02

Criando o modelo Produto

Transcrição

[00:00] Fala, aluno. Tudo bom? Dando continuidade ao nosso curso JDBC, gostaria de voltar na nossa classe `TestaInsercao` e verificarmos aqui uma questão. Hoje, na minha base de dados, eu tenho uma tabela chamada produto. Essa tabela produto, ela tem os seus atributos: o ID, nome e descrição.

[00:25] Do lado do Java, nós viemos tratando esses atributos apenas como strings, ou seja, eu não tenho uma representatividade de produto igual nós temos do lado do banco de dados. Então, por exemplo, quando eu quero fazer o *bind* dessas interrogações do `VALUES (?, ?)`, que eu quero *setar* de fato os valores dessas interrogações, eu estou recebendo nesse método `adicionarVariavel` uma `(String nome, String descricao, PreparedStatement stm)`.

[00:53] E estou *setando* essas strings no meu `setString` do meu `PreparedStatement`. Só que isso não fica muito legal, eu queria de fato ter a mesma representatividade do lado do Java que eu tenho no banco de dados. Então eu queria ter um modelo de produto. Tem uma forma de fazer isso no Java e nós vamos criar agora dessa forma.

[01:25] Dentro do "New Java Class", do "package" "br.com.alura.jdbc.modelo", eu vou criar uma classe chamada "Produto". Essa `public class Produto` vai conter exatamente o que eu tenho na minha tabela: `private Integer id;`, eu vou ter `private String nome;` e uma `private String descricao;`. Colocamos todos os atributos como privados por conta do encapsulamento da nossa classe.

[02:07] Por enquanto a classe vai ficar dessa forma porque eu não quero gerar métodos e nem nada desnecessário nesse momento. Agora eu quero fazer um teste inserindo um produto com essa classe, com a representatividade dessa classe `Produto` e não apenas strings soltas no código. Então eu vou criar mais uma classe e ela vai se chamar `TestaInsercaoComProduto`, que nós vamos utilizar o `Produto` para inserir um produto.

[02:41] Eu já vou mandar gerar o `public static void main`, para não precisarmos ficar gerando ele. Então, primeira coisa: vamos instanciar o nosso produto. E o nosso produto vai ser uma cômoda agora no banco de dados. Vou dar um `Produto comoda = new Produto("", "");`. Só que eu já quero instanciar o meu produto passando o nome dele e a descrição, vai ser `Produto comoda = new Produto("Cômoda" , "Cômoda Vertical");`.

[03:15] Ele está reclamando, falando que eu não tenho ainda esse construtor que recebe duas strings. Vamos na nossa classe `Produto` então, "Ctrl + 3" e eu vou na opção "Generate Constructor using Fields - Choose fields to initialize and constructor from superclass to call". Então vamos criar um construtor que tenha os atributos. Não vou selecionar o ID, só o "nome" e a "descricao". Mandou gerar.

[03:44] Deu um espaço aqui no código, para ficar organizado. Agora na nossa `TestaInsercaoComProduto` está tudo certo. Pronto. Agora para começarmos a conversar com o banco de dados, eu preciso perguntar para o meu Pool de conexões, se tem uma conexão disponível para nós.

[04:05] Então eu chamo o `try()`, e estou recuperando uma connection. Vou perguntar para a nossa Connection factory se tem uma conexão disponível no Pool de conexões. Então `try(Connection connection = new ConnectionFactory().recuperarConexao())`. Vamos adicionar o `throws SQLException`, porque senão ele vai reclamar.

[04:33] Estamos querendo testar o `TestaInsercaoComProduto`, então o nosso comando vai ser o Insert. Então eu vou escrever aqui `String sql = "INSERT INTO PRODUTO (NOME, DESCRICAO) VALUES (?, ?)";`, o `VALUES` com dois atributos que ainda vão receber o seu conteúdo, o nome e a descrição do produto.

[05:05] Agora eu vou preparar o Statement, então eu vou pegar `try(PreparedStatement pstm = connection.preparedStatement(sql))`, passo o `sql`, só que eu quero, no momento da inserção, recuperar a nossa chave gerada, então vai ser `try(PreparedStatement pstm = connection.preparedStatement(sql, Statement.RETURN_GENERATED_KEYS))`.

[05:31] Agora nós vamos *setar* os atributos. São duas strings, então `pstm.setString(1, x);` no primeiro parâmetro, e o nome vai ser o `pstm.setString(1, comoda.getNome());`. Obviamente esse parâmetro não existe ainda porque nós não criamos o método `.getNome()` na nossa classe `Produto`. E também vamos ter o mesmo problema com o segundo parâmetro, que é `pstm.setString(2, comoda.getDescricao);`.

[06:05] Para o `TestaInsercaoComProduto` compilar, nós temos que ir na nossa classe `Produto` e vamos mandar criar. "Ctrl + 3" novamente. Vou mandar um "Generate Getters and Setters - Generate Getter and Setter methods for type's fields". Expando o "descricao", só vou usar o "getDescricao()". Expando o "nome" e só vou usar o "getNome()".

[06:22] A ideia, na nossa classe `Produto` é criarmos só os métodos que vão ser utilizados mesmo, para não ficar uma classe com getters e setters desnecessários. E agora vou mandar executar o `TestaInsercaoComProduto` com o `pstm.execute();`, bem padrão, como já fizemos isso em outras oportunidades.

[06:45] Agora eu quero recuperar a chave gerada. Nós vimos que para recuperar a chave gerada, nós temos o `try(ResultSet rst = )` e, para pegarmos essa chave, vai ser `try(ResultSet rst = pstm.getGeneratedKeys())`

Enquanto eu tiver um próximo, eu vou *setar* então no meu `Produto` o ID. Eu vou pegar `while(rst.next()){ , comoda.setId(rst.getInt(1)); ,` que no primeiro index é o ID.

[07:27] Obviamente não vai compilar também, porque nós não temos o `setId` em `Produto` . Então deixa eu dar um "Ctrl + 3" de novo , dei um "Ctrl + F3" sem querer. "Ctrl + 3", "Generate Getters and Setters - Generate Getter and Setter methods for type's fields", expando o ID, "setId(Integer)".

[07:49] Vamos só tirar os espaços que ele criou aqui, desnecessários. Também quero tirar esses dois espaços. Agora pegou aqui, a nossa classe `TestaInsercaoComProduto` , ela já não está mais com nenhum erro, está compilando perfeitamente. Só que agora eu quero fazer o seguinte, eu quero mostrar qual foi o produto criado.

[08:16] Então para isso, em `TestaInsercaoComProduto` , eu vou dar um `System.out.println(comoda);` . Se eu mandar executar dessa forma, ele vai me mostrar o valor da variável na memória, então ele vai me trazer aquele número esquisito e eu não quero isso. Mas também eu não quero sair escrevendo getters e setters desnecessários, só para mostrar no nosso console o produto de forma bonita.

[08:45] Então, para isso, eu tenho uma solução: em `Produto` , eu vou sobrescrever o `public String ToString()` e para esse `return super.toString();` eu troco para `return String.format();` e vou falar `("O produto criado foi: %d, %s, %s", );` . Aqui eu estou falando que o primeiro vai ser substituído pelo ID, o segundo pelo nome e o terceiro pela descrição. Já vamos ver o resultado agora.

[09:32] Então vou passar `return String.format("O produto criado foi: %d, %s, %s", this.id, this.nome, this.descricao);` . Se mandarmos testar agora a nossa classe `TestaInsercaoComProduto` , ele vai *logar* as informações no nosso

Pool de conexões, que nós vimos anteriormente e vai mostrar que o produto criado foi o de ID 110, a nossa cômoda, cômoda vertical.

[10:08] Agora nós temos, de fato, a representatividade de um produto do lado do Java também, só que agora resolvemos um problema, mas eu queria chamar a atenção de vocês para uma outra questão: toda vez que eu vou escrever as classes, os meus main, eu tenho que repetir o `try`, com o `recuperarConnection`, tenho que escrever o `String sql`, tenho que preparar um Statement.

[10:41] Enfim, eu tenho que fazer todo esse passo, seja para inserir, seja para listar. Então fica um alerta aqui. Nós já vimos sobre repetir código em várias classes, isso é um exemplo disso. Como nós podemos melhorar esse código? Talvez tenha uma forma de extrair isso para um método? Enfim, vou deixar essa questão com vocês, porque a nossa aula de agora fica por aqui.

[11:16] Espero que vocês tenham gostado agora de trabalhar com o `Produto`, que é de fato a representatividade de um produto, então agora acho que fica até mais fácil de ler o código, vai ficar mais organizado. Mas vamos deixar essa indagação e nas próximas aulas nós vamos resolver essa questão. Então, espero que você tenha gostado e vejo você no próximo vídeo.