**WEEK - 3**

Module 5 - Spring Core and Maven

**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>com.library</groupId>  
 <artifactId>LibraryManagement</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <properties>  
 <maven.compiler.source>24</maven.compiler.source>  
 <maven.compiler.target>24</maven.compiler.target>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
  
 <dependencies>  
 <!-- Spring Core (Spring Context includes Beans and Core) -->  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>5.3.36</version>  
 </dependency>  
 </dependencies>  
  
</project>

applicationContext.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 (Injects bookRepository) -->  
 <bean id="bookService" class="com.library.service.BookService">  
 <property name="bookRepository" ref="bookRepository" />  
 </bean>  
</beans>

BookRepository.java

package com.library.repository;  
  
public class BookRepository {  
 public void getBooks() {  
 System.*out*.println("BookRepository: Getting all books from database");  
 }  
}

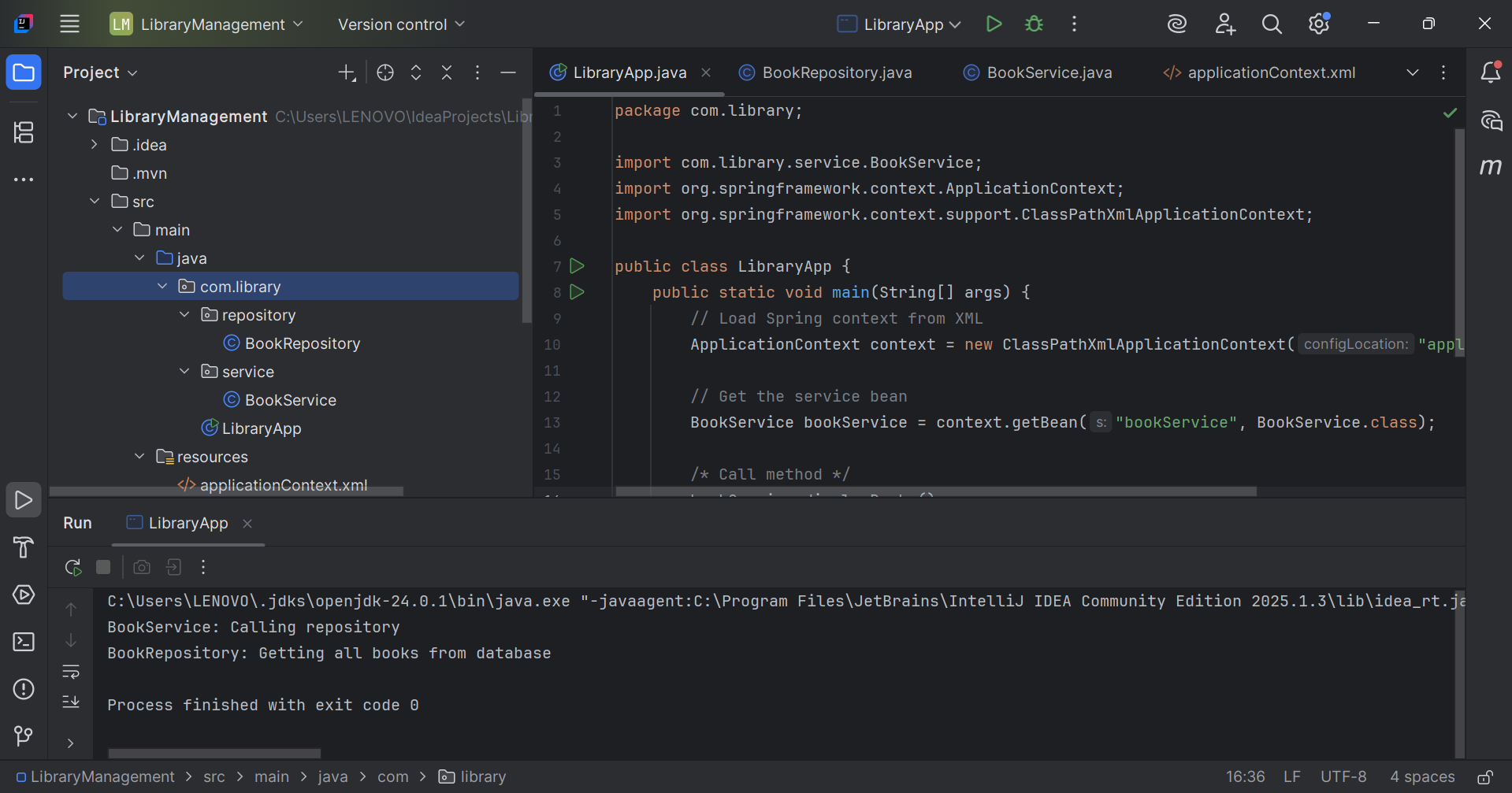
BookService.java

package com.library.service;  
  
import com.library.repository.BookRepository;  
  
public class BookService {  
 private BookRepository bookRepository;  
  
 // Setter method for Spring DI  
 public void setBookRepository(BookRepository bookRepository) {  
 this.bookRepository = bookRepository;  
 }  
  
 public void displayBooks() {  
 System.*out*.println("BookService: Calling repository");  
 bookRepository.getBooks();  
 }  
}

LibraryApp.java

package com.library;  
  
import com.library.service.BookService;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
  
public class LibraryApp {  
 public static void main(String[] args) {  
 // Load Spring context from XML  
 ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");  
  
 // Get the service bean  
 BookService bookService = context.getBean("bookService", BookService.class);  
  
 /\* Call method \*/  
 bookService.displayBooks();  
 }  
}

Output:



Implementation Description:

In this exercise, a basic Spring application was configured to serve as the backend for a Library Management system. A Maven project named LibraryManagement was created, and the required Spring Core dependency was added to the pom.xml file to enable the use of Spring’s core features. An XML-based configuration file named applicationContext.xml was created under the src/main/resources directory to define and manage beans. Two primary components were developed: BookRepository, responsible for handling data access operations, and BookService, which coordinates service-level logic. These classes were placed under appropriately named packages following standard conventions. Finally, a main class was implemented to load the Spring application context and retrieve the BookService bean to test the configuration. This setup established a foundational Spring application structure using XML configuration and demonstrated how Spring can manage components and dependencies effectively.

**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 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: Injecting bookRepository using setter -->  
 <bean id="bookService" class="com.library.service.BookService">  
 <property name="bookRepository" ref="bookRepository" />  
 </bean>  
  
</beans>

BookService.java

package com.library.service;  
  
import com.library.repository.BookRepository;  
  
public class BookService {  
 private BookRepository bookRepository;  
  
 // Setter method for DI  
 public void setBookRepository(BookRepository bookRepository) {  
 this.bookRepository = bookRepository;  
 }  
  
 public void displayBooks() {  
 System.*out*.println("BookService: Calling repository");  
 bookRepository.getBooks();  
 }  
}

BookRepository.java

package com.library.repository;  
  
public class BookRepository {  
 public void getBooks() {  
 System.*out*.println("BookRepository: Getting all books from database");  
 }  
}

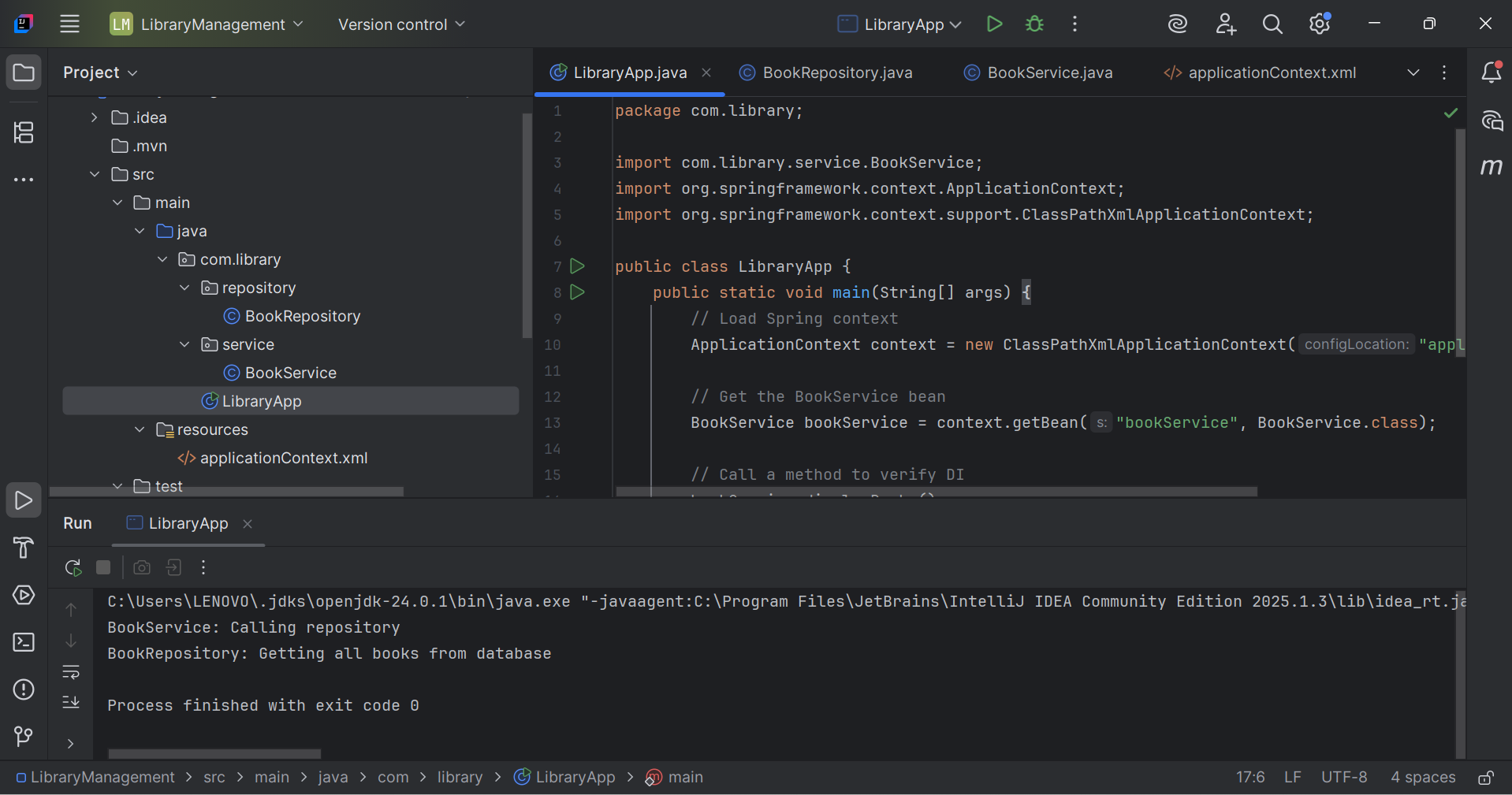
LibraryApp.java

package com.library;  
  
import com.library.service.BookService;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
  
public class LibraryApp {  
 public static void main(String[] args) {  
 // Load Spring context  
 ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");  
  
 // Get the BookService bean  
 BookService bookService = context.getBean("bookService", BookService.class);  
  
 // Call a method to verify DI  
 bookService.displayBooks();  
 }  
}

Implementation Description

In this exercise, dependency injection was implemented in the Library Management application using Spring's Inversion of Control (IoC) container. The applicationContext.xml configuration file was updated to define beans for both BookRepository and BookService, and to wire the repository into the service using setter-based injection. A public setter method was provided in the BookService class to facilitate this injection. This approach allows Spring to manage object creation and dependency management, promoting loose coupling and enhancing modularity. The configuration was tested by running the main application class, which successfully demonstrated that the BookService bean was correctly initialized with its required dependency, confirming that the injection mechanism was functioning as intended.

Output:



**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:

<?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>24</maven.compiler.source>  
 <maven.compiler.target>24</maven.compiler.target>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
  
 <dependencies>  
 <!-- Spring Core Context -->  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>5.3.36</version>  
 </dependency>  
  
 <!-- Spring AOP -->  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-aop</artifactId>  
 <version>5.3.36</version>  
 </dependency>  
  
 <!-- Spring Web MVC -->  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-webmvc</artifactId>  
 <version>5.3.36</version>  
 </dependency>  
 </dependencies>  
  
 <build>  
 <plugins>  
 <!-- Compiler Plugin to use Java 1.8 -->  
 <plugin>  
 <groupId>org.apache.maven.plugins</groupId>  
 <artifactId>maven-compiler-plugin</artifactId>  
 <version>3.10.1</version>  
 <configuration>  
 <source>1.8</source>  
 <target>1.8</target>  
 </configuration>  
 </plugin>  
 </plugins>  
 </build>  
  
  
</project>

Implementation Description

The implementation of this exercise began with the creation of a new Maven project named LibraryManagement using IntelliJ IDEA. Maven was selected as the build tool to manage project dependencies and structure. Upon project creation, the default Maven directory structure was generated, including folders for source code (src/main/java) and resources (src/main/resources), along with a pom.xml file at the root level. The pom.xml file was then configured to include dependencies for key Spring modules, namely spring-context for core container support, spring-aop for aspect-oriented programming, and spring-webmvc for building web-based components. These dependencies were added under the <dependencies> section to enable Spring functionality throughout the application. Following this, the Maven Compiler Plugin was added within the <build> section of the pom.xml to specify the Java version for compilation. The plugin was configured to use Java version 1.8 for both source and target compatibility. Finally, the Maven project was reloaded within IntelliJ IDEA to ensure that all configurations and dependencies were properly recognized and integrated, establishing a complete and ready-to-use Spring-based development environment.