**Spring Core and Maven**

**Exercise – 1,2: Configuring a Basic Spring Application & Implementing Dependency Injection**

**Code:**MainApp.java

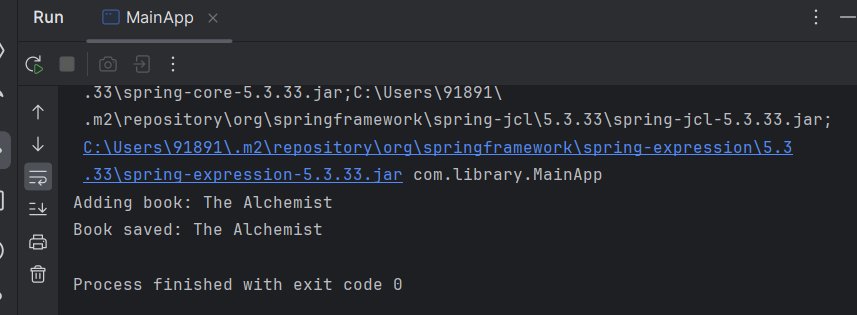
package com.library;  
  
import com.library.service.BookService;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
  
public class MainApp {  
 public static void main(String[] args) {  
 ApplicationContext context = new ClassPathXmlApplicationContext("applicationContext.xml");  
 BookService bookService = (BookService) context.getBean("bookService");  
 bookService.addBook("The Alchemist");  
 }  
}

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.saveBook(bookName);  
 }  
}

BookRepository.java

package com.library.repository;  
  
public class BookRepository {  
 public void saveBook(String bookName) {  
 System.*out*.println("Book saved: " + bookName);  
 }  
}

**Output: **

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

<?xml version="1.0" encoding="UTF-8"?>  
<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>  
  
 <properties>  
 <maven.compiler.source>21</maven.compiler.source>  
 <maven.compiler.target>21</maven.compiler.target>  
 <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  
 </properties>  
 <dependencies>  
 <!-- Spring Core Dependency -->  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-context</artifactId>  
 <version>5.3.33</version>  
 </dependency>  
 <!-- Spring 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 Web MVC -->  
 <dependency>  
 <groupId>org.springframework</groupId>  
 <artifactId>spring-webmvc</artifactId>  
 <version>5.3.33</version>  
 </dependency>  
 </dependencies>  
 <!-- Maven Compiler Plugin -->  
 <build>  
 <plugins>  
 <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>

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

1. A Sample Example on Spring Data JPA

**DEMOAPPLICATION.JAVA**

package com.example.demo;  
  
import org.springframework.boot.CommandLineRunner;  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.context.annotation.Bean;  
  
@SpringBootApplication  
public class DemoApplication {  
 public static void main(String[] args) {  
 SpringApplication.*run*(DemoApplication.class, args);  
 }  
  
 @Bean  
 CommandLineRunner run(UserRepository repo) {  
 return args -> {  
 repo.save(new User("Alice", "alice@example.com"));  
 repo.save(new User("Bob", "bob@example.com"));  
 repo.findAll().forEach(System.*out*::println);  
 };  
 }  
}  
  
**USER.JAVA**

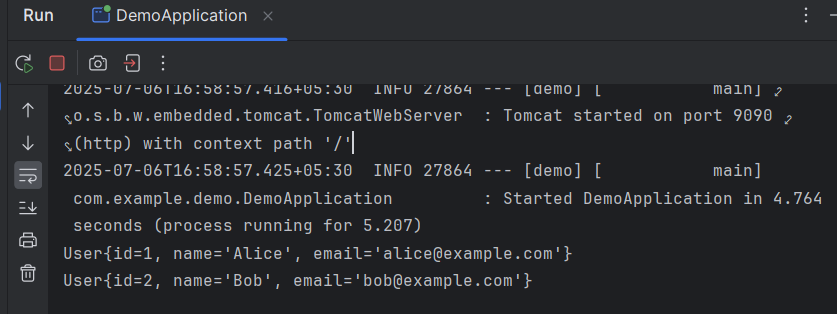
package com.example.demo;  
  
import jakarta.persistence.\*;  
  
@Entity  
@Table(name = "users")  
public class User {  
 @Id  
 @GeneratedValue(strategy = GenerationType.*IDENTITY*)  
 private Long id;  
 private String name;  
 private String email;  
  
 public User() {}  
 public User(String name, String email) {  
 this.name = name;  
 this.email = email;  
 }  
  
 public Long getId() { return id; }  
 public String getName() { return name; }  
 public String getEmail() { return email; }  
  
 @Override  
 public String toString() {  
 return "User{id=" + id + ", name='" + name + "', email='" + email + "'}";  
 }  
}

package com.example.demo;  
  
import org.springframework.data.jpa.repository.JpaRepository;  
  
public interface UserRepository extends JpaRepository<User, Long> {}

**POM.XML**

<?xml version="1.0" encoding="UTF-8"?>  
<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 https://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
 <parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>3.5.3</version>  
 <relativePath/> <!-- lookup parent from repository -->  
 </parent>  
 <groupId>com.example</groupId>  
 <artifactId>demo</artifactId>  
 <version>0.0.1-SNAPSHOT</version>  
 <name>demo</name>  
 <description>Demo project for Spring Boot</description>  
 <url/>  
 <licenses>  
 <license/>  
 </licenses>  
 <developers>  
 <developer/>  
 </developers>  
 <scm>  
 <connection/>  
 <developerConnection/>  
 <tag/>  
 <url/>  
 </scm>  
 <properties>  
 <java.version>21</java.version>  
 </properties>  
 <dependencies>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-data-jpa</artifactId>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-web</artifactId>  
 </dependency>  
  
 <dependency>  
 <groupId>com.h2database</groupId>  
 <artifactId>h2</artifactId>  
 <scope>runtime</scope>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
 </dependencies>  
  
 <build>  
 <plugins>  
 <plugin>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-maven-plugin</artifactId>  
 </plugin>  
 </plugins>  
 </build>  
  
</project>

Output:



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

1. Java Persistence API (JPA)

JPA is not a framework or a library. It is a standard specification defined by Java (JSR 338). Its main goal is to specify how Java applications should connect with relational databases using object-oriented principles. JPA outlines the rules but does not provide actual working code. To use JPA, developers must depend on implementations like Hibernate or Eclipse Link.

In short:

🡪 JPA is the blueprint, not the builder.

2. Hibernate

Hibernate is one of the most popular implementations of the JPA specification. It is a complete ORM (Object-Relational Mapping) framework that makes it easy to store and retrieve Java objects from a database. Hibernate follows JPA and includes additional features such as caching, lazy loading, and a custom query language (HQL).

You can use Hibernate in two ways:

- With JPA (standardized)

- Using its native API (Hibernate-specific)

In short:

🡪 Hibernate is a real working tool that implements JPA and goes beyond it.

3. Spring Data JPA

Spring Data JPA is part of the larger Spring Data project. It builds on JPA and typically uses Hibernate under the hood. It goes a step further by helping to remove repetitive boilerplate code, such as writing query methods and setting up repositories.

With Spring Data JPA, developers can simply create an interface and extend a Spring-provided base interface (Jpa Repository). Spring automatically provides implementations for common database operations like save(), findById(), delete(), and more.

In short:

🡪 Spring Data JPA simplifies working with JPA by leveraging the power of Spring.