**Spring Core And Maven**

**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.

**BookRepository.java**

package com.library.repository;

public class BookRepository {

public void saveBook(String title) {

System.***out***.println("Saving book: " + title);

}

}

**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 title) {

System.***out***.println("Adding book in service: " + title);

bookRepository.saveBook(title);

}

}

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

service.addBook("The Spring Journey");

}

}

**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">

<!-- Define BookRepository Bean -->

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

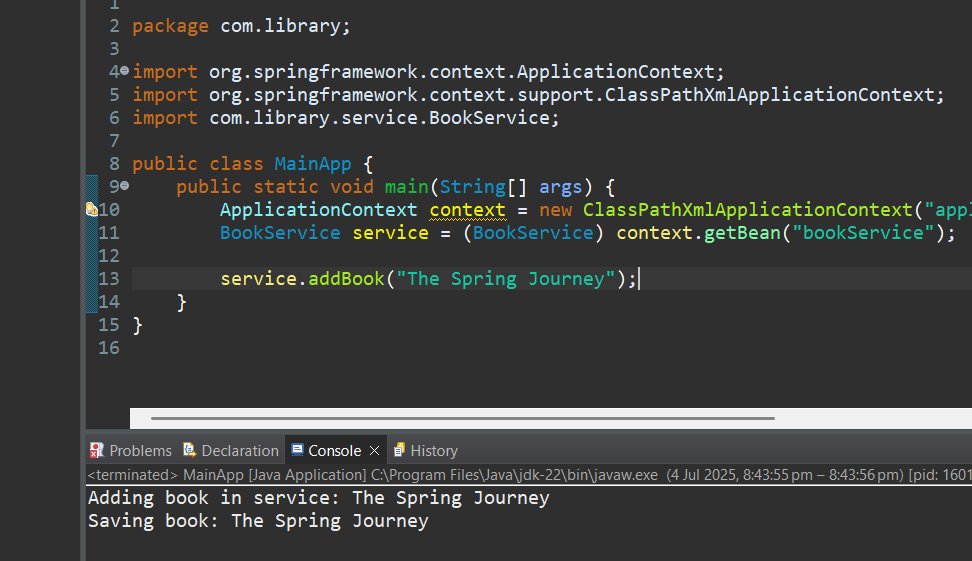
<!-- Define BookService Bean and Inject BookRepository -->

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

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

</bean>

</beans>



**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.

**BookService.java**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

// Required setter for Spring Dependency Injection

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void addBook(String title) {

System.***out***.println("Adding book in service: " + title);

bookRepository.saveBook(title);

}

}

**BookRepository.java**

package com.library.repository;

public class BookRepository {

public void saveBook(String title) {

System.out.println("Saving book: " + title);

}

}

**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) {

// Load Spring Application Context

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

// Retrieve Bean from Context

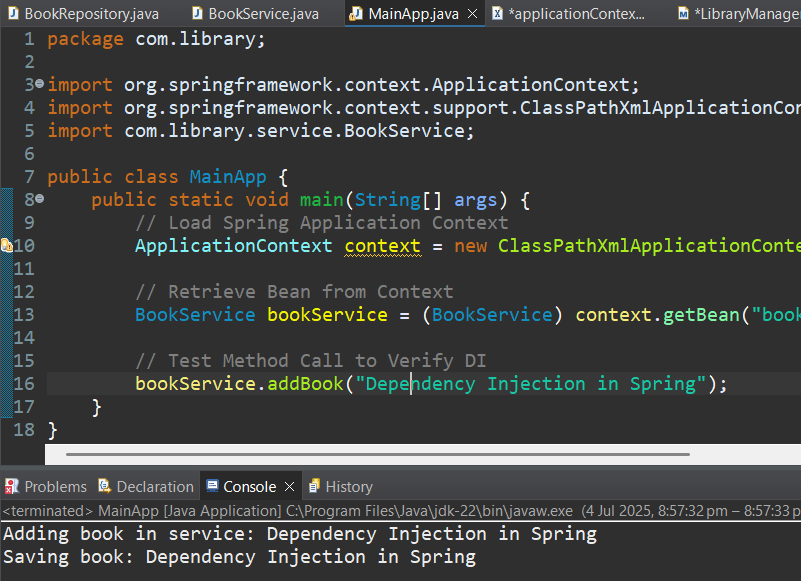
BookService bookService = (BookService) context.getBean("bookService");

// Test Method Call to Verify DI

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

}

}



**Exercise 3: Implementing Logging with Spring AOP**

**Scenario:**

The library management application requires logging capabilities to track method execution times.

**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.class);

bookService.addBook();

bookService.deleteBook();

}}

**LoggingAspect.java**

package com.library.aspect;

import org.aspectj.lang.ProceedingJoinPoint;

import org.aspectj.lang.annotation.Around;

import org.aspectj.lang.annotation.Aspect;

import org.springframework.stereotype.Component;

*@Aspect*

*@Component*

public class LoggingAspect {

*@Around*("execution(\* com.library.service.\*.\*(..))")

public Object logExecutionTime(ProceedingJoinPoint joinPoint) throws Throwable {

long start = System.*currentTimeMillis*();

Object result = joinPoint.proceed();

long duration = System.*currentTimeMillis*() - start;

System.***out***.println("Execution time of " + joinPoint.getSignature() + ": " + duration + " ms");

return result;

}

}

**Applicationcontext.xml**

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

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:aop="http://www.springframework.org/schema/aop"

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

http://www.springframework.org/schema/context https://www.springframework.org/schema/context/spring-context.xsd

http://www.springframework.org/schema/aop https://www.springframework.org/schema/aop/spring-aop.xsd">

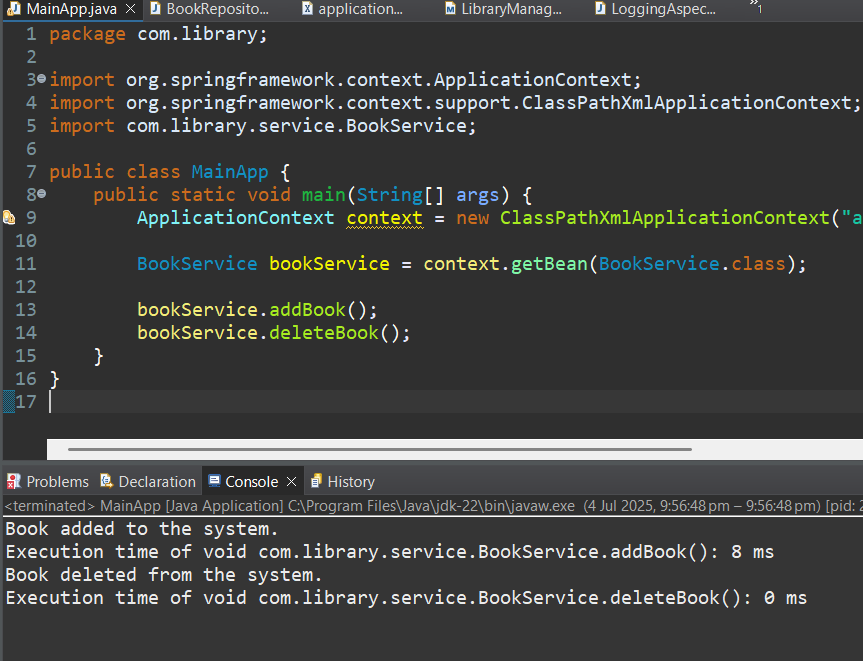
<!-- Enable @Component and @Aspect scanning -->

<context:component-scan base-package="com.library" />

<!-- Enable AOP -->

<aop:aspectj-autoproxy />

</beans>



**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.

**MainApp.java**

package com.library;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

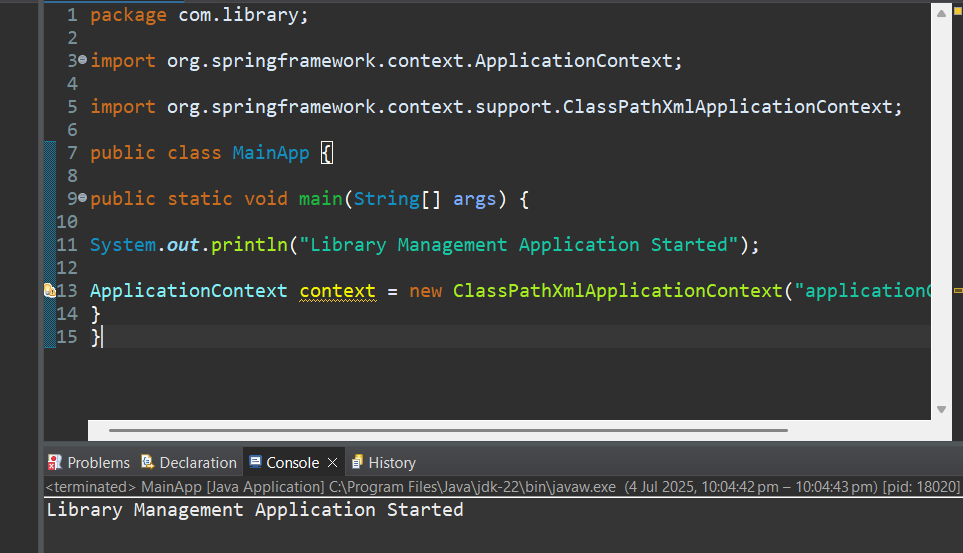
public static void main(String[] args) {

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

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

}

}



**Exercise 5: Configuring the Spring IoC Container**

**Scenario:**

The library management application requires a central configuration for beans and dependencies.

**BookRepository.java**

package com.library.repository;

public class BookRepository {

public void displayBooks() {

System.out.println("Displaying books from the repository...");

}

}

**BookService.java**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

// Setter injection

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void addBook() {

System.out.println("Book added to the system.");

bookRepository.displayBooks();

}

public void deleteBook() {

System.out.println("Book deleted from the system.");

}

}

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

bookService.addBook();

bookService.deleteBook();

}

}

**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">

<!-- BookRepository Bean -->

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

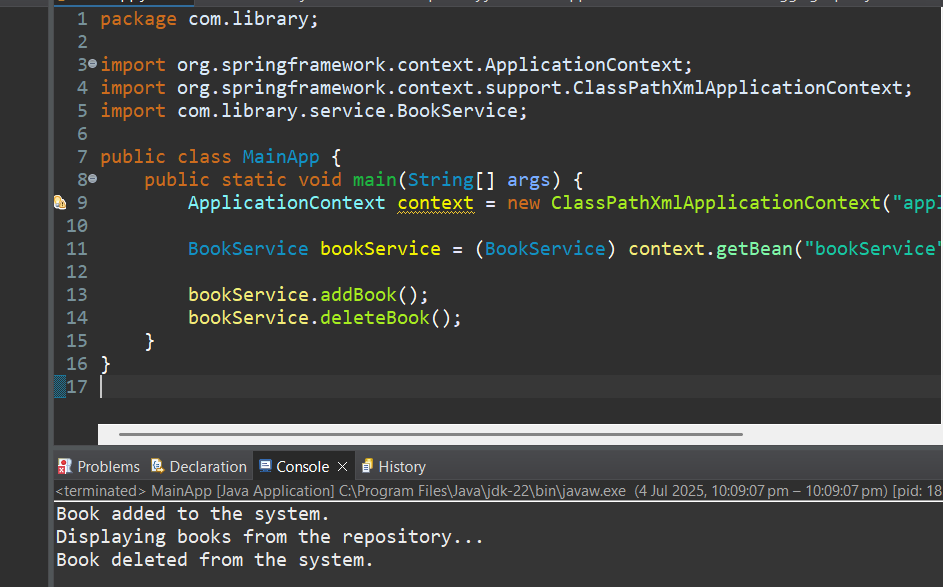
<!-- BookService Bean with setter injection -->

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

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

</bean>

</beans>



**Exercise 6: Configuring Beans with Annotations**

**Scenario:**

You need to simplify the configuration of beans in the library management application using annotations.

**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");

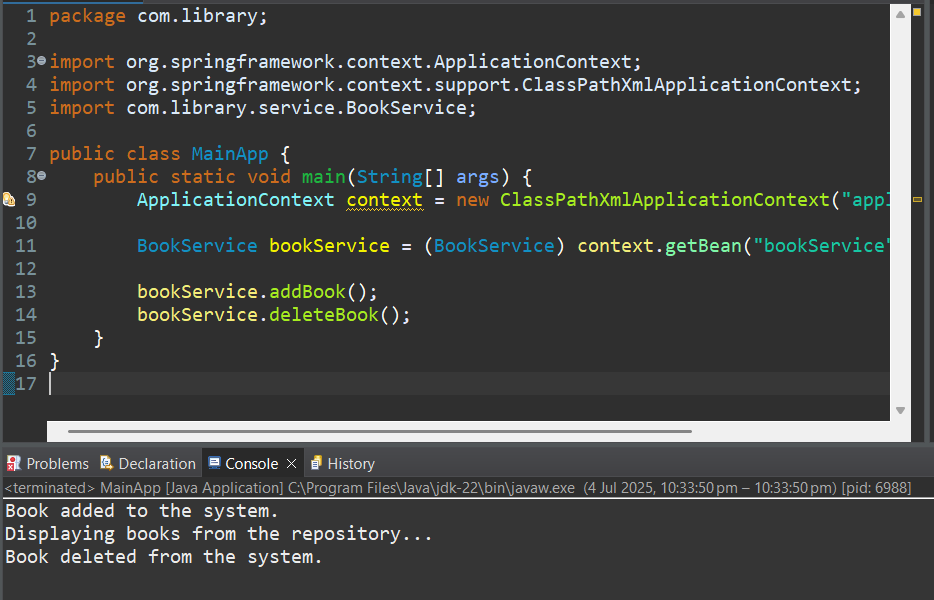
BookService bookService = context.getBean(BookService.class);

bookService.addBook();

bookService.deleteBook();

}

}



**Exercise 7: Implementing Constructor and Setter Injection**

**Scenario:**

The library management application requires both constructor and setter injection for better control over bean initialization.

**BookService.java**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private String librarianName;

private BookRepository bookRepository;

// Constructor for injecting librarian name

public BookService(String librarianName) {

this.librarianName = librarianName;

}

// Setter for injecting repository

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void addBook() {

System.out.println("[" + librarianName + "] Book added to the system.");

bookRepository.displayBooks();

}

public void deleteBook() {

System.out.println("[" + librarianName + "] Book deleted from the system.");

}

}

**BookRepository.java**

package com.library.repository;

public class BookRepository {

public void displayBooks() {

System.out.println("Displaying books from the repository...");

}

}

**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">

<!-- BookRepository Bean -->

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

<!-- BookService Bean with constructor and setter injection -->

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

<!-- Constructor injection for librarian name -->

<constructor-arg value="Priya Sharma" />

<!-- Setter injection for BookRepository -->

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

</bean>

</beans>

**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");

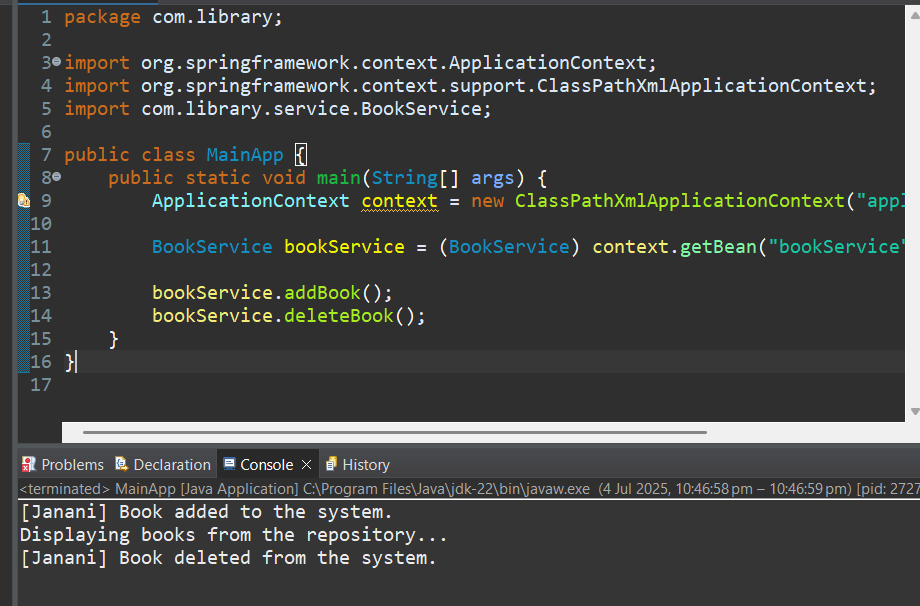
BookService bookService = (BookService) context.getBean("bookService");

bookService.addBook();

bookService.deleteBook();

}

}



**Exercise 8: Implementing Basic AOP with Spring**

**Scenario:**

The library management application requires basic AOP functionality to separate cross-cutting concerns like logging and transaction management.

**LoggingAspect.java**

package com.library.aspect;

import org.aspectj.lang.JoinPoint;

import org.aspectj.lang.annotation.Aspect;

import org.aspectj.lang.annotation.Before;

import org.aspectj.lang.annotation.After;

import org.springframework.stereotype.Component;

@Aspect

@Component

public class LoggingAspect {

@Before("execution(\* com.library.service.BookService.\*(..))")

public void beforeMethod(JoinPoint joinPoint) {

System.out.println("Before: " + joinPoint.getSignature().getName());

}

@After("execution(\* com.library.service.BookService.\*(..))")

public void afterMethod(JoinPoint joinPoint) {

System.out.println("After: " + joinPoint.getSignature().getName());

}

}

**BookService.java**

package com.library.service;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import com.library.repository.BookRepository;

@Service

public class BookService {

@Autowired

private BookRepository bookRepository;

public void addBook() {

System.out.println("Book added to the system.");

bookRepository.displayBooks();

}

public void deleteBook() {

System.out.println("Book deleted from the system.");

}

}

**BookRepository.java**

package com.library.repository;

import org.springframework.stereotype.Repository;

@Repository

public class BookRepository {

public void displayBooks() {

System.out.println("Displaying books from the repository...");

}

}

**applicationContext.xml**

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

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:context="http://www.springframework.org/schema/context"

xmlns:aop="http://www.springframework.org/schema/aop"

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

http://www.springframework.org/schema/context https://www.springframework.org/schema/context/spring-context.xsd

http://www.springframework.org/schema/aop https://www.springframework.org/schema/aop/spring-aop.xsd">

<!-- Enable component scanning -->

<context:component-scan base-package="com.library" />

<!-- Enable AspectJ auto proxy support -->

<aop:aspectj-autoproxy />

</beans>

**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");

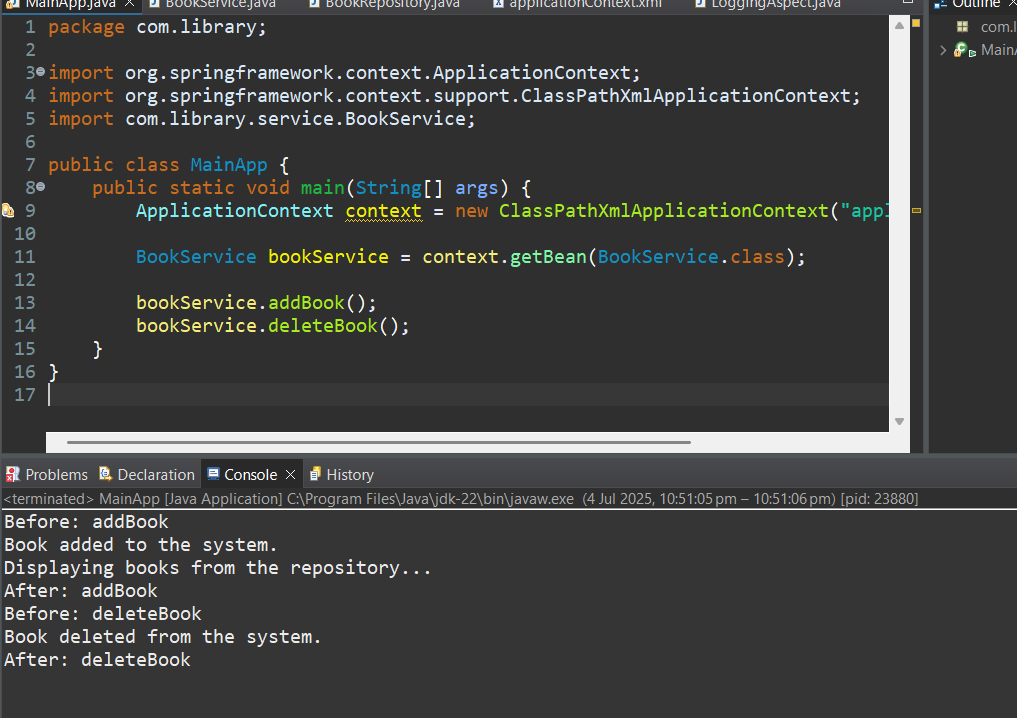
BookService bookService = context.getBean(BookService.class);

bookService.addBook();

bookService.deleteBook();

}

}



**Exercise 9: Creating a Spring Boot Application**

**Scenario:**

You need to create a Spring Boot application for the library management system to simplify configuration and deployment.

**1.Book.java**

package com.library.entity;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String title;

private String author;

public Book() {}

public Book(String title, String author) {

this.title = title;

this.author = author;

}

// Getters & setters

public Long getId() { return id; }

public String getTitle() { return title; }

public void setTitle(String title) { this.title = title; }

public String getAuthor() { return author; }

public void setAuthor(String author) { this.author = author; }

}

**2.BookController.java**

package com.library.controller;

import com.library.entity.Book;

import com.library.repository.BookRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/books")

public class BookController {

@Autowired

private BookRepository bookRepository;

@GetMapping

public List<Book> getAllBooks() {

return bookRepository.findAll();

}

@PostMapping

public Book createBook(@RequestBody Book book) {

return bookRepository.save(book);

}

@GetMapping("/{id}")

public Book getBook(@PathVariable Long id) {

return bookRepository.findById(id).orElse(null);

}

@PutMapping("/{id}")

public Book updateBook(@PathVariable Long id, @RequestBody Book bookDetails) {

Book existingBook = bookRepository.findById(id)

.orElseThrow(() -> new RuntimeException("Book not found with ID: " + id));

existingBook.setTitle(bookDetails.getTitle());

existingBook.setAuthor(bookDetails.getAuthor());

return bookRepository.save(existingBook);

}

@DeleteMapping("/{id}")

public void deleteBook(@PathVariable Long id) {

bookRepository.deleteById(id);

}

}

**3.BookRepository.java**

package com.library.repository;

import com.library.entity.Book;

import org.springframework.data.jpa.repository.JpaRepository;

public interface BookRepository extends JpaRepository<Book, Long> {

}

**4.LibraryManagementBootApplication.java**

package com.library;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class LibraryManagementBootApplication {

public static void main(String[] args) {

SpringApplication.run(LibraryManagementBootApplication.class, args);

}

}

**5.application.properties**

spring.datasource.url=jdbc:h2:mem:librarydb

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=

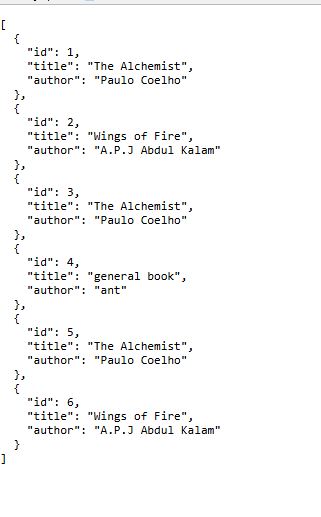
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

spring.jpa.hibernate.ddl-auto=update

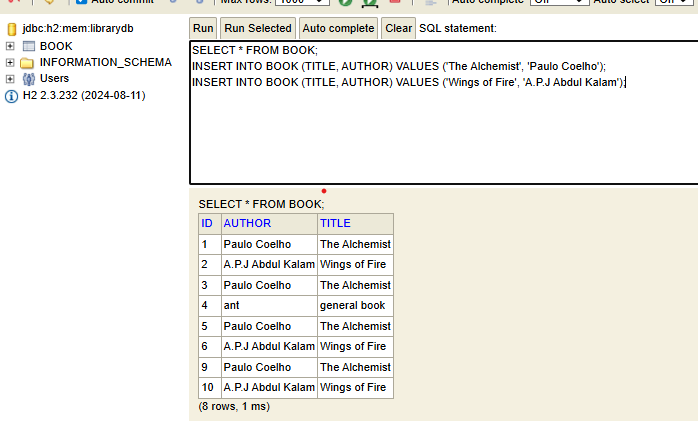
spring.h2.console.enabled=true

server.port=808

**#http://localhost:8081/books -> JSON**



**#http://localhost:8081/h2-console -> SQL DB**



#**using postman commands- POST,GET,PUT,DELETE**

