**Spring Core\_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.

**Library Management**

**Step1 : 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>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.30</version>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

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

<version>3.1.0</version>

<configuration>

<mainClass>com.library.MainApp</mainClass>

</configuration>

</plugin>

</plugins>

</build>

</project>

**Step 2: Configure Spring XML Application Context**

<?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>

**Step 3: BookRepository.java**

package com.library.repository;

public class BookRepository {

public void saveBook() {

System.out.println("BookRepository: Book saved successfully.");

}

}

**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() {

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

bookRepository.saveBook();

}

}

**Step 4: MainApp.java (Main Class)**

package com.library;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

public class MainApp {

public static void main(String[] args) {

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

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

service.addBook();

((ClassPathXmlApplicationContext) context).close();

}

}

**OUTPUT:**

BookService: Adding a new book...

BookRepository: Book saved successfully.

**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.

**Step1: Modify the XML Configuration**

<?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 the BookService Class:**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

// Dependency to be injected

private BookRepository bookRepository;

// Setter method for DI

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void addBook() {

System.out.println("BookService: Initiating addBook()...");

bookRepository.saveBook();

}

}

**BookRepository**

package com.library.repository;

public class BookRepository {

public void saveBook() {

System.out.println("BookRepository: Book saved to the database.");

}

}

**Step 3: Test the Configuration with MainApp**

package com.library;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

public class MainApp {

public static void main(String[] args) {

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

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

service.addBook();

((ClassPathXmlApplicationContext) context).close();

}

}

**OUTPUT:**

BookService: Initiating addBook()...

BookRepository: Book saved to the database.

**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.

**Step 1: New Maven Project**

mvn archetype:generate -DgroupId=com.library -DartifactId=LibraryManagement -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

**Step 2: Add Spring Dependencies 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

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>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.30</version>

</dependency>

<!-- Spring AOP (for aspect-oriented programming) -->

<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>

</dependencies>

<build>

<plugins>

<plugin>

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

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

<version>3.11.0</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

</plugin>

<plugin>

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

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

<version>3.1.0</version>

<configuration>

<mainClass>com.library.MainApp</mainClass>

</configuration>

</plugin>

</plugins>

</build>

</project>

**Step 3: Configure Maven Plugins Output**

[INFO] Scanning for projects...

[INFO] Building LibraryManagement 1.0-SNAPSHOT

[INFO] --- maven-compiler-plugin:3.11.0:compile (default-compile) @ LibraryManagement ---

[INFO] BUILD SUCCESS

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

**Scenario:**

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

**Step 1: 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

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

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

<!-- Define the BookService bean and inject BookRepository using setter -->

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

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

</bean>

</beans>

**Step 2: BookService Class**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

// Setter method for Spring Dependency Injection

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

// Simple business logic method

public void displayBookInfo() {

System.out.println("BookService: Requesting book info...");

bookRepository.showBookDetails();

}

}

**BookRepository Class**

package com.library.repository;

public class BookRepository {

public void showBookDetails() {

System.out.println("BookRepository: Showing book details - 'Spring in Action, 5th Edition'");

}

}

**Step 3: Run the Application**

package com.library;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

public class MainApp {

public static void main(String[] args) {

// Load Spring configuration from XML

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

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

// Call method to test dependency injection

bookService.displayBookInfo();

// Close the context

((ClassPathXmlApplicationContext) context).close();

}

}

**OUTPUT:**

BookService: Requesting book info...

BookRepository: Showing book details - 'Spring in Action, 5th Edition'

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

**Scenario:**

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

**Step 1: 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

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

<!-- BookRepository -->

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

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

<!-- Constructor Injection -->

<constructor-arg value="Library Service for Advanced Readers"/>

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

</bean>

</beans>

**Step 2: Setter Injection**

**BookService.java**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private String title;

private BookRepository bookRepository;

// Constructor injection

public BookService(String title) {

this.title = title;

}

// Setter injection

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

// Method to test both injections

public void performService() {

System.out.println("BookService Title: " + title);

bookRepository.displayBook();

}

}

**BookRepository.java**

package com.library.repository;

public class BookRepository {

public void displayBook() {

System.out.println("BookRepository: Displaying 'Effective Java - 3rd Edition'");

}

}

**Step 3: MainApp.java**

package com.library;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

public class MainApp {

public static void main(String[] args) {

// Load Spring container

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

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

// Test the injection results

service.performService();

((ClassPathXmlApplicationContext) context).close();

}

}

**OUTPUT:**

BookService Title: Library Service for Advanced Readers

BookRepository: Displaying 'Effective Java - 3rd Edition’

**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.

**Step 1: Spring Boot Project**

Project Name: LibraryManagement

Group: com.library

Dependencies: Spring Web, Spring Data JPA, H2 Database

Packaging: Jar

Java Version: 11

**Step 2: Add Dependencies (in pom.xml)**

<dependencies>

<!-- 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>

<!--H2 Database -->

<dependency>

<groupId>com.h2database</groupId>

<artifactId>h2</artifactId>

<scope>runtime</scope>

</dependency>

</dependencies>

**Step 3: application.properties**

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

**Step 4: Create Book Entity & Repository (Book.java)**

package com.library.model;

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;

public Book() {}

public Book(String title, String author) {

this.title = title;

this.author = author;

}

// Getters and Setters

public Long getId() {

return 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> {

}

**Step 5: Create REST Controller (BookController.java)**

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;

import java.util.Optional;

@RestController

@RequestMapping("/api/books")

public class BookController {

@Autowired

rivate BookRepository bookRepository;

@PostMapping

public Book addBook(@RequestBody Book book) {

return bookRepository.save(book);

}

// Read all

@GetMapping

public List<Book> getAllBooks() {

return bookRepository.findAll();

}

// Read by ID

@GetMapping("/{id}")

public Optional<Book> getBook(@PathVariable Long id) {

return bookRepository.findById(id);

}

// Update

@PutMapping("/{id}")

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

return bookRepository.findById(id)

.map(book -> {

book.setTitle(updatedBook.getTitle());

book.setAuthor(updatedBook.getAuthor());

return bookRepository.save(book);

}).orElseGet(() -> {

updatedBook.setId(id);

return bookRepository.save(updatedBook);

});

}

// Delete

@DeleteMapping("/{id}")

public void deleteBook(@PathVariable Long id) {

bookRepository.deleteById(id);

}

}

**Step 6: Run the Application (**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);

}

}

**OUTPUT:**

POST

GET

{

"id": 1,

"title": "Head First Java",

"author": "Bert Bates"

}

PUT

DELETE