**Name:** Paarthivi D

**Superset ID:** 6410961

**DN 4.0 - Java FSE Mandatory Hands-on**

**WEEK – 2**

**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>

        <dependency>

            <groupId>org.springframework</groupId>

            <artifactId>spring-context</artifactId>

            <version>5.3.20</version>

        </dependency>

    </dependencies>

    <build>

        <plugins>

            <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>

</project>

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

package com.library;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

public class MainApp {

public static void main(String[] args) {

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

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

bookService.addBook("The Harry Potter");

}

}

**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) {

bookRepository.saveBook(bookName);

}

}

**BookRepositary.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) {

bookRepository.saveBook(bookName);

}

}

**MainApp.java**

package com.library;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

public class MainApp {

public static void main(String[] args) {

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

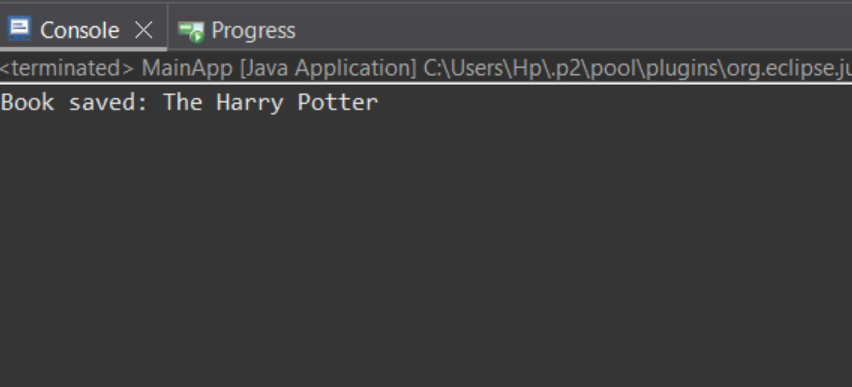
BookService bookService = (BookService) context.getBean("bookService");

bookService.addBook("The Harry Potter");

}

}

**Output:**



**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 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<groupId>com.library</groupId>

<artifactId>LibraryManagement-DI</artifactId>

<version>0.0.1-SNAPSHOT</version>

<dependencies>

<dependency>

<groupId>org.springframework</groupId>

<artifactId>spring-context</artifactId>

<version>5.3.30</version>

</dependency>

</dependencies>

</project>

**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

https://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>

**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) {

bookRepository.saveBook(bookName);

}

}

**MainApp.java**

package com.library;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.library.service.BookService;

public class MainApp {

public static void main(String[] args) {

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

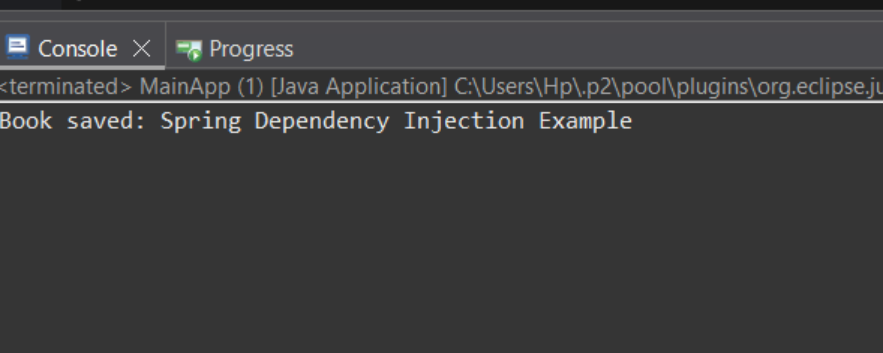
BookService bookService = (BookService) context.getBean("bookService");

bookService.addBook("Spring Dependency Injection Example");

}

}

**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 -->

        <dependency>

            <groupId>org.springframework</groupId>

            <artifactId>spring-context</artifactId>

            <version>5.3.20</version>

        </dependency>

        <!-- Spring AOP -->

        <dependency>

            <groupId>org.springframework</groupId>

            <artifactId>spring-aop</artifactId>

            <version>5.3.20</version>

        </dependency>

        <!-- Spring WebMVC -->

        <dependency>

            <groupId>org.springframework</groupId>

            <artifactId>spring-webmvc</artifactId>

            <version>5.3.20</version>

        </dependency>

    </dependencies>

    <build>

        <plugins>

            <!-- 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>

            <!-- Exec Plugin -->

            <plugin>

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

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

                <version>3.1.0</version>

                <configuration>

                    <mainClass>com.library.LibraryManagementApplication</mainClass>

                </configuration>

            </plugin>

        </plugins>

    </build>

</project>

**LibraryManagementApplication.java**

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

        bookService.displayBook();

    }

}

**BookRepository.java**

package com.library.repository;

public class BookRepository {

    public void fetchBook() {

        System.out.println("Fetching book from repository...");

    }

}

**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"/>

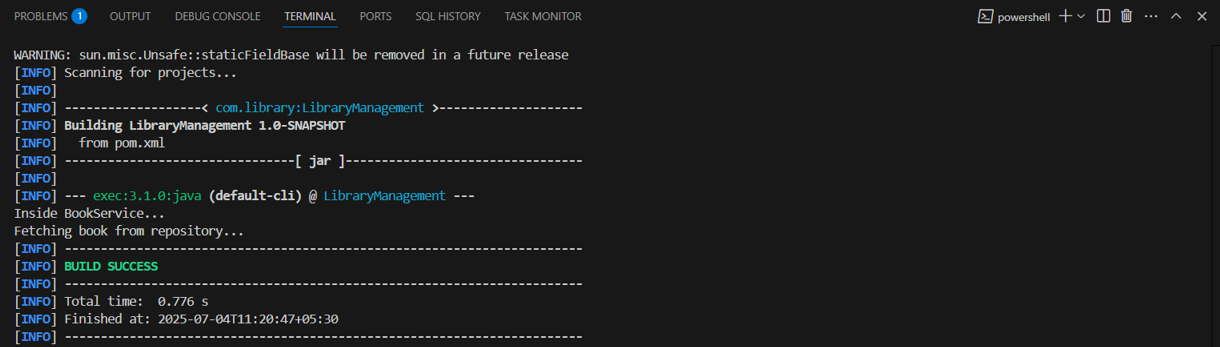
    <bean id="bookService" class="com.library.service.BookService">

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

    </bean>

</beans>

**Output:**



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

**Spring Data JPA - Quick Example**

[**hibernate-hands-on**](https://github.com/selvavignesh1402/Cognizant-Java-FSE-Exercises/tree/main/Week%203/Spring%20Data%20JPA%20with%20Spring%20Boot/hibernate-hands-on)

**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.example</groupId>

  <artifactId>orm-learn</artifactId>

  <version>1.0.0</version>

  <parent>

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

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

    <version>3.2.0</version>

  </parent>

  <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>

    <dependency>

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

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

    </dependency>

  </dependencies>

  <build>

    <plugins>

      <plugin>

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

        <artifactId>spring-boot-maven-plugin</artifactId>

      </plugin>

    </plugins>

  </build>

</project>

**Main.java**

package com.employee;

import com.employee.DAO.EmployeeDAO;

import com.employee.model.Employee;

public class Main {

public static void main(String[] args) {

EmployeeDAO dao = new EmployeeDAO();

Employee emp = new Employee();

emp.setName("Sachin");

emp.setSalary(80000);

dao.addEmployee(emp);

}

}

**Employee.java**

package com.employee.model;

import javax.persistence.\*;

@Entity

@Table(name = "employee")

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int id;

private String name;

private double salary;

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public double getSalary() {

return salary;

}

public void setSalary(double salary) {

this.salary = salary;

}

// Getters and setters

}

**EmployeeDAO.java**

package com.employee.DAO;

import org.hibernate.Session;

import org.hibernate.SessionFactory;

import org.hibernate.Transaction;

import org.hibernate.cfg.Configuration;

import com.employee.model.Employee;

public class EmployeeDAO {

public Integer addEmployee(Employee employee) {

SessionFactory factory = new Configuration().configure().buildSessionFactory();

Session session = factory.openSession();

Transaction tx = null;

Integer empId = null;

try {

tx = session.beginTransaction();

empId = (Integer) session.save(employee);

tx.commit();

} catch (Exception e) {

if (tx != null) tx.rollback();

e.printStackTrace();

} finally {

session.close();

}

return empId;

}

}

**hibernate.cfg.xml**

<!DOCTYPE hibernate-configuration PUBLIC

"-//Hibernate/Hibernate Configuration DTD 3.0//EN"

"http://www.hibernate.org/dtd/hibernate-configuration-3.0.dtd">

<hibernate-configuration>

<session-factory>

<property name="hibernate.connection.driver\_class">com.mysql.cj.jdbc.Driver</property>

<property name="hibernate.connection.url">jdbc:mysql://localhost:3306/ormlearn</property>

<property name="hibernate.connection.username">root</property>

<property name="hibernate.connection.password">root</property>

<property name="hibernate.dialect">org.hibernate.dialect.MySQL8Dialect</property>

<property name="show\_sql">true</property>

<property name="hibernate.hbm2ddl.auto">update</property>

<mapping class="com.employee.model.Employee"/>

</session-factory>

</hibernate-configuration>

**.project**

<?xml version="1.0" encoding="UTF-8"?>

<projectDescription>

<name>Hibernate</name>

<comment></comment>

<projects>

</projects>

<buildSpec>

<buildCommand>

<name>org.eclipse.jdt.core.javabuilder</name>

<arguments>

</arguments>

</buildCommand>

<buildCommand>

<name>org.eclipse.m2e.core.maven2Builder</name>

<arguments>

</arguments>

</buildCommand>

</buildSpec>

<natures>

<nature>org.eclipse.jdt.core.javanature</nature>

<nature>org.eclipse.m2e.core.maven2Nature</nature>

</natures>

</projectDescription>

**.classpath**

<?xml version="1.0" encoding="UTF-8"?>

<classpath>

<classpathentry kind="src" output="target/classes" path="src/main/java">

<attributes>

<attribute name="optional" value="true"/>

<attribute name="maven.pomderived" value="true"/>

</attributes>

</classpathentry>

<classpathentry excluding="\*\*" kind="src" output="target/classes" path="src/main/resources">

<attributes>

<attribute name="maven.pomderived" value="true"/>

<attribute name="optional" value="true"/>

</attributes>

</classpathentry>

<classpathentry kind="src" output="target/test-classes" path="src/test/java">

<attributes>

<attribute name="optional" value="true"/>

<attribute name="maven.pomderived" value="true"/>

<attribute name="test" value="true"/>

</attributes>

</classpathentry>

<classpathentry excluding="\*\*" kind="src" output="target/test-classes" path="src/test/resources">

<attributes>

<attribute name="maven.pomderived" value="true"/>

<attribute name="test" value="true"/>

<attribute name="optional" value="true"/>

</attributes>

</classpathentry>

<classpathentry kind="con" path="org.eclipse.jdt.launching.JRE\_CONTAINER/org.eclipse.jdt.internal.debug.ui.launcher.StandardVMType/JavaSE-1.8">

<attributes>

<attribute name="maven.pomderived" value="true"/>

</attributes>

</classpathentry>

<classpathentry kind="con" path="org.eclipse.m2e.MAVEN2\_CLASSPATH\_CONTAINER">

<attributes>

<attribute name="maven.pomderived" value="true"/>

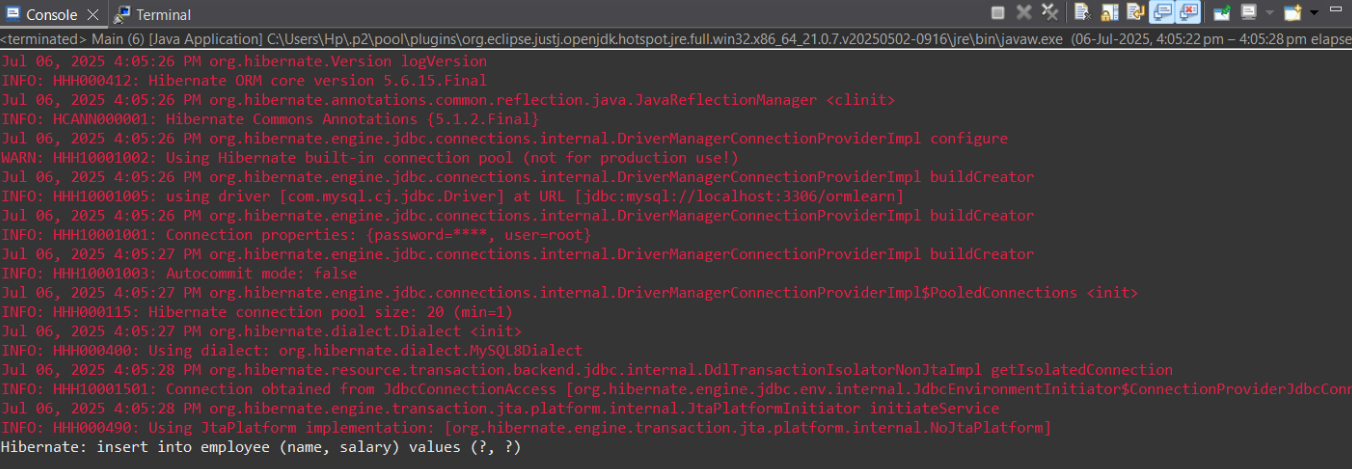
</attributes>

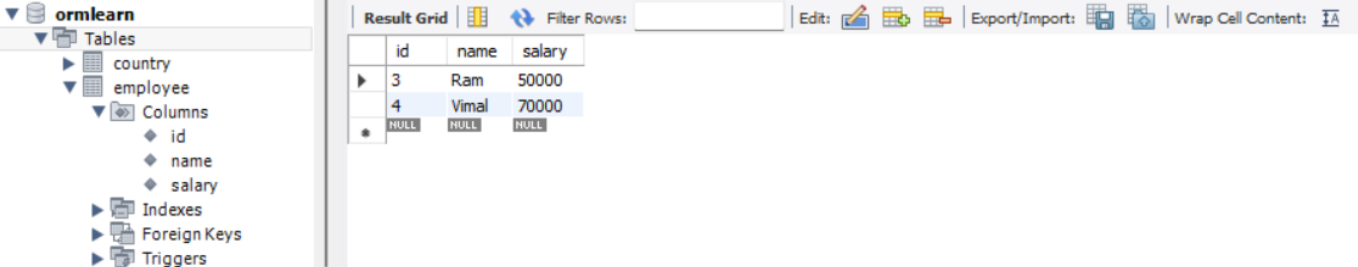
</classpathentry>

<classpathentry kind="output" path="target/classes"/>

</classpath>

**Output**





[**orm-learn**](https://github.com/selvavignesh1402/Cognizant-Java-FSE-Exercises/tree/main/Week%203/Spring%20Data%20JPA%20with%20Spring%20Boot/orm-learn)

**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 com.cognizant.ormlearn.model.Country;

import com.cognizant.ormlearn.service.CountryService;

import org.springframework.context.ApplicationContext;

@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");

}

}

**Country.java**

package com.cognizant.ormlearn.model;

import jakarta.persistence.Column;

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

import jakarta.persistence.Table;

@Entity

@Table(name = "country")

public class Country {

@Id

@Column(name = "co\_code")

private String code;

@Column(name = "co\_name")

private String name;

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 + '\'' +

'}';

}

}

**CountryService.java**

package com.cognizant.ormlearn.service;

import java.util.List;

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

import org.springframework.stereotype.Service;

import com.cognizant.ormlearn.model.Country;

import com.cognizant.ormlearn.repository.CountryRepository;

import jakarta.transaction.Transactional;

@Service

public class CountryService {

@Autowired

private CountryRepository countryRepository;

@Transactional

public List<Country> getAllCountries() {

return countryRepository.findAll();

}

}

**application.properties**

spring.application.name=orm-learn

# Spring Framework and application log

logging.level.org.springframework=info

logging.level.com.cognizant=debug

# Hibernate logs for displaying executed SQL, input and output

logging.level.org.hibernate.SQL=trace

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

# Log pattern

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.database-platform=org.hibernate.dialect.MySQL8Dialect

**OrmLearnApplicationTests.java**

package com.cognizant.orm\_learn;

import org.junit.jupiter.api.Test;

import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest

class OrmLearnApplicationTests {

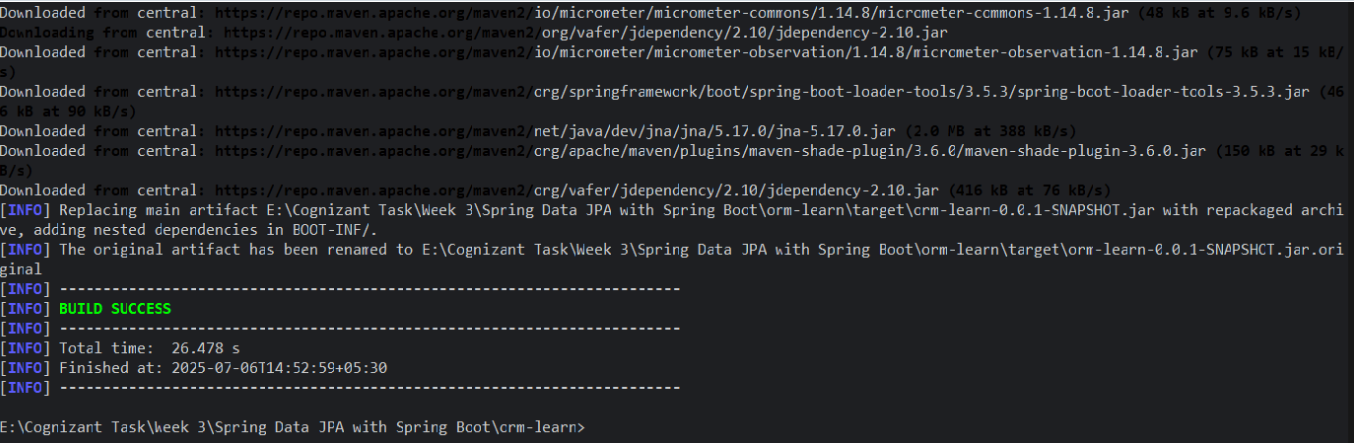
@Test

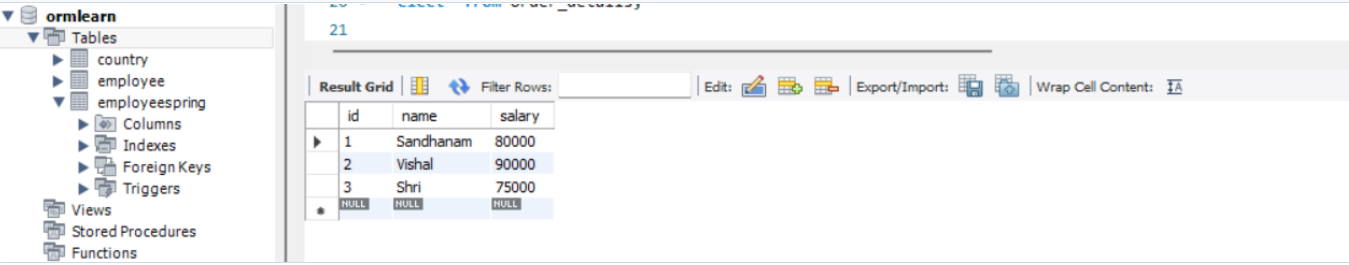
void contextLoads() {

}

}

**Output**





**springdatajpa**

**EmployeeRepository.java**

package com.employee.repository;

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

import org.springframework.stereotype.Repository;

import com.employee.model.Employee;

@Repository

public interface EmployeeRepository extends JpaRepository<Employee, Integer> {

}

**Employee.java**

package com.employee.model;

import jakarta.persistence.Entity;

import jakarta.persistence.GeneratedValue;

import jakarta.persistence.GenerationType;

import jakarta.persistence.Id;

import jakarta.persistence.Table;

@Entity

@Table(name = "employeespring")

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private int id;

private String name;

private double salary;

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public double getSalary() {

return salary;

}

public void setSalary(double salary) {

this.salary = salary;

}

// Getters and setters

}

**SpringdatajpaHandsOnApplication.java**

package com.employee.springdatajpa\_hands\_on;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import org.springframework.boot.autoconfigure.domain.EntityScan;

import com.employee.model.Employee;

import com.employee.service.EmployeeService;

import org.springframework.context.ApplicationContext;

import org.springframework.context.annotation.ComponentScan;

import org.springframework.data.jpa.repository.config.EnableJpaRepositories;

@SpringBootApplication

@EntityScan(basePackages = "com.employee.model")

@EnableJpaRepositories(basePackages = "com.employee.repository")

@ComponentScan(basePackages = "com.employee")

public class SpringdatajpaHandsOnApplication {

private static EmployeeService employeeService;

public static void main(String[] args) {

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

employeeService = context.getBean(EmployeeService.class);

Employee emp = new Employee();

emp.setName("Shri");

emp.setSalary(75000);

employeeService.addEmployee(emp);

}

}

**EmployeeService.java**

package com.employee.service;

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

import org.springframework.stereotype.Service;

import com.employee.model.Employee;

import com.employee.repository.EmployeeRepository;

import jakarta.transaction.Transactional;

@Service

public class EmployeeService {

@Autowired

private EmployeeRepository employeeRepository;

@Transactional

public void addEmployee(Employee employee) {

employeeRepository.save(employee);

}

}

**SpringdatajpaHandsOnApplicationTests.java**

package com.employee.springdatajpa\_hands\_on;

import org.junit.jupiter.api.Test;

import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest

class SpringdatajpaHandsOnApplicationTests {

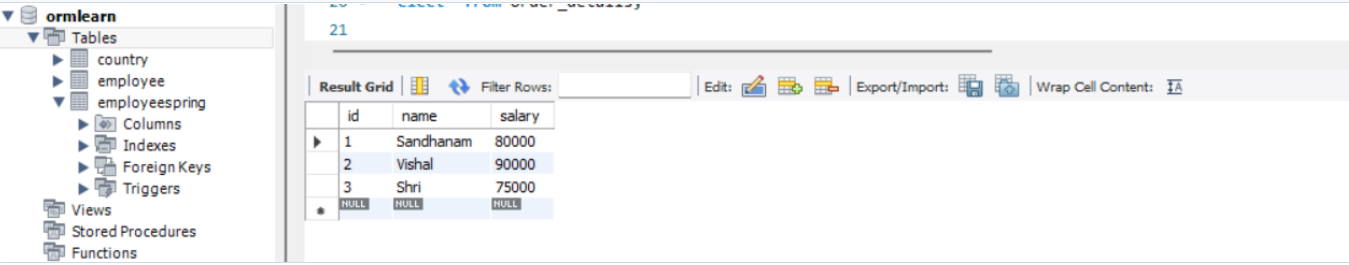
@Test

void contextLoads() {

}

}

**Output:**



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

**1. JPA – Java Persistence API**

JPA is a **standard specification** used for object-relational mapping (ORM) in Java. Developed as part of Java EE (now Jakarta EE), JPA provides a set of **interfaces and annotations** to manage relational data in Java applications.

**Key features of JPA:**

* Provides annotations like @Entity, @Id, @ManyToOne, etc.
* Offers APIs such as EntityManager, TypedQuery, and CriteriaBuilder
* JPA itself doesn’t perform ORM—it needs an implementation like **Hibernate** to function.

In essence, JPA defines *how* data should be persisted, not *how* it is implemented.

**2. Hibernate**

Hibernate is a **popular ORM framework** and the most commonly used JPA provider. It fully implements the JPA standard and offers a range of **advanced capabilities** beyond JPA.

**Features of Hibernate:**

* Support for **caching** (first and second-level)
* **Lazy/eager loading** of relationships
* Querying via **HQL (Hibernate Query Language)**
* Automatic **schema creation and updates**
* Provides batch fetching, dirty checking, and session management

Hibernate can work **with or without** using the JPA interface—it can be used as a standalone ORM framework.

**3. Spring Data JPA**

Spring Data JPA is a module of the **Spring Framework** that simplifies JPA-based data access using **repositories**. It acts as an abstraction layer over JPA and Hibernate, offering **ready-to-use** features that cut down on boilerplate code.

**Why Spring Data JPA is useful:**

* Offers interfaces like JpaRepository, CrudRepository, and PagingAndSortingRepository
* Automatically generates queries based on method names
* Integrates seamlessly with **Spring Boot**
* Supports pagination, sorting, and custom JPQL queries using @Query

Internally, it uses JPA annotations and Hibernate as the default ORM provider.

**Comparison Summary**

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **JPA** | **Hibernate** | **Spring Data JPA** |
| Type | Specification (API) | ORM Framework | Spring Module (built on JPA) |
| Developed By | Oracle / Jakarta EE | Red Hat | Spring Team (Pivotal) |
| Can run independently? | No, needs a provider | Yes | Yes, but depends on JPA provider |
| Main Use | Defines ORM rules | Performs ORM operations | Eases JPA usage in Spring |
| Typical Usage | With a provider like Hibernate | Standalone or via JPA | With Spring Boot & JPA repositories |

**Understanding the Role of Each**

**What Problem Does JPA Solve?**

Before JPA, developers relied on JDBC, which required:

* Manual connection handling
* Writing SQL queries for CRUD
* Mapping rows to Java objects manually

JPA introduced an **object-oriented approach**, allowing you to:

* Persist Java classes as database records
* Use annotations instead of SQL
* Interact with entities rather than rows and columns

**Why Hibernate Gained Popularity**

Hibernate grew in popularity due to:

* Powerful ORM features not available in early JPA versions
* Support for complex mappings and queries
* Automatic DDL generation
* Advanced features like second-level caching, interceptors, and auditing

**Advantages of Spring Data JPA**

Spring Data JPA streamlines persistence logic by:

* Providing **repository interfaces** with ready-to-use methods
* Eliminating the need for writing repetitive DAO classes
* Supporting method-based query derivation
* Enabling @Query annotation for custom queries

You can get results with lines like:

List<User> findByRole(String role);

**Real-Life Analogy**

Imagine you're building a house:

* **JPA** is the **architect's design** (specification)
* **Hibernate** is the **builder** who constructs based on that design
* **Spring Data JPA** is the **construction manager** who speeds up the process with tools and pre-defined tasks

**Conclusion**

* **JPA** sets the standard.
* **Hibernate** implements the standard and goes further.
* **Spring Data JPA** makes development faster and cleaner by integrating ORM into the Spring ecosystem.

Together, they offer a complete and powerful solution for data persistence in modern Java applications.