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

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

**Solution**

**pom.xml**

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

<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>0.0.1-SNAPSHOT</version>

<name>LibraryManagement</name>

<url>http://www.example.com</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<maven.compiler.release>17</maven.compiler.release>

</properties>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.junit</groupId>

<artifactId>junit-bom</artifactId>

<version>5.11.0</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<dependencies>

<!-- Spring Core (IoC Container) -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.33</version>

</dependency>

<!-- Spring AOP -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aspects</artifactId>

<version>5.3.33</version>

</dependency>

<!-- Spring Web MVC -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.33</version>

</dependency>

<!-- JUnit 5 (Test) -->

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter-api</artifactId>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter-params</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<pluginManagement>

<plugins>

<plugin>

<artifactId>maven-clean-plugin</artifactId>

<version>3.4.0</version>

</plugin>

<plugin>

<artifactId>maven-resources-plugin</artifactId>

<version>3.3.1</version>

</plugin>

<plugin>

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

<version>3.13.0</version>

</plugin>

<plugin>

<artifactId>maven-surefire-plugin</artifactId>

<version>3.3.0</version>

</plugin>

<plugin>

<artifactId>maven-jar-plugin</artifactId>

<version>3.4.2</version>

</plugin>

<plugin>

<artifactId>maven-install-plugin</artifactId>

<version>3.1.2</version>

</plugin>

<plugin>

<artifactId>maven-deploy-plugin</artifactId>

<version>3.1.2</version>

</plugin>

<plugin>

<artifactId>maven-site-plugin</artifactId>

<version>3.12.1</version>

</plugin>

<plugin>

<artifactId>maven-project-info-reports-plugin</artifactId>

<version>3.6.1</version>

</plugin>

</plugins>

</pluginManagement>

</build>

</project>

**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">

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

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

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

</bean>

</beans>

**BookRepository.java**

package com.library.repository;

public class BookRepository {

public void saveBook(String title) {

System.out.println("Book saved: " + title);

}

}

**BookService.java**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

// Setter injection for exercise 2

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void addBook(String title) {

System.out.println("BookService: Adding book - " + title);

bookRepository.saveBook(title);

}

}

**LibraryManagementApplication.java**

package com.library.LibraryManagement;

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");

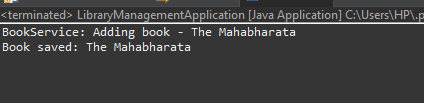
BookService bookService = (BookService) context.getBean("bookService");

bookService.addBook("The Mahabharata");

}

}

**Output**



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

**Solution**

**pom.xml (updated)**

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

<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>0.0.1-SNAPSHOT</version>

<name>LibraryManagement</name>

<url>http://www.example.com</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<maven.compiler.release>17</maven.compiler.release>

</properties>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.junit</groupId>

<artifactId>junit-bom</artifactId>

<version>5.11.0</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<dependencies>

<!-- Spring Core (IoC Container) -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.33</version>

</dependency>

<!-- Spring AOP -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aspects</artifactId>

<version>5.3.33</version>

</dependency>

<!-- Spring Web MVC -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.33</version>

</dependency>

<!-- JUnit 5 (Test) -->

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter-api</artifactId>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter-params</artifactId>

<scope>test</scope>

</dependency>

**<dependency>**

**<groupId>org.springframework</groupId>**

**<artifactId>spring-aop</artifactId>**

**<version>5.3.33</version>**

**</dependency>**

**<dependency>**

**<groupId>org.aspectj</groupId>**

**<artifactId>aspectjweaver</artifactId>**

**<version>1.9.21</version>**

**</dependency>**

</dependencies>

<build>

<pluginManagement>

<plugins>

<plugin>

<artifactId>maven-clean-plugin</artifactId>

<version>3.4.0</version>

</plugin>

<plugin>

<artifactId>maven-resources-plugin</artifactId>

<version>3.3.1</version>

</plugin>

<plugin>

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

<version>3.13.0</version>

</plugin>

<plugin>

<artifactId>maven-surefire-plugin</artifactId>

<version>3.3.0</version>

</plugin>

<plugin>

<artifactId>maven-jar-plugin</artifactId>

<version>3.4.2</version>

</plugin>

<plugin>

<artifactId>maven-install-plugin</artifactId>

<version>3.1.2</version>

</plugin>

<plugin>

<artifactId>maven-deploy-plugin</artifactId>

<version>3.1.2</version>

</plugin>

<plugin>

<artifactId>maven-site-plugin</artifactId>

<version>3.12.1</version>

</plugin>

<plugin>

<artifactId>maven-project-info-reports-plugin</artifactId>

<version>3.6.1</version>

</plugin>

</plugins>

</pluginManagement>

</build>

</project>

**ApplicationContext.xml (updated)**

<?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

https://www.springframework.org/schema/beans/spring-beans.xsd

**http://www.springframework.org/schema/aop**

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

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

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

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

</bean>

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

**<aop:aspectj-autoproxy />**

</beans>

**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.BookService.\*(..))")

public Object logExecutionTime(ProceedingJoinPoint joinPoint) throws Throwable {

long start = System.currentTimeMillis();

Object result = joinPoint.proceed();

long duration = System.currentTimeMillis() - start;

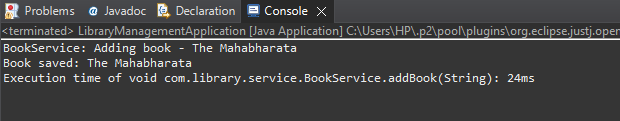
System.out.println("Execution time of " + joinPoint.getSignature() + ": " + duration + "ms");

return result;

}

}

**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.

**Steps:**

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

**Solution**

**pom.xml**

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

<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>Library\_Management</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>Library\_Management</name>

<!-- FIXME change it to the project's website -->

<url>http://www.example.com</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<maven.compiler.release>17</maven.compiler.release>

</properties>

<dependencyManagement>

<dependencies>

<dependency>

<groupId>org.junit</groupId>

<artifactId>junit-bom</artifactId>

<version>5.11.0</version>

<type>pom</type>

<scope>import</scope>

</dependency>

</dependencies>

</dependencyManagement>

<dependencies>

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter-api</artifactId>

<scope>test</scope>

</dependency>

<!-- Optionally: parameterized tests support -->

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter-params</artifactId>

<scope>test</scope>

</dependency>

**<!-- Spring Core (IoC Container) -->**

**<dependency>**

**<groupId>org.springframework</groupId>**

**<artifactId>spring-context</artifactId>**

**<version>5.3.33</version>**

**</dependency>**

**<!-- Spring AOP -->**

**<dependency>**

**<groupId>org.springframework</groupId>**

**<artifactId>spring-aop</artifactId>**

**<version>5.3.33</version>**

**</dependency>**

**<dependency>**

**<groupId>org.springframework</groupId>**

**<artifactId>spring-aspects</artifactId>**

**<version>5.3.33</version>**

**</dependency>**

**<!-- Spring Web MVC (for future use in web project or REST) -->**

**<dependency>**

**<groupId>org.springframework</groupId>**

**<artifactId>spring-webmvc</artifactId>**

**<version>5.3.33</version>**

**</dependency>**

**<!-- AspectJ for AOP -->**

**<dependency>**

**<groupId>org.aspectj</groupId>**

**<artifactId>aspectjweaver</artifactId>**

**<version>1.9.21</version>**

**</dependency>**

**<!-- JUnit for testing -->**

**<dependency>**

**<groupId>org.junit.jupiter</groupId>**

**<artifactId>junit-jupiter-api</artifactId>**

**<scope>test</scope>**

**</dependency>**

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter-params</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<pluginManagement><!-- lock down plugins versions to avoid using Maven defaults (may be moved to parent pom) -->

<plugins>

<!-- clean lifecycle, see https://maven.apache.org/ref/current/maven-core/lifecycles.html#clean\_Lifecycle -->

<plugin>

<artifactId>maven-clean-plugin</artifactId>

<version>3.4.0</version>

</plugin>

<!-- default lifecycle, jar packaging: see https://maven.apache.org/ref/current/maven-core/default-bindings.html#Plugin\_bindings\_for\_jar\_packaging -->

<plugin>

<artifactId>maven-resources-plugin</artifactId>

<version>3.3.1</version>

</plugin>

<plugin>

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

<version>3.13.0</version>

</plugin>

<plugin>

<artifactId>maven-surefire-plugin</artifactId>

<version>3.3.0</version>

</plugin>

<plugin>

<artifactId>maven-jar-plugin</artifactId>

<version>3.4.2</version>

</plugin>

<plugin>

<artifactId>maven-install-plugin</artifactId>

<version>3.1.2</version>

</plugin>

<plugin>

<artifactId>maven-deploy-plugin</artifactId>

<version>3.1.2</version>

</plugin>

<!-- site lifecycle, see https://maven.apache.org/ref/current/maven-core/lifecycles.html#site\_Lifecycle -->

<plugin>

<artifactId>maven-site-plugin</artifactId>

<version>3.12.1</version>

</plugin>

<plugin>

<artifactId>maven-project-info-reports-plugin</artifactId>

<version>3.6.1</version>

</plugin>

</plugins>

</pluginManagement>

**<plugins>**

**<plugin>**

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

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

**<version>3.10.1</version>**

**<configuration>**

**<source>17</source>**

**<target>17</target>**

**</configuration>**

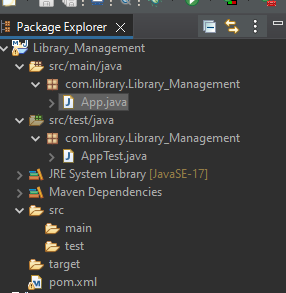
**</plugin>**

**</plugins>**

</build>

</project>

**Maven Project Library\_Management created**



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

**Solution**

**Application.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>

**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: " + title);

bookRepository.saveBook(title);

}

}

**BookRepository.java**

package com.library.repository;

public class BookRepository {

public void saveBook(String title) {

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

}

}

**LibraryManagementApplication.java**

package com.library.Library\_Management;

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");

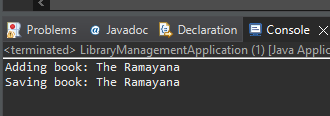
BookService bookService = context.getBean("bookService", BookService.class);

bookService.addBook("The Ramayana");

}

}

**Output**



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

**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 component scanning -->

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

</beans>

**BookService.java**

package com.library.service;

import org.springframework.stereotype.Service;

import com.library.repository.BookRepository;

@Service

public class BookService {

private BookRepository bookRepository;

public void addBook(String title) {

System.out.println("Book added: " + title);

}

// Setter for DI

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

}

**BookRepository.java**

package com.library.repository;

import org.springframework.stereotype.Repository;

@Repository

public class BookRepository {

public void save(String title) {

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

}

}

**LibraryManagementApplication.java**

package com.library.Library\_Management;

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");

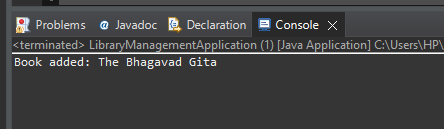
BookService bookService = context.getBean(BookService.class);

bookService.addBook("The Bhagavad Gita");

}

}

**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:**
   1. Update applicationContext.**xml** to configure constructor injection for **BookService**.
2. **Configure Setter Injection:**
   1. Ensure that the **BookService** class has a setter method for **BookRepository** and configure it in **applicationContext.xml**.
3. **Test the Injection:**
   1. Run the **LibraryManagementApplication** main class to verify both constructor and setter injection.

**Solution (Constructor)**

**ApplicationContext.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">

<!-- BookRepository Bean -->

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

<!-- BookService Bean using Constructor Injection -->

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

<constructor-arg ref="bookRepository"/>

<!-- Setter injection can also be used here (optional) -->

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

</bean>

</beans>

**BookService.java**

package com.library.service;

import com.library.repository.BookRepository;

import org.springframework.stereotype.Service;

@Service

public class BookService {

private BookRepository bookRepository;

// Constructor Injection

public BookService(BookRepository bookRepository) {

System.out.println("Constructor injection called");

this.bookRepository = bookRepository;

}

public void addBook(String title) {

System.out.println("Book added: " + title);

bookRepository.save(title);

}

}

**BookRepository.java**

package com.library.repository;

import org.springframework.stereotype.Repository;

@Repository

public class BookRepository {

public void save(String title) {

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

}

}

**LibraryManagementApplication.java**

package com.library.Library\_Management;

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");

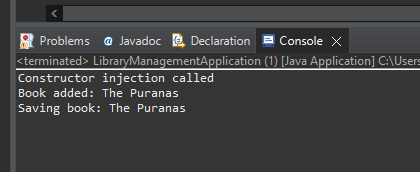
BookService bookService = context.getBean("bookService", BookService.class);

bookService.addBook("The Puranas");

}

}

**Output**



**Setter Injection**

**ApplicationContext.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">

<!-- 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>

**BookService.java**

package com.library.service;

import com.library.repository.BookRepository;

import org.springframework.stereotype.Service;

@Service

public class BookService {

private BookRepository bookRepository;

// Setter Injection

public void setBookRepository(BookRepository bookRepository) {

System.out.println("Setter injection called");

this.bookRepository = bookRepository;

}

public void addBook(String title) {

System.out.println("Book added: " + title);

bookRepository.save(title);

}

}

**BookRepository.java**

package com.library.repository;

import org.springframework.stereotype.Repository;

@Repository

public class BookRepository {

public void save(String title) {

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

}

}

**LibraryManagementApplication.java**

package com.library.Library\_Management;

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");

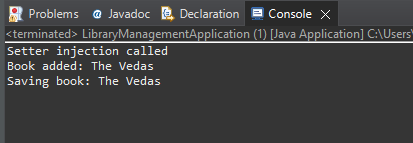
BookService bookService = context.getBean("bookService", BookService.class);

bookService.addBook("The Vedas");

}

}

**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.

**Steps:**

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

**Solution**

**applicationContext.xml**

<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

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

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

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

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

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

<!-- Enable annotation configuration and scanning -->

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

<!-- Enable AOP -->

<aop:aspectj-autoproxy/>

</beans>

**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 injection (optional, preferred)

@Autowired

public BookService(BookRepository bookRepository) {

this.bookRepository = bookRepository;

System.out.println("Constructor injection called");

}

// Setter injection (optional if constructor is used)

@Autowired

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

System.out.println("Setter injection called");

}

public void addBook(String title) {

System.out.println("Book added: " + title);

bookRepository.save(title); // NullPointerException if not injected

}

}

**BookRepository.java**

package com.library.repository;

import org.springframework.stereotype.Repository;

@Repository

public class BookRepository {

public void save(String title) {

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

}

}

**LoggingAspect.java**

package com.library.aspect;

import org.aspectj.lang.JoinPoint;

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

import org.springframework.stereotype.Component;

@Aspect

@Component

public class LoggingAspect {

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

public void logBefore(JoinPoint joinPoint) {

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

}

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

public void logAfter(JoinPoint joinPoint) {

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

}

}

**LibraryManagementApplication.java**

package com.library.Library\_Management;

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");

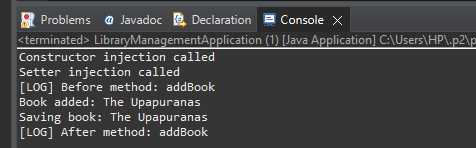
BookService bookService = context.getBean("bookService", BookService.class);

bookService.addBook("The Upapuranas");

}

}

**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.

**Steps:**

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

**Solution**

**application.properties**

spring.application.name=Library\_\_Management

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.show-sql=true

spring.jpa.hibernate.ddl-auto=update

**Book.java**

package com.library.Library\_\_Management.entity;

import jakarta.persistence.\*;

@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String title;

private String author;

private String genre;

@Column(name = "publication\_year") // avoid reserved keyword

private int publicationYear;

public Book() {}

public Book(String title, String author, String genre, int publicationYear) {

this.title = title;

this.author = author;

this.genre = genre;

this.publicationYear = publicationYear;

}

// 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; }

public String getGenre() { return genre; }

public void setGenre(String genre) { this.genre = genre; }

public int getPublicationYear() { return publicationYear; }

public void setPublicationYear(int publicationYear) { this.publicationYear = publicationYear; }

}

**BookRepository.java**

package com.library.Library\_\_Management.repository;

import com.library.Library\_\_Management.entity.Book;

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

public interface BookRepository extends JpaRepository<Book, Long> {

}

**BookController.java**

package com.library.Library\_\_Management.controller;

import com.library.Library\_\_Management.entity.Book;

import com.library.Library\_\_Management.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 a book by ID

@GetMapping("/{id}")

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

return bookRepository.findById(id);

}

// Add a new book

@PostMapping

public Book addBook(@RequestBody Book book) {

return bookRepository.save(book);

}

// Update a book

@PutMapping("/{id}")

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

book.setId(id);

return bookRepository.save(book);

}

// Delete a book

@DeleteMapping("/{id}")

public void deleteBook(@PathVariable Long id) {

bookRepository.deleteById(id);

}

}

**LibraryManagementApplication.java**

package com.library.Library\_\_Management;

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 the REST endpoints**

