**WEEK 3  
 Spring Core and Maven**

**Exercise 1  
Configure a Basic Spring Application**

**Project Structure:**

LibraryExercise1/

├── src/

│ └── main/

│ ├── java/

│ │ └── com/library/

│ │ ├── repository/BookRepository.java

│ │ ├── service/BookService.java

│ │ └── main/LibraryManagementApplication.java

│ └── resources/

│ └── applicationContext.xml

└── pom.xml

**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>LibraryExercise1</artifactId>

<version>1.0</version>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.20</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>

**LibraryManagementApplication.java**

**package** com.library.main;

**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 bookService = context.getBean("bookService", BookService.**class**);

bookService.addBook();

}

}

**BookRepository.java:**

**package** com.library.repository;

**public** **class** BookRepository {

**public** **void** save() {

System.***out***.println("Book saved to the repository.");

}

}

**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** addBook() {

bookRepository.save();

System.***out***.println("Book added through service.");

}

}

**ApplicationContext.xml:**

<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">

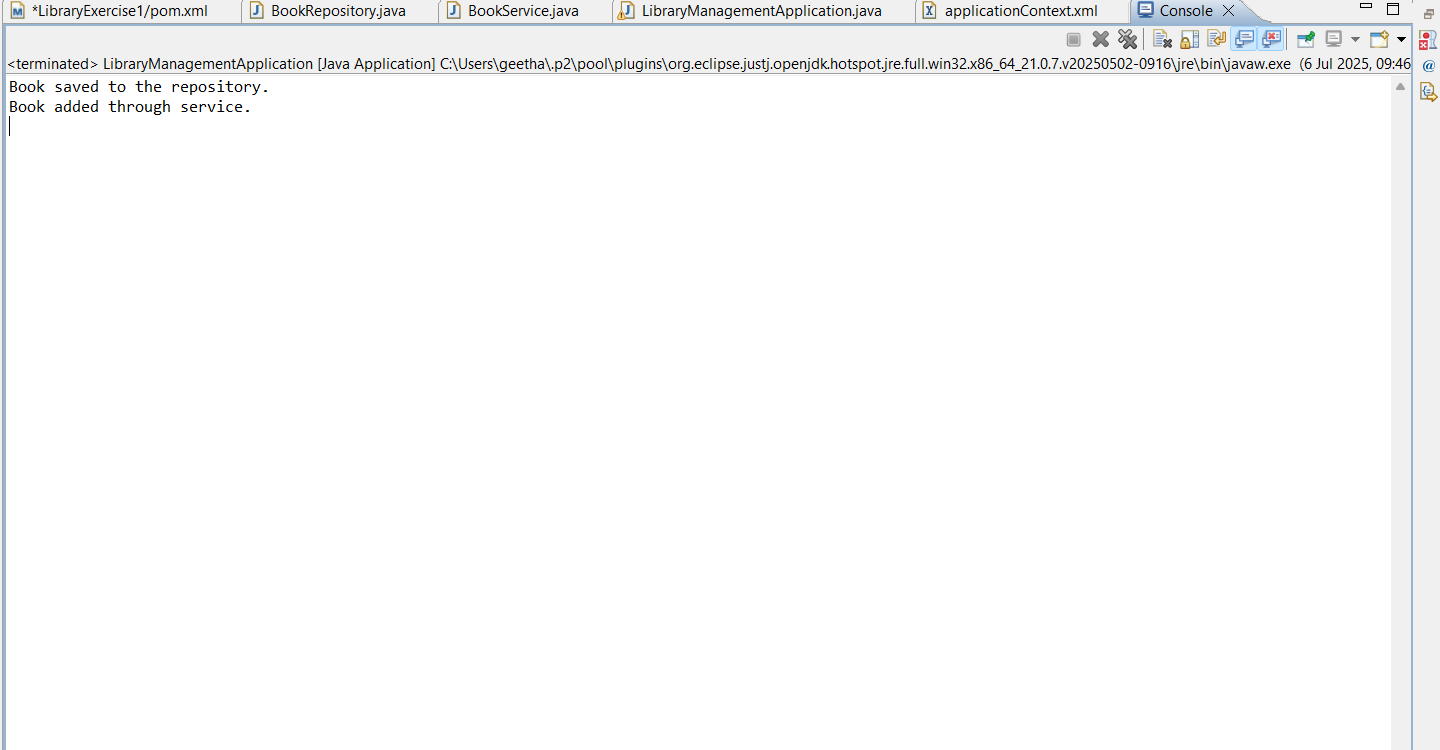
<bean id="bookRepository" class="com.library.repository.BookRepository"/>

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

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

</bean>

</beans>

**OUTPUT:**  


**Exercise 2  
Implementing Dependency injection**

I reused the project from Exercise 1 and made minor modifications to demonstrate dependency injection using Spring’s IoC container. The modified parts are listed below.

**BookService.java:**

public void performBookOperation() {

bookRepository.save();

System.out.println("Book operation performed using DI.");

}

**LibraryManagementApplication.java:**

**BookService service = context.getBean("bookService", BookService.class);**

**service.performBookOperation();**

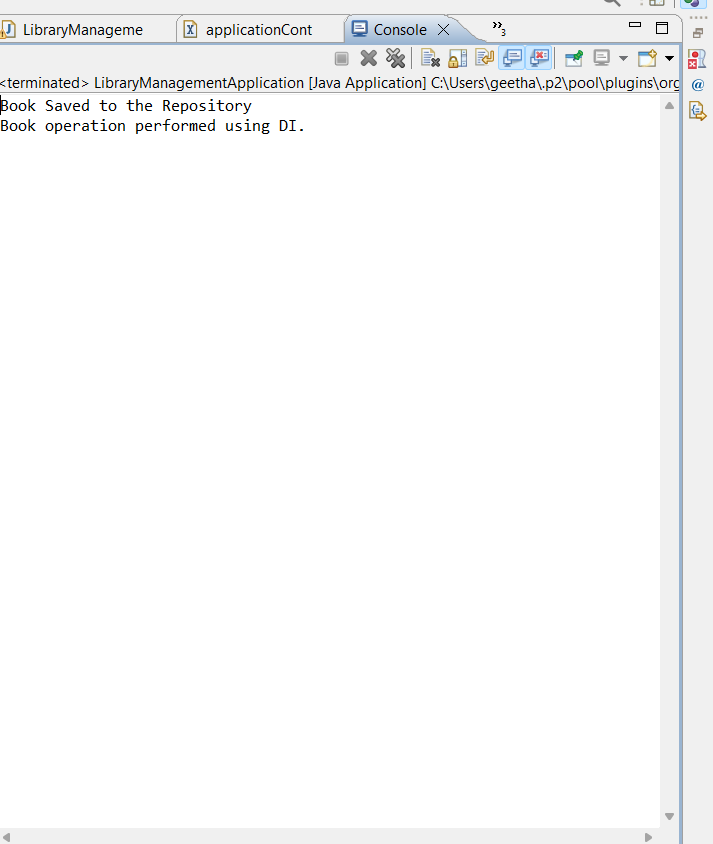
application.xml:

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

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

</bean>

**OUTPUT:**



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

***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>LibraryExercise4</artifactId>

<version>1.0</version>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.20</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.20</version>

</dependency>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.20</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>

**LibraryManagementApplication.java:**  
**package** com.library;

**public** **class** LibraryManagementApplication {

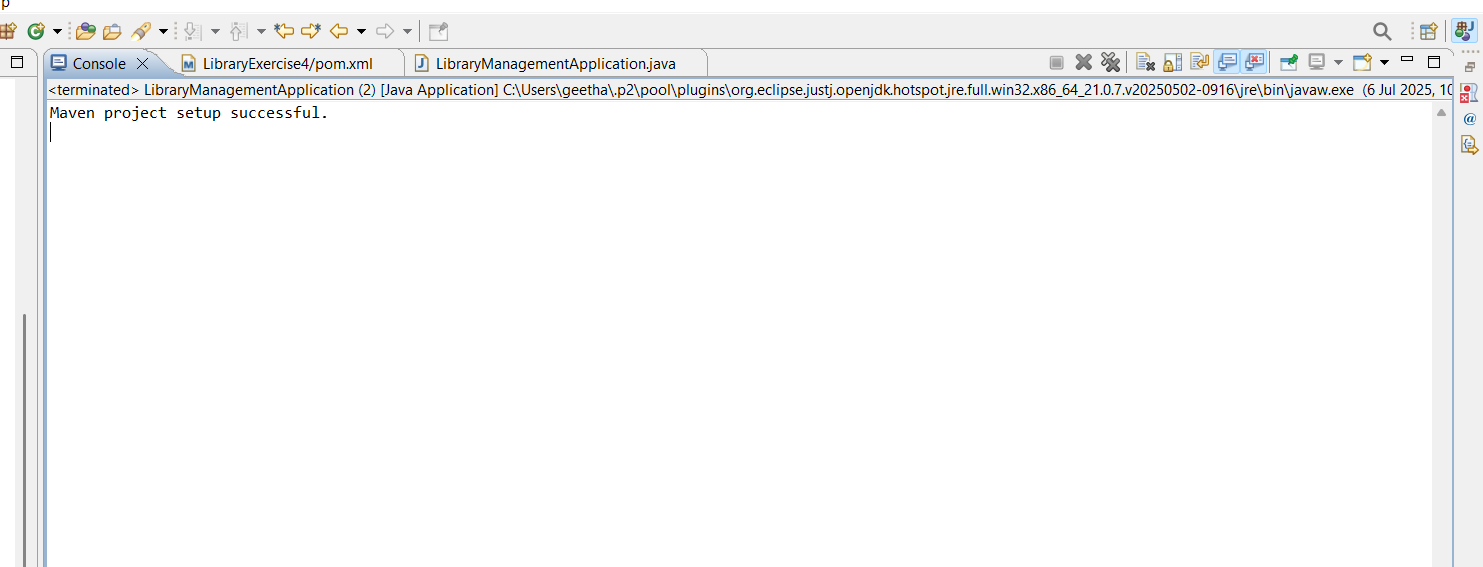
**public** **static** **void** main(String[] args) {

System.***out***.println("Maven project setup successful.");

}

}

**OUTPUT:**



**Spring Data JPA**  
**Exercise 1  
Spring Data JPA-Quick Example**  
  
**Project Structure:**  
orm-learn/

├── src/

│ ├── main/

│ │ ├── java/

│ │ │ └── com/example/

│ │ │ ├── OrmLearnApplication.java

│ │ │ ├── entity/

│ │ │ │ └── Country.java

│ │ │ └── repository/

│ │ │ └── CountryRepository.java

│ └── resources/

│ ├── application.properties

│ └── data.sql

└── pom.xml  
  
**Pom.XML:**

<dependencies>

<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-web</artifactId>

</dependency>

</dependencies>  
  
**application.properties:**  
spring.datasource.url=jdbc:h2:mem:testdb

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

spring.datasource.username=sa

spring.datasource.password=

spring.jpa.hibernate.ddl-auto=update

spring.jpa.show-sql=true

spring.h2.console.enabled=true  
  
**country.java:**

package com.example.entity;

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

import jakarta.persistence.Table;

@Entity

@Table(name = "country")

public class Country {

@Id

private String code;

private String name;

// Constructors, Getters, Setters, toString()

}  
**CountryRepository.java:**  
package com.example.repository;

import com.example.entity.Country;

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

import java.util.List;

public interface CountryRepository extends JpaRepository<Country, String> {

List<Country> findByNameContaining(String keyword);

List<Country> findByNameStartingWith(String prefix);

List<Country> findByNameContainingOrderByNameAsc(String keyword);

}  
  
**OrmLearnApplication.java:**

@SpringBootApplication

public class OrmLearnApplication implements CommandLineRunner {

@Autowired

private CountryRepository countryRepository;

public static void main(String[] args) {

SpringApplication.run(OrmLearnApplication.class, args);

}

@Override

public void run(String... args) throws Exception {

System.out.println("✅ Countries containing 'ou':");

countryRepository.findByNameContaining("ou").forEach(System.out::println);

System.out.println("\n✅ Countries starting with 'L':");

countryRepository.findByNameStartingWith("L").forEach(System.out::println);

System.out.println("\n✅ Sorted countries with 'ou':");

countryRepository.findByNameContainingOrderByNameAsc("ou").forEach(System.out::println);

}

}  
**data.sql:**

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

INSERT INTO country (code, name) VALUES ('ZA', 'South Africa');

INSERT INTO country (code, name) VALUES ('LU', 'Luxembourg');

**Exercise 2**

**Difference Between JPA , Hibernate and Spring Data JPA**  
Java Persistence API(JPA):

| **Feature** | **Description** |
| --- | --- |
| Type | **Specification (Interface)** |
| Purpose | Defines standard APIs for object-relational mapping (ORM) in Java |
| Maintained by | Oracle |
| What It Does | Allows developers to manage relational data using Java objects |
| Key Point | It’s **not an implementation**, just a set of rules (like EntityManager, @Entity) |
| Example | JPA says "you must have an entity class," but doesn’t say **how** it should be saved — that’s up to the implementation (like Hibernate) |

Hibernate:

| **Feature** | **Description** |
| --- | --- |
| Type | **Implementation of JPA** |
| Purpose | Provides actual code to persist Java objects to the database |
| Maintained by | Red Hat |
| What It Does | Implements JPA interfaces and adds additional features (like caching, lazy loading) |
| Key Point | You can use Hibernate **with or without** JPA |
| Example | When you write entityManager.persist(), Hibernate runs the actual SQL INSERT behind the scenes |

Spring Data JPA:

| **Feature** | **Description** |
| --- | --- |
| Type | **Framework on top of JPA** (Spring module) |
| Purpose | Simplifies JPA/Hibernate by generating most of the DAO/Repository code |
| Maintained by | Spring Team |
| What It Does | Automatically implements JPA repositories (like findByNameContaining()) without writing SQL or HQL |
| Key Point | Focuses on reducing boilerplate and improves developer productivity |
| Example | With Spring Data JPA, you don’t write SELECT \* FROM country WHERE name LIKE '%ou%'; you just write findByNameContaining("ou") in an interface |

Comparison:

| **Feature** | **JPA** | **Hibernate** | **Spring Data JPA** |
| --- | --- | --- | --- |
| Type | Specification | Implementation | Framework (Spring module) |
| Maintainer | Oracle | Red Hat | Spring Team |
| Purpose | Define ORM standards | Implement ORM | Simplify JPA usage |
| Requires SQL? | No | Some HQL | No (auto-generated by method names) |
| Boilerplate Code | Yes | Yes | Very minimal |
| Common Use In Spring | With Hibernate | As default JPA provider | Most preferred choice |