**Digital Nurture 4.0 – Week 3**

**Spring Core and Maven**

**Mandatory Hands-On**

**Exercise 1: Configuring a Basic Spring Application**

**Objective:**

Create a simple Spring project for **Library Management** using **Maven** and configure beans using **applicationContext.xml**.

**Step 1: Set Up Maven Project**

**IntelliJ :**

* Create Maven project → Name it LibraryManagement
* Group ID: com.library
* Artifact ID: LibraryManagement

**pom.xml**

<dependencies>

<!-- Spring Core Dependency -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.22</version> <!-- Use a stable version -->

</dependency>

</dependencies>

**Step 2:** **Configure 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 for Repository -->

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

<!-- Bean for Service (will use DI later) -->

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

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

</bean>

</beans>

**Step 3: Create Service and Repository Classes**

**BookRepository.java**

package com.library.repository;

public class BookRepository {

public void getBook() {

System.out.println("Fetching book from repository...");

}

}

BookService.java

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

// Setter for DI

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void displayBook() {

System.out.println("Service Layer: Displaying book...");

bookRepository.getBook();

}

}

**Step 4: Create Main Class to Test Configuration**

**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 Container

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

// Fetch bookService bean from Spring context

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

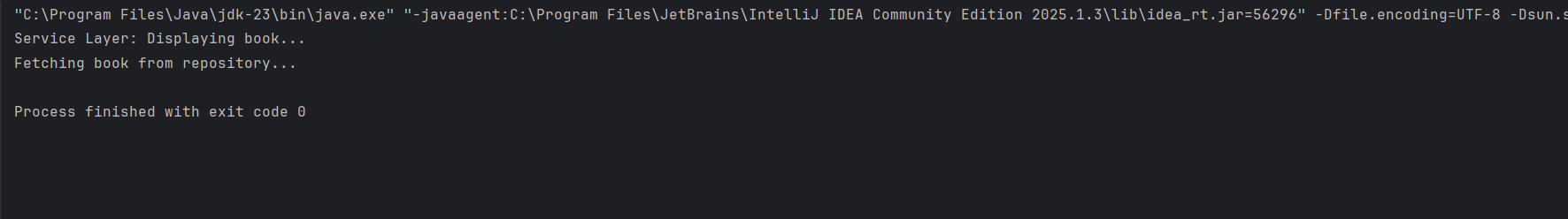
// Call a method to test the flow

service.displayBook();

}

}

**Output:**

****

**Exercise 2: Implementing Dependency Injection**

**Objective:**

Use **Spring’s IoC container** to automatically inject dependencies between BookService and BookRepository using **setter-based Dependency Injection** in applicationContext.xml.

**Step 1: Modify 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">

<!-- BookRepository Bean -->

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

<!-- BookService Bean with Setter Injection -->

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

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

</bean>

</beans>

**Step 2: Update BookService.java**

**BookService.java**

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) {

this.bookRepository = bookRepository;

System.out.println("BookRepository has been injected into BookService via Setter.");

}

public void displayBook() {

System.out.println("BookService: Initiating book display operation...");

bookRepository.getBook();

}

}

**Step 3: Update BookRepository.java**

**BookRepository.java**

package com.library.repository;

public class BookRepository {

public void getBook() {

System.out.println("BookRepository: Retrieving book details from the database...");

}

}

**Step 4: Run and Test using Main Class**

**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) {

System.out.println("Starting Library Management - Exercise 2: DI Demo");

// Load Spring container from XML

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

// Get BookService bean from Spring container

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

// Call the method to test DI

service.displayBook();

System.out.println("Dependency Injection test completed successfully.");

}

}

**Output:**

A screen shot of a computer

AI-generated content may be incorrect.

**Exercise 4: Creating and Configuring a Maven Project**

**Objective:**

Verify that Spring Framework is working correctly by:

* Creating a simple service class
* Registering it as a Spring bean
* Loading it from Spring container
* Printing output from a method

**Step 1: Create HelloService.java**

package com.library.service;

public class HelloService {

public void sayHello() {

System.out.println("Hello from Spring Framework! Dependency loaded successfully.");

}

}

**Step 2: Configure Spring Bean in 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">

<!-- Register HelloService as a Spring bean -->

<bean id="helloService" class="com.library.service.HelloService" />

</beans>

**Step 3: Create MainApp.java to Load Spring**

**MainApp.java**

package com.library;

import com.library.service.HelloService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

// Step 1: Load Spring container

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

// Step 2: Get bean from Spring

HelloService helloService = context.getBean("helloService", HelloService.class);

// Step 3: Call the method

helloService.sayHello();

}

}

**Output:**

A screen shot of a computer

AI-generated content may be incorrect.

**Additional Hands-On**

**Exercise 5: Configuring the Spring IoC Container**

**Objective:**

Set up the Spring IoC container using applicationContext.xml and wire the dependencies between BookService and BookRepository.

**Step 1: Create XML Configuration File**

**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">

<!-- BookRepository bean -->

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

<!-- BookService bean with DI (Setter Injection) -->

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

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

</bean>

</beans>

**Step 2: Create Java Classes**

**BookRepository.java**

package com.library.repository;

public class BookRepository {

public void getBook() {

System.out.println("📚 BookRepository: Fetching book...");

}

}

**BookService.java**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

// Setter method for DI

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void displayBook() {

System.out.println("🛠 BookService: Displaying book...");

bookRepository.getBook(); // Calling repository method

}

}

**Step 3: Create Main Class to Run**

**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 container from XML

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

// Get the BookService bean from Spring

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

// Call method to test the flow

service.displayBook();

}

}

**Output:**

A screen shot of a computer

AI-generated content may be incorrect.

**Exercise 7: Implementing Constructor and Setter Injection**

**Objective:**

Use both constructor and setter injection to initialize BookService in the Spring IoC container and verify proper dependency management.

**Step 1: Create BookRepository.java**

**BookRepository.java**

package com.library.repository;

public class BookRepository {

public void getBook() {

System.out.println("Repository Connected: Accessing book inventory database...");

System.out.println("Book retrieval successful!");

}

}

**Step 2: Create BookService.java**

**BookService.java**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

private String serviceName;

// Constructor Injection

public BookService(String serviceName) {

this.serviceName = serviceName;

}

// Setter Injection

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void displayBook() {

System.out.println("BookService Activated: " + serviceName);

bookRepository.getBook();

}

}

**Step 3: Update Spring Configuration**

**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">

<!-- BookRepository Bean -->

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

<!-- BookService Bean with Constructor + Setter Injection -->

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

<!-- Constructor Injection -->

<constructor-arg value="Library Book Service V1.0"/>

<!-- Setter Injection -->

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

</bean>

</beans>

**Step 4: Create and Run Main Class**

**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

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

// Get BookService Bean

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

// Call method to verify DI

service.displayBook();

}

}

**Output:**

A screen shot of a computer

AI-generated content may be incorrect.

**Exercise 9: Creating a Spring Boot Application**

**Objective:**

To develop a Spring Boot application named **LibraryManagement** that performs CRUD operations on books using **Spring Data JPA** and stores the data in a **MySQL database**.

**Step 1: Create Spring Boot Project**

* Go to <https://start.spring.io>
* Fill in the following:
  + **Group**: com.example
  + **Artifact**: demo
  + **Name**: LibraryManagement
  + **Package Name**: com.example.LibraryManagement
  + **Packaging**: Jar
  + **Java Version**: 21
  + **Dependencies**:
    - Spring Web
    - Spring Data JPA
    - MySQL Driver
* Click **Generate**, extract the ZIP, and open in **IntelliJ IDEA**.

**Step 2: Configure Dependencies in pom.xml**

**pom.xml**

<!-- Spring Web -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<!-- Spring Data JPA -->

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-data-jpa</artifactId>

</dependency>

<!-- MySQL Connector -->

<dependency>

<groupId>com.mysql</groupId>

<artifactId>mysql-connector-j</artifactId>

</dependency>

**Step 3: Configure application.properties**

**application.properties**

spring.datasource.url=jdbc:mysql://localhost:3306/librarydb

spring.datasource.username=root

spring.datasource.password=290319

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect

server.port=8080

**Step 4: Create MySQL Database**

CREATE DATABASE librarydb;

**Step 5: Create Book Entity Class**

**com.example.LibraryManagement.model.Book.java**

package com.example.LibraryManagement.model;

import jakarta.persistence.\*;

@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String title;

private String author;

private int yearPublished;

}

**Step 6: Create Repository Interface**

**com.example.LibraryManagement.repository.BookRepository.java**

package com.example.LibraryManagement.repository;

import com.example.LibraryManagement.model.Book;

import org.springframework.data.jpa.repository.JpaRepository;

public interface BookRepository extends JpaRepository<Book, Long> {

}

**Step 7: Create REST Controller**

**com.example.LibraryManagement.controller.BookController.java**

package com.example.LibraryManagement.controller;

import com.example.LibraryManagement.model.Book;

import com.example.LibraryManagement.repository.BookRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/books")

public class BookController {

@Autowired

private BookRepository bookRepository;

@PostMapping

public Book addBook(@RequestBody Book book) {

return bookRepository.save(book);

}

@GetMapping

public List<Book> getAllBooks() {

return bookRepository.findAll();

}

@GetMapping("/{id}")

public Book getBook(@PathVariable Long id) {

return bookRepository.findById(id).orElse(null);

}

@PutMapping("/{id}")

public Book updateBook(@PathVariable Long id, @RequestBody Book updatedBook) {

Book book = bookRepository.findById(id).orElse(null);

if (book != null) {

book.setTitle(updatedBook.getTitle());

book.setAuthor(updatedBook.getAuthor());

book.setYearPublished(updatedBook.getYearPublished());

return bookRepository.save(book);

}

return null;

}

@DeleteMapping("/{id}")

public void deleteBook(@PathVariable Long id) {

bookRepository.deleteById(id);

}

}

**Step 8: Run the Application**

**LibraryManagementApplication.java**

package com.example.LibraryManagement;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class LibraryManagementApplication {

public static void main(String[] args) {

SpringApplication.run(LibraryManagementApplication.class, args);

}

}

**Output:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer program

AI-generated content may be incorrect.**