**Student Name:** KARTHYAYENI P

**Registration No:** 22CSR089

**Course/Batch:** KONGU ENGINEERING COLLEGE (B.E COMPUTER SCIENCE AND ENGINEERING)

**EXERCISE 1: CONFIGURING A BASIC SPRING APPLICATION**

**Introduction:**

This application demonstrates a basic Library Management System using Spring Framework’s XML-based configuration and setter-based dependency injection to manage bean relationships.

**Objective:**

* To implement setter-based dependency injection using Spring's XML configuration.
* To separate concerns between the service (BookService) and repository (BookRepository) layers.
* To simulate a simple book listing and searching feature within a library system.

**Implementation Breakdown:**

**MainApp.java:**

package org.example;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

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

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

System.out.println("Welcome to the Library Management System\n");

bookService.displayAllBooks();

bookService.getBookById(1);

bookService.getBookById(10);

System.out.println("Application executed successfully.");

}

}

**BookService.java:**

package org.example;

import java.util.List;

public class BookService {

private BookRepository bookRepository;

private String librarianName;

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void setLibrarianName(String librarianName) {

this.librarianName = librarianName;

}

public void displayAllBooks() {

System.out.println("Librarian: " + librarianName);

System.out.println("Available Books in Library:");

List<String> books = bookRepository.getAllBooks();

for (int i = 0; i < books.size(); i++) {

System.out.println(" " + (i + 1) + ". " + books.get(i));

}

System.out.println();

}

public void getBookById(int id) {

System.out.println("Searching for Book ID " + id + "...");

String result = bookRepository.findBookById(id);

System.out.println("Result: " + result + "\n");

}

}

**BookRepository.java:**

package org.example;

import java.util.ArrayList;

import java.util.List;

public class BookRepository {

public List<String> getAllBooks() {

List<String> books = new ArrayList<>();

books.add("Java Programming by John Doe");

books.add("Spring Framework Essentials by Jane Smith");

books.add("Data Structures in Depth by Alan Turing");

return books;

}

public String findBookById(int id) {

List<String> books = getAllBooks();

if (id >= 0 && id < books.size()) {

return books.get(id);

} else {

return "Book not found.";

}

}

}

**Application.properties:**

<?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="org.example.BookRepository"/>

<bean id="bookService" class="org.example.BookService">

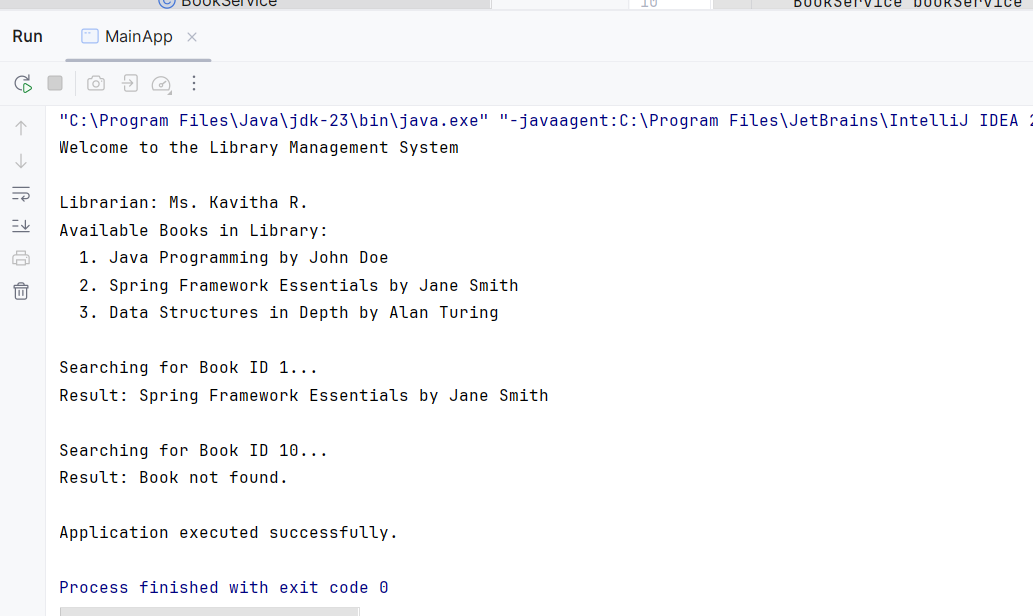
<property name="bookRepository" ref="bookRepository"/>

<property name="librarianName" value="Ms. Kavitha R."/>

</bean>

</beans>

**Output:**

****

**Conclusion:**

The project effectively applies Spring’s IoC and DI concepts, allowing for clean, modular, and testable code by decoupling object creation from business logic.

**EXERCISE 2: IMPLEMENTING A DEPENDENCY INJECTION**

**Introduction:**

This Spring-based application uses **interface-driven development and setter-based dependency injection** to manage service-repository communication within a modular Library Management System.

**Objective:**

* To implement **interface-based programming** by defining a BookRepository contract and injecting its implementation using Spring XML configuration.
* To demonstrate **setter-based dependency injection**, where the repository dependency is injected into the service without tight coupling.
* To build a basic **book management flow**, allowing the service layer to retrieve and print all available books through the injected repository.

**Implementation Breakdown:**

**LibraryManagementApplication.java:**

package org.example;

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.printBooks();

}

}

**Book.java:**

package org.example;

public class Book {

private String title;

public Book(String title) {

this.title = title;

}

@Override

public String toString() {

return title;

}

}

**BookRepository.java:**

package org.example;

import java.util.List;

public interface BookRepository {

List<Book> findAllBooks();

}

**BookService.java:**

package org.example;

import java.util.List;

public class BookService {

private BookRepository bookRepository;

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void printBooks() {

if (bookRepository != null) {

List<Book> books = bookRepository.findAllBooks();

books.forEach(System.out::println);

} else {

System.out.println("BookRepository not injected!");

}

}

}

**InMemoryBookRepository.java:**

package org.example;

import java.util.Arrays;

import java.util.List;

public class InMemoryBookRepository implements BookRepository {

@Override

public List<Book> findAllBooks() {

return Arrays.asList(

new Book("The Alchemist"),

new Book("Spring in Action"),

new Book("Clean Code")

);

}

}

**Application.properties:**

<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="org.example.InMemoryBookRepository" />

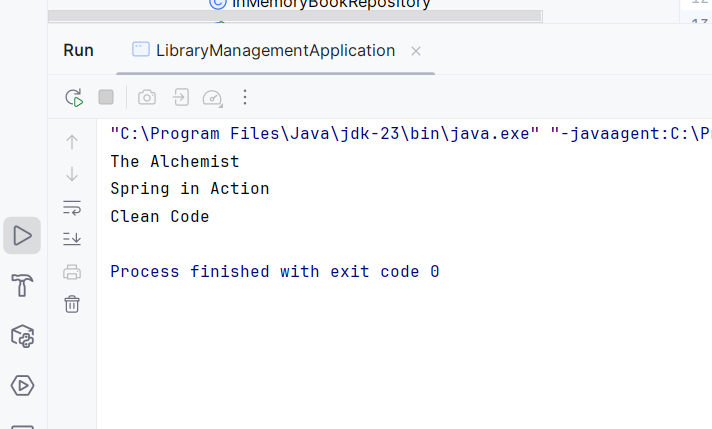
<bean id="bookService" class="org.example.BookService">

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

</bean>

</beans>

**Output:**

****

**Conclusion:**

This setup promotes a **clean separation of concerns** and supports flexibility, as different implementations of BookRepository can be injected without modifying the BookService logic — a core advantage of Spring's dependency injection model.

**EXERCISE 4: CREATING AND CONFIGURING A MAVEN PROJECT**

**Introduction:**

This project demonstrates a Spring-based Library Management System where core concepts such as setter-based dependency injection, interface-based design, and Aspect-Oriented Programming (AOP) are implemented using Maven for build configuration and Spring XML for bean wiring.

**Objective:**

* To create and configure a Maven project that supports Spring and AOP dependencies for modular development.
* To implement setter-based dependency injection for injecting BookRepository into BookService using XML configuration.
* To apply Aspect-Oriented Programming (AOP) using @Aspect and @Before advice to log method executions, demonstrating cross-cutting concerns.

**Implementation:**

**MainApp.java:**

package org.example;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

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

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

bookService.displayBooks();

}

}

**Book.java:**

package org.example;

public class Book {

private String title;

private String author;

public Book(String title, String author) {

this.title = title;

this.author = author;

}

public String getTitle() {

return title;

}

public String getAuthor() {

return author;

}

}

**BookRepository.java:**

package org.example;

import java.util.ArrayList;

import java.util.List;

public class BookRepository {

public List<Book> getAllBooks() {

List<Book> books = new ArrayList<>();

books.add(new Book("Effective Java", "Joshua Bloch"));

books.add(new Book("Spring in Action", "Craig Walls"));

books.add(new Book("Clean Code", "Robert C. Martin"));

books.add(new Book("Head First Design Patterns", "Eric Freeman"));

books.add(new Book("Java Concurrency in Practice", "Brian Goetz"));

books.add(new Book("Pro Spring 5", "Iuliana Cosmina"));

books.add(new Book("Beginning Spring Boot 3", "Jude Joseph"));

books.add(new Book("Introduction to Algorithms", "Thomas H. Cormen"));

books.add(new Book("Refactoring", "Martin Fowler"));

books.add(new Book("The Pragmatic Programmer", "Andrew Hunt"));

return books;

}

}

**BookService.java:**

package org.example;

import java.util.List;

public class BookService {

private BookRepository bookRepository;

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void displayBooks() {

printBanner();

List<Book> books = bookRepository.getAllBooks();

System.out.printf("%-30s | %-20s%n", "Title", "Author");

for (Book book : books) {

System.out.printf("%-30s | %-20s%n", book.getTitle(), book.getAuthor());

}

System.out.println("Total Books: " + books.size());

}

private void printBanner() {

System.out.println(" WELCOME TO SPRING LIBRARY MANAGEMENT SYSTEM ");

}

}

**LoggingAspect.java:**

package org.example;

import org.aspectj.lang.JoinPoint;

import org.aspectj.lang.annotation.Aspect;

import org.aspectj.lang.annotation.Before;

@Aspect

public class LoggingAspect {

@Before("execution(\* org.example.BookService.displayBooks(..))")

public void logBeforeDisplay(JoinPoint joinPoint) {

System.out.println("LOG: " + joinPoint.getSignature().getName() + "() method is about to be called.");

}

}

**Application.properties:**

<?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="bookRepository" class="org.example.BookRepository" />

<bean id="bookService" class="org.example.BookService">

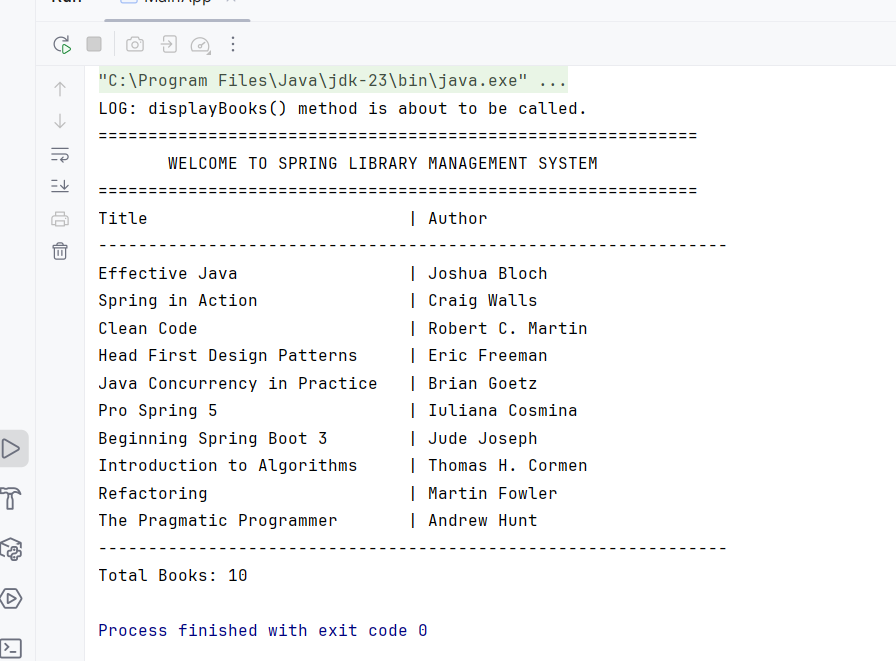
<property name="bookRepository" ref="bookRepository" />

</bean>

<bean id="loggingAspect" class="org.example.LoggingAspect" />

</beans>

**Output:**

****

**Conclusion:**

This project effectively showcases how to integrate **Spring Core with AOP** in a clean, maintainable architecture. By leveraging **Maven for dependency management** and Spring's IoC container, the application achieves better modularity, separation of concerns, and enhanced logging through aspects.