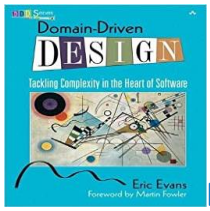
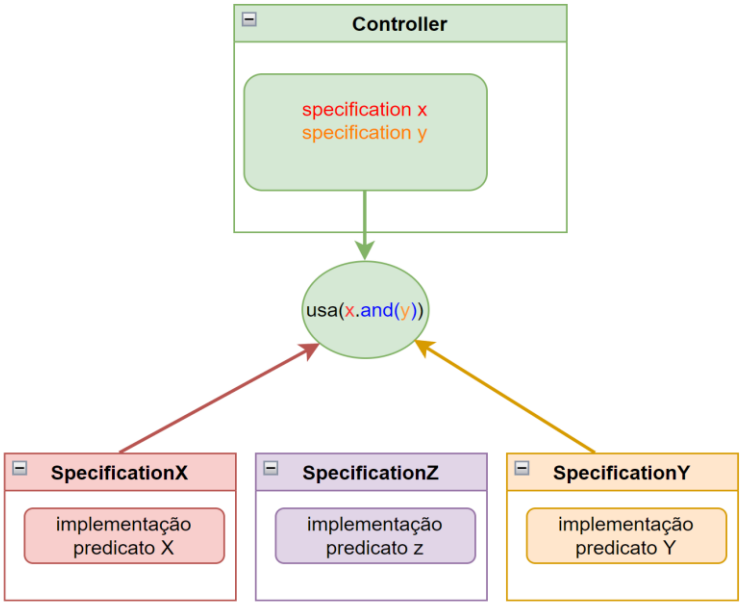


DDD Specifications – Spring Data JPA

Specifications é um padrão do DDD. Basicamente falando ele tem como objetivo criar objetos que podem ser combinados para chegar a um determinado fim. Specification é uma restrição no estado de algum outro Objeto.

Por exemplo, se eu tiver uma **Entidade chamada Restaurante** e eu quero fazer uma consulta onde a taxa de frete esteja grátis, eu posso criar uma specification disso, algo como **"RestauranteComFretegratisSpec",** essa specification poderia então ser combinada com outras especifications também relacionadas a restaurante, por exemplo uma **"RestauranteComNomeSemelhanteSpec",** no fim eu teria ***uma consulta com restaurantes que o frete esteja grátis e o nome é parecido com algum valor informado.***



Então os objetos de Specifications são usados como predicados.

==========

Especification no Spring Data JPA

O Spring data JPA tem classes de suporte a esse padrão, ele oferece uma interface Specification que tem um método "toPredicate" que precisa ser sobrescrito, na prática esse **método vai simplesmente servir para implementar o predicado que a specification se refere**.

Então ele já recebe um **Root** ***(entidade/objeto raiz da query)***, um **CriteriaQuery** ***(responsável pelo "where")*** e o próprio **CriteriaBuilder** ***(responsável pelas clausulas)***

```java
public class RestauranteComFreteGratisSpec implements Specification<Restaurante> {

    private static final long serialVersionUID = 1L;

    @Override
    public Predicate toPredicate(Root<Restaurante> root, CriteriaQuery<?> query, CriteriaBuilder builder) {
        return builder.equal(root.get("taxaFrete"), BigDecimal.ZERO);
    }
}
```

```java
@AllArgsConstructor
public class RestauranteComNomeSemelhanteSpec implements Specification<Restaurante> {

    private String nome;

    @Override
    public Predicate toPredicate(Root<Restaurante> root, CriteriaQuery<?> query, CriteriaBuilder builder) {
        return builder.like(root.get("nome"), pattern: "%"+nome+"%");
    }
}
```

Para que o repositório reconheça as nossas Specifications temos que extender a interface **JpaSpecificationExecutor<T>**, essa interface tem vários métodos que recebem alguma Specification:

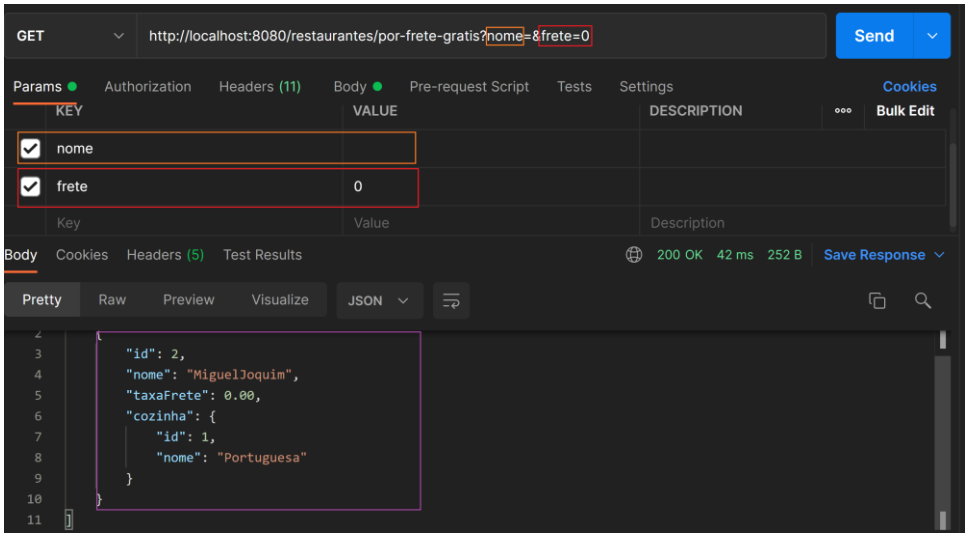
```java
@Repository
public interface RestauranteRepository extends JpaRepository<Restaurante, Long>, RestauranteRepositoryQueries,
        JpaSpecificationExecutor<Restaurante> {
```

Agora na implementação disso em algum endpoint, podemos executar o **método do repositório que recebe nossas** Specifications, dessa maneira encadeando uma specification a outra, obviamente dá pra acrescentar mais specifications ou tirar:

```java
@GetMapping("/por-frete-gratis")
public List<Restaurante> acharFreteGratis( String nome){

    var freteGratisSpec = new RestauranteComFreteGratisSpec();
    var nomeSemelhanteSpect = new RestauranteComNomeSemelhanteSpec(nome);

    return restauranteRepository.findAll(freteGratisSpec.and(nomeSemelhanteSpect));
}
```



==========

Factory de Specifications

É possível fazer a criação de predicados de Specification de forma bem simples com a ajuda do Java. Da pra criar uma fábrica com métodos ***statics*** que retornam Specifications, então passar o comportamento através de um lambda:

```java
public class RestauranteSpecs {

    public static Specification<Restaurante> comFreteGratis(){
        return (root, query, builder) -> builder.equal(root.get("taxaFrete"), BigDecimal.ZERO);
    }

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

Dessa maneira podemos reservar uma classe unicamente como uma Fábrica de specification, **simplificando muito o código usado em outros locais do programa além de reduzir a quantidade de classes sem perder a coesão:**

```java
@GetMapping("/por-frete-gratis")
public List<Restaurante> acharFreteGratis( String nome){
    return restauranteRepository.findAll(comFreteGratis().and(comNomeSemelhante(nome)));
}
```

==========

Referência Circular

****Ponto negativo**:** Usando o padrão de specification é nítido que a r**esponsabilidade de usar os filtros é nossa e não do repositório**, isso pode acabar gerando códigos duplicados e códigos duplicados podem zuar uma refatoração.

Então uma boa **alternativa é centralizar o método que utiliza specification no repositório**, para caso seja necessário altera-lo no futuro, saibamos exatamente onde fazer sem quebrar o projeto ou gerar algum bug.

Já que tenho uma interface de **repositório customizado** é nele que vou criar esse método:

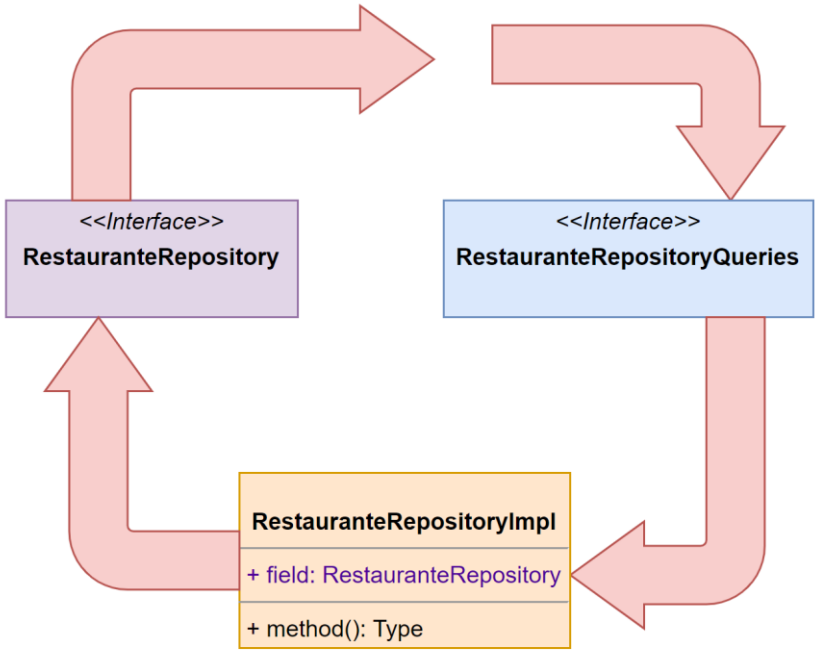
```java
public interface RestauranteRepositoryQueries {
    List<Restaurante> achar(String nome, BigDecimal taxa);

    List<Restaurante> acharComFreteGratis(String nome);
}
```

Agora preciso fazer a implementação:

```java
@Override
public List<Restaurante> acharComFreteGratis(String nome) {
    return restauranteRepository.findAll(comFreteGratis().and(comNomeSemelhante(nome)));
}
```

E nesse ponto tem algo muito importante a respeito do RestauranteRepository, o que acontece é que o RestauranteRepository estende o RestauranteRepositoryQueries, o RestauranteRepositoryQueries usa a RestauranteRepositoryImpl como implementação **QUE INJETA** o RestauranteRepository , dessa maneira criamos uma referência circular:



Quando o **Spring** inicializa ele vai instanciar todas as classes do projeto **(componentes/Beans)** e essas classes que fazem referência circular são problemáticas então o container não sobe.

Pra resolver isso é necessário usar a anotação **@Lazy**, dessa maneira o Spring só vai instanciar a classe quando ela realmente for ser usada:

```java
@Repository
public class RestauranteRepositoryImpl implements RestauranteRepositoryQueries {

    @PersistenceContext
    private EntityManager entityManager;

    @Autowired @Lazy
    private RestauranteRepository restauranteRepository;
```

Com o problema resolvido, basta usar o método customizado no controller:

```java
@GetMapping("/por-frete-gratis")
public List<Restaurante> acharFreteGratis( String nome){
    return restauranteRepository.acharComFreteGratis(nome);
}
```