**WEEK - 3**

**Spring Core and Maven**

**EXERCISE 1:** Configuring a Basic Spring Application.

**STEP 1:** Set Up a Spring Project.

**1.1. Create Maven Project**

You can use an IDE like IntelliJ IDEA or Eclipse, or create manually:

LibraryManagement/

├── pom.xml

└── src/

├── main/

│ ├── java/

│ │ └── com/

│ │ └── library/

│ │ ├── service/

│ │ │ └── BookService.java

│ │ ├── repository/

│ │ │ └── BookRepository.java

│ │ └── MainApp.java

│ └── resources/

│ └── applicationContext.xml

**1.2. pom.xml with Spring Core dependencies:**

<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>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.32</version>

</dependency>

</dependencies>

</project>

**STEP 2:** Configure the Application Context

src/main/resources/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

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 -3:** Define Service and Repository Classes.

**com.library.repository.BookRepository.java.**

package com.library.repository;

public class BookRepository {

public void saveBook(String title) {

System.out.println("Book '" + title + "' saved to the repository.");

}

}

**com.library.service.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(String title) {

System.out.println("Adding book: " + title);

bookRepository.saveBook(title);

}

}

**STEP – 4:** Run the Application

**com.library.MainApp.java**

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");

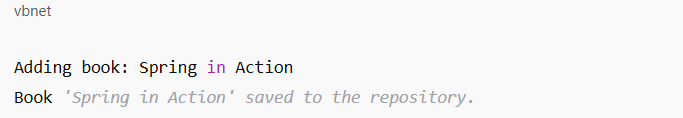
BookService bookService = (BookService) context.getBean("bookService");

bookService.addBook("Spring in Action");

}

}

**OUTPUT:**



**EXERCISE 2:** Implementing Dependency Injection.

**STEP 1:** Modify the XML Configuration.

**src/main/resources/applicationContext.xml**

This file will wire BookRepository into BookService using **setter-based 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"

xsi:schemaLocation="

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

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

<!-- Repository Bean -->

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

<!-- Service Bean with Dependency Injection -->

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

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

</bean>

</beans>

**STEP 2:** Update the BookService Class.

Ensure that BookService has a **setter method** for BookRepository.

**com.library.service.BookService.java**

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 addBook(String title) {

System.out.println("Adding book: " + title);

bookRepository.saveBook(title);

}

}

**STEP 3:** Test the Configuration.

**com.library.LibraryManagementApplication.java**

This class loads the Spring context and uses the injected BookService.

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");

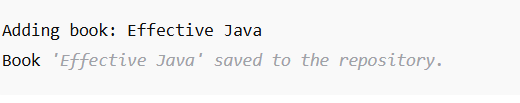
BookService bookService = (BookService) context.getBean("bookService");

bookService.addBook("Effective Java");

}

}

**OUTPUT:**



**EXERCISE 4:** Creating and Configuring a Maven Project.

**STEP 1:** Create a New Maven Project.

You can create the Maven project using:

* **IDE (like IntelliJ IDEA or Eclipse)**: File → New → Maven Project
* Or manually create this folder structure:

LibraryManagement/

├── pom.xml

└── src/

├── main/

│ ├── java/

│ └── resources/

└── test/

└── java/

**STEP -2:** Add Spring Dependencies in pom.xml.

Here’s a complete pom.xml with dependencies for:

* Spring **Context**
* Spring **AOP**
* Spring **WebMVC**

<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>

<packaging>jar</packaging>

<properties>

<maven.compiler.source>1.8</maven.compiler.source>

<maven.compiler.target>1.8</maven.compiler.target>

</properties>

<dependencies>

<!-- Spring Core + Context -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.32</version>

</dependency>

<!-- Spring AOP -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-aop</artifactId>

<version>5.3.32</version>

</dependency>

<!-- Spring Web MVC -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.32</version>

</dependency>

<!-- (Optional) Servlet API for WebMVC to compile -->

<dependency>

<groupId>javax.servlet</groupId>

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

<version>4.0.1</version>

<scope>provided</scope>

</dependency>

</dependencies>

<build>

<plugins>

<!-- Maven Compiler Plugin -->

<plugin>

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

<artifactId>maven-compiler-plugin</artifactId>

<version>3.10.1</version>

<configuration>

<source>1.8</source>

<target>1.8</target>

</configuration>

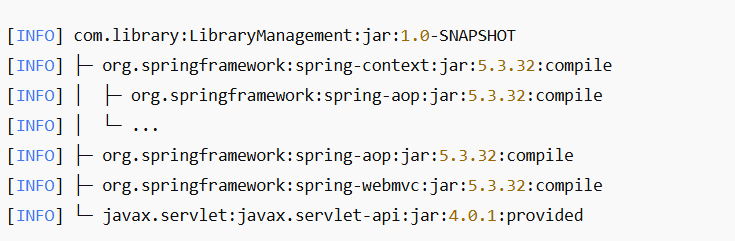
</plugin>

</plugins>

</build>

</project>

**OUTPUT:**

****

**Spring Data JPA with Spring Boot, Hibernate**

**1. Add Dependencies (pom.xml)**

For Maven:

<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>

</dependencies>

**2. Configure application.properties**

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

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=

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

spring.h2.console.enabled=true

**3. Create an Entity.**

import jakarta.persistence.\*;

@Entity

public class User {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

private String email;

// Getters and setters

}

**4. Create Repository Interface.**

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

public interface UserRepository extends JpaRepository<User, Long> {

// Custom query example

User findByEmail(String email);

}

**5. Use Repository in a Service or Controller.**

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

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

import java.util.List;

@RestController

@RequestMapping("/users")

public class UserController {

@Autowired

private UserRepository userRepository;

@PostMapping

public User addUser(@RequestBody User user) {

return userRepository.save(user);

}

@GetMapping

public List<User> getAllUsers() {

return userRepository.findAll();

}

@GetMapping("/{email}")

public User getUserByEmail(@PathVariable String email) {

return userRepository.findByEmail(email);

}

}

**6. Run the Application**

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class MyApp {

public static void main(String[] args) {

SpringApplication.run(MyApp.class, args);

}

}

**OUTPUT:**

* POST /users → Create user
* GET /users → List all users
* GET /users/{email} → Find user by email

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

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

* What it is: A standard interface for object-relational mapping (ORM) in Java.
* Who provides it: Part of the Jakarta EE (formerly Java EE) specification.
* What it does: Defines how Java objects are mapped to relational database tables, and how CRUD operations are done.
* Contains: Interfaces like EntityManager, Query, and annotations like @Entity, @Id, etc.
* Doesn't provide: Any implementation — it’s just a specification.

**2. Hibernate — *JPA Implementation (and more)***

* **What it is:** A **popular implementation** of the JPA specification.
* **Who provides it:** Red Hat.
* **What it does:**
  + Implements all JPA interfaces.
  + Adds **extra features** beyond JPA, such as caching, better performance tuning, native SQL support, and custom ID generation.
* **Used as:** A JPA provider — e.g., hibernate-entitymanager.

**Hibernate is a tool that actually performs the work defined by JPA.**

**3. Spring Data JPA — *Abstraction & Convenience Layer***

* What it is: A Spring project that builds on top of JPA and simplifies DAO (Data Access Object) implementation.
* What it does:
  + Automatically implements repository interfaces (e.g., CrudRepository, JpaRepository).
  + Reduces boilerplate code — you just define interfaces and use method naming conventions.
  + Provides query methods, pagination, sorting, and integration with Spring.
* Works with: Any JPA provider, but typically uses Hibernate by default.

Spring Data JPA makes working with JPA (and Hibernate) easier and more productive.