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

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

1. **Create a Maven Project:** Create a new Maven project named LibraryManagement.

**Add Spring Core dependencies in the pom.xml file:**  
<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>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.24</version>

</dependency>

</dependencies>

</project>

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

**Create applicationContext.xml in src/main/resources:**<?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">

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

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

</beans>

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

**Create BookService class in com.library.service package:**  
package com.library.service;

public class BookService {

// Business methods

}

**Create BookRepository class in com.library.repository package:**  
package com.library.repository;

public class BookRepository {

// Data access methods

}

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

**Create a main class to load the Spring context and test the configuration:**package com.library;

import com.library.service.BookService;

org.springframework.context.ApplicationContext;

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");

System.out.println("BookService Bean: " + bookService);

}

}

### **Exercise 2: Implementing Dependency Injection**

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

**Update applicationContext.xml to wire BookRepository into BookService:**  
<?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">

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

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

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

</bean>

</beans>

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

**Ensure BookService class has a setter method for BookRepository:**package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

// Business methods

}

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

**Run the LibraryManagementApplication main class:**  
package com.library;

import com.library.service.BookService;

org.springframework.context.ApplicationContext;

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");

System.out.println("BookService Bean: " + bookService);

}

}

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

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

**Update pom.xml to include Spring AOP dependency:**<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.24</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.24</version>

</dependency>

</dependencies>

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

**Create LoggingAspect class in com.library.aspect package:**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 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">

<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>

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

**Run the LibraryManagementApplication main class and observe the console for log messages:**  
package com.library;

import com.library.service.BookService;

org.springframework.context.ApplicationContext;

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");

System.out.println("BookService Bean: " + bookService);

}

}

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

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

1. **Create a new Maven project named LibraryManagement.**

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

**Include dependencies for Spring Context, Spring AOP, and Spring WebMVC:**  
<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.24</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.24</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.24</version>

</dependency>

</dependencies>

#### **Step 3: Configure Maven Plugins**

**Configure the Maven Compiler Plugin for Java version 1.8 in pom.xml:**  
<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>

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

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

**Create applicationContext.xml in src/main/resources:**<?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">

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

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

</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;

}

// Business methods

}

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

**Create a main class to load the Spring context and test the configuration:**package com.library;

import com.library.service.BookService;

org.springframework.context.ApplicationContext;

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");

System.out.println("BookService Bean: " + bookService);

}

}

### **Exercise 6: Configuring Beans with Annotations**

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

**Update applicationContext.xml to include component scanning for the com.library package:**<?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">

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

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

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

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

</bean>

</beans>

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

**Use @Service annotation for the BookService class:**  
package com.library.service;

import org.springframework.stereotype.Service;

import com.library.repository.BookRepository;

@Service

public class BookService {

private BookRepository bookRepository;

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

// Business methods

}

**Use @Repository annotation for the BookRepository class:**package com.library.repository;

import org.springframework.stereotype.Repository;

@Repository

public class BookRepository {

// Data access methods

}

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

**Run the LibraryManagementApplication main class:**package com.library;

import com.library.service.BookService;

org.springframework.context.ApplicationContext;

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");

System.out.println("BookService Bean: " + bookService);

}

}

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

#### **Step 1: Configure Constructor Injection**

**Update applicationContext.xml to configure constructor injection for BookService:**<?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">

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

<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;

public class BookService {

private BookRepository bookRepository;

public BookService(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

// Business methods

}

#### **Step 3: Test the Injection**

**Run the LibraryManagementApplication main class:**  
package com.library;

import com.library.service.BookService;

org.springframework.context.ApplicationContext;

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");

System.out.println("BookService Bean: " + bookService);

}

}

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

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

**Create LoggingAspect class in com.library.aspect package:**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 2: Create Advice Methods**

**Define advice methods in LoggingAspect for logging before and after method execution:**  
package com.library.aspect;

import org.aspectj.lang.annotation.Aspect;

import org.aspectj.lang.annotation.Before;

import org.aspectj.lang.annotation.After;

import org.aspectj.lang.JoinPoint;

@Aspect

public class LoggingAspect {

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

public void logBefore(JoinPoint joinPoint) {

System.out.println("Method " + joinPoint.getSignature().getName() + " is about to start.");

}

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

public void logAfter(JoinPoint joinPoint) {

System.out.println("Method " + joinPoint.getSignature().getName() + " has finished.");

}

}

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

**Update applicationContext.xml to register the aspect and enable AspectJ auto-proxying:**<?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">

<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>

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

**Run the LibraryManagementApplication main class to verify the AOP functionality:**  
  
package com.library;

import com.library.service.BookService;

org.springframework.context.ApplicationContext;

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");

System.out.println("BookService Bean: " + bookService);

}

}

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

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

1. **Use Spring Initializr to create a new Spring Boot project named LibraryManagement.**

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

**Include dependencies for Spring Web, Spring Data JPA, and H2 Database in pom.xml:**<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

</dependencies>

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

**Configure database connection properties in application.properties:**  
spring.datasource.url=jdbc:h2:mem:testdb

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=password

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

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

**Create Book entity:**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

}

**Create BookRepository interface:**  
  
package com.library.repository;

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

import com.library.entity.Book;

public interface BookRepository extends JpaRepository<Book, Long> {

}

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

**Create BookController class to handle CRUD operations:**  
package com.library.controller;

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

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

import com.library.entity.Book;

import com.library.repository.BookRepository;

import java.util.List;

@RestController

@RequestMapping("/books")

public class BookController {

@Autowired

private BookRepository bookRepository;

@GetMapping

public List<Book> getAllBooks() {

return bookRepository.findAll();

}

@PostMapping

public Book createBook(@RequestBody Book book) {

return bookRepository.save(book);

}

@GetMapping("/{id}")

public Book getBookById(@PathVariable Long id) {

return bookRepository.findById(id).orElseThrow(() -> new RuntimeException("Book not found"));

}

@PutMapping("/{id}")

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

Book book = bookRepository.findById(id).orElseThrow(() -> new RuntimeException("Book not found"));

book.setTitle(bookDetails.getTitle());

book.setAuthor(bookDetails.getAuthor());

return bookRepository.save(book);

}

@DeleteMapping("/{id}")

public void deleteBook(@PathVariable Long id) {

Book book = bookRepository.findById(id).orElseThrow(() -> new RuntimeException("Book not found"));

bookRepository.delete(book);

}

}