# Exercise 1: Configuring a Basic Spring Application

## Step 1: Set Up a Spring Project

<project xmln[s="h](http://maven.apache.org/POM/4.0.0)ttp[://maven.apache.org/POM/4.0.0"](http://maven.apache.org/POM/4.0.0)

xmlns:xsi="[http://www.w3.org/2001/XMLSchema-instance"](http://www.w3.org/2001/XMLSchema-instance) xsi:schemaLocati[on="h](http://maven.apache.org/POM/4.0.0)ttp[:/](http://maven.apache.org/POM/4.0.0)/[maven.apache.org/POM/4.0.0](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 Core -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.20</version>

</dependency>

<!-- Spring Beans -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-beans</artifactId>

<version>5.3.20</version>

</dependency>

<!-- Logging -->

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.36</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-simple</artifactId>

<version>1.7.36</version>

</dependency>

</dependencies>

</project>

## Step 2: Configure the Application Context

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

<beans xmln[s="h](http://www.springframework.org/schema/beans)ttp[://www.springframework.org/schema/beans"](http://www.springframework.org/schema/beans) xmlns:xsi[="h](http://www.w3.org/2001/XMLSchema-instance)tt[p://www.w3.org/2001/XMLSchema-instance"](http://www.w3.org/2001/XMLSchema-instance)

xsi:schemaLocation=["htt](http://www.springframework.org/schema/beans)p[://www.springframework.org/schema/beans](http://www.springframework.org/schema/beans) <http://www.springframework.org/schema/beans/spring-beans.xsd>">

<!-- Bean definitions -->

<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: Define Service and Repository Classes

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 performService() {

System.out.println("Service is being performed."); bookRepository.doSomething();

}

}

## Step 4: Run the Application

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

ApplicationContext context = new

ClassPathXmlApplicationContext("applicationContext.xml");

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

}

}

## Output

**Service is being performed. Repository is doing something.**

# Exercise 2: Implementing Dependency Injection

## Step 1: Modify the XML Configuration

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

<beans xmln[s="h](http://www.springframework.org/schema/beans)ttp[://www.springframework.org/schema/beans"](http://www.springframework.org/schema/beans) xmlns:xsi[="h](http://www.w3.org/2001/XMLSchema-instance)tt[p://www.w3.org/2001/XMLSchema-instance"](http://www.w3.org/2001/XMLSchema-instance)

xsi:schemaLocation=["htt](http://www.springframework.org/schema/beans)p[://www.springframework.org/schema/beans](http://www.springframework.org/schema/beans) <http://www.springframework.org/schema/beans/spring-beans.xsd>">

<!-- Bean definitions -->

<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;

// Setter method for dependency injection

public void setBookRepository(BookRepository bookRepository) { this.bookRepository = bookRepository;

}

public void performService() {

System.out.println("Service is being performed."); bookRepository.doSomething();

}

}

## Step 3: Test the Configuration

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) { ApplicationContext context = new

ClassPathXmlApplicationContext("applicationContext.xml");

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

}

}

## Output

**Service is being performed. Repository is doing something.**

# Exercise 3: Implementing Logging with Spring AOP

## Step 1: Add Spring AOP Dependency

<dependencies>

<!-- Existing dependencies -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.20</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-beans</artifactId>

<version>5.3.20</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.36</version>

</dependency>

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-simple</artifactId>

<version>1.7.36</version>

</dependency>

<!-- Spring AOP -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.20</version>

</dependency>

<!-- AspectJ weaver -->

<dependency>

<groupId>org.aspectj</groupId>

<artifactId>aspectjweaver</artifactId>

<version>1.9.9</version>

</dependency>

</dependencies>

## Step 2: Create an Aspect for Logging

package com.library.aspect;

import org.aspectj.lang.ProceedingJoinPoint; import org.aspectj.lang.annotation.Around;

import org.aspectj.lang.annotation.Aspect; import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.stereotype.Component;

@Aspect @Component

public class LoggingAspect {

private static final Logger logger = LoggerFactory.getLogger(LoggingAspect.class);

@Around("execution(\* com.library.service.\*.\*(..))")

public Object logExecutionTime(ProceedingJoinPoint joinPoint) throws Throwable { long start = System.currentTimeMillis();

Object proceed = joinPoint.proceed();

long executionTime = System.currentTimeMillis() - start;

logger.info(joinPoint.getSignature() + " executed in " + executionTime + "ms");

return proceed;

}

}

## Step 3: Enable AspectJ Support

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

<beans xmln[s="h](http://www.springframework.org/schema/beans)ttp[://www.springframework.org/schema/beans"](http://www.springframework.org/schema/beans) xmlns:xsi[="h](http://www.w3.org/2001/XMLSchema-instance)tt[p://www.w3.org/2001/XMLSchema-instance"](http://www.w3.org/2001/XMLSchema-instance) xmlns:aop[="h](http://www.springframework.org/schema/aop)tt[p://www.springframework.org/schema/aop](http://www.springframework.org/schema/aop)"

xsi:schemaLocation=["htt](http://www.springframework.org/schema/beans)p[://www.springframework.org/schema/beans](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>">

<!-- Enable AspectJ auto-proxying -->

<aop:aspectj-autoproxy />

<!-- Bean definitions -->

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

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

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

</bean>

<!-- Register the Logging Aspect -->

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

</beans>

## Step 4: Test the Aspect

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) { ApplicationContext context = new

ClassPathXmlApplicationContext("applicationContext.xml");

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

}

}

# Exercise 4: Creating and Configuring a Maven Project

## Step 1: Create a 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 xmln[s="h](http://maven.apache.org/POM/4.0.0)ttp[://maven.apache.org/POM/4.0.0"](http://maven.apache.org/POM/4.0.0)

xmlns:xsi="[http://www.w3.org/2001/XMLSchema-instance"](http://www.w3.org/2001/XMLSchema-instance) xsi:schemaLocati[on="h](http://maven.apache.org/POM/4.0.0)ttp[:/](http://maven.apache.org/POM/4.0.0)/[maven.apache.org/POM/4.0.0](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 Context -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.20</version>

</dependency>

<!-- Spring AOP -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.20</version>

</dependency>

<!-- Spring WebMVC -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.20</version>

</dependency>

<!-- SLF4J API for Logging -->

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.36</version>

</dependency>

<!-- SLF4J Simple Implementation -->

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-simple</artifactId>

<version>1.7.36</version>

</dependency>

</dependencies>

</project>

## Step 3: Configure Maven Plugins

<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>

## ‘pom.xml’

<project xmln[s="h](http://maven.apache.org/POM/4.0.0)ttp[://maven.apache.org/POM/4.0.0"](http://maven.apache.org/POM/4.0.0)

xmlns:xsi="[http://www.w3.org/2001/XMLSchema-instance"](http://www.w3.org/2001/XMLSchema-instance) xsi:schemaLocati[on="h](http://maven.apache.org/POM/4.0.0)ttp[:/](http://maven.apache.org/POM/4.0.0)/[maven.apache.org/POM/4.0.0](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 Context -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.20</version>

</dependency>

<!-- Spring AOP -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.20</version>

</dependency>

<!-- Spring WebMVC -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.20</version>

</dependency>

<!-- SLF4J API for Logging -->

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.36</version>

</dependency>

<!-- SLF4J Simple Implementation -->

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-simple</artifactId>

<version>1.7.36</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

## Step 1: Create Spring Configuration File

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

<beans xmln[s="h](http://www.springframework.org/schema/beans)ttp[://www.springframework.org/schema/beans"](http://www.springframework.org/schema/beans) xmlns:xsi[="h](http://www.w3.org/2001/XMLSchema-instance)tt[p://www.w3.org/2001/XMLSchema-instance"](http://www.w3.org/2001/XMLSchema-instance)

xsi:schemaLocation=["htt](http://www.springframework.org/schema/beans)p[://www.springframework.org/schema/beans](http://www.springframework.org/schema/beans) <http://www.springframework.org/schema/beans/spring-beans.xsd>">

<!-- Define the BookRepository bean -->

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

<!-- Define the BookService bean and inject 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;

// Setter method for dependency injection

public void setBookRepository(BookRepository bookRepository) { this.bookRepository = bookRepository;

}

public void performService() {

System.out.println("Service is being performed."); bookRepository.doSomething();

}

}

## Step 3: Run the Application

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

// Load Spring application context from XML configuration file ApplicationContext context = new

ClassPathXmlApplicationContext("applicationContext.xml");

// Retrieve the BookService bean

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

// Call a method on the BookService bean bookService.performService();

}

}

## Project Structure

LibraryManagement/

│

├── src/

│ ├── main/

│ │ ├── java/

│ │ │ └── com/

│ │ │ └── library/

│ │ │ ├── MainApp.java

│ │ │ ├── service/

│ │ │ │ └── BookService.java

│ │ │ └── repository/

│ │ │ └── BookRepository.java

│ │ └── resources/

│ │ └── applicationContext.xml

└── pom.xml

# Exercise 6: Configuring Beans with Annotations

## Step 1: Enable Component Scanning

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

<beans xmln[s="h](http://www.springframework.org/schema/beans)ttp[://www.springframework.org/schema/beans"](http://www.springframework.org/schema/beans) xmlns:xsi[="h](http://www.w3.org/2001/XMLSchema-instance)tt[p://www.w3.org/2001/XMLSchema-instance"](http://www.w3.org/2001/XMLSchema-instance)

xsi:schemaLocation=["htt](http://www.springframework.org/schema/beans)p[://www.springframework.org/schema/beans](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 -->

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

<!-- Define additional beans here if needed -->

</beans>

## Step 2: Annotate Classes

package com.library.service;

import org.springframework.stereotype.Service; import com.library.repository.BookRepository;

@Service

public class BookService {

private BookRepository bookRepository;

// Constructor injection (recommended)

public BookService(BookRepository bookRepository) { this.bookRepository = bookRepository;

}

// Alternatively, you can use a setter for dependency injection

// public void setBookRepository(BookRepository bookRepository) {

// this.bookRepository = bookRepository;

// }

public void performService() {

System.out.println("Service is being performed."); bookRepository.doSomething();

}

}

## Step 3: Test the Configuration

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

// Load Spring application context from XML configuration file ApplicationContext context = new

ClassPathXmlApplicationContext("applicationContext.xml");

// Retrieve the BookService bean

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

// Call a method on the BookService bean bookService.performService();

}

}

## “applicationContext.xml”

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

<beans xmln[s="h](http://www.springframework.org/schema/beans)ttp[://www.springframework.org/schema/beans"](http://www.springframework.org/schema/beans) xmlns:xsi[="h](http://www.w3.org/2001/XMLSchema-instance)tt[p://www.w3.org/2001/XMLSchema-instance"](http://www.w3.org/2001/XMLSchema-instance)

xmlns:context[="h](http://www.springframework.org/schema/context)tt[p://www.springframework.org/schema/context](http://www.springframework.org/schema/context)" xsi:schemaLocation=["htt](http://www.springframework.org/schema/beans)p[://www.springframework.org/schema/beans](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 -->

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

</beans>

# Exercise 7: Implementing Constructor and Setter Injection

## Step 1: Configure Constructor Injection

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

<beans xmln[s="h](http://www.springframework.org/schema/beans)ttp[://www.springframework.org/schema/beans"](http://www.springframework.org/schema/beans) xmlns:xsi[="h](http://www.w3.org/2001/XMLSchema-instance)tt[p://www.w3.org/2001/XMLSchema-instance"](http://www.w3.org/2001/XMLSchema-instance)

xmlns:context[="h](http://www.springframework.org/schema/context)tt[p://www.springframework.org/schema/context](http://www.springframework.org/schema/context)" xsi:schemaLocation=["htt](http://www.springframework.org/schema/beans)p[://www.springframework.org/schema/beans](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 -->

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

<!-- Configure BookService with constructor injection -->

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

<constructor-arg ref="bookRepository" />

</bean>

<!-- Define the BookRepository bean -->

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

</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;

// Constructor injection @Autowired

public BookService(BookRepository bookRepository) { this.bookRepository = bookRepository;

}

// Setter injection @Autowired

public void setBookRepository(BookRepository bookRepository) { this.bookRepository = bookRepository;

}

public void performService() {

System.out.println("Service is being performed."); bookRepository.doSomething();

}

}

## Step 3: Test the Injection

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

// Load Spring application context from XML configuration file ApplicationContext context = new

ClassPathXmlApplicationContext("applicationContext.xml");

// Retrieve the BookService bean

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

// Call a method on the BookService bean bookService.performService();

}

}

## “applicationContext.xml”

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

<beans xmln[s="h](http://www.springframework.org/schema/beans)ttp[://www.springframework.org/schema/beans"](http://www.springframework.org/schema/beans) xmlns:xsi[="h](http://www.w3.org/2001/XMLSchema-instance)tt[p://www.w3.org/2001/XMLSchema-instance"](http://www.w3.org/2001/XMLSchema-instance)

xmlns:context[="h](http://www.springframework.org/schema/context)tt[p://www.springframework.org/schema/context](http://www.springframework.org/schema/context)" xsi:schemaLocation=["htt](http://www.springframework.org/schema/beans)p[://www.springframework.org/schema/beans](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 -->

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

<!-- Configure BookService with constructor injection -->

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

<constructor-arg ref="bookRepository" />

</bean>

<!-- Define the BookRepository bean -->

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

</beans>

# Exercise 8: Implementing Basic AOP with Spring

## Step 1: Define an Aspect

package com.library.aspect;

import org.aspectj.lang.JoinPoint;

import org.aspectj.lang.annotation.After;

import org.aspectj.lang.annotation.Aspect; import org.aspectj.lang.annotation.Before;

import org.springframework.stereotype.Component;

@Aspect @Component

public class LoggingAspect {

@Before("execution(\* com.library.service.BookService.performService(..))") public void logBefore(JoinPoint joinPoint) {

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

}

@After("execution(\* com.library.service.BookService.performService(..))") public void logAfter(JoinPoint joinPoint) {

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

}

}

## Step 2: Create Advice Methods

1. **Advice Methods in LoggingAspect**

The LoggingAspect class above already includes the advice methods:

* + logBefore(JoinPoint joinPoint) logs a message before the performService method executes.
  + logAfter(JoinPoint joinPoint) logs a message after the performService method executes

## Step 3: Configure the Aspect

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

<beans xmln[s="h](http://www.springframework.org/schema/beans)ttp[://www.springframework.org/schema/beans"](http://www.springframework.org/schema/beans) xmlns:xsi[="h](http://www.w3.org/2001/XMLSchema-instance)tt[p://www.w3.org/2001/XMLSchema-instance"](http://www.w3.org/2001/XMLSchema-instance)

xmlns:context[="h](http://www.springframework.org/schema/context)tt[p://www.springframework.org/schema/context](http://www.springframework.org/schema/context)" xmlns:aop[="h](http://www.springframework.org/schema/aop)tt[p://www.springframework.org/schema/aop](http://www.springframework.org/schema/aop)"

xsi:schemaLocation=["htt](http://www.springframework.org/schema/beans)p[://www.springframework.org/schema/beans](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> <http://www.springframework.org/schema/aop>

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

<!-- Enable component scanning -->

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

<!-- Enable AspectJ auto-proxying -->

<aop:aspectj-autoproxy />

<!-- Configure beans if needed -->

</beans>

## Step 4: Test the Aspect

package com.library;

import com.library.service.BookService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class MainApp {

public static void main(String[] args) {

// Load Spring application context from XML configuration file ApplicationContext context = new

ClassPathXmlApplicationContext("applicationContext.xml");

// Retrieve the BookService bean

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

// Call a method on the BookService bean bookService.performService();

}

}

## Output

**Before method: performService Service is being performed.**

**Repository is doing something. After method: performService**

# Exercise 9: Creating a Spring Boot Application

Step 1: Create a Spring Boot Project

1. Use Spring Initializr

Go to [Spring Initializr](https://start.spring.io/) and create a new Spring Boot project with the following settings:

* + Project: Maven Project
  + Language: Java
  + Spring Boot: Choose the latest stable version
  + Group: com.library
  + Artifact: LibraryManagement
  + Name: LibraryManagement
  + Description: A project for managing a library
  + Package Name: com.library
  + Packaging: Jar
  + Java: 11 or later (depending on your environment) Dependencies:
  + Spring Web
  + Spring Data JPA
  + H2 Database

## Step 2: Add Dependencies

<dependencies>

<dependency>

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

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

</dependency>

<dependency>

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

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

## Step 3: Create Application Properties

# DataSource Configuration

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

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

spring.datasource.password=password

# JPA Configuration

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

spring.jpa.hibernate.ddl-auto=update

# H2 Console (for testing)

spring.h2.console.enabled=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; private String isbn;

// 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 String getIsbn() { return isbn; }

public void setIsbn(String isbn) { this.isbn = isbn; }

}

## 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) {

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

Book existingBook = book.get();

existingBook.setTitle(bookDetails.getTitle());

existingBook.setAuthor(bookDetails.getAuthor()); existingBook.setIsbn(bookDetails.getIsbn());

return ResponseEntity.ok(bookRepository.save(existingBook));

} else {

return ResponseEntity.notFound().build();

}

}

@DeleteMapping("/{id}")

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

if (bookRepository.existsById(id)) { bookRepository.deleteById(id);

return ResponseEntity.noContent().build();

} else {

return ResponseEntity.notFound().build();

}

}

}

## Step 6: Run the Application

./mvnw spring-boot:run

## Summary

1. **Spring Boot Project Setup:**
   * Created using Spring Initializr with dependencies for Spring Web, Spring Data JPA, and H2 Database.

## Dependencies:

* + Verified in pom.xml.

## Configuration:

* + Set up database properties in application.properties.

## Entities and Repositories:

* + Created Book entity and BookRepository interface.

## REST Controller:

* + Implemented BookController for CRUD operations.

## Running and Testing:

* + Ran the application and tested REST endpoints.

This setup provides a complete Spring Boot application for managing a library, simplifying the configuration and deployment process.