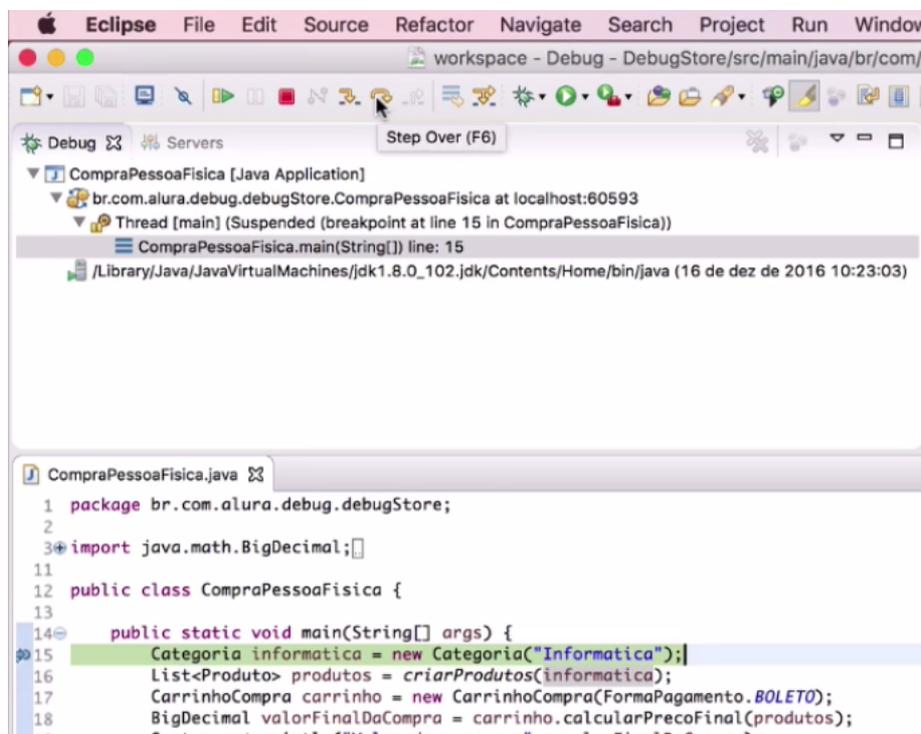


Caminhando pela aplicação

Transcrição

Agora que já sabemos como pausar a aplicação em um determinado *breakpoint*, podemos "andar" nela para identificarmos o local exato do erro. Imagine, por exemplo, uma situação em que você está em uma empresa nova e precisa corrigir um código legado, mas não sabe onde está o erro.

Mas como assim, "andar" pelo código? Analisaremos o método `main`, passando por cada linha, e assim entenderemos por que o cálculo da soma dos valores dos produtos vistos no vídeo anterior está errado. Para caminhar linha a linha, começaremos pelo próprio `main`, com objeto do tipo `Categoria`, que passa um `string` como construtor criando um objeto com o nome `Informatica` (da categoria `Informatica`). Para passarmos à próxima linha, clicaremos no botão "Step over" (por meio do atalho `F6` do teclado), localizado na barra de ferramentas no topo do programa:



Na próxima linha, há o método `criarProdutos()`, passando pela categoria criada anteriormente, a qual retorna uma lista de produtos. Usando `F6` mais uma vez, vimos que foi criado um objeto do tipo `CarrinhoCompra`, passando um enum no construtor, `FormaPagamento`. São três formas de pagamento: boleto, cartão de crédito ou débito. Existem diferenças entre elas, porém o que importa no momento é que o objeto foi criado, sendo jogado à variável `carrinho`, um objeto, o carrinho de compras.

Passamos para a próxima linha, em que a variável `carrinho` chama o método `calcularPrecoFinal` passando `produtos` para calcular o preço final, retornando e colocando a variável `valorFinaldaCompra`, que é o `BigDecimal`. Não encontramos o erro ali, portanto, passamos à próxima linha: vemos um "sysout" ("System out") com a variável que acabamos de criar na linha anterior (`valorFinaldaCompra`).

Utilizaremos o atalho de "Step over" novamente, caindo no valor final da compra (" R\$268,11 ") a partir da concatenação da variável `valorFinaldaCompra`, que é o `BigDecimal`:

```

1 package br.com.alura.debug.debugStore;
2
3 import java.math.BigDecimal;
11
12 public class CompraPessoaFisica {
13
14     public static void main(String[] args) {
15         Categoria informatica = new Categoria("Informatica");
16         List<Produto> produtos = criarProdutos(informatica);
17         CarrinhoCompra carrinho = new CarrinhoCompra(FormaPagamento.BOLETO);
18         BigDecimal valorFinalDaCompra = carrinho.calcularPrecoFinal(produtos);
19         System.out.println("Valor da compra: " + valorFinalDaCompra);
20     }
21
22
23     private static List<Produto> criarProdutos(Categoria categoria) {

```

Console [Java Application] /Library/Java/JavaVirtualMachines/jdk1.8.0_102.jdk/Contents/Home/bin/java (16 de dez de 2016 10:23:03)
Valor da compra: 268.11

O cálculo não está ali, só faz a chamada para os outros métodos, `criarProdutos` e `calcularPrecoFinal`. Chegamos ao fim do método `main`, e o cálculo continua errado. No entanto, vimos que o problema não se encontra no `main`.

Temos mais duas opções: seguir a aplicação, fazendo-se um "Resume" apertando o `F8`, ou terminá-la, parando-a e voltando a um ponto inicial. Vamos escolher a primeira opção. Na prática, não houve nenhuma diferença.

Recolocaremos a aplicação em modo *Debug*, já que o erro ainda não foi encontrado. Este processo costuma ser assim mesmo, sendo necessário que se rode a aplicação diversas vezes. Clicaremos com o botão direito do mouse, e depois em "Debug As > Java Application". A aplicação foi parada assim que atingiu o *breakpoint*.

Continuaremos apertando o `F6`, procurando o erro não apenas no método principal, mas em outros também. Como conseguimos "entrar" nestes métodos? Selecionando o botão "Step into", localizado ao lado do "Step over", ou `F5` no teclado, "entrarmos" no método.

Veja por exemplo o `private static`, que retorna uma lista do tipo `Produto`, recebendo `categoria` como argumento. Agora que estamos dentro do método, navegaremos por ele utilizando o atalho `F6`.

```

24 private static List<Produto> criarProdutos(Categoria categoria) {
25     List<Produto> produtos = new ArrayList<Produto>();
26     Produto mouse = new Produto("Mouse sem fio Microsoft", new BigDecimal("120"), categoria);
27     produtos.add(mouse);
28     Produto teclado = new Produto("Teclado de Gamer Alien", new BigDecimal("350"), categoria);
29     produtos.add(teclado);
30     Produto monitor = new Produto("Monitor Lg 17 Widescreen", new BigDecimal("250"), categoria);
31     produtos.add(monitor);
32     Produto processador = new Produto("Processador Intel Core I7", new BigDecimal("1500"), categoria);
33     produtos.add(processador);
34     Produto cadeira = new Produto("Cadeira Racing Gamer", new BigDecimal("759"), categoria);
35     produtos.add(cadeira);
36     return produtos;
37 }
38
39 }
40

```

É criado um array `list<Produto>`, e na próxima linha há um objeto do tipo `Produto`, em cujo construtor se passa o string que é o nome do produto ("Mouse sem fio Microsoft", por exemplo). Passa-se então um `BigDecimal` de valor "120" (reais) e o argumento `categoria`, criado no método `main`, que se trata de "Informatica".

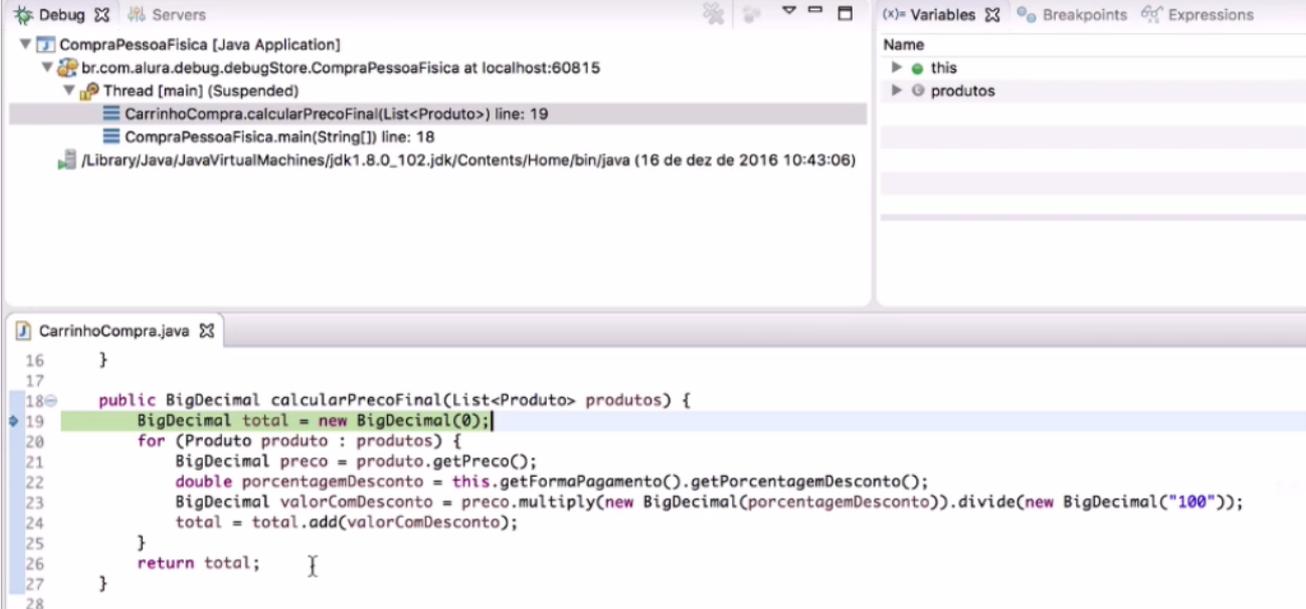
Continuando na próxima linha, teremos `produtos`, array `list`, `add` (de "adiciona") o "mouse". Ou seja, o produto foi criado e adicionado à lista. O mesmo acontece com o produto `teclado`, temos a string "Teclado de Gamer Alien", passaremos o preço "350", e a `categoria`. Na linha seguinte, o produto foi adicionado no array `list`, e o mesmo ocorre nas próximas linhas, com todos os outros produtos, alcançando-se finalmente o retorno dos produtos (`return produtos;`).

Utilizando o atalho `F6` novamente, retornamos a um método. Nenhum cálculo é feito, portanto não é ali que está o erro. Apertamos `F6` mais umas vezes, e ele "impronta" (de *imprint* do inglês) novamente. Vimos que o problema não consta no método `main` nem em `criarProdutos`. O erro persiste, porém ainda não foi encontrado.

Alcançamos o final do `main` de novo, e temos as mesmas opções: continuar (`F8`), ou parar a aplicação (`F2`). Na vez anterior, continuamos, mas desta vez vamos parar a aplicação. Depois, iremos à terceira rodada, analisando o método `calcularPrecoFinal`, colocando no modo *Debug* clicando em "Debug As > Java Application" pelo lado direito do mouse.

Repetiremos os cliques em `F6`, entrando dentro do método `criarProdutos` através do `F5` ("Step into"). Passando-se por todos os produtos e listas, queremos retornar ao método principal e, para isto, existe o botão "Step return" (`F7` no teclado), que nos faz retornar ao método pela qual fizemos a chamada inicial. Lembrando que ele só funciona caso já estejamos dentro de algum método.

Voltamos então `main` e, analisando linha a linha, chegamos ao `calcularPrecoFinal`. Vamos entrar nele para verificar se o erro está ali, apertando-se `F5`. O `public` retorna o `BigDecimal` `calcularPrecoFinal`, recebendo como argumento uma lista de produtos. Agora, sim, calcula-se o valor total, ou seja, chegamos onde queríamos:



```
16 }
17
18 public BigDecimal calcularPrecoFinal(List<Produto> produtos) {
19     BigDecimal total = new BigDecimal(0);
20     for (Produto produto : produtos) {
21         BigDecimal preco = produto.getPreco();
22         double porcentagemDesconto = this.getFormaPagamento().getPorcentagemDesconto();
23         BigDecimal valorComDesconto = preco.multiply(new BigDecimal(porcentagemDesconto)).divide(new BigDecimal("100"));
24         total = total.add(valorComDesconto);
25     }
26     return total;
27 }
28
```

Descobriremos o motivo por que a lista de produtos está sendo somada de forma errada no próximo vídeo. O importante é que aprendemos a "andar" (ou "navegar") pela aplicação, entrando nos métodos e retornando a um ponto anterior depois.