**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:**
   * Create a Maven project named **LibraryManagement**.
   * Add Spring Core dependencies in the **pom.xml** file.

Ans)

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-core</artifactId>

<version>5.3.20</version

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.20</version

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.36</version>

</dependency>

<dependency>

<groupId>ch.qos.logback</groupId>

<artifactId>logback-classic</artifactId>

<version>1.2.11</version>

</dependency>

</dependencies>

1. **Configure the Application Context:**
   * Create an XML configuration file named **applicationContext.xml** in the **src/main/resources** directory.

* + Define beans for **BookService** and **BookRepository** in the XML file.

Ans)

<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>

1. **Define Service and Repository Classes:**
   * Create a package **com.library.service** and add a class **BookService**.

Ans)

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public String getBookDetails(int id) {

return bookRepository.findBookById(id);

}

}

* + Create a package **com.library.repository** and add a class **BookRepository**

Ans)

package com.library.repository;

public class BookRepository {

public String findBookById(int id) {

return "Book with ID: " + id;

}

}

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;n

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public String getBookDetails(int id) {

return bookRepository.findBookById(id);

}

}

1. **Run the Application:**
   * Create a main class to load the Spring context and test the configuration.

package com.library;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

public class Main {

public static void main(String[] args) {

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

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

String bookDetails = bookService.getBookDetails(1);

System.out.println(bookDetails); // Output: Book with ID: 1

}

}

Ans)

**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:**
   * Update **applicationContext.xml** to wire **BookRepository** into **BookService**.

Ans)

<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>

1. **Update the BookService Class:**
   * Ensure that **BookService** class has a setter method for **BookRepository**.

Ans)

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public String getBookDetails(int id) {

return bookRepository.findBookById(id);

}

}

1. **Test the Configuration:**
   * Run the **LibraryManagementApplication** main class to verify the dependency injection.

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public String getBookDetails(int id) {

return bookRepository.findBookById(id);

}

}

Ans)

**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:**
   * Update **pom.xml** to include Spring AOP dependency.

Ans)

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.20</version>

</dependency>

<dependency>

<groupId>org.aspectj</groupId>

<artifactId>aspectjweaver</artifactId>

<version>1.9.9</version>

</dependency>

</dependencies>

1. **Create an Aspect for Logging:**
   * Create a package **com.library.aspect** and add a class **LoggingAspect** with a method to log execution times.

Ans)

package com.library.aspect;

import org.aspectj.lang.JoinPoint;

import org.aspectj.lang.annotation.Around;

import org.aspectj.lang.annotation.Aspect;

import org.aspectj.lang.annotation.Pointcut;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.stereotype.Component;

public class LoggingAspect {

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

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

public void serviceMethods() {}

@Around("serviceMethods()")

public Object logExecutionTime(JoinPoint joinPoint) throws Throwable {

long start = System.currentTimeMillis();

Object proceed = joinPoint.proceed();

long executionTime = System.currentTimeMillis() - start;

logger.info(joinPoint.getSignature() + " executed in " + executionTime + "ms");

return proceed;

}

}

1. **Enable AspectJ Support:**
   * Update **applicationContext.xml** to enable **AspectJ** support and register the aspect.

Ans)

<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">

<aop:aspectj-autoproxy />

<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"/>

</beans>

1. **Test the Aspect:**
   * Run the **LibraryManagementApplication** main class and observe the console for log messages indicating method execution times.

package com.library;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

public class Main {

public static void main(String[] args) {

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

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

String bookDetails = bookService.getBookDetails(1);

System.out.println(bookDetails); // Output: Book with ID: 1

}

}

Ans)

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

Ans)

mvn archetype:generate -DgroupId=com.library -DartifactId=LibraryManagement -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

cd LibraryManagement

1. **Add Spring Dependencies in pom.xml:**
   * Include dependencies for Spring Context, Spring AOP, and Spring WebMVC.

Ans)

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.20</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.20</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.20</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.36</version>

</dependency>

<dependency>

<groupId>ch.qos.logback</groupId>

<artifactId>logback-classic</artifactId>

<version>1.2.11</version>

</dependency>

</dependencies>

1. **Configure Maven Plugins:**
   * Configure the Maven Compiler Plugin for Java version 1.8 in the pom.xml file.

Ans)

<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**

**Scenario:**

The library management application requires a central configuration for beans and dependencies.

**Steps:**

1. **Create Spring Configuration File:**
   * Create an XML configuration file named **applicationContext.xml** in the **src/main/resources** directory.

* + Define beans for **BookService** and **BookRepository** in the XML file.

Ans)

<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>

1. **Update the BookService Class:**
   * Ensure that the **BookService** class has a setter method for **BookRepository**.

Ans)

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public String getBookDetails(int id) {

return bookRepository.findBookById(id);

}

}

1. **Run the Application:**
   * Create a main class to load the Spring context and test the configuration.

Ans)

package com.library;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

public class Main {

public static void main(String[] args) {

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

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

String bookDetails = bookService.getBookDetails(1);

System.out.println(bookDetails); // Output: Book with ID: 1

}

}

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

Ans)

<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">

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

<aop:aspectj-autoproxy />

</beans>

1. **Annotate Classes:**
   * Use **@Service** annotation for the **BookService** class.

Ans)

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;

}

public String getBookDetails(int id) {

return bookRepository.findBookById(id);

}

}

* + Use **@Repository** annotation for the **BookRepository** class.

package com.library.repository;

import org.springframework.stereotype.Repository;

@Repository

public class BookRepository {

public String findBookById(int id) {

return "Book with ID: " + id;

}

}

Ans)

1. **Test the Configuration:**
   * Run the **LibraryManagementApplication** main class to verify the annotation-based configuration.

Ans)

package com.library;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

public class Main {

public static void main(String[] args) {

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

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

String bookDetails = bookService.getBookDetails(1);

System.out.println(bookDetails); // Output: Book with ID: 1

}

}

**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:**
   * Update applicationContext.**xml** to configure constructor injection for **BookService**.

Ans)

<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"

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/aop

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

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

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

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

<aop:aspectj-autoproxy />

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

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

<constructor-arg ref="bookRepository"/>

</bean>

</beans>

1. **Configure Setter Injection:**
   * Ensure that the **BookService** class has a setter method for **BookRepository** and configure it in **applicationContext.xml**.

Ans)

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;

}

@Autowired

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public String getBookDetails(int id) {

return bookRepository.findBookById(id);

}

}

1. **Test the Injection:**
   * Run the **LibraryManagementApplication** main class to verify both constructor and setter injection.

Ans)

package com.library;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

public class Main {

public static void main(String[] args) {

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

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

String bookDetails = bookService.getBookDetails(1);

System.out.println(bookDetails); // Output: Book with ID: 1

}

}

**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:**
   * Create a package **com.library.aspect** and add a class **LoggingAspect**.

Ans)

package com.library.aspect;

import org.aspectj.lang.JoinPoint;

import org.aspectj.lang.annotation.After;

import org.aspectj.lang.annotation.AfterReturning;

import org.aspectj.lang.annotation.Before;

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);

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

public void logBefore(JoinPoint joinPoint) {

logger.info("Method {} is about to be called", joinPoint.getSignature());

}

@AfterReturning(pointcut = "execution(\* com.library.service..\*(..))", returning = "result")

public void logAfterReturning(JoinPoint joinPoint, Object result) {

logger.info("Method {} returned with value {}", joinPoint.getSignature(), result);

}

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

public void logAfter(JoinPoint joinPoint) {

logger.info("Method {} has been executed", joinPoint.getSignature());

}

}

1. **Create Advice Methods:**
   * Define advice methods in **LoggingAspect** for logging before and after method execution.

Ans)

package com.library.aspect;

import org.aspectj.lang.JoinPoint;

import org.aspectj.lang.annotation.After;

import org.aspectj.lang.annotation.AfterReturning;

import org.aspectj.lang.annotation.Before;

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);

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

public void logBefore(JoinPoint joinPoint) {

logger.info("Method {} is about to be called", joinPoint.getSignature());

}

@AfterReturning(pointcut = "execution(\* com.library.service..\*(..))", returning = "result")

public void logAfterReturning(JoinPoint joinPoint, Object result) {

logger.info("Method {} returned with value {}", joinPoint.getSignature(), result);

}

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

public void logAfter(JoinPoint joinPoint) {

logger.info("Method {} has been executed", joinPoint.getSignature());

}

}

1. **Configure the Aspect:**
   * Update **applicationContext.xml** to register the aspect and enable **AspectJ** auto-proxying.

Ans)

<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"

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/aop

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

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

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

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

<aop:aspectj-autoproxy />

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

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

<constructor-arg ref="bookRepository"/>

</bean>

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

</beans>

1. **Test the Aspect:**
   * Run the **LibraryManagementApplication** main class to verify the AOP functionality.

Ans)

package com.library;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

public class Main {

public static void main(String[] args) {

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

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

String bookDetails = bookService.getBookDetails(1);

System.out.println(bookDetails); // Output: Book with ID: 1

}

}

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

Ans) Setup successfully implemented.

1. **Add Dependencies:**
   * Include dependencies for **Spring Web, Spring Data JPA, and H2 Database**.

Ans)

<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>

<dependency>

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

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

<scope>test</scope>

</dependency>

</dependencies>

1. **Create Application Properties:**
   * Configure database connection properties in **application.properties**.

Ans)

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

spring.h2.console.enabled=true

spring.h2.console.path=/h2-console

1. **Define Entities and Repositories:**
   * Create **Book** entity and **BookRepository** interface.

Ans)

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;

private String isbn;

}

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> {

}

1. **Create a REST Controller:**
   * Create a **BookController** class to handle CRUD operations.

Ans)

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;

import java.util.Optional;

@RestController

@RequestMapping("/books")

public class BookController {

@Autowired

private BookRepository bookRepository;

@GetMapping

public List<Book> getAllBooks() {

return bookRepository.findAll();

}

@GetMapping("/{id}")

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

return bookRepository.findById(id);

}

@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).orElseThrow();

book.setTitle(bookDetails.getTitle());

book.setAuthor(bookDetails.getAuthor());

book.setIsbn(bookDetails.getIsbn());

return bookRepository.save(book);

}

@DeleteMapping("/{id}")

public void deleteBook(@PathVariable Long id) {

bookRepository.deleteById(id);

}

}

1. **Run the Application:**
   * Run the Spring Boot application and test the REST endpoints.

Ans)

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);

}

}