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

**Set Up a Spring Project:**

* Create a Maven project named **LibraryManagement**.
* In pom.xml, add the Spring Core dependency:

xml

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.30</version> *<!-- Use the latest stable version -->*

</dependency>

**Configure the Application Context:**

* Create src/main/resources/applicationContext.xml.
* Define beans:

xml

<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"/>

</beans>

**Define Service and Repository Classes:**

* com.library.service.BookService:

java

**package** com.library.service;

**public** **class** BookService {

*// Business logic methods*

}

* com.library.repository.BookRepository:

java

**package** com.library.repository;

**public** **class** BookRepository {

*// Data access methods*

}

**Run the Application:**

* Create a main class to load the context:

java

**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");

*// Test bookService methods*

}

}

**Exercise 2: Implementing Dependency Injection**

**Modify the XML Configuration:**

* Wire BookRepository into BookService:

xml

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

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

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

</bean>

**Update the BookService Class:**

* Add a setter for BookRepository:

java

**public** **class** BookService {

**private** BookRepository bookRepository;

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

**this**.bookRepository = bookRepository;

}

}

**Test the Configuration:**

* Run LibraryManagementApplication to verify DI works (no exceptions, beans wired).

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

**Add Spring AOP Dependency:**

* In pom.xml:

xml

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.30</version>

</dependency>

**Create an Aspect for Logging:**

* com.library.aspect.LoggingAspect:

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;

}

}

**Enable AspectJ Support:**

* In applicationContext.xml:

xml

<beans ...>

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

<aop:aspectj-autoproxy xmlns:aop="http://www.springframework.org/schema/aop"/>

</beans>

**Test the Aspect:**

* Run the main class and observe logs for execution times.

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

**Create a New Maven Project:**

* Project name: **LibraryManagement**.

**Add Spring Dependencies in pom.xml:**

xml

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.30</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.30</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.30</version>

</dependency>

**Configure Maven Plugins:**

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 9: Creating a Spring Boot Application**

**Create a Spring Boot Project:**

* Use Spring Initializr, name: **LibraryManagement**.

**Add Dependencies:**

* Spring Web, Spring Data JPA, H2 Database.

**Create Application Properties:**

* In src/main/resources/application.properties:

text

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

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=

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

**Define Entities and Repositories:**

* Book entity:

java

@Entity

**public** **class** Book {

@Id @GeneratedValue

**private** Long id;

**private** String title;

*// getters/setters*

}

* BookRepository:

java

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

**Create a REST Controller:**

* BookController:

java

@RestController

@RequestMapping("/books")

**public** **class** BookController {

@Autowired

**private** BookRepository bookRepository;

@GetMapping

**public** List<Book> getAllBooks() {

**return** bookRepository.findAll();

}

*// Additional CRUD endpoints*

}

**Run the Application:**

* Run LibraryManagementApplication and test endpoints

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

**Define an Aspect:**

* com.library.aspect.LoggingAspect as in Exercise 3.

**Create Advice Methods:**

* Add @Before and @After advice in LoggingAspect:

java

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

**public** **void** beforeMethod() {

System.out.println("Before method execution");

}

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

**public** **void** afterMethod() {

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

}

**Configure the Aspect:**

* Register aspect and enable auto-proxying in applicationContext.xml.

**Test the Aspect:**

* Run the main class and observe before/after logs.

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

**Configure Constructor Injection:**

* In applicationContext.xml:

xml

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

<constructor-arg ref="bookRepository"/>

</bean>

**Configure Setter Injection:**

* Ensure BookService has a setter for BookRepository and configure <property name="bookRepository" ref="bookRepository"/> if needed.

**Test the Injection:**

* Run the main class to verify both injections work

**Exercise 6: Configuring Beans with Annotations**

**Enable Component Scanning:**

* In applicationContext.xml:

xml

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

**Annotate Classes:**

* In BookService:

java

@Service

**public** **class** BookService { ... }

* In BookRepository:

java

@Repository

**public** **class** BookRepository { ... }

**Test the Configuration:**

* Run the main class, ensure beans are auto-wired.