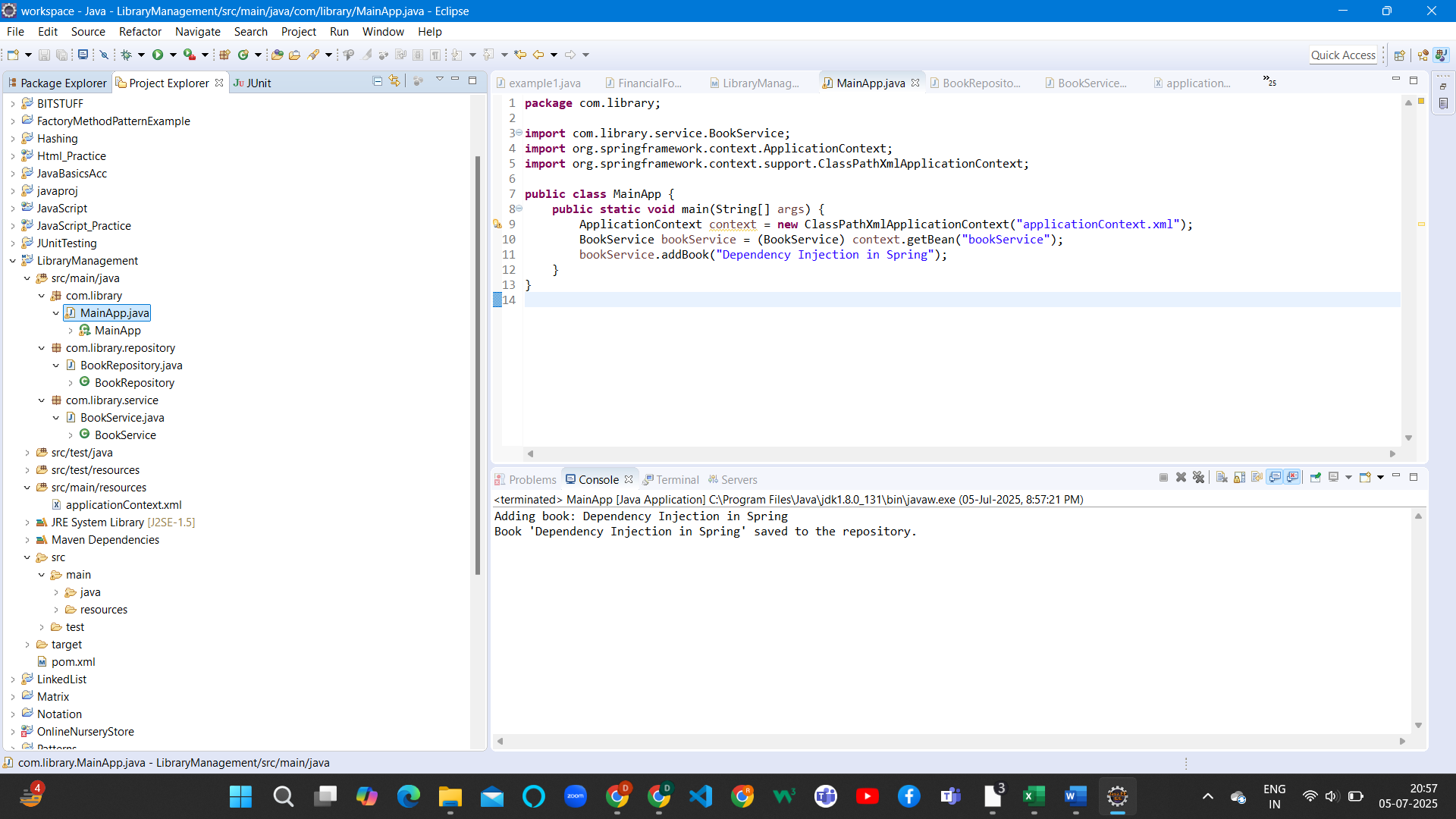
ApplicationContext context = **new** ClassPathXmlApplicationContext("applicationContext.xml");

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

bookService.addBook("Dependency Injection in Spring");

}

}



**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.

pom.xml:

<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>

<properties>

<maven.compiler.source>1.8</maven.compiler.source>

<maven.compiler.target>1.8</maven.compiler.target>

</properties>

<dependencies>

<!-- Spring Context (Core IoC container) -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.29</version>

</dependency>

<!-- Spring AOP (for Aspect-Oriented Programming) -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.29</version>

</dependency>

<!-- Spring Web MVC (for controllers and web applications) -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.29</version>

</dependency>

<!-- Commons Logging (used by Spring internally) -->

<dependency>

<groupId>commons-logging</groupId>

<artifactId>commons-logging</artifactId>

<version>1.2</version>

</dependency>

</dependencies>

<build>

<plugins>

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

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.10.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

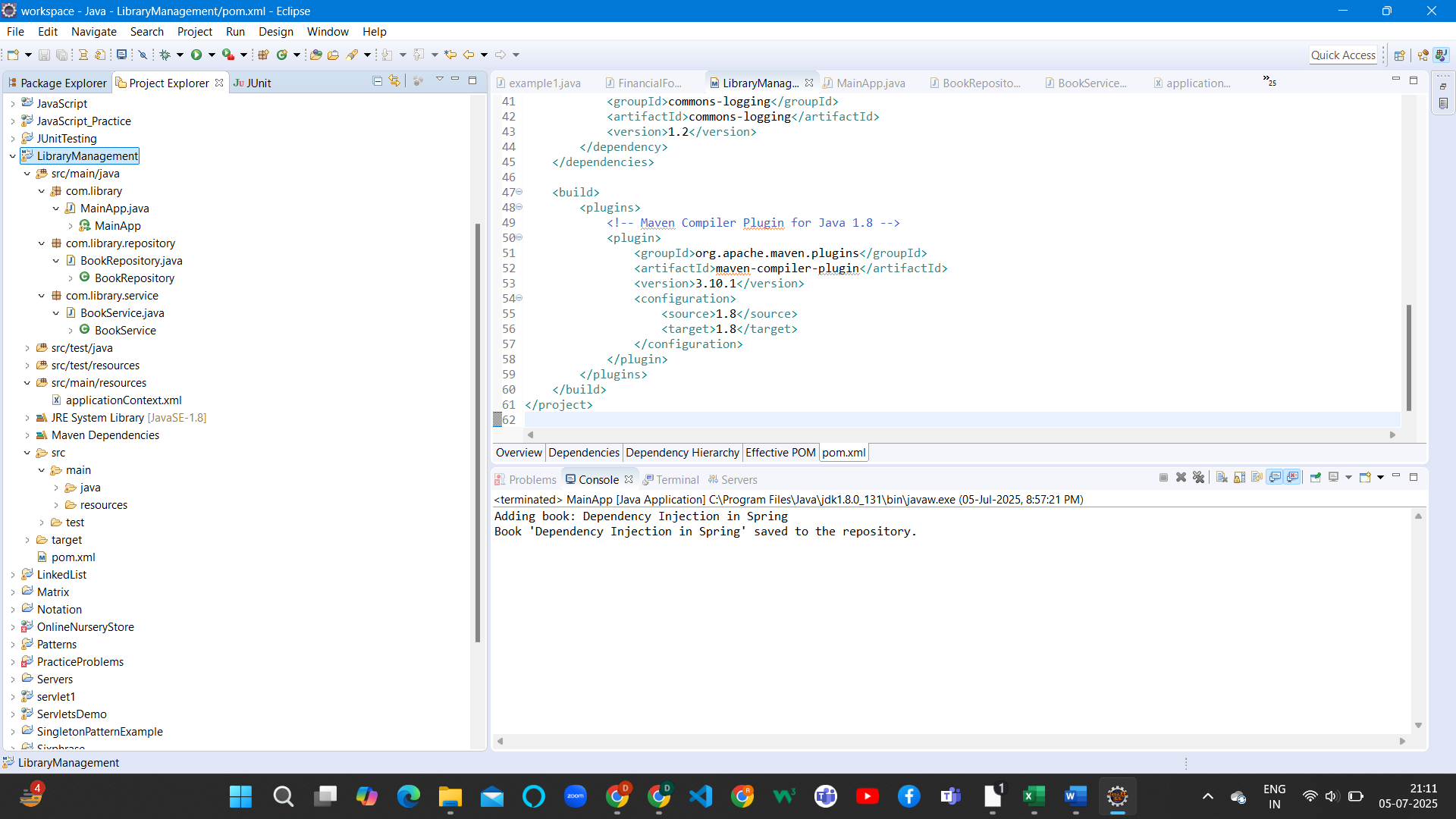
</configuration>

</plugin>

</plugins>

</build>

</project>



**4.Spring Data JPA - Quick Example**

**Spring Data JPA** is a part of the Spring ecosystem that simplifies the implementation of JPA (Java Persistence API) based data access layers. In this exercise, I implemented a quick example to demonstrate how to use Spring Data JPA for performing CRUD operations on a simple Book entity.

The main components I used in this example are:

1. **Entity Class**:  
   A Book class was created and annotated with @Entity, representing a table in the database. Fields like id, title, and author were used to define columns.
2. **Repository Interface**:  
   An interface BookRepository was created which extends JpaRepository<Book, Integer>. This provides built-in methods like save(), findAll(), findById(), and deleteById() without writing any SQL.
3. **Main Application Class**:  
   A Spring Boot application class annotated with @SpringBootApplication was used to bootstrap the app using SpringApplication.run() method.
4. **Database Configuration**:  
   Database properties such as driver class, URL, username, and password were set using application.properties.

**5. Difference between JPA, Hibernate and Spring Data JPA**

➤ JPA (Java Persistence API):

* JPA is a specification provided by Java to define a standard way to manage relational data using Java objects.
* It only defines interfaces and annotations like @Entity, @Id, and EntityManager.
* JPA does not provide actual implementation.

➤ Hibernate:

* Hibernate is the most widely used implementation of JPA.
* It provides advanced ORM (Object Relational Mapping) features and allows writing queries using HQL (Hibernate Query Language).
* Hibernate implements all the interfaces defined by JPA and adds its own features.

➤ Spring Data JPA:

* Spring Data JPA is built on top of JPA and Hibernate.
* It simplifies the development by removing the need to write DAO (Data Access Object) implementations manually.
* Just by extending JpaRepository, we can perform most database operations through method names like findByTitle(), save(), etc.

| **Feature** | **JPA** | **Hibernate** | **Spring Data JPA** |
| --- | --- | --- | --- |
| Type | Specification (Interface) | Implementation of JPA | Abstraction built on JPA/Hibernate |
| Provider? | No | Yes | No (uses JPA provider internally) |
| Ease of Use | Moderate | Easier | Easiest |
| Boilerplate Code | More | Less than JPA | Very minimal |
| Used In | Java EE | Java/Spring apps | Spring Boot |