**WEEK 3\_Spring Core and Maven Hands-on**

**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.

**com.library.repository/BookService.java:**

**package** com.library.service;

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

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

**private** BookRepository bookRepository;

// Setter for Spring DI

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

**this**.bookRepository = bookRepository;

}

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

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

bookRepository.save(bookName);

System.***out***.println("BookService: Book added successfully.");

}

}

**Com.library.repository/BookRepository.java:**

**package** com.library.repository;

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

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

System.***out***.println("BookRepository: Book '" + bookName + "' saved.");

}}

**com.library/MainApp.java:**

package com.library;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

public class MainApp {

public static void main(String[] args) {

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

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

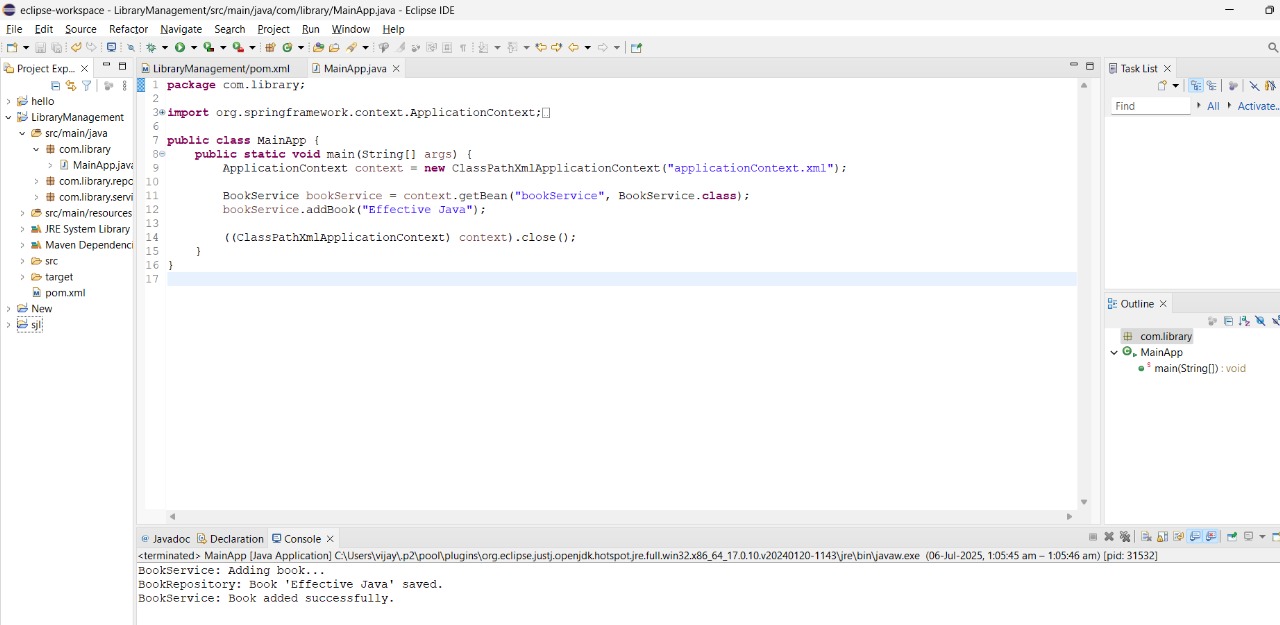
bookService.addBook("Effective Java");

((ClassPathXmlApplicationContext) context).close();

}

}

**OUTPUT:**



**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.

**CODE:**

**Book.java:**

package com.example.library;

public class Book {

private String title;

public Book(String title) {

this.title = title;

}

public String getTitle() {

return title;

}

}

**BookRepository.java:**

package com.example.library;

import java.util.Arrays;

import java.util.List;

public class BookRepository {

public List<Book> getAllBooks() {

return Arrays.asList(

new Book("The Hobbit"),

new Book("1984"),

new Book("Pride and Prejudice"));}}

**BookService.java:**

**package** com.example.library;

**import** java.util.List;

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

**private** BookRepository bookRepository;

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

**this**.bookRepository = bookRepository;

}

**public** **void** displayBooks() {

List<Book> books = bookRepository.getAllBooks();

System.***out***.println("Books in Library:");

**for** (Book book : books) {

System.***out***.println("- " + book.getTitle());

}

}

}

**LibraryManagementApplication.java:**

package com.example.library;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryManagementApplication {

public static void main(String[] args) {

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

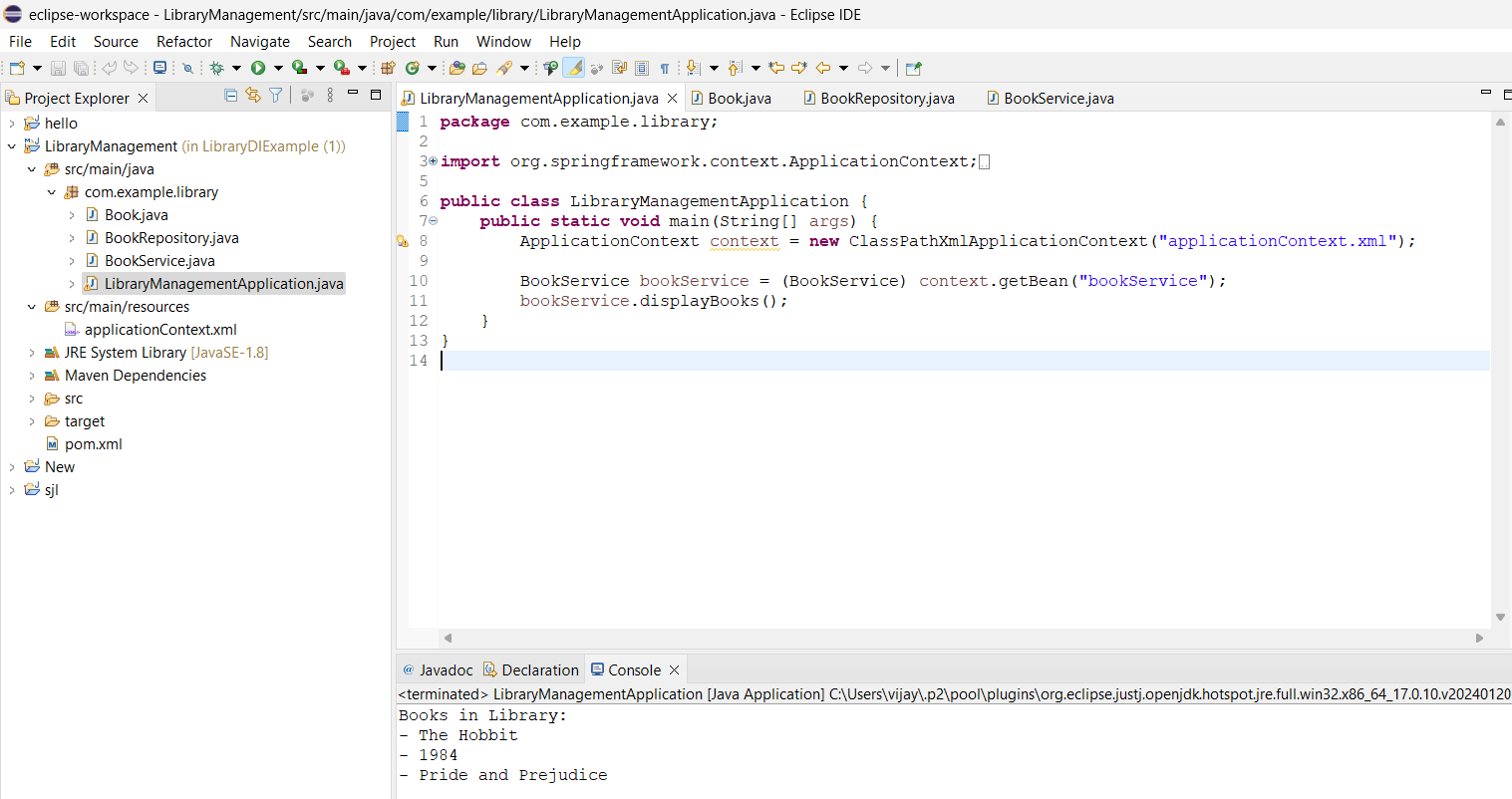
BookService bookService = (BookService) context.getBean("bookService");

bookService.displayBooks();

}

}

**OUTPUT:**

****

**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.

**CODE:**

**App.java:**

package com.library;

public class App {

public static void main(String[] args) {

System.*out*.println("Library Management Application Initialized.");

}

}

**Apptest.java:**

package com.library;

public class AppTest {

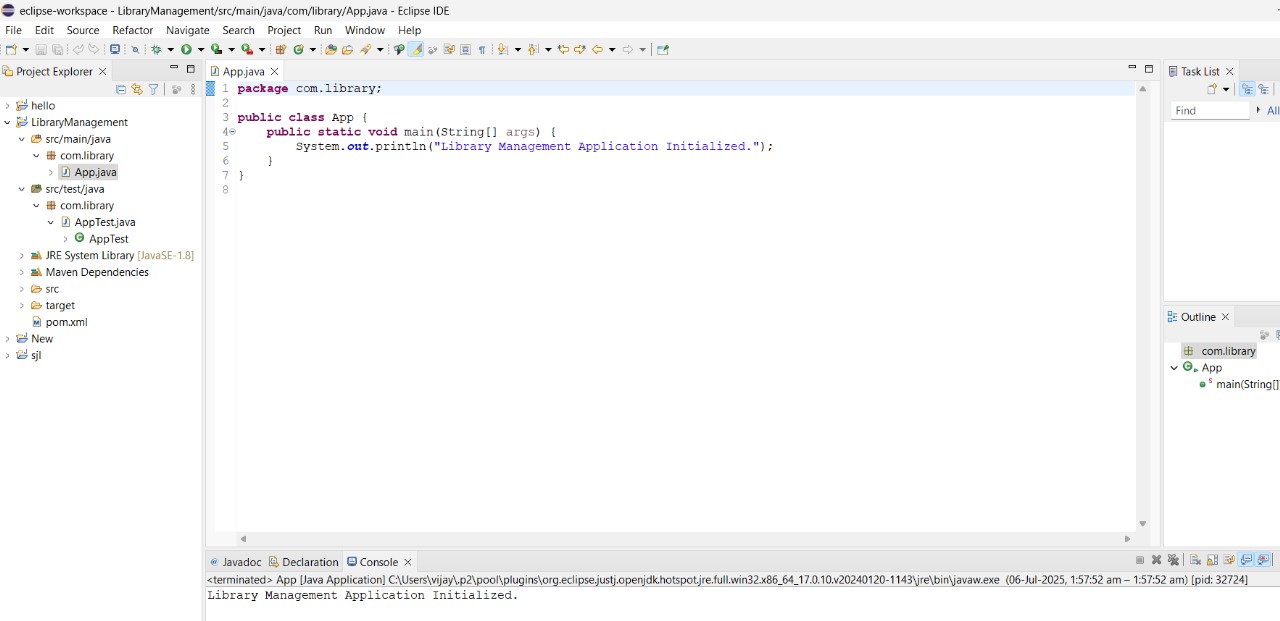
public void testApp() {

System.*out*.println("Running App Test.");

}

}

**OUTPUT:**



**Exercise 5: Spring Data JPA - Quick Example**

**Scenario:**

You are developing a simple library management system using Spring Boot and Spring Data JPA. The application must Add a few books to an in-memory H2 database.

**CODE:**

**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.example</groupId>  
 <artifactId>demo</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <packaging>jar</packaging>  
 <name>demo</name>  
 <description>Spring Boot Demo</description>  
 <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>3.2.0</version>  
 <relativePath/>  
 </parent>  
 <properties>  
 <java.version>21</java.version>  
 <maven.compiler.source>21</maven.compiler.source>  
 <maven.compiler.target>21</maven.compiler.target>  
 </properties>  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-jpa</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>com.h2database</groupId>  
 <artifactId>h2</artifactId>  
 <scope>runtime</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
 </dependencies>  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 </plugins>  
 </build>  
</project>

**OUTPUT:**

****

**Exercise 6: Difference between JPA, Hibernate and Spring Data JPA**

**JPA (Java Persistence API)**

* **Definition:** JPA is a Java specification that defines a standard for ORM (Object-Relational Mapping) between Java objects and relational databases.
* **Role:** Acts as an API layer or contract that provides guidelines for managing relational data in Java applications.
* **Key Features:**
  + Annotations like @Entity, @Table, @Id, @OneToMany
  + Entity lifecycle management (persist, merge, remove, etc.)
  + JPQL (Java Persistence Query Language)
  + EntityManager to interact with the database
* Note: JPA does not provide implementation — it must be used with an implementation like Hibernate, EclipseLink, etc.

Example: When you write @Entity, you’re using JPA — but it’s Hibernate (or another provider) that actually processes that annotation.

**Hibernate**

* **Definition:** Hibernate is the most popular implementation of the JPA specification.
* **Role:** Acts as the JPA provider, executing all operations defined by JPA and offering additional advanced features.
* **Key Features:**
  + Implements everything from JPA
  + Hibernate-specific configurations (e.g., hibernate.cfg.xml)
  + Advanced features like:
    - Caching (L1, L2)
    - Lazy and Eager loading
    - Hibernate Query Language (HQL)
    - Support for native SQL queries
    - Custom ID generation strategies
* **Beyond JPA:** You can use Hibernate APIs directly (e.g., SessionFactory, Session) for more control, but that increases complexity.

Hibernate is like a powerful engine running behind the scenes of JPA.

**Spring Data JPA**

* **Definition:** A part of the Spring Data family, Spring Data JPA is a Spring module that makes it easy to implement JPA-based data access layers using auto-generated code and repository interfaces.
* **Role:** Abstracts the JPA/Hibernate complexities and reduces boilerplate code.
* **Key Features:**
  + JpaRepository<T, ID> and CrudRepository<T, ID> interfaces
  + Auto-implementation of methods like findById, findAll, save, deleteById
  + Method query derivation: findByTitleContainingIgnoreCase(String title)
  + Custom queries with @Query
  + Paging and sorting support
* **Spring Boot Integration:**
  + Auto configuration of EntityManager
  + No XML required (uses application.properties)
  + Can integrate with Spring Web, Security, etc.

Spring Data JPA lets you focus on business logic instead of writing repetitive DAO code.

**Comparision Table:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature / Aspect** | **JPA** | **Hibernate** | **Spring Data JPA** |
| **Type** | Specification / API | ORM Framework / JPA Implementation | Spring Module (abstraction over JPA) |
| **Provider** | Oracle (Java EE) | Red Hat | Spring (VMware) |
| **Purpose** | Standard ORM mapping | ORM engine + JPA implementation | Simplify JPA + DB access in Spring apps |
| **Needs Implementation?** | Yes | No (It is an implementation) | Yes (uses JPA & Hibernate internally) |
| **Boilerplate Code** | Medium (EntityManager, queries) | High (SessionFactory, config needed) | Very Low (Repository auto-wiring, auto-queries) |
| **Query Language** | JPQL | JPQL, HQL, Native SQL | JPQL, Method names, @Query |
| **Transaction Management** | Manual or integrated | Manual or integrated | Auto-handled by Spring |
| **Advanced ORM Features** | No | Yes (caching, batch fetch, interceptors) | No (delegates to Hibernate/JPA) |
| **Ease of Use** | Moderate | Complex | Very Easy |
| **Code to Write** | More | More | Less |
| **Typical Use Case** | Portable, vendor-neutral ORM | Full control, advanced ORM | Rapid development with Spring Boot |
| **Spring Boot Support** | Manual integration | Integrated via JPA | Fully integrated with Spring Boot |