**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:

o Create a Maven project named LibraryManagement.

o Add Spring Core dependencies in the pom.xml file.

2. Configure the Application Context:

o Create an XML configuration file named applicationContext.xml in the src/main/resources directory.

o Define beans for BookService and BookRepository in the XML file.

3. Define Service and Repository Classes:

o Create a package com.library.service and add a class BookService.

o Create a package com.library.repository and add a class BookRepository.

4. Run the Application:

o Create a main class to load the Spring context and test the configuration.

**CODE:**

// Spring project

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

<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 https://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>

<name>LibraryManagement</name>

<properties>

<java.version>11</java.version>

<spring.version>5.3.20</spring.version>

</properties>

<dependencies>

*<!-- Spring Core -->*

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>${spring.version}</version>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.8.1</version>

<configuration>

<source>${java.version}</source>

<target>${java.version}</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

// Application context

<?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

https://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>

//service and repository

package com.library.repository;

public class BookRepository {

*// This would normally contain database operations*

*// For now, we'll just add a simple method for demonstration*

public String getBookInfo(String isbn) {

return "Book details for ISBN: " + isbn;

}

}

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 String getBookDetails(String isbn) {

return bookRepository.getBookInfo(isbn);

}

}

//run

**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:

o Update applicationContext.xml to wire BookRepository into BookService.

2. Update the BookService Class:

o Ensure that BookService class has a setter method for BookRepository.

3. Test the Configuration:

o Run the LibraryManagementApplication main class to verify the dependency injection.

**CODE:**

**//**xml config

<?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

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

*<!-- BookRepository bean definition -->*

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

*<!-- You could add property configurations here if needed -->*

</bean>

*<!-- BookService bean definition with explicit dependency injection -->*

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

*<!-- Setter-based dependency injection -->*

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

*<!-- Alternatively, you could use constructor injection:*

*<constructor-arg ref="bookRepository"/>*

*-->*

</bean>

</beans>

//update bookservice class

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

*// Setter method for dependency injection*

public void setBookRepository(BookRepository bookRepository) {

System.out.println("Dependency Injection: Setting BookRepository");

this.bookRepository = bookRepository;

}

*// Business method that uses the repository*

public String getBookDetails(String isbn) {

if (bookRepository == null) {

throw new IllegalStateException("BookRepository dependency not injected!");

}

return bookRepository.getBookInfo(isbn);

}

*// Additional business methods can be added here*

public int getTotalBooks() {

*// This would normally come from the repository*

return 100; *// Example value*

}

}

// update bookrepository class

package com.library.repository;

public class BookRepository {

*// Simulated database operations*

public String getBookInfo(String isbn) {

return "Book details for ISBN: " + isbn + " (Title: Spring in Action, Author: Craig Walls)";

}

public int countAllBooks() {

*// Simulated count*

return 100;

}

public boolean isBookAvailable(String isbn) {

*// Simulated availability check*

return true;

}

}

//test

package com.library;

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) {

System.out.println("Starting Library Management Application...");

*// Load the Spring application context*

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

*// Retrieve the BookService bean*

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

*// Test the service methods*

System.out.println("\n=== Testing Book Service ===");

String bookDetails = bookService.getBookDetails("123-456-789");

System.out.println("Book Details: " + bookDetails);

int totalBooks = bookService.getTotalBooks();

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

*// Verify the dependency was properly injected*

System.out.println("\n=== Dependency Injection Verification ===");

try {

bookService.getBookDetails("TEST-ISBN");

System.out.println("Dependency injection successful - BookRepository is available");

} catch (IllegalStateException e) {

System.err.println("Dependency injection failed: " + e.getMessage());

}

*// Close the context*

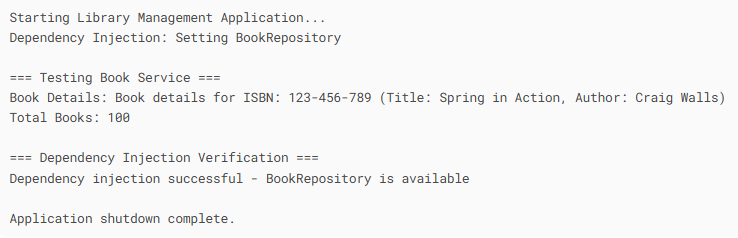
((ClassPathXmlApplicationContext) context).close();

System.out.println("\nApplication shutdown complete.");

}

}

**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:

o Create a new Maven project named LibraryManagement.

2. Add Spring Dependencies in pom.xml:

o Include dependencies for Spring Context, Spring AOP, and Spring WebMVC.

3. Configure Maven Plugins:

o Configure the Maven Compiler Plugin for Java version 1.8 in the pom.xml file.

**CODE:**

**//** xml with dependencies

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

<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>

<packaging>jar</packaging>

<name>LibraryManagement</name>

<description>Library Management System using Spring Framework</description>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<java.version>1.8</java.version>

<spring.version>5.3.20</spring.version>

</properties>

<dependencies>

*<!-- Spring Context (includes Core, Beans, Expression) -->*

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>${spring.version}</version>

</dependency>

*<!-- Spring AOP -->*

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>${spring.version}</version>

</dependency>

*<!-- Spring Web MVC -->*

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>${spring.version}</version>

</dependency>

*<!-- For testing -->*

<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>${java.version}</source>

<target>${java.version}</target>

</configuration>

</plugin>

*<!-- Maven Surefire Plugin for testing -->*

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-surefire-plugin</artifactId>

<version>2.22.2</version>

</plugin>

</plugins>

</build>

</project>

**OUTPUT:**

