**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.

**Code:**

**project1/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</version>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.32</version>

</dependency>

</dependencies>

</project>

**project1/src/main/java/com/library/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 service = context.getBean(BookService.class);

service.displayBook();

}

}

**project1/src/main/java/com/library/service/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 displayBook() {

System.out.println("Book Found: " + bookRepository.findBook());

}

}

**project1/src/main/java/com/library/repository/bookrepository.java**

package com.library.repository;

public class BookRepository {

public String findBook() {

return "Spring in Action";

}

}

**project1/src/main/resources/applicationContext.xml**

<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"/>

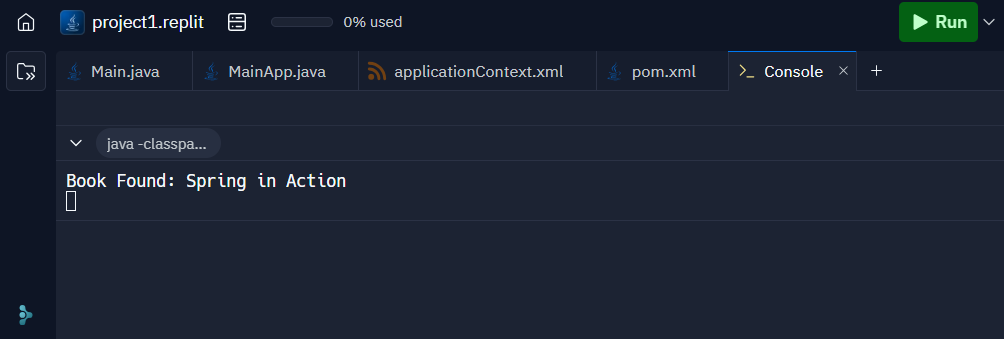
<bean id="bookService" class="com.library.service.BookService">

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

</bean>

</beans>

**Output:**

****

**Explanation:**

* I made a Java project using Maven to administer a library system, and I included a Spring Core dependency to use the Spring Framework to manage the backend.
* I created a BookService class to use that repository for business logic and a BookRepository class to mimic data access.
* Without implementing any functionality inside the main class, I configured Spring beans and carried out dependency injection using an XML file called applicationContext.xml.
* The service method, which accesses the repository internally to retrieve the book name, is called by the main class (MainApp) after loading the Spring context.
* Spring correctly wired the classes and printed the result when I ran the program, demonstrating that the configuration and bean setup were accurate.

**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.

**Code:**

**project1/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</version>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.32</version>

</dependency>

</dependencies>

</project>

**project1/src/main/java/com/library/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 service = context.getBean(BookService.class);

service.displayBook();

}

}

**project1/src/main/java/com/library/service/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 displayBook() {

System.out.println("Book Found: " + bookRepository.findBook());

}

}

**project1/src/main/java/com/library/repository/bookrepository.java**

package com.library.repository;

public class BookRepository {

public String findBook() {

return "Spring in Action";

}

}

**project1/src/main/resources/applicationContext.xml**

<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"/>

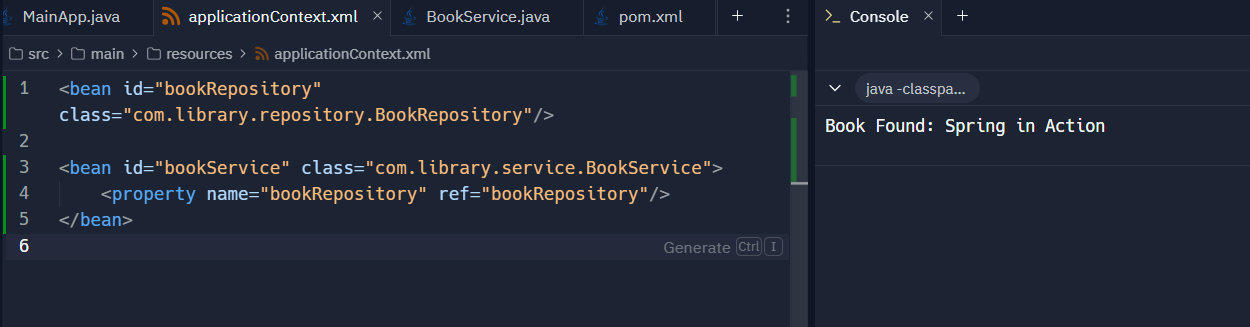
<bean id="bookService" class="com.library.service.BookService">

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

</bean>

</beans>

**Output:**



**Explanation:**

* For this work, I managed the connection between the BookService and BookRepository classes using Spring's Dependency Injection functionality.
* In order to enable Spring to automatically inject the repository into the service, I modified the applicationContext.xml file to wire these two classes using the element.
* For Spring to carry out the injection, I made sure the BookService class has a public setter function named setBookRepository().
* To check if the wiring was correct, I imported the Spring context from the XML file into the MainApp class and retrieved the BookService bean.
* The program's output verified that dependency injection had occurred correctly when I executed it, demonstrating that Spring managed the object relationships without the need for human instantiation.

**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.

**Code:**

**exercise4-week3/src/main/java/Main.java**

package com.library;

public class Main {

public static void main(String[] args) {

System.out.println("Library Management System Started!");

System.out.println("Spring dependencies loaded successfully.");

}

}

**exercise4-week3/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>

<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.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.32</version>

</dependency>

</dependencies>

<build>

<plugins>

<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>

<plugin>

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

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

<version>3.1.0</version>

<configuration>

<mainClass>com.library.Main</mainClass>

</configuration>

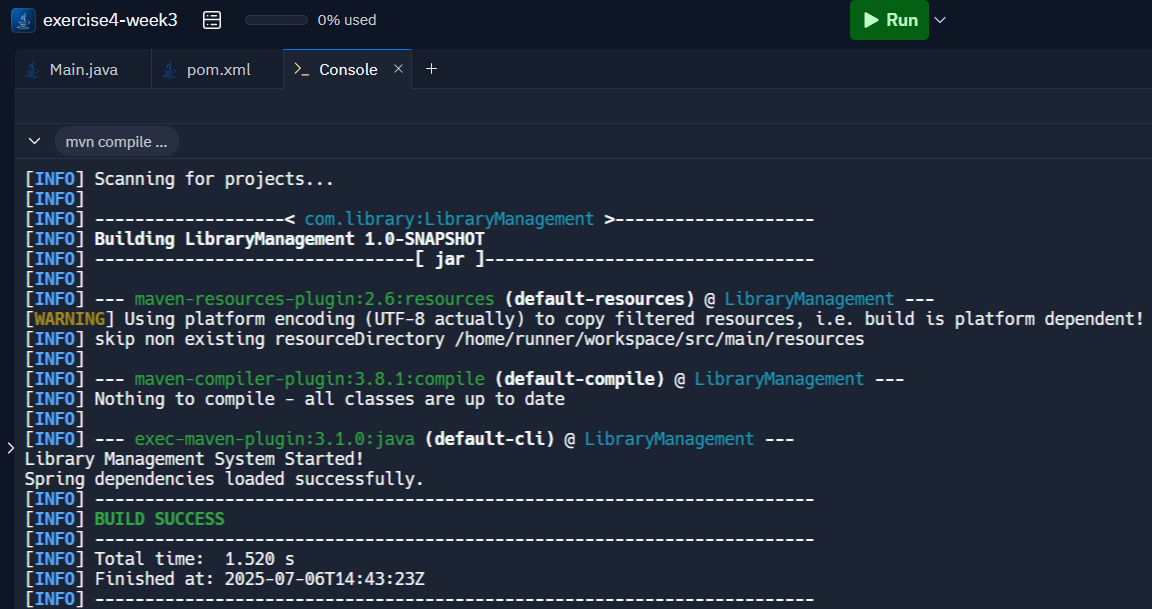
</plugin>

</plugins>

</build>

</project>

**Output:**

****

**Explanation:**

* I used the typical folder structure (src/main/java and pom.xml) appropriate for Spring apps to create a Maven project called LibraryManagement.
* In order to enable core functionality, aspect-based programming, and web controllers, I included the necessary Spring dependencies—Spring Context, AOP, and WebMVC—in the pom.xml file.
* To ensure that all classes compile correctly and continue to work with the Spring version, I also set up the Maven Compiler Plugin to utilize Java version 1.8.
* I set the main class to com.library and applied the exec-maven-plugin to make the project executable. main to use Maven to start the application.
* The output from my main class was successfully printed when I ran the project, indicating that the build, dependency loading, and execution were all operating as they should have.