**Spring Core and Maven**

**Exercise 1:**

**Configuring a Basic Spring Application**

**Step 1 :Setting up a spring project**

<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>5.3.21</version>

</dependency>

</dependencies>

</project>

**Step 2 :Configure the 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="bookService" class="com.library.service.BookService"/>

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

</beans>

**Step 3 :Define Service and Repository Classes**

package com.library.service;

public class BookService {

public void displayService() {

System.out.println("BookService is working...");

}

}

package com.library.repository;

public class BookRepository {

public void displayRepository() {

System.out.println("BookRepository is working...");

}

}

**Step 4: Run the Application**

package com.library;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

import com.library.repository.BookRepository;

public class Main {

public static void main(String[] args) {

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

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

bookService.displayService();

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

bookRepository.displayRepository();

}

}

**Exercise 2:**

**Implementing Dependency Injection**

**Step 1: 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">

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

<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 {

private BookRepository bookRepository;

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void displayService() {

System.out.println("BookService is working...");

if (bookRepository != null) {

bookRepository.displayRepository();

} else {

System.out.println("BookRepository is not injected.");

}

}

}

**Step 3: Test the Configuration**

package com.library;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

public class Main {

public static void main(String[] args) {

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

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

bookService.displayService();

}

}

**Exercise 3:**

**Implementing Logging with Spring AOP**

**Step 1: Add Spring AOP Dependency**

<dependencies>

<!-- Existing dependencies -->

<!-- Spring AOP dependency -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.21</version>

</dependency>

<!-- AspectJ dependency for compiling aspects -->

<dependency>

<groupId>org.aspectj</groupId>

<artifactId>aspectjrt</artifactId>

<version>1.9.8</version>

</dependency>

</dependencies>

**Step 2: Create an Aspect for Logging**

package com.library.aspect;

import org.aspectj.lang.JoinPoint;

import org.aspectj.lang.annotation.Aspect;

import org.aspectj.lang.annotation.Before;

import org.aspectj.lang.annotation.After;

import org.springframework.stereotype.Component;

@Aspect

@Component

public class LoggingAspect {

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

public void logBefore(JoinPoint joinPoint) {

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

}

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

public void logAfter(JoinPoint joinPoint) {

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

}

}

**Step 3: Enable AspectJ Support**

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

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:aop="http://www.springframework.org/schema/aop"

xsi:schemaLocation="http://www.springframework.org/schema/beans

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

http://www.springframework.org/schema/aop

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

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

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

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

</bean>

<!-- Enable AspectJ auto proxying -->

<aop:aspectj-autoproxy/>

<!-- Register the aspect -->

<bean id="loggingAspect" class="com.library.aspect.LoggingAspect"/>

</beans>

**Step 4: Test the Aspect**

package com.library;\

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

public class Main {

public static void main(String[] args) {

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

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

bookService.displayService();

}

}

**Exercise 4:**

**Creating and Configuring a Maven Project**

<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>

<!-- Spring dependencies -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.21</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.21</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.21</version>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

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

<artifactId>maven

**Exercise 5:**

**Configuring the Spring IoC Container**

**Step 1: Create Spring Configuration 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

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

<!-- Bean definition for BookRepository -->

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

<!-- Bean definition for BookService with dependency 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 {

private BookRepository bookRepository;

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void displayService() {

System.out.println("BookService is working...");

if (bookRepository != null) {

bookRepository.displayRepository();

} else {

System.out.println("BookRepository is not injected.");

}

}

}

**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 Main {

public static void main(String[] args) {

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

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

bookService.displayService();

}

}

**Exercise 6:**

**Configuring Beans with Annotations**

**Step 1: Enable Component Scanning**

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

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:context="http://www.springframework.org/schema/context"

xsi:schemaLocation="http://www.springframework.org/schema/beans

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

http://www.springframework.org/schema/context

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

<!-- Enable component scanning for the com.library package -->

<context:component-scan base-package="com.library"/>

</beans>

**Step 2: Annotate Classes**

package com.library.service;

import com.library.repository.BookRepository;

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

import org.springframework.stereotype.Service;

@Service

public class BookService {

private final BookRepository bookRepository;

@Autowired

public BookService(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void displayService() {

System.out.println("BookService is working...");

if (bookRepository != null) {

bookRepository.displayRepository();

} else {

System.out.println("BookRepository is not injected.");

}

}

}

**Step 3: Test the Configuration**

package com.library;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

public class Main {

public static void main(String[] args) {

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

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

bookService.displayService();

}

}

**Exercise 7:**

**Implementing Constructor and Setter Injection**

**Step 1: Configure Constructor Injection**

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

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:context="http://www.springframework.org/schema/context"

xsi:schemaLocation="http://www.springframework.org/schema/beans

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

http://www.springframework.org/schema/context

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

<!-- Enable component scanning for the com.library package -->

<context:component-scan base-package="com.library"/>

<!-- Bean definition for BookRepository -->

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

<!-- Bean definition for BookService with constructor injection -->

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

<constructor-arg ref="bookRepository"/>

</bean>

</beans>

**Step 2: Configure Setter Injection**

package com.library.service;

import com.library.repository.BookRepository;

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

import org.springframework.stereotype.Service;

@Service

public class BookService {

private BookRepository bookRepository;

@Autowired

public BookService(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

@Autowired

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void displayService() {

System.out.println("BookService is working...");

if (bookRepository != null) {

bookRepository.displayRepository();

} else {

System.out.println("BookRepository is not injected.");

}

}

}

**Step 3: Test the Injection**

package com.library;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

public class Main {

public static void main(String[] args) {

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

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

bookService.displayService();

}

}

**Exercise 8:**

**Implementing Basic AOP with Spring**

**Step 1: Define an Aspect and Step 2: Create Advice Methods**

package com.library.aspect;

import org.aspectj.lang.JoinPoint;

import org.aspectj.lang.annotation.Aspect;

import org.aspectj.lang.annotation.Before;

import org.aspectj.lang.annotation.After;

import org.springframework.stereotype.Component;

@Aspect

@Component

public class LoggingAspect {

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

public void logBefore(JoinPoint joinPoint) {

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

}

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

public void logAfter(JoinPoint joinPoint) {

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

}

}

**Step 3: Configure the Aspect**

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

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xmlns:aop="http://www.springframework.org/schema/aop"

xmlns:context="http://www.springframework.org/schema/context"

xsi:schemaLocation="http://www.springframework.org/schema/beans

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

http://www.springframework.org/schema/aop

http://www.springframework.org/schema/aop/spring-aop.xsd

http://www.springframework.org/schema/context

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

<!-- Enable component scanning for the com.library package -->

<context:component-scan base-package="com.library"/>

<!-- Enable AspectJ auto proxying -->

<aop:aspectj-autoproxy/>

</beans>

**Step 4: Test the Aspect**

package com.library;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

public class Main {

public static void main(String[] args) {

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

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

bookService.displayService();

}

}

**Exercise 9:**

**Creating a Spring Boot Application**

**Step 1: Create a Spring Boot Project**

**Step 2: Add Dependencies**

<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 for in-memory database -->

<dependency>

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

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

</dependency>

</dependencies>

**Step 3: Create Application Properties**

# H2 Database Configuration

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

spring.datasource.driver-class-name=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=password

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

# Show SQL statements

spring.jpa.show-sql=true

**Step 4: Define Entities and Repositories**

package com.library.model;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

@Entity

public class Book {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String title;

private String author;

public Long getId() { return id; } //getset

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; }

}

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 a REST Controller**

package com.library.controller;

import com.library.model.Book;

import com.library.repository.BookRepository;

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

import org.springframework.http.ResponseEntity;

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

import java.util.List;

import java.util.Optional;

@RestController

@RequestMapping("/books")

public class BookController {

@Autowired

private BookRepository bookRepository;

@GetMapping

public List<Book> getAllBooks() {

return bookRepository.findAll();

}

@GetMapping("/{id}")

public ResponseEntity<Book> getBookById(@PathVariable Long id) {

Optional<Book> book = bookRepository.findById(id);

return book.map(ResponseEntity::ok).orElseGet(() -> ResponseEntity.notFound().build());

}

@PostMapping

public Book createBook(@RequestBody Book book) {

return bookRepository.save(book);

}

@PutMapping("/{id}")

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

if (!bookRepository.existsById(id)) {

return ResponseEntity.notFound().build();

}

bookDetails.setId(id);

Book updatedBook = bookRepository.save(bookDetails);

return ResponseEntity.ok(updatedBook);

}

@DeleteMapping("/{id}")

public ResponseEntity<Void> deleteBook(@PathVariable Long id) {

if (!bookRepository.existsById(id)) {

return ResponseEntity.notFound().build();

}

bookRepository.deleteById(id);

return ResponseEntity.noContent().build();

}

}

**Step 6: Run the Application**

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);

}

}

**//Command: mvn spring-boot:run**