Exercise 1: Configuring a Basic Spring Application

step 1: Set Up a Spring Project

1. Create a Maven project named LibraryManagement:

mvn archetype:generate -DgroupId=com.library -DartifactId=LibraryManagement -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

2. Add Spring Core dependencies in the pom.xml file:

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.10</version>

</dependency>

</dependencies>

Step 2: Configure the Application Context

1. Create an XML configuration file named applicationContext.xml` in the src/main/resources directory:

<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="bookService" class="com.library.service.BookService"/>

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

</beans>

Step 3: Define Service and Repository Classes

1. Create a package com.library.service and add a class BookService:

```
package com.library.service;

public class BookService {

}
```

2. Create a package com.library.repository and add a class BookRepository:

```
package com.library.repository;

public class BookRepository {
}
```

Step 4: Run the Application

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

```
package com.library;
import com.library.service.BookService;
org.springframework.context.ApplicationContext;
org.springframework.context.support.ClassPathXmlApplicationContext;
public class LibraryManagementApplication {
public static void main(String[] args) {
ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");
BookService bookService = context.getBean("bookService", BookService.class);
System.out.println("BookService bean initialized: " + (bookService != null));
}
}
```

Exercise 2: Implementing Dependency Injection

Step 1: Modify the XML Configuration

1. 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">

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

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

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

</bean>

</beans>

Step 2: Update the BookService Class

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

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

}

Step 3: Test the Configuration

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

```
package com.library;

import com.library.service.BookService;
org.springframework.context.ApplicationContext;
org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryManagementApplication {
public static void main(String[] args) {
ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");
BookService bookService = context.getBean("bookService", BookService.class);
System.out.println("BookService bean initialized: " + (bookService != null));
System.out.println("BookRepository injected: " + (bookService.getBookRepository() != null));
}
}
```

Exercise 3: Implementing Logging with Spring AOP

Step 1: Add Spring AOP Dependency

1. Update `pom.xml` to include Spring AOP dependency:

```
<dependencies>
<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-context</artifactId>
<version>5.3.10</version>
</dependency>
<dependency>
<groupId>org.springframework</groupId>
<artifactId>spring-aop</artifactId>
<version>5.3.10</version>
```

</dependency>

</dependencies>

Step 2: Create an Aspect for Logging

1. Create a package com.library.aspect and add a class LoggingAspect:

```
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;

}

}
```

Step 3: Enable AspectJ Support

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

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

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

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

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">

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

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

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

</bean>

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

<aop:aspectj-autoproxy/>

</beans>
```

Step 4: Test the Aspect

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

```
package com.library;

import com.library.service.BookService;

org.springframework.context.ApplicationContext;

org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryManagementApplication {

public static void main(String[] args) {

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

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

System.out.println("BookService bean initialized: " + (bookService != null));

}

}
```

Exercise 4: Creating and Configuring a Maven Project

Step 1: Create a New Maven Project

1. Create a new Maven project named LibraryManagement:

mvn archetype:generate -DgroupId=com.library -DartifactId=LibraryManagement -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

Step 2: Add Spring Dependencies in pom.xml

1. Include dependencies for Spring Context, Spring AOP, and Spring WebMVC:

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.10</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.10</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.10</version>

</dependency>

</dependencies>

Step 3:

Configure Maven Plugins

1. 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>

Exercise 5: Configuring the Spring IoC Container

Step 1: Create Spring Configuration File

1. Create an XML configuration file named `applicationContext.xml` in the src/main/resources directory:

<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="bookService" class="com.library.service.BookService"/>

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

</beans>

Step 2: Update the BookService Class

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

package com.library.service;

```
import com.library.repository.BookRepository;

public class BookService {
private BookRepository bookRepository;

public void setBookRepository(BookRepository bookRepository) {
this.bookRepository = bookRepository;
}

}
```

Step 3: Run the Application

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

```
package com.library;

import com.library.service.BookService;
org.springframework.context.ApplicationContext;
org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryManagementApplication {
public static void main(String[] args) {
ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");
BookService bookService = context.getBean("bookService", BookService.class);
System.out.println("BookService bean initialized: " + (bookService != null));
}
}
```

Exercise 6: Configuring Beans with Annotations

Step 1: Enable Component Scanning

1. Update applicationContext.xml` to include component scanning for the com.library package:

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:context="http://www.springframework.org/schema/context"
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">

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

Step 2: Annotate Classes

1. Use @Service annotation for the BookService class:

```
package com.library.service;

import org.springframework.stereotype.Service;

@Service
public class BookService {
}
```

2. Use @Repository annotation for the BookRepository class:

```
package com.library.repository;

import org.springframework.stereotype.Repository;

@Repository
public class BookRepository {
```

}

Step 3: Test the Configuration

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

package com.library;

import com.library.service.BookService;

org.springframework.context.ApplicationContext;

org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryManagementApplication {

public static void main(String[] args) {

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

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

System.out.println("BookService bean initialized: " + (bookService != null));

}

}

Exercise 7: Implementing Constructor and Setter Injection

Step 1: Configure Constructor Injection

1. Update applicationContext.xml to configure constructor injection for BookServic:

<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">

<constructor-arg ref="bookRepository"/>

</bean>

</beans>

Step 2: Configure Setter Injection

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

```
package com.library.service;

import com.library.repository.BookRepository;

public class BookService {
private BookRepository bookRepository;

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

public void setBookRepository(BookRepository bookRepository) {
this.bookRepository = bookRepository;
}

}
```

2. Update applicationContext.xml for setter injection:

```
<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>

Step 3: Test the Injection

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

```
package com.library;

import com.library.service.BookService;
org.springframework.context.ApplicationContext;
org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryManagementApplication {
public static void main(String[] args) {
ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");
BookService bookService = context.getBean("bookService", BookService.class);
System.out.println("BookService bean initialized: " + (bookService != null));
System.out.println("BookRepository injected via constructor: " + (bookService.getBookRepository() !=
null));
}
}
```

Exercise 8: Implementing Basic AOP with Spring

Step 1: Define an Aspect

1. Create a package com.library.aspect and add a class LoggingAspect:

```
package com.library.aspect;
```

```
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.aspectj.lang.annotation.After;

@Aspect
public class LoggingAspect {
@Before("execution(* com.library.service.*.*(..))")
public void logBefore() {
System.out.println("Method execution started");
}

@After("execution(* com.library.service.*.*(..))")
public void logAfter() {
System.out.println("Method execution finished");
}
}
```

Step 2: Create Advice Methods

1. Define advice methods in LoggingAspect for logging before and after method execution:

Step 3: Configure the Aspect

1. Update applicationContext.xml to register the aspect and enable AspectJ auto-proxying:

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:aop="http://www.springframework.org/schema/aop"
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">
```

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

<bean id="bookService" class="com.library.service.BookService">
<property name="bookRepository" ref="bookRepository"/>
</bean>

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

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

Step 4: Test the Aspect

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

```
package com.library;

import com.library.service.BookService;
org.springframework.context.ApplicationContext;
org

.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryManagementApplication {
public static void main(String[] args) {
ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");
BookService bookService = context.getBean("bookService", BookService.class);
bookService.someMethod();  // Ensure this method exists to trigger the aspect
}
}
```

Exercise 9: Creating a Spring Boot Application

Step 2: Add Dependencies

```
<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>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-test</artifactId>
<scope>test</scope>
</dependency>
</dependencies>
```

Step 3: Create Application Properties

```
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
```

Step 4: Define Entities and Repositories

```
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;
}
```

2.Create BookRepository interface:

```
package com.library.repository;

import com.library.entity.Book;
import org.springframework.data.jpa.repository.JpaRepository;

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

Step 5: Create a REST Controller

1. Create BookController class to handle CRUD operations:

```
package com.library.controller;

import com.library.entity.Book;
import com.library.repository.BookRepository;
```

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

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);
}

@GetMapping("/{id}")
public Book getBookById(@PathVariable Long id) {
return bookRepository.findById(id).orElse(null);
}

@PutMapping("/{id}")
public Book updateBook(@PathVariable Long id, @RequestBody Book bookDetails) {
Book book = bookRepository.findById(id).orElse(null);
book.setTitle(bookDetails.getTitle());
book.setAuthor(bookDetails.getAuthor());
```

```
return bookRepository.save(book);
}

@DeleteMapping("/{id}")
public void deleteBook(@PathVariable Long id) {
bookRepository.deleteById(id);
}
}
```

Step 6: Run the Application

```
package com.library;

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);
}
}
```