06

Laço com for

Transcrição

O `for` tem a sintaxe um pouco mais estranha. O `while` é uma estrutura de laço, e o `for` realiza a mesma tarefa, porém possui algumas vantagens em relação à legibilidade, mesmo que o resultado final - o *bytecode* - seja o mesmo. Criaremos `TestaFor`, em que incluiremos algo equivalente ao laço feito anteriormente, que conta de `0` a `10` imprimindo todos os números.

Diferentemente do `while`, não é preciso declararmos `contador` fora dele, pois o `for`, palavra chave do Java, tem uma sintaxe muito diferente. Até então, utilizamos apenas ponto e vírgula no fim dos *statements*, isto é, das linhas. Neste caso, usaremos o ponto e vírgula **dentro dos parênteses** (isto também herança do C).

Dentro dos parênteses, então, serão criados três "espaços" intercalados por ponto e vírgula, e então abriremos e fecharemos as chaves normalmente. O primeiro espaço é opcional e costuma ter a declaração e inicialização da variável, sendo executado **apenas uma vez**.

O segundo espaço é executado **todas as vezes** e contém a condição booleana para saber se ele deve ou não entrar no laço, ou seja, executar a próxima iteração. No nosso caso, queremos saber se `contador` é menor ou igual a `10`, como no `while`.

O terceiro espaço geralmente é ocupado por aquilo a ser executado ao fim de cada iteração, o que acaba sendo um tanto estranho para quem não está bem ambientado com isto. O código ficará desta maneira:

```
public class TestaFor {

    public static void main(String[] args) {
        for(int contador = 0; contador <= 10; contador++) {
            System.out.println(contador);
        }
    }
}
```

Salvaremos e rodaremos o código, e obteremos o esperado, como em `while`:

```
0
1
2
3
4
5
6
7
8
9
10
```

Diferentemente do `while`, apesar de `int contador = 0` valer no escopo do `for` inteiro em todas as iterações, ele não é zerado, sendo executado apenas uma vez, e por isto sua sintaxe não é muito intuitiva. Se quisermos imprimir o último

valor que o `contador` estava lendo, não conseguiremos, por conta do escopo.

O `for` oferece a possibilidade de haver uma variável que participa de todas as iterações, que é o que precisamos, mas depois do `for`, ela deixa de valer.

Não é melhor usarmos o `while`, então? Depende. Muitas vezes queremos utilizar a variável temporariamente, somente dentro do laço, e é por isso que o `for` é mais atrativo, e se adequa melhor a este tipo de caso.

No entanto, `while` e `for` são intercambiáveis, e inclusive existe outro laço, denominado *do-while*, que não veremos neste curso, mas que também poderia ser utilizado.