**Exercise2: Implementing Dependency Injection**

**Scenario:**

In the Library Management Application, Spring’s Inversion of Control and Dependency Injection is to be used to manage dependencies between BookService and BookRepository.

In this Application:

* Wire BookRepository into BookService using Spring IoC.
* Use Setter Injection.
* Test the configuration with a main class.

**Steps for Implementation Dependency Injection:**

**Step1**: Create a project named LibraryManagement.

Now create an .xml file i.e. path for applicationContext.xml is src/main/resources/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">  <!-- Define BookRepository Bean -->  <bean id="bookRepository" class="com.library.repository.BookRepository"/>  <!-- Define BookService Bean and wire BookRepository -->  <bean id="bookService" class="com.library.service.BookService">  <property name="bookRepository" ref="bookRepository"/>  </bean>  </beans> |

In this,

<property> tag tells Spring to inject bookRepository bean into bookService using setter injection.

**Step2:** Create the BookService.java and BookRepository.java

To enable setter Injection, BookService must have a setter for BookRepository.

BookRepository.java

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

BookService.java

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

The setter method allows the Spring to inject the dependency defined in applicationContext.xml

**Step3**: Test the Configuration

To verify, create a class naming MainApp.java:

* Loads Spring’s ApplicationContext
* Gets BookService bean
* Calls addBook() to check if dependency injection worked

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("Effective Java");  ((ClassPathXmlApplicationContext) context).close();  }  } |

**Step4**: Run the LibraryManagement Application.

Run As -> java application.

**Expected Output:**

