**WEEK - 3**

**Spring Core and Maven**

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

**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>LibraryManagement</artifactId>

<version>1.0-SNAPSHOT</version>

<dependencies>

<!-- Spring Core -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.33</version>

</dependency>

</dependencies>

</project>

**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>

**BookRepository.java**

public class BookRepository {

public void saveBook(String bookName) {

System.out.println("Book saved: " + bookName);

}

}

**BookService.java**

public class BookService {

private BookRepository bookRepository;

// Setter for Spring injection

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void addBook(String name) {

System.out.println("Adding book...");

bookRepository.saveBook(name);

}

}

**AppMain.java**

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class AppMain {

public static void main(String[] args) {

ApplicationContext context =

new ClassPathXmlApplicationContext("applicationContext.xml");

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

bookService.addBook("Spring in Action");

}

}

**Exercise 2: Implementing Dependency Injection**

**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>LibraryManagement</artifactId>

<version>1.0-SNAPSHOT</version>

<dependencies>

<!-- Spring Core -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.33</version>

</dependency>

</dependencies>

</project>

**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>

**BookRepository.java**

public class BookRepository {

public void saveBook(String bookName) {

System.out.println("Book saved: " + bookName);

}

}

**BookService.java**

public class BookService {

private BookRepository bookRepository;

// Setter for Spring injection

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void addBook(String name) {

System.out.println("Adding book...");

bookRepository.saveBook(name);

}

}

**AppMain.java**

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

public class AppMain {

public static void main(String[] args) {

ApplicationContext context =

new ClassPathXmlApplicationContext("applicationContext.xml");

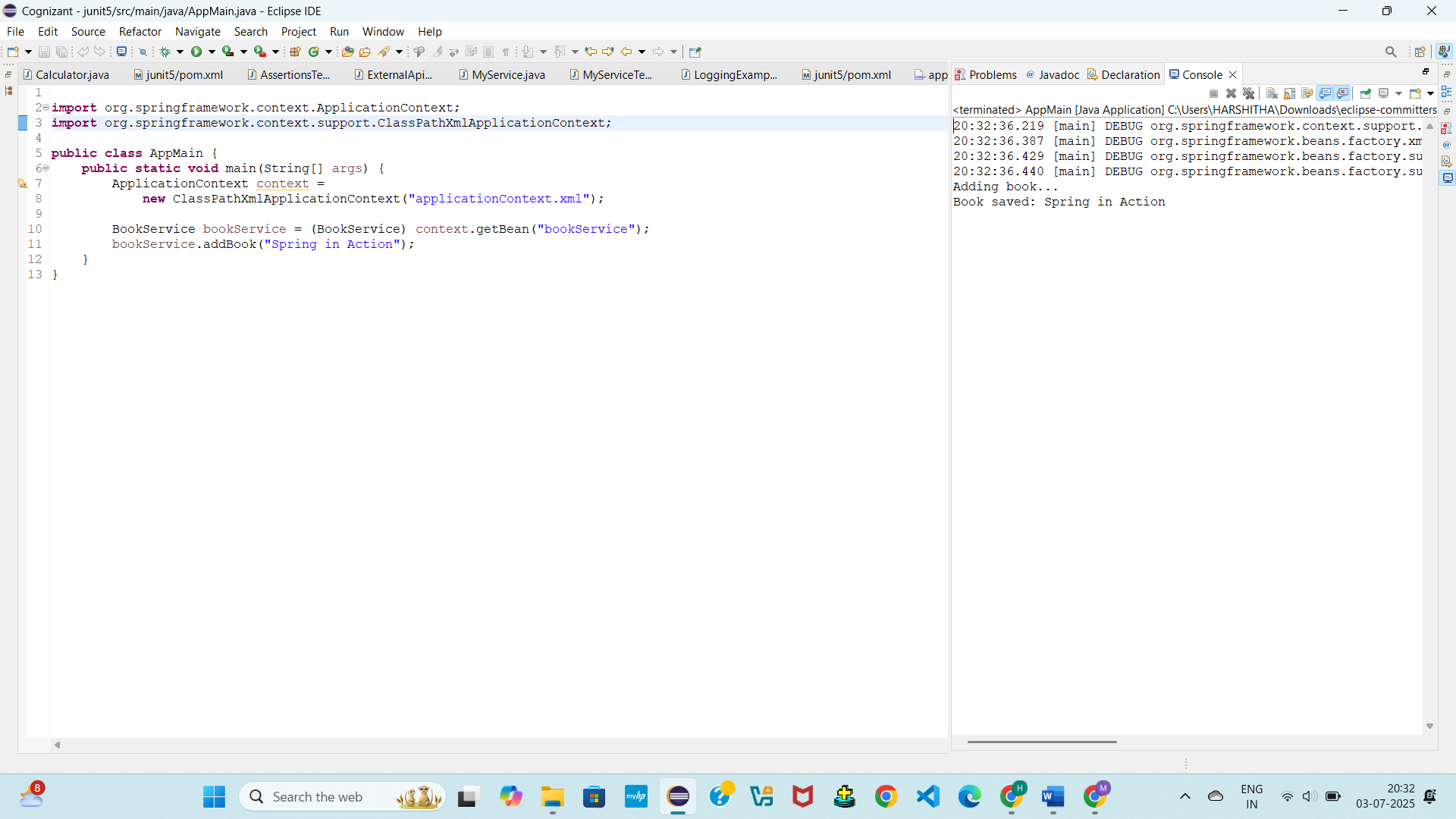
BookService bookService = (BookService) context.getBean("bookService");

bookService.addBook("Spring in Action");

}

}

**OUTPUT:**

****

**Exercise 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>LibraryManagement</artifactId>

<version>1.0-SNAPSHOT</version>

<dependencies>

<!-- Spring Core & 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 WebMVC -->

<dependency>

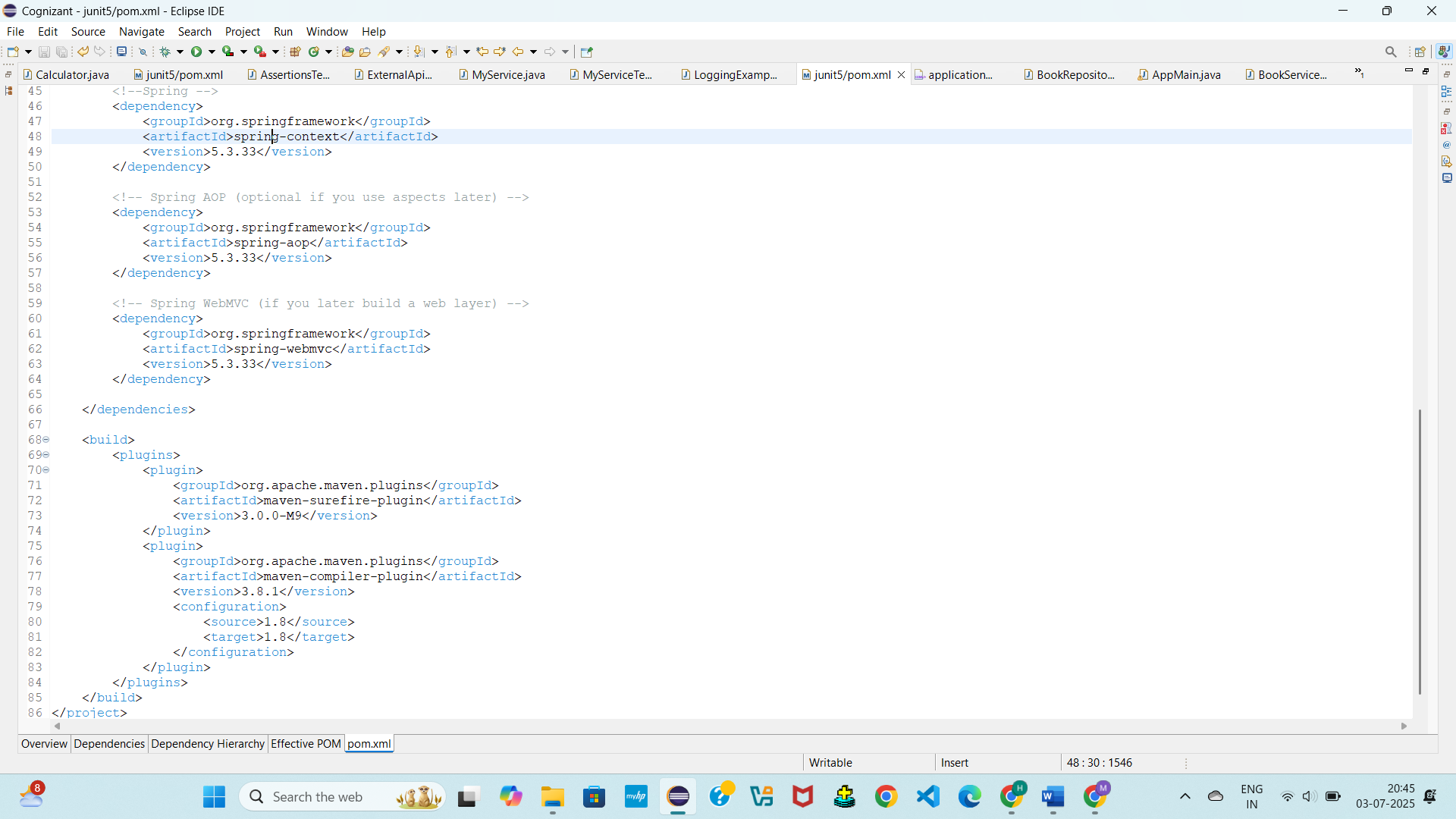
<groupId>org.springframework</groupId>

<artifactId>spring-webmvc</artifactId>

<version>5.3.33</version>

</dependency>

</dependencies>

****

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

**Spring Data JPA - Quick Example**

Spring Data JPA is a powerful abstraction provided by the Spring framework to simplify database interactions. It reduces boilerplate code required for data access layers by allowing developers to use interfaces and annotations instead of verbose SQL or JDBC.

Spring Data JPA builds on top of JPA (Java Persistence API) and integrates seamlessly with Spring Boot. It enables rapid development of repository-based data access layers by leveraging method naming conventions and runtime proxy generation.

**Key Features**

* No SQL needed; methods are auto-implemented
* Integrates with Spring Boot for auto-configuration
* Supports CRUD, pagination, sorting, and custom queries

**Example:**

**1. Entity Definition**

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

@Entity

public class Student {

@Id

private Long id;

private String name;

private int age;

public Student() {}

public Student(Long id, String name, int age) {

this.id = id; this.name = name; this.age = age;

}

}

**2. Repository Interface**

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

public interface StudentRepository extends JpaRepository<Student, Long> {}

**3. Service Usage**

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

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class StudentService {

@Autowired

private StudentRepository repo;

public void saveStudent(Student s) {

repo.save(s);

}

public List<Student> fetchAll() {

return repo.findAll();

}

}

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

In Java-based enterprise applications, data persistence is commonly managed using JPA, Hibernate, and Spring Data JPA. While often used together, these three technologies serve distinct purposes.

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

**Definition:**  
JPA is a **specification** provided by Java EE for object-relational mapping (ORM). It defines **standard interfaces and annotations** for persisting Java objects into relational databases.

**Key Points:**

* JPA is **just an interface layer**, not an implementation.
* It defines how ORM should work, not how it is done.
* Common annotations: @Entity, @Id, @OneToMany, etc.

**2. Hibernate**

**Definition:**  
Hibernate is a **concrete implementation of the JPA specification**. It is an ORM framework that provides the actual code and tools to map Java classes to database tables.

**Key Points:**

* Hibernate is both a **JPA implementation** and an **extended ORM tool**.
* It offers additional features like caching, lazy loading, batch fetching.
* You can use Hibernate directly (without JPA), or as the JPA provider.

**3. Spring Data JPA**

**Definition:**  
Spring Data JPA is a **higher-level abstraction** built on top of JPA (and usually Hibernate) provided by the Spring framework. It simplifies JPA-based data access layers by eliminating boilerplate code.

**Key Points:**

* Uses interfaces like JpaRepository to perform CRUD automatically.
* Supports custom queries via method names or @Query.
* Requires a JPA provider (usually Hibernate) underneath.