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

1. **Set Up a Spring Project**
   * Create a Maven project named LibraryManagement.
   * Add the following dependencies in your pom.xml:

xml

Copy code

<dependencies>

<!-- Spring Core -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.9</version>

</dependency>

</dependencies>

1. **Configure the Application Context**

Create an applicationContext.xml file in the src/main/resources directory:

xml

Copy code

<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 beans -->

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

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

</bean>

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

</beans>

1. **Define Service and Repository Classes**

Create BookService and BookRepository classes.

java

Copy code

package com.library.service;

public class BookService {

private com.library.repository.BookRepository bookRepository;

public void setBookRepository(com.library.repository.BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void getBookInfo() {

bookRepository.findBook();

}

}

package com.library.repository;

public class BookRepository {

public void findBook() {

System.out.println("Fetching book information...");

}

}

1. **Run the Application**

Create a main class to load the Spring context:

java

Copy code

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");

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

bookService.getBookInfo();

}

}

**Exercise 2: Implementing Dependency Injection**

1. **Modify the XML Configuration**

Ensure the applicationContext.xml has a setter injection for BookService:

xml

Copy code

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

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

</bean>

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

1. **Update the BookService Class**

Ensure the BookService class has a setter method for BookRepository (already shown in Exercise 1).

1. **Test the Configuration**

Run the LibraryManagementApplication class to verify the dependency injection.

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

1. **Add Spring AOP Dependency**

Add Spring AOP dependency in pom.xml:

xml

Copy code

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.9</version>

</dependency>

1. **Create an Aspect for Logging**

Create a class LoggingAspect:

java

Copy code

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;

}

}

1. **Enable AspectJ Support**

Update applicationContext.xml to enable AspectJ support:

xml

Copy code

<aop:aspectj-autoproxy/>

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

1. **Test the Aspect**

Run the LibraryManagementApplication class and observe the console for log messages.

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

1. **Create a New Maven Project**
   * Named LibraryManagement.
2. **Add Spring Dependencies in pom.xml**:

xml

Copy code

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.9</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.9</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.9</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>

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

Repeat Exercise 1 and ensure that applicationContext.xml is centrally configured for all beans and dependencies.

**Exercise 6: Configuring Beans with Annotations**

1. **Enable Component Scanning**

Update applicationContext.xml:

xml

Copy code

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

1. **Annotate Classes**

java

Copy code

package com.library.service;

import org.springframework.stereotype.Service;

@Service

public class BookService {

// Same as before

}

package com.library.repository;

import org.springframework.stereotype.Repository;

@Repository

public class BookRepository {

// Same as before

}

1. **Test the Configuration**

Run the LibraryManagementApplication to verify annotation-based configuration.

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

1. **Configure Constructor Injection**

Update applicationContext.xml:

xml

Copy code

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

<constructor-arg ref="bookRepository"/>

</bean>

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

Update BookService class:

java

Copy code

public class BookService {

private final BookRepository bookRepository;

public BookService(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

}

1. **Test the Injection**

Run the LibraryManagementApplication class to verify both constructor and setter injection.

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

Repeat Exercise 3 for implementing basic AOP with Spring.

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

1. **Create a Spring Boot Project**
   * Use Spring Initializr to create a project named LibraryManagement.
2. **Add Dependencies**

Add the following dependencies in pom.xml:

xml

Copy code

<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>

1. **Create Application Properties**

Create application.properties in src/main/resources:

properties

Copy code

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

1. **Define Entities and Repositories**

java

Copy code

package com.library.entity;

import javax.persistence.Entity;

import javax.persistence.Id;

@Entity

public class Book {

@Id

private Long id;

private String title;

private String author;

// Getters and setters

}

package com.library.repository;

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

import com.library.entity.Book;

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

1. **Create a REST Controller**

java

Copy code

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);

}

@PutMapping("/{id}")

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

book.setId(id);

return bookRepository.save(book);

}

@DeleteMapping("/{id}")

public void deleteBook(@PathVariable Long id) {

bookRepository.deleteById(id);

}

}

1. **Run the Spring Boot Application**

Create a LibraryManagementApplication class:

java

Copy code

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);

}

}