Spring Core Maven Hands-On

**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.

Code:

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:p=*"http://www.springframework.org/schema/p"*

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

xmlns:jee=*"http://www.springframework.org/schema/jee"* xmlns:tx=*"http://www.springframework.org/schema/tx"*

xmlns:task=*"http://www.springframework.org/schema/task"*

xsi:schemaLocation=*"http://www.springframework.org/schema/aop* [*http://www.springframework.org/schema/aop/spring-aop-3.2.xsd*](http://www.springframework.org/schema/aop/spring-aop-3.2.xsd)[*http://www.springframework.org/schema/beans*](http://www.springframework.org/schema/beans)[*http://www.springframework.org/schema/beans/spring-beans-3.2.xsd*](http://www.springframework.org/schema/beans/spring-beans-3.2.xsd)[*http://www.springframework.org/schema/context*](http://www.springframework.org/schema/context)[*http://www.springframework.org/schema/context/spring-context-3.2.xsd*](http://www.springframework.org/schema/context/spring-context-3.2.xsd)[*http://www.springframework.org/schema/jee*](http://www.springframework.org/schema/jee)[*http://www.springframework.org/schema/jee/spring-jee-3.2.xsd*](http://www.springframework.org/schema/jee/spring-jee-3.2.xsd)[*http://www.springframework.org/schema/tx*](http://www.springframework.org/schema/tx)[*http://www.springframework.org/schema/tx/spring-tx-3.2.xsd*](http://www.springframework.org/schema/tx/spring-tx-3.2.xsd)[*http://www.springframework.org/schema/task*](http://www.springframework.org/schema/task)[*http://www.springframework.org/schema/task/spring-task-3.2.xsd*](http://www.springframework.org/schema/task/spring-task-3.2.xsd)*"*>

<!-- Define the BookRepository bean -->

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

<!-- No properties to set for this simple example -->

</bean>

<!-- Define the BookService bean -->

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

<!-- Inject the bookRepository bean into the bookService bean's bookRepository property -->

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

</bean>

</beans>

BookService.java

**public** **class** BookService {

**private** BookRepository bookRepository;

// Setter for dependency injection

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

**this**.bookRepository = bookRepository;

}

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

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

bookRepository.save();

}

**public** String getBook(String id) {

System.***out***.println("BookService: Getting book with id " + id);

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

}

}

BookRepository.java

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

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

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

}

**public** String findById(String id) {

System.***out***.println("BookRepository: Finding book with id " + id);

**return** "The Lord of the Rings";

}

}

Main.java

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

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

/\*\*

\* Main class to run the Spring application.

\*/

**public** **class** Main {

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

// Load the Spring configuration file from the classpath

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

// Retrieve the BookService bean from the Spring container

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

// Use the service to perform operations

System.***out***.println("--- Calling addBook() ---");

bookService.addBook();

System.***out***.println("\n--- Calling getBook() ---");

String bookTitle = bookService.getBook("123");

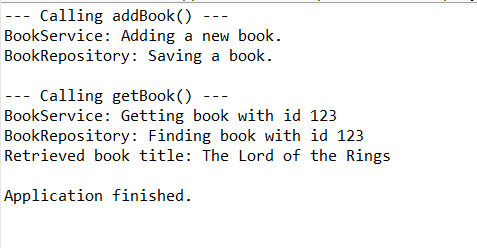
System.***out***.println("book title: " + bookTitle);

System.***out***.println("\nApplication finished.");

}

}

Output:



**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.

Output:

Successfully Updated **applicationContext.xml** to wire **BookRepository** into **BookService and our application has a dependency injection between bookservice and bookrepository, in the above exercise 1 code**

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

**Scenario:**

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

**Code:**

**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"*

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

xsi:schemaLocation=*"*

[*http://www.springframework.org/schema/beans*](http://www.springframework.org/schema/beans)

[*http://www.springframework.org/schema/beans/spring-beans.xsd*](http://www.springframework.org/schema/beans/spring-beans.xsd)

[*http://www.springframework.org/schema/context*](http://www.springframework.org/schema/context)

[*http://www.springframework.org/schema/context/spring-context.xsd*](http://www.springframework.org/schema/context/spring-context.xsd)

[*http://www.springframework.org/schema/aop*](http://www.springframework.org/schema/aop)

[*http://www.springframework.org/schema/aop/spring-aop.xsd*](http://www.springframework.org/schema/aop/spring-aop.xsd)*"*>

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

<aop:aspectj-autoproxy/>

</beans>

**LoggingAspects.java**

**package** com.library.aspect;

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

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

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

**import** org.slf4j.Logger;

**import** org.slf4j.LoggerFactory;

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

@Aspect

@Component

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

**private** **static** **final** Logger ***LOGGER*** = LoggerFactory.*getLogger*(LoggingAspect.**class**);

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

**public** Object logExecutionTime(ProceedingJoinPoint joinPoint) **throws** Throwable {

**long** startTime = System.*currentTimeMillis*();

Object result = joinPoint.proceed();

**long** endTime = System.*currentTimeMillis*();

**long** executionTime = endTime - startTime;

***LOGGER***.info(

joinPoint.getSignature(),

executionTime

);

**return** result;

}

}

**BookService.java**

**package** com.library.service;

**import** org.springframework.stereotype.Service;

**import** java.util.concurrent.TimeUnit;

@Service

**public** **class** BookService {

**public** **void** addBook(String title) {

**try** {

// Simulate a delay

TimeUnit.***MILLISECONDS***.sleep(50);

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

} **catch** (InterruptedException e) {

Thread.*currentThread*().interrupt();

}

}

**public** **void** findBook(String title) {

**try** {

// Simulate another delay

TimeUnit.***MILLISECONDS***.sleep(120);

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

} **catch** (InterruptedException e) {

Thread.*currentThread*().interrupt();

}

}

}

**Main.java**

**package** com.library;

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

**import** org.springframework.boot.SpringApplication;

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

**import** org.springframework.context.ConfigurableApplicationContext;

@SpringBootApplication

**public** **class** Main {

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

// Run the application

ConfigurableApplicationContext context = SpringApplication.*run*(Main.**class**, args);

// Get the BookService bean from the context

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

// Call service methods

System.***out***.println("--- Calling Service Methods ---");

bookService.addBook("The Lord of the Rings");

bookService.findBook("The Hobbit");

System.***out***.println("--- Finished Calling Methods ---");

// Close the context

context.close();

}

}

**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.

Code:

Pom.xml

<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 <https://maven.apache.org/xsd/maven-4.0.0.xsd>">

<modelVersion>4.0.0</modelVersion>

<groupId>LibraryManagement</groupId>

<artifactId>ibraryManagement</artifactId>

<version>0.0.1-SNAPSHOT</version>

<properties>

<maven.compiler.source>1.8</maven.compiler.source>

<maven.compiler.target>1.8</maven.compiler.target>

<spring.version>5.3.37</spring.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

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

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

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

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

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

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

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

<version>3.8.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

</plugins>

</build>

</project>