01

Default Methods

Transcrição

As tarefas mais comuns de um programador podem ser desafiadoras no Java. O motivo? A sintaxe com mais de 20 anos, tornando-a uma linguagem burocrática. Felizmente isso mudou significativamente com o Java 8. Um exemplo? Ordenação de objetos.

Vamos usar o Eclipse nos nossos exemplos. Você pode utilizar qualquer outra IDE, ou até mesmo a linha de comando. O antigo Eclipse Kepler 4.3 possui suporte ao Java 8 via download, a partir do Eclipse Luna 4.4, esse suporte já vem ativado. Lembre-se de verificar se você tem o Java 8 instalado, indo ao console/terminal e digitando `java -version`. Deve sair `1.8.0`. Caso contrário, atualize a versão do seu java (http://www.oracle.com/technetwork/pt/java/javase/downloads/index.html?ssSourceSiteId=otnes).

No eclipse, crie um novo projeto chamado `Java8` e, através das propriedades do projeto, escolha a opção Java Compiler e ative a versão 8. Se ela não está disponível e você tem certeza que instalou o Java 8, basta adicionar esse JDK em Windows, Preferences, Java, Installed JREs.

Em uma nova classe `OrdenaStrings`, crie o método `main` e vamos fazer uma lista de strings e trabalhar com ele sem nenhum dos novos recursos da linguagem:

```
List<String> palavras = new ArrayList<>();
palavras.add("alura online");
palavras.add("casa do código");
palavras.add("caelum");
```

Vale lembrar que podemos criar uma lista de objetos diretamente via `Arrays.asList`, fazendo `List<String> palavras = Arrays.asList("", "", ...)`. A diferença é que não se pode mudar a quantidade de elementos de uma lista devolvida por esse método.

Como fazemos para ordenar essa lista? Podemos fazer isso sem usar nenhuma novidade: com o `Collections.sort`:

```
Collections.sort(palavras);
System.out.println(palavras);
```

E se quisermos ordenar essas palavras em uma ordem diferente? Por exemplo, pela ordem do tamanho das palavras. Nesse caso, utilizaremos um `Comparator`. Podemos criá-la como uma outra classe, por enquanto apenas o esqueleto:

```
class ComparadorDeStringPorTamanho implements Comparator<String> {

    public int compare(String s1, String s2) {
        return 0;
    }

}
```

O que preencher aí dentro? Se você lembrar, o contrato da interface `Comparator` diz que devemos devolver um número negativo se o primeiro objeto for menor que o segundo, um número positivo caso contrário e zero se forem equivalentes. Esse "maior", "menor" e "equivalente" é algo que nós decidimos. No nosso caso, vamos dizer que uma `String` é "menor" que outra se ela tiver menos caracteres. Então podemos fazer:

```
class ComparadorDeStringPorTamanho implements Comparator<String> {
    public int compare(String s1, String s2) {
        if(s1.length() < s2.length())
            return -1;
        if(s1.length() > s2.length())
            return 1;
        return 0;
    }
}
```

E, para ordenar com esse novo critério de comparação, podemos fazer:

```
Comparator<String> comparador = new ComparadorDeStringPorTamanho();
Collections.sort(palavras, comparador);
```

Até aqui, nenhuma novidade. No decorrer do curso, você verá como esse código ficará muito, muito menor, mais sucinto e expressivo com cada recurso que formos estudar do Java 8. Vamos ao primeiro deles. Em vez de usar o `Collections.sort`, podemos invocar essa operação na própria `List`! Veja:

```
Comparator<String> comparador = new ComparadorDeStringPorTamanho();
palavras.sort(comparador);
```

Parece pouco, mas há muita coisa por trás. Em primeiro lugar, esse método `sort` não existia antes na interface `List` (http://docs.oracle.com/javase/8/docs/api/java/util/List.html), nem em suas mães (`Collection` e `Iterable`).

Será então que simplesmente adicionaram um novo método? Se tivessem feito assim, haveria um grande problema: todas as classes que implementam `List` parariam de compilar, pois não teriam o método `sort`. E há muitas, muitas classes que implementam essas interfaces básicas do Java. Há implementações no Hibernate, no Google Collections e muito mais.

Default Methods

Para evitar essa quebra, o Java 8 optou por criar um novo recurso que possibilitasse adicionar métodos em interfaces e implementá-los ali mesmo! Se você abrir o código fonte da interface `List`, verá esse método:

```
    default void sort(Comparator<? super E> c) {
        Collections.sort(this, c);
    }
```

É um default method! Um método de interface que você não precisa implementar na sua classe se não quiser, pois você terá já essa implementação default. Repare que ele simplesmente delega a invocação para o bom e velho `Collections.sort`, mas veremos que outros métodos fazem muito mais.

Default methods foi uma forma que o Java encontrou para evoluir interfaces antigas, sem gerar incompatibilidades. Não é uma novidade da linguagem: Scala, C# e outras possuem recursos similares e até mais poderosos. E repare que é diferente de uma classe abstrata: em uma interface você não pode ter atributos de instância, apenas esses métodos que delegam chamadas ou trabalham com os próprios métodos da interface.

foreach, Consumer e interfaces no java.util.functions

Vamos a um outro método default adicionado as coleções do Java: o `forEach` na interface `Iterable`. Como `Iterable` é mãe de `Collection`, temos acesso a esse método na nossa lista.

Se você abrir o JavaDoc ou utilizar o auto complete do Eclipse, verá que `List.forEach` recebe um `Consumer`, que é uma das muitas interfaces do novo pacote `java.util.functions`. Então vamos criar um consumidor de `String`:

```
class ConsumidorDeString implements Consumer<String> {
    public void accept(String s) {
        System.out.println(s);
    }
}
```

E podemos passar uma instância dessa para o `forEach`:

```
Consumer<String> consumidor = new ConsumidorDeString();
palavras.forEach(consumidor);
```

Interessante? Ainda não muito. Talvez fosse mais direto e simples escrever um `for(String s : lista)`.

Default methods é o primeiro recurso que conhecemos. Sim, é bastante simples e parece não trazer grandes melhorias. O segredo é utilizá-los junto com lambdas, que você verá a seguir, e trará um impacto significativo para o seu código.