Exercise1: **Configuring a Basic Spring Application**

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>  
  
 <dependencies>  
 <!-- Spring Context (Core) -->  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>5.3.30</version>  
 </dependency>  
 </dependencies>  
  
 <build>  
 <plugins>  
 <!-- Optional: Used for running via mvn exec:java -->  
 <plugin>  
 <groupId>org.codehaus.mojo</groupId>  
 <artifactId>exec-maven-plugin</artifactId>  
 <version>3.1.0</version>  
 <configuration>  
 <mainClass>com.library.Main</mainClass>  
 </configuration>  
 </plugin>  
 </plugins>  
 </build>  
  
</project>

Main.java

package com.library;  
  
import com.library.service.BookService;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
  
import java.util.Scanner;  
  
public class Main {  
 public static void main(String[] args) {  
 ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");  
 BookService bookService = (BookService) context.getBean("bookService");  
  
 Scanner scanner = new Scanner(System.*in*);  
 int choice;  
  
 do {  
 System.*out*.println("\nLibrary Menu");  
 System.*out*.println("1. Add Book");  
 System.*out*.println("2. Show All Books");  
 System.*out*.println("3. Search Book");  
 System.*out*.println("4. Exit");  
 System.*out*.print("Enter choice: ");  
  
 choice = scanner.nextInt();  
 scanner.nextLine(); // consume newline  
  
 switch (choice) {  
 case 1:  
 System.*out*.print("Enter book title: ");  
 String title = scanner.nextLine();  
 bookService.addBook(title);  
 break;  
 case 2:  
 bookService.showAllBooks();  
 break;  
 case 3:  
 System.*out*.print("Enter book title to search: ");  
 String searchTitle = scanner.nextLine();  
 bookService.searchBook(searchTitle);  
 break;  
 case 4:  
 System.*out*.println("Exiting... ");  
 break;  
 default:  
 System.*out*.println("❗ Invalid choice. Try again.");  
 }  
 } while (choice != 4);  
  
 scanner.close();  
 }  
}

BookRepository.java

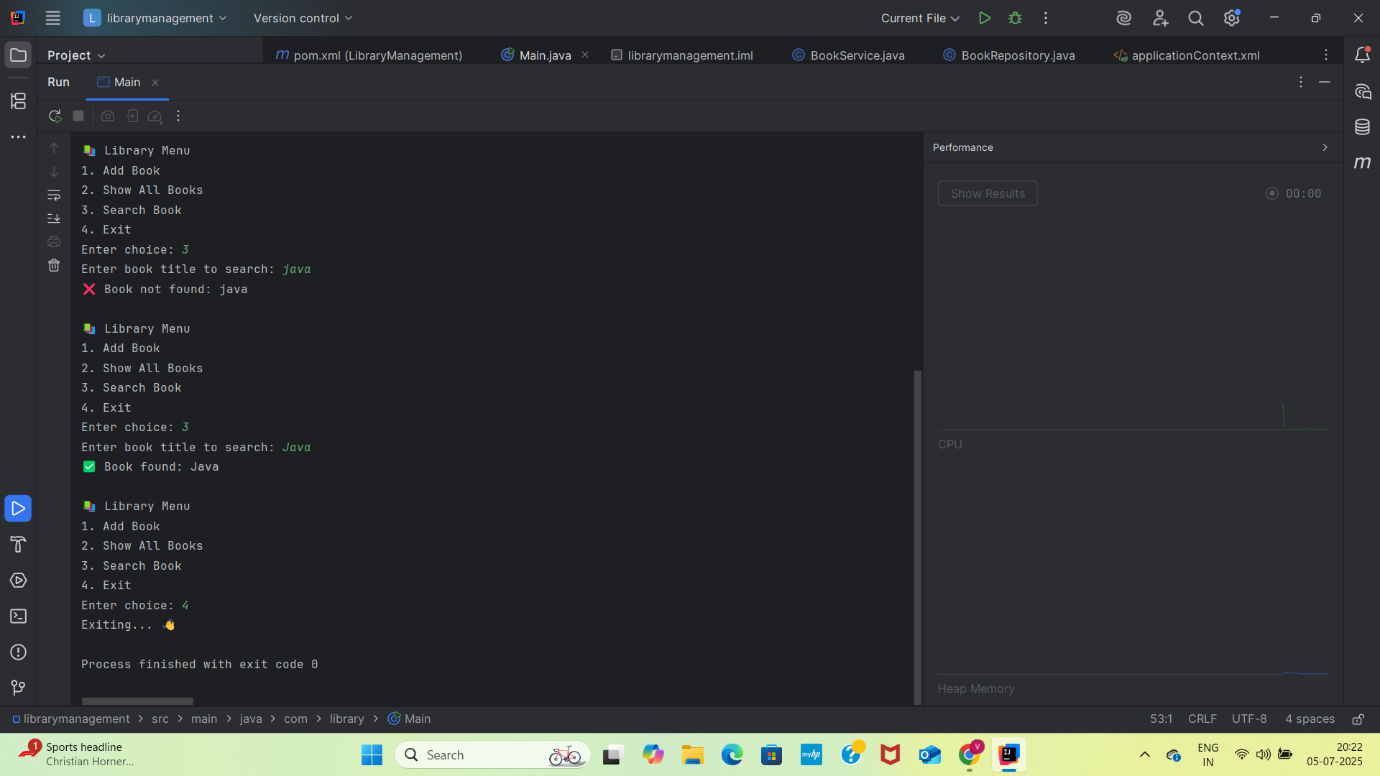
package com.library.repository;  
  
import java.util.ArrayList;  
import java.util.List;  
  
public class BookRepository {  
 private List<String> books = new ArrayList<>();  
  
 public void saveBook(String title) {  
 books.add(title);  
 System.*out*.println("Book saved: " + title);  
 }  
  
 public List<String> getAllBooks() {  
 return books;  
 }  
  
 public boolean searchBook(String title) {  
 return books.contains(title);  
 }  
}

BookService.java

package com.library.service;  
  
import com.library.repository.BookRepository;  
import java.util.List;  
  
public class BookService {  
 private BookRepository bookRepository;  
  
 public void setBookRepository(BookRepository bookRepository) {  
 this.bookRepository = bookRepository;  
 }  
  
 public void addBook(String title) {  
 System.*out*.println("Adding book: " + title);  
 bookRepository.saveBook(title);  
 }  
  
 public void showAllBooks() {  
 List<String> books = bookRepository.getAllBooks();  
 System.*out*.println("All books:");  
 for (String book : books) {  
 System.*out*.println("- " + book);  
 }  
 }  
  
 public void searchBook(String title) {  
 boolean found = bookRepository.searchBook(title);  
 if (found) {  
 System.*out*.println(" Book found: " + title);  
 } else {  
 System.*out*.println(" Book not found: " + title);  
 }  
 }  
}

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  
 http://www.springframework.org/schema/beans/spring-beans.xsd">  
  
 <bean id="bookRepository" class="com.library.repository.BookRepository"/>  
  
 <bean id="bookService" class="com.library.service.BookService">  
 <property name="bookRepository" ref="bookRepository"/>  
 </bean>  
</beans>



**Exercise2:**

**Configuring a Basic Spring Application**

**BookRepository.java**

package com.library.repository;  
  
import java.util.ArrayList;  
import java.util.List;  
  
public class BookRepository {  
 private List<String> books = new ArrayList<>();  
  
 public void saveBook(String title) {  
 books.add(title);  
 System.*out*.println("Book saved: " + title);  
 }  
  
 public List<String> getAllBooks() {  
 return books;  
 }  
  
 public boolean searchBook(String title) {  
 return books.contains(title);  
 }  
}

**BookService.java**

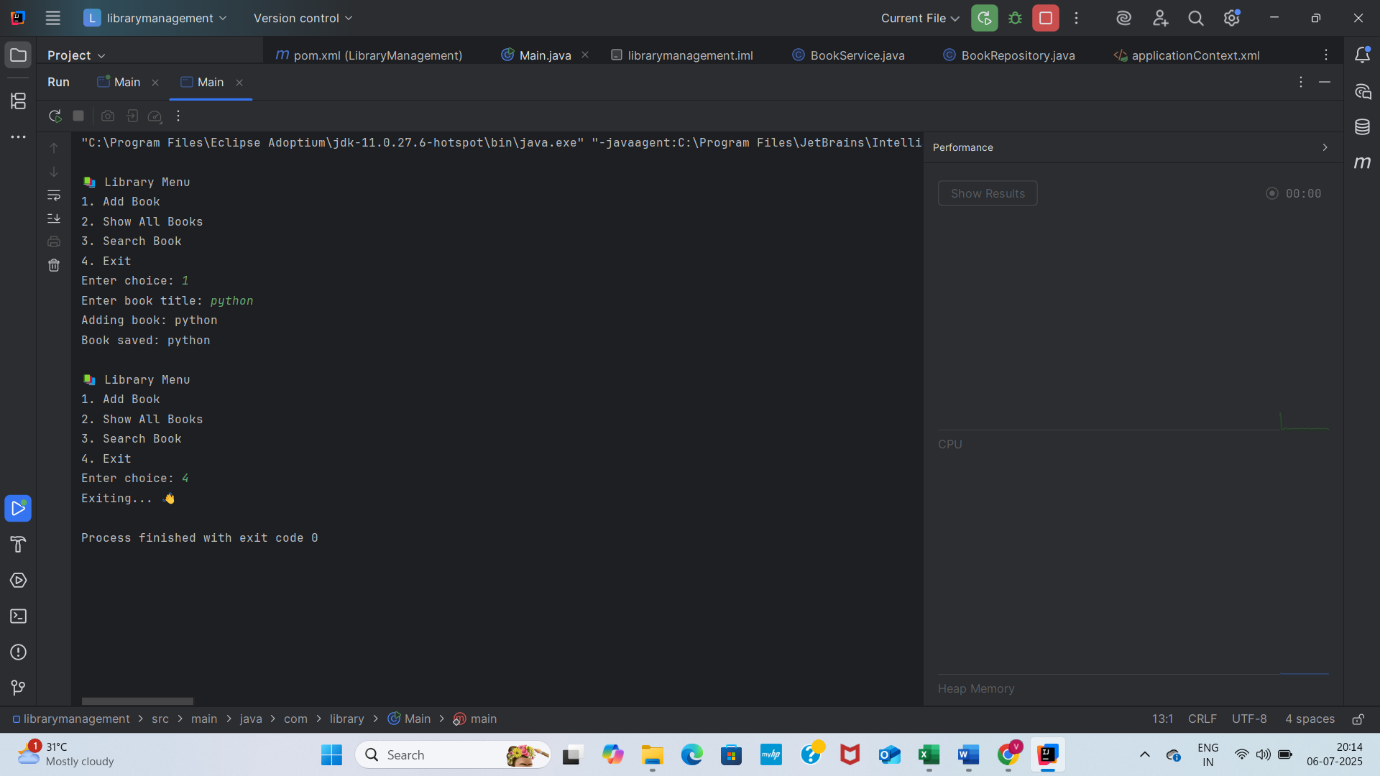
package com.library.service;  
  
import com.library.repository.BookRepository;  
import java.util.List;  
  
public class BookService {  
 private BookRepository bookRepository;  
  
 public void setBookRepository(BookRepository bookRepository) {  
 this.bookRepository = bookRepository;  
 }  
  
 public void addBook(String title) {  
 System.*out*.println("Adding book: " + title);  
 bookRepository.saveBook(title);  
 }  
  
 public void showAllBooks() {  
 List<String> books = bookRepository.getAllBooks();  
 System.*out*.println("All books:");  
 for (String book : books) {  
 System.*out*.println("- " + book);  
 }  
 }  
  
 public void searchBook(String title) {  
 boolean found = bookRepository.searchBook(title);  
 if (found) {  
 System.*out*.println(" Book found: " + title);  
 } else {  
 System.*out*.println(" Book not found: " + title);  
 }  
 }  
}

**Main.java**

package com.library;  
  
import com.library.service.BookService;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
  
import java.util.Scanner;  
  
public class Main {  
 public static void main(String[] args) {  
 ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");  
 BookService bookService = (BookService) context.getBean("bookService");  
  
 Scanner scanner = new Scanner(System.*in*);  
 int choice;  
  
 do {  
 System.*out*.println("\nLibrary Menu");  
 System.*out*.println("1. Add Book");  
 System.*out*.println("2. Show All Books");  
 System.*out*.println("3. Search Book");  
 System.*out*.println("4. Exit");  
 System.*out*.print("Enter choice: ");  
  
 choice = scanner.nextInt();  
 scanner.nextLine(); // consume newline  
  
 switch (choice) {  
 case 1:  
 System.*out*.print("Enter book title: ");  
 String title = scanner.nextLine();  
 bookService.addBook(title);  
 break;  
 case 2:  
 bookService.showAllBooks();  
 break;  
 case 3:  
 System.*out*.print("Enter book title to search: ");  
 String searchTitle = scanner.nextLine();  
 bookService.searchBook(searchTitle);  
 break;  
 case 4:  
 System.*out*.println("Exiting... ");  
 break;  
 default:  
 System.*out*.println("❗ Invalid choice. Try again.");  
 }  
 } while (choice != 4);  
  
 scanner.close();  
 }  
}

**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>  
  
 <dependencies>  
 <!-- ✅ Spring Context (Core) -->  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>5.3.30</version>  
 </dependency>  
 </dependencies>  
  
 <build>  
 <plugins>  
 <!-- Optional: Used for running via mvn exec:java -->  
 <plugin>  
 <groupId>org.codehaus.mojo</groupId>  
 <artifactId>exec-maven-plugin</artifactId>  
 <version>3.1.0</version>  
 <configuration>  
 <mainClass>com.library.Main</mainClass>  
 </configuration>  
 </plugin>  
 </plugins>  
 </build>  
  
</project>



**Difference between JPA, Hibernate and Spring Data JPA** **Java Persistence API (JPA)**

1. JPA is a **specification** (JSR 338) for object-relational mapping (ORM).
2. It defines **interfaces and annotations**, but has **no implementation**.
3. Requires a provider like **Hibernate** or **EclipseLink** to work.
4. Provides **standardized APIs** like EntityManager, @Entity, @Table, etc.
5. Supports **JPQL (Java Persistence Query Language)**.
6. Can be used in both **Java SE and EE** environments.
7. Offers **database independence** and a consistent ORM layer.

**Hibernate**

1. Hibernate is an **ORM tool** and also the **most common JPA implementation**.
2. Can work with or without JPA (has its own API like Session, Transaction, HQL).
3. Offers **rich features** like caching, lazy/eager loading, and custom queries.
4. Requires **more boilerplate code** for session and transaction management.
5. Uses **HQL (Hibernate Query Language)** as its native query language.
6. Provides **native support** for features like **second-level caching**, **interceptors**, and more.
7. Is the **default provider** used in Spring Data JPA if not overridden.

**Spring Data JPA**

1. A **Spring module** that sits on top of JPA and uses an implementation like Hibernate.
2. Provides **ready-made repository interfaces** like JpaRepository, CrudRepository.
3. Removes **boilerplate code** (no need to write queries or DAO implementations).
4. Uses **Spring’s dependency injection** and **transaction management**.
5. Supports **derived query methods**, custom @Query, pagination, and sorting.
6. Perfect for **rapid development** with **Spring Boot**.
7. Cannot function alone—needs **JPA + Hibernate** or another provider.

**Summary (1-line each)**

| **Feature** | **JPA** | **Hibernate** | **Spring Data JPA** |
| --- | --- | --- | --- |
| Type | Specification | ORM tool | Spring abstraction over JPA |
| Implementation Provided | No | Yes | Uses JPA provider |
| Boilerplate Code | Medium | High | Low |
| Query Support | JPQL | HQL | JPQL, Native SQL, Derived Methods |
| API Example | EntityManager | Session, Transaction | JpaRepository, @Query |
| Use with Spring | Possible but manual setup | Possible but manual setup | Fully integrated |
| Goal | Define standard persistence API | Provide full ORM functionality | Simplify DB access in Spring apps |