03

DAO com inserção do produto

Transcrição

[00:00] Fala, aluno. Tudo bom? Anteriormente nós vimos como utilizar uma classe de modelo para fazer persistência dos nossos dados. Só que ao realizar os testes, nós encontramos um outro problema: nós vimos que toda vez que nós precisamos inserir ou buscar informações no nosso banco de dados, nas nossas classes de teste nós estamos repetindo código.

[00:27] Então se eu quero inserir um produto, eu pego e repito uma string com a query, preparo o Statement, executo o meu SQL, enfim, fazemos uma série de repetições em várias classes. A ideia é melhorar isso. Nós já vimos que não é nunca aceitável sairmos repetindo código na nossa aplicação. Então qual seria um ponto interessante?

[01:06] Talvez, se eu estou inserindo um produto, se eu estou trabalhando com um produto, eu poderia ter uma classe onde quando eu instanciasse essa classe, eu tivesse um método salvar produto, porque eu só precisaria passar o produto, que vai ser inserido, passar uma conexão para esse método e ele faria o trabalho para mim. Então eu deixaria uma classe mais específica para as operações de banco de produto.

[01:43] Então, para testarmos isso, eu vou criar uma classe e eu vou chamar ela de "PersistenciaProduto". Essa classe, eu quero que ela tenha um construtor e que ela receba uma Connection, então `public PersistenciaProduto(Connection connection)`. Eu vou ter também um atributo, o `private Connection connection`.

[02:22] Vou pegar então `this.connection = connection;` e, como falado, eu quero um método `public void salvarProduto()`. Esse método, ele vai receber um produto, então `public void salvarProduto(Produto produto)`. Eu vou mandar importar esse primeiro `Produto`. E qual seria a lógica do meu `salvarProduto`?

[02:53] Seria essa lógica que nós utilizamos para fazer o teste na classe `TestaInsercaoComProduto`. Então toda essa `String sql = "INSERT INTO PRODUTO (NOME, DESCRICAO) VALUES (?, ?)"` eu posso extrair da classe `TestaInsercaoComProduto` e posso jogar para dentro do `PersistenciaProduto`.

[03:14] Só que antes nós trabalhávamos com um produto específico, agora não, agora eu quero receber qualquer produto, então substitui o `pstm.setString(1, comoda.getNome());` por `pstm.setString(1, produto.getNome());` e o `pstm.setString(2, comoda.getDescricao());` por `pstm.setString(2, produto.getDescricao());`.

[03:25] E o `comoda.setId(rst.getInt(1));` por `produto.setId(rst.getInt(1));`. Esse produto vai ser adicionado à nossa base de dados. O que ele está reclamando aqui? Vou adicionar o `throws SQLException`. Nosso código parou de reclamar. Aqui dentro de `try(Connection connection = newConnectionFactory().recuperarConexao())`, eu só vou precisar dar um `new PersistenciaProduto(Connection).salvarProduto(produto);`.

[03:56] No nosso caso, eu vou passar `new PersistenciaProduto(Connection), salvarProduto(comoda);`. Veja a diferença já no nosso código. Eu tenho uma classe especializada em trabalhar com a persistência das informações de produto e nas minhas classes de teste, ela fica mais sucinta, bem mais fácil de ler, de dar manutenção. Só que vamos supor que nessa situação, após eu salvar o produto, eu queira listar esse produto.

[04:36] Então eu não vou poder mais chamar ele dessa forma, com `new PersistenciaProduto(Connection), salvarProduto(comoda);` - quer dizer, vou

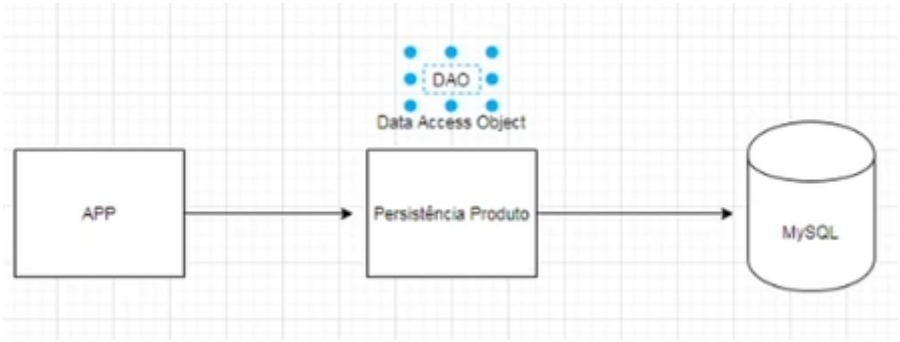
poder, mas vai ser mais fácil então eu instanciar a minha `PresistenciaProduto persistenciaProduto = new PersistenciaProduto(connection);` , passando a minha `connection` para ser o meu construtor.

[04:59] E, no lugar de `new PersistenciaProduto(Connection), salvarProduto(comoda);` , eu uso só a minha referência, a `persistenciaProduto.salvarProduto(comoda);` . Então, se depois de salvar eu precisar listar, eu vou criar aqui o `//persistenciaProduto.listar();` . Esse comando devolve uma `Lista = //persistenciaProduto.listar();` . Enfim, esse comando não existe ainda, só estamos pensando como ficaria a nossa situação.

[05:33] Vamos ver essa classe `PersistenciaProduto` , o que ela está fazendo. No desenho - vou fazer um desenho aqui. Eu tenho o nosso banco de dados, ele é o nosso MySQL. Eu tenho a nossa aplicação. Eu tenho agora um item que se chama Persistência Produto. O que ele faz? Ele acessa os dados do nosso objeto.

[06:30] Então eu tenho um objeto produto e quem está fazendo acesso desse objeto no nosso banco de dados é essa classe Persistência Produto. Então, a minha aplicação chama a Persistência Produto, a Persistência Produto vai no MySQL e nos traz as informações. Essa Persistência Produto, ela é um Data Access Object.

[07:08] O Data Access Object é quem trabalha com as informações do nosso objeto no banco de dados. Então ele é uma DAO. Essa Persistência Produto, se ela é um Data Access Object, não faz muito sentido eu trabalhar com o nome `PersistenciaProduto` . Eu posso utilizar o DAO, produto DAO, porque seu usar Produto Data Access Object, então vai ficar muito grande o nosso nome.



[07:50] Então eu vou fazer um "Refactor > Rename" em `PersistenciaProduto` e vou utilizar "ProdutoDAO". Vou clicar em "Finish". Eu já tenho um pacote criado, vocês podem criar o de vocês, mas para ficar mais organizado, eu vou mandar o "Produto DAO", vou dar um "Refactor > Move" para esse pacote "br.com.alura.jdbc.dao". Perfeito.

[08:21] Então agora, com esse `ProdutoDAO`, tudo o que é referente a acesso ao banco de dados para trabalhar com as informações do meu objeto vão ficar nessa classe. Então, se eu quiser alterar uma informação de produto, que está no meu banco de dados, eu vou criar o método dentro de `ProdutoDAO`.

[08:50] A vantagem é que quando eu tenho um `ProdutoDAO`, eu não preciso mais especializar aqui os meus métodos para informar que eles são um produto, porque ele já está dentro de uma classe que só trabalha com produto. Então, `TestaInsercaoComProduto`, eu vou mudar `persistenciaProduto.salvarProduto(comoda);`, eu vou falar `produtoDao.salvar(comoda);`.

[09:11] Vai ficar bem bacana, porque se eu mudar de `ProdutoDAO persistenciaProduto = new ProdutoDAO(connection);` para `ProdutoDAO produtoDao = new ProdutoDAO(connection);`, quando eu for ler as minhas chamadas ao método, eu vou ver `produtoDao`, chama o seu método `.salvar();`. Então eu já estou sabendo aqui que o que o método `.salvar();`, ele é de produto, ele vai salvar um produto.

[09:43] É importante ressaltar que esse DAO, não sou eu quem estou inventando, ele é um *pattern*, então ele é o padrão da linguagem, que tudo qu

é sobre acesso a objeto, seja banco de dados, seja algo externo à sua aplicação, geralmente ele vai ter o sufixo DAO. É obrigatório? Não.

[10:13] Mas é aquilo: tudo o que é convenção, tudo o que é padrão, mesmo que não seja obrigatório, eu recomendo fortemente utilizar, porque vai ser assim na sua aplicação, vai ser na minha aplicação e vai ser nas aplicações corporativas espalhados pelo mundo todo. Então a ideia da DAO é exatamente isso: criamos um local onde eu vou trabalhar com a parte de persistência da minha aplicação, ela fica toda centralizada nessa DAO.

[10:46] A DAO tem por objetivo conversar com o nosso banco de dados, então onde eu vou salvar produto, eu posso chamar de qualquer main desses que eu já criei. Só que eu sempre vou instanciar a minha DAO de produto e vou passar o meu produto para ela. Dentro da DAO, esse método, o `.salvar()`, o `.listar();`, eles vão se virar, mas em um único lugar vamos ter centralizado todos eles e eles que vão executar essas querys com o nosso banco de dados.

[11:26] Então a nossa aplicação agora já começa a ficar com essa nossa camada de dados de persistência bem mais organizada, o nosso código tende a ficar mais organizado também. Agora, quando eu for fazer os próximos testes, eu não preciso mais sair criando de novo todos esses SQLs, porque ele vai estar centralizado já no mesmo lugar.

[11:54] E agora já fica utilizando a DAO, utilizando um Pool de conexões, as interfaces JDB, Datasource, já começamos a ter uma aplicação do mundo real. Quando você for trabalhar em aplicações corporativas, elas vão ter todos esses conceitos que estamos aplicando aqui e isso é muito legal. Então, aluno, eu espero que você tenha gostado e até a próxima aula.