**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.

**<dependency>**

**<groupId>org.springframework</groupId>**

**<artifactId>spring-context</artifactId>**

**<version>5.3.8</version>**

**</dependency>**

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.

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

/**/ BookRepository.java**

**public class BookRepository {**

**public void saveBook() {**

**System.out.println("Book saved.");**

**}**

**}**

**// BookService.java**

**public class BookService {**

**private BookRepository bookRepository;**

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

**this.bookRepository = bookRepository;**

**}**

**public void addBook() {**

**bookRepository.saveBook();**

**}**

**}**

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

**public class LibraryManagementApplication {**

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

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

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

**bookService.addBook();**

**}**

**}**

**Output-**

Book saved.

**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**.

**<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 with setter-based DI -->**

**<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**.

**package com.library.service;**

**import com.library.repository.BookRepository;**

**public class BookService {**

**private BookRepository bookRepository;**

**// Setter for Dependency Injection**

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

**this.bookRepository = bookRepository;**

**}**

**public void addBook() {**

**bookRepository.saveBook();**

**}**

**}**

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

**import org.springframework.context.ApplicationContext;**

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

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

**public class LibraryManagementApplication {**

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

**// Load the Spring Application Context**

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

**// Get the BookService bean**

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

**// Call the method to test DI**

**bookService.addBook();**

**}**

**}**

**Output-**

Book saved to repository.

**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.

**<dependencies>**

**<!-- Spring Context -->**

**<dependency>**

**<groupId>org.springframework</groupId>**

**<artifactId>spring-context</artifactId>**

**<version>5.3.8</version>**

**</dependency>**

**<!-- Spring AOP -->**

**<dependency>**

**<groupId>org.springframework</groupId>**

**<artifactId>spring-aop</artifactId>**

**<version>5.3.8</version>**

**</dependency**

**<!-- AspectJ Weaver (Required for AOP) -->**

**<dependency>**

**<groupId>org.aspectj</groupId>**

**<artifactId>aspectjweaver</artifactId>**

**<version>1.9.6</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.

**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 result = joinPoint.proceed(); // Call the actual method**

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

**System.out.println(joinPoint.getSignature() + " executed in " + (end - start) + "ms");**

**return result;**

**}**

**}**

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

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

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

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

**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**

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

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

**<!-- Component scanning (optional if using annotations like @Service) -->**

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

**<!-- AOP support -->**

**<aop:aspectj-autoproxy/>**

**<!-- Register the logging aspect -->**

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

**<!-- Define Repository and Service beans -->**

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

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

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

**</bean>**

**</beans>**

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

**import org.springframework.context.ApplicationContext;**

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

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

**public class LibraryManagementApplication {**

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

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

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

**bookService.addBook();**

**}**

**}**

**Output-**

Book saved to repository.

void com.library.service.BookService.addBook() executed in 6ms

**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**.

**mvn archetype:generate -DgroupId=com.example.library \**

**-DartifactId=LibraryManagement \**

**-DarchetypeArtifactId=maven-archetype-quickstart \**

**-DinteractiveMode=false**

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

**<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.example.library</groupId>**

**<artifactId>LibraryManagement</artifactId>**

**<version>1.0-SNAPSHOT</version>**

**<dependencies>**

**<!-- Spring Context -->**

**<dependency>**

**<groupId>org.springframework</groupId>**

**<artifactId>spring-context</artifactId>**

**<version>5.3.31</version>**

**</dependency>**

**<!-- Spring AOP -->**

**<dependency>**

**<groupId>org.springframework</groupId>**

**<artifactId>spring-aop</artifactId>**

**<version>5.3.31</version>**

**</dependency>**

**<!-- Spring Web MVC -->**

**<dependency>**

**<groupId>org.springframework</groupId>**

**<artifactId>spring-webmvc</artifactId>**

**<version>5.3.31</version>**

**</dependency>**

**<!-- Servlet API (required for Spring WebMVC) -->**

**<dependency>**

**<groupId>javax.servlet</groupId>**

**<artifactId>javax.servlet-api</artifactId>**

**<version>4.0.1</version>**

**<scope>provided</scope>**

**</dependency>**

**</dependencies>**

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

**<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**

**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.

**<?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">**

**<!-- BookRepository Bean -->**

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

**<!-- BookService Bean with setter injection -->**

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

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

**</bean>**

**</beans>**

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

**<?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.example.library.BookRepository" />**

**<!-- Define BookService Bean and inject BookRepository -->**

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

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

**</bean>**

**</beans>**

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

**package com.example.library;**

**public class BookService {**

**private BookRepository bookRepository**

**// Setter for dependency injection**

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

**this.bookRepository = bookRepository;**

**}**

**public void displayBooks() {**

**bookRepository.listBooks();**

**}**

**}**

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

**package com.example.library;**

**import org.springframework.context.ApplicationContext;**

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

**public class LibraryApp {**

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

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

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

**bookService.displayBooks(); // Should print the message from BookRepository**

**}**

**}**

**Output-**

Listing all books from the repository...

**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.

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

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

**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/context**

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

**<!-- Enable component scanning for the com.library package -->**

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

**</beans>**

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

**package com.library.repository;**

**import org.springframework.stereotype.Repository;**

**@Repository**

**public class BookRepository {**

**public void saveBook() {**

**System.out.println("Book saved to repository.");**

**}**

**}**

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

**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 {**

**@Autowired // Field injection**

**private BookRepository bookRepository;**

**public void addBook() {**

**bookRepository.saveBook();**

**}**

**}**

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

**import org.springframework.context.ApplicationContext;**

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

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

**public class LibraryManagementApplication {**

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

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

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

**bookService.addBook();**

**}**

**}**

**Output-**

Book saved to repository.

**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**.

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

**<constructor-arg ref="bookRepository"/>**

**</bean>**

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

**public class BookService {**

**private BookRepository bookRepository;**

**public BookService(BookRepository bookRepository) {**

**this.bookRepository = bookRepository;**

**}**

**}**

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

**Output-**

Book saved.

**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**.

**package com.library.aspect;**

**import org.aspectj.lang.JoinPoint;**

**public class LoggingAspect {**

**// Before method execution**

**public void logBefore(JoinPoint joinPoint) {**

**System.out.println("🔍 Before executing method: " + joinPoint.getSignature().getName());**

**}**

**// After method execution**

**public void logAfter(JoinPoint joinPoint) {**

**System.out.println("✅ After executing method: " + joinPoint.getSignature().getName());**

**}**

**}**

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

**package com.library.aspect;**

**import org.aspectj.lang.JoinPoint;**

**public class LoggingAspect {**

**// Advice method to run BEFORE target method**

**public void logBefore(JoinPoint joinPoint) {**

**System.out.println(" [BEFORE] Executing: " + joinPoint.getSignature().getName());**

**}**

**// Advice method to run AFTER target method**

**public void logAfter(JoinPoint joinPoint) {**

**System.out.println("[AFTER] Completed: " + joinPoint.getSignature().getName());**

**}**

**}**

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

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 AOP auto-proxying -->

<aop:aspectj-autoproxy/>

<!-- Define the BookRepository bean -->

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

<!-- Define the BookService bean -->

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

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

</bean>

<!-- Define the Logging Aspect -->

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

<!-- Register the Aspect -->

<aop:config>

<aop:aspect ref="loggingAspect">

<aop:pointcut id="allServiceMethods" expression="execution(\* com.library.service.\*.\*(..))"/>

<aop:before pointcut-ref="allServiceMethods" method="logBefore"/>

<aop:after pointcut-ref="allServiceMethods" method="logAfter"/>

</aop:aspect>

</aop:config>

</beans>

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

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

**service.displayBooks();**

**}**

**}**

**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**.

**Use Spring Initializr with dependencies: Spring Web, Spring Data JPA, H2 DB.**

1. **Add Dependencies:**
   * Include dependencies for **Spring Web, Spring Data JPA, and H2 Database**.
2. **Create Application Properties:**
   * Configure database connection properties in **application.properties**.

**spring.datasource.url=jdbc:h2:mem:librarydb**

**spring.jpa.hibernate.ddl-auto=create**

**spring.h2.console.enabled=true**

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

**@Entity**

**public class Book {**

**@Id**

**@GeneratedValue**

**private Long id;**

**private String title;**

**}**

**public interface BookRepository extends JpaRepository<Book, Long> {}**

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

**@RestController**

**@RequestMapping("/books")**

**public class BookController {**

**@Autowired private BookRepository repo;**

**@PostMapping public Book save(@RequestBody Book book) { return repo.save(book); }**

**@GetMapping public List<Book> getAll() { return repo.findAll(); }**

**}**

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

**Output-**

**GET http://localhost:8080/books**