

**Programação**  
**Programação III**

# **Introdução à Linguagem Java**

**Marco Veloso**  
marco.veloso@estgoh.ipc.pt

# Agenda

## Introdução à Linguagem Java

### Introdução à Linguagem Java

Paradigmas de Programação

Linguagem Java

### Programação Orientada a Objectos

Objectos

Classes

Heranças

Polimorfismo

### Tratamento de Excepções

### Estruturas de dados

Tabelas unidimensionais

Tabelas multidimensionais

Vectores

Dicionários (*Hashtables*)

*Collections*

### Ficheiros

Manipulação do sistema de ficheiros

Ficheiros de Texto

Ficheiros Binários

Ficheiros de Objectos

Leitura de dados do dispositivo de entrada

**Resenha Histórica**

**Paradigmas de Programação**

**Linguagens Compiladas e Linguagens Interpretadas**

**Estrutura de um programa em Java**

**Variáveis e operadores**

**Controlo de fluxo**

**Plataforma Java e Ambiente de Desenvolvimento**

## Resenha Histórica

# História da Programação Orientada a Objectos

## Introdução à Linguagem Java

A ideia **Orientado a Objectos** (OO) teve origem na linguagem ***SIMULA-67*** (década de **1960** – Noruega) que já implementava alguns conceitos da **Programação Orientada a Objectos** (POO): **objectos**; **classes**; **subclasses**

Durante a década de **1970's**, a Xerox PARC desenvolveu a ***Smalltalk*** (primeira linguagem designada de POO) usando a plataforma ***SIMULA***

***Smalltalk*** foi lançada em 1981

A POO desenvolveu-se durante a década de 1990

Outras linguagens OO: ***Java***; ***C++***; ***Object Pascal*** (*Delphi*); ***C#***

# História da linguagem Java

## Introdução à Linguagem Java

Surgiu no início da década de **1990's**

Desenvolvida pela **Sun Microsystems** (James Gosling e Bill Joy), mais tarde adquirida pela **Oracle**

Concebida como **linguagem independente do computador** (*machine independent programming language*) onde será implementada a aplicação

Objectivo inicial: desenvolvimento de aplicações embebidas para a **world wide web** (*applets*).

Popular em redes

Java 2 – tornou-se plataforma de desenvolvimento de software com **interfaces gráficos**.

Referência para o desenvolvimento de **aplicações distribuídas**.

# Top Linguagens de Programação

## Introdução à Linguagem Java

### Programming Community Index

Sep 2019	Sep 2018	Change	Programming Language	Ratings	Change
1	1		Java	16.661%	-0.78%
2	2		C	15.205%	-0.24%
3	3		Python	9.874%	+2.22%
4	4		C++	5.635%	-1.76%
5	6	▲	C#	3.399%	+0.10%
6	5	▼	Visual Basic .NET	3.291%	-2.02%
7	8	▲	JavaScript	2.128%	-0.00%
8	9	▲	SQL	1.944%	-0.12%
9	7	▼	PHP	1.863%	-0.91%
10	10		Objective-C	1.840%	+0.33%

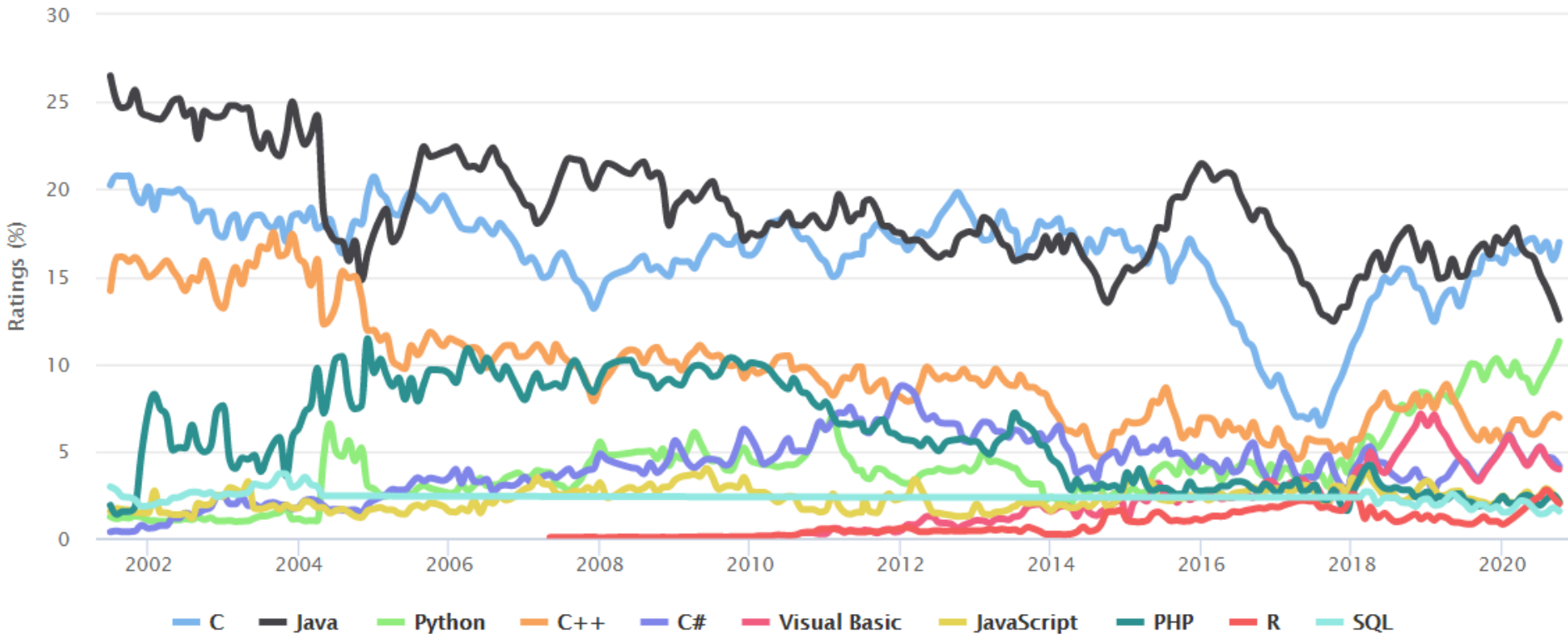
<http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html>

# Evolução Linguagens de Programação

## Introdução à Linguagem Java

### TIOBE Programming Community Index

Source: [www.tiobe.com](http://www.tiobe.com)


























# Top Linguagens de Programação























## Introdução à Linguagem Java

### IEEE Top Programming Languages

2015-2018

Language Rank	Types	Spectrum Ranking	Spectrum Ranking
1. Java	  	100.0	100.0
2. C	  	99.9	99.3
3. C++	  	99.4	95.5
4. Python	 	96.5	93.5
5. C#	  	91.3	92.4
6. R		84.8	
7. PHP		84.5	
8. JavaScript	 	83.0	
9. Ruby	 	76.2	
10. Matlab		72.4	

2019-2020

Language Rank	Types	Jobs Ranking	Spectrum Ranking
1. Python	  	100.0	100.0
2. Java	  	99.2	99.7
3. C	  	98.8	99.4
4. C++	  	94.6	97.4
5. C#	  	86.2	88.8
6. JavaScript	 	85.7	88.8
7. Assembly		83.4	86.2
8. PHP		83.1	82.3
9. HTML		81.3	77.2
10. Scala	 	76.5	76.5

<http://spectrum.ieee.org/computing/software/the-2015-top-ten-programming-languages>

## Paradigmas de Programação

Um **paradigma** pode-se definir como sendo um (a)

modelo

teoria

interpretação

ponto de vista

...

de uma **determinada realidade**

A partir de cada “visão” da realidade, escolhe-se a melhor forma de agir sobre ela

Em programação, um **paradigma** é uma forma de construir a estrutura e os elementos de um programa informático

## Paradigmas de Programação

*Programação Procedimental*

*Programação Estruturada*

*Programação Orientada a Objectos*

*Programação Funcional*

...

**Separação clara entre dados e código** (o código manipula os dados)

**Dividir um problema em pequenos problemas** pode levar um programador a criar um emaranhado de subprogramas que se chamam entre si.

Pode gerar:

- Grandes **dependências** no sistema

- Dificuldade na manutenção** (futura ou não) do código

- Não há muito reaproveitamento de código**, existindo, muitas vezes **duplicação de código**.

Consideram-se **objectos** que solucionem os problemas.

Os objectos possuem determinadas características que os definem e é **possível realizar determinadas acções** sobre eles.

Da **interacção** entre objectos surge a **solução** para o problema.

Assim, **não há distinção clara entre dados e código**

Os **dados passam a estar encapsulados** em objectos.

**Pensa-se em termos de objectos e suas relações**, em vez de dados e algoritmos.

Esta estrutura **facilita a gestão de grandes projectos**.

As desvantagens apresentadas para Programação Procedimental são “corrigidas” em Programação Orientada a Objectos.

Algumas vantagens de POO:

- aumento de **produtividade**

- reutilização de código**

- redução do número de linhas de código**

- maior **flexibilidade** do sistema

- facilidade na manutenção

O facto de se usar uma Linguagem Orientada para Objectos não significa que o programa siga o paradigma POO.

**Programação Orientada para Objectos também é (ou pode ser) Programação Estruturada.**

## Linguagens Compiladas e Linguagens Interpretadas

# Linguagens Compiladas e Linguagens Interpretadas

Introdução à Linguagem Java

## Compilador

**traduz**

programa que **converte** código escrito numa linguagem de alto nível para linguagem máquina (0's e 1's)

## Interpretador (ou Intérprete)

**interpreta**

programa que **traduz e executa, instrução a instrução**, o código da linguagem de alto nível para linguagem máquina



Realizada por **compilador** ou **interpretador**



### Compilador

O código executável gerado pelo compilador é dependente do sistema operativo da linguagem de máquina para o qual o código fonte foi traduzido.

Há geração de **código intermédio**

Linguagens: C; C++; Pascal; etc.

### Interpretador

Traduz o código linha a linha (lê e traduz).

O código fonte não é totalmente traduzido antes de ser executado.

Não há fases distintas **nem se produz código intermédio**.

Linguagens e tecnologias: Internet; Excel; Word; Access; Basic; SmallTalk; AutoLisp; Lisp; etc.

# Linguagens Compiladas e Linguagens Interpretadas

## Introdução à Linguagem Java

	Vantagens	Desvantagens
Compiladores	Execução mais <b>rápida</b>	Várias etapas de tradução
	Permite estruturas de programação mais completa	Programação final é maior, necessitando <b>mais memória</b> para a sua execução
	Permite a optimização do código fonte	Processo de correcção de erros e <b>depuração é mais demorado</b>
Interpretadores	Depuração do programa é mais simples	Execução do programa é mais <b>lenta</b>
	Consome <b>menos memória</b>	Estruturas de dados demasiado simples
	<b>Resultado imediato</b> do programa ou rotina desenvolvida	Necessário fornecer o programa fonte ao utilizador

# Linguagens Compiladas e Linguagens Interpretadas

## Introdução à Linguagem Java

Java é uma **linguagem compilada e interpretada**

Usa um **compilador** – `javac` – para fazer tradução dos programas.

Porém, esta tradução, em vez de ser feita para linguagem máquina, é feita para um código, chamado **bytecode** que é **independente das especificidades da máquina usada**.

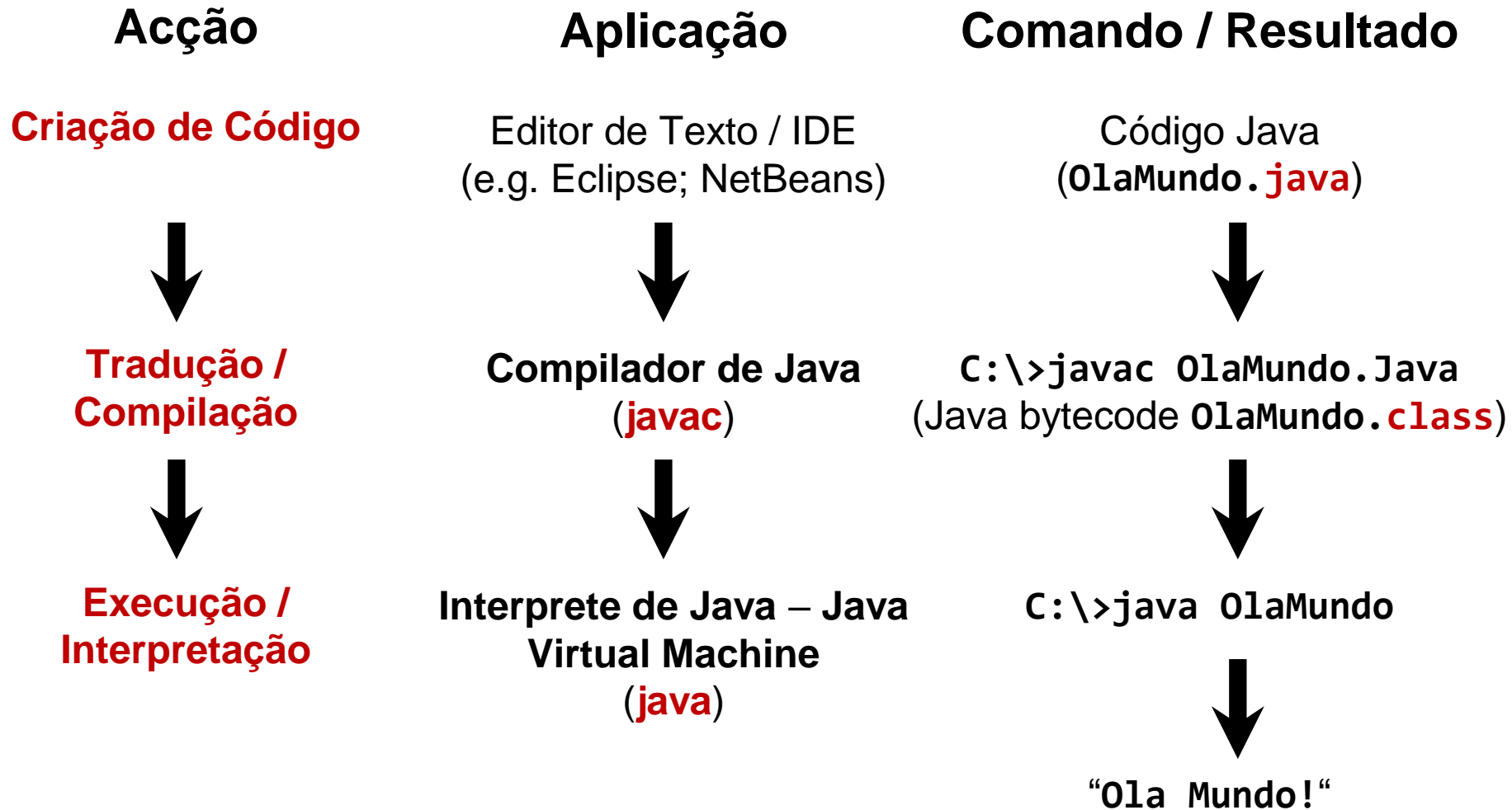
O *bytecode* é, depois interpretado por um **intérprete** – `java` – adequado à plataforma computacional usada (Windows, Unix, Mac-OS, ...).

BASIC é outra linguagem compilada e interpretada

## Estrutura de um programa em Java

# Java: Linguagem compilada e linguagem interpretada

Introdução à Linguagem Java



# Java: Linguagem compilada e linguagem interpretada

## Introdução à Linguagem Java

```
1 public class OlaMundo {  
2     public static void main (String [] args) {  
3         System.out.println("Olá Mundo!");  
4     }  
5 }
```



```
Administrator: C:\windows\system32\cmd.exe  
D:\Development\Eclipse\TesteJava>javac OlaMundo.java
```



```
1 Êp°%NULNULNUL4NULGS  
2 NULACKNULSI NULDLENULDC1BSNULDC2  
3 NULDC3NULDC4BELNULNAKBELNULSYNSOHNULACK  
4 NULNULNULSTXNULSO
```



```
Administrator: C:\windows\system32\cmd.exe  
D:\Development\Eclipse\TesteJava>java OlaMundo  
Picked up _JAVA_OPTIONS: -Xmx2048m  
Olá Mundo!
```

**Linguagem de alto nível**  
(código com significado humano)

**Compilação (JSE)**  
Multiplataforma,  
independente do CPU

**Linguagem de baixo nível**  
(*bytecode*)

**Interpretação (JRE ou JVM)**  
Específico do SO  
Output (resultado final)

# Estrutura de um programa em Java

## Introdução à Linguagem Java

Primeiro exemplo em Java:

```
TesteJava ▶ (default package) ▶ OlaMundo ▶ main(String[]) : void
1
2 /**
3  * Primeiro programa em Java
4  */
5
6 public class OlaMundo {
7
8     /**
9     * Método principal de execução
10    */
11
12    public static void main (String [] args) {
13
14        // Imprime a mensagem "Olá Mundo!"
15        System.out.println("Olá Mundo!");
16
17    }
18
19 }
```

# Estrutura de um programa em Java

## Introdução à Linguagem Java

Comentário `/**`  
(descreve a classe)

```
/**  
 * Primeiro programa em Java  
 */
```

Comentário `/**`  
(descreve o método)

```
/**  
 * Método principal de execução  
 */
```

Comentário de linha `//`

```
public static void main (String [] args) {  
    // Imprime a mensagem "Olá Mundo!"  
    System.out.println("Olá Mundo!");  
}
```

**Método** com nome *main*

Todos os programas necessitam do método *main* para serem executados

Invocação ao **método** `println()` sobre o **objecto** `out` da **classe** `System` com **argumento** `"Olá Mundo!"`  
Todas as instruções terminam com `;`

**Classe** com nome *OLaMundo*

O código deverá estar alojado num ficheiro com o mesmo nome da classe, neste caso *OLaMundo.java*.  
Por regra, cada classe tem o seu próprio ficheiro.

**Modificadores de acesso**

(*public, private, protected*)

**Argumentos de entrada**

do método: tipo / variável  
(tipo = `String []`;  
variável = `args`)

**Nome** único do método

Tipo de **retorno** do método  
(`void` significa sem valor de retorno)



# Método main

## Introdução à Linguagem Java

Um programa é composto por **uma ou mais classes**

Um programa necessita de uma classe que possua um **método main** para execução, com a seguinte assinatura:

```
public static void main (String [] args) {/* Code */}
```

**public** método público, acessível por classes externas;

**static** método estático, único, não pode ser instanciado, aplicável apenas a esta classe;

**void** método não retorna qualquer tipo de valor ou objecto;

**(String [] args)** argumentos de entrada do método, neste caso concreto o método recebe uma tabela (ou *array*) do tipo *String*. Os argumentos são passados no momento de execução, quando é invocado o comando

Comando

C: \> java OlaMundo 1 "dados"

Ficheiro com instruções java compiladas para execução

Argumento 1

Argumento 2

## Variáveis e operadores

# Tipo de variáveis

## Introdução à Linguagem Java

Tipo	Descrição	Tamanho	Mínimo	Máximo	Omissão
<b>boolean</b>	Lógico	1 bit	true, false		false
<b>char</b>	Character unicode	16 bit	\u0000 ou 0	\uFFFF ou 65535	\u0000
<b>byte</b>	Inteiro (com sinal)	8 bit	-128	127	0
<b>short</b>		16 bit	$-2^{15} = -32768$	$2^{15} - 1 = 32767$	0
<b>int</b>		32 bit	$-2^{31} = -2147483648$	$2^{31} - 1 = 2147483647$	0
<b>long</b>		64 bit	$-2^{63} =$ -9223372036854775808	$-2^{63} - 1 =$ 9223372036854775807	0L
<b>float</b>	Real (virgula flutuante)	32 bit	$-3.4 \times 10^{38}$	$3.4 \times 10^{38}$	0.0f
<b>double</b>		64 bit	$-1.8 \times 10^{308}$	$1.8 \times 10^{308}$	0.0
<b>void</b>	vazio				

**Nota:** tipos de **dados primitivos** (*primitive types*, 8): *boolean, char, byte, short, int, long, float, double*.

# Variáveis

## Introdução à Linguagem Java

As **variáveis** têm obrigatoriamente um **tipo**, **nome** e **valor** (se não for explicitamente atribuído pelo programador, no acto de criação recebe um valor por defeito).

O **nome** deve ser **claro e único** (evitar nomes abstractos como 'a' ou 'xpto').

**Declaração:** **TipoDados NomeVariável;**

**Declaração e atribuição de valor:** **TipoDados NomeVariável = Valor;**

```
char c = 'A', d = '\u0041';      int octonum = 00;
int num = 0;                     byte hexax = 0x0f;
float var = 0.0f;                long exay = 0xaf1;
```

Podem ser **declaradas várias variáveis na mesma declaração**. Neste caso, só é inserido um tipo de dados e os nomes das variáveis são separados por vírgula. Exemplo:

```
int a = 1, b = 2;
```

Para **cada variável primitiva** existe um classe designada **Wrapper Class** que inclui funcionalidades avançadas para criar e manipular objectos do tipo primitivo: **int** → **Integer**; **char** → **Character**; **boolean** → **Boolean**

Todas as **variáveis** que não sejam tipo primitivos são na realidade **Objectos**

Java disponibiliza variáveis (objectos) a partir de **classes pré-definidas**:

**Strings** conjunto de caracteres;

**Tabelas** estruturas organizadas com tamanho fixo e índices para acesso;

Exemplos:

```
String texto = "Olá Mundo!"; //variável texto representa um conjunto de caracteres
```

```
int[] lista = {1, 2, 3}; //tabela para armazenar 3 inteiros, inicializada
```

```
int[] tabela = new int [10]; //tabela para armazenar 10 inteiros, não inicializada
```

```
String[] nomes = {"João", "Ana"}; //tabela para armazenar 2 strings, inicializada
```

```
int i = nomes.length; //variável inteira que armazena o tamanho da tabela
```

```
System.out.println(texto + nomes[0]); //impressão do conteúdo da variável texto e o  
//conteúdo da posição 0 da tabela nomes
```

### Casting

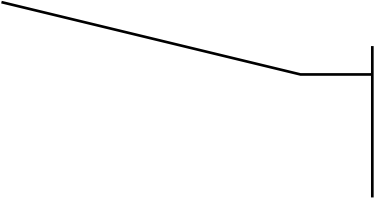
**Força** a consideração de um tipo de dados noutro, **podendo perder precisão**.

```
double d = 2.3d;
```

```
int i = 65;
```

```
int i = (int) d;
```

```
char c = (char) i;
```



**Cast:** forçar um valor do tipo *double* a ser armazenado numa variável do tipo *int*, com menor precisão (sem parte decimal)

### Conversão

**Transforma** o tipo de dados de uma variável para atribuição a outra variável.

Realizado através de métodos de conversão, disponíveis nas *wrapper classes*.

```
String numString = "123";
```

```
int num = Integer.parseInt(numString);
```

### Constantes

Distinguem-se das variáveis pelo uso da palavra reservada **final** antes da declaração.

Algumas regras no uso de constantes:

- nomes devem ser em **maiúsculas**;
- devem ser **declaradas no início do programa**.

```
public static void main(String[] args) {  
    final double JURO = 3.5;  
    //...  
}
```

### Aritméticos e de Atribuição

**+**, **-**      adição, subtração  
**\***, **/**      multiplicação, divisão (inteira se o dividendo e divisor são inteiros)  
**%**      resto da divisão inteira  
**exp++**, **exp--**, **++exp**, **--exp**  
**=**, **+=**, **-=**, **\*=**, **/=**, **%=**

### Lógicos (binários)

**||** (ou/*OR*), **&&** (e/*AND*), **^** (ou exclusivo/*XOR*), **==**, **!=**

### Lógicos (unários/*Bitwise*)

**!**      (complemento ou negação)      devolve valor lógico contrário  
**~**      (complemento)      complementa cada bit na representação interna do valor

### Relacionais ou de Comparação

**<**, **<=**, **>**, **>=**, **==**, **!=**



### Outros operadores

- >> deslocação (da representação interna) para a direita o número de vezes indicado pelo segundo operador
- >>> deslocação para a direita com extensão 0 como o anterior mas os bits inseridos à esquerda têm o valor 0
- ? Operador ternário: **condition ? if true : if false**

### Comandos

expressão de atribuição

formas pré-fixadas ou pós-fixadas de ++ e --

chamada de métodos (.)

criação de objectos (**new**)

comandos de controlo de fluxo

bloco = lista de comandos

```
x = 5;
```

```
x++ ;
```

```
System.out.println();
```

```
String s = new String();
```

```
while(x > 1) { /* ... */ }
```

# Precedência de operadores

## Introdução à Linguagem Java

### Maior precedência

```
++ -- + - ! ~ (cast)
* / %
+ -
<< >> >>>
< > <= >= instanceof
== !=
&
^
|
&&
||
?:
= *= /= %= += -= <<= >>= >>>= &= |=
```

### Menor precedência

## Controlo de fluxo

O **controlo de fluxo** de um programa pode ser realizado com as seguintes instruções, baseadas em testes lógicos:

**if-else**

**Seleccção simples**

**switch**

**Seleccção múltipla**

**while**

**Repetição** (condição de paragem no inicio)

**do-while**

**Repetição** (condição de paragem no fim)

**for**

**Repetição**

**labels-break**

**if** é uma estrutura **condicional simples**, composta por 2 elementos: **if**, **else**

```
if(condição)
{
    // bloco de instruções se condição é verdadeira
}
else
{
    // bloco de instruções se condição é falsa
}
```

**switch** é estrutura condicional de **selecção múltipla**, composta por 4 elementos: **switch**, **case**, **break**, **default**

```
switch(variável)
{
    case valor1:
        // bloco de instruções se variável = valor1
        break;
    case valor2:
        // bloco de instruções se variável = valor2
        break;
    //...
    [default:
        // bloco de instruções se variável diferente de todos os casos
        anteriores
    ]
}
```

O teste lógico funciona apenas com tipos de dados simples: **char**, **int**, **short**, **long** e **byte**

**while** é uma estrutura de repetição com **paragem no início**

```
while(condição)
{
    // bloco de instruções se condição verdadeira
}
```

**do-while** é uma estrutura de repetição com **paragem no fim**

```
do
{
    // bloco de instruções se condição verdadeira
}while(condição);
```

**for** é uma estrutura de **repetição**

```
for (inicialização; condição; incremento)
{
    //bloco de instruções
}
```

na manipulação de estruturas (e.g. tabelas) pode ser representada pela **forma contraída **for(each)**, sem condição de saída ou de incremento explícitos**, com o propósito de percorrer integralmente tabelas

```
for (variável de controlo: tabela)
{
    //bloco de instruções
}
```



# Controlo de Fluxo

## Introdução à Linguagem Java

O fluxo de dados pode ser interrompido com as instruções **break** ou **return**:

<b>break</b>	comando de interrupção da iteração
<b>continue</b>	comando de interrupção da iteração corrente
<b>return</b>	comando de retorno
<b>label</b>	comando de continuação de execução num determinado local do programa

```
cicloP:
{
    // ...
    for (...;...;...){
        // ...
        while(...){
            // ...
            if(...){
                // ...
                break cicloP;
            }
            // ...
        }
        // ...
    }
    // ...
}

//Código fora do ciclo principal
```

Instrução termina o fluxo normal do código, passando a execução para fora do ciclo

## **Entrada e saída de dados**

### **Interacção com o utilizador**

Classe **System** controla a entrada (*in*) e saída de dados (*out*)

A impressão de mensagens no ecrã corresponde à saída de dados para o dispositivo de saída por defeito (*out*): **System.out**

A impressão é realizada através de dois possíveis métodos, que recebem como argumento uma *String* (conjunto de caracteres):

**print()** Imprime uma mensagem,  
e.g. `System.out.print("Ola Mundo!");`

**println()** Imprime uma mensagem e muda de linha,  
e.g. `System.out.println("Ola Mundo!" + 100);`

Caracteres reservados	Resultado
<code>\t</code>	Tabulação ( <i>tab</i> )
<code>\n</code>	Nova linha ( <i>new line</i> )
<code>\'</code>	' ( <i>single quote</i> )
<code>\"</code>	" ( <i>double quote</i> )
<code>\\</code>	\ ( <i>backslash</i> )

# Saída de dados (formatados)

## Introdução à Linguagem Java

É possível formatar a saída de dados através do método `printf()`

```
System.out.printf("... %i$flagx.yt... \n...", var1, ..., varn);
```

***i*** i-ésimo argumento depois da vírgula – ***vari*** (de *var1* a *varn*)

***\$flag*** indicações formato do número:

- alinhar à esquerda
- + incluir o sinal do número
- 0 colocar zeros no início
- ' ' colocar espaços no início
- ( colocar parêntesis se número negativo

***x*** número de espaços a ocupar

***y*** número de casas decimais

***t*** tipo da variável:

<b><i>c</i></b>	caractere	<b><i>d</i></b>	inteiro
<b><i>b</i></b>	booleano	<b><i>f</i></b>	float
<b><i>o</i></b>	octal	<b><i>s</i></b>	string
<b><i>h</i></b>	hexadecimal	<b><i>e</i></b>	double

# Saída de dados (formatados)

## Introdução à Linguagem Java

### Código Exemplo:

```
float f=-123.456f, g=123.456f;
```

```
System.out.println("f= "+f+" g= "+g+".");
```

```
System.out.printf("f= %f g= %f.\n",f,g);
```

```
System.out.printf("f= %2$8.2f g= %1$8.2f.\n",f,g);
```

```
System.out.printf("f= %2$-8.2f g= %1$-8.2f.\n",f,g);
```

```
System.out.printf("f= %+8.2f g= %+8.2f.\n",f,g);
```

```
System.out.printf("f= %08.2f g= %08.2f.\n",f,g);
```

```
System.out.printf("f= % 8.2f g= % 8.2f.\n",f,g);
```

```
System.out.printf("f= %(8.2f g= %(8.2f.\n",f,g);
```

### Resultado:

```
f= -123.456 g= 123.456.
```

```
f= -123.456001 g= 123.456001.
```

```
f= 123.46 g= -123.46.
```

```
f= 123.46 g= -123.46 .
```

```
f= -123.46 g= +123.46.
```

```
f= -0123.46 g= 00123.46.
```

```
f= -123.46 g= 123.46.
```

```
f= (123.46) g= 123.46.
```

# Saída de dados (formatados)

## Introdução à Linguagem Java

É possível **formatar o estilo (cor) do texto na consola** do sistema. Esta funcionalidade dependerá da forma como a consola do **sistema operativo suporta e lida com codificação ANSI**, pelo que em alguns sistemas operativos os comandos podem não ter efeito

Para o efeito segue-se a sintaxe:

```
Escape_character[<<code>>m <<output text>>
```

**Caracteres de escape** usuais: `\033` ou `\u001B`

**Code** possui os seguintes parâmetros, usando o formato **cor;estilo**, ou seja, separando os vários estilos por ';' :

- 0** Remove todas formatações (Reset)
- 1** Negrito (**Bold**)
- 3** Itálico (*Italic*)
- 4** Sublinhado (Underline)
- 30-37** Cada número representa uma cor (30 = Preto; 31 = Vermelho; 32 = Verde; 33 = Amarelo; 34 = Azul; 35 = Roxo; 36 = Cinzento; 37 = Branco)

Para **cor de fundo (background)**, o número de codificação da cor inicia pelo dígito 4 (40 = Preto; 41 = Vermelho; 42 = Verde; 43 = Amarelo; 44 = Azul; 45 = Roxo; 46 = Cinzento; 47 = Branco)

# Saída de dados (formatados)

## Introdução à Linguagem Java

```
public class Coloring
{
    public static void main(String args[])
    {
        System.out.println("\033[31;1mHello
                           \033[0m,
                           \033[32;1mworld!
                           \033[0m");
        System.out.println("\033[31mRed
                           \033[32m, Green
                           \033[33m, Yellow
                           \033[34m, Blue
                           \033[0m");
    }
}
```

### Output

```
Hello, world!
Red, Green, Yellow, Blue
```

# Saída de dados (formatados)

## Introdução à Linguagem Java

Por uma questão de organização, é comum **usar constantes**, declaradas no início de cada classe:

```
ANSI_RESET    = "\u001B[0m";  
ANSI_BLACK    = "\u001B[30m";  
ANSI_RED      = "\u001B[31m";  
ANSI_GREEN    = "\u001B[32m";  
ANSI_YELLOW   = "\u001B[33m";  
ANSI_BLUE     = "\u001B[34m";  
ANSI_PURPLE   = "\u001B[35m";  
ANSI_CYAN     = "\u001B[36m";  
ANSI_WHITE    = "\u001B[37m";
```

```
System.out.println(ANSI_RED + "This text is red!" + ANSI_RESET);
```

De notar que, por boa prática, é definido o formato antes do texto e removido (*reset*) após o envio da mensagem para a consola



# Saída de dados (formatados – esquemas de cores)

## Introdução à Linguagem Java

### *// Reset*

```
RESET = "\033[0m"; // Text Reset
```

### *// Regular Colors*

```
BLACK = "\033[0;30m"; // BLACK
RED = "\033[0;31m"; // RED
GREEN = "\033[0;32m"; // GREEN
YELLOW = "\033[0;33m"; // YELLOW
BLUE = "\033[0;34m"; // BLUE
PURPLE = "\033[0;35m"; // PURPLE
CYAN = "\033[0;36m"; // CYAN
WHITE = "\033[0;37m"; // WHITE
```

### *// Bold*

```
BLACK_BOLD = "\033[1;30m"; // BLACK
RED_BOLD = "\033[1;31m"; // RED
GREEN_BOLD = "\033[1;32m"; // GREEN
YELLOW_BOLD = "\033[1;33m"; // YELLOW
BLUE_BOLD = "\033[1;34m"; // BLUE
PURPLE_BOLD = "\033[1;35m"; // PURPLE
CYAN_BOLD = "\033[1;36m"; // CYAN
WHITE_BOLD = "\033[1;37m"; // WHITE
```

### *// Underline*

```
BLACK_UNDERLINED = "\033[4;30m"; // BLACK
RED_UNDERLINED = "\033[4;31m"; // RED
GREEN_UNDERLINED = "\033[4;32m"; // GREEN
YELLOW_UNDERLINED = "\033[4;33m"; // YELLOW
BLUE_UNDERLINED = "\033[4;34m"; // BLUE
PURPLE_UNDERLINED = "\033[4;35m"; // PURPLE
CYAN_UNDERLINED = "\033[4;36m"; // CYAN
WHITE_UNDERLINED = "\033[4;37m"; // WHITE
```

### *// Background*

```
BLACK_BACKGROUND = "\033[40m"; // BLACK
RED_BACKGROUND = "\033[41m"; // RED
GREEN_BACKGROUND = "\033[42m"; // GREEN
YELLOW_BACKGROUND = "\033[43m"; // YELLOW
BLUE_BACKGROUND = "\033[44m"; // BLUE
PURPLE_BACKGROUND = "\033[45m"; // PURPLE
CYAN_BACKGROUND = "\033[46m"; // CYAN
WHITE_BACKGROUND = "\033[47m"; // WHITE
```

### *// High Intensity*

```
BLACK_BRIGHT = "\033[0;90m"; // BLACK
RED_BRIGHT = "\033[0;91m"; // RED
GREEN_BRIGHT = "\033[0;92m"; // GREEN
YELLOW_BRIGHT = "\033[0;93m"; // YELLOW
BLUE_BRIGHT = "\033[0;94m"; // BLUE
PURPLE_BRIGHT = "\033[0;95m"; // PURPLE
CYAN_BRIGHT = "\033[0;96m"; // CYAN
WHITE_BRIGHT = "\033[0;97m"; // WHITE
```

### *// Bold High Intensity*

```
BLACK_BOLD_BRIGHT = "\033[1;90m"; // BLACK
RED_BOLD_BRIGHT = "\033[1;91m"; // RED
GREEN_BOLD_BRIGHT = "\033[1;92m"; // GREEN
YELLOW_BOLD_BRIGHT = "\033[1;93m"; // YELLOW
BLUE_BOLD_BRIGHT = "\033[1;94m"; // BLUE
PURPLE_BOLD_BRIGHT = "\033[1;95m"; // PURPLE
CYAN_BOLD_BRIGHT = "\033[1;96m"; // CYAN
WHITE_BOLD_BRIGHT = "\033[1;97m"; // WHITE
```

### *// High Intensity backgrounds*

```
BLACK_BACKGROUND_BRIGHT = "\033[0;100m"; // BLACK
RED_BACKGROUND_BRIGHT = "\033[0;101m"; // RED
GREEN_BACKGROUND_BRIGHT = "\033[0;102m"; // GREEN
YELLOW_BACKGROUND_BRIGHT = "\033[0;103m"; // YELLOW
BLUE_BACKGROUND_BRIGHT = "\033[0;104m"; // BLUE
PURPLE_BACKGROUND_BRIGHT = "\033[0;105m"; // PURPLE
CYAN_BACKGROUND_BRIGHT = "\033[0;106m"; // CYAN
WHITE_BACKGROUND_BRIGHT = "\033[0;107m"; // WHITE
```

A **leitura de dados a partir do teclado** pode ser realizada através da classe **System.in** que permite ler caracteres com o método **read()**

```
import java.io.IOException;

public class LeChar
{
    public static void main(String[] args)
                                throws java.io.IOException
    {
        char c = 'a';
        System.out.println("inserir caractere: ");
        c = (char) System.in.read();
        System.out.print("caractere escrito: "+c);
    }
}
```

# Entrada de dados

## Introdução à Linguagem Java

Para realizar a leitura de outros tipos de dados deve-se recorrer à classe **Scanner** que disponibiliza métodos de acordo com o tipo de dados (e.g. `readInt()`, `readFloat()`).

De notar que a classe **Scanner** não pertence à package `java.lang` (automaticamente importada) mas sim à package `java.util`, pelo que tem que ser explicitamente importada na primeira linha

```
import java.util.Scanner;

public class LeDados
{
    public static void main(String[] args)
    {
        Scanner teclado = new Scanner(System.in);

        System.out.print("inserir string: ");
        String b = teclado.next();

        System.out.print("inserir valor inteiro: ");
        int c = teclado.nextInt();

        System.out.print("inserir valor real: ");
        float d = teclado.nextFloat();

        System.out.println("string: "+b+"\n inteiro: "+c+"\n real: "+d);
    }
}
```

A invocação do método `nextInt()` **não interpreta a última quebra de linha** (Enter) do valor lido, uma vez que apenas interpreta números e não caracteres alfanuméricos.

Desta forma, o `nextLine()` seguinte irá receber como primeira entrada uma quebra de linha.

A solução simples será após a invocação do `nextInt()` realizar uma invocação ao `nextLine()`:

```
Scanner teclado = new Scanner(System.in);  
  
int valor = teclado.nextInt();  
teclado.nextLine(); // limpa a quebra de linha  
String palavra = teclado.nextLine();
```

Para simplificação, é disponibilizada a classe **LeituraDados** que permite ler diferentes tipos de dados, disponibilizando métodos específicos para cada tipo de dados

```
import LeituraDados;

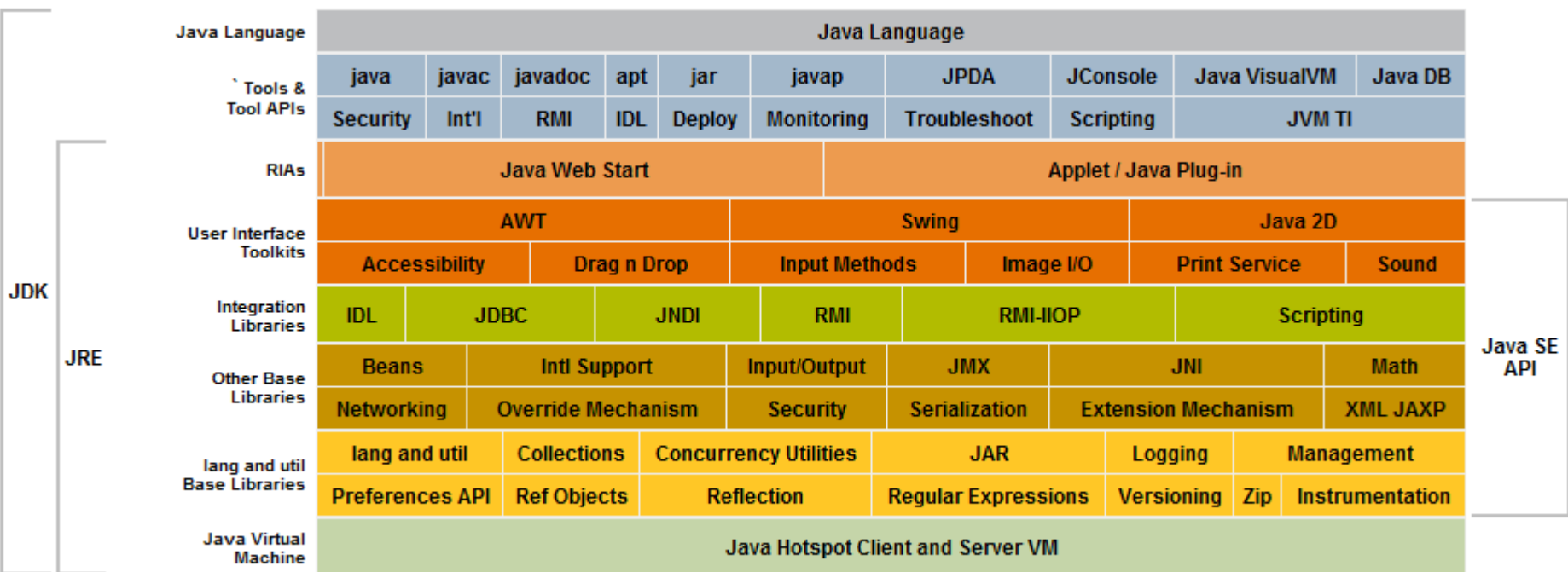
public class LeDados
{
    public static void main(String[] args)
    {
        System.out.print("inserir string: ");
        String a = LeituraDados.LeituraString();
        System.out.println("string: "+a);

        System.out.print("inserir inteiro: ");
        int b = LeituraDados.LeituraInt();
        System.out.println("inteiro: "+b);
    }
}
```

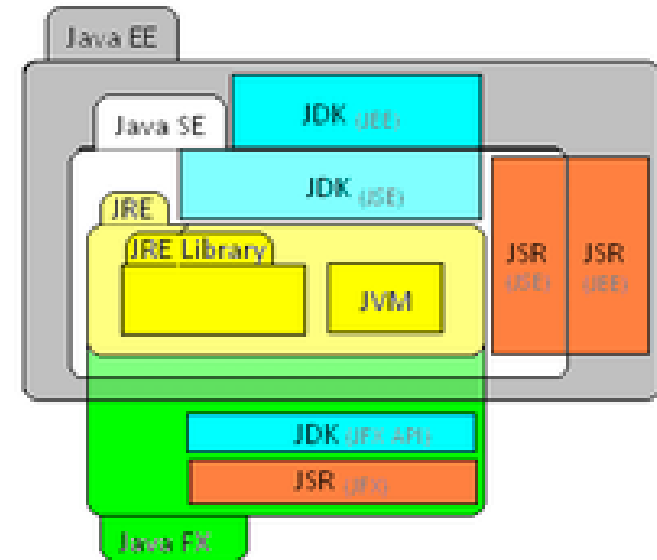
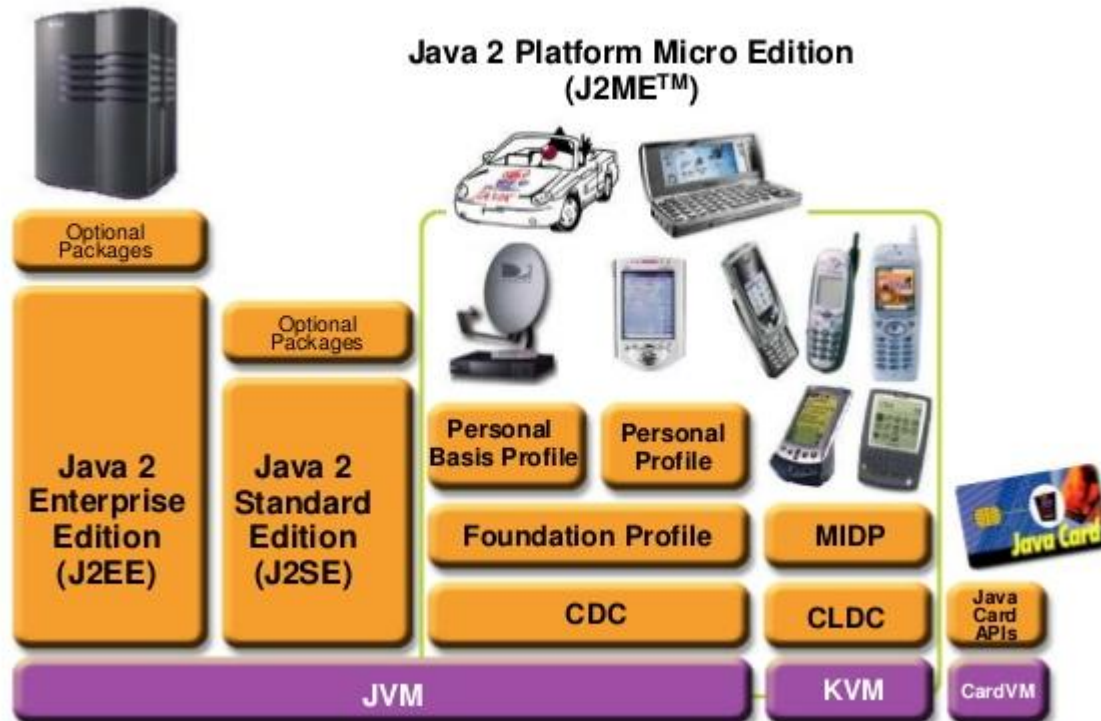
## **Plataforma Java e Ambiente de Desenvolvimento**

# Plataforma Java

## Introdução à Linguagem Java



### The Java™ Platform





A **biblioteca Java** é uma **coleção de Classes**, organizadas por ***packages*** (directorias de classes com funções análogas). Cada classe é composta por métodos que, sendo públicos, podem ser invocados.

Uma descrição das funcionalidades destas classes e *packages* está disponível em <http://docs.oracle.com/javase/tutorial/>

Adicionalmente, a **Application Program Interface** (API), disponível em <https://docs.oracle.com/javase/8/docs/api/> ou <https://docs.oracle.com/en/java/javase/15/docs/api/> descreve os métodos públicos, passíveis de serem invocados pelos programadores

Duas classes são de extrema utilidade diária:

**String** (`java.lang.String`) representa e manipula cadeias de caracteres  
<https://docs.oracle.com/javase/8/docs/api/java/lang/String.html>

**Math** (`java.lang.Math`) disponibiliza métodos para operações aritméticas  
<https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html>

### Classe **Math**

#### Métodos públicos:

```
static double sqrt(double a)
static double abs(double a)
static float abs(float a)
static int abs(int a)
static long abs(long a)
static double random()
static double pow(double a, double b)
static int round(float a)
static double toDegrees(double angrad)
```

### Class **String**

#### Métodos públicos:

```
char charAt(int index)
boolean endsWith(String suffix)
int lastIndexOf(int ch)
int indexOf(String str)
int length()
String toLowerCase()
String concat(String str)
boolean equals(Object anObject)
boolean equalsIgnoreCase(String aString)
```

As classes **Math** e **String** como outras classes essenciais na programação, nomeadamente as *Wrapper Classes*, **System**, **Thread** ou **StringBuffer**, fazem parte da **package java.lang**, importada automaticamente em cada programa

# Biblioteca Java e API

## Introdução à Linguagem Java

### Packages

java.applet  
java.awt  
java.awt.color  
java.awt.datatransfer  
java.awt.dnd  
java.awt.event  
java.awt.font  
java.awt.geom  
java.awt.im  
java.awt.im.spi  
java.awt.image  
java.awt.image.renderable

### Classes

Boolean  
Byte  
Character  
Character.Subset  
Character.UnicodeBlock  
Class  
ClassLoader  
ClassValue  
Compiler  
Double  
Enum  
Float  
InheritableThreadLocal  
Integer  
Long  
Math  
Number  
Object  
Package  
Process  
ProcessBuilder  
ProcessBuilder.Redirect  
Runtime  
RuntimePermission  
SecurityManager  
Short  
StackTraceElement  
StrictMath  
String  
StringBuffer  
StringBuilder  
System  
Thread

### Method Summary

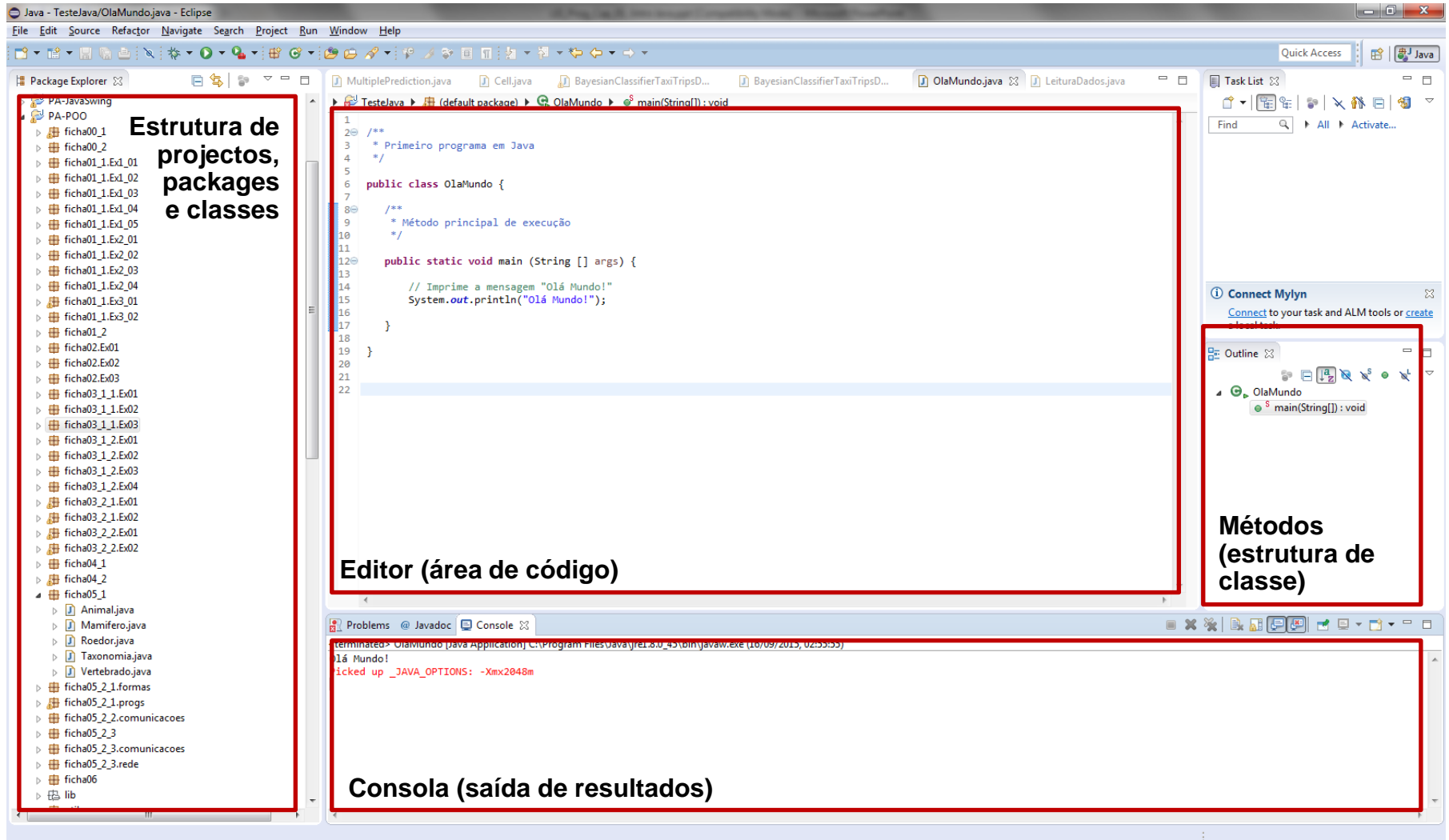
All Methods	Static Methods	Instance Methods	Concrete Methods	Deprecated Methods
Modifier and Type		Method and Description		
char		<b>charAt</b> (int index)	Returns the char value at the specified index.	
int		<b>codePointAt</b> (int index)	Returns the character (Unicode code point) at the specified index.	
int		<b>codePointBefore</b> (int index)	Returns the character (Unicode code point) before the specified index.	
int		<b>codePointCount</b> (int beginIndex, int endIndex)	Returns the number of Unicode code points in the specified text range of this String.	
int		<b>compareTo</b> (String anotherString)	Compares two strings lexicographically.	
int		<b>compareToIgnoreCase</b> (String str)	Compares two strings lexicographically, ignoring case differences.	
String		<b>concat</b> (String str)	Concatenates the specified string to the end of this string.	
boolean		<b>contains</b> (CharSequence s)	Returns true if and only if this string contains the specified sequence of char values.	
boolean		<b>contentEquals</b> (CharSequence cs)	Compares this string to the specified CharSequence.	
boolean		<b>contentEquals</b> (StringBuffer sb)	Compares this string to the specified StringBuffer.	
static String		<b>copyValueOf</b> (char[] data)	Equivalent to <b>valueOf(char[])</b> .	
static String		<b>copyValueOf</b> (char[] data, int offset, int count)	Equivalent to <b>valueOf(char[], int, int)</b> .	
boolean		<b>endsWith</b> (String suffix)		

API <https://docs.oracle.com/javase/8/docs/api/>

<https://docs.oracle.com/en/java/javase/15/docs/api/>

# Ambiente de Desenvolvimento (IDE)

## Introdução à Linguagem Java



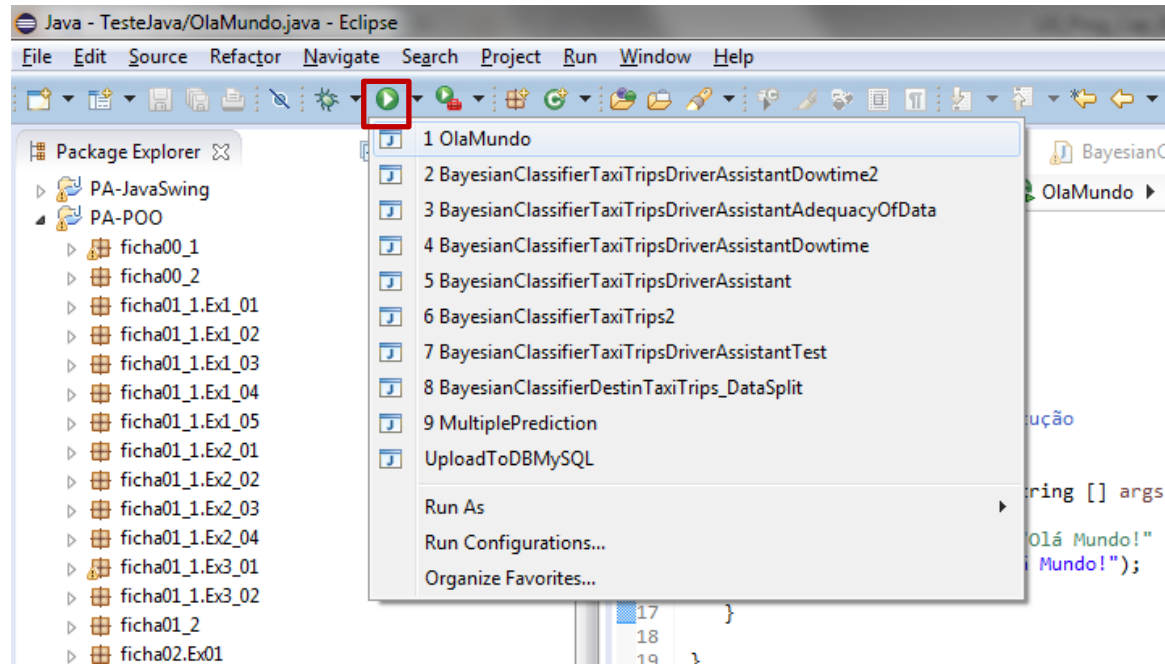
# Ambiente de Desenvolvimento (IDE)

## Introdução à Linguagem Java

### Compilação e execução de código

Menu: Run → Run (CTRL+F11)

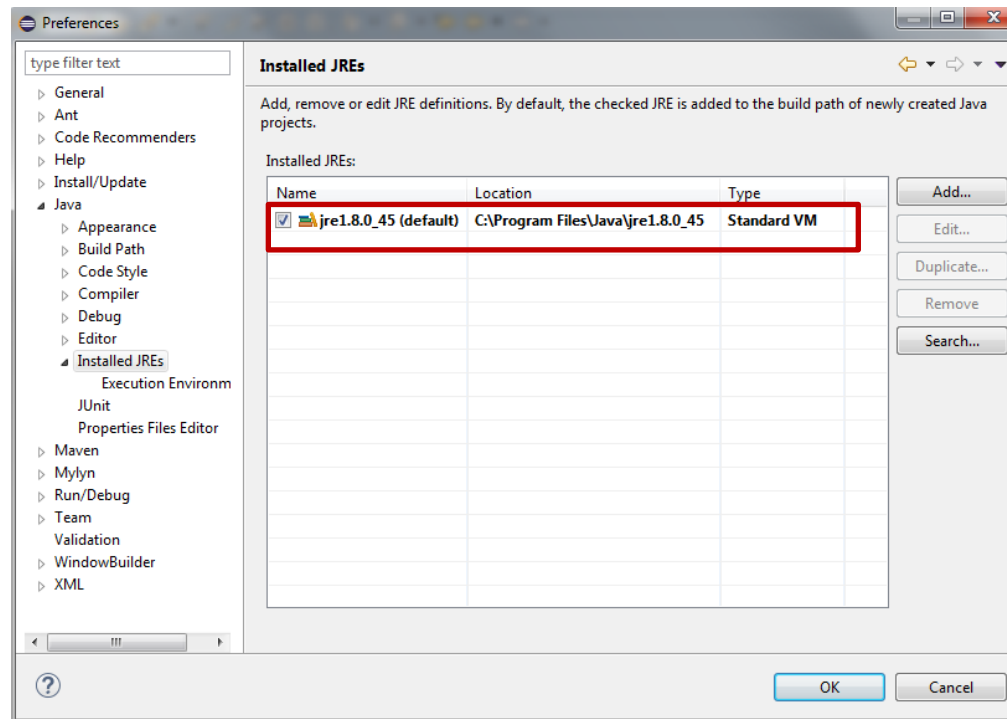
O IDE irá invocar os comandos **JAVAC** (compilação) e **JAVA** (execução)



# Introdução à Linguagem Java

## Verificar a existência de uma **Máquina Virtual** (VM, também designado **JRE: Java Runtime Environment**)

Menu: Window → Preferences



# Ambiente de Desenvolvimento (IDE) - Referências

## Introdução à Linguagem Java

Bibliotecas de programação (**JSE – Java Standard Environment**; também conhecido por **JDK – Java Development Kit**):

<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

**Máquina Virtual (JRE – Java Runtime Environment**, integrada no JSE)

<https://java.com/en/download/>

JSE **API** Documentation (**Application Program Interface**)

<http://docs.oracle.com/javase/8/docs/api/>

<https://docs.oracle.com/en/java/javase/15/docs/api/>

**Java tutoriais**

<http://docs.oracle.com/javase/tutorial/>

**IDE Eclipse**

<https://eclipse.org/downloads/>

**IDE NetBeans**

<https://netbeans.org/downloads/index.html>

# Referências

Introdução à Linguagem Java

## ***“Programação Orientada a Objectos”***

António José Mendes

Departamento de Engenharia Informática, Universidade de Coimbra

## ***“Java in a Nutshell”, 4ª Edição, Capítulo 3 “Object-Oriented Programming in Java”***

David Flanagan

O'Reilly, ISBN: 0596002831

## ***“Thinking in Java”, 4ª Edição,***

Capítulo 1 “Introduction to Objects”; Capítulo 2 “Everything is an Object”

Bruce Eckel

Prentice Hall, ISBN: 0131872486

## ***“The Java Tutorial - Learning the Java Language: Object-Oriented Programming Concepts”***

Java Sun Microsystems

<http://java.sun.com/docs/books/tutorial/java/concepts/index.html>

## ***“The Java Tutorial – Learning the Java Language: Classes and Objects ”***

<http://java.sun.com/docs/books/tutorial/java/javaOO/index.html>



**"Fundamentos de Programação em Java 2"**, Capítulo 8 "*Classes e Objectos*"

António José Mendes, Maria José Marcelino

FCA, ISBN: 9727224237

**"Java 5 e Programação por Objectos"**,

Capítulo 1 "*Paradigma da Programação por Objectos*", Capítulo 3 "*Classes e Instâncias*"

F. Mário Martins

FCA, ISBN: 9727225489