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

Business Scenario:

Create repositories for Employee and Department entities to perform CRUD operations.

**1. Overview of Spring Data Repositories:**

* Learn the benefits of using Spring Data repositories.

 **Simplified Data Access Layer**:

**Declarative Data Access**: Spring Data repositories abstract the data access layer, allowing you to focus on the domain logic rather than boilerplate code for CRUD operations.

**Built-in Query Methods**: You can define query methods in your repository interfaces by simply declaring method names. Spring Data automatically implements these methods based on the method name.

 **Automatic Implementation**:

**Repository Abstraction**: Spring Data provides automatic implementation of repository interfaces, reducing the need for manual implementation of data access logic.

**Custom Implementations**: If needed, you can still provide custom implementations by extending repository interfaces and defining your own query methods.

 **Easy Integration with JPA, MongoDB, and Other Data Stores**:

**Unified API**: Spring Data provides a consistent programming model across different data stores (e.g., JPA, MongoDB, Redis, Neo4j), making it easier to switch or integrate multiple databases.

**Repository Definitions**: You can define repositories for different data stores with minimal configuration.

 **Powerful Query Capabilities**:

**Query Derivation**: Spring Data can derive queries from method names, simplifying the creation of queries for common use cases.

**Custom Queries**: You can also use the @Query annotation to define custom JPQL or SQL queries, giving you more control over complex queries.

 **Pagination and Sorting**:

**Built-in Support**: Repositories support pagination and sorting through PagingAndSortingRepository and JpaRepository, which makes it easy to handle large datasets and organize results.

 **Event Handling**:

**Lifecycle Events**: Spring Data repositories support event handling for CRUD operations, allowing you to react to entity lifecycle events like save, update, and delete.

 **Less Boilerplate Code**:

**Automatic Implementation**: The framework reduces the amount of boilerplate code required for common data access operations, leading to cleaner and more maintainable code.

 **Integration with Spring Ecosystem**:

**Seamless Integration**: Spring Data repositories integrate seamlessly with other Spring components (e.g., Spring Security, Spring Boot), leveraging Spring’s powerful features for transaction management, dependency injection, and configuration.

 **Testing Support**:

**Mock Repositories**: Spring Data makes it easier to test repositories by providing mock implementations and integration with testing frameworks, which can improve test coverage and reliability.

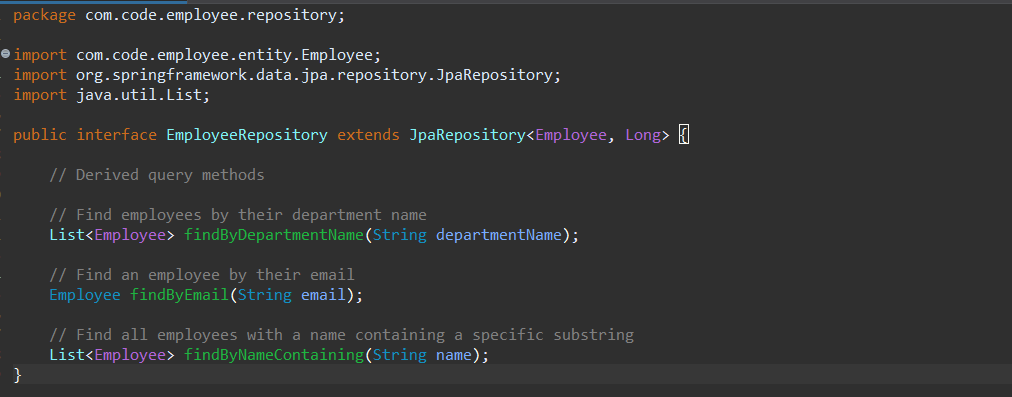
 **Community and Documentation**:

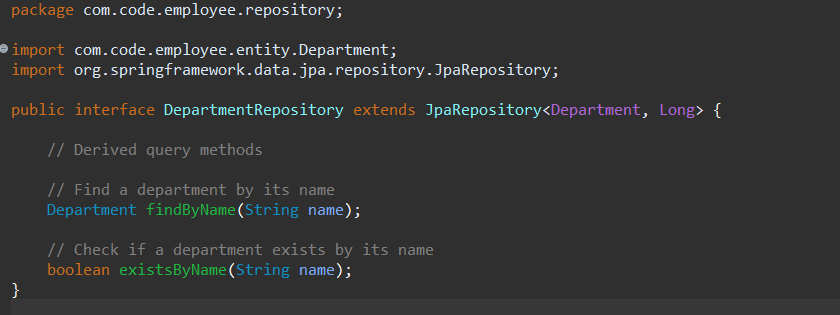
**Active Community**: Spring Data has a large, active community and extensive documentation, making it easier to find support and resources for development challenges.

In the given problem, I can now use these repository interfaces to interact with the Employee and Department entities in the service layer, leveraging the power of Spring Data JPA to perform various database operations without writing boilerplate code.

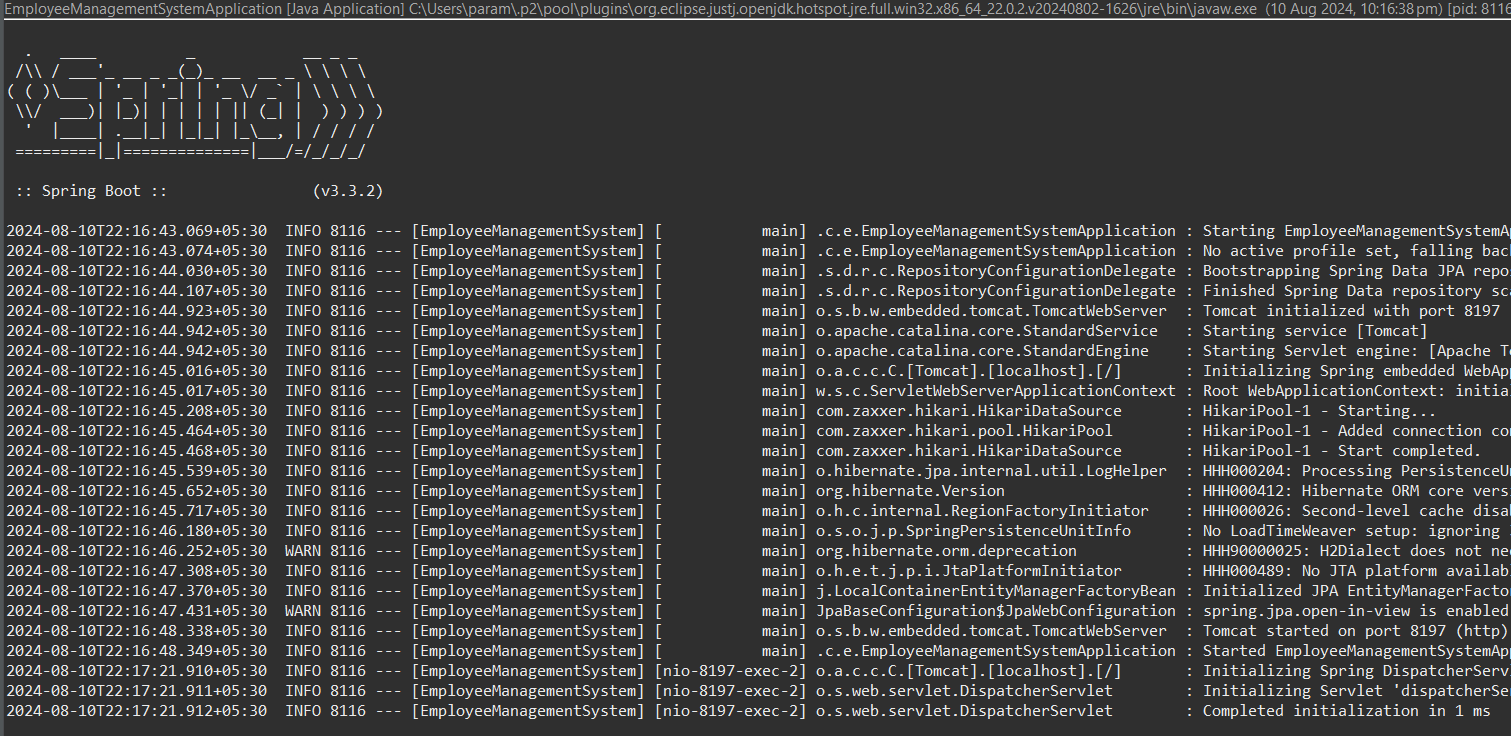
**2. Creating Repositories:**

I have created **EmployeeRepository** and **DepartmentRepository** interfaces that extends **JpaRepository**, and also defined derived query methods in these repositories

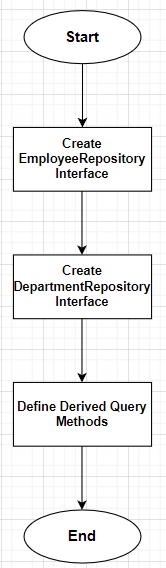




**Output:**

****

**Flowchart:**

****

**1. Start**

* This is the entry point of the flowchart. It signifies the beginning of the process where you start creating the repository interfaces.

**2. Create EmployeeRepository Interface**

* **Action**: Create the EmployeeRepository interface in your Spring Boot project.
* **Details**:
  + This interface should extend JpaRepository<Employee, Long>.
  + By extending JpaRepository, you inherit methods for basic CRUD operations without needing to write them manually.
* **Purpose**: This step sets up the repository for interacting with the Employee entity in the database.

**3. Create DepartmentRepository Interface**

* **Action**: Create the DepartmentRepository interface.
* **Details**:
  + This interface should extend JpaRepository<Department, Long>.
  + Similar to EmployeeRepository, this interface will allow you to perform CRUD operations on the Department entity.
* **Purpose**: This step sets up the repository for interacting with the Department entity in the database.

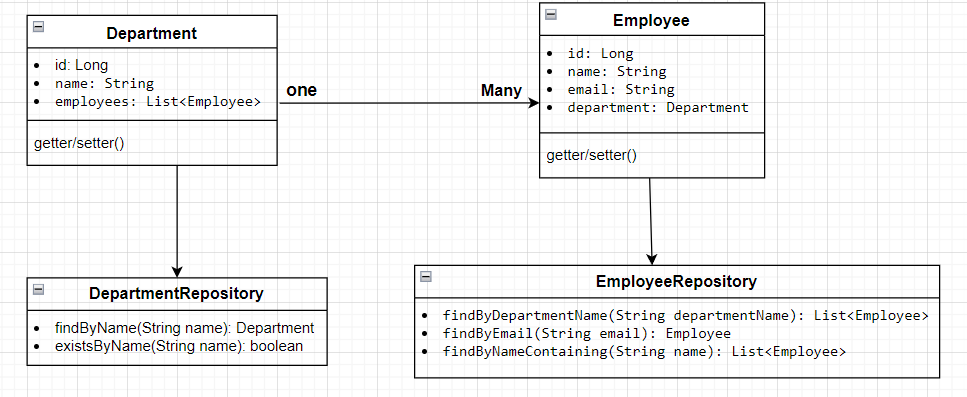
**4. Define Derived Query Methods**

* **Action**: Add specific query methods to the EmployeeRepository and DepartmentRepository interface.
* **Details**:
  + Derived query methods are automatically implemented by Spring Data JPA based on the method names.
* **Purpose**: These methods allow for custom queries that go beyond basic CRUD operations, enabling more specific data retrieval from the Employee table.

**5. End**

* This is the exit point of the flowchart, indicating that the process of creating and defining the repository interfaces is complete.

**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**

**Analysis:**

**1. Extending JpaRepository:**

* Both repositories extend JpaRepository, which provides generic CRUD operations (Create, Read, Update, Delete) and pagination capabilities.
* The JpaRepository<T, ID> interface requires two parameters:
  + T: The entity type (e.g., Employee, Department).
  + ID: The type of the primary key (e.g., Long).

**2. Defining Derived Query Methods:**

Derived query methods are automatically implemented by Spring Data JPA based on the method names. The methods are named according to specific conventions that allow Spring to derive the necessary SQL queries.

* **In EmployeeRepository:**
  + findByDepartmentName(String departmentName): Finds all employees in a department with a given name.
  + findByEmail(String email): Finds an employee by their email address.
  + findByNameContaining(String name): Finds employees whose names contain a specified substring.
* **In DepartmentRepository:**
  + findByName(String name): Finds a department by its name.
  + existsByName(String name): Checks if a department with a given name exists in the database.

**Relationship:**

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