**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.

**Creation of a Maven project**

* Create a folder named LibraryManagement
* Add Spring Core Dependency in 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 https://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>5.3.33</version>

</dependency>

</dependencies>

</project>

**Configure the Application Context**

**Creation of applicationContext.xml in src/main/resources**

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

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

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

</bean>

</beans>

**Creation of classes**

**BookRepository.java:**

package com.example.demo.repository;

import org.springframework.stereotype.Repository;

@Repository

public class BookRepository {

    public void displayBookList() {

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

    }

}

**BookService.java**

package com.example.demo.service;

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

import org.springframework.stereotype.Service;

import com.example.demo.repository.BookRepository;

@Service

public class BookService {

    @Autowired

    private BookRepository bookRepository;

    public void displayBooks() {

        System.out.println("Inside BookService...");

        bookRepository.displayBookList();

    }

}

**DemoApplication.java**

package com.example.demo.service;

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

import org.springframework.stereotype.Service;

import com.example.demo.repository.BookRepository;

@Service

public class BookService {

    @Autowired

    private BookRepository bookRepository;

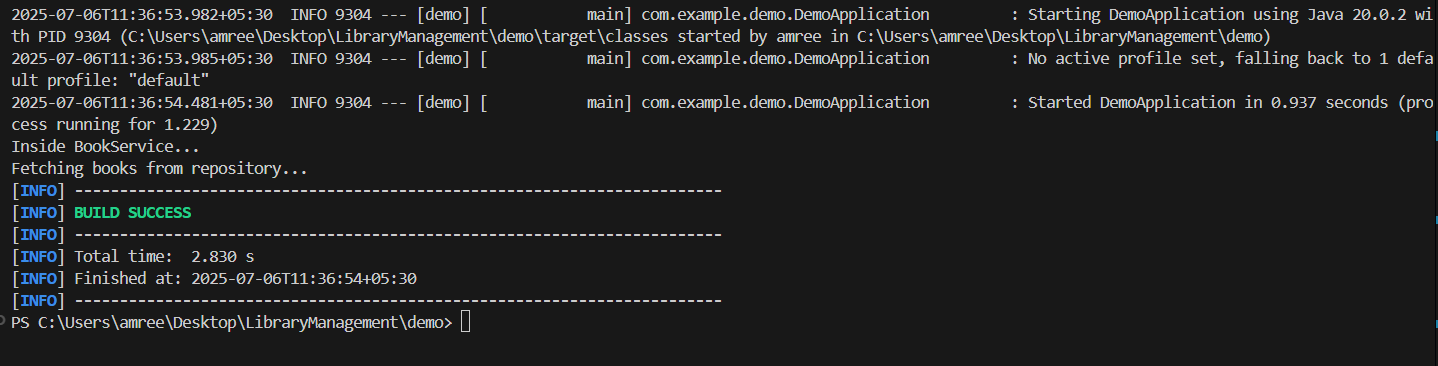
    public void displayBooks() {

        System.out.println("Inside BookService...");

        bookRepository.displayBookList();

    }

}



**Output:**

Inside BookService...

Fetching books from repository...

**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.

**Modify the XML Configuration**

Update applicationContext.xml to wire BookRepository into BookService using the <property> tag.

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

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

</bean>

**Updating the BookService class**

package com.example.demo.service;

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

import org.springframework.stereotype.Service;

import com.example.demo.repository.BookRepository;

@Service

public class BookService {

    @Autowired

    private BookRepository bookRepository;

    public void setBookRepository(BookRepository bookRepository) {

    this.bookRepository = bookRepository;

}

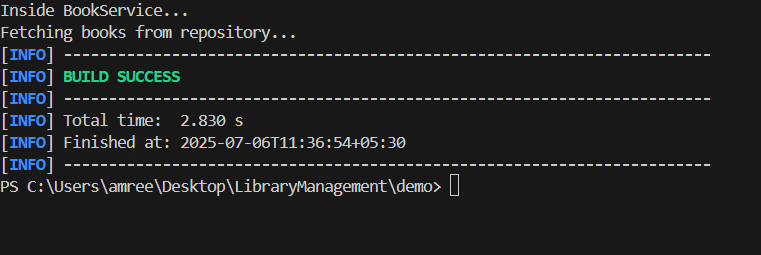
    public void displayBooks() {

        System.out.println("Inside BookService...");

        bookRepository.displayBookList();

    }

}



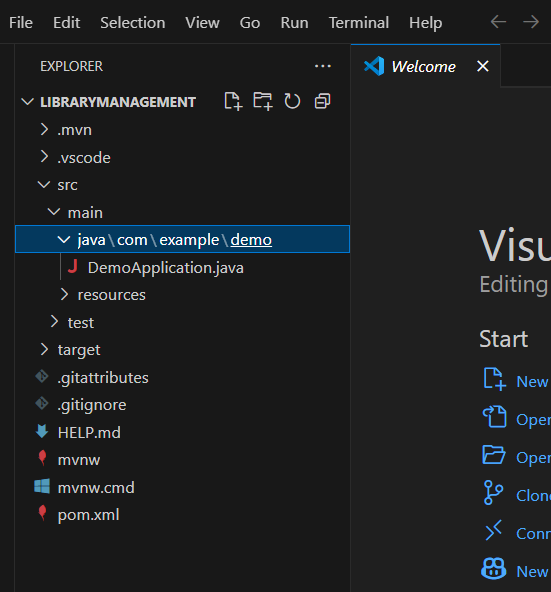
Inside BookService...

Fetching books from the repository...

**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.

**Creation of a New Maven Project named LibraryManagement.**

**Add Spring Dependencies in pom.xml**

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.33</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.33</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.33</version>

</dependency>

</dependencies>

**Configure Maven Compiler Plugin**

<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>

</plugins>

</build>

This sets up our project for Spring-based development with AOP and WebMVC support.

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

**Scenario:** The library management application requires a central configuration for beans and dependencies.

**Creation of Spring Configuration File**

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">

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

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

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

</bean>

</beans>

**Updating the BookService Class for setter method**

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

**Main method**

package com.example.demo.service;

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

import org.springframework.stereotype.Service;

import com.example.demo.repository.BookRepository;

@Service

public class BookService {

    @Autowired

    private BookRepository bookRepository;

    public void displayBooks() {

        System.out.println("Inside BookService...");

        bookRepository.displayBookList();

    }

}

**Run the application**

**Testing the configuration**

**Output:**

Inside BookService...

Fetching books from the repository...

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

**Scenario:**

The library management application requires both constructor and setter injection for better control over bean initialization.

**Configure Constructor Injection in 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">

<!-- Repository Bean -->

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

<!-- Service Bean with Constructor and Setter Injection -->

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

<constructor-arg ref="bookRepository"/>

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

</bean>

</beans>

This injects the dependency both via constructor and setter.

**Update BookService.java**

public class BookService {

private BookRepository bookRepository;

public BookService() {}

public BookService(BookRepository bookRepository) {

System.out.println("Constructor Injection Called");

this.bookRepository = bookRepository;

}

public void setBookRepository(BookRepository bookRepository) {

System.out.println("Setter Injection Called");

this.bookRepository = bookRepository;

}

public void displayBooks() {

System.out.println("Inside BookService...");

bookRepository.fetchBooks();

}

}

**MainApp.java**

package com.example.demo;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.example.demo.service.BookService;

public class MainApp {

    public static void main(String[] args) {

        try (ClassPathXmlApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml")) {

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

            bookService.displayBooks();

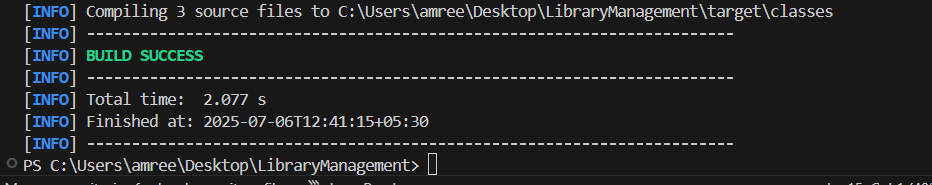
        }

    }

}

Compile:

mvn compile

Ouput after compilation:

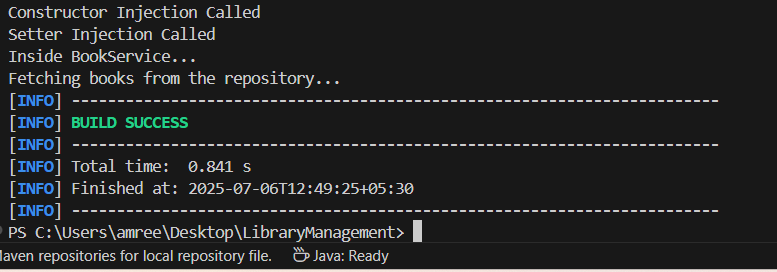
Output:

Constructor Injection called

Setter injection called

Inside BookService..

Fetching books from the repository.

**Output**

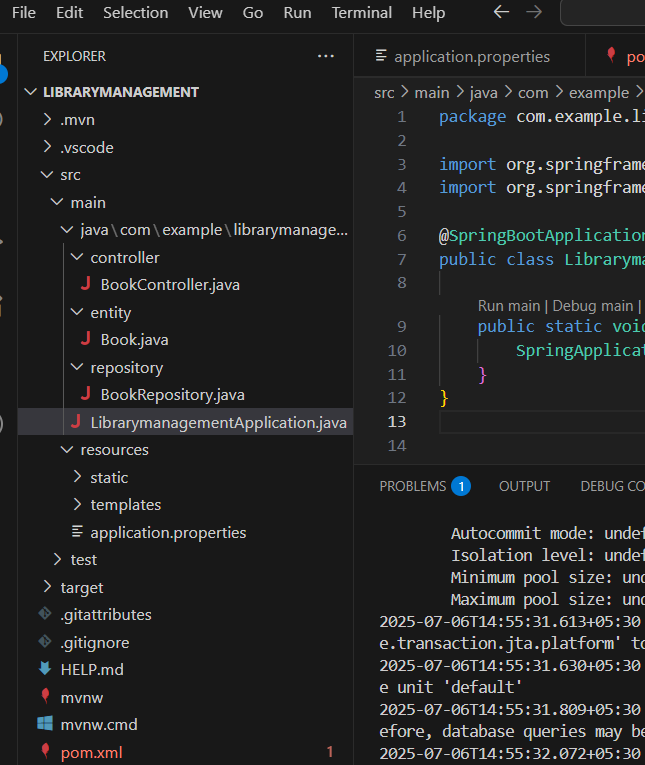
**Exercise 9: Creating a Spring Boot Application**

**Scenario:**

You need to create a Spring Boot application for the library management system to simplify configuration and deployment.

1. **Create a Spring Boot Project:**

Creation of a new Spring Boot project named LibraryManagement.



Adding dependencies of Spring web,Spring Data JPA,H2 Database

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependency>

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

</dependencies>

**Configure Application Properties**

In src/main/resources/application.properties:

spring.datasource.url=jdbc:h2:mem:testdb

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

spring.h2.console.enabled=true

**Defining Entity and Repository**

**Book.java:**

package com.library.model;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.Id;

@Entity

public class Book {

@Id

@GeneratedValue

private Long id;

private String title;

private String author;

// Getters and setters

public Long getId() { return id; }

public void setId(Long id) { this.id = id; }

public String getTitle() { return title; }

public void setTitle(String title) { this.title = title; }

public String getAuthor() { return author; }

public void setAuthor(String author) { this.author = author; }

}

**BookRepository.java**

package com.library.repository;

import com.library.model.Book;

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

public interface BookRepository extends JpaRepository<Book, Long> {

}

**Creation of a REST Controller**

package com.library.controller;

import com.library.model.Book;

import com.library.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;

@GetMapping

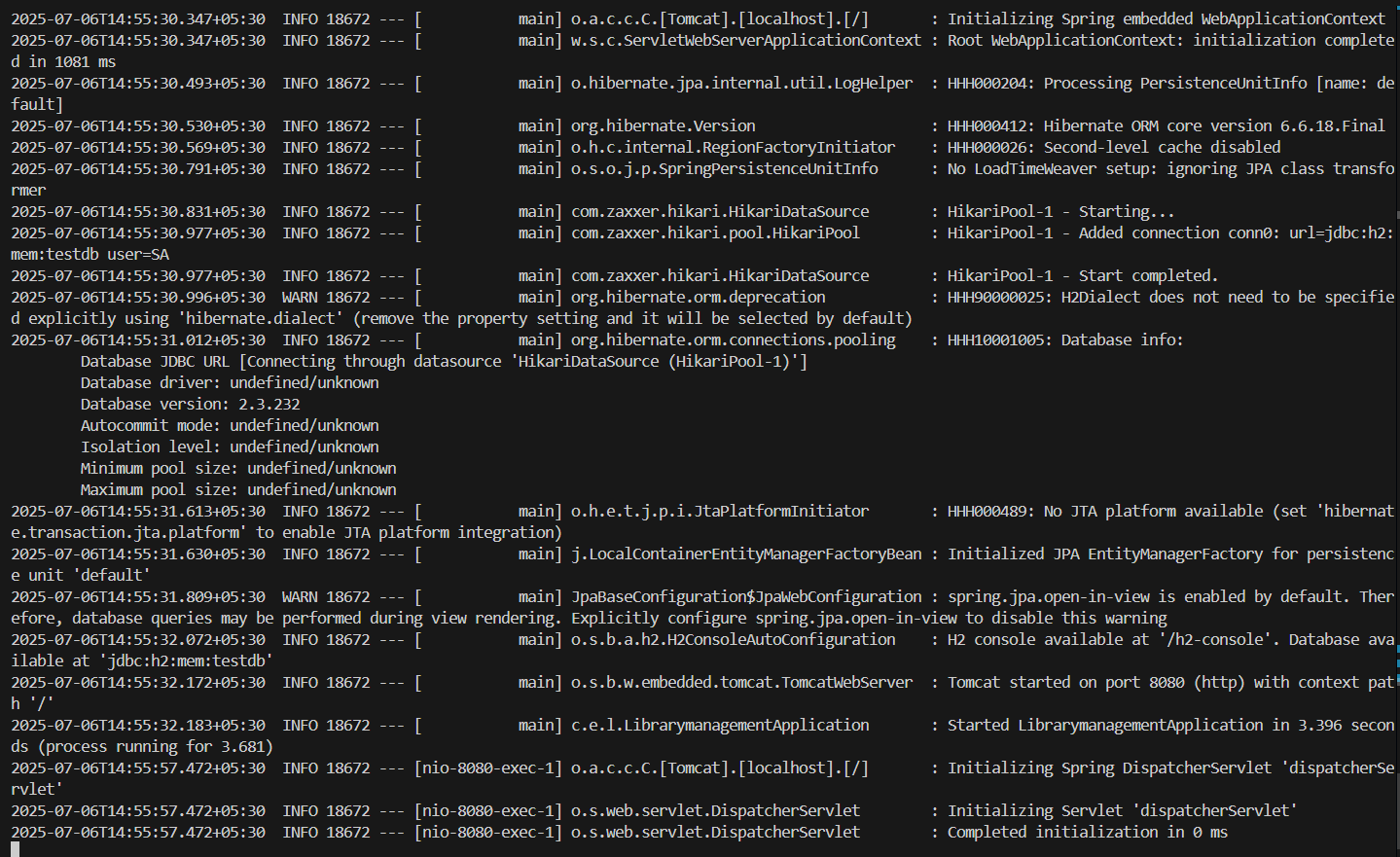
public List<Book> getAllBooks() {

return bookRepository.findAll();}

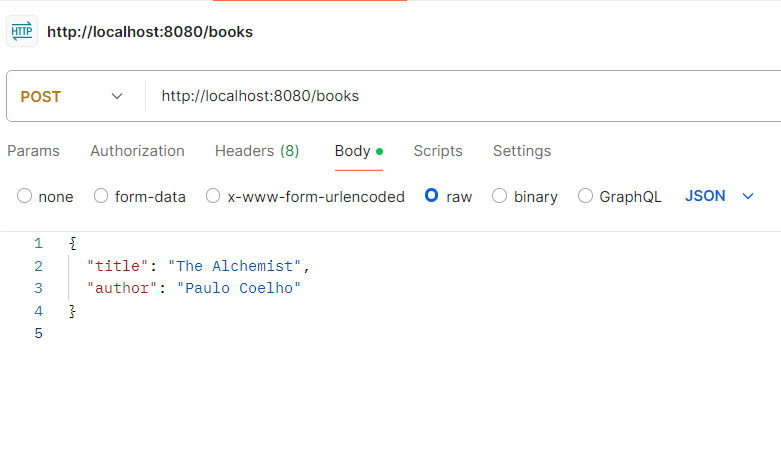
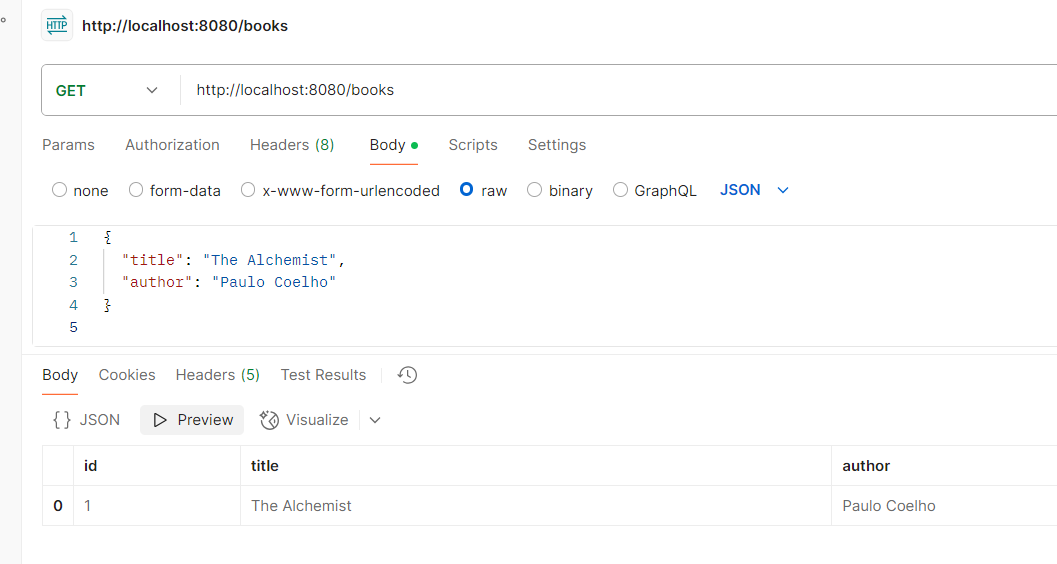
@PostMapping

public Book createBook(@RequestBody Book book) {

return bookRepository.save(book);}}

**Running the Application**:

Tomcat started on port 8080 (http) with context path '/'

**Testing in Postman**