

Linguagem Java

Variáveis e tipos primitivos

Arrays

Operadores

Controlo de fluxo

<https://docs.oracle.com/javase/tutorial/java/nutsandbolts>

Tipos de dados primitivos

- Existem oito tipos de dados primitivos suportados pela linguagem Java:
 - byte (8 *bits*): -128..127
 - short (16 *bits*): -32768..32767
 - int (32 *bits*): $-2^{31}..2^{31}-1$
 - long (64 *bits*): $-2^{63}..2^{63}-1$
 - float (IEEE754 32 *bits*)
 - double (IEEE754 64 *bits*)
 - boolean (*true* ou *false* => 1 *bit*)
 - char (16 *bits*): *Unicode*

Data Type	Default Value (for fields)
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	'\u0000'
String (or any object)	null
boolean	false

Variáveis

- Permitem o armazenamento de informação tipificada ou referências para objetos
- A declaração de uma variável simples é realizada da seguinte forma:

<tipo> <nome_da_variável> [= <valor_por_omissão>]

- O nome da variável pode ser um conjunto de caracteres maiúsculos ou minúsculos, números ou os caracteres ‘_’ ou ‘\$’ (este não deve ser usado!)
 - Não deve ser iniciada por um número

Exemplo de variáveis tipificadas

```
// The number 26, in decimal
int decVal = 26;
// The number 26, in hexadecimal
int hexVal = 0x1a;
// The number 26, in binary
int binVal = 0b11010;

double d1 = 123.4;
// same value as d1, but in scientific notation
double d2 = 1.234e2;
float f1 = 123.4f;

char c = 'A';
```

- Podem ser usados '_' para melhorar a legibilidade de números:

```
long creditCardNumber = 1234_5678_9012_3456L;
long socialSecurityNumber = 999_99_9999L;
float pi = 3.14_15F;
long hexBytes = 0xFF_EC_DE_5E;
long hexWords = 0xCAFE_BABE;
long maxLong = 0x7fff_ffff_ffff_ffffL;
byte nybbles = 0b0010_0101;
long bytes = 0b11010010_01101001_10010100_10010010;
```

... mas não são permitidos os seguintes casos:

```
// Invalid: cannot put underscores
// adjacent to a decimal point
float pi1 = 3_.1415F;
// Invalid: cannot put underscores
// adjacent to a decimal point
float pi2 = 3._1415F;
// Invalid: cannot put underscores
// prior to an L suffix
long socialSecurityNumber1 = 999_99_9999_L;

// OK (decimal literal)
int x1 = 5_2;
// Invalid: cannot put underscores
// At the end of a literal
int x2 = 52_;
// OK (decimal literal)
int x3 = 5_____2;

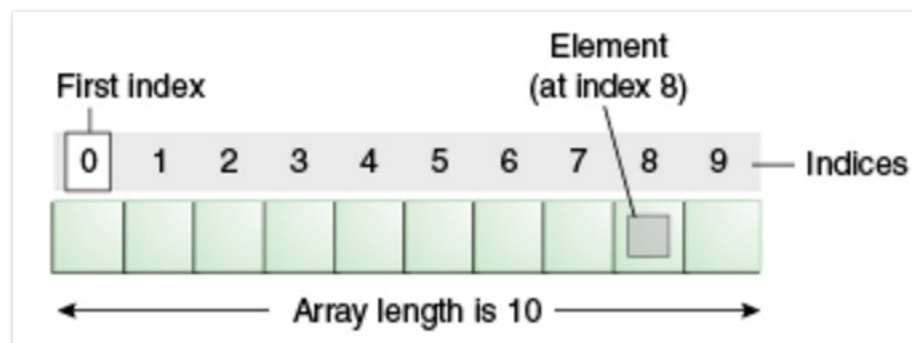
// Invalid: cannot put underscores
// in the 0x radix prefix
int x4 = 0_x52;
// Invalid: cannot put underscores
// at the beginning of a number
int x5 = 0x_52;
// OK (hexadecimal literal)
int x6 = 0x5_2;
// Invalid: cannot put underscores
// at the end of a number
int x7 = 0x52_;
```

Classes *wrapper*

- Sendo a linguagem Java totalmente orientada a objetos, todos os dados primitivos têm um equivalente definido através de um tipo de classe de objetos
 - Estas classes fornecem algumas funcionalidades que poderão ser úteis, por exemplo, para realizar conversões
- Classes equivalentes:
 - `byte` ⇔ `Byte`
 - `short` ⇔ `Short`, `int` ⇔ `Integer`, `long` ⇔ `Long`
 - `float` ⇔ `Float`, `double` ⇔ `Double`
 - `boolean` ⇔ `Boolean`
 - `char` ⇔ `Character`

Arrays

- Permitem o armazenamento de múltiplos valores
 - O número máximo de elementos é definido aquando da sua criação
 - O *array* possui um nome, correspondente ao nome da variável que o suporta
 - Cada valor é acedido através da variável do *array* e do respetivo índice (0.. $n-1$)



Declaração de Arrays

- A declaração pode ser realizada usando os seguinte formatos
 - *<tipo> [] <nome>*
 - *<tipo> <nome> []*
- Exemplo:

```
int [ ] idades;  
float meses [ ];
```

Criação de arrays

- Quando são apenas declarados, os *arrays* possuem o valor `null`, correspondendo à não definição do *array*
- Para criar (definir) o *array* é necessário criar o número de células/valores pretendidos para o mesmo usando a palavra chave `new`

```
int [ ] idades;
```

```
idades = new int[10];
```

```
float [ ] meses = new float[12];
```


Exemplos de utilização de arrays

```
idades[0] = 10;
```

```
idades[1] = 15;
```

```
idades[2] = 20;
```

```
int i1 = idades[0] + 1;
```

```
float m = (idades[0]+idades[1])/2.0;
```

Iniciação de arrays

- Quando um *array* é criado é possível iniciá-lo com valores por omissão

```
int [ ] tab1 = { 10, 20, 30, 40, 50 };
```

```
char [ ] tab2 = { 'a', 'b', 'c' };
```

Arrays multidimensionais

- Em Java são permitidos arrays de múltiplas dimensões, os quais são declarados com a inserção de pares de [] por cada dimensão

```
int [ ][ ] array1 = new int [5][2];
```

```
double [ ][ ][ ] array2 = new double [5][4][3];
```

- O acesso é realizado da seguinte forma

```
array2[0][1][2] = array2[2][1][0] * 3.1415;
```

Arrays multidimensionais

- Os arrays poderão ter um número de elementos variável por linha, nesse caso a criação de cada linha deve ser realizada de forma independente

```
int [][] b = new int[3][];
```

```
b[0] = new int[3];
```

```
b[1] = new int[4];
```

```
b[2] = new int[2];
```

Iteração sobre arrays

- Obtenção de informação sobre o número de elementos de um array
 - Assumindo `int [][] b = ...`
 - `b.length` – número de linhas do *array* `b`
 - `b[i].length` – número de elementos da linha `i`
- Exemplo:

```
for ( int i=0 ; i < b.length ; i++)  
    for ( int j=0 ; j< b[i].length ; j++){  
        // ... b[i][j] ...  
    }
```

Operadores

- Os operadores são similares aos existentes noutras linguagens (C/C++,C#,...)

Operator Precedence	
Operators	Precedence
postfix	<i>expr</i> ++ <i>expr</i> --
unary	++ <i>expr</i> -- <i>expr</i> + <i>expr</i> - <i>expr</i> ~ !
multiplicative	* / %
additive	+ -
shift	<< >> >>>
relational	< > <= >= instanceof
equality	== !=
bitwise AND	&
bitwise exclusive OR	^
bitwise inclusive OR	
logical AND	&&
logical OR	
ternary	? :
assignment	= += -= *= /= %= &= ^= = <<= >>= >>>=

Controlo de fluxo – if

- *if-then*

```
if (condição) {  
    ...  
}
```

- *if-then-else*

```
if (condição) {  
    ...  
} else {  
    ...  
}
```

```
if (condição1) {  
    ...  
} else if (condição2) {  
    ...  
} else if (condição3) {  
    ...  
} else {  
    ...  
}
```

Controlo de fluxo – switch, break

- *switch-case*

```
switch (source) {  
    case op1:  
    [case op2:]  
        ...  
        break;  
    ...  
    default:  
        ...  
        break;  
}
```

- *switch-arrow case*

```
switch (source) {  
    case op1 -> ... ;  
    case op2 -> ... ;  
    ...  
    default -> ... ;  
}
```

- Não são necessários *breaks*

- Podem ser usadas *strings* nos *cases*

Controlo de fluxo – ciclos

- *for*

```
for(iniciacão; condição; evolução) {  
    ...  
}
```

- *for-each*

```
for(<type> <var>:<coleção/array>) {  
    ...  
}
```

- *while*

```
while (cond) {  
    ...  
}
```

- *do-while*

```
do {  
    ...  
} while (cond);
```

Controlo de fluxo

- Outras instruções de controlo de fluxo na execução de um programa Java
 - `continue`
 - `break`
 - O *continue* e o *break* podem ser seguidas por uma *label* associada previamente ao ciclo em que devem atuar

```
ciclo1: for(int i=0;i<10;i++)  
    for(int j=10;j>0;j--)  
        if (i == j) break ciclo1;
```
- `return`

Interação com o utilizador

- O Java disponibiliza um conjunto de objetos que permitem representar as entradas e saídas típicas de um programa/processo
 - `System.in` – representa a entrada de dados por omissão, normalmente associada à entrada de dados da consola
 - `System.out` – representa a saída de dados por omissão, normalmente associada à saída de dados para a consola
 - `System.err` – representa a saída para erros, normalmente associada à saída de dados para a consola

Saída de dados

- `System.out.print(...)`
- `System.out.println(...)`
- `System.out.printf(<format>, param1, param2, ...)`
- `System.err.print(...)`
- `System.err.println(...)`
- `System.err.printf(<format>, param1, param2, ...)`

Entrada de dados

- Embora o objeto `System.in` permita obter dados introduzidos pelo utilizador na consola, esses dados são interpretados como *bytes*
- Para facilitar o acesso a essa informação, de forma mais facilitada e tipificada, pode-se usar um objeto `Scanner`
`Scanner sc = new Scanner(System.in);`
- Um objeto `Scanner` permite ler sequências de caracteres separados por delimitadores.
`int n = sc.nextInt(); // Lê um inteiro`
`double x = sc.nextDouble() ; // Lê um double`
- Por omissão, os delimitadores são os espaços em branco, *tabs* e mudanças de linha. No entanto pode ser definido outro delimitador:
`sc.useDelimiter("-");`
- É possível testar o tipo do próximo valor a ser lido:
`if(sc.hasNextDouble()) // Verifica se próximo valor é um double`
`x = sc.nextDouble() ;`

Classe Math

- A classe Math, pertencente ao package `java.lang`, disponibiliza um conjunto de funções matemáticas
 - `sin`, `cos`, `tan`, `abs`, `max`, `min`, `round`, `pow`, ...
- disponibiliza também um método que gera números pseudo-aleatórios: `random`
 - `double r = Math.random();`
 - $0.0 \leq r < 1.0$
 - Exemplo para gerar números entre 1 e 100, inclusive
 - `int i = (int) (Math.random() * 100) + 1;`

Classe Random

- No *package* `java.util` é disponibilizada uma classe `Random` a qual possui um conjunto de métodos que permitem um acesso mais flexível a sequências de números pseudo-aleatórios

```
Random rnd = new Random();  
int i = rnd.nextInt();  
int j = rnd.nextInt(100) + 1;  
double d = rnd.nextDouble();  
...  
rnd.setSeed(1234);  
...
```

Exercício

- Resolução do exercício 2 da ficha

Classes

- Para definir uma nova classe de objetos usa-se a seguinte sintaxe:

```
class <nome> {  
    <variáveis>  
    <métodos/funções>  
}
```

- Não é obrigatório as variáveis serem definidas todas no início ou os métodos no fim, mas é uma boa política para a sua organização

- Exemplo:

```
class Ponto {  
    int x,y;  
  
    void move(int dx,int dy) {  
        x += dx;  
        y += dy;  
    }  
}
```

Criação de objetos

- Para criar um objeto de uma classe usa-se a instrução `new`
`Ponto p = new Ponto();`
- Sempre que um objeto é criado, é chamado automaticamente um construtor para fazer a iniciação desse objeto
 - Os construtores são métodos com o mesmo nome da classe e sem tipo de retorno declarado
 - Podem existir vários construtores, com diferentes parâmetros
 - Um construtor sem parâmetros é designado "construtor por omissão"
 - Em Java as variáveis não iniciadas explicitamente são automaticamente iniciadas com valores por omissão (0 ou null)

```
public class Ponto {  
    int x,y;  
  
    public Ponto(int xi, int yi) {  
        x = xi;  
        y = yi;  
    }  
}
```

Exercício

- Resolução do exercício 4 da ficha