WEEK – 3

**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.

**Pom.xml:**

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.32</version>

</dependency>

</dependencies>

**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

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>

**com.library.repository.BookRepository:**

package com.library.repository;

public class BookRepository {

public void saveBook(String title) {

System.out.println("Book saved: " + title);

}

}

**com.library.service.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 addBook(String title) {

System.out.println("Adding book: " + title);

bookRepository.saveBook(title);

}

}

**Com.library.main:**

package com.library.main;

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) {

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

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

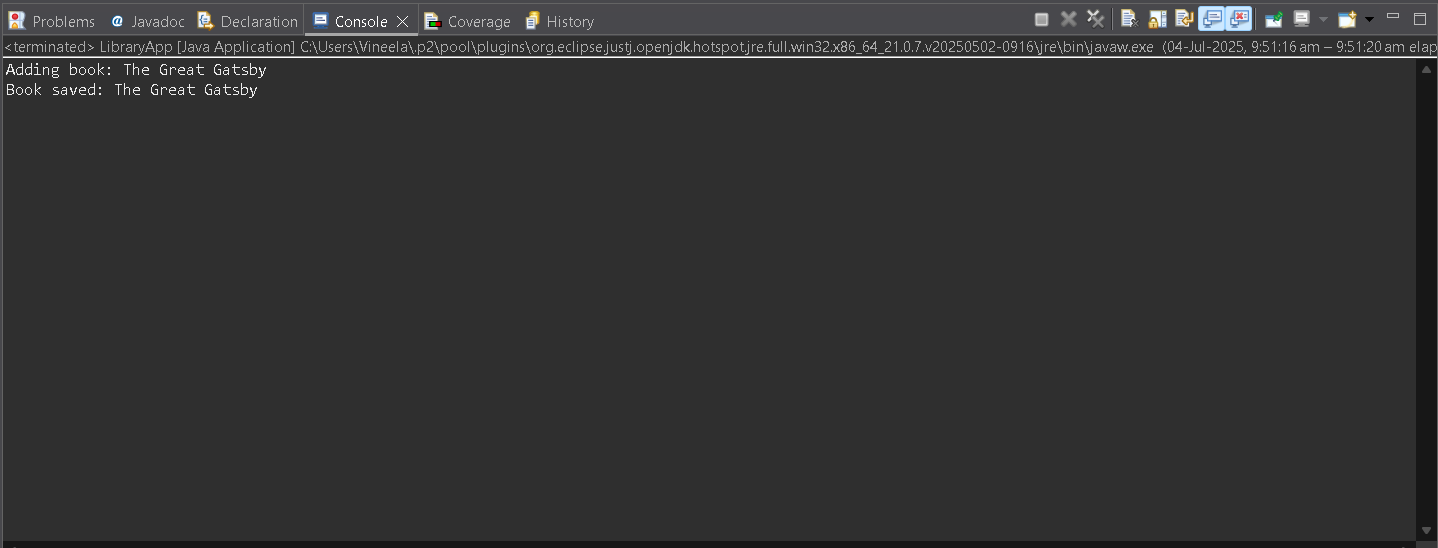
bookService.addBook("The Great Gatsby");

((ClassPathXmlApplicationContext) context).close();

}

}

**OUTPUT:**

****

**Excercise-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.

**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

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>

**BookRepository.java:**

package com.library.repository;

import java.util.Arrays;

import java.util.List;

public class BookRepository {

public List<String> getBooksByGenre(String genre) {

if ("fiction".equalsIgnoreCase(genre)) {

return Arrays.asList("The Alchemist", "The Great Gatsby", "1984");

} else if ("science".equalsIgnoreCase(genre)) {

return Arrays.asList("A Brief History of Time", "Cosmos", "The Selfish Gene");

} else {

return Arrays.asList("No books available for this genre.");

}

}

}

**BookService.java:**

package com.library.service;

import com.library.repository.BookRepository;

import java.util.List;

public class BookService {

private BookRepository bookRepository;

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void recommendBooks(String genre) {

List<String> books = bookRepository.getBooksByGenre(genre);

System.out.println("Recommended books for genre: " + genre);

for (String book : books) {

System.out.println("- " + book);

}

}

}

**LibraryManagementApplication.java:**

package com.library.main;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryManagementApplication {

public static void main(String[] args) {

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

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

bookService.recommendBooks("fiction");

System.out.println();

bookService.recommendBooks("science");

System.out.println();

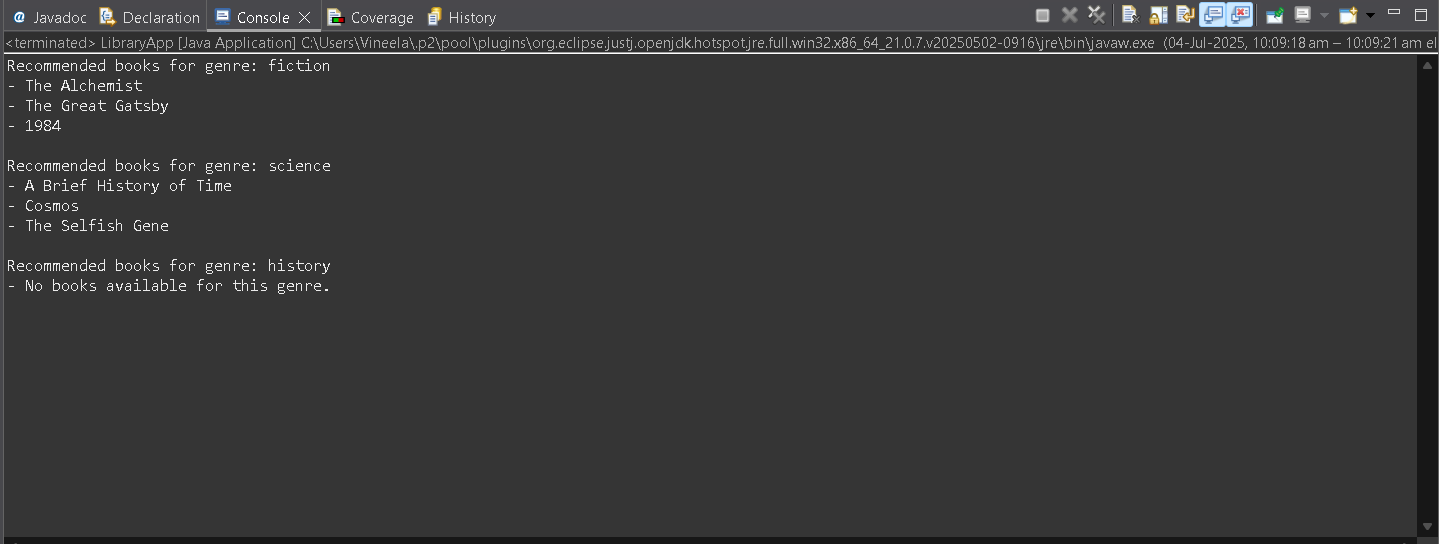
bookService.recommendBooks("history");

((ClassPathXmlApplicationContext) context).close();

}

}

**OUTPUT:**



**Exercise-3:** **Implementing Logging with Spring AOP**

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

**pom.xml:**

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aspects</artifactId>

<version>5.3.32</version>

</dependency>

**LogginAspect.java:**

package com.library.aspect;

import org.aspectj.lang.ProceedingJoinPoint;

public class LoggingAspect {

public Object logExecutionTime(ProceedingJoinPoint joinPoint) throws Throwable {

long start = System.currentTimeMillis();

Object result = joinPoint.proceed(); // run the actual method

long end = System.currentTimeMillis();

System.out.println("[LOG] Method " + joinPoint.getSignature().getName() +

" executed in " + (end - start) + " ms");

return result;

}

}

**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"

xmlns:aop="http://www.springframework.org/schema/aop"

xsi:schemaLocation="

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/aop

http://www.springframework.org/schema/aop/spring-aop.xsd">

<aop:config>

<aop:aspect ref="loggingAspect">

<aop:around method="logExecutionTime"

pointcut="execution(\* com.library.service.\*.\*(..))" />

</aop:aspect>

</aop:config>

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

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

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

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

</bean>

</beans>

**LibraryManagementApplication:**

package com.library.main;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryManagementApplication {

public static void main(String[] args) {

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

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

bookService.recommendBooks("fiction");

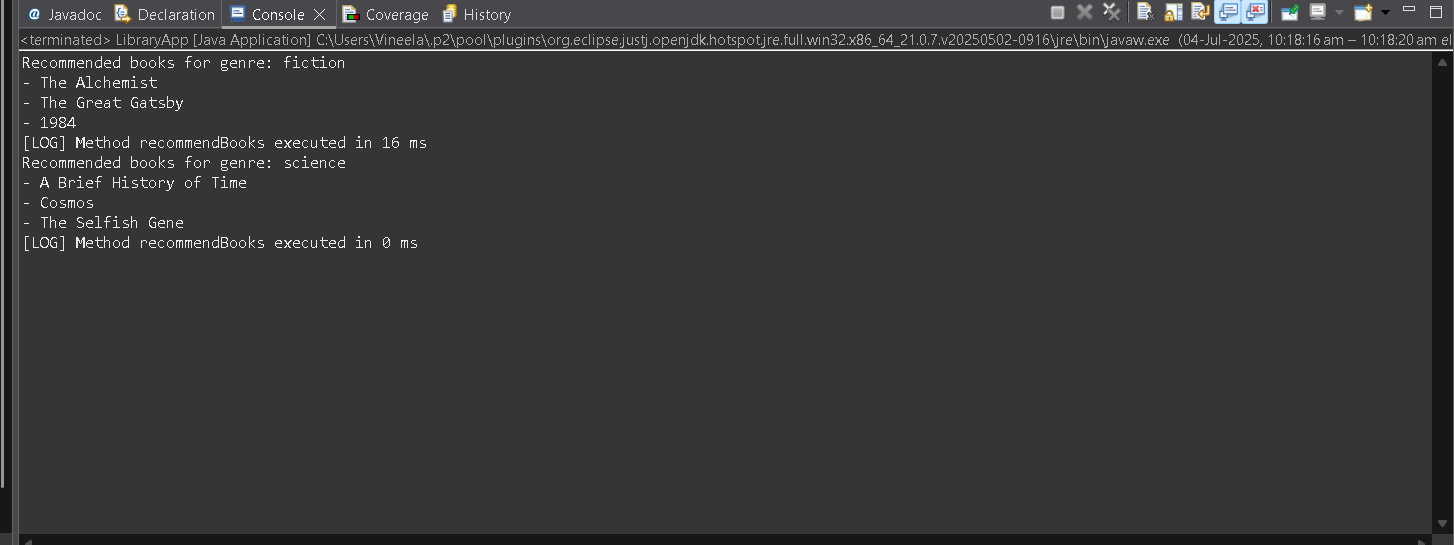
bookService.recommendBooks("science");

((ClassPathXmlApplicationContext) context).close();

}

}

**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.

**pom.xml:**

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.32</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aspects</artifactId>

<version>5.3.32</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.32</version>

</dependency>

</dependencies>

<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>

**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

http://www.springframework.org/schema/beans/spring-beans.xsd">

<bean id="welcomeService" class="com.library.service.WelcomeService" />

</beans>

**WelcomeService.java:**

package com.library.service;

public class WelcomeService {

public void showMessage() {

System.out.println("✅ Welcome to the Library Management System!");

}

}

**LibraryApp:**

package com.library.main;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.WelcomeService;

public class LibraryApp {

public static void main(String[] args) {

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

WelcomeService service = context.getBean("welcomeService", WelcomeService.class);

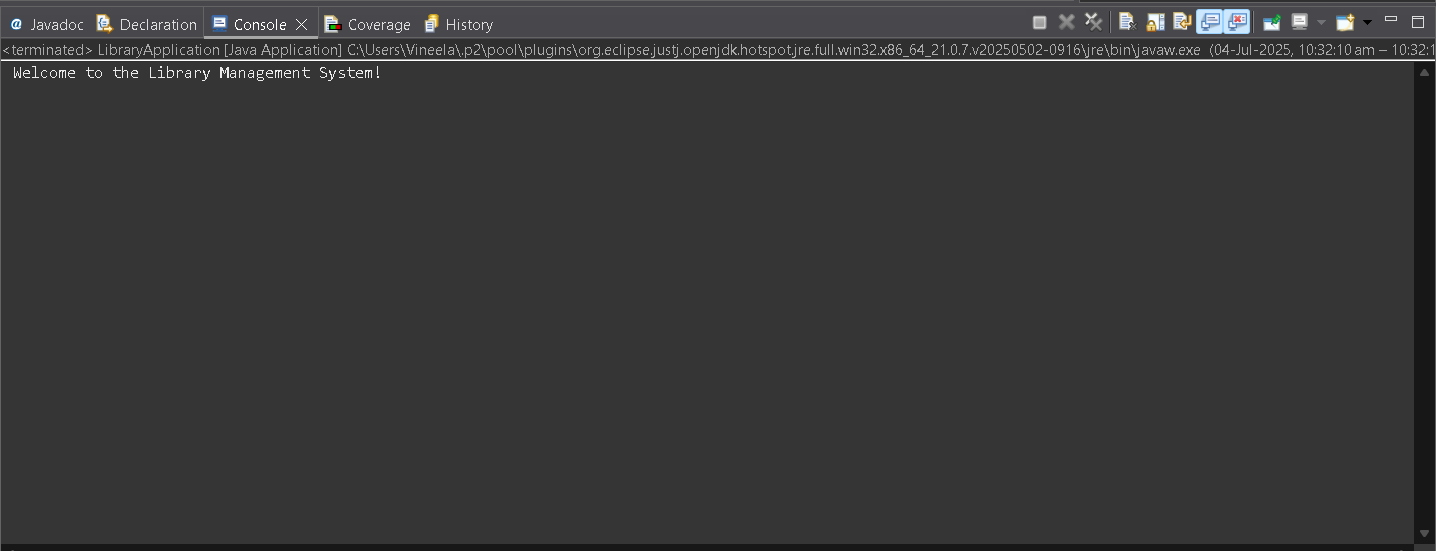
service.showMessage();

((ClassPathXmlApplicationContext) context).close();

}

}

**OUTPUT:**



**Exercise-5: Configuring the Spring IoC Container**

Scenario: The library management application requires a central configuration for beans and dependencies.

**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

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>

**BookRepository.java:**

package com.library.repository;

public class BookRepository {

public void displayBooks() {

System.out.println("📚 Displaying list of books from the repository...");

}

}

**BookService.java:**

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 listBooks() {

System.out.println("📖 BookService is calling the repository...");

bookRepository.displayBooks();

}

}

**LibraryApp.java:**

package com.library.main;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

public class LibraryApp {

public static void main(String[] args) {

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

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

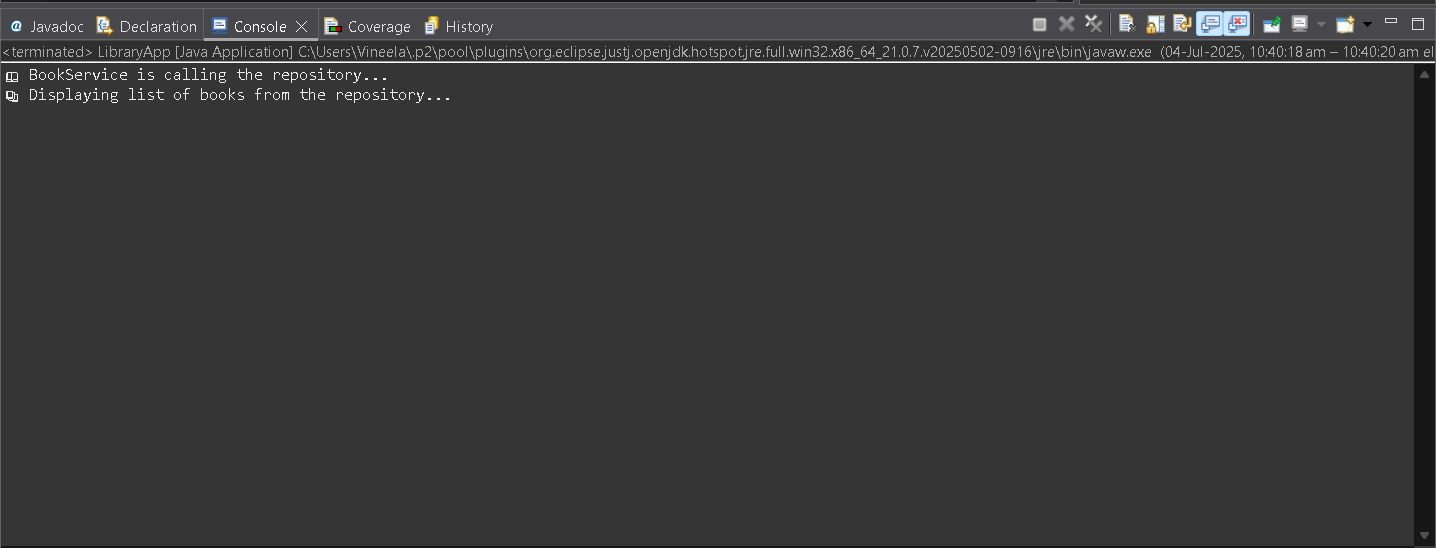
bookService.listBooks();

((ClassPathXmlApplicationContext) context).close();

}

}

**OUTPUT:**



**Exercise-6: Configuring Beans with Annotations**

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

**applicationContext.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

http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/context

http://www.springframework.org/schema/context/spring-context.xsd">

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

</beans>

**UserRepository.java:**

package com.library.repository;

import org.springframework.stereotype.Repository;

import java.util.Arrays;

import java.util.List;

public class UserRepository {

public List<String> getAllUsers() {

return Arrays.asList("Vineela", "Deepak", "Priya");

}

}

**UserService.java:**

package com.library.service;

import com.library.repository.UserRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class UserService {

private UserRepository userRepository;

public void showUsers() {

System.out.println("🔐 UserService: Fetching user list...");

List<String> users = userRepository.getAllUsers();

System.out.println("👤 UserRepository: List of users:");

for (String user : users) {

System.out.println("- " + user);

}

}

}

**LIbraryManagementApplication:**

package com.library.main;

import com.library.service.UserService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryManagementApplication {

public static void main(String[] args) {

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

UserService userService = context.getBean(UserService.class);

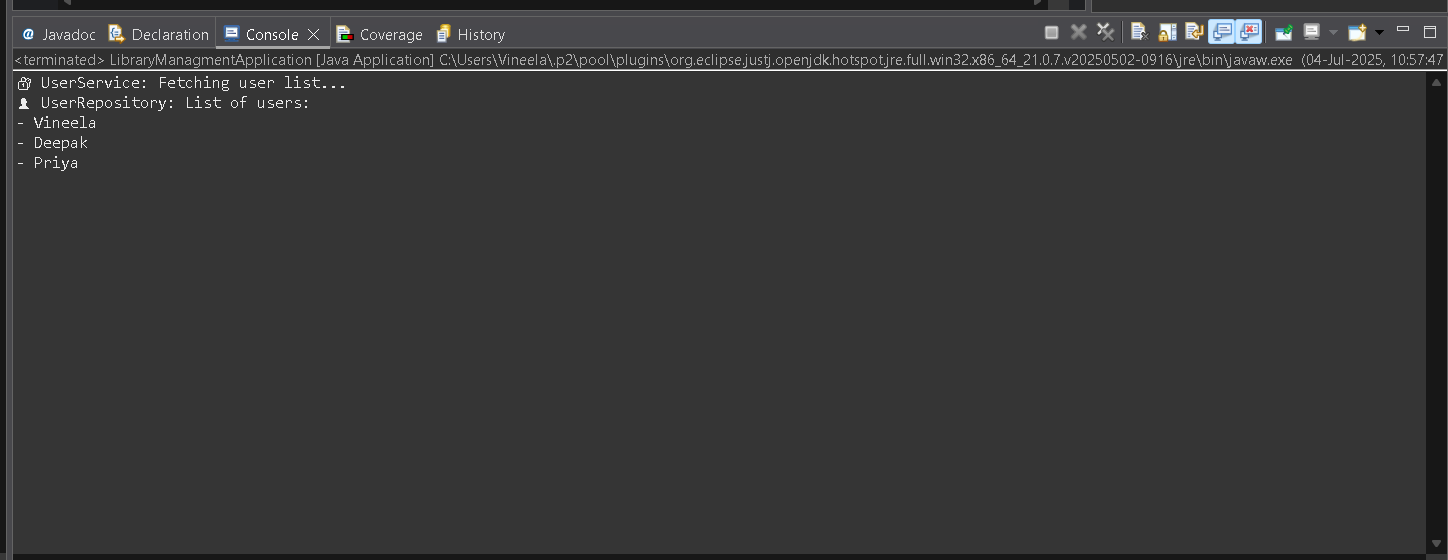
userService.showUsers();

((ClassPathXmlApplicationContext) context).close();

}

}

**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.

**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

http://www.springframework.org/schema/beans/spring-beans.xsd">

<bean id="userRepository" class="com.library.repository.UserRepository" />

<bean id="notificationService" class="com.library.service.NotificationService">

<constructor-arg value="Reminder: Return books on time." />

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

</bean>

</beans>

**UserRepository.java:**

package com.library.repository;

import java.util.Arrays;

import java.util.List;

public class UserRepository {

public List<String> getAllUsers() {

return Arrays.asList("Alice", "Bob", "Charlie");

}

}

**NotificationService:**

package com.library.service;

import com.library.repository.UserRepository;

public class NotificationService {

private String message;

private UserRepository userRepository;

public NotificationService(String message) {

this.message = message;

}

public void setUserRepository(UserRepository userRepository) {

this.userRepository = userRepository;

}

public void sendNotifications() {

System.out.println("📢 Notification: " + message);

System.out.println("👤 Sending to users:");

for (String user : userRepository.getAllUsers()) {

System.out.println("- Notified " + user);

}

}

}

**LibraryManagementApplication.java:**

package com.library.main;

import com.library.service.NotificationService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryManagementApplication {

public static void main(String[] args) {

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

NotificationService service = context.getBean("notificationService", NotificationService.class);

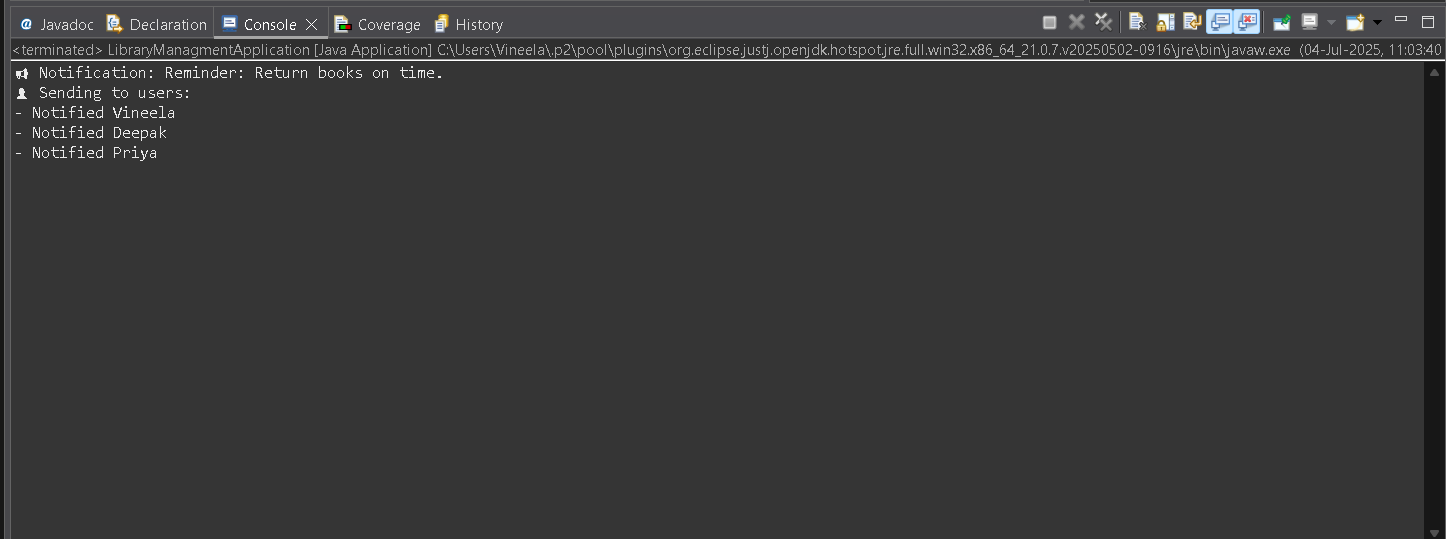
service.sendNotifications();

((ClassPathXmlApplicationContext) context).close();

}

}

**OUTPUT:**



**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.

**applicationContext.xml:**

<?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"

xsi:schemaLocation="

http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd

http://www.springframework.org/schema/aop

http://www.springframework.org/schema/aop/spring-aop.xsd">

<aop:aspectj-autoproxy />

<bean id="membershipService" class="com.library.service.MembershipService" />

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

<aop:config>

<aop:aspect ref="loggingAspect">

<aop:before method="beforeAdvice"

pointcut="execution(\* com.library.service.MembershipService.\*(..))" />

<aop:after method="afterAdvice"

pointcut="execution(\* com.library.service.MembershipService.\*(..))" />

</aop:aspect>

</aop:config>

</beans>

**MembershipService:**

package com.library.service;

public class MembershipService {

public void registerMember() {

System.out.println("🎫 Registering new member...");

}

public void cancelMembership() {

System.out.println("❌ Cancelling membership...");

}

}

**LoggingAspect.java:**

package com.library.aspect;

import org.aspectj.lang.JoinPoint;

public class LoggingAspect {

public void beforeAdvice(JoinPoint joinPoint) {

System.out.println("[LOG] Method " + joinPoint.getSignature().getName() + "() is about to be called");

}

public void afterAdvice(JoinPoint joinPoint) {

System.out.println("[LOG] Method " + joinPoint.getSignature().getName() + "() has been executed");

}

}

**LibraryManagementApplication.java:**

package com.library.main;

import com.library.service.MembershipService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryManagementApplication {

public static void main(String[] args) {

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

MembershipService service = context.getBean("membershipService", MembershipService.class);

service.registerMember();

System.out.println();

service.cancelMembership();

((ClassPathXmlApplicationContext) context).close();

}

}

**OUTPUT:**

