**Spring Core\_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.

**Steps:**

1. **Set Up a Spring Project:**
   * Create a Maven project named **LibraryManagement**.
   * Add Spring Core dependencies in the **pom.xml** file.
2. **Configure the Application Context:**
   * Create an XML configuration file named **applicationContext.xml** in the **src/main/resources** directory.
   * Define beans for **BookService** and **BookRepository** in the XML file.
3. **Define Service and Repository Classes:**
   * Create a package **com.library.service** and add a class **BookService**.
   * Create a package **com.library.repository** and add a class **BookRepository**.
4. **Run the Application:**
   * Create a main class to load the Spring context and test the configuration.

### **Step 1: Set Up a Spring Project**

**Project Name:** LibraryManagement

1. Create a Maven project named LibraryManagement.
2. Add the following dependencies to the pom.xml file to include **Spring Core**:

<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 Core Dependency -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.34</version>

</dependency>

</dependencies>

</project>

### **Step 2: Configure the Application Context**

1. In the directory src/main/resources, create a new XML file named applicationContext.xml.
2. Define beans for BookService and BookRepository as shown below:

<?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 3: Define Service and Repository Classes**

1. Create a package com.library.repository and add the class BookRepository.java:

package com.library.repository;

public class BookRepository {

public void displayBooks() {

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

}

}

1. Create a package com.library.service and add the class 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 showBooks() {

System.out.println("BookService: Calling BookRepository...");

bookRepository.displayBooks();

}

}

### **Step 4: Run the Application**

1. Create a main class MainApp.java in a package like com.library.main:

package com.library.main;

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 Context

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

// Get BookService bean

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

// Call method to test

bookService.showBooks();

}

}

### **Output:**

vbnet

BookService: Calling BookRepository...

Displaying books from the repository...

**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.

**Steps:**

1. **Modify the XML Configuration:**
   * Update **applicationContext.xml** to wire **BookRepository** into **BookService**.
2. **Update the BookService Class:**
   * Ensure that **BookService** class has a setter method for **BookRepository**.
3. **Test the Configuration:**
   * Run the **LibraryManagementApplication** main class to verify the dependency injection.

### **Scenario:**

We need to manage the dependencies between the BookService and BookRepository classes using **Spring's IoC (Inversion of Control)** and **Dependency Injection (DI)** mechanisms.

### 

### **Step 1: Modify the XML Configuration**

Update the existing applicationContext.xml file located in src/main/resources to inject the BookRepository bean into the BookService bean using **setter-based dependency injection**.

<?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 using setter -->

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

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

</bean>

</beans>

The property tag uses the name attribute to match the **setter method name** in the BookService class (i.e., setBookRepository()).

### **Step 2: Update the BookService Class**

Ensure that the BookService class has a **setter method** to receive the BookRepository dependency.

**Location:** 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 Dependency Injection

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void showBooks() {

System.out.println("BookService: Calling BookRepository...");

bookRepository.displayBooks();

}

}

Spring will automatically call this setter method and inject the BookRepository bean at runtime.

### **Step 3: Test the Configuration**

Create or update the main class to load the Spring context and verify the dependency injection.

**Location:** src/main/java/com/library/main/MainApp.java

package com.library.main;

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 the Spring application context

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

// Get the BookService bean

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

// Call a method to verify DI

bookService.showBooks();

}

}

### 

### **Output on Console:**

vbnet

BookService: Calling BookRepository...

Displaying books from the repository...

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

**Scenario:**

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

**Steps:**

1. **Add Spring AOP Dependency:**
   * Update **pom.xml** to include Spring AOP dependency.
2. **Create an Aspect for Logging:**
   * Create a package **com.library.aspect** and add a class **LoggingAspect** with a method to log execution times.
3. **Enable AspectJ Support:**
   * Update **applicationContext.xml** to enable **AspectJ** support and register the aspect.
4. **Test the Aspect:**
   * Run the **LibraryManagementApplication** main class and observe the console for log messages indicating method execution times.

### **Scenario:**

The application should log **method execution times** using **Spring AOP (Aspect-Oriented Programming)**.

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

Update your pom.xml to include the **Spring AOP** and **AspectJ** dependencies.

<dependencies>

<!-- Spring Context -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.34</version>

</dependency>

<!-- Spring AOP -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aspects</artifactId>

<version>5.3.34</version>

</dependency>

<!-- AspectJ Weaver -->

<dependency>

<groupId>org.aspectj</groupId>

<artifactId>aspectjweaver</artifactId>

<version>1.9.21</version>

</dependency>

</dependencies>

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

1. Create the package: com.library.aspect.
2. Inside it, create the class LoggingAspect.java:

package com.library.aspect;

import org.aspectj.lang.ProceedingJoinPoint;

import org.aspectj.lang.annotation.Around;

import org.aspectj.lang.annotation.Aspect;

@Aspect

public class LoggingAspect {

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

public Object logExecutionTime(ProceedingJoinPoint joinPoint) throws Throwable {

long start = System.currentTimeMillis();

Object result = joinPoint.proceed(); // execute the method

long end = System.currentTimeMillis();

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

return result;

}

}

The @Around annotation logs method execution time for all methods in com.library.service.

### **Step 3: Enable AspectJ Support in applicationContext.xml**

Update applicationContext.xml to enable AspectJ support and register the LoggingAspect bean.

<?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 AspectJ Auto Proxy -->

<aop:aspectj-autoproxy/>

<!-- Bean Definitions -->

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

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

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

</bean>

<!-- Register Logging Aspect -->

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

</beans>

<aop:aspectj-autoproxy/> is required to activate AspectJ support in Spring.

### **Step 4: Test the Aspect**

Run the main application class to test the logging functionality.

**Main Class (if not already created):**

package com.library.main;

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.showBooks(); // This should now also log execution time

}

}

### **Output on Console:**

csharp

BookService: Calling BookRepository...

Displaying books from the repository...

Method void com.library.service.BookService.showBooks() executed in 10 ms

**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.

**Steps:**

1. **Create a New Maven Project:**
   * Create a new Maven project named **LibraryManagement**.
2. **Add Spring Dependencies in pom.xml:**
   * Include dependencies for Spring Context, Spring AOP, and Spring WebMVC.
3. **Configure Maven Plugins:**
   * Configure the Maven Compiler Plugin for Java version 1.8 in the pom.xml file.

### **Scenario:**

Set up a new Maven project for the Library Management application and configure it with all required Spring dependencies and compiler settings.

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

Create a new Maven project with the name:

nginx

LibraryManagement

You can do this manually in an IDE like **Eclipse**, **IntelliJ**, or by using the command line:

mvn archetype:generate -DgroupId=com.library -DartifactId=LibraryManagement -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

This will generate a basic project structure:

css

LibraryManagement/

├── pom.xml

└── src/

├── main/

│ ├── java/

│ └── resources/

└── test/

### **Step 2: Add Spring Dependencies in pom.xml**

Open the pom.xml file and add the following dependencies to include **Spring Context**, **Spring AOP**, and **Spring WebMVC**:

<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 -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.34</version>

</dependency>

<!-- Spring AOP -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aspects</artifactId>

<version>5.3.34</version>

</dependency>

<!-- Spring WebMVC -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.34</version>

</dependency>

<!-- AspectJ Weaver for AOP -->

<dependency>

<groupId>org.aspectj</groupId>

<artifactId>aspectjweaver</artifactId>

<version>1.9.21</version>

</dependency>

</dependencies>

### **Step 3: Configure Maven Compiler Plugin**

Still in the pom.xml, add the Maven Compiler Plugin inside the <build> tag to ensure the project compiles with **Java 1.8**:

<build>

<plugins>

<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>

### **✅ Final pom.xml Overview (Ready to Copy):**

<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 -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.34</version>

</dependency>

<!-- Spring AOP -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aspects</artifactId>

<version>5.3.34</version>

</dependency>

<!-- Spring WebMVC -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.34</version>

</dependency>

<!-- AspectJ Weaver -->

<dependency>

<groupId>org.aspectj</groupId>

<artifactId>aspectjweaver</artifactId>

<version>1.9.21</version>

</dependency>

</dependencies>

<build>

<plugins>

<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>

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

**Scenario:**

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

**Steps:**

1. **Create Spring Configuration File:**
   * Create an XML configuration file named **applicationContext.xml** in the **src/main/resources** directory.
   * Define beans for **BookService** and **BookRepository** in the XML file.
2. **Update the BookService Class:**
   * Ensure that the **BookService** class has a setter method for **BookRepository**.
3. **Run the Application:**
   * Create a main class to load the Spring context and test the configuration.

### **Scenario:**

Set up a centralized Spring configuration file (applicationContext.xml) that manages the application's beans and dependencies using the Spring IoC (Inversion of Control) container.

### **Step 1: Create Spring Configuration File**

**Location:** src/main/resources/applicationContext.xml

**Content of 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 with dependency injected -->

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

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

</bean>

</beans>

### **Step 2: Update the BookService Class**

**File:** 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 Dependency Injection

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void showBooks() {

System.out.println("BookService: Calling BookRepository...");

bookRepository.displayBooks();

}

}

**Also create this class if not already done:**

**File:** src/main/java/com/library/repository/BookRepository.java

package com.library.repository;

public class BookRepository {

public void displayBooks() {

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

}

}

### **Step 3: Run the Application**

**File:** src/main/java/com/library/main/MainApp.java

package com.library.main;

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 the Spring application context from XML

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

// Get BookService bean from the context

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

// Call method to test if DI works

bookService.showBooks();

}

}

### **Console Output:**

vbnet

BookService: Calling BookRepository...

Displaying books from the repository...

**Exercise 6: Configuring Beans with Annotations**

**Scenario:**

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

**Steps:**

1. **Enable Component Scanning:**
   * Update **applicationContext.xml** to include component scanning for the **com.library** package.
2. **Annotate Classes:**
   * Use **@Service** annotation for the **BookService** class.
   * Use **@Repository** annotation for the **BookRepository** class.
3. **Test the Configuration:**
   * Run the **LibraryManagementApplication** main class to verify the annotation-based configuration.

### **Scenario:**

To simplify Spring configuration, we will use **annotations** instead of defining beans in applicationContext.xml.

### **Step 1: Enable Component Scanning**

Update applicationContext.xml to enable component scanning for the com.library package.

**File:** 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"

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/context

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

<!-- Enable annotation-based configuration and scan the com.library package -->

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

</beans>

✅ This tells Spring to automatically detect and register beans annotated with @Component, @Service, @Repository, or @Controller in the specified package.

### **Step 2: Annotate Classes**

#### BookService class with @Service

**File:** src/main/java/com/library/service/BookService.java

package com.library.service;

import com.library.repository.BookRepository;

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

import org.springframework.stereotype.Service;

@Service

public class BookService {

private BookRepository bookRepository;

// Constructor-based injection (can also use setter)

@Autowired

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void showBooks() {

System.out.println("BookService: Calling BookRepository...");

bookRepository.displayBooks();

}

}

#### BookRepository class with @Repository

**File:** src/main/java/com/library/repository/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...");

}

}

@Autowired automatically injects the BookRepository bean into BookService.

### **Step 3: Test the Configuration**

**Main Application Class:** **File:** src/main/java/com/library/main/MainApp.java

package com.library.main;

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 the Spring context

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

// Retrieve the BookService bean

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

// Call the method to test annotation-based configuration

bookService.showBooks();

}

}

### **Console Output:**

vbnet

BookService: Calling BookRepository...

Displaying books from the repository...

**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.

### **Scenario:**

The goal is to configure **both constructor and setter injection** in the Spring application to demonstrate multiple ways of injecting dependencies.

### **Step 1: Configure Constructor Injection in applicationContext.xml**

Update the applicationContext.xml file to define a BookService bean that receives a BookRepository object via **constructor injection**, and then modify it to also use **setter injection**.

**File:** 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 with constructor and setter injection -->

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

<!-- Constructor Injection -->

<constructor-arg ref="bookRepository"/>

<!-- Setter Injection -->

<property name="message" value="Library initialized with both constructor and setter injection."/>

</bean>

</beans>

### **Step 2: Update the BookService Class**

Modify the BookService class to:

* Accept BookRepository via **constructor injection**.
* Accept a String message via **setter injection**.

**File:** src/main/java/com/library/service/BookService.java

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

private String message;

// Constructor Injection

public BookService(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

// Setter Injection

public void setMessage(String message) {

this.message = message;

}

public void showBooks() {

System.out.println("BookService: Calling BookRepository...");

bookRepository.displayBooks();

System.out.println("Setter-injected message: " + message);

}

}

### **BookRepository Class (No Change if Already Present)**

**File:** src/main/java/com/library/repository/BookRepository.java

package com.library.repository;

public class BookRepository {

public void displayBooks() {

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

}

}

### **Step 3: Test the Injection in the Main Class**

Run the main class to verify that both constructor and setter injection work correctly.

**File:** src/main/java/com/library/main/MainApp.java

package com.library.main;

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 the Spring context

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

// Get the BookService bean

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

// Call method to verify injection

bookService.showBooks();

}

}

### **Output:**

sql

BookService: Calling BookRepository...

Displaying books from the repository...

Setter-injected message: Library initialized with both constructor and setter injection.

**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.

**Steps:**

1. **Define an Aspect:**
   * Create a package **com.library.aspect** and add a class **LoggingAspect**.
2. **Create Advice Methods:**
   * Define advice methods in **LoggingAspect** for logging before and after method execution.
3. **Configure the Aspect:**
   * Update **applicationContext.xml** to register the aspect and enable **AspectJ** auto-proxying.
4. **Test the Aspect:**
   * Run the **LibraryManagementApplication** main class to verify the AOP functionality.

### **Scenario:**

To implement **Aspect-Oriented Programming (AOP)** for handling cross-cutting concerns like logging and transaction management using Spring.

### **Step 1: Define an Aspect**

Create a new package and class for the logging aspect.

**Location:** src/main/java/com/library/aspect/LoggingAspect.java

package com.library.aspect;

import org.aspectj.lang.JoinPoint;

import org.aspectj.lang.annotation.After;

import org.aspectj.lang.annotation.Aspect;

import org.aspectj.lang.annotation.Before;

@Aspect

public class LoggingAspect {

// Before advice

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

public void logBeforeMethod(JoinPoint joinPoint) {

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

}

// After advice

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

public void logAfterMethod(JoinPoint joinPoint) {

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

}

}

This aspect logs method names before and after the execution of any method inside com.library.service.

### **Step 2: Create Advice Methods**

Already included in the LoggingAspect class above:

* logBeforeMethod() is called **before** any service method.
* logAfterMethod() is called **after** any service method.

### **Step 3: Configure the Aspect**

Update applicationContext.xml to:

* Enable AspectJ auto-proxying.
* Register the aspect bean.

**File:** 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"

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 component scanning if needed -->

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

<!-- Enable AOP proxy support -->

<aop:aspectj-autoproxy/>

<!-- Define repository and service beans -->

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

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

<constructor-arg ref="bookRepository"/>

</bean>

<!-- Register the LoggingAspect bean -->

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

</beans>

aop:aspectj-autoproxy tells Spring to look for @Aspect classes and enable AOP.

### **Step 4: Test the Aspect**

Run the main class to verify the AOP logging.

**File:** src/main/java/com/library/main/MainApp.java

package com.library.main;

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.showBooks(); // This should trigger logging aspect

}

}

**Console Output:**

sql

Before method: showBooks

BookService: Calling BookRepository...

Displaying books from the repository...

After method: showBooks

**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.

**Steps:**

1. **Create a Spring Boot Project:**
   * Use **Spring Initializr** to create a new Spring Boot project named **LibraryManagement**.
2. **Add Dependencies:**
   * Include dependencies for **Spring Web, Spring Data JPA, and H2 Database**.
3. **Create Application Properties:**
   * Configure database connection properties in **application.properties**.
4. **Define Entities and Repositories:**
   * Create **Book** entity and **BookRepository** interface.
5. **Create a REST Controller:**
   * Create a **BookController** class to handle CRUD operations.
6. **Run the Application:**
   * Run the Spring Boot application and test the REST endpoints.

### **Scenario:**

You are required to build a Spring Boot-based library management system that supports CRUD operations with simplified configuration using Spring Boot and in-memory H2 database.

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

Use [https://start.spring.io](https://start.spring.io/) to generate the Spring Boot project.

* **Project Name:** LibraryManagement
* **Project Type:** Maven
* **Language:** Java
* **Spring Boot Version:** 2.7.18 or 3.x
* **Packaging:** Jar
* **Java Version:** 1.8 or 17 (use 1.8 if mentioned)
* **Group:** com.library
* **Artifact:** LibraryManagement

### **Step 2: Add Dependencies**

Select the following dependencies on Spring Initializr:

* **Spring Web**
* **Spring Data JPA**
* **H2 Database**

These will be automatically added to pom.xml like below:

<dependencies>

<!-- Spring Boot Starter for Web -->

<dependency>

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

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

</dependency>

<!-- Spring Boot Starter for JPA -->

<dependency>

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

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

</dependency>

<!-- In-Memory H2 Database -->

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

</dependencies>

### **Step 3: Create Application Properties**

Create the application.properties file inside src/main/resources:

# H2 Console

spring.h2.console.enabled=true

spring.h2.console.path=/h2-console

# DataSource Configuration

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

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=

# JPA Configuration

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

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

### **Step 4: Define Entities and Repositories**

#### Book Entity

**File:** src/main/java/com/library/model/Book.java

package com.library.model;

import javax.persistence.\*;

@Entity

public 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 Interface

**File:** src/main/java/com/library/repository/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> {

}

### **Step 5: Create a REST Controller**

#### BookController

**File:** src/main/java/com/library/controller/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;

import java.util.Optional;

@RestController

@RequestMapping("/books")

public class BookController {

@Autowired

private BookRepository bookRepository;

// Get all books

@GetMapping

public List<Book> getAllBooks() {

return bookRepository.findAll();

}

// Get book by ID

@GetMapping("/{id}")

public Optional<Book> getBookById(@PathVariable Long id) {

return bookRepository.findById(id);

}

// Add new book

@PostMapping

public Book addBook(@RequestBody Book book) {

return bookRepository.save(book);

}

// Update book

@PutMapping("/{id}")

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

Book book = bookRepository.findById(id).orElseThrow();

book.setTitle(bookDetails.getTitle());

book.setAuthor(bookDetails.getAuthor());

return bookRepository.save(book);

}

// Delete book

@DeleteMapping("/{id}")

public void deleteBook(@PathVariable Long id) {

bookRepository.deleteById(id);

}

}

### **Step 6: Run the Application**

#### Main Application Class

**File:** src/main/java/com/library/LibraryManagementApplication.java

package com.library;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class LibraryManagementApplication {

public static void main(String[] args) {

SpringApplication.run(LibraryManagementApplication.class, args);

}

}

### **Testing REST Endpoints**

You can use **Postman** or **curl** to test the following endpoints:

| **Method** | **Endpoint** | **Description** |
| --- | --- | --- |
| GET | /books | Get all books |
| GET | /books/{id} | Get book by ID |
| POST | /books | Add a new book |
| PUT | /books/{id} | Update existing book |
| DELETE | /books/{id} | Delete a book |

Sample JSON for POST/PUT:

{

"title": "The Alchemist",

"author": "Paulo Coelho"

}

H2 Console available at: http://localhost:8080/h2-console  
 (JDBC URL: jdbc:h2:mem:librarydb)