02

Use operadores Java

Para manipular os valores armazenados das variáveis, tanto as primitivas quanto as não primitivas, a linguagem de programação deve oferecer operadores. Um dos operadores mais importantes é o que permite guardar um valor em uma variável. Esse operador é denominado **operador de atribuição**.

No Java, o símbolo `=` representa o operador de atribuição. Para atribuir um valor precisamos de uma variável à qual será atribuído o valor, e do valor:

```
long idade = ; // não compila, onde está o valor?
long = 15; // não compila, onde está o nome da variável?
long idade = 15; // compila
idade = 15;
// compila desde que a variável tenha sido declarada
// anteriormente
```

COPIAR CÓDIGO

Para um valor ser atribuído a uma variável, ambos devem ser compatíveis. Um valor é compatível com uma variável se ele for do mesmo tipo dela ou de um tipo menos abrangente.

```
// Iniciando uma variável com o operador de atribuição "=".
int idade = 10;

// O valor literal 10 é do tipo int e a variável é do tipo long
// Como int é menos abrangente que long essa atribuição está
```

```
// correta.
long idade = 10;
```

COPIAR CÓDIGO

Procure sempre se lembrar dos tamanhos dos primitivos quando estiver fazendo a prova. `int` é um número médio, será que ele "cabe" em uma variável do tipo `long` (número grande)? Sim, logo o código compila. Mais exemplos:

```
int a = 10;       // tipos iguais
long b = 20;      // int cabe em um long
float c = 10f;  // tipos iguais
double d = 20.0f; // float cabe em um double
double e = 30.0; // tipos iguais
float f = 40.0; // erro, double não cabe em um float.
int g = 10l; // erro, long não cabe em int
float h = 10l; // inteiros cabem em decimais
double i = 20; // inteiros cabem em decimais
long j = 20f; // decimais não cabem em inteiros
```

COPIAR CÓDIGO

A exceção a essa regra ocorre quando trabalhamos com tipos inteiros menos abrangentes que `int` (`byte` , `short` e `char`). Nesses casos, o compilador permite que atribuamos um valor inteiro, desde que compatível com o tipo:

```
byte b1 = 10;
byte b2 = 200; // não compila, estoura byte

char c1 = 10;
char c2 = -3; // não compila, char não pode ser negati...
```

COPIAR CÓDIGO

Atribuição e referência

Quando trabalhamos com referências, temos que lembrar do polimorfismo:

```
List<String> lista = new ArrayList<String>();
```

COPIAR CÓDIGO

E no caso do Java 7, quando atribuímos com ::generics:: podemos usar o operador diamante:

```
// operador diamante na atribuição e inicialização:
ArrayList<String> lista = new ArrayList<>();

// operador diamante e polimorfismo junto:
List<String> lista = new ArrayList<>();
```

COPIAR CÓDIGO

Lembre que as atribuições em Java são por cópia de valor, sempre. No tipo primitivo, copiamos o valor, em referências a objetos, copiamos o valor da referência (não duplicamos o objeto):

```
List<String> lista = new ArrayList<>();
```

```
// copia o valor da referência, o objeto é o mesmo
List<String> lista2 = lista;
lista2.add("Guilherme");

// verdadeiro
System.out.println(lista.size() == lista2.size());

int idade = 15;

int idade2 = idade; // copia o valor
idade2 = 20;

System.out.println(idade == idade2); // falso
```

COPIAR CÓDIGO