02

Operadores de incremento e decremento

Incrementos e decrementos

Para facilitar a codificação, ainda podemos ter operadores que fazem cálculos (aritméticos) e atribuição em uma única operação. Para somar ou subtrair um valor em 1, podemos usar os operadores de incremento/decremento:

```
int i = 5;

// 5 - pós-incremento, i agora vale 6
System.out.println(i++);

// 6 - pós-decremento, i agora vale 5
System.out.println(i--);

// 5
System.out.println(i);
```

COPIAR CÓDIGO

E incrementos e decrementos antecipados:

```
int i = 5;

System.out.println(++i);    // 6 - pré-incremento
System.out.println(--i);    // 5 - pré-decremento
System.out.println(i);        // 5
```

COPIAR CÓDIGO

Cuidado com os incrementos e decrementos em relação a **pré** e **pós**. Quando usamos pós-incremento, essa é a última coisa a ser executada. E quando usamos o pré-incremento, é sempre a primeira.

```
int i = 10;

// 10, primeiro imprime, depois incrementa
System.out.println(i++);

// 11, valor já incrementado.
System.out.println(i);

// 12, incrementa primeiro, depois imprime
System.out.println(++i);

// 12, valor incrementado.
System.out.println(i);
```

COPIAR CÓDIGO

Existem ainda operadores para realizar operações e atribuições de uma só vez:

```
int a = 10;

// para somar 2 em a
a = a + 2;

//podemos obter o mesmo resultado com:
a += 2;

//exemplos de operadores:
```

```
int i = 5;

i += 10; //soma e atribui
System.out.println(i);        // 15

i -= 10; //subtrai e atribui
System.out.println(i);        // 5

i *= 3; // multiplica e atribui
System.out.println(i);        // 15

i /= 3; // divide a atribui
System.out.println(i);        // 5

i %= 2; // divide por 2, e atribui o resto
System.out.println(i);        // 1

System.out.println(i+=3); // soma 3 e retorna o resultado:
4
```

COPIAR CÓDIGO

Nesses casos, o compilador ainda dá um desconto para operações com tipos teoricamente incompatíveis. Veja:

```
byte b1 = 3; // compila, dá um desconto
b1 = b1 + 4; // não compila, conta com int devolve int

byte b2 = 3; // compila, dá um desconto
b2 += 4; // compila também, compilador gente boa!
```

COPIAR CÓDIGO

Esse último caso compila inclusive se passar valores absurdamente altos: `b2+=400` é diferente de `b2 = b2 + 400`. Ele faz o ::casting:: e roda normalmente.

Cuidado também com o caso de atribuição com o próprio autoincremento:

```
int a = 10;
a += ++a + a + ++a;
```

COPIAR CÓDIGO

Como a execução é do primeiro para o último elemento das somas, temos as reduções:

```
a += ++a + a + ++a;
a = a + ++a + a + ++a;
a = 10 +11 + a + ++a;
a = 10 + 11 + 11 + ++a;
a = 10 + 11 + 11 + 12;
a = 44; // a passa a valer 44
```

COPIAR CÓDIGO

Um outro exemplo de operador pós-incremento, cujo resultado é 1 e 2:

```
int j = 0;
int i = (j++ * j + j++);
System.out.println(i);
System.out.println(j);
```

COPIAR CÓDIGO

Pois:

```
i = (0 * j + j++); // j vale 1
i = (0 * 1 + j++); // j vale 1
i = (0 * 1 + 1); // j vale 2
i = 1; // j vale 2
```

COPIAR CÓDIGO

Podemos fazer diversas atribuições em sequência, que serão executadas da direita para a esquerda. O resultado de uma atribuição é sempre o valor da atribuição:

```
int a = 15, b = 20, c = 30;
a = b = c; // b = 30, portanto a = 30
```

COPIAR CÓDIGO

Outro exemplo mais complexo:

```
int a = 15, b = 20, c = 30;
a = (b = c + 5) + 5; // c = 30, portanto b = 35, portanto a =
40
```

COPIAR CÓDIGO