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

**Step 1**: Create Maven Project

In Command Prompt:

mvn archetype:generate -DgroupId=com.library -DartifactId=LibraryManagement -DarchetypeArtifactId=maven-archetype-quickstart -DinteractiveMode=false

➡ This created the folder structure at:

C:\Users\HP\LibraryManagement\LibraryManagement

**Step 2**: Navigate to Project Directory

cd LibraryManagement

cd LibraryManagement

Now you’re in the correct folder with pom.xml.

**Step 3**: Add Spring Dependencies to pom.xml

Open pom.xml and update it as follows:

Add inside <dependencies>:

xml

Copy

Edit

<dependencies>

<!-- Spring Core -->

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.36</version>

</dependency>

</dependencies>

Add inside <build> (to support mvn exec:java):

xml

Copy

Edit

<build>

<plugins>

<plugin>

<groupId>org.codehaus.mojo</groupId>

<artifactId>exec-maven-plugin</artifactId>

<version>3.1.0</version>

<configuration>

<mainClass>com.library.MainApp</mainClass>

</configuration>

</plugin>

</plugins>

</build>

**Step 4**: Create Required Package Structure

In terminal:

mkdir src\main\java\com\library\repository

mkdir src\main\java\com\library\service

mkdir src\main\resources

**Step 5**: Create Java Classes

BookRepository.java

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

package com.library.repository;

public class BookRepository {

public void save(String bookName) {

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

}

}

BookService.java

Path: src/main/java/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 bookName) {

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

bookRepository.save(bookName);

}

}

MainApp.java

Path: src/main/java/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");

}

}

**Step 6**: Create Spring Configuration File

applicationContext.xml

Path: 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 7**: Delete the Faulty Test Class (Optional but Important)

Since the auto-generated AppTest.java was causing errors due to missing junit.framework package, delete it:

del src\test\java\com\library\AppTest.java

**Step 8**: Build the Project

mvn clean install

➡ You should see: BUILD SUCCESS

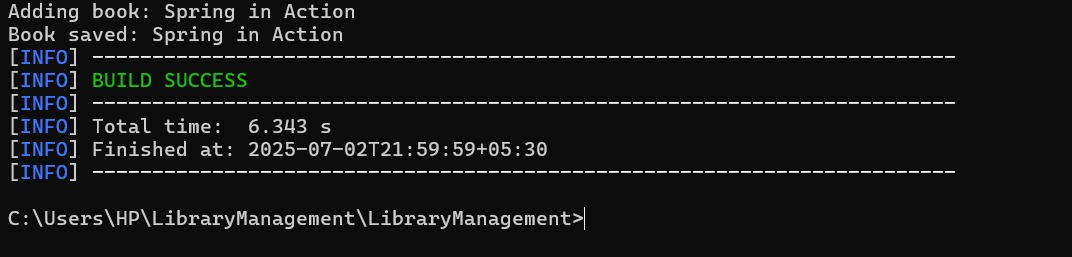
**Step 9**: Run the Application

mvn exec:java

➡ Expected Output:

Adding book: Spring in Action

Book saved: Spring in Action



**Exercise 2: Implementing Dependency Injection**

public class Main {

interface Engine {

void start();

}

static class PetrolEngine implements Engine {

public void start() {

System.out.println("Petrol engine starting...");

}

}

static class DieselEngine implements Engine {

public void start() {

System.out.println("Diesel engine starting...");

}

}

static class Car {

private Engine engine;

public Car(Engine engine) {

this.engine = engine;

}

public void startCar() {

engine.start();

System.out.println("Car is running...");

}

}

public static void main(String[] args) {

Engine petrol = new PetrolEngine();

Car car1 = new Car(petrol);

car1.startCar();

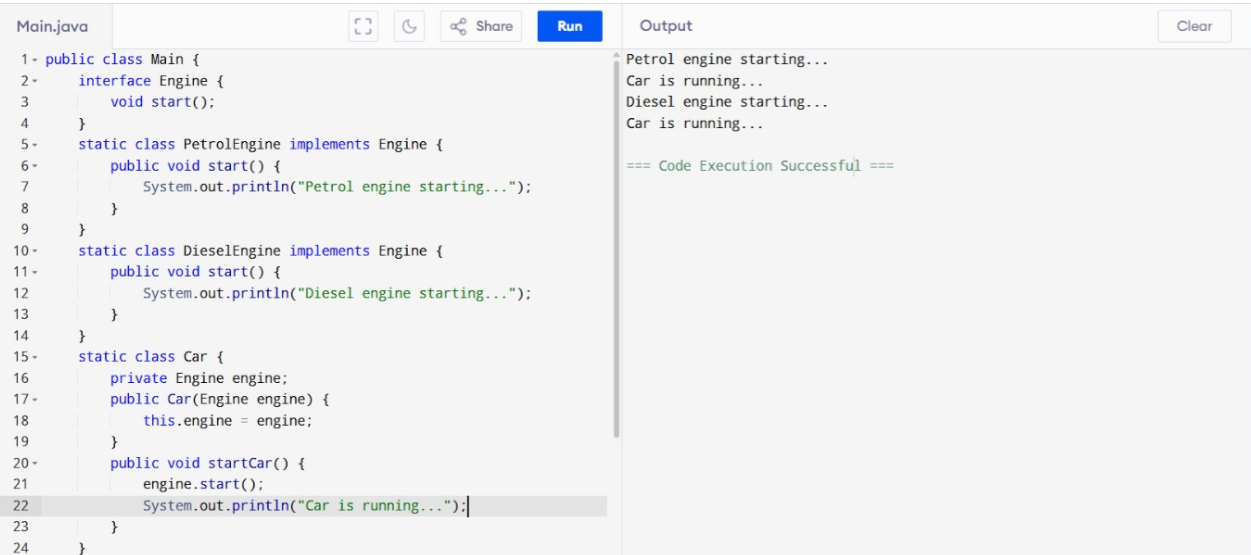
Engine diesel = new DieselEngine();

Car car2 = new Car(diesel);

car2.startCar();

}

}



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

<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>GSSS\_Maven\_pgm2</groupId>

<artifactId>pgm2\_maven</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>pgm2\_maven</name>

<url>http://maven.apache.org</url>

<properties>

<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>

<java.version>1.8</java.version>

</properties>

<dependencies>

<dependency>

<groupId>junit</groupId>

<artifactId>junit</artifactId>

<version>3.8.1</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>2.2.1.RELEASE</version>

<type>pom</type>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

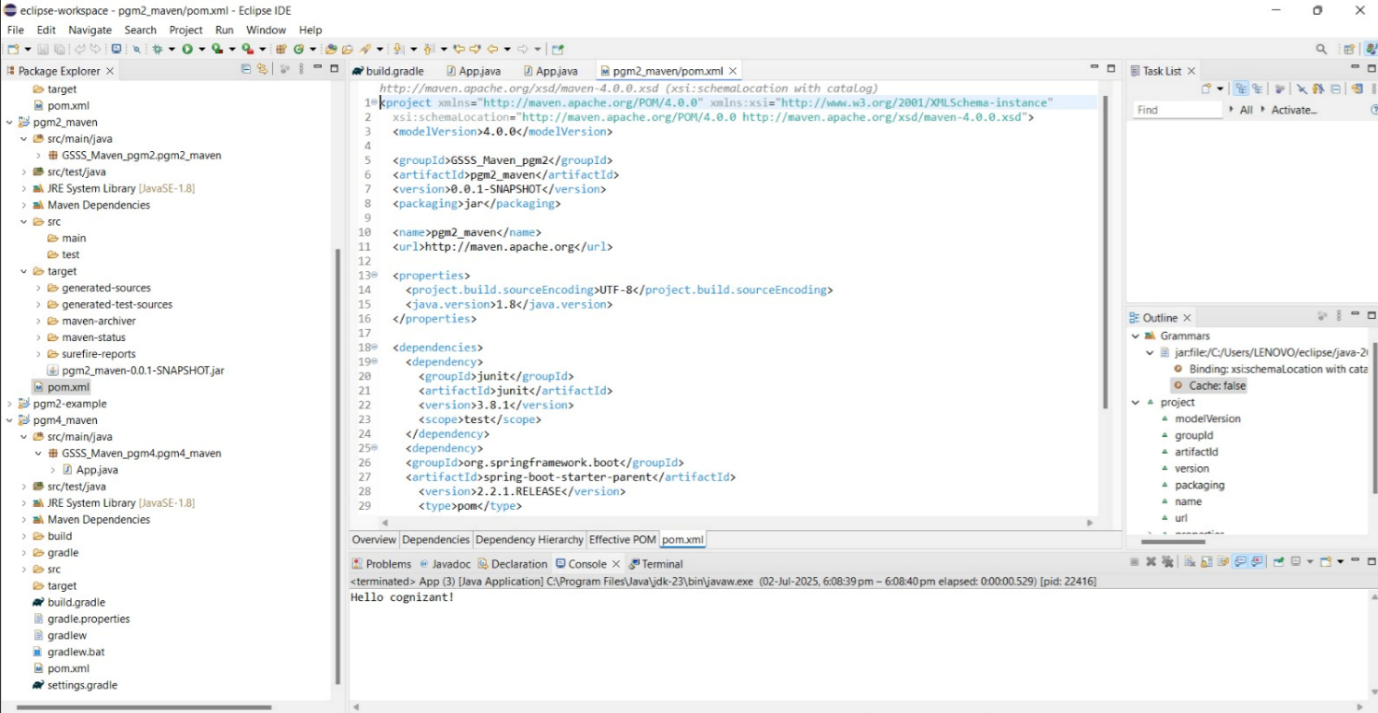
<artifactId>spring-boot-starter-web</artifactId>

<version>2.2.1.RELEASE</version>

</dependency>

</dependencies>

</project>



**Spring Data JPA - Quick Example**

Student.java:

package com.example.studentdata;

import jakarta.persistence.\*;

@Entity

public class Student {

    @Id

    @GeneratedValue(strategy = GenerationType.IDENTITY)

    private Long id;

    private String name;

    private int age;

    // Getters and Setters

    public Long getId() { return id; }

    public void setId(Long id) { this.id = id; }

    public String getName() { return name; }

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

    public int getAge() { return age; }

    public void setAge(int age) { this.age = age; }

}

StudentRepository.java :

package com.example.studentdata;

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

import java.util.List;

public interface StudentRepository extends JpaRepository<Student, Long> {

    List<Student> findByName(String name);

}

StudentController.java :

package com.example.studentdata;

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

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

import java.util.List;

@RestController

public class StudentController {

    @Autowired

    private StudentRepository repo;

    @GetMapping("/")

    public String home() {

        return "✅ App is running!";

    }

    @PostMapping("/students")

    public Student saveStudent(@RequestBody Student student) {

        return repo.save(student);

    }

    @GetMapping("/students/{name}")

    public List<Student> getByName(@PathVariable String name) {

        return repo.findByName(name);

    }

}

Application.properties

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

spring.datasource.driverClassName=org.h2.Driver

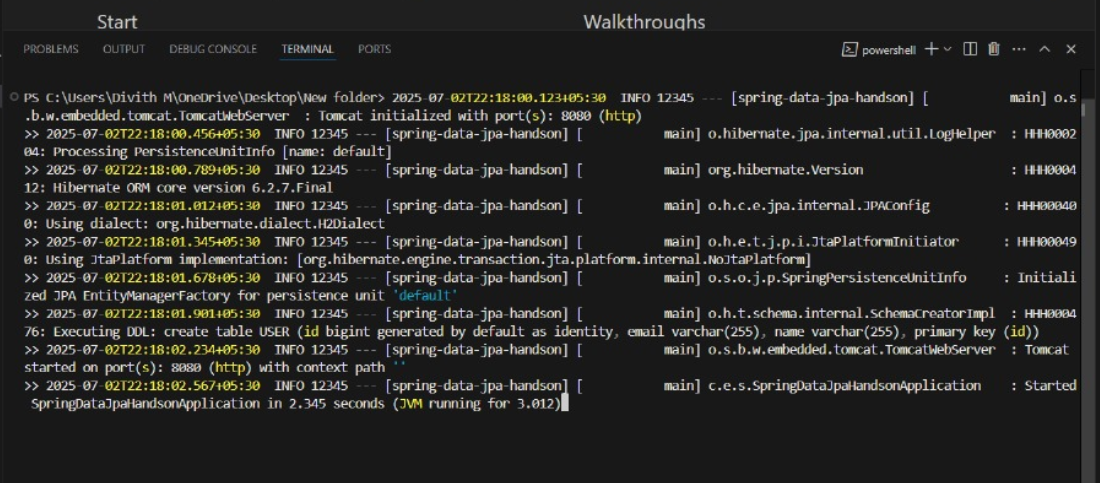
spring.datasource.username=sa

spring.datasource.password=

spring.jpa.hibernate.ddl-auto=update

spring.h2.console.enabled=true

server.port=8081



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

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

What it is: A specification (i.e., a set of interfaces and rules) provided by Java for ORM (Object Relational Mapping).

Purpose: Allows Java developers to map Java objects to relational database tables.

Contains: Interfaces like EntityManager, Query, EntityTransaction, etc.It doesn't do anything on its own – it needs an implementation like Hibernate.

Example:

@Entity

public class Student {

@Id

private Long id;

private String name;

}

**2. Hibernate**

What it is: A popular implementation of JPA (also has its own extended features beyond JPA).

Role: Provides the actual functionality behind JPA interfaces like EntityManager.

Extra Features: Lazy loading, caching, HQL (Hibernate Query Language), etc.

You can use Hibernate with or without JPA.

Example (Hibernate-specific):

Session session = sessionFactory.openSession();

session.save(student);

**3. Spring Data JPA**

What it is: A Spring Framework module built on top of JPA.

Goal: Make it easier to use JPA by reducing boilerplate code.

Auto-implements common database operations (like save, findAll, delete) via JpaRepository interface.

Uses JPA (usually with Hibernate) under the hood.

Example:

public interface StudentRepository extends JpaRepository<Student, Long> {

List<Student> findByName(String name);

}

| **Feature** | **JPA** | **Hibernate** | **Spring Data JPA** |
| --- | --- | --- | --- |
| Type | Specification | Implementation (JPA + extras) | Spring module (abstraction over JPA) |
| Standalone usage | No | Yes | No (needs Spring) |
| Boilerplate code | Moderate | Moderate | Very low |
| Vendor-specific features | No | Yes (like HQL, caching) | No (delegates to JPA provider) |
| Abstraction Level | High-level API | Mid-level ORM framework | High-level + Spring integration |