**Exercise 3: Employee Management System - Creating Repositories**

Create EmployeeRepository Interface:

package com.example.employeemanagementsystem.repository;

import com.example.employeemanagementsystem.entity.Employee;

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

import java.util.List;

public interface EmployeeRepository extends JpaRepository<Employee, Long> {

// Derived query method to find employees by department List<Employee> findByDepartmentId(Long departmentId);

// Derived query method to find employees by name List<Employee> findByName(String name);

}

Create DepartmentRepository Interface:

package com.example.employeemanagementsystem.repository;

import com.example.employeemanagementsystem.entity.Department;

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

public interface DepartmentRepository extends JpaRepository<Department, Long> {

// Derived query method to find departments by name Department findByName(String name);

}

* **JpaRepository<T, ID>**: This interface provides JPA-related methods for standard CRUD operations. T is the entity type, and ID is the type of the entity's primary key.
* **Derived Query Methods:** Spring Data JPA allows you to define methods in the repository interface that follow a certain naming convention, like findBy, countBy, deleteBy, etc. Spring Data will automatically generate the necessary query based on the method name.

**Final Steps**

1. **Inject Repositories into Services:** You can now inject these repositories into your service classes to perform business logic operations.
2. **Testing:** Test the repositories by creating, reading, updating, and deleting entities via your service layer or directly in a test class.
3. **Extend Functionality :** Add more custom queries or business logic as needed.

With these repositories in place, you can now efficiently perform CRUD operations on Employee and Department entities in your Employee Management System.