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

Your company is developing a web application for managing a library. You need to use the Spring Framework to handle the backend operations.

1. **Set up a Maven project named LibraryManagement**

Add this to pom.xml:

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.22</version>

</dependency>

1. **Create applicationContext.xml in src/main/resources**

<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. **Create Service and Repository Classes**

// 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 repo) {

this.bookRepository = repo;

}

public void addBook() {

bookRepository.saveBook();

}

}

1. **Run with Main Class**

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

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

service.addBook();

**Exercise 2: Implementing Dependency Injection**

In the library management application, manage the dependencies between BookService and BookRepository using Spring's IoC and DI.

1. **In applicationContext.xml**  
   DI already configured using <property name="bookRepository" ref="bookRepository"/>.
2. **In BookService.java**

public void setBookRepository(BookRepository repo) {

this.bookRepository = repo;

}

1. **Run the Main Class**  
   Outputs:

Book saved.

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

Add logging to track method execution times using Spring AOP.

1. **Add to pom.xml:**

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.22</version>

</dependency>

<dependency>

<groupId>org.aspectj</groupId>

<artifactId>aspectjweaver</artifactId>

<version>1.9.9.1</version>

</dependency>

1. **Create LoggingAspect.java**

@Aspect

public class LoggingAspect {

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

public Object log(ProceedingJoinPoint joinPoint) throws Throwable {

long start = System.currentTimeMillis();

Object result = joinPoint.proceed();

long end = System.currentTimeMillis();

System.out.println(joinPoint.getSignature() + " took " + (end - start) + "ms");

return result;

}

}

1. **Enable AspectJ in XML**

<aop:aspectj-autoproxy/>

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

1. **Run Main Class**

Observe execution time logs in console.

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

Set up a new Maven project and add Spring dependencies.

1. **Create Maven project named LibraryManagement**
2. **Add to pom.xml:**

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.22</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.22</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.22</version>

</dependency>

1. **Configure Maven Compiler Plugin**

<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**

Configure a central XML file for all beans and dependencies.

1. **applicationContext.xml** already created in src/main/resources.
2. **BookService.java**  
   Must have:

public void setBookRepository(BookRepository repo) {

this.bookRepository = repo;

}

1. **Test via Main Class**  
   Output confirms IoC:

Book saved.

**Exercise 6: Configuring Beans with Annotations**

Simplify bean configuration using annotations.

1. **Enable component scan in XML**

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

<context:annotation-config/>

1. **Annotate classes**

@Service

public class BookService {

@Autowired

private BookRepository bookRepository;

public void addBook() {

bookRepository.saveBook();

}

}

@Repository

public class BookRepository {

public void saveBook() {

System.out.println("Saved with annotations.");

}

}

1. **Run Main Class**

Output:

Saved with annotations.

**Exercise 7: Constructor and Setter Injection**

Implement both constructor and setter-based injection.

1. **Constructor Injection in XML**

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

<constructor-arg ref="bookRepository"/>

</bean>

1. **BookService.java**

public BookService(BookRepository repo) {

this.bookRepository = repo;

}

public void setBookRepository(BookRepository repo) {

this.bookRepository = repo;

}

1. **Run Main Class**

Output:

Book saved.

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

Use AOP for logging and separation of concerns.

1. **Define Aspect: LoggingAspect.java**

@Aspect

public class LoggingAspect {

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

public void before() {

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

}

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

public void after() {

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

}

}

1. **Configure in XML**

<aop:aspectj-autoproxy/>

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

1. **Run Main Class**

Output:

Before method

Book saved

After method

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

Create a Spring Boot app for the library system.

1. **Use Spring Initializr**

Project name: LibraryManagement  
Dependencies: Web, JPA, H2

1. **application.properties**

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

spring.datasource.driverClassName=org.h2.Driver

spring.jpa.show-sql=true

spring.h2.console.enabled=true

1. **Book.java**

@Entity

public class Book {

@Id @GeneratedValue

private Long id;

private String title;

}

1. **BookRepository.java**

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

1. **BookController.java**

@RestController

@RequestMapping("/books")

public class BookController {

@Autowired

private BookRepository repo;

@PostMapping

public Book add(@RequestBody Book book) {

return repo.save(book);

}

@GetMapping

public List<Book> all() {

return repo.findAll();

}

}

1. **Main Class**

@SpringBootApplication

public class LibraryManagementApplication {

public static void main(String[] args) {

SpringApplication.run(LibraryManagementApplication.class, args);

}

}

1. **Run and Test**  
   Use Postman or browser at:  
   http://localhost:8080/books