**Exercise 1: Employee Management System - Overview and Setup**

**application.properties**

# H2 DB Configuration

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

spring.datasource.driverClassName=org.h2.Driver

spring.datasource.username=sa

spring.datasource.password=password

spring.jpa.database-platform=org.hibernate.dialect.H2Dialect

# JPA Behavior

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=update

# H2 Console

spring.h2.console.enabled=true

spring.h2.console.path=/h2-console

**Department.java**

package com.example.ems.model;

import jakarta.persistence.\*;

import lombok.Data;

@Entity

@Data

public class Department {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

}

**Employee.java**

package com.example.ems.model;

import jakarta.persistence.\*;

import lombok.Data;

@Entity

@Data

public class Employee {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String name;

@ManyToOne

@JoinColumn(name = "department\_id")

private Department department;

}

**DepartmentRepository.java**

package com.example.ems.repository;

import com.example.ems.model.Department;

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

public interface DepartmentRepository extends JpaRepository<Department, Long> {

}

**EmployeeRepository.java**

package com.example.ems.repository;

import com.example.ems.model.Employee;

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

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

}

**EmployeeController.java**

package com.example.ems.controller;

import com.example.ems.model.Employee;

import com.example.ems.repository.EmployeeRepository;

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

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

import java.util.List;

@RestController

@RequestMapping("/employees")

public class EmployeeController {

@Autowired

private EmployeeRepository employeeRepository;

@GetMapping

public List<Employee> getAllEmployees() {

return employeeRepository.findAll();

}

@PostMapping

public Employee createEmployee(@RequestBody Employee employee) {

return employeeRepository.save(employee);

}

}

**DepartmentController.java**

package com.example.ems.controller;

import com.example.ems.model.Department;

import com.example.ems.repository.DepartmentRepository;

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

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

import java.util.List;

@RestController

@RequestMapping("/departments")

public class DepartmentController {

@Autowired

private DepartmentRepository departmentRepository;

@GetMapping

public List<Department> getAllDepartments() {

return departmentRepository.findAll();

}

@PostMapping

public Department createDepartment(@RequestBody Department department) {

return departmentRepository.save(department);

}

}

**EmployeeManagementSystemApplication.java**

package com.example.ems;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class EmployeeManagementSystemApplication {

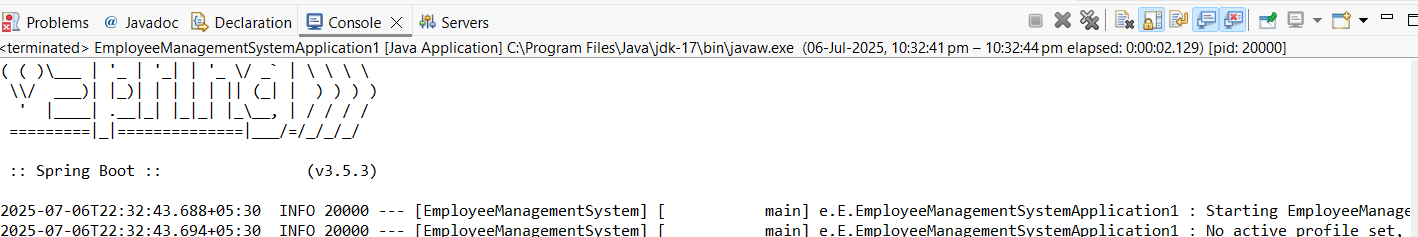
public static void main(String[] args) {

SpringApplication.run(EmployeeManagementSystemApplication.class, args);

}

}

**Output**

****

Difference Between JPA, Hibernate, and Spring Data JPA

# 1. JPA (Java Persistence API)

• JPA is a Java specification for accessing, persisting, and managing data between Java objects and relational databases.

• It is not a framework; it only defines a set of interfaces and annotations (e.g., @Entity, @Id, @OneToMany).

• Provided by Jakarta EE (formerly Java EE).

• It requires an implementation like Hibernate to perform operations.

• Example:  
 EntityManager em = entityManagerFactory.createEntityManager();  
 em.persist(new Employee());

# 2. Hibernate

• Hibernate is a powerful ORM framework that implements the JPA specification.

• Provided by Red Hat.

• Can be used with JPA or with its own native APIs (e.g., SessionFactory, HQL).

• Offers additional features like caching, batch processing, and native SQL support.

• Example:  
 Session session = sessionFactory.openSession();  
 session.save(new Employee());

# 3. Spring Data JPA

• Spring Data JPA is part of the Spring ecosystem that builds on top of JPA.

• Provided by the Spring team (VMware).

• Simplifies repository layer by reducing boilerplate code.

• Offers auto-generated queries and integrates seamlessly with Spring Boot.

• Example:  
 public interface EmployeeRepository extends JpaRepository<Employee, Long> {  
 List<Employee> findByName(String name);  
 }

# 4. Relationship Among Them

• Spring Data JPA uses JPA under the hood.

• JPA requires an implementation like Hibernate.

• Hibernate performs the actual ORM work.

• Flow: Spring Data JPA → JPA → Hibernate → Database

# 5. Comparison Table

| **Feature** | **JPA** | **Hibernate** | **Spring Data JPA** |
| --- | --- | --- | --- |
| **Type** | Specification (interface) | Framework (implementation) | Framework/Abstraction Layer |
| **Ownership** | Jakarta EE / Oracle | Red Hat | Spring (VMware) |
| **Uses** | Interfaces & annotations | Native APIs + JPA APIs | JPA + Spring magic |
| **Boilerplate Code** | Medium | Medium | Very Low |
| **Ease of Use** | Moderate | Moderate | Easy |
| **Custom Queries** | JPQL | HQL | JPQL, @Query, Query Methods |
| **Transactions** | Yes | Yes | Spring-managed with @Transactional |
| **Caching** | Not specified | Yes (1st and 2nd level) | Inherited from JPA provider |
| **Spring Boot Friendly** | Partial | Yes | Yes (fully integrated) |

# 6. Conclusion

• JPA defines the rules (standard interfaces).

• Hibernate implements those rules (actual logic).

• Spring Data JPA makes working with JPA much easier and faster by reducing code.