

# APROG – Algoritmia e Programação



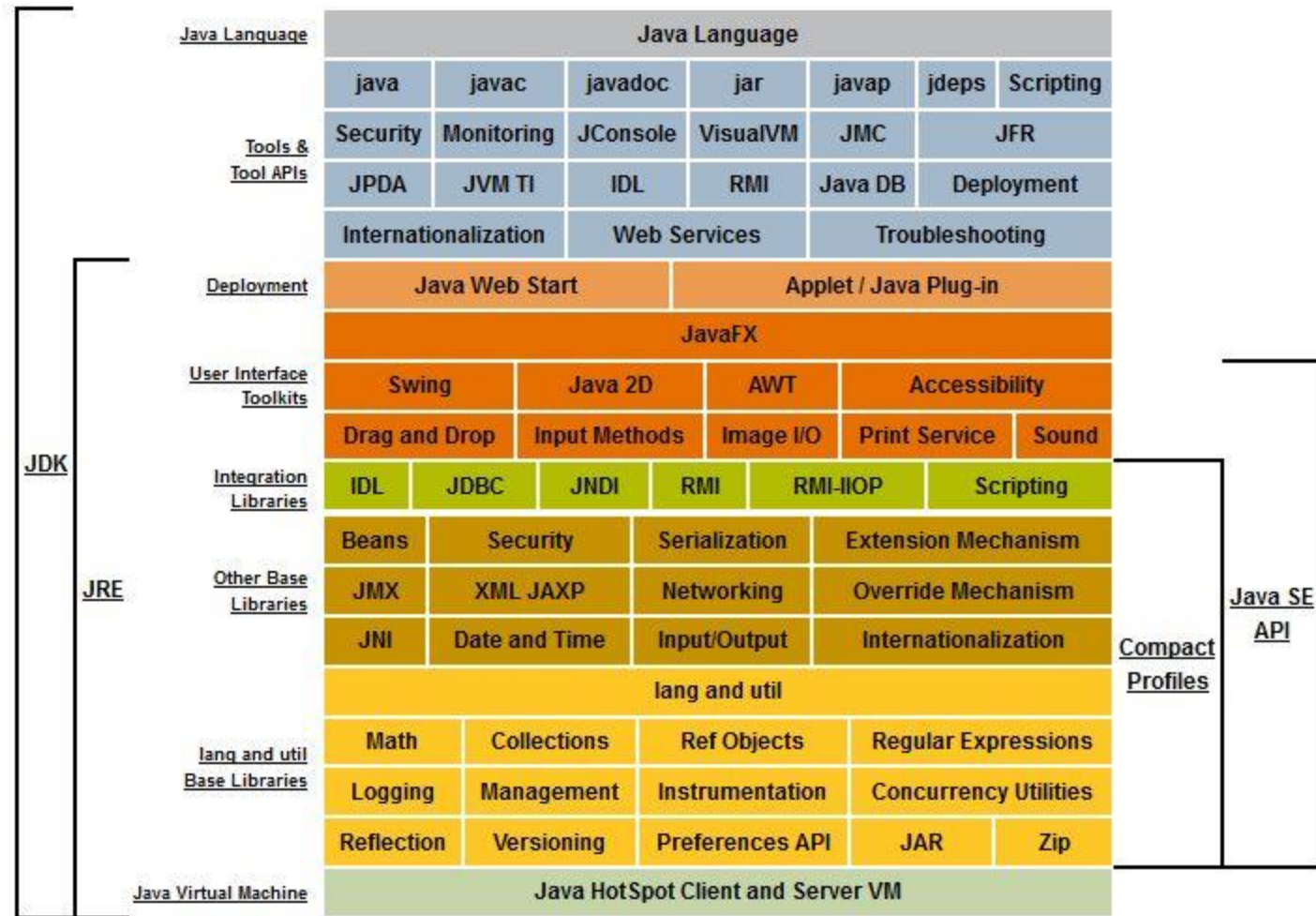
Emanuel Cunha Silva

[ecs@isep.ipp.pt](mailto:ecs@isep.ipp.pt)

# Apresentação

- É uma linguagem de programação
- O Java foi projetado para permitir o desenvolvimento de aplicações portáteis de alto desempenho para a mais ampla variedade possível de plataformas de computação
- Lançada em 1995 por Sun Microsystems
- É uma linguagem orientada a objetos
  - Objetos “são designados” por Classes
  - Todas as instruções que podem ser executadas têm de estar dentro de uma Classe

# Apresentação



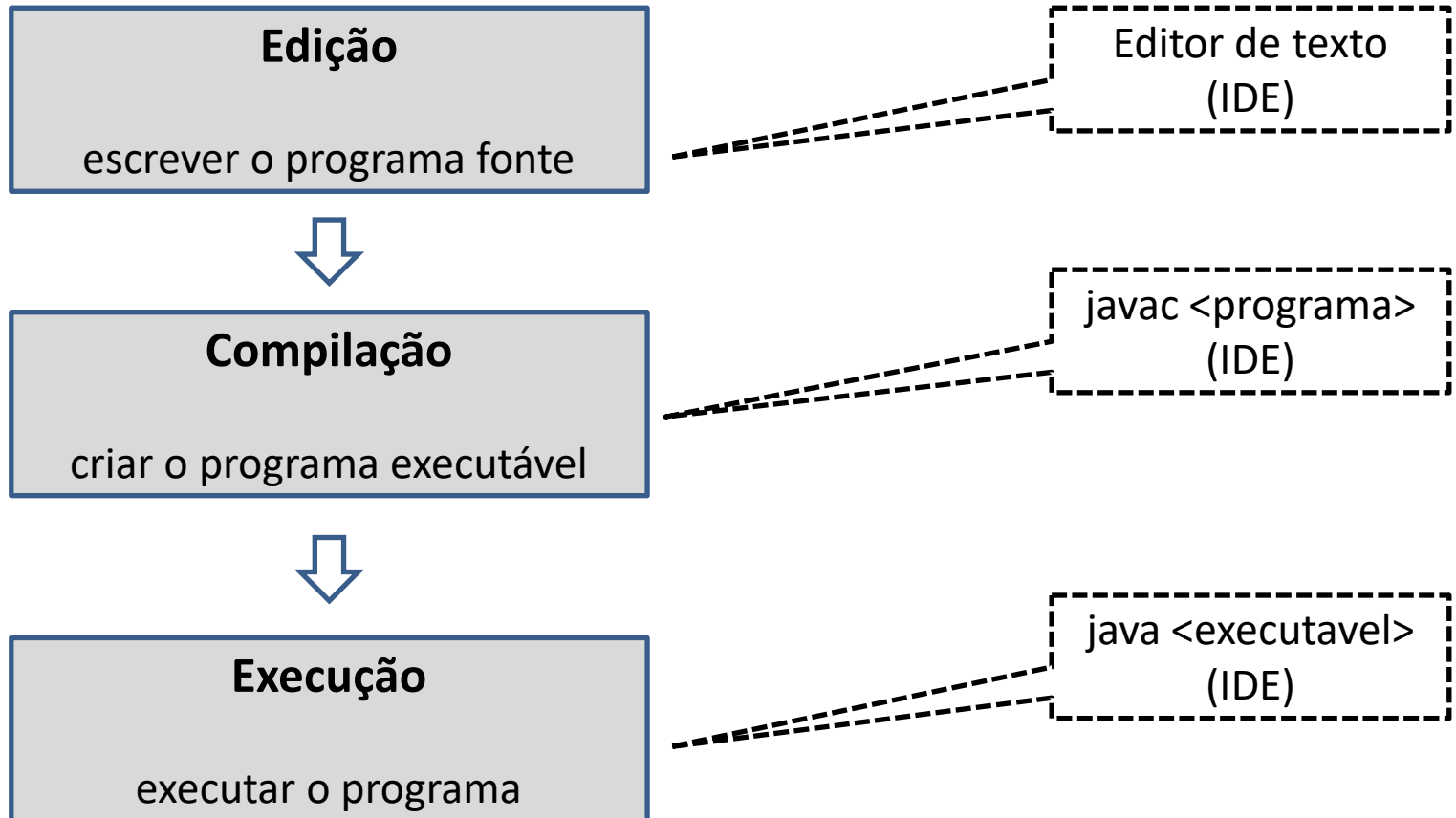
# Apresentação

## Desenvolver programas em Java

- Instalar o JDK (JDK JavaSE)
  - Download  
<https://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>
  - Java API (Classes já programadas do Java que podemos usar)  
<https://docs.oracle.com/javase/8/docs/api/>
  - Ferramentas
    - javac      compilador de código
    - java        interpretador de código
- Instalar um IDE (Integrated Developed Environment)
  - IntelliJ IDEA (<https://www.jetbrains.com/edu-products/download/>)
  - Netbeans (<https://netbeans.org/downloads/>)

# Apresentação

## Fases de desenvolvimento de um programa

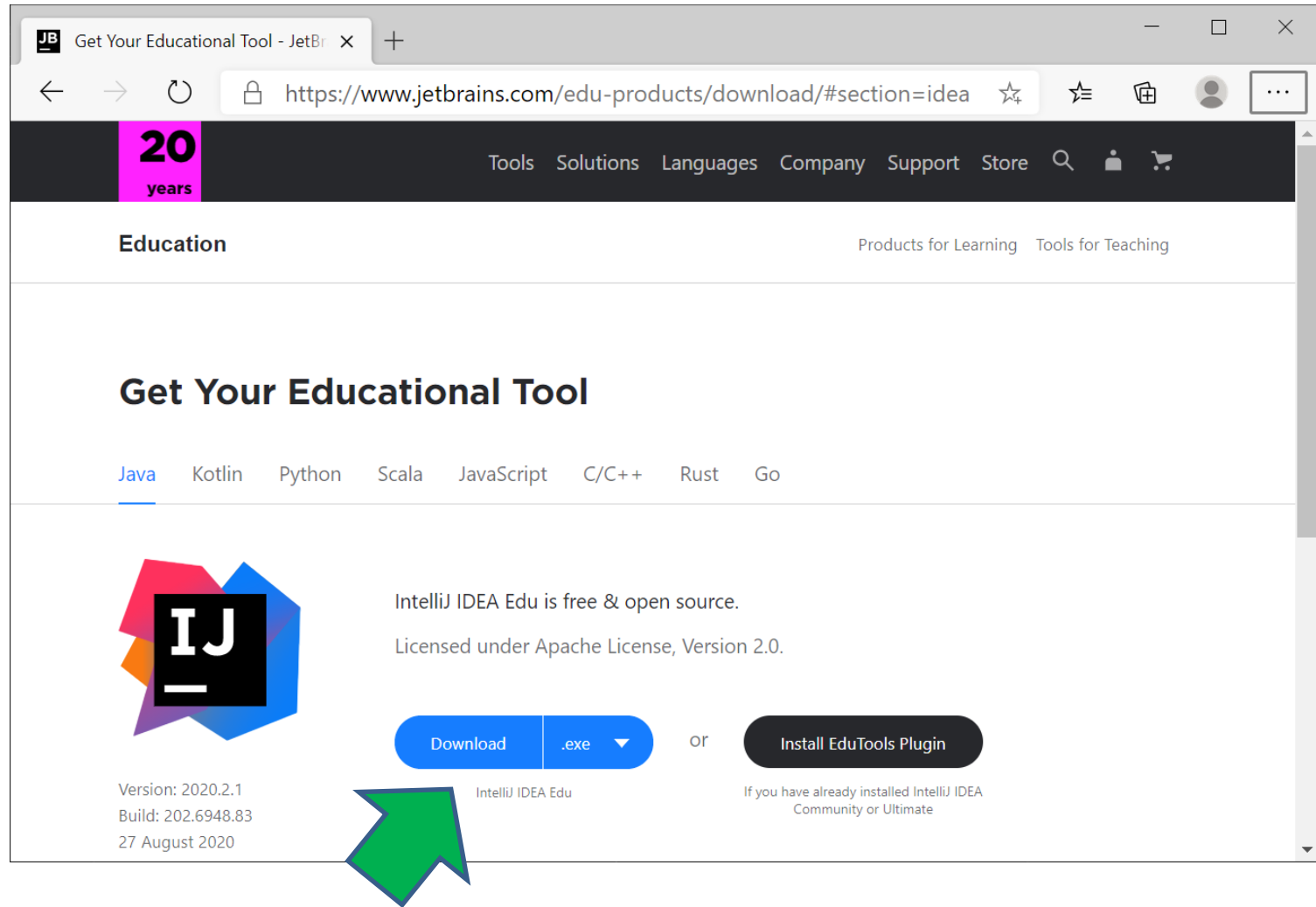


# Apresentação

Ex: Criar um programa em Java com o nome “Ola”

- Criar o ficheiro fonte “Ola.java”
  - Primeira letra maiúscula
  - Extensão “.java”
- Compilar o ficheiro fonte e criar o ficheiro executável
  - “javac Ola.java”
  - É criado o ficheiro executável “Ola.class”
- Executar o ficheiro executável
  - “java Ola”

# Apresentação



The screenshot shows a web browser window with the URL <https://www.jetbrains.com/edu-products/download/#section=idea>. The page features a dark header with the JetBrains logo, a '20 years' badge, and navigation links: Tools, Solutions, Languages, Company, Support, Store. Below the header, the 'Education' section is highlighted, with sub-links for 'Products for Learning' and 'Tools for Teaching'. The main heading is 'Get Your Educational Tool'. Underneath, there are tabs for different languages: Java (selected), Kotlin, Python, Scala, JavaScript, C/C++, Rust, and Go. The IntelliJ IDEA logo is displayed on the left. To its right, the text states: 'IntelliJ IDEA Edu is free & open source. Licensed under Apache License, Version 2.0.' Below this, there are two main options: 'Download .exe' (with a dropdown arrow) and 'Install EduTools Plugin'. A green arrow points to the 'Download .exe' button. At the bottom left, version information is provided: 'Version: 2020.2.1', 'Build: 202.6948.83', and '27 August 2020'. At the bottom right, a note says: 'If you have already installed IntelliJ IDEA Community or Ultimate'.

JB Get Your Educational Tool - JetBrains


20 years

Tools Solutions Languages Company Support Store

Education Products for Learning Tools for Teaching

## Get Your Educational Tool

Java Kotlin Python Scala JavaScript C/C++ Rust Go



IntelliJ IDEA Edu is free & open source.  
Licensed under Apache License, Version 2.0.

Download .exe or Install EduTools Plugin

Version: 2020.2.1  
Build: 202.6948.83  
27 August 2020

IntelliJ IDEA Edu

If you have already installed IntelliJ IDEA Community or Ultimate

# Java - Terminologia

- Classe

Definição de um tipo de dados que especifica os atributos (ou propriedades) e os comportamentos (ou métodos) disponíveis para os objetos dessa classe.

- Ex: projeto de uma casa, onde são definidos atributos como as cores, materiais, etc.

- Objeto

Instância de uma classe

- Ex: construir uma casa referente a um projeto, dando significado às propriedades desse projeto (cores, materiais)

- Instanciar um objeto

Criar e inicializar um objeto referente a um tipo de classe

- Pode-se criar vários objetos distintos com base na mesma classe, usando diferentes valores para as propriedades
  - Ex: construir várias casas com base no mesmo projeto, aplicando diferentes cores e materiais



# Java

- Composto por uma ou mais classes
- Uma classe possui:
  - dados (atributos)
  - métodos (funções) – módulos de código que contêm instruções para realizar uma tarefa e quando terminam voltam ao local onde foram invocados, podendo retornar um valor.
- Uma aplicação executável em java tem de possuir numa das suas classes o método *main* que é o local por onde é iniciada a execução

```
public class Ola {  
    public static void main (String[] args) {  
        ...  
    }  
    ...  
}
```

# Java

```
package nome;  
import nomeCompletoClasse;  
public class NomePrograma {  
    public static void main(String[] args) {  
        ...  
    }  
}
```

**NomePrograma** é nome da classe principal com inicial maiúscula

cabeçalho do método main é imutável

- Java distingue letras maiúsculas das minúsculas
- Declaração **package**
  - Especifica nome da package a que pertencerá nova classe
  - Packages permitem organizar classes
    - Semelhante às pastas dos sistemas de ficheiros que facilitam a gestão de ficheiros
    - Concretamente, são pastas dos sistemas de ficheiros

# Java

```
package nome;  
import nomeCompletoClasse;  
public class NomePrograma {  
    public static void main(String[] args) {  
        ...  
    }  
}
```

- Declaração **import**
  - Permite à nova classe usar classes pertencentes a outras packages
  - Nome completo de uma classe
    - nome da package seguido do nome da classe
    - Exemplo: `import java.util.Scanner;` // `java.util` é package da classe `Scanner`
- Modificador de acesso **public**
  - Especifica se classe/método pode ou não ser usada por outras classes
  - Classes/métodos públicos podem ser usados por outras classes

# Java

```
package nome;
```

```
...
```

```
public class NomePrograma {  
    public static void main(String[] args) {  
        ...  
    }  
}
```

- **NOTA:**  
Nesta fase inicial, e para manter os programas o mais simples possível, os mesmos não devem estar associados a qualquer package. Por isso, a referência a “**package**” não deve existir.

- Os programas incluem normalmente três tipos componentes
  - **Declarações** – reservar memória para o armazenamento das estruturas de dados
  - **Instruções** – indicar ao computador o que deve efetuar
    - As instruções são separadas por ponto e vírgula ( ; )
  - **Comentários** – registar anotações sobre o significado do código ou das estruturas de dados. Permitem também auxiliar na documentação do programa. São ignorados pelo computador
    - `//` - ignora até ao fim da linha em que se encontra
    - `/* ... */` - ignora tudo o que está no interior
    - `/** ... */` - ignora tudo o que está no interior. Usado para criar documentação automaticamente

# Java - Estruturas de dados

- Categorias
  - Variáveis - Valor pode variar durante a execução do programa
  - Constantes - Valor é constante durante a execução do programa
- Identificadores
  - Palavras usadas para identificar variáveis, constantes, classes, métodos, etc.
  - Pode conter letras, dígitos e os caracteres “\_” e “\$”
  - Não podem começar por um dígito
  - Java é *case sensitive* – maiúsculas e minúsculas são diferentes
    - `pessoa` ≠ `Pessoa` ≠ `PESSOA`

# Java - Identificadores

Os identificadores, em Java, normalmente, seguem a seguinte convenção

- **Variáveis e métodos**
  - Iniciam por letra minúscula
  - Podem ter várias palavras agregadas sem espaços e capitalizadas
    - Ex: `nomeAluno`, `alturaDoAtleta`, `numeroDoAtletaDeRemo`
- **Constantes**
  - Apenas maiúsculas
    - Ex: `MAXIMO`, `TAXA_IVA`
- **Classes**
  - Iniciam por letra maiúscula
  - Podem ter várias palavras agregadas sem espaços e capitalizadas
    - Ex: `Aluno`, `CarroDeF1`

# Java - Identificadores

**Palavras reservadas pelo java que não podem ser usadas como identificadores**

abstract	continue	for	new	switch
assert***	default	goto*	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum****	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp**	volatile
const*	float	native	super	while

\* not used

\*\* added in 1.2

\*\*\* added in 1.4

\*\*\*\* added in 5.0



# Java - Estruturas de dados

## Tipos de dados primitivos

### ■ Categorias

- Inteiros: 

<code>byte</code>	1 byte	(-128, 127)
<code>short</code>	2 bytes	(-32 768, 32 767)
<code>int</code>	4 bytes	(-2 147 483 648, 2 147 483 647)
<code>long</code>	8 bytes	(-9x10 <sup>18</sup> , 9x10 <sup>18</sup> )
- Reais: 

<code>float</code>	4 bytes	(-/+ 3.4 x 10 <sup>38</sup> )
<code>double</code>	8 bytes	(-/+ 1.7 x 10 <sup>308</sup> )
- Outros: 

<code>char</code>		
<code>boolean</code>		

### ■ Exemplos

```
char c = 'R', opcao = 'a';    // caracteres são delimitados por plicas
boolean flag = false;        // valores lógicos: false e true
double nota = 18.5;          // separador decimal é o . (ponto)
float altura = 1.83f;         // os valores float devem ser seguidos pela letra f
```

# Java - Estruturas de dados

## Tipos de dados não primitivos (referências)

- **Definidos por Classes**

- Exemplos

- String       // guarda texto

- Scanner      // leitura de dados (ex: teclado)

- Formatter    // escrita formatada (ex: ecrã)

- **Variável de Tipo Referência**

- Exemplos

- String cidade;

- String nome = "Nico";

- String ave = "águia";

# Java - Estruturas de dados

## Declaração de variáveis

- **Sintaxe** `// para N variáveis do mesmo tipo`  
`Tipo_de_Dado identificador_1[=valor inicial] [ , ..., identificador_N[=valor inicial] ] ;` [...] = opcional

- Exemplos

`int numero;` `// guarda inteiros; por omissão, variáveis numéricas são inicializadas a 0`

`int preco=5;` `// variável declarada e inicializada; = é operador de atribuição`

`int x=5, y, z=1;` `// múltiplas variáveis, do mesmo tipo, declaradas na mesma linha`

- **Local**

- Em qualquer parte do corpo do método

```
public class NomePrograma {  
    public static void main(String[] args) {  
        int numero;  
        ...  
        int preco = 5;  
        ...  
        int x = 5, y, z = 1;  
    }  
}
```

# Java - Estruturas de dados

## Declaração de constantes

- **Sintaxe**

// para N constantes do mesmo tipo

**final** Tipo\_de\_Dado identificador\_1=valor1 [ , ..., identificador\_N=valorN] ;

[...] = opcional

- Exemplos

**final int** NUMERO=10; // por convenção, nomes em maiúsculas

**final int** X=5, PRECO=5; // múltiplas constantes, do mesmo tipo, declaradas na mesma linha

- **Local**

- Em qualquer parte do corpo do método

```
public class NomePrograma {  
    public static void main(String[] args) {  
        final int NUMERO=10;  
        ...  
        final int X=5, PRECO = 5;  
        ...  
    }  
}
```

# Java - Operadores

## ▪ Aritméticos

+ // Soma  
- // Subtracção  
\* // Produto  
/ // Divisão real ou inteira; Divisão inteira se ambos os operandos forem inteiros.  
% // Resto da divisão inteira

## ▪ Relacionais

> // Maior  
>= // Maior ou igual  
< // Menor  
<= // Menor ou igual  
!= // Diferente  
== // Igual

## ▪ Atribuição

= // Ex: x = 5;  
+= // Ex: x += 5      ou      x = x + 5;  
-=  
\*=  
/=

# Java - Operadores

## ▪ Lógicos

&&    // AND

||    // OR

!    // NOT ;   Ex: !(x<0 && y>10)

## ▪ Outros

++    // Incrementa variável de 1 unidade;

--    // Decrementa variável de 1 unidade;

Ex: contador++    ou    ++contador

Ex: contador--    ou    --contador

# Java - Instrução atribuição

## ▪ Sintaxe

variável = expressão;      // expressão tem de resultar num tipo compatível com a variável

## ▪ Exemplos

```
public class NomePrograma {  
  
    public static void main(String[] args){  
        int x, y;  
        double z;  
        int d, idade;  
        ...  
        x = 5;  
        ...  
        d = idade - 12;           // idade e d têm de ser do mesmo tipo  
        ...  
        z = x * 10 - Math.pow(2,5); // Math.pow(2,5) = 25  
    }  
}
```

# Java - Instrução decisão

## ▪ Sintaxe

```
if (condição) {           // SE ... ENTÃO ... FSE
    // acção
}                          // { ... } ; obrigatório para acção com mais de 1 instrução
```

```
if (condição) {           // SE ... ENTÃO ... SENÃO ... FSE
    // acção 1
} else {
    // acção 2
}
```

```
                                // CASO ... SEJA
switch(expressão) {          // expressão tem de ser do tipo inteiro, carácter ou string
    case valor_1[: case valor_2: ...: case valor_m] : // acção_1; break;
    case valor_n[: case valor_o: ...: case valor_v] : // acção_2; break;
    ...
    default : // acção_por_omissão;                      //opcional
}
```



# Java - Instrução decisão

## Exemplos

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

```
        int x, y, z;
```

```
        ...
```

```
        if (x>5)
```

```
            y=4;
```

// {...} não é obrigatório para apenas uma instrução

```
        ...
```

```
        if(x>5 && x<10){
```

// {...} é obrigatório por haver mais de uma instrução

```
            y=10;
```

```
            z=100;
```

```
        }
```

```
        ...
```

```
        if (x>0) {
```

```
            System.out.println(x + "é um nº positivo");
```

```
        } else {
```

```
            System.out.println(x + "não é um nº positivo");
```

```
        }
```

```
    }
```

```
}
```

para imprimir no ecrã

# Java - Instrução decisão

## Exemplos

**Problema:** a partir da idade de uma pessoa indicar se é maior de idade ou, caso a idade seja negativa escrever inválido

```
public class NomePrograma {  
  
    public static void main(String[] args){  
        int idade;  
        ...  
        if (idade > 0)  
            if (idade >= 18)  
                System.out.println("maior de idade");  
            else  
                System.out.println("inválido");  
    }  
}
```

**Errado**

Exemplo:  
Idade = 10

```
public class NomePrograma {  
  
    public static void main(String[] args){  
        int idade;  
        ...  
        if (idade > 0) {  
            if (idade >= 18)  
                System.out.println("maior de idade");  
        } else  
            System.out.println("inválido");  
    }  
}
```

# Java - Instrução decisão

## ▪ Exemplos (com valores numéricos)

```
public class NomePrograma {  
    public static void main(String[] args){  
        int x, y, z;  
        ...  
        switch(x){  
            case 1: case 3:  
                z = y * 2;  
                break;  
            case 6:  
                z = 12;  
                break;  
            case 2: case 4: case 7:  
                y = 45;  
                z = 23;  
                break;  
            default:  
                z=50;  
        }  
        ...  
    }  
}
```

**break** termina execução do switch;  
senão, são executadas as instruções  
do caso seguinte

# Java - Instrução decisão

## ▪ Exemplos (com Strings)

```
public class NomePrograma {  
    public static void main(String[] args){  
        String mes;  
        ...  
        switch(mes){  
            case "janeiro": case "fevereiro":  
                ...  
                break;  
            case "marco":  
                ...  
                break;  
            default:  
                ...  
        }  
        ...  
    }  
}
```

**break** termina execução do switch;  
senão, são executadas as instruções  
do caso seguinte

# Java - Instrução repetição

## ▪ Sintaxe

```
while (condição) {                                     // ENQUANTO ... REPETIR
    // corpo do ciclo
}
```

```
do {                                                    // REPETIR ... ENQUANTO
    // corpo do ciclo
} while (condição);    // termina com ponto e vírgula ( ; )
```

```
for(inicialização; condição de funcionamento; passo){ // REPETIR PARA
    // corpo do ciclo
}
```

# Java - Instrução repetição



# Java - Instrução repetição

## ▪ Exemplos

```
public class NomePrograma {  
    public static void main(String[] args){  
        int hora, numero;  
        Scanner ler = new Scanner(System.in);  
        hora = 0;
```

```
        while (hora<=23) {  
            System.out.println(hora);  
            hora++;  
        }
```

```
        do {  
            numero = ler.nextInt();  
        } while (numero>0);
```

```
        for (hora=0; hora<=23; hora++) {           // REPETIR PARA hora←0 ATÉ 23 PASSO 1  
            System.out.println(hora);  
        }
```

```
        for (hora=23; hora>=0; hora--)           // for(...); não terminar com ponto-e-vírgula  
            System.out.println(hora);             se existirem instruções associadas à repetição
```

...

# Java – Blocos de Instruções

Um bloco de instruções refere-se a um conjunto de instruções que pertencem a um contexto comum. Um bloco é delimitado por chavetas **{ ... }**

- Uma variável declarada dentro de um bloco é local ao bloco.
  - Quando o bloco termina, a variável desaparece.
- Quando é necessário usar uma variável dentro e fora de um bloco então, ela deve ser declarada fora do bloco.

```
for (int x =1; x<10; x++) {  
    ...  
}  
System.out.println( x );
```

***Errado***

```
int x  
  
for (x =1; x<10; x++) {  
    ...  
}  
System.out.println( x );
```



# Java – Blocos de Instruções

```
int a;  
{  
    ...  
    {  
        int b;  
        ...  
        {  
            int c;  
            ...  
        }  
    }  
    {  
        int d;  
        ...  
    }  
}
```

Zonas de validade das variáveis

# Java – Conversão de tipos numéricos

- **Conversão implícita (sem perda de informação)**

- Conversão para tipos compatíveis com igual ou maior capacidade

**byte → short → int → long → double**

- Ex:     int a = 10;  
          double b = a;

- **Conversão explícita (com possível perda de informação)**

- Declarada explicitamente através de um *cast* porque pode originar perda de informação
- Sintaxe de um cast

- *(tipo destino)* nomeVariavel

- Ex:     double pi = 3.1415;  
          int a = **(int)** pi;

//parte decimal é descartada (a=3)

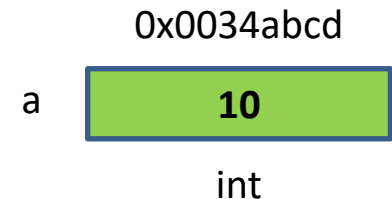
# Java – tipos de dados

Um tipo de dados é um conjunto de valores relacionados por um conjunto de operações

## Tipos primitivo (simples) **char, byte, short, int, long, float, double, boolean**

- Guardam valores atômicos, isto é, valores que não se podem decompor

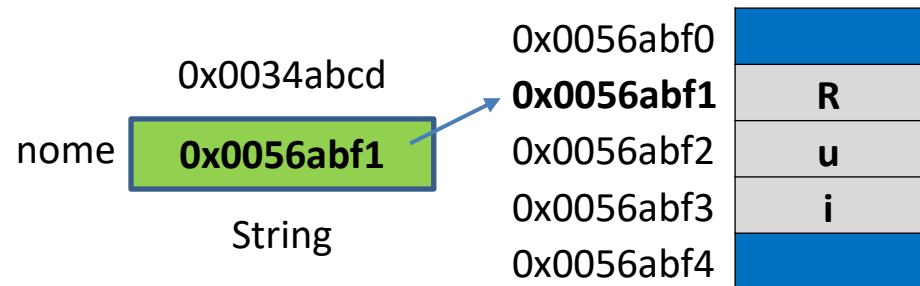
- Ex: `int a = 10;`  
`double b = 3.1415;`  
`boolean c = true;`



## Tipos referência **arrays, classes (String, ...)**

- Guardam o endereço de memória onde estão representados objetos

- Ex: `String nome = "Rui";`



**</Repetições>**



java.util

**Class Scanner**

java.lang

**Class Math**

java.lang

**Class String**

java.lang

**Class System**

# Java - Entrada de dados

Suportada por **objeto** da classe **Scanner** // **java.util.Scanner**

- **Classe Scanner**

- Serve para ler informação de uma entrada específica
  - Ficheiros de texto
  - Consola (teclado) é tratada como ficheiro de texto

- **Declaração de objeto Scanner para leitura da entrada padrão (teclado)**

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

- Cria objeto (instância) da classe Scanner (objeto *ler*)
- **System.in** representa dispositivo de entrada standard (teclado)
  - **System** é classe que representa sistema de computação
  - **in** é objeto da classe System que representa dispositivo de entrada padrão
    - Por omissão, é o teclado

# Java - Entrada de dados

## ▪ Leitura

- Preciso invocar método do objeto Scanner adequado ao tipo de dado a ler

- Sintaxe: `nomeObjetoScanner.nomeMétodo()`

- Exemplo: `ler.nextInt()` // para ler int

## ▪ Métodos de instância da classe Scanner que podem ser aplicados ao objeto (*ler*)

- `next()` Lê próxima string simples do teclado (i.e., cadeia de caracteres terminada pelo caráter espaço ou newline ('\n'))
- `nextLine()` Lê próxima linha do teclado (i.e., cadeia de caracteres terminada em \n)
- `nextInt()` Lê próximo int do teclado
- `nextLong()` Lê próximo long do teclado
- `nextFloat()` Lê próximo float do teclado
- `nextDouble()` Lê próximo double do teclado

# Java - Entrada de dados

## ■ Exemplos de instruções para ler informação e guardá-la numa variável

### ■ Leitura de inteiro:

```
int num = ler.nextInt();
```

### ■ Leitura de float:

```
float num = ler.nextFloat();
```

### ■ Leitura de strings:

```
String s = ler.next();           // palavra simples (apenas 1 palavra)  
//ex: "Ana", "maria", "domingo"
```

```
String s = ler.nextLine();       // frase (todas as palavras até ao fim da linha)  
//ex: "ana Maria", "Ola eu sou o Rui"
```



# Java - Entrada de dados

```
import java.util.Scanner;

public class NomePrograma {
    public static void main(String[] args){
        int numero;
        long altura;
        double peso;
        String nome, nomeComposto;
        ...
        Scanner ler = new Scanner(System.in);
        ...
        numero = ler.nextInt();
        altura = ler.nextLong();
        peso = ler.nextDouble();
        ...
        nome = ler.next();
        nomeComposto= ler.nextLine();
        ...
    }
}
```

// importa classe Scanner para usar em main

// declara um objeto Scanner para ler do teclado

// lê um int do teclado e guarda em numero  
// lê um long do teclado e guarda em altura  
// lê um double do teclado e guarda em peso

// lê uma string do teclado e guarda em nome  
// lê uma linha e guarda em nomeComposto

# Java - Entrada de dados

## ■ Problemas quando se combina a leitura de números, strings e linhas de texto

- Os métodos para leitura de números (`nextInt()`, `nextDouble()`, `nextFloat()`) apenas capturam os valores numéricos.
- O método para leitura de uma palavra (`next()`) apenas captura a sequência de caracteres até encontrar um espaço, tab ou fim de linha, mas não os inclui.
- Qualquer um dos métodos anteriores **não captura** o carácter de mudança de linha `'\n'`.
- Apenas o método `nextLine()` captura na íntegra uma linha de texto, incluindo o `'\n'`.
- Os caracteres não capturados pelos métodos podem ser capturados pelo método seguinte, originando problemas.

# Java - Entrada de dados

## ■ Problemas quando se combina a leitura de números, strings e linhas de texto

### ■ Exemplo

Pretende-se ler a idade e o nome de uma pessoa. Considere os valores 5 e “Rui Sá”.

```
Scanner ler = new Scanner(System.in);  
int idade = ler.nextInt();  
String nome = ler.nextLine();
```

Na leitura da idade foram pressionadas 2 teclas: [5]+[ENTER].

O método `nextInt()` capturou o valor 5 para a variável `idade` e deixou ficar o [ENTER].

De seguida o método `nextLine()` deteta que já existe conteúdo terminado por [ENTER] e captura-o para a variável `nome`.

Ou seja, `nome` fica vazio!!!!!!

# Java - Entrada de dados

## ■ Problemas quando se combina a leitura de números, strings e linhas de texto

### ■ Sugestões

1. Quando se pretende usar o método `nextLine()` conjuntamente com outros métodos de leitura deve-se “limpar”, imediatamente antes do `nextLine()`, qualquer conteúdo lido que possa existir, através da invocação do método `nextLine()` .

```
Scanner ler = new Scanner(System.in);  
int idade = ler.nextInt();  
ler.nextLine();  
String nome = ler.nextLine();
```

limpa o [ENTER] deixado  
pelo `nextInt()` anterior

2. Usar `nextLine()` para todas as leituras e converter os valores numéricos lidos através de métodos de classe apropriados, por exemplo:

```
int idade = Integer.parseInt( ler.nextLine() );  
float peso = Float.parseFloat( ler.nextLine() );  
double altura = Double.parseDouble( ler.nextLine() );
```

# Java - Saída de dados

Suportada pelos métodos **print**, **println** e **printf** do objeto **System.out**

- **System** é classe que representa sistema de computação
- **out** é objeto da classe System que representa dispositivo de saída padrão
  - Por omissão, ecrã
- Métodos de escrita
  - `System.out.print("mensagem");` //mantém-se na mesma linha
  - `System.out.println("mensagem");` //muda de linha
  - `System.out.printf(string_formatação, lista_parâmetros);`

# Java - Saída de dados

```
public class NomePrograma {  
    public static void main(String[] args){  
  
        ...  
        // Escreve mensagem e coloca cursor no fim  
  
        System.out.print("Operação impossível!");  
  
        ...  
        // Escreve mensagem e coloca cursor no início da linha seguinte  
  
        System.out.println("Divisão por zero!!");  
  
        ...  
        // Saída formatada; Escreve a media (double) com uma casa decimal. Exemplo: "Média=15.3"  
  
        double media = 15.3456;  
        System.out.printf("Média=%.1f",media);  
    }  
}
```

output

```
Operação impossível!Divisão por zero!!  
Média=15.3
```

# Java - Strings

## String

- Cadeia de caracteres = texto
- **Exemplos**
  - "ISEP"
  - "Algoritmia e Programação"
  - "JUPITER"

- **Declaração**

```
String v = "";                // String vazia
String s = "ISEP";
String nome = "Ana Berta Carla Dionísio";
```

# Java - Strings

## Carateres especiais

- Strings podem conter caracteres especiais com funções específicas (sequência de escape)
- Cada sequência de escape é um único carácter embora seja escrito com dois símbolos.

<code>\t</code>	Inserir um <i>tab</i> no texto
<code>\b</code>	Inserir um <i>backspace</i> no texto
<code>\n</code>	Inserir uma mudança de linha no texto
<code>\r</code>	Inserir um <i>carriage return</i> no texto
<code>\'</code>	Inserir uma plica no texto.
<code>\"</code>	Inserir uma aspa no texto.
<code>\\</code>	Inserir uma barra para trás ( <i>backslash</i> ) no texto

## ▪ Exemplos

```
String s = "Aprog\t, é \tfixe\t e fácil";
```

```
//s="Aprog   , é "fixe" e fácil"
```



# Java - Strings

## String

```
String s = "Bom dia, Rui";
```

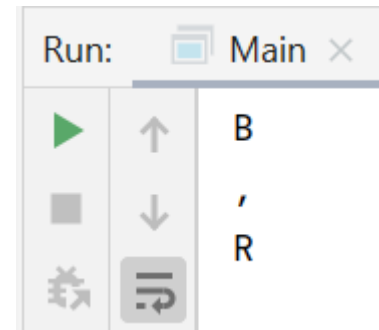
B	o	m		d	i	a	,		R	u	i
0	1	2	3	4	5	6	7	8	9	10	11

conteúdo

posições

a posição inicial é zero (0) e não um (1)

```
System.out.println( s.charAt(0) );  
System.out.println( s.charAt(7) );  
System.out.println( s.charAt(9) );
```



# Java - Strings

## Métodos

- int **length()**
- String **toLowerCase()**
- String **toUpperCase()**
- int **compareTo**(String outraString)
- boolean **equals**(String outraString)
- boolean **equalsIgnoreCase**(String outraString)
- String **trim()**
- char **charAt**( int índice )
- ...

# Java - Strings

- **Exemplos** // considerando: `String s = "Aprog";`

- `int length()`

- Retorna comprimento da string (quantidade de caracteres)

```
int comprimento = s.length();           // comprimento = 5
```

- `String toLowerCase()`

- Retorna a string com todas as letras minúsculas

```
String s2 = s.toLowerCase();           // s2 = "aprog"
```

- `String toUpperCase()`

- Retorna a string com todas as letras maiúsculas

```
String s2 = s.toUpperCase();           // s2 = "APROG"
```

# Java - Strings

- **Comparação de Strings**

**Não se pode usar os operadores relacionais para comparar Strings**

É necessário verificar se o conjunto de caracteres é o mesmo e pela mesma ordem

- **Exemplos** // considerando: `String s = "Aprog";`
- `int compareTo(String outraString)`
  - Compara duas strings alfabeticamente e retorna um número inteiro
    - Negativo                      Ex: `s.compareTo("PPROG")`    s é anterior a "PPROG"
    - Positivo                        Ex: `s.compareTo("ALIN")`     s é posterior a "ALIN"
    - Zero                             Ex: `s.compareTo("Aprog")`    s e "Aprog" são iguais

```
if ( s.compareTo("APROG")==0 )  
    System.out.println(" s=APROG ");  
else  
    System.out.println(" s≠APROG ");
```

# Java - Strings

- **Exemplos** // considerando: String s = "Aprog";

- boolean **equals**(String outraString)

- Distingue maiúsculas de minúsculas

s ≠ "APROG"

```
if ( s.equals("APROG") )  
    System.out.println(" s=APROG ");  
else  
    System.out.println(" s≠APROG ");
```

- boolean **equalsIgnoreCase**(String outraString)

- Compara duas strings alfabeticamente sem distinguir maiúsculas de minúsculas e retorna:

- true            caso sejam iguais
- false          caso sejam diferentes

s = "APROG"

```
if ( s.equalsIgnoreCase("APROG") )  
    System.out.println(" s=APROG ");  
else  
    System.out.println(" s≠APROG ");
```

# Java - Strings

- **Exemplos** // considerando: `String s = "Aprog";`

- `char charAt( int índice )`

- Retorna o caráter que se encontra na posição índice

```
char c = s.charAt(0);           // c = 'A'
```

```
char c = s.charAt(1);           // c = 'p'
```

- `String trim()`

- Retorna cópia da string sem espaços brancos iniciais e finais

```
String s1 = "    APROG    ";
```

```
String s2 = s1.trim();           // s2 = "APROG"
```

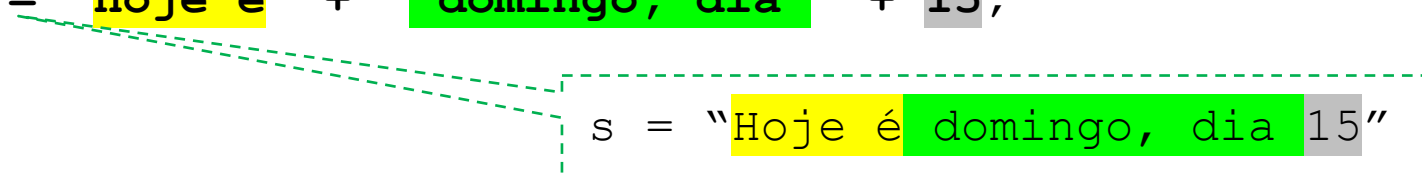
# Java - Strings

- **Concatenar Strings** - Usa-se o operador **+**

Permite criar uma nova string a partir da junção dos valores de várias strings ou números. O primeiro elemento da junção tem de ser uma string.

- **Exemplos**

```
String s = "Hoje é" + " domingo, dia " + 15;
```



```
s = "Hoje é domingo, dia 15"
```

```
String s1 = " D ";
```

```
String s2 = "ia";
```

```
int dia = 15;
```

```
String s = s1.trim() + s2 + dia;
```



```
s = "Dia15"
```

# Java - Math

Biblioteca que disponibiliza constantes matemáticas (**atributos** da classe) e operações numéricas básicas e funções trigonométricas (**métodos** da classe)

## ▪ Atributos

- final double **PI** // 3.14... Exemplo: **Math.PI**
- final double **E** // 2.71... Exemplo: **Math.E**

## ▪ Métodos de Classe

- double **sqrt**(double **a**) // Retorna raiz quadrada de **a**; Ex: **Math.sqrt(3)**;
- tipo **abs**(tipo **a**) // **tipo** = double, float, int ou long  
// Retorna valor absoluto de **a**. Ex: **Math.abs(-5)**
- double **pow**(double **a**, double **b**) // Retorna **a<sup>b</sup>**; Exemplo:  $2^5 \Leftrightarrow$  **Math.pow(2,5)**
- double **exp**(double **a**) // Retorna **e<sup>a</sup>**; Exemplo:  $e^5 \Leftrightarrow$  **Math.exp(5)**
- double **random**() // Retorna nº real aleatório do intervalo [0,1[
- double **cos**(double **angulo**) // Retorna coseno do **angulo** em radianos
- double **sin**(double **angulo**) // Retorna seno do **angulo** em radianos
- double **acos**(double **a**) // Retorna arco coseno de **a** em radianos



**</API>**