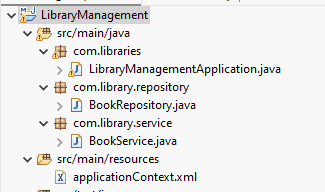
**Exercise 1: Configuring a Basic Spring Application**

**Scenario:**

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



**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 <https://maven.apache.org/xsd/maven-4.0.0.xsd>">

<modelVersion>4.0.0</modelVersion>

<groupId>com.library</groupId>

<artifactId>LibraryManagement</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>LibraryManagement</name>

<dependencies>

<!-- Spring Core -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>5.3.34</version>

</dependency>

<!-- Spring Context -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.34</version>

</dependency>

</dependencies>

<build>

<plugins>

<!-- Compiler plugin -->

<plugin>

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

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

<version>3.8.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

**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.

**Program**

**BookRespository.java**

**package** com.library.repository;

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

**public** **void** save() {

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

}

}

**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() {

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

bookRepository.save();

}

}

**LibraryManagementApplication.java**

**package** com.libraries;

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

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

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

**public** **class** LibraryManagementApplication {

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

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

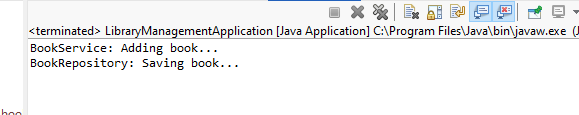
BookService bookService = (BookService) context.getBean("bookService");

bookService.addBook();

}

}

**Output**



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

**Scenario:**

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

**Program**

**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"

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

xsi:schemaLocation="http://www.springframework.org/schema/beans

<http://www.springframework.org/schema/beans/spring-beans.xsd>

<http://www.springframework.org/schema/aop>

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

<!-- Enable AOP -->

<aop:aspectj-autoproxy />

<!-- Beans -->

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

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

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

</bean>

<!-- Aspect -->

<bean class="com.library.aspect.LoggingAspect"/>

</beans>

**LibraryManagementApplication.java**

**package** com.libraries;

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

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

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

**public** **class** LibraryManagementApplication {

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

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

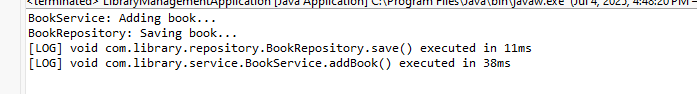
BookService bookService = (BookService) context.getBean("bookService");

bookService.addBook();

}

}

**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.

Pom.xml

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

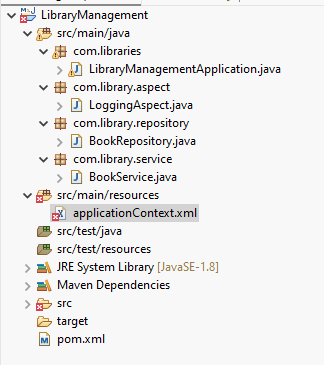
<version>5.3.34</version>

</dependency>

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

**Scenario:**

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



**Beans**

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

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

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

</bean>

**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() {

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

bookRepository.save();

}

}

**Exercise 6: Configuring Beans with Annotations**

**Scenario:**

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

**Program**

**BookRepository.java**

**package** com.library.repository;

**import** org.springframework.stereotype.Repository;

@Repository

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

**public** **void** save() {

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

}

}

**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 {

**private** BookRepository bookRepository;

@Autowired

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

**this**.bookRepository = bookRepository;

}

**public** **void** addBook() {

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

bookRepository.save();

}

}

**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"

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

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

xsi:schemaLocation="http://www.springframework.org/schema/beans

<http://www.springframework.org/schema/beans/spring-beans.xsd>

<http://www.springframework.org/schema/aop>

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

<http://www.springframework.org/schema/context>

<http://www.springframework.org/schema/context/spring-context.xsd>">

<!-- Enable AOP -->

<aop:aspectj-autoproxy />

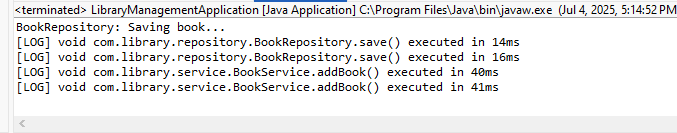
<bean class="com.library.aspect.LoggingAspect"/>

<!--Context-->

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

</beans>

**Output**



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

**Scenario:**

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

**Program**

**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 {

**private** BookRepository bookRepository;

@Autowired

**public** BookService(BookRepository bookRepository) {

**this**.bookRepository = bookRepository;

System.***out***.println("[INFO] Constructor injection used.");

}

@Autowired

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

**this**.bookRepository = bookRepository;

System.***out***.println("[INFO] Constructor injection used.");

}

**public** **void** addBook() {

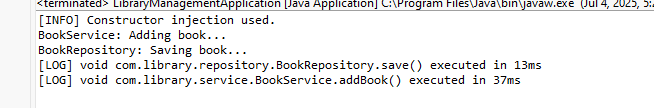
System.***out***.println("BookService: Adding book...");

bookRepository.save();

}

}

**Output**



**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.

Program

**LoggingAspect.java**

**package** com.library.aspect;

**import** org.aspectj.lang.JoinPoint;

**import** org.aspectj.lang.ProceedingJoinPoint;

**import** org.aspectj.lang.annotation.Around;

**import** org.aspectj.lang.annotation.Aspect;

**import** org.aspectj.lang.annotation.After;

**import** org.aspectj.lang.annotation.AfterReturning;

**import** org.aspectj.lang.annotation.AfterThrowing;

**import** org.aspectj.lang.annotation.Before;

**import** org.springframework.stereotype.Component;

@Aspect

@Component

**public** **class** LoggingAspect {

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

**public** **void** beforeServiceMethods(JoinPoint joinPoint) {

System.***out***.println("[AOP BEFORE] " + joinPoint.getSignature().getName() + " is about to execute.");

}

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

**public** **void** afterServiceMethods(JoinPoint joinPoint) {

System.***out***.println("[AOP AFTER] " + joinPoint.getSignature().getName() + " just finished.");

}

@AfterReturning(pointcut = "execution(\* com.library.service.\*.\*(..))", returning = "result")

**public** **void** afterReturning(JoinPoint joinPoint, Object result) {

System.***out***.println("[AOP AFTER RETURNING] " + joinPoint.getSignature().getName() + " returned successfully.");

}

@AfterThrowing(pointcut = "execution(\* com.library.repository.\*.\*(..))", throwing = "ex")

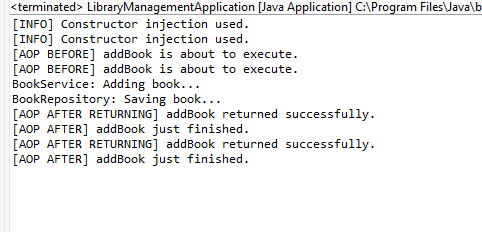
**public** **void** afterThrowing(JoinPoint joinPoint, Throwable ex) {

System.***out***.println("[AOP AFTER THROWING] " + joinPoint.getSignature().getName() + ": " + ex);

}

}

**Output**



**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.

Program

**package** com.library.model;

**import** jakarta.persistence.\*;

@Entity

**public** **class** Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

**private** Long id;

**private** String title;

**private** String author;

// Getters & Setters

**public** Long getId() { **return** id; }

**public** **void** setId(Long id) { **this**.id = 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; }

}

BookRepository.java

**package** com.library.repository;

**import** com.library.model.Book;

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

**public** **interface** BookRepository **extends** JpaRepository<Book, Long> {

}

BookController.java

**package** com.library.controller;

**import** com.library.model.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("/ping")

**public** String ping() {

**return** "pong";

}

@GetMapping

**public** List<Book> getAll() {

**return** bookRepository.findAll();

}

@PostMapping

**public** Book create(@RequestBody Book book) {

**return** bookRepository.save(book);

}

@GetMapping("/{id}")

**public** Book getById(@PathVariable Long id) {

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

}

@DeleteMapping("/{id}")

**public** **void** delete(@PathVariable Long id) {

bookRepository.deleteById(id);

}

}

Application.properties

spring.application.name=LibraryManagementBoot

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

spring.datasource.driver-class-name=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=

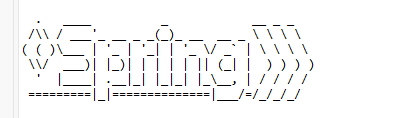
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

spring.jpa.hibernate.ddl-auto=update

spring.h2.console.enabled=true

spring.jpa.show-sql=true

Output:



:: Spring Boot :: (v3.5.3)

2025-07-05T17:23:21.198+05:30 INFO 19000 --- [LibraryManagementBoot] [ main] c.l.LibraryManagementBootApplication : Starting LibraryManagementBootApplication using Java 17.0.14 with PID 19000 (C:\Users\Administrator\eclipse-workspace\LibraryManagementBoot\target\classes started by Administrator in C:\Users\Administrator\eclipse-workspace\LibraryManagementBoot)

2025-07-05T17:23:21.205+05:30 INFO 19000 --- [LibraryManagementBoot] [ main] c.l.LibraryManagementBootApplication : No active profile set, falling back to 1 default profile: "default"

2025-07-05T17:23:22.218+05:30 INFO 19000 --- [LibraryManagementBoot] [ main] c.l.LibraryManagementBootApplication : Started LibraryManagementBootApplication in 1.943 seconds (process running for 2.629)