**Anotações – Java 2024**

Auxilio: Curso mais didático e completo de Java e OO, UML, JDBC, JavaFX, Spring Boot, JPA, Hibernate, MySQL, MongoDB e muito mais.

Plataforma: Udemy

Professor: Nelio Alves

Algoritmo

Sequência finita de instruções para se resolver um problema.

Exemplo:

Problema: lavar roupa suja

Algoritmo:

1. Colocar a roupa em um recipiente;
2. Colocar um pouco de sabão e amaciante;
3. Encher de água;
4. Mexer tudo até dissolver todo o sabão;
5. Deixar de molho por vinte minutos;
6. Esfregar a roupa;
7. Enxaguar;
8. Torcer.

Automação

Consiste em utilizar máquina(s) para executar o procedimento desejado de forma automática ou semiautomática.

Computador

Hardware – parte física (a máquina em si)

Software – parte lógica (programas)

* Sistema operacional (Windows, Linux, iOS);
* Aplicativos (apps de escritório, app de câmera, navegador web);
* Jogos;
* Utilitários (Antivírus, compactador de arquivos);
* Outros.

Programa > Algoritmo

Programas de computador **são algoritmos** executados pelo computador (em linhas gerais).

Conclusão: o computador é uma máquina de **automatiza** a execução de **algoritmos**.

Qualquer algoritmo? Não. Apenas algoritmos computacionais:

* Processamento de dados;
* Cálculos.

Linguagem de programação, léxica, sintática.

Linguagem de programação:

É um conjunto de regras **léxicas** (ortografia) e **sintáticas** (gramática) para se escrever programas.

Léxica

Diz respeito à correção das palavras “isoladas” (ortografia).

Exemplo (Português):

* cachorro – correto
* caxorro – errado

Exemplo (Linguagem de programação):

* main – correto
* maim – errado

Sintática

Diz respeito à correção das **sentenças** (gramática).

Exemplo (Português):

* O cachorro está com fome - certo
* A cachorro está com fome – errado

Exemplo (Linguagem de programação):

* x = 2 + y; - certo
* x = + 2 y; - errado

Exemplo de um programa:

Suponha um programa que solicita do usuário dois números e depois mostra na média aritmética deles:

*Digite o primeiro numero: 3*

*Digite o segundo numero: 6*

*Media = 4.5*

Solução em linguagem C

*#include <stdio.h>*

*int main() {*

*double x, y, media;*

*printf(“Digite o primeiro numero: “);*

*scanf(“%1f”, &x);*

*printf(“Digite o segundo numero: “);*

*scanf(“%1f”, &y);*

*media = (x + y) / 2.0;*

*printf(“Media = %.1f\n”, media);*

*return 0 ;*

*}*

Solução em linguagem C++

*#include <iostream>*

*using namespace std;*

*int main() {*

*double x, y, media;*

*cout << “Digite o primeiro numero: “;*

*cout >> x;*

*cin >> y;*

*media = (x + y) / 2.0;*

*cout << “Media = “ << media << endl;*

*return 0;*

*}*

Solução em linguagem C#

*using System;*

*namespace programa {*

*class Program {*

*static void Main(string[] args) {*

*double x, y, media;*

*Console.Write(“Digite o primeiro numero: “);*

*x = double.Parse(Console.ReadLine());*

*y = double.Parse(Console.ReadLine());*

*media = (x + y) / 2.0;*

*Console.WriteLine(“Media = “ + media);*

*}*

*}*

*}*

Solução em linguagem Java

*import java.util.Scanner;*

*public class Main {*

*public static void main(String[] args) {*

*Scanner sc = new Scanner(System.in);*

*double x, y, media;*

*System.out.print(“Digite o primeiro numero: “);*

*x = sc.nextDouble();*

*System.out.print(“Digite o segundo numero: “);*

*y = sc.nextDouble();*

*media = (x + y) / 2.0;*

*System.out.println(“Media = “ + media);*

*sc.close();*

*}*

*}*

**IDE – Ambiente Integrado de Desenvolvimento**

É um conjunto de softwares utilizado para a construção de programas.

Exemplos:

* C/C++ : Code Blocks
* Java: Eclipse, NetBeans
* C#: Microsoft Visual Studio

Funcionalidades de uma IDE

* Edição de código fonte (endentação, autocompletar, destaque de palavras, etc.);
* Depuração e testes;
* Construção do produto final (build);
* Sugestão de modelos (templates);
* Auxiliar em várias tarefas do seu projeto;
* Etc.

Compilação e interpretação – Código fonte e objeto Máquina virtual

Código fonte: é aquele escrito pelo programador em linguagem de programação

Compilação

Código fonte -> **compilador: compilação** | *análise léxica + análise sintática* -> Código objeto -> **Gerador de código: construção (build)** | *geração de código* -> Código executável -> **Execução**

**Exemplos de linguagem que tipicamente usam essa abordagem: C, C++**

Interpretação

Código fonte -> **Interpretador: interpretação** | *análise léxica + análise sintática + geração de código SOB DEMANDA* -> **Execução**

Abordagem híbrida

Código fonte **-> Compilador: precompilação** | *análise léxica + análise sintática* -> Bytecode -> **Máquina virtual: interpretação** | *geração de código SOB DEMANDA* -> Execução

**Exemplos de linguagem que tipicamente usam essa abordagem: Java (JVM), C# (Microsoft .NET Framework)**

**Vantagens**

**Compilação:**

* **velocidade do programa**
* **auxílio do compilador antes da execução**

**Interpretação:**

* **flexibilidade de manutenção do aplicativo em produção**
* **expressividade da linguagem**
* **código fonte não precisa ser recompilado para rodar em plataformas diferentes**

**Abordagem híbrida:**

* velocidade do programa
* **auxílio do compilador antes da execução**
* flexibilidade de manutenção do aplicativo em produção
* **código fonte não precisa ser recompilado para rodar em plataformas diferentes**

**C/C++**

Código fonte

Código executável (específico para o sistema operacional)

Sistema Operacional

Hardware

**PHP, Python, JavaScript**

Código fonte

Interpretador (específico para o sistema operacional)

Sistema Operacional

Hardware

**Java, C#**

Código fonte

Bytecode (código precompilado)

Máquina virtual (específica para o sistema operacional)

Sistema Operacional

Hardware

**Java – contextualização**

O que é Java?

Linguagem de programação (regras sintáticas)

Plataforma de desenvolvimento e execução

* Bibliotecas (API)
* Ambientes de execução

Histórico

Problemas resolvidos e motivo do seu sucesso:

* Ponteiros / gerenciamento de memória
* Portabilidade falha: reescrever parte do código ao mudar de SO
* Utilização em dispositivos diversos
* Custo

Criada pela Sun Microsystems no meio da década de 1990

Adquirida pela Oracle Corporation em 2010.

**Aspectos notáveis**

* Código compilado para bytecode e executado em máquina virtual (JVM)
* Portável, segura, robusta
* Roda em vários tipos de dispositivos
* Domina o mercado corporativo desde o fim do século 20
* Padrão Android por muitos anos

**Edições**

* Java ME – Java Micro Edition – dispositivos embarcados e móveis – IoT
* <http://www.oracle.com/technetwork/java/javame>
* Java SE – Java Standard Edition – core – desktop e servidores
* <http://www.oracle.com/technetwork/java/javase>
* Java EE – Java Enterprise Edition – aplicações corporativas
* <http://www.oracle.com/technetwork/java/javaee>

**Plataforma Java SE**

**JVM – Java Virtual Machine**

* Máquina virtual do Java – necessário para executar sistemas Java

**Compilação e interpretação**

* Linguagens **compiladas**: C, C++
* Linguagens **interpretadas**: PHP, JavaScript
* Linguagens **pré-compiladas + máquina virtual**: Java, C#

**Estrutura de uma aplicação Java**

Uma aplicação é composta por classes.

package = agrupamento LÓGICO de classes relacionadas.

Módulo (Java 9+) = Agrupamento lógico de pacotes relacionados.

Runtime = Agrupamento físico.

Aplicação = Agrupamento de módulo relacionados.

**Expressões aritméticas**

Expressão – resultado -> valor numérico

4 + 5 resultado -> 9

**Operadores aritméticos**

*C, C++, Java, C#*

**Operador Significado**

+ adição

- subtração

\* multiplicação

/ divisão

% resto da divisão (“mod”)

**Precedência**: 1º lugar: \* / % 2º lugar: + -

Exemplos de expressões aritméticas

2 \* 6 / 3 Resultado = 4

3 + 2 \* 4 Resultado = 11

(3 + 2) \* 4 Resultado = 20

60 / (3 + 2) \* 4 Resultado = 48

60 / ((3 + 2) \* 4) Resultado = 3

Exemplos com operador “mod”

14 % 3 Resultado = 2

19 % 5 Resultado = 4

**Variáveis - Tipos básicos em Java**

Um programa de computador em execução lida com dados.

Como esses dados são armazenados?

* Em **variáveis**!

**Variáveis**

Definição informal:

Em programação, uma variável é uma porção de memória (RAM) utilizada para armazenar dados durante a execução dos programas.

Computador > Memória RAM > x = 3; salario = 5000.0; nome = “Maria”

**Declaração de variáveis**

**Sintaxe:**

<tipo> <nome> = <valor inicial>;

Valor inicial é opcional.

**Exemplos:**

int idade = 25;

double altura = 1.68;

char sexp = ‘F’;

*Memória RAM:*

idade = 25

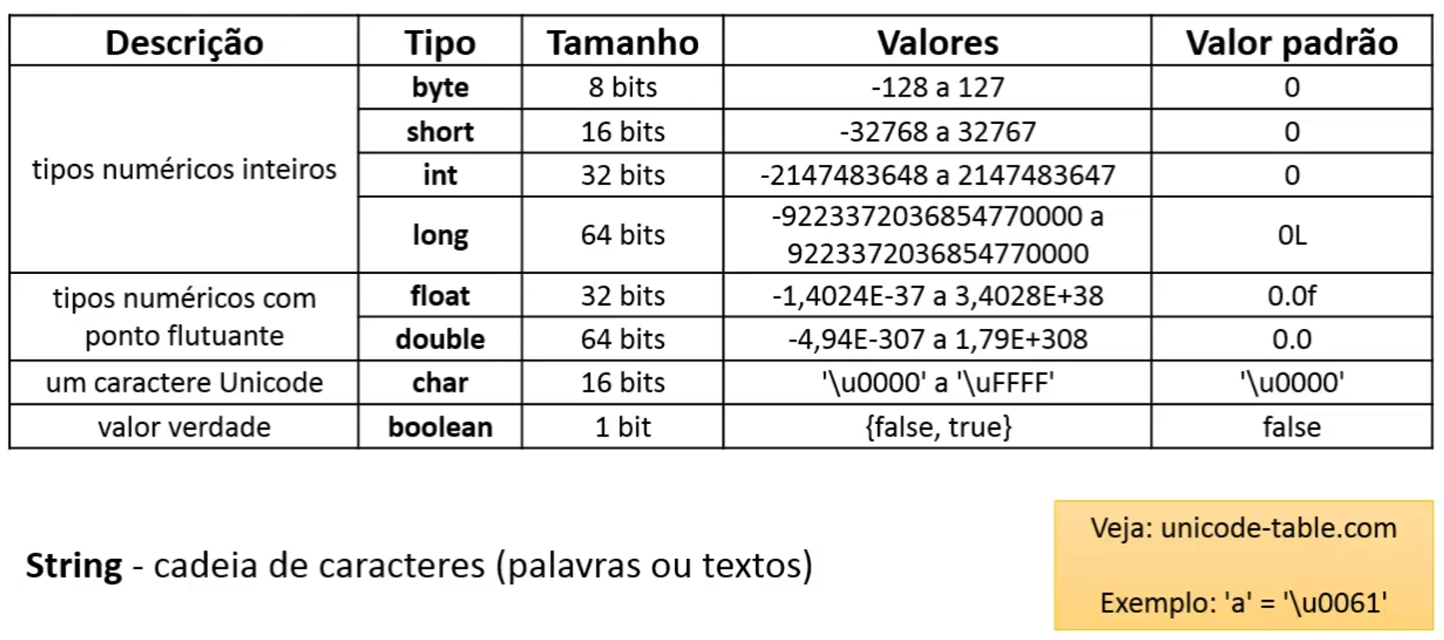
altura = 1.68

sexo = F

**Uma variável possui:**

* Nome (ou identificador)
* Tipo
* Valor
* Endereço

Tipos primitivos em Java



Um bit pode armazenar 2 valores possíveis (0 ou 1)

Cada bit = 2 possibilidades

8 bits:

2 x 2 x 2 x 2 x 2 x 2 x 2 x 2 = 28 = 256 possibilidades

**Nomes de variáveis**

* Não pode começar com dígito: use uma letra ou \_
* Não pode ter espaço em branco
* Não usar acentos ou til
* Sugestão: use o padrão “camel case”

**Errado: Correto:**

*int 5minutos; int \_5minutos;*

*int salário; int salario;*

*int salário do funcionário; int salarioDoFuncionario;*

**As três operações básicas de programação**

Um programa de computador é capaz de realizar essencialmente três operações:

* Entrada de dados;
* Processamento de dados;
* Saída de dados.

**Entrada de dados:**

Usuário -> Programa (dentro de variáveis). Também chamada de LEITURA: “O programa está lendo dados.”

**Processamento de dados:**

É quando o programa realiza os cálculos. O processamento de dados se dá por um comando chamado ATRIBUIÇÃO. media = ( x + y ) / 2.0;

**Saída de dados:**

Programa -> Usuário. Também chamada de ESCRITA: “O programa está escrevendo dados.”

**Saída de dados em Java:**

Para escrever na tela um texto qualquer

**Sem quebra de linha ao final:**

System.out.print(“Bom dia!”);

**Com quebra de linha ao final:**

System.out.println(“Bom dia!”);

Para escrever o conteúdo de uma variável de algum tipo básico:

Suponha uma variável tipo int declarada e iniciada:

*int y = 32;*

*System.out.println(y);*

Para escrever o conteúdo de uma variável com ponto flutuante:

Suponha uma variável tipo double declarada e iniciada:

*double x = 10.35784;*

%n = quebra de linha (independente de plataforma)

*System.out.println(x);*

*System.out.printf(“%.2f%n”, x); - System.out.printf(“%.4f%n”, x);* - Localidade do sistema

**ATENÇÃO:**

Para considerar o separador de decimais como ponto, ANTES da declaração do Scanner, faça:

*Locale.setDefault(Locale.US);*

Para concatenar vários elementos em um mesmo comando de escrita:  
Regra geral para print e println:

elemento1 + elemento2 + elemento3 + ... + elementoN

*System.out.println(“RESULTADO = “* ***+*** *x* ***+*** *“ METROS”);*

Para concatenar vários elementos em um mesmo comando de escrita:

Regra geral para **printf**:

“TEXTO1 %f TEXTO2 %f TEXT03”, variavel1, variavel2

%f = ponto flutuante

%n = quebra de linha

*System.out.printf(“RESULTADO = %.2f metros%n”, x);*

%f = ponto flutuante

%d = inteiro

%s = texto

%n = quebra de linha

*String nome = “Maria”;*

*int idade = 31;*

*double renda = 4000.0;*

*System.out.printf(“%s tem %d anos e ganha R$ %.2f reais%n”, nome, idade, renda);*

*Mais informações: https://docs.oracle.com/javase/tutorial/java/data/numberformat.html*

**Processamento de dados em Java, Casting**

Processamento de dados

Comando de atribuição.

**Sintaxe:**

<variável> = <expressão>;

= Lê-se “recebe”

**Regra:**

1. A expressão é calculada;
2. O resultado da expressão é armazenado na variável.

**Exemplo 1:**

*int x, y;*

*x = 5;*

*y = 2 \* x;*

*System.out.println(x);*

*System.out.println(y);*

**Exemplo 2:**

*int x;*

*double y;*

*x = 5;*

*y = 2 \* x;*

*System.out.println(x);*

*System.out.println(y);*

**Exemplo 3:**

*double b, B, h, area;*

*b = 6.0;*

*B = 8.0;*

*h = 5.0;*

*area = (b + B) / 2.0 \* h;*

*System.out.println(area);*

**Boa prática:**

Sempre indique o tipo do número, se a expressão for de ponto flutuante (não inteira).

Para **double** use:

*.0*

Para **float** use:

*f*

**Exemplo:**

***float*** *b, B, h, area;*

*b = 6f;*

*B = 8f;*

*h = 5f;*

*area = (b + B) / 2f \* h;*

*System.out.println(area);*

**Exemplo 4:**

*int a, g;*

*double resultado;*

*a = 5;*

*g = 2;*

*resultado = (double) a / b;* ***-> casting***

*System.out.println(resultado);*

**Exemplo 5:**

*double j;*

*int p;*

*j = 5.0;*

*p = (int) j;* ***-> casting***

*System.out.println(p);*

Entrada de Dados

**Usuário** -> **Programa** (dentro de variáveis)

Dispositivo de ENTRADA (teclado)

*Também chamada de LEITURA:*

*"O programa está lendo dados."*

**Scanner**

Para fazer entrada de dados, nós vamos criar um objeto do tipo "Scanner" da seguinte forma:

*Scanner sc = new Scanner(System.in);*

**E para funcionar precisamos utilizar:**

*import java.util.Scanner;*

faça sc.close() quando não precisar mais do objeto sc

**Para ler uma palavra (texto sem espaços)**

Suponha uma variável tipo String declarada:

*String x;*

*x = sc.next();*

Exemplo 01:

package entradaDeDados;

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner keyboard = new Scanner(System.***in***);

String nome;

nome = keyboard.next();

System.***out***.println("Voce digitou: " + nome);

keyboard.close();

}

}

**Para ler um número inteiro**

Suponha uma variável tipo int declarada:

*int x;*

*x = keyboard.nextInt();*

Exemplo 02:

package entradaDeDados;

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner keyboard = new Scanner(System.***in***);

String nome;

int idade;

System.***out***.println("Digite o seu nome:");

nome = keyboard.next();

System.***out***.println("Voce digitou: " + nome);

System.***out***.println("Digite a sua idade:");

idade = keyboard.nextInt();

System.***out***.println("Voce digitou: " + idade);

keyboard.close();

}

}

**Para ler um número com ponto flutuante**

Suponha uma variável tipo double declarada:

*double x;*

*x = sc.nextDouble(); -> Localidade do sistema*

ATENÇÃO:

Para considerar o separador de decimais como ponto, ANTES da declaração do Scanner, faça:

*Locale.setDefault(Locale.US);*

Exemplo 03:

package entradaDeDados;

import java.util.Locale;

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Locale.*setDefault*(Locale.***US***);

Scanner keyboard = new Scanner(System.***in***);

String nome;

int idade;

double altura;

System.***out***.println("Digite o seu nome:");

nome = keyboard.next();

System.***out***.println("Voce digitou: " + nome);

System.***out***.println("\nDigite a sua idade:");

idade = keyboard.nextInt();

System.***out***.println("Voce digitou: " + idade);

System.***out***.println("\nDigite a sua altura:");

altura = keyboard.nextDouble();

System.***out***.printf("Voce digitou: %.2f%n", altura);

keyboard.close();

}

}

Locale precisa ser feito antes do Scanner para conseguir receber entrada de dados com . Ao invés de virgula.

Se não quiser usar . e sim virgula, basta não usar o Locale.

**Para ler um caractere**

Suponha uma variável char declarada:

*char x;*

*x = sc.next().charAt(0);*

**Para ler vários dados na mesma linha**

*string x;*

*int y;*

*double z;*

*x = sc.next();*

*y = sc;nextInt();*

*z = sc.nextDouble();*

Para ler um texto ATÉ A QUEBRA DE LINHA

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

String s1, s2, s3;

s1 = sc.nextLine();

s2 = sc.nextLine();

s3 = sc.nextLine();

System.out.println(“Dados Digitados:”);

System.out.println(s1);

System.out.println(s2);

System.out.println(s3);

sc.close();

}

}

**ATENÇÃO: quebra de linha pendente**

int x;

String s1, s2, s3;

x = sc.nextInt();

s1 = sc.nextLine();

s2 = sc.nextLine();

s3 = sc.nextLine();

System.out.println(“DADOS DIGITADOS:”);

System.out.println(x);

System.out.println(s1);

System.out.println(s2);

System.out.println(s3);

Quando você usa um comado de leitura diferente do nextLine() e dá alguma quebra de linha, essa quebra de linha fica “pendente” na entrada padrão.

Se você então fizer um nextLine(), aquela quebra de linha pendente será absorvida pelo nextLine().

**Solução:**

Faça um nextLine() extra antes de fazer o nextLine() do seu interesse.

**Exemplo**:

package entradaDeDados02;

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner sc = new Scanner(System.***in***);

int x;

String s1, s2, s3;

x = sc.nextInt();

sc.nextLine();

s1 = sc.nextLine();

s2 = sc.nextLine();

s3 = sc.nextLine();

System.***out***.println("Dados Digitados:");

System.***out***.println(x);

System.***out***.println(s1);

System.***out***.println(s2);

System.***out***.println(s3);

sc.close();

}

}

Resumo da aula:

**Scanner**

* next()
* nextInt()
* nextDouble()
* next().charAt(0)

**Locale**

**Como ler até a quebra de linha**

* nextLine()
* como limpar o buffer de leitura

**Funções matemáticas:**

Algumas funções matemáticas em Java

**Exemplo: Significado**

A = Math.sqrt(x); Variável A recebe a raiz quadrada de x

A = Math.pow(x, y); Variável A recebe o resultado de x elevado a y

A = Math.abs(x); Variável A recebe o valor absoluto de x

Exemplo:

package funcoesMatematica;

public class Application {

public static void main(String[] args) {

double x = 3.0;

double y = 4.0;

double z = -5.0;

double A, B, C;

A = Math.*sqrt*(x);

B = Math.*sqrt*(y);

C = Math.*sqrt*(25.0);

System.***out***.println("Raiz quadrada de " + x + " = " + A);

System.***out***.println("Raiz quadrada de " + y + " = " + B);

System.***out***.println("Raiz quadrada de 25 = " + C);

A = Math.*pow*(x, y);

B = Math.*pow*(x, 2.0);

C = Math.*pow*(5.0, 2.0);

System.***out***.println(x + " elevado a " + y + " = " + A);

System.***out***.println(x + " elevado ao quadrado = " + B);

System.***out***.println("5 elevado ao quadrado = " + C);

A = Math.*abs*(y);

B = Math.*abs*(z);

System.***out***.println("Valor absoluto de " + y + " = " + A);

System.***out***.println("Valor absoluto de " + z + " = " + B);

}

}

**Incluindo funções em expressões maiores**

*x = -b +- raiz de delta / 2.a*

delta = b² -4ac

delta = Math.pow(b, 2.0) – 4\*a\*c;

x1 = (-b + Math.sqrt(delta)) / (2.0 \* a);

x2 = (-b – Math.sqrt(delta)) / (2.0 \* a);

**Exercício resolvido 01:**

Fazer um programa para ler as medidas da largura e comprimento de um terreno retangular com uma casa decimal, bem como o valor do metro quadrado do terreno com duas casas decimais. Em seguida, o programa deve mostrar o valor da área do terreno, bem como o valor do preço do terreno, ambos com duas casas decimais, conforme exemplo.

Exemplo:

Entrada: Saída:

10.0 AREA = 300.00

30.0 PRECO = 60000.00

200.00

área = largura x comprimento

preço = área x preço do metro quadrado

**Exercícios propostos (No arquivo: 02-exercicios1-estrutura-sequencial.pdf)**

**Exercício 01**

*Correção: https://github.com/acenelio/nivelamento-java/blob/master/src/uri1003.java*

Faça um programa para ler dois valores inteiros, e depois mostrar na tela a soma desses números com uma mensagem explicativa, conforme exemplos.

**Exemplos:**

**Entrada:**

10

30

**Saída:**

SOMA = 40

**Entrada:**

-30

10

**Saída:**

SOMA = -20

**Entrada:**

0

0

**Saída:**

SOMA = 0

**Exercício 02**

*Correção: https://github.com/acenelio/nivelamento-java/blob/master/src/uri1002.java*

Faça um programa para ler o valor do raio de um círculo, e depois mostrar o valor da área deste círculo com quatro

casas decimais conforme exemplos.

Fórmula da área: area = π . raio2

Considere o valor de π = 3.14159

**Exemplos:**

**Entrada:**

2.00

**Saída:**

A=12.5664

**Entrada:**

100.64

**Saída:**

A=31819.3103

**Entrada:**

150.00

**Saída:**

A=70685.7750

**Exercício 03**

*Correção: https://github.com/acenelio/nivelamento-java/blob/master/src/uri1007.java*

Fazer um programa para ler quatro valores inteiros A, B, C e D. A seguir, calcule e mostre a diferença do produto

de A e B pelo produto de C e D segundo a fórmula: DIFERENCA = (A \* B - C \* D).

**Exemplos:**

**Entrada:**

5

6

7

8

**Saída:**

DIFERENCA = -26

**Entrada:**

5

6

-7

8

**Saída:**

DIFERENCA = 86

**Exercício 04**

*Correção: https://github.com/acenelio/nivelamento-java/blob/master/src/uri1008.java*

Fazer um programa que leia o número de um funcionário, seu número de horas trabalhadas, o valor que recebe por

hora e calcula o salário desse funcionário. A seguir, mostre o número e o salário do funcionário, com duas casas

decimais.

Exemplos:

**Entrada:**

25

100

5.50

**Saída:**

NUMBER = 25

SALARY = U$ 550.00

**Entrada:**

1

200

20.50

**Saída:**

NUMBER = 1

SALARY = U$ 4100.00

**Entrada:**

6

145

15.55

**Saída:**

NUMBER = 6

SALARY = U$ 2254.75

**Exercício 05**

*Correção: https://github.com/acenelio/nivelamento-java/blob/master/src/uri1010.java*

Fazer um programa para ler o código de uma peça 1, o número de peças 1, o valor unitário de cada peça 1, o

código de uma peça 2, o número de peças 2 e o valor unitário de cada peça 2. Calcule e mostre o valor a ser pago.

Exemplos:

**Entrada:**

12 1 5.30

16 2 5.10

**Saída:**

VALOR A PAGAR: R$ 15.50

**Entrada:**

13 2 15.30

161 4 5.20

**Saída:**

VALOR A PAGAR: R$ 51.40

**Entrada:**

1 1 15.10

2 1 15.10

**Saída:**

VALOR A PAGAR: R$ 30.20

**Exercício 06**

Correção: https://github.com/acenelio/nivelamento-java/blob/master/src/uri1012.java

Fazer um programa que leia três valores com ponto flutuante de dupla precisão: A, B e C. Em seguida, calcule e

mostre:

a) a área do triângulo retângulo que tem A por base e C por altura.

b) a área do círculo de raio C. (pi = 3.14159)

c) a área do trapézio que tem A e B por bases e C por altura.

d) a área do quadrado que tem lado B.

e) a área do retângulo que tem lados A e B.

Exemplos:

**Entrada:**

3.0 4.0 5.2

**Saída:**

TRIANGULO: 7.800

CIRCULO: 84.949

TRAPEZIO: 18.200

QUADRADO: 16.000

RETANGULO: 12.000

**Entrada:**

12.7 10.4 15.2

**Saída:**

TRIANGULO: 96.520

CIRCULO: 725.833

TRAPEZIO: 175.560

QUADRADO: 108.160

RETANGULO: 132.080

**Expressões comparativas**

expressão -> resultado -> valor verdade

5 > 10 -> Falso

**Operadores comparativos**

C, C++, Java, C#

**Operador Significado**

> maior

< menor

>= maior ou igual

<= menor ou igual

== igual

!= diferente

**Exemplos de expressões comparativas**

(*suponha x igual a 5*)

X > 0 Resultado: V

X == 3 Resultado: F

10 <= 30 Resultado: V

X != 2 Resultado: V

**Expressões lógicas**

expressão -> resultado -> valor verdade

**Operadores lógicos**

C, C++, Java, C#

Operador Significado

&& E

|| OU

! NÃO

***Ideia por trás do operador "E"***

Você pode obter uma habilitação de motorista se:

* For aprovado no exame psicotécnico,

**E**

* For aprovado no exame de legislação,

**E**

* For aprovado no exame de direção

*Todas as condições devem ser verdadeiras!*

**Exemplos de expressões lógicas**

(*suponha x igual a 5*):

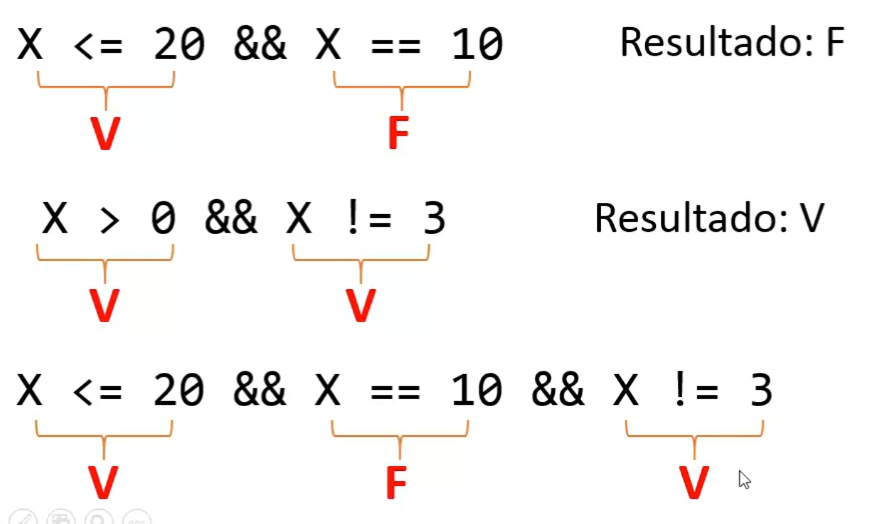


Tabela verdade do operador "**E**"

**A B A&&B**

F F F

F V F

V F F

V V V

**Ideia por trás do operador "OU"**

Você pode obter estacionar na vaga especial se:

* For idoso(a),

**OU**

* For uma pessoa com deficiência,

**OU**

* For uma gestante

*Pelo menos uma condição deve ser verdadeira!*

**Exemplos de expressões lógicas**

(*suponha x igual a 5*)

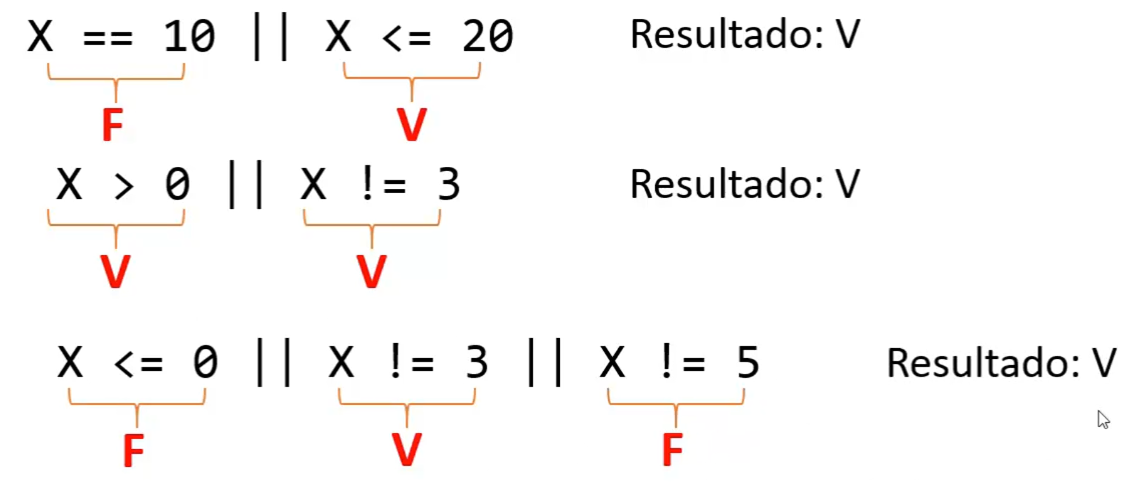


Tabela verdade do operador "**OU**"

**A B A || B**

F F F

F V V

V F V

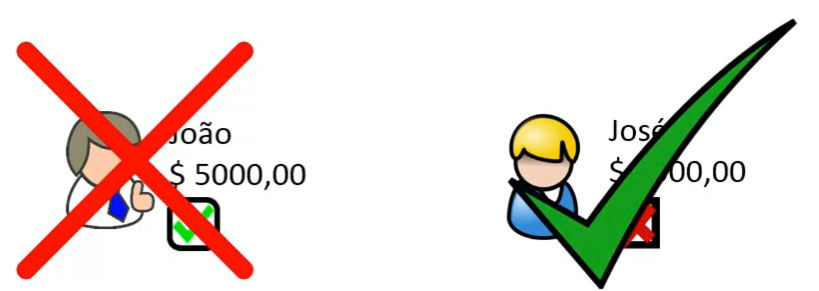
V V V

**Ideia por trás do operador "NÃO"**

Você tem direito a receber uma bolsa de estudos se você:

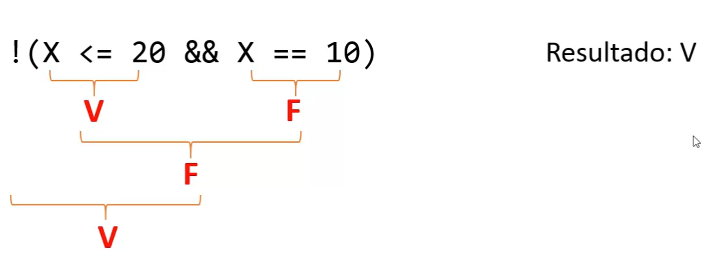
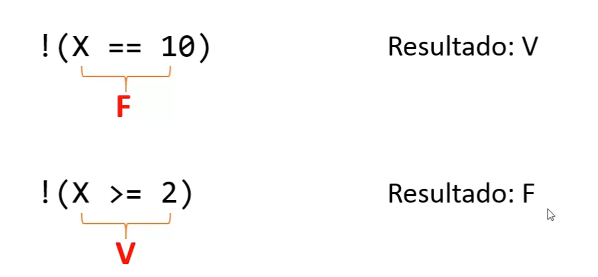
**Não**

* Possuir renda maior que $ 3000,00

*O operador "NÃO" inverte a condição*

**Exemplos de expressões lógicas**

(*suponha x igual a 5*)

Tabela verdade do operador "**NÃO**"

**A !A**

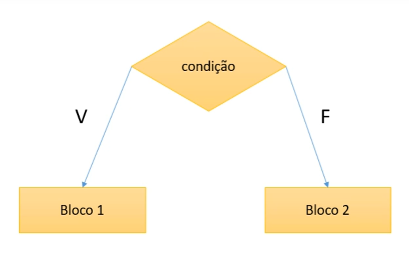
F V

V F

**Estrutura condicional**

**Conceito**

Estrutura condicional: É uma **estrutura de controle** que permite definir que um certo **bloco de comandos** somente será executado dependendo de uma **condição**.



**Sintaxe da estrutura condicional**

**Simples**:

if (<condição>) {

<comando 1>

<comando 2>

}

**Exemplo**:

public class Main {

public static void main(String[] args) {

int x = 5;

System.out.println("Bom dia");

if (x < 0){

System.out.println("Boa tarde");

}

System.out.println("Boa noite");

}

}

**Resultado:**

*Bom dia*

*Boa noite*

**REGRA:**

**V**: *executa o bloco de comandos*

**F**: *pula o bloco de comandos*

*Sintaxe da estrutura condicional*

**Composta**:

if (<condição>){

<comando 1>

<comando 2>

}

else {

<comando 3>

<comando 4>

}

**REGRA**:

**V**: *executa somente o bloco do if*

**F**: *executa somente o bloco do eles*

**Exemplo:**

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner keyboard = new Scanner(System.in);

int hora;

System.out.println("Quantas horas?");

hora = keyboard.nextInt();

if (hora < 12){

System.out.println("Bom dia");

}

else {

System.out.println("Boa tarde");

}

keyboard.close();

}

}

**Resultado:**

10

Bom dia

**Resultado:**

15

Boa tarde

E se tiver mais de duas possibilidades:

horas < 12 Bom dia

12 <= horas < 18 Boa tarde

horas >= 18 Boa noite

**Encadeamento de estruturas condicionais**

**if** (condição 1) {

comando 1

comando 2

}

**else** {

**if** (condição 2) {

comando 3

comando 4

}

**else** {

comando 5

comando 6

}

}

**Exemplo:**

import java.util.Scanner;

public class Main {

public static void main(String[] args) {

Scanner keyboard = new Scanner(System.in);

int hora;

System.out.println("Quantas horas?");

hora = keyboard.nextInt();

if (hora < 12){

System.out.println("Bom dia");

}

else if (hora < 18) {

System.out.println("Boa tarde");

}

else {

System.out.println("Boa noite");

}

keyboard.close();

}

}

**Encademaneto de estruturas condicionais**

if (condição 1) {

comando 1

comando 2

}

else if (condição 2) {

comando 3

comando 4

}

else if (condição 2) {

comando 5

comando 6

}

else {

comando 7

comando 8

}

**Sintaxe opcional: operadores de atribuição cumulativa**

Problema exemplo

Uma operadora de telefonia cobra R$ 50.00 por um plano básico que dá direito a 100 minutos de telefone. Cada minuto que exceder a franquia de 100 minutos custa R 2.00. Fazer um programa para ler a quantidade de minutos que uma pessoa consumiu, daí mostrar o valor a ser pago.

Entrada:

22

Saída:

Valor a pagar: R$ 50.00

Entrada:

103

Saída:

Valor a pagar: R$ 56.00

**Operadores de atribuição cumulativa:**

a += b; a = a + b;

a -= b; a = a – b;

a \* = b; a = a \* b;

a /= b; a = a / b;

a %= b; a = a % b;

**Sintaxe opcional: estrutura switch-case**

Quando se tem várias opções de fluxo a serem tratadas com base no valor de uma variável, ao invés de várias estruturas if-else encadeadas, alguns preferem utilizar a estrutura switch-case.

Problema exemplo

Fazer um programa para ler um valor inteiro de 1 a 7 representado um dia da semana (sendo 1 = domingo, 2 = segunda, e assim por diante).

Escrever na tela o dia da semana correspondente, conforme exemplos.

**Entrada Saída**

1 Dia da semana:domingo

4 Dia da semana:quarta

9 Dia da semana:valor inválido

**Sintaxe**:

*switch ( expressão ) {*

*case valor1:*

*comando1*

*comando2*

*break;*

*case valor2:*

*comando3*

*comando4*

*break;*

*default:*

*comando5*

*comando6*

*break;*

*}*

**Expressão condicional ternária**

Estrutura opcional ao if-else quando se deseja decidir um VALOR com base em uma condição.

**Sintaxe:**

( condição ) ? valor\_se\_verdadeiro : valor\_se\_falso

**Exemplos:**

( 2 > 4 ) ? 50 : 80 -> 80

( 10 != 3 ) ? “Maria” : “Alex” -> “Maria”

**Demonstração:**

double preco = 34.5;

double desconto;

if (preco < 20.0) {

desconto = preco \* 0.1;

} else {

desconto = preco \* 0.05;

}

**Condicional ternária (Código):**

double preco = 34.5;

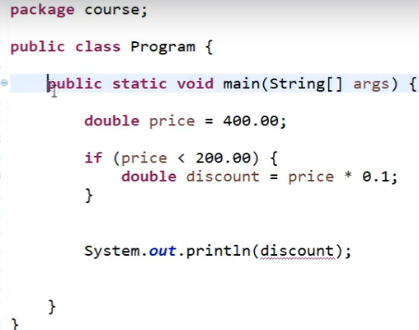
double desconto = (preco < 20.0) ? preco \* 0.1 : preco \* 0.05;

**Escopo e inicialização**

* Escopo de uma variável: é a região do programa onde a variável é válida, ou seja, onde ela pode ser referenciada;
* Uma variável não pode ser usada se não for iniciada;
* Escopo de métodos no Capítulo 5.

Inicialização – Uma variável deve ser inicializada de alguma forma, antes de apresentar a saída.   
Deve conter algum valor.

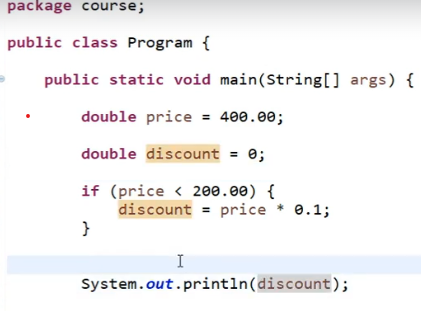
Escopo – A variável não pode ficar fora do escopo. Exemplo:



discount está fora do escopo, no caso a declaração feita da variável está dentro do if, não sendo possível localizar o resultado fora dele.

Mesma coisa se a variável discount estivesse sido declarada antes, porém, só tivesse um valor atribuído dentro do if. E ao tentar apresentar no println fora da estrutura condicional, o programa iria apresentar erro. Pois o valor só foi atribuído dentro do if e caso não fosse verdadeiro, não iria processar o resultado do if. Nesse caso, precisamos atribuir algum valor antes, na variável discount para poder obter o resultado (ou utilizar um else com valor zero na variável)

Exemplo:



**Como utilizar o Debug no Eclipse (execução passo a passo)**

**Para marcar uma linha de breakpoint:**

* Run -> Toggle Breakpoint

**Para iniciar o debug:**

* Botão direito na classe -> Debug as -> Java Application

**Para executar uma linha:**

* F6

**Para interromper o debug:**

* **Botão stop**

**While**

Estrutura repetitiva “enquanto” é uma **estrutura de controle** que **repete** um bloco de comandos enquanto uma **condição** for verdadeira. **Quando usar**: quando **não** se sabe previamente a quantidade de repetições que será realizada.

Problema exemplo:

Fazer um programa que lê números inteiros até que um zero seja lido. Ao final mostra a soma dos números lidos.

Entrada: Saída:

5 11

2

4

0

Sintaxe / regra:

While ( condição ) {

comando 1

comando 2

}

**Regra**:

**V**: executa e volta

**F**: pula fora

**Estrutura repetitiva “para” (for)**

É uma estrutura de controle que repete um bloco de comandos para um certo intervalo de valores.

Quando usar: quando se sabe previamente a quantidade de repetições, ou o intervalo de valores.

Por exemplo:

Fazer um programa que lê um valor inteiro N e depois N números inteiros.

Ao final, mostra a soma dos N números lidos:

Entrada: Saída:

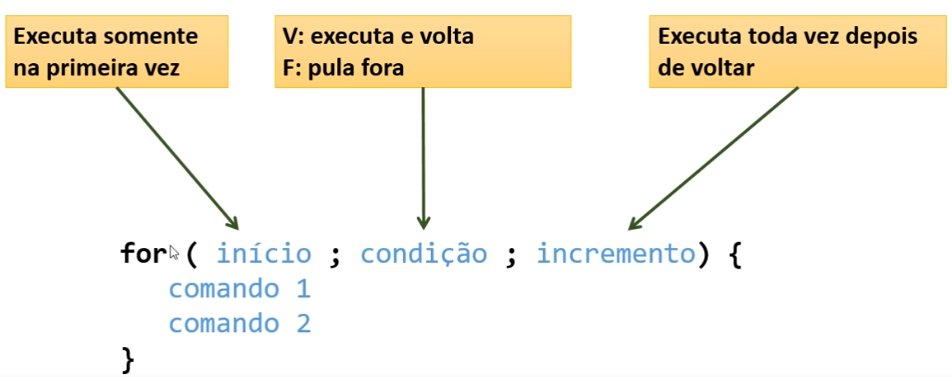
3 11

5

2

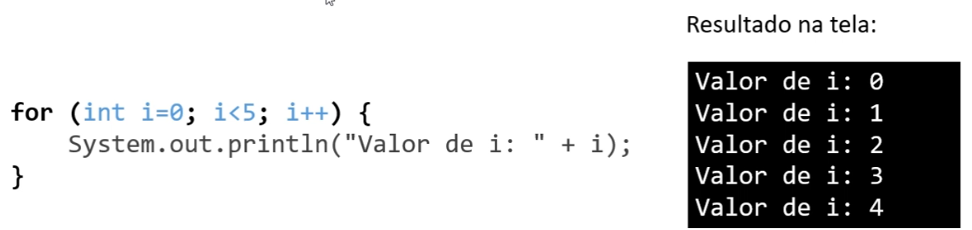
4

Sintaxe/regra:



Importante

Perceba que a estrutura “para” é ótima para se fazer uma repetição baseada em uma CONTAGEM:



Estrutura repetitiva “faça-enquanto” – Do While

**Sintaxe/regra**

***Do*** *{*

*Comando1*

*Comando2*

*}* ***while*** *(condição);*

**Regra:**

V: volta

F: pula fora

**Problema exemplo:**

Fazer um programa para ler uma temperatura em Celsius e mostrar o equivalente em Fahrenheit. Perguntar se o usuário deseja repetir (s/n). Caso o usuário digite “s”, repetir o programa.

Fórmula: F = 9C/5 + 32

**Exemplo:**

*Digite a temperatura em Celsius: 30.0*

*Equivalente em Fahrenheit: 86.0*

**Deseja repetir (s/n)? s**

*Digite a temperatura em Celsius: 21.0*

*Equivalente em Fahrenheit: 69.8*

**Deseja repetir (s/n)? s**

*Digite a temperatura em Celsius: -10.5*

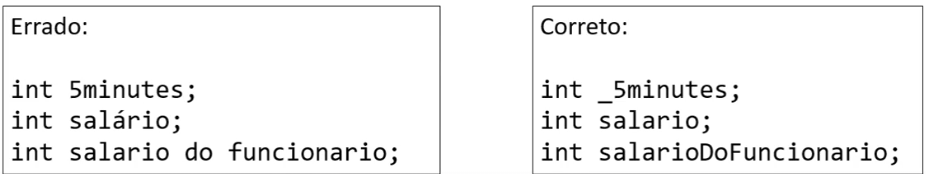
*Equivalente em Fahrenheit: 13.1*

**Deseja repetir (s/n)? n**

**Restrições e convenções para nomes**

Nomes de variáveis:

* Não pode começar com dígito: use uma letra ou \_
* Não usar acentos ou til
* Não pode ter espaço em branco
* Sugestão: use nomes que tenham um significado



Convenções

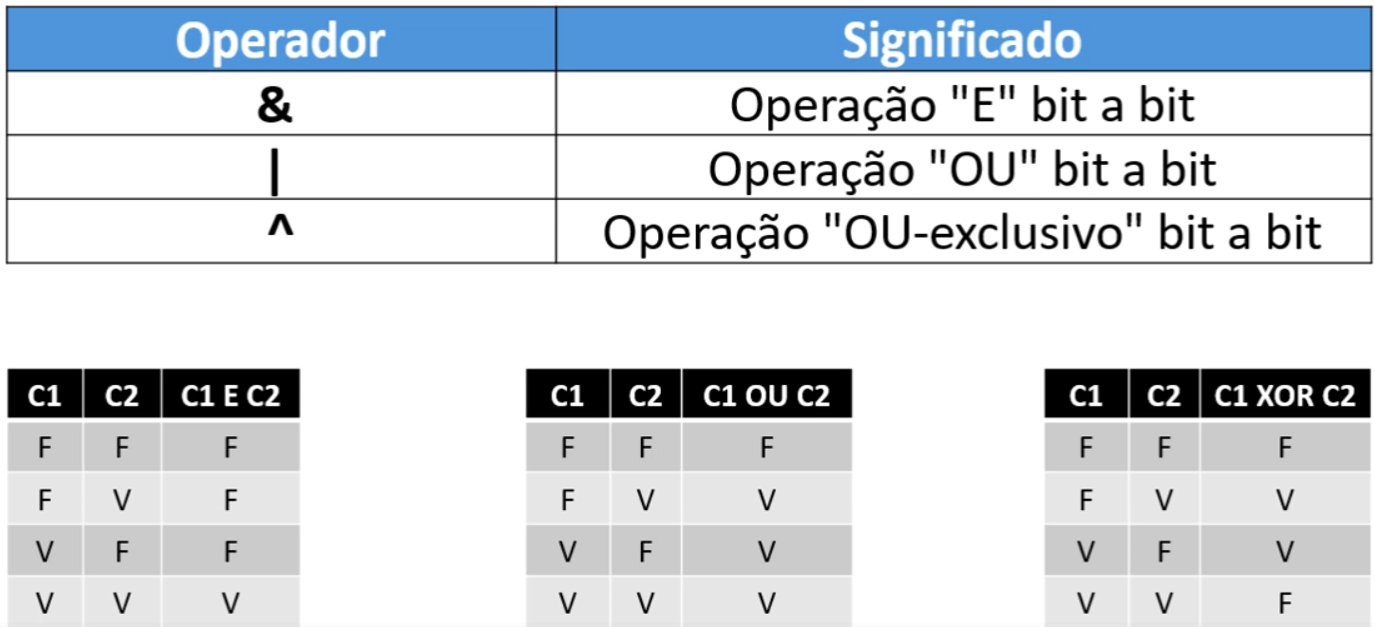
Camel Case: lastName

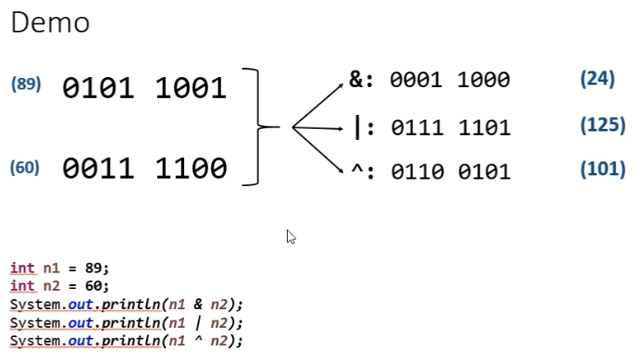
* pacotes;
* atributos;
* métodos
* variáveis e parâmetros

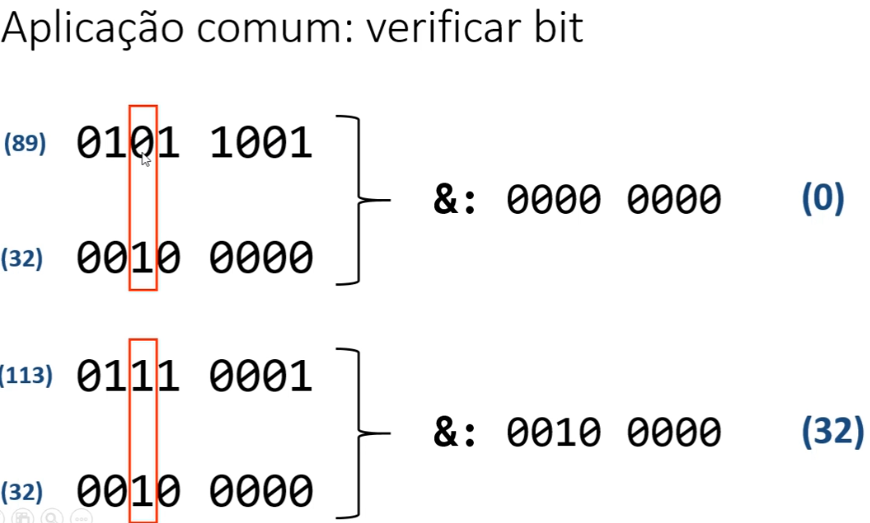
Pascal Case: ProductService

* classes

**Operadores bitwise**







**Funções interessantes para String**

Checklist

* **Formatar**: toLowerCase(),toUpperCase(),trim()
* **Recortar**: substring(inicio),substring(inicio, fim)
* **Substituir**: Replace(char, char), Replace(string, string)
* **Buscar**: IndexOf, LastIndexOf
* str.Split(“ “)

**Operação split**

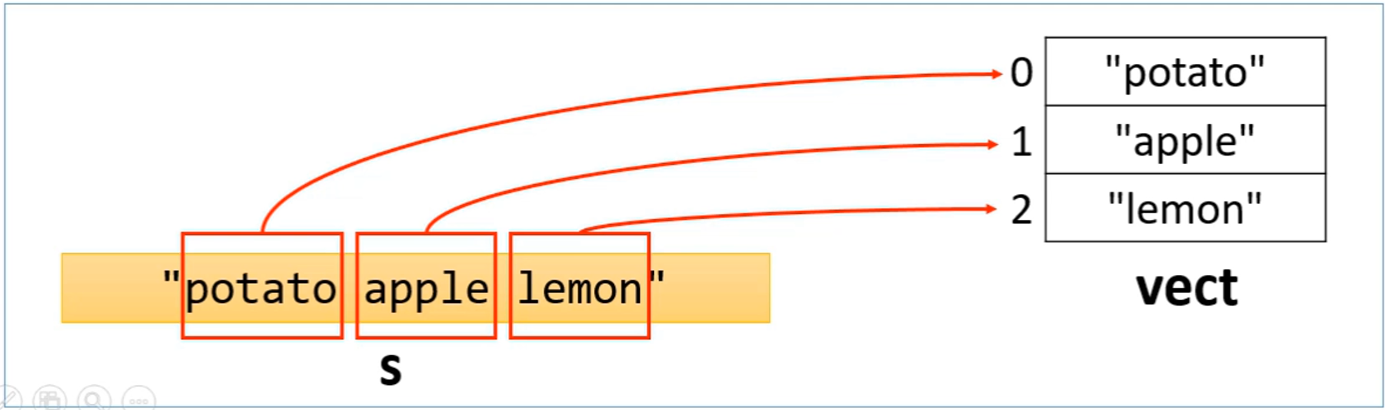
*String s = “potato apple lemon”;*

*String [] vect = s.split(“ “);*

*String word1 = vect[0];*

*String word2 = vect[1];*

*String word3 = vect[2];*



**Funções**

Representam um processamento que possui um significado

* Math.sqrt(double)
* System.out.println(string)

Principais vantagens: modularização, delegação e reaproveitamento

Dados de entrada e saída

* Funções podem receber dados de entrada (parâmetros ou argumentos)
* Funções podem ou não retornar uma saída

Em orientação a objetos, funções em classes recebem o nome de “métodos”

**Problema exemplo:**

Fazer um programa para ler três números inteiros e mostrar na tela o maior deles.

Exemplo:

*Enter three numbers:*

**5**

**8**

**3**

*Higher = 8*

Vamos criar uma função logo após a função principal main.

Usaremos public static, public para publico (ficar disponível para outras classes) e static para que essa função possa ser chamada independente de criar um objeto.

Programa criado:

package problema01Aula61;

import java.util.Scanner;

public class Application {

public static void main(String[] args) {

Scanner keyboard = new Scanner(System.***in***);

System.***out***.println("Enter three numbers: ");

int a = keyboard.nextInt();

int b = keyboard.nextInt();

int c = keyboard.nextInt();

int higher = *max*(a, b, c);

*showResult*(higher);

keyboard.close();

}

public static int max(int x, int y, int z) {

int aux;

if (x > y && x > z) {

aux = x;

} else if (y > z) {

aux = y;

} else {

aux = z;

}

return aux;

}

public static void showResult(int value) {

System.***out***.println("Higher = " + value);

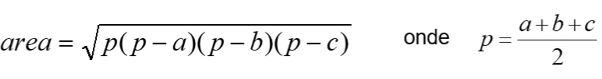
}

}

**Resolvendo um problema sem orientação a objetos**

Fazer um programa para ler as medida dos lados de dois triângulos X e Y (suponha medidas válidas). Em seguida, mostrar o valor das áreas dos dois triângulos e dizer qual dos dois triângulos possui a maior área.

A fórmula para calcular a área de um triângulo a partir das medidas de seus lados a, b e c é a seguinte (fórmula de Heron):



Exemplo:

Enter the measures of triangule X:

3.00

4.00

5.00

Enter the measures of triangle Y:

7.50

4.50

4.02

Triangle X area: 6.0000

Triangle Y area: 7.5638

Larger area: Y

**Criando uma classe com três atributos para representar melhor o triângulo**

Discussão

Triângulo é uma entidade com três atributos: a, b, c.

Estamos usando três variáveis distintas para representar cada triângulo:

double aX, bX, cX, aY, bY, cY;

Para melhorar isso, vamos usar uma CLASSE para representar um triângulo.

Memória:



**Classe**

É um tipo estrutura que pode conter (membros):

* Atributos (dados / campos)
* Métodos (funções /operações)

A classe também pode prover muitos outros recursos, tais como:

* Construtores
* Sobrecarga
* Encapsulamento
* Herança
* Polimorfismo

Exemplos:

* Entidades: produto, Cliente, Triangulo
* Serviços: ProdutoService, ClienteService, EmailService, StorageService
* Controladores: ProdutoController, ClienteController
* Utilitários: Calculadora, Compactador
* Outros (views, repositórios, gerenciadores, etc.)

package entities;

public class Triangle **{**

public double a;

public double b;

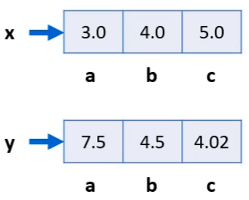
public double c;

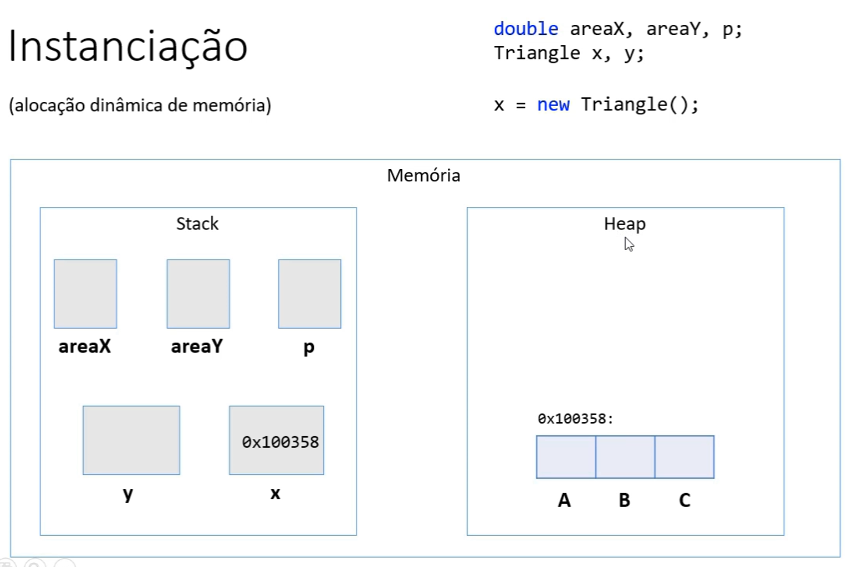
**}**

Triangle x, y;

x = new Triangle();

y = new Triangle();





**Criando um método para obtermos os benefícios de reaproveitamento e delegação**

Discussão

Com o uso de CLASSE, agora nós temos uma variável composta do tipo “Triangle” para representar cada triângulo:

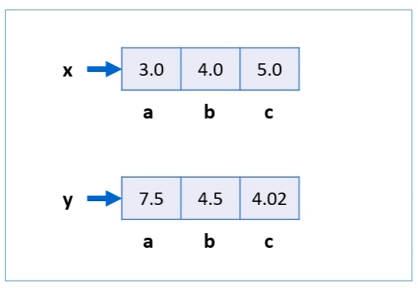
Triangle x, y;

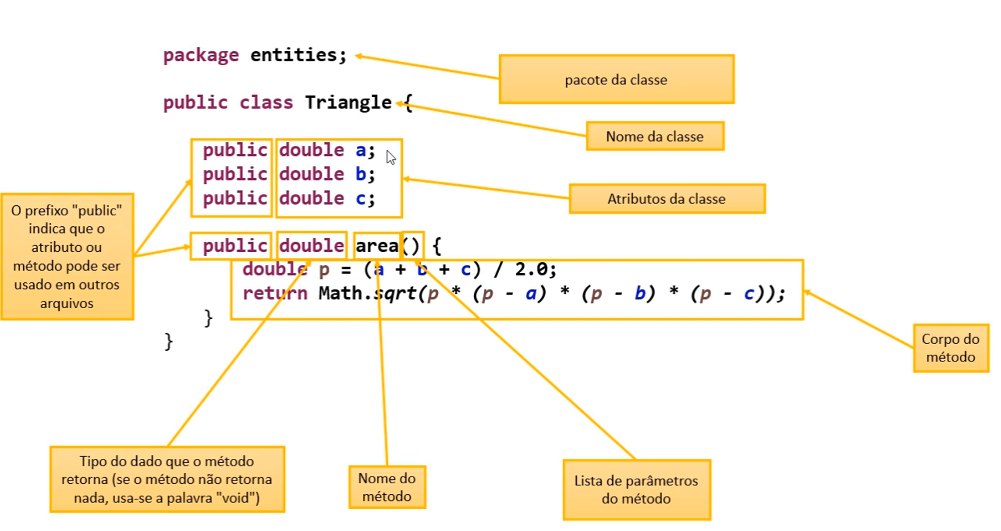
x = new Triangle();

y = new Triangle();

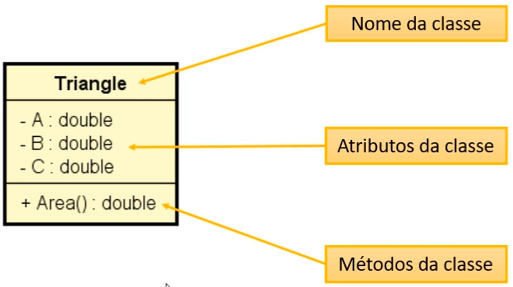
Agora vamos melhorar nossa CLASSe acrescentando nela um MÉTODO para calcular a área.

**Memória:**





**Projeto de classe (UML)**



Benefícios de se calcular a área de um triângulo por meio de um **MÉTODO** dentro da **CLASSE**:

1. **Reaproveitamento de código:** nós eliminamos o código repetido (cálculo das áreas dos triângulos x e y) no programa principal.
2. **Delegação de responsabilidades:** quem deve ser responsável por saber como calcular a área não deve estar em outro lugar.

**Resolvendo um segundo problema exemplo.**

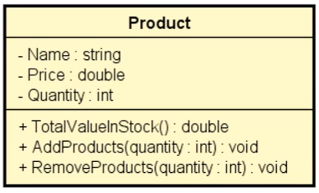
Fazer um programa para ler os dados de um produto em estoque (nome, preço e quantidade no estoque). Em seguida:

Mostrar os dados do produto (nome, preço, quantidade no estoque, valor total no estoque)

Realizar uma entrada no estoque e mostrar novamente os dados do produto.

Realizar uma saída no estoque e mostrar novamente os dados do produto.

Para resolver este problema, você deve criar uma CLASSE conforme projeto ao lado:



Exemplo:

Enter product data:

Name: TV

Price: 900.00

Quantity in stock: 10

Product data: TV, $ 900.00, 10 units, Total: $ 9000.00

Enter the number of products to be added in stock: 5

Update data: TV, $ 900.00, 15 units, Total: $ 13500.00

Enter the number of products to be removed from stock: 3

Updated data: TV, $ 900.00, 12 units, Total: $ 10800.00

**Object e toString**

**Discussão**

Toda classe em Java é uma subclasse da classe Object.

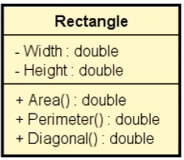
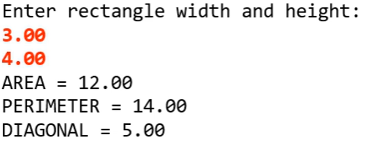
Object possui os seguintes métodos:

* getClass - retorna o tipo do objeto;
* equals - compara se o objeto é igual a outro;
* hashCode - retorna um código hash do objeto;
* toString - converte o objeto para string

Aula 70

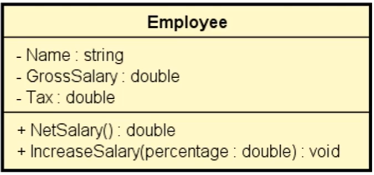
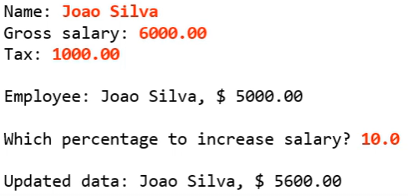
Exercício 1

Fazer um programa para ler os valores da largura e altura de um retângulo. Em seguida, mostrar na tela o valor de sua área, perímetro e diagonal. Usar uma classe como mostrado no projeto ao lado.

Exercício 2

Fazer um programa para ler os dados de um funcionário (nome, salário bruto e imposto). Em seguida, mostrar os dados do funcionário (nome e salário líquido). Em seguida, aumentar o salário do funcionário com base em uma porcentagem dada (somente o salário bruto é afetado pela porcentagem) e mostrar novamente os dados do funcionário. Use a classe projetada abaixo.

Exercício 3

Fazer um programa para ler o nome de um aluno e as três notas que ele obteve nos três trimestres do ano (primeiro trimestre vale 30 e o segundo e terceiro valem 35 cada). Ao final, mostrar qual a nota final do aluno no ano. Dizer também se o aluno está aprovado (PASS) ou não (FAILED) e, em caso negativo, quantos pontos faltam para o aluno obter o mínimo para ser aprovado (que é 60% da nota). Você deve criar uma classe Student para resolver este problema.

Exemplos:

**Entrada: Saída:**

Alex Green FINAL GRADE = 90.00

27.00 PASS

31.00

32.00

**Entrada: Saída:**

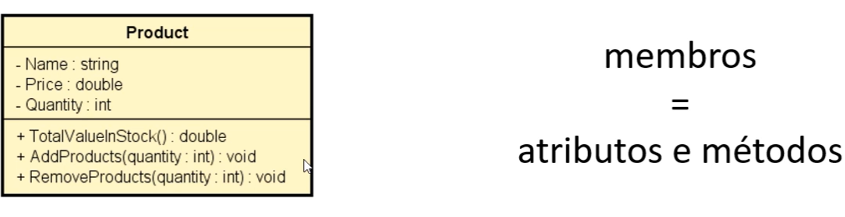
Alex Green FINAL GRADE = 52.00

17.00 FAILED

20.00 MISSING 8.00 POINTS

15.00

**Membros estáticos**



Também chamados membros de classe

Em oposição a membros e instância

São membros que fazem sentido independentemente de objetos. Não precisam de objeto para serem chamados. São chamados a partir do próprio nome da classe.

Aplicações comuns:

Classes utilitárias

Declaração de constantes

Uma classe que possui somente membros estáticos, pode ser uma classe estática também. Esta classe não poderá ser instanciada.

Problema exemplo

Fazer um programa para ler um valor numérico qualquer, e daí mostrar quanto seria o valor de uma circunferência e do volume de uma esfera para um raio daquele valor. Informar também o valor de PI com duas casas decimais.

Exemplo:

Enter radius: 3.0

Circumference: 18.84

Volume:113.04

PI value: 3.14

Checklist

**Versão 1**: métodos na própria classe do programa

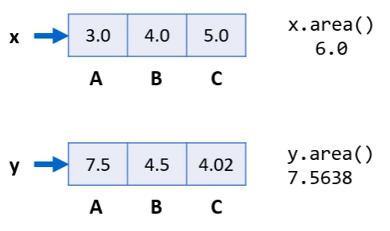
* Nota: dentro de um método estático você não pode chamar membros de instância da mesma classe.

**Versão 2**: classe Calculator com membros de instância

**Versão 3**: classe Calculator com método estático

**Discussão**

* No problema dos triângulos, cada triângulo possui sua área.
* Area() é uma operação concernente ao objeto: cada triângulo possui sua área.



* Já no caso da calculadora, os valores dos cálculos não mudam para calculadoras diferentes, ou seja, são cálculos estáticos. O valor de Pi também é estático.

Calculator calc1 = **new Calculator();**

Calculator calc2 = **new Calculator();**

**Exercício de fixação**

Faça um programa para ler a cotação do dólar, e depois um valor em dólares a ser comprado por uma pessoa em reais. Informar quantos reais a pessoa vai pagar pelos dólares, considerando ainda que a pessoa terá que pagar 6% de IOF sobre o valor em dólar. Criar uma classe CurrencyConverter para ser responsável pelos cálculos.

**Exemplo**:

Wha tis the dollar price? 3.10

How many dollars will be bought? 200.00

Amiunt to be paid in reais = 657.20

**Construtores**

É uma operação especial da classe, que executa no momento da instanciação do objeto.

**Usos comuns:**

* Iniciar valores dos atributos
* Permitir ou obrigar que o objeto receba dados / dependências no momento de sua instanciação (injeção de dependência)

Se um construtor customizado não for especificado, a classe disponibiliza o construtor padrão:

*Product p = new Product();*

É possível especificar mais de um construtor na mesma classe (sobrecarga)

Problema exemplo:

Enter product data:

Name: TV

Price: 900.00

Quantity in stock: 10

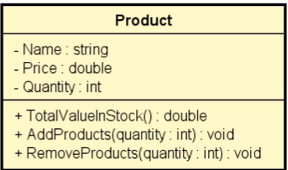
Product data: TV, $ 900.00, 10 units, Total: $ 9000.00

Enter the number of products to be added in stock: 5

Updated data: TV, $ 900.00, 15 units, Total: 13500.00

Enter the number of products to be removed from stock: 3

Updated data: TV, $ 900.00, 12 units, Total: $ 10800.00



Proposta de melhoria:

Quando executamos o comando abaixo, instanciamos um produto “product” com seus atributos “vazios”:

product = new Product();



Entretanto, faz sentido um produto que não tem nome? Faz sentido um produto que não tem preço?

Com o intuito de evitar a existência de produtos sem nome e sem preço, é possível fazer com que seja “obrigatória” a iniciação desses valores?

**Palavra this**

É uma referência para o próprio objeto

Usos comuns:

* Diferenciar atributos de variáveis locais
* Passa o próprio objeto como argumento na chamada de um método ou construtor

Diferenciar atributos de variáveis locais





Passa o próprio objeto como argumento na chamada de um método ou construtor



**Sobrecarga**

É um recurso que uma classe possui de oferecer mais de uma operação com o mesmo nome, porém com diferentes listas de parâmetros.

Proposta de melhoria

Vamos criar um construtor opcional, o qual recebe apenas nome e preço do produto. A quantidade em estoque deste novo produto, por padrão, deverá então ser iniciada com o valor zero.

Nota: é possível também incluir um construtor padrão.

**Encapsulamento**

É um princípio que consiste em esconder detalhes de implementação de uma classe, expondo apenas operações seguras e que mantenham os objetos em um estado consistente.

Regra de ouro: o objeto deve sempre estar em um estado consistente, e a própria classe deve garantir isso.

**Regra geral básica**

Um objeto NÃO deve expor nenhum atributo (modificador de acesso private)

Os atributos devem ser acessados por meio de métodos get e set

Padrão JavaBeans: <https://en.wikipedia.org/wiki/JavaBeans>

Padrão para implementação de getters e setters:

private String nome;

private double price;

public String getName(){

return name;

}

public void setName(String name){

this.name = name;

}

public double getPrice(){

return price;

}

public void setPrice(double price){

this.price = price;

}

**Modificadores de acesso**

* <https://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html>
* **private**: o membro só pode ser acessado na **própria classe**
* (nada): o membro só pode ser acessado nas classes do **mesmo pacote**
* **protected**: o membro só pode ser acessado no **mesmo pacote**, bem como em **subclasses de pacotes diferentes**
* **public**: o membro é acessado por todas classes (ao menos que ele resida em um módulo diferente que não exporte o pacote onde ele está)

**Aula 82 – Exercício**

Em um banco, para se cadastrar uma conta bancária, é necessário informar o número da conta, o nome do titular da conta, e o valor de depósito inicial que o titular depositou ao abrir a conta. Este valor de depósito inicial, entretanto, é opcional, ou seja: se o titular não tiver dinheiro a depositar no momento de abrir sua conta, o depósito inicial não será feito e o saldo inicial da conta será, naturalmente, zero.

Importante: uma vez que uma conta bancária foi aberta, o número da conta nunca poderá ser alterado. Já o nome do titular pode ser alterado (pois uma pessoa pode mudar de nome por ocasião de casamento, por exemplo).

Por fim, o saldo da conta não pode ser alterado livremente. É preciso haver um mecanismo para proteger isso. O saldo só aumenta por meio de depósitos, e só diminui por meio de saques.

Para cada saque realizado, o banco cobra uma taxa de $ 5.00. Nota: a conta pode ficar com o saldo negativo se o saldo não for suficiente para realizar o saque e/ou pagar a taxa.

Você deve fazer um programa que realize o cadastro de uma conta, dando opção para que seja ou não informado o valor de depósito inicial. Em seguida, realizar um depósito e depois um saque, sempre mostrando os dados da conta após cada operação.

**Exemplo 01:**

Enter account number: 8532

Enter account holder: Alex Green

Is there na initial deposit (y/n)? y

Enter initial deposit value: 500.00

Account data:

Account 8532, Holder: Alex Green, Balance: % 500.00

Enter a deposit value: 200.00

Updated account data:

Account 8532, Holder: Alex Green, Balance: $ 700.00

Enter a withdraw value: 300.00

Updated account data:

Account 8532, Holder: Alex Green, Balance: $ 385.00

**Exemplo 02:**

Enter account number: 7801

Enter account holder: Maria Brown

Is there na initial deposit (y/n)? n

Account data:

Account 7801, Holder: Maria Brown, Balance: $ 0.00

Enter a deposit value: 200.00

Updated account data:

Account 7801, Holder: Maria Brown, Balance: $ 200.00

Enter a withdraw value: 198.00

Updated account data:

Account 7801, Holder: maria Brown, Balance: $ -3.00

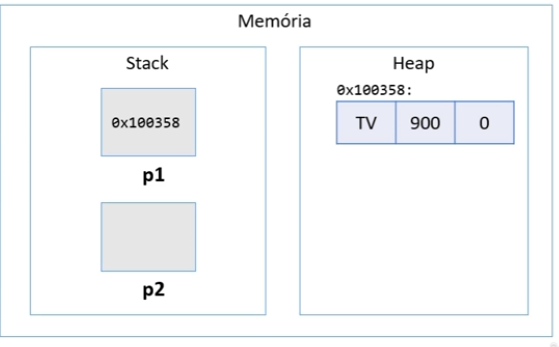
**Tipos referências vs tipos valor**

Classes são tipos referência

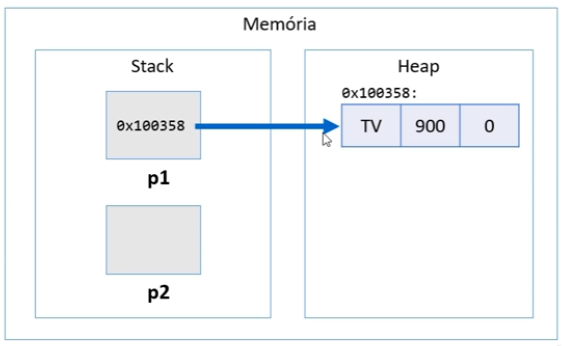
Variáveis cujo tipo são classes não devem ser entendidas como caixas, mas sim “tentáculos” (ponteiros) para caixas

Product p1, p2;

p1 = new Product(“TV”, 900.00, 0);

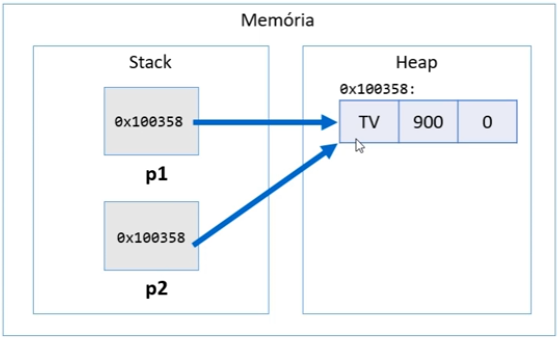


Ponteiro ou “tentáculo”:



p2 = p1;

p2 passa a apontar para onde p1 aponta:





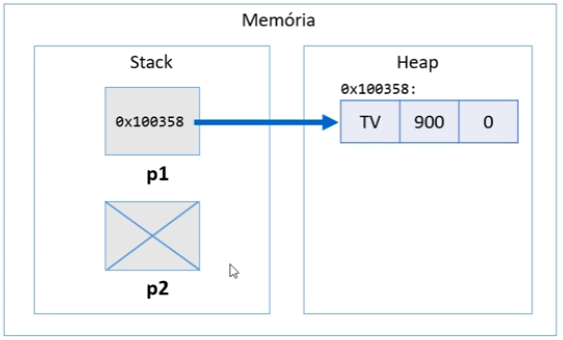
Valor “null”

Tipos referência aceitam o valor “null”, que indica que a variável aponta pra ninguém.

Product p1, p2;

p1 = new Product(“TV”, 900.00, 0);

p2 = null;



**Tipos primitivos são tipos valor**

Em Java, tipos primitivos são tipos valor. Tipos valor são CAIXAS e não ponteiros.

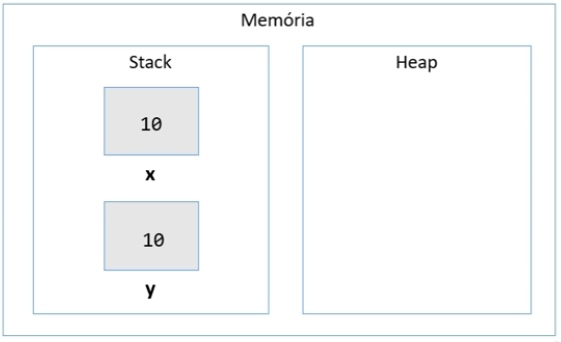
double x, y;

Double é um tipo primitivo em Java.

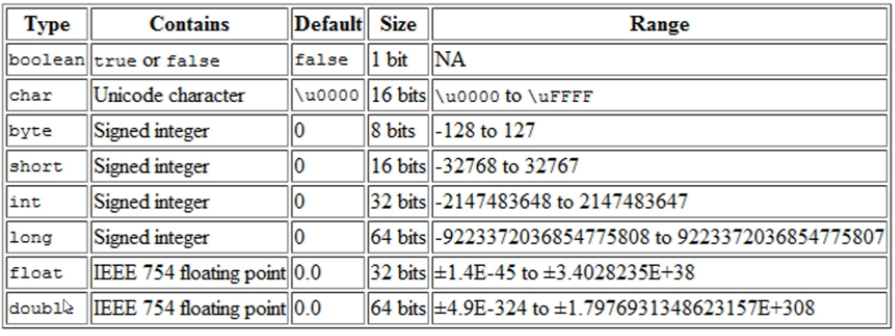
x = 10;

y = x;

y = x então y recebe uma CÓPIA de x



**Tipos primitivos em Java:**



**Tipos primitivos e inicialização**

Demo:

int p;

System.out.println(p); // erro: variável não iniciada

p = 10;

System.out.println(p);

**Valores padrão**

Quando alocamos (new) qualquer tipo estruturado (classe ou array), são atribuídos valores padrão aos seus elementos

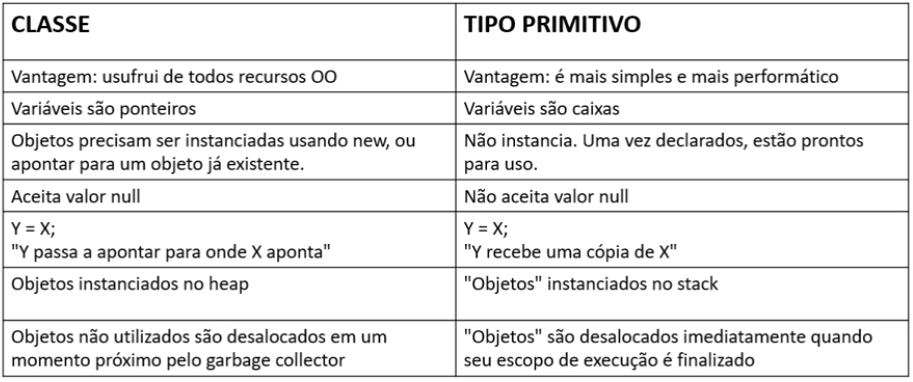
* números: 0
* boolean: false
* char: caractere código 0
* objeto: null

Product p = new Product();

Memória:



**Tipos referência vs tipos valor**



**Deslocação de memória – garbage collector e escopo local**

Garbage collector

* É um processo que automotiza o gerenciamento de memória de um programa em execução
* O garbage collector monitora os objetos alocados dinamicamente pelo programa (no heap), desalocando aqueles que não estão mais sendo utilizados.

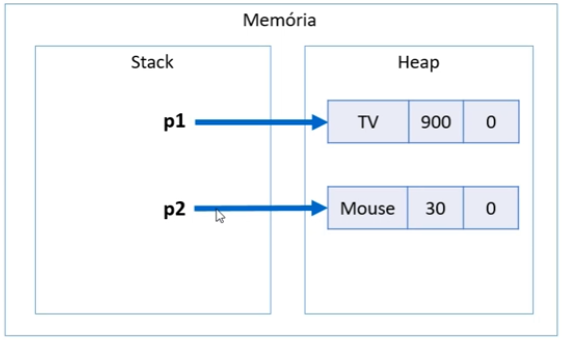
Desalocação por garbage collector

Product p1, p2;

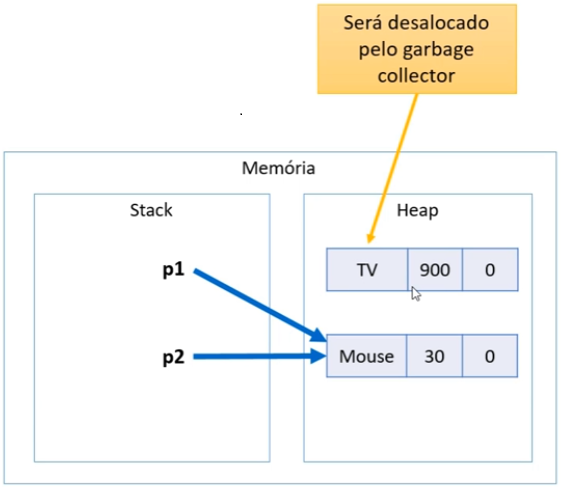
p1 = new Product(“TV”, 900.00, 0);

p2 = new Product(“Mouse”, 30.00, 0);

**Memória:**



**Se eu fizer p1 = p2:**



Desalocação por escopo

void method1(){

int x = 10;

if (x > 0) {

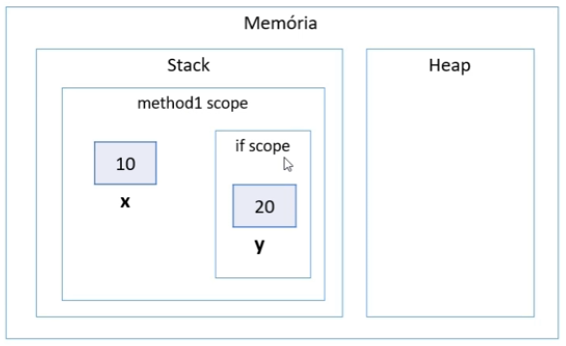
int y = 20;

}

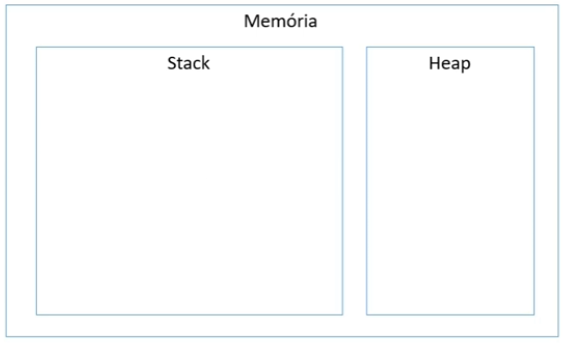
System.out.println(x);

}

Dentro do escopo do if:

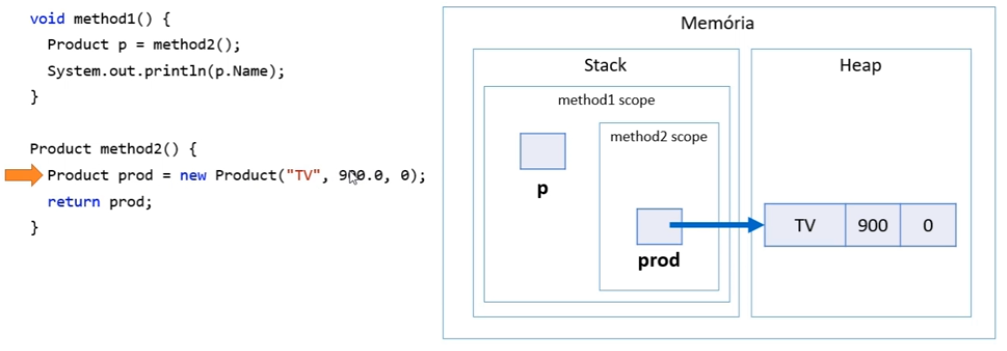


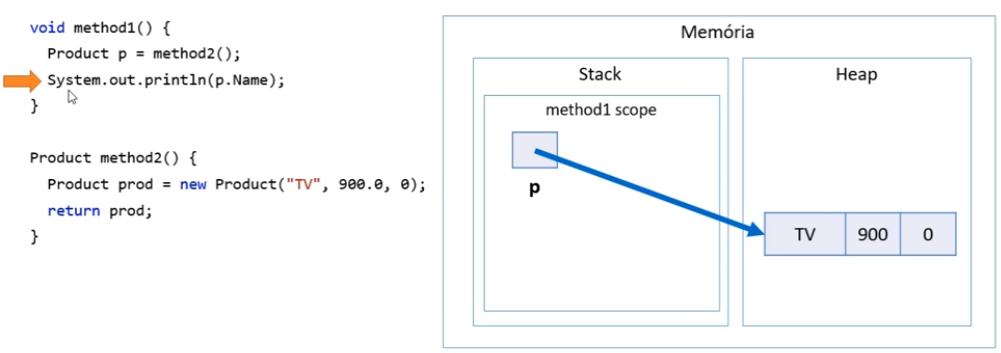
Acabando o método:



**Outro exemplo:**

**Sub escopo (pilha) – O method1 chama o method2 e assim é feto um sub escopo como na imagem abaixo:**





**Resumo**:

* Objetos alocados dinamicamente, quando não possuem mais referência para eles, serão desalocados pelo garbage collector
* Variáveis locais são desalocadas imediatamente assim que seu escopo local sai de execução

**Vetores**

Em programação, “vetor” é o nome dado a arranjos unidimensionais

Arranjo (array) é uma estrutura de dados:

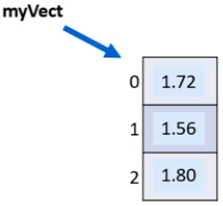
* Homogênea (dados do mesmo tipo)
* Ordenada (elementos acessados por meio de posições)
* Alocada de uma vez só, em um bloco contíguo de memória

Vantagens:

* Acesso imediato aos elementos pela sua posição

Desvantagens:

* Tamanho fixo
* Dificuldade para se realizar inserções e delegações



*Problema exemplo 1*

Fazer um programa para ler um número inteiro N e a altura de N pessoas. Armazene as N alturas em um vetor. Em seguida, mostrar a altura média dessas pessoas.

*Exemplo:*

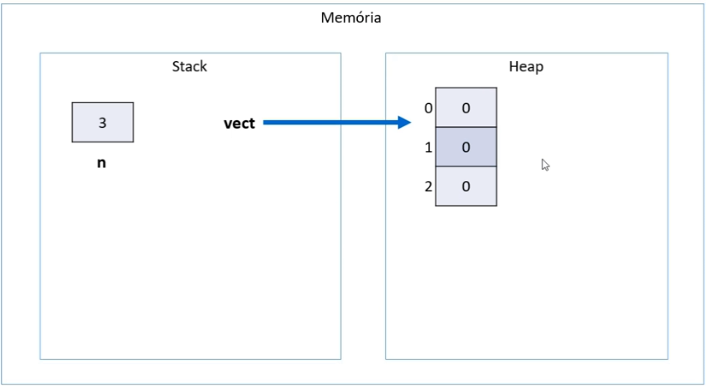
**Input: Output:**

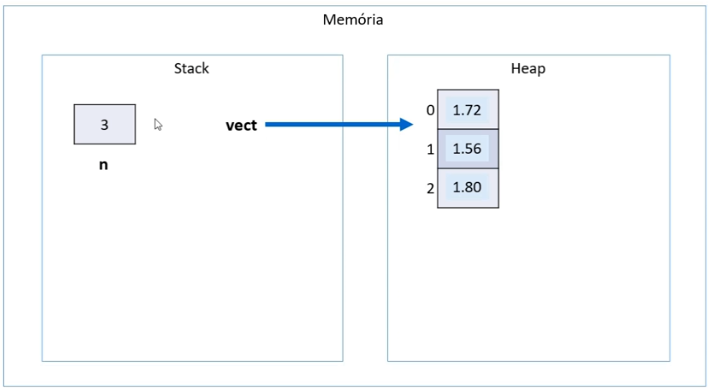
3 AVERAGE HEIGHT = 1.69

1.72

1.56

1.80





package application;

import java.util.Locale;

import java.util.Scanner;

public class Program {

public static void main(String[] args) {

Locale.setDefault(Locale.US);

Scanner keyboard = new Scanner(System.in);

int n = keyboard.nextInt();

double[] vect = new double[n];

for(int i = 0; i < n; i++) {

vect[i] = keyboard.nextDouble();

}

double sum = 0.0;

for(int i = 0; i < n; i++) {

sum += vect[i];

}

double avg = sum / n;

System.out.printf("AVERAGE HEIGHT: %.2f%n", avg);

keyboard.close();

}

}

Problema exemplo 2

Fazer um programa para ler um número inteiro N e os dados (nome e preço) de N Produtos. Armazene os N produtos em um vetor. Em seguida, mostrar o preço médio dos produtos.

Exemplo:

**Input: Output:**

3 AVERAGE PRICE = 700.00

TV

900.00

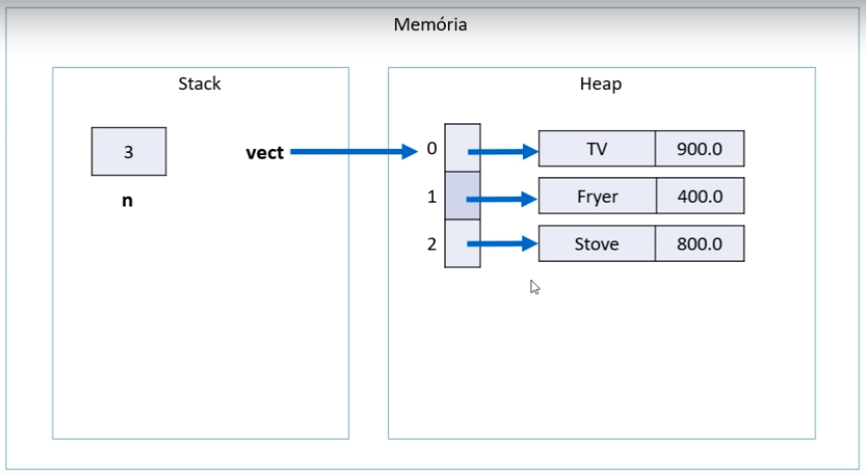
Fryer

400.00

Stove

800.00





Aula 94 Desafio sobre vetores (pensionato)

Exercício de fixação

A dona de um pensionato possui dez quartos para alugar para estudantes, sendo esses quartos identificados pelos números 0 e 9.

Fazer um programa que inicie com todos os dez quartos vazios, e depois leia uma quantidade N representando o número de estudantes que vão alugar quartos (N pode ser de 1 a 10). Em seguida, registre o aluguel dos N estudantes. Para cada registro de aluguel, informar o nome e email do estudante, bem como qual dos quartos ele escolheu (de 0 a 9). Suponha que seja escolhido um quarto vago. Ao final, seu programa deve imprimir um relatório de todas ocupações do pensionato, por ordem de quarto, conforme exemplo.

How many rooms will be rented? 3

Rent #1

Name: Maria Green

Email: maria@gmail.com

Room: 5

Rent #2

Name: Marco Antonio

Email: marco@gmail.com

Room: 1

Rent #3

Name: Alex Brown

Email: alex@gmail.com

Room: 8

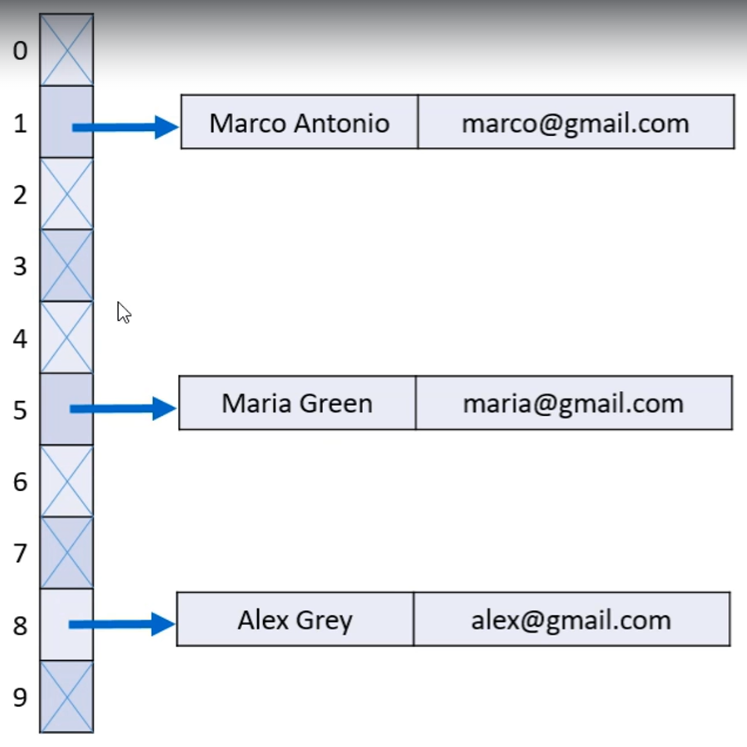
Busy rooms:

1: Marco Antonio, [marco@gmail.com](mailto:marco@gmail.com)

5: Maria Green, [maria@gmail.com](mailto:maria@gmail.com)

8: Alex Brown, [alex@gmail.com](mailto:alex@gmail.com)

Sugestão:



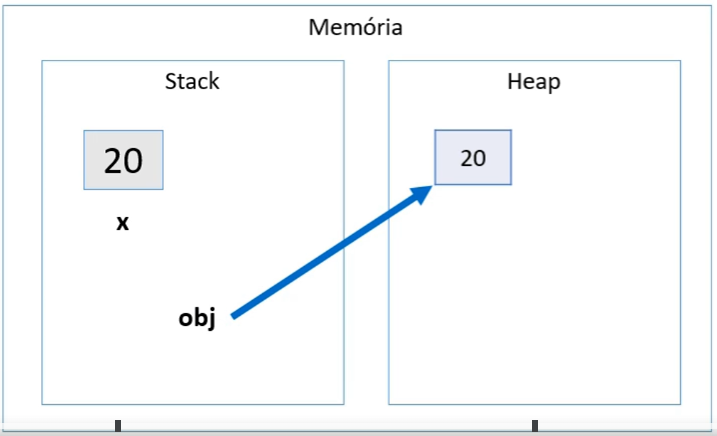
**Aula 96 – Boxing, unboxing e wrapper classes**

**Boxing**

É o processo de conversão de um objeto tipo valor para um objeto tipo referência compatível

int x = 20;

Object obj = x;



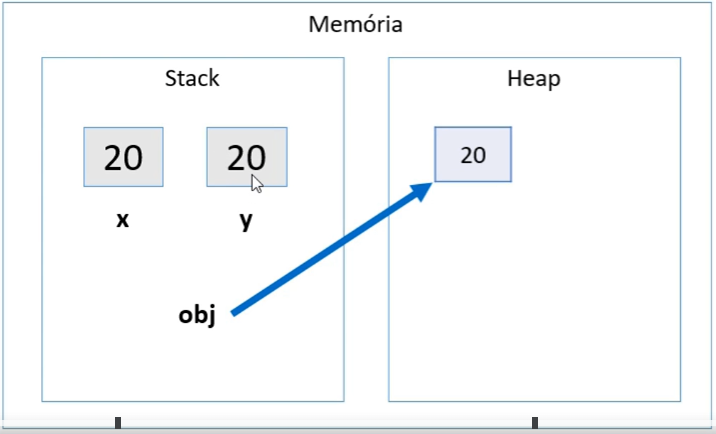
**Unboxing**

É o processo de conversão de um objeto tipo referência para um objeto tipo valor compatível

int x = 20;

Object obj = x;

int y = (int) obj;

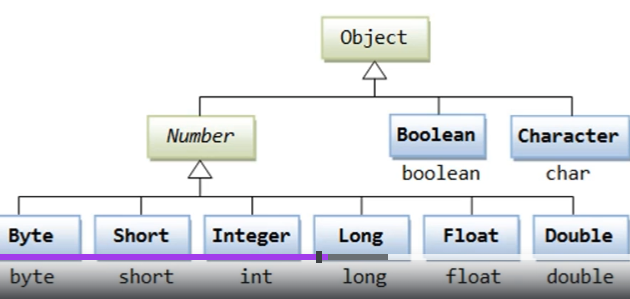


**Wrapper Classes**

* São classes equivalentes aos tipos primitivos em Java (por exemplo, int -> Integer, double -> Double).
* Boxing (conversão de tipo primitivo para Wrapper Class) e unboxing (conversão de Wrapper Class para tipo primitivo) são realizados automaticamente pela linguagem desde o Java 5.

O uso comum das Wrapper Classes ocorre principalmente em sistemas de informação, como campos de entidades e frameworks, porque:

Tipos de referência (como Wrapper Classes) podem armazenar o valor null, algo que tipos primitivos não permitem. Tipos de referência usufruem dos recursos da Orientação a Objetos, como métodos utilitários e armazenamento em coleções.



**Laço “for each”**

Sintaxe opcional e simplificada para percorrer coleções

Sintaxe:

*for (Tipo apelido* ***:*** *coleção){*

*<comando 1>*

*<comando 2>*

*}*

**Demo**

Leitura: “para cada objeto ‘obj’ contido em vect, faça:”

String[] vect = new String[] {“Maria”, “Bob”, “Alex”};

for(int i = 0; i < vect.length; i++) {

System.out.println(vect[i]);

}

for(String obj : vect) {

System.out.println(obj);

}

**Listas – Parte 1**

Referência: <https://docs.oracle.com/javase/10/docs/api/java/util/List.html>

Referência: <https://docs.oracle.com/en/java/javase/23/docs/api/index.html>

Listas

**Lista é uma estrutura de dados:**

* Homogênea (dados do mesmo tipo)
* Ordenada (elementos acessados por meio de posições)
* Inicia vazia, e seus elementos são alocados sob demanda
* Cada elemento ocupa um “nó” (ou nodo) da lista

**Tipo (interface):** List

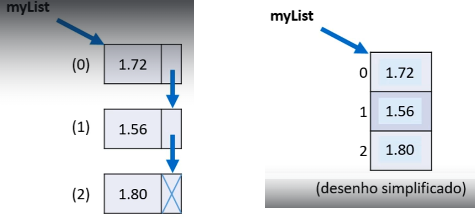
**Classes que implementam:** ArrayList, LinkedList, etc.

**Vantagens:**

* Tamanho variável
* Facilidade para se realizar inserções e deleções

**Desvantagens:**

* Acesso sequencial aos elementos.



**Aula 99 Listas – Parte 02**

Demo

Tamanho da lista: size()

Inserir elemento na lista: add(obj), add(int, obj)

Remover elementos da lista: remove(obj), remove(int), removei(Predicate)

Encontrar posição de elemento: indexOf(obj), lastIndexOf(obj)

Filtrar lista com base em predicado: List<Integer> result = list.stream().filter(x -> x > 4).collect(Collectors.toList());

Encontrar primeira ocorrência com base em predicado: Integer result = list.stream().filter(x -> x > 4).findFirst().orElse(null);

Assuntos pendentes:

Interfaces

Generics

Predicados (lambda)

Uma lista só aceita o tipo primitivo: Integer

Se você quiser usar uma com tipo inteiro, exemplo:

List<Integer> list;

Para instanciar:

List<Integer> list = new ArrayList<>();

Outro exemplo com String:

package application;

import java.util.ArrayList;

import java.util.List;

public class Program {

public static void main(String[] args) {

List<String> list = new ArrayList<>();

list.add("Bruna");

list.add("Bruno");

list.add("Kabeça");

list.add("Pepe Antonio");

list.add("Lola");

list.add("Belinha");

list.add("Fred");

list.add("Jenna");

list.add("Pandora");

list.add("Luna");

for(String x : list) {

System.***out***.println(x);

}

}

}

**Aula 100 Exercício proposto**

Fazer um programa para ler um número inteiro N e depois os dados (id, nome e salario) de N funcionários. Não deve haver repetição de id.

Em seguida, efetuar o aumento de X por cento no salário de um determinado funcionário.  
Para isso, o programa deve ler um id e o valor X. Se o id informado não existir, mostrar uma mensagem e abortar a operação. Ao final, mostrar a listagem atualizada dos funcionários, conforme exemplos.

Lembre-se de aplicar a técnica de encapsulamento para não permitir que o salário possa ser mudado livremente. Um salário só pode ser aumentado com base em uma operação de aumento por porcentagem dada.

Exemplo:

**How many employess will be registered? 3**

**Emplyoee #1:**

**Id: 333**

**Name: Maria Brown**

**Salary: 4000.00**

**Emplyoee #2:**

**Id: 536**

**Name: Anderson Grey**

**Salary: 3000.00**

**Emplyoee #3:**

**Id: 772**

**Name: Bob Green**

**Salary: 5000.00**

**Enter the employee id that will have salary increase : 536**

**Enter the percentagem: 10.0**

**List of employees:**

**333, Maria Brown, 4000.00**

**536, Anderson Grey, 3300.00**

**772, Bob Green, 5000.00**

Exemplo 02:

**How many employess will be registered? 2**

**Emplyoee #1:**

**Id: 333**

**Name: Maria Brown**

**Salary: 4000.00**

**Emplyoee #2:**

**Id: 536**

**Name: Anderson Grey**

**Salary: 3000.00**

**Enter the employee id that will have salary increase : 776**

**This is does not exist!**

**List of employees:**

**333, Maria Brown, 4000.00**

**536, Anderson Grey, 3300.00**

**Matrizes**

Em programação, “matriz” é o nome dado a arranjos bidimensionais

* Atenção: “vetor de vetores”

Arranjo (array) é uma estrutura de dados:

* Homogênea (dados do mesmo tipo)
* Ordenada (elementos acessados por meio de posições)
* Alocada de uma vez só, em um bloco contíguo de memória

Vantagens:

* Acesso imediato aos elementos pela sua posição

Desvantagens:

* Tamanho fixo
* Dificuldade para se realizar inserções e deleções

Exercício resolvido

Fazer um programa para ler um número inteiro N e uma matriz de ordem N contendo números inteiros. Em seguida, mostrar a diagonal principal e a quantidade de valores negativos da matriz.

Exemplo:

Input: Output:

3 Main diagonal:

5 -3 10 5 8 -4

15 8 2 Negative numbers = 2

7 9 -4

**Exercício de fixação**

Fazer um programa para ler dois números inteiros M e N, e depois ler uma matriz de M linhas por N colunas contendo números inteiros, podendo haver repetições. Em seguida, ler um número inteiro X que pertence à matriz. Para cada ocorrência de X, mostrar os valores à esquerda, acima, à direita e abaixo de X, quando houver conforme exemplo.

Exemplo:

3 4

10 8 15 12

21 11 23 8

14 5 13 19

8

Position 0,1:

Left: 10

Right: 15

Down: 11

Position 1,3:

Left: 23

Up: 12

Down: 19

**Revisando:**

**1. O que é um Vetor (Array) em Java?**

* Um vetor (ou array) é uma **estrutura de dados estática** usada para armazenar múltiplos valores de **mesmo tipo**.
* **Estático** significa que seu tamanho deve ser definido no momento da criação e **não pode ser alterado**.
* É ideal para armazenar coleções de dados onde o número de elementos é fixo ou conhecido previamente.

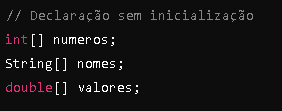
**2. Características do Vetor**

* **Tamanho fixo:** Uma vez definido, não pode ser alterado.
* **Baseado em índices:** Cada elemento do vetor pode ser acessado por seu índice, que começa em **0**.
* **Tipos homogêneos:** Todos os elementos do vetor devem ser do mesmo tipo (como int, double, String).
* Pode armazenar **tipos primitivos** e **objetos**.

**3. Declaração e Inicialização de Vetores**

Existem várias formas de declarar e inicializar vetores em Java.

**Declaração**



**Inicialização**

1. **Definindo o tamanho e inicializando com valores padrão:**
   * Os valores padrão dependem do tipo de dado:
     + int, double, float, etc.: **0**
     + boolean: **false**
     + char: **'\u0000'** (caractere nulo)
     + Objetos (como String): **null**



**Inicialização com valores específicos:**

* Você pode criar e atribuir os valores diretamente.



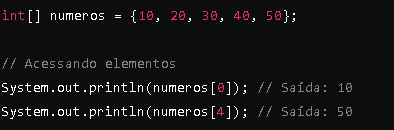
Dividindo declaração e inicialização:



### ****4. Acessando Elementos de um Vetor****

Os elementos de um vetor são acessados usando seu **índice**.

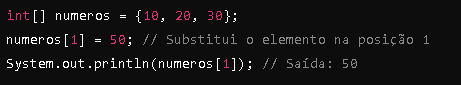
#### ****Exemplo****:



Se você tentar acessar um índice fora do intervalo do vetor, ocorrerá uma **ArrayIndexOutOfBoundsException**.

### ****5. Modificando Elementos****

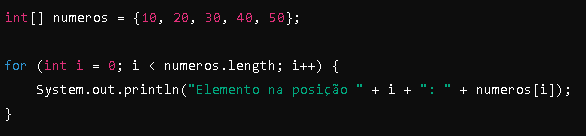
Os elementos de um vetor podem ser alterados diretamente através de seus índices:



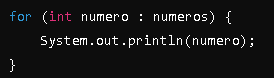
**6. Percorrendo um Vetor**

Para percorrer todos os elementos de um vetor, geralmente utilizamos **loops**.

1. **Usando for tradicional**:

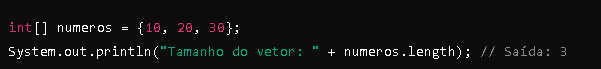


**Usando for-each**:



### ****7. Tamanho do Vetor****

O tamanho de um vetor pode ser obtido com a propriedade length:



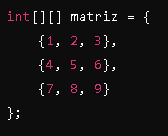
### ****8. Vetores Multidimensionais****

Além de vetores unidimensionais, Java também suporta **vetores multidimensionais** (como matrizes).

#### ****Declaração de um vetor bidimensional (matriz):****



#### ****Inicialização com valores específicos:****



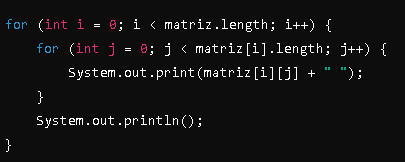
#### ****Acessando elementos em matrizes:****

Os elementos são acessados usando dois índices:



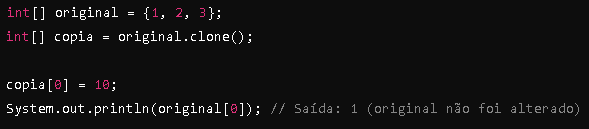
#### ****Percorrendo uma matriz:****

Você pode usar dois loops aninhados:

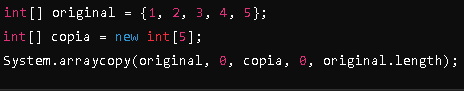


**9. Copiando Vetores**

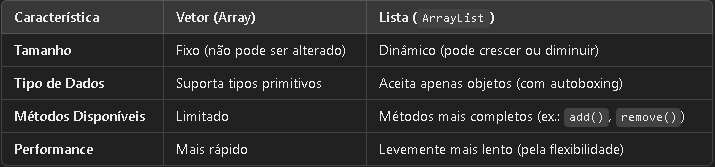
1. **Usando o método clone()**:



1. **Usando System.arraycopy()**:

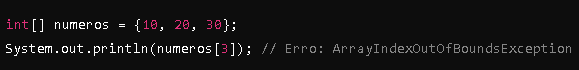


10. Diferença entre Vetores e Listas

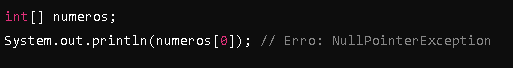


**11. Erros Comuns ao Usar Vetores**

* **Acessar um índice fora do intervalo**:



* **Esquecer de inicializar o vetor**:

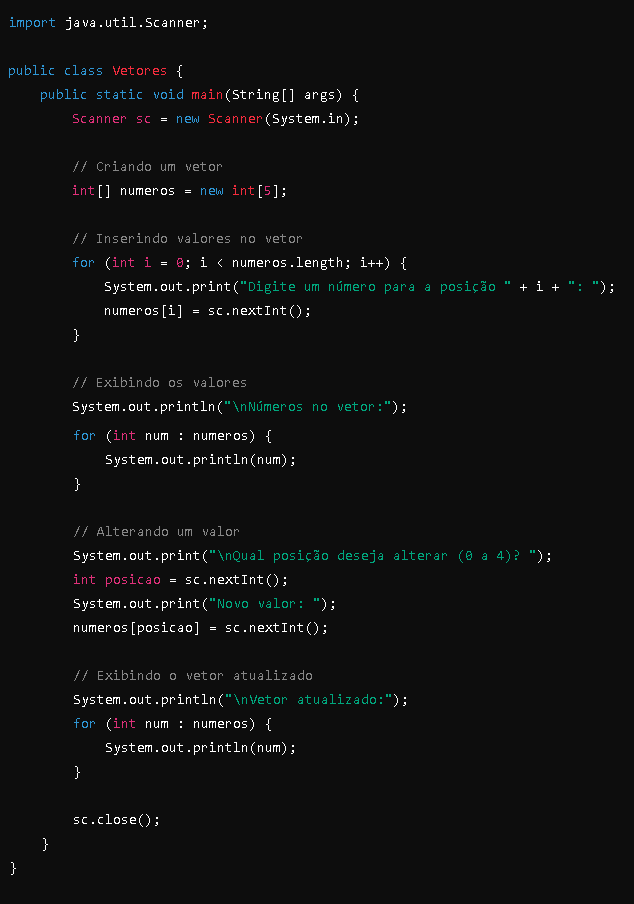


**Confundir o índice inicial (0) com o tamanho do vetor**:

* O índice do último elemento de um vetor é sempre length - 1.

### ****12. Exemplo Completo****

Aqui está um exemplo que utiliza várias operações com vetores:



### ****Conclusão****

Os **vetores** são uma estrutura de dados poderosa e essencial em Java. Apesar de serem estáticos, eles são rápidos e eficientes para armazenar dados de tamanho fixo.

**Listas:**

**1. O que é uma Lista em Java?**

* Uma **lista** em Java é uma **estrutura de dados dinâmica** usada para armazenar um conjunto de elementos em **ordem sequencial**.
* Ao contrário de arrays (que têm tamanho fixo), as listas podem **crescer** ou **diminuir** conforme elementos são adicionados ou removidos.
* Em Java, a interface List é usada para representar listas, e ela pertence ao pacote java.util.

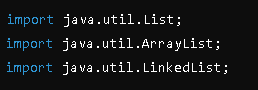
**2. Principais Implementações de List**

Existem várias implementações da interface List, mas as mais comuns são:

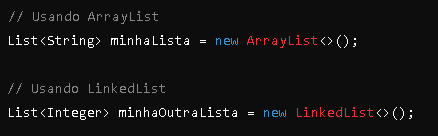
1. **ArrayList**:
   * Baseada em um array dinâmico.
   * Melhor para **acessos rápidos** a elementos pelo índice.
   * Exemplo: Boas para listas de leitura, onde há poucas operações de inserção/remoção.
2. **LinkedList**:
   * Baseada em uma lista duplamente encadeada.
   * Melhor para **inserção e remoção rápidas** de elementos no início ou no meio da lista.
   * Exemplo: Boas para filas e pilhas, ou onde há muitas alterações na lista.
3. **Vector** (pouco usado atualmente):
   * Parecido com ArrayList, mas **sincronizado** (suporta múltiplas threads).

**3. Sintaxe Básica para Usar List**

Antes de usar uma lista, você deve importá-la:



Criação de uma lista:



**4. Principais Métodos das Listas**

Aqui estão os métodos mais usados:

* **Adicionar elementos**:



* **Acessar elementos**:



* **Alterar elementos**:



* **Remover elementos**:



* **Verificar tamanho da lista**:



* **Verificar se a lista contém um elemento**:

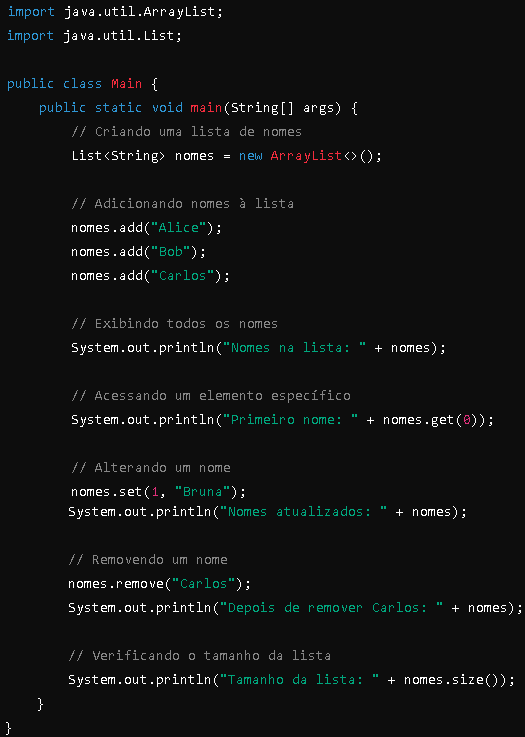


* **Limpar a lista**:

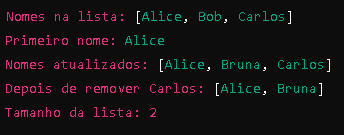


### ****5. Exemplo Prático com ArrayList****

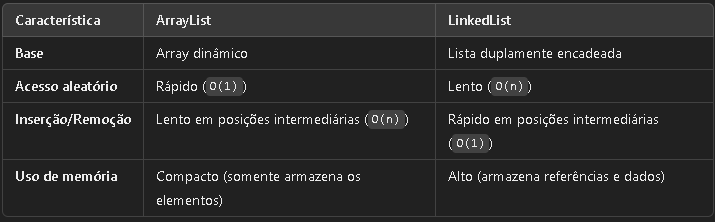
Vamos criar uma lista de nomes e manipular os elementos:



**Saída**:



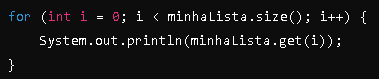
### ****6. Diferença entre**** ArrayList ****e**** LinkedList



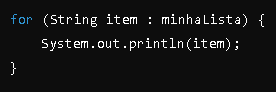
**7. Como Iterar sobre uma Lista?**

Há várias formas de percorrer uma lista:

* **Usando um for tradicional (com índice)**:



* **Usando um for-each**:



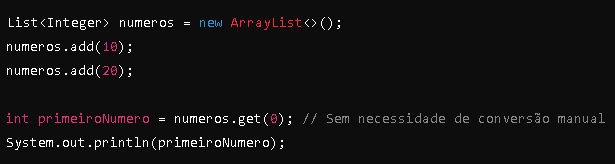
* **Usando o método forEach com Lambda**:



### ****8. Trabalhando com Tipos Primitivos em Listas****

As listas em Java não aceitam tipos primitivos diretamente (como int, double), mas aceitam seus equivalentes **wrappers** (como Integer, Double).

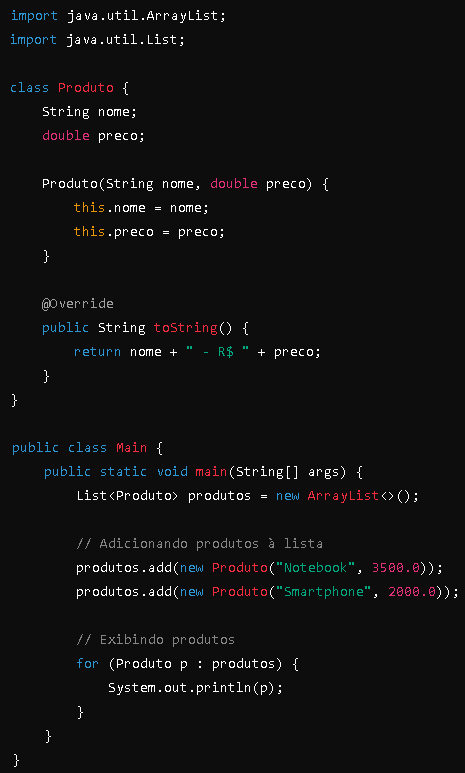
Por exemplo:



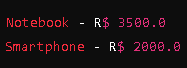
O Java faz o **autoboxing** (conversão automática entre primitivos e objetos) para você.

### ****9. Exemplo de Lista de Objetos****

Você também pode criar uma lista de objetos personalizados. Exemplo:

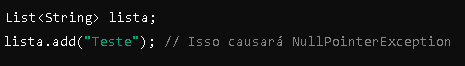


**Saída**:



**10. Erros Comuns com Listas**

* **Esquecer de inicializar a lista**:



Correção:



**Acessar índices inválidos**:



**Confundir tipos primitivos e wrappers**:



Correção:



**11. Quando usar List?**

* Use listas quando você precisar de uma coleção ordenada que permita elementos duplicados e precise de acesso pelo índice.
* Para requisitos mais específicos, considere outras coleções como Set (não permite duplicados) ou Map (chave-valor).

**Matrizes**

Em Java, uma **matriz** é uma estrutura de dados bidimensional (ou seja, uma "tabela" de elementos organizados em **linhas** e **colunas**). Enquanto um **vetor** (array unidimensional) armazena dados em uma única linha, uma matriz (array bidimensional) pode ser vista como uma **tabela** ou **grade**.

### 📌 ****Como declarar uma matriz em Java?****

Para criar uma matriz, usamos **colchetes duplos [][]** para indicar que há **duas dimensões** (linhas e colunas). Veja a sintaxe básica:



📌 **Exemplo**: Criando uma matriz de inteiros **3x3** (3 linhas e 3 colunas):

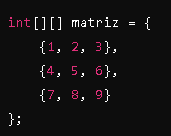


💡 **Isso significa que a matriz tem 9 espaços de armazenamento (3x3 = 9 posições).**

### 📌 ****Como inicializar uma matriz?****

Podemos inicializar uma matriz de duas formas:

#### ✅ ****1. Atribuindo valores manualmente****

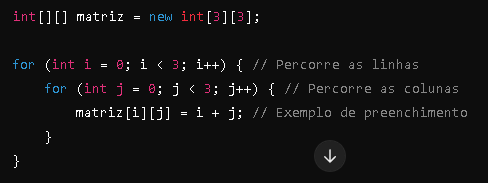


Isso cria a seguinte matriz:



#### ****2. Usando um loop para preencher a matriz****

Podemos usar **dois for aninhados** para preencher ou exibir os valores de uma matriz:

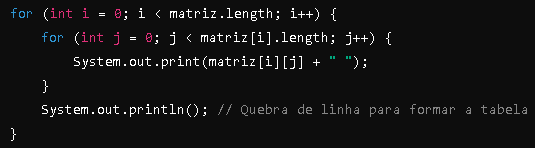


🛠 **Explicação**:

* O primeiro for percorre **as linhas**.
* O segundo for percorre **as colunas**.
* O valor de cada posição é definido como i + j.

### 📌 ****Como exibir os valores da matriz?****

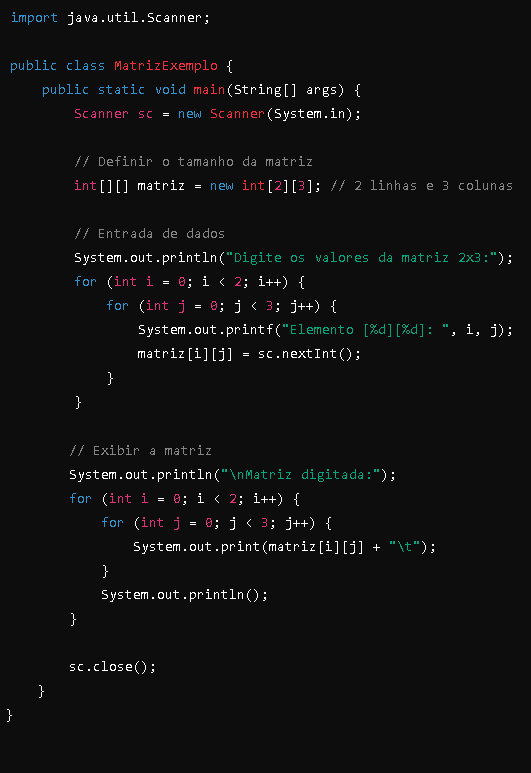
Podemos imprimir os valores da matriz no formato de **tabela**:



💡 O método matriz.length retorna a quantidade de **linhas**, e matriz[i].length retorna a quantidade de **colunas** de cada linha.

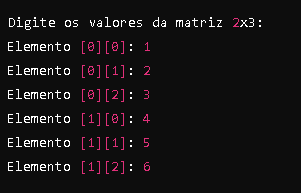
### 📌 ****Exemplo Completo: Lendo e imprimindo uma matriz****

Vamos criar um programa que recebe valores de uma matriz do usuário e depois os exibe formatados:

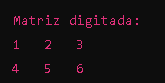


### 🖥 ****Exemplo de Entrada e Saída****

**Entrada:**



**Saída:**



### 📌 ****Exercício Prático****

💡 **Problema**: Faça um programa que leia uma matriz quadrada **3x3**, calcule a **soma dos elementos da diagonal principal** e exiba o resultado.

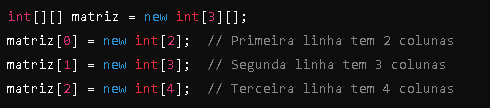
📌 **Exemplo de Matriz:**



💡 **Diagonal principal** = 1 + 5 + 9 = 15

### 📌 ****Dicas importantes****

✔ **Tamanho Dinâmico**: O Java permite criar matrizes sem tamanho fixo:



✔ **Usando List<List<Integer>> como alternativa flexível**:



* Permite uma matriz onde cada linha pode ter um número diferente de colunas.

## 🎯 **Resumo**

✅ Uma **matriz** é um array bidimensional (linhas e colunas).  
✅ Podemos **declarar, preencher e acessar** elementos usando loops for.  
✅ Podemos **imprimir** a matriz formatada no console.  
✅ Podemos usar **métodos auxiliares** para cálculos como soma da diagonal.  
✅ Em Java, existe alternativa dinâmica com List<List<Integer>.

**Introdução a data-hora e duração**

Conceitos importantes

**Data-[hora] local:**

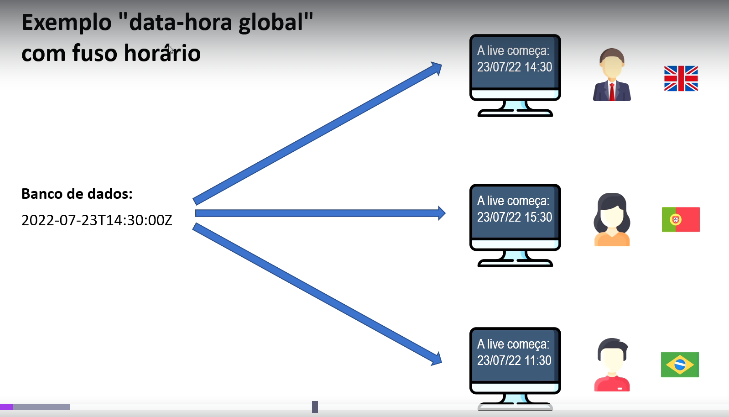
* ano-mês-dia-[hora] sem fuso horário
* [hora] opcional

**Data-hora global:**

* Ano-mês-dia-hora com fuso horário

**Duração:**

Tempo decorrido entre duas data-horas



Quando usar?

**Data-[hora] local:**

* Quando o momento exato não interessa a pessoas de outro fuso horário.
* Uso comum: sistemas de região única, Excel.

Data de nascimento: “15/06/2001”

Data-hora da venda: “13/08/2022 às 15:32” (presumindo não interessar fuso horário)

**Data-hora global:**

* Quando o momento exato interessa a pessoas de outro fuso horário.
* Uso comum: sistemas multi-região, web.
* Quando será o sorteio? “21/08/2022 às 20h (horário de São Paulo)”
* Quando o comentário foi postado? “há 17 minutos”
* Quando foi realizada a venda? “13/08/2022 às 15:32 (horário de São Paulo)”
* Início e fim do evento? “21/08/2022 às 14h até 16h (horário de São Paulo)”

**Entendendo timezone (fuso horário)**

Time (fuso horário)

**GMT – Greenwich Mean Time**

* Horário de Londres
* Horário do padrão UTC – Coordinated Universal Time
* Também chamado de “Z” time, ou Zulu time

**Outros fuso horários são relativos ao GMT/UTC:**

* São Paulo: GMT-3
* Manaus: GMT -4
* Portugal: GMT +1

**Muitas linguagens/tecnologias usam nomes para as timezones:**

* “US/Pacific”
* “America/São\_Paulo”
* Etc.

**Padrão ISSO 8601**

Data-[hora] local:

2022-07-21

2022-07-21T4:52

2022-07-22T14:52:09

2022-07-22T14:52:09.4073

Data-hora global:

2022-07-23T14:52:09Z

2022-07-23T14:52:09.254935Z

2022-07-23T14:52:09-03:00

**Operações importantes com data-hora**

**Instanciação**

* (agora) **->** Data-hora
* Texto ISSO 8601 **->** Data-hora
* Texto formato customizado -> Data-hora
* dia, mês, ano, [horário] -> Data-hora local

**Formatação**

* Data-hora **->** Texto ISSO 8601
* Data-hora **->** Texto formato customizado

**Operações importantes com data-hora**

Obter dados de uma data-hora local

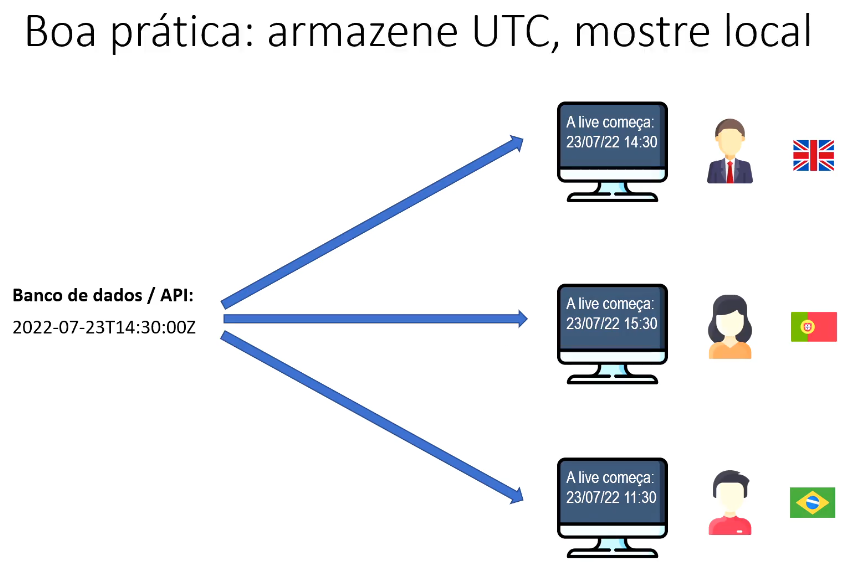
* Data-hora local **->** dia, mês, ano, horário

Converter data-hora global para local

* Data-hora global, timezone (sistema local) **->** Data-hora local

Cálculos comdata-hora

* Data-hora +/- tempo **->** Data-hora
* Data-hora 1, Data-hora 2 **->** Duração



**Instanciando data-hora em Java**

Principais tipos Java (versão 8+)

Data-hora local

* LocalDate
* LocalDateTime

Data-hora global

* Instant

Duração

* Duration

Outros

* Zoneld
* ChronoUnit

**Operações importantes com data-hora**

**Instanciação**

* (agora) -> Data-hora
* Texto ISSO 8601 -> Data-hora
* Texto formato customizado -> Data-hora
* Dia, mês, ano, [horário] -> Data-hora local

**Formatação**

* Data-hora -> Texto ISSO 8601
* Data-hora -> Texto formato customizado

**Converter data-hora para texto**

package application;

import java.time.Instant;

import java.time.LocalDate;

import java.time.LocalDateTime;

import java.time.ZoneId;

import java.time.format.DateTimeFormatter;

public class Program {

public static void main(String[] args) {

LocalDate d04 = LocalDate.*parse*("2025-02-16");

LocalDateTime d05 = LocalDateTime.*parse*("2025-02-16T17:44:18");

Instant d06 = Instant.*parse*("2025-02-16T17:44:51Z");

DateTimeFormatter fmt1 = DateTimeFormatter.*ofPattern*("dd/MM/yyyy");

DateTimeFormatter fmt2 = DateTimeFormatter.*ofPattern*("dd/MM/yyyy HH:mm");

DateTimeFormatter fmt3 = DateTimeFormatter.*ofPattern*("dd/MM/yyyy HH:mm").withZone(ZoneId.*systemDefault*());

DateTimeFormatter fmt4 = DateTimeFormatter.***ISO\_DATE\_TIME***;

DateTimeFormatter fmt5 = DateTimeFormatter.***ISO\_INSTANT***;

System.***out***.println("d04 = " + d04.format(fmt1));

System.***out***.println("d04 = " + fmt1.format(d04));

System.***out***.println("d04 = " + d04.format(DateTimeFormatter.*ofPattern*("dd/MM/yyyy")));

System.***out***.println("d05 = " + d05.format(fmt1));

System.***out***.println("d05 = " + d05.format(fmt2));

System.***out***.println("d05 = " + d05.format(fmt4));

System.***out***.println("d06 = " + fmt3.format(d06));

System.***out***.println("d06 = " + fmt5.format(d06));

System.***out***.println("d06 = " + d06.toString());

}

}

**Convertendo data-hora global para local**

package application;

import java.time.Instant;

import java.time.LocalDate;

import java.time.LocalDateTime;

import java.time.ZoneId;

public class Program {

public static void main(String[] args) {

LocalDate d04 = LocalDate.parse("2025-02-16");

LocalDateTime d05 = LocalDateTime.parse("2025-02-16T18:24:46");

Instant d06 = Instant.parse("2025-02-16T18:25:19Z");

//for (String s : ZoneId.getAvailableZoneIds()) {

// System.out.println(s);

//} datas globais

LocalDate r1 = LocalDate.ofInstant(d06, ZoneId.systemDefault());

LocalDate r2 = LocalDate.ofInstant(d06, ZoneId.of("Portugal"));

LocalDateTime r3 = LocalDateTime.ofInstant(d06, ZoneId.systemDefault());

LocalDateTime r4 = LocalDateTime.ofInstant(d06, ZoneId.of("Portugal"));

System.out.println("r1 = " + r1);

System.out.println("r2 = " + r2);

System.out.println("r3 = " + r3);

System.out.println("r4 = " + r4);

System.out.println("d04 dia = " + d04.getDayOfMonth());

System.out.println("d04 mês = " + d04.getMonthValue());

System.out.println("d04 ano = " + d04.getYear());

System.out.println("d05 hora = " + d05.getHour());

System.out.println("d05 minuto = " + d05.getMinute());

}

}

**Operações importantes com data-hora**

**Converter data-hora global para local**

* Data-hora global, timezone(sistema local) -> Data-hora local

**Obter dados em uma data-hora local**

* Data-hora local -> dia, mês, ano, horário

**Cálculos com data-hora**

* Data-hora +/- tempo -> Data-hora
* Data-hora 1, Data-hora 2 -> Duração

package application;

import java.time.Duration;

import java.time.Instant;

import java.time.LocalDate;

import java.time.LocalDateTime;

import java.time.temporal.ChronoUnit;

public class Program {

public static void main(String[] args) {

LocalDate d04 = LocalDate.*parse*("2025-02-16");

LocalDateTime d05 = LocalDateTime.*parse*("2025-02-16T18:24:46");

Instant d06 = Instant.*parse*("2025-02-16T18:25:19Z");

LocalDate pastWeekLocalDate = d04.minusDays(7);

LocalDate nextWeekLocalDate = d04.plusDays(7);

System.***out***.println("pastWeekLocalDate = " + pastWeekLocalDate);

System.***out***.println("nextWeekLocalDate = " + nextWeekLocalDate);

LocalDateTime pastWeekLocalDateTime = d05.minusDays(7);

LocalDateTime nextWeekLocalDateTime = d05.plusDays(7);

System.***out***.println("pastWeekLocalDateTime = " + pastWeekLocalDateTime);

System.***out***.println("nextWeekLocalDateTime = " + nextWeekLocalDateTime);

Instant pastWeekInstant = d06.minus(7, *ChronoUnit*.***DAYS***);

Instant nextWeekInstant = d06.plus(7, *ChronoUnit*.***DAYS***);

System.***out***.println("pastWeekInstant = " + pastWeekInstant);

System.***out***.println("nextWeekInstant = " + nextWeekInstant);

Duration t1 = Duration.*between*(pastWeekLocalDate.atTime(0, 0), d04.atTime(0, 0));

Duration t2 = Duration.*between*(pastWeekLocalDateTime, d05);

Duration t3 = Duration.*between*(pastWeekLocalDate.atStartOfDay(), d04.atStartOfDay());

Duration t4 = Duration.*between*(pastWeekInstant, d06);

Duration t5 = Duration.*between*(d06, pastWeekInstant);

System.***out***.println("t1 dias = " + t1.toDays());

System.***out***.println("t2 dias = " + t2.toDays());

System.***out***.println("t3 dias = " + t3.toDays());

System.***out***.println("t4 dias = " + t4.toDays());

System.***out***.println("t5 dias = " + t5.toDays());

}

}

**Trabalhando com datas – Date**

Representa um INSTANTE

Pacote java.util

<https://docs.oracle.com/javase/10/docs/api/java/util/Date.html>

Um objeto Date internamente armazena:

* O número de milissegundos desde a meia noite do dia 1 de janeiro de 1970 GMT(UTC)
* GMT: Greenwich Mean Time (time zone)
* UTC: Coordinated Universal Time (time standard)

SimpleDateFormat

* <https://docs.oracle.com/javase/10/docs/api/java/text/SimpleDateFormat.html>
* Define formatos para conversão entre Date e String
* dd/MM/yyyy -> 16/02/2025
* dd/MM/yyyy HH:mm:ss -> 16/02/2025 19:02:17

Padrão ISSO 8601 e classe Instant

* Formato: yyyy-MM-ddTHH:mm:ssZ
* Exemplo: “2025-02-16T19:02:17Z”
* Date y3 = Date.from(Instant.parse(“2025-02-16T19:02:17Z”));
* package application;
* import java.text.ParseException;
* import java.text.SimpleDateFormat;
* import java.time.Instant;
* import java.util.Date;
* import java.util.TimeZone;
* public class Program {
* public static void main(String[] args) throws ParseException {
* SimpleDateFormat sdf1 = new SimpleDateFormat("dd/MM/yyyy");
* SimpleDateFormat sdf2 = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
* SimpleDateFormat sdf3 = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
* sdf3.setTimeZone(TimeZone.*getTimeZone*("GMT"));
* Date y1 = sdf1.parse("16/02/2025");
* Date y2 = sdf2.parse("16/02/2025 19:10:05");
* Date y3 = Date.*from*(Instant.*parse*("2025-02-16T19:16:44Z"));
* Date x1 = new Date();
* Date x2 = new Date(System.*currentTimeMillis*());
* Date x3 = new Date(0L);
* Date x4 = new Date(1000L \* 60L \* 60L \* 5L);
* System.***out***.println(y1);
* System.***out***.println(y2);
* System.***out***.println("-------------------");
* System.***out***.println("x1: " + x1);
* System.***out***.println("x2: " + x2);
* System.***out***.println("x3: " + x3);
* System.***out***.println("x4: " + x4);
* System.***out***.println("y1: " + y1);
* System.***out***.println("y2: " + y2);
* System.***out***.println("y3: " + y3);
* System.***out***.println("-------------------");
* System.***out***.println("x1: " + sdf2.format(x1));
* System.***out***.println("x2: " + sdf2.format(x2));
* System.***out***.println("x3: " + sdf2.format(x3));
* System.***out***.println("x4: " + sdf2.format(x4));
* System.***out***.println("y1: " + sdf2.format(y1));
* System.***out***.println("y2: " + sdf2.format(y2));
* System.***out***.println("y3: " + sdf2.format(y3));
* System.***out***.println("-------------------");
* System.***out***.println("x1: " + sdf3.format(x1));
* System.***out***.println("x2: " + sdf3.format(x2));
* System.***out***.println("x3: " + sdf3.format(x3));
* System.***out***.println("x4: " + sdf3.format(x4));
* System.***out***.println("y1: " + sdf3.format(y1));
* System.***out***.println("y2: " + sdf3.format(y2));
* System.***out***.println("y3: " + sdf3.format(y3));
* }
* }

**Manipulando uma data com Calendar**

Somando uma unidade de tempo

SimpleDateFormat sdf = new SimpleDateFormat(“dd/MM/yyyy HH:mm:ss”);

Date d = Date.from(Instant.parse(“2025-02-16T19:23:46Z”));

System.out.println(sdf.format(d));

Calendar cal = Calendar.getInstance();

cal.setTime(d);

cal.add(Calendar.**HOUR\_OF\_DAY**, **4**);

d = cal.getTime();

System.out.println(sdf.format(d));

package application;

import java.text.ParseException;

import java.text.SimpleDateFormat;

import java.time.Instant;

import java.util.Calendar;

import java.util.Date;

public class Programm {

public static void main(String[] args) throws ParseException {

SimpleDateFormat sdf = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");

Date d = Date.from(Instant.parse("2025-02-16T19:30:22Z"));

System.out.println(sdf.format(d));

Calendar cal = Calendar.getInstance();

cal.setTime(d);

cal.add(Calendar.HOUR\_OF\_DAY, 4);

d = cal.getTime();

int minutes = cal.get(Calendar.MINUTE);

int month = 1 + cal.get(Calendar.MONTH);

System.out.println(sdf.format(d));

System.out.println("Minutes: " + minutes);

System.out.println("Month: " + month);

}

}

04/03/2025

**Enumerações**

É um tipo especial que serve para especificar de forma literal um conjunto de constantes relacionadas.

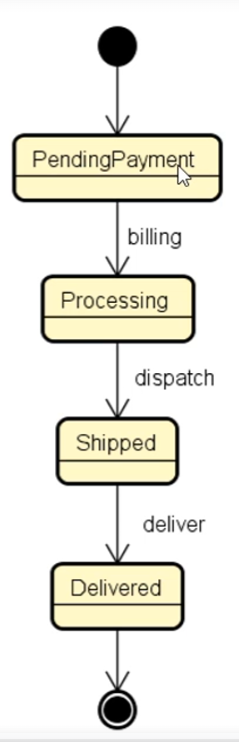
Palavra chave em Java: enum

Vantagem: melhor semântica, código mais legível e auxiliado pelo compilador

Referência: <https://docs.oracle.com/javase/tutorial/java/javaOO/enum.html>

Exemplo:

Ciclo de vida de um pedido.



package entities.enums;

public enum OrderStatus {

PENDING\_PAYMENT,

PROCESSING,

SHIPPED,

DELIVERED;

}

package entities;

import java.util.Date;

import entities.enums.OrderStatus;

public class Order {

private Integer id;

private Date moment;

private OrderStatus status;

}

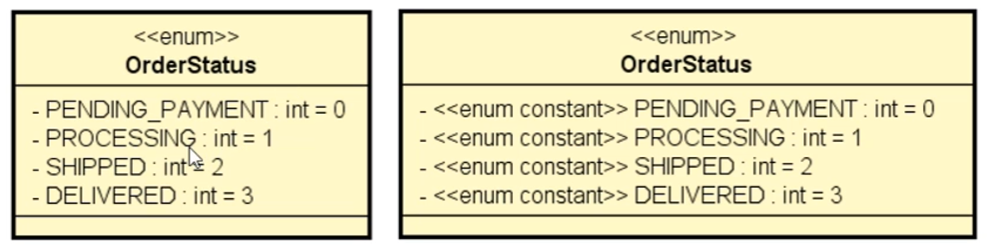
Exemplo pratico no GitHub aula 120.

Conversão de String par enum

OrderStatus os1 = OrderStatus.***DELIVERED***;

OrderStatus os2 = OrderStatus.valueOf(“*DELIVERED*”);

**Notação UML:**



**Sobre design**

Categorias de classes

Em um sistema orientado a objetos, de modo geral “tudo” é objeto.

Por questões de design tais como organização, flexibilidade, reuso, delegação, etc., há várias categorias de classes:

*Views;*

*Controllers;*

*Entities;*

*Services;*

*Repositories*

