

Programação Orientada a Objetos – Java

(Material de apoio do Aluno)

Professor: Fábio Renato de Almeida

<https://github.com/fabiorenatodealmeida>

e-mail: fabiorenatodealmeida@hotmail.com

Bibliografia recomendada...

1. Java Como Programar

10ª edição, 2017, Pearson

Paul Deitel, Harvey Deitel

2. OCA – Oracle Certified Associate

Java SE 8 Programmer I

Study Guide, 2015

Jeanne Boyarsky, Scott Selikoff

3. OCA – Java SE 8 Programmer I

Certification Guide

Manning, 2017

Mala Gupta

4. Lógica de Programação e Estruturas de Dados

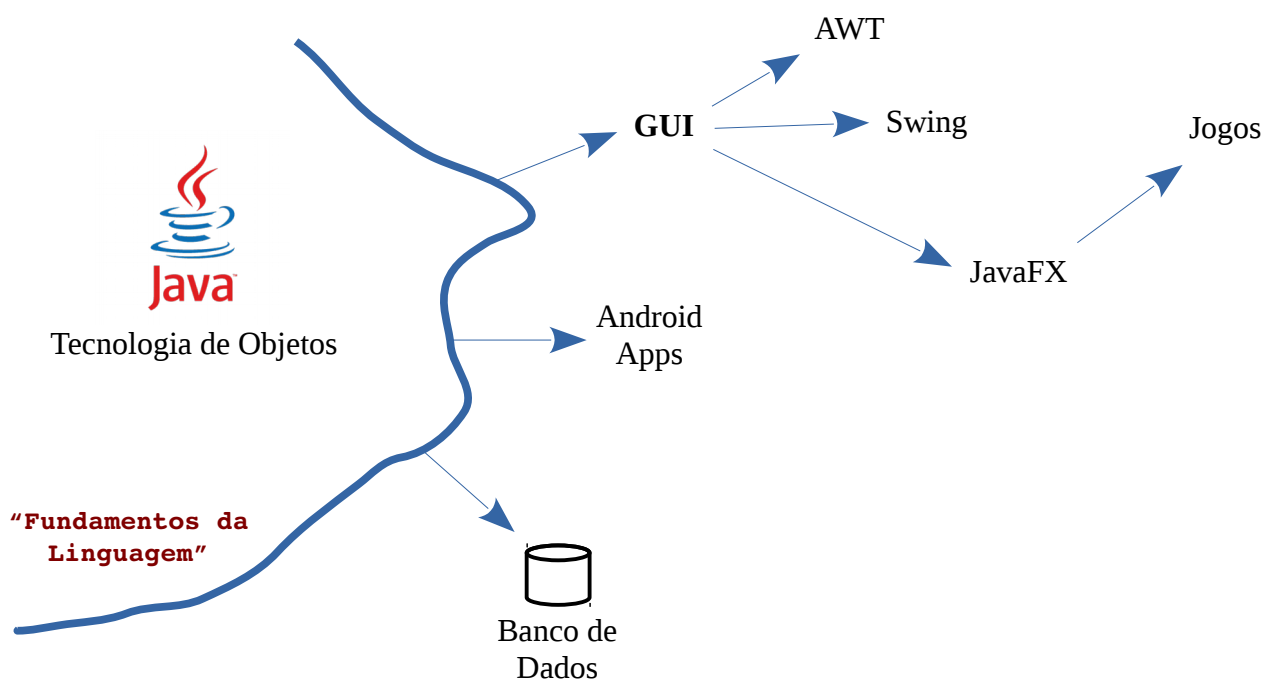
3ª edição, 2016, Pearson

Sandra Puga, Gerson Rissetti

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

6. JDK → src.zip (Java API)

Panorama Geral



1 Introdução ao Java

Java ME (Micro Edition): Direcionada a dispositivos com recursos limitados (memória, CPU, armazenamento). Sistemas Android adotam uma versão customizada de Java ME → Smart phones, tablets, smart TVs, smart watches, etc.

Google Android (+80% do mercado) → Ambiente de desenvolvimento...

Android Studio + Android SDK + JDK (SE)

Restante do mercado: Apple iOS, Windows Phone e outros

Java SE (Standard Edition): Direcionada a computadores de propósito geral → Notebooks, desktops, servidores.

Java EE (Enterprise Edition): Direcionada a ambientes que exigem processamento distribuído em larga escala e aplicações web → Servidores, cluster.

Windows, Linux, macOS

Ambiente de desenvolvimento: NetBeans + JDK (SE ou EE)

JDK → www.oracle.com/technetwork/java/javase/downloads/index.html

JAVA_HOME = <jdk-installation-directory>

PATH = <jdk-installation-directory>/bin

CLASSPATH = .

Habilidades...

Entender o propósito das edições Java. Preparar um ambiente de desenvolvimento Java: NetBeans + JDK SE.

keywords...

abstract	continue	for	new	switch
assert	default	if	package	synchronized
boolean	do	goto	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

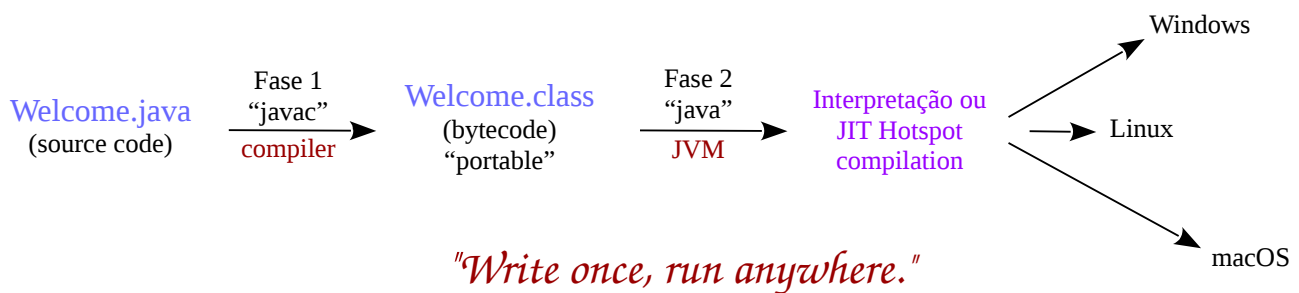
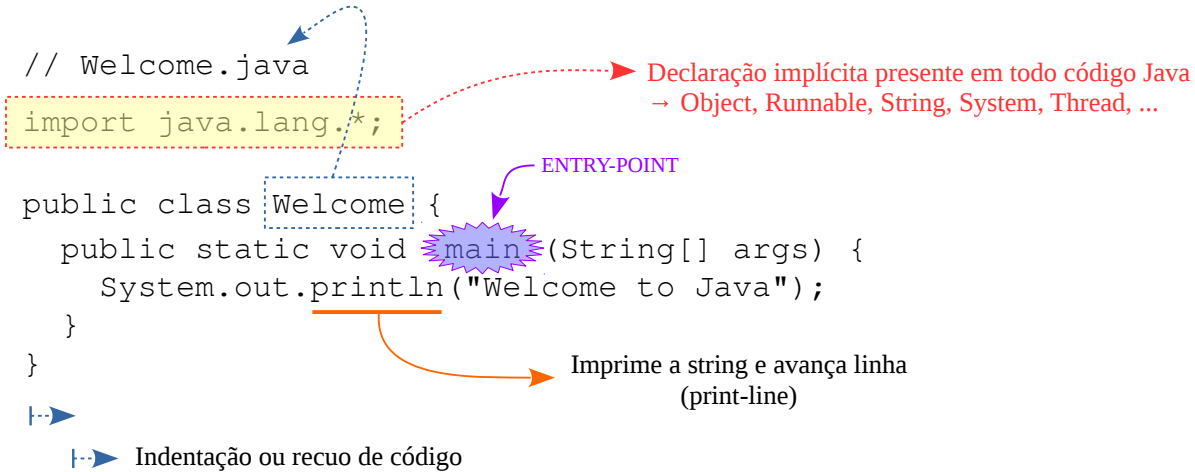
Literais: `true`, `false`, `null`

Java → Linguagem de programação orientada a objetos projetada para ser "simples".

Habilidades...

Entender o que são palavras-chave (keywords). Reconhecer que Java é mais fácil que C++.

App: Welcome



Compilação: javac Welcome.java

Execução: java Welcome

Habilidades...

Compreender a estrutura básica de um código Java. Entender a importância do recuo de código. Entender o propósito das diferentes fases de compilação. Reconhecer o ponto de partida em um programa Java. Construir e executar um código Java simples. Entender a portabilidade 32-64 bits.

App: FichaPessoal


Nome: Joaquim José da Silva Xavier

Endereço: Rua Tiradentes, 1000

Bairro: Centro

Cep: 15100-000 Cidade: São José do Rio Preto – SP

Telefone: 9.9090-1234

- 
1. `System.out.println`
 2. `System.out.print + System.out.println`

App: ArvoreNatal

```
  *  
 * * *  
* * * * *  
_ | _
```

Habilidades...

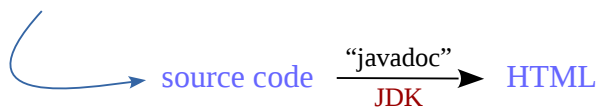
Imprimir saída de texto no terminal de comandos. Entender a diferença entre `System.out.print` e `System.out.println`.

Comentários em Java

```
// Comentário de linha
```

```
/*
  Comentário de bloco
  Pode conter diversas linhas
*/
```

```
/**
  Comentário javadoc
  Contém código para gerar documentação
*/
```



Exemplos...

```
salario = salario * 1.1; // 10% de aumento
```

```
/*
 *
 * Programa escrito por Ada Augusta
 * 01/01/1850
 *
 */
```

```
if (x == 2) // checa se x é igual a 2
```

```
/*
  int x = 1;
  int y = 2;
*/
```

Péssimo!!!

Habilidades...

Reconhecer a importância de comentários. Saber quando, onde e como documentar/comentar trechos de código Java.

Identificadores em Java

Identificadores são utilizados para dar nomes únicos a classes, interfaces, enumerações, arrays, variáveis, métodos, etc.

Regras de nomeação...

1. Podem conter letras (A..Z, a..z), dígitos (0..9), underscore (_) e cifrão (\$)
2. Não devem começar com um dígito
3. Não podem conter espaços
4. Letras maiúsculas e minúsculas são consideradas diferentes (case sensitive)
5. Caracteres acentuados, chineses, asiáticos, etc. são permitidos mas não recomendados
6. Não há restrições de tamanho

Identificadores válidos...

```
Pessoa  
pessoa  
x1  
MAX_SIZE  
salario$ → não recomendável
```

Identificadores inválidos...


```
1x  
nome da pessoa  
int
```

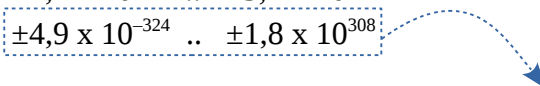
Habilidades...

Entender as regras de construção de identificadores. Reconhecer a importância da escolha de um nome simples e adequado.

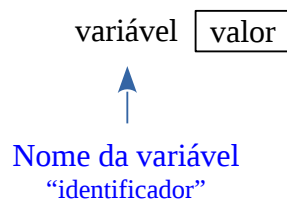
2 Tipos primitivos

<u>Tipo</u>	<u>Bits</u>	<u>Valores</u>
boolean	JVM	true, false
char	16	'\u0000' .. '\uFFFF' (0 .. 65535) Unicode UTF-16
byte	8	$-2^7 \dots 2^7 - 1 = -128 \dots 127$
short	16	$-2^{15} \dots 2^{15} - 1 = -32768 \dots 32767$
int	32	$-2^{31} \dots 2^{31} - 1 = -2.147.483.648 \dots 2.147.483.647$
long	64	$-2^{63} \dots 2^{63} - 1 = -9.223.372.036.854.775.808 \dots 9.223.372.036.854.775.807$
float	32	$\pm 1,4 \times 10^{-45} \dots \pm 3,4 \times 10^{38}$
double	64	$\pm 4,9 \times 10^{-324} \dots \pm 1,8 \times 10^{308}$


 Números de ponto flutuante
(NÃO EXATOS)


 Maior magnitude e precisão numérica
(dígitos depois da vírgula)

Uma variável de um tipo primitivo contém um valor dentro do intervalo permitido para o tipo em questão.



Exemplos...


estudante (boolean) true

idade (short) 28

sexo (char) ‘M’

salario (double) 9000.0

quantidade (byte) 200

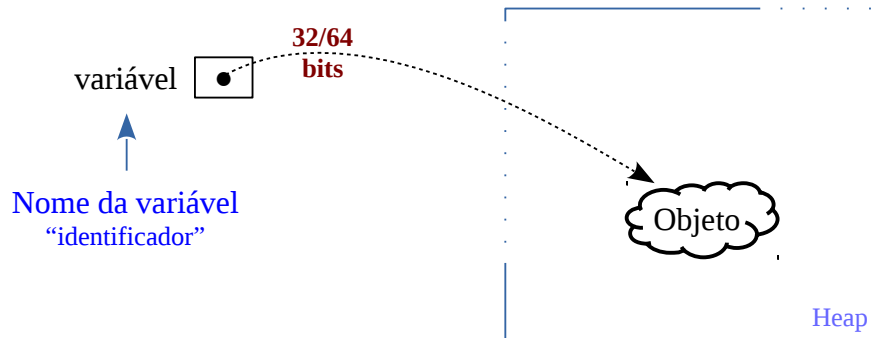

 Impossível

Habilidades...

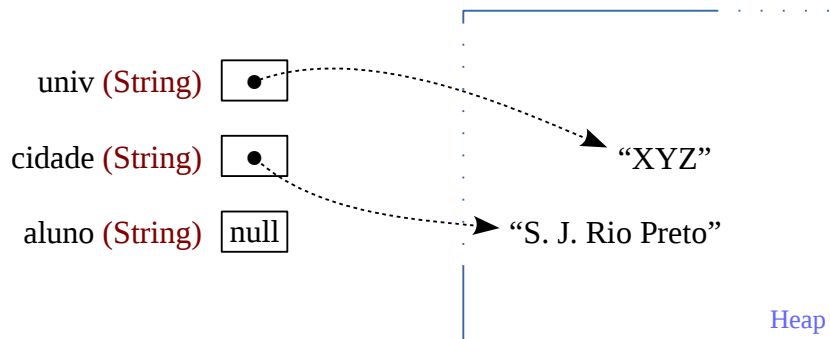
Identificar os tipos de dados primitivos da linguagem Java e compreender o propósito e a capacidade de armazenamento de cada um. Entender o porquê do padrão Unicode. Reconhecer a imprecisão numérica envolvendo cálculos com números de ponto flutuante.

3 Tipos referência

Uma variável de um tipo referência (classe, interface, enumeração, array) contém um ponteiro (endereço ou referência) para um objeto do tipo em questão.



Exemplos...



Em Java, strings ("abc") são gerenciadas por objetos da classe String.

null: Ponteiro nulo → A variável não referencia nenhum objeto

Habilidades...

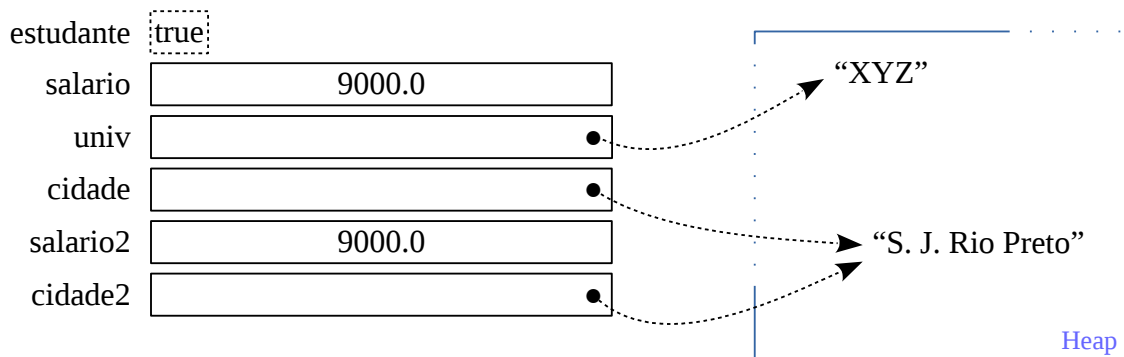
Entender a diferença entre alocação de dados do tipo primitivo e alocação de dados do tipo referência.

4 Declaração e inicialização de variáveis

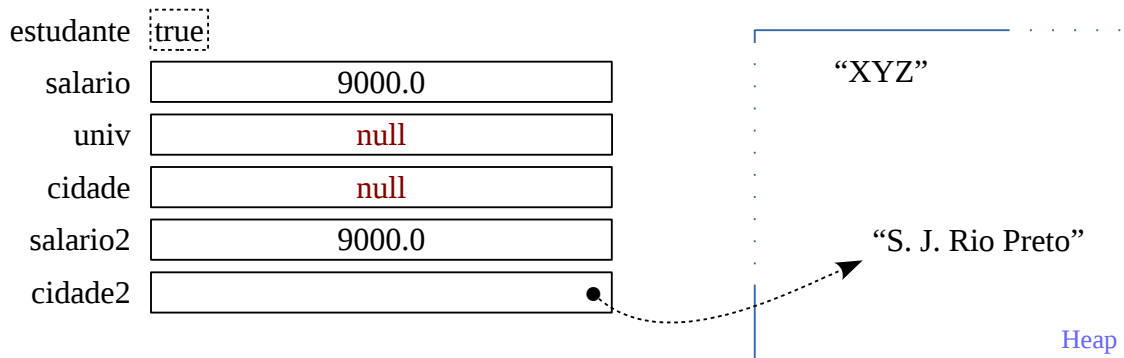
tipo variável [= <inicialização>] ;

Exemplos...

```
boolean estudante = true;
double salario = 9000.0;
String univ = "XYZ";
String cidade = "S. J. Rio Preto";
double salario2 = salario;
String cidade2 = cidade;
```



```
univ = null;
cidade = null;
```



O objeto String "XYZ", agora sem qualquer referência, ainda permanece no Heap. Contudo ele se torna "elegível" para a coleta de lixo.

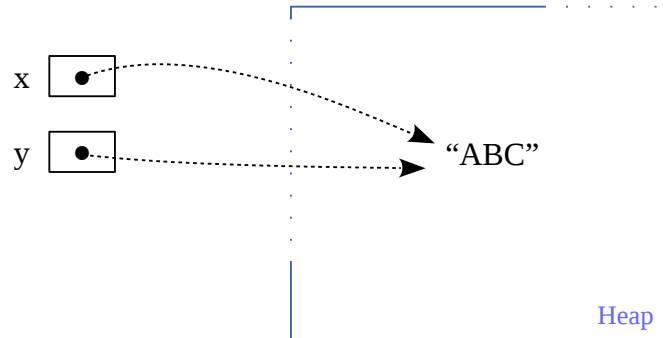
Habilidades...

Entender como declarar e inicializar variáveis. Compreender o mecanismo básico de coleta de lixo.

5 Imutabilidade de strings

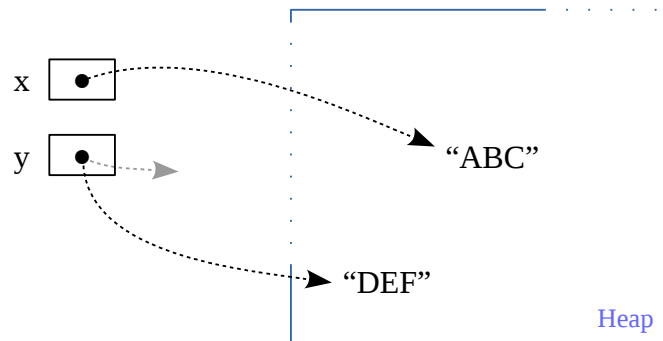
Considere o trecho de código a seguir:

```
String x = "ABC";  
String y = x;
```



O que irá acontecer se fizermos:

```
y = "DEF";
```



Strings em Java são imutáveis. Toda vez que uma string é criada ou modificada, um novo objeto String é alocado no Heap.

Habilidades...

Entender o conceito de imutabilidade de strings e o impacto no gerenciamento de memória.

App: Dados

```
public class Dados {
    public static void main(String[] args) {
        String nome = "João";
        char sexo = 'M';
        short idade = 28;
        double salario = 9000.0;
    }
}
```

Imprimir...

```
Nome: João
Sexo: M
Idade: 28
Salário: 9000.0
```

Solução 1:

```
System.out.print("Nome: ");
System.out.println(nome);
...
```

Solução 2:

```
System.out.println("Nome: " + nome);
...
```

Operador de adição (+) utilizado para concatenação (junção/união) de strings. Java converte argumentos não-string em strings durante o processo de concatenação.

Cuidado!

```
System.out.println("A soma de 1 e 2 resulta em " + 1 + 2);
```

Imprime: A soma de 1 e 2 resulta em 12

```
System.out.println("1 + 2 = " + (1 + 2));
```

Imprime: 1 + 2 = 3

Habilidades...

Compreender o conceito de concatenação de strings. Entender a diferença entre imprimir uma string ou o conteúdo de uma variável.

App: Parabens

```
import java.util.Scanner;
```

Pacote onde se encontra a classe **Scanner**

```
public class Parabens {  
    public static void main(String[] args) {  
        Scanner scanner = new Scanner(System.in);
```

O objeto **scanner** irá gerenciar a entrada de dados pelo terminal de comandos...

```
        System.out.print("Qual seu nome? ");  
        String nome = scanner.nextLine();  
        System.out.print("Quantos anos você faz hoje? ");  
        short idade = scanner.nextShort();  
        System.out.println(  
            "Olá " + nome + ", parabéns pelos seus " + idade + " anos."  
        );  
    }  
}
```

Gera um objeto da classe **Scanner**

Exibindo mensagem por meio de concatenação de strings...

`scanner.nextLine()` → recebe uma linha de texto

`scanner.nextShort()` → recebe um número do tipo short

Habilidades...

Compreender o mecanismo básico de geração de objetos. Possibilitar a entrada de dados do tipo texto e numérico pelo terminal de comandos.

App: CalculadoraInteiros

```
import java.util.Scanner;

public class CalculadoraInteiros {
    public static void main(String[] args) {
        int x, y;
        int adicao, subtracao, multiplicacao, divisao, resto;
        Scanner scanner = new Scanner(System.in);
        System.out.println("Calculadora de números inteiros");
        System.out.print("Primeiro valor: ");
        x = scanner.nextInt();
        System.out.print("Segundo valor: ");
        y = scanner.nextInt();
        adicao = x + y;
        subtracao = x - y;
        multiplicacao = x * y;
        divisao = x / y;
        resto = x % y;
        System.out.printf("%d + %d = %d\n", x, y, adicao);
        System.out.printf("%d - %d = %d\n", x, y, subtracao);
        System.out.printf("%d * %d = %d\n", x, y, multiplicacao);
        System.out.printf("%d / %d = %d\n", x, y, divisao);
        System.out.printf("%d %% %d = %d\n", x, y, resto);
    }
}
```

Imprime com base em regras de formatação...

Regras de formatação com printf...

%b	placeholder para um valor lógico – boolean
%c, %C	placeholder para um caractere – char
%d	placeholder para um número inteiro – byte, short, int, long
%f	placeholder para um número de ponto flutuante – float, double
%s, %S	placeholder para um objeto string – String
%n	avança uma linha – não requer um argumento na lista de argumentos
%%	insere o caractere %

Desafio: Criar calculadora de números reais...

Desafio: Re-escrever printf usando println e concatenação de strings.

Habilidades...

Saber quando utilizar os métodos print, println e printf. Reconhecer a simplicidade do uso de printf em situações onde a concatenação de strings levaria a um código mais complexo.

6 Operadores aritméticos, relacionais, lógicos, etc.

Operadores aritméticos

+	Adição
-	Subtração
*	Multiplicação
/	Divisão
%	Resto

Operadores relacionais

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

Operadores lógicos

&	AND (avaliação de curto-circuito: &&)
	OR (avaliação de curto-circuito:)
^	XOR
!	NOT

Operadores de atribuição

=	Recebe
+=	Adiciona
-=	Subtrai
*=	Multiplica
/=	Divide
%=	Resto
&=	AND
=	OR
^=	XOR

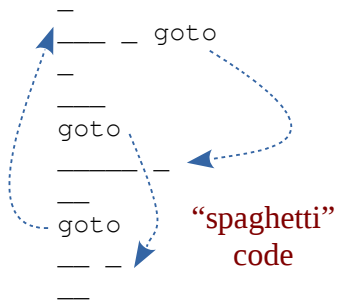
Outros

++	Incremento
--	Decremento
?:	Condicional

Habilidades...

Conhecer os operadores disponibilizados pela linhagem Java.

7 Programação estruturada



Na programação estruturada os algoritmos são implementados sem o uso de **goto** e com base em três estruturas fundamentais:

- Sequência
- Decisão (**if**, **if-else**, **switch**)
- Repetição (**for**, **while**, **do-while**)

As estruturas de sequência, decisão e repetição podem ser encadeadas (uma após a outra) e também aninhadas (nested – uma dentro da outra). A programação estruturada está incorporada na programação orientada a objetos.

Estrutura de sequência

```
instrução-1;
instrução-2;
...
instrução-n;
```

Exemplo...




```
System.out.print("1 + 2 = ");
int x = 1;
int y = 2;
System.out.println(x + y);
```

Habilidades...

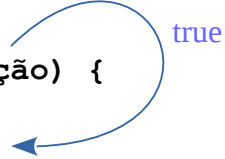
Reconhecer o problema com códigos do tipo “spaghetti”. Conhecer as três estruturas fundamentais da programação estruturada. Entender a diferença entre código indentado (recuado), encadeado e aninhado.

Estruturas de decisão: if, if-else

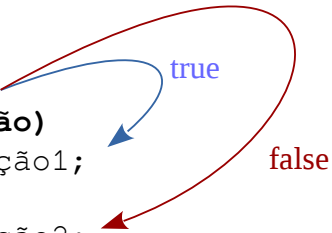
```
if (condição) instrução;
```



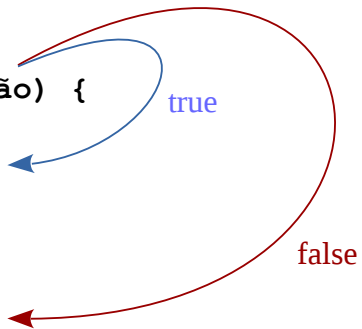
```
if (condição) {  
    bloco  
}
```



```
if (condição)  
    instrução1;  
else  
    instrução2;
```

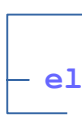


```
if (condição) {  
    bloco1  
} else {  
    bloco2  
}
```



Cuidado!

```
if (condição1)  
    if (condição2)  
        if (condição3)  
            instrução1;  
    else  
        instrução2;
```



Indentação incorreta!

Como escrever um **else** para cada uma das condições?

condição: expressão que resulta sempre em um valor lógico (**true** ou **false**)

Habilidades...

Compreender as construções básicas das estruturas de decisão if e if-else.

Qual o valor final da variável x?

1.

```
int x = 1;
if (x > 0) x = x + 1;
```
2.

```
int x = 10;
int y = 9;
if (x + y > 20)
    x = 1;
else
    x++;
```
3.

```
short idade = 17;
char sexo = 'M';
int x = 0;
if (sexo == 'M' && idade >= 18)
    System.out.println("Homem adulto");
    x = 1;
```
4.

```
int x = 9;
if (x > 10) ;
    x = 0;
```
5.

```
int a = 1;
int b = 2;
int c = 3;
int x = 0;
if (a > 10 && b > 0)
    if (c > 0)
        if (x > 0)
            x = 1;
else
    x = 2;
```
6.

```
int x = 0;
if (true) {
    x = 1;
}
```

Habilidades...

Ficar atento a pequenos detalhes que podem mudar radicalmente o comportamento de um programa.

Qual o conceito final do aluno?

A → 9.0 – 10.0

B → 8.0 – abaixo de 9.0

C → 7.0 – abaixo de 8.0

F → Abaixo de 7.0

```
double nota = 8.2;
char conceito;
if (nota >= 9.0)
    conceito = 'A';
else if (nota >= 8.0 && nota < 9.0)
    conceito = 'B';
else if (nota >= 7.0 && nota < 8.0)
    conceito = 'C';
else
    conceito = 'F';
```

Habilidades...

Recuar apropriadamente instruções if aninhadas. Identificar condições redundantes.

Estrutura de decisão: switch

```
switch (expressão) {  
    case constante1:  
        ...  
        break;  
    case constante2:  
        ...  
        break;  
    default:  
        ...  
}
```

char, byte, short, int, valor de uma enumeração, String

Abandona estrutura switch

Exemplo...

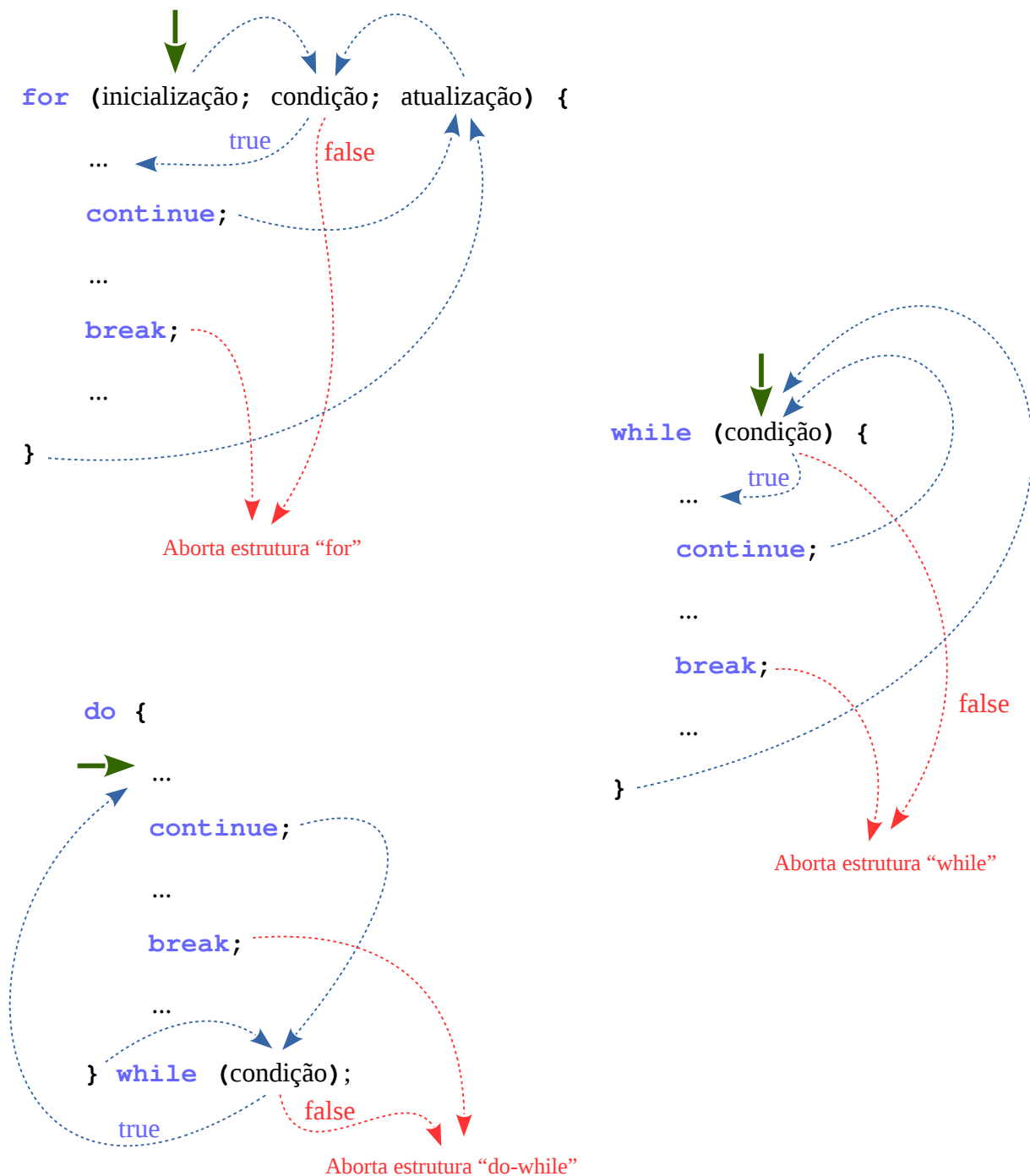
```
int mes = 1;  
switch (mes) {  
    case 1:  
    case 7:  
    case 12:  
        System.out.println("Férias");  
        break;  
    default:  
        System.out.println("Aulas");  
}
```

Desafio: Re-escrever usando if.

Habilidades...

Utilizar a estrutura de decisão switch. Saber quando utilizar switch e quando utilizar if.

Estruturas de repetição: for, while, do-while



Habilidades...

Compreender as estruturas de repetição e a diferença entre elas.

Loops com for, while e do-while...

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

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

```
int i = 1;  
do {  
    System.out.println(i);  
    i++;  
} while (i <= 10);
```

```
int i = 1;  
while (i <= 10) System.out.println(i++);
```

```
int i = 0;  
while (++i <= 10) System.out.println(i);
```

```
int i = 0;  
while (i++ < 10) System.out.println(i);
```

Contagem regressiva...

```
for (int i = 10; i >= 1; i--) {  
    System.out.println(i);  
}
```

Habilidades...

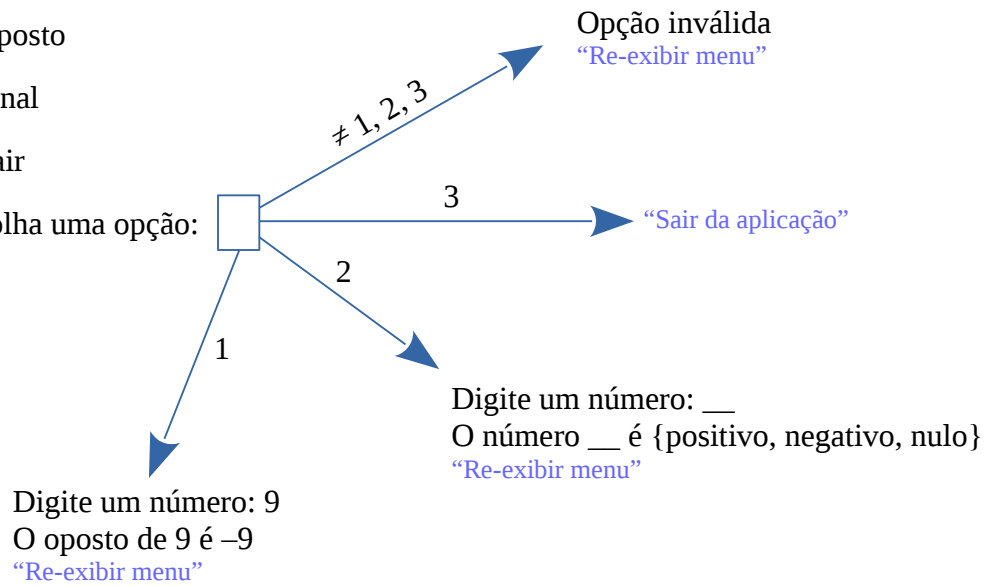
Utilizar a estrutura de repetição mais simples para cada tipo de situação.

Desafio → App: Menu

Menu

1. Oposto
2. Sinal
3. Sair

Escolha uma opção:

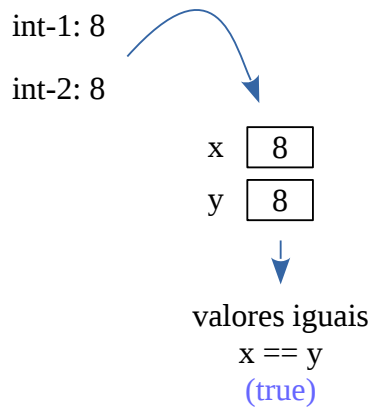


App: ComparaTiposPrimitivos

```
import java.util.Scanner;

public class ComparaTiposPrimitivos {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("int-1: ");
        int x = scanner.nextInt();
        System.out.print("int-2: ");
        int y = scanner.nextInt();
        if (x == y) System.out.println("Iguais");
    }
}
```

Execução...

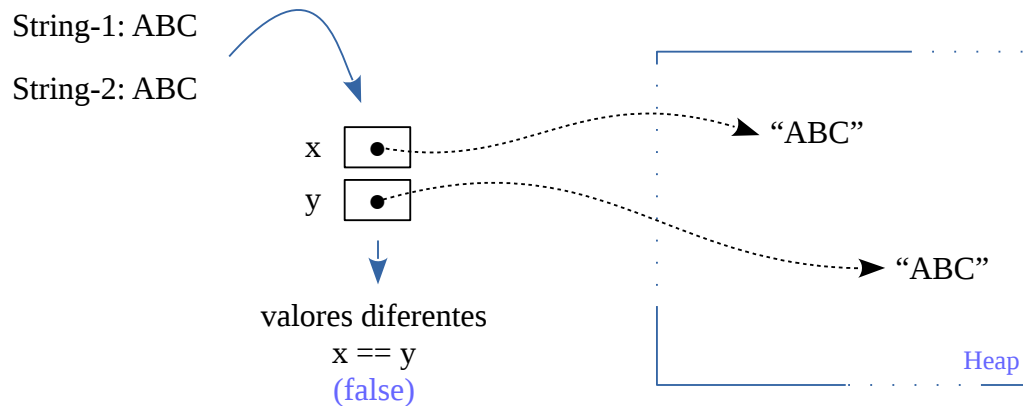


App: ComparaTiposReferencia

```
import java.util.Scanner;

public class ComparaTiposReferencia {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("String-1: ");
        String x = scanner.nextLine();
        System.out.print("String-2: ");
        String y = scanner.nextLine();
        if (x == y) System.out.println("Iguais");
    }
}
```

Execução...



Cuidado!

Nunca compare tipos referência com o operador ==

Use: `if (x.equals(y)) ...`

Habilidades...

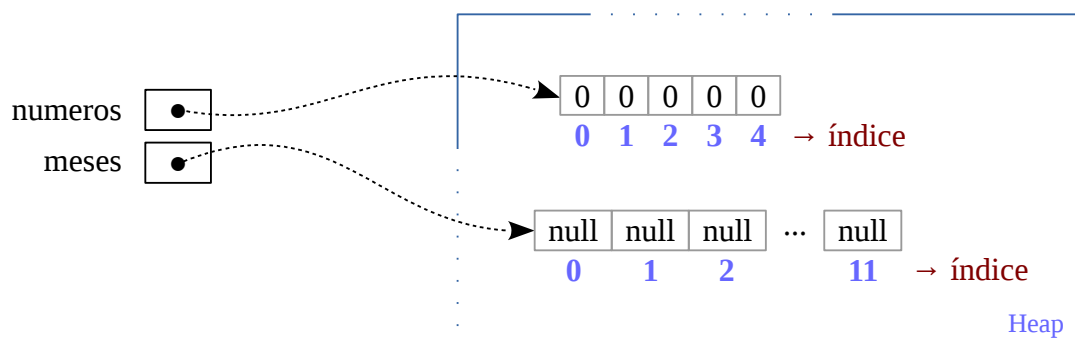
Entender que o operador == compara valores de tipos primitivos e valores de tipos referência (os ponteiros).

8 Arrays

Estrutura de dados → Conjunto de dados que se inter-relacionam de modo a abstrair as complexidades de um conceito ou funcionalidade: pilha, lista, fila, árvore, etc.

Array → Estrutura de dados do tipo coleção onde os elementos são todos de um mesmo tipo.

```
int[] numeros = new int[5]; // inicializa com zeros
String[] meses = new String[12]; // inicializa com "null"
```



Uma vez instanciado, o tamanho de um array não pode ser alterado, ou seja, não é possível inserir novos elementos ou excluir elementos existentes, contudo é possível alterá-los.

Arrays possuem alocação estática → Tamanho fixo pré-definido

`java.util.ArrayList` → Tratamento dinâmico de arrays

Exemplo...

```
String[] meses = new String[] {
    "Janeiro", "Fevereiro", "Março", "Abril", "Maio", "Junho",
    "Julho", "Agosto", "Setembro", "Outubro", "Novembro", "Dezembro"
};
```

Qual o 5º mês do ano?

```
System.out.println(meses[4]);
```

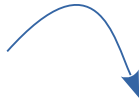
Forma alternativa de declaração + instânciação + inicialização

```
String[] meses = {  
    "Janeiro", "Fevereiro", "Março", "Abril", "Maio", "Junho",  
    "Julho", "Agosto", "Setembro", "Outubro", "Novembro", "Dezembro"  
};
```

Percorrendo os elementos de um array...

```
for (int i = 0; i < meses.length; i++)  
    System.out.println(meses[i]);
```

```
for (String mes : meses)  
    System.out.println(mes);
```



for direcionado a coleções.


Atenção! O parâmetro “mes” recebe uma cópia dos elementos do array.

Habilidades...

Entender que arrays são do tipo referência e possuem alocação estática. Compreender a indexação dos elementos em um array (zero-based). Saber quando se pode usar a instrução *for* aprimorada.

Arrays multi-dimensionais

```
int[][] x = { {1, 2}, {3, 4}, {5, 6} };
              x[0]   x[1]   x[2]
```



```

x[0][0] = 1
x[0][1] = 2

```

```
int[][] y = { {1, 2}, {3, 4, 5, 6}, {7}, {8, 9, 10} };
              y[0]   y[1]   y[2]   y[3]
```

Desafio: Exibir os valores do array y.

```
1 2
3 4 5 6
7
8 9 10
```

Estratégia 1: Utilizar duas instruções for aninhadas (for tradicional)

Estratégia 2: Utilizar duas instruções for aninhadas (for aprimorado para coleções)

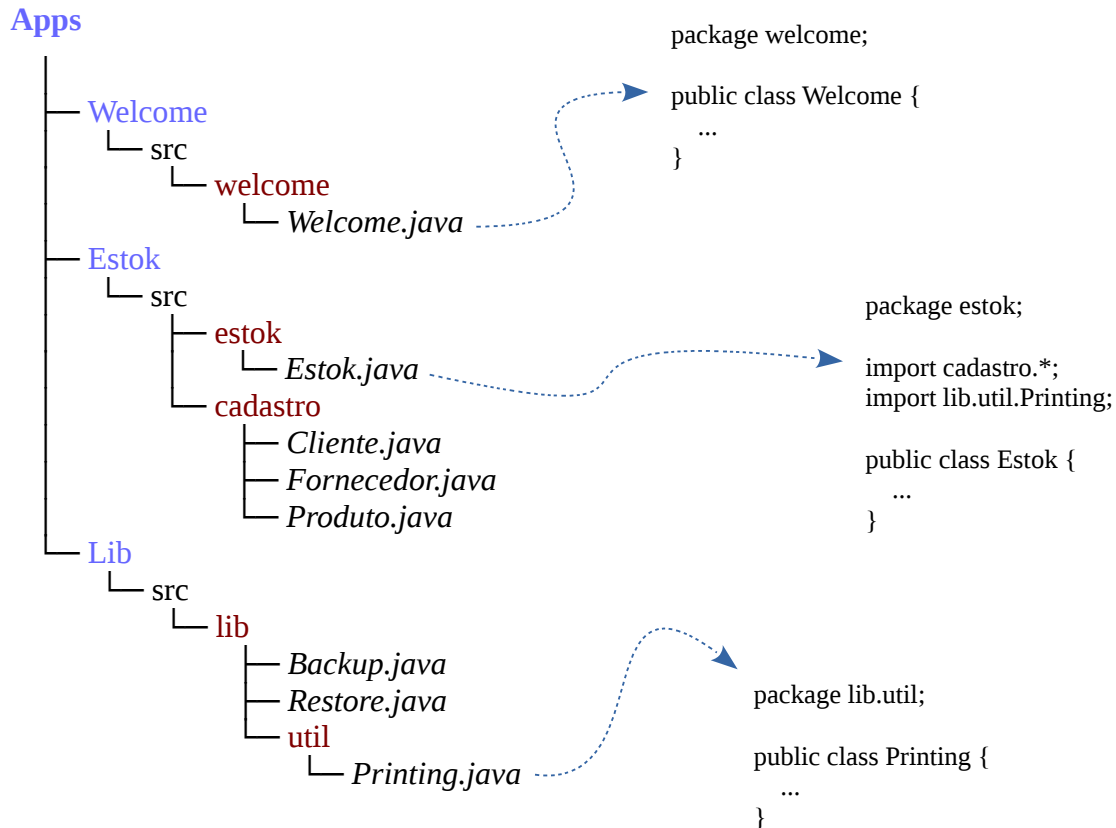
Bônus...

```
import java.util.Arrays;
int[] numeros = { 30, 20, 10, 60, 40, 50 };
Arrays.sort(numeros);
int pos = Arrays.binarySearch(numeros, 60);
```

Habilidades...

Compreender que arrays multi-dimensionais em Java são, na verdade, arrays de arrays. Manipular arrays multi-dimensionais com loops aninhados.

9 Pacotes em Java



Pacotes permitem o agrupamento de tipos relacionados, gerenciamento de espaços de nomes e oferecem um mecanismo de proteção de acesso. Declarações `import` entre arquivos contidos em um mesmo pacote não são necessárias.

Compilação do projeto Estok:

Linux, macOS → `CLASSPATH = ./~/Apps/Lib/src`

Windows → `CLASSPATH = .;C:\Apps\Lib\src`

Alternativa...

```
javac -classpath ./~/Apps/Lib/src
```

```
java -classpath .:<path-to-lib-version>
```

NetBeans: Project – Libraries (sub-item de Project) – Add Folder

Sugestão: `package com.github.usuario.app;`

Habilidades...

Entender a importância do uso de pacotes para a organização do projeto e distribuição de código.

10 Programação Orientada a Objetos

Motivação...

Equação do 2º grau: $ax^2 + bx + c = 0 \Rightarrow \Delta = b^2 - 4ac$

$\Delta < 0 \Rightarrow \nexists$ raiz real

$\Delta = 0 \Rightarrow x = \frac{-b}{2a}$

$\Delta > 0 \Rightarrow x_1 = \frac{-b + \sqrt{\Delta}}{2a} ; x_2 = \frac{-b - \sqrt{\Delta}}{2a}$

1. Como resolver a equação $x^2 + 2x = 0$?

```
double a = 1.0, b = 2.0, c = 0.0;
double delta = b * b - 4 * a * c;
if (delta < 0.0) {
    System.out.println("Não existe raiz real");
} else if (delta == 0.0) {
    double x = -b / (2 * a);
    System.out.printf("x = %f\n", x);
} else {
    double x1 = (-b + Math.sqrt(delta)) / (2 * a);
    double x2 = (-b - Math.sqrt(delta)) / (2 * a);
    System.out.printf("x1 = %f; x2 = %f\n", x1, x2);
}
```

2. E quanto a equação $x^2 + 2x + 1 = 0$?

3. Qual o valor de Δ na equação $3x^2 + 2x - 2 = 0$?

Tecnologia de Objetos

```
1. Eq2G eq = new Eq2G(1.0, 2.0, 0.0);  
   System.out.println(eq.raizes());  
  
2. eq.setC(1.0);  
   System.out.println(eq.raizes());  
  
3. eq.setABC(3.0, 2.0, -2.0);  
   System.out.println(eq.getDelta());
```

Mas a classe `Eq2G` não existe → Basta criar e implementar as funcionalidades desejadas.

Na tecnologia de objetos, as classes são componentes de software reutilizáveis que dão origem aos objetos e ajudam a modularizar a aplicação. A reutilização se dá por herança, composição e instanciação. Qualquer conceito, seja este concreto ou abstrato, pode ser representado por meio de um objeto contendo atributos e comportamentos.

Pilares da Tecnologia de Objetos: Classes, Objetos, Interfaces, Encapsulamento, Herança e Polimorfismo.

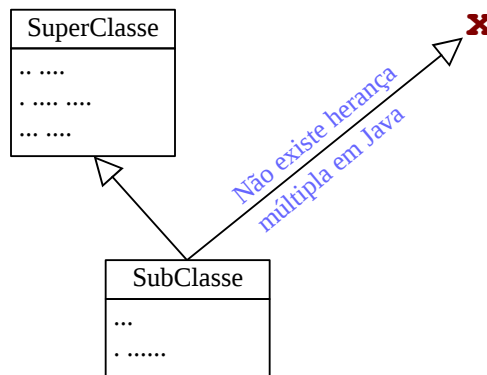
Habilidades...

Capacidade de decidir quando realizar uma tarefa da forma tradicional (métodos) e quando trabalhar com objetos. Reaproveitamento de código e ocultação de funcionalidade. Distribuição de biblioteca (API).

11 Classes

```
public class SubClasse extends SuperClasse {
    :
    Campos, atributos, variáveis de instância
    Definem o conjunto de estados possíveis
    :
    Construtores
    Auxiliam na inicialização dos objetos
    :
    Métodos
    Executam as funcionalidades (comportamentos)
    :
}
```

Opcional (default = java.lang.Object)
 Componentes da classe (Membros)



SuperClasse: Versão mais genérica, normalmente englobando um conjunto maior de objetos.

SubClasse: Versão mais específica, onde novos membros podem ser adicionados e comportamentos herdados podem ser sobrescritos (especialização).

Herança: Subclasses herdam, direta ou indiretamente, o estado e comportamento de classes superiores. O mecanismo de herança proporciona reutilização (ou reaproveitamento) de código.

Habilidades...

Compreender a estrutura geral de uma classe (membros). Entender que não existem destrutores em Java. Entender a hierarquia Object – Classe – SubClasse – SubClasse... Identificar o relacionamento É-UM (IS-A).

12 Modificadores de acesso

private: O componente da classe só pode ser acessado de dentro da própria classe.

public: O componente é acessível dentro da classe e também fora dela (visibilidade externa).

protected: Subclasses e classes no mesmo pacote podem acessar o componente.

Ausência de um modificador de acesso: O componente só pode ser acessado por classes no mesmo pacote (package-access).

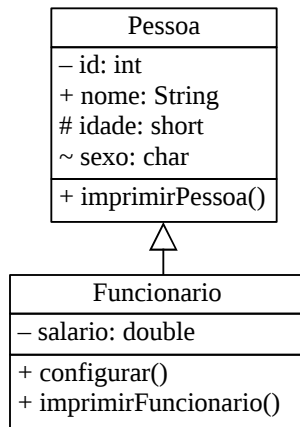
Encapsulamento

As classes definem (agrupam) os atributos e comportamentos dos objetos. Os modificadores de acesso controlam a visibilidade dos componentes da classe, ocultando recursos que não devem ser expostos ou restringindo o acesso a um subconjunto de classes.

Habilidades...

Reconhecer a importância dos modificadores de acesso para ocultação (encapsulamento) e/ou exposição de funcionalidades.

13 Campos



public class Pessoa {
 private int id;
 public String nome;
 protected short idade;
 char sexo = 'M'; // package-access
 public void imprimirPessoa() {
 System.out.printf(
 "id = %d\nnome = %s\nidade = %d\nsexo = %c\n",
 id, nome, idade, sexo);
 }
 }

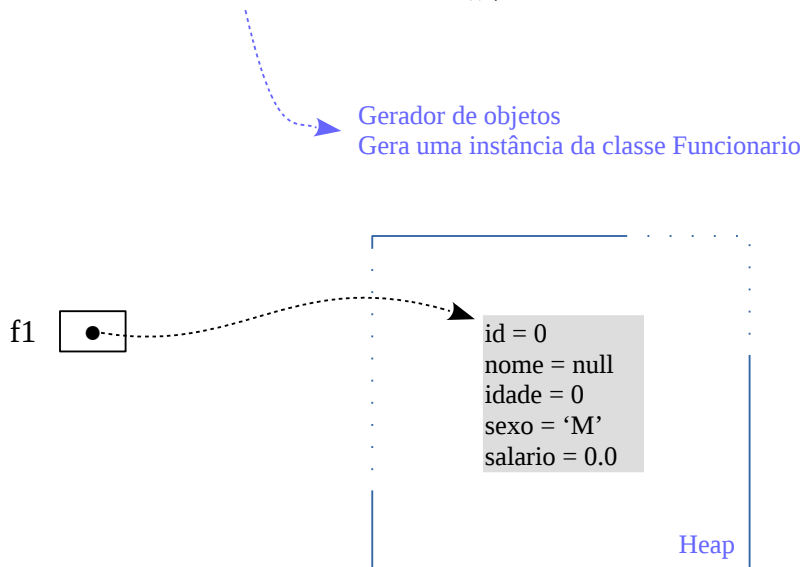
public class Funcionario extends Pessoa {
 private double salario;
 public void configurar() {
 id = 1; // **NÃO COMPILA**
 nome = "ABC";
 idade = 22;
 sexo = 'M'; // **OK se no mesmo pacote**
 salario = 9000.0;
 }
 public void imprimirFuncionario() {
 System.out.printf(
 "id = %d\nnome = %s\nidade = %d\nsexo = %c\nsalário = %f\n",
 id, nome, idade, sexo, salario);
 }
 }

Implicito: extends Object

Quando não especificado, variáveis de instância são inicializadas com o valor default:
 boolean = false
 char, byte, short, int, long, float, double = 0
 tipo referência = null

Gerando e trabalhando com objetos

```
Funcionario f1 = new Funcionario();
```



O que ocorre?

```
f1.imprimirPessoa();  
f1.configurar();  
f1.imprimirFuncionario();  
f1.imprimirPessoa();  
f1 = null;
```

O que ocorre?

```
Pessoa f2 = new Funcionario();  
f2.nome = "xyz";  
f2.idade = 18;  
f2.sexo = 'F';  
f2.imprimirFuncionario();  
f2.imprimirPessoa();
```

Habilidades...

Entender o mecanismo de downcasting (coerção). Reconhecer que as classes sempre herdam, implícita ou explicitamente, de Object. Entender conceitos de herança direta/indireta, restrições e/ou controle de acesso a membros da classe e referência mais genérica (objeto referenciado por um tipo ancestral).

14 Construtores

```
public class Horario {  
  
}
```

Classe sem um construtor explicitamente definido.
Neste caso o compilador fornece um construtor padrão.

```
public class Horario extends Object {  
    public Horario() {  
        super();  
    }  
}
```

Invoca construtor sem argumentos da
super classe (Object)

Construtores são responsáveis por inicializar os objetos em um estado consistente.

Construtores possuem o mesmo nome da classe e tem comportamento análogo ao de métodos, porém não possuem um tipo de retorno (nem mesmo void) e são invocados apenas uma vez: durante a construção do objeto (operador **new**).

```
Horario horario = new Horario();
```

Atenção! Um construtor padrão sem argumentos é fornecido pelo compilador somente quando não escrevemos nenhum construtor para a classe.

Sobrecarga de construtores...

Uma classe pode ter mais de um construtor, permitindo que um objeto seja inicializado de diversas formas, mas sempre de modo consistente.

```

public class Horario {
    private int hora;
    private int minuto;
    private int segundo;
    public Horario(int hora, int minuto, int segundo) {
        super();
        this.hora = hora;
        this.minuto = minuto;
        this.segundo = segundo;
    }
    public Horario(int hora, int minuto) {
        this(hora, minuto, 0);
    }
    public Horario(int hora) {
        this(hora, 0, 0);
    }
    public Horario() {
        this(0, 0, 0);
    }
    public void print() {
        System.out.printf("%02d:%02d:%02d", hora, minuto, segundo);
    }
}

```

Diagram annotations:

- shadowing...** (blue dashed arrow) points from the `segundo` parameter in the constructor to the `segundo` field declaration.
- Declaração local oculta declaração da classe** (blue dashed arrow) points from the `segundo` parameter to the `segundo` field.
- Instrução implícita** (purple arrow) points to the `super();` line.
- Concede acesso a variável de instância** (green dashed arrow) points to the `this.segundo` assignment.
- Invoca construtor principal...** (red dashed arrows) points to the `this(hora, minuto, 0);` and `this(hora, 0, 0);` lines.
- De fato, não é necessário fazer nada.** (purple arrow) points to the `this(0, 0, 0);` line.

Exemplos...

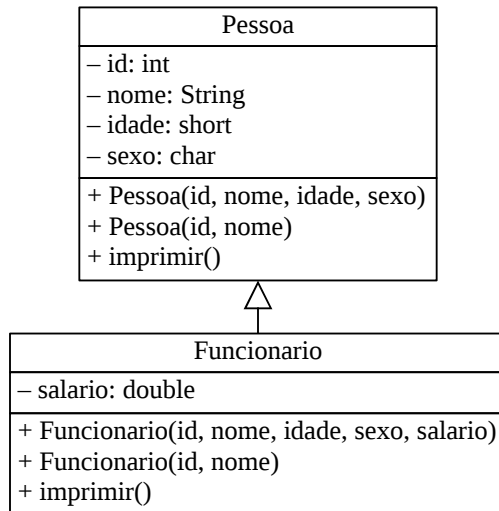
```

Horario h1 = new Horario(19, 20, 30);
Horario h2 = new Horario(19, 20);
Horario h3 = new Horario(19);
Horario h4 = new Horario();

h1.print();
h2.print();
h3.print();
h4.print();

```

Versão 2 da hierarquia Object – Pessoa – Funcionario



```

public class Pessoa {
    private int id;
    private String nome;
    private short idade;
    private char sexo;
    public Pessoa(int id, String nome, short idade, char sexo) {
        super();
        this.id = id;
        this.nome = nome;
        this.idade = idade;
        this.sexo = sexo;
    }
    public Pessoa(int id, String nome) {
        this(id, nome, (short)0, '?');
    }
    public void imprimir() {
        System.out.printf("id = %d\nnome = %s\nidade = %d\nsexo = %c\n",
            id, nome, idade, sexo);
    }
}

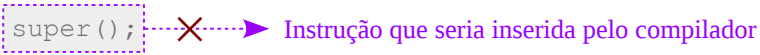
```


`super();` → Inserido pelo compilador

```

public class Funcionario extends Pessoa {
    private double salario;
    public Funcionario(int id, String nome, short idade,
        char sexo, double salario) {
        super();
        super(id, nome, idade, sexo);
        this.salario = salario;
    }
    public Funcionario(int id, String nome) {
        this(id, nome, (short)0, '?', 0.0);
    }
    @Override
    public void imprimir() {
        super.imprimir();
        System.out.printf("salário = $%,.2f%n", salario);
    }
}

```

 Instrução que seria inserida pelo compilador

 Notifica o compilador que estamos “re-escrevendo” um método herdado

Exemplos...

```

Funcionario f1 = new Funcionario(1, "X", (short)33, 'F', 9000.0);
Funcionario f2 = new Funcionario(2, "ABC");
Funcionario f3 = new Funcionario(); // Compila?

f1.imprimir();
f2.imprimir();
f3.imprimir();

Pessoa p = new Pessoa(3, "DEF");
p.imprimir();

```

Habilidades...

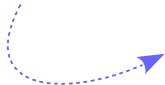
Entender o uso de construtores (sub-rotinas de inicialização) e conceitos relacionados: construtor padrão, sobrecarga, operador "new", "shadowing" e o emprego das palavras super e this.

15 Métodos

Métodos são sub-rotinas (procedimentos ou funções) que pertencem a uma classe e realizam uma determinada tarefa. Variáveis declaradas em um método possuem escopo local e existem a partir do ponto onde foram declaradas e somente dentro do bloco onde foram declaradas. Ao contrário das variáveis de classe, nos métodos as variáveis devem ser explicitamente inicializadas antes do seu uso.

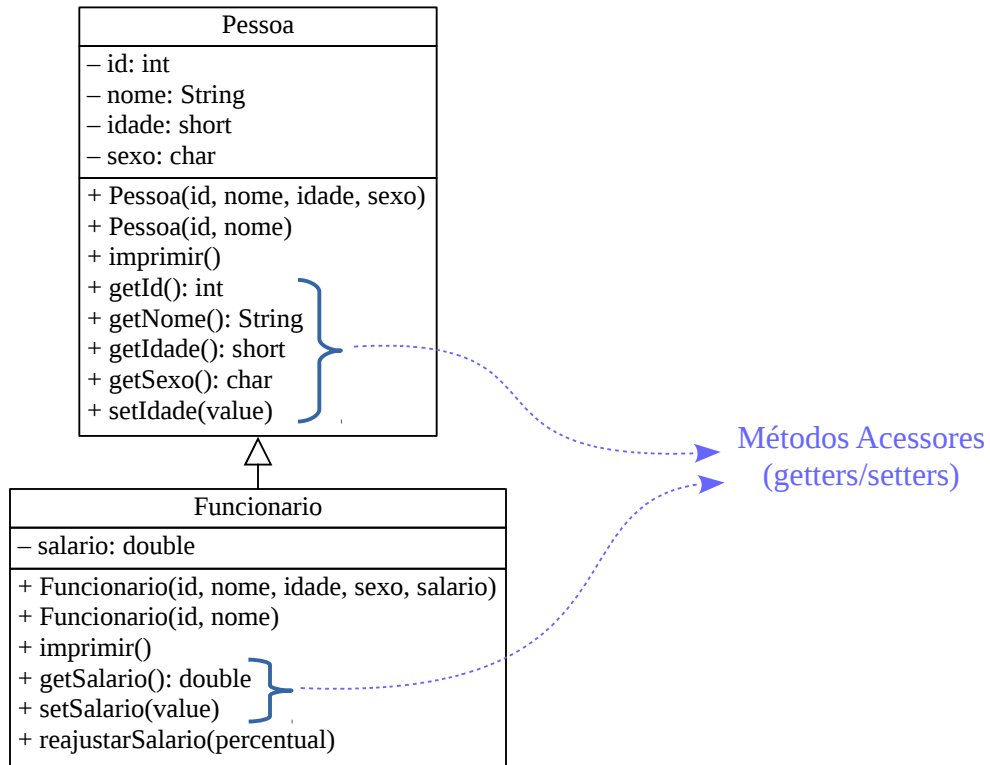
Declaração...

```
{ tipo } nomeMetodo(tipo parametro1, tipo parametro2, ...) {  
    _____  
    _____  
    return [valor];  
}
```



Somente quando não for “void”

A instrução “return” também pode ser utilizada para sair prematuramente de um método.

Versão 3 final da hierarquia Object – Pessoa – Funcionario

```
public class Pessoa {
    private int id;
    private String nome;
    private short idade;
    private char sexo;
    public Pessoa(int id, String nome, short idade, char sexo) {
        this.id = id;
        this.nome = nome;
        this.idade = idade;
        this.sexo = sexo;
    }
    public Pessoa(int id, String nome) {
        this(id, nome, (short)0, '?');
    }
    public void imprimir() {
        System.out.printf("id = %d\nnome = %s\nidade = %d\nsexo = %c\n",
            id, nome, idade, sexo);
    }
    public int getId() {
        return id;
    }
    public String getNome() {
        return nome;
    }
    public short getIdade() {
        return idade;
    }
    public char getSexo() {
        return sexo;
    }
    public void setIdade(short value) {
        idade = value;
    }
}
```

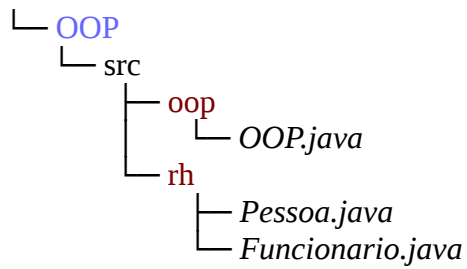
```
public class Funcionario extends Pessoa {
    private double salario;
    public Funcionario(int id, String nome, short idade,
        char sexo, double salario) {
        super(id, nome, idade, sexo);
        this.salario = salario;
    }
    public Funcionario(int id, String nome) {
        this(id, nome, (short)0, '?', 0.0);
    }
    @Override
    public void imprimir() {
        super.imprimir();
        System.out.printf("salário = $%,.2f%n", salario);
    }
    public double getSalario() {
        return salario;
    }
    public void setSalario(double value) {
        salario = value;
    }
    public void reajustarSalario(double percentual) {
        salario *= percentual / 100.0 + 1.0;
    }
}
```

Habilidades...

Entender o conceito de sub-rotinas e, principalmente, saber quando definir um método. Compreender o que são métodos acessores.

App: OOP

Apps



Em OOP.java, testar:

1. Geração de instâncias de Pessoa e Funcionario.
2. Invocação de métodos.
3. Acessibilidade de membros (Intellisense do NetBeans).

O que você faria para distribuir a funcionalidade (biblioteca) Pessoa – Funcionario?

package com.github.usuario.rh;

16 Argumentos x Parâmetros

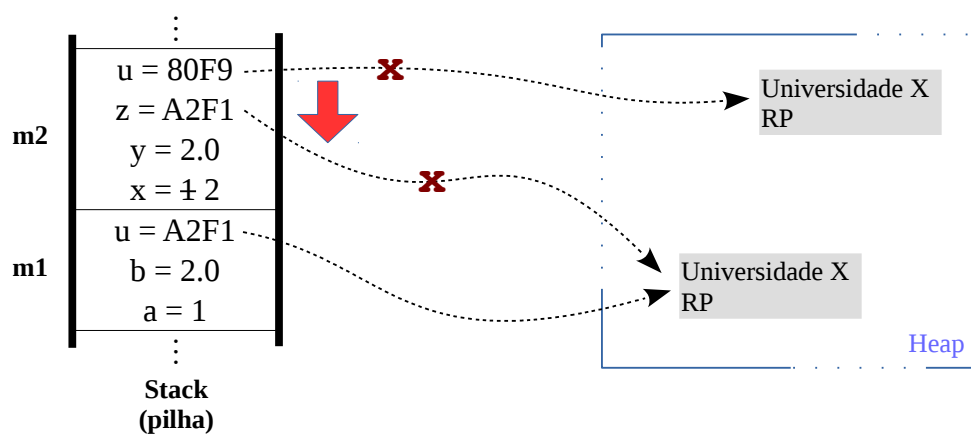
Parâmetros: variáveis que aparecem na definição do método

Argumentos: valores atribuídos (passados) aos parâmetros durante uma chamada

```
void m1() {
    int a = 1;
    double b = 2.0;
    Univ u = new Univ("Universidade X", "RP");
    m2(a, b, u);
    :
    u = null;
    :
}
```

Argumentos em Java são copiados
sempre por valor

```
void m2(int x, double y, Univ z) {
    x += 1;
    Univ u = new Univ(z);
    :
}
```



Habilidades...

Reconhecer que alterações em tipos primitivos locais não afetam o valor original dos argumentos. Compreender o mecanismo de alteração de um argumento do tipo referência por meio de acesso a um de seus membros.

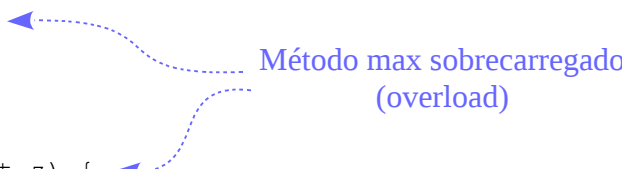
17 Sobrecarga de métodos (overload)

Considere o trecho de código a seguir:

```
int max2(int x, int y) {  
    return x > y ? x : y;  
}  
  
int max3(int x, int y, int z) {  
    return max2(x, max2(y, z));  
}
```

A finalidade de ambos os métodos é retornar o maior número inteiro. Métodos com a mesma finalidade podem compartilhar de um mesmo nome, desde que apresentem assinaturas diferentes.

```
int max(int x, int y) {  
    return x > y ? x : y;  
}  
  
int max(int x, int y, int z) {  
    return max(x, max(y, z));  
}
```



Método max sobrecarregado (overload)

Exemplos:

```
System.out.println(max(1, 2));  
System.out.println(max(3, 4, 2));
```

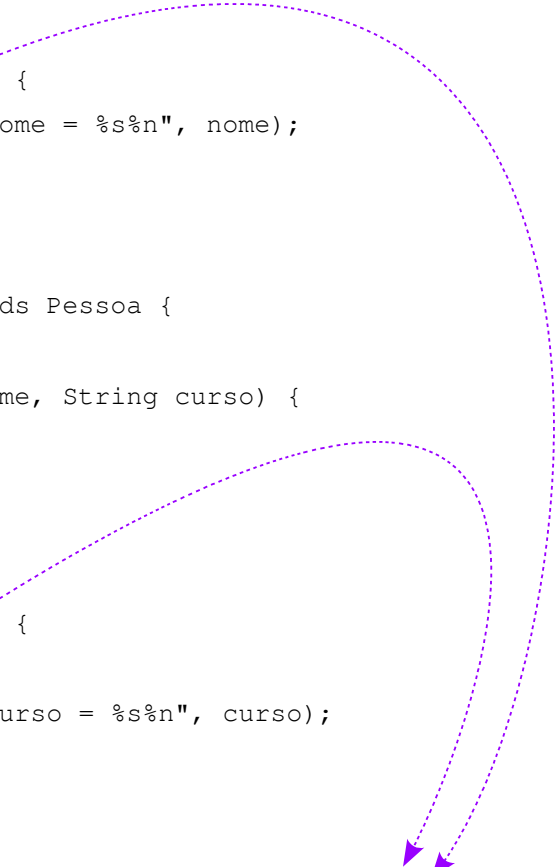
O compilador sabe qual método invocar com base no número e tipo dos argumentos e parâmetros envolvidos na chamada.

Habilidades...

Saber quando e como sobrecarregar um método. Entender as regras para que uma sobrecarga seja permitida.

18 Sobrescrita de métodos (override)

```
public class Pessoa {  
    private String nome;  
    public Pessoa(String nome) {  
        this.nome = nome;  
    }  
    public void imprimir() {  
        System.out.printf("Nome = %s\n", nome);  
    }  
}  
  
public class Aluno extends Pessoa {  
    private String curso;  
    public Aluno(String nome, String curso) {  
        super(nome);  
        this.curso = curso;  
    }  
    @Override  
    public void imprimir() {  
        super.imprimir();  
        System.out.printf("Curso = %s\n", curso);  
    }  
}
```



A sobrescrita (override) ocorre quando o mesmo nome e assinatura (parâmetros) são utilizados por classes diferentes que se relacionam por herança.

Métodos sobrescritos apresentam comportamento polimórfico. A versão a ser invocada não depende do tipo da variável, mas sim do tipo do objeto referenciado (instância).

Exemplo:

```
Pessoa p = new Aluno("Eurípides", "Matemática");  
p.imprimir();
```

Habilidades...

Compreender a diferença entre overload e override. Entender a importância da anotação `@Override` em Java.

19 Membros estáticos

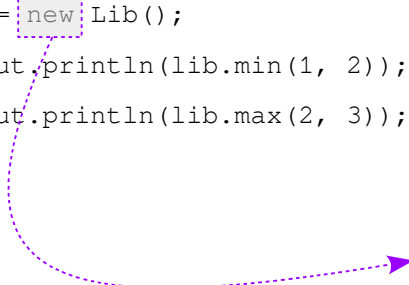
Membros não-estáticos estão associados aos objetos de uma classe (instâncias). Por sua vez, membros estáticos estão associados às classes.

Considere a classe Lib a seguir:

```
public class Lib {  
    public int min(int a, int b) {  
        return a < b ? a : b;  
    }  
    public int max(int a, int b) {  
        return a > b ? a : b;  
    }  
}
```

Para utilizarmos os métodos da classe, fazemos:

```
Lib lib = new Lib();  
System.out.println(lib.min(1, 2));  
System.out.println(lib.max(2, 3));
```



Uma instância da classe é necessária porque os métodos são membros de instância: **métodos não-estáticos**.

Membros cujo comportamento/finalidade não dependem ou não exigem um objeto com um estado em particular para agir podem ser declarados como estáticos.

```
public class Lib {  
    public static int min(int a, int b) {  
        return a < b ? a : b;  
    }  
    public static int max(int a, int b) {  
        return a > b ? a : b;  
    }  
}
```

Forma de uso:

```
System.out.println(Lib.min(1, 2));  
System.out.println(Lib.max(2, 3));
```

Pergunta!

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

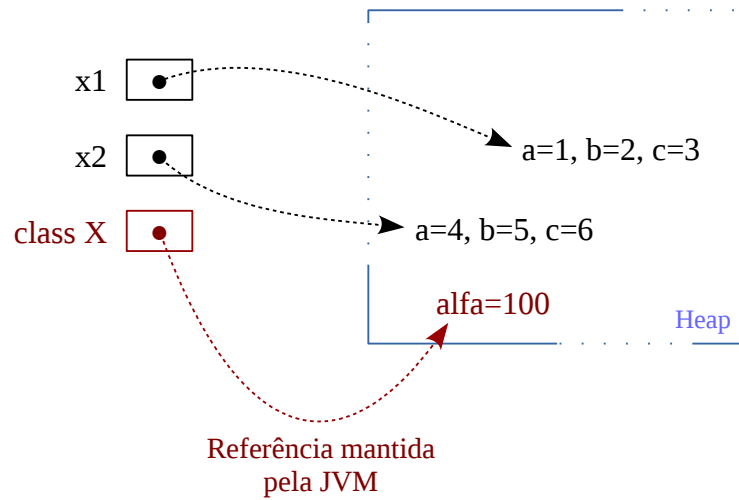
Por que o método “main” é declarado como estático?

Campos estáticos

```
public class X {  
    public static int alfa = 100;  
    private int a, b, c;  
    public X(int a, int b, int c) {  
        this.a = a;  
        this.b = b;  
        this.c = c;  
    }  
}
```

```
X x1 = new X(1, 2, 3);
```

```
X x2 = new X(4, 5, 6);
```



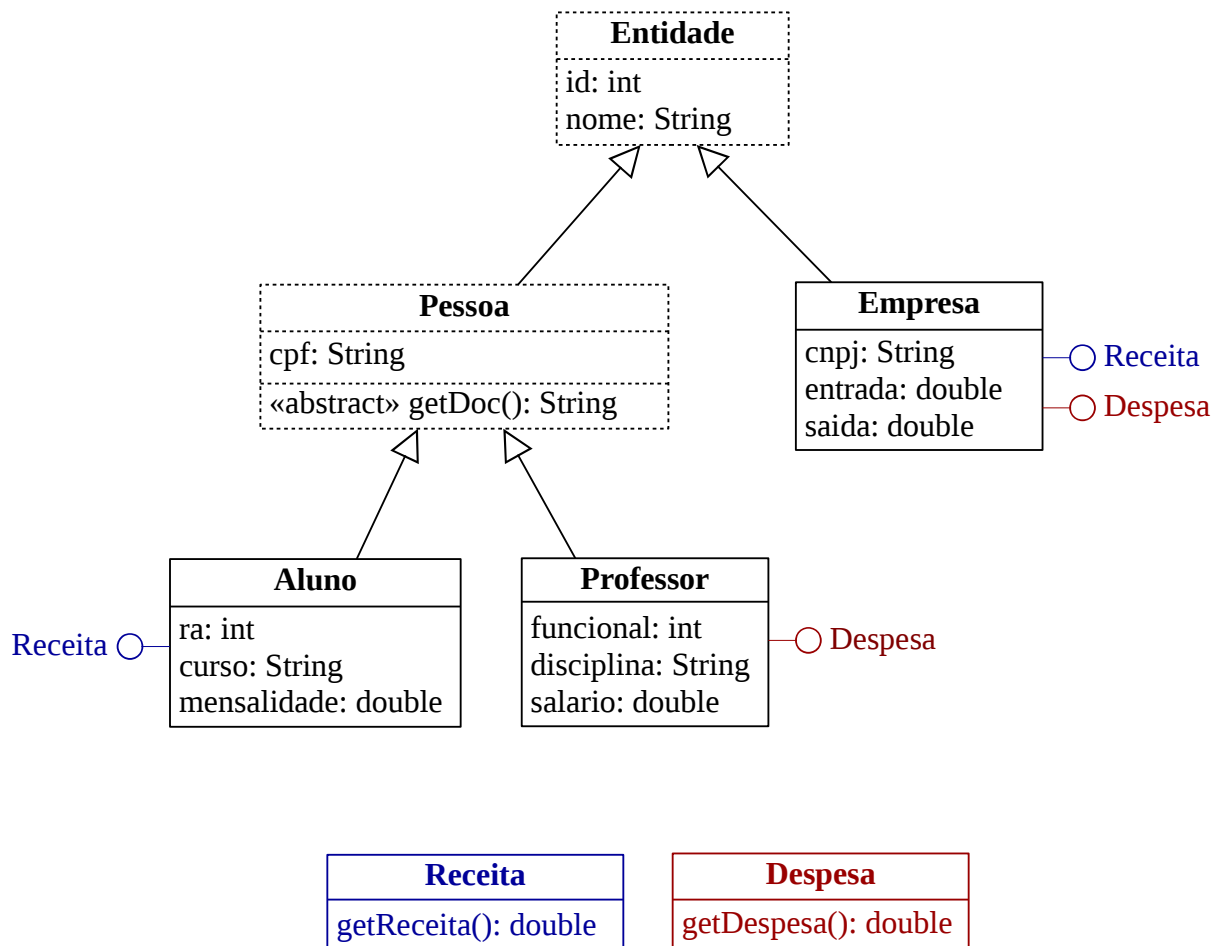
Habilidades...

Compreender o conceito e entender quando definir membros estáticos.

20 Interfaces e classes abstratas

Universidade

“versão simplificada”



Uma interface possibilita que classes não relacionadas por uma hierarquia compartilhem de constantes e métodos comuns.

```
public interface Receita {  
    double getReceita();  
}
```

```
public interface Despesa {  
    double getDespesa();  
}
```

```
public abstract class Entidade {  
    ...  
}
```

➤ Não existirão instâncias da classe Entidade

```
public abstract class Pessoa extends Entidade {  
    ...  
    public abstract String getDoc();  
}
```

➤ Sub-classes concretas devem implementar o método

```
public class Aluno extends Pessoa implements Receita {  
    ...  
    @Override  
    public String getDoc() {  
        ...  
    }  
    @Override  
    public double getReceita() {  
        ...  
    }  
}
```

Ao implementar uma interface, o programador “assina” um contrato com o compilador prometendo definir todos os métodos abstratos da interface ou, alternativamente, adiar a definição declarando a própria classe como abstrata.

```
public class Professor extends Pessoa implements Despesa {  
    ...  
}
```

```
public class Empresa extends Entidade implements Receita, Despesa {  
    ...  
}
```

Exemplos...

```
Aluno a1 = new Aluno(...);
Aluno a2 = new Aluno(...);
Empresa e1 = new Empresa(...);
```

Utilizando a super-classe comum mais próxima...

```
Entidade[] entidades = new Entidade[] { a1, a2, e1 };
double total = 0.0;
for (Entidade entidade : entidades) {
    if (entidade instanceof Aluno) {
        Aluno aluno = (Aluno)entidade;
        total += aluno.getMensalidade();
    } else if (entidade instanceof Empresa) {
        Empresa empresa = (Empresa)entidade;
        total += empresa.getEntrada();
    }
}
System.out.println(total);
```

Utilizando uma interface...

```
Receita[] receitas = new Receita[] { a1, a2, e1 };
double total = 0.0;
for (Receita receita : receitas) total += receita.getReceita();
System.out.println(total);
```

Polimorfismo → Permite que objetos e comportamentos específicos sejam manipulados por meio de uma referência mais genérica, neste caso, interfaces.

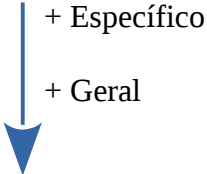
Habilidades...

Compreender que a aplicação pode ser facilmente estendida sem a necessidade de alterações onde o processamento dos dados é feito de forma polimórfica.

21 Exceptions

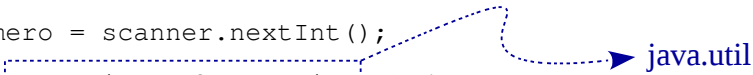
Mecanismo geral de tratamento de exceções...

```
try {
    Bloco de código a ser protegido e/ou aquisição de recursos
    "Possível ocorrência de uma exceção"
} catch (TipoExceção1 e) {
    Tratamento para exceções que se enquadram em TipoExceção1
} catch (TipoExceção2 e) {
    Tratamento para exceções que se enquadram em TipoExceção2
} finally {
    Liberação de recursos
}
```



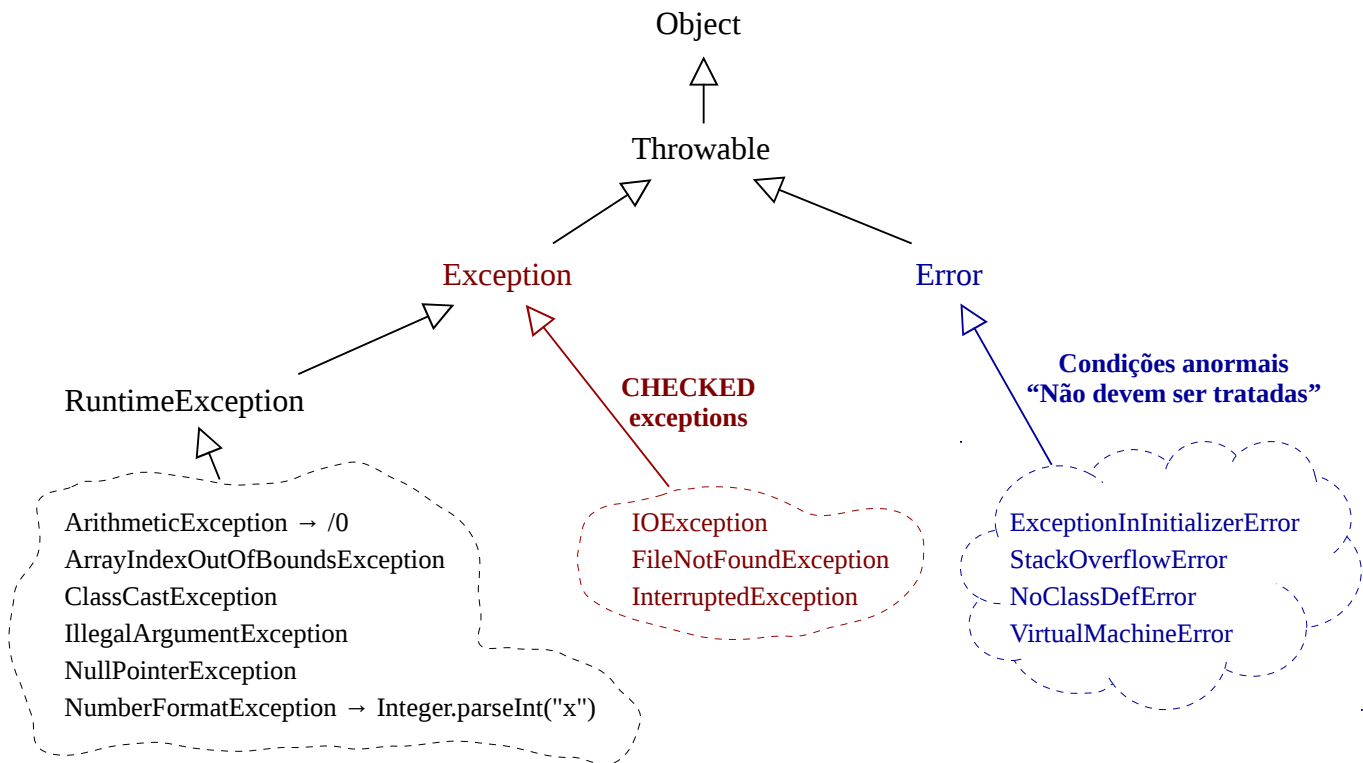
Introduzindo tolerância a falhas...

```
Scanner scanner = new Scanner(System.in);
try {
    int numero = scanner.nextInt();
} catch (InputMismatchException e) {
    System.out.println("Entrada inválida");
}
```



Desafio!

Reconstruir App CalculadoraInteiros. A aplicação não deve abortar abruptamente caso o usuário não digite um número válido.



Regra para CHECKED exceptions...

1. Capturar e tratar

```

try {
    ...
} catch (...) {
    // Rotina de tratamento da exceção
}
  
```

2. Declarar

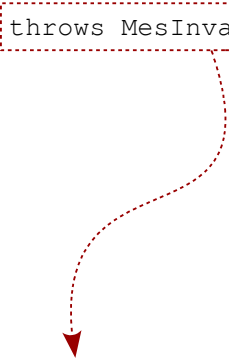
```

void m() throws ... {
    ...
}
  
```


Criando exceções...

```
public class MesInvalidoException extends RuntimeException {
    private int mes;
    public MesInvalidoException(int mes) {
        super(String.format("Mês %d inválido", mes));
        this.mes = mes;
    }
    public int getMes() {
        return mes;
    }
}

public class Lib {
    private static String[] meses = {
        "Janeiro", "Fevereiro", "Março", "Abril", "Maio", "Junho",
        "Julho", "Agosto", "Setembro", "Outubro", "Novembro", "Dezembro" };
    public static String getDescricaoMes(int mes) throws MesInvalidoException {
        if (mes >= 1 && mes <= 12)
            return meses[mes - 1];
        else
            throw new MesInvalidoException(mes);
    }
}
```



Obrigatório se o método lançar uma exceção
MesInvalidoException do tipo CHECKED

Obrigatório se o método não tratar uma possível ocorrência de uma exceção `MesInvalidoException` do tipo `CHECKED`

```
public class App {  
    public static void main(String[] args) throws MesInvalidoException {  
        System.out.println(Lib.getDescricaoMes(1));  
        try {  
            System.out.println(Lib.getDescricaoMes(13));  
        } catch (MesInvalidoException e) {  
            System.out.println(e.getMessage());  
        }  
        System.out.println(Lib.getDescricaoMes(13));  
    }  
}
```

Exceção lançada e capturada

Exceção lançada e não capturada

Resultado...

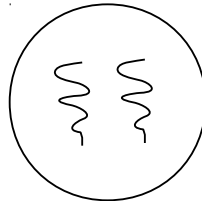
Janeiro
Mês 13 inválido
Exception in thread ...

Habilidades...

Entender o mecanismo de tratamento de exceções e a diferença entre exceções comuns e do tipo "checked".

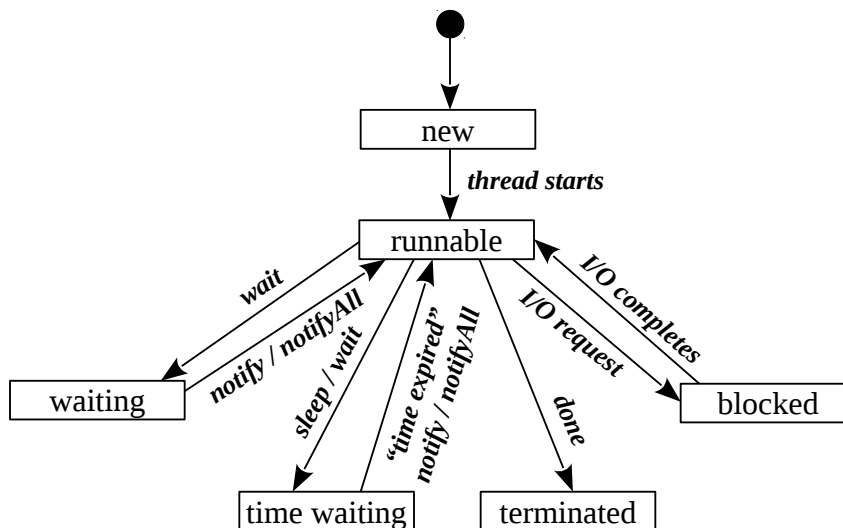
22 Concorrência x Paralelismo x Multithreading

A programação concorrente é feita por meio de threads. O número de CPUs disponíveis influencia fortemente na possibilidade ou não de execução paralela de segmentos de código independentes.

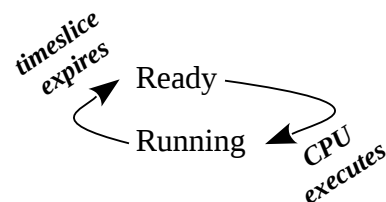


Processo com duas threads

Ciclo de vida de uma thread...



Quando no estado **runnable**...



“Java é projetado para ser simples, mas a programação concorrente não é fácil.”

1º passo → Definir classe responsável pela tarefa concorrente

```
import java.security.SecureRandom;

public class Task implements Runnable {
    private static SecureRandom r = new SecureRandom();
    private String task;
    private int time;
    public Task(String task) {
        this.task = task;
        this.time = r.nextInt(5001);
        System.out.println("Task " + task + " created");
    }
    @Override
    public void run() {
        try {
            System.out.println("Task " + task + ", sleep = " + time);
            Thread.sleep(time);
        } catch (InterruptedException e) {
            return;
        }
        System.out.println("Task " + task + " completed");
    }
}
```

Objeto gerador de número aleatórios

0 .. 5000 ms

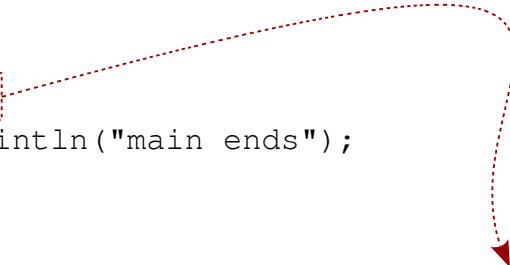
“Thread entry-point”
Tarefa a ser executada
concorrentemente

A thread pode ser interrompida enquanto dormente

2º passo → Iniciar execução concorrente

```
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class TaskRunner {
    private static ExecutorService s = Executors.newCachedThreadPool();
    public static void main(String[] args) {
        System.out.println("main starts");
        Task a = new Task("A");
        Task b = new Task("B");
        Task c = new Task("C");
        s.execute(a);
        s.execute(b);
        s.execute(c);
        s.shutdown();
        System.out.println("main ends");
    }
}
```



Tarefas agendadas e em execução permanecem ativas,
mas o serviço não aceita novas requisições de execução.

Framework Executor → Balanceamento automático de carga com base na capacidade de processamento, ou seja, número de CPUs e disponibilidade de recursos (memória). Tarefas concorrentes são automaticamente alocadas a uma thread disponível no **pool**.

Threads: Problemas de concorrência

```
public class Number {
    private int value;
    public void inc() {
        value++;
    }
    public int getValue() {
        return value;
    }
}

public class Add implements Runnable {
    private Number number;
    public Add(Number number) {
        this.number = number;
    }
    @Override
    public void run() {
        for (int i = 0; i < 1_000_000; i++) {
            number.inc();
        }
    }
}
```

```

import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class App {
    private static ExecutorService s = Executors.newCachedThreadPool();
    public static void main(String[] args) throws InterruptedException{
        Number number = new Number();
        Add add = new Add(number);
        s.execute(add);
        s.execute(add);
        s.execute(add);
        s.execute(add);
        s.shutdown();
        while (!s.isTerminated()) {
            System.out.println("Computing...");
            Thread.sleep(1000);
        }
        System.out.println(number.getValue());
    }
}

```

Data Race Problem
 inc() não é thread-safe

Solução

Tornar a operação atômica por meio de exclusão mútua

```

public synchronized void inc() {
    value++;
}

```

ou...

```

public void inc() {
    synchronized (this) {
        value++;
    }
}

```

Habilidades...

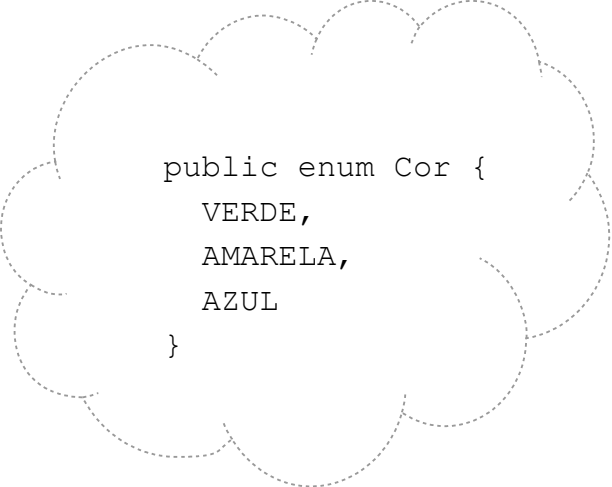
Entender o conceito e os fundamentos básicos de multiprogramação em Java.

23 Enumerações

Enumerações em Java são um tipo especial de classe com um conjunto de constantes representadas por identificadores únicos (os enumeradores). As constantes são, de fato, instâncias da própria classe.

enum ^{class} → CONSTANTES...
campos
construtores
métodos

```
public enum Pais {
    ALEMANHA(49),
    ARGENTINA(54),
    AUSTRALIA(61),
    BRASIL(55),
    EUA(1);
    private final int ddi;
    private Pais(int ddi) {
        this.ddi = ddi;
    }
    public int getDDI() {
        return ddi;
    }
}
```



```
public enum Cor {
    VERDE,
    AMARELA,
    AZUL
}
```

Exemplo...

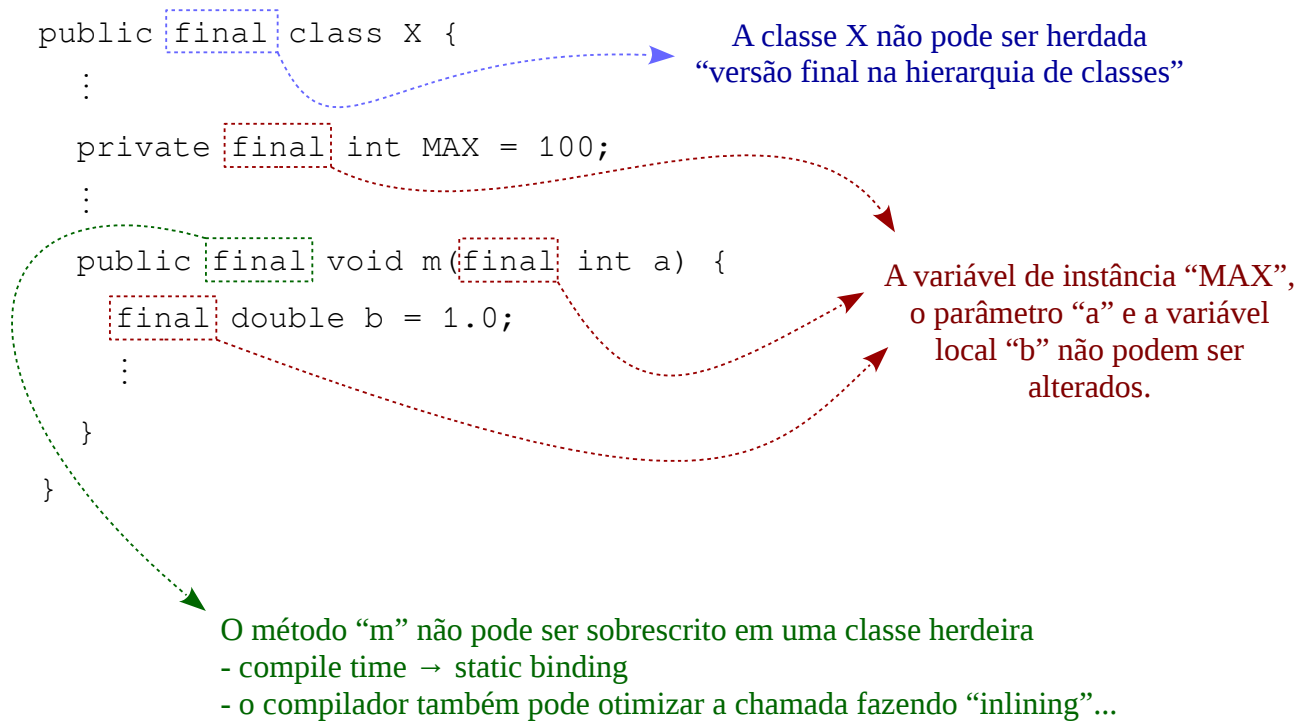
```
for (Pais pais : Pais.values()) {
    System.out.println(pais + ", " + pais.getDDI());
}
```

Habilidades...

Reconhecer a existência de enumerações e, principalmente, quando utilizá-las.

24 A palavra-chave “final”

Utilizada para indicar o estado definitivo (final) de um conceito e impedir alterações acidentais.



Habilidades...

Compreender a importância de "final".

25 Revisão e trabalho

keywords...

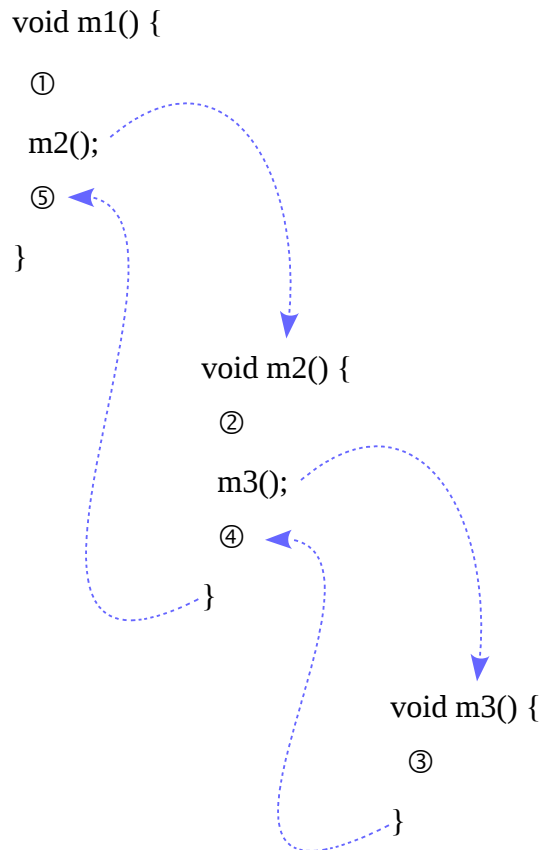
abstract	continue	for	new	switch
assert	default	if	package	synchronized
boolean	do	goto	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile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

Literais: true, false, null

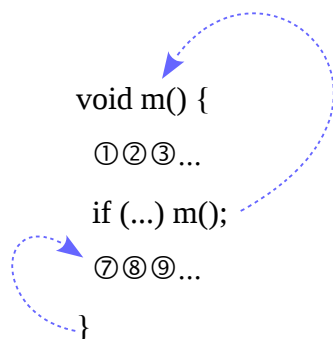
Trabalho →

26 Tópicos adicionais

Encadeamento tradicional de chamada de métodos (empilhamento)



Recursividade: Um método recursivo é um método que chama a si próprio direta ou indiretamente por meio de outro método.



BigInteger

```
import java.math.BigInteger;

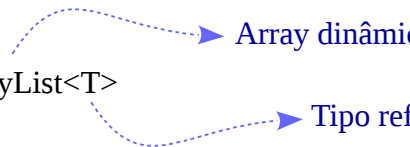
public class AppFatorial {
    public static long fatorialLong(long number) {
        if (number <= 1)
            return 1;
        else
            return number * fatorialLong(number - 1);
    }

    public static BigInteger fatorialBig(BigInteger number) {
        if (number.compareTo(BigInteger.ONE) <= 0)
            return BigInteger.ONE;
        else
            return number.multiply(fatorialBig(number.subtract(BigInteger.ONE)));
    }

    public static void main(String[] args) {
        for (int i = 0; i <= 50; i++) {
            System.out.printf("%d! = %d%n", i, fatorialLong(i));
        }
    }
}
```

► Experimentar...
fatorialBig(BigInteger.valueOf(i))

Boxing / Unboxing



 java.util.ArrayList<T>

ArrayList<int> n = new ArrayList<>(); → **NÃO compila**

Solução: Utilizar uma classe wrapper

Tipo primitivo	Classe wrapper (java.lang)
boolean	Boolean
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double

Uma classe wrapper permite que tipos primitivos sejam manipulados por meio de um objeto.

Exemplos...

```

ArrayList<Integer> n = new ArrayList<>();
n.add(new Integer(10));
n.add(new Integer(20));
n.add(new Integer(30));
int x = n.get(0).intValue();

```

ou...

```

ArrayList<Integer> n = new ArrayList<>();
n.add(10); // boxing: 10 é transformado em um objeto Integer(10)
n.add(20); // boxing: 20 é transformado em um objeto Integer(20)
n.add(30); // boxing: 30 é transformado em um objeto Integer(30)
int x = n.get(0); // unboxing: n.get(0).intValue()

```

Boxing / Unboxing: Conversões automáticas entre tipos primitivos e classes wrapper.

Exemplos...

```
void m(Object o) {
    System.out.println(o);
}
```

Chamadas... `m("abc");`
 `m(99); // boxing: new Integer(99)`

```
void n(int x) {
    System.out.println(x);
}
```

Chamadas... `n(9)`
 `Integer i = new Integer(99);`
 `n(i); // unboxing: i.intValue()`

Sistemas numéricos

Nr. inteiro 267

`int x = 267; // Sistema numérico decimal`

$(267)_{10} = (1.0000.1011)_2 = (413)_8 = (10B)_{16}$
 decimal binário octal hexadecimal

`int x = 267;`

`int x = 0b100001011; // ou 0b1_0000_1011`

`int x = 0413; // octal (Cuidado: 10 + 0413 = 277)`

`int x = 0x10B;`

Regras de agrupamento

	Certo	Errado
Decimal	1_000_000	1_000_000_
Binário	0b1111_1111	0b_1111_1111
Octal	0_413	0_413_
Hexadecimal	0xFF_FF	0x_FF_FF