**Spring Framework Library Management Exercises**

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

**Steps:**

1. **Set Up a Spring Project (LibraryManagement)**
   * Create a Maven project.
   * Add Spring Core dependency in pom.xml:

<dependencies>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>5.3.36</version>  
 </dependency>  
</dependencies>

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. **Service and Repository Classes:**

* BookRepository.java

package com.library.repository;  
public class BookRepository {  
 public void saveBook() {  
 System.out.println("Book saved.");  
 }  
}

* BookService.java

package com.library.service;  
import com.library.repository.BookRepository;  
  
public class BookService {  
 private BookRepository bookRepository;  
 public void setBookRepository(BookRepository bookRepository) {  
 this.bookRepository = bookRepository;  
 }  
 public void addBook() {  
 bookRepository.saveBook();  
 }  
}

1. **Main Class:**

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

### **Exercise 2: Implementing Dependency Injection**

* Already configured with setter injection in applicationContext.xml.
* BookService uses setBookRepository(BookRepository) method.
* Run the main class to verify Spring DI.

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

**Steps:**

1. Add Spring AOP dependency in pom.xml:

<dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-aop</artifactId>  
 <version>5.3.36</version>  
</dependency>

1. Create Aspect class:

package com.library.aspect;  
import org.aspectj.lang.ProceedingJoinPoint;  
import org.aspectj.lang.annotation.\*;  
  
@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 end = System.currentTimeMillis();  
 System.out.println("Execution time: " + (end - start) + "ms");  
 return proceed;  
 }  
}

1. Update applicationContext.xml:

<aop:aspectj-autoproxy/>  
<bean class="com.library.aspect.LoggingAspect"/>

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

* Add Spring Context, AOP, WebMVC dependencies.

<dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-webmvc</artifactId>  
 <version>5.3.36</version>  
</dependency>

* 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: Spring IoC Container**

* Already completed in Exercise 1 with applicationContext.xml and setter method.

### **Exercise 6: Annotations for Bean Configuration**

1. Update applicationContext.xml:

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

1. Use annotations:

@Service  
public class BookService { ... }  
  
@Repository  
public class BookRepository { ... }

1. Remove XML bean definitions for these classes.

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

* Constructor Injection XML:

<bean id="bookService" class="com.library.service.BookService">  
 <constructor-arg ref="bookRepository"/>  
</bean>

* BookService.java:

public BookService(BookRepository bookRepository) {  
 this.bookRepository = bookRepository;  
}

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

* Add before and after advice:

@Before("execution(\* com.library.service.\*.\*(..))")  
public void beforeAdvice() {  
 System.out.println("Before method execution.");  
}  
  
@After("execution(\* com.library.service.\*.\*(..))")  
public void afterAdvice() {  
 System.out.println("After method execution.");  
}

### **Exercise 9: Spring Boot Application**

1. Use Spring Initializr with Spring Web, Spring Data JPA, H2 DB.
2. application.properties:

spring.datasource.url=jdbc:h2:mem:testdb  
spring.datasource.driverClassName=org.h2.Driver  
spring.datasource.username=sa  
spring.datasource.password=  
spring.jpa.hibernate.ddl-auto=update

1. Book.java Entity:

@Entity  
public class Book {  
 @Id  
 @GeneratedValue(strategy = GenerationType.IDENTITY)  
 private Long id;  
 private String title;  
 private String author;  
 // Getters & Setters  
}

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

1. Run with @SpringBootApplication main class.