**Week3\_Configuring a Basic Spring Application**

## 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:**

**Set Up a Spring Project:**

* Create a Maven project named LibraryManagement.
* Add Spring Core dependencies in the pom.xml file.

*<?*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>1.8</maven.compiler.source>  
 <maven.compiler.target>1.8</maven.compiler.target>  
 <spring.version>5.3.23</spring.version>  
 </properties>  
<dependencies><dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>${spring.version}</version>  
 </dependency><dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>4.13.2</version>  
 <scope>test</scope>  
 </dependency>  
 </dependencies>  
</project>

**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.

*<?*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  
 http://www.springframework.org/schema/beans/spring-beans.xsd">  
  
 *<!-- Define the BookRepository bean -->* <bean id="bookRepository" class="com.library.repository.BookRepository">  
 *<!-- No properties to set for this simple bean -->* </bean>  
  
 *<!-- Define the BookService bean and inject the BookRepository -->* <bean id="bookService" class="com.library.service.BookService">  
 *<!-- 'name' refers to the property in BookService class (i.e., setBookRepository) -->  
 <!-- 'ref' refers to the 'id' of the bean to be injected -->* <property name="bookRepository" ref="bookRepository"/>  
 </bean>  
  
</beans>

**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.

package com.library.repository;  
public class BookRepository {  
 public void findBook() {System.*out*.println("BookRepository: Finding a book in the database...");  
 }  
}

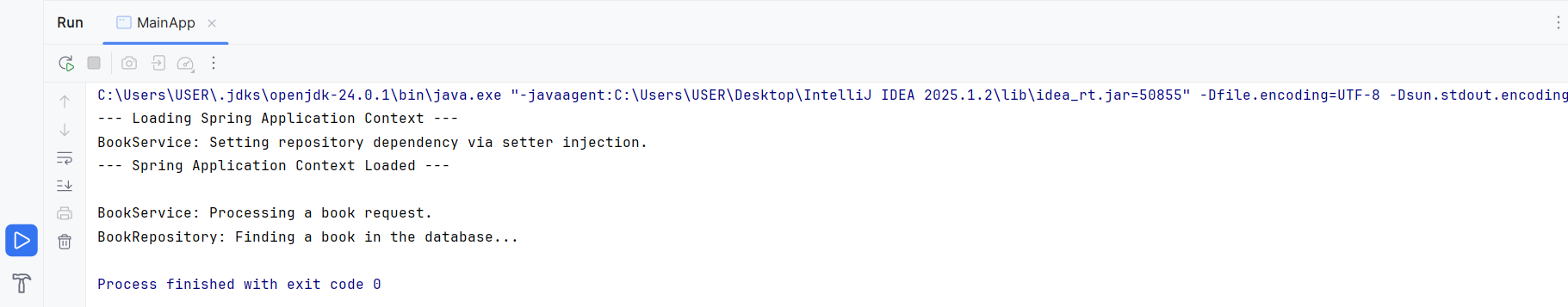
package com.library.service;  
import com.library.repository.BookRepository;  
  
public class BookService {  
  
 *// A dependency on BookRepository* private BookRepository bookRepository;  
  
 *// A setter method for Spring to inject the dependency* public void setBookRepository(BookRepository bookRepository) {  
 System.*out*.println("BookService: Setting repository dependency via setter injection.");  
 this.bookRepository = bookRepository;  
 }  
  
 *// A business logic method that uses the dependency* public void processBook() {  
 System.*out*.println("BookService: Processing a book request.");  
 *// We can now use the injected repository* bookRepository.findBook();  
 }  
}

**Run the Application:**

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

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) {  
 *// 1. Load the Spring configuration file and initialize the container* System.*out*.println("--- Loading Spring Application Context ---");  
 ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");  
 System.*out*.println("--- Spring Application Context Loaded ---\n");  
  
 *// 2. Retrieve the 'bookService' bean from the Spring container  
 // Spring has already created this object and injected its dependencies.* BookService bookService = context.getBean("bookService", BookService.class);  
  
 *// 3. Call a method on the service to test it* bookService.processBook();  
  
 *// 4. Close the context (good practice to release resources)* ((ClassPathXmlApplicationContext) context).close();  
 }  
}

Output



**Week3\_Implementing Dependency Injection**

## 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:**

**Modify the XML Configuration:**

* Update applicationContext.xml to wire BookRepository into BookService.

<!-- Initial State -->

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

<bean id="bookService" class="com.library.service.BookService"></bean>

**Modify the XML Configuration:**

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

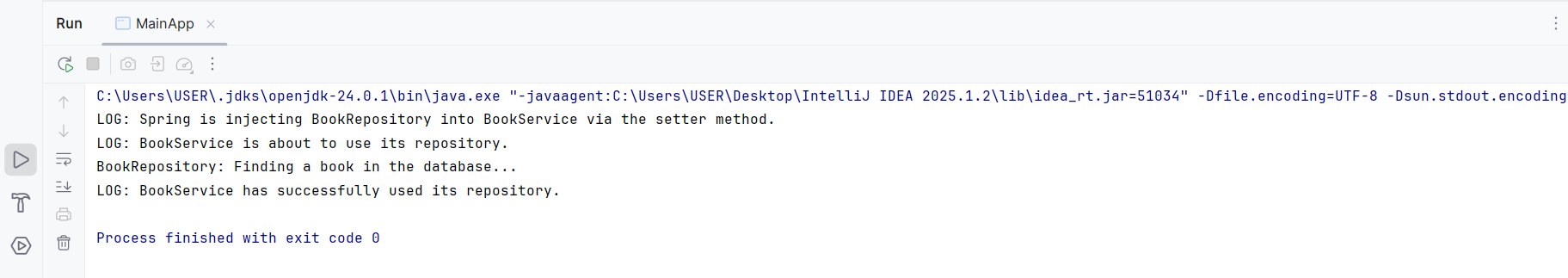
package com.library.service;  
import com.library.repository.BookRepository;  
public class BookService {  
  
 *// 1. A private field to hold the dependency.* private BookRepository bookRepository;  
  
 *// 2. A public setter method for Spring to call.  
 // The name "setBookRepository" matches the `name="bookRepository"` in the XML.* public void setBookRepository(BookRepository bookRepository) {  
 System.*out*.println("LOG: Spring is injecting BookRepository into BookService via the setter method.");  
 this.bookRepository = bookRepository;  
 }  
 *// 3. A business method that uses the injected dependency.* public void processBookLookup() {  
 System.*out*.println("LOG: BookService is about to use its repository.");  
  
 *// If injection fails, this line will cause a NullPointerException.  
 // If it succeeds, it will call the method on the injected object.* bookRepository.findBook();  
  
 System.*out*.println("LOG: BookService has successfully used its repository.");  
 }  
}

**Modify the XML Configuration:**

* Run the LibraryManagementApplication main class to verify the dependency injection.

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) {  
 *// 1. Load the Spring configuration file.  
 // During this step, Spring creates all beans and performs dependency injection.* ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");  
  
 *// 2. Retrieve the fully configured 'bookService' bean from the container.* BookService bookService = context.getBean("bookService", BookService.class);  
  
 *// 3. Call the business method that relies on the injected dependency.* bookService.processBookLookup();  
 *// 4. Close the context.* ((ClassPathXmlApplicationContext) context).close();  
 }  
}

Output



**Week3\_Creating and Configuring a Maven Project**

## Exercise 4: Creating and Configuring a Maven Project

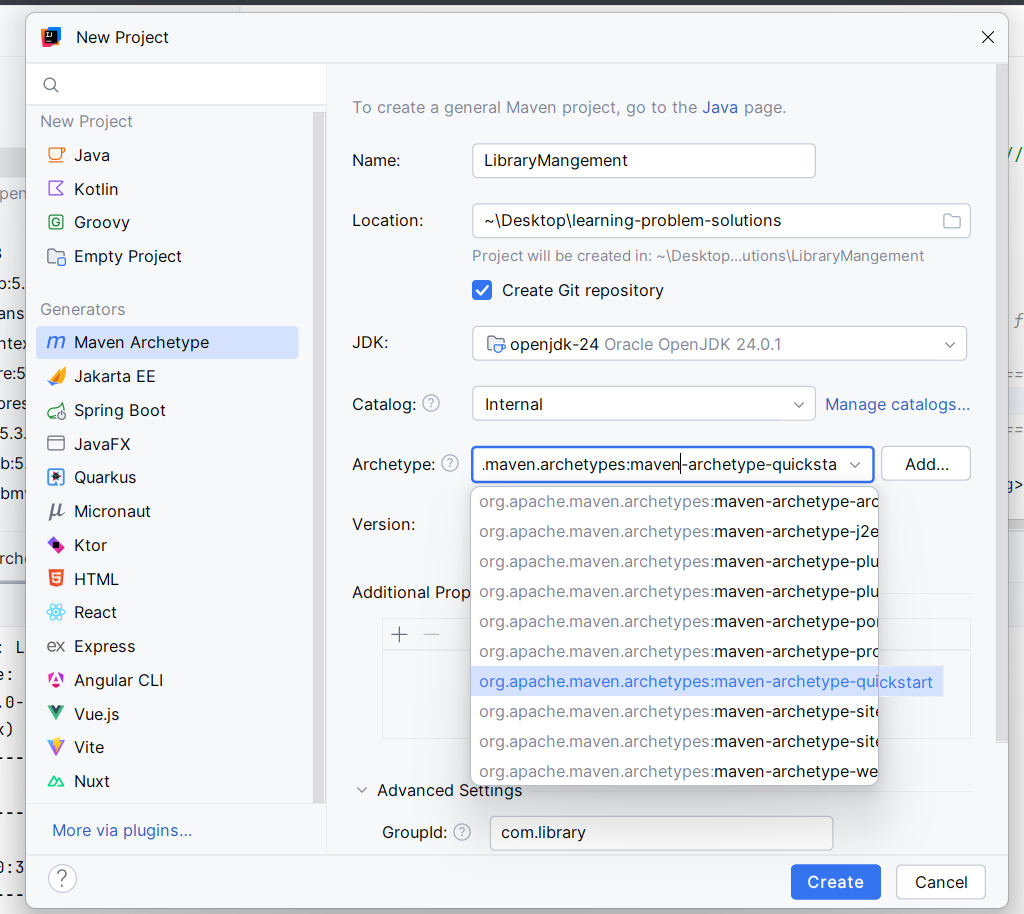
**Scenario:**

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

**Steps:**

**Create a New Maven Project:**

* Create a new Maven project named LibraryManagement.

****

**Add Spring Dependencies in pom.xml:**

* Include dependencies for Spring Context, Spring AOP, and Spring WebMVC.

*<?*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>  
 <packaging>jar</packaging> *<!-- Specifies the output will be a JAR file -->  
  
 <!-- =============================================================== -->  
 <!-- 1. PROPERTIES: Define versions in one place for easy updates -->  
 <!-- =============================================================== -->* <properties>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 <spring.version>5.3.23</spring.version>  
 *<!-- This configures the Maven Compiler Plugin for Java 1.8 -->* <maven.compiler.source>1.8</maven.compiler.source>  
 <maven.compiler.target>1.8</maven.compiler.target>  
 </properties>  
  
 *<!-- =============================================================== -->  
 <!-- 2. DEPENDENCIES: Declare the libraries your project needs -->  
 <!-- =============================================================== -->* <dependencies>  
 *<!-- Spring Context: The core IoC container (includes Core, Beans, SpEL) -->* <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>${spring.version}</version>  
 </dependency>  
  
 *<!-- Spring AOP: For Aspect-Oriented Programming (e.g., logging, transactions) -->* <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-aop</artifactId>  
 <version>${spring.version}</version>  
 </dependency>  
  
 *<!-- Spring WebMVC: For building web applications with the Model-View-Controller pattern -->* <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-webmvc</artifactId>  
 <version>${spring.version}</version>  
 </dependency>  
  
 *<!-- JUnit: A standard library for unit testing -->* <dependency>  
 <groupId>junit</groupId>  
 <artifactId>junit</artifactId>  
 <version>4.13.2</version>  
 <scope>test</scope>  
 </dependency>  
 </dependencies>  
  
</project>

**Configure Maven Plugins:**

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

Output

