01

Código mais sucinto com Method references

Transcrição

Com os lambdas e métodos default, conseguimos escrever a ordenação das Strings de uma forma bem mais sucinta:

```
palavras.sort((s1, s2) -> Integer.compare(s1.length(), s2.length()));
```

Podemos ir além.

Métodos default em Comparator

Há vários métodos auxiliares no Java 8. Até em interfaces como o `Comparator`. E você pode ter um método default que é estático. Esse é o caso do `Comparator.comparing`, que é uma fábrica, uma factory, de `Comparator`. Passamos o lambda para dizer qual será o critério de comparação desse `Comparator`, repare:

```
palavras.sort(Comparator.comparing(s -> s.length()));
```

Veja a expressividade da linha, está escrito algo como "palavras ordene comparando s.length". Podemos quebrar em duas linhas para ver o que esse novo método faz exatamente:

```
Comparator<String> comparador = Comparator.comparing(s -> s.length());
palavras.sort(comparador);
```

Dizemos que `Comparator.comparing` recebe um lambda, mas essa é uma expressão do dia a dia. Na verdade, ela recebe uma instância de uma interface funcional. No caso é a interface `Function` (http://docs.oracle.com/javase/8/docs/api/java/util/function/Function.html) que tem apenas um método, o `apply`. Para utilizarmos o `Comparator.comparing`, nem precisamos ficar decorando os tipos e assinatura do método dessas interfaces funcionais. Essa é uma vantagem dos lambdas. Você também vai acabar programando dessa forma. É claro que, com o tempo, você vai conhecer melhor as funções do pacote `java.util.functions`. Vamos quebrar o código mais um pouco. Não se esqueça de dar os devidos `imports`.

```
Function<String, Integer> funcao = s -> s.length();
Comparator<String> comparador = Comparator.comparing(funcao);
palavras.sort(comparador);
```

A interface `Function` vai nos ajudar a passar um objeto para o `Comparator.comparing` que diz qual será a informação que queremos usar como critério de comparação. Ela recebe dois tipos genéricos. No nosso caso, recebe uma `String`, que é o tipo que queremos comparar, e um `Integer`, que é o que queremos extrair dessa string para usar como critério. Poderia até mesmo criar uma classe anônima para implementar essa `Function` e seu método `apply`, sem utilizar nenhum lambda. O código ficaria grande e tedioso.

Quisemos quebrar em três linhas para que você pudesse enxergar o que ocorre por trás exatamente. Sem dúvida o `palavras.sort(Comparator.comparing(s -> s.length()))` é mais fácil de ler. Dá para melhorar ainda mais? Sim!

Method reference

É muito comum escrevermos lambdas curtos, que simplesmente invocam um único método. É o exemplo de `s -> s.length()`. Dada uma `String`, invoque e retorne o método `length`. Por esse motivo, há uma forma de escrever esse tipo de lambda de uma forma ainda mais reduzida. Em vez de fazer:

```
palavras.sort(Comparator.comparing(s -> s.length()));
```

Fazemos uma referência ao método (method reference):

```
palavras.sort(Comparator.comparing(String::length));
```

São equivalentes nesse caso! Sim, é estranho ver `String::length` e dizer que é equivalente a um lambda, pois não há nem a `->` e nem os parênteses de invocação ao método. Por isso é chamado de method reference. Ela pode ficar ainda mais curta com o `import static`:

```
import static java.util.Comparator.*;
palavras.sort(comparing(String::length));
```

Vamos ver melhor a semelhança entre um lambda e seu method reference equivalente. Veja as duas declarações a seguir:

```
Function<String, Integer> funcao1 = s -> s.length();
Function<String, Integer> funcao2 = String::length;
```

Elas ambas geram a mesma função: dada um `String`, invoca o método `length` e devolve este `Integer`. As duas serão avaliadas/resolvidas (*evaluated*) para `Functions` equivalentes.

Quer um outro exemplo? Vejamos o nosso `forEach`, que recebe um `Consumer`:

```
palavras.forEach(s -> System.out.println(s));
```

Dada uma `String`, invoque o `System.out.println` passando-a como argumento. É possível usar method reference aqui também! Queremos invocar o `println` de `System.out`:

```
palavras.forEach(System.out::println);
```

Novamente pode parecer estranho. Não há os parênteses, não há a flechinha (`->`), nem os argumentos que o `Consumer` recebe. Fica tudo implícito. Dessa vez, o argumento recebido (isso é, cada palavra dentro da lista `palavras`), não será a variável onde o método será invocado. O Java 8 consegue perceber que tem um `println` que recebe objetos, e invocará esse método, passando a `String` da vez.

Quando usar lambda e quando usar method reference? Algumas vezes não é possível usar method references. Se você tiver, por exemplo, um lambda que dada uma `String`, pega os 5 primeiros caracteres, faríamos `s -> s.substring(0,`

`5)` . Esse lambda não pode ser escrito como method reference! Pois não é uma simples invocação de métodos onde os parâmetros são os mesmos que os do lambda.