**Digital Nurture 4.0 Deep Skilling - Java FSE**  
**WEEK –3 Additional Hands-on Exercises**  
**Module 5 - Spring Core and Maven**

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

**Scenario:**

The library management application requires a central configuration for beans and

dependencies.

**Solution:**

**Code:**

**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>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>LibraryManagement</name>

<url>http://www.example.com</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

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

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

</properties>

<dependencies>

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter-engine</artifactId>

<version>5.7.1</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter-params</artifactId>

<version>5.7.1</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.33</version>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

<artifactId>maven-surefire-plugin</artifactId>

<version>3.0.0-M5</version>

</plugin>

</plugins>

</build>

</project>

**BookRepository.java**

package com.library.repository;

public class BookRepository {

public void saveBook(String bookName) {

System.out.println("BookRepository: Book saved - " + bookName);

}

}

**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 bookName) {

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

bookRepository.saveBook(bookName);

}

}

**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>

**LibraryApp.java**

package com.library;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

public class LibraryApp {

public static void main(String[] args) {

ApplicationContext context = new

ClassPathXmlApplicationContext("applicationContext.xml");

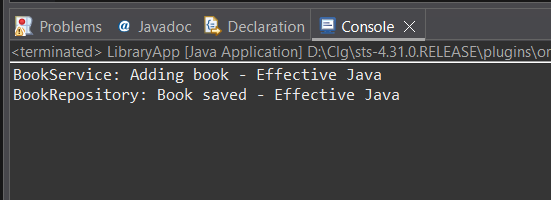
BookService service = (BookService) context.getBean("bookService");

service.addBook("Effective Java");

}

}

**Output:**



**Explanation:**

1. Create applicationContext.xml in src/main/resources.
2. Define beans for BookService and BookRepository in the XML file.
3. Add a setter method in BookService to accept BookRepository.
4. Create a main class to load the Spring context using ClassPathXmlApplicationContext.
5. Retrieve BookService bean from the context and test the configuration.

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

**Scenario:**

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

**Solution:**

**Code:**

**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="LibraryService v1.0"/>

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

</bean>

</beans>

**BookService.java**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private String serviceName; // constructor injection

private BookRepository bookRepository; // setter injection

public BookService(String serviceName) {

this.serviceName = serviceName;

System.out.println("Constructor injected: " + serviceName);

}

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void addBook(String bookName) {

System.out.println(serviceName + " - Adding Book: " + bookName);

bookRepository.saveBook(bookName);

}

}

**BookRepository.java**

package com.library.repository;

public class BookRepository {

public void saveBook(String bookName) {

System.out.println("BookRepository: Saved book - " + bookName);

}

}

**LibraryApp.java**

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryApp {

public static void main(String[] args) {

ApplicationContext context = new

ClassPathXmlApplicationContext("applicationContext.xml");

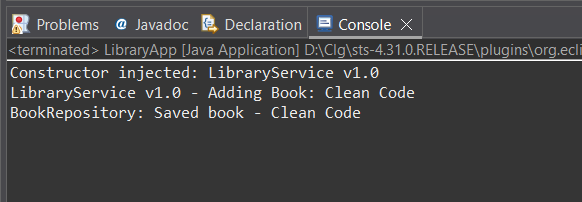
BookService bookService = (BookService) context.getBean("bookService");

bookService.addBook("Clean Code");

}

}

**Output:**



**Explanation:**

1. In applicationContext.xml, configure constructor injection for the BookService bean using the <constructor-arg> tag.
2. Ensure the BookService class has a setter method for BookRepository.
3. Configure setter injection in applicationContext.xml using the <property> tag.
4. Run the LibraryManagementApplication main class to verify that both constructor and setter injection work correctly.

**3.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.

**Solution:**

**Code:**

**applicationContext.xml**

spring.application.name=LibraryManagement2

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

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

spring.jpa.hibernate.ddl-auto=update

**Book.java**

package com.example.Library;

import jakarta.persistence.\*;

@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String title;

private String author;

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.example.Library.repository;

import com.example.Library.Book;

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

public interface BookRepository extends JpaRepository<Book, Long> {

}

**BookController.java**

package com.example.Library.controller;

import com.example.Library.Book;

import com.example.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 repository;

@GetMapping

public List<Book> getAllBooks() {

return repository.findAll();

}

@PostMapping

public Book addBook(@RequestBody Book book) {

return repository.save(book);

}

@GetMapping("/{id}")

public Book getBook(@PathVariable Long id) {

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

}

@PutMapping("/{id}")

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

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

if (book != null) {

book.setTitle(updatedBook.getTitle());

book.setAuthor(updatedBook.getAuthor());

return repository.save(book);

}

return null;

}

@DeleteMapping("/{id}")

public void deleteBook(@PathVariable Long id) {

repository.deleteById(id);

}

}

**BookController.java**

package com.example.Library;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class LibraryManagement2Application {

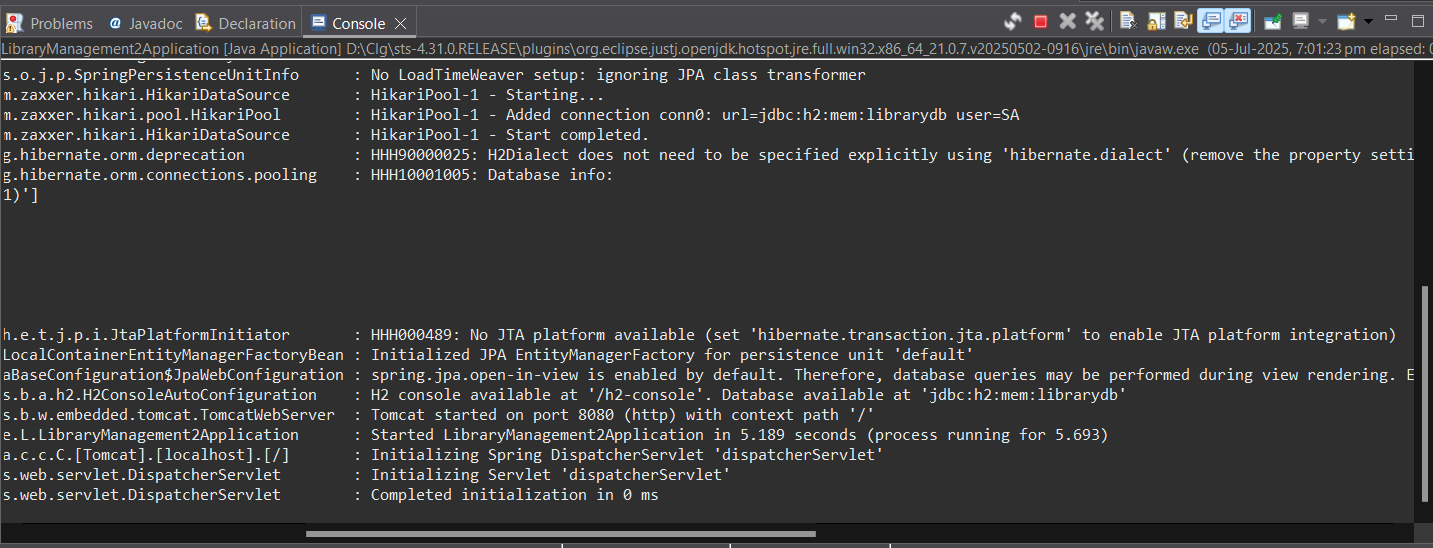
public static void main(String[] args) {

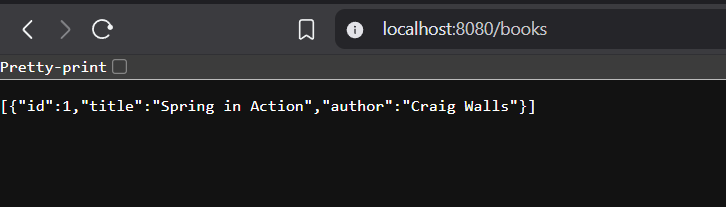
SpringApplication.run(LibraryManagement2Application.class, args);

}

}

**Output:**





**Explanation:**

1. Use Spring Initializr to create a new project named LibraryManagement.
2. Add dependencies: Spring Web, Spring Data JPA, H2 Database.
3. Configure database settings in application.properties.
4. Create the Book entity and BookRepository interface.
5. Implement BookController to expose REST endpoints for CRUD operations.
6. Run the Spring Boot application and test the API using Postman or browser.