###Exercise 1: Configuring a Basic Spring Application

### Step 1: Set Up a Spring Project

#### 1.1 Create a Maven project named `LibraryManagement`

1. Open your IDE (e.g., IntelliJ IDEA, Eclipse) and create a new Maven project named `LibraryManagement`.

2. Ensure your `pom.xml` file is set up to include the necessary Spring dependencies.

#### 1.2 Add Spring Core dependencies in the `pom.xml` file

Open the `pom.xml` file and add the following dependencies:

```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 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>5.3.22</version>

</dependency>

</dependencies>

</project>

```

### Step 2: Configure the Application Context

#### 2.1 Create an XML configuration file named `applicationContext.xml`

In the `src/main/resources` directory, create a file named `applicationContext.xml` and define the beans for `BookService` and `BookRepository`.

```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 -->

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

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

</bean>

</beans>

```

### Step 3: Define Service and Repository Classes

#### 3.1 Create `BookService` class

Create the package `com.library.service` and add a class named `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 manageBooks() {

System.out.println("Managing books in the library...");

bookRepository.displayBooks();

}

}

```

#### 3.2 Create `BookRepository` class

Create the package `com.library.repository` and add a class named `BookRepository`.

```java

package com.library.repository;

public class BookRepository {

public void displayBooks() {

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

}

}

```

### Step 4: Run the Application

#### 4.1 Create a main class to load the Spring context and test the configuration

Create a main class `LibraryManagementApp` in the package `com.library`.

```java

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryManagementApp {

public static void main(String[] args) {

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

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

bookService.manageBooks();

}

}

```

### Running the Application

1. Right-click on `LibraryManagementApp` and select `Run`.

2. You should see the following output in the console:

```

Managing books in the library...

Displaying books from the repository...

```

This output confirms that the Spring context has been successfully loaded and the beans have been correctly configured and injected.

### Exercise 2: Implementing Dependency Injection

#### Step 1: Modify the XML Configuration

Update `applicationContext.xml` to wire `BookRepository` into `BookService`.

```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 -->

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

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

</bean>

</beans>

```

#### Step 2: Update the BookService Class

Ensure that the `BookService` class has a setter method for `BookRepository`.

```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 manageBooks() {

System.out.println("Managing books in the library...");

bookRepository.displayBooks();

}

}

```

#### Step 3: Test the Configuration

Run the `LibraryManagementApp` main class to verify the dependency injection.

```java

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryManagementApp {

public static void main(String[] args) {

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

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

bookService.manageBooks();

}

}

```

When you run `LibraryManagementApp`, you should see the following output:

```

Managing books in the library...

Displaying books from the repository...

```

This confirms that the dependency injection is working correctly.

### Exercise 3: Implementing Logging with Spring AOP

#### Step 1: Add Spring AOP Dependency

Update `pom.xml` to include the Spring AOP dependency.

```xml

<dependencies>

<!-- Spring Core Dependency -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.22</version>

</dependency>

<!-- Spring AOP Dependency -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.22</version>

</dependency>

<!-- AspectJ Dependency -->

<dependency>

<groupId>org.aspectj</groupId>

<artifactId>aspectjweaver</artifactId>

<version>1.9.7</version>

</dependency>

</dependencies>

```

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

Create a package `com.library.aspect` and add a class `LoggingAspect` with a method to log execution times.

```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 proceed = joinPoint.proceed();

long executionTime = System.currentTimeMillis() - start;

System.out.println(joinPoint.getSignature() + " executed in " + executionTime + "ms");

return proceed;

}

}

```

#### Step 3: Enable AspectJ Support

Update `applicationContext.xml` to enable AspectJ support and register the aspect.

```xml

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

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

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

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

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

<!-- Define BookRepository bean -->

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

<!-- Define BookService bean -->

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

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

</bean>

<!-- Enable AspectJ support -->

<aop:aspectj-autoproxy/>

<!-- Register LoggingAspect -->

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

</beans>

```

#### Step 4: Test the Aspect

Run the `LibraryManagementApp` main class and observe the console for log messages indicating method execution times.

```java

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryManagementApp {

public static void main(String[] args) {

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

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

bookService.manageBooks();

}

}

```

When you run `LibraryManagementApp`, you should see output similar to the following:

```

Managing books in the library...

Displaying books from the repository...

public void com.library.service.BookService.manageBooks() executed in [execution time]ms

public void com.library.repository.BookRepository.displayBooks() executed in [execution time]ms

```

This confirms that the logging aspect is working correctly and is tracking the method execution times.

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

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

1. Open your IDE (e.g., IntelliJ IDEA, Eclipse) and create a new Maven project named `LibraryManagement`.

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

Update the `pom.xml` file to include dependencies for Spring Context, Spring AOP, and Spring WebMVC.

```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 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 -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.22</version>

</dependency>

<!-- Spring AOP Dependency -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.22</version>

</dependency>

<!-- Spring WebMVC Dependency -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.22</version>

</dependency>

<!-- AspectJ Dependency -->

<dependency>

<groupId>org.aspectj</groupId>

<artifactId>aspectjweaver</artifactId>

<version>1.9.7</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>

```

### Exercise 5: Configuring the Spring IoC Container

#### Step 1: Create Spring Configuration File

Create an XML configuration file named `applicationContext.xml` in the `src/main/resources` directory and define beans for `BookService` and `BookRepository`.

```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 -->

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

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

</bean>

</beans>

```

#### Step 2: Update the BookService Class

Ensure that the `BookService` class has a setter method for `BookRepository`.

```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 manageBooks() {

System.out.println("Managing books in the library...");

bookRepository.displayBooks();

}

}

```

#### Step 3: Run the Application

Create a main class to load the Spring context and test the configuration.

```java

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryManagementApp {

public static void main(String[] args) {

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

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

bookService.manageBooks();

}

}

```

### Running the Application

1. Ensure your Maven project is set up correctly and all dependencies are resolved.

2. Run the `LibraryManagementApp` main class.

You should see the following output in the console:

```

Managing books in the library...

Displaying books from the repository...

```

This confirms that the Spring IoC container is configured correctly, and the dependencies are injected properly.

### Exercise 6: Configuring Beans with Annotations

#### Step 1: Enable Component Scanning

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

```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>

```

#### Step 2: Annotate Classes

Use `@Service` annotation for the `BookService` class and `@Repository` annotation for the `BookRepository` class.

```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;

@Autowired

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void manageBooks() {

System.out.println("Managing books in the library...");

bookRepository.displayBooks();

}

}

```

```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...");

}

}

```

#### Step 3: Test the Configuration

Run the `LibraryManagementApp` main class to verify the annotation-based configuration.

```java

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryManagementApp {

public static void main(String[] args) {

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

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

bookService.manageBooks();

}

}

```

When you run `LibraryManagementApp`, you should see the following output:

```

Managing books in the library...

Displaying books from the repository...

```

This confirms that the annotation-based configuration is working correctly.

### Exercise 7: Implementing Constructor and Setter Injection

#### Step 1: Configure Constructor Injection

Update `BookService` to use constructor injection and update `applicationContext.xml` to configure constructor injection.

##### Update `BookService` Class

```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;

@Autowired

public BookService(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

// Setter method for setter injection

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void manageBooks() {

System.out.println("Managing books in the library...");

bookRepository.displayBooks();

}

}

```

##### Update `applicationContext.xml` for Constructor Injection

```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>

```

#### Step 2: Ensure Setter Method for BookRepository

The `BookService` class already has a setter method for `BookRepository` from the previous step.

#### Step 3: Test the Injection

Run the `LibraryManagementApp` main class to verify both constructor and setter injection.

```java

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryManagementApp {

public static void main(String[] args) {

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

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

bookService.manageBooks();

}

}

```

When you run `LibraryManagementApp`, you should see the following output:

```

Managing books in the library...

Displaying books from the repository...

```

This confirms that both constructor and setter injections are working correctly. The Spring framework will use constructor injection by default, but the setter method is also available if needed.

### Exercise 8: Implementing Basic AOP with Spring

#### Step 1: Define an Aspect

Create a package `com.library.aspect` and add a class `LoggingAspect`.

```java

package com.library.aspect;

import org.aspectj.lang.annotation.After;

import org.aspectj.lang.annotation.Aspect;

import org.aspectj.lang.annotation.Before;

import org.springframework.stereotype.Component;

@Aspect

@Component

public class LoggingAspect {

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

public void logBefore() {

System.out.println("Method execution started...");

}

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

public void logAfter() {

System.out.println("Method execution finished...");

}

}

```

#### Step 2: Create Advice Methods

The advice methods `logBefore` and `logAfter` in the `LoggingAspect` class handle logging before and after method execution.

#### Step 3: Configure the Aspect

Update `applicationContext.xml` to register the aspect and enable AspectJ auto-proxying.

```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/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 component scanning -->

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

<!-- Enable AspectJ auto-proxying -->

<aop:aspectj-autoproxy/>

<!-- Register LoggingAspect -->

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

</beans>

```

#### Step 4: Test the Aspect

Run the `LibraryManagementApp` main class to verify the AOP functionality.

```java

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryManagementApp {

public static void main(String[] args) {

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

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

bookService.manageBooks();

}

}

```

When you run `LibraryManagementApp`, you should see output similar to the following:

```

Method execution started...

Managing books in the library...

Displaying books from the repository...

Method execution finished...

```

This confirms that the AOP functionality is working correctly.

### Exercise 9: Creating a Spring Boot Application

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

Use [Spring Initializr](https://start.spring.io/) to create a new Spring Boot project named `LibraryManagement`.

- \*\*Project:\*\* Maven Project

- \*\*Language:\*\* Java

- \*\*Spring Boot:\*\* 2.7.6

- \*\*Group:\*\* com.library

- \*\*Artifact:\*\* LibraryManagement

- \*\*Name:\*\* LibraryManagement

- \*\*Dependencies:\*\* Spring Web, Spring Data JPA, H2 Database

Download the project and open it in your IDE.

#### Step 2: Add Dependencies

Ensure your `pom.xml` includes the dependencies for Spring Web, Spring Data JPA, and H2 Database.

```xml

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

<dependency>

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

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

<scope>test</scope>

</dependency>

</dependencies>

```

#### Step 3: Create Application Properties

Configure database connection properties in `src/main/resources/application.properties`.

```properties

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

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

```

#### Step 4: Define Entities and Repositories

Create `Book` entity and `BookRepository` interface.

##### Book Entity

```java

package com.library.entity;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String title;

private String 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

```java

package com.library.repository;

import com.library.entity.Book;

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

import org.springframework.stereotype.Repository;

@Repository

public interface BookRepository extends JpaRepository<Book, Long> {

}

```

#### Step 5: Create a REST Controller

Create a `BookController` class to handle CRUD operations.

```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();

}

@GetMapping("/{id}")

public Book getBookById(@PathVariable Long id) {

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

}

@PostMapping

public Book createBook(@RequestBody Book book) {

return bookRepository.save(book);

}

@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

Run the Spring Boot application and test the REST endpoints.

1. In your IDE, run the `LibraryManagementApplication` main class.

2. Use a tool like Postman or curl to test the REST endpoints.

##### Example REST Endpoints

- \*\*Get all books:\*\* `GET http://localhost:8080/books`

- \*\*Get a book by ID:\*\* `GET http://localhost:8080/books/{id}`

- \*\*Create a book:\*\* `POST http://localhost:8080/books` (with JSON body)

- \*\*Update a book:\*\* `PUT http://localhost:8080/books/{id}` (with JSON body)

- \*\*Delete a book:\*\* `DELETE http://localhost:8080/books/{id}`

This confirms that the Spring Boot application is set up correctly and the REST endpoints are functional.