**Digital Nurture 4.0 – Week 2**

**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.

**Objective:**

To configure a basic Spring Core application using Maven in IntelliJ IDEA for a Library Management system, which separates concerns using BookService and BookRepository components, and manages them via XML-based Dependency Injection.

**Steps:**

* Open IntelliJ IDEA → File → New → Project.
* Select **Maven** (not Spring Initializr).
* Project name: LibraryManagement.
* Set groupId: com.library, artifactId: LibraryManagement.
* Finish. IntelliJ will generate the basic Maven structure

**Add Spring Core Dependency:**

**pom.xml.**

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.30</version>

</dependency>

**Create Folder Structure:**

src/main/java, create:

* com.library.service → for BookService.java
* com.library.repository → for BookRepository.java

src/main/resources, create:

* applicationContext.xml

**Create BookRepository Class:**

package com.library.repository;

public class BookRepository {

public void printBookInfo() {

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

}

}

**Create BookService Class:**

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("BookService: Displaying book");

bookRepository.printBookInfo();

}

}

**Configure Spring Beans 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

https://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>

**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");

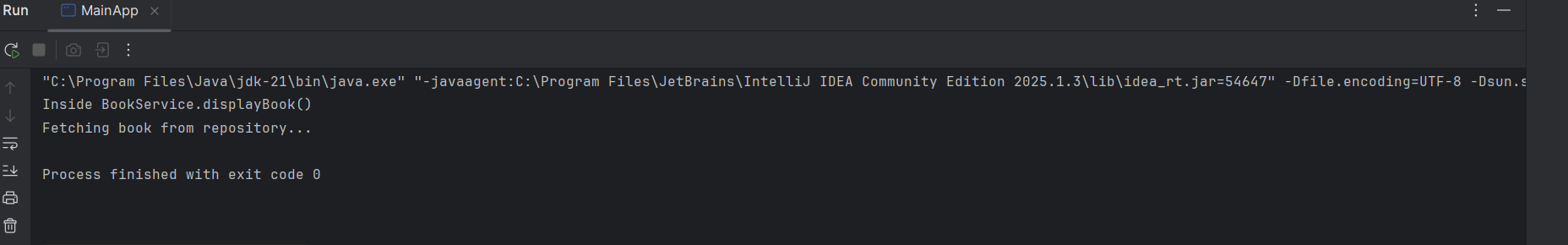
BookService bookService = context.getBean("bookService", BookService.class);

bookService.displayBook();

}

}

**Output:**

****

**Exercise 2: Implementing Dependency Injection**

**Objective:**

To implement **Dependency Injection (DI)** in a **Library Management System** using **Spring Framework**, where the BookService class depends on the BookRepository class.Spring's **Inversion of Control (IoC)** container will manage the creation and wiring of these objects through **XML-based configuration**.

**Steps:**

**Step 1:** **Create BookRepository Class:**

package com.library.repository;

public class BookRepository {

public void updateBookStatus(String bookId, String status) {

System.out.println("Book " + bookId + " status updated to: " + status);

}

}

**Step 2: Create BookService Class:**

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 issueBook(String bookId) {

System.out.println("Issuing Book: " + bookId);

bookRepository.updateBookStatus(bookId, "issued");

}

}

**Step 3: Create 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>

**Step 4: Create LibraryManagementApplication.java:**

package com.library;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

public class LibraryManagementApplication {

public static void main(String[] args) {

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

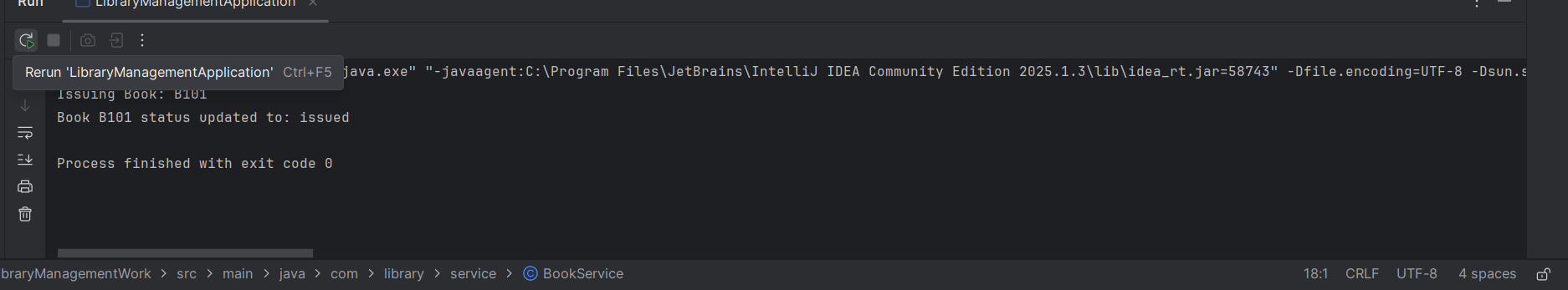
BookService bookService = context.getBean("bookService", BookService.class);

bookService.issueBook("B101");

}

}

**Output:**



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

**Objective**

To create a Maven-based Java project named LibraryManagement, add Spring framework dependencies, and configure Maven to compile with Java 1.8 using the Maven Compiler Plugin.

**Step 1: Create a New Maven Project:**

* Open IntelliJ IDEA → File → New → Project
* Choose Maven (from the left panel).
* Uncheck “Create from archetype” (keep it simple).
* Click Next.
* Enter:
  + GroupId: com.library
  + ArtifactId: LibraryManagement
  + Version: Leave default or use 1.0-SNAPSHOT
* Click Finish.

**Step 2: Add Spring Dependencies in pom.xml:**

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.30</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.30</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.30</version>

</dependency>

**Step4:** **MainApp.java:**

package com.library;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

ClassPathXmlApplicationContext context =

new ClassPathXmlApplicationContext("applicationContext.xml");

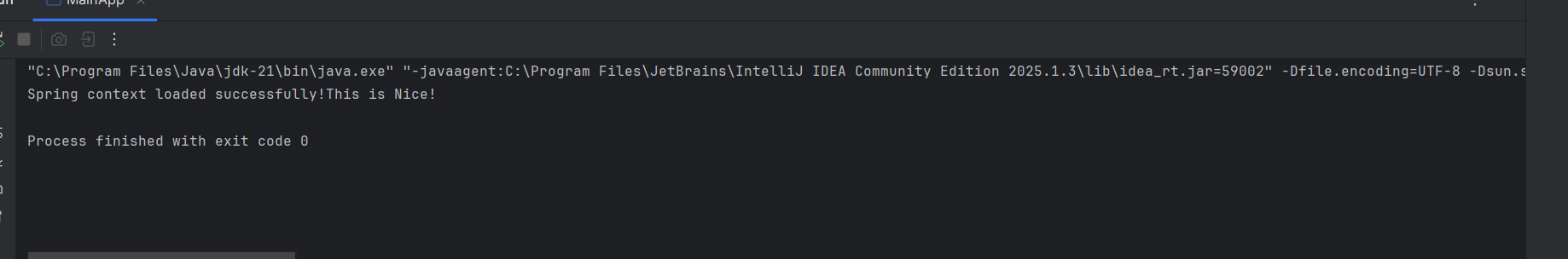
System.out.println("Spring context loaded successfully!This is Nice!");

context.close();

}

}

**Output:**

****

**Additional Hands-on**

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

**Objective:**

To configure and test the Spring IoC (Inversion of Control) Container using XML. This helps automatically manage object creation and dependency injection in your application.

**Steps:**

**Step 1: Create Spring 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

https://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>

**Step2:** **BookRepository.java**

package com.library.repository;

public class BookRepository {

public void fetchBookData() {

System.out.println(“!!!BookRepository: Fetching book data from the database...");

}}

**Step 3: 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("!!!BookService: Displaying book information...");

bookRepository.fetchBookData();

}

}

**Step 4: 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 = context.getBean("bookService", BookService.class);

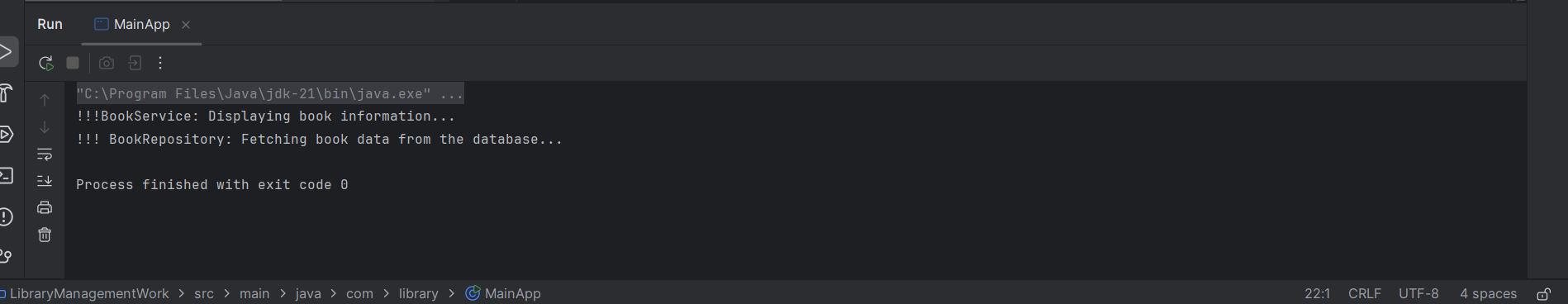
bookService.displayBook();

((ClassPathXmlApplicationContext) context).close();

}

}

**Output:**

****

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

**Objective**

To configure constructor and setter injection for Spring beans using applicationContext.xml, and test the setup through a main class.

**Steps:**

**Step1:** **BookRepository.java**

package com.library.repository;

public class BookRepository {

public void fetchBookData() {

System.out.println("!!!BookRepository: Fetching book data from the database...");

}

}

**Step2:** **BookService.java**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private String message;

private BookRepository bookRepository;

public BookService(String message) {

this.message = message;

System.out.println("!!! Constructor called: " + message);

}

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void displayBook() {

System.out.println("!!! BookService: " + message);

bookRepository.fetchBookData();

}

}

**Step3:** **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

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

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

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

<constructor-arg value="Library Service Initialized via Constructor Injection" />

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

</bean>

</beans>

**Step4:** **LibraryManagementApplication.java**

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) {

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

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

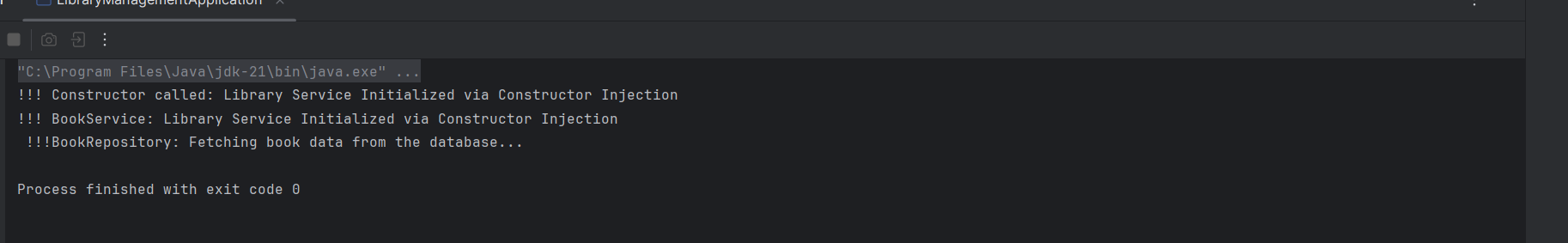
bookService.displayBook();

((ClassPathXmlApplicationContext) context).close();

}

}

**Output:**

****

**Excersice 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.

**Steps:**

**Step 1: Create Spring Boot Project**

• Go to https://start.spring.io

• Fill in the following:

o Group: com.example

o Artifact: demo

o Name: LibraryManagement

o Package Name: com.example.LibraryManagement

o Packaging: Jar

o Java Version: 21

o 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:**

