Exercise 5: Defining Query Methods

Understanding the Requirements

We'll enhance the `EmployeeRepository` and `DepartmentRepository` with custom query methods to retrieve specific data based on various criteria.

Derived Query Methods

We can leverage Spring Data JPA's ability to create query methods based on method names.

Java

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

public interface EmployeeRepository extends JpaRepository<Employee,
Long> {
    List<Employee> findByDepartmentName(String   departmentName);
    List<Employee> findByNameStartingWith(String namePrefix);
    List<Employee> findBySalaryGreaterThan(double salary);
}
```

Custom Query Methods with @Query

For more complex queries, we can use the `@Query` annotation:

Java

```
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.data.jpa.repository.Query;
import   org.springframework.data.repository.query.Param;

public interface   EmployeeRepository extends JpaRepository<Employee,
Long> {
```

```
    @Query("SELECT e FROM Employee e WHERE e.department.name    =
:departmentName")
    List<Employee> getEmployeesByDepartmentName(@Param("departmentName")
String departmentName);
}
```

- The `@Query` annotation specifies the JPQL query to be executed.
- The `@Param` annotation binds the method parameter to the query parameter.

Named Queries

For reusable queries, we can define named queries:

Java

```
import javax.persistence.NamedQuery;

@Entity
@NamedQuery(name = "Employee.findByDepartment", query = "SELECT e FROM
Employee e WHERE e.department.name = :departmentName")
public class Employee {
    // ...
}
```

To execute a named query:

Java

```
import javax.persistence.EntityManager;
import javax.persistence.TypedQuery;

@Autowired
private EntityManager entityManager;

public List<Employee> getEmployeesByDepartmentName(String departmentName)
{
```

```
    TypedQuery<Employee> query =
entityManager.createNamedQuery("Employee.findByDepartment",
Employee.class);
    query.setParameter("departmentName", departmentName);
    return query.getResultList();
}
```

Key Points

- Derived query methods are convenient for simple queries based on method names.
- `@Query` annotation provides flexibility for complex queries.
- Named queries are useful for reusable queries.
- Choose the appropriate approach based on query complexity and reusability.