Exercise 1: Configuring a Basic Spring Application

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

Steps:

1. Set Up a Spring Project:
   - Create a Maven project named LibraryManagement.
   - Add Spring Core dependencies in the pom.xml file.
2. Configure the Application Context:
   - Create an XML configuration file named applicationContext.xml in the src/main/resources directory.
   - Define beans for BookService and BookRepository in the XML file.
3. Define Service and Repository Classes:
   - Create a package com.library.service and add a class BookService.
   - Create a package com.library.repository and add a class BookRepository.
4. Run the Application:
   - Create a main class to load the Spring context and test the configuration.

CODE:

LibraryManagementApplication.java

java

Copy code

```
package com.library;


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");
    }
}
```

applicationContext.xml

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">

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

</beans>
```

BookService.java

java

Copy code

```
package com.library.service;

import com.library.repository.BookRepository;

public class BookService {
    private BookRepository bookRepository;
    public void setBookRepository(BookRepository bookRepository) {
        this.bookRepository = bookRepository;
    }
}
```

BookRepository.java

java

Copy code

```
package com.library.repository;


public class BookRepository {
}
```

Exercise 2: Implementing Dependency Injection

Scenario:
In the library management application, you need to manage the dependencies between the BookService and BookRepository classes using Spring's IoC and DI.

Steps:

1. Modify the XML Configuration:
   - Update applicationContext.xml to wire BookRepository into BookService.
2. Update the BookService Class:
   - Ensure that BookService class has a setter method for BookRepository.
3. Test the Configuration:
   - Run the LibraryManagementApplication main class to verify the dependency injection.

CODE:

applicationContext.xml

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">

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

</beans>
```

BookService.java

java

Copy code

```java
package com.library.service;

import com.library.repository.BookRepository;

public class BookService {
   private BookRepository bookRepository;

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

LibraryManagementApplication.java

java

Copy code

```
package com.library;


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");

   }

}
```

Exercise 3: Implementing Logging with Spring AOP

Scenario:
The library management application requires logging capabilities to track method execution times.

Steps:

1. Add Spring AOP Dependency:
   - Update pom.xml to include Spring AOP dependency.
2. Create an Aspect for Logging:
   - Create a package com.library.aspect and add a class LoggingAspect with a method to log execution times.
3. Enable AspectJ Support:
   - Update applicationContext.xml to enable AspectJ support and register the aspect.
4. Test the Aspect:
   - Run the LibraryManagementApplication main class and observe the console for log messages indicating method execution times.

CODE:

applicationContext.xml

xml

Copy code

```
<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">

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

</beans>
```

LoggingAspect.java

java

Copy code

```
package com.library.aspect;

import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.After;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.springframework.stereotype.Component;

@Aspect
```

```
@Component
public class LoggingAspect {

    @Before("execution(* com.library..*(..))")
    public void logBefore(JoinPoint joinPoint) {
        System.out.println("Before method: " + joinPoint.getSignature());
    }

    @After("execution(* com.library..*(..))")
    public void logAfter(JoinPoint joinPoint) {
        System.out.println("After method: " + joinPoint.getSignature());
    }
}
```

LibraryManagementApplication.java

java

Copy code

```
package com.library;

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");
    }
```

}

Exercise 4: Creating and Configuring a Maven Project

Scenario:
You need to set up a new Maven project for the library management application and add Spring dependencies.

Steps:

1. Create a New Maven Project:
   - Create a new Maven project named LibraryManagement.
2. Add Spring Dependencies in pom.xml:
   - Include dependencies for Spring Context, Spring AOP, and Spring WebMVC.
3. Configure Maven Plugins:
   - Configure the Maven Compiler Plugin for Java version 1.8 in the pom.xml file.

CODE:

LibraryManagementApp.java

java

Copy code

```
import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;


@SpringBootApplication

public class LibraryManagementApp {

  public static void main(String[] args) {

    System.out.println("Welcome to Library Management Application!");

  }

}
```

LibraryManagementAppTests.java

java

Copy code

```java
import org.springframework.boot.test.context.SpringBootTest;
import org.junit.jupiter.api.Test;


@SpringBootTest
class LibraryManagementAppTests {
  @Test
  void contextLoads() {
  }
}
```

pom.xml

xml

Copy code

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.lms</groupId>
  <artifactId>exercise4</artifactId>
  <version>1.0-SNAPSHOT</version>
  <properties>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
  </properties>
  <dependencies>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>6.1.11</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-aop</artifactId>
      <version>6.1.11</version>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
      <version>6.1.11</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>

</project>
```

Exercise 5: Integrating Spring with a Database

Scenario:
Integrate a database into the library management application using Spring Data JPA.

Steps:

1. Add Spring Data JPA Dependency:
   - Update pom.xml to include Spring Data JPA and MySQL dependencies.
2. Configure Database Connection:
   - Create application.properties to configure the database connection.
3. Create Entity and Repository:
   - Define an Entity class for Book and a Repository interface for data access.
4. Test the Repository:
   - Write a simple test to ensure that the repository can perform CRUD operations.

CODE:

Book.java

java

Copy code

```
package com.library.entity;


import javax.persistence.Entity;

import javax.persistence.Id;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;


@Entity

public class Book {
    @Id

    @GeneratedValue(strategy = GenerationType.AUTO)

    private Long id;

    private String title;

    private String author;

    private String isbn;


    // Getters and Setters

}
```

BookRepository.java

java

Copy code

```
package com.library.repository;


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

import com.library.entity.Book;


public interface BookRepository extends JpaRepository<Book, Long> {

}
```

application.properties

properties

Copy code

```
spring.datasource.url=jdbc:mysql://localhost:3306/librarydb

spring.datasource.username=root

spring.datasource.password=password

spring.jpa.hibernate.ddl-auto=update
```

LibraryManagementApplication.java

java

Copy code

```java
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);

  }

}
```

Exercise 6: Configuring Beans with Annotations

Scenario:
You need to simplify the configuration of beans in the library management application using annotations.

Steps:

1. Enable Component Scanning:
   - Update applicationContext.xml to include component scanning for the com.library package.
2. Annotate Classes:
   - Use @Service annotation for the BookService class.
   - Use @Repository annotation for the BookRepository class.
3. Test the Configuration:
   - Run the LibraryManagementApplication main class to verify the annotation-based configuration.

CODE:

applicationContext.xml

xml

Copy code

```
<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>
```

BookService.java

java

Copy code

```
package com.library.service;

import org.springframework.stereotype.Service;
import com.library.repository.BookRepository;

@Service
public class BookService {
  private BookRepository bookRepository;

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

```
    // Other service methods
}
```

BookRepository.java

java

Copy code

```
package com.library.repository;

import org.springframework.stereotype.Repository;

@Repository
public class BookRepository {
    // Repository methods
}
```

LibraryManagementApplication.java

java

Copy code

```
package com.library;

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
    }
}
```

Exercise 7: Implementing Constructor and Setter Injection

Scenario:
The library management application requires both constructor and setter injection for better control over bean initialization.

Steps:

1. Configure Constructor Injection:
    - Update applicationContext.xml to configure constructor injection for BookService.
2. Configure Setter Injection:
    - Ensure that the BookService class has a setter method for BookRepository and configure it in applicationContext.xml.
3. Test the Injection:
    - Run the LibraryManagementApplication main class to verify both constructor and setter injection.

CODE:

applicationContext.xml

xml

Copy code

```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">
        <constructor-arg ref="bookRepository"/>
    </bean>

</beans>
```

BookService.java

java

Copy code

```java
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;
    }

    // Other service methods
}
```

LibraryManagementApplication.java

java

Copy code

```
package com.library;


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

  }

}
```

Exercise 8: Implementing Basic AOP with Spring

Scenario:
The library management application requires basic AOP functionality to separate cross-cutting concerns like logging and transaction management.

Steps:

1. Define an Aspect:
    - Create a package com.library.aspect and add a class LoggingAspect.
2. Create Advice Methods:
    - Define advice methods in LoggingAspect for logging before and after method execution.
3. Configure the Aspect:
    - Update applicationContext.xml to register the aspect and enable AspectJ auto-proxying.
4. Test the Aspect:

- Run the LibraryManagementApplication main class to verify the AOP functionality.

CODE:

applicationContext.xml

xml

Copy code

```
<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">

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

</beans>
```

LoggingAspect.java

java

Copy code

```
package com.library.aspect;

import org.aspectj.lang.JoinPoint;
import org.aspectj.lang.annotation.After;
import org.aspectj.lang.annotation.Aspect;
import org.aspectj.lang.annotation.Before;
import org.springframework.stereotype.Component;
```

```
@Aspect

@Component

public class LoggingAspect {


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

   public void logBefore(JoinPoint joinPoint) {

      System.out.println("Before method: " + joinPoint.getSignature());

   }


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

   public void logAfter(JoinPoint joinPoint) {

      System.out.println("After method: " + joinPoint.getSignature());

   }

}
```

LibraryManagementApplication.java

java

Copy code

```
package com.library;


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
    }
}
```

Exercise 9: Creating a Spring Boot Application

Scenario:
You need to create a Spring Boot application for the library management system to simplify configuration and deployment.

Steps:

1. Create a Spring Boot Project:
   - Use Spring Initializr to create a new Spring Boot project named LibraryManagement.
2. Add Dependencies:
   - Include dependencies for Spring Web, Spring Data JPA, and H2 Database.
3. Create Application Properties:
   - Configure database connection properties in application.properties.
4. Define Entities and Repositories:
   - Create entities and repositories for Book in the com.library package.
5. Run the Application:
   - Create a main class with @SpringBootApplication and run the application.

CODE:

LibraryManagementApplication.java

java

Copy code

```
package com.library;


import org.springframework.boot.SpringApplication;
```

```java
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class LibraryManagementApplication {
    public static void main(String[] args) {
        SpringApplication.run(LibraryManagementApplication.class, args);
    }
}
```

application.properties

properties

Copy code

```properties
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driver-class-name=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=password
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
```

Book.java

java

Copy code

```java
package com.library.model;

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
}
```

BookRepository.java

java

Copy code

```java
package com.library.repository;

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

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

application.properties

properties

Copy code

```properties
spring.datasource.url=jdbc:h2:mem:testdb
spring.datasource.driver-class-name=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=password
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
```