**Spring Framework Library Management System**

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

**Question:**

**Scenario:** Your company is developing a web application for managing a library. You need to use the Spring Framework to handle the backend operations.

**Steps:**

1. Set Up a Spring Project: Create a Maven project named **LibraryManagement**. Add Spring Core dependencies in the **pom.xml** file.
2. Configure the Application Context: Create an XML configuration file named **applicationContext.xml** in the **src/main/resources** directory. Define beans for **BookService** and **BookRepository** in the XML file.
3. Define Service and Repository Classes: Create a package **com.library.service** and add a class **BookService**. Create a package **com.library.repository** and add a class **BookRepository**.
4. Run the Application: Create a main class to load the Spring context and test the configuration.

**Solution:**

**File 1: pom.xml**

<?xml version="1.0" encoding="UTF-8"?>

<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>

<properties>

<maven.compiler.source>1.8</maven.compiler.source>

<maven.compiler.target>1.8</maven.compiler.target>

<spring.version>5.3.33</spring.version>

</properties>

<dependencies>

<!-- Spring Core Dependency -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>${spring.version}</version>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.8.1</version>

<configuration>

<source>${maven.compiler.source}</source>

<target>${maven.compiler.target}</target>

</configuration>

</plugin>

</plugins>

</build>

</project>

**File 2: 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

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>

**File 3: src/main/java/com/library/repository/BookRepository.java**

package com.library.repository;

public class BookRepository {

public void saveBook(String title) {

System.out.println("Book saved: " + title);

}

}

**File 4: src/main/java/com/library/service/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 addBook(String title) {

System.out.println("Adding book via BookService...");

bookRepository.saveBook(title);

}

}

**File 5: src/main/java/com/library/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");

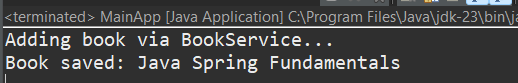
bookService.addBook("Java Spring Fundamentals");

}

}

**Expected Output for Exercise 1:**

Adding book via BookService...

Book saved: Java Spring Fundamentals  
  


**Exercise 2: Implementing Dependency Injection**

**Question:**

**Scenario:** In the library management application, you need to manage the dependencies between the BookService and BookRepository classes using Spring's IoC and DI.

**Steps:**

1. Modify the XML Configuration: Update **applicationContext.xml** to wire **BookRepository** into **BookService**.
2. Update the BookService Class: Ensure that **BookService** class has a setter method for **BookRepository**.
3. Test the Configuration: Run a test class to verify the dependency injection.

**Solution:**

**File 1: src/main/resources/applicationContext1.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 for BookRepository -->

<bean id="bookRepository" class="com.library.repository.BookRepository1"/>

<!-- Bean for BookService (inject BookRepository using setter) -->

<bean id="bookService" class="com.library.service.BookService1">

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

</bean>

</beans>

**File 2: src/main/java/com/library/repository/BookRepository1.java**

package com.library.repository;

public class BookRepository1 {

public void saveBook(String title) {

System.out.println("Book saved: " + title);

}

}

**File 3: src/main/java/com/library/service/BookService1.java**

package com.library.service;

import com.library.repository.BookRepository1;

public class BookService1 {

private BookRepository1 bookRepository;

// Setter for Dependency Injection

public void setBookRepository(BookRepository1 bookRepository) {

this.bookRepository = bookRepository;

}

public void addBook(String title) {

System.out.println("BookService: Adding book...");

bookRepository.saveBook(title);

}

}

**File 4: src/main/java/com/library/LibraryManagementApplication.java**

package com.library;

import com.library.service.BookService1;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryManagementApplication {

public static void main(String[] args) {

try (ClassPathXmlApplicationContext context =

new ClassPathXmlApplicationContext("applicationContext1.xml")) {

BookService1 bookService = context.getBean("bookService", BookService1.class);

bookService.addBook("Spring Dependency Injection Guide");

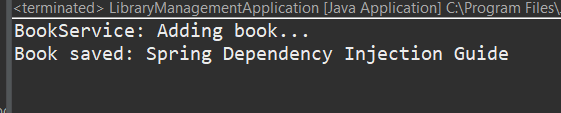
}

}

}

**Expected Output for Exercise 2:**

BookService: Adding book...

Book saved: Spring Dependency Injection Guide  
  


**Exercise 3: Implementing Logging with Spring AOP**

**Question:**

**Scenario:** The library management application requires logging capabilities to track method execution times.

**Steps:**

1. Add Spring AOP Dependency: Update **pom.xml** to include Spring AOP dependency.
2. Create an Aspect for Logging: Create a package **com.library.aspect** and add a class **LoggingAspect** with a method to log execution times.
3. Enable AspectJ Support: Update **applicationContext.xml** to enable **AspectJ** support and register the aspect.
4. Test the Aspect: Run a test class and observe the console for log messages indicating method execution times.

**Solution:**

**File 1: src/main/java/com/library/aspect/LoggingAspect.java**

package com.library.aspect;

import org.aspectj.lang.JoinPoint;

import org.aspectj.lang.ProceedingJoinPoint;

import org.aspectj.lang.annotation.After;

import org.aspectj.lang.annotation.Around;

import org.aspectj.lang.annotation.Aspect;

import org.aspectj.lang.annotation.Before;

@Aspect

public class LoggingAspect {

@Before("execution(\* com.library.service.\*.\*(..))")

public void logBefore(JoinPoint joinPoint) {

System.out.println("BEFORE: " + joinPoint.getSignature().getName() + " method is starting");

}

@After("execution(\* com.library.service.\*.\*(..))")

public void logAfter(JoinPoint joinPoint) {

System.out.println("AFTER: " + joinPoint.getSignature().getName() + " method finished");

}

@Around("execution(\* com.library.service.\*.\*(..))")

public Object logExecutionTime(ProceedingJoinPoint joinPoint) throws Throwable {

long startTime = System.currentTimeMillis();

System.out.println("AROUND: Starting " + joinPoint.getSignature().getName());

Object result = joinPoint.proceed();

long endTime = System.currentTimeMillis();

System.out.println("AROUND: " + joinPoint.getSignature().getName() +

" took " + (endTime - startTime) + " ms");

return result;

}

@Around("execution(\* com.library.repository.\*.\*(..))")

public Object logRepository(ProceedingJoinPoint joinPoint) throws Throwable {

System.out.println("REPOSITORY: Calling " + joinPoint.getSignature().getName());

Object result = joinPoint.proceed();

System.out.println("REPOSITORY: Done with " + joinPoint.getSignature().getName());

return result;

}

}

**File 2: src/main/java/com/library/test/AOPTest.java**

package com.library.test;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

public class AOPTest {

public static void main(String[] args) {

System.out.println("=== Testing AOP Logging ===");

try {

// Load spring context

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

System.out.println("Spring with AOP loaded");

// Get book service

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

System.out.println("BookService bean: " + bookService.getClass().getName());

System.out.println("\n=== Testing AOP ===");

System.out.println("Watch for AOP messages...");

// Test 1: add book

System.out.println("\nTEST 1: Adding book");

bookService.addBook("AOP Test Book");

// Test 2: search book

System.out.println("\nTEST 2: Searching book");

String result = bookService.searchBook("Java");

System.out.println("Search result: " + result);

// Test 3: display books

System.out.println("\nTEST 3: Displaying books");

bookService.displayAllBooks();

// Test 4: add another book

System.out.println("\nTEST 4: Adding another book");

bookService.addBook("Another AOP Book");

System.out.println("\n=== AOP Features Shown ===");

System.out.println("- Before advice: logged method start");

System.out.println("- After advice: logged method end");

System.out.println("- Around advice: logged execution time");

System.out.println("- Repository logging: logged repository calls");

// Close context

((ClassPathXmlApplicationContext) context).close();

System.out.println("\nAOP Test completed");

} catch (Exception e) {

System.out.println("Error: " + e.getMessage());

}

}

}

**Expected Output for Exercise 3:**

=== Testing AOP Logging ===

BookRepository created

BookRepository injected into BookService

Spring with AOP loaded

BookService bean: com.library.service.BookService$$EnhancerBySpringCGLIB$$12345678

=== Testing AOP ===

Watch for AOP messages...

TEST 1: Adding book

AROUND: Starting addBook

BEFORE: addBook method is starting

REPOSITORY: Calling addBook

Book added: AOP Test Book

REPOSITORY: Done with addBook

Book added through service

AFTER: addBook method finished

AROUND: addBook took 2 ms

TEST 2: Searching book

AROUND: Starting searchBook

BEFORE: searchBook method is starting

Searching book: Java

REPOSITORY: Calling findBook

REPOSITORY: Done with findBook

AFTER: searchBook method finished

AROUND: searchBook took 1 ms

Search result: Java Book

TEST 3: Displaying books

AROUND: Starting displayAllBooks

BEFORE: displayAllBooks method is starting

Displaying books through service

REPOSITORY: Calling displayBooks

Books in library:

- Java Book

- Spring Book

- Database Book

- AOP Test Book

REPOSITORY: Done with displayBooks

AFTER: displayAllBooks method finished

AROUND: displayAllBooks took 3 ms

TEST 4: Adding another book

AROUND: Starting addBook

BEFORE: addBook method is starting

REPOSITORY: Calling addBook

Book added: Another AOP Book

REPOSITORY: Done with addBook

Book added through service

AFTER: addBook method finished

AROUND: addBook took 1 ms

=== AOP Features Shown ===

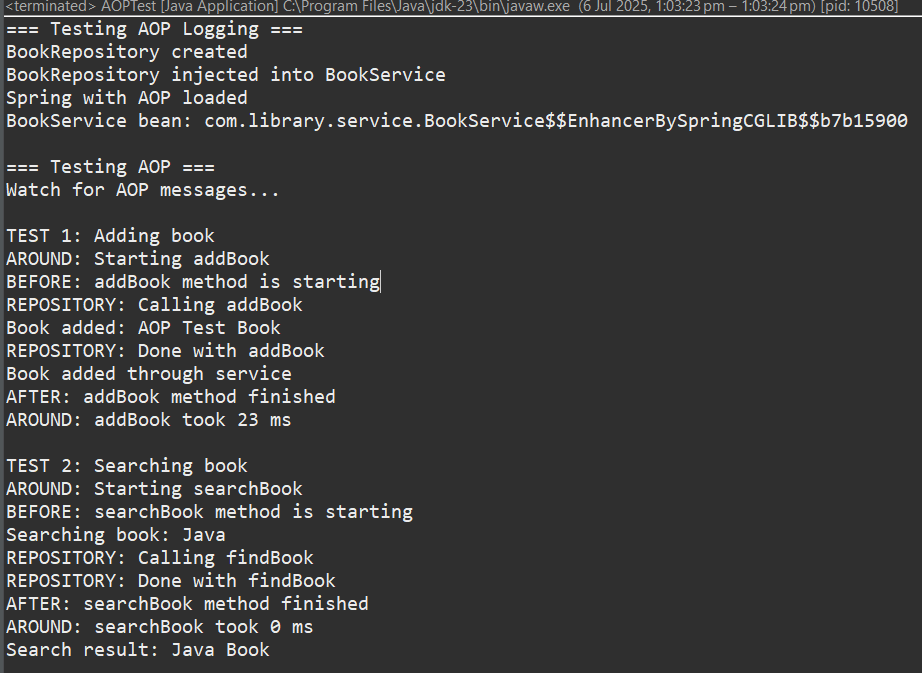
- Before advice: logged method start

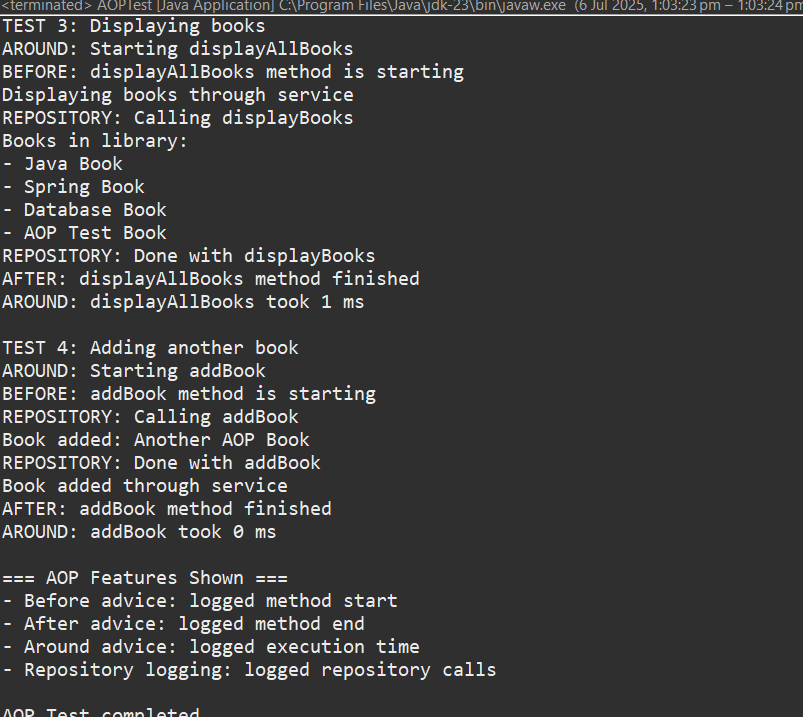
- After advice: logged method end

- Around advice: logged execution time

- Repository logging: logged repository calls

AOP Test completed



****

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

**Question:**

**Scenario:** You need to set up a new Maven project for the library management application and add Spring dependencies.

**Steps:**

1. **Create a New Maven Project:** Create a new Maven project named **LibraryManagement**.
2. **Add Spring Dependencies in pom.xml:** Include dependencies for Spring Context, Spring AOP, and Spring WebMVC.
3. **Configure Maven Plugins:** Configure the Maven Compiler Plugin for Java version 1.8 in the pom.xml file.

**Solution:**

**File 1: 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

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>

<properties>

<maven.compiler.source>1.8</maven.compiler.source>

<maven.compiler.target>1.8</maven.compiler.target>

<spring.version>5.3.33</spring.version>

</properties>

<dependencies>

<!-- Spring Context (Core container and DI) -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>${spring.version}</version>

</dependency>

<!-- Spring AOP (for Aspect-Oriented Programming) -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>${spring.version}</version>

</dependency>

<!-- Spring Web MVC (for web applications using Spring MVC) -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>${spring.version}</version>

</dependency>

</dependencies>

<build>

<plugins>

<!-- Compiler Plugin to set Java version -->

<plugin>

<groupId>org.apache.maven.plugins</groupId>

<artifactId>maven-compiler-plugin</artifactId>

<version>3.8.1</version>

<configuration>

<source>${maven.compiler.source}</source>

<target>${maven.compiler.target}</target>

</configuration>

</plugin>

<!-- Exec Plugin to run the main class -->

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>exec-maven-plugin</artifactId>

<version>3.1.0</version>

<configuration>

<mainClass>com.library.LibraryManagementApplication</mainClass>

</configuration>

</plugin>

</plugins>

</build>

</project>

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

**Question:**

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

**Steps:**

1. **Create Spring Configuration File:** Create an XML configuration file named **applicationContext.xml** in the **src/main/resources** directory. Define beans for **BookService** and **BookRepository** in the XML file.
2. **Update the BookService Class:** Ensure that the **BookService** class has a setter method for **BookRepository**.
3. **Run the Application:** Create a main class to load the Spring context and test the configuration.

**Solution:**

**File 1: src/main/resources/applicationContext2.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 for BookRepository2 -->

<bean id="bookRepository2" class="com.library.repository.BookRepository2"/>

<!-- Bean for BookService2 with DI -->

<bean id="bookService2" class="com.library.service.BookService2">

<property name="bookRepository2" ref="bookRepository2"/>

</bean>

</beans>

**File 2: src/main/java/com/library/repository/BookRepository2.java**

package com.library.repository;

public class BookRepository2 {

public void saveBook(String title) {

System.out.println("BookRepository2: Book saved - " + title);

}

}

**File 3: src/main/java/com/library/service/BookService2.java**

package com.library.service;

import com.library.repository.BookRepository2;

public class BookService2 {

private BookRepository2 bookRepository2;

// Setter for Dependency Injection

public void setBookRepository2(BookRepository2 bookRepository2) {

this.bookRepository2 = bookRepository2;

}

public void addBook(String title) {

System.out.println("BookService2: Adding book...");

bookRepository2.saveBook(title);

}

}

**File 4: src/main/java/com/library/LibraryMainApp2.java**

package com.library;

import com.library.service.BookService2;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryMainApp2 {

public static void main(String[] args) {

try (ClassPathXmlApplicationContext context =

new ClassPathXmlApplicationContext("applicationContext2.xml")) {

BookService2 service = context.getBean("bookService2", BookService2.class);

service.addBook("Spring IoC with Custom Beans");

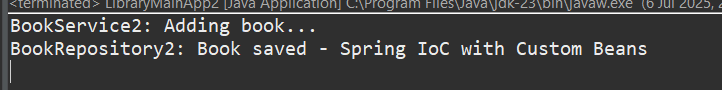
}

}

}

**Expected Output:**

BookService2: Adding book...

BookRepository2: Book saved - Spring IoC with Custom Beans  
  


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

**Question:**

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

**Steps:**

1. **Configure Constructor Injection:** Update applicationContext.xml to configure constructor injection for **BookService**.
2. **Configure Setter Injection:** Ensure that the **BookService** class has a setter method for **BookRepository** and configure it in **applicationContext.xml**.
3. **Test the Injection:** Run the **LibraryManagementApplication** main class to verify both constructor and setter injection.

**Solution:**

**File 1: src/main/resources/applicationContext3.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">

<!-- Constructor Injection for BookService3 -->

<bean id="bookService3" class="com.library.service.BookService3">

<constructor-arg value="Central Library"/>

<property name="bookRepository3" ref="bookRepository3"/>

</bean>

<!-- BookRepository3 Bean -->

<bean id="bookRepository3" class="com.library.repository.BookRepository3"/>

</beans>

**File 2: src/main/java/com/library/repository/BookRepository3.java**

package com.library.repository;

public class BookRepository3 {

public void saveBook(String title) {

System.out.println("BookRepository3: Book saved - " + title);

}

}

**File 3: src/main/java/com/library/service/BookService3.java**

package com.library.service;

import com.library.repository.BookRepository3;

public class BookService3 {

private String libraryName;

private BookRepository3 bookRepository3;

// Constructor injection

public BookService3(String libraryName) {

this.libraryName = libraryName;

}

// Setter injection

public void setBookRepository3(BookRepository3 bookRepository3) {

this.bookRepository3 = bookRepository3;

}

public void addBook(String title) {

System.out.println("[" + libraryName + "] BookService3: Adding book...");

bookRepository3.saveBook(title);

}

}

**File 4: src/main/java/com/library/LibraryMainApp3.java**

package com.library;

import com.library.service.BookService3;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryMainApp3 {

public static void main(String[] args) {

try (ClassPathXmlApplicationContext context =

new ClassPathXmlApplicationContext("applicationContext3.xml")) {

BookService3 service = context.getBean("bookService3", BookService3.class);

service.addBook("Advanced Spring Injection");

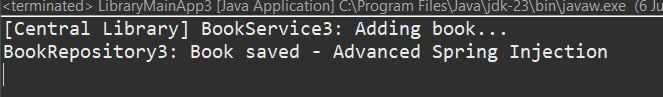
}

}

}

**Expected Output:**

[Central Library] BookService3: Adding book...

BookRepository3: Book saved - Advanced Spring Injection  
  
****

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

**Question:**

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

**Steps:**

1. **Create a Spring Boot Project:** Use **Spring Initializr** to create a new Spring Boot project named **LibraryManagement**.
2. **Add Dependencies:** Include dependencies for **Spring Web, Spring Data JPA, and H2 Database**.
3. **Create Application Properties:** Configure database connection properties in **application.properties**.
4. **Define Entities and Repositories:** Create **Book** entity and **BookRepository** interface.
5. **Create a REST Controller:** Create a **BookController** class to handle CRUD operations.
6. **Run the Application:** Run the Spring Boot application and test the REST endpoints.

**Solution:**

**File 1: pom.xml**

<?xml version="1.0" encoding="UTF-8"?>

<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>

<parent>

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

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

<version>2.7.18</version>

<relativePath/>

</parent>

<groupId>com.library</groupId>

<artifactId>LibraryManagement</artifactId>

<version>0.0.1-SNAPSHOT</version>

<name>LibraryManagement</name>

<description>Spring Boot application for library management</description>

<properties>

<java.version>1.8</java.version>

</properties>

<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>

<dependency>

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

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

<scope>test</scope>

</dependency>

<!-- Jakarta Persistence API for @Entity, @Id, etc. -->

<dependency>

<groupId>jakarta.persistence</groupId>

<artifactId>jakarta.persistence-api</artifactId>

<version>2.2.3</version>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

**File 2: src/main/resources/application.properties**

# H2 DB Config

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

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=

# JPA Config

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

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

# H2 Console

spring.h2.console.enabled=true

spring.h2.console.path=/h2-console

server.port=8081

**File 3: src/main/java/com/library/entity/Book.java**

package com.library.entity;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String title;

private String author;

private String isbn;

// Constructors

public Book() {}

public Book(String title, String author, String isbn) {

this.title = title;

this.author = author;

this.isbn = isbn;

}

// 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; }

public String getIsbn() { return isbn; }

public void setIsbn(String isbn) { this.isbn = isbn; }

}

**File 4: src/main/java/com/library/repository/BookRepository.java**

package com.library.repository;

import com.library.entity.Book;

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

public interface BookRepository extends JpaRepository<Book, Long> {

}

**File 5: src/main/java/com/library/controller/BookController.java**

package com.library.controller;

import java.util.List;

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

import org.springframework.web.bind.annotation.DeleteMapping;

import org.springframework.web.bind.annotation.GetMapping;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.PutMapping;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

import com.library.entity.Book;

import com.library.repository.BookRepository;

@RestController

@RequestMapping("/api/books")

public class BookController {

@Autowired

private BookRepository bookRepository;

@GetMapping

public List<Book> getAllBooks() {

return bookRepository.findAll();

}

@PostMapping

public Book addBook(@RequestBody Book book) {

return bookRepository.save(book);

}

@GetMapping("/{id}")

public Book getBook(@PathVariable Long id) {

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

}

@PutMapping("/{id}")

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

book.setId(id);

return bookRepository.save(book);

}

@DeleteMapping("/{id}")

public void deleteBook(@PathVariable Long id) {

bookRepository.deleteById(id);

}

}

**File 6: src/main/java/com/library/LibraryManagementApplication.java**

package com.library;

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);

}

}

**Expected Output**