**WEEK 3:**

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

**Step 1: Add Spring Core Dependencies to pom.xml**

pom.xml

<dependencies>

<!-- Spring Core -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.33</version>

</dependency>

</dependencies>

**Step 2: Create XML Configuration File**

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

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

<!-- BookRepository Bean -->

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

<!-- BookService Bean -->

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

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

</bean>

</beans>

**Step 3: Create Java Packages and Classes**

**Create Package: com.library.repository**

**Class:BookRepository.java**

package com.library.repository;

public class BookRepository {

public void saveBook() {

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

}

}

**Create Package: com.library.service**

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() {

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

bookRepository.saveBook();

}

}

**Step 4: Create a Main Class to Run**

**Package: com.library**

**Create class: MainApp**

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");

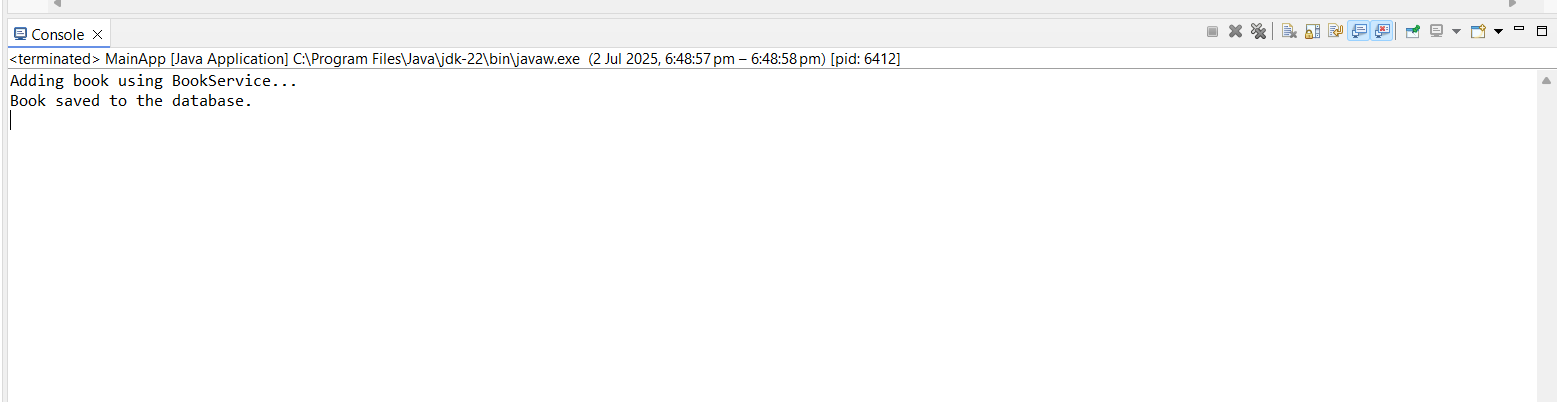
BookService bookService = context.getBean("bookService", BookService.class);

bookService.addBook();

}

}

* **OUTPUT:**



**Exercise 2: Implementing Dependency Injection**

**Step 1: 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

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

<!-- Bean for BookRepository -->

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

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

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

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

</bean>

</beans>

**Step 2: BookService Class**

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

// Setter method for DI

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void addBook() {

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

bookRepository.saveBook();

}

}

**Step 3: Test the Configuration**

**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) {

System.*out*.println("Studying dependency injection");

// Load Spring container

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

// Get BookService bean

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

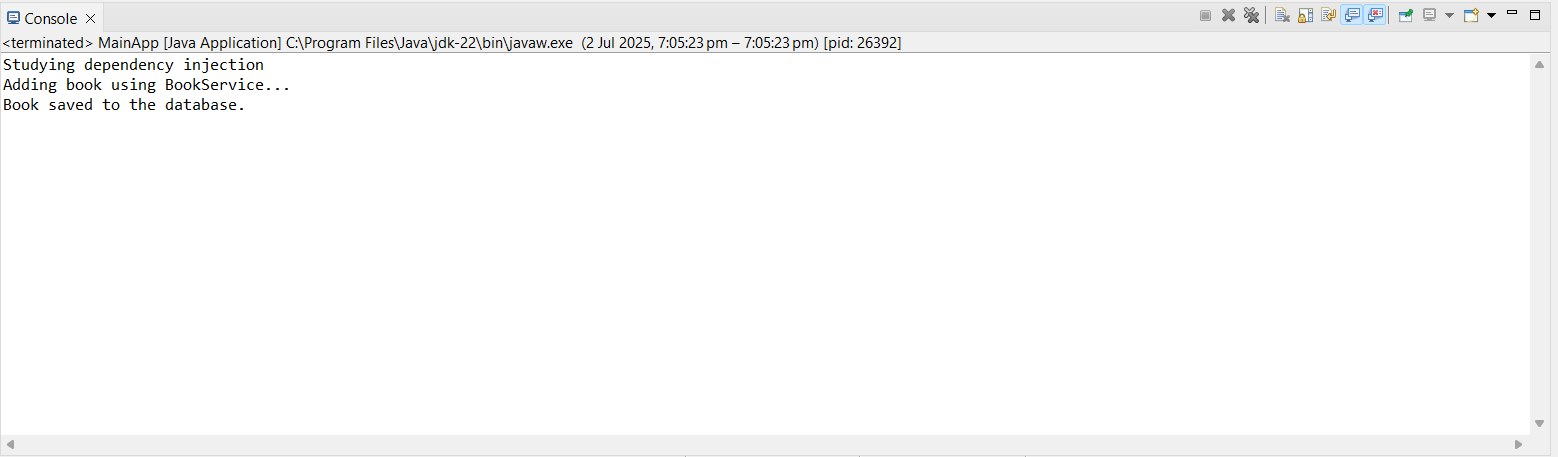
// Call method to test DI

bookService.addBook();

}

}

* **OUTPUT:**

****

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

**Add Spring Dependencies in pom.xml**

<dependencies>

<!-- Spring Context (Core + Beans + Context modules) -->

<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>

<!-- Servlet API (optional, required for Spring WebMVC) -->

<dependency>

<groupId>javax.servlet</groupId>

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

<version>4.0.1</version>

<scope>provided</scope>

</dependency>

</dependencies>

**Configure the Maven Compiler Plugin**

<build>

<plugins>

<!-- Maven Compiler Plugin -->

<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>

**Spring Data JPA - Quick Example**

**STEP 1: Create MySQL Database Schema**

1. Open MySQL client:

CREATE SCHEMA ormlearn;

**2: Add Configuration in application.properties**

# Logging configuration

logging.level.org.springframework=info

logging.level.com.cognizant=debug

logging.level.org.hibernate.SQL=trace

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

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

# Database configuration

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 configuration

spring.jpa.hibernate.ddl-auto=validate

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

**STEP 2: Create country Table in MySQL**

In MySQL:

CREATE TABLE country (

co\_code VARCHAR(2) PRIMARY KEY,

co\_name VARCHAR(50)

);

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

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

**STEP 3: Create Model Class Country**

1. Create package: com.cognizant.ormlearn.model
2. Create class: Country.java

package com.cognizant.ormlearn.model;

import javax.persistence.\*;

@Entity

@Table(name = "country")

public class Country {

@Id

@Column(name = "co\_code")

private String code;

@Column(name = "co\_name")

private String name;

// Getters and Setters

public String getCode() { return code; }

public void setCode(String code) { this.code = code; }

public String getName() { return name; }

public void setName(String name) { this.name = name; }

@Override

public String toString() {

return "Country [code=" + code + ", name=" + name + "]";

}

}

**STEP 4: Create Repository Interface**

1. Create package: com.cognizant.ormlearn.repository
2. Create interface: CountryRepository.java

package com.cognizant.ormlearn.repository;

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

import org.springframework.stereotype.Repository;

import com.cognizant.ormlearn.model.Country;

@Repository

public interface CountryRepository extends JpaRepository<Country, String> {

}

**STEP 5: Create Service Class**

1. Create package: com.cognizant.ormlearn.service
2. Create class: CountryService.java

package com.cognizant.ormlearn.service;

import java.util.List;

import javax.transaction.Transactional;

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

import org.springframework.stereotype.Service;

import com.cognizant.ormlearn.model.Country;

import com.cognizant.ormlearn.repository.CountryRepository;

@Service

public class CountryService {

@Autowired

private CountryRepository countryRepository;

@Transactional

public List<Country> getAllCountries() {

return countryRepository.findAll();

}

}

**STEP 6: Modify OrmLearnApplication.java**

package com.cognizant.ormlearn;

import java.util.List;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.context.ApplicationContext;

import com.cognizant.ormlearn.model.Country;

import com.cognizant.ormlearn.service.CountryService;

@SpringBootApplication

public class OrmLearnApplication {

private static final Logger LOGGER = LoggerFactory.getLogger(OrmLearnApplication.class);

private static CountryService countryService;

public static void main(String[] args) {

ApplicationContext context = SpringApplication.run(OrmLearnApplication.class, args);

LOGGER.info("Inside main");

countryService = context.getBean(CountryService.class);

testGetAllCountries();

}

private static void testGetAllCountries() {

LOGGER.info("Start");

List<Country> countries = countryService.getAllCountries();

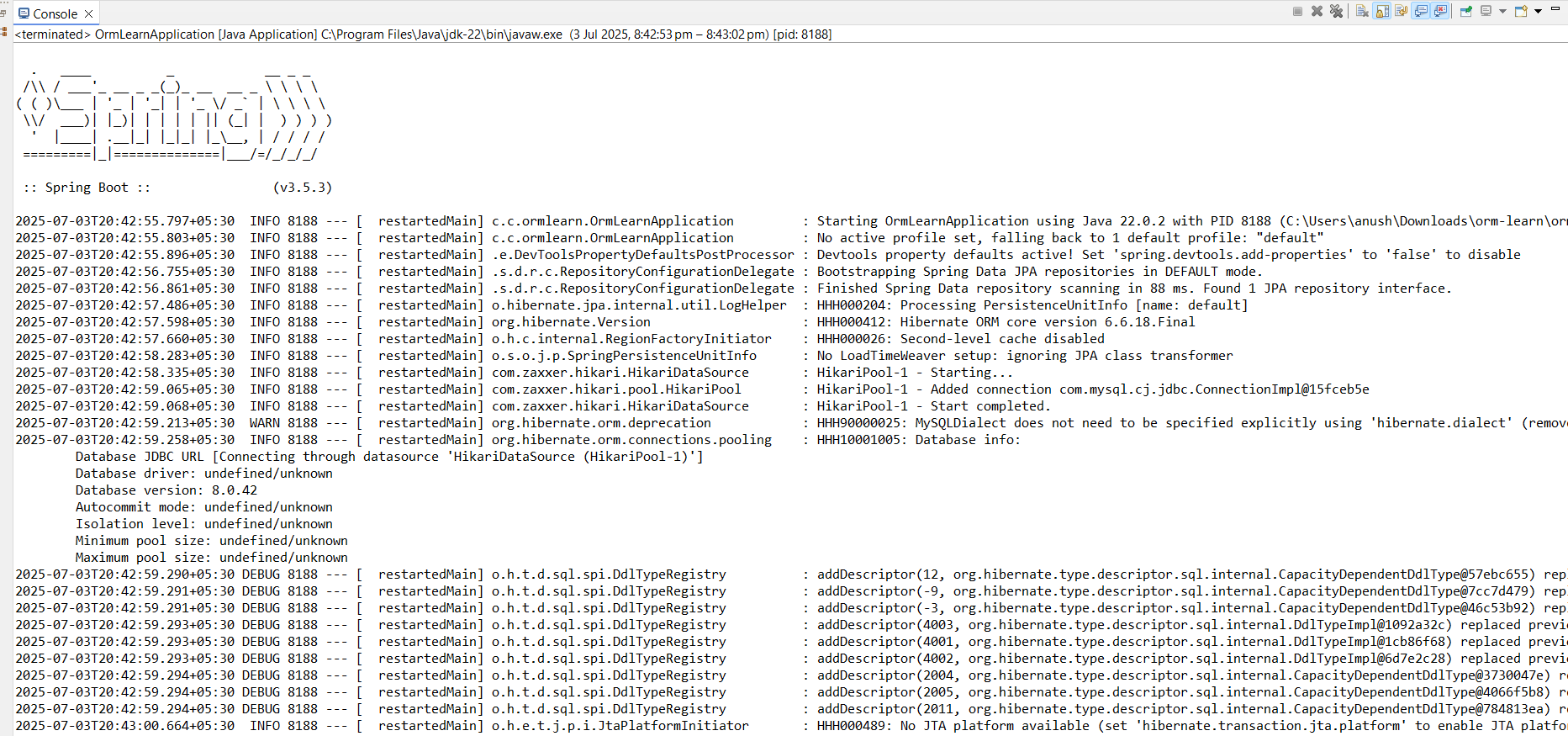
LOGGER.debug("countries={}", countries);

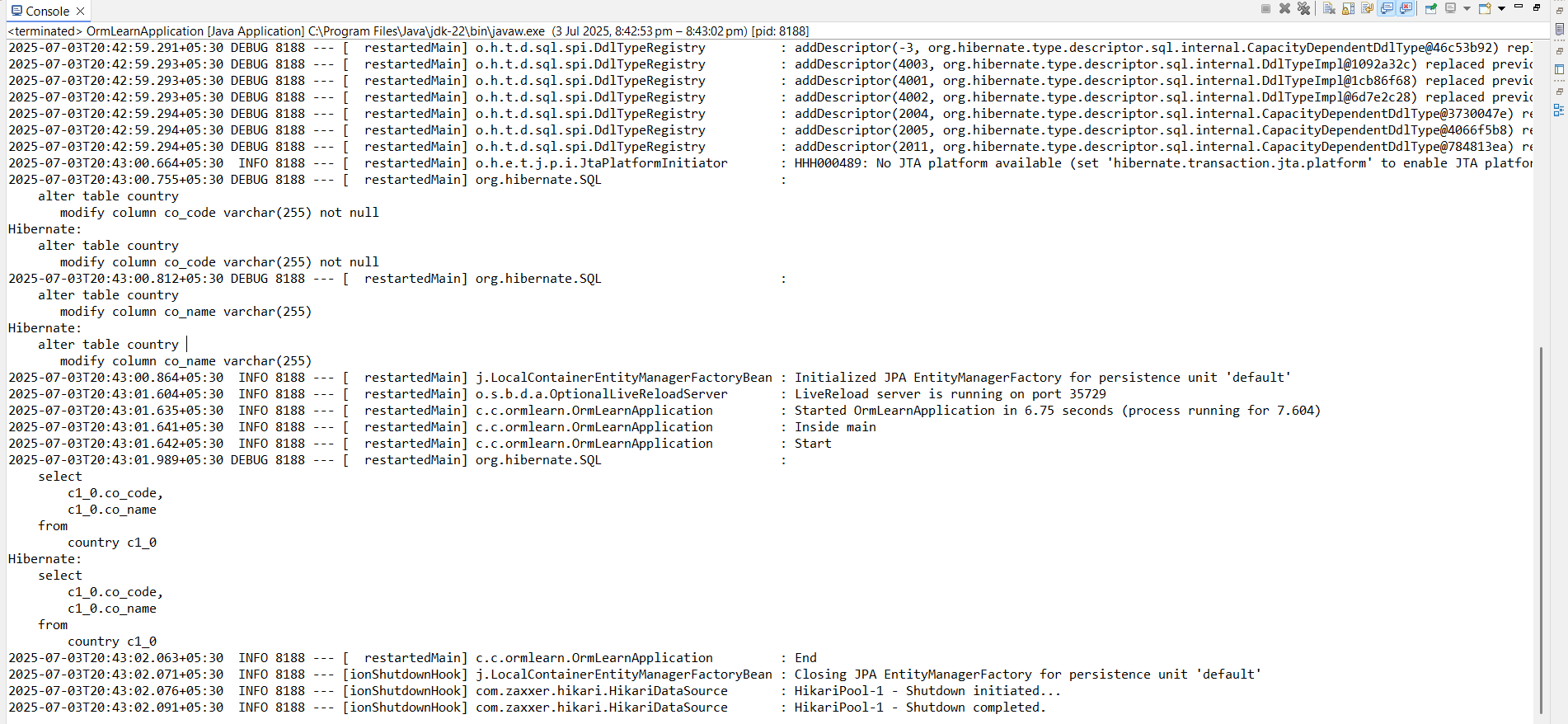
LOGGER.info("End");

}

}

* **OUTPUT:**





**Q.Explain the difference between Java Persistence API, Hibernate and Spring Data JPA**

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

* **JPA is a specification (interface)** for accessing, persisting, and managing data between Java objects and relational databases.
* It is defined in **Jakarta EE (formerly Java EE)**.
* It provides **standard annotations and interfaces** like @Entity, @Id, EntityManager, etc.

**Key Points:**

* Not an implementation — just a set of **rules and APIs**.
* Think of it as the **"what to do"**, not **"how to do it"**.

**Example:**

@Entity

public class Book {

@Id

private Long id;

private String title;

}

**2. Hibernate**

* Hibernate is a **popular implementation of the JPA specification**.
* It provides the **actual logic** behind the interfaces defined in JPA.
* Also includes **extra features** beyond JPA like:
  + Lazy loading
  + Second-level caching
  + Custom HQL (Hibernate Query Language)

**Key Points:**

* It **implements** JPA but also allows using **native Hibernate APIs**.
* You can use Hibernate **with or without JPA**.

**Example:**

Session session = sessionFactory.openSession(); // Hibernate-specific

**3. Spring Data JPA**

* A **Spring framework module** that **builds on top of JPA** (usually with Hibernate as the provider).
* It **simplifies** JPA by:
  + Eliminating boilerplate code
  + Automatically implementing common repository methods
  + Providing **CrudRepository, JpaRepository, etc.**

**Key Points:**

* It uses JPA annotations under the hood.
* You don’t need to write even basic SQL or JPQL for common operations.

**Example:**

public interface BookRepository extends JpaRepository<Book, Long> {

List<Book> findByTitle(String title); // Automatically implemented!

}

**Summary Table**

| **Feature** | **JPA** | **Hibernate** | **Spring Data JPA** |
| --- | --- | --- | --- |
| Type | Specification (interface) | Implementation | Abstraction layer over JPA (helper module) |
| Part of | Jakarta EE / Java EE | Third-party ORM | Spring Framework |
| Requires Boilerplate | Yes | Less than JDBC | Minimal/none |
| Usage | Defines how ORM should work | Does ORM + extra features | Simplifies JPA usage with Spring |
| Common In | Enterprise Java applications | Standalone & Spring apps | Spring Boot app |