

# Programação I

**Dep. de Electrónica, Telecomunicações e Informática**  
**Universidade de Aveiro**

*Arnaldo Martins*  
(Prof. Catedrático)  
Email: jam@ua.pt

<http://elearning.ua.pt>

# Aula 1

- Apresentação da disciplina
- Organização de um computador (*Livro, pág. 3-10*)
- Desenvolvimento de um programa (*Livro, pág. 24-33*)
- Conceitos base da linguagem JAVA (*Livro, pág. 115-145*)
  - Estrutura de um programa
  - Tipos de dados
  - Variáveis e constantes
  - Operadores e expressões
  - Classes da linguagem JAVA
  - Leitura e escrita de dados (*Livro, pág. 156-162*)
  - Escrita formatada (*Livro, pág. 169-174*)

# Objetivos

- Compreensão clara, ainda que elementar, do que é um computador, como funciona, para que serve, que limitações tem e como se comunica com ele.
- Desenvolvimento de estratégias para a especificação precisa do problema que se pretende pôr o computador a resolver.
- Estabelecimento de métodos para descrição detalhada e rigorosa de soluções que possam ser implementadas num computador.
- Aprendizagem de uma linguagem de programação (JAVA).
- Familiarização com um ambiente de desenvolvimento onde os programas possam ser escritos, documentados, testados e validados.

# Programa

- Introdução à Linguagem JAVA: elementos
- Estruturas de controlo: instruções decisórias
- Estruturas de controlo: instruções repetitivas
- Programação modular (Funções/Métodos)
- Matrizes/Sequências (Arrays)
- Criação de novos tipos de dados (Registos/Classes)
- Sequências de caracteres (Strings)
- Ficheiros de texto
- Pesquisa e ordenação

# Metodologia e Organização das aulas

- *“o que ouço, esqueço; o que vejo, recordo; o que faço, compreendo” [confucio]*
- **Aulas teórico-práticas:**
  - aulas baseadas em slides e exemplos que serão colocados on-line;
  - apresentação e discussão dos temas da disciplina;
- **Aulas práticas:**
  - Aplicação dos conhecimentos à resolução de problemas concretos;

# Bibliografia

## Livro recomendado

- António Adrego da Rocha, Osvaldo Rocha Pacheco, *"Introdução à Programação em Java"*, 1ª edição, FCA editores, 2009.

## Bibliografia complementar

- Allen Downey and Chris Mayfield, Think Java: How to Think Like a Computer Scientist, 2ed  
<http://greenteapress.com/wp/think-java-2e/>

# Avaliação

- A disciplina tem **avaliação contínua** com 6 momentos de avaliação:
  - Teste Prático 1 (25%) – 4 dez
  - Teste Prático 2 (35%) - 15 jan
  - Teste Teórico Prático 1 (10%) – 27 out a 2 nov (durante as aulas)
  - Teste Teórico Prático 2 (10%) – 10 a 16 nov (durante as aulas)
  - Teste Teórico Prático 3 (10%) – 15 a 21 dez (durante as aulas)
  - Questões nas aulas TP (10%) – questões no e-learning
- A frequência das aulas práticas é obrigatória (Limite de faltas 20% - 3 faltas)

Os repetentes tem a TP3 (em EaD) de frequência opcional.
- Ou avaliação por **Exame Final** (30% TP + 70% P)

# Aula 1

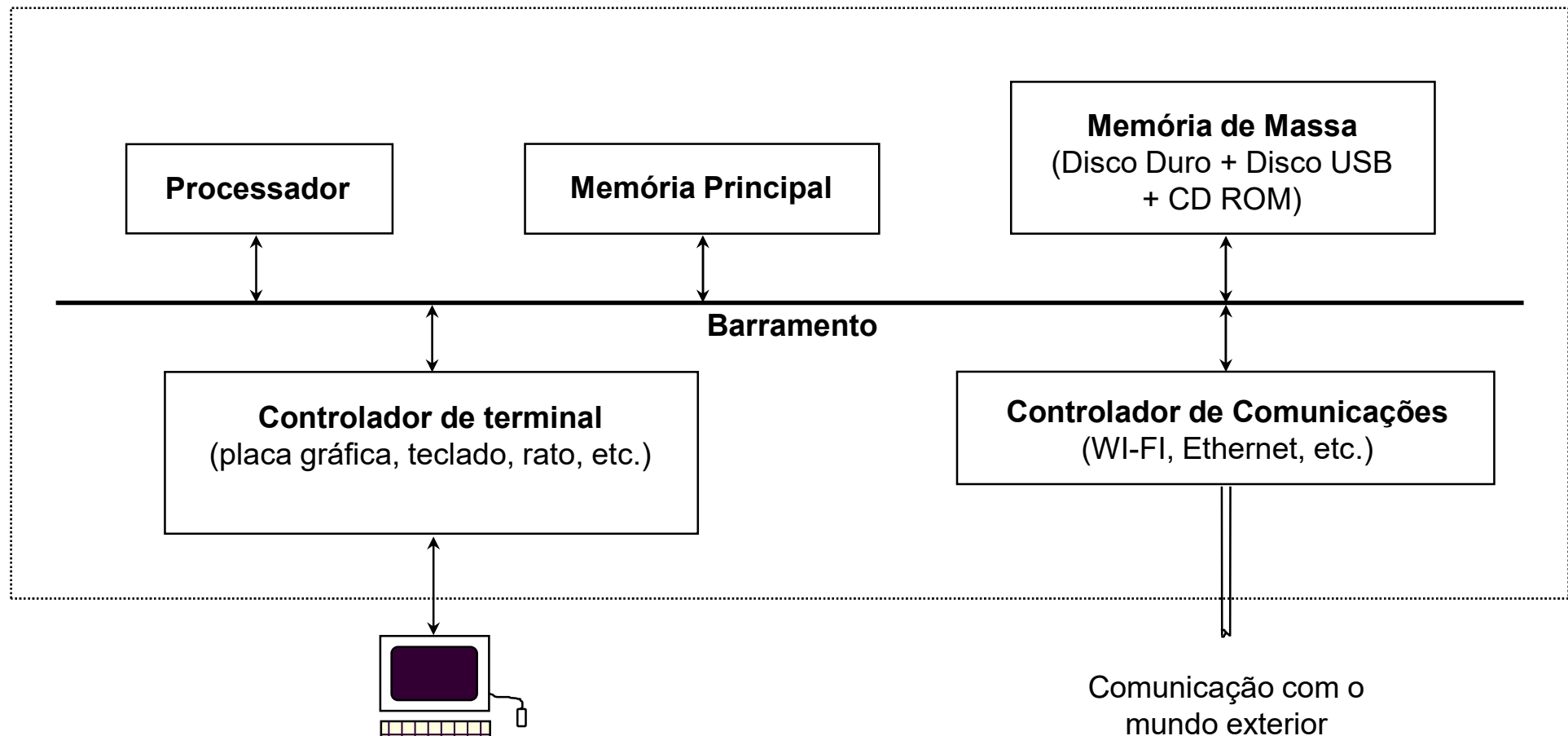
## O computador e os elementos básicos da linguagem JAVA



# Computador...

- Máquina programável que processa informação de forma autónoma.
- Executa, com uma cadência muito rápida, sequências de operações elementares sobre informação recebida, devolvendo ao utilizador resultados.
- A sequência de operações elementares, designada habitualmente por **programa**, pode ser alterada ou substituída por outra, sempre que se deseje.
- Durante a execução do programa, a sequência de operações elementares e os valores temporários produzidos estão armazenados num dispositivo interno, chamado memória.

# Organização de um computador



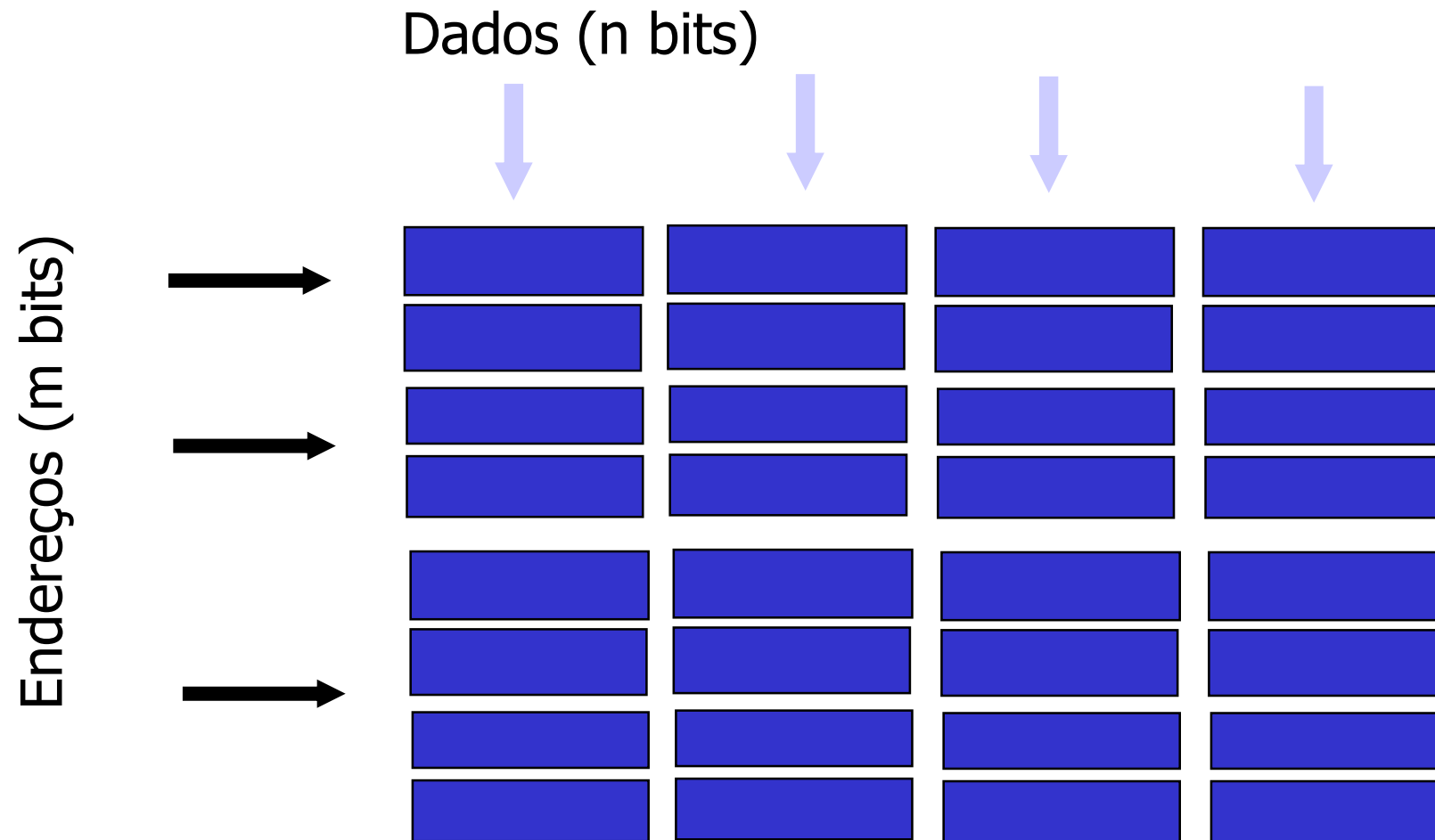
# Organização de um computador

- O computador utiliza tecnologia e lógica binária (valor ‘0’ ou ‘1’).
- Todos os dados (números inteiros, reais, texto, etc.) são armazenados em *bits*. Um conjunto de 8 bits corresponde a um *byte*.
- A memória do computador organiza-se em endereços (normalmente com um identificador associado) e dados :

Endereços	“Identificador”	Dados	Significado
0xFF0000	idade	0011...1001	40
0xFF0001	peso	1001...0101	34.50
...	...	...	...
0xFF00FE	fimDeCiclo	0000...0000	false
0xFF00FF	msg	1101...1001	‘Olá’



# Memória: Endereços e Dados



# Homem Vs. Computador

## Homem

### a abordagem é criativa

- aprende com a experiência passada;
- associa conceitos distintos, conseguindo isolar elementos comuns;
- usa em larga medida um raciocínio de tipo indutivo (intuição);

## Computador

### a abordagem é não criativa

- não tem capacidade directa de aprendizagem;
- só associa conceitos cuja afinidade foi previamente estabelecida;
- usa mecanismos de raciocínio dedutivo;

# Homem Vs. Computador

## Homem

propõe soluções

- descobre métodos de resolução;

comete erros

- as inferências produzidas são muitas vezes incorrectas;
- está sujeito a lapsos de concentração provocados por cansaço.

## Computador

não propõe soluções

- possibilita a validação das soluções encontradas;

não comete erros

- salvo avaria, limita-se a executar de um modo automático a sequência de operações estabelecida.

# Tipos de problemas que o computador resolve

Problemas completamente especificados:

- as variáveis de entrada e de saída estão perfeitamente identificadas;
- se conhece uma **solução**; ou seja, um método que permite obter, de forma unívoca, os valores das variáveis de saída em função dos valores das variáveis de entrada;
- deve considerar-se sempre a resolução dos problemas no âmbito mais lato possível; ou seja, deve considerar-se a resolução de classes de problemas e não de problemas particulares;

# Tipos de problemas que o computador resolve

Problemas completamente especificados:

- a gama de valores permitida para as variáveis de entrada deve ser claramente estabelecida;
- a solução descrita deve contemplar alternativas para toda a gama de valores das variáveis de entrada, eliminando toda e qualquer ambiguidade.



# Exemplo de um problema

Conversão de distâncias (milhas para Km)

- Dada uma distância, expressa em milhas, que é lida do teclado, convertê-la para quilómetros e escrevê-la no ecrã do computador (terminal).

## Variável de entrada:

MILHAS (distância expressa em milhas)  
valor numérico positivo ou nulo

## Variável de saída:

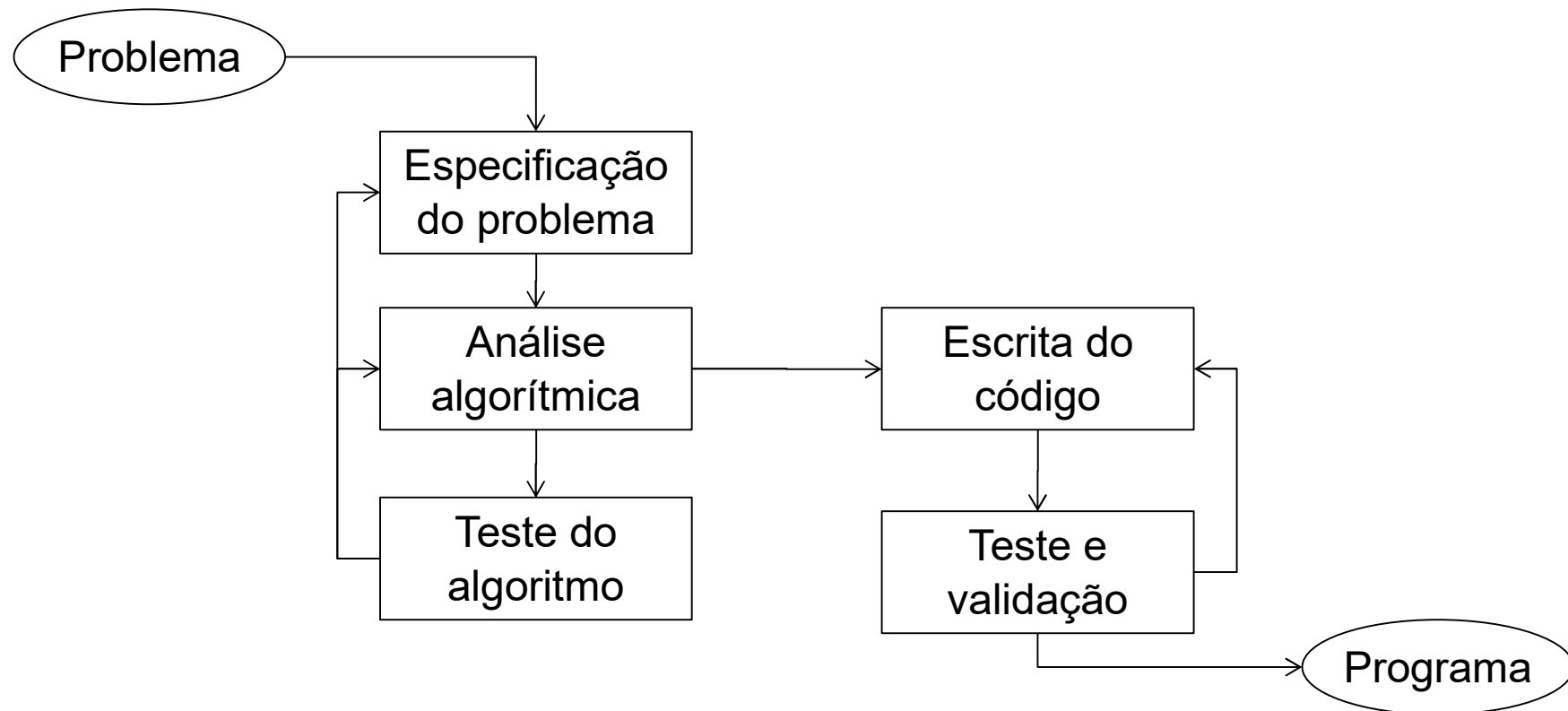
KILOMETROS (distância expressa em quilómetros)  
valor numérico representado com 3 casas decimais

## Solução:

$$\text{KILOMETROS} = 1.609 * \text{MILHAS}$$

# Fases de desenvolvimento de um programa

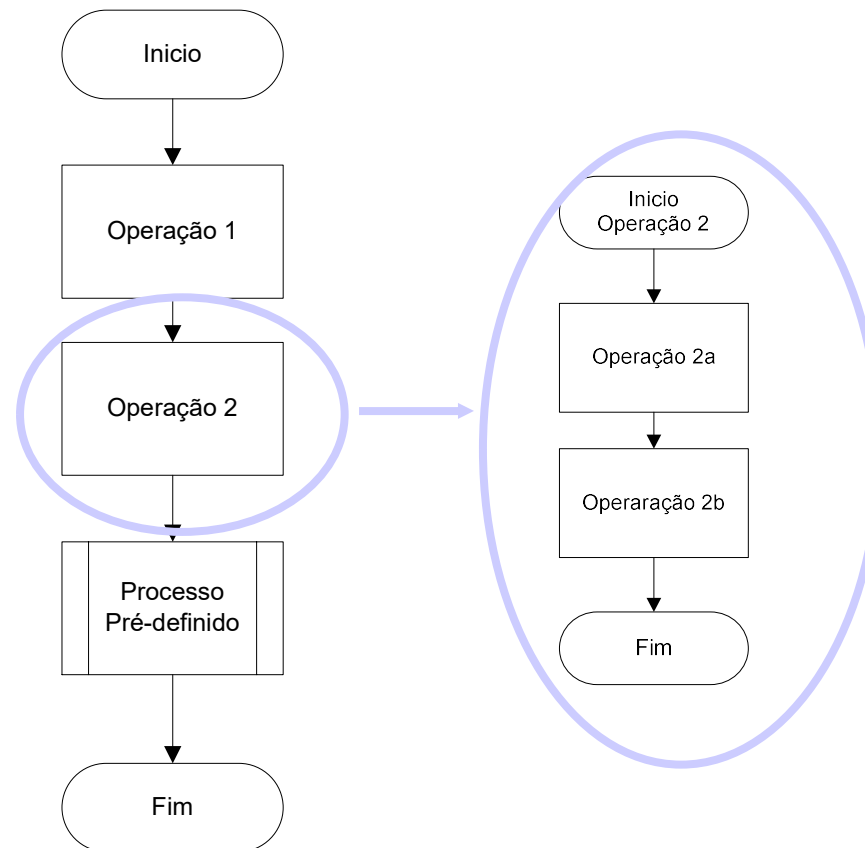
- As duas etapas básicas do desenvolvimento de um programa são a **análise do problema** e a **implementação da aplicação**.



# Algoritmo

- Designa-se por **algoritmo** a descrição detalhada e rigorosa da solução do problema.
- A transcrição do algoritmo para uma linguagem de programação dá origem ao **programa**.
- Supõe-se que o conjunto de operações descrito no algoritmo é realizado segundo uma ordem pré-estabelecida: só se inicia uma dada operação, quando a anterior estiver terminada - **execução sequencial**.
- Exemplo:
  - leitura dos valores das variáveis de entrada
  - processamento
  - escrita dos valores das variáveis de saída

# Diagramas de Fluxo – *Flowchart* (Operações)



# Estrutura de um programa

inclusão de classes externas

```
public class Programa
```

```
{
```

declaração de constantes e variáveis globais

```
public static void main (String[] args)
```

```
{
```

declaração de constantes e variáveis  
locais

sequências de instruções

```
}
```

```
}
```

definição de tipos de dados (registos)



# Exemplo de um programa

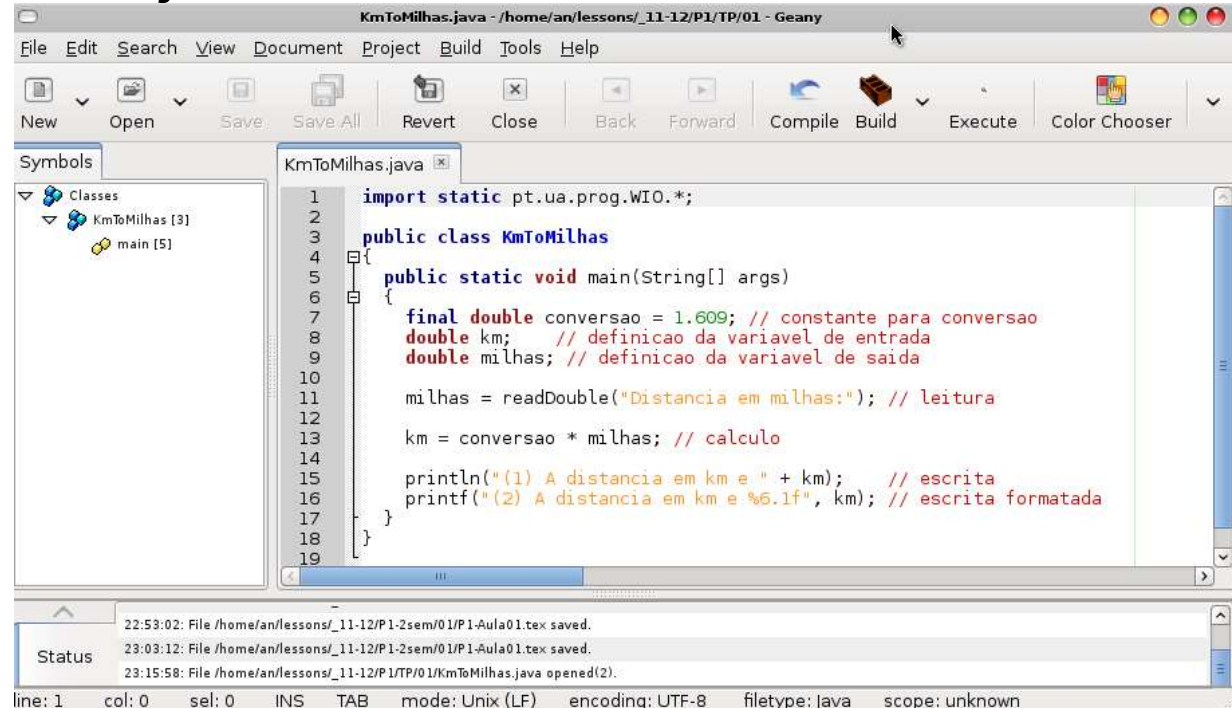
## Ficheiro **KmToMilhas.java**

```
import java.util.Scanner;
public class KmToMilhas {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        final double CONVERSAO = 1.609; // constante
        double km, milhas;
        System.out.print("Distancia em milhas:");
        milhas = sc.nextDouble();
        km = CONVERSAO * milhas;
        System.out.println("A distancia em km é " +
        km);
    }
}
```

Nome ficheiro = nome da class  
Nomes das classes começam por maiúsculas;  
Nomes das variáveis começam por minúsculas;  
Constantes em maiúsculas;

# Desenvolvimento de um programa

- Edição:
  - geany KmToMilhas.java



```
1 import static pt.ua.prog.WIO.*;
2
3 public class KmToMilhas
4 {
5     public static void main(String[] args)
6     {
7         final double conversao = 1.609; // constante para conversao
8         double km; // definicao da variavel de entrada
9         double milhas; // definicao da variavel de saida
10
11         milhas = readDouble("Distancia em milhas:"); // leitura
12
13         km = conversao * milhas; // calculo
14
15         println("(1) A distancia em km e " + km); // escrita
16         printf("(2) A distancia em km e %6.1f", km); // escrita formatada
17     }
18 }
19
```

Status

22:53:02: File /home/an/lessons/\_11-12/P1-2sem/01/P1-Aula01.tex saved.  
23:03:12: File /home/an/lessons/\_11-12/P1-2sem/01/P1-Aula01.tex saved.  
23:15:58: File /home/an/lessons/\_11-12/P1/TP/01/KmToMilhas.java opened(2).

line: 1 col: 0 sel: 0 INS TAB mode: Unix (LF) encoding: UTF-8 filetype: java scope: unknown

- Compilação
  - javac KmToMilhas.java
- Execução
  - java KmToMilhas

# Elementos básicos da linguagem JAVA

- **Palavras reservadas** – símbolos que têm um significado bem definido em JAVA e que não podem ser usadas para outro fim (ex. `class`, `break`, `switch`, `final`, `if`, `then`, `else`, `while`, ...).
- **Identificadores** – nomes utilizados para designar todos os objectos existentes num programa. Devem começar por uma letra ou por símbolo '\_' e só podem conter letras, números e o símbolo '\_' (ex. `nome`, `idade`, `i`, `j`, `cont_1`, `dia_mes`, `res`, `_km` ...).
- **Comentários** – melhoram a legibilidade de um programa (todos os caracteres na mesma linha que se seguem ao símbolos `//` e blocos `/*` comentários (podem ser várias linhas) `*/`).



# Elementos básicos da linguagem JAVA

- **Constantes** – “valor específico” de um certo tipo (ex. 10, -10, 5.5, .5, -0.8, “Aveiro”, true, ...).
- **Operadores e separadores** – símbolos ou combinações de símbolos que especificam operações e usados na construção de instruções: ( ) [ ] { } < > ; . , : ? ! ' " & | = + - \* / % ~ ^ # \ \_ \$

# Tipos de dados

## Tipos primitivos

- `byte, short, int, long` - números inteiros (10, -10, 0, ...)
- `float, double` - números reais (10.5, -10.5, .2, ...)
- `boolean` – apenas dois valores possíveis (`true, false`)
- `char` – caracteres (`'a', '1', '!', ...`)

## Tipo String

- `String` – sequência de caracteres (`"ria de aveiro", "aula 1", ...`)

## Definição de uma variável:

`tipo identificador variável1, variável2, ...;`

# Tipos de dados

- Uma variável (posição de memória) pode ser considerada como uma caixa cujo conteúdo inicialmente não está definido.
- Exemplos de definição de variáveis e constantes:
  - `double peso, altura, largura, erro;`
  - `int idade, dia_mes, ano;`
  - `boolean resultado;`
  - `char letra, op;`
  - `String nome, cidade;`
  - `final double PI = 3.1415; //def. constante real`
  - `final int LIMITE = 100; //def. constante inteira`

# Tipos de dados – Gama de valores

Type	Storage requirement	Range (inclusive)
int	4 bytes	-2,147,483,648 to 2,147,483,647 (just over 2 billion)
long	8 bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
short	2 bytes	-32,768 to 32,767
byte	1 byte	-128 to 127

Type	Storage requirement	Range
float	4 bytes	Approximately $\pm 3.40282347\text{E}+38\text{F}$ (6–7 significant decimal digits)
double	8 bytes	Approximately $\pm 1.79769313486231570\text{E}+308$ (15 significant decimal digits)

# Inicialização de variáveis

- Antes de uma variável poder ser utilizada deve ser-lhe atribuído um valor:
  - na altura da definição

```
double num = 10.5;
int idade = 18;
String cidade = "Aveiro";
```
  - usando uma instrução de atribuição (símbolo '=')

```
double peso;
peso = 50.5;
```
  - lendo um valor do teclado ou de outro dispositivo (ex. ficheiro)

```
double milhas;
milhas = sc.nextDouble("Valor real:");
```



# Conversões

- Sempre que uma expressão tenha operandos aritméticos de tipos diferentes, os operandos com menor capacidade de armazenamento são automaticamente convertidos para o tipo com maior capacidade:

```
byte -> short (ou char) ->  
int -> long -> float -> double
```

- A conversão inversa não é admitida e gera um erro de compilação.
- Podemos sempre forçar uma conversão através de um operador de conversão (*cast* em inglês):

```
double x;  
int y;  
y = (int)x; //estamos a forçar a conversão  
           para int
```

# Operadores e expressões (1)

- **Operadores:**

- Aritméticos:  $*$ ,  $/$ ,  $+$ ,  $-$ ,  $\%$
- Relacionais:  $<$ ,  $<=$ ,  $>$ ,  $>=$ ,  $==$ ,  $!=$
- Lógicos:  $!$ ,  $||$ ,  $\&\&$
- Condicional: `boolean-exp ? value0 : value1`
- Manipulação de bits:  $\&$ ,  $\sim$ ,  $|$ ,  $\wedge$ ,  $>>$ ,  $<<$

- **Expressões:**

```
int x;
```

```
double y;
```

```
x = 10 + 20; //o valor 30 é armazenado em x
```

```
y = 8.4 / 4.2; //o valor 2.0 é armazenado em y
```

## Operadores e expressões (2)

```
int x, z = 2;
```

```
String nome1, nome2 = "ana";
```

```
x = 10 + 20 * z;    //x armazena o valor 50
```

```
x = (10 + 20) * z; //x armazena o valor 60
```

- As expressões são calculadas da esquerda para a direita.
- Atenção às prioridades dos operadores e aos parênteses.

```
nome1 = nome2 + " maria";
```

```
// nome1 armazena "ana maria"
```

- O operador `+` aplicado aos Strings faz a concatenação.



# Operadores - prioridades

Operators	Associativity
[] . () (method call)	Left
! ~ ++ -- + (unary) - (unary) () (cast) new	Right
* / % (modulus)	Left
+ -	Left
<< >> >>> (arithmetic shift)	Left
< > <= >= instanceof	Left
== !=	Left
& (bitwise and)	Left
^ (bitwise exclusive or)	Left
(bitwise or)	Left
&& (logical and)	Left
(logical or)	Left
? : (conditional)	Left
= += -= *= /= <<= >>= >>>= &= ^=  =	Right

Operadores JAVA por prioridade decrescente



# Operadores aritméticos unários

- simétrico:  $-$  ( $-x$ )
- incremento de 1:  $++$  ( $++x$ ,  $x++$ )
- decremento de 1:  $--$  ( $--x$ ,  $x--$ )
- Os operadores unários de incremento e decremento só podem ser utilizados com variáveis e atualizam o seu valor de uma unidade.
- Colocados antes são pré-incremento e pré-decremento. Neste caso a variável é primeiro alterada antes de ser usada.
- Colocados depois são pós-incremento e pós-decremento e neste caso a variável é primeiro usada na expressão onde está inserida e depois atualizada.

# Algumas classes da linguagem JAVA

- A linguagem java disponibiliza um vasto conjunto de classes que permitem manipular dados e realizar diversas operações. Ficam alguns exemplos e serão apresentadas à medida das necessidades.
- **Classe Math:** <https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/lang/Math.html>
  - `double Math.cos(double);`
  - `double Math.acos(double);`
  - `double Math.sin(double);`
  - `double Math.asin(double);`
  - `double Math.sqrt(double);`
  - `double Math.pow(double, double);`
  - `double Math.toRadians(double);`
  - `double Math.random();`
- **Classe Integer e Double:**

• <code>Integer.MAX_VALUE</code>	<code>Double.MAX_VALUE</code>
• <code>Integer.MIN_VALUE</code>	<code>Double.MIN_VALUE</code>

# Algumas classes da linguagem JAVA (2)

- **Classe Character:**

`Character c;`

**Métodos que devolvem um valor booleano**

- `Character.isLetter(c)`
- `Character.isDigit(c)`
- `Character.isLetterOrDigit(c)`
- `Character.isLowerCase(c)`
- `Character.isUpperCase(c)`

**Métodos que devolvem um character**

- `Character.toLowerCase(c)`
- `Character.toUpperCase(c)`

- **Classe String:**

- `String s = "aveiro";`
- `s.length() → 6`
- `s.charAt(1) → 'v'`

# Leitura e escrita de dados

- **Classe Scanner:** Leitura do teclado (System.in)

- `import java.util.Scanner;`
- `nextInt()`, `nextDouble()`, `nextLine()`, `next()`...

- **Exemplos**

```
Scanner sc = new Scanner(System.in);  
int x; double y; String z;  
x = sc.nextInt();      // lê um valor inteiro  
y = sc.nextDouble();   // lê um valor decimal  
z = sc.nextLine();      // lê um string, toda a linha  
z = sc.next();          // lê um string, a próxima palavra
```

- **Classe System:** Escrita no terminal

- `print()`, `println()`, `printf()`;
- **Exemplos:**

```
System.out.print("O valor de x é " + x); // não muda de linha
```

```
System.out.println("O valor de x é " + x); // muda de linha
```

```
System.out.printf("O valor de x é %3d %n", x); // formatada
```



# Escrita formatada

- A função **printf** permite escrever informação formatada.

```
System.out.printf("formato de escrita", lista de variáveis);
```

- O formato de escrita é uma sequência de caracteres, que pode conter especificadores de conversão.
- O especificador de conversão é composto pelo símbolo **%** seguido de um caracter que indica qual o tipo de dados que queremos escrever (ver tabela de caracteres de conversão):

**%d, %f, %c, %s, ...**

- Este caracter pode ser precedido de um **número** com o qual se controla o formato e opcionalmente por uma **flag** (ver tabela):

**%03d, %5.1f, %3c, %-10s, ...**

- Exemplo:

```
System.out.printf("Int.: %6d", 15);           // Int.: _ _ _ _ 1 5
System.out.printf("Real: %6.2f", 14.2);       // Real: _ 1 4 . 2 0
```

# Escrita formatada `printf` – caracteres conversão

Conversion Character	Purpose	Example
d	Decimal integer	159
x or X	Hexadecimal integer	9f or 9F
o	Octal integer	237
f	Fixed floating-point	15.9
e or E	Exponential floating-point	1.59e+01 or 1.59E+01
g or G	General floating point: the shorter of e/E and f/F	—
a or A	Hexadecimal floating-point	0x1.fccdp3 or 0X1.FCCDP3
s or S	String	Java or JAVA
c or C	Character	j or J
b or B	boolean	false or FALSE
h or H	Hash code (see Chapter 4)	42628b2 or 42628B2
t or T	Date and time (obsolete; see Chapter 12 instead)	—
%	The percent symbol	%
n	The platform-dependent line separator	—



# Escrita formatada printf – flags de conversão

Flag	Purpose	Example
+	Prints sign for positive and negative numbers	+3333.33
space	Adds a space before positive numbers	_3333.33
-	Left-justifies field	3333.33__
0	Adds leading zeroes	003333.33
(	Encloses negative values in parentheses	(3333.33)
,	Uses group separators	3,333.33
# (for f format)	Always includes a decimal point	3333.
# (for x or o format)	Adds 0x or 0 prefix	0xcafe
\$	Specifies the index of the argument to be formatted; for example, %1\$d %1\$x prints the first argument in decimal and hexadecimal.	159 9f
<	Formats the same value as the previous specification; for example, %d %<x prints the same number in decimal and hexadecimal.	159 9f





# Tipos enumerados – enum (conjunto de constantes)

```
// Define dois tipos enumerados
// -- definidos fora do método main()
import java.util.*;
public class Enums {

    static Scanner ler = new Scanner(System.in);

    enum Day { SUNDAY, MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY,
SATURDAY }
    enum Month { JAN, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC }

    public static void main(String[] args) {
        Day dia;                // Declara uma variavel do tipo Day.
        Month mes;              // Declara uma variavel do tipo Month.
        dia = Day.FRIDAY;       // Atribui um valor do tipo Day ao dia.
        mes = Month.OCT;        // Atribui um valor do tipo Month ao mes.
        System.out.print("O meu signo é Balança, porque nasci em ");
        System.out.println(mes);
        System.out.print("que é o mês ");
        System.out.print( mes.ordinal() );
        System.out.println(" do ano.");
        System.out.println("(A contar de 0, lembrem-se!)");
        System.out.print("É agradável chegar a ");
        System.out.println(dia);
        System.out.println( "que é o " + dia.ordinal()
                             + "-to dia da semana.")
    }
}
```

O meu signo é Balança, porque nasci em OCT  
que é o mês 9 do ano.  
(A contar de 0, lembrem-se!)  
É agradável chegar a FRIDAY  
que é o 5-to dia da semana.



# Leitura e escrita de dados – redirecção e ficheiros

>java KmToMilhas < entrada.txt

Com o operador de redirecção < o programa lê os dados do ficheiro *entrada.txt* em vez do teclado.

>java KmToMilhas > saida.txt

Com o operador de redirecção > o programa escreve os dados no ficheiro *saida.txt* em vez do ecrã.

Isto é uma forma expedita de ler e escrever num ficheiro sem alterar nada no código, usando os métodos da classe *scanner* e *print*, *println* e *printf* da mesma forma, sem modificações.

## Questão 1

Qual é a diferença entre os tipos **int** e **long**?

- 1) Um guarda valores inteiros e outro decimais;
- 2) Guardam ambos valores decimais com precisão diferente;
- 3) Usam um número de *bytes* (e *bits*) diferente para armazenar valores inteiros;
- 4) Nenhuma das outras respostas é válida;

## Questão 1b

Qual é o resultado armazenado em r?

```
int k = 3;
```

```
double r = k/2 + 10.666;
```

- 1) 12.166
- 2) 11.666
- 3) 14
- 4) Erro de compilação