01

Analisando nossas variáveis

Transcrição

Realizamos a compra por pessoa física anteriormente. Agora, partiremos para a compra da pessoa jurídica, ou seja, de empresas. Em vez de comprarem produtos de informática, os usuários comprarão de perfumaria, outro segmento da nossa loja. Na hora da criação de categorias, foi definida outra *string* (" `Perfumaria` "), dentro da categoria `perfumaria` , sendo objeto da classe `Categoria` .

Para se criar os produtos, usaremos o método `criarProdutos()` , tal como ocorria com produtos de informática, mas há diferenças: o construtor foi quantificado, após definição de descrição, preço e categoria referentes, sendo criada uma nova lista de produtos.

CompraPessoaJuridica.java | CarrinhoCompra.java

```
11
12 public class CompraPessoaJuridica {
13
14     public static void main(String[] args) {
15         Categoria perfumaria = new Categoria("Perfumaria");
16         List<Produto> produtos = criarProdutos(perfumaria);
17         CarrinhoCompra carrinho = new CarrinhoCompra(FormaPagamento.CARTAO_CREDITO);
18         BigDecimal valorFinalDaCompra = carrinho.calcularPrecoFinal(produtos);
19         System.out.println("Valor da compra: " + valorFinalDaCompra);
20     }
21
22
23
24     private static List<Produto> criarProdutos(Categoria categoria) {
25
26         //5 Mulheres na equipe
27         Produto _212 = new Produto("212 Sexy Feminino",new BigDecimal(("179.99")), categoria, 5);
28
29         //2 Diretoras na equipe
30         Produto chanel = new Produto("Chanel N°5 ", new BigDecimal("665"), categoria, 2);
31
32
33         //7 Homens da equipe
34         Produto sauvage = new Produto("Sauvage", new BigDecimal("375"), categoria, 7);
35
36         return  Arrays.asList(_212, chanel, sauvage);
37     }
38
39 }
40
```

Foi criado também o objeto `carrinho` mas, ao passar pelo construtor `CarrinhoCompra` , passa-se a forma de pagamento `CARTAO_CREDITO` , sobre a qual podemos clicar mantendo o "Ctrl" apertado ("Cmd" no Mac). Cairemos no `ENUM` e verificamos que, como já visto anteriormente, o cartão de crédito dá `1.5%` de desconto. O boleto, por sua vez, dá `9%` , e o cartão de débito, `5%` .

Voltando-se à aba `CompraPessoaJuridica.java` , há o método `calcularPrecoFinal` . Para fomentarmos as vendas para as pessoas jurídicas, há um desconto do tipo "leve 3 e pague 2". Mantendo o mouse pressionado sobre este método com o "Ctrl" ou "Cmd", verificaremos maiores detalhes sobre ele. Vemos que há um método novo chamado `calcularPromocao` , que verifica se há `3` ou mais itens similares na compra, o que valida a promoção.

Vamos rodar a aplicação para conferirmos seu funcionamento e consequente valor da compra. A partir do método `main` , clicaremos com o botão direito do mouse selecionando "Run as > Java Application". A mensagem retornada pelo console nos mostra "Valor da compra: `4227.13575` ".

O problema é que quando a empresa que está realizando a compra fez os cálculos, viu que o valor final dado pela nossa aplicação não é o mesmo a que eles chegaram (`3852.13`). O valor da compra sem promoção seria `4782.12` (esta quantia não é relevante, servindo apenas como curiosidade).

Temos um bug a ser resolvido! Na aba `CompraPessoaJuridica.java`, vamos clicar em `calcularPrecoFinal` para abri-lo na aba `CarrinhoCompra.java`. Inserimos um *breakpoint* ao *BigDecimal* inicial, rodando a aplicação novamente, desta vez em modo *Debug*. Voltando ao método principal, clicaremos com o botão direito do mouse e em "Debug As > Java Application", e trocaremos a perspectiva (de Java para Debug).

Durante todo este processo, verificaremos o valor total da compra, quais valores estão sendo utilizados para se chegar nele, como as contas são feitas, entre outros. Selecionaremos `total` em `BigDecimal total = new BigDecimal(0);`, clicando com o lado direito do mouse e apertando a opção "*Watch*", conseguiremos observar uma variável específica. Feito isto, uma nova aba é mostrada na parte superior direita, denominada "*Expressions*".

Esta aba nos mostrará exatamente o que acontece nesta variável, acompanhando seus valores. No momento, nenhuma foi criada, pois ainda não passamos pela primeira linha. Neste instante, o que nos importa é o valor total, conforme a aplicação for rodando.

Utilizaremos o atalho do teclado `F6` ("*Step Over*") para prosseguirmos à próxima linha. É criada a variável `total`, com valor `0`. Vamos ver como ela está sendo alterada conforme o cálculo. Seguiremos linha a linha por meio do `F6`, e vimos preço, valor da porcentagem de desconto, valor do produto com desconto; logo em seguida, verificaremos se o valor é maior do que `R$700`.

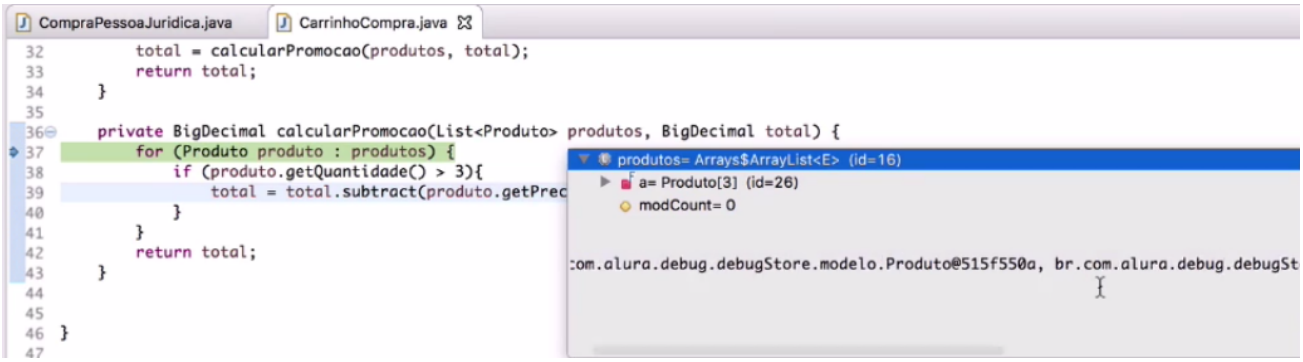
É criada uma variável nova, `multiplicador`, pela qual o valor total é multiplicado. Abrindo a aba "*Variables*", selecionaremos o `multiplicador` e depois, veremos que `5` é a quantidade do respectivo construtor. Queremos multiplicar o valor com desconto por este número.

Agora, a aplicação passará pela variável `total`, após clicarmos em "*Step Over*". A variável foi alterada para `886.45875`, valor do primeiro produto. É possível observar o valor da variável também por meio da aba de variáveis. A vantagem da aba "*Expressions*" é poder separar apenas o que se quer observar, e de maneira mais rápida.

Seguiremos acompanhando o código linha a linha, com outro produto. O `multiplicar` agora vale `2`, cuja soma referente resulta em `2196.50075`. Podemos passar para o próximo produto, repetindo os mesmos procedimentos. O `multiplicar` seguinte é `7`, e o total, `4782.12575`.

Saindo do laço `for`, caímos no método `calcularPromocao` que, como o próprio nome indica, realiza o cálculo da promoção, dada à compra de `3` ou mais produtos iguais. Entraremos neste método para verificar o que acontece lá dentro, apertando `F5` no teclado. O método recebe `produtos` e `BigDecimal`, que é o total.

Uma funcionalidade muito interessante desta IDE (Eclipse) é o "*Show*", que mostrará o valor da variável em um determinado momento. Quais produtos vieram para cá? Podemos passar o mouse em cima, saber que se trata de um `array list`, seu valor, e podemos navegar por eles, clicando nas setas que se encontram ao lado dos mesmos.



Neste caso, existem três objetos, ou produtos, `1`, `2` e `3`. A variável `produto` ainda não foi criada, isto ocorre apenas a partir do laço. O "*Show*", portanto, serve para mostrar de maneira mais rápida o que acontece com uma variável durante

a debugação, sem que seja preciso recorrer à aba de variáveis.

Até o momento, tudo está funcionando perfeitamente. Para aplicação da promoção, temos o primeiro produto, cuja quantia é de `5`, então precisamos tirar `1` deles do total. Do segundo produto, foram comprados `7` itens, portanto passaremos `5` (pois a cada `3` produtos, subtraímos o valor de `1` deles).

Passaremos à próxima linha, em que encontramos `produto.getQuantidade() > 3`. Pela opção "*Show*", verificamos que a quantidade deste produto é `5`, que é maior que `3`. Pulamos à próxima linha e vimos um novo produto, cuja quantidade é `2`, que é menor que `3`, não se aplicando a promoção.

Seguimos, e a quantidade do outro produto é `7`, ou seja, é preciso subtrair o valor de `2` deles, mas vemos que se tira de apenas `1`. De toda forma, vamos continuar rodando a aplicação para ver se isso realmente deixa de acontecer. O laço não foi rodado, acabaram-se os produtos da compra e a conta final se mostra realmente errada.

O valor final é retornado: `4227.13575`. Podemos apertar `F6` ou `F7` para voltarmos. Observamos o método durante todo este tempo, e vamos seguir a aplicação normalmente. A mensagem que aparece no console é "Valor da compra: `4227.13575` ".

Reforçando que valores múltiplos de `3` entram na promoção, o método não está calculando corretamente, pois ele apenas verifica se a quantidade é maior que `3` e, se sim, retira-se o valor de apenas `1` produto. Se o cliente comprou `7` perfumes, é preciso retirar da soma total a quantia referente a `2` perfumes, não `1`.

Dentro do laço `for`, precisamos dividir a quantidade por `3` para saber a quantidade de produtos que entram no critério da promoção, e precisam ser subtraídos do valor total da compra. Para isto, acrescentaremos as seguintes linhas na aba `CarrinhoCompra.java`:

```
int quantidadeDescontar = produto.getQuantidade() / 3;
for(int cont = 0; quantidadeDescontar > cont; cont++ ){
  total = total.substract(produto.getPreco());
}
```

Na alteração que acabamos de fazer, a estrutura `if` que havia anteriormente não existe mais, e a subtração passa a integrar o laço `for`, como demonstrado acima. Rodaremos novamente a aplicação em modo *Debug* para verificar se as alterações feitas resultarão no valor reportado pelo cliente.

Lembrando que temos um *breakpoint* na linha `19`, onde se encontra `BigDecimal`, sabemos que esta parte do método está funcionando corretamente, então não há necessidade de debugarmos mais uma vez por ali. Precisamos parar no método `calcularPromocao()`, então tiraremos o *breakpoint* dali, criando outro na linha `total = calcularPromocao(produtos, total)`.

Rodaremos a app de novo, em modo *Debug*, percorrendo por todo o código, já acrescentando a variável `total` (`4782.12575`) antes de passar pelo método `calcularPromocao`, que é nosso objetivo atual. Queremos que ele seja executado e retornar o valor total. Como faremos isto? Existe alguma maneira de fazê-lo sem necessidade de debugá-lo?

Sim! Basta selecionarmos o método desejado, clicando com o lado direito do mouse sobre ele e escolhendo a opção "*Inspect*", responsável pela execução daquele código, o qual nos mostrará seu resultado. Ao fazermos isto, teremos o valor `3852.13575`, que é exatamente a quantia informada pela empresa. Houve de fato um erro, eles tinham razão, o nosso código estava com erro.

Apertaremos "*Step Over*" e seguiremos com a aplicação clicando em `F8` (ou "*Resume*"), e a mesma mensagem com o valor total aparece no console.

Conseguimos consertar o bug do programa utilizando "*Watch*" para a variável `total`. Falamos sobre "*Show*", útil em mostrar informações quando passamos o mouse em cima do código, e "*Inspect*", que executa um determinado método cujo resultado queremos obter sem que precisemos entrar nele e debugar.