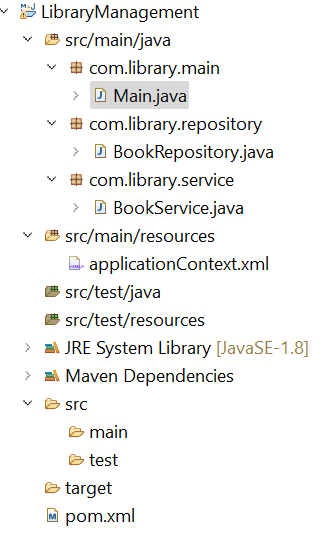
**Exercise 1: Configuring a Basic Spring Application**

**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.
5. **Set Up a Spring Project:**

**2.Configure the Application Context:**

<?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

file:///C:/Users/bhara/Downloads/spring-beans.xsd.xml">

<!-- Define Repository Bean -->

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

<!-- Define Service Bean and inject Repository -->

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

<property name="bookRepository" ref="bookRepository"/>

</bean>

</beans>

**3.Define Service and Repository Classes:**

**BookService Java Class:**

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 displayBooks() {

System.*out*.println("BookService: Calling repository...");

bookRepository.getAllBooks();

}

}

**BookRepository class**

package com.library.repository;

public class BookRepository {

public void getAllBooks() {

System.*out*.println("Fetching all books from the repository...");

}

}

4. **Run the Application:**

Main.java:

package com.library.main;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

public class Main { private static ApplicationContext *context*;

public static void main(String[] args) {

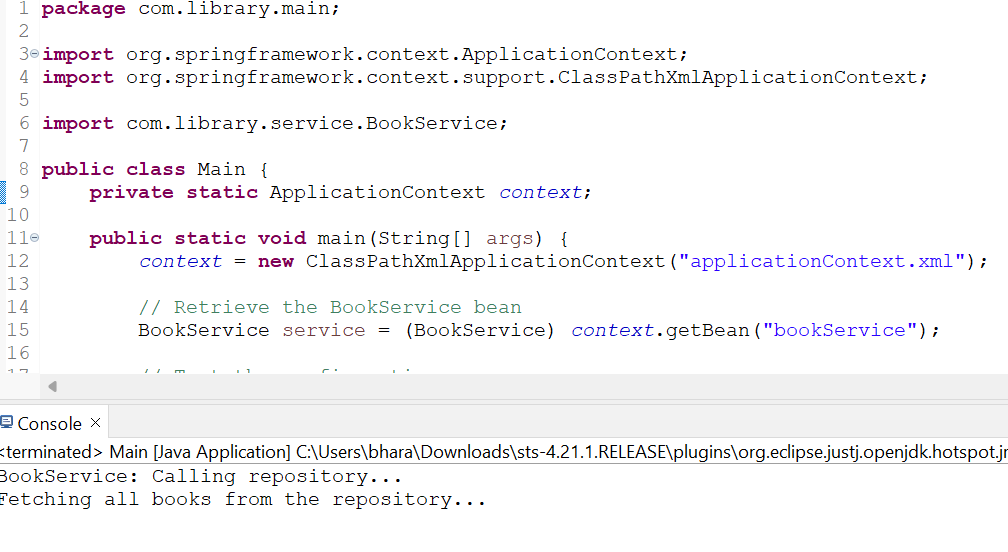
*context* = new ClassPathXmlApplicationContext("applicationContext.xml");

BookService service = (BookService) *context*.getBean("bookService");

service.displayBooks();

}

}

**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**.
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.

**1.Modify the XML Configuration:**

**Code**:

<?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">

<!-- BookRepository Bean -->

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

<!-- BookService Bean -->

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

<property name="bookRepository" ref="bookRepository"/>

</bean>

</beans>

**2 Update the BookService Class:**

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 displayBooks() {

System.out.println("BookService: Calling repository...");

bookRepository.getAllBooks();

}

}

3. **Test the Configuration in the Main Class:**

package com.library.main;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

public class Main {

public static void main(String[] args) {

@SuppressWarnings("resource")

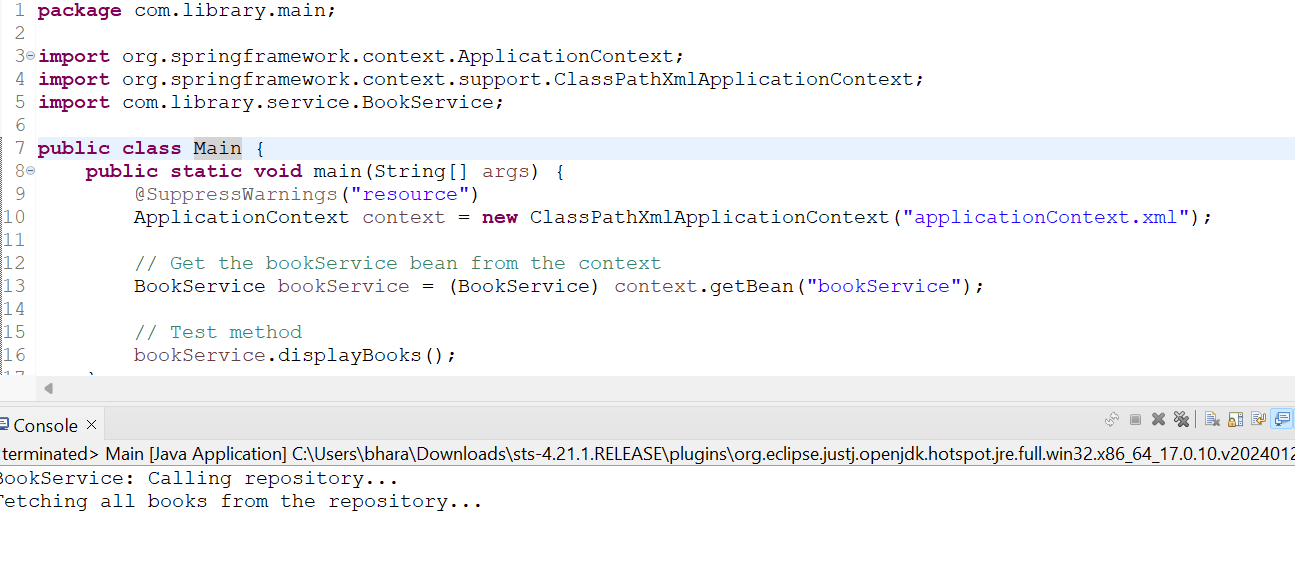
ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

BookService bookService = (BookService) context.getBean("bookService");

bookService.displayBooks();

}

}

**OUTPUT:**

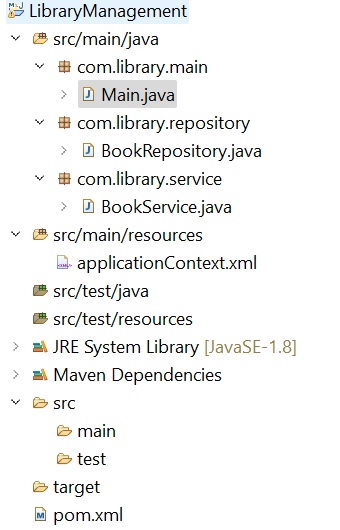
**Exercise 4: Creating and Configuring a Maven Project**

**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.

**1.Create a New Maven Project:**

**2.Add Spring Dependencies in pom.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>1.0-SNAPSHOT</version>

<dependencies>

<!-- Spring Core Context -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.32</version>

</dependency>

<!-- Spring AOP -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.32</version>

</dependency>

<!-- Spring Web MVC -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.32</version>

</dependency>

</dependencies>

<build>

<plugins>

<!-- Maven Compiler Plugin for 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>

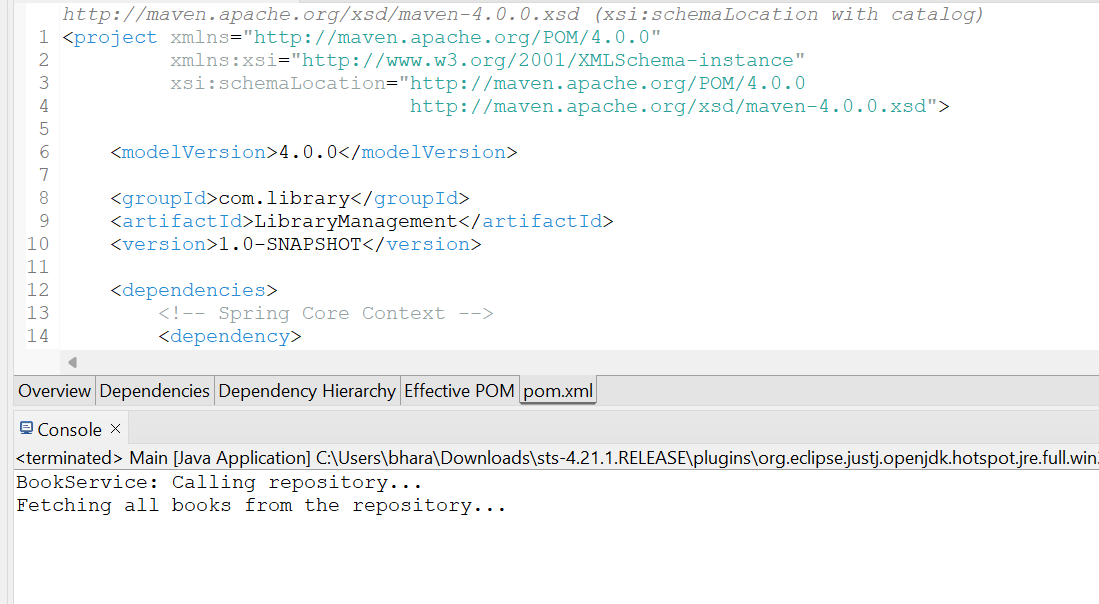
</plugins>

</build>

</project>

**3.Configure Maven Plugins:**

Output:

****

**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.

**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.

**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

file:///C:/Users/bhara/Downloads/spring-beans.xsd.xml">

<!-- BookRepository Bean -->

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

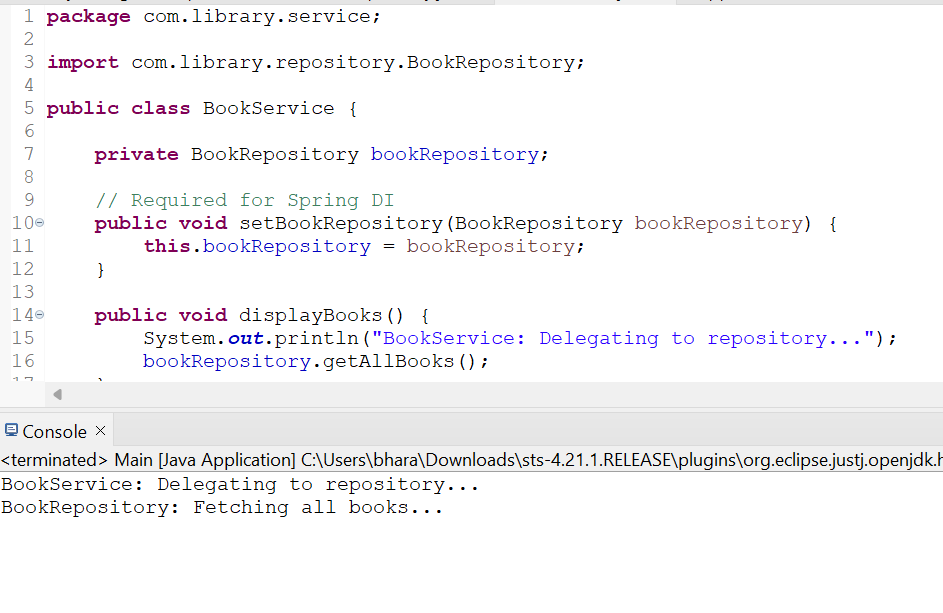
<!-- BookService Bean -->

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

<property name="bookRepository" ref="bookRepository"/>

</bean>

</beans>

****

**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.

**1. Create Spring Configuration 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

file:///C:/Users/bhara/Downloads/spring-beans.xsd.xml">

<!-- BookRepository Bean -->

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

<!-- BookService Bean -->

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

<property name="bookRepository" ref="bookRepository"/>

</bean>

</beans>

**BookRepository:**

package com.library.repository;

public class BookRepository {

public void getAllBooks() {

System.*out*.println("BookRepository: Returning all books.");

}

}

**2 Update the BookService Class:**

**BookService:**

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 displayBooks() {

System.*out*.println("BookService: Fetching books via repository...");

bookRepository.getAllBooks();

} }

**3.Run the Application:**

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

package com.library.main;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

public class Main {

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

BookService bookService = context.getBean("bookService", BookService.class);

bookService.displayBooks();

}

}

**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.

**Enable Component Scanning:**

<?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">

<!-- Enable component scanning for com.library -->

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

</beans>

**2.Annotate Classes:**

package com.library.repository;

import org.springframework.stereotype.Repository;

@Repository

public class BookRepository {

public void getAllBooks() {

System.*out*.println("BookRepository: Returning all books.");

}

}

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 {

@Autowired

private BookRepository bookRepository;

public void displayBooks() {

System.*out*.println("BookService: Fetching books via repository...");

bookRepository.getAllBooks();

}

}

**Main.java:**

package com.library.main;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

public class Main {

public static void main(String[] args) {

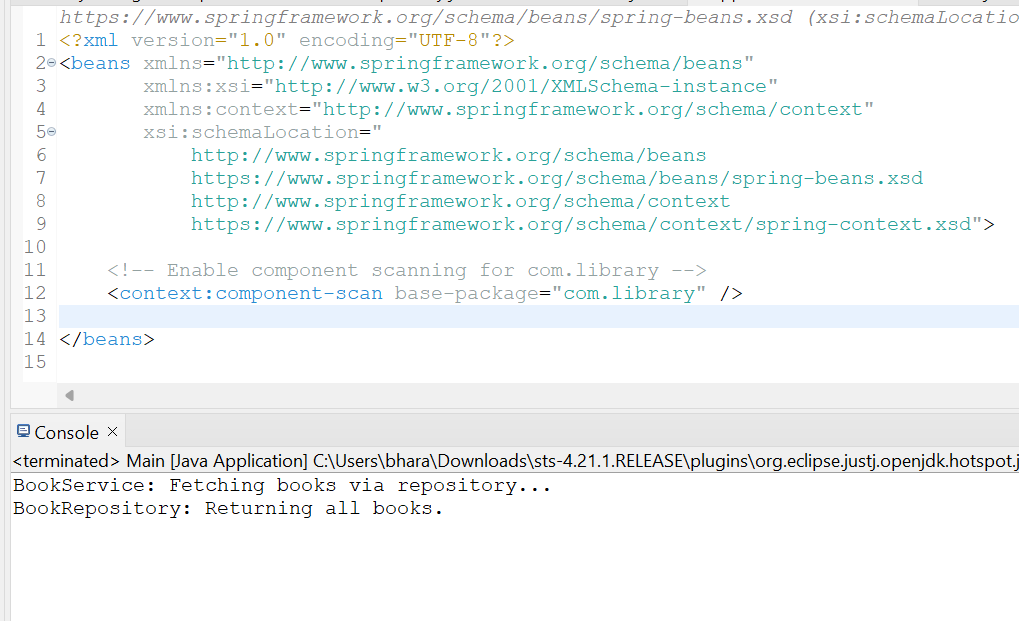
ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");

BookService bookService = context.getBean(BookService.class);

bookService.displayBooks();

}

}

**OUTPUT:**

**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.

**1.Configure Constructor Injection:**

**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">

<!-- Bean for BookRepository -->

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

<!-- Bean for BookService with constructor and setter injection -->

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

<!-- Constructor injection -->

<constructor-arg value="Central Library Service" />

<!-- Setter injection -->

<property name="bookRepository" ref="bookRepository" />

</bean>

</beans>

**2.Configure Setter Injection:**

**BookRepository Class:**

package com.library.repository;

public class BookRepository {

public void listBooks() {

System.*out*.println("BookRepository: Listing books...");

}

}

**BookService Class:**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private String serviceName;

private BookRepository bookRepository;

// Constructor injection

public BookService(String serviceName) {

this.serviceName = serviceName;

}

// Setter injection

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void displayBooks() {

System.*out*.println("Service Name: " + serviceName);

bookRepository.listBooks();

}

}

**3.Test the Injection:**

**package** com.library.main;

**import** org.springframework.context.ApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**import** com.library.service.BookService;

**public** **class** Main {

**public** **static** **void** main(String[] args) {

ApplicationContext context = **new** ClassPathXmlApplicationContext("applicationContext.xml");

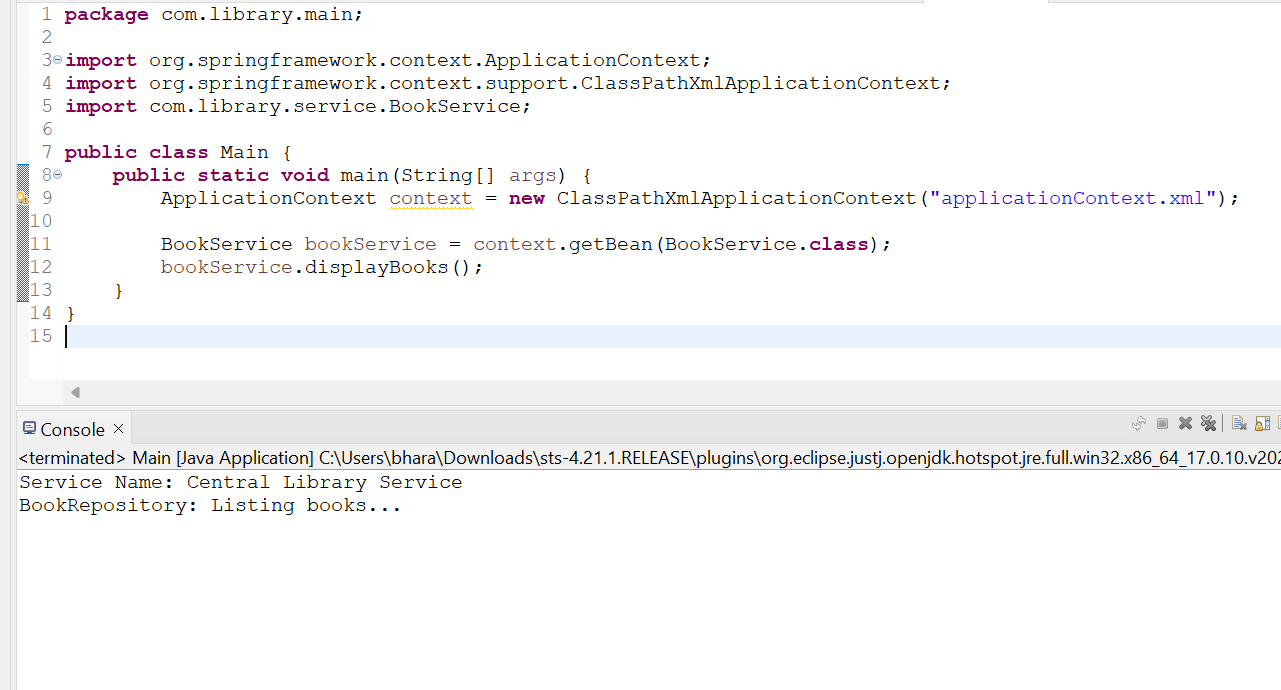
BookService bookService = context.getBean(BookService.**class**);

bookService.displayBooks();

}

}

**OUTPUT:**

****

**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.