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.

1. **Set Up a Spring Project:**

Create a Maven project named **LibraryManagement**

Add Spring Core dependencies in the **pom.xml** file.

<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>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.29</version>

</dependency>

</dependencies>

</project>

1. **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.

<?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">

</bean>

</beans>

1. **Define Service and Repository Classes:**

Create a package **com.library.service** and add a class **BookService**.

package com.library.service;

public class BookService {

public void testService() {

System.***out***.println("BookService bean loaded successfully.");

}

}

Create a package **com.library.repository** and add a class **BookRepository**.

package com.library.repository;

public class BookRepository {

public void testRepository() {

System.***out***.println("BookRepository bean loaded successfully.");

}

}

1. **Run the Application:**

package com.library.main;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

import com.library.repository.BookRepository;

public class LibraryApp {

public static void main(String[] args) {

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

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

BookRepository bookRepository = (BookRepository) context.getBean ("bookRepository");

bookService.testService();

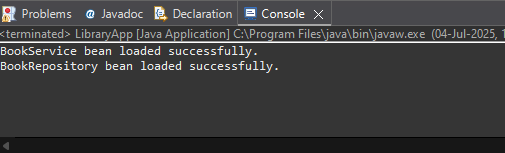
bookRepository.testRepository();

((ClassPathXmlApplicationContext) context).close();

}

}

OUTPUT:



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.

1. **Modify the XML Configuration:**

Update **applicationContext.xml** to wire **BookRepository** into **BookService**.

<?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>

1. **Update the BookService Class:**

Ensure that **BookService** class has a setter method for **BookRepository**.

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

// Setter for DI

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void addBook(String bookName) {

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

bookRepository.saveBook(bookName);

}

package com.library.repository;

public class BookRepository {

public void saveBook(String bookName) {

System.***out***.println("Book '" + bookName + "' saved to repository.");

}

}

1. **Test the Configuration:**

Run the **LibraryApp** main class to verify the dependency injection.

package com.library.main;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import org.springframework.context.ApplicationContext;

import com.library.service.BookService;

public class LibraryApp {

public static void main(String[] args) {

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

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

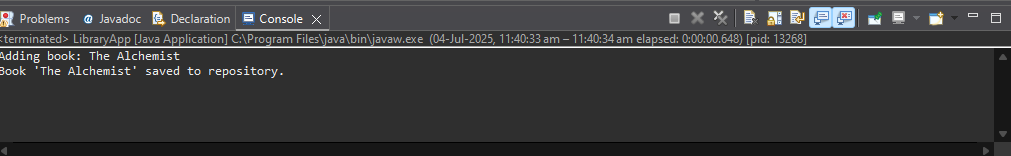
bookService.addBook("The Alchemist");

((ClassPathXmlApplicationContext) context).close();

}

}

OUTPUT:



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.

<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>

<dependencies>

<!-- Spring Context -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.29</version>

</dependency>

<!-- Spring AOP -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.29</version>

</dependency>

<!-- Spring WebMVC -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.29</version>

</dependency>

</dependencies>

<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>

</project>

Exercise 5: Configuring the Spring IoC Container

**Scenario:**

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

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.

<?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>

1. **Update the BookService Class:**

Ensure that the **BookService** class has a setter method for **BookRepository**.

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("Adding book: " + bookName);

bookRepository.saveBook(bookName);

}

}

package com.library.repository;

public class BookRepository {

public void saveBook(String bookName) {

System.***out***.println("Book '" + bookName + "' saved to repository.");

}

}

1. **Run the Application:**

Create a main class to load the Spring context and test the configuration.

package com.library.main;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import org.springframework.context.ApplicationContext;

import com.library.service.BookService;

public class LibraryApp {

public static void main(String[] args) {

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

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

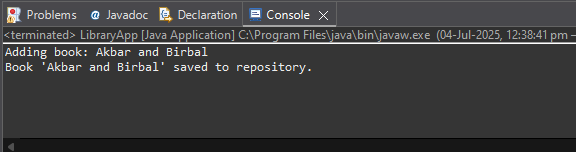
bookService.addBook("Akbar and Birbal");

((ClassPathXmlApplicationContext) context).close();

}

}

OUTPUT:



Exercise 7: Implementing Constructor and Setter Injection

**Scenario:**

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

1. **Configure Constructor Injection:**

Update applicationContext.**xml** to configure constructor injection for **BookService**.

1. **Configure Setter Injection:**

Ensure that the **BookService** class has a setter method for **BookRepository** and configure it 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">

<!-- Define BookRepository Bean -->

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

<!-- Define BookService Bean with Constructor and Setter Injection -->

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

<!-- Constructor Injection -->

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

<!-- Setter Injection -->

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

</bean>

</beans>

package com.library.repository;

public class BookRepository {

public void displayRepositoryDetails() {

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

}

}

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

private String serviceName;

// Constructor Injection

public BookService(String serviceName) {

this.serviceName = serviceName;

System.***out***.println("BookService constructor called with serviceName: " + serviceName);

}

// Setter Injection

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

System.***out***.println("BookRepository setter called");

}

public void displayServiceDetails() {

System.***out***.println("Service Name: " + serviceName);

bookRepository.displayRepositoryDetails();

}

}

1. **Test the Injection:**

Run the **LibraryApp** main class to verify both constructor and setter injection.

package com.library.main;

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");

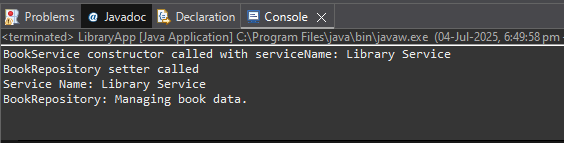
bookService.displayServiceDetails();

((ClassPathXmlApplicationContext) context).close();

}

}

OUTPUT:



Exercise 9: Creating a Spring Boot Application

Use **Spring Initializr,** to create a new Spring Boot project named LibraryManagement.

1. **Add Dependencies:**

Include dependencies for **Spring Web, Spring Data JPA, and H2 Database**.

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

</dependencies>

1. **Create Application Properties:**

Configure database connection properties in **application.properties**.

# H2 Database configuration

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

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=

# Hibernate configuration

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

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

# Enable H2 Console

spring.h2.console.enabled=true

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

1. **Define Entities and Repositories:**

Create **Book** entity and **BookRepository** interface.

package com.example.LibraryManagementBoot.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 double price;

// Constructors

public Book() {

}

public Book(Long id, String title, String author, double price) {

this.id = id;

this.title = title;

this.author = author;

this.price = price;

}

// 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 double getPrice() {

return price;

}

public void setPrice(double price) {

this.price = price;

}

}

package com.example.LibraryManagementBoot.repository;

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

import com.example.LibraryManagementBoot.entity.Book;

public interface BookRepository extends JpaRepository<Book, Long> {

}

1. **Create a REST Controller:**

Create a **BookController** class to handle CRUD operations.

package com.example.LibraryManagementBoot.controller;

import com.example.LibraryManagementBoot.entity.Book;

import com.example.LibraryManagementBoot.repository.BookRepository;

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

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

*@RestController*

*@RequestMapping*("/books")

public class BookController {

*@Autowired*

private BookRepository bookRepository;

// GET all books

*@GetMapping*

public List<Book> getAllBooks() {

return bookRepository.findAll();

}

// GET book by id

*@GetMapping*("/{id}")

public Book getBookById(*@PathVariable* Long id) {

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

}

// POST create a new book

*@PostMapping*

public Book createBook(*@RequestBody* Book book) {

return bookRepository.save(book);

}

// PUT update a book

*@PutMapping*("/books/{id}")

public ResponseEntity<Book> updateBook(*@PathVariable* Long id, *@RequestBody* Book updatedBook) {

return bookRepository.findById(id)

.map(book -> {

book.setTitle(updatedBook.getTitle());

book.setAuthor(updatedBook.getAuthor());

book.setPrice(updatedBook.getPrice());

bookRepository.save(book);

return ResponseEntity.*ok*(book);

})

.orElseGet(() -> ResponseEntity.*notFound*().build());

}

// DELETE a book

*@DeleteMapping*("/{id}")

public void deleteBook(*@PathVariable* Long id) {

bookRepository.deleteById(id);

}

}

1. **Run the Application:**

Run the Spring Boot application and test the REST endpoints.

package com.example.LibraryManagementBoot;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

*@SpringBootApplication*

public class LibraryManagementBootApplication {

public static void main(String[] args) {

SpringApplication.*run*(LibraryManagementBootApplication.class, args);

}

}

