02

Declarar e inicializar variáveis

```
// Declaração
int idade;

System.out.println(idade); // erro de compilação
```

COPIAR CÓDIGO

```
// Declaração
int idade;

// Inicialização explícita de uma variável
idade = 10;

// Utilização da variável
System.out.println(idade); // ok
```

COPIAR CÓDIGO

Podemos declarar e iniciar a variável na mesma instrução:

```
// Declaração e inicialização explícita na mesma linha
double pi = 3.14;
```

COPIAR CÓDIGO

Se eu tenho um `if`, a inicialização deve ser feita em todos os caminhos possíveis.

```
void metodo(int a) {
    double x;
    if(a > 1) {
        x = 6;
    }
    System.out.println(x); // talvez x não tenha sido
                           // inicializado, portanto não compila
}
```

COPIAR CÓDIGO

Quando a variável é membro de uma classe, ela é iniciada implicitamente junto com o objeto com um valor *default*:

```
class Prova {
     double tempo;
}

// Implicitamente, na criação de um objeto Prova,
// o atributo tempo é iniciado com 0
Prova prova = new Prova();

// Utilização do atributo tempo
System.out.println(prova.tempo);
```

COPIAR CÓDIGO

Outro momento em que ocorre a inicialização implícita é na criação de arrays:

```
    int[] numeros = new int[10];
    System.out.println(numeros[0]); // imprime 0
```

COPIAR CÓDIGO

Quando iniciadas implicitamente, os valores default para as variáveis são:

- primitivos numéricos inteiros - **0**
- primitivos numéricos com ponto flutuante - **0.0**
- `boolean` - **false**
- `char` - **vazio**, equivalente a 0
- referências - **null**

Os tipos das variáveis do Java podem ser classificados em duas categorias: primitivos e não primitivos (referências).

Tipos primitivos

Todos os tipos primitivos do Java já estão definidos e não é possível criar novos tipos primitivos. São oito os tipos primitivos do Java: `byte`, `short`, `char`, `int`, `long`, `float`, `double` e `boolean`.

O `boolean` é o único primitivo não numérico. Todos os demais armazenam números: `double` e `float` são ponto flutuante, e os demais, todos inteiros (incluindo `char`). Apesar de representar um caractere, o tipo `char` armazena seu valor como um número positivo. Em Java, não é possível declarar variáveis com ou sem sinal (*unsigned*), todos os números (exceto `char`) podem ser positivos e negativos.

Cada tipo primitivo abrange um conjunto de valores. Por exemplo, o tipo `byte` abrange os números inteiros de -128 até 127. Isso depende do tamanho em bytes do tipo sendo usado.

Os tipos inteiros têm os seguintes tamanhos:

- `byte` - 1 byte (8 bits, de -128 a 127);
- `short` - 2 bytes (16 bits, de –32.768 a 32.767);
- `char` - 2 bytes (só positivo), (16 bits, de 0 a 65.535);
- `int` - 4 bytes (32 bits, de –2.147.483.648 a 2.147.483.647);
- `long` - 8 bytes (64 bits, de –9.223.372.036.854.775.808 a 9.223.372.036.854.775.807).

Decorar o tamanho dos primitivos para prova

Não há a necessidade de decorar o intervalo e tamanho de todos os tipos de primitivos para a prova. O único intervalo cobrado é o do `byte` (-128 a 127).

É importante também saber que o `char`, apesar de ter o mesmo tamanho de um `short`, não consegue armazenar todos os números que cabem em um `short`, já que o `char` só armazena números positivos.

Para saber mais: calculando o intervalo de valores

Dado o número de bits **N** do tipo primitivo inteiro, para saber os valores que ele aceita usamos a seguinte conta:

```
-2^(n-1) a 2^(n-1) -1
```

COPIAR CÓDIGO

O `char`, por ser apenas positivo, tem intervalo:

```
0 a 2^(16) -1
```

COPIAR CÓDIGO

Os tipos ponto flutuante têm os seguintes tamanhos em notação científica:

- `float` - 4 bytes (32 bits, de +/–1.4 *10 ^ 45 a +/–3.4028235* 10 ^ 38);
- `double` - 8 bytes (64 bits, de +/–4.9 *10 ^ 324 a +/–1.79769313486231 57* 10 ^ 308).

Todos os números de ponto flutuante também podem assumir os seguintes valores:

- +/– infinity
- +/- 0
- NaN (Not a Number)

Literais

Na codificação, muitas vezes o programador coloca os valores das variáveis diretamente no código-fonte. Quando isso ocorre, dizemos que o valor foi literalmente escrito no código, ou seja, é um **valor literal**.

Todos os valores primitivos maiores que `int` podem ser expressos literalmente. Por outro lado, as referências (valores não primitivos) não podem ser expressas de maneira literal (não conseguimos colocar direto os endereços de memória dos objetos).

Ao inicializar uma variável, podemos explicitar que queremos que ela seja do tipo `double` ou `long` usando a letra específica:

```
// compila pois 737821237891232 é um double válido
```

```
System.out.println(737821237891232d);

// compila pois 737821237891232 é um long válido
System.out.println(7378212378912321);

// nao compila pois 737821237891232 é um valor maior que
// o int aceita
System.out.println(737821237891232);
```

COPIAR CÓDIGO

Da mesma maneira, o compilador é um pouco esperto e percebe se você tenta quebrar o limite de um `int` muito facilmente:

```
// compila pois 7378212378912321 é um long válido
long l = 7378212378912321;

// não compila pois o compilador não é bobo assim
int i = l;
```

COPIAR CÓDIGO

```
// booleanos
System.out.println(true); // booleano verdadeiro
System.out.println(false); // booleano falso

// números simples são considerados inteiros
System.out.println(1); // int

// números com casa decimal são considerados double.
// Também podemos colocar uma letra "D" ou "d" no final
System.out.println(1.0); //double
```

```
System.out.println(1.0D); //double

// números inteiros com a letra "L" ou "l"
// no final são considerados long.
System.out.println(1L); //long

// números com casa decimal com a letra "F" ou "f"
// no final são considerados float.
System.out.println(1.0F); //float
```

COPIAR CÓDIGO

Bases diferentes

No caso dos números inteiros, podemos declarar usando bases diferentes. O Java suporta a base **decimal** e mais as bases **octal**, **hexadecimal** e **binária**.

Um número na base octal tem que começar com um zero à esquerda e pode usar apenas os algarismos de 0 a 7:

```
int i = 0761; // base octal

System.out.println(i); // saída: 497
```

COPIAR CÓDIGO

E na hexadecimal, começa com `0x` ou `0X` e usa os algarismos de 0 a 15. Como não existe um algarismo "15", usamos letras para representar algarismos de "10" a "15", no caso, "A" a "F", maiúsculas ou minúsculas:

```
int j = 0xAB3400; // base hexadecimal
System.out.println(j); // saída: 11219968
```

COPIAR CÓDIGO

Já na base binária, começamos com `0b`, e só podemos usar "0" e "1":

```
int b = 0b100001011; // base binária
System.out.println(b); // saída: 267
```

COPIAR CÓDIGO

Não é necessário aprender a fazer a conversão entre as diferentes bases e a decimal. Apenas saber quais são os valores possíveis em cada base, para identificar erros de compilação como o que segue:

```
int i = 0769; // erro, base octal não pode usar 9
```

COPIAR CÓDIGO

Notação científica

Ao declarar `doubles` ou `floats`, podemos usar a notação científica:

```
double d = 3.1E2;
System.out.println(d); // 310.0

float e = 2e3f;
System.out.println(e); // 2000.0
```

```
float f = 1E4F;
System.out.println(f); // 10000.0
```

COPIAR CÓDIGO

Usando underlines em literais

A partir do Java 7, existe a possibilidade de usarmos *underlines* (_) quando estamos declarando literais para facilitar a leitura do código:

```
int a = 123_456_789;
```

COPIAR CÓDIGO

Existem algumas regras sobre onde esses *underlines* podem ser posicionados nos literais, e caso sejam colocados em locais errados resultam em erros de compilação. A regra básica é que eles só podem ser posicionados com **valores numéricos em ambos os lados**. Vamos ver alguns exemplos:

```
int v1 = 0_100_267_760;         // ok
int v2 = 0_x_4_13;              // erro, _ antes e depois do x
int v3 = 0b_x10_BA_75;          // erro, _ depois do b
int v4 = 0b_10000_10_11;      // erro, _ depois do b
int v5 = 0xa10_AF_75;          // ok, apesar de ser letra
                              // representa dígito
int v6 = _123_341;              // erro, inicia com _
int v7 = 123_432_;              // erro, termina com _
int v8 = 0x1_0A0_11;          // ok
int v9 = 144__21_12;          // ok
```

COPIAR CÓDIGO

A mesma regra se aplica a números de ponto flutuante:

```
double d1 = 345.45_e3;          // erro, _ antes do e
double d2 = 345.45e_3;          // erro, _ depois do e
double d3 = 345.4_5e3;          // ok
double d4 = 34_5.45e3_2;     // ok
double d5 = 3_4_5.4_5e3;     // ok
double d6 = 345._45F;          // erro, _ depois do .
double d7 = 345_.45;         // erro, _ antes do .
double d8 = 345.45_F;          // erro, _ antes do indicador de
                             // float
double d9 = 345.45_d;          // erro, _ antes do indicador de
                             // double
```

COPIAR CÓDIGO

Iniciando chars

Os chars são iniciados colocando o caractere desejado entre **aspas simples**:

```
char c = 'A';
```

COPIAR CÓDIGO

Mas podemos iniciar com números também. Neste caso, o número representa a posição do caractere na tabela unicode:

```
char c = 65;
System.out.println(c); // imprime A
```

COPIAR CÓDIGO

Não é necessário decorar a tabela unicode, mas é preciso prestar atenção a pegadinhas como a seguinte:

```
char sete = 7; // número, pois não está entre aspas simples
System.out.println(sete);  // Não imprime nada!!!!
```

COPIAR CÓDIGO

Quando usando programas em outras línguas, às vezes queremos usar caracteres unicode, mas não temos um teclado com tais teclas (árabe, chinês etc.). Neste caso, podemos usar uma representação literal de um caractere unicode em nosso código, iniciando o `char` com `\u` :

```
char c = '\u03A9'; // unicode
System.out.println(c); // imprime a letra grega ômega
```

COPIAR CÓDIGO

Identificadores

Quando escrevemos nossos programas, usamos basicamente dois tipos de termos para compor nosso código: identificadores e palavras reservadas.

Chamamos de **identificadores** as palavras definidas pelo programador para nomear variáveis, métodos, construtores, classes, interfaces etc.

Já **palavras reservadas** ou **palavras-chave** são termos predefinidos da linguagem que podemos usar para definir comandos (`if` , `for` , `class` , entre outras).

São diversas palavras-chave na linguagem java:

- abstract
- assert
- boolean
- break
- byte
- case
- catch
- char
- class
- const
- continue
- default
- do
- double
- else
- enum
- extends
- false
- final
- finally
- float
- for
- goto
- if
- implements
- import

- instanceof
- int
- interface
- long
- native
- new
- null
- package
- private
- protected
- public
- return
- short
- static
- strictfp
- super
- switch
- synchronized
- this
- throw
- throws
- transient
- true
- try
- void
- volatile
- while

null, false e true

Outras três palavras reservadas que não aparecem nessa lista são `true`, `false` e `null`. Mas, segundo a especificação na linguagem Java, esses três termos são considerados *literais* e não palavras-chave (embora também sejam reservadas), totalizando 53 palavras reservadas.

(http://java.sun.com/docs/books/tutorial/java/nutsandbolts/_keywords.html)http://java.
(http://java.sun.com/docs/books/tutorial/java/nutsandbolts/_keywords.html)

Identificadores válidos devem seguir as seguintes regras:

- Não podem ser igual a uma palavra-chave;
- Podem usar letras (unicode), números, `$` e `_`;
- O primeiro caractere **não** pode ser um número;
- Podem possuir qualquer número de caracteres.

Os identificadores são *case sensitive*, ou seja, respeitam maiúsculas e minúsculas:

```
int umNome; // ok
int umnome;    // ok, diferente do anterior
int _num;    // ok
int $_ab_c; // ok
int x_y;    // ok
int false;    // inválido, palavra reservada
int x-y;     // inválido, traço
int 4num;     // inválido, começa com número
int av#f;     // inválido, #
int num.spc; // inválido, ponto no meio
```

COPIAR CÓDIGO