**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.

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>org.example</groupId>

<artifactId>LibraryManagement</artifactId>

<version>1.0-SNAPSHOT</version>

<packaging>jar</packaging>

<name>LibraryManagement</name>

<properties>

<maven.compiler.source>17</maven.compiler.source>

<maven.compiler.target>17</maven.compiler.target>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>6.1.6</version>

</dependency>

</dependencies>

</project>

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>

BookRepository.java

package com.library.repository;

public class BookRepository {

public String getBookName() {

return "Spring in Action";

}

}

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 displayBooks() {

System.out.println("Book: " + bookRepository.getBookName());

}

}

App.java

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class App {

public static void main(String[] args) {

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

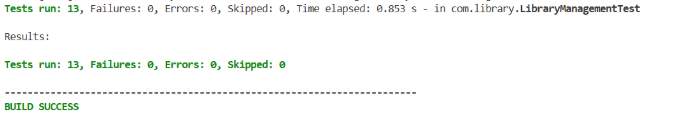
BookService bookService = (BookService) context.getBean("bookService");

bookService.displayBooks();

}

}

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.

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>

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 displayBooks() {

System.out.println("Book: " + bookRepository.getBookName());

}

}

App.java package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class App {

public static void main(String[] args) {

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

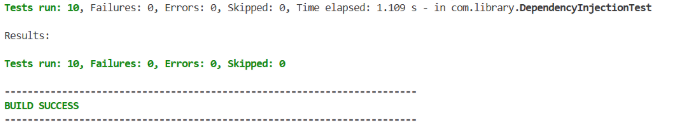
BookService bookService = (BookService) context.getBean("bookService");

bookService.displayBooks();

}

}

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.

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>6.0.12</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>6.0.12</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>6.0.12</version>

</dependency>

</dependencies>

</project>

BookRepository.java

package com.library.repository;

public class BookRepository {

public String getBookName() {

return "Spring in Action";

}

}

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 displayBooks() {

System.out.println("Book: " + bookRepository.getBookName());

}

}

LibraryManagementApplication.java

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class LibraryManagementApplication {

public static void main(String[] args) {

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

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

service.displayBooks();

}

}

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>

OUTPUT:



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

**Scenario:**

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

BookRepository.java

package com.library.repository;

public class BookRepository {

public String getBookName() {

return "Spring in Action";

}

}

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 displayBooks() {

System.out.println("Book Retrieved from Repository: \"" + bookRepository.getBookName() + "\"");

}

}

App.java (main class)

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class App {

public static void main(String[] args) {

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

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

bookService.displayBooks();

}

}

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" />

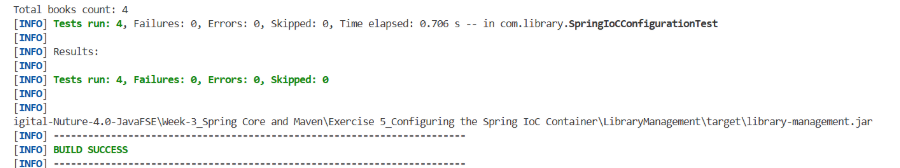
<bean id="bookService" class="com.library.service.BookService">

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

</bean>

</beans>

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.

BookRepository.java

package com.library.repository; public class BookRepository { public String getBookName() { return "Effective Java"; } }

BookService.java

package com.library.service; import com.library.repository.BookRepository; public class BookService { private BookRepository bookRepository; private String libraryName; public BookService(String libraryName) { this.libraryName = libraryName; } public void setBookRepository(BookRepository bookRepository) { this.bookRepository = bookRepository; } public void displayBooks() { System.out.println("Welcome to " + libraryName + " Library!"); System.out.println("Book available: " + bookRepository.getBookName());

} }

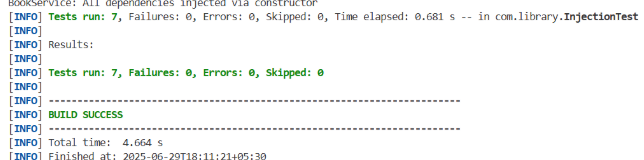
App.java

package com.library; import com.library.service.BookService; import org.springframework.context.ApplicationContext; import org.springframework.context.support.ClassPathXmlApplicationContext; public class App { public static void main(String[] args) { ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml"); BookService bookService = (BookService) context.getBean("bookService"); bookService.displayBooks(); } }

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="Central City"/> <property name="bookRepository" ref="bookRepository"/> </bean> </beans>

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. Book.java – Model

package com.libraryapp.librarybootapp.model;

import jakarta.persistence.\*;

@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;

}

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; }

}

2. BookRepository.java – JPA Repository

package com.libraryapp.librarybootapp.repository;

import com.libraryapp.librarybootapp.model.Book;

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

public interface BookRepository extends JpaRepository<Book, Long> {

}

3. BookController.java – REST API Controller

package com.libraryapp.librarybootapp.controller;

import com.libraryapp.librarybootapp.model.Book;

import com.libraryapp.librarybootapp.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 addBook(@RequestBody Book book) {

return bookRepository.save(book);

}

@GetMapping("/{id}")

public Book getBookById(@PathVariable Long id) {

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

}

@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);

}).orElse(null);

}

@DeleteMapping("/{id}")

public void deleteBook(@PathVariable Long id) {

bookRepository.deleteById(id);

}

}

4. LibraryBootAppApplication.java – Main App Class

package com.libraryapp.librarybootapp;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class LibraryBootAppApplication {

public static void main(String[] args) {

SpringApplication.run(LibraryBootAppApplication.class, args);

}

}

5. application.properties – H2 DB Config

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

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=

spring.h2.console.enabled=true

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=update

Maven Dependencies (pom.xml)

<dependencies>

<!-- Spring Boot Starter Web -->

<dependency>

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

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

</dependency>

<!-- Spring Boot Starter 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>

<!-- Jakarta Persistence API -->

<dependency>

<groupId>jakarta.persistence</groupId>

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

<version>3.1.0</version>

</dependency>

</dependencies>

**OUTPUT:**

