**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:**

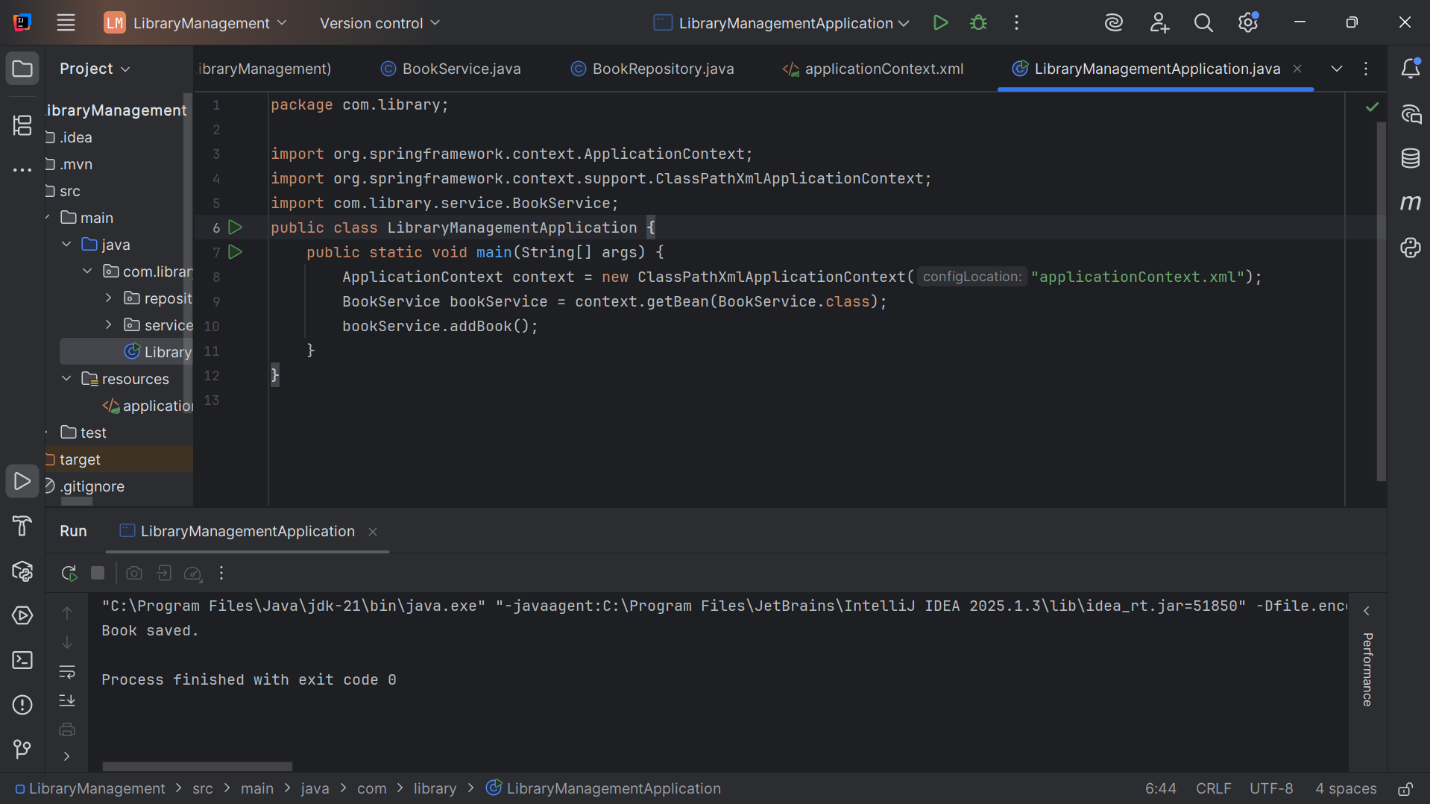
1. **Set Up a Spring Project:**
   * Create a Maven project named **LibraryManagement**.
   * Add Spring Core dependencies in the **pom.xml** file.

<dependencies>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>5.3.34</version>  
 </dependency>  
</dependencies>

1. **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.

<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. **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**.
2. **Run the Application:**
   * Create a main class to load the Spring context and test the configuration.
3. 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 = context.getBean(BookService.class);  
    bookService.addBook();  
    }  
   }

**Output:**

**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:**

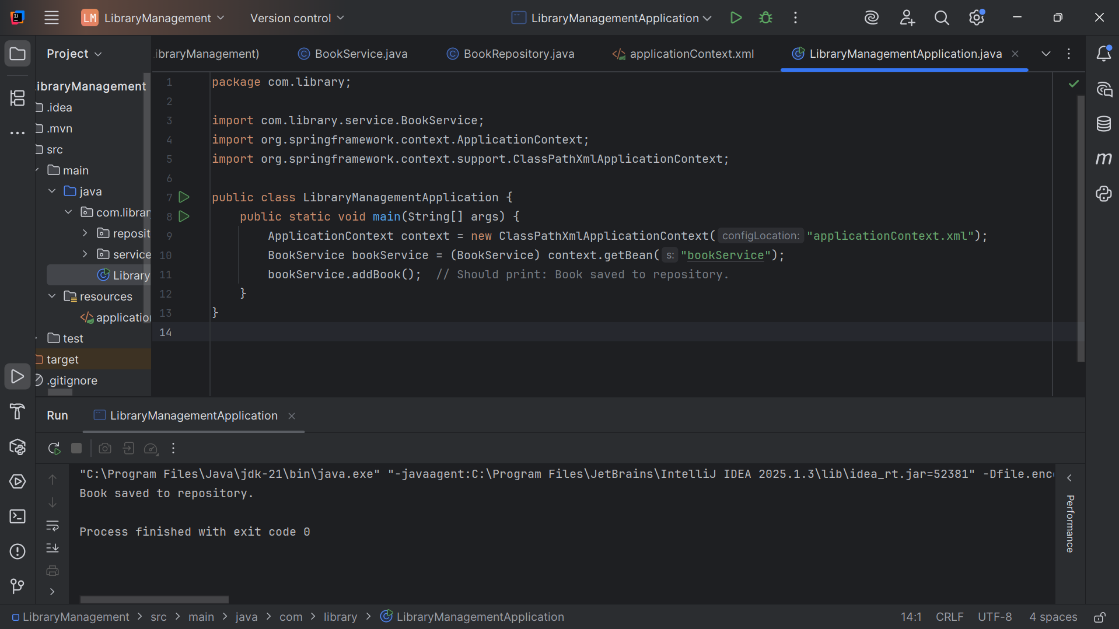
1. **Modify the XML Configuration:**
   * Update **applicationContext.xml** to wire **BookRepository** into **BookService**.

<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. **Update the BookService Class:**
   * Ensure that **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;  
 }  
  
 public void addBook() {  
 bookRepository.saveBook();  
 }  
}

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

****

**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.

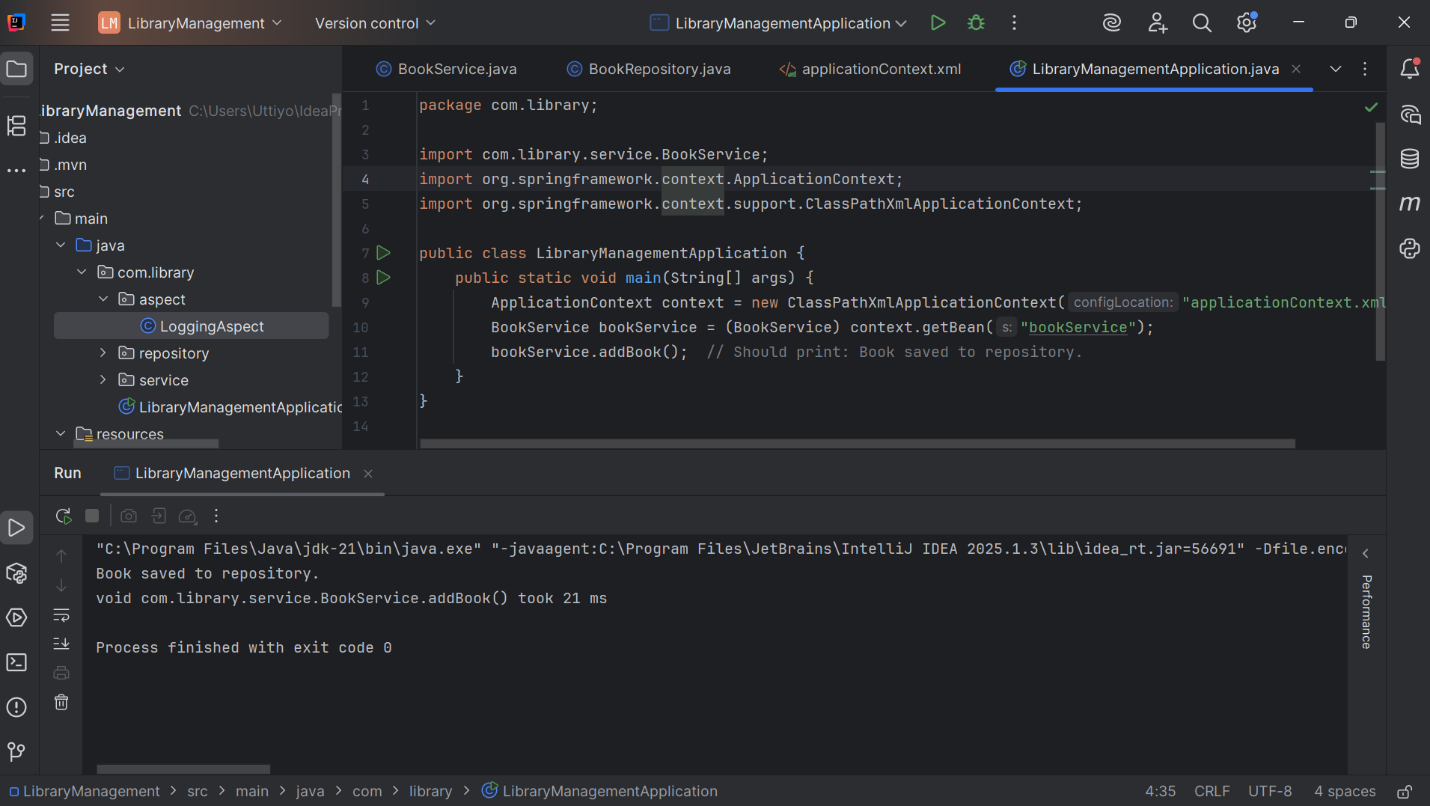
<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>com.library</groupId>  
 <artifactId>LibraryManagement</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <properties>  
 <maven.compiler.source>21</maven.compiler.source>  
 <maven.compiler.target>21</maven.compiler.target>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>5.3.34</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-aop</artifactId>  
 <version>5.3.34</version>  
 </dependency>  
  
 <dependency>  
 <groupId>org.aspectj</groupId>  
 <artifactId>aspectjweaver</artifactId>  
 <version>1.9.21</version>  
 </dependency>  
  
 </dependencies>  
  
</project>

1. **Create an Aspect for Logging:**
   * Create a package **com.library.aspect** and add a class **LoggingAspect** with a method to log execution times.

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 result = joinPoint.proceed();  
 long end = System.*currentTimeMillis*();  
 System.*out*.println(joinPoint.getSignature() + " took " + (end - start) + " ms");  
 return result;  
 }  
}

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

<?xml version="1.0" encoding="UTF-8"?>  
<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"  
 xmlns:aop="http://www.springframework.org/schema/aop"  
 xsi:schemaLocation="  
 http://www.springframework.org/schema/beans  
 https://www.springframework.org/schema/beans/spring-beans.xsd  
 http://www.springframework.org/schema/context  
 https://www.springframework.org/schema/context/spring-context.xsd  
 http://www.springframework.org/schema/aop  
 https://www.springframework.org/schema/aop/spring-aop.xsd">  
  
 <context:component-scan base-package="com.library"/>  
  
 <aop:aspectj-autoproxy/>  
  
 <bean id="bookRepository" class="com.library.repository.BookRepository"/>  
 <bean id="bookService" class="com.library.service.BookService">  
 <property name="bookRepository" ref="bookRepository"/>  
 </bean>  
  
 <bean class="com.library.aspect.LoggingAspect"/>  
</beans>

1. **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.

**Ans)** <dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.34</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.34</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.34</version>

</dependency

<dependency>

<groupId>org.aspectj</groupId>

<artifactId>aspectjweaver</artifactId>

<version>1.9.21</version>

</dependency>

</dependencies>

1. **Configure Maven Plugins:**
   * Configure the Maven Compiler Plugin for Java version 1.8 in the pom.xml file.

**Ans)** <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**

**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.

**Ans)** Same as Exercise 1 and 2.

**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.

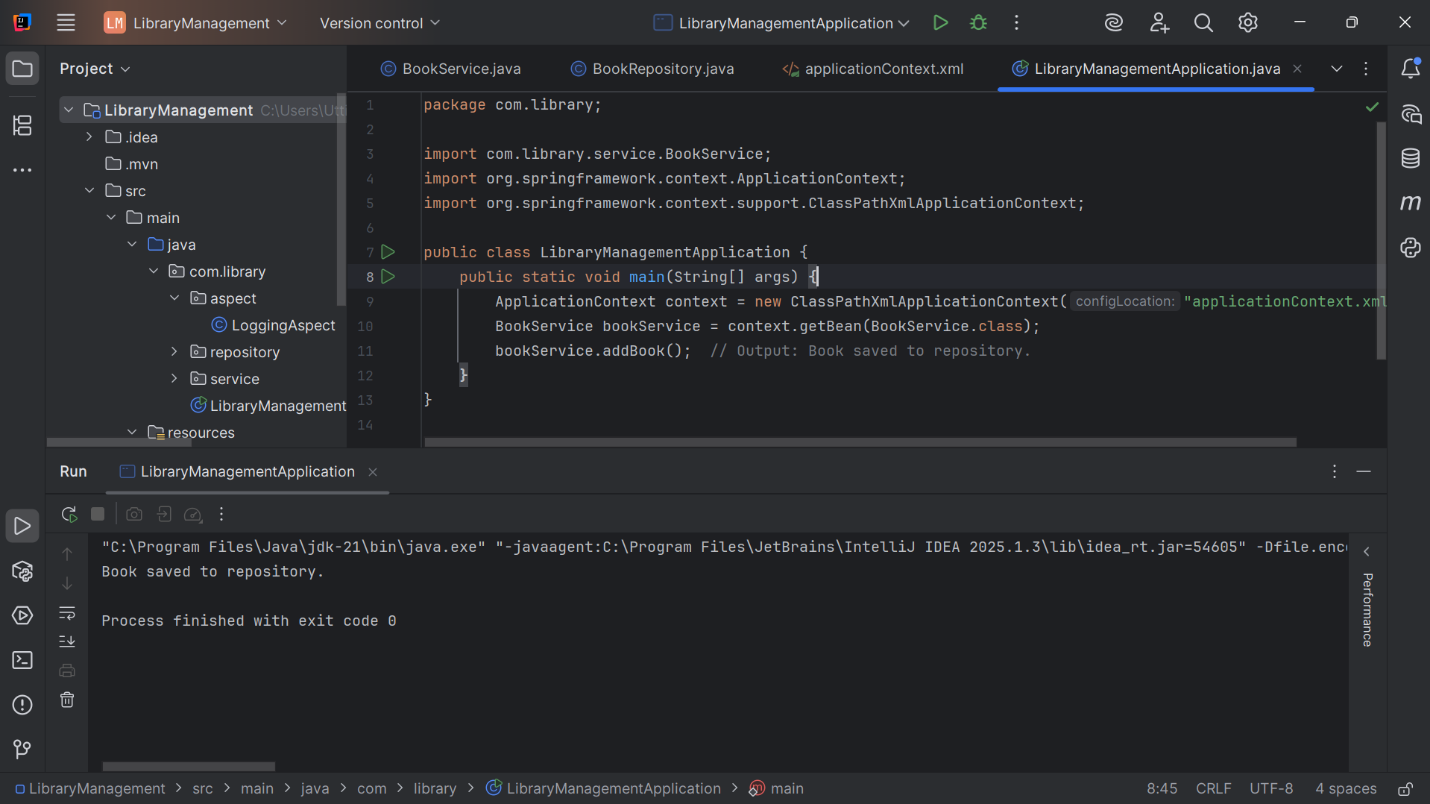
<?xml version="1.0" encoding="UTF-8"?>  
<beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xmlns:context="http://www.springframework.org/schema/context"  
 xsi:schemaLocation="  
 http://www.springframework.org/schema/beans  
 https://www.springframework.org/schema/beans/spring-beans.xsd  
 http://www.springframework.org/schema/context  
 https://www.springframework.org/schema/context/spring-context.xsd">  
   
 <context:component-scan base-package="com.library"/>  
  
</beans>

1. **Annotate Classes:**
   * Use **@Service** annotation for the **BookService** class.

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 {  
  
 private BookRepository bookRepository;  
  
 @Autowired  
 public void setBookRepository(BookRepository bookRepository) {  
 this.bookRepository = bookRepository;  
 }  
  
 public void addBook() {  
 bookRepository.saveBook();  
 }  
}

* + Use **@Repository** annotation for the **BookRepository** class.

package com.library.repository;  
  
import org.springframework.stereotype.Repository;  
  
@Repository  
public class BookRepository {  
 public void saveBook() {  
 System.*out*.println("Book saved to repository.");  
 }  
}

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

**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:**

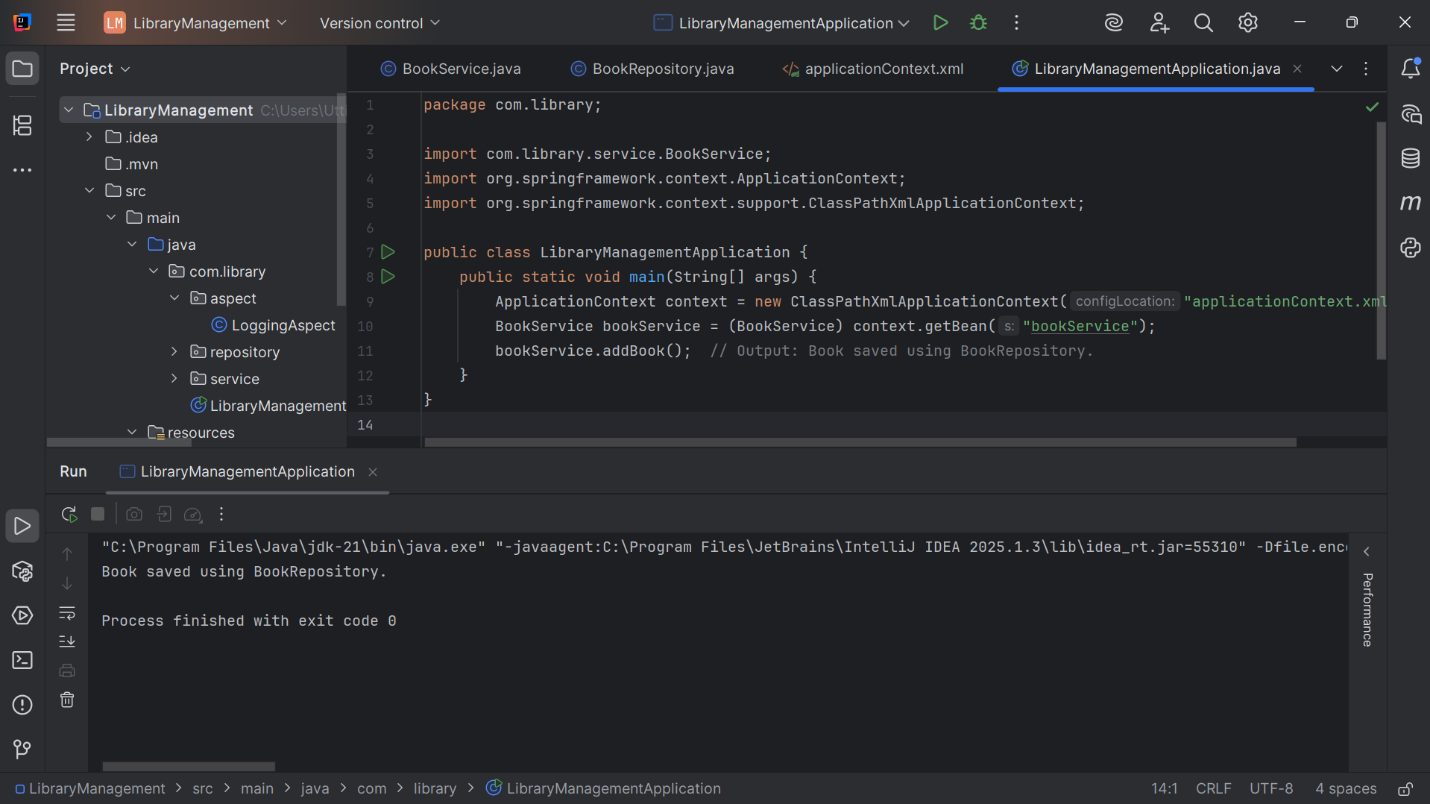
1. **Configure Constructor Injection:**
   * Update applicationContext.**xml** to configure constructor injection for **BookService**.

<?xml version="1.0" encoding="UTF-8"?>  
<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  
 https://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"/>  
  
 <property name="bookRepository" ref="bookRepository"/>  
 </bean>  
  
</beans>

1. **Configure Setter Injection:**
   * 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;  
 }  
  
 public void addBook() {  
 bookRepository.saveBook();  
 }  
}

1. **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:**

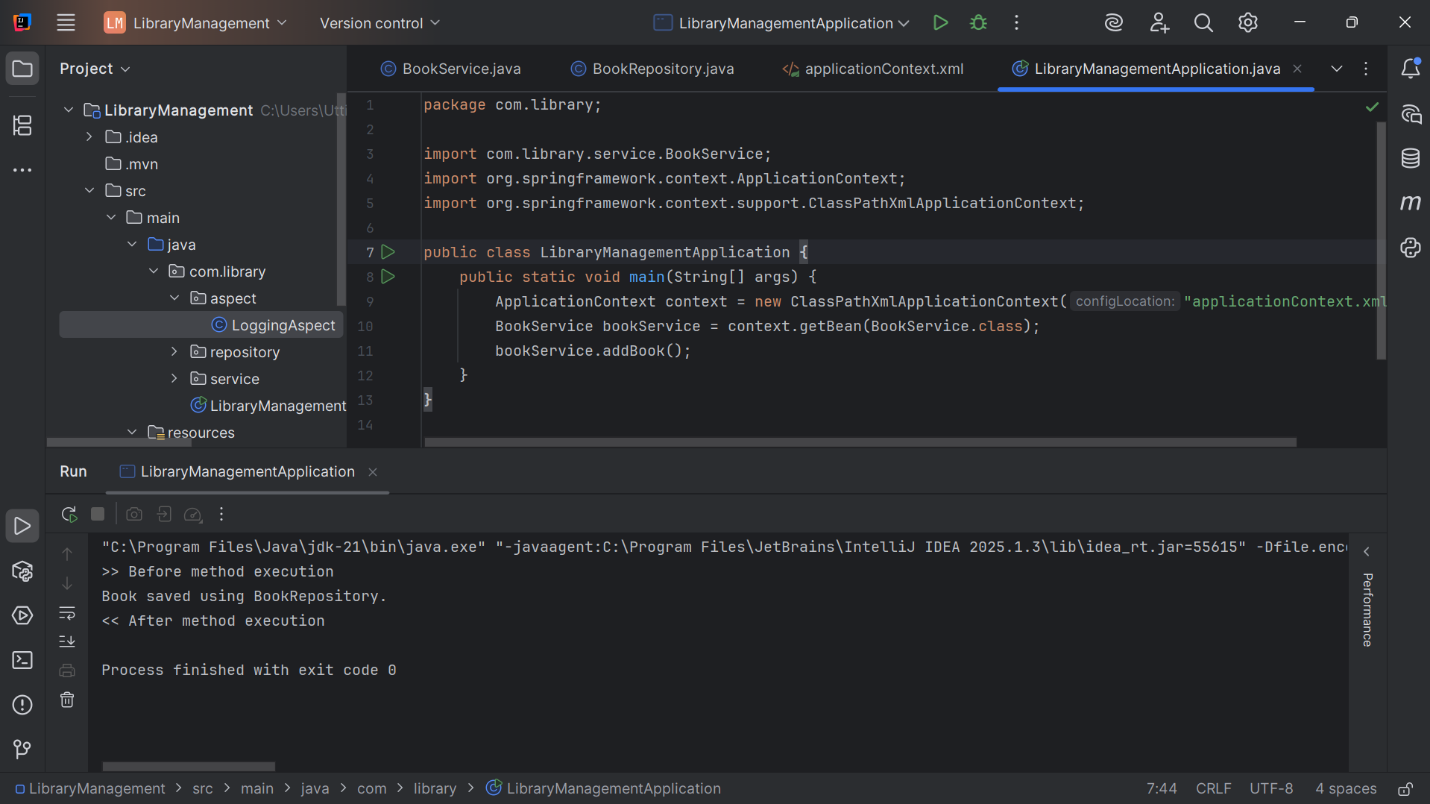
1. **Define an Aspect:**
   * Create a package **com.library.aspect** and add a class **LoggingAspect**.

package com.library.aspect;  
  
import org.aspectj.lang.annotation.After;  
import org.aspectj.lang.annotation.Aspect;  
import org.aspectj.lang.annotation.Before;  
  
@Aspect  
public class LoggingAspect {  
  
 @Before("execution(\* com.library.service.\*.\*(..))")  
 public void logBefore() {  
 System.*out*.println(">> Before method execution");  
 }  
  
 @After("execution(\* com.library.service.\*.\*(..))")  
 public void logAfter() {  
 System.*out*.println("<< After method execution");  
 }  
}

1. **Create Advice Methods:**
   * Define advice methods in **LoggingAspect** for logging before and after method execution.
2. **Configure the Aspect:**
   * Update **applicationContext.xml** to register the aspect and enable **AspectJ** auto-proxying.

<?xml version="1.0" encoding="UTF-8"?>  
<beans xmlns="http://www.springframework.org/schema/beans"  
 xmlns:aop="http://www.springframework.org/schema/aop"  
 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  
 https://www.springframework.org/schema/beans/spring-beans.xsd  
 http://www.springframework.org/schema/context  
 https://www.springframework.org/schema/context/spring-context.xsd  
 http://www.springframework.org/schema/aop  
 https://www.springframework.org/schema/aop/spring-aop.xsd">  
  
 <context:component-scan base-package="com.library"/>  
 <aop:aspectj-autoproxy/>  
  
 <!-- Manual bean registration if not using annotations -->  
 <bean id="bookRepository" class="com.library.repository.BookRepository"/>  
 <bean id="bookService" class="com.library.service.BookService">  
 <property name="bookRepository" ref="bookRepository"/>  
 </bean>  
 <bean class="com.library.aspect.LoggingAspect"/>  
</beans>

1. **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**.

<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  
 https://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>com.library</groupId>  
 <artifactId>LibraryManagementBoot</artifactId>  
 <version>1.0.0</version>  
 <packaging>jar</packaging>  
  
 <name>LibraryManagementBoot</name>  
 <description>Spring Boot Library Management System</description>  
  
 <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>3.2.4</version>  
 <relativePath/>  
 </parent>  
  
 <properties>  
 <java.version>17</java.version>  
 </properties>  
  
 <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>jakarta.persistence</groupId>  
 <artifactId>jakarta.persistence-api</artifactId>  
 <version>3.1.0</version>  
 </dependency>  
  
 </dependencies>  
  
 <build>  
 <plugins>  
  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 </plugins>  
 </build>  
</project>

1. **Create Application Properties:**
   * Configure database connection properties in **application.properties**.

# H2 Database settings  
spring.datasource.url=jdbc:h2:mem:librarydb  
spring.datasource.driver-class-name=org.h2.Driver  
spring.datasource.username=sa  
spring.datasource.password=  
  
# JPA  
spring.jpa.hibernate.ddl-auto=update  
spring.jpa.show-sql=true  
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect  
  
# H2 Console  
spring.h2.console.enabled=true  
spring.h2.console.path=/h2-console

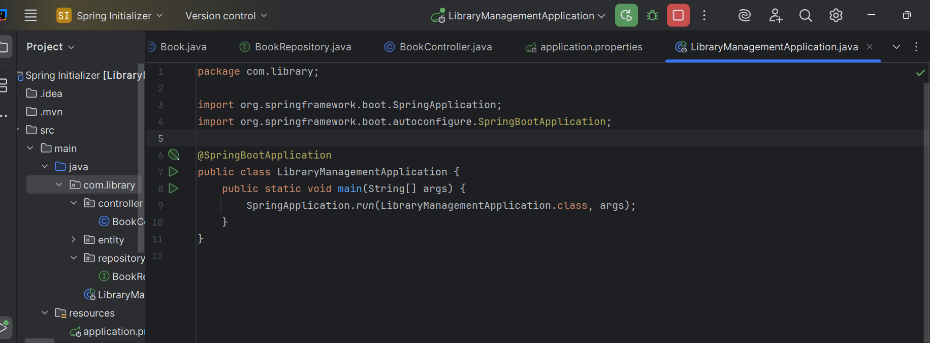
1. **Define Entities and Repositories:**
   * Create **Book** entity and **BookRepository** interface.

package com.library.entity;  
  
import jakarta.persistence.Entity;  
import jakarta.persistence.GeneratedValue;  
import jakarta.persistence.GenerationType;  
import jakarta.persistence.Id;  
  
@Entity  
public class Book {  
  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
  
 private String title;  
  
 // Constructors  
 public Book() {}  
 public Book(String title) {  
 this.title = title;  
 }  
  
 // Getters & Setters  
 public Long getId() { return id; }  
 public String getTitle() { return title; }  
 public void setId(Long id) { this.id = id; }  
 public void setTitle(String title) { this.title = title; }  
}

package com.library.repository;  
  
import com.library.entity.Book;  
import org.springframework.data.jpa.repository.JpaRepository;  
  
public interface BookRepository extends JpaRepository<Book, Long> {}

1. **Create a REST Controller:**
   * Create a **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> getBooks() {  
 return bookRepository.findAll();  
 }  
  
 @PostMapping  
 public Book addBook(@RequestBody Book book) {  
 return bookRepository.save(book);  
 }  
  
 @GetMapping("/{id}")  
 public Book getBook(@PathVariable Long id) {  
 return bookRepository.findById(id).orElse(null);  
 }  
  
 @DeleteMapping("/{id}")  
 public void deleteBook(@PathVariable Long id) {  
 bookRepository.deleteById(id);  
 }  
}

1. **Run the Application:**
   * Run the Spring Boot application and test the REST endpoints.

