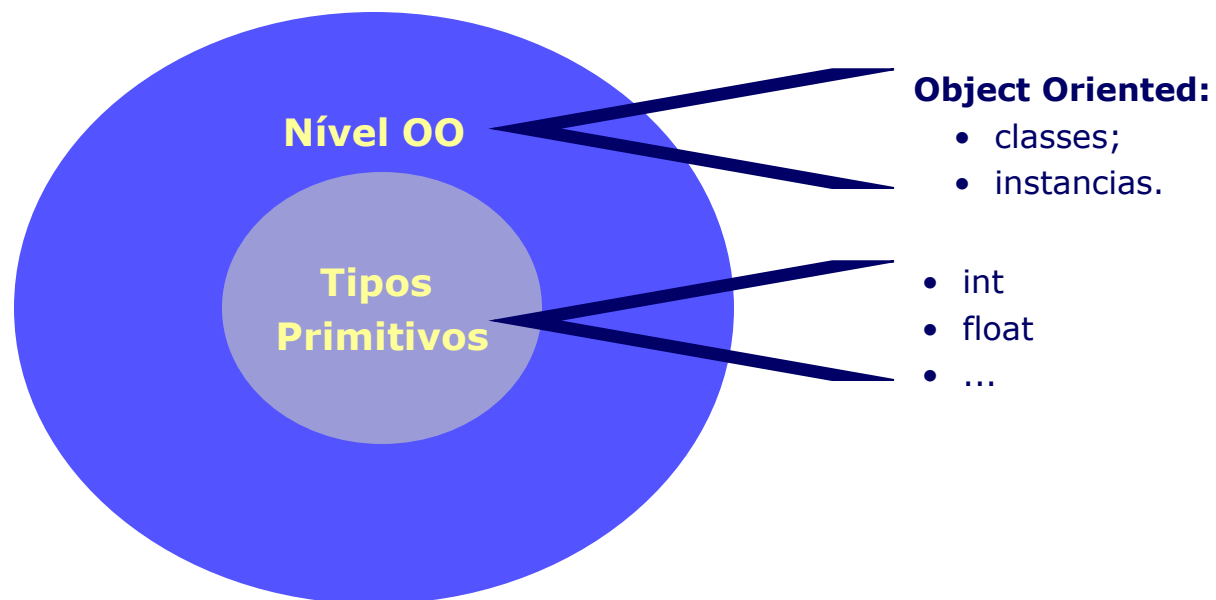


PG II

Programação Orientada aos Objectos em Java

Tipos básicos:

- Nível das Entidades
- Tipos Primitivos
- Constantes
- Coerção (*casting*) de Tipos Primitivos
- Casting
- Vírgula flutuante: IEEE 754 FP - Float
- Vírgula flutuante: IEEE 754 FP - Double
- Tipos Referenciados
- Array
- String
- Array de Strings como parâmetro
- pg2.io.In e pg2.io.Out

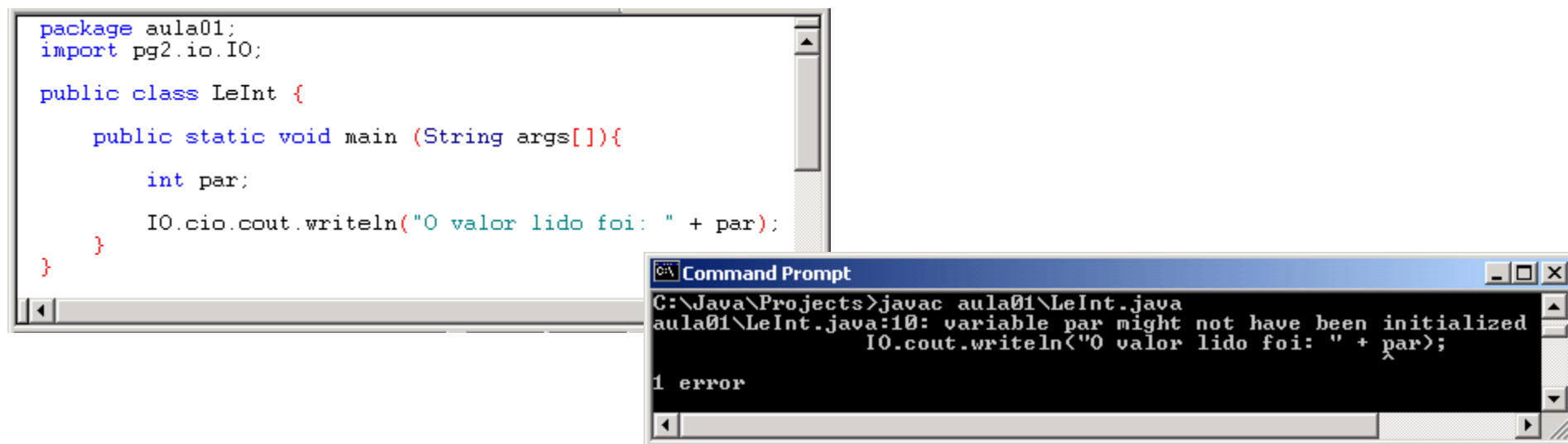


Tipos Primitivos

Declaração de **variáveis** com e sem inicialização:

id_tipo id_variável [= valor] [,id_variável [= valor]...];

“... a qualquer variável **local** deve ser atribuído um valor de forma explícita antes de ser utilizada, seja por inicialização ou por atribuição, de uma forma verificável para o compilador, caso contrário este indicará um erro de sintaxe.”



```

package aula01;
import pg2.io.IO;

public class LeInt {

    public static void main (String args[]){

        int par;

        IO.cio.cout.writeln("O valor lido foi: " + par);

    }
}

```

```

C:\Java\Projects>javac aula01\LeInt.java
aula01\LeInt.java:10: variable par might not have been initialized
        IO.cout.writeln("O valor lido foi: " + par);
                                   ^
1 error

```

Constantes

A declaração de constante é semelhante à declaração de variáveis, mas requer a inclusão do qualificador ***final***, que especifica que tal inicialização é imutável:


```
final double PI = 3.14159273269;
```

As constantes estão assim associadas a variáveis cujo valor não tem significado alterar. Tratam-se assim de variáveis usadas apenas para consulta do seu valor, “*readonly*”.

```
public class Circulo {  
    static final double PI = 3.14159273269;  
    private int raio;  
    public Circulo(int r) {  
        raio = r;  
    }  
    public double perimetro() {  
        return 2*PI*raio;  
    }  
    public double area() {  
        return PI*raio*raio;  
    }  
}
```

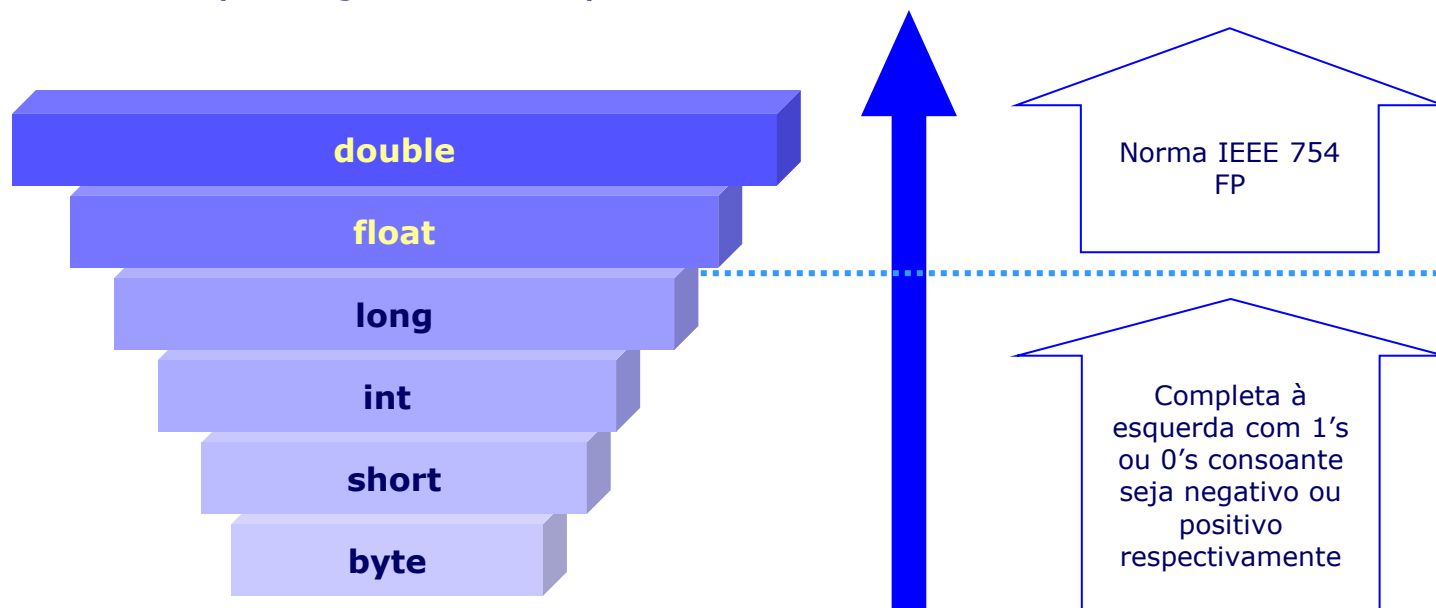
... Tipos Primitivos



Tipo	Valores	Default	Bits	Gama
boolean	true, false	false	1	{ true; false }
char 	Caracteres Unicode 3.2 (38.885 caracteres) Primeiros 128 caracteres iguais aos ASCII <ul style="list-style-type: none"> char var_char = 'A'; ASCII (256 caractéres): char var_char = 65; Unicode 3.2: char var_char = '\u0065'; (e.g. € = \u20AC)	\u0000	16	\u0000 a \uFFFF
byte	Inteiro com sinal: <ul style="list-style-type: none"> byte b1 = 0x49; Byte b1 = 73; 	0	8	[-128; 127]
short	Inteiro com sinal	0	16	[-32.768; 32.767]
int	Inteiro com sinal	0	32	[-2.147.483.648; 2.147.483.647]
long	Inteiro com sinal	0	64	$\sim -1 \text{ E } +20 \text{ a } \sim 1 \text{ E } + 20$
float	Vírgula flutuante: IEEE 754 FP	0,0	32	$\sim \pm 3,4 \text{ E } +38 \text{ a}$ $\sim \pm 1,4 \text{ E } -45$
double	Vírgula flutuante: IEEE 754 FP	0,0	64	$\sim \pm 1,8 \text{ E } +308 \text{ a}$ $\sim \pm 5 \text{ E } -324$

Coerção de Tipos Primitivos

Coerção de tipos automática (*casting* automático)



```
package aula01;
import pg2.io.IO;
public class Teste1{
    public static void main(String [] args){
        short var_sh = 480;          // 0000 0001 1110 0000
        byte var_byte = var_sh;      // 1110 0000
        IO.cout.writeln("O valor short:" + var_sh+ " ...\n... convertido para byte: " + var_byte);
    }
}
```

```
C:\ Command Prompt
D:\work>javac aula01\Teste1.java
aula01\Teste1.java:6: possible loss of precision
found   : short
required: byte
    byte var_byte = var_sh;    // 1110 0000
                   ^
1 error
```

No sentido contrário a coerção de tipos não é tão linear.

Operador de casting em Java: (<id_tipo>)

<<...

```
double var_db = 56.78;
```

```
float var_fl = (float) var_db;
```

...>>

Quando forçamos o casting o que acontece?

<<...

```
short var_sh = 480; // 0000 0001 1110 0000
```

```
byte var_byte = (byte) var_sh; // 1110 0000
```

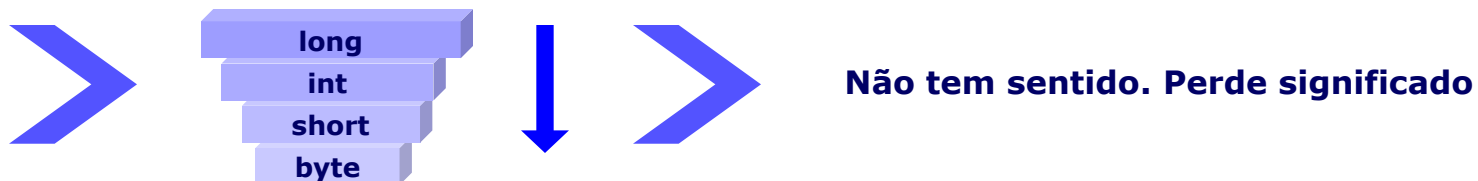
```
IO.cout.writeln("O valor short:" + var_sh+ " ...\n... convertido para byte: " + var_byte);
```

...>>

```
C:\Java\Projects>java aula01.Teste1
```

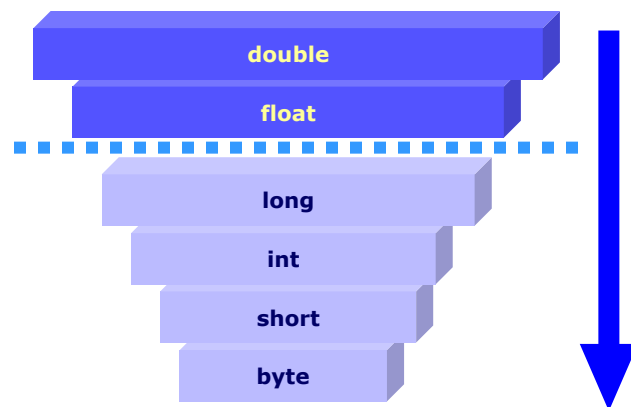
```
O valor short: 480 ...
```

```
... convertido para byte: -32
```



... Casting

E de números reais para inteiros ?



> Retira as casas decimais e

```

C:\ Command Prompt
D:\work>java aula01.Teste3
0 float:480.938 ...
... convertido para byte: -32
D:\work>

```

```

C:\ Command Prompt
D:\work>java aula01.Teste3
0 double:480.62395649560233 ...
... convertido para byte: -32
D:\work>

```

(passa para uma representação de inteiro com sinal e corta os bits à esquerda consoante o tipo da conversão)

Vírgula flutuante: IEEE 754 FP - **Float**

Float



> F são potências negativas a ler da esquerda para a direita

- $V = (-1)^S * 2^{(E-127)} * (1 + F)$ com $0 < E < 255$ e $1 \leq 1, F < 2$

- if (E == 255 && F != 0) then V = NaN ("Not a number");
- if (E == 255 && F == 0 && S == 0) then V = infinity;
- if (E == 255 && F == 0 && S == 1) then V = - infinity;
- if (E == 0 && F == 0 && S == 0) then V = 0;
- if (E == 0 && F == 0 && S == 1) then V = -0;
- if (E == 0 && F != 0) then V isUnnormalizedValue = true

$$V = (-1)^S * 2^{(-126)} * (F)$$

... Vírgula flutuante: IEEE 754 FP - **Float**

$$\begin{aligned}
 \text{O maior float} &= 0 \underbrace{1111\ 1110}_{\text{exponent}} \underbrace{1111\ 1111\ 1111\ 1111\ 1111\ 1111}_{\text{mantissa}} \\
 &= (-1)^0 * 2^{(254-127)} * (1,9999999881) \\
 &= 3,40282E+38
 \end{aligned}$$

$$\begin{aligned}
 \text{O número mais próximo de zero} &= 0\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000\ 001 \\
 &= (-1)^0 * 2^{(-126)} * (2^{-23}) \\
 &= 2^{-149} \\
 &= 1,4013E-45
 \end{aligned}$$

Exemplo:

$$\begin{aligned}
 6,5 &= 2 * 3,25 \\
 6,5 &= 2^2 * 1,625 \quad 1 \leq 1,625 < 2 \\
 &\quad \downarrow \\
 &2 = E - 127 \Leftrightarrow E = 129 \qquad \qquad \qquad 0,625 = 2^{-1} + 2^{-3} = 0,5 + 0,125 \\
 &\qquad \qquad \qquad \underbrace{0\ 1000\ 0001}_{\text{exponent}} \underbrace{1010\ 0000\ 0000\ 0000\ 0000\ 0000}_{\text{mantissa}}
 \end{aligned}$$

Vírgula flutuante: IEEE 754 FP - **Double**

Double

..... 3 2 1 0 -1 -2 -3 -4
 S E.....E F.....F > F são potências negativas a ler da esquerda para a direita
 0 1.....11 12.....63

$$V = (-1)^S * 2^{(E-1023)} * (1 + F) \text{ com } 0 < E < 2047 \text{ e } 1 \leq 1, F < 2$$

- if (E == 2047 && F != 0) then V = NaN ("Not a number");
- if (E == 2047 && F == 0 && S == 0) then V = infinity;
- if (E == 2047 && F == 0 && S == 1) then V = - infinity;
- if (E == 0 && F == 0 && S == 0) then V = 0;
- if (E == 0 && F == 0 && S == 1) then V = -0;
- if (E == 0 && F != 0) then V isUnnormalizedValue = true

$$V = (-1)^S * 2^{(-1022)} * (F)$$

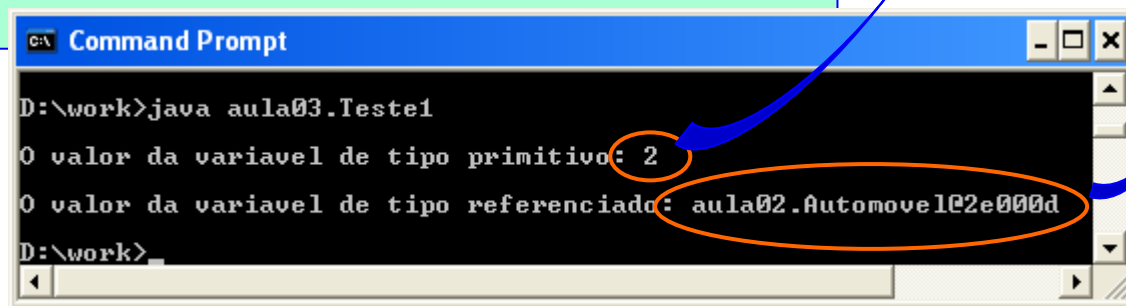
Tipos Referenciados

Por que se designam de tipos **referenciados**?

- Variáveis de tipos **primitivos**, guardam directamente os seus valores ("*value types*");
- Variáveis de tipos **referenciados**, guardam uma referência para o endereço de memória onde está armazenado o seu valor efectivo ("*reference type*").

```
package aula03;
import aula02.Automovel;
import pg2.io.IO;

public class Teste1{
    public static void main (String args[]) {
        int var_prim = 2;
        Automovel var_obj = new Automovel(1500, 120, 200);
        IO.cout.writeln("\nO valor da variavel de tipo primitivo: " + var_prim);
        IO.cout.writeln("\nO valor da variavel de tipo referenciado: " + var_obj);
    }
}
```

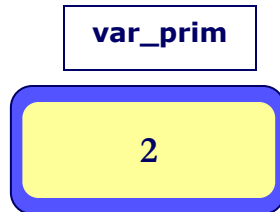


```
C:\> Command Prompt
D:\work>java aula03.Teste1
O valor da variavel de tipo primitivo: 2
O valor da variavel de tipo referenciado: aula02.Automovel@2e0000d
D:\work>
```

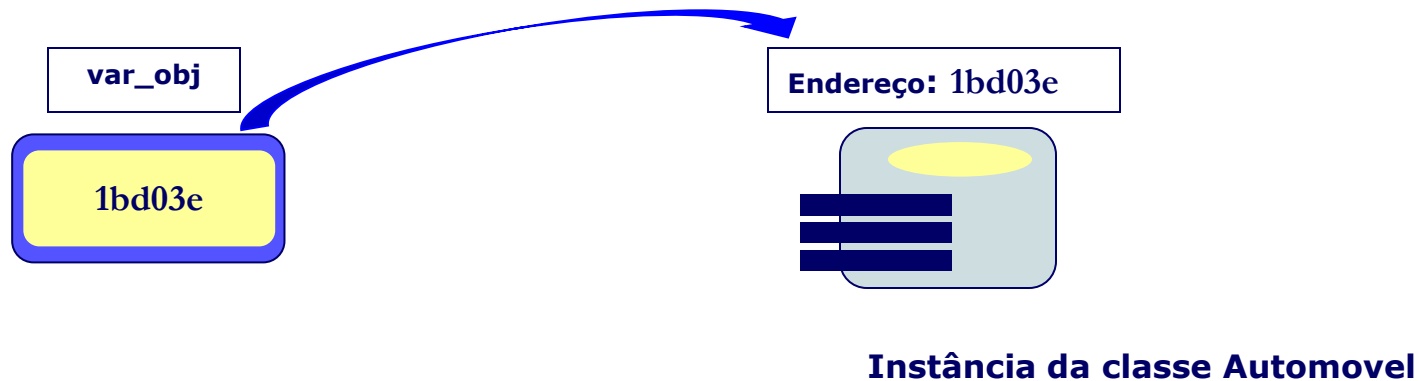
Valor

Referência

Variável "var_prim" do tipo "int", só poderá guardar valores do tipo "int".



Variável "var_obj" do tipo "*Automovel*", só poderá referenciar instâncias da classe Ponto.



... Tipos Referenciados

Uma variável de um tipo referenciado, que seja global (variável de instância ou de classe) e não seja iniciada, ficará com o valor *default*: null.

```
package aula03;
import aula02.Automovel;
import pg2.io.IO;

public class FrotaAutomovel{

    Automovel carro, camiao, tractor;

    public static void main (String args[]) {
        FrotaAutomovel frota = new FrotaAutomovel();
        IO.cout.writeln("\nValor da variavel de instancia carro: " + frota.carro);
    }
}
```

carro

null

C:\ Command Prompt

```
D:\work>java aula03.FrotaAutomovel
Valor da variavel de instancia carro: null
D:\work>_
```

... Tipos Referenciados

Quando manipulamos directamente variáveis de tipos referenciados, estamos a afectar apenas os endereços referenciados por essas variáveis.

```
package aula03;
import aula02.Automovel;
import pg2.io.IO;

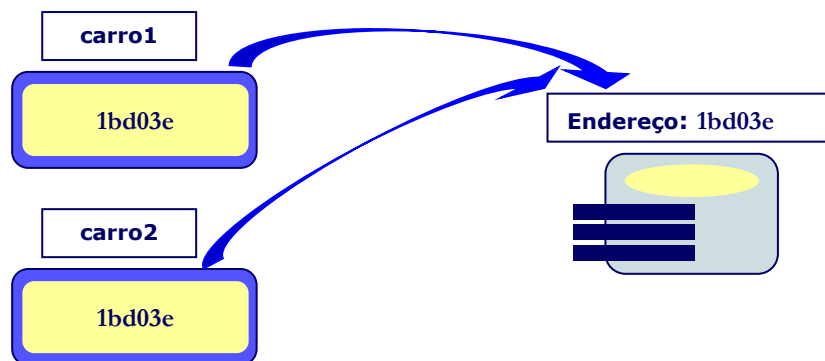
public class Teste2{
    public static void main (String args[]) {
        Automovel carro1 = new Automovel(1500, 120, 200);
        Automovel carro2 = carro1;
        carro2.setMudanca(2);
        pg2.io.IO.cout.writeln("\n0 carro1 esta em " + carro1.getMudanca() + "a e tem a velocidade maxima de: " +
                                carro1.velocidadeMax() + "kms/h");
        pg2.io.IO.cout.writeln("\n0 carro2 esta em " + carro2.getMudanca() + "a e tem a velocidade maxima de: " +
                                carro2.velocidadeMax() + "kms/h");
    }
}
```

Command Prompt

```
D:\work>java aula03.Teste2

0 carro1 esta em 2a e tem a velocidade maxima de: 88.0kms/h
0 carro2 esta em 2a e tem a velocidade maxima de: 88.0kms/h

D:\work>
```



Logo, fazer:

`carro2.setMudanca(2);`

é igual a fazer:

`carro1.setMudanca(2);`

Array

São entidades referenciadas mas não são objectos, uma vez que não são criados a partir de uma classe.

Exemplos:

```
char var_arr []; //declaração de uma variável lista que referencia uma array de caracteres
```

```
char [] var_arr; //declaração semelhante à anterior
```

```
char [] var_arr = {'I', 'S', 'E', 'L'}; //declaração e inicialização
```

var_arr



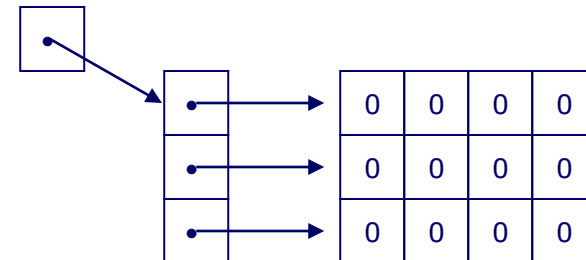
```
int var_arr [] = new int[4]; //declaração e "alocação" de 4 inteiros
```

var_arr



```
int var_arr [][] = new int[3][4];
```

var_arr



Nota: Java não obriga à especificação da segunda dimensão.

... Array

Conclusões:

- Depois de instanciado podemos saber a sua dimensão consultando o atributo *readonly* **length**.

- A declaração de um array inicializado:

```
int[] a = { 10, 20, 30 };
```

é equivalente a:

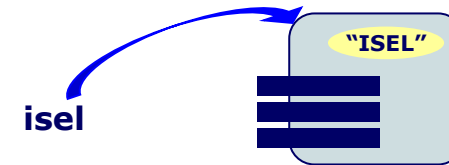
```
int[] a = new int[3];
```

```
a[0] = 10; a[1] = 20; a[2] = 30;
```

O que é uma String ?

- Uma String **não** é um array de caracteres;
- Uma String é uma **classe** vulgar (java.lang.String) que usa internamente um **array de caracteres**:

```
char[] aux = { 'I', 'S', 'E', 'L' };  
String isel = new String(aux);
```



- A linguagem JAVA tem algumas características específicas para facilitar o uso de objectos desta classe:

```
String isel = "ISEL";  
String isel = new String ("ISEL");
```

- Os métodos mais usados desta classe são:
 - `int length()` – o número de caracteres.
 - `char charAt(int)` – o carácter na posição indicada.
 - `String.valueOf(tipoPrimitivo)` – tipo Primitivo para String.

O conteúdo de um objecto String não pode ser alterado.

Conclusões:

- A conversão de um tipo primitivo para String, pode sempre ser realizada chamando `String.valueOf(prim)`.
- O operador '+' entre duas Strings faz a concatenação. Caso um dos argumentos não seja uma String, este é implicitamente convertido para String antes da concatenação:

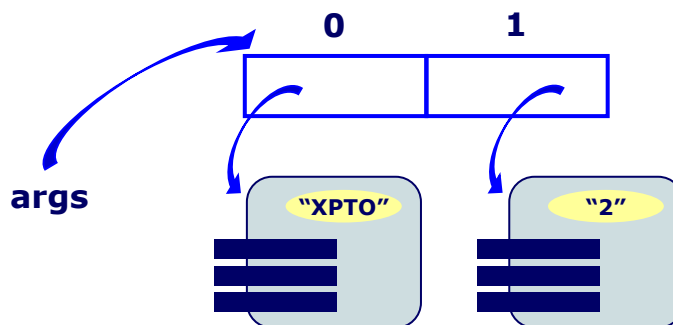
"ISEL " + 2002	⇔	
"ISEL " + String.valueOf(2002)	⇔	
"ISEL " + "2002"	⇔	"ISEL 2002"

Array de Strings como parâmetro

```
package aula03;
import pg2.io.IO;

public class Echo{
    public static void main(String [] args){
        for(int i=0; i<args.length; i++){
            IO.cout.writeln(args[i]);
        }
    }
}
```

```
C:\ Command Prompt
D:\work>java aula03.Echo
D:\work>java aula03.Echo XPTO 2
XPTO
2
                        args[0] args[1]
D:\work>
```



Quando passado, juntamente com a “linha de comando”, uma sequência de argumentos separados por espaços, a variável **args** (parâmetro da função main) é instanciada com um array de dimensão igual ao número de argumentos.

Para cada um destes argumentos é instanciada uma String, cuja referência é guardada no array **args** na posição correspondente à ordem em que este foi passado.

... Array de Strings como parâmetro

Alguns dos exemplos apresentados e algumas das aplicações solicitadas aos alunos, serão desenvolvidas no *package*, com a designação **pg2.util**. A primeira classe a introduzir neste *package* é a classe **Arrays**, que irá disponibilizar um serviço para ordenação de arrays de inteiros.

```
package pg2.util;
import pg2.io.IO;

public class Arrays{
    public static void sort(int [] a){
        for(int i=a.length-1; i>0; i--){
            for(int j=0; j<i; j++){
                if(a[j] > a[j+1]){
                    int aux = a[j];
                    a[j]=a[j+1];
                    a[j+1]=aux;
                }
            }
        }
    }

    public static void main(String [] args){
        int [] inteiros = new int[args.length];
        for(int i=0; i<args.length; i++){
            inteiros[i] = Integer.parseInt(args[i]);
        }
        Arrays.sort(inteiros);
        for(int i=0; i<inteiros.length; i++){
            IO.cout.writeln(inteiros[i]);
        }
    }
}
```

Command Prompt

```
D:\work>java pg2.util.Arrays 123 324 2 32 5 7 1323 4423 12 2124 234 23 34
2
5
7
12
23
32
34
123
234
324
1323
2124
4423
D:\work>
```

java.lang

Class Integer

static int

parseInt(String s)

Parses the string argument as a signed decimal integer.

Uma vez introduzidos os tipos primitivos, já se entende claramente qual a funcionalidade de cada um dos métodos disponibilizados pelas classes pg2.io.In e pg2.io.Out.

char readChar()	Lê um caracter e devolve um caracter em formato Unicode 3.2
int getChar()	Lê um caracter e devolve um valor inteiro correspondente ao código Unicode 3.2: Igual a fazer: <code>int var_int = ReadChar ();</code>
short readShort()	
byte readByte()	
int readInt()	
long readLong()	
float readFloat()	
double readDouble()	
java.lang.String readLine()	
java.lang.String readWord()	Lê a primeira de uma sequência de palavras inseridas.
void skipLine()	Salta a primeira linha introduzida na consola
boolean eof()	Indicação de fim de ficheiro

void dec()

void hex()

void left()

void oct()

void right()

void width(int i)

void write(byte b)

void write(char c)

void write(double d)

void write(float f)

void write(int i)

void write(long l)

void write(java.lang.Object o)

void write(short s)

void write(java.lang.String s)

void writeln()

void writeln(byte b)

void writeln(char c)

void writeln(double d)

void writeln(float f)

void writeln(int i)

void writeln(long l)

void writeln(java.lang.Object o)

void writeln(short s)

void writeln(java.lang.String s)