As vezes precisamos fazer ***querys dinâmicas***, e o **Spring Data JPA** tem uma **feature interessante para isso**, que são as **Specifications**.

A primeira coisa a ser feita é dar o poder de executar **Specifications** ao **repositório**, dá pra fazer isso estendendo **JpaSpecificationExecutor:**

```
@Repository
public interface FuncionarioRepository extends PagingAndSortingRepository<Funcionario, Long>,
        JpaSpecificationExecutor<Funcionario> {
```

Agora é necessário que **uma classe contenha a implementação** das ***querys dinâmicas*** baseada nos **atributos**. A ideia aqui é que o **Spring Data JPA** nos **poupa de todo a verbosidade de escrever uma query com Criteria API.**

Então ele já fornece um **Objeto do tipo Specification** com todos os **outros objetos** que são necessários para escrever uma query dinâmica, restando para nós apenas fazer a **implementação do atributo do jeito que quisermos**:

```
public class SpecificationFuncionario {

    public static Specification<Funcionario> nome(String nome){
        return ((root, query, criteriaBuilder) ->
            criteriaBuilder.like(root.get("nome"), pattern: "%"+nome+"%"));
    }

}
```

Para **executar essa query dinâmica** eu preciso apenas chamar os **métodos padrões** dos **repositories do SDJ** passando a **Specification.where** com as **especificações implementadas** *(se a implementação que você fizer já conter o WHERE então nem precisa chamar estaticamente):*

```
private void inicial(Scanner scanner){
    scanner.nextLine();
    System.out.println("Pesquise um nome");
    String nome = scanner.nextLine();

    funcionarioRepository.findAll(Specification.where(SpecificationFuncionario.nome(nome)));
}
```

A **facilidade em criar esses atributos dinâmicos** é tanta que eu posso simplesmente **fazer um monte** de **métodos de especificação** na minha **Specification:**

```java
public class SpecificationFuncionario {

    public static Specification<Funcionario> nome(String nome){
        return ((root, query, criteriaBuilder) ->
            criteriaBuilder.like(root.get("nome"), pattern: "%"+nome+"%"));
    }

    public static Specification<Funcionario> cpf(String cpf){
        return ((root, query, criteriaBuilder) ->
                criteriaBuilder.equal(root.get("cpf"), cpf));
    }

    public static Specification<Funcionario> salario(BigDecimal salario){
        return ((root, query, criteriaBuilder) ->
                criteriaBuilder.greaterThanOrEqualTo(root.get("salario"), salario));
    }

    public static Specification<Funcionario> unidadeTrabalho(String endereco){
        return ((root, query, criteriaBuilder) ->
            criteriaBuilder.like(root.get("unidadeTrabalho.endereco"), pattern: "%"+endereco+"%"));
    }

}
```

Considerando que agora **a consulta pode usar todas essas especificações ou NENHUMA**, eu posso implementar **checando se as informações passadas são nulas ou não**.

Depois posso chamar um método padrão do **repositorie SDJ** encadeando as **especificações**, o que **tiver valor nullo vai ser simplesmente ignorado** na consulta. Pode encadear com ***"and"*** ou ***"or"*** aí depende de como você quer fazer a querie:

```java
String nome = scanner.nextLine();
if(nome.isEmpty()){
    nome = null;
}

System.out.println("Informe o CPF:");
String cpf = scanner.nextLine();
if(cpf.isEmpty()){
    cpf = null;
}

List<Funcionario> funcionarios = funcionarioRepository.findAll(Specification.where(
        SpecificationFuncionario.nome(nome)
        .or(SpecificationFuncionario.cpf(cpf))
));

funcionarios.forEach(System.out::println);
```