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

### Step 1: Set Up a Maven Project

**Project Name**: LibraryManagemet

LibraryManagement/

├── src/

│ ├── main/

│ │ ├── java/

│ │ └── resources/

└── pom.xml

So these is the structure to create maven project

**Add Spring Core Dependency in** pom.xml**:**

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>6.0.11</version>

</dependency></dependencies>

### Step 2: Configure applicationContext.xml

Path: src/main/resources/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>

### Step 3: Create Java Classes

com.library.repository.BookRepository.java

package com.library.repository;

public class BookRepository {

public void saveBook(String bookName) {

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

}

}

com.library.service.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);

}

}

Step 4: Create Main Class to Run

com.library.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");

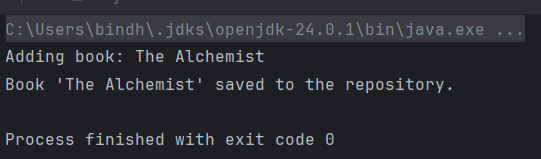
BookService bookService = (BookService) context.getBean("bookService");

bookService.addBook("The Alchemist");

}

}

**OUTPUT:**



**Exercise 2: Implementing Dependency Injection**

### Step 1: applicationContext.xml

src/main/resources/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 Repository Bean -->

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

<!-- Define Service Bean and Inject Repository -->

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

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

</bean>

</beans>

Step 2: BookService.java

src/main/java/com/library/service/BookService.java

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

// Setter method for DI

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void addBook(String bookName) {

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

bookRepository.saveBook(bookName);

}

}

Step 3: BookRepository.java

src/main/java/com/library/repository/BookRepository.java

package com.library.repository;

public class BookRepository {

public void saveBook(String bookName) {

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

}

}

Step 4: MainApp.java

src/main/java/com/library/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");

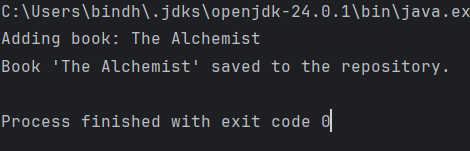
BookService bookService = (BookService) context.getBean("bookService");

bookService.addBook("The Alchemist");

}

}

**OUTPUT:**



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

### ****Step 1: Add Spring AOP Dependency****

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-aop</artifactId></dependency>

Make sure to also include:

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter</artifactId></dependency>

**Step 2: Create an Aspect for Logging**

Create package: com.library.aspect  
 Create class: LoggingAspect.java

package com.library.aspect;

import org.aspectj.lang.ProceedingJoinPoint;import org.aspectj.lang.annotation.\*;import org.springframework.stereotype.Component;

@Aspect@Componentpublic class LoggingAspect {

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

public Object logExecutionTime(ProceedingJoinPoint joinPoint) throws Throwable {

long start = System.currentTimeMillis();

Object result = joinPoint.proceed(); // proceed with actual method

long end = System.currentTimeMillis();

System.out.println(joinPoint.getSignature() + " executed in " + (end - start) + "ms");

return result;

}

}

This logs execution time of **all methods** in com.library.service package.

**Step 3: Enable AspectJ Support**

In a Spring Boot app, you don’t need applicationContext.xml. Just use:

t

@SpringBootApplication@EnableAspectJAutoProxypublic class LibraryManagementApplication {

public static void main(String[] args) {

SpringApplication.run(LibraryManagementApplication.class, args);

}

}

@EnableAspectJAutoProxy enables AOP support.

**Step 4: Test the Aspect**

Create a sample service method:

package com.library.service;

import org.springframework.stereotype.Service;

@Servicepublic class BookService {

public void issueBook() {

try {

Thread.sleep(500); // simulate delay

} catch (InterruptedException e) {

e.printStackTrace();

}

System.out.println("Book issued.");

}

}

Call this method in your controller or main app to test:

@Autowired

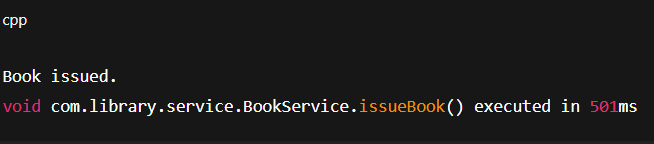
BookService bookService;

@PostConstructpublic void run() {

bookService.issueBook();

}

**OUTPUT:**

****

**Exercise 4: Creating and Configuring a Maven Project**

## ****Step 1: Create a New Maven Project****

You can use any IDE (like IntelliJ or Eclipse) or create the project manually.

Project Name: LibraryManagement

Packaging: jar

**Step 2:** pom.xml **with Spring Dependencies and Plugins**

<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>1.0.0</version>

<packaging>jar</packaging>

<name>Library Management</name>

<description>Spring AOP and Web MVC Library Project</description>

<properties>

<java.version>1.8</java.version>

<spring.version>5.3.31</spring.version> <!-- Compatible Spring 5 for Java 8 -->

</properties>

<dependencies>

<!-- Spring Context -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>${spring.version}</version>

</dependency>

<!-- Spring AOP -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>${spring.version}</version>

</dependency>

<!-- Spring Web MVC -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>${spring.version}</version>

</dependency>

<!-- AspectJ (for AOP support) -->

<dependency>

<groupId>org.aspectj</groupId>

<artifactId>aspectjweaver</artifactId>

<version>1.9.19</version>

</dependency>

<!-- For logging -->

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.36</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-simple</artifactId>

<version>1.7.36</version>

</dependency>

</dependencies>

<!-- Maven Plugins -->

<build>

<plugins>

<!-- Maven Compiler Plugin -->

<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>

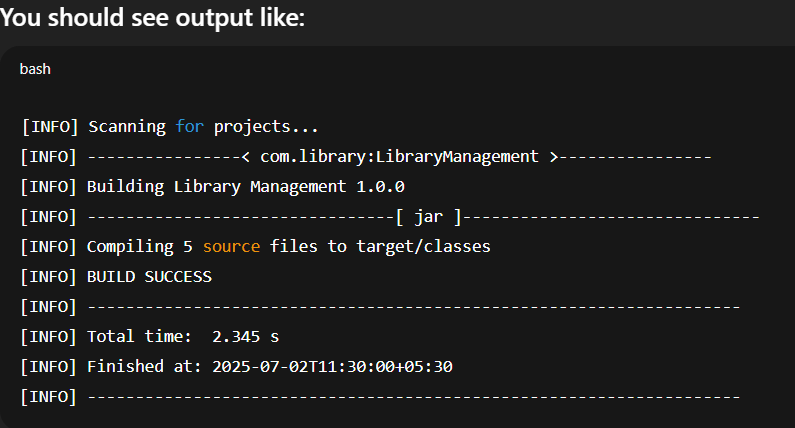
</configuration>

</plugin>

</plugins>

</build></project>

**OUTPUT:**



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

## ****Step 1: Create**** applicationContext.xml

Create the file in this path:  
src/main/resources/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>

**Step 2: BookRepository Class**

📄 src/main/java/com/library/repository/BookRepository.java

package com.library.repository;

public class BookRepository {

public void saveBook() {

System.out.println("Book saved to the repository.");

}

}

**Step 3: BookService Class with Setter Injection**

src/main/java/com/library/service/BookService.java

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

// Setter for Dependency Injection

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void issueBook() {

System.out.println("Issuing book...");

bookRepository.saveBook();

}

}

**Step 4: Main Class to Load Spring Context and Test**

📄 src/main/java/com/library/LibraryMain.java

package com.library;

import com.library.service.BookService;import org.springframework.context.ApplicationContext;import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryMain {

public static void main(String[] args) {

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

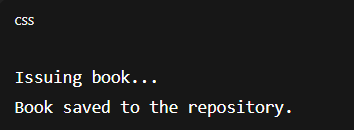
BookService bookService = context.getBean("bookService", BookService.class);

bookService.issueBook();

}

}

**OUTPUT:**



**Exercise 6: Configuring Beans with Annotations**

## Step 1: Enable Component Scanning in applicationContext.xml

src/main/resources/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: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

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

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

<!-- Enable annotation-based configuration -->

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

</beans>

**Step 2: Annotate the Java Classes**

### BookRepository.java

package com.library.repository;

import org.springframework.stereotype.Repository;

@Repositorypublic class BookRepository {

public void saveBook() {

System.out.println("Book saved to the repository.");

}

}

BookService.java

package com.library.service;

import com.library.repository.BookRepository;import org.springframework.beans.factory.annotation.Autowired;import org.springframework.stereotype.Service;

@Servicepublic class BookService {

@Autowired

private BookRepository bookRepository;

public void issueBook() {

System.out.println("Issuing book...");

bookRepository.saveBook();

}

}

**Step 3: Run the Application**

### LibraryManagementApplication.java

package com.library;

import com.library.service.BookService;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");

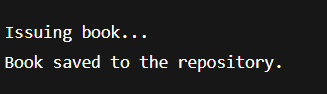
BookService bookService = context.getBean(BookService.class);

bookService.issueBook();

}

}

**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.

Steps:

1. Configure Constructor Injection:

- Update applicationContext.xml to configure constructor injection for BookService.

2. Configure Setter Injection:

- Ensure that the BookService class has a setter method for BookRepository and configure it in applicationContext.xml.

3. Test the Injection:

- Run the LibraryManagementApplication main class to verify both constructor and setter injection.

**BookRepository.java**

package com.library.repository;

public class BookRepository {

public void saveBook() {

System.out.println("Book saved to the repository.");

}

}

**BookService.java**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private String librarianName;

private BookRepository bookRepository;

// Constructor

public BookService(String librarianName) {

this.librarianName = librarianName;

}

// Setter

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void issueBook() {

System.out.println("Librarian " + librarianName + " is issuing book...");

bookRepository.saveBook();

}

}

**applicationContext.xml**xml

<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">

<constructor-arg value="Bindhu Mechineeni"/>

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

</bean>

</beans>

**LibraryManagementApplication.java**

package com.library;

import com.library.service.BookService;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");

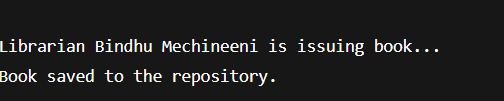
BookService bookService = context.getBean("bookService", BookService.class);

bookService.issueBook();

}

}

**OUTPUT:**



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

## ****Step 1: Define the Aspect Class****

**Package:** com.library.aspect  
**File:** LoggingAspect.java

package com.library.aspect;

import org.aspectj.lang.JoinPoint;

public class LoggingAspect {

// Before Advice

public void beforeAdvice(JoinPoint joinPoint) {

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

}

// After Advice

public void afterAdvice(JoinPoint joinPoint) {

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

}

}

**Step 2: Update BookService Class**

com.library.service.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 issueBook() {

System.out.println("Issuing book...");

bookRepository.saveBook();

}

}

**Step 3: Configure** applicationContext.xml

📄 src/main/resources/applicationContext.xml

<?xml version="1.0" encoding="UTF-8"?><beans xmlns="http://www.springframework.org/schema/beans"

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

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 />

<!-- Register Repository and Service beans -->

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

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

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

</bean>

<!-- Register Aspect bean -->

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

<!-- AOP Configuration -->

<aop:config>

<aop:aspect ref="loggingAspect">

<aop:pointcut id="bookServiceMethods"

expression="execution(\* com.library.service.BookService.\*(..))" />

<aop:before pointcut-ref="bookServiceMethods" method="beforeAdvice" />

<aop:after pointcut-ref="bookServiceMethods" method="afterAdvice" />

</aop:aspect>

</aop:config>

</beans>

**Step 4: Main Class to Test AOP**

com.library.LibraryManagementApplication.java

package com.library;

import com.library.service.BookService;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");

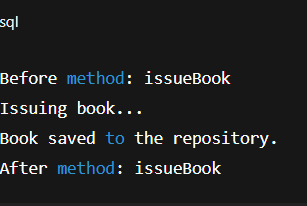
BookService bookService = context.getBean("bookService", BookService.class);

bookService.issueBook();

}

}

**OUTPUT:**



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

## ****Step 1: Create a Spring Boot Project****

Go to [https://start.spring.io](https://start.spring.io" \t "_new) and configure:

**Project Name**: LibraryManagement

**Packaging**: Jar

**Java Version**: 17 (or 8 if needed)

**Dependencies**:

Spring Web

Spring Data JPA

H2 Database

Download and extract the project.

**Step 2: Add Dependencies (automatically added)**

In pom.xml:

<dependencies>

<!-- Spring Boot Starter Web -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<!-- Spring Data JPA -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<!-- H2 Database -->

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency></dependencies>

**Step 3: Configure** application.properties

src/main/resources/application.properties

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.h2.console.enabled=true

spring.jpa.hibernate.ddl-auto=update

**Step 4: Define Entity and Repository**

### Book.java

package com.library.entity;

import jakarta.persistence.Entity;import jakarta.persistence.GeneratedValue;import jakarta.persistence.GenerationType;import jakarta.persistence.Id;

@Entitypublic class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String title;

private String author;

// Constructors

public Book() {}

public Book(String title, String author) {

this.title = title;

this.author = author;

}

// Getters and 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.entity.Book;import org.springframework.data.jpa.repository.JpaRepository;

public interface BookRepository extends JpaRepository<Book, Long> {

}

**Step 5: Create a REST Controller**

### 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 addBook(@RequestBody Book book) {

return bookRepository.save(book);

}

@GetMapping("/{id}")

public Book getBookById(@PathVariable Long id) {

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

}

@PutMapping("/{id}")

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

Book book = bookRepository.findById(id).orElse(null);

if (book != null) {

book.setTitle(bookDetails.getTitle());

book.setAuthor(bookDetails.getAuthor());

return bookRepository.save(book);

}

return null;

}

@DeleteMapping("/{id}")

public void deleteBook(@PathVariable Long id) {

bookRepository.deleteById(id);

}

}

**Step 6: Run the Application**

### LibraryManagementApplication.java

package com.library;

import org.springframework.boot.SpringApplication;import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplicationpublic class LibraryManagementApplication {

public static void main(String[] args) {

SpringApplication.run(LibraryManagementApplication.class, args);

}

}

**OUTPUT:**