RUBINA S SUPERSET ID:6383615

# 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.

## 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.

# SOLUTION

## 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>

<packaging>jar</packaging>

<version>1.0-SNAPSHOT</version>

<name>LibraryManagement</name>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.36</version>

</dependency>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

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

<version>3.1.0</version>

<configuration>

<mainClass>com.library.main.MainApp</mainClass>

</configuration>

</plugin>

</plugins>

</build>

</project>

## MainApp.java

package com.library.main;

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 = (BookService) context.getBean("bookService"); bookService.addBook("Clean Code");

}}

## 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 addBook(String bookName) { System.out.println("Adding book: " + bookName); bookRepository.saveBook(bookName);

}

}

## BookRepository.java

package com.library.repository; public class BookRepository {

public void saveBook(String bookName) {

System.out.println("Book \"" + bookName + "\" saved to the database.");

}

}

## 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>

## AppTest.java

package com.library; import org.junit.Test;

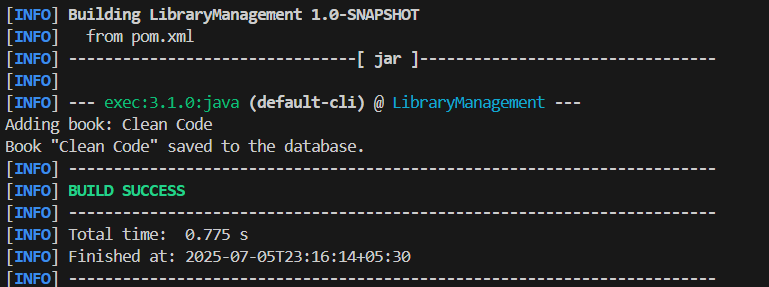
public class AppTest { @Test

public void testApp() { System.out.println("Test executed.");

}

}

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

# SOLUTION

Use Spring’s Inversion of Control (IoC) to inject the BookRepository into BookService using an XML file (applicationContext.xml).

## 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>

## 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 addBook(String title) { System.out.println("Adding book: " + title); bookRepository.save(title);

}

}

## 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

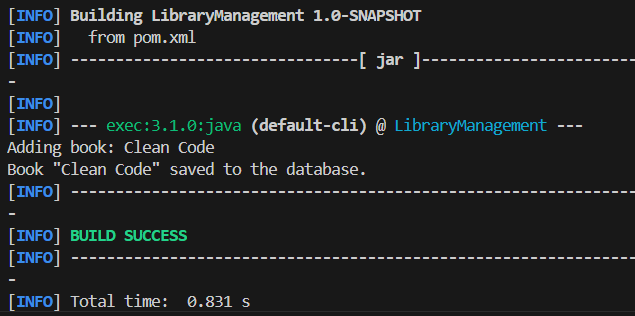
ClassPathXmlApplicationContext("applicationContext.xml");

BookService bookService = (BookService) context.getBean("bookService"); bookService.addBook("Clean Code");

}

}

# 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.

# SOLUTION

## 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 Context -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.33</version>

</dependency>

<!-- Spring AOP -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.33</version>

</dependency>

<!-- Spring WebMVC -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.33</version>

</dependency>

<!-- JUnit -->

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>4.13.2</version>

<scope>test</scope>

</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>

<!-- Exec Plugin to run main class -->

<plugin>

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

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

<version>3.1.0</version>

<executions>

<execution>

<goals>

<goal>java</goal>

</goals>

</execution>

</executions>

</plugin>

</plugins>

</build>

</project>

## MainApp.java

package com.library.main;

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 = (BookService) context.getBean("bookService"); bookService.addBook("Clean Code");

}

}

## 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 addBook(String bookName) { System.out.println("Adding book: " + bookName); bookRepository.saveBook(bookName);

}

}

## BookRepository.java

package com.library.repository;

public class BookRepository {

public void saveBook(String bookName) {

System.out.println("Book \"" + bookName + "\" saved to the database.");

}

}

## 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" />

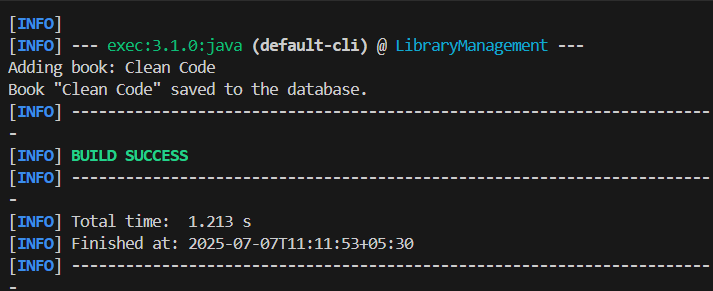
<bean id="bookService" class="com.library.service.BookService">

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

</bean>

</beans>

**OUTPUT**

****