

## 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:

#### **pom.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/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>org.example</groupId>
  <artifactId>LibraryManagementSystem</artifactId>
  <version>1.0-SNAPSHOT</version>

  <properties>
    <maven.compiler.source>17</maven.compiler.source>
    <maven.compiler.target>17</maven.compiler.target>
  </properties>
```

```
<dependencies>
  <!-- Spring Core -->
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-context</artifactId>
    <version>5.3.30</version>
  </dependency>

  <!-- Spring AOP -->
  <dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-aop</artifactId>
    <version>5.3.30</version>
  </dependency>

  <!-- AspectJ for AOP -->
  <dependency>
    <groupId>org.aspectj</groupId>
    <artifactId>aspectjweaver</artifactId>
    <version>1.9.20.1</version>
  </dependency>

  <!-- JUnit for testing (optional) -->
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.13.2</version>
    <scope>test</scope>
  </dependency>
</dependencies>

<build>
  <plugins>
    <!-- Maven Compiler Plugin -->
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-compiler-plugin</artifactId>
      <version>3.11.0</version>
      <configuration>
        <source>${maven.compiler.source}</source>
        <target>${maven.compiler.target}</target>
      </configuration>
    </plugin>
  </plugins>
</build>
```

```
    </plugins>
</build>
</project>
```

### **BookRepository.java**

src/main/java/com.library/repository

```
package com.library.repository;
```

```
import org.springframework.stereotype.Repository;
```

```
@Repository
```

```
public class BookRepository {
    public void save() {
        System.out.println("BookRepository: book saved.");
    }
}
```

### **BookService.java**

src/main/java/com.library/service

```
package com.library.service;
```

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

```
import org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.stereotype.Service;
```

```
@Service("bookService")
```

```
public class BookService {
```

```
    @Autowired
```

```
    private BookRepository bookRepository;
```

```
    public void performSave() {
```

```
        System.out.println("BookService: calling repository...");
```

```
        bookRepository.save();
```

```
    }
```

```
}
```

### **MainApp.java**

Src/main/java/com.library/MainApp

```

package com.library;

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

public class MainApp {
    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");
        BookService service = context.getBean("bookService", BookService.class);
        service.performSave();
    }
}

```

### **src/main/resources**

```

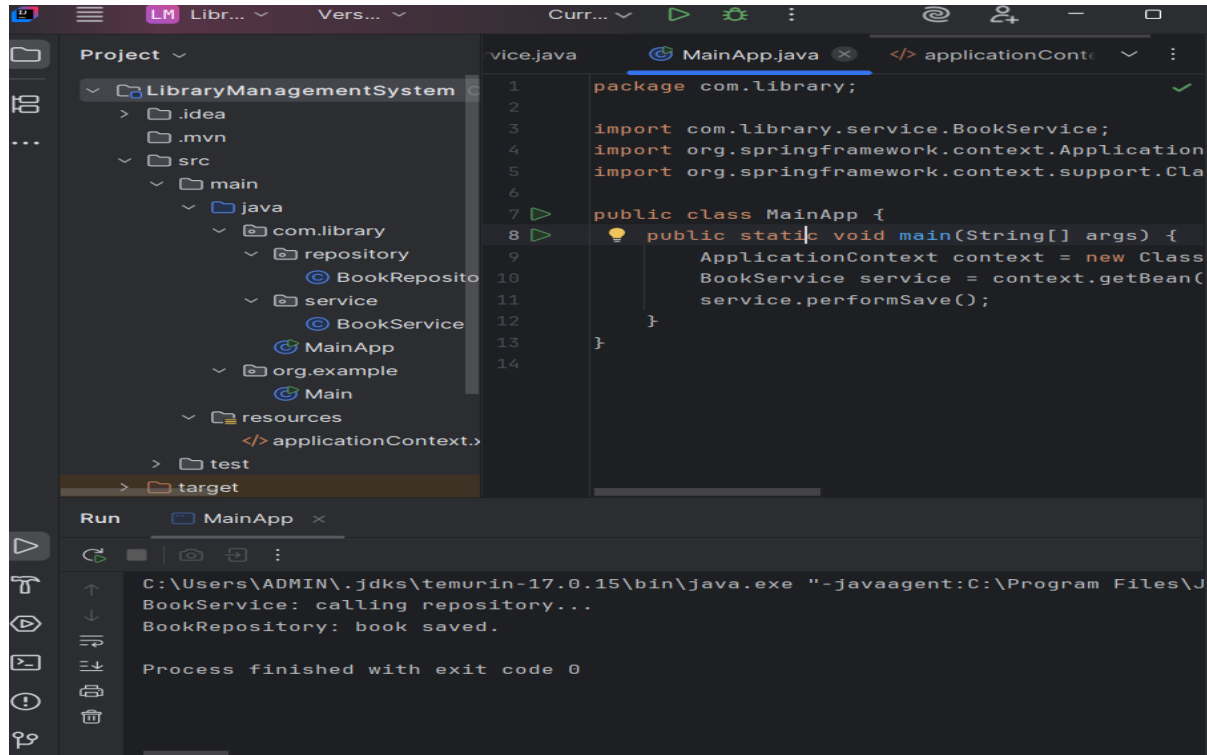
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:context="http://www.springframework.org/schema/context"
    xmlns:aop="http://www.springframework.org/schema/aop"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="
        http://www.springframework.org/schema/beans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

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

    <!-- Enable AOP -->
    <aop:aspectj-autoproxy />
</beans>

```

OUTPUT:



```
Project
├── LibraryManagementSystem
│   ├── .idea
│   ├── .mvn
│   └── src
│       ├── main
│       │   ├── java
│       │   │   ├── com
│       │   │   │   ├── library
│       │   │   │   │   ├── repository
│       │   │   │   │   │   ├── BookRepository
│       │   │   │   │   ├── service
│       │   │   │   │   │   ├── BookService
│       │   │   │   │   ├── MainApp
│       │   │   │   ├── org
│       │   │   │   │   ├── example
│       │   │   │   │   │   ├── Main
│       │   │   │   ├── resources
│       │   │   │   │   ├── applicationContext.xml
│       │   ├── test
│       └── target
└── Run
    ├── MainApp
    └── Console
        ├── C:\Users\ADMIN\jdk\temurin-17.0.15\bin\java.exe "-javaagent:C:\Program Files\J
        ├── BookService: calling repository...
        ├── BookRepository: book saved.
        └── Process finished with exit code 0
```

## Exercise 2: Implementing Dependency Injection

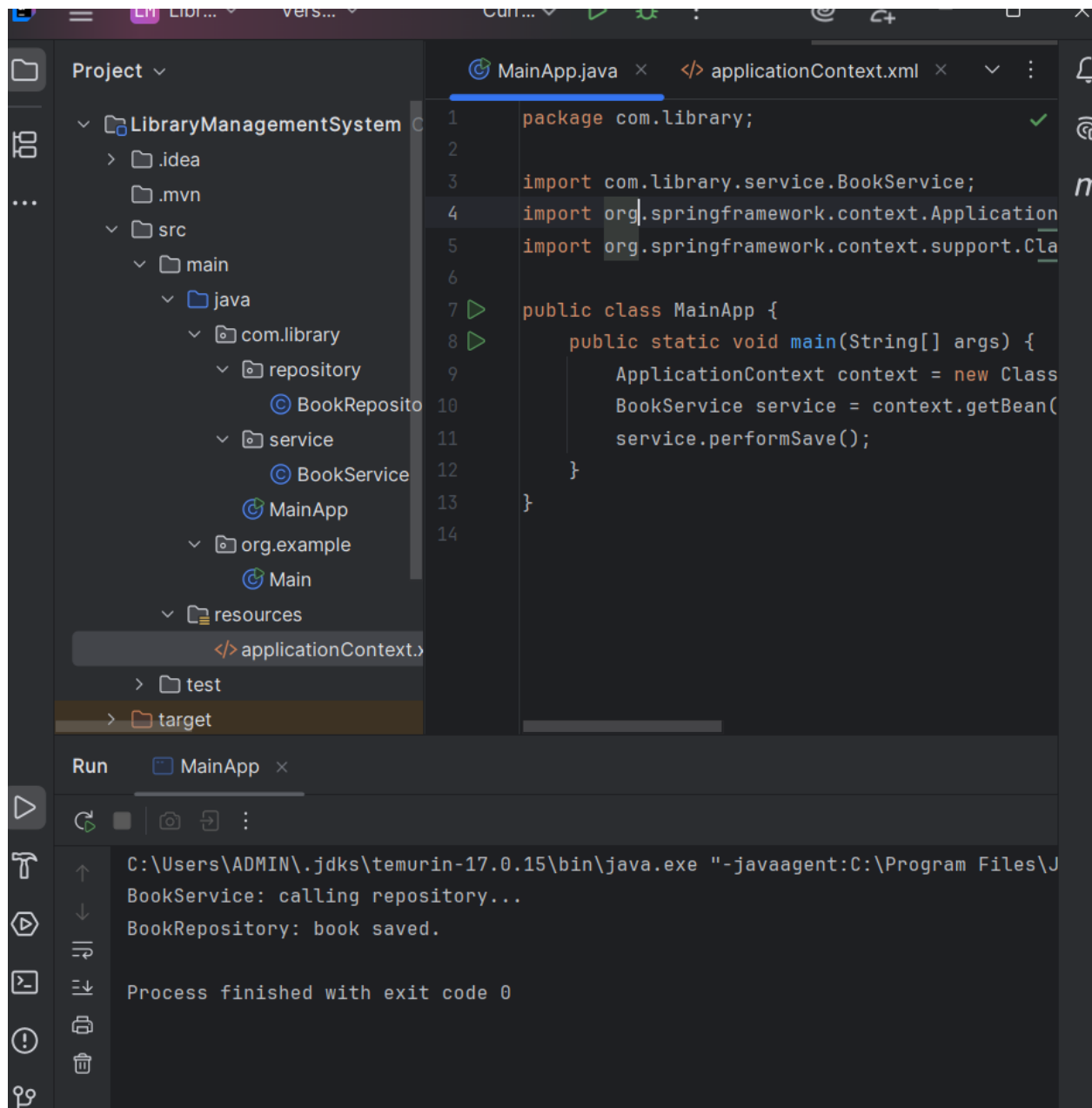
### Scenario:

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.

OUTPUT:



The screenshot displays an IDE interface with a project named 'LibraryManagementSystem'. The project structure on the left includes folders for '.idea', '.mvn', 'src' (containing 'main' and 'test'), and 'target'. The 'main' folder contains a 'java' package with sub-packages 'com.library' (containing 'repository' and 'service') and 'org.example'. The 'service' package contains 'BookRepository' and 'BookService'. The 'org.example' package contains 'MainApp' and 'Main'. The 'resources' folder is also visible. The 'MainApp.java' file is open in the editor, showing the following code:

```
1 package com.library;
2
3 import com.library.service.BookService;
4 import org.springframework.context.ApplicationContext;
5 import org.springframework.context.support.ClassPathXmlApplicationContext;
6
7 public class MainApp {
8     public static void main(String[] args) {
9         ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");
10        BookService service = context.getBean(BookService.class);
11        service.performSave();
12    }
13 }
14
```

The 'Run' tab at the bottom shows the execution of 'MainApp'. The output is as follows:

```
C:\Users\ADMIN\.jdk\temurin-17.0.15\bin\java.exe "-javaagent:C:\Program Files\Java\jdk-17.0.15\lib\jrt-fs.jar;-Dspring.config.location=src/main/resources/applicationContext.xml"
BookService: calling repository...
BookRepository: book saved.
Process finished with exit code 0
```

## Exercise 4: 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.

## 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:

#### LoggingAspect.java

```
package com.library.aspect;

import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.*;
import org.springframework.stereotype.Component;

@Aspect
@Component
```

```

public class LoggingAspect {

    @Around("execution(* com.library.service.*(..))")
    public Object logExecutionTime(ProceedingJoinPoint joinPoint) throws Throwable {
        long start = System.currentTimeMillis();
        Object result = joinPoint.proceed();
        long timeTaken = System.currentTimeMillis() - start;
        System.out.println(joinPoint.getSignature() + " executed in " + timeTaken + "ms");
        return result;
    }
}

```

## OUTPUT:

The screenshot shows an IDE with the following components:

- Project Explorer:** Shows a project named 'LibraryManagementSystem' with a package structure: `com.library.aspect` (containing `LoggingAspect`), `com.library.repository` (containing `BookRepository`), and `com.library.service` (containing `BookService`).
- Editor:** Displays the `LoggingAspect.java` file. The code is as follows:
 

```

package com.library.aspect;

import org.aspectj.lang.ProceedingJoinPoint;
import org.aspectj.lang.annotation.*;
import org.springframework.stereotype.Component;

@Aspect
@Component
public class LoggingAspect {

    @Around("execution(* com.library.service.*(..))")
    public Object logExecutionTime(ProceedingJoinPoint joinPoint) throws Throwable {
        long start = System.currentTimeMillis();
        Object result = joinPoint.proceed();
        long timeTaken = System.currentTimeMillis() - start;
        System.out.println(joinPoint.getSignature() + " executed in " + timeTaken + "ms");
        return result;
    }
}

```
- Run Console:** Shows the output of the application. The output is:
 

```

C:\Users\ADMIN\.jdk\temurin-17.0.15\bin\java.exe "-javaagent:C:\Program Files
BookService: calling repository...
BookRepository: book saved.
void com.library.service.BookService.performSave() executed in 22ms
Process finished with exit code 0

```



## Exercise 5: Configuring the Spring IoC Container

### Scenario:

The library management application requires a central configuration for beans and dependencies.

### Steps:

#### 1. Create Spring Configuration File:

- Create an XML configuration file named **applicationContext.xml** in the **src/main/resources** directory.
- Define beans for **BookService** and **BookRepository** in the XML file.

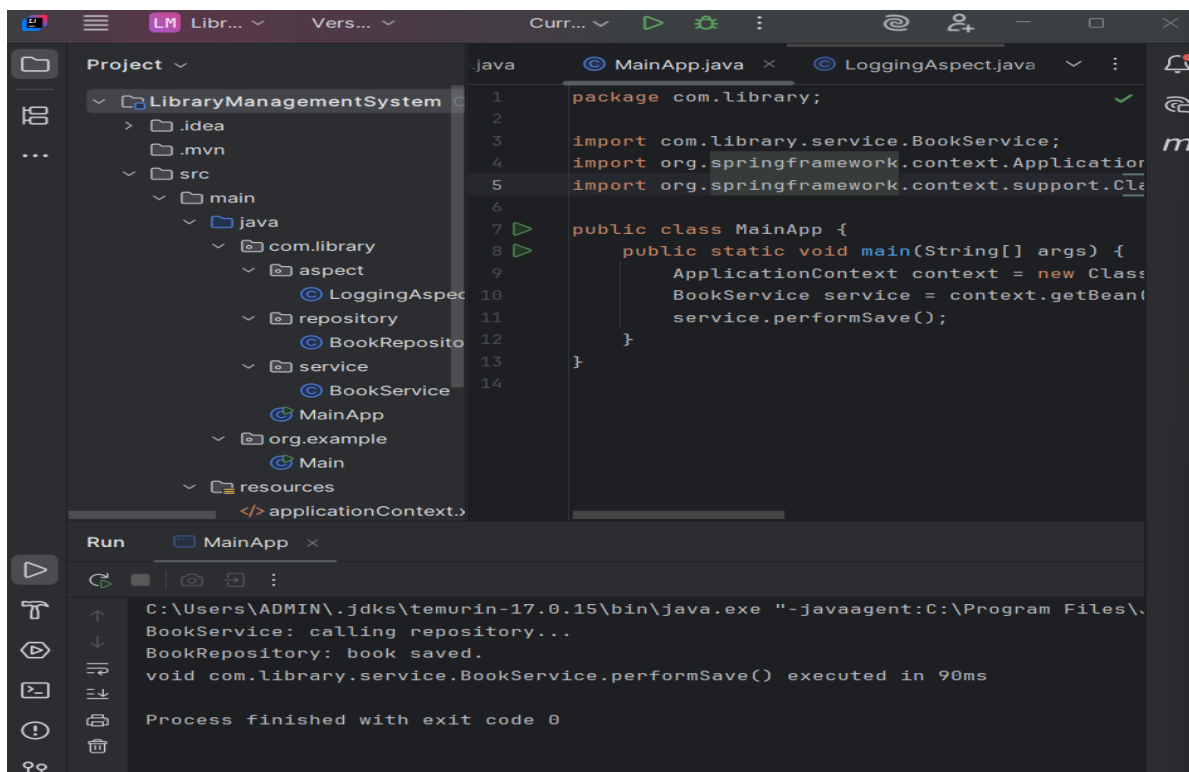
#### 2. Update the BookService Class:

- Ensure that the **BookService** class has a setter method for **BookRepository**.

#### 3. Run the Application:

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

### OUTPUT:



The screenshot displays an IDE interface with the following components:

- Project View:** Shows the project structure for 'LibraryManagementSystem'. The 'src/main/resources' directory contains the 'applicationContext.xml' file.
- Code Editor:** Displays the 'MainApp.java' file with the following code:

```
1 package com.library;  
2  
3 import com.library.service.BookService;  
4 import org.springframework.context.ApplicationContext;  
5 import org.springframework.context.support.ClassPathXmlApplicationContext;  
6  
7 public class MainApp {  
8     public static void main(String[] args) {  
9         ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");  
10        BookService service = context.getBean(BookService.class);  
11        service.performSave();  
12    }  
13 }  
14
```
- Run Console:** Shows the output of the application execution:

```
Run MainApp x  
C:\Users\ADMIN\jdk\temurin-17.0.15\bin\java.exe "-javaagent:C:\Program Files\...  
BookService: calling repository...  
BookRepository: book saved.  
void com.library.service.BookService.performSave() executed in 90ms  
Process finished with exit code 0
```

## 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.

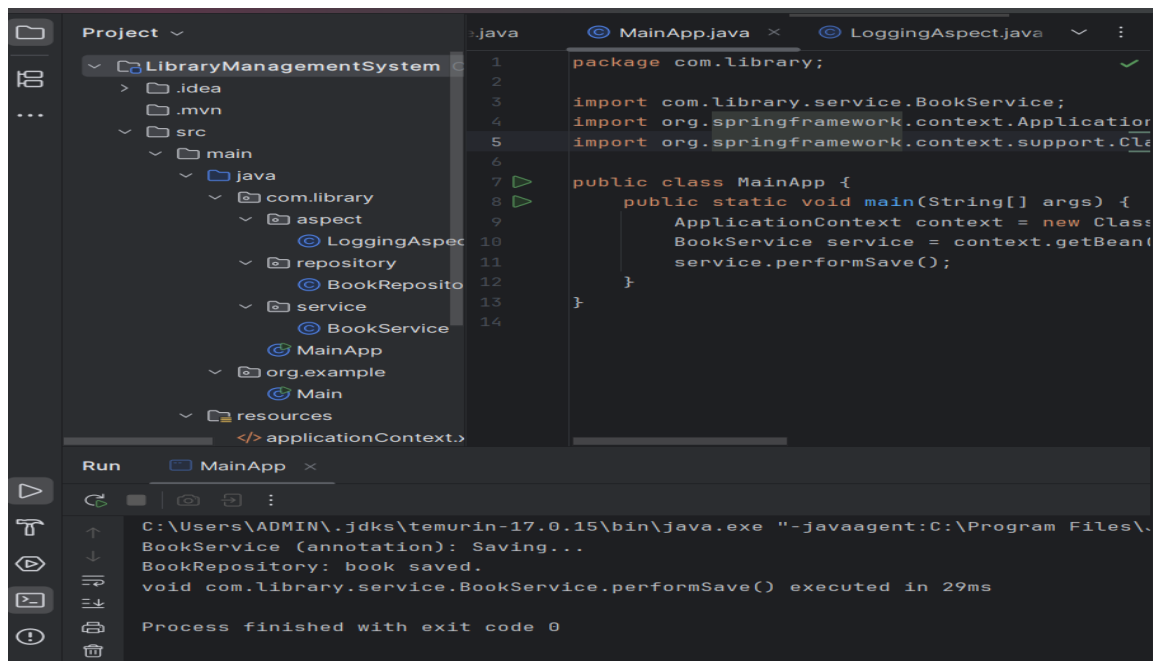
#### 2. Annotate Classes:

- 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.

### OUTPUT:



The screenshot displays an IDE interface. On the left, the 'Project' view shows the directory structure of 'LibraryManagementSystem', including 'src/main/java/com/library' and its sub-packages 'aspect', 'repository', 'service', and 'resources'. The 'MainApp' class is highlighted under the 'service' package. The main editor window shows the code for 'MainApp.java', which includes a package declaration, imports for 'BookService' and 'ApplicationContext', and a 'main' method that creates an 'ApplicationContext', retrieves a 'BookService' bean, and calls 'performSave()'. Below the editor, the 'Run' tab shows the execution output for 'MainApp'. The output text is: 'C:\Users\ADMIN\jdk\temurin-17.0.15\bin\java.exe "-javaagent:C:\Program Files\... BookService (annotation): Saving... BookRepository: book saved. void com.library.service.BookService.performSave() executed in 29ms Process finished with exit code 0'.

```
package com.library;

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

public class MainApp {
    public static void main(String[] args) {
        ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");
        BookService service = context.getBean(BookService.class);
        service.performSave();
    }
}
```

Run MainApp

C:\Users\ADMIN\jdk\temurin-17.0.15\bin\java.exe "-javaagent:C:\Program Files\... BookService (annotation): Saving... BookRepository: book saved. void com.library.service.BookService.performSave() executed in 29ms Process finished with exit code 0

## 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.

## 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.

## 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 **Book** entity and **BookRepository** interface.
5. **Create a REST Controller:** ○ Create a **BookController** class to handle CRUD operations.
6. **Run the Application:**
  - Run the Spring Boot application and test the REST endpoints.

### CODE:

#### **BookService.java**

```
package com.library.service;
```

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

```
import org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.stereotype.Service;
```

```
@Service
```

```
public class BookService {
```

@Autowired

```
private BookRepository bookRepository;
```

```
public void performSave() {
```

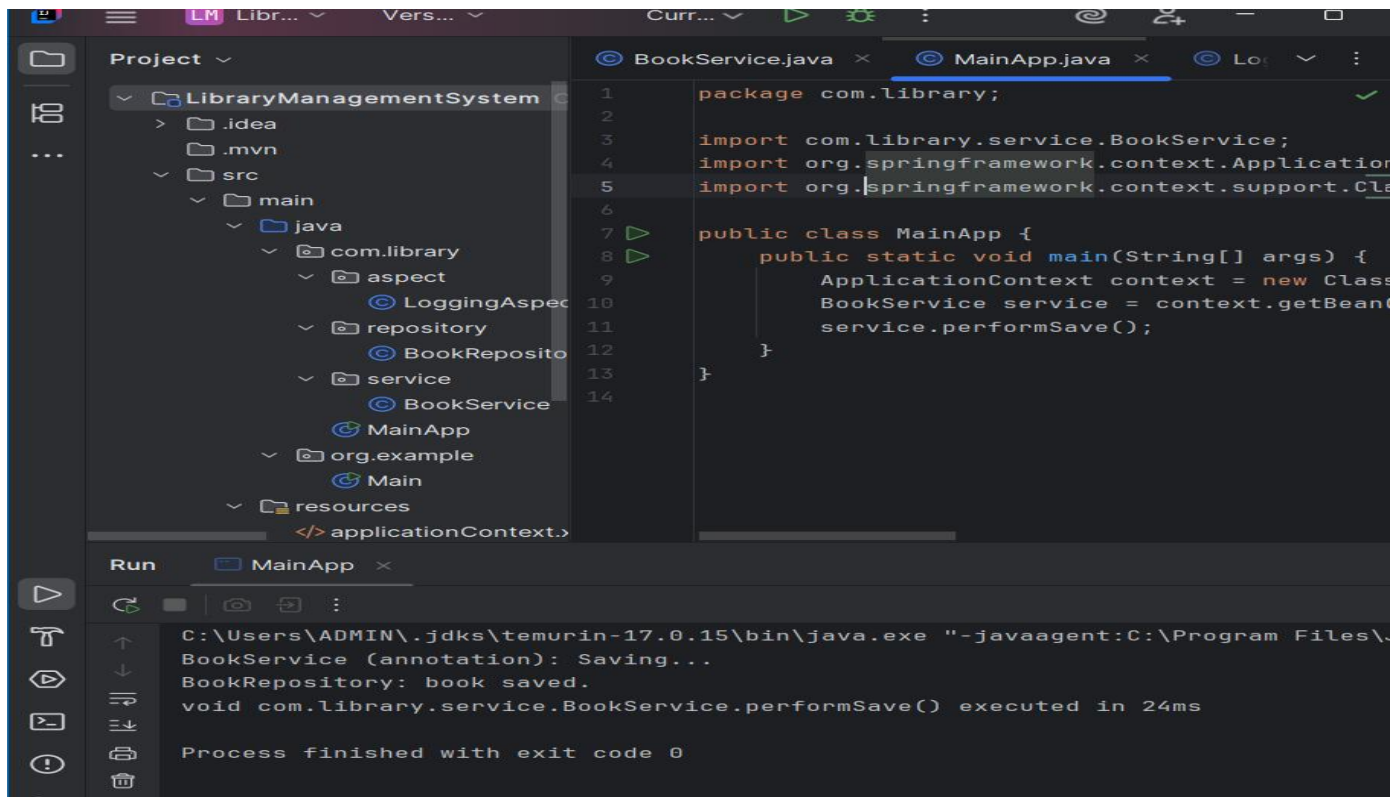
```
    System.out.println("BookService (annotation): Saving...");
```

```
    bookRepository.save();
```

```
}
```

```
}
```

OUTPUT:



The screenshot displays an IDE interface with a project named 'LibraryManagementSystem'. The project structure on the left includes folders for '.idea', '.mvn', 'src', 'main', 'java', 'com.library', 'aspect', 'repository', 'service', 'org.example', and 'resources'. The 'MainApp.java' file is open in the editor, showing the following code:

```
1 package com.library;
2
3 import com.library.service.BookService;
4 import org.springframework.context.ApplicationContext;
5 import org.springframework.context.support.ClassPathXmlApplicationContext;
6
7 public class MainApp {
8     public static void main(String[] args) {
9         ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");
10         BookService service = context.getBean(BookService.class);
11         service.performSave();
12     }
13 }
14
```

The 'Run' tab at the bottom shows the execution output:

```
C:\Users\ADMIN\jdk\temurin-17.0.15\bin\java.exe "-javaagent:C:\Program Files\Java\jdk-17.0.15\lib\jrt-fs.jar" -Dspring.config.location=classpath:/applicationContext.xml
BookService (annotation): Saving...
BookRepository: book saved.
void com.library.service.BookService.performSave() executed in 24ms
Process finished with exit code 0
```