02

Simplificando entidades com Embeddable

Transcrição

[00:00] Olá, pessoal! Nessa aula vamos conhecer alguns recursos a mais da JPA para complementar o nosso treinamento. Eu estou aqui com a nossa entidade "Cliente" e vou fazer uma pequena alteração nessa classe. Até então só temos três atributos aqui, o ID, o nome e o CPF.

[00:17] Mas, obviamente, essa classe provavelmente vai crescer e vai ter novos atributos, então vou precisar de outras informações do cliente: telefone, endereço, e-mail, enfim. No endereço em si, os dados do endereço, logradouro, rua, bairro, CEP, estado. Então essa classe vai começar a crescer e ficar com vários atributos.

[00:39] Talvez você já tenha se esbarrado com isso, uma classe que tem 30, 40, 60, 100 atributos. Uma maneira de você começar a organizar esse código é tentar agrupar esses atributos que são comuns e extrair para classes. Por exemplo, ao invés de deixar um nome e CPF aqui na classe Cliente, e outros dados pessoais, eu poderia criar uma classe chamada dados pessoais e lá dentro eu coloco os atributos que são de dados pessoais.

[01:06] Os atributos do endereço, rua, logradouro, CEP, bairro, UF, ao invés de deixar tudo solto na classe Cliente, eu podia ter uma classe endereço e esses atributos estarem lá dentro. Então o "Cliente", ele será composto desses outros objetos. Vamos fazer essa mudança e ver como isso vai funcionar perante à JPA.

[01:26] Então eu vou criar uma classe aqui no pacote do "modelo", "Ctrl + N", "class", e eu vou chamar essa classe de "DadosPessoais". Na classe "Cliente", eu

vou extrair o nome e o CPF, vou dar um "Ctrl + X", vou jogar para os "DadosPessoais". Extrai para cá. Eu preciso extrair também os *getters* e *setters*, agora os *getters* e *setters* vão ficar em "DadosPessoais", vou dar um "Ctrl +X", vou jogar para "DadosPessoais".

[01:56] Em "DadosPessoais" eu não vou colocar os métodos *setters*, só os *getters*, vou gerar um construtor para receber logo esses dois parâmetros. Construtor, que recebe o nome e o CPF. Ok, está aqui a nossa classe "DadosPessoais". O "Cliente" agora, ele vai ter o construtor que recebe o `Cliente(String nome, String cpf)`, só que ele não atribui mais a atributos da classe.

[02:22] Ao invés de ter os atributos soltos aqui, eu vou ter um atributo do tipo `privateDadosPessoais dadosPessoais;`. Vou deixar dessa maneira.

[02:34] Quando eu instancio um Cliente, eu passo o nome, eu passo o CPF e eu atribuo esse parâmetro, `this.dadosPessoais = new DadosPessoais (nome, cpf);` e passo o nome e o CPF que chegou no construtor.

[02:49] Então todo o acesso ao nome e ao CPF, e às outras informações de dados pessoais, ficam encapsuladas nessa classe de "DadosPessoais". Em "Cliente", eu só vou precisar ter um *get* `DadosPessoais`. Poderia ter um método *get* nome aqui, que delegava, que o *return* seria `dadosPessoais.getNome`, facilitaria também de fora, mas para simplificar aqui, eu vou deixar sem.

[03:11] Está aqui a nossa classe. Porém, temos um problema. Como a JPA vai mapear esse atributo `DadosPessoais`? Porque agora, esse atributo, `DadosPessoais`, não é mais um tipo primitivo, não é uma classe do Java, não é *string*, não é *int*, não é *big decimal*, não é *local date*. E não é uma entidade, eu não tenho uma tabela "DadosPessoais", eu tenho a tabela "Cliente", nome e CPF são colunas da tabela "Cliente".

[03:37] Eu não vou colocar `@Entity` aqui, não vai ter um `@Id`. Eu só quero separar as classes. Só que precisamos dizer isso para a JPA, precisamos falar:

JPA, entra nessa classe "DadosPessoais" e considera que esses atributos, embora eles estejam em "DadosPessoais", considere que eles pertencem a mim, a classe Cliente", considere que são colunas da tabela de "Clientes".

[04:00] Então tem como você organizar o código e indicar isso para a JPA. Você só vai precisar fazer o seguinte, em cima da classe `DadosPessoais`, não vai ter um `@Entity`, mas vai ter uma outra anotação, que é o `@Embeddable`, essa anotação do "javax persistence". Eu estou falando: JPA, essa classe é embutível, eu consigo embuti-la dentro de uma entidade. Você precisa indicar isso para a JPA.

[04:25] E na classe `Cliente`, em cima do atributo `DadosPessoais`, `@Embedded`: embute para mim os atributos dessa classe aqui, do meu atributo "DadosPessoais". Esse é um recurso bem interessante da JPA, os atributos *embeddable*, você consegue organizar o código, quebrar em classes separadas, ao invés de encher uma classe com 300 milhões de atributos.

[04:52] De resto, tudo continua igual, não muda nada. Obviamente, como eu fiz essa mudança, vai dar pau. Por exemplo, deu pau na classe "PerformanceConsultas" por conta do `getNome`, do `getCpf`, que mudou. Agora o "Cliente", ele não tem mais um `getNome`, ele tem um `getDadosPessoais.getNome`.

[05:12] Se você quiser evitar esses erros de compilação, provavelmente se você fizer essa mudança, vão estourar uma série de erros de compilação no seu projeto. Vou fazer aquele método *delegate*. Eu disse que não ia fazer, mas vou fazer aqui só para mostrar de exemplo. Eu posso ter aqui, no "Cliente", um método `public String getNome()`, porque já tinha um monte de classes chamando o `getNome` do "Cliente".

[05:35] Ao invés de ser um `return this.nome`, vai ser um `return this.dadosPessoais.getNome();`. Pronto, olha lá, voltou a compilar. Então isso é um método *delegate*, ele está delegando essa chamada do `getNome` para o

atributo `dadosPessoais`. Posso fazer a mesma coisa para o CPF. `public String getCpf()`, `return this.dadosPessoais.getCpf();`. Pronto.

[06:04] Não tinha nenhum lugar chamando `getCpf`, mas se tiver não vai dar mais problema. Vamos rodar essa classe "PerfomanceConsultas"? Só para ver se tudo continua funcionando, se ele vai criar as tabelas corretamente. Deu erro.

[06:15] Vamos dar uma olhada. O construtor *default*, esqueci. Na classe *embeddable*, eu gerei esse construtor, mas tem que ter um construtor *default* por causa da JPA, ela exige um construtor *default*. Vamos rodar aqui novamente.

[06:30] Rodou, carregou aqui o registro. Vamos ver os *create tables*, estão lá para cima. *Create table* clientes.

[06:39] O ID, cpf, nome. Ela continua trazendo os campos CPF, o campo nome, embora esteja em outra classe, ela fez o *embeddable*, o *embedded*, ela embutiu esses atributos como se pertencesse à classe "Cliente". Esse é um recurso bem interessante, talvez você se esbarre com ele.

[06:59] Se não conhecia, passe a utilizar. Ele é bem útil quando você tem principalmente telefone e endereço, quando você não tem uma tabela de endereço ou de telefone, está tudo misturado em uma mesma tabela. Como está tudo na mesma tabela, o natural é encher de atributos a classe que está mapeando aquela tabela.

[07:18] Mas você pode quebrar em classes pequenas, para organizar o código, deixar mais coeso e mais simples de dar manutenção, usando esses recursos de *embeddable* e *embedded* da JPA. Espero que vocês tenham gostado. No próximo vídeo continuaremos vendo mais recursos da JPA. Um abraço e até lá.