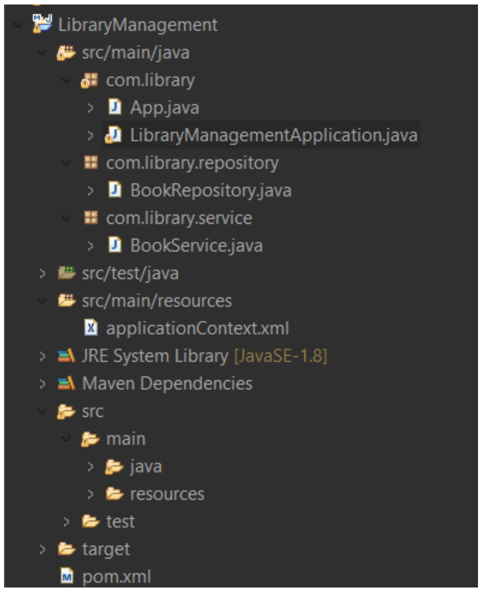
Exercise 1: Configuring a Basic Spring Application

Scenario:

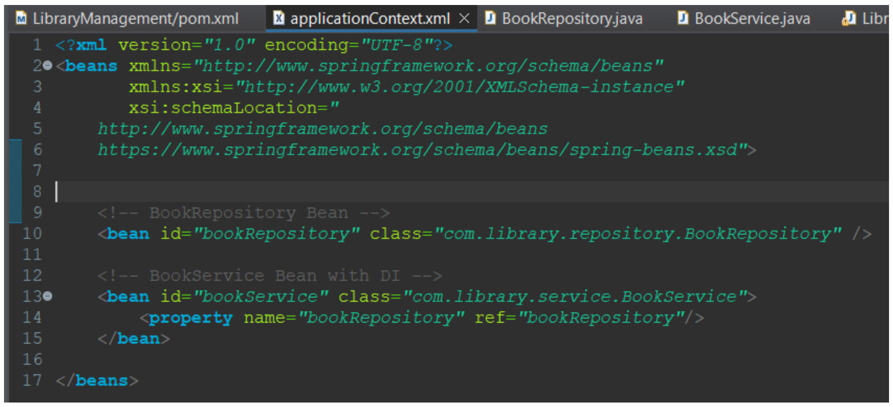
Your company is developing a web application for managing a library. You need to use the Spring Framework to handle the backend operations.

Steps:

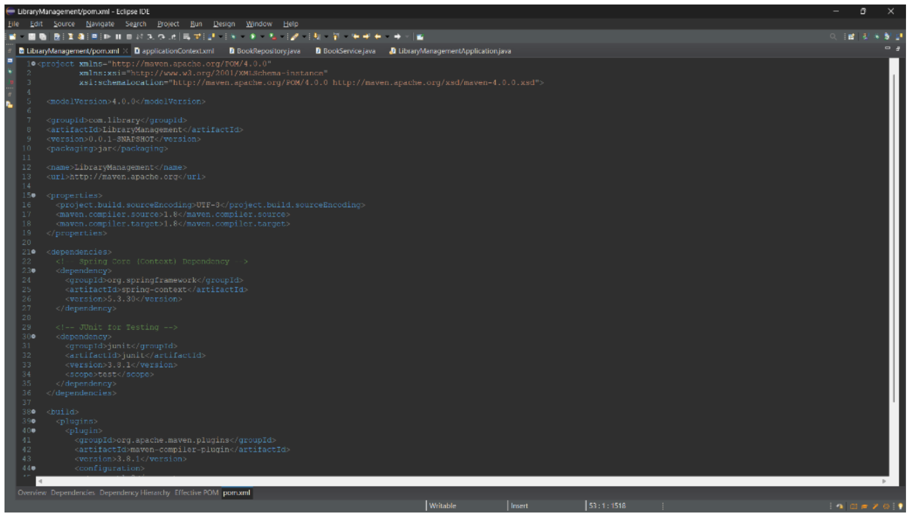
1. **Set Up a Spring Project:**
   - Create a Maven project named **LibraryManagement**.
   - Add Spring Core dependencies in the **pom.xml** file.
2. **Configure the Application Context:**
   - Create an XML configuration file named **applicationContext.xml** in the **src/main/resources** directory.
   - 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.



```
LibraryManagement
  src/main/java
    com.library
      App.java
      LibraryManagementApplication.java
    com.library.repository
      BookRepository.java
    com.library.service
      BookService.java
  src/test/java
  src/main/resources
    applicationContext.xml
  JRE System Library [JavaSE-1.8]
  Maven Dependencies
  src
    main
      java
      resources
    test
  target
  pom.xml
```

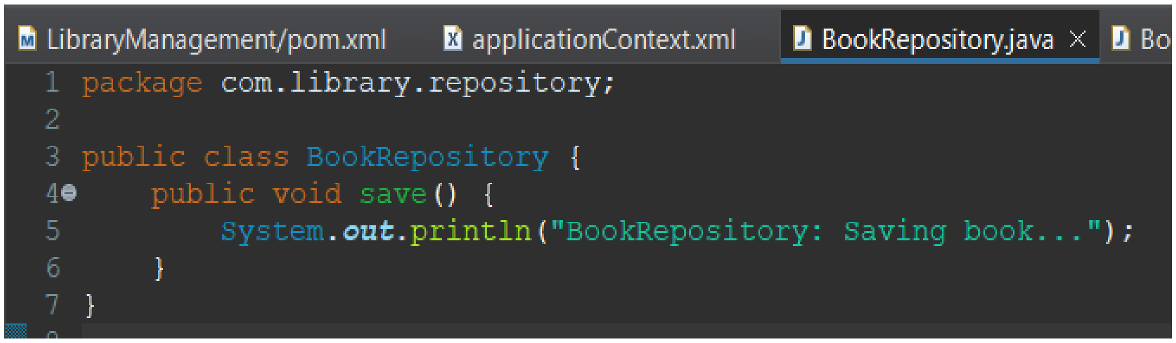


```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
    http://www.springframework.org/schema/beans
    https://www.springframework.org/schema/beans/spring-beans.xsd">


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

    <!-- BookService Bean with DI -->
    <bean id="bookService" class="com.library.service.BookService">
        <property name="bookRepository" ref="bookRepository"/>
    </bean>

</beans>
```



LibraryManagement/pom.xml - Eclipse IDE
File Edit Source Navigate Search Project Run Design Window Help
LibraryManagement/pom.xml
applicationContext.xml
BookRepository.java
BookService.java
LibraryManagementApplication.java
Overview Dependencies Dependency Hierarchy Effective POM pom.xml
Writable
Insert
53 : 1 : 1518

```xml
<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>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>LibraryManagement</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.8</maven.compiler.source>
    <maven.compiler.target>1.8</maven.compiler.target>
  </properties>

  <dependencies>
    <!-- Spring Core (Context) Dependency -->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>5.3.30</version>
    </dependency>

    <!-- JUnit for Testing -->
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>

  <build>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.8.1</version>
        <configuration>
```



```java
package com.library.repository;

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

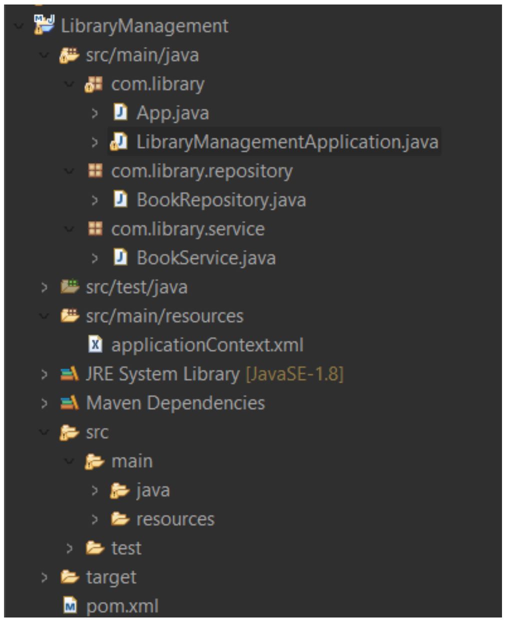
Exercise 2: Implementing Dependency Injection

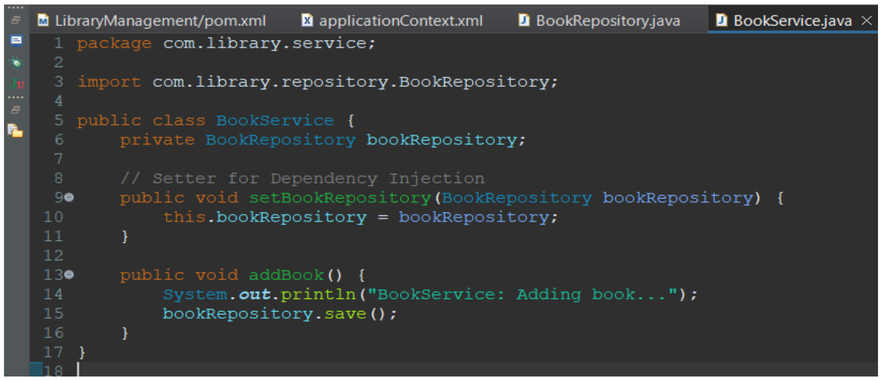
Scenario:

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

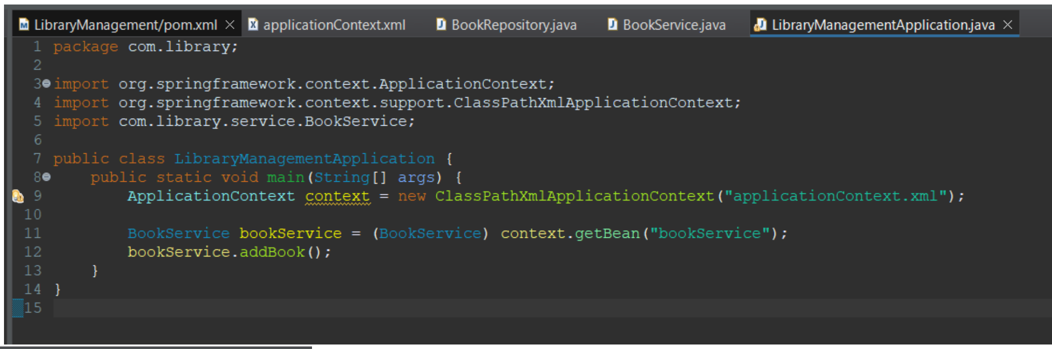
Steps:

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.

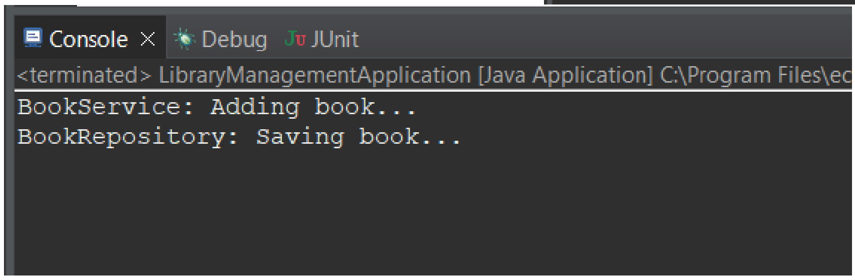




```java
package com.library.service;

import com.library.repository.BookRepository;

public class BookService {
    private BookRepository bookRepository;

    // Setter for Dependency Injection
    public void setBookRepository(BookRepository bookRepository) {
        this.bookRepository = bookRepository;
    }

    public void addBook() {
        System.out.println("BookService: Adding book...");
        bookRepository.save();
    }
}
```



```java
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");
        bookService.addBook();
    }
}
```



```
<terminated> LibraryManagementApplication [Java Application] C:\Program Files\ec
BookService: Adding book...
BookRepository: Saving book...
```

Exercise 3: Implementing Logging with Spring AOP

Scenario:

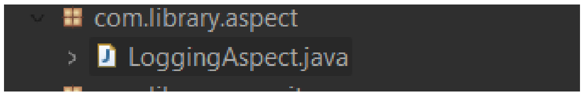
The library management application requires logging capabilities to track method execution times.

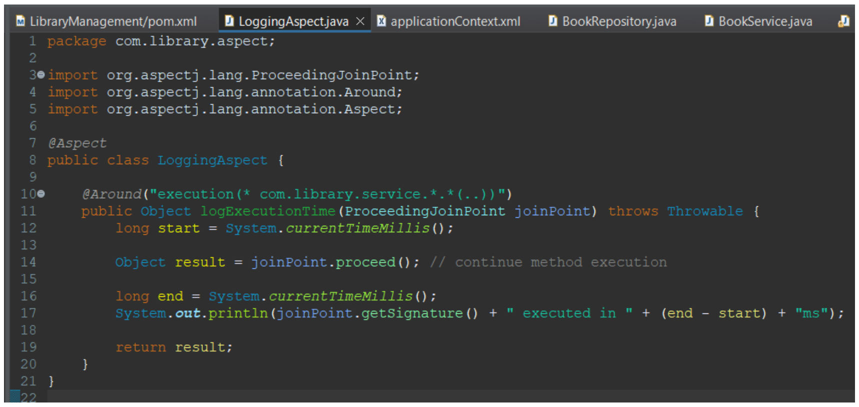
Steps:

1. **Add Spring AOP Dependency:**
   - Update **pom.xml** to include Spring AOP dependency.

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

<!-- AspectJ Weaver -->
<dependency>
  <groupId>org.aspectj</groupId>
  <artifactId>aspectjweaver</artifactId>
  <version>1.9.21.1</version>
</dependency>
```

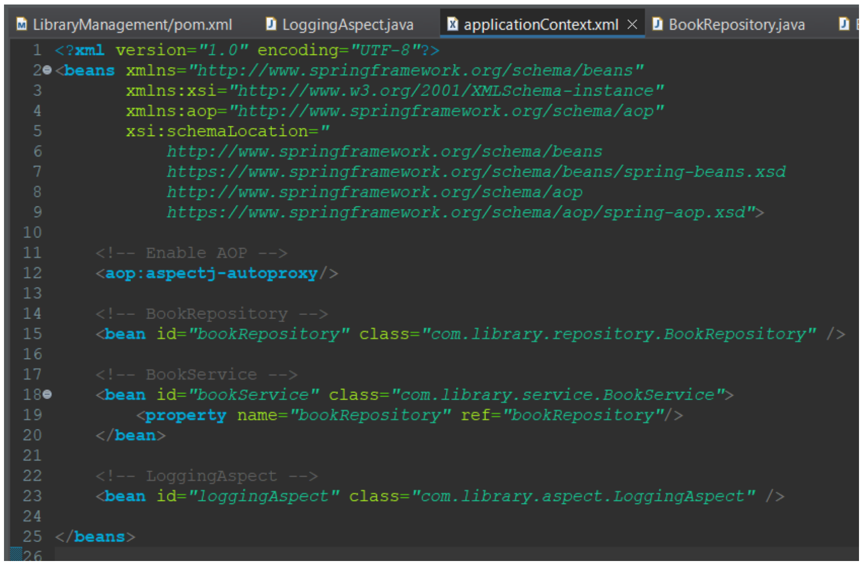
2. **Create an Aspect for Logging:**
   - Create a package **com.library.aspect** and add a class **LoggingAspect** with a method to log execution times.





```java
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(); // continue method execution

        long end = System.currentTimeMillis();
        System.out.println(joinPoint.getSignature() + " executed in " + (end - start) + "ms");

        return result;
    }
}
```

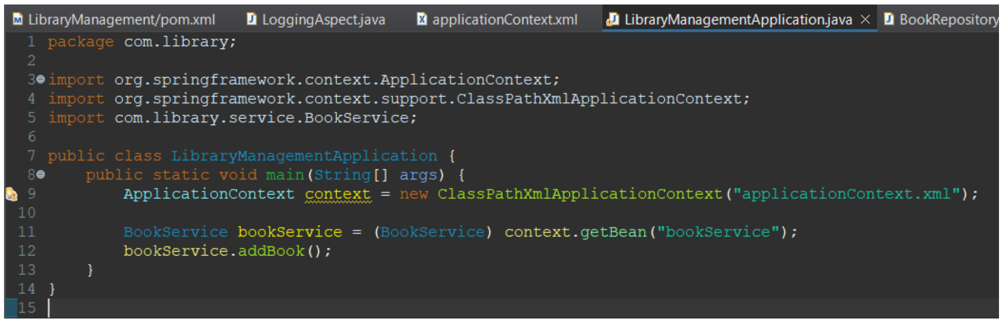
3. **Enable AspectJ Support:**
   - Update **applicationContext.xml** to enable **AspectJ** support and register the aspect.



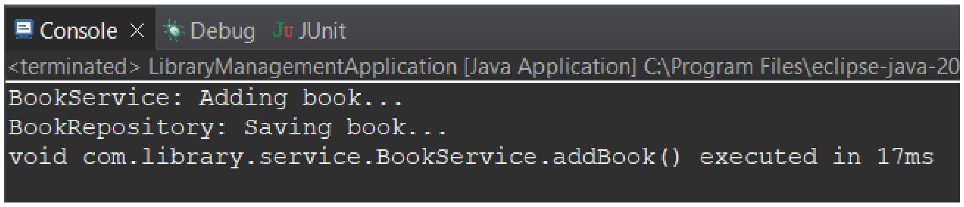
```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xsi:schemaLocation="
           http://www.springframework.org/schema/beans
           https://www.springframework.org/schema/beans/spring-beans.xsd
           http://www.springframework.org/schema/aop
           https://www.springframework.org/schema/aop/spring-aop.xsd">

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

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

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

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

</beans>
```

4. **Test the Aspect:**

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



```java
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");
        bookService.addBook();
    }
}
```



```
<terminated> LibraryManagementApplication [Java Application] C:\Program Files\eclipse-java-20
BookService: Adding book...
BookRepository: Saving book...
void com.library.service.BookService.addBook() executed in 17ms
```

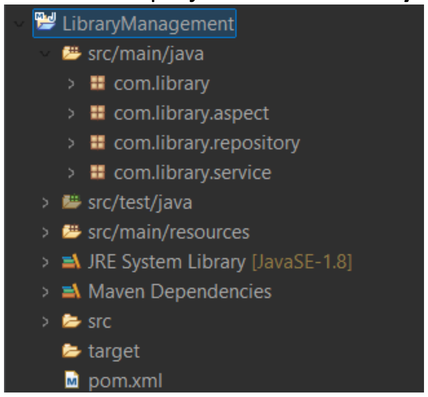
Exercise 4: Creating and Configuring a Maven Project

Scenario:

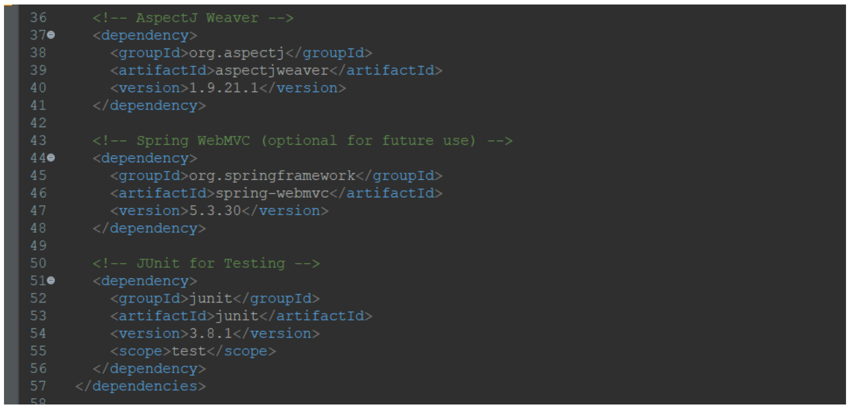
You need to set up a new Maven project for the library management application and add Spring dependencies.

Steps:

1. **Create a New Maven Project:**
   - Create a new Maven project named **LibraryManagement**.

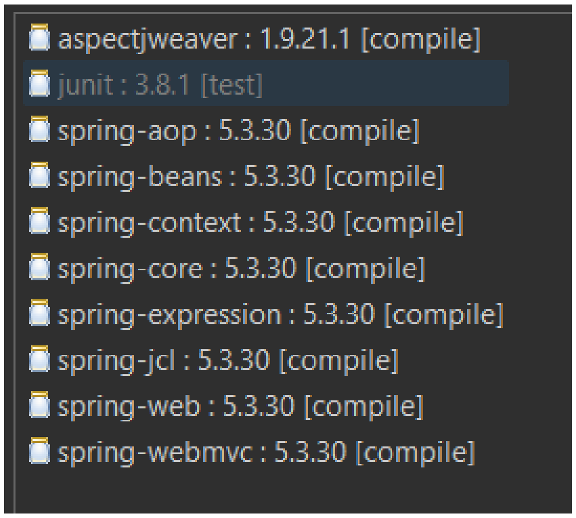


2. **Add Spring Dependencies in pom.xml:**
   - Include dependencies for Spring Context, Spring AOP, and Spring WebMVC.



```xml
    <!-- AspectJ Weaver -->
    <dependency>
      <groupId>org.aspectj</groupId>
      <artifactId>aspectjweaver</artifactId>
      <version>1.9.21.1</version>
    </dependency>

    <!-- Spring WebMVC (optional for future use) -->
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
      <version>5.3.30</version>
    </dependency>

    <!-- JUnit for Testing -->
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>
  </dependencies>
```

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



```
aspectjweaver : 1.9.21.1 [compile]
junit : 3.8.1 [test]
spring-aop : 5.3.30 [compile]
spring-beans : 5.3.30 [compile]
spring-context : 5.3.30 [compile]
spring-core : 5.3.30 [compile]
spring-expression : 5.3.30 [compile]
spring-jcl : 5.3.30 [compile]
spring-web : 5.3.30 [compile]
spring-webmvc : 5.3.30 [compile]
```

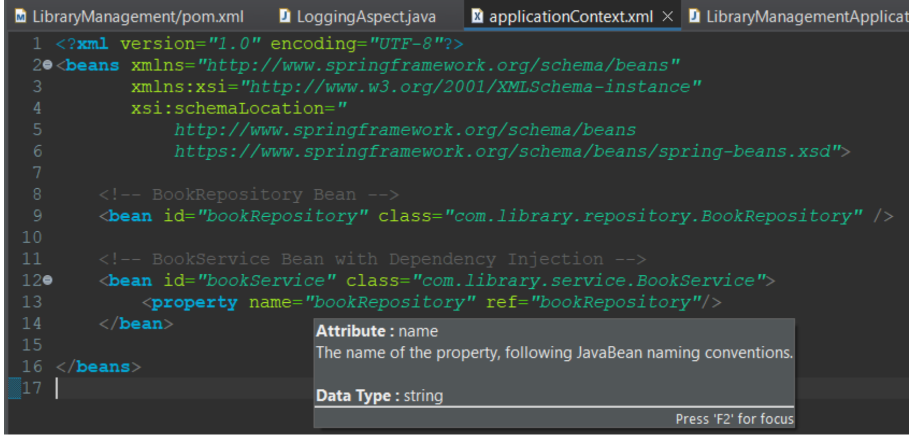
Exercise 5: Configuring the Spring IoC Container

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.



LibraryManagement/pom.xml | LoggingAspect.java | applicationContext.xml | LibraryManagementApplicat

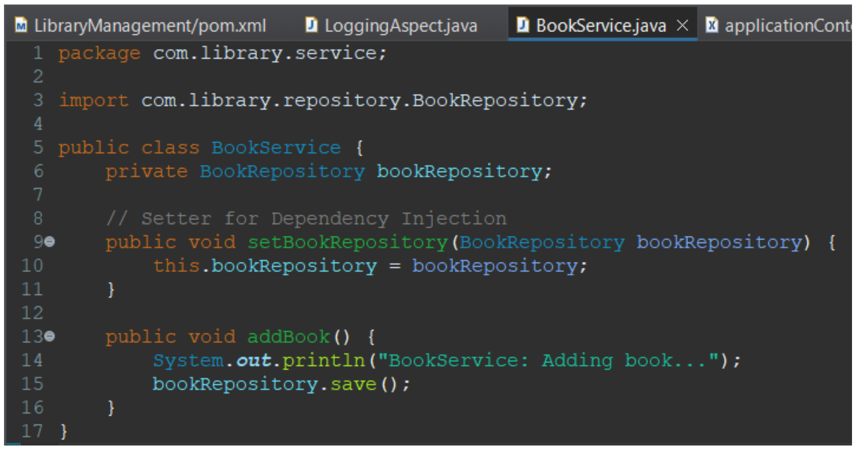
```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
           http://www.springframework.org/schema/beans
           https://www.springframework.org/schema/beans/spring-beans.xsd">

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

    <!-- BookService Bean with Dependency Injection -->
    <bean id="bookService" class="com.library.service.BookService">
        <property name="bookRepository" ref="bookRepository"/>
    </bean>

</beans>
```

Attribute : name
The name of the property, following JavaBean naming conventions.

Data Type : string

Press 'F2' for focus

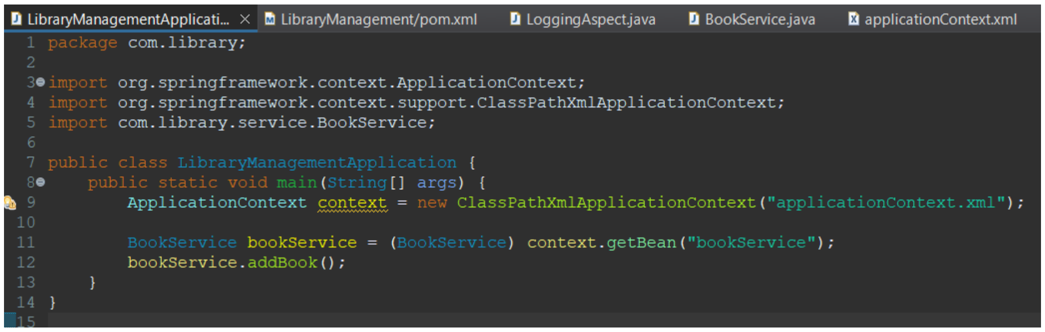
2. **Update the BookService Class:**
   - Ensure that the **BookService** class has a setter method for **BookRepository**.



LibraryManagement/pom.xml | LoggingAspect.java | BookService.java | applicationCont

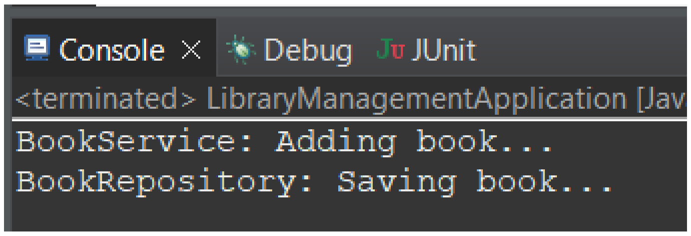
```java
package com.library.service;

import com.library.repository.BookRepository;

public class BookService {
    private BookRepository bookRepository;

    // Setter for Dependency Injection
    public void setBookRepository(BookRepository bookRepository) {
        this.bookRepository = bookRepository;
    }

    public void addBook() {
        System.out.println("BookService: Adding book...");
        bookRepository.save();
    }
}
```

3. **Run the Application:**
   - Create a main class to load the Spring context and test the configuration.



LibraryManagementApplicati... | LibraryManagement/pom.xml | LoggingAspect.java | BookService.java | applicationContext.xml

```java
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");
        bookService.addBook();
    }
}
```



Console | Debug | JUnit

```
<terminated> LibraryManagementApplication [Jav
BookService: Adding book...
BookRepository: Saving book...
```

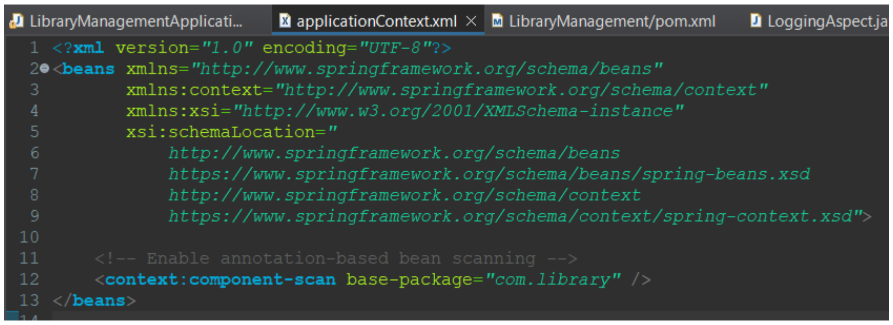
Exercise 6: Configuring Beans with Annotations

Scenario:

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

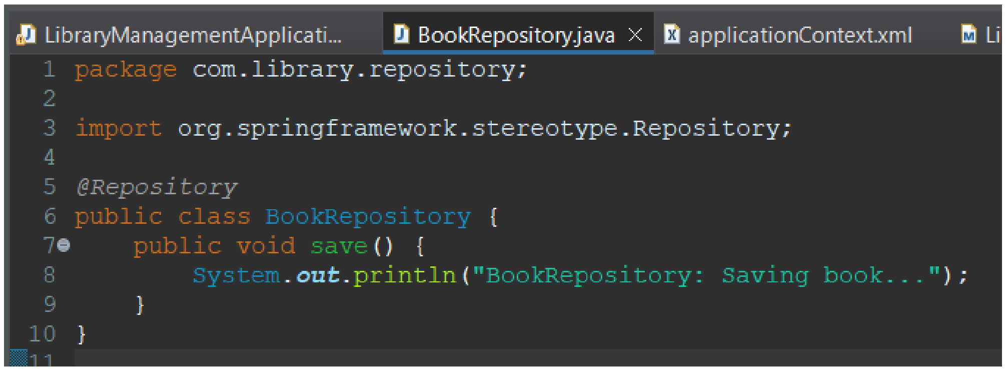
Steps:

1. **Enable Component Scanning:**
   - Update **applicationContext.xml** to include component scanning for the **com.library** package.

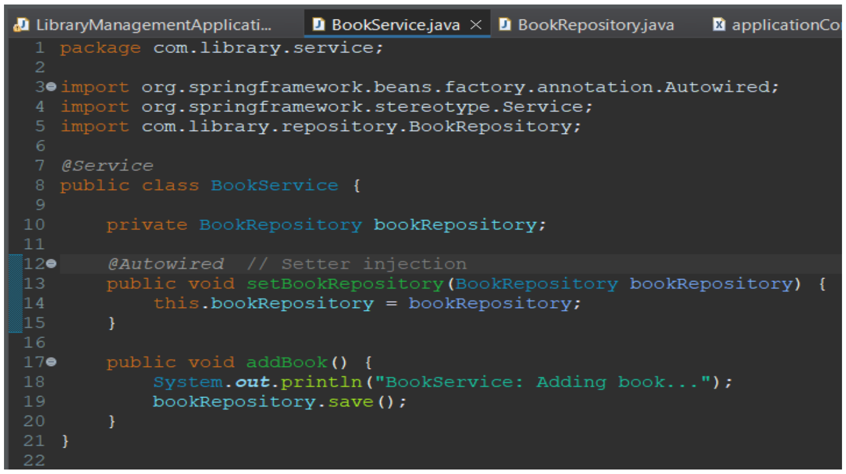


```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
           http://www.springframework.org/schema/beans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

    <!-- Enable annotation-based bean scanning -->
    <context:component-scan base-package="com.library" />
</beans>
```

2. **Annotate Classes:**
   - Use **@Service** annotation for the **BookService** class.
   - Use **@Repository** annotation for the **BookRepository** class.

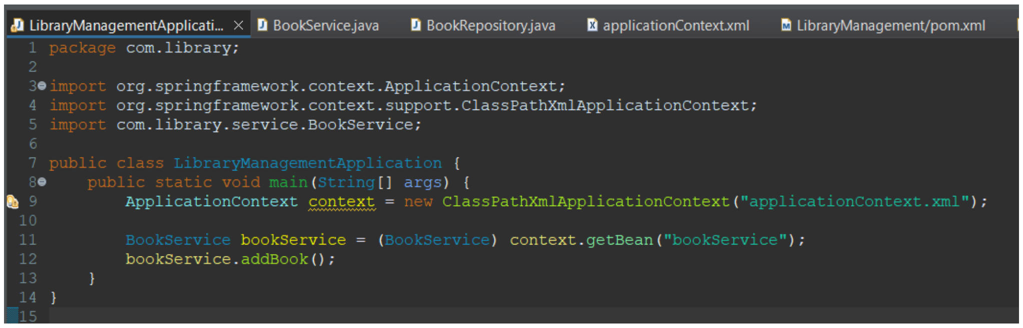


```java
package com.library.repository;

import org.springframework.stereotype.Repository;

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



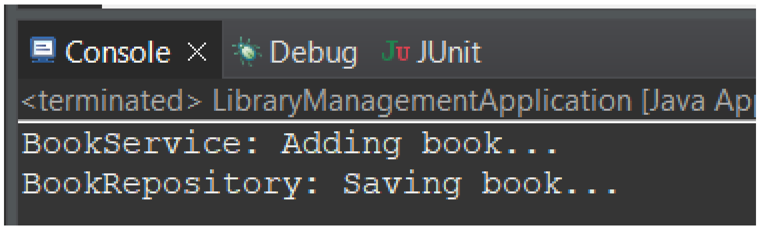
```java
package com.library.service;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import com.library.repository.BookRepository;

@Service
public class BookService {

    private BookRepository bookRepository;

    @Autowired  // Setter injection
    public void setBookRepository(BookRepository bookRepository) {
        this.bookRepository = bookRepository;
    }

    public void addBook() {
        System.out.println("BookService: Adding book...");
        bookRepository.save();
    }
}
```

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



LibraryManagementApplicati... × BookService.java BookRepository.java applicationContext.xml LibraryManagement/pom.xml

```java
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");
        bookService.addBook();
    }
}
```



```
<terminated> LibraryManagementApplication [Java Ap
BookService: Adding book...
BookRepository: Saving book...
```

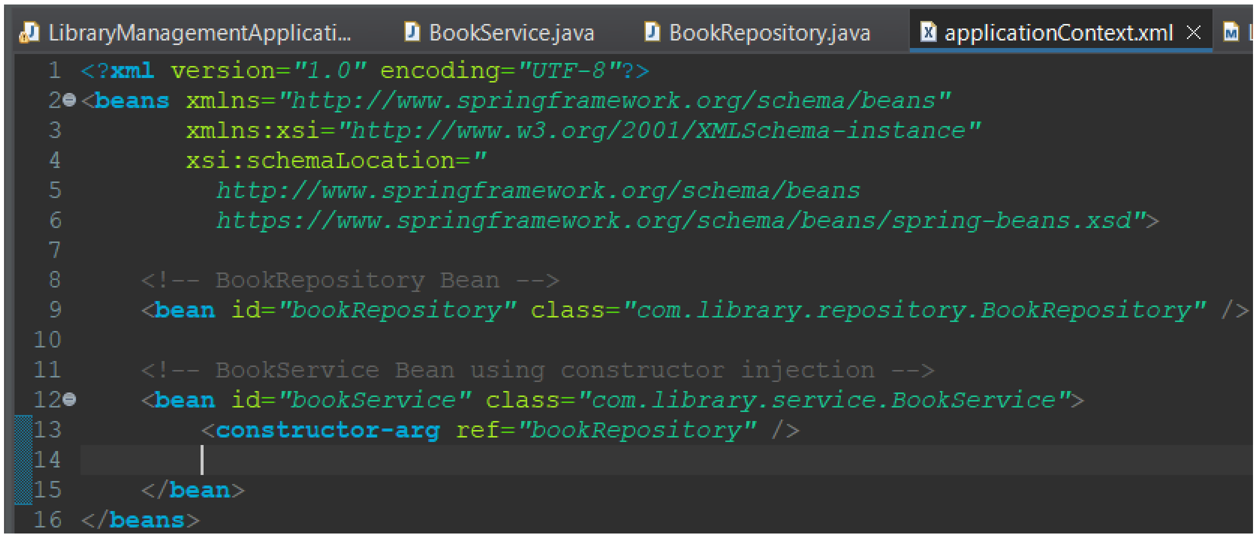
Exercise 7: Implementing Constructor and Setter Injection

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**.

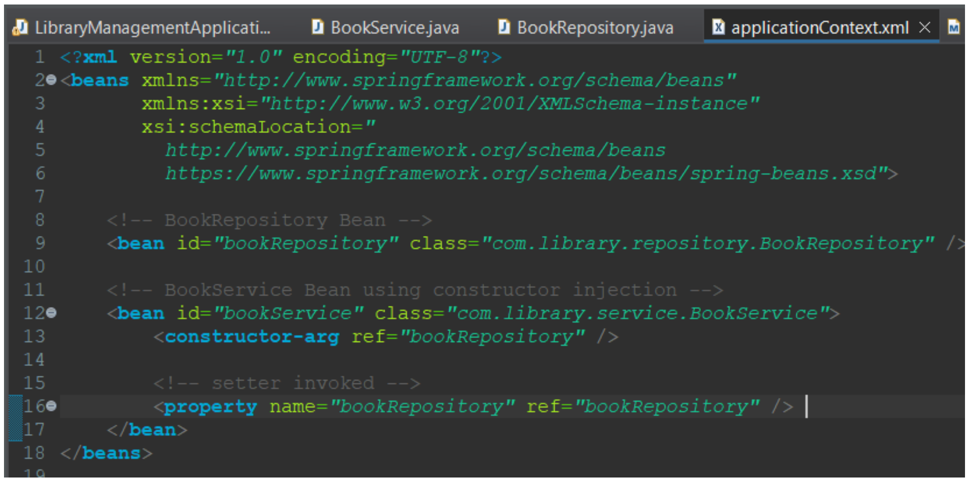


LibraryManagementApplicati... BookService.java BookRepository.java applicationContext.xml ×

```xml
<?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">

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

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

    </bean>
</beans>
```

2. **Configure Setter Injection:**

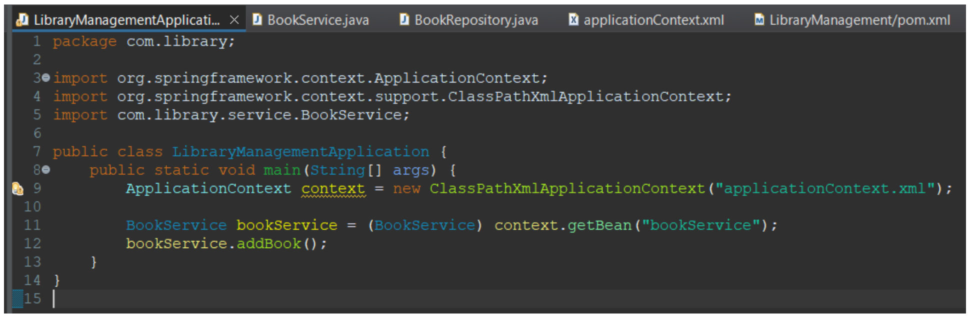
    ○ Ensure that the **BookService** class has a setter method for **BookRepository** and configure it in **applicationContext.xml**.



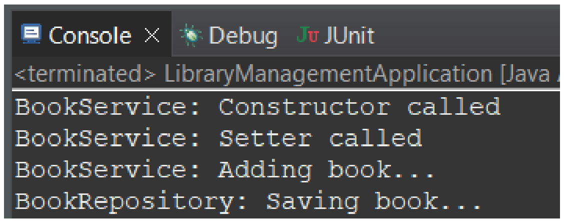
```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
         http://www.springframework.org/schema/beans
         https://www.springframework.org/schema/beans/spring-beans.xsd">

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

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

        <!-- setter invoked -->
        <property name="bookRepository" ref="bookRepository" />
    </bean>
</beans>
```

3. **Test the Injection:**

    ○ Run the **LibraryManagementApplication** main class to verify both constructor and setter injection.



```java
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");
        bookService.addBook();
    }
}
```



```
<terminated> LibraryManagementApplication [Java
BookService: Constructor called
BookService: Setter called
BookService: Adding book...
BookRepository: Saving book...
```

Exercise 8: Implementing Basic AOP with Spring

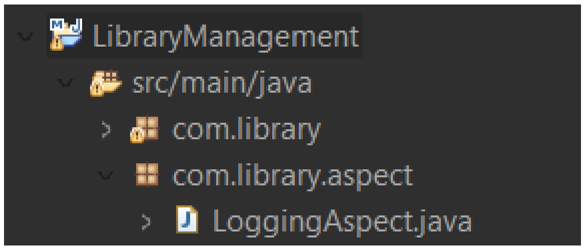
Scenario:

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

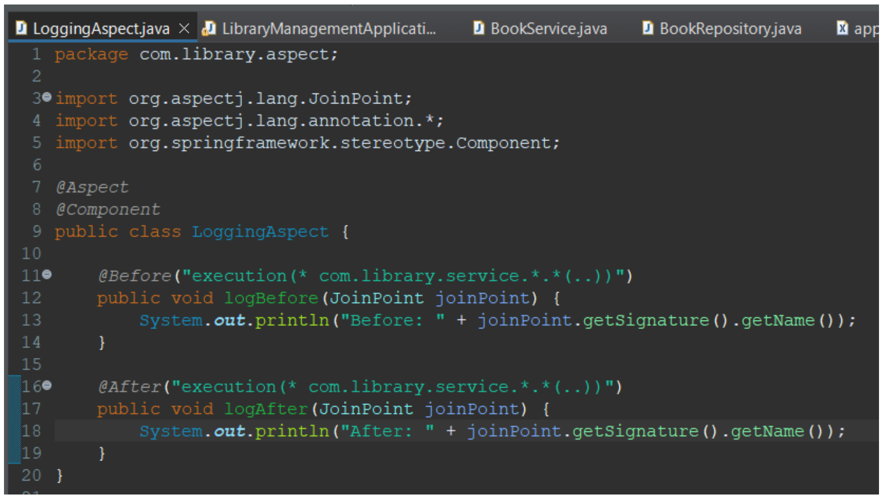
Steps:

1. **Define an Aspect:**

    ○ Create a package **com.library.aspect** and add a class **LoggingAspect**

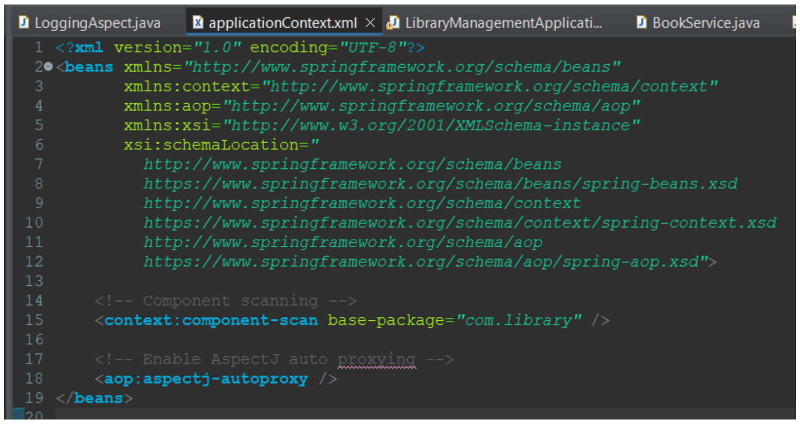


2. **Create Advice Methods:**
   - Define advice methods in **LoggingAspect** for logging before and after method execution.



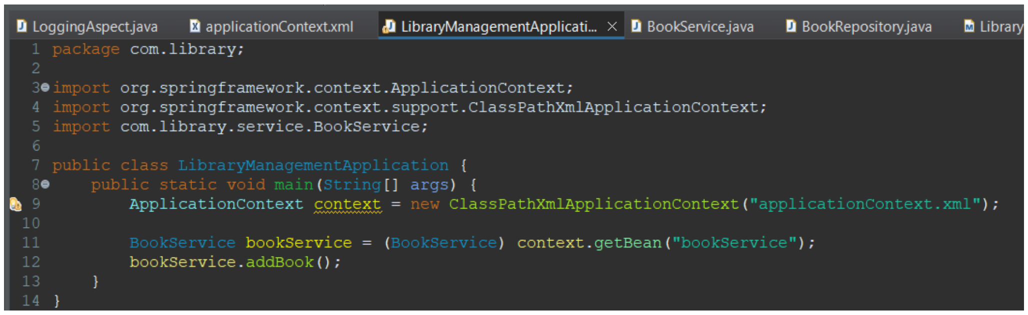
```java
package com.library.aspect;

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

@Aspect
@Component
public class LoggingAspect {

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

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

3. **Configure the Aspect:**
   - Update **applicationContext.xml** to register the aspect and enable **AspectJ** auto-proxying.

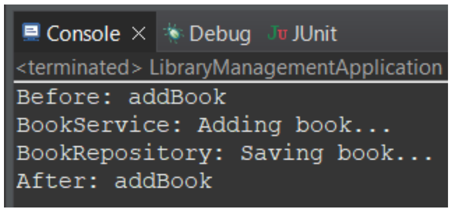


```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:context="http://www.springframework.org/schema/context"
       xmlns:aop="http://www.springframework.org/schema/aop"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
         http://www.springframework.org/schema/beans
         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">

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

    <!-- Enable AspectJ auto proxying -->
    <aop:aspectj-autoproxy />
</beans>
```

4. **Test the Aspect:**
   - Run the **LibraryManagementApplication** main class to verify the AOP functionality.



```java
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");
        bookService.addBook();
    }
}
```



```
<terminated> LibraryManagementApplication
Before: addBook
BookService: Adding book...
BookRepository: Saving book...
After: addBook
```

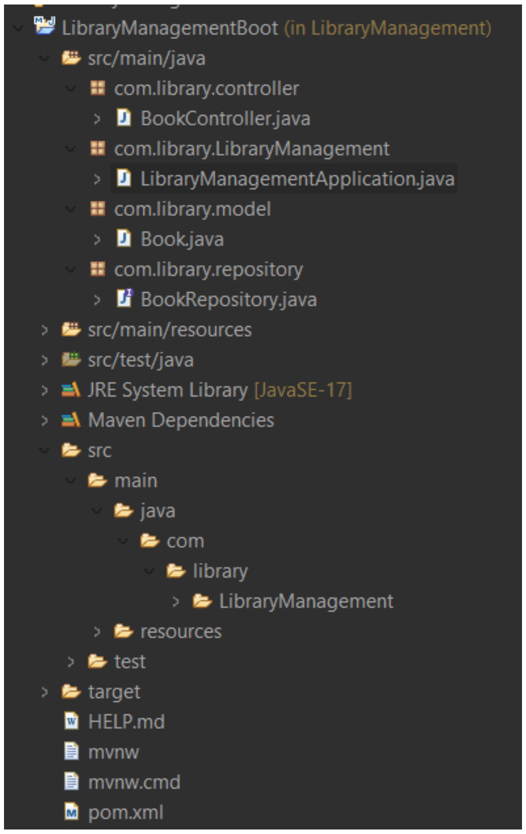
Exercise 9: Creating a Spring Boot Application

Scenario:

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

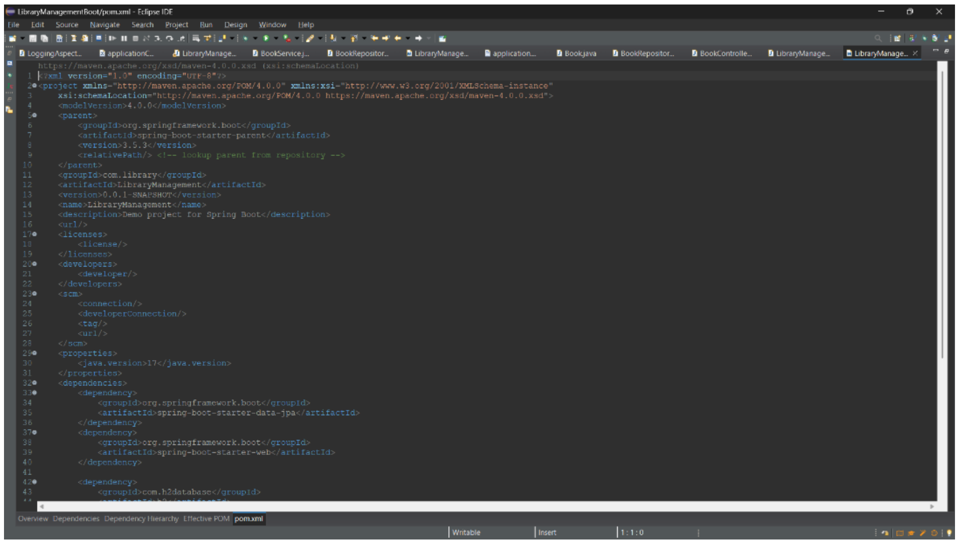
Steps:

1. **Create a Spring Boot Project:**
   - Use **Spring Initializr** to create a new Spring Boot project named **LibraryManagement**.



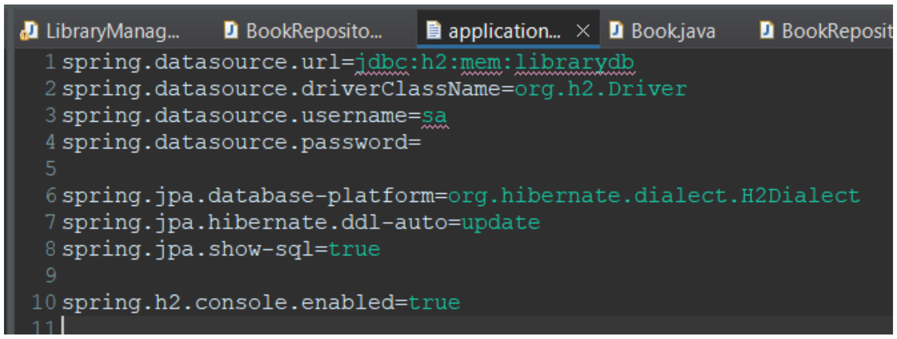
```
LibraryManagementBoot (in LibraryManagement)
  src/main/java
    com.library.controller
      BookController.java
    com.library.LibraryManagement
      LibraryManagementApplication.java
    com.library.model
      Book.java
    com.library.repository
      BookRepository.java
  src/main/resources
  src/test/java
  JRE System Library [JavaSE-17]
  Maven Dependencies
  src
    main
      java
        com
          library
            LibraryManagement
      resources
    test
  target
  HELP.md
  mvnw
  mvnw.cmd
  pom.xml
```

2. **Add Dependencies:**
   - Include dependencies for **Spring Web, Spring Data JPA, and H2 Database**.



```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 https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>3.5.3</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.library</groupId>
    <artifactId>LibraryManagement</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>LibraryManagement</name>
    <description>Demo project for Spring Boot</description>
    <url/>
    <licenses>
        <license/>
    </licenses>
    <developers>
        <developer/>
    </developers>
    <scm>
        <connection/>
        <developerConnection/>
        <tag/>
        <url/>
    </scm>
    <properties>
        <java.version>17</java.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
        </dependency>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>

        <dependency>
            <groupId>com.h2database</groupId>
```

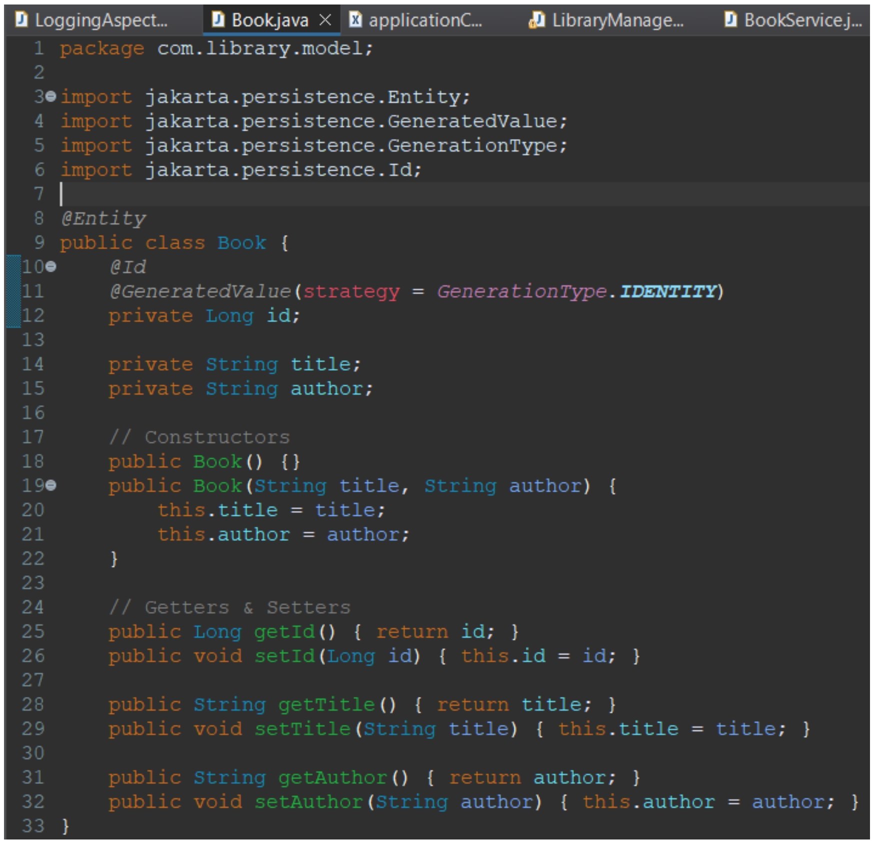
3. **Create Application Properties:**
   - Configure database connection properties in **application.properties**.



LibraryManag... BookReposito... application... Book.java BookReposit

```
spring.datasource.url=jdbc:h2:mem:librarydb
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true

spring.h2.console.enabled=true

```

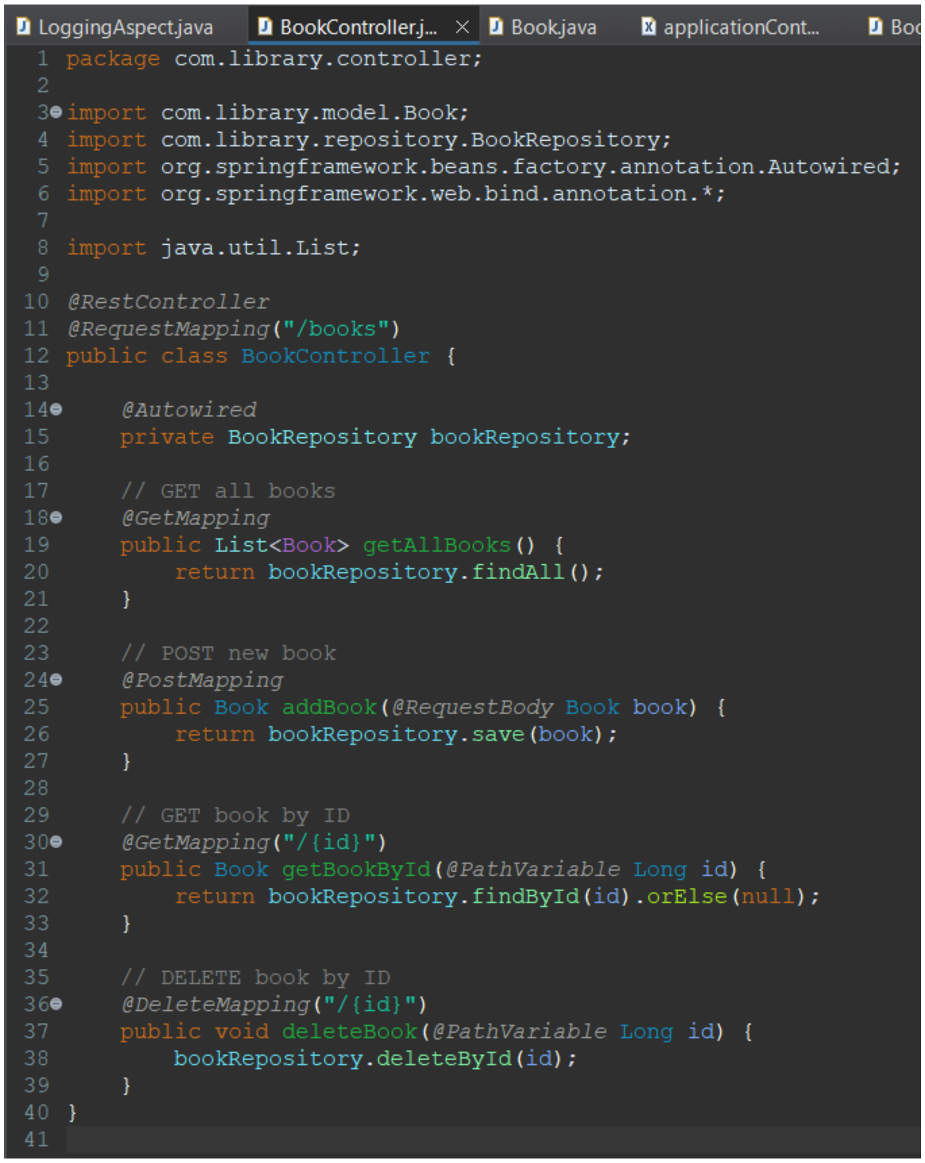
4. **Define Entities and Repositories:**
   - Create **Book** entity and **BookRepository** interface.



LoggingAspect... Book.java applicationC... LibraryManage... BookService.j...

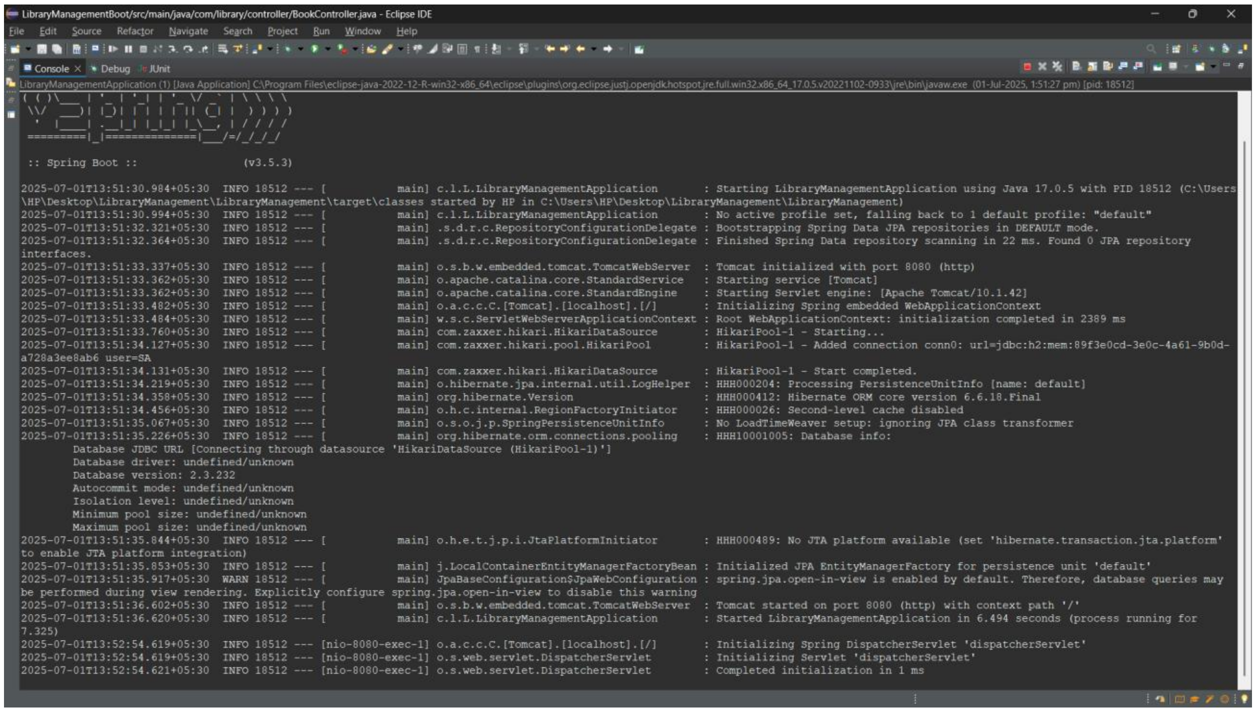
```java
package com.library.model;

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;
    private String author;

    // Constructors
    public Book() {}
    public Book(String title, String author) {
        this.title = title;
        this.author = author;
    }

    // Getters & Setters
    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }

    public String getTitle() { return title; }
    public void setTitle(String title) { this.title = title; }

    public String getAuthor() { return author; }
    public void setAuthor(String author) { this.author = author; }
}
```

5. **Create a REST Controller:**
   - Create a **BookController** class to handle CRUD operations.



```java
package com.library.controller;

import com.library.model.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;

    // GET all books
    @GetMapping
    public List<Book> getAllBooks() {
        return bookRepository.findAll();
    }

    // POST new book
    @PostMapping
    public Book addBook(@RequestBody Book book) {
        return bookRepository.save(book);
    }

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

    // DELETE book by ID
    @DeleteMapping("/{id}")
    public void deleteBook(@PathVariable Long id) {
        bookRepository.deleteById(id);
    }
}

```

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


LibraryManagementBoot/src/main/java/com/library/controller/BookController.java - Eclipse IDE
File
Edit
Source
Refactor
Navigate
Search
Project
Run
Window
Help
Console
Debug
JUnit

```
LibraryManagementApplication (1) [Java Application] C:\Program Files\eclipse-java-2022-12-R-win32-x86_64\eclipse\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.5.v20221102-0933\jre\bin\javaw.exe  (01-Jul-2025, 1:51:27 pm) [pid: 18512]
( ( )\___ | '_ | '_| | '_ \/ _` | \ \ \ \
 \\/  ___)| |_)| | | | | || (_| |  ) ) ) )
  '  |____| .__|_| |_|_| |_\__, | / / / /
 =========|_|==============|___/=/_/_/_/

 :: Spring Boot ::                (v3.5.3)

2025-07-01T13:51:30.984+05:30  INFO 18512 --- [           main] c.l.L.LibraryManagementApplication       : Starting LibraryManagementApplication using Java 17.0.5 with PID 18512 (C:\Users
\HP\Desktop\LibraryManagement\LibraryManagement\target\classes started by HP in C:\Users\HP\Desktop\LibraryManagement\LibraryManagement)
2025-07-01T13:51:30.994+05:30  INFO 18512 --- [           main] c.l.L.LibraryManagementApplication       : No active profile set, falling back to 1 default profile: "default"
2025-07-01T13:51:32.321+05:30  INFO 18512 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2025-07-01T13:51:32.364+05:30  INFO 18512 --- [           main] .s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 22 ms. Found 0 JPA repository
interfaces.
2025-07-01T13:51:33.337+05:30  INFO 18512 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat initialized with port 8080 (http)
2025-07-01T13:51:33.362+05:30  INFO 18512 --- [           main] o.apache.catalina.core.StandardService   : Starting service [Tomcat]
2025-07-01T13:51:33.362+05:30  INFO 18512 --- [           main] o.apache.catalina.core.StandardEngine    : Starting Servlet engine: [Apache Tomcat/10.1.42]
2025-07-01T13:51:33.482+05:30  INFO 18512 --- [           main] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring embedded WebApplicationContext
2025-07-01T13:51:33.484+05:30  INFO 18512 --- [           main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 2389 ms
2025-07-01T13:51:33.760+05:30  INFO 18512 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Starting...
2025-07-01T13:51:34.127+05:30  INFO 18512 --- [           main] com.zaxxer.hikari.pool.HikariPool        : HikariPool-1 - Added connection conn0: url=jdbc:h2:mem:89f3e0cd-3e0c-4a61-9b0d-
a728a3ee8ab6 user=SA
2025-07-01T13:51:34.131+05:30  INFO 18512 --- [           main] com.zaxxer.hikari.HikariDataSource       : HikariPool-1 - Start completed.
2025-07-01T13:51:34.219+05:30  INFO 18512 --- [           main] o.hibernate.jpa.internal.util.LogHelper  : HHH000204: Processing PersistenceUnitInfo [name: default]
2025-07-01T13:51:34.358+05:30  INFO 18512 --- [           main] org.hibernate.Version                    : HHH000412: Hibernate ORM core version 6.6.18.Final
2025-07-01T13:51:34.456+05:30  INFO 18512 --- [           main] o.h.c.internal.RegionFactoryInitiator    : HHH000026: Second-level cache disabled
2025-07-01T13:51:35.067+05:30  INFO 18512 --- [           main] o.s.o.j.p.SpringPersistenceUnitInfo      : No LoadTimeWeaver setup: ignoring JPA class transformer
2025-07-01T13:51:35.226+05:30  INFO 18512 --- [           main] org.hibernate.orm.connections.pooling    : HHH10001005: Database info:
	Database JDBC URL [Connecting through datasource 'HikariDataSource (HikariPool-1)']
	Database driver: undefined/unknown
	Database version: 2.3.232
	Autocommit mode: undefined/unknown
	Isolation level: undefined/unknown
	Minimum pool size: undefined/unknown
	Maximum pool size: undefined/unknown
2025-07-01T13:51:35.844+05:30  INFO 18512 --- [           main] o.h.e.t.j.p.i.JtaPlatformInitiator       : HHH000489: No JTA platform available (set 'hibernate.transaction.jta.platform'
to enable JTA platform integration)
2025-07-01T13:51:35.853+05:30  INFO 18512 --- [           main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2025-07-01T13:51:35.917+05:30  WARN 18512 --- [           main] JpaBaseConfiguration$JpaWebConfiguration : spring.jpa.open-in-view is enabled by default. Therefore, database queries may
be performed during view rendering. Explicitly configure spring.jpa.open-in-view to disable this warning
2025-07-01T13:51:36.602+05:30  INFO 18512 --- [           main] o.s.b.w.embedded.tomcat.TomcatWebServer  : Tomcat started on port 8080 (http) with context path '/'
2025-07-01T13:51:36.620+05:30  INFO 18512 --- [           main] c.l.L.LibraryManagementApplication       : Started LibraryManagementApplication in 6.494 seconds (process running for
7.325)
2025-07-01T13:52:54.619+05:30  INFO 18512 --- [nio-8080-exec-1] o.a.c.c.C.[Tomcat].[localhost].[/]       : Initializing Spring DispatcherServlet 'dispatcherServlet'
2025-07-01T13:52:54.619+05:30  INFO 18512 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Initializing Servlet 'dispatcherServlet'
2025-07-01T13:52:54.621+05:30  INFO 18512 --- [nio-8080-exec-1] o.s.web.servlet.DispatcherServlet        : Completed initialization in 1 ms
```

<u>Submitted By:</u>

Name : Lingaraj Nayak

Superset ID : 6387607