**Exercise 5: Employee Management System – Defining Query Methods**

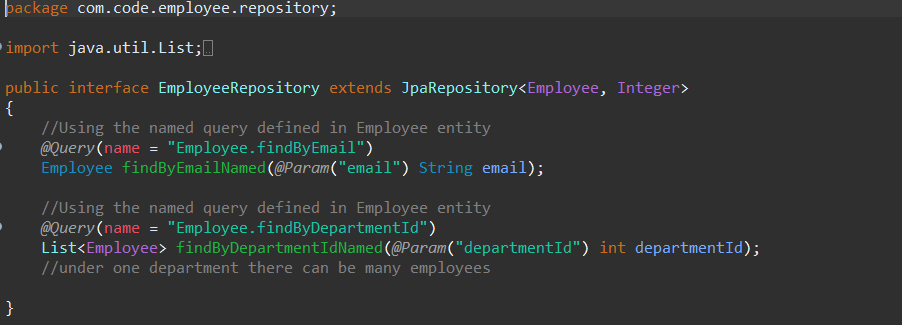
Business Scenario:

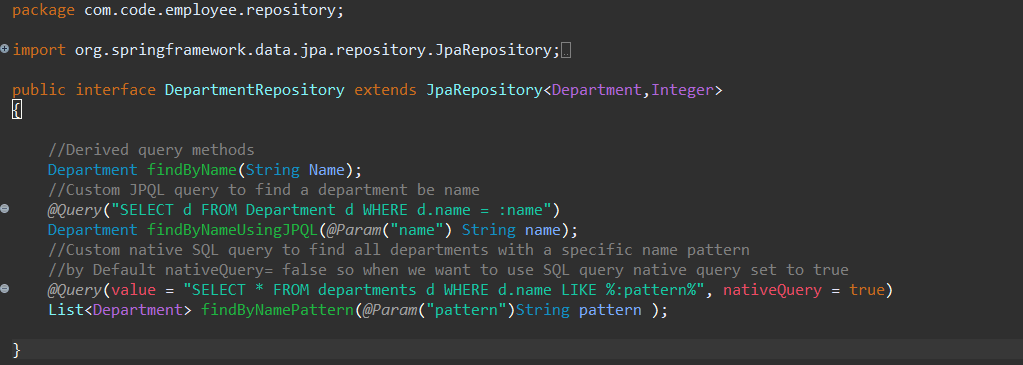
Enhance your repository to support custom queries.

**1. Defining Query Methods:**

* Using keywords in method names to create custom query methods and implemented the custom query methods using the **@Query** annotation in EmployeeRepository and DepartmentRepository.

In this exercise I modified the existing code by using keywords in method names to create custom query methods. Spring Data JPA provides a way to create custom query methods by defining method names according to specific keywords. These methods are automatically implemented by Spring Data JPA based on the method names. This method uses JPQL (Java Persistence Query Language) to explicitly define a query. JPQL operates on the entity objects rather than directly on the database tables, and later implemented the methods using the @Query annotation.





**2. Named Queries:**

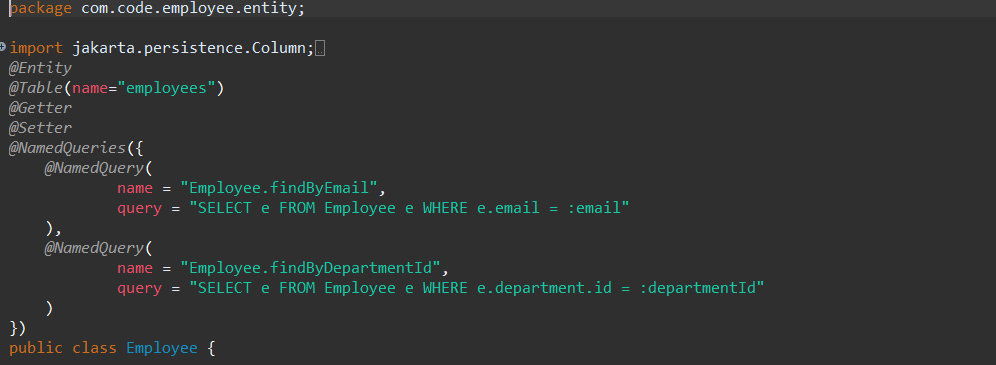
I defined and executed named queries with **@NamedQuery** and **@NamedQueries**.

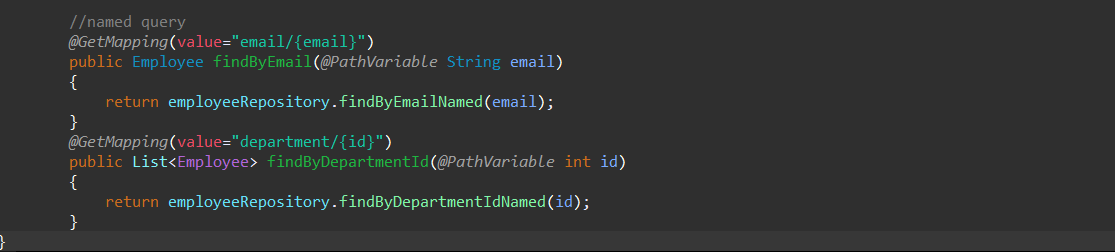
**@NamedQuery:**

* **The @NamedQuery annotation is used to define a query at the entity level, which can be executed in any repository method by referencing the query name.**

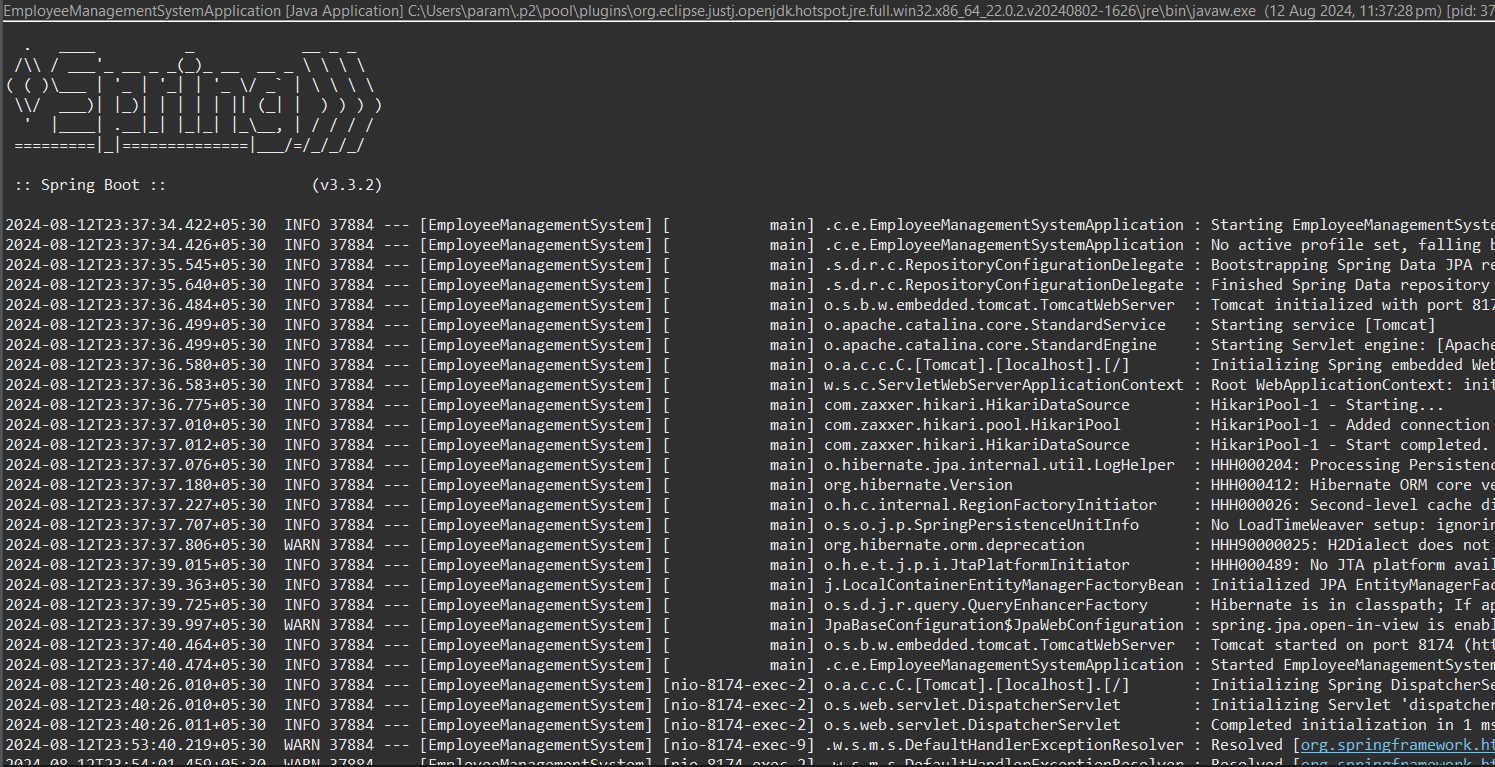
**@NamedQueries:**

* **Use @NamedQueries to group multiple @NamedQuery annotations together.**

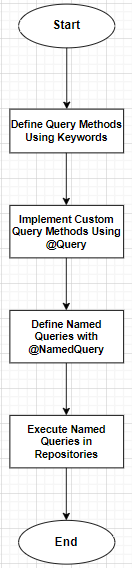
****



**Output:**

****

**Flowchart:**

****

1: "Start"

* + This is the starting point of the process.

2: "Define Query Methods Using Keywords"

* + This step involves defining query methods in the repository interfaces using keywords like findBy, Containing, etc.

3: "Implement Custom Query Methods Using @Query"

* + This step involves implementing custom queries using the @Query annotation for more complex queries.

4: "Define Named Queries with @NamedQuery"

* + This step involves defining named queries at the entity level using the @NamedQuery and @NamedQueries annotations.

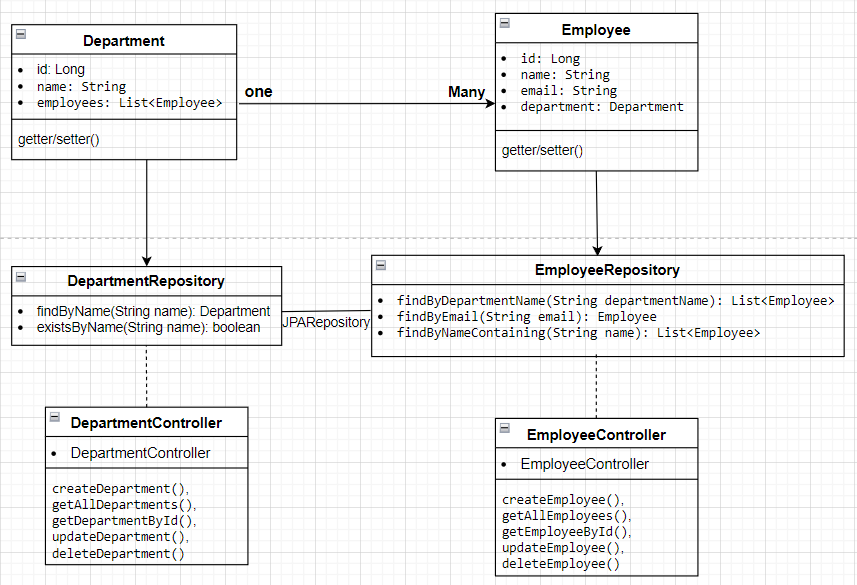
5: "Execute Named Queries in Repositories"

* + This step involves executing the named queries in repository methods by referencing their names.

6: "End"

* + This is the end of the process.

**Class Diagram:**

****

** Department Class:**

* **Attributes: Represents the id, name, and a list of Employee objects.**
* **Relationship: Has a one-to-many relationship with Employee.**

** Employee Class:**

* **Attributes: Represents the id, name, email, and a reference to the Department object.**
* **Relationship: Each Employee belongs to a single Department.**

** EmployeeRepository Interface:**

* **Methods:**
  + **findByDepartmentName(String departmentName): List<Employee>**
  + **findByEmail(String email): Employee**
  + **findByNameContaining(String name): List<Employee>**

** DepartmentRepository Interface:**

* **Methods:**
  + **findByName(String name): Department**
  + **existsByName(String name): Boolean**

** EmployeeController Class:**

* **Methods:**
* **createEmployee()**
* **getAllEmployees()**
* **getEmployeeById()**
* **updateEmployee()**
* **deleteEmployee().**

** DepartmentController Class:**

* **Methods**
* **createDepartment()**
* **getAllDepartments()**
* **getDepartmentById()**
* **updateDepartment()**
* **deleteDepartment()**

**Analysis:**

**1. Defining Query Methods Using Keywords**

Spring Data JPA provides a way to create custom query methods by defining method names according to specific keywords. These methods are automatically implemented by Spring Data JPA based on the method names.

* **Keywords**:
  + findBy: Fetches records based on the specified fields.
  + Containing: Checks if a field contains a specified substring.

**2. Implementing Custom Query Methods Using @Query Annotation**

If we need more control over the query, we can use the @Query annotation to write JPQL (Java Persistence Query Language) or native SQL queries.

* **@Query Annotation**:
  + Allows us to write custom JPQL or SQL queries.
  + Use @Param to bind method parameters to query parameters.

**3. Named Queries with @NamedQuery and @NamedQueries**

Named queries are predefined, static queries that can be defined at the entity level and referenced by name in the repository methods.

**@NamedQuery**:

* The @NamedQuery annotation is used to define a query at the entity level, which can be executed in any repository method by referencing the query name.

**@NamedQueries**:

* Use @NamedQueries to group multiple @NamedQuery annotations together.

**Relationship:**

* A Department can have many Employees (OneToMany relationship).
* An Employee belongs to one Department (ManyToOne relationship).
* Department to DepartmentRepository
* Employee to EmployeeRepository