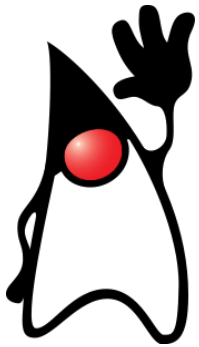
Introdução à Linguagem Java

Pedro Ribeiro

DCC/FCUP

2020/2021



(baseado e/ou inspirado parcialmente nos slides de Luís Lopes e de Fernando Silva)

Um pouco de História sobre a Linguagem Java

- desenvolvida pela Sun Microsystems
- início em 1991 (com o nome de Oak)
- disponibilizada em 1995 com o nome Java
- fundamental no desenvolvimento da Web
- passou para a Oracle em 2010
- orientada a objectos (Object Oriented/OO)

(James Gosling, autor original do Java - imagem da Wikipedia)

Versões

- JDK 1.0 (January, 1996)
- JDK 1.1 (February, 1997)
- J2SE 1.2 (December, 1998)
- J2SE 1.3 (May, 2000)
- J2SE 1.4 (February, 2002)
- J2SE 5.0 (September, 2004)
- Java SE 6 (December, 2006)
- Java SE 7 (July, 2011)
- Java SE 8 LTS (March, 2014)
- Java SE 9 (September 2017)
- Java SE 10 (March, 2018)
- **Java SE 11 LTS (September, 2018)** *(usada no Mooshak)*
- Java SE 12 (March, 2019)
- Java SE 13 (September, 2019)
- Java SE 14 (March, 2020)
- Java SE 15 (September, 2020)

Popularidade

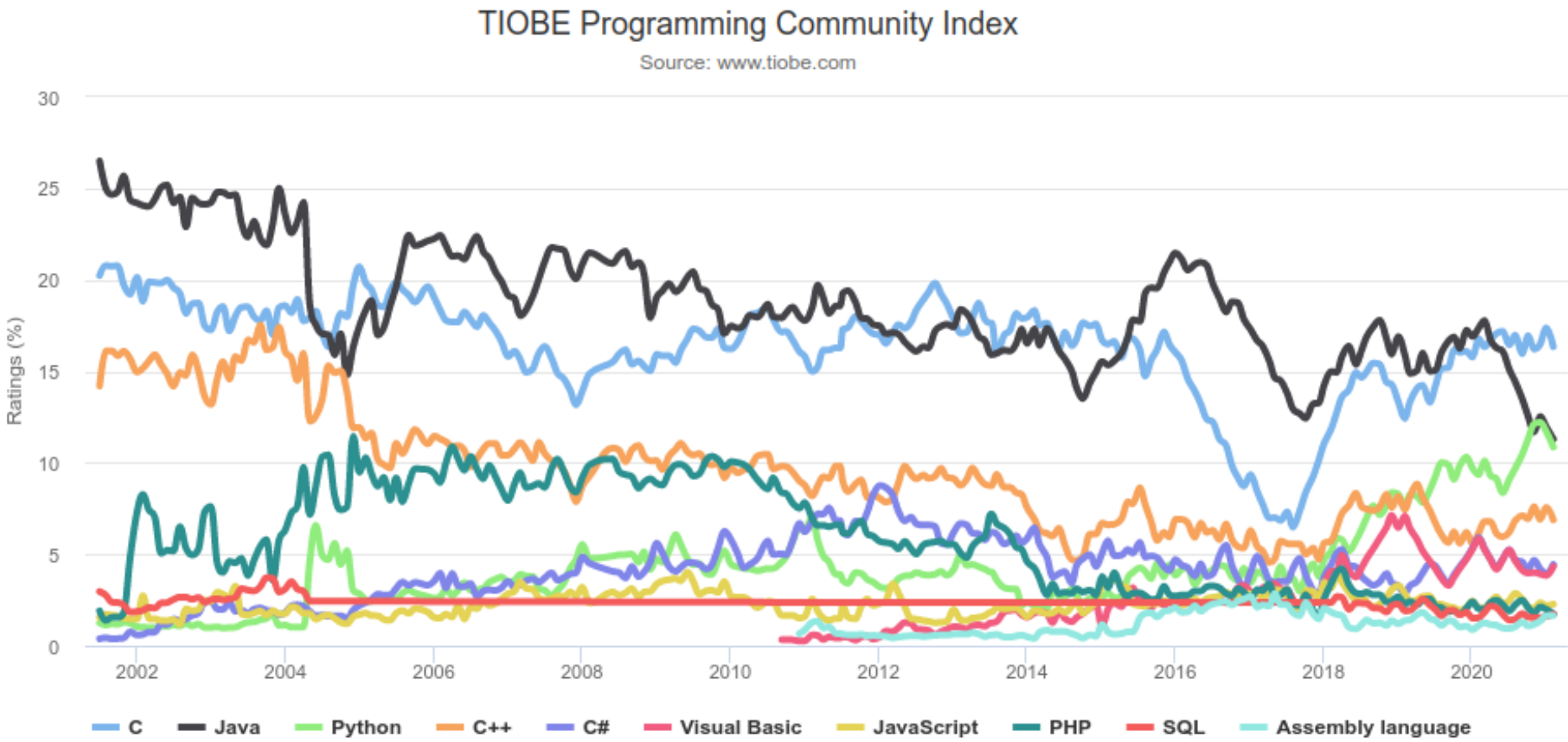
TIOBE Index: https://www.tiobe.com/tiobe-index/

Feb 2021	Feb 2020	Change	Programming Language	Ratings	Change
1	2	▲	C	16.34%	-0.43%
2	1	▼	Java	11.29%	-6.07%
3	3		Python	10.86%	+1.52%
4	4		C++	6.88%	+0.71%
5	5		C#	4.44%	-1.48%
6	6		Visual Basic	4.33%	-1.53%
7	7		JavaScript	2.27%	+0.21%
8	8		PHP	1.75%	-0.27%
9	9		SQL	1.72%	+0.20%
10	12	▲	Assembly language	1.65%	+0.54%
11	13	▲	R	1.56%	+0.55%

(Fevereiro de 2021)

Popularidade

TIOBE Index: https://www.tiobe.com/tiobe-index/



Vantagens

- **Portabilidade**
  - compilada para bytecode
  - executado por uma máquina virtual (**JVM**)
  - basta ter a JVM instalada para executar qualquer programa Java
- **Segurança e Robustez**
  - verificação de tipos estática
  - gestão de memória automática
  - excepções para tratar erros de execução
- **Ferramentas de Desenvolvimento (JDK)**
  - compilador: `javac`
  - interpretador de bytecode: `java`
  - ferramentas (`jar`, `javadoc`, ...)
  - Aplication Programming Interfaces (APIs)

Modelo de Programação

- programa é composto por um conjunto de classes
  (mais sobre classes durante as próximas aulas)
- cada classe `X` deve estar num ficheiro `X.java`
- (apenas) uma das classes deve ter o método `main()`

```
public class HelloWorld {
   public static void main(String[] args) {
     System.out.println("Hello World!");
   }
}
```

- para compilar e executar este programa na linha de comando:

```
$ javac HelloWorld.java
$ java HelloWorld
```

Comentários

- Comentários de uma linha começam com "//"
- Comentários de múltiplas linhas iniciados com "/*" e terminados com "*/"

```
/*************************************************************
   Estruturas de Dados - CC1007
   Pedro Ribeiro (DCC/FCUP)
   ---------------------------------------------------
   HelloWorld - um primeiro programa em Java
*************************************************************/

public class HelloWorld {
   public static void main(String[] args) {
      System.out.println("Hello World!"); // A minha primeira instrução
   }
}
```

Tipos de Dados

- Tipos básicos (ou primitivos):
  manipulados directamente na máquina virtual
  - inteiros (byte, short, int, long)
  - vírgula-flutuante (float, double)
  - verdadeiro/falso (boolean)
  - caracteres (char)

```
int     i = 2;
float   f = 3.192;
boolean b = true;
char    c = 'a';
```

Tipos de dados básicos

- **Inteiros**

`byte`	8 bits	$[-2^7 : 2^7 - 1]$
`short`	16 bits	$[-2^{15} : 2^{15} - 1]$
`int`	32 bits	$[-2^{31} : 2^{31} - 1]$
`long`	64 bits	$[-2^{63} : 2^{63} - 1]$

- **Vírgula flutuante (IEEE 754)**

`float`	32 bits	$[-3.4029E + 38 : +3.4029E + 38]$
`double`	64 bits	$[-1.79769E + 308 : +1.79769E + 308]$

- **Outros**

`boolean`	depende JVM	valor booleano (`true` ou `false`)
`char`	16 bits	ISO Unicode char set

Exemplo de atribuição de tipos

```
public class TestBasicTypes {
   public static void main(String[] args) {
      boolean flag = true;
      char      ch = 'A';
      byte       b = 12;
      short      s = 24;
      int        i = 257;
      long       l = 2147483648L;  // sem a letra é interpretado como int
      float      f = 3.1415f;      // sem a letra é interpretado como double
      double     d = 2.1828;
      System.out.println("flag = " + flag);
      System.out.println("ch = "   + ch);
      System.out.println("b = "    + b);
      System.out.println("s = "    + s);
      System.out.println("i = "    + i);
      System.out.println("l = "    + l);
      System.out.println("f = "    + f);
      System.out.println("d = "    + d);
   }
}
```

Conversão entre tipos

- É possível converter entre tipos "compatíveis"
  - De menor precisão para maior precisão é sempre possível
  - No sentido inverso é preciso fazer *casting* explícito

```
double d1 = 3.2;
double d2 = 3.9;

// Casting explícito
int    i1 = (int)d1;    // i1 fica com o valor de 3
int    i2 = (int)d2;    // i2 fica com o valor de 3
double d3 = (double)i2; // d3 fica com o valor de 3.0

// Casting implícito
int    i3 = 42;
double d4 = i3;         // d4 fica com o valor de 42.0
int    i4 = d4;         // erro: "possible loss of precision"

// Conversão entre char e int
char   ch1 = 'A';
int    i5 = ch1;        // i5 fica com 65 (código ascii de 'A')
char   ch2 = 66;        // ch2 fica com 'B' (código ASCII 66)

// Exemplo de tipos incompatíveis: boolean e int
boolean b1 = i1;        // erro: "incompatible types"
```

Exemplo com limites de tipos numéricos

```
public class TestLimits {
   public static void main(String args[]) {
      // inteiros
      byte   largestByte = Byte.MAX_VALUE;
      short largestShort = Short.MAX_VALUE;
      int largestInteger = Integer.MAX_VALUE;
      long   largestLong = Long.MAX_VALUE;

      // virgula flutuante
      float largestFloat = Float.MAX_VALUE;
      double largestDouble = Double.MAX_VALUE;

      // mostrar limites
      System.out.println("Largest byte value: " + largestByte);
      System.out.println("Largest short value: " + largestShort);
      System.out.println("Largest integer value: " + largestInteger);
      System.out.println("Largest long value: " + largestLong);

      System.out.println("Largest float value: " + largestFloat);
      System.out.println("Largest double value: " + largestDouble);
    }
}
```

Âmbito das variáveis

- Variáveis podem ser declaradas em qualquer ponto do código (não necessariamente no início do procedimento/função)

```
class TestDeclare {
   public static void main(String[] args) {
      int num1 = 10;
      System.out.println(num1);
      int num2 = 20; // variável pode ser declarada em qualquer ponto
      System.out.println(num2);
   }
}
```

- Variáveis só são válidas no bloco de código (entre chavetas) onde foram criadas

```
class TestScope {
   public static void main(String[] args) {
      for (int i=0; i<10; i++) {
         System.out.println("Iteracao " + i);
      }
      System.out.println(i); // linha dá erro porque i já "não existe"
   }
}
```

Operadores

Aritméticos:

+	adição	-	subtracção
*	multiplicação	/	divisão
%	módulo	++	incremento
--	decremento		

Lógicos:

!	NOT lógico
&&	AND lógico
||	OR lógico

Bits:

~	NOT binário	&	AND binário
|	OR binário	^	XOR binário
<<	shift binário esquerda	>>	shift binário direita

Relacionais ou de comparação:

==	igualdade	!=	diferente
<	menor que	<=	menor ou igual que
>	maior que	>=	maior ou igual que

Divisão inteira

- Quando dividimos por inteiros, o quociente é inteiro.

```
double d1 = 3/2;    // dá 1 e não 1.5 (int/int dá um int)
double d2 = 3.0/2; // dá 1.5 (double/int dá um double)
double d3 = 3/2.0; // dá 1.5 (int/double dá um double)

int    i1 = 3;
double d4 = i1/2; // dá 1 e não 1.5 (int/int)

double d5 = 3;
int    i2 = d5/2; // resultado da divisão é 1.5 mas depois
                  // dá erro "possible loss of precision"
                  // (double/int = double)
```

- Divisão por zero dá um erro durante a execução

```
int i1 = 3/0; // Erro:  "java.lang.ArithmeticException:  / by zero"
```

Resto da divisão inteira

- O operador % calcula o resto da divisão inteira.
  - `18 % 4` dá 2
  - `59 % 5` dá 4
- Exemplos de aplicações do operador %:
  - obter o último dígito de um número: `12345 % 10` é `5`
  - obter os últimos três dígitos: `734528 % 1000` é `528`
  - verificar se um número é par: `9 % 2` é `1` e `16 % 2` é `0`

Regras de precedência entre operadores

Precedência: ordem de avaliação dos operadores.

- a regra geral é avaliação da esquerda para a direita.
  - `5-2-7` é igual a `(5-2)-7` que é `-4`
- mas os operadores `* / %` têm maior precedência que `+ -`
  - `5+2*3` é `5+6` que dá `11`
  - `5+10/2*3` é `5+5*3` que dá `20`
- os parênteses forçam a ordem de avaliação
  - `(5+2)*3` é `7*3` que dá `21`

Regras de precedência entre operadores

Tabela retirada do livro recomendado (*Data Structrures and Algorithms in Java*):

Operator Precedence		
	Type	**Symbols**
1	array index method call dot operator	[ ] ( ) .
2	postfix ops prefix ops cast	*exp*++ *exp*−− ++*exp* −−*exp* +*exp* −*exp* ~*exp* !*exp* (*type*) *exp*
3	mult./div.	* / %
4	add./subt.	+ −
5	shift	<< >> >>>
6	comparison	< <= > >= **instanceof**
7	equality	== !=
8	bitwise-and	&
9	bitwise-xor	^
10	bitwise-or	|
11	and	&&
12	or	||
13	conditional	*booleanExpression* ? *valueIfTrue* : *valueIfFalse*
14	assignment	= += −= *= /= %= <<= >>= >>>= &= ^= |=

Operadores de incremento/decremento

- operadores incremento (++) e decremento (--) têm significado diferente consoante posição em relação à variável
- Primeiro usar o valor da variável e só depois incrementar/decrementar:
  - `variavel++;` $\Longleftrightarrow$ `variavel = variavel + 1;`
  - `variavel--;` $\Longleftrightarrow$ `variavel = variavel - 1;`
- Primeiro incrementar/decrementar e só depois usar o valor da variável:
  - `++variavel;` $\Longleftrightarrow$ `variavel = variavel + 1;`
  - `--variavel;` $\Longleftrightarrow$ `variavel = variavel - 1;`
- Cuidado com o uso destes operadores sobre variáveis no meio de expressões. Consideremos o seguinte:

```
int x = 5, y = 3, z;
x++; // incrementa x (i.e., x = x+1)
--y; // decrementa y (i.e., y = y-1)

z = x-- * ++y; // Com que valores ficam x, y, e z?
```

Operadores de modificação-e-atribuição

- operadores modifica-e-atribui:
  - `variavel += valor;` $\Longleftrightarrow$ `variavel = variavel + valor;`
  - `variavel -= valor;` $\Longleftrightarrow$ `variavel = variavel - valor;`
  - `variavel *= valor;` $\Longleftrightarrow$ `variavel = variavel * valor;`
  - `variavel /= valor;` $\Longleftrightarrow$ `variavel = variavel / valor;`
- Exemplos:
  - `x -= 1;` $\Longleftrightarrow$ `x--;` $\Longleftrightarrow$ `x= x-1;`
  - `y /= 2;` $\Longleftrightarrow$ `y = y/2;`
  - `x *= y+2;` $\Longleftrightarrow$ `x= x*(y+2);`

Instrução condicional if-else

- Exemplo de uso:

```
class TestIfElse {
  public static void main(String[] args) {
    int testscore = 76;
    char grade;

    if (testscore >= 90) {
      grade = 'A';
    } else if (testscore >= 80) {
      grade = 'B';
    } else if (testscore >= 70) {
      grade = 'C';
    } else if (testscore >= 60) {
      grade = 'D';
    } else {
      grade = 'F';
    }

    System.out.println("Grade = " + grade);
  }
}
```

Instrução condicional switch-case

- Exemplo de uso:

```
public class TestSwitch {
  public static void main(String[] args) {
    int month = 2;
    String monthString;
    switch (month) {
    case 1:  monthString = "January";
             break;
    case 2:  monthString = "February";
             break;
    case 3:  monthString = "March";
             break;
    ...
    case 12: monthString = "December";
             break;
    default: monthString = "Invalid month";
             break;
    }
    System.out.println(monthString);
  }
}
```

Instrução de ciclo while

- Exemplo de uso:

```
class TestWhile {

   public static void main(String[] args) {
      System.out.println("isPrime(19) = " + isPrime(19));
   }

   public static boolean isPrime(int n) {
      int divisor = 2;
      while (divisor*divisor <= n) {
         if ( (n % divisor) == 0 )
            return false;
         divisor++;
      }
      return true;
   }
}
```

Instrução de ciclo for

- Exemplo de uso:

```
class TestFor {
  public static void main(String[] args) {
    System.out.println("isPrime(19) = " + isPrime(19));
  }

  public static boolean isPrime(int n) {
    for(int divisor = 2; divisor < n/2; divisor++)
      if ( (n % divisor) == 0 )
        return false;
    return true;
  }
}
```

Instrução de ciclo do-while

- Exemplo de uso:

```
class TestDoWhile {
  public static void main(String[] args) {
    System.out.println("isPrime(19) = " + isPrime(19));
  }

  public static boolean isPrime(int n) {
      int divisor = 2;
      do {
         if ( (n % divisor) == 0 )
            return false;
         divisor++;
      } while (divisor*divisor <= n);
      return true;
   }
}
```

Instruções de controle de fluxo em ciclos

Válidos para for, while ou do-while.

- break sai do ciclo mais interno onde está.

  Resume execução na linha a seguir ao ciclo

```
class TestBreak {
   public static void main(String[] args) {
      for (int i=1; i<=2; i++) {
         for (int j=1; j<=10; j++) {
            if (j == 3) break;
            // Linha seguinte só será executada quando j < 3
            System.out.println("ciclo j = " + j + " | i = " + i);
         }
         System.out.println("ciclo i = " + i);
      }
   }
}
```

```
ciclo j = 1 | i = 1
ciclo j = 2 | i = 1
ciclo i = 1
ciclo j = 1 | i = 2
ciclo j = 2 | i = 2
ciclo i = 2
```

Instruções de controle de fluxo em ciclos

Válidos para for, while ou do-while.

- `continue` passa à próxima iteração do ciclo mais interno onde está
  Resume execução no topo do ciclo, ignorando as restantes linhas na iteração actual

```
class TestContinue {
   public static void main(String[] args) {
      for (int i=1; i<=4; i++) {
         System.out.println("[antes] i = " + i);
         if (i > 2) continue;
         // Linha seguinte só será executada quando i <= 2
         System.out.println("[depois] i = " + i);
      }
   }
}
```

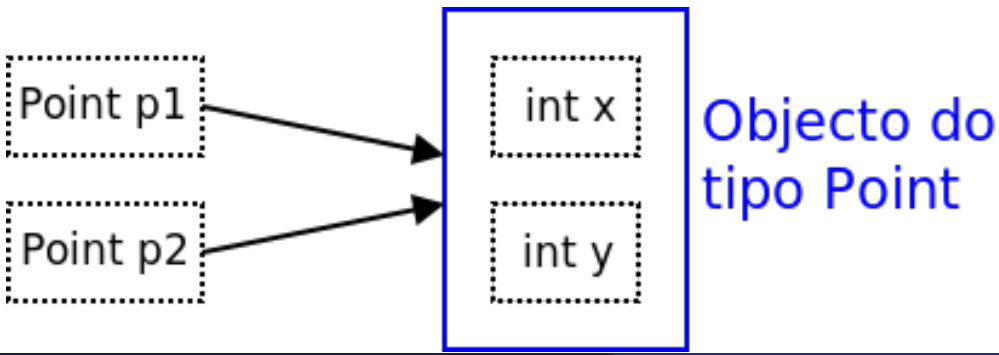
```
[antes] i = 1
[depois] i = 1
[antes] i = 2
[depois] i = 2
[antes] i = 3
[antes] i = 4
```

Classes

- Em Java um "tipo não básico" é uma **classe**
- Uma variável do tipo de uma classe é apenas uma **referência**, um endereço de memória a apontar para o conteúdo em si (como que o "equivalente" de um apontador em C)
- A uma instância de uma classe chamamos de **objecto**.
- Para criar um objecto de uma classe usamos o operador **new**
- Às variáveis de uma classe chamamos de **atributos**.
- Para aceder aos atributos de um objecto usamos o operador **ponto**

Um primeiro exemplo de classe só com atributos

```
class Point {
    int x;
    int y;
}

class TestPoint {
    public static void main(String[] args) {
        Point p1 = new Point();    // Criação de um objecto do tipo Point
        p1.x = 42;                 // Atribui um valor ao atributo x
        System.out.println(p1.x); // Escreve 42

        Point p2 = p1;             // p2 é referência para o mesmo objecto de p1
        System.out.println(p2.x); // Escreve 42

        System.out.println(p1);    // Escreve referência (endereço de memória),
    }                              // e não o conteúdo de p1
}
```



Classes - Métodos

- A um procedimento/função de uma classe chamamos de **método**

```
class Point {
    int x;
    int y;

    // Método para mostrar o conteúdo das variáveis x e y
    void show() {
        System.out.println("(" + x + "," + y + ")");
    }
}

class TestPoint {
    public static void main(String[] args) {

        Point p = new Point();
        p.x = 42;
        p.y = 13;

        p.show(); // Escreve "(42,13)"
    }
}
```

Classes - Métodos construtores

- Quando é criado um objecto é chamado o método **construtor**
  - Método com o nome da classe e sem nenhum tipo de retorno

```
class Point {
    int x;
    int y;

    // Construtor que recebe o conteúdo a colocar nos atributos
    Point(int xvalue, int yvalue) {
        x = xvalue;
        y = yvalue;
    }

    // Método para mostrar o conteúdo das variáveis x e y
    void show() {
        System.out.println("(" + x + "," + y + ")");
    }
}

class TestPoint {
    public static void main(String[] args) {
        Point p = new Point(42, 13);
        p.show(); // Escreve "(42,13)"
    }
}
```

Classes - Métodos construtores

- Se não for criado nenhum construtor apenas existe um construtor padrão sem argumentos (que não faz nada)
- Criado um construtor qualquer, deixa de existir o construtor padrão...

```
class Point {
    int x;
    int y;

    // Construtor que recebe o conteúdo a colocar nos atributos
    Point(int xvalue, int yvalue) {
        x = xvalue;
        y = yvalue;
    }
}

class TestPoint {
    public static void main(String[] args) {
        Point p = new Point(); // Erro:  não existe construtor sem argumentos
    }                          // Apenas existe um que recebe int, int
}
```

Classes - Overload de Métodos

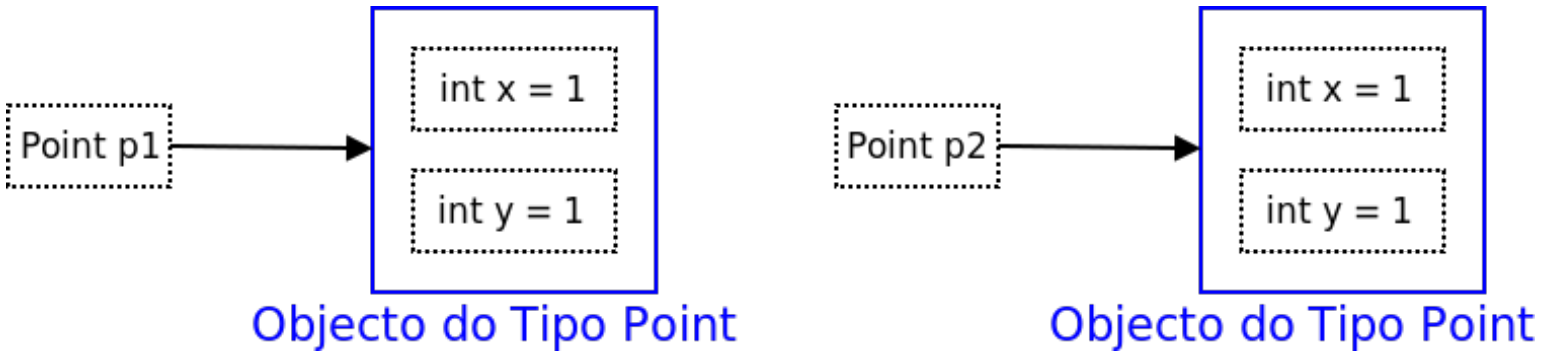
- Java permite **overload** de métodos (métodos com o mesmo nome mas lista de argumentos diferente)

```
class Point {
    int x;
    int y;

    Point() { // Construtor padrão (sem argumentos)
        x = 0;
        y = 0;
    }

    Point(int xvalue, int yvalue) { // Construtor para 2 ints
        x = xvalue;
        y = yvalue;
    }
}

class TestPoint {
    public static void main(String[] args) {
        Point p1 = new Point();       // p1 fica com (0,0)
        Point p2 = new Point(42,13); // p2 fica com (42,13)
    }
}
```

Classes - Comparação de objectos

- Comparar directamente com == duas variáveis do tipo de uma classe, compara as referências e não o conteúdo!
  (normalmente as classes têm um método "equals" para comparar)

```
class Point {
   int x, y;
   Point(int xvalue, int yvalue) { x = xvalue; y = yvalue;}
}

class TestPoint {
   public static void main(String[] args) {
      Point p1 = new Point(1,1);  // p1 fica com (1,1)
      Point p2 = new Point(1,1);  // p2 fica com (1,1)
      System.out.println(p1==p2); // escreve false, porque são referências
   }                              // para objectos diferentes, embora tenham
}                                 // o mesmo conteúdo
```



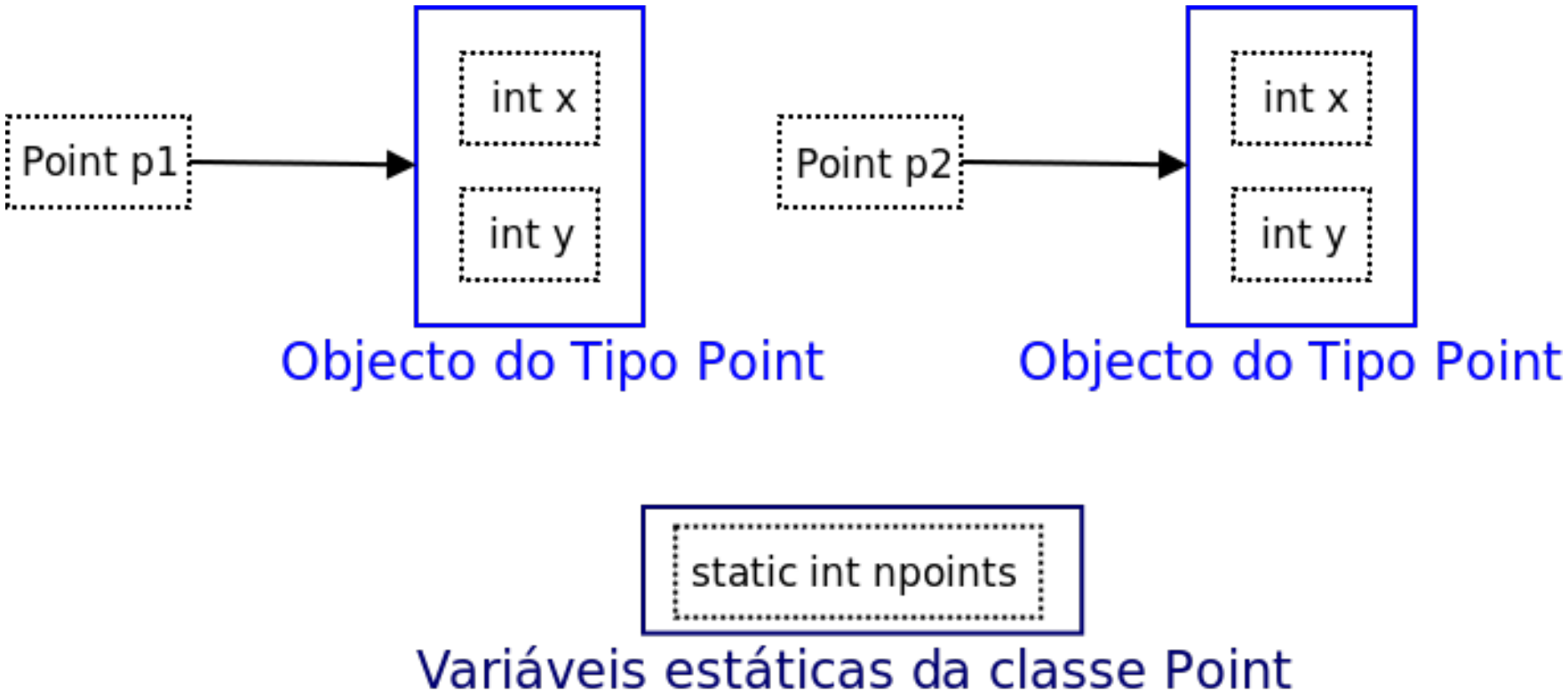
Classes - Variáveis e métodos estáticos

- Se quisermos que uma variável (ou método) seja acessível para todos os objectos da mesma classe podemos declará-la como **static**

```
class Point {
   static int npoints = 0; // Variavel estática com nr pontos criados
   int x, y;

   Point() { npoints++; }  // Construtor

   static void showNumPoints() { // mostra conteudo da variavel estatica
      System.out.println("npoints = " + npoints);
   }
}

class TestPoint {
   public static void main(String[] args) {
      // É possível chamar método estático mesmo sem objectos criados
      Point.showNumPoints();   // Escreve npoints = 0
      Point p1 = new Point();
      Point.showNumPoints();   // Escreve npoints = 1
      Point p2 = new Point();
      Point.showNumPoints();   // Escreve npoints = 2
   }
}
```

Classes - Variáveis e métodos estáticos

- Um esquema de como fica em memória o programa anterior:



Classes - Variáveis e métodos estáticos

- Um método estático só pode chamar variáveis estáticas
  (como saberia de qual objecto chamar a variável? Até pode ser chamado antes de ser criado qualquer objecto dessa classe...)

```
class Point {
   int x;
   int y;

   static void showX() {
      // Erro:  non-static variable x cannot be referenced from a static context
      System.out.println(x);
   }
}
```

- Nota: `main` (procedimento usado para iniciar o programa) é estático,
  (O Java chama o método `NomeClasse.main` para iniciar o programa sem criar nenhum objecto do tipo `NomeClasse`)

API do Java

- Java dispõe de muitas classes já implementadas prontas a usar. Exemplos:
  - String
  - Scanner
  - Arrays
  - ...

- Vamos ir aprendendo algumas aos poucos, mas a documentação de todas as classes está disponível online:

  Java API: https://docs.oracle.com/javase/8/docs/api/

Strings

API: https://docs.oracle.com/javase/8/docs/api/java/lang/String.html

- uma string é uma sequência de caracteres
- Em Java não são um tipo básico, mas sim instâncias da classe `String`
- não se comportam como arrays, pois são imutáveis
  (uma vez criadas não se podem modificar)
- criação/inicialização invocando métodos construtores:
  - `String c1 = new String("CC");`
- criação simplificada:
  - `String c2 = "1007";`
- operador de concatenação:
  - `String c3 = c1 + c2 + " rocks";`
- o valor que fica em `c3` é:
  - `"CC1007 rocks";`

Strings

- ex: `String s = "algoritmos";`

índice	0	1	2	3	4	5	6	7	8	9
caracter	'a'	'l'	'g'	'o'	'r'	'i'	't'	'm'	'o'	's'

- podemos ler caracteres individuais mas não alterá-los

```
String s = "algoritmos";
System.out.println(s.charAt(6)); // 't'
```

- tamanho obtido através do método `length()`

```
for(int i = 0; i < s.length(); i++)
  System.out.print(s.charAt(i));
```

- quando se escreve, o comportamento por omissão é mostrar conteúdo e não apenas a referência
  (tem implementado o método `toString()`)

```
String s = "algoritmos";
System.out.print(s);      // Escreve "algoritmos"
```

Strings

- comparação de duas strings
  - usar o método `equals()` e não o habitual operador "==".
- dadas duas strings `s1` e `s2`
  - `s1 == s2` apenas compara as referências dos dois objectos
  - `s1.equals(s2)` compara as strings, caracter a caracter

```
String s1= "Hello";
String s2= new String("Hello");
System.out.println(s1.equals(s2)); // true
System.out.println(s1 == s2);      // false

String s3= "Good-bye";
String s4= "HELLO";
System.out.println(s1.equals(s3));           // false
System.out.println(s1.equals(s4));           // false
System.out.println(s1.equalsIgnoreCase(s4)); // true (é um outro método)
```

Strings

API: https://docs.oracle.com/javase/8/docs/api/java/lang/String.html

A API contém muitos mais métodos que pode usar. Alguns exemplos:

- O método `s1.compareTo(s2)` compara as strings `s1` e `s2` por ordem lexicográfica
  - `< 0` se `s1` precede `s2`, `== 0` se forem iguais, `> 0` se `s1` vem a seguir a `s2`

```
String s1 = "algoritmos";
String s2 = "estruturas";
s1.compareTo(s2); // < 0
```

- `substring(a,b)` devolve substrings da posição a à $b-1$

```
String s1 = "algoritmos";
String s2 = s1.substring(2,5); // "gor"
```

- podemos converter strings em arrays de caracteres e vice-versa

```
String s1 = "algoritmos";
char[] cs = s1.toCharArray();
String s2 = new String(cs);
```

- mini-exercício: API para verificar se um caracter ocorre numa String?

Arrays

Os **arrays** são objectos que guardam, em posições contíguas de memória, um conjunto de valores de um mesmo tipo (primitivo ou não). Os valores são localizados por um índice inteiro ≥ 0.

- criar variável para guardar a referência para um array de um dado tipo (não reserva espaço em memória!)
  - `int[] values;`
  - `Point[] coordinates;`
- o operador `new` cria o array com a capacidade indicada em memória
  - `int[] values = new int[20];`
  - `Point[] coordinates = new Point[1024];`
  - os elementos do array são acessíveis pelo nome da variável e um índice, e.g., `values[17]` ou `coordinates[221]`

Arrays

- Ao trabalharmos com arrays (e outros objectos) é necessário ter atenção a possíveis excepções e erros que sejam gerados:
- tentar usar uma referência sem a ter inicializado

```
int[] v;
v[0] = 2; // variable v might not have been initialized
          // nalguns compiladores pode devolver um NullPointerException
```

- `ArrayIndexOutOfBounds` – aceder ao array fora dos limites

```
int[] v = new int[4];
v[0] = 2;
v[4] = 5; // ArrayIndexOutOfBoundsException
```

- quando declara um array deve sempre inicializá-lo de imediato
- de resto, este conselho é válido para todas as variáveis
- o atributo `length` de um array dá-nos o seu tamanho (o número de posições em memória que foram criadas, ou o máximo de elementos que pode conter)

Packages

O java inclui uma série de classes *"built-in"* agrupadas em **packages**. Exemplos:

- `java.lang` - Classes fundamentais da classe (ex: `String`) https://docs.oracle.com/javase/8/docs/api/java/lang/package-summary.html
- `java.util` - Classes utilitárias (ex: `Arrays`, `Scanner`) https://docs.oracle.com/javase/8/docs/api/java/util/package-summary.html

Para poder usar-se uma classe de um package que não seja do `java.lang`, deve usar-se a palavra-chave **import**. Exemplo:

- `import java.util.Scanner; //importar a classe Scanner`
- `import java.util.*; //importar todas as classes do package java.util`

Package Arrays

- Como imprimimos os elementos de um array?
  (necessita no início de **import** `java.util.Arrays;`)

```
int[] primes = {2,3,5,7,11,13}; // Cria e inicializa array
System.out.println(primes);     // Apenas imprime a referência
System.out.println(Arrays.toString(primes)); // Converte para String
```

output:

```
[I@75b84c92
[2,3,5,7,11,13]
```

- Como comparámos dois arrays?
  - usar um ciclo que compara os elementos, um a um
  - usar o método `Arrays.equals()`

```
int[] a = {1,2,3,4,5};
int[] b = {1,2,3,4,5};
if( Arrays.equals(a,b) )
  System.out.println("same contents");
```

A classe `Arrays` contém muitos mais métodos estáticos úteis:
API: https://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html

Arrays

- Um método pode ter parâmetros na chamada que são arrays e pode dar como resultado um array.
- Exemplo: um método que recebe 2 arrays (de inteiros) de igual tamanho e retorna um array com a soma dos dois

```
int[] addArrays (int[] u, int[] v) {
  int[] res = new int[u.length];
  for ( int i = 0 ; i < u.length ; i++ )
    res[i] = u[i] + v[i];
  return res;
}
```

Arrays Multidimensionais

- os arrays podem ser multidimensionais

```
int[][]   v = new int[4][4];     // bi-dimensional 4x4

int[][][] u = new int[5][3][7]; // tri-dimensional 5x3x7
```

- Exemplo: multiplicação de uma matriz $a[N][M]$ por um vector $u[M]$ para dar $v[N]$:

  i.e.

$$v_i = \sum_{j=0}^{M-1} a_{ij} \times u_j \quad (0 \leq i < N)$$

Exemplo:

1	2	3										
4	5	6	*	7	8	9	=	1*7+2*8+3*9	4*7+5*8+6*9	=	50	122

Multiplicação de matriz por array

```
import java.util.Arrays;

class TestMatrixVectorProduct {
   public static void main(String[] args) {
      int[][]  a = {{1,2,3},{4,5,6}};
      int[]    u = {7,8,9};
      int[]    v = matrixVectorMult(a,u);

      System.out.println(Arrays.toString(v));
   }
   static int[] matrixVectorMult(int[][] a, int[] u) {
      int[] v = new int[a.length];
      for( int i = 0 ; i < v.length ; i++ ) {
         v[i] = 0;
         for( int j = 0 ; j < u.length ; j++ )
            v[i] += a[i][j] * u[j];
      }
      return v;
   }
}
```

Input/Output

As classes mais importantes que lidam com I/O no Java estão definidas nas *packages* `java.io` e `java.lang`

- a leitura e escrita faz-se através de canais (*streams*) que podem representar um qualquer periférico físico.
- a classe **System**, definida em `java.lang`, inclui muitas definições de sistema, nomeadamente 3 canais: **in**, **out**, e **err**.
  - `InputStream System.in` – objecto que representa o standard input stream (por omissão é o teclado);
  - `PrintStream System.out` – objecto que representa o standard output stream (por defeito a consola);
  - `PrintStream System.err` – objecto que representa o standard error stream (consola).

A classe Scanner

- simplifica muito o processamento do input vindo de:
  - teclado (tipo `InputStream`)

```
Scanner stdIn= new Scanner(System.in);
```

  - através de uma `String`

```
String line = new String("Hello World!");
Scanner strIn= new Scanner(line);
```

  - ou de um ficheiro

```
File file= new File(fileName);
Scanner fileIn= new Scanner(file);
```

- divide input em strings separadas por *delimitadores*.
- `useDelimiter(expr)` permite especificar delimitadores; ex:

`scanner.useDelimiter("\r\n")`

A Classe Scanner

Para se poder usar a classe Scanner é necessário declarar no programa:

- **import** `java.util.Scanner;`

Alguns métodos relevantes desta classe:

`hasNext()`	**true** se e só se existir mais uma palavra no input
`next()`	retorna a próxima palavra (String) do input
`hasNextLine()`	**true** se e só se o input tiver mais uma linha de texto
`nextLine()`	retorna a próxima linha de texto do input
`hasNextType()`	**true** se e só se a próxima palavra for do tipo `Type` onde `Type` pode ser qualquer tipo básico: **int**, **float**, ...
`nextType()`	retorna a próxima palavra convertida para o tipo básico definido por `Type`.

Scanner: leitura a partir de uma String

```
import java.util.Scanner;

public class TestScannerFromString {
  public static void main (String[] args) {
    Scanner strIn = new Scanner("1 - 2 - 3 - 4 - 5");
    strIn.useDelimiter(" - ");
    while ( strIn.hasNextInt() ) {
      int n = strIn.nextInt();
      System.out.println(n);
    }
  }
}
```

Scanner: leitura a partir da entrada padrão

```
import java.util.Scanner;

class TestScannerFromStdIn {
  public static void main (String[] args) {
    Scanner stdIn = new Scanner(System.in);
    System.out.println("Number of persons: ");

    int n = stdIn.nextInt();
    String[] names = new String[n];
    int[]    ages  = new int[n];

    for( int i = 0; i < n ; i++ ) {
      System.out.println("input name[space]age: ");
      names[i] = stdIn.next();
      ages[i]  = stdIn.nextInt();
    }

    for( int i = 0; i < n ; i++ )
      System.out.println("name:"+names[i]+" age: "+ages[i]);
  }
}
```

Scanner: leitura a partir de um ficheiro

```
import java.io.File;
import java.io.IOException;
import java.util.Scanner;

class TestScannerFromFile {
  public static void main (String args[]) {
    try {
      File file = new File("./example.txt");
      Scanner fileIn = new Scanner(file);
      while( fileIn.hasNextLine() )
        System.out.println(fileIn.nextLine());
    }
    catch (IOException e) { // Mais sobre a instrução catch noutra aula
      System.out.println("Error Opening File");
    }
  }
}
```

Output formatado

O Java permite output formatado ao estilo do `printf` do C.

- usar o método `printf()`
  Exemplo: `System.out.printf("pi= %5.3f%n", pi);`

  `System.out.printf ("string format", parameters)`
- Uma string de formato contém "encaixes" para inserir os valores dos parâmetros:
  - `%d` - inteiro base 10
  - `%x` - inteiro base hexadecimal
  - `%o` - inteiro base octal
  - `%f` - número em vírgula flutuante
  - `%s` - string
  - `%c` - caracter
- o `printf()` não faz a mudança de linha
- incluir o caracter `'%n'` (*newline*) na string de formato (é usada a maneira de mudar de linha do S.O. onde o Java está a ser executado)

Formatos especiais (inteiros)

- `%Wd` - inteiro com `W` caracteres de largura, alinhado à direita.
- `%-Wd` - inteiro com `W` caracteres de largura, alinhado à esquerda.

```
long n = 461012;
System.out.printf("%d%n", n);       // --> "461012"
System.out.printf("%8d%n", n);      // --> "__461012"
System.out.printf("%-8d%n", n);     // --> "461012__"
System.out.printf("%08d%n", n);     // --> "00461012"
```

Formatos especiais (floating point)

- `%.Df` - nº real, arredondado para `D` casas decimais.
- `%W.Df` - nº real, com `W` espaços no total para os dígitos, e `D` casas decimais
- `%-W.Df` - o mesmo que o anterior mas com os dígitos alinhados à esquerda
- exemplo:

```
double pi = Math.PI;

System.out.printf("%f%n", pi);         // --> "3.141593"
System.out.printf("%.3f%n", pi);       // --> "3.142"
System.out.printf("%10.3f%n", pi);     // --> "_____3.142"
System.out.printf("%-10.3f%n", pi);    // --> "3.142_____"
```

Algumas regras de etiqueta

- nomes de classes são substantivos e começam com maiúscula (ex: `Point`)
- se o nome tiver várias palavras, todas começam com maiúscula (ex: `IntegerPoint`)
- nomes de atributos são substantivos e começam com minúscula (ex: `distance`)
- se o nome tiver várias palavras, apenas a primeira começa com minúscula (ex: `distanceOrigin`)
- nomes de métodos são verbos começam com minúscula (ex: `show`)
- se o nome tiver várias palavras, apenas a primeira começa com minúscula (ex: `computeMagnitude`)
- sempre que possível devem ser utilizados nomes por extenso, excepto em casos onde haja convenção (ex: `x`, `y`, `z` para as coordenadas, `i`, `j` para variáveis de iteração num ciclo)

Algumas regras de etiqueta

- há excepções às regras de etiqueta, ex:
  - `sqrt`
  - `minus`
  - `scalarProduct`
  - `modulus`
- usar o bom senso
- o fundamental é que o nome/verbo transmita com clareza a semântica pretendida