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

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

Create a Maven Project

<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>5.3.15</version>

</dependency>

</dependencies>

</project>

**Step 2: Configure the Application Context**

Create an XML Configuration File

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

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

xsi:schemaLocation="

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

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

<!-- Bean for BookRepository -->

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

<!-- Bean for BookService -->

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

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

</bean>

</beans>

**Step 3: Define Service and Repository Classes**

Create BookService Class

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 performSomeService() {

// Example service method

System.out.println("Performing service operation...");

}

}

Create BookRepository Class

package com.library.repository;

public class BookRepository {

public void performSomeRepositoryOperation() {

// Example repository method

System.out.println("Performing repository operation...");

}

}

**Step 4: Run the Application**

Create a Main Class

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryManagementApplication {

public static void main(String[] args) {

// Load the Spring context

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

// Retrieve the BookService bean

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

// Test the configuration

bookService.performSomeService();

}

}

**Exercise 2: Implementing Dependency Injection**

**Step 1: Modify the XML Configuration**

Update applicationContext.xml to Wire BookRepository into BookService

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

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

xsi:schemaLocation="

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

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

<!-- Bean for BookRepository -->

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

<!-- Bean for BookService -->

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

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

</bean>

</beans>

**Step 2: Update the BookService Class**

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 performSomeService() {

// Example service method

System.out.println("Performing service operation...");

bookRepository.performSomeRepositoryOperation();

}

}

**Step 3: Test the Configuration**

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryManagementApplication {

public static void main(String[] args) {

// Load the Spring context

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

// Retrieve the BookService bean

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

// Test the configuration

bookService.performSomeService();

}

}

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

**Step 1: Add Spring AOP Dependency**

<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://www.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>5.3.15</version>

</dependency>

<!-- Spring AOP Dependency -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.15</version>

</dependency>

<!-- AspectJ Weaver Dependency -->

<dependency>

<groupId>org.aspectj</groupId>

<artifactId>aspectjweaver</artifactId>

<version>1.9.7</version>

</dependency>

</dependencies>

</project>

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

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**

<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

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

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

<!-- Enable AspectJ auto proxying -->

<aop:aspectj-autoproxy/>

<!-- Bean for LoggingAspect -->

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

<!-- Bean for BookRepository -->

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

<!-- Bean for BookService -->

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

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

</bean>

</beans>

**Step 4: Test the Aspect**

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryManagementApplication {

public static void main(String[] args) {

// Load the Spring context

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

// Retrieve the BookService bean

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

// Test the configuration and logging aspect

bookService.performSomeService();

}

}

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

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

Create a new Maven project named LibraryManagement.

**Step 2: Add Spring Dependencies in 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 http://www.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>5.3.15</version>

</dependency>

<!-- Spring AOP Dependency -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.15</version>

</dependency>

<!-- Spring WebMVC Dependency -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.15</version>

</dependency>

<!-- AspectJ Weaver 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>

**Step 3: Configure Maven Plugins**

Configure the Maven Compiler Plugin for Java version 1.8 in the pom.xml file, as shown above. This ensures that the project is compiled using Java 1.8.

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

**Step 1: Create Spring Configuration File**

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

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

xsi:schemaLocation="

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

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

<!-- Bean for BookRepository -->

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

<!-- Bean for BookService -->

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

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

</bean>

</beans>

**Step 2: Update the BookService Class**

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 performSomeService() {

// Example service method

System.out.println("Performing service operation...");

bookRepository.performSomeRepositoryOperation();

}

}

**Step 3: Run the Application**

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryManagementApplication {

public static void main(String[] args) {

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

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

bookService.performSomeService();

}

}

**Exercise 6: Configuring Beans with Annotations**

**Step 1: Enable Component Scanning**

<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 for com.library package -->

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

</beans>

**Step 2: Annotate Classes**

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 performSomeService() {

// Example service method

System.out.println("Performing service operation...");

bookRepository.performSomeRepositoryOperation();

}

}

**Annotate BookRepository Class**

package com.library.repository;

import org.springframework.stereotype.Repository;

@Repository

public class BookRepository {

public void performSomeRepositoryOperation() {

// Example repository method

System.out.println("Performing repository operation...");

}

}

**Step 3: Test the Configuration**

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryManagementApplication {

public static void main(String[] args) {

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

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

bookService.performSomeService();

}

}

**Exercise 7: Implementing Constructor and Setter Injection**

**Step 1: Configure Constructor Injection**

<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 for com.library package -->

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

<!-- Bean for BookRepository -->

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

<!-- Bean for BookService using constructor injection -->

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

<constructor-arg ref="bookRepository" />

</bean>

</beans>

**Step 2: Configure Setter Injection**

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;

injection

@Autowired

public BookService(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

// Setter injection

@Autowired

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void performSomeService() {

// Example service method

System.out.println("Performing service operation...");

bookRepository.performSomeRepositoryOperation();

}

}

**Step 3: Test the Injection**

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryManagementApplication {

public static void main(String[] args) {

// Load the Spring context

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

// Retrieve the BookService bean

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

// Test the configuration

bookService.performSomeService();

}

}

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

**Step 1: Define an Aspect**

**Create LoggingAspect Class**

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("LoggingAspect: Before method execution");

}

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

public void logAfter() {

System.out.println("LoggingAspect: After method execution");

}

}

**Step 2: Create Advice Methods**

The LoggingAspect class above already contains the advice methods for logging before and after method execution.

**Step 3: Configure the Aspect**

<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 for com.library package -->

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

<!-- Enable AspectJ auto proxying -->

<aop:aspectj-autoproxy/>

<!-- Bean for BookRepository -->

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

<!-- Bean for BookService using constructor injection -->

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

<constructor-arg ref="bookRepository" />

</bean>

</beans>

**Step 4: Test the Aspect**

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryManagementApplication {

public static void main(String[] args) {

// Load the Spring context

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

// Retrieve the BookService bean

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

// Test the configuration

bookService.performSomeService();

}

}

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

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

**Use Spring Initializr to Create a New Spring Boot Project**

1. Navigate to [Spring Initializr](https://start.spring.io/).
2. Configure the project with the following settings:
   * Project: Maven Project
   * Language: Java
   * Spring Boot: 2.7.x (or the latest version)
   * Group: com.library
   * Artifact: LibraryManagement
   * Name: LibraryManagement
   * Packaging: Jar
   * Java: 11 or 17 (depending on your preference)
3. Add the following dependencies:
   * Spring Web
   * Spring Data JPA
   * H2 Database
4. Click on the "Generate" button to download the project zip file.
5. Extract the zip file and open the project in your IDE.

**Step 2: Add Dependencies**

<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**

# H2 Database configuration

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

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=password

spring.h2.console.enabled=true

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

spring.jpa.hibernate.ddl-auto=update

**Step 4: Define Entities and Repositories**

**Create Book Entity**

Create a Book class in the com.library.model package:

java

Copy code

package com.library.model;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.AUTO)

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;

}

}

**Create BookRepository Interface**

package com.library.repository;

import com.library.model.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 BookController Class**

Create a BookController class in the com.library.controller package to handle CRUD operations:

java

Copy code

package com.library.controller;

import com.library.model.Book;

import com.library.repository.BookRepository;

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

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

import java.util.Optional;

@RestController

@RequestMapping("/api/books")

public class BookController {

@Autowired

private BookRepository bookRepository;

@GetMapping

public List<Book> getAllBooks() {

return bookRepository.findAll();

}

@GetMapping("/{id}")

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

Optional<Book> book = bookRepository.findById(id);

return book.map(ResponseEntity::ok).orElseGet(() -> ResponseEntity.notFound().build());

}

@PostMapping

public Book createBook(@RequestBody Book book) {

return bookRepository.save(book);

}

@PutMapping("/{id}")

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

Optional<Book> book = bookRepository.findById(id);

if (book.isPresent()) {

Book updatedBook = book.get();

updatedBook.setTitle(bookDetails.getTitle());

updatedBook.setAuthor(bookDetails.getAuthor());

bookRepository.save(updatedBook);

return ResponseEntity.ok(updatedBook);

} else {

return ResponseEntity.notFound().build();

}

}

@DeleteMapping("/{id}")

public ResponseEntity<Void> deleteBook(@PathVariable Long id) {

Optional<Book> book = bookRepository.findById(id);

if (book.isPresent()) {

bookRepository.delete(book.get());

return ResponseEntity.ok().build();

} else {

return ResponseEntity.notFound().build();

}

}

}

**Step 6: Run the Application**

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);

}

}