02

Crie e use laços do tipo for, incluindo o enhanced for

Observando um pouco os códigos que utilizam `while`, dá para perceber que eles são formados por quatro partes: inicialização, condição, comandos e atualização.

```
int i = 1; // Inicialização
while (i < 10) { // Condição
    System.out.println(i); // Comandos
    i++; // Atualização
}
```

COPIAR CÓDIGO

A inicialização é importante para que o laço execute adequadamente. Mesmo com essa importância, a inicialização fica separada do `while`.

A atualização é fundamental para que não aconteça um "loop infinito". Porém, a sintaxe do `while` não a coloca em evidência.

Há um outro laço que coloca em destaque a inicialização, a condição e a atualização. Esse laço é o `for`.

```
for (int i = 1; i < 10; i++) {
    System.out.println(i);
}
```

COPIAR CÓDIGO

O `for` tem três argumentos separados por `;` . O primeiro é a inicialização, o segundo, a condição, e o terceiro, a atualização.

A inicialização é executada somente uma vez no começo do `for` . A condição é verificada no começo de cada rodada (iteração). A atualização é executada no fim de cada iteração.

Todos os três argumentos do `for` são opcionais. Desta forma, você poderia escrever o seguinte código:

```
for(;;){
    // CODIGO
}
```

COPIAR CÓDIGO

O que acontece com esse laço? Para responder essa pergunta é necessário saber quais são os "valores default" colocados nos argumentos do `for` , quando não é colocado nada pelo programador. A inicialização e a atualização ficam realmente vazias. Agora, a condição recebe por padrão o valor `true` . Então, o código anterior depois de compilado fica assim:

```
//loop infinito
for (;true;){
    // CODIGO
}
```

COPIAR CÓDIGO

Nos exemplos anteriores, basicamente o que fizemos na inicialização foi declarar e inicializar apenas uma variável qualquer. Porém, é permitido declarar diversa

variáveis de um mesmo tipo ou inicializar diversas variáveis.

Na inicialização, não é permitido declarar variáveis de tipos diferentes. Mas é possível informar variáveis de nomes diferentes. Veja os exemplos:

```
// Declarando três variáveis do tipo int e inicializando as três.
// Repare que o "," separa as declarações e inicializações.
for (int i = 1, j = 2, k = 3;;){
    // CODIGO
}

// Declarando três variáveis de tipos diferentes
int a;
double b;
boolean c;

// Inicializando as três variáveis já declaradas
for (a = 1, b = 2.0, c = true;;){
    // CODIGO
}
```

COPIAR CÓDIGO

Na atualização, é possível fazer diversas atribuições separadas por `,`.

```
//a cada volta do laço, incrementamos o i e decrementamos o j
for (int i=1,j=2;; i++,j--){
    //código
}
```

COPIAR CÓDIGO

Como já citamos anteriormente, não é possível, na inicialização, declarar variáveis de tipos diferentes:

```
for (int i=1, long j=0; i< 10; i++){ // erro
    //código
}
```

COPIAR CÓDIGO

No campo de condição, podemos passar qualquer expressão que resulte em um `boolean`. São exatamente as mesmas regras do `if` e `while`.

No campo de atualização, não podemos só usar os operadores de incremento, podemos executar qualquer trecho de código:

```
    for (int i = 0; i < 10; i += 3) { //somatório
    //código
    }

    for (int i = 0; i < 10; System.out.println(i++)) { //
bizarro
    //código
    }
```

COPIAR CÓDIGO

Enhanced for

Quando vamos percorrer uma coleção de objetos ou um array, podemos usar uma versão simplificada do `for` para percorrer essa coleção de maneira simplificada. Essa forma simplificada é chamada de ::enhanced for::, ou ::foreach:: :

```
int[] numeros = {1,2,3,4,5,6};
for (int num : numeros) { //enhanced for
    System.out.println(num);
}
```

COPIAR CÓDIGO

A sintaxe é mais simples, temos agora 2 partes dentro da declaração do `for` :

```
for(VARIAVEL : COLEÇÃO){
    CODIGO
}
```

COPIAR CÓDIGO

Nesse caso, declaramos uma variável que irá receber cada um dos membros da coleção ou array que estamos percorrendo. O próprio `for` irá a cada iteração do laço atribuir o próximo elemento da lista à variável. Seria o equivalente a fazer o seguinte:

```
int[] numeros = {1,2,3,4,5,6};

for( int i=0; i < numeros.length; i++){
    int num = numeros[i]; //declaração da variável e
atribuição
    System.out.println(num);
}
```

COPIAR CÓDIGO

Se fosse uma ::collection::, o código fica mais simples ainda se comparado com o `for` original:

```
ArrayList<String> nomes = //lista com vários nomes

//percorrendo a lista com o for simples
for(Iterator<String> iterator = nomes.iterator();
    iterator.hasNext();){
    String nome = iterator.next();
    System.out.println(nome);
}

//percorrendo com o enhanced for
for (String nome : nomes) {
    System.out.println(nome);
}
```

COPIAR CÓDIGO

Existem, porém, algumas limitações no ::enhanced for::. Não podemos, por exemplo, modificar o conteúdo da coleção que estamos percorrendo usando a variável que declaramos:

```
ArrayList<String> nomes = //lista com vários nomes

//tentando remover nomes da lista
for (String nome : nomes) {
    nome = null;
}

//o que imprime abaixo?
for (String nome : nomes) {
```

```
        System.out.println(nome);
    }
```

COPIAR CÓDIGO

Ao executar esse código, você perceberá que a coleção não foi modificada, nenhum elemento mudou de valor para `null` .

Outra limitação é que não há uma maneira natural de saber em qual iteração estamos, já que não existe nenhum contador. Para saber em qual linha estamos, precisaríamos de um contador externo. Também não é possível percorrer duas coleções ao mesmo tempo, já que não há um contador centralizado. Para todos esses casos, é recomendado usar o `for` simples.