**Exercise 1: Configuring a Basic Spring Application**

**✅ 1. Set Up a Spring Maven Project**

**➤ pom.xml**

Create a Maven project named LibraryManagement, and use the following minimal pom.xml:

xml

CopyEdit

<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 Core Dependency -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.33</version>

</dependency>

</dependencies>

</project>

**✅ 2. Configure the Application Context**

**➤ Directory structure:**

Create the file:  
src/main/resources/applicationContext.xml

**➤ applicationContext.xml**

xml

CopyEdit

<?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 for BookRepository -->

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

<!-- Bean for BookService with dependency injection -->

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

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

</bean>

</beans>

**✅ 3. Define Service and Repository Classes**

**➤ BookRepository.java**

Location: src/main/java/com/library/repository/BookRepository.java

java

CopyEdit

package com.library.repository;

public class BookRepository {

public String getBook() {

return "Effective Java by Joshua Bloch";

}

}

**➤ BookService.java**

Location: src/main/java/com/library/service/BookService.java

java

CopyEdit

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 displayBook() {

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

}

}

**✅ 4. Run the Application**

**➤ Main class to load Spring context**

Location: src/main/java/com/library/MainApp.java

java

CopyEdit

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.displayBook();

}

}

**Exercise 2: Implementing Dependency Injection**

**1. Modify the XML Configuration**

Ensure your applicationContext.xml file (in src/main/resources) correctly **injects BookRepository into BookService** using **setter injection**.

**🔹 applicationContext.xml**

xml

CopyEdit

<?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 for BookRepository -->

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

<!-- Bean for BookService with DI via setter -->

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

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

</bean>

</beans>

**✅ 2. Update the BookService Class**

Make sure BookService.java has a **setter method** to accept a BookRepository object.

**🔹 BookService.java**

java

CopyEdit

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 displayBook() {

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

}

}

**✅ 3. Test the Configuration**

Use the main application class to **load the Spring container** and test the dependency injection.

**🔹 MainApp.java**

java

CopyEdit

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 ApplicationContext from XML

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

// Get the BookService bean

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

// Call method to check if DI worked

bookService.displayBook();

}

}

**✅ Expected Output**

When you run MainApp.java, you should see:

vbnet

CopyEdit

Book: Effective Java by Joshua Bloch

This confirms that:

* Spring successfully created the beans
* BookRepository was injected into BookService using setter injection

**Exercise 4: Creating and Configuring a Maven Project**

**1. Create a New Maven Project**

In **Eclipse** or any IDE:

* File → New → Maven Project
* Choose maven-archetype-quickstart
* Set:
  + **GroupId**: com.library
  + **ArtifactId**: LibraryManagement
* Finish

This creates the basic Maven structure.

**✅ 2. Add Spring Dependencies to pom.xml**

Open pom.xml and add the following dependencies for:

* **Spring Context**: Core functionality and DI
* **Spring AOP**: Aspect-oriented programming
* **Spring WebMVC**: For creating web applications

xml

CopyEdit

<dependencies>

<!-- Spring Context -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.33</version>

</dependency>

<!-- Spring AOP -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.33</version>

</dependency>

<!-- Spring Web MVC -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.33</version>

</dependency>

<!-- Servlet API (for Web MVC) -->

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>4.0.1</version>

<scope>provided</scope>

</dependency>

</dependencies>

**✅ 3. Configure Maven Compiler Plugin**

Still in pom.xml, configure the **Maven Compiler Plugin** for **Java 1.8**:

xml

CopyEdit

<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>

**✅ Final pom.xml Example**

xml

CopyEdit

<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>5.3.33</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.33</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.33</version>

</dependency>

<dependency>

<groupId>javax.servlet</groupId>

<artifactId>javax.servlet-api</artifactId>

<version>4.0.1</version>

<scope>provided</scope>

</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>

**✅ After Setup**

* Right-click project → Maven → Update Project
* Build and run to ensure no errors
* You are now ready to start using **Spring MVC**, **IoC**, and **AOP in your app**

**Spring Data JPA - Quick Example**

**🔧 1. Project Generation**

* Used **Spring Initializr** with:
  + Group: com.cognizant
  + Artifact: orm-learn
  + Dependencies:
    - Spring Boot DevTools
    - Spring Data JPA
    - MySQL Driver

**🛠️ 2. MySQL Setup**

* MySQL version: **8.0**
* Created schema:

sql

CopyEdit

CREATE SCHEMA ormlearn;

* Created table and inserted records:

sql

CopyEdit

CREATE TABLE country (

code VARCHAR(2) PRIMARY KEY,

name VARCHAR(50)

);

INSERT INTO country VALUES ('IN', 'India');

INSERT INTO country VALUES ('US', 'United States of America');

**⚙️ 3. Configuration in application.properties**

properties

CopyEdit

# Logging

logging.level.org.springframework=info

logging.level.com.cognizant=debug

logging.level.org.hibernate.SQL=trace

logging.level.org.hibernate.type.descriptor.sql=trace

# Log format

logging.pattern.console=%d{dd-MM-yy} %d{HH:mm:ss.SSS} %-20.20thread %5p %-25.25logger{25} %25M %4L %m%n

# DB Config

spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver

spring.datasource.url=jdbc:mysql://localhost:3306/ormlearn

spring.datasource.username=root

spring.datasource.password=root

# Hibernate

spring.jpa.hibernate.ddl-auto=validate

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQL5Dialect

**Difference between JPA, Hibernate and Spring Data JPA**

**1. JPA (Java Persistence API)**

* **What it is**: A **specification** (set of interfaces and annotations).
* **Goal**: Standardize ORM in Java across vendors.
* **Does not** do any real work — it needs a provider like Hibernate.
* **Examples**:

java

CopyEdit

@Entity

public class User {

@Id

private Long id;

private String name;

}

* **Common interfaces**: EntityManager, Query, EntityTransaction
* **Provider examples**: Hibernate, EclipseLink, OpenJPA

**📌 2. Hibernate**

* **What it is**: A **JPA implementation** + **more** features.
* **Extends JPA**: Offers more flexibility than JPA (caching, filters, criteria API, etc.)
* **Can be used with or without JPA**.
* **Vendor-specific** features:
  + @CreationTimestamp, @UpdateTimestamp
  + Native queries
  + Hibernate Session API
* **When you use Hibernate directly**, you might write:

java

CopyEdit

Session session = sessionFactory.openSession();

Transaction tx = session.beginTransaction();

session.save(user);

tx.commit();

**📌 3. Spring Data JPA**

* **What it is**: A **Spring module** that builds on JPA and Hibernate to:
  + Reduce boilerplate
  + Simplify database access
  + Use **repository interfaces** instead of writing queries
* **Key features**:
  + Interface-based repositories (JpaRepository)
  + Auto-generated queries from method names
  + Pagination, sorting
  + Custom queries with @Query
* **Example**:

java

CopyEdit

public interface UserRepository extends JpaRepository<User, Long> {

List<User> findByName(String name);

}