**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.

 <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>

    <build>

    <plugins>

        <plugin>

            <groupId>org.codehaus.mojo</groupId>

            <artifactId>exec-maven-plugin</artifactId>

            <version>3.1.0</version>

        </plugin>

    </plugins>

</build>

    <groupId>com.library</groupId>

    <artifactId>LibraryManagement</artifactId>

    <version>1.0-SNAPSHOT</version>

    <dependencies>

        <dependency>

            <groupId>org.springframework</groupId>

            <artifactId>spring-context</artifactId>

            <version>5.3.32</version>

        </dependency>

    </dependencies>

</project>

1. **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.
2. **Define Service and Repository Classes:**
   * Create a package **com.library.service** and add a 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 addBook(String bookName) {

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

        bookRepository.saveBook(bookName);

    }

}

* + Create a package **com.library.repository** and add a class **BookRepository**.

package com.library.repository;

public class BookRepository {

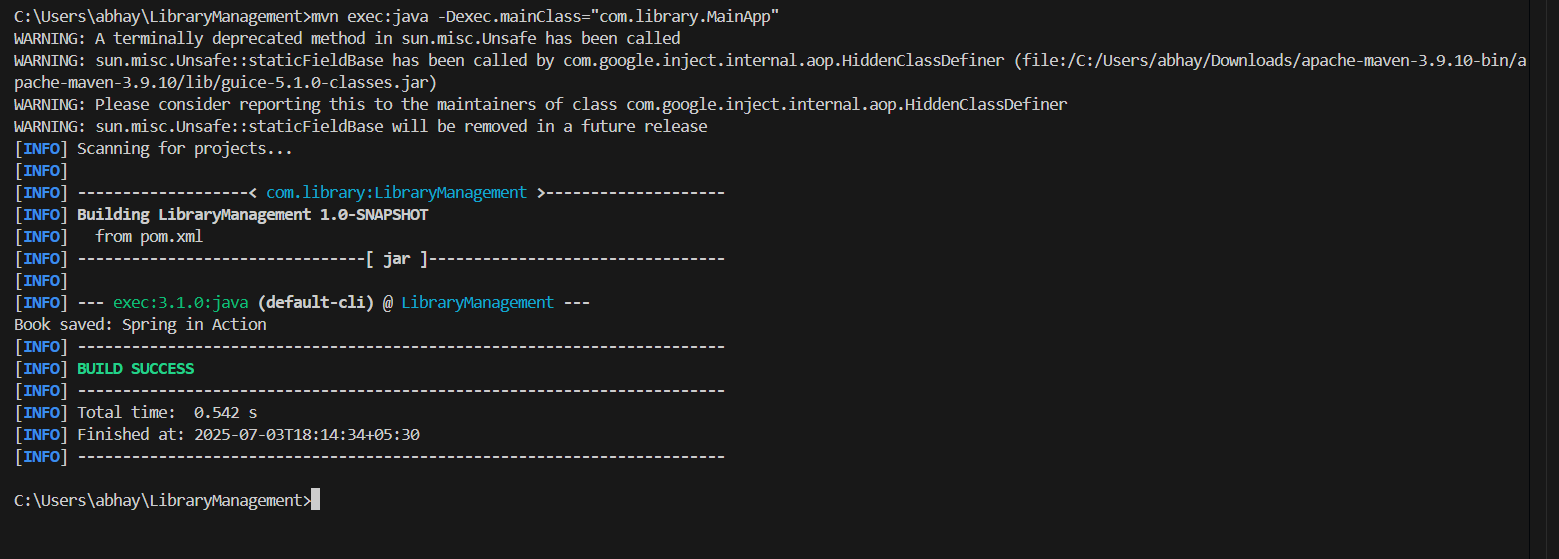
    public void saveBook(String bookName) {

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

    }

}

1. **Run the Application:**
   * Create a main class to load the Spring context and test the configuration.

Test 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**.

<?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 with Dependency Injection -->

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

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

    </bean>

</beans>

1. **Update the BookService Class:**
   * Ensure that **BookService** class has a setter method for **BookRepository**.

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

    private BookRepository bookRepository;

    // Setter for dependency injection

    public void setBookRepository(BookRepository bookRepository) {

        this.bookRepository = bookRepository;

    }

    public void addBook(String bookName) {

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

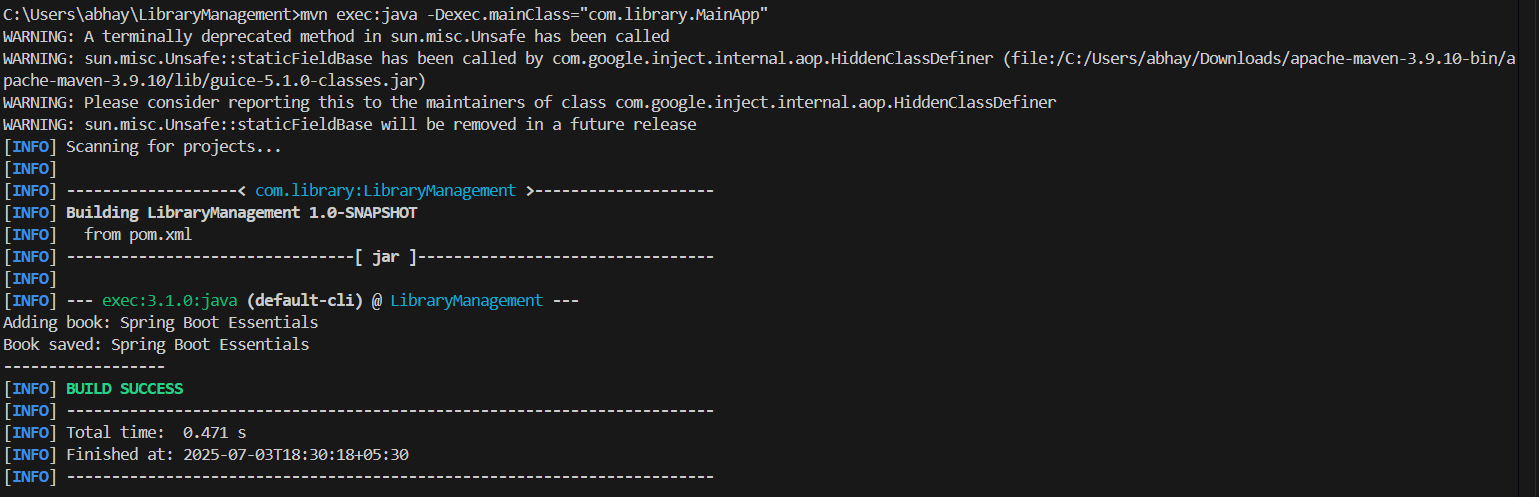
        bookRepository.saveBook(bookName);

    }

}

1. **Test the Configuration:**
   * Run the **LibraryManagementApplication** main class to verify the dependency injection.

Test output:



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

**Scenario:**

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

**Steps:**

1. **Add Spring AOP Dependency:**
   * Update **pom.xml** to include Spring AOP dependency.

 <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>

    <build>

    <plugins>

        <plugin>

            <groupId>org.codehaus.mojo</groupId>

            <artifactId>exec-maven-plugin</artifactId>

            <version>3.1.0</version>

        </plugin>

    </plugins>

</build>

    <groupId>com.library</groupId>

    <artifactId>LibraryManagement</artifactId>

    <version>1.0-SNAPSHOT</version>

    <dependencies>

    <dependency>

        <groupId>org.springframework</groupId>

        <artifactId>spring-context</artifactId>

        <version>5.3.32</version>

    </dependency>

    <dependency>

        <groupId>org.springframework</groupId>

        <artifactId>spring-aop</artifactId>

        <version>5.3.32</version>

    </dependency>

    <dependency>

        <groupId>org.aspectj</groupId>

        <artifactId>aspectjrt</artifactId>

        <version>1.9.21</version>

    </dependency>

    <dependency>

        <groupId>org.aspectj</groupId>

        <artifactId>aspectjweaver</artifactId>

        <version>1.9.21</version>

    </dependency>

</dependencies>

</project>

1. **Create an Aspect for Logging:**
   * Create a package **com.library.aspect** and add a class **LoggingAspect** with a method to log execution times.

package com.library.aspect;

import org.aspectj.lang.ProceedingJoinPoint;

import org.aspectj.lang.annotation.Around;

import org.aspectj.lang.annotation.Aspect;

@Aspect

public class LoggingAspect {

    @Around("execution(\* com.library.service.\*.\*(..))")

    public Object logExecutionTime(ProceedingJoinPoint joinPoint) throws Throwable {

        long start = System.currentTimeMillis();

        Object result = joinPoint.proceed();  // proceed with method

        long end = System.currentTimeMillis();

        System.out.println("[AOP] Method " + joinPoint.getSignature() + " executed in " + (end - start) + " ms");

        return result;

    }

}

1. **Enable AspectJ Support:**
   * Update **applicationContext.xml** to enable **AspectJ** support and register the aspect.

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

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

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

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

    <!-- Enable annotation scanning and AOP -->

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

    <aop:aspectj-autoproxy/>

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

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

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

    </bean>

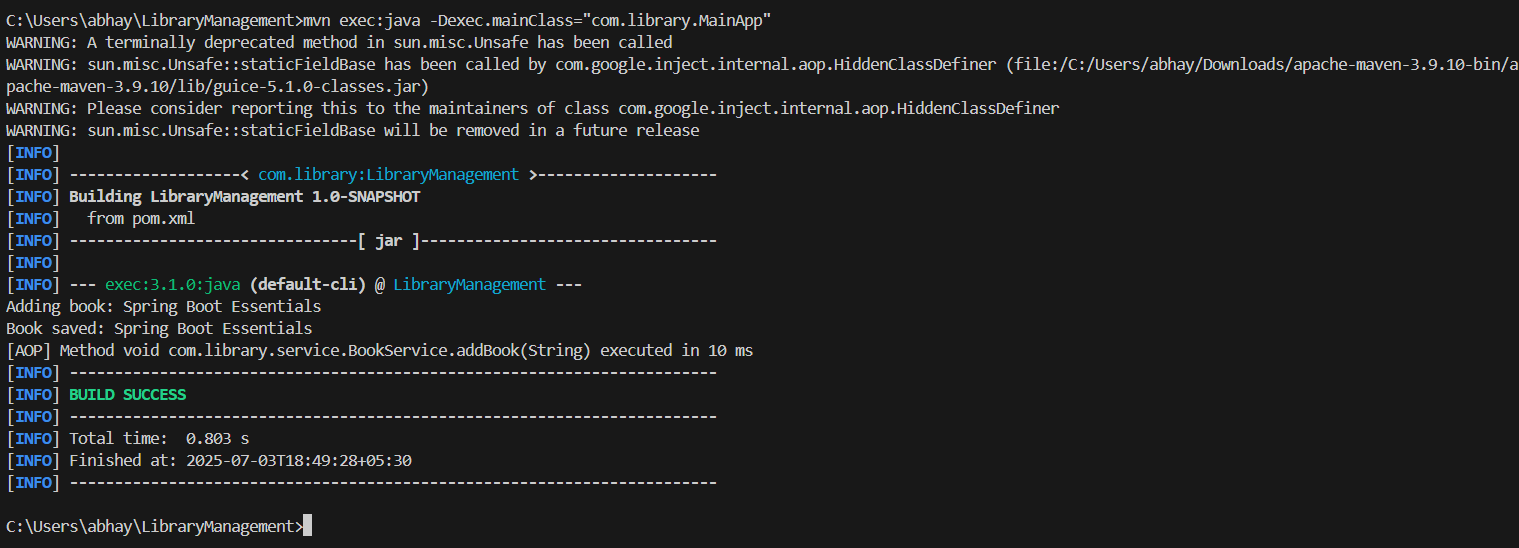
    <!-- Register the Aspect -->

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

</beans>

1. **Test the Aspect:**
   * Run the **LibraryManagementApplication** main class and observe the console for log messages indicating method execution times.

Test 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.

<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>

    </properties>

    <dependencies>

        <!-- Spring Core -->

        <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>

        <dependency>

            <groupId>org.aspectj</groupId>

            <artifactId>aspectjrt</artifactId>

            <version>1.9.21</version>

        </dependency>

        <dependency>

            <groupId>org.aspectj</groupId>

            <artifactId>aspectjweaver</artifactId>

            <version>1.9.21</version>

        </dependency>

        <!-- Spring WebMVC (newly added) -->

        <dependency>

            <groupId>org.springframework</groupId>

            <artifactId>spring-webmvc</artifactId>

            <version>5.3.32</version>

        </dependency>

    </dependencies>

    <build>

        <plugins>

            <!-- Maven Compiler Plugin -->

            <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>

            <!-- Exec plugin (for running main method) -->

            <plugin>

                <groupId>org.codehaus.mojo</groupId>

                <artifactId>exec-maven-plugin</artifactId>

                <version>3.1.0</version>

            </plugin>

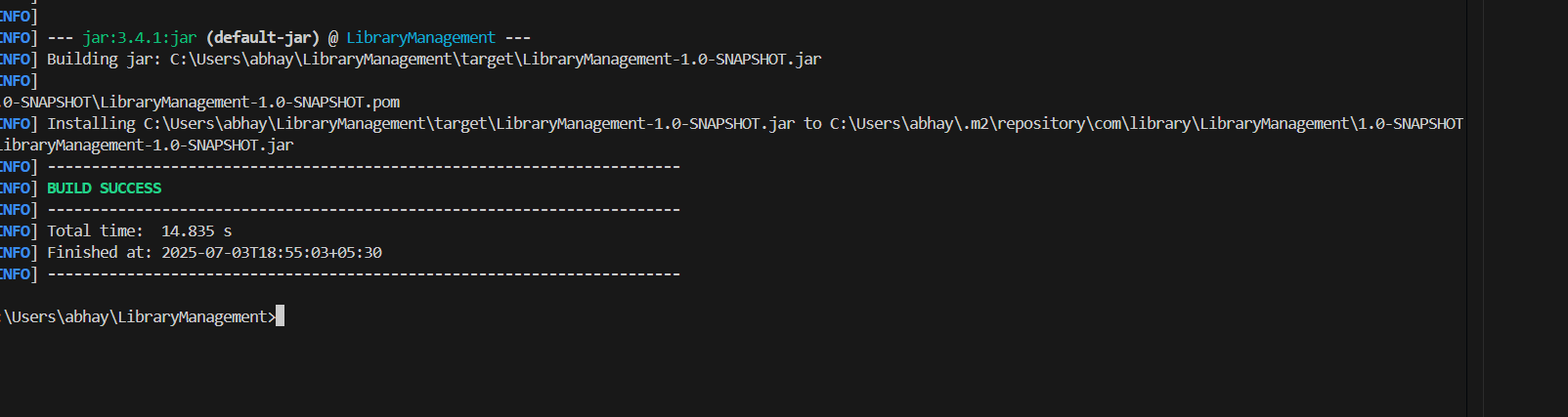
        </plugins>

    </build>

</project>

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

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.

<?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 BookRepository bean -->

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

    <!-- Define BookService bean and inject bookRepository -->

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

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

    </bean>

</beans>

* + Define beans for **BookService** and **BookRepository** in the XML file.

1. **Update the BookService Class:**
   * Ensure that the **BookService** class has a setter method for **BookRepository**.

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

    private BookRepository bookRepository;

    // Setter for DI

    public void setBookRepository(BookRepository bookRepository) {

        this.bookRepository = bookRepository;

    }

    public void addBook(String bookName) {

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

        bookRepository.saveBook(bookName);

    }

}

1. **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) {

        // Load Spring IoC container

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

        // Get BookService bean from context

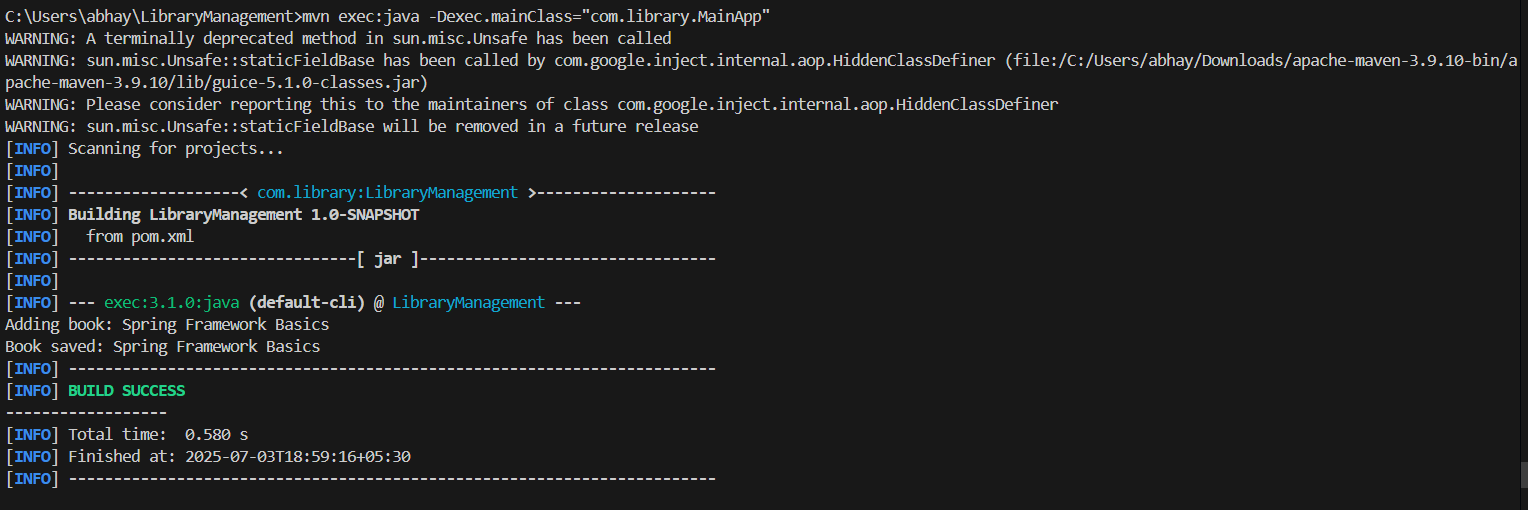
        BookService service = (BookService) context.getBean("bookService");

        service.addBook("Spring Framework Basics");

    }

}

Output:



**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.

<?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">

    <!-- Enable annotation-based configuration -->

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

</beans>

1. **Annotate Classes:**
   * Use **@Service** annotation for the **BookService** class.

package com.library.service;

import com.library.repository.BookRepository;

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

import org.springframework.stereotype.Service;

@Service

public class BookService {

    @Autowired

    private BookRepository bookRepository;

    public void addBook(String bookName) {

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

        bookRepository.saveBook(bookName);

    }

}

* + Use **@Repository** annotation for the **BookRepository** class.

package com.library.repository;

import org.springframework.stereotype.Repository;

@Repository

public class BookRepository {

    public void saveBook(String bookName) {

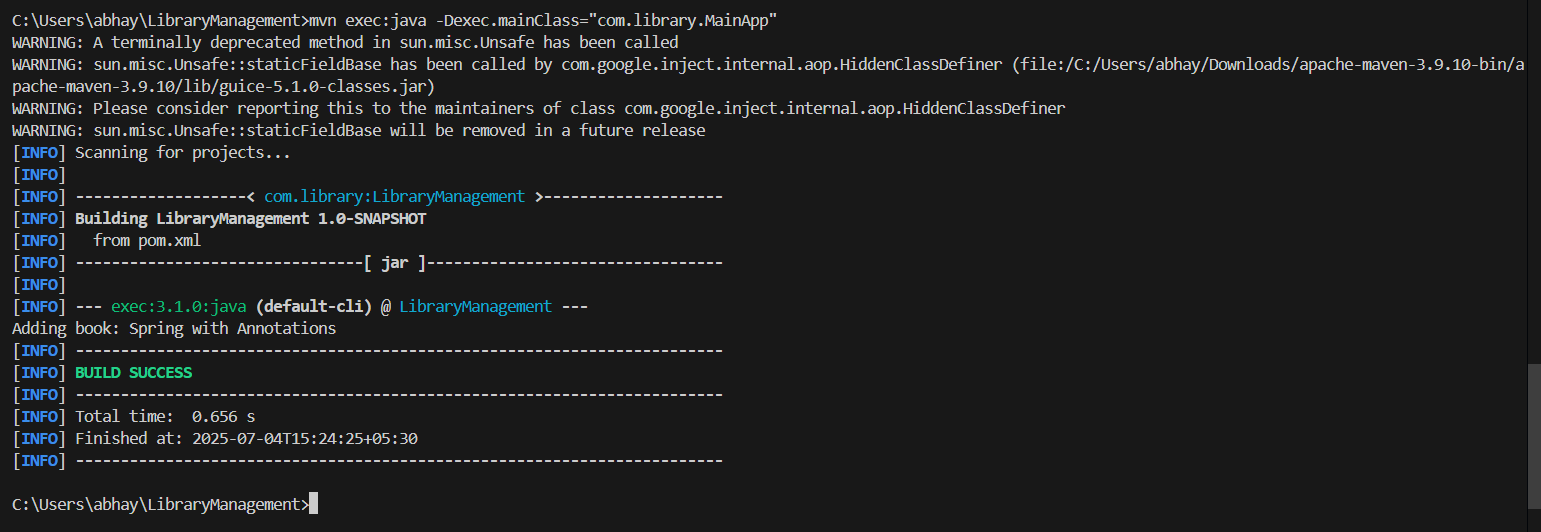
        System.out.println("Book saved: " + bookName);

    }

}

1. **Test the Configuration:**
   * Run the **LibraryManagementApplication** main class to verify the annotation-based configuration.

**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.**

<?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 repository bean -->

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

    <!-- Define BookService with constructor + setter injection -->

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

        <!-- Constructor Injection -->

        <constructor-arg value="LibraryService V1"/>

        <!-- Setter Injection -->

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

    </bean>

</beans>

1. **Configure Setter Injection:**
   * **Ensure that the BookService class has a setter method for BookRepository and configure it in applicationContext.xml.**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

    private String serviceName;                // Constructor-injected

    private BookRepository bookRepository;     // Setter-injected

    // Constructor

    public BookService(String serviceName) {

        this.serviceName = serviceName;

        System.out.println("BookService initialized with name: " + serviceName);

    }

    // Setter for DI

    public void setBookRepository(BookRepository bookRepository) {

        this.bookRepository = bookRepository;

    }

    public void addBook(String bookName) {

        System.out.println("[" + serviceName + "] Adding book: " + bookName);

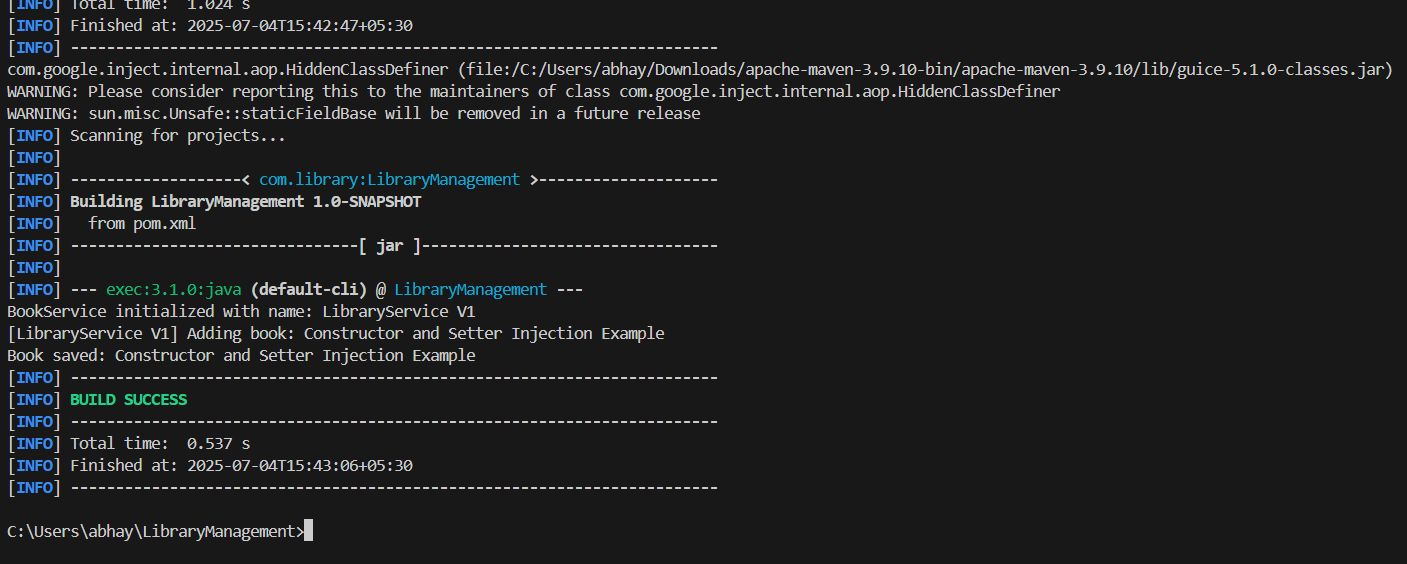
        bookRepository.saveBook(bookName);

    }

}

1. **Test the Injection:**
   * **Run the LibraryManagementApplication main class to verify both constructor and setter injection.**

**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.**

**Steps:**

1. **Define an Aspect:**
   * **Create a package com.library.aspect and add a class LoggingAspect.**
2. **Create Advice Methods:**
   * **Define advice methods in LoggingAspect for logging before and after method execution.**

package com.library.aspect;

import org.aspectj.lang.JoinPoint;

import org.aspectj.lang.annotation.After;

import org.aspectj.lang.annotation.Aspect;

import org.aspectj.lang.annotation.Before;

import org.springframework.stereotype.Component;

@Aspect

@Component

public class LoggingAspect {

    @Before("execution(\* com.library.service.BookService.\*(..))")

    public void beforeMethod(JoinPoint joinPoint) {

        System.out.println("[AOP Before] Method called: " + joinPoint.getSignature().getName());

    }

    @After("execution(\* com.library.service.BookService.\*(..))")

    public void afterMethod(JoinPoint joinPoint) {

        System.out.println("[AOP After] Method executed: " + joinPoint.getSignature().getName());

    }

}

1. **Configure the Aspect:**
   * **Update applicationContext.xml to register the aspect and enable AspectJ auto-proxying.**

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

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

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

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

    <!-- Enable component scanning -->

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

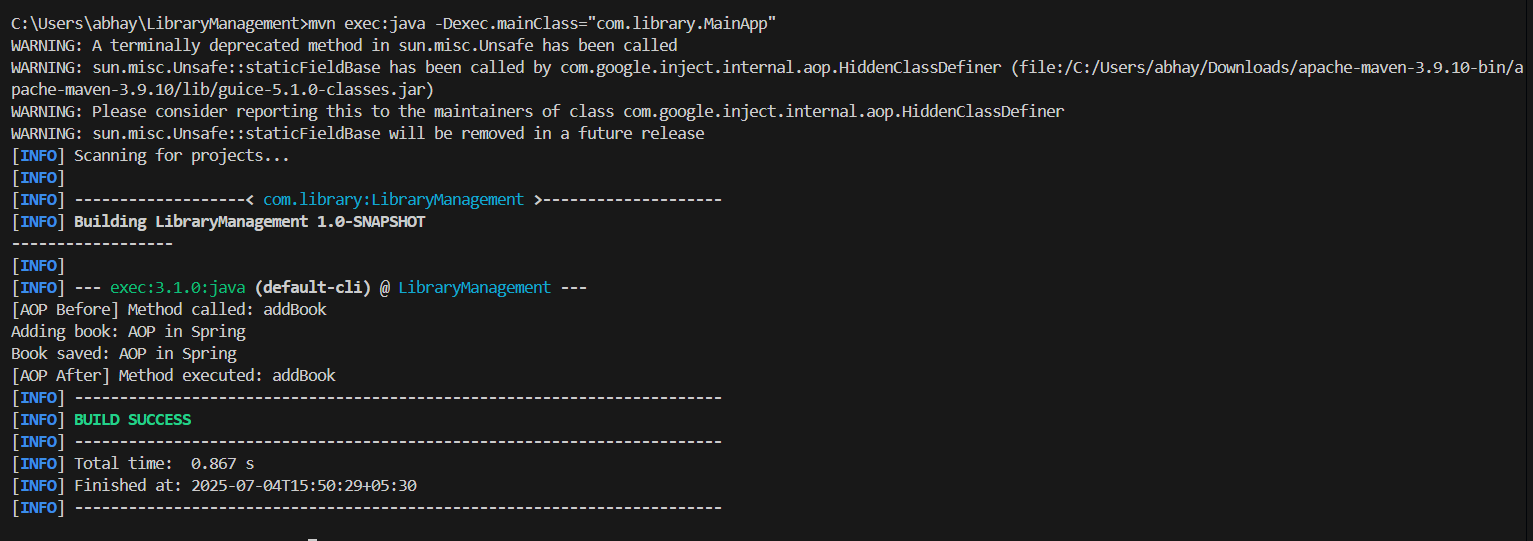
    <!-- Enable AOP proxy support -->

    <aop:aspectj-autoproxy/>

</beans>

1. **Test the Aspect:**
   * **Run the LibraryManagementApplication main class to verify the AOP functionality.**

**Output:**

****