**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:**
   1. Create a Maven project named **LibraryManagement**.
   2. Add Spring Core dependencies in the **pom.xml** file.
2. **Configure the Application Context:**
   1. Create an XML configuration file named **applicationContext.xml** in the **src/main/resources** directory.
   2. Define beans for **BookService** and **BookRepository** in the XML file.
3. **Define Service and Repository Classes:**
   1. Create a package **com.library.service** and add a class **BookService**.
   2. Create a package **com.library.repository** and add a class **BookRepository**.
4. **Run the Application:**
   1. Create a main class to load the Spring context and test the configuration.

**Code**

**BookRepository.java**

package com.library.repository;

public class BookRepository {

public void saveBook(String bookName) {

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

}

}

**BookService.java**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

// Setter (used for dependency injection via XML)

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void addBook(String bookName) {

System.***out***.println("Service: Adding book...");

bookRepository.saveBook(bookName);

}

}

**MainApp.java**

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) {

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

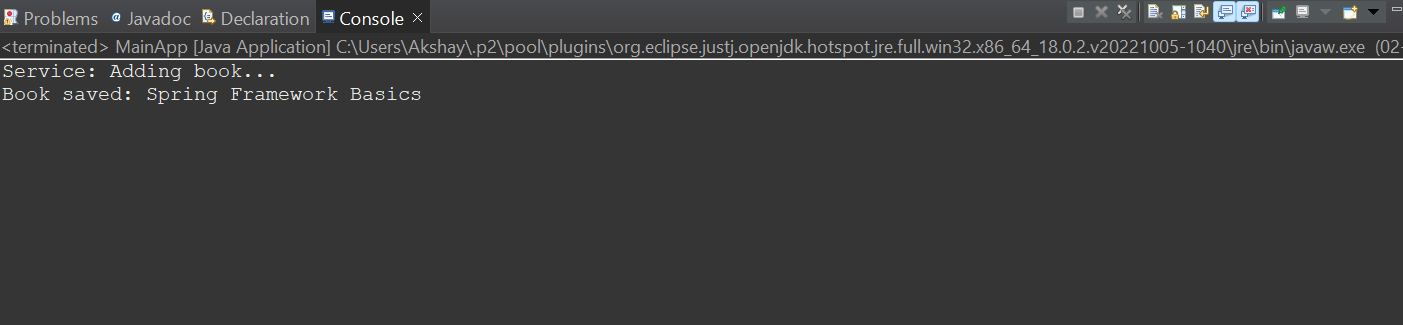
BookService bookService = context.getBean("bookService", BookService.class);

bookService.addBook("Spring Framework Basics");

}

}

**Output:**



**Exercise 2 :Same as Ex1.**

**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:**
   1. Create a new Maven project named **LibraryManagement**.
2. **Add Spring Dependencies in pom.xml:**
   1. Include dependencies for Spring Context, Spring AOP, and Spring WebMVC.
3. **Configure Maven Plugins:**
   1. Configure the Maven Compiler Plugin for Java version 1.8 in the pom.xml file.

**Code**

**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 -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.34</version>

</dependency>

<!-- Spring AOP -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.34</version>

</dependency>

<!-- Spring Web MVC -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.34</version>

</dependency>

</dependencies>

<build>

<plugins>

<!-- Maven Compiler Plugin -->

<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>

</project>

**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:**
   1. Create an XML configuration file named **applicationContext.xml** in the **src/main/resources** directory.
   2. Define beans for **BookService** and **BookRepository** in the XML file.
2. **Update the BookService Class:**
   1. Ensure that the **BookService** class has a setter method for **BookRepository**.
3. **Run the Application:**
   1. Create a main class to load the Spring context and test the configuration.

**Code:**

**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">

<!-- BookRepository Bean -->

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

<!-- BookService Bean (depends on BookRepository) -->

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

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

</bean>

</beans>

**MainApp.java**

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) {

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

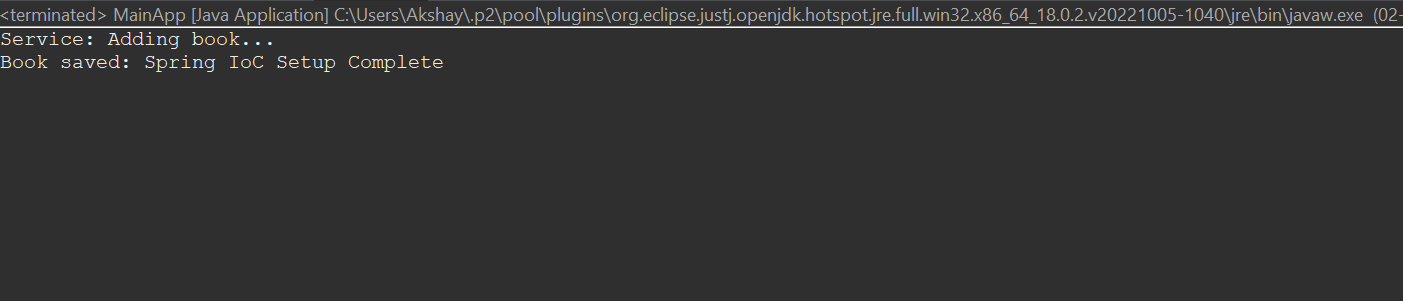
BookService bookService = context.getBean("bookService", BookService.class);

bookService.addBook("Spring IoC Setup Complete");

}

}

**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:**
   1. Update applicationContext.**xml** to configure constructor injection for **BookService**.
2. **Configure Setter Injection:**
   1. Ensure that the **BookService** class has a setter method for **BookRepository** and configure it in **applicationContext.xml**.
3. **Test the Injection:**
   1. Run the **LibraryManagementApplication** main class to verify both constructor and setter injection.

**Code**

**BookService.java**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService

private BookRepository bookRepository;

private String serviceName;

public BookService(String serviceName) {

this.serviceName = serviceName;

}

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void addBook(String bookName) {

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

bookRepository.saveBook(bookName);

}

}

**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">

<!-- BookRepository Bean -->

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

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

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

<!-- Constructor injection -->

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

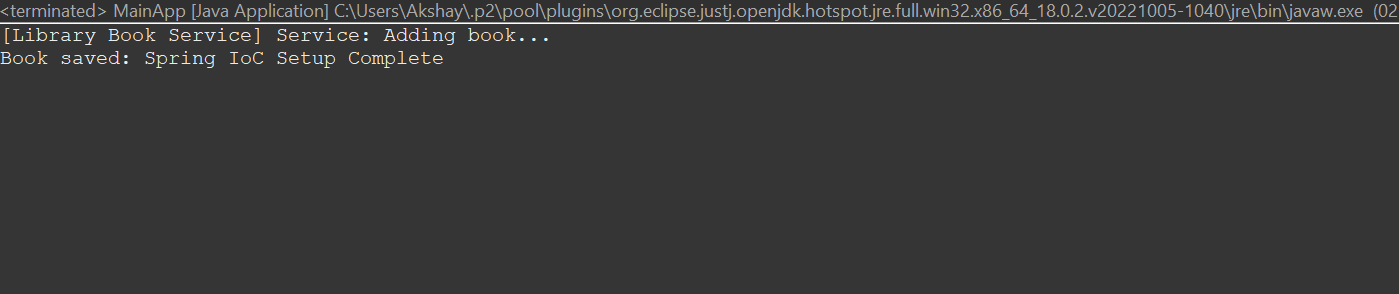
<!-- Setter injection -->

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

</bean>

</beans>

**Output:**

****