**Exercise1: Configuring a Basic Spring Application**

**Scenario:**

A company is developing a web application for managing a library. It needs to use the Spring Framework to handle the backend operations.

In this,

* Create a Maven project named LibraryManagement.
* Configure Spring IOC using XML.
* Define BookService and BookRepository.
* Test the configuration with a simple main class.

Steps in construction of Basic Spring Application:

**Step1:** Set up a Spring Project.

Spring applications rely on IOC to manage objects. Hence Maven is used to manage dependencies and project structure.

**Project Structure:**

LibraryManagement/

├── src/

│ ├── main/

│ │ ├── java/

│ │ │ └── com/

│ │ │ └── library/

│ │ │ ├── service/

│ │ │ │ └── BookService.java

│ │ │ ├── repository/

│ │ │ │ └── BookRepository.java

│ │ │ └── MainApp.java

│ │ └── resources/

│ │ └── applicationContext.xml

├── pom.xml

After the creation of maven project i.e. file -> new -> Maven Project,

The **pom.xml** is:

|  |
| --- |
| <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>  <!-- SLF4J Simple Logger (Optional) -->  <dependency>  <groupId>org.slf4j</groupId>  <artifactId>slf4j-simple</artifactId>  <version>1.7.36</version>  </dependency>  </dependencies>  </project> |

Here,

spring-context: Core Spring module for IoC container.

Slf4j-simple: Logging support

**Step2:** Configure the Application.

Spring needs a configuration file to know which objects to manage and how to write them.

For defining beans applicationContext.xml is used.

The file path is:

src/main/resources/application.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">  <!-- Define BookRepository Bean -->  <bean id="bookRepository" class="com.library.repository.BookRepository"/>  <!-- Define BookService Bean with Dependency Injection -->  <bean id="bookService" class="com.library.service.BookService">  <property name="bookRepository" ref="bookRepository"/>  </bean>  </beans> |

In this file,

<bean> - defines Spring managed objects.

<property> - writes BookRepository into BookService (a setter injection).

**Step3:** Defining Service and Repository classes.

A layered approach is followed.

* Repository Layer: used for data operations.
* Service Layer: used to implement business logic.

Also, BookService.java

|  |
| --- |
| package com.library.service;  import com.library.repository.BookRepository;  public class BookService {  private BookRepository bookRepository;  // Setter Injection  public void setBookRepository (BookRepository bookRepository) {  this.bookRepository = bookRepository;  }  public void addBook (String bookName) {  System.out.println("BookService: Adding book - " + bookName);  bookRepository.saveBook(bookName);  }  } |

BookRepository.java

|  |
| --- |
| package com.library.repository;  public class BookRepository {  public void saveBook (String bookName) {  System.out.println("BookRepository: Saving book - " + bookName);  }  } |

Hence, BookService depends on BookRepository, setter method allows spring to inject the dependency defined in XML.

**Step4:** Run Application as Java Application.

To test, we create MainApp.java class as main class.

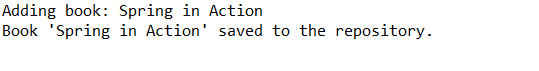
* Load spring content.
* Then, get beans from container.
* Later, call a method to check wiring.

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) {  // Load Spring Context from XML  ApplicationContext context =  new ClassPathXmlApplicationContext("applicationContext.xml");  // Get BookService Bean  BookService bookService = (BookService) context.getBean("bookService");  // Test adding a book  bookService.addBook("Spring in Action");  // Close Context  ((ClassPathXmlApplicationContext) context).close();  }  } |

Hence in the MainApp.java class we run it as java application once we update the maven project.

**Expected Outcome:**



Creation of LibraryManagement maven project overview:

