

# WAGENGEN DE PROGRAMAS I

## UNIDADE 1

- 1 Conceitos de Programas de Computadores.
- 2 Modelos *(Máquinas e Arquiteturas e origens e objetos.)*
- 3 Paradigma de Programação a Objetos.

- 4 Encapsulamento *(Exemplo de nomes e lista)*

- 5 Classes, Objetos, Atributos e Métodos

- 6 Desenvolvimento de Programas Classes *(AIXTAC) dicas*
- 7 TIPOS PONTUAOS *(Parâmetros recursos, tipos de limite de limite de tipos)*

- 8 Operadores

- 9 Conversões entre TIPOS NÚMEROS *Linha 2 OK*

- 10 Operadores Relacionais e Lógicos

- 11 Aplicações

- 12 Escopo de Variáveis e Objetos

- 13 Manipulações do Processo

- 14 Entrada e Saída de DADOS

- 15 Estruturas Condicionais

- 16 Estruturas de Repetição

- 17 Operador NEW

Garbage Collection

Construtores

[L3] x [L4]

Sobrecargas

Váriaveis

Matrizes

Simulado Prática

## ① CONCEITOS DE PROGRAMA DE COMPUTADOR

↪ Programa de Computador.

\* CONJUNTO DE **COMANDOS** E **REGRAS** PARA A **MANIPULAÇÃO** DE UM **COMPUTADOR**.

Programas / Comandos regras

\* SÃO **ESCRITOS** EM UMA **LINGUAGEM DE PROGRAMAÇÃO**.

Programa → Código ↓, compilam ↓ Computar.

\* Para um **PROGRAMA** SER EXECUTADO OS **COMANDOS** DEVEM SER **TRANSLATOS** P/ UMA **LINGUAGEM** QUE POSSA SER **COMPREENDIDA** PELO **COMPUTADOR** AFIMOS DO **COMPILADOR**.

\* Programas processam dados.

\* O PARADIGMA DE PROGRAMAÇÃO CONCEDE A OBSTACO CONVERSÃO DOS DADOS A SEREM PROCESSADOS E OS MECANISMOS DE PROCESSAMENTO DE DADOS DEVEM SER CONSIDERADOS EM CONJUNTO

\* OS **MODELOS** REPRESENTAM **CONCRETAMENTE** **DADOS** E **OPERAÇÕES** SOBRE ESSES DADOS.

P.O.  
↓  
DADOS  
↓  
Mecanismos manipulando dados

(Modelo) Dados  
Operações

## ② MODELOS

\* REPRESENTAÇÃO SIMPLIFICADA DE:

Objetos  
Pessoas  
Itens  
Tarefas

Processos  
conceitos  
informações

Restaurante Casino Hiperativo

MESA 1	MEAL 2	MESA 3
kg Refeições	kg Refeições	kg Refeições
Somatório	Somatório	Somatório
Refrigerante	Refrigerante	Refrigerante
Conta	Conta	Conta

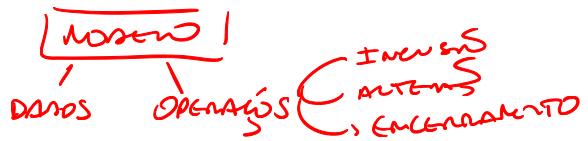
\* Representação de forma simplificada das informações do restaurante necessárias para controlar os pedidos.

Modelo  
Representando → Dados

\* O **MODELO** REPRESENTA CERTOS **DADOS** OU **INFORMAÇÕES**.

\* Os **DADOS** CONTINOS NO **MODELO** SÃO SOMENTE **RELEVANTES** À **ARQUITETURA** DO NÚMERO REAL FONTE.

## 2.1 - MODELOS E OPERAÇÕES



- \*) Um **Modelo** **CONTÉM** **OPERAÇÕES** ou PROCEDIMENTOS ASSOCIADOS A ELE
- \*) Incluem de um PESSOAS P/MESA.
  - \*) MODIFICAÇÃO DE ESTADOS DO PESSOAS
  - \*) ENCAPSULAMENTO DOS PESSOAS DE UMA MESA.
- \*) **Classe** de **Modelos** SÓMENTE DE **DADOS** **OU** SÓMENTE DE **OPERAÇÕES**.
- \*) **Modelos** PODEM CONTAR **Submodelos** **a** SER FONTE DE **OUTROS Modelos**.
- 
- ```
graph TD; Modelo[Modelo] --> dados[dados]; Modelo --> subModelo1[Modelo 1]; subModelo1 --> subDados1[dados]; subModelo1 --> subModelo2[Modelo 2]
```

## 2.2 - Modelos e Orientação a Objetos

- \*) A **SIMPLEZA** DE UM **Modelo** SEM AS **INFORMAÇÕES** SONÁ NECESSÁRIA, E, DEPENDENDO DO **CONTEXTO**, ALGUMAS **INFORMAÇÕES** DEVERÃO SER **OCULTADAS** OU **ESCONDEDIDAS**.
- Modelo - Dados** → PESSOAS SÃO EMPREGOS DE UMA EMPRESA.
- Simulação** → PESSOAS SÃO FALANTE DA UMA DIVERSA MÍDIA.
- Automação/Informações** → PESSOAS SÃO CONTATO COMERCIAL.
- \*) A **ENAGEM** E **USO** DE **Modelos** É UMA TAREFA NATURAL E A **EXTENSÃO** DESTA ABORDAGEM À **Programação** DEU ORIGEM AO **PARADIGMA DA PROGRAMAÇÃO** ORIENTADA A **OBJETOS**.

## ③ PARADIGMA DE ORIENTAÇÃO A OBJETOS

- \*) **Modelos Passando em Objetos:**
- COMPUTAÇÕES DE PROBLEMAS;
  - COMUNICAÇÃO COM OS USUÁRIOS DAS APLICAÇÕES;
  - MANUTENÇÃO DE EMPRESAS;
  - PRODUZIR PROGRAMAS, ETC;
- \*) **POO** É UM **PARADIGMA DE PROGRAMAÇÃO** DE COMPUTAÇÕES ONDE SE USAM **classes** E **objetos**, CRIANDO A PARTIR DE **Modelos** DESSENTOS ANTERIORMENTE, PARA **REPRESENTAR** E **PROCESSAR** **DADOS** COMO **Software** **Sistema Operacional** **Programas de COMPUTADORES**.
- 
- ```
graph TD; Problem[Problema] --> Models[Modelos]; Models --> Classes[Classes]; Classes --> Objects[Objetos]; Objects --> Representacao[REPRESENTAR E PROCESSAR DADOS]; Objects --> Software[Software Sistema Operacional Programas de COMPUTADORES]
```

## 4 ENCAPSULAMENTO

- \*) A capacidade de ocultar dados dentro dos objetos, permittendo só acessar especificamente as informações que descriuem os dados.
- \*) Um dos benefícios mais importantes da P.O.O.
- \*) Modelos de Encapsulamento os dados possuem a forma de programar com menos erros e mais eficiência.
- \*) Modelos podem contar dados para representação das informações ou dados recursivos ao uso se desear modelar e operar pl manipular dos dados. Modelo representação de informações + Operações de manipulação de dados
- \*) As índices serão desenvolvidos de modo que os dados não possam ser acessados ou usados diretamente, mas somente através das operações.

\*) Em muitos sistemas temos vantagens em usar mecanismos de encapsulamento de dados:

Obs -> sempre que fazemos uma interface de usuário ao modelo a capacidade e responsabilidade para manipular os dados de um dos seres vivos, devem ser unidas com juntas pl para fazê-lo.

Linha branca -> DP. Retângulo -> Programa

### Exemplos de Modelos

I) -> Lâmpada Inversamente

Lâmpada
estadoDaLampada <u>Dados</u>
acende()
apaga()
mostraEstado()

Modelo Lampada  
 <inicio\_modelo>  
 dado estadoDaLampada;  
 operação acende()  
 <ini>  
 estadoDaLampada = aceso;  
 <fim>  
 operação apaga()  
 <ini>  
 estadoDaLampada = apagado;  
 <fim>  
 operação showData()  
 <ini>  
 se (estadoDaLampada == aceso)  
 <ini>  
 imprime "A lampada esta acesa"  
 <fim>  
 senão  
 <ini>  
 imprime "A lampada esta apagada"  
 <fim>  
 <fim>  
 <fim\_modelo>

Representa lâmpada em uso.  
 Status da lâmpada  
 Op. P/ Acender lâmpada  
 D.P P/ Apagar lâmpada  
 D.P P/ Mostrar status da lâmpada

[2] → Conta Bancária Simplificada

ContaBancariaSimplificada
nomeDoCorrentista
saldo
contaEEspecial
abreConta(nome,deposito,éEspecial)
abreContaSimple(nome)
deposita(valor)
retira(valor)
mostraDados()

Modelo ContaBancariaSimplificada

```

<inicio modelo>
< dado, nomeDoCorrentista, saldo, contaEEspecial;
  operação abreConta(nome, deposito, especial)
  <ini>
    nomeDoCorrentista = nome;
    saldo = deposito;
    contaEEspecial = especial;
  <fim>
  operação abreContaSimple(nome)
  <ini>
    nomeDoCorrentista = nome;
    saldo = 0.00;
    contaEEspecial = false;
  <fim>
  operação depositaValor(valor)
  <ini>
    saldo = saldo + valor;
  <fim>
  operação retira(valor)
  <ini>
    se (contaEEspecial == false)
      se (valor <= saldo)
        saldo = saldo - valor;
      <fim>
    <fim>
    senão
      saldo = saldo - valor
    <fim>
  <fim>
  operação showData()
  <ini>
    imprime "o nome do correntista é " + nomeDoCorrentista;
    imprime " o saldo é " + saldo;
    imprime
    <ini>
      se (contaEEspecial) imprime "A conta é especial"
    <fim>
  <fim>
<fim_modelo>
```

*dados da conta*

*argumentos ? / A operação.*

*anterior nome.*

*Iniciando Simultaneamente todos os dados do bloco.*

*Mais - saldo = movimento;*

*Depois um na conta*

*Acumular valor contas*

*se is for Especial e maior menor em Tanto > ao saldo.*

*caso sem Especial, Pode Retirar a Conta.*

*mostrar os dados INFORMAÇÕES dos valores.*

[3] → DataSimples

Data
dia
mês
ano
initializaData(d,m,a)
dataÉVálida(d,m,a)
mostraData()

Modelo DataSimples

```

<inicio modelo>

dados dia, mês, ano;

operação inicializaData(d,m,a)
<ini>
  dia = d;
  mês = m;
  ano = a;
<fim>
operação dataValida(d,m,a)
<ini>
  se (d <= 30 && m <= 12 && a > 2021)
    imprime "data é valida e " + dia + "/" + mes + "/" + ano;
  <fim>
  senão
    <ini>
      imprime "data não é valida ";
    <fim>
  <fim>
operação showData()
<ini>
  imprime "data : " + dia + "/" + mes + "/" + ano;
<fim>
<fim_modelo>
```

# Exercícios Week 1 - Modelagem

## 1) → Modelo Restaurante

Modele um restaurante que oferece a seus clientes comida a Kg, sobremesa, refrigerante e cerveja. O valor do Kg de comida é R\$ 30,00, a unidade da sobremesa fica por R\$ 6,00, o valor do refrigerante é R\$ 5,00 e a cerveja custa R\$ 10,00. A conta é contabilizada através do número da mesa, sendo de responsabilidade do estabelecimento informar o total geral e o valor por pessoa, que é calculado através da divisão do valor total pelo número de integrantes da mesa.

As principais ações do modelo são referentes a abertura do pedido, a adição de novas pessoas na mesa caso cheguem mais tarde, a adição de comida, sobremesa, refrigerante e cerveja a qualquer instante que seja necessário, o fechamento da conta e a emissão da nota fiscal com o valor total e com o valor por pessoa.

### Modelo restaurante

<início do modelo>

```
dados numeroMesa, kg, qtdSobremesa, qtdRefrigerante, qtdCerveja, total,
numeroPessoas, totalPorPessoa;
valorKg = 30.00; valorSobremesa = 6.00; valorRefrigerante = 5.00;
valorCerveja = 10.00;
```

operação inicializaçãoDoPedido(nMesa, qtdR, qtdC, nPessoas)

<início>

```
numeroMesa = nMesa;
qtdRefrigerante = qtdR;
qtdCerveja = qtdC;
numeroPessoas = nPessoas;
kg = 0;
qtdSobremesa = 0;
```

<fim>

operação adiçãoDeComida(peso)

<início>

```
Kg = kg + peso;
```

<fim>

*Adicionar peso*

operação adiçãoDeSobremesa(qtd)

<início>

```
qtdSobremesa = qtdSobremesa + qtd;
```

<fim>

*Adicionar sobremesa*

operação adiçãoDeRefrigerante(qtd)

<início>

```
qtdRefrigerante = qtdRefrigerante + qtd;
```

<fim>

*Adicionar refrigerante*

operação adiçãoDeCerveja(qtd)

<início>

```
qtdCerveja = qtdCerveja + qtd;
```

<fim>

*Adicionar cerveja*

operação adiçãoDePessoas(n)

<início>

```
numeroPessoas = numeroPessoas + n;
```

<fim>

*Fechar conta*

operação Fechamento()

<início>

```
total = (qtdCerveja * valorCerveja) + (qtdRefrigerante * valorRefrigerante) +
(qtdSobremesa * valorSobremesa) + (kg * valor kg);
```

<fim>

operação divisãoDaConta()

<início>

```
totalPorPessoa = total / numeroPessoas;
```

<fim>

operação emissãoNota()

<início>

```
imprime("O valor da conta é = "+total);
imprime("O valor por pessoa é = "+totalPorPessoa);
```

<fim>

<fim do modelo>

### Restaurante

Número da mesa

Kg

QTD	Sobremesa
	Refrigerante
	Cerveja
	Pessoas

Total

Total R\$ por pessoa

V	Numero
	Sobremesa
	Refrigerante

- Início()
- ADD Comida()
- ADD Sobremesa()
- ADD Refri()
- ADD Cerveja()
- Adicionar()
- FEchar conta()
- Divisão da conta()
- Emissão Nota()

## 2] → Lâmpada Sem Mercado

Escreva um modelo para representar uma lâmpada que está à venda em um supermercado. Que dados devem ser representados por este modelo?

Lâmpada

Marca  
Potência  
Preço  
Quantidade  
Voltagem  
  
Iniciar()  
Vender()  
Lampada()  
Reajuste()

```

Modelo LampadaMercado
<Início do modelo>

dados marca, potência, preço, qtd, voltagem;

operação inicializa(Marca, Potência, Preço, Quantidade, Voltagem)
<ini>
    marca = Marca;
    potência = Potência;
    qtd = Quantidade;
    preço = Preço;
    voltagem = Voltagem;
<fim>

operação venda(Quantidade)
<ini>
    Se (Quantidade <= qtd)
        <ini>
            qtd=qtd-Quantidade;
            retorna Quantidade * preço;
        <fim>
    <fim>

operação compra(Quantidade)
<ini>
    qtd = qtd + quantidade;
<fim>

operação reajuste(percentual)
<ini>
    preço = preço + (preço * percentual);
<fim>
<Fim do modelo>
```

## 3] → Lâmpada Tres Estados

Imagine uma lâmpada que possa ter três estados: apagada, acesa e meia-luz. Usando o modelo “Lâmpada” como base, escreva o modelo “LampadaTresEstados”.

Lâmpada

Marca  
Potência  
Preço  
Quantidade  
Voltagem  
  
Iniciar()  
Vender()  
Lampada()  
Reajuste()

LÂMPADA

**ESTADO** DA LÂMPADA
 

- ↳ ACEITA
- ↳ APAGADA
- ↳ MEIA-LUZ

 Marca  
Potência  
  
IniciarLampada()  
Acesa()  
Apaga()  
MeiaLuz()

Modelo LampadaTresEstados

```

<Início do modelo>

 dado estadoDaLampada, marca, potência;

 operação acende()
 <ini>
    estadoDaLampada = aceso;
 <fim>

 operação apaga()
 <ini>
    estadoDaLampada = apagado;
 <fim>

 operação meiaLuz()
 <ini>
    estadoDaLampada = meiaLuz;
 <Fim>

 operação mostraEstado()
 <ini>
    se (estadoDaLampada == aceso)
        <ini>
            imprime "A lâmpada está acesa"
        <fim>
    senão
    <ini>
        se (estadoDaLampada == apagado)
            <ini>
                imprime "A lâmpada está apagada";
            <fim>
        senão
        <ini>
            imprime "A lâmpada está em meia luz";
        <fim>
    <fim>
 <fim>
<Fim do modelo>
```

4)  $\rightarrow$  Lâmpada + hora 80min

Inclua, no modelo “Lâmpada”, uma operação “estáLigada” que retorne verdadeiro se a lâmpada estiver ligada e falso, caso contrário.



5)  $\rightarrow$  Livro

Crie um modelo Livro que represente os dados básicos de um livro, sem se preocupar com a sua finalidade.



Modelo LampadaTresEstados  
<Início do modelo>

```

    dado estadoDaLampada, marca, potencia;
    operação acende()
    <ini>
        estadoDaLampada = aceso;
    <fim>

    operação apaga()
    <ini>
        estadoDaLampada = apagado;
    <fim>

    operação meiaLuz()
    <ini>
        estadoDaLampada = meiaLuz;
    <Fim>

    operação mostraEstado()
    <ini>
        se (estadoDaLampada == aceso)
            <ini>
                imprime "A lâmpada está acesa"
            <fim>
        senão
        <ini>
            se (estadoDaLampada == apagado)
                <ini>
                    imprime "A lâmpada está apagada";
                <fim>
            senão
            <ini>
                imprime "A lâmpada está em meia luz";
            <fim>
        <fim>

    operação estáLigada()
    <ini>
        se (estadoDaLampada == apagada)
    <ini>
        return false;
    <fim>
    <retorna verdadeiro;>
    <fim>

<Fim do modelo>

```

Modelo Livro

<Início do modelo>

Dados ISBN, título, autor, editora, edicao, paginas;

InicializarLivro(Isbn, Titulo, Autor, Editora, Edicao, Paginas)

<Ini>

```

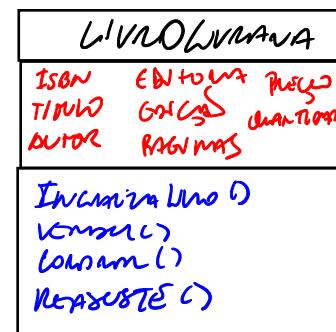
ISBN = Isbn;
título = Titulo;
autor = Autor;
editora = Editora;
edicao = Edicao;
paginas = Paginas;

```

<fim>

<Fim do modelo>

6)  $\rightarrow$  Modelo Livro Livraria



Modelo LivroLivraria

<Início>

Dados ISBN, título, autor, editora, edicao, paginas, preco, qtd;

InicializarLivreLivraria(Isbn, Titulo, Autor, Editora, Edicao, Paginas, Preco, Qtd)

<Ini>

```

ISBN = Isbn;
título = Titulo;
autor = Autor;
editora = Editora;
edicao = Edicao;
paginas = Paginas;
preco = Preco;
qtd = QTD;

```

<fim>

vender(Quantidade)

<Ini>

Se (Quantidade <= qtd)

<ini>

```

qtd = qtd - Quantidade;
retorna (Quantidade * preco);

```

<fim>

Senão

Imprime "Não tenho esta qtd no estoque";  
retorna 0;

<fim>

Comprar(Quantidade)

<Ini>

qtd = qtd + Quantidade

<fim>

reajustePreco(percentual)

<Ini>

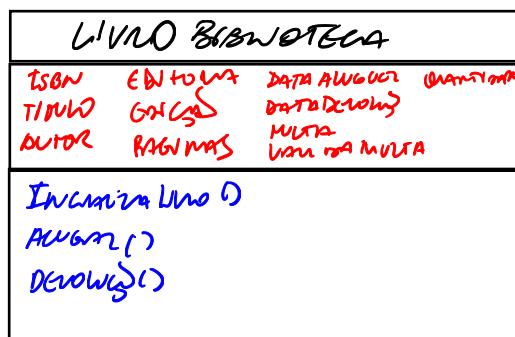
preco = preco + (preco \* percentual);

<fim>

<Fim do modelo>

# 7 → Modelo Livro de Biblioteca

Usando o resultado do modelo “Livro” como base, crie um modelo “LivroDeBiblioteca” que represente os dados básicos de um livro de uma biblioteca, que pode ser emprestado a leitores.



## Modelo LivroBiblioteca

```

<Inicio>
    Dados ISBN, titulo, autor, editora, edicao, paginas, qtd, dataAluguel,
    dataDevolucao, multa, valorMult=5;

    Operacao InicializarLivroBiblioteca(Isbn, Titulo, Autor, Editora, Edicao,
                                         Paginas, Qtd)

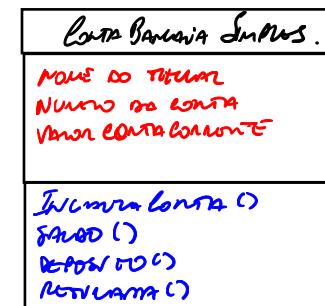
    <Ini>
        ISBN = Isbn;
        titulo = Titulo;
        autor = Autor;
        editora = Editora;
        edicao = Edicao;
        paginas = Paginas;
        qtd = QTD;
    <fim>

    Operacao Alugar(data)
    <ini>
        Se (qtd > 0)
        <ini>
            qtd = qtd-1;
            dataAluguel = data;
            dataDevolucao = data + 7;
            retorna (dataDevolucao);
        <fim>
        Senao
            Imprime "Não tenho o livro disponivel";
            retorna (data);
        <fim>
    <fim do modelo>

    Operacao Devolucao(data)
    <ini>
        Dado atraso=data - dataDevolucao
        Dado valor=0;
        qtd = qtd + 1;
        Se (atraso > 0)
        <ini>
            valor = atraso * valorMult;
        <fim>
        retorna valor;
    <fim>
<fim do modelo>
```

# 8 → Modelo Conta Bancaria Simplificada

Escreva um modelo para representar uma conta bancária simplificada com os seguintes dados: nome do titular, número da conta e valor na conta corrente. Crie uma operação para abrir a conta inicializando os dados. Além disso, especifique as operações para mostrar os dados da conta (emitir saldo), e fazer movimentações de depósitos e retiradas



## Modelo ContaBancaria

```

<inicio do modelo>

    Dado nomeDoTitular, numeroDaConta, saldo;

    operacao inicializa(Nome, Numero, Saldo)
    <ini>
        nomeDoTitular = Nome;
        numeroDaConta = Numero;
        saldo = Saldo;
    <fim>

    operacao emitirSaldo()
    <ini>
        imprime("Nome do titular: "+nomeDoTitular);
        imprime("Número da conta: "+numeroDaConta);
        imprime("Saldo: "+saldo);
    <fim>

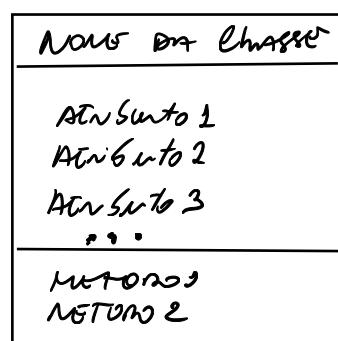
    operacao deposita(valor)
    <ini>
        saldo = saldo + valor;
    <fim>

    operacao retira(valor)
    <ini>
        Se (valor <= saldo)
        <ini>
            saldo = saldo - valor;
        <fim>
        Senao
        <ini>
            Imprime "Saldo insuficiente";
        <fim>
    <fim>

<Fim do modelo>
```

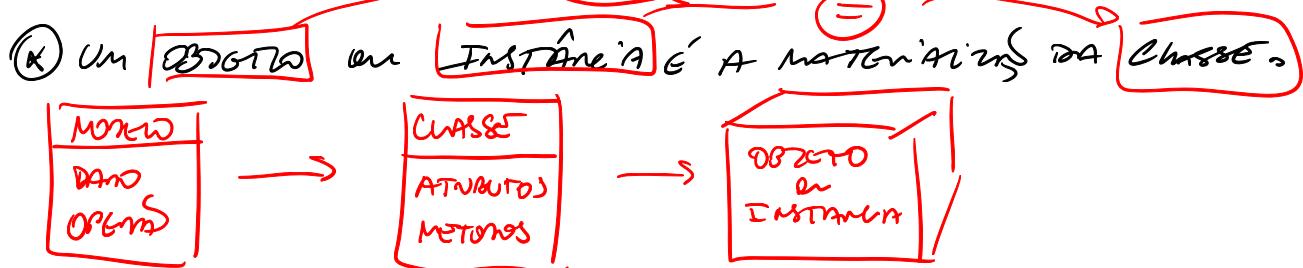
## 5 Classes, Objetos, Métodos & Atributos

④ A partir do P.O.O podemos criar e usar **OBETOS** a partir de **CLASSES**, que são estruturas dinamicamente com os métodos descritos.



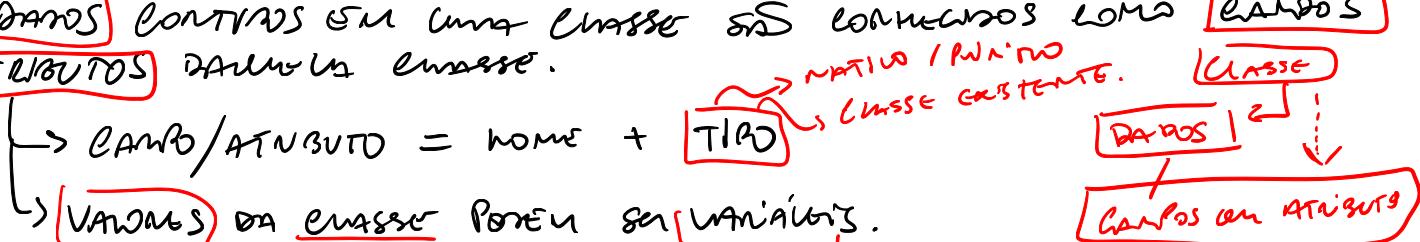
⑤ **CLASSES** são estruturas organizadas para resguardar os **MÉTODOS**.

- ↳ Contêm a descrição dos **DADOS** (**ATRIBUTOS**)
- ↳ Contêm a descrição das **OPERAÇÕES** (**MÉTODOS**)



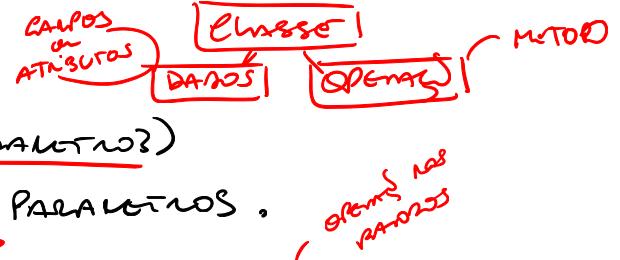
⑦ Os **DADOS** contidos em uma classe são conhecidos como **CAMPOS** ou **ATRIBUTOS** daquela classe.

- Campo / ATRIBUTO = nome + **TIPO**
- **VARIÁVEIS** da classe podem ser variáveis.

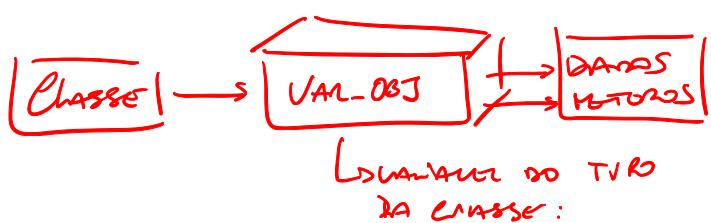
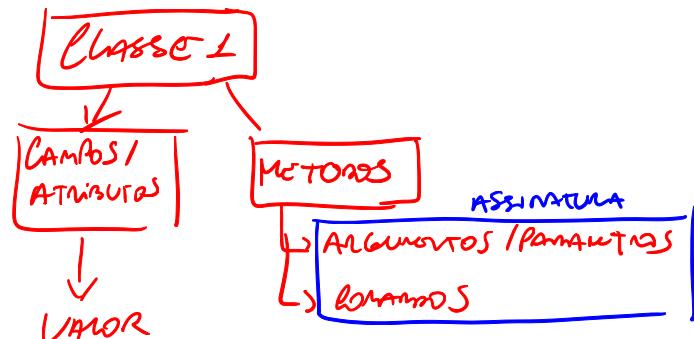


⑧ As **OPERAÇÕES** contidas são os **MÉTODOS**.

- ↳ Podem receber **ARGUMENTOS** (Parâmetros?)
- ↳ ASSINATURA → nome do método + Parâmetros.



⑨ Para **OBETOS** em **INSTÂNCIAS** serem criados, é necessário a existência de **REFERÊNCIAS** a estes **OBETOS**, que são VARIÁVEIS do **TIPO** da **CLASSE**.



## ⑥ DESENVOLVIMENTO DA PRIMEIRAS CLASSES

### 6.1 → Classes: Sintaxe Básica

#### ① Declaração:

↳ class + nome\_da\_classe

↳ Sem espaços  
↳ Usadas por convenção  
↳ Sem palavras reservadas  
↳ Nunca utilizar cláusulas

#### ② Conteúdo:

↳ Um ou mais métodos { }

```
class Empregado {  
    String nome;  
    public String ApresentarNome(){  
        return Nome;  
    }  
}
```

### 6.2 → PALAVRAS RESERVADAS

abstract	final	protected
boolean	finally	public
break	float	return
byte	for	short
case	if	static
catch	implements	super
char	import	switch
class	instanceof	Synchronized
const	int	this
Continue	interface	throw
default	long	throws
do	native	transient
double	new	true
else	null	try
extends	package	void
false	private	while

### 6.3 → Classes: Campos na Classe

- ① Campos/ATRIBUTOS devem ser DECLARADOS dentro do CORPO da classe.
- ② CADA CAMPO/ATRIBUTO DEVE ser um TIPO DE DADO.
- ③ CAMPOS/ATRIBUTOS de uma classe PODEM ser REFERENCIAS a INSTÂNCIAS de OUTRAS CLASSES.
- ④ Campos = TIPOS PONTUAIS (variáveis)

## 6.4 → DECLARAÇÃO DE CAMPOS

[TIPO-DE-DADO] (+) [NAME-DO-OS-CAMPOS / ATRIBUTOS]

- ⑩ Para cada uma das 32 variáveis criadas em campo na classe.  
32 DADOS = 32 CAMPOS
- ⑪ Campos podem ser referentes a uma Classe.  
↳ Exemplo no topo da classe

### 7 TIPOS PRIMITIVOS

Tipo	Faixa de Valores	Armazenamento
byte	-128 a 127	Inteiro de 8 bits
short	-32.768 a 32.767	Inteiro de 16 bits
int	-2.147.483.648 até 2.147.483.647	Inteiro de 32 bits
long	-9.223.372.036.854.775.808 a 9.223.372.036.854.775.807	Inteiro de 64 bits
float	1.401298464324811707e-45 a 3.40282346638528860e+38	Ponto Flutuante de 32 bits
double	4.94065645841246544e-324 a 1.79769313486231570e+308	8 bytes ou 64 bits
char	Representam um único caractere	16 bits
boolean	True ou False	1 bit

[String] é uma Classe.

- ⑫ A Classe String é usada para representar caracteres no caractere.  
(As 32 variáveis MATHS, somo instâncias da classe String)

## ⑧ Operaciones Aritméticas

- (+) → SUMA
- (-) → RESTA
- (\*) → MULTIPLICACIÓN

- (/) → DIVISIÓN
- (%) → RESTO DE UNA DIVISIÓN
- (INT(A))  
(INT(B)) → Parte Entera De Una División

(+) → Operación Con INSTRUCCIONES DE CLASE STRING (concatenación)

## ⑨ Conversões Entre TIPOS numéricos

### 9.1 - Conversões Entre TIPOS numéricos

↳ Operações numéricas entre TIPOS DIFERENTES, considerando como se passam INSTRUÇÕES COM AS PREMISSAS DO TIPO maior.

maior

int x START  
int / int  
int  
int

Byte > SHORT > int > long > float > double

### 9.2 → Conversão Garicuta

↳ Ns seguimos a regra anterior

[double x = 9.987;  
int yx = (int) x;  
, yx = 9] } Sintaxe II  
Conversão  
Garicuta

## 10 MÉTODOS DE CLASSES

① Classes = DADOS + OPERAÇÕES

② OPERAÇÕES = MÉTODOS

✗ MÉTODO? MÉTODO? → ERRO

✗ MÉTODOS CLASSE? → ERRO

④ MÉTODOS (TIPO PRIMITIVO Param)

MÉTODOS (TIPO DA CLASSE Param)

```
class DataSimples{  
    byte dia, mes;  
    short ano;  
}  
  
void iniDataSimples(byte d, byte m, byte, short a){  
    if(dataValida(d,m,a) == true){  
        dia = d;  
        mes = m;  
        ano = a;  
    }else{  
        dia = 0;  
        mes = 0;  
        ano = 0;  
    }  
}  
  
boolean dataValida(byte d, byte m, short a){  
    boolean validade = false;  
    if((d>=1)&&(d<=31) && (m>=1) && (m<=12)){  
        return true;  
    }else{  
        return false;  
    }  
}  
  
boolean igual(DataSimples data1){  
    if((dia == data1.dia) && (mes == data1.mes) && (ano == data1.ano)){  
        return true;  
    }else{  
        return false;  
    }  
}  
  
void mostraDataSimples(){  
    System.out.println(dia + "/" + mes + "/" + ano);  
}
```

TIPOS PRIMITIVOS  
✓ ATRIBUTOS (variáveis)

PARÂMETROS  
BASEADOS EM TIPOS PRIMITIVOS

PARÂMETROS PRIMITIVOS  
EM UM OBJETO DA PRÓPRIA CLASSE.

## Exercícios Lista 2 → Classes em Java

### [1] → Classe Livro (Lista 1)

Escreva uma classe “Livro” que represente o modelo desenvolvido na lista 1.

LIVRO
ISBN      EDITORA
TÍTULO    GÊNERO
AUTOR
INICIAR LIVRO ()
SET & GET ()

```
public class Book
{
    // instance variables
    private int isbn;
    private String titulo, edicao, autor, editora;

    // Constructor for objects of class Book
    public Book(int is, String t, String edic, String a, String edit)
    {
        // initialise instance variables
        this.isbn = is;
        this.titulo = t;
        this.edicao = edic;
        this.editora = edit;
        this.autor = a;
    } This returns the object to the class?

    // An example of a method - replace this comment with your own

    public void setisbn(int is){
        this.isbn = is;
    }
    public int getisbn(){
        return isbn;
    }

    public void setTitulo(String t){
        this.titulo = t;
    }
    public String getTitulo(){
        return titulo;
    }

    public void setAutor(String a){
        this.autor = a;
    }
    public String getAutor(){
        return autor;
    }

    public void setEditora(String edit){
        this.editora = edit;
    }
    public String getEditora(){
        return editora;
    }

    public void setEdicao(String edic){
        this.edicao = edic;
    }
    public String getEdicao(){
        return edicao;
    }

    public void mostraDados(){
        System.out.println("ISBN = "+isbn);
        System.out.println("Título = "+titulo);
        System.out.println("Autor = "+autor);
        System.out.println("Editora = "+ editora);
        System.out.println("Edição = "+ edicao);
    }
}
```

*Constructor*

*Setters & Getters*

*Print vars.*

## [2] → Classe Livro Livraria

Escreva uma classe "LivroLivraria" que represente o modelo desenvolvido na lista 1.

LIVRO LIVRARIA	
ISBN	EDITORA preco
TITULO	GÊNERO QUANTIDADE
AUTOR	PAGINAS
Inclui: ISBN() Autor() Gênero() Páginas()	

### ① Instances Variables

```
private int ISBN, PAGINAS, QUANTIDADE;
float preco;
String TITULO, AUTOR, EDITORA, GÊNERO;
```

### ② Construtor

```
public BookStore(int id, String Titulo, String autor, String edit, String gênero,
                 int pag, float p) {
    this.ISBN = id;
    this.Paginas = p;
}
```

### ③ Setters and Getters

```
public void setISBN(int Is) {
    this.ISBN = Is;
}

public int getISBN() {
    return ISBN;
}
```

### ④ Show DATA

```
public void Show DATA() {
    S.O.PLN("ISBN = " + ISBN);
}
```

## [3] → Classe Livro Biblioteca

LIVRO BIBLIOTECA	
ISBN	EDITORA DATA ALUGUEL QUANTIDADE
TITULO	GÊNERO DATA DEVOLUÇÃO USO
AUTOR	MULTA VERSÃO DA MULTA
Inclui: ISBN() Aluguel() Devolução()	

### ① Instance Variables

DA, MA, TA DD, HD, AD

```
private int ISBN, Paginas, DATAALUGUEL, DATADEVOLUÇÃO, QTDS;
private float multa, valorDaMulta;
private String TITULO, AUTOR, EDITORA, GÊNERO, USO.
```

### ② Construtor

on multa = 0

```
public void BookLibrary(int —, float —, String —) {
```

This.ISBN = Is;

### ④ Show DATA

```
public void Show DATA() {
    S.O.PLN("ISBN = " + ISBN);
```

This.GÊNERO = GÊNERO;  
MULTA = 0;

### ③ Setters and Getters

```
public void setISBN(int Is) {
    This.ISBN = Is;
}

public int getISBN() {
    return ISBN;
}
```

## ⑤ Euros

Public vars euros (Strg usr, int d, Int m, int a) {

    This. UsrNO =usr;

    DA = D;

    MA = M;

    AA = A;

    DD = D + 8;

    MD = M;

    AD = A;

## ⑥ Devolvi

Public vars devolvi (int d, int m, int a) {

    ↓ Int DifC = 0; DifM = 0, difA = 0;

    DifA = A - AD;

    DifM = M - MD;

    DifD = D - DD;

    IF (DifD > 0) {

        1    MULTA = DifD \* 2;

    } IF (DifM > 0) {

        3    MULTA = MULTA + (DifM \* 31 \* 2);

    } IF (DifA > 0) {

        3    MULTA = MULTA + (DifA \* 360 \* 2);

    } IF ((DifA > 0) || (DifM > 0) || (DifD > 0)) {

        1    S.O. PLN ("Su multa es RD " + multa);

    } S.O. PLN ("Umo Devolvi");  
    UsrNO = " ";

    3

## 4) → Arquivos com Instâncias das Classes

1. → 2 TIPOS DE ENTRADA PODEM SER:

1.1 → BufferedReader import java.io.\*;

↳ CADA OBGETO DESTA CLASSE NÃO POSSUI.

1.2 → Scanner import java.util.Scanner;

↳ CADA OBGETO DESTA CLASSE POSSUI.

### APLICAÇÃO Buffer Reader → LIB = JAVA.ID.

↳ Class BookControl

↳ PUBLIC STATIC VOID main (String args[]) throws java.io.IOException

↳ VARIABLES

→ STRING (aux, titulo, auto, corona, esq, usura);

→ INT (ISBN, DA, MA, AA, DD, MD, AD, PAGINAS, QUANTIDADE,  
QTDVENDA);

→ FLOAT (altura, preco);

→ OBGETO P/ ENTRADA DO TECLADO (BRD).

INSTÂNCIA

→ S.O. PLN ("informe");

→ AUX = OB(BRD). READLINE();

↳ CONVERSÃO

→ manAux = INTEGER. VALUEOF (aux). intValue();

P/ MÚLTOS RECORRER  
DO TECLADO.

STLS

→ URL = OB5.RL0

→ OBGETO LIVRO → MOSTRAR DADOS ()

↳ ENCONTRAR()

→ OBGETO LIVRO(LIVRO) → QTD (QUANT)

→ OBGETO LIVRO BIBLIOTECA → EMPRESTIMO ()

→ DEPOIS

→ MOSTRAR DADOS ()

### APLICAÇÃO Scanner → [OB]

↳ nextLine() / nextInt() / nextFloat() ...

↳ [BUG] → MUNDO → STRING

↳ PEGAR COM STRING COM VAM AUX E DEPOIS  
PAR CONVERSÃO.

## (11) ESCOPO DE VARIÁVEIS E OBSTÉCOS

- Escopo dos campos e variáveis = visibilidade
- Campos de uma classe = Torna a classe (ATE com declaração) depois dos métodos)
- Variáveis e instruções (obs) de chamadas dentro dos métodos só serão vistos dentro desse método.
- Variáveis e objetos dentro de métodos e blocos de comando devem ser declarados antes de serem utilizadas.
- Variáveis (argumentos de retorno) = variável dentro do método.

## (12) MÉTODOS DE ACESSO

MDA → ATIBUTOS (variáveis)

- MDA pode ser usado tanto em campos como em métodos de uma classe.

- Objetivo = proteger a integridade e a consistência dos dados e operações da sua classe manipula.

[Ex] → Procedimentos (métodos)

classes → DataMembers

- ↳ Consistência retorno **dataMember**.
- ↳ Sem modulação, sem garantia da data variável.
- ↳ Campos podem ser acessados diretamente sem a utilização dos métodos da classe!

• Resposta: modulação p/ proteger dados

↳ Acesso na API ao campo e não a uma função que retorna o valor.

[Ex] → Proteger os métodos (sigos)



Obs → Claro → Existe implementação Interface sem Implementação

④ Resposta: montarão p/ restrição o acesso ao método Implementação.

12.1 → Tipos de modificadores de acesso

**PUBLIC** → Campo/método acessível/exclusivo = Querer outra classe.

**PRIVATE** → Acessíveis/montáveis/executáveis = Métodos da propria classe.  
→ Ocultos = Instâncias da classe + classes herdadas/derivadas.

**PROTECTED** → = PRIVATE  
→ Exág: classes herdadas/derivadas possuem acesso.

**PACKAGE** → Campos/métodos podem ser declarados sem modificadores.  
→ Campos/métodos = Tornar os membros do pacote.

④ Em classes = private com set / get ?  
Política de ocultos com acesso à pacote + métodos internos

Regras de Implementação

- Campos = PRIVATE ou PROTECTED - Classes as implementações em pacotes para omitir.
- Métodos accessíveis = PUBLIC.
- Métodos p/ dentro do campo = PUBLIC.
- Se reservar, métodos = PRIVATE.

# Ex) → ESCOPO E ACESSO

```

public class DataSimples{
    private byte dia, mes;
    private short ano;

    public void initDataSimples(byte d, byte m, byte, short a){
        if(dataValida(d,m,a)==true){
            dia = d;
            mes = m;
            ano = a;
        }else{
            dia = 0;
            mes = 0;
            ano = 0;
        }
    }

    private boolean dataValida(byte d, byte m, short a){
        boolean validade = false;
        if((d>=1)&&(d<=31) && (m>=1) && (m<=12)){
            return true;
        }else{
            return false;
        }
    }

    public boolean igual(DataSimples data1){
        if((dia == data1.dia) && (mes == data1.mes) && (ano == data1.ano)){
            return true;
        }else{
            return false;
        }
    }

    protected void mostraDataSimples(){
        System.out.println(dia + "/" + mes + "/" + ano);
    }
}

```

→ PUBLIC → Classe pode ser instanciada dentro de outra classe.  
 → PRIVATE → Dados só podem ser manipulados dentro da própria classe.  
 → VARIÁVEL GLOBAL (ATTRIBUTOS)  
 ↳ Declaração fora do método.  
 → PARÂMETROS (ARGUMENTOS)  
 ↳ São parâmetros manipulados dentro do método.  
 → MÉTODO ACESSO = OUTROS MÉTODOS DA PRÓPRIA CLASSE.  
 ↳ VALIDADE = VARIÁVEL LOCAL (MANIP → MÉTODOS)  
 → PROTEGIDO → Acesso = MÉTODOS DA CLASSE ou classes herigadas / derivadas.

## (13) Programas Passados em Origem → Objetos

### \* Composição

- ↳ CONSUMIDORES DE OBJETOS
  - ↳ Representar os objetos negócios locais;
  - ↳ Representar estruturas p/ a manipulação de dados.
- OBJETOS DO PROGRAMA (consumidores) = TROCA DE MENSAGENS.
- ↳ PONTO DE ENTRADA DO PROGRAMA

```

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

```

- ↳ JVM (Java Virtual Machine) Executa.
- ↳ PONTO DE ENTRADA.

## 14) Atribuições em Java

Public static void main(String args[])

**PUBLIC** → Visível a outras outras classes.

**STATIC** → Dispõe a evaçõe de objetos.

**MAIN** → Comunicação com a [JVM] para o ponto de entrada da aplicação.

String args[]

em  
String[] args

→ Parâmetros passados para Aplicação via linha de comando.  
→ ARGS é o IDOMÍNIO.

[Ex] → Aplicação em Java

```
public class Aplicação{
    private Ponto p1,p2;
    public void moverPontos(){
        p1.mover(4F,2F);
        p2.mover(-2F,-4F);
    }
    public void CriarPontos(){
        p1 = new Ponto();
        p2 = new Ponto();
    }
    public void rodar(){
        CriarPontos();
        moverPontos();
    }
    public static void main(String args[]){
        Aplicação ap = new Aplicação();
        ap.rodar();
    }
    public class Ponto{
        private float x,y;
    }
    public void mover(float newX, float newY){
        x = newX;
        y = newY;
    }
}
```

→ Declaração de objetos da classe Ponto como atributos.

→ Utilização do método [mover()] da classe [Ponto].

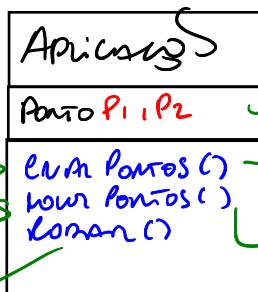
→ Instância de um [objeto] da classe [Ponto].

→ [Método Principal] executado pela Jvm.

→ Objeto da própria classe [Aplicação]

→ Declaração das instâncias de tipo Punto privativo da classe [Ponto]

→ Método da classe Ponto.



p<sub>1</sub> → {x, y}  
p<sub>2</sub> → {x, y}

class Ponto  
[NEW]

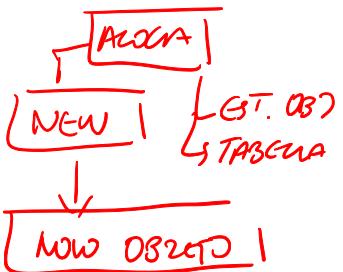
class Ponto → método mover()

sub-regras  
criarPontos  
moverPontos



## 15 Operador NEW

- ④ NEW = Instanciar um novo OBGETO.
- ⑤ Alocar Objetos = Alocar espaço no OBGETO  
+  
TARGET P/ INICIAR O LARO DO  
CONSTRUTOR NECESSARIO P/ GETAR  
SUS ORGANIGOS.
- ⑥ NEW EXCLUI AS TAREFAS DE INICIAR CONSTRUTOR P/ SER CONSTRUTOR.
- ⑦ NEW "Retorno"  $\Rightarrow$  IDENTIFICADOR DO OBGETO CRIADO. new retorno
- ⑧ Operador DE ATRIBUICAO = COVAR ENTRADA P/ CONSTRUTOR.  
 $\hookrightarrow$  VARIAVEL → REFERENCIA A OBGETOS.  
||  
IDENTIFICADOR DO OBGETO
- ⑨ CONSTRUTOR + JVM VORILHAT  $\rightarrow$  TIPICO DA MATERIA  
+  
TIPICO DA REFERENCIA



## 16 Garbage Collection

- ⑩ Gerenciamento de memoria para armazenar dados.
- ⑪ Thread Chama o Automatic Garbage Collector.
- ⑫ Retorno = memoria gerenciada p/ OBGETOS que não estão mais SENDO usados.
- ⑬ Monitoriza a lista de OBGETOS ATIVOS DO NEW.
- ⑭ Contador de Referencia P/ cada OBGETO
- ⑮ Ligam a memoria quando o contador chegar a zero.

## 17 REFERÊNCIA NULA

- \*) OBJETOS → ARMazenam uma REFERÊNCIA NULA ATÉ SEREM INICIALIZADOS.
- \*) REFERÊNCIA NULA = Ponteiro [NULL].
- \*) Desfazer REFERÊNCIA P/ OBJETO, ATRIBUINDO PONTUA [NULL]
  - ↳ OBJETO = NULL;

## 18 PRIMITIVAS & REFERÊNCIAS

- \*) VARIÁVEIS DE TIPOS PRIMITIVOS ARMazenam UM VALOR;
  - ↳ DECLARAÇÃO = Aloca-se o espaço na memória SEGUINTE ao armazenamento da variável.
- \*) VARIÁVEIS DE TIPOS REFERENCIAIS (OBJETO) ARMazenam IDENTIFICADORES para OBJETOS.
  - ↳ DECLARAÇÃO = Aloca-se espaço para a REFERÊNCIA AO OBJETO.
  - ↳ A Alocação do OBJETO é realizada SEMPRE DO DENTRO NEW.

## 19 CONSTRUTORES

- ④ ATRAVÉS = CRIA MÚLTIPLOS INSTÂNCIAS DE CLASSES
- ④ NEW = CRIA UMA INSTÂNCIA DA CLASSE (OBJETO)
- ④ MÉTODOS PARA INICIAR OS EXEMPLOS (CONSTRUTOR)
  - ↳ DECLARAÇÃO = ADICIONA SEUS PARÂMETROS PARA A RETORNO DA OBSTACO.
- ④ POSSUEM CRIAR UMA INSTÂNCIA DE CLASSE SEM INICIARLA.

### CONSTRUTOR

- ↳ TIPO ESPECIAIS DE MEMBROS DA CLASSE (MÉTODOS)
- ↳ CHAMAM AUTOMATICAMENTE QUANDO CRIAMOS EXEMPLOS USANDO A PALAVRA NEW.

### CONSTRUTOR CRIADORES

- ↳ INICIARIZAM OS ATRIBUTOS DA SUA CLASSE;
- ↳ RETORNAM NOVAS EXEMPLOAS DE INICIARIZAÇÃO
- ↳ RETORNAM INICIARIZADOS SEGUNDO OS PARÂMETROS PASSADOS NO MOMENTO DA CRIAÇÃO DO OBJETO.

### DECLARAÇÕES

↳ MESMO NOME DA CLASSE (CAPITALIZADO)

↳ NÃO POSSUEM TIPO DE RETORNO.

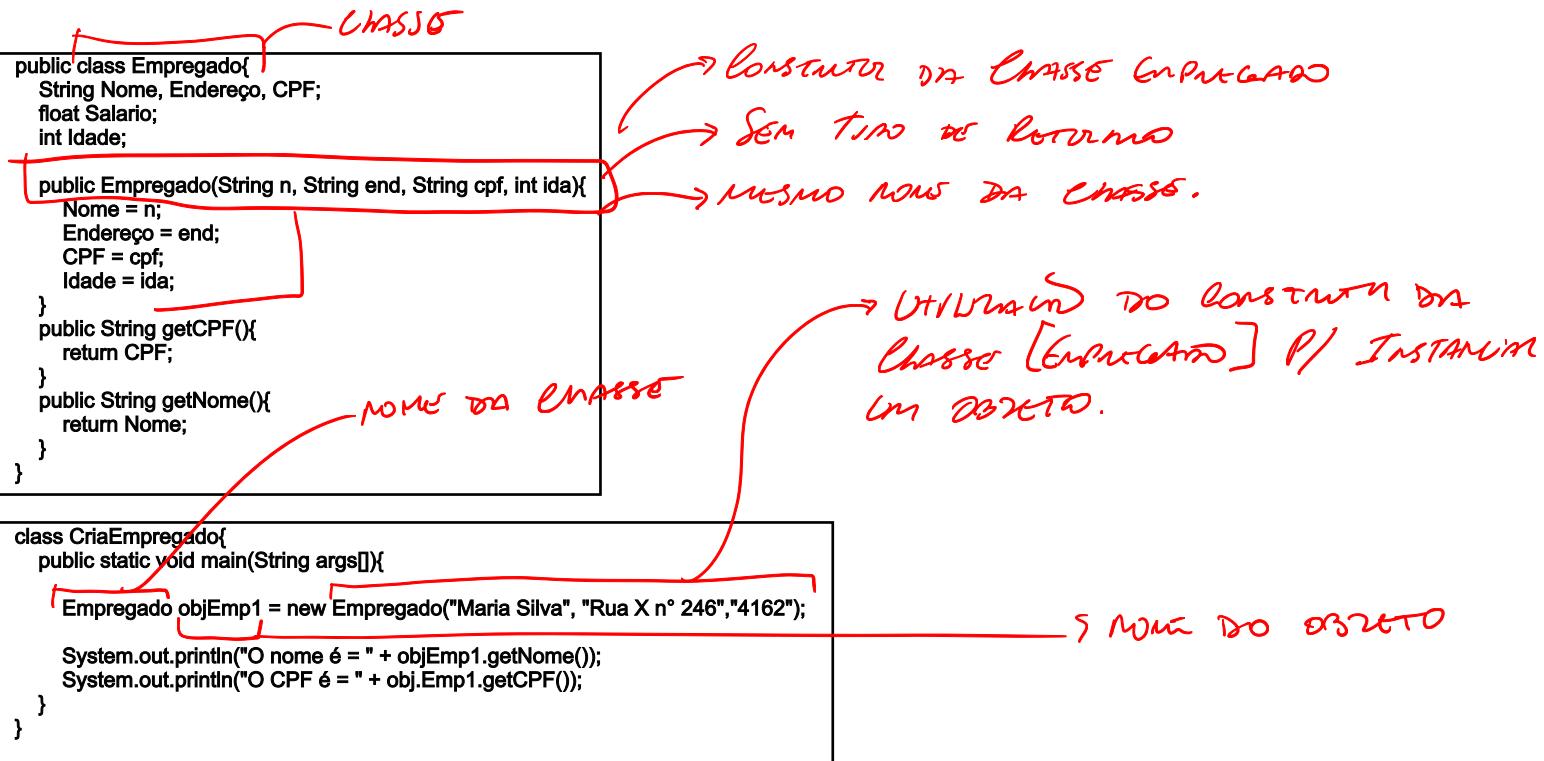
↳ MÉTODO DE ACESSO (DEFAULTE) = PUBLIC

↳ PODEM SER CHAMADAS ANÔNIMAMENTE

### FORMA CLASSE POSSUI UM CONSTRUTOR;

- ④ CONSTRUTOR Vazio = VERSÃO IMPLÍCITA PARA CLASSE SEM LIGAÇÕES COM CONSTRUTOR COM PARÂMETROS (VERSÃO EXPLICATIVA)
  - ↳ SE DIFERENTES OS 2 TIPOS, ESCREVEROS EXPLICITAMENTE OS 2.

## [Ex] → Construtores



## (20) Saida de Dados Básicos

④ **Saida = Atributo estatico da classe [System].**

↳ `System.out.println(...)`

⑤ **TIPOS PRIMITIVOS = Concatenação com strings com Op (+)**

## [Ex] → Saida de Dados

```
class saidaDados{  
    public static void main(String args[]){  
        System.out.println("Saida de dados");  
    }  
}
```

## 21 Entrada de dados Ponto

\* Tom classe que for ser utilizada para Entrada de dados via teclado deve conter o PACKAGE JAVA.io.

- ↳ PACOTE com classes que tratam de Entrada de dados.
- ↳ Import JAVA.io.\*

\* Objetos usados para tecendo em uma classe Java

- ↳ Criar objeto da classe BufferedReader.
- ↳ BufferedReader bf = new BufferedReader(new InputStreamReader(System.in));

\* Entrada Sobre a Entrada = Exceção

- ↳ TRATAMENTO → Sairia da A exceção não sera tratada localmente.

- ↳ Throws java.io.IOException.

## 22 Conversão String P/ TIPOS PRIMITIVOS

④ Entrada de DADOS = String (strut)

④ Entrada Vários numeros = Comunicação String Númerico.

**String → Int** → int i = Integer.valueOf(line).intValue();  
↳ Variável tipo String

**String → long** → long l = Long.valueOf(line).longValue();

**String → float** → float f = Float.valueOf(line).floatValue();

**String → double** → double d = Double.valueOf(line).doubleValue();

[Ex] → Entrada e Saída de Dados

```
import java.io.*;  
  
class soma{  
    public static void main(String args[]) throws java.io.IOException{  
        String aux;  
        int a, b, soma;  
        ↳ Entrada = String.  
        BufferedReader objTeclado = new BufferedReader(new InputStreamReader(System.in));  
        System.out.println("Digite o primeiro numero: ");  
        aux = objTeclado.readLine();  
        a = Integer.valueOf(aux).intValue();  
        a = a + b;  
        System.out.println("Digite o segundo numero: ");  
        aux = objTeclado.readLine();  
        b = Integer.valueOf(aux).intValue();  
        soma = a + b;  
        System.out.println("O Resultado de " + a + " + " + b + " = " + soma);  
    }  
}
```

→ INPUT US com classe de Entrada de DADOS.  
→ IMPRESSÃO NO LOCAL OS ERROS DE ENTRADA DO USUÁRIO.  
→ Métodos do objeto leitura na classe BufferedReader PARA ENTRADA DE DADOS UMA TECMO RECEBIMENTO EM UMA VARIAVEL AUXILIAR DE UM DADO EM FORMA STRING.  
→ Conversão de String P/ Inteiro.

## 23 Entrada de Dados com Scanner

```
import java.util.Scanner;  
  
public class MeuPrograma{  
    public static void main(String args[]){  
        int i;  
  
        Scanner objTeclado = new Scanner(System.in);  
        i = objTeclado.nextInt();  
  
        System.out.println("Valor digitado = " + i);  
    } }
```

Lis → Java. UTIL → Classe Scanner

Objeto da classe Scanner

Informa de onde vem os dados [System.in]

||

Usamos o objeto [objTeclado] para fazer a leitura do teclado.

teclado.

Leitura de um novo Inteiro, na forma decimal.

\* BUG → Leitura inválida, demais, leitura String (usar var (aux) p/ conversão)

## Entradas do Scanner

↳ nextByte()

↳ nextFloat()

↳ nextShort()

↳ nextDouble()

↳ nextInt()

↳ nextBoolean()

↳ nextLong()

↳ nextLine() → String

## 24 Operadores Relacionais e Lógicos

### Op. Relacionais

[<] menor → [<=] menor/igual

[>] maior → [>=] maior/igual

[==] igual → [!=] diferente

### Op. Lógicos

[&&] E lógico

[||] ou lógico

[!] não lógico

### Op. Relacionais String

↳ Concatenação [+]

## (25) ESTRUTURA DECISÃO E CONTROLE - COMANDOS

IF → ELSE

```
if (expressão booleana){  
    bloco de comandos do if;  
}else{  
    bloco de comandos do else;  
}
```

→ ESTRUTURA DO COMANDO

↳ ELSE = OPERADOR

↳ IF'S ANINHADOS → ELSE = IF + PROX.

↳ USAM 2 CHAVES {} P/ DELIMITAR O BLOCO.

↳ OPERADOR DE COMPARAÇÃO → Término → [= =]

↳ Bloco de comandos pode ser 1 comando (;) em Várias ({}3)

↳ Parênteses garantirão MA (EXPRESSÃO - Booleanos)

SWITCH

```
switch(variavelDeControle){  
    case constante1:  
        bloco1;  
        break;  
    case constante 2:  
        bloco2;  
        break;  
    default:  
        bloco3;  
        break;  
}
```

→ ESTRUTURA

↳ Serve para um tipo de estrutura de alternativas

↳ Variável de Controle = INT / CHAR / STRING

↳ CASE = Ponto de Entrada da Estrutura.

↳ BREAK = Um bloco sem execução sempre.  
↳ Ignora os outros CASE'S.

↳ DEFAULT = Operador.

## (26) ESTRUTURAS DE REPETIÇÃO

WHILE

```
while(expressão booleana){  
    bloco de comandos;  
}
```

→ ESTRUTURA

↳ uso parenteses em (expressão Booleana).

↳ Loop = Expressão Booleana (verdadeira)

↳ Erro: NÃO ATÉ UM CONTROLE LOOP = Loop do

↳ CHAMOS ? Bloco de comandos ?

DO-WHILE

```
do{  
    bloco de comandos;  
}while(expressão booleana);
```

→ ESTRUTURA.

↳ Semelhante while ≠ enquanto loop testava depois da  
 (já execução) do bloco.

↳ Bloco executado pelo menos 1º vez.

↳ Mesmas considerações do while.

FOR

```
for( inicialização; terminação; iteração){  
    bloco de comandos;  
}
```

→ ESTRUTURA

↳ Loop = Executar as 3 partes de uma ITERAÇÃO.

inicialização → int i=0;

terminação → i <= 10;

Iteração → i++;

# [Ex] → Estruturas de Decisão / Controle + Repetição

```
public class exemploCondicaoRepeticao{  
    public static void main(String args[]){  
        double valor = 1;  
        char letra = 'A';  
        int contador;  
  
        while(valor<=20){  
            System.out.println(valor);  
            valor = valor * 2;  
            if(valor >=20){  
                System.out.println("Fim da exemplificação do while e if");  
            }  
        }  
  
        for(contador = 0; contador < 10; contador++){  
            System.out.println(contador);  
        }  
        System.out.println("Fim da explicação do FOR");  
  
        do{  
            switch(letra){  
                case 'A':  
                    System.out.println(letra + "é vogal");  
                    break;  
                case 'E':  
                    System.out.println(letra + "é vogal");  
                    break;  
                case 'I':  
                    System.out.println(letra + "é vogal");  
                    break;  
                case 'O':  
                    System.out.println(letra + "é vogal");  
                    break;  
                case 'U':  
                    System.out.println(letra + "é vogal");  
                    break;  
            }  
            letra++;  
        }while( letra!= 'Z');  
        System.out.println("fim da explicação do switch e do-while");  
    }  
}
```

## 27) Sobrecarga

- ① MÉTODO COM NOME IGUAL A OUTRO MÉTODO NA CLASSE.
- ② USANDO MESMO NOME DE 2 OU + FUNÇÕES DE SE CONTRIBUIR A MESMA SANTÍSSIMA = MÉTODOS SOBRECARGADOS
- ③ MÉTODOS COM NOMES IGUAIS → ASSINATURAS DIFERENTES.
  - ↳ NOME + TIPOS DE ARGUMENTOS
- ④ ASSINATURA DIFERENTE = TIPOS DE PARÂMETROS DE CADA MÉTODO.
- ⑤ CONSTANTES = SOBRECARGA = PARÂMETROS DIFERENTES / MESMO NOME.
- ⑥ CONSTRUTOR SOBRECARGADO P/ OUTRO = REFERÊNCIA THIS

## Ex) → Sobrecarga

```
class criaEmpregado{  
    public static void main(String args[]){  
        Empregado objEmp1 = new Empregado("Maria","12321312", "rua das flores", 25,500);  
        System.out.println("A idade de maria é = " + objEmp1.Devolveldade());  
        System.out.println("A idade de maria daqui a 10 anos é = " + objEmp1.Devolveldade(10));  
        Empregado objEmp2 = new Empregado("Tadeu",700);  
        System.out.println("O salario de Tadeu é = " + objEmp2.salario);  
    }  
}
```

constroi Sobrecarga  
métodos sobrecarregados  
construir Sobrecarga

```
public class Empregado{  
    String nome, endereço, cpf;  
    float salario;  
    int idade;  
  
    public Empregado(String n, String end, String CPF, int idade, float sal){  
        this(n,sal);  
        endereço = end;  
        cpf = c;  
        idade = ida;  
    }  
    public Empregado (String n, float sal){  
        nome = n;  
        salario = sal;  
    }  
    public int Devolveldade(){  
        return idade;  
    }  
    public int Devolveldade(int numAnos){  
        return idade+numAnos  
    }  
}
```

↳ ASSINATURA = Empregado + (S, S, S, I, F)  
↳ uso da referência [this] para chamar outro construtor.  
↳ Sobrecarga

## 23 A Referência THIS

① NA CHAMADA DOS MÉTODOS DE UMA CLASSE, A JVM PEGA IMPLICITAMENTE, COMO PARÂMETRO, UMA REFERÊNCIA AO OBJETO AO QUAL A MENSAGEM FOI ENVIADA.

② PARA RECUPERAR essa referência DE DENTRO DO MÉTODO DA CLASSE, USAR-SE A PALAVRA RESERVADA **THIS**

③ CONSTRUTORES SOBRECARREGADOS = USAR **THIS** PARA REFERENCIAR OUTRO CONSTRUTOR SOBRECARREGADO

↳ Nesse caso, this é uma referência ao objeto, para que essa seja o parâmetro para o construtor.

## \* Resumo THIS

↳ Referencia PROPRIO OBSTETO

↳ Passo DADOS e METODOS na Instancia de um OBSTETO.

↳ Passa o OBSTETO como PARÂMETRO.

### [Ex] → Referencia THIS

```
public class ExemploThis{  
    public static void main(String args[]){  
        Funcionario objFunc1 = new Funcionario("Maria",500);  
        Funcionario objFunc2 = new Funcionario("Joao",700);  
  
        objFunc1.salario = objFunc1.ajustaSalario(20);  
        objFunc2.salario = objFunc2.ajustaSalario(30);  
  
        System.out.println("O novo salario de " + objFunc1.nome + "é = " + objFunc1.salario);  
        System.out.println("O novo salario de " + objFunc2.nome + "é = " + objFunc2.salario);  
    }  
}
```

→ Classe / ARVGS Bndlpar para  
a execução da Referencia  
THIS

→ CUP dos OBSTETOS Classe [Funcionarios]

```
public class Funcionario{  
  
    String nome; ) CAMPOS DA CLASSE.  
    float salario;  
  
    public Funcionario(String n, float s){  
        nome = n;  
        salario = s;  
    }  
    public float ajustaSalario(float percentual){  
  
        float novoSal;  
        novoSal = this.salario + calculo(percentual,this); ) CONSTRUTOR  
  
        return novoSal;  
    }  
    public float calcular(float percentual, Funcionario objFunc){  
  
        float valorAcrescido;  
  
        valorAcrescido = objFunc.salario * percentual / 100;  
  
        return valorAcrescido;  
    }  
}
```

→ O THIS ESTA REFERENCIANDO  
UM OBSTETO DA ARVGS

→ USO DO [THIS] PARA ACESSAR  
UM ATRIBUTO DO OBSTETO E/OU  
NA ARVGS [objFunc]

→ USO DO [THIS] COMO PAR-  
METRO PARA REFERENCIAL O  
OBSTETO CONCRETO (ARVGS [objFunc])

# EXERCÍCIOS LSOA 3 → Construtores e Sobrecarga

Questão 1 → Construir classe Livro. Questão 2 → 2 construtores usando Sobrecarga e referenciar this para Livro Linharia.

Classe Livro

ISBN  
TÍTULO  
AUTOR  
EDICAO  
VALOR

Início/2º = Construir

```
public class Book {
    // instance variables - replace the example below with your own
    private int isbn;
    private String titulo, autor, edicao, editora;

    // Constructor for objects of class Book
    public Book(int is, String t, String a, String edic, String edit) {
        // initialise instance variables
        this.isbn = is;
        this.titulo = t;
        this.autor = a;
        this.edicao = edic;
        this.editora = edit;
    }

    public void showData() {
        System.out.println("ISBN = " + this.isbn);
        System.out.println("Título = " + this.titulo);
        System.out.println("Autor = " + this.autor);
        System.out.println("Editora = " + this.editora);
        System.out.println("Edicao = " + this.edicao);
    }

    //setter and getters

    // metodos específicos

    public void showData(){
        System.out.println("ISBN = " + this.isbn);
        System.out.println("Título = " + this.titulo);
        System.out.println("Autor = " + this.autor);
        System.out.println("Editora = " + this.editora);
        System.out.println("Edicao = " + this.edicao);
    }
}
```

} CONSTRUTOR com PARÂMETROS

Classe Livro Linharia

+ Classe Livro +  
PÁGINAS  
Preço  
Quantidade

```
public class BookStore {
    // instance variables - replace the example below with your own
    private int isbn, paginas, quantidade;
    private float preco;
    private String titulo, autor, edicao, editora;

    // Constructor for objects of class Book
    public BookStore(int is, String t, String a, String edic, String edit, int pag, int qtd, float p) {
        this(is, t, a, edic, edit);
        this.paginas = pag;
        this.quantidade = qtd;
        this.preco = p;
    }

    public BookStore(int is, String t, String a, String edic, String edit) {
        // initialise instance variables
        this.isbn = is;
        this.titulo = t;
        this.autor = a;
        this.edicao = edic;
        this.editora = edit;
    }

    //setter and getters

    // metodos específicos

    public void showData(){
        System.out.println("ISBN = " + this.isbn);
        System.out.println("Título = " + this.titulo);
        System.out.println("Autor = " + this.autor);
        System.out.println("Editora = " + this.editora);
        System.out.println("Edicao = " + this.edicao);
        System.out.println("Preço = " + this.preco);
        System.out.println("Quantidade = " + quantidade);
    }

    public void vender(int qtd){
        float value;

        if(qtd <= quantidade){
            quantidade = quantidade - qtd;
            value = qtd * preco;
            System.out.println("O valor de " + quantidade + " livros é = " + value);
        }else{
            System.out.println("Não tenho esta quantidade em estoque!");
        }
    }

    public void comprar(int qtd){
        quantidade = quantidade + qtd;
    }

    public void reajuste(float perct){
        preco = preco + ((preco * perct)/100);
    }
}
```

AS = BS + [I,S,S,S,F,J,F] → REFERÊNCIA A UM CONSTRUTO USANDO THIS

ASS = BS + [I,S,S,S,S] → CONSTRUTORES E SobreCARGA

## Questão 3 → Classe Professor

Classe Professor

Nome do Professor  
Nome do Departamento  
Data Admissão  
Número de Registro

Constructor()  
Setters() / Getters()  
ShowData()

```
public class Professor
{
    // instance variables - replace the example below with your own
    private byte dia, mes;
    private short ano;
    private int numRegistro;
    private String nomeProfessor, nomeDepartamento;

    /**
     * Constructor for objects of class Professor
     */
    public Professor(byte dia, byte mes, short ano, int numR, String np, String nd){
        this.dia = dia;
        this.mes = mes;
        this.ano = ano;
        this.numRegistro = numR;
        this.nomeProfessor = np;
        this.nomeDepartamento = nd;
    }

    // setters and getters

    // methods específicos
    private void showData(){
        System.out.println("Nome Professor = " + nomeProfessor);
        System.out.println("Nome Departamento = " + nomeDepartamento);
        System.out.println("Número do Registro = " + numRegistro);
        System.out.println("data Admissão = ");
    }
}
```

## Questão 4 → Classe Data

Classe Data

DIA  
MES  
ANO

IniData()  
Setters() / Getters()  
ShowData()

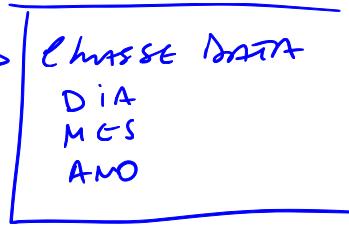
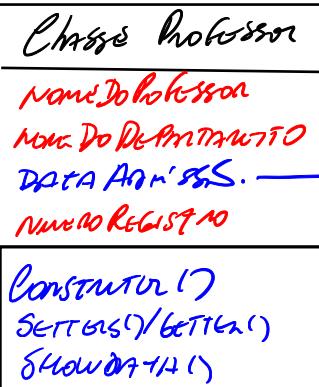
```
public class Data
{
    // instance variables - replace the example below with your own
    private byte dia, mes;
    private short ano;

    /**
     * Constructor for objects of class Data
     */
    public Data(byte d, byte m, short a){
        // initialise instance variables
        this.dia = d;
        this.mes = m;
        this.ano = a;
    }

    // An example of a method - replace this comment with your own

    public void setDia(byte d){
        this.dia = d;
    }
    public void setMes(byte m){
        this.mes = m;
    }
    public void setAno(short a){
        this.ano = a;
    }
    public byte getDia(){
        return dia;
    }
    public byte getMes(){
        return mes;
    }
    public short getAno(){
        return ano;
    }
    public void showDate(){
        System.out.println("Data = " + dia + "/" + mes + "/" + ano);
    }
}
```

Questão 5 → Classe Professor com DATA ADMISSÃO como OBRETO.



```

public class ProfessorData {
    // instance variables - replace the example below with your own
    private Data dataAdmissao;
    private int numRegistro;
    private String nomeProfessor, nomeDepartamento;

    /**
     * Constructor for objects of class Professor
     */
    public ProfessorData(int numR, String np, String nd, Data da) {
        this.dataAdmissao = da;
        this.numRegistro = numR;
        this.nomeProfessor = np;
        this.nomeDepartamento = nd;
    }

    //setter and getters

    // métodos específicos
    private void showData() {
        System.out.println("Nome Professor = " + nomeProfessor);
        System.out.println("Nome Departamento = " + nomeDepartamento);
        System.out.println("Número do Registro = " + numRegistro);
        dataAdmissao.showDate();
    }
}

```

Questão 6 → Fazia na Questão 4

Questão 7 → Aplicar o/instance 3 objetos da classe Professor e Imprimir os dados

```

import java.io.*;

public class AppProfessorBRD{
    public static void main(String args[]) throws java.io.IOException{
        String nome, departamento, aux;
        byte dia,mes;
        short ano;
        int numeroRegistro;

        //criação dos objetos da classe professor com valores nulos.
        ProfessorData objProf1 = null, objProf2 = null, objProf3 = null;

        // criação objeto do teclado
        BufferedReader objTeclado = new BufferedReader(new InputStreamReader(System.in));
        System.out.println("***** Informe os dados do Professor " + i + " *****");
        System.out.println("Digite o Nome : ");
        nome = objTeclado.readLine();
        System.out.println("Digite o Departamento : ");
        departamento = objTeclado.readLine();
        System.out.println("Digite Numero de registro : ");
        aux = objTeclado.readLine();
        numeroRegistro = Integer.valueOf(aux).intValue();
        System.out.println("Digite Dia da admissão : ");
        aux = objTeclado.readLine();
        dia = Byte.valueOf(aux).byteValue();
        System.out.println("Digite Mes da admissão : ");
        aux = objTeclado.readLine();
        mes = Byte.valueOf(aux).byteValue();
        System.out.println("Digite Ano da admissão : ");
        aux = objTeclado.readLine();
        ano = Short.valueOf(aux).shortValue();

        // criação de um objeto do tipo data, para add os valores da admissao
        Data admissao = new Data(dia,mes,ano);

        // criando os objetos dos professores passando os dados coletados pelo teclado.
        if(i == 1){
            objProf1 = new ProfessorData(nome, departamento, admissao, numeroRegistro);
        }else if(i == 2){
            objProf2 = new ProfessorData(nome, departamento, admissao, numeroRegistro);
        }else if(i == 3){
            objProf3 = new ProfessorData(nome, departamento, admissao, numeroRegistro);
        }

        System.out.println("***** Dados do prof. 1 *****");
        objProf1.showData();
        System.out.println("***** Dados do prof. 2 *****");
        objProf2.showData();
        System.out.println("***** Dados do prof. 3 *****");
        objProf3.showData();
    }
}

```

JAVAM SCANNER

→ Ler o BGG (1m).

# Exercícios Lista 4 – Estruturas Decisão e Repetição

Questão 1 → Classe Comparável

Classe Comparável
VALUE (INT)
É maior ou igual (P)
É menor ou igual (P)
É diferente de (P)
R → True / False

```
public class Comparavel {
    private int value;

    // Constructor for objects of class Comparavel
    public Comparavel(int v){
        this.value = v;
    }

    // setters and getters
    public void setValue(int v){
        this.value = v;
    }

    public int getValue(){
        return value;
    }

    // An example of a method - replace this comment with your own
    public Boolean maiorOuIgual(int v){
        if(v >= value){
            return true;
        }else{
            return false;
        }
    }

    public Boolean menorOuIgual(int v){
        if( v <= value){
            return true;
        }else{
            return false;
        }
    }

    public Boolean eDiferenteDe(int v){
        if(v != value){
            return true;
        }else{
            return false;
        }
    }
}
```

Questão 2 → Classe Calculadora

Classe Calculadora
VALOR 1 (DOUBLE)
VALOR 2 (DOUBLE)
SÍMBOLO (CHAR) [+/-/*/÷]
CONSTRUTOR (V1, V2, S)
CALCULAR() → $\frac{V_1}{V_2} = \text{RESULTADO}$
MOSTRAR()

```
public class Calculator {
    // instance variables - replace the example below with your own
    private double value1, value2, result;
    private char simbolo;
```

// Constructor for objects of class Calculator

```
public Calculator(double v1, double v2, char sim){
    // initialise instance variables
    this.value1 = v1;
    this.value2 = v2;
    this.simbolo = sim;
}

// setters and getters
public void setValue1(double v1){
    this.value1 = v1;
}
public void setValue2(double v2){
    this.value2 = v2;
}
public void setSymbolo(char sim){
    this.simbolo = sim;
}

public double getValue1(){
    return value1;
}

public double getValue2(){
    return value2;
}

public char getSymbolo(){
    return simbolo;
}
```

// An example of a method - replace this comment with your own

public void showData(){
 System.out.println("O resultado eh: " + result);
}

public void calculate(double n1, double n2, char sim){

```
switch(simbolo){
    case '+':
        result = value1 + value2;
        System.out.println(" Soma\n");
        break;
    case '-':
        result = value1 - value2;
        System.out.println(" Subtração\n");
        break;
    case '*':
        result = value1 * value2;
        System.out.println(" Multiplicação\n");
        break;
    case '/':
        if(value2 == 0){
            System.out.println("Não é possível dividir por 0");
        }else{
            result = value1/value2;
            System.out.println(" Divisão\n");
        }
        break;
    default:
        System.out.println("Este operador não é valido");
        return;
}
```

showData();

import java.util.Scanner;

```
public class CalcApp{
    public static void main(String args[]){
        Double n1, n2, result;
        char op;
        Scanner teclado = new Scanner(System.in);

        do{
            System.out.println("Digite valor 1: ");
            n1 = teclado.nextDouble();
            System.out.println("Digite valor 2: ");
            n2 = teclado.nextDouble();
            System.out.println("Digite a operação: ");
            teclado.nextLine();
            op = teclado.nextLine().charAt(0);
            Calculator calc = new Calculator(n1,n2,op);
            calc.calculate(n1,n2,op);
        }while(n1!= 0);
    }
}
```

APP

~~Classe~~

### Orientação 3 → Classe Livro + Metodos equals

Classe Livro
ISBN
TITULO
AUTOR
ENCOMA
ENCOMA
CONSTRUTOR()
SETORS() GETORS()
EQUALS.

```
public Boolean equals(String title){  
    if(titulo.equals(title) == true){  
        return true;  
    }else{  
        return false;  
    }  
}
```

### Orientação 4 → Classes Entrada de Cinema

Classe EntradaCinema
DATA DATE Movie
FLOAT TIME
ROOM
FLOAT VALOR
CONSTRUTOR (ALL)
CALL DESCONT () Sobrecarga
(CALL DESCONT TIME)
TO STRING ()

#### CALL DESCONT () [Sobrecarga]

- DATA MAXIMO LIGADA (DATA)  
< 12 → 50% DESCONT
- DATA MAXIMO + numero Student ID  
= 12 L=15 → 40% DESCONT
- = 16 L=20 → 30% DESCONT  
→ 20 → 20% DESCONT

#### CALL DESCONT TIME ()

- METO DEVE SER EXECUTADO APÓS TUTAS AS OUTRAS OPÇÕES.
- 10% SOBRE VALOR APÓS OUTROS DESCONTOS.

#### TO STRING ()

- ↳ IMPRIMIR DADOS DO INGRESSO.

```

public class TicketMove
{
    // instance variables - replace the example below with your own
    private Date dateMove;
    int room;
    float time, value;

    // Constructor for objects of class TicketMove
    public TicketMove(Date dm, int r, float t, float v){
        // initialise instance variables
        this.dateMove = dm;
        this.room = r;
        this.time = t;
        this.value = v;
    }

    // An example of a method - replace this comment with your own
    public void calcDescont(Date dbirth){
        if((dateMove.getYear() - dbirth.getYear()) <=12){
            value = value/2; USANDO GETTER
        }
    }

    public void calcDescont(Date dbirth, int numStudentID){ DE OUTRA CLASSE
        int age = dateMove.getYear() - dbirth.getYear();
        if((age > 12) && (age <=15)){ EXEMPLO DE ACESO SIGUENTE A METROS!
            value = value - ((40*value)/100);
        }else if((age >=16) && (age < 20)){
            value = value - ((30*value)/100);
        }else if(age >=20){
            value = value - ((20*value)/100);
        }
    }

    public void calcDescontTime(){
        if(time < 16){
            value = value - ((value*10)/100);
        }
    }

    public void showTicket(){
        System.out.println("Date of move: ");
        dateMove.showDate();
        System.out.println("Session time : " + time);
        System.out.println("Session Room : " + room);
        System.out.println("Session price : " + value);
    }

    //setters and getters
}

```

MOTIVO 5 → App P1 INPNMR Incisos

- 1) ELEGIRÉS NOVOS;
- 2) ELEGIRÉS L 12 ANOS;
- 3) ESTUDARES

```

import java.util.Scanner;
public class SellTicket
{
    public static void main(String args[]){
        float time, value;
        int room, studentID;
        byte day, month;
        short year;
        Date dateMove, dateBirth;
        TicketMove objTicketMove;

        Scanner keyboard = new Scanner(System.in);

        System.out.println("**** Type session data ****");
        System.out.println("Day of session: ");
        day = keyboard.nextByte();
        System.out.println("Month of session: ");
        month = keyboard.nextByte();
        System.out.println("Year of session: ");
        year = keyboard.nextByte();

        dateMove = new Date(day,month,year);

        System.out.println("Type session time: ");
        time = keyboard.nextFloat();
        System.out.println("Type session price: ");
        value = keyboard.nextFloat();
        System.out.println("Type session Room: ");
        room = keyboard.nextInt();

        System.out.println("Type day of birth: ");
        day = keyboard.nextByte();
        System.out.println("Type month of birth: ");
        month = keyboard.nextByte();
        System.out.println("Type year of birth: ");
        year = keyboard.nextShort();

        dateBirth = new Date(day,month,year);

        System.out.println("Type studentID number or 0 (normal)");
        studentID = keyboard.nextInt();

        // instancie the object created earlier
        objTicketMove = new TicketMove(dateMove, room, time, value);

        if(studentID == 0){
            objTicketMove.calcDescont(dateBirth,studentID);
        }else{
            objTicketMove.calcDescont(dateBirth);
        }

        objTicketMove.calcDescontTime();
        objTicketMove.showTicket();
    }
}

```

USO DA ENTRADA SIMPLES  
DATAS REPRESENTAMOS  
PONTOS SINTÁTICOS  
DIFERENTES

## Oussois 6 → Classe DATA + METODO retorna MES (P)

```

public class Date {
    // instance variables - replace the example below with your own
    private byte day, month;
    private short year;

    /**
     * Constructor for objects of class Date
     */
    public Date(byte d, byte m, short y) {
        // initialise instance variables
        if(dateisValid(d,m,y) == true){
            this.day = d;
            this.month = m;
            this.year = y;
        }else{
            day = 0;
            month = 0;
            year = 0;
        }
    }

    // An example of a method - replace this comment with your own
    public void showDate(){
        System.out.println("Date = " + day + "/" + month + "/" + year);
    }

    public Boolean dateisValid(byte d, byte m, short y){
        if((d>=1) && (d<=31) && (m>=1) && (m<=12)){
            return true;
        }else{
            return false;
        }
    }

    public boolean isEqual(Date auxDate){
        if(
            (this.day == auxDate.day) &&
            (this.month == auxDate.month) &&
            (this.year == auxDate.year)
        ){
            return true;
        }else{
            return false;
        }
    }

    public String returnMonth(){
        String result = null;
        switch(month){
            case 1: result = "Janeiro";
                break;
            case 2: result = "February";
                break;
            case 3: result = "March";
                break;
            case 4: result = "April";
                break;
            case 5: result = "May";
                break;
            case 6: result = "June";
                break;
            case 7: result = "July";
                break;
            case 8: result = "August";
                break;
            case 9: result = "September";
                break;
            case 10: result = "October";
                break;
            case 11: result = "November";
                break;
            case 12: result = "December";
                break;
        }
        return result;
    }

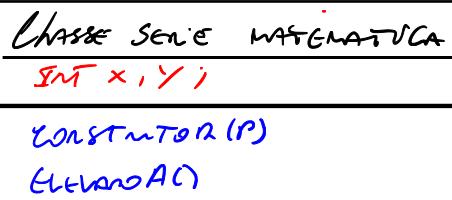
    // setters and getters
}

```

*(LATERAL DA CLASSE DATA, COLORIDA)*

*DATAS, COLORIR  
NOS OUTROS  
EXERCICIOS*

## Oussois 7 → SENSO MATEMÁTICAS



```

public class MathSeries {
    // instance variables - replace the example below with your own
    private int y;
    private int x;

    /**
     * Constructor for objects of class MathSeries
     */
    public MathSeries(int x, int y) {
        // initialise instance variables
        this.x = x;
        this.y = y;
    }

    public void showElevateTo(){
        System.out.println(elevateTo());
    }

    public float elevateTo(){
        int aux = 1, result;
        result = x;
        while(aux < this.y){
            result = result * x;
            aux++;
        }
        return result;
    }

    // An example of a method - replace this comment with your own
    public void setX(int x){
        this.x = x;
    }

    public void setY(int y){
        this.y = y;
    }

    public int getX(){
        return x;
    }

    public int getY(){
        return y;
    }
}

```

**Questão 8** → Classe Series Matemáticas {  $\underbrace{\frac{1}{1^2}}_2 + \underbrace{\frac{1}{3^2}}_2 + \underbrace{\frac{1}{5^2}}_2 + \underbrace{\frac{1}{7^2}}_2 + \underbrace{\frac{1}{9^2}}_2$  } → METODO PI QUADRADO SORTE

### METODO PI\_Square8

Denominador = 1

RESULT = 0

QTD (5)

Denominador = 3

RESULT = 0

$$\text{RESULT} = \text{RESULT} + \left( \frac{1}{\text{Denominador} * \text{Denominador}} \right) \rightarrow \text{RESULT} = \frac{1}{1^2}$$

$$\text{Denominador} = \text{Denominador} + 2 \rightarrow \text{Denominador} = 3$$

$$\text{RESULT} = \text{RESULT} + \frac{1}{\text{Denominador} * \text{Denominador}} \rightarrow \text{RESULT} = \frac{1}{3^2}$$

$$\rightarrow \text{Denominador} = 3 + 2 = 5$$

→ while (QTD > 0);

### public class MathSeries

```
// instance variables - replace the example below with your own
private int y;
private int x;

/**
 * Constructor for objects of class MathSeries
 */
public MathSeries(int x, int y)
{
    // initialise instance variables
    this.x = x;
    this.y = y;
}
public void showElevateTo(){
    System.out.println(elevateTo());
}
public float elevateTo(){
    int aux = 1, result;
    result = x;
    while(aux < this.y){
        result = result * x;
        aux++;
    }
    return result;
}
public float piSquare8(int qtd){
    float denominator = 1, result = 0;
    do{
        result = result + (1 / (denominator * denominator));
        denominator = denominator + 2;
        qtd--;
    }while(qtd > 0);
    return result;
}
public void setX(int x){
    this.x = x;
}
public void setY(int y){
    this.y = y;
}
public int getX(){
    return x;
}
public int getY(){
    return y;
}
```

**Questão 9** → Classe Series Matemáticas + METODO CALCULAR PI

### METODO CALCULAR PI

CONTAM = 2

RESULT = 2

NUMERADOR = 2

DENOMINADOR = 1

QTD

$$2 \times \left( \frac{2}{1} \right) \times \left( \frac{2}{3} \right) \times \left( \frac{4}{3} \right) \left( \frac{4}{5} \right)$$

$$\left( \frac{6}{5} \right) \times \left( \frac{6}{7} \right)$$

$$\text{RESULT} = \text{RESULT} + \frac{\text{Numerador}}{\text{Denominador}}$$

$$= 2 \times \frac{2}{1}$$

$$\text{Denominador} = 1 + 2 = 3$$

$$\text{Numerator} = 2 \times \frac{2}{1} \times \frac{\text{Numerador}}{3}$$

$$\text{Numerador} = \text{Numerador} + 2$$

```
public float calculatePi(int qtd){
    int cont = 1;
    float result = 2, numerator = 2, denominator = 1;
    while(cont < qtd){
        result = result * (numerator / denominator);
        System.out.println(numerator + "/" + denominator + "=" + result);
        cont++;
        denominator = denominator + 2;
        result = result * (numerator / denominator);
        System.out.println(numerator + "/" + denominator + "=" + result);
        cont++;
        numerator = numerator + 2;
    }
    return result;
}
```

## Oursso 10 → Classe Fibonacci

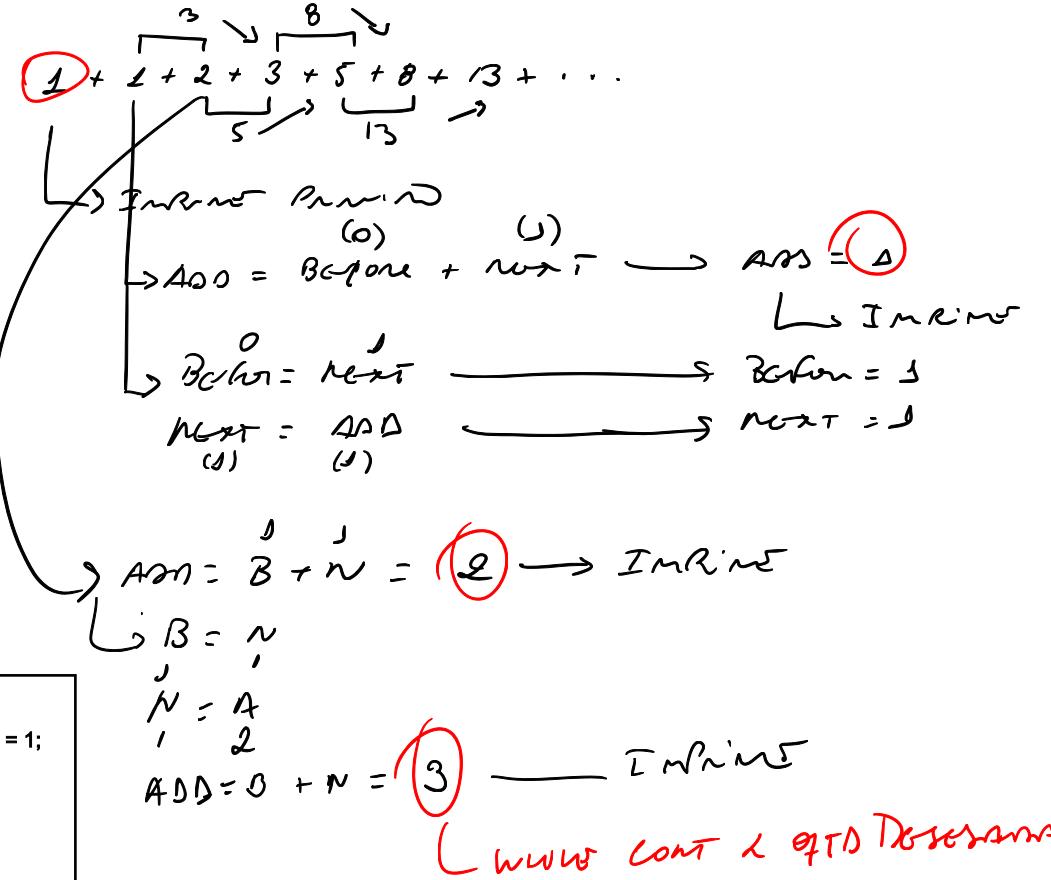
Classe Fibonacci

before = 0

next = 1

add

cont = 1



```
public Fibonacci(int qtd){
    // Initialise instance variables
    int before = 0, next = 1, add, cont = 1;
    System.out.println(next);
    do{
        add = before + next;
        System.out.println(add);
        before = next;
        next = add;
        cont++;
    }while(cont < qtd);
}
```

Boletim

## Oursso 11 → Classe Fibonacci + Método Retorna A Série (true)

Método findNumber (n)

before = 0, next = 1, add;

```
public boolean findNumber(int num){
    int before = 0, next = 1, add;
    do{
        add = before + next;
        before = next;
        next = add;
    }while(add < num);
    if(add == num){
        return true;
    }else{
        return false;
    }
}
```

- ⑦ → O loop tem ate ADD ser maior que o numero passado com paramentro.
- ⑧ → ADD é a variável de controle.
- ⑨ → Retorne a logica do Fibonacci P/ dentro do loop, se retorna a res.

Aussois 12 → Classe Primes + Méthode primeRange

⑥ Classe de méthodes sonrent.

↳ méthodes renvoyées (Primes) { ⑬ }  $\frac{6}{2} = \frac{3}{2} = 3.0$   $6 \% 2 = 0,5$   $6 \% 2 = 0^*$

$\frac{6}{2} = 3$   
 $6 \% 2 = 0$   $3 \% 2 = 1$   
 $6 \% 2 = 0,5$   
final result RMO.

Boolean ok = true;

I = 2 ; i < num ; i++

↳ soit  $\rightarrow$  num % i == 0  $\rightarrow$  S'ET RMO  
ok = false

```
public class Primes{  
    public Boolean Prime(int num){  
        boolean ok = true;  
        for(int i = 2; i < num; i++){  
            if((num % i)==0){  
                ok = false;  
            }  
        }  
        return ok;  
    }  
}
```

Aussois 13 → Classe Primes + Méthode primeRange(2 paramètres)

Méthode primeRange (start, end)

▷ start <= end → END

↳ Imprime toutes

S      end  
2 → 13  
  <= ⊗

2 → 13

  <=  $\rightarrow$  Prime(start) == true ?  
    → Prime(2) == true ? → ⑤  $\rightarrow$  start++ = 3

3 → 13  
  <=  $\rightarrow$  Prime(3) == true ?  $\rightarrow$  S → start++ = 4

4 → 13  
  <=  $\rightarrow$  Prime(4) == true ?

  ↳ N → start++  
    ③

start (start)  
↳ somme ou non  
utilisation P1  
RMO.

```
public void primeRange(int start, int end){  
    if(start <= end){  
        System.out.println("Invalid Range");  
    }else{  
        System.out.println("***** Prime's List at Range : " + start + " - " + end + " *****");  
        while(start <= end){  
            if(Prime(start) == true){  
                System.out.println(start);  
            }  
            start++;  
        }  
    }  
}
```

## 29 VETORES

- ④ **VETOR** → sequência de objetos ou valores de tipos diferentes, todos do mesmo tipo e combinados sob um único identificador.
- ④ **Vetores = Estáticos** → Tamanho definido no momento da execução.
- ④ **JAVA** → **VET = OBGETO** → Na Prática HERDAM DE **OBGETO**.
- ④ **Arrays** → **LENGTH** → Atributo público que informa o seu tamanho.
- ④ **Arrays** → Início na posição (**index**) 0.
- ④ **Declaração** → **tipo[] identificador;**  
**int[] VET;** → Declaração de um vetor de tipo inteiro  
**BUTTON[6];** → Declaração de um vetor de obgetto.
- ④ **Construção** → **Identificador = new tipo[TAMANHO];**  
**VET = new int[12];** → tipo RUNDO  
**b = new BUTTON[10]** → tipo de obgetto.
- ④ **Inicialização** → **int VET = new int {0,1,2,3,4,5,6,7,8,9,10,11}**  
(Declaração, construção e inicialização).

**[Ex] → Vetores**

```
class DiasDosMeses{  
    public static void main(String[] args){  
        int[] maxDiasMes = new int [12];  
        String[] nomeMes = {"JAN", "FEV", "MAR", "ABR", "MAI", "JUN", "JUL", "AGO", "SET", "OUT", "NOV", "DEZ"};  
        for(int i = 0; i < maxDiasMes.length; i++){  
            if( ( (i+1) < 8 ) && ((i+1)%2==1) ) || ( (i+1) >= 8 ) && ( (i+1)%2==0 ) ){  
                maxDiasMes[i] = 31;  
            } else{  
                maxDiasMes[i] = 30;  
            }  
        }  
        maxDiasmes[1] = 28;  
        for(int i = 0; i < 12; i++){  
            System.out.println(nomeMes[i] + ":" + maxDiasMes[i]);  
        }  
    }  
}
```

## (30) MATRIZES

① ARRAYS MULTIDIMENSIONAIS → ARRANJO DE VETORES.

② CLASSES DIMENSÃO E' REPRESENTADA POR UM PAR DE COORDENADAS.

[EX] → MATRIZES

```
class ManipulaMatrix{  
    public static void main(String args[]){  
        int mat[][] = new int[5][2];  
  
        for(int i = 0; i < Mat.length; i++){  
            for(int j = 0; j < Mat[0].length; j++){  
                Mat[i][j] = (i * 2) + j;  
                System.out.println("Mat[" + i + "]" + "[" + j + "] = " + Mat[i][j]);  
            }  
            System.out.println(" ");  
        }  
    }  
}
```

## Exercícios UFGA 5 → VETORES E MATRIZES

Questão 1 → Programa → ler VETOR com 8 ELEMENTOS e IMPRIMIR A SOMA DOS ELEMENTOS -

Declarar → TIPO[] IDENTIFICADOR  
Criar → VET = new TIPO[TAMANHO]  
Inserir → VET = {0, 1, 2, 3, 4, 5}

[ALL] INT VET = new int{0, 1, 2, 3, 4, 5};

```
import java.util.Scanner;  
public class PrintVector  
{  
    public static void main(String args[]){  
        int[] vet = new int[8];  
  
        Scanner keyboard = new Scanner(System.in);  
  
        for(int i = 0; i < vet.length; i++){  
            System.out.println(" Type value vet[" + i + "] : ");  
            vet[i] = keyboard.nextInt();  
        }  
        for(int i = 0; i < vet.length; i++){  
            System.out.println("Vet[" + i + "] = " + vet[i]);  
        }  
    }  
}
```

**Questão 2** → Programa, 2 VETORES ( $A + B$ ) = 8 elementos.  
 ↳ Conclusão  $VET(C) = \text{Soma } (A + B)$ .

```
import java.util.Scanner;
public class AddVector {
    public static void main(String args[]){
        int[] A, B, C;
        A = new int[8];
        B = new int[8];
        C = new int[8];

        Scanner keyboard = new Scanner(System.in);

        for(int i = 0; i < A.length;i++){
            System.out.println("Type Vector A[" + i + "] :");
            A[i] = keyboard.nextInt();
        }
        for(int i = 0; i < B.length;i++){
            System.out.println("Type Vector B[" + i + "] :");
            B[i] = keyboard.nextInt();
            C[i] = A[i] + B[i];
        }
        for(int i = 0; i < C.length; i++){
            System.out.println("Vetor C[" + i + "] = " + C[i]);
        }
    }
}
```

**Questão 3** → 2 VETORES ( $VET_1$  e  $VET_2$ ) = 10 elementos

① Somar os números em ordem crescente

3 5 7 12  
 —————  
 2 5 3 ~~X~~

② Imprimir  $[VET_3]$  = Enunciado.

$$VET_3[0] = VET_1[0]$$

$$VET_3[1] = VET_2[0]$$

**Questão 4** → Programa ler TANANHO vetor, elementos e deixa's IRMADA ordenada

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

public class ReadQtdVector {
    public static void main(String args[]){
        int[] vet;
        int i = 0;
        Scanner kb = new Scanner(System.in);

        System.out.println("Type Vector size:");
        int qtd = kb.nextInt();

        vet = new int[qtd];

        for(i = 0; i < vet.length;i++){
            System.out.println("Type position of vet[" + i + "] : " + i);
            vet[i] = kb.nextInt();
        }
        System.out.println("Vector not ordered");
        for(int elemento : vet){
            System.out.println(elemento);
        }
        System.out.println();
        Arrays.sort(vet);
        System.out.println("Vetor ordered");
        for(int elemento : vet){
            System.out.println(elemento);
        }
        System.out.println();
    }
}
```

## Questão 5 → Classe VETOR & CHAM

Classe VETOR & CHAM

VETOR CHAM

Construtor FRASE)  
↳ Passar frase p/  
o VETOR

NÚM. VOCALIS()  
NÚM. PAVARINAS JAVANIS()

\* DICA → METODO [toString] classe String p/  
colocar os ELEMENTOS da frase no  
VETOR  
↳ Chamar VETOR[] = frase. toCharArray();

TÉCNICA EXTRÍA

1) String = SÉRIE DE CARACTERES UTILIZADA P/ REPRESEN  
TAR A E MANIPULAR O TEXTO.

2) VERSO TECNICA = String É UMA CLASSE PRESENTE NO JAVA, QUE IMPLEM  
ENTA A INTERFACE **CharSequence** (ALÉM DA SERIALIZABLE E COMPARABLE).

3) StringS = MUDA ORDEM DOS CARACTERES.

4) A classe String É UM "WRAPPER" DE UM VETOR DE "CHAR". EM OUTRAS  
PALAVRAS, O String GESTA PARA UM VETOR DE "CHAR" ASSIM COMO O Integer  
ESTÁ PARA O TIPO PRIMITIVO **INT**.

char DATA[] = {'a', 'b', 'c'};  
String STR = new String(DATA);  
A PRINCIPAL DIFERENÇA É QUE  
COM O "CHAR DATA[]" NÃO  
TEM-SE TODA O "PODER" QUE  
A CLASSE String PROPORCIONA P/  
MANIPULAR ESSE VETOR.

5) METODO CHARAT()

↳ METODO SIMPLES QUE CONSISTE EM RETORNAR APENAS UM CARACTERS  
EM DETERMINADO POSIÇÃO DA NOSSA String.

String VADOR = "DEVMENDA - JONAS";  
S.O.PLN(VADOR. charAt(0));  
RETORNO → **D**

6) METODO CONEPONTAT()

↳ RETORNA O VALOR EM UNIDADE DO CARACTERE ESPECIFICADO NO INDEX  
DO PARAMETRO.

String VADOR = "DEVMENDA - JONAS";  
S.O.PLN(VADOR. conEPONTAT(0));  
→ **68**

\* NOTE QUE A EQUIVALENCIA DA LETRA 'D' CORRESPONDE A POSIÇÃO 0 NA NOSSA  
String E TAMBÉM A 68, CONSIDERANDO A CONVENÇÃO **UNICODE**.

## 7) Método Componente e componenteIgnoreCase

- ↳ Ambos fazem comparações de duas strings.
- ↳ **Componente** = considera letras maiúsculas e minúsculas na comparação.
- ↳ **componenteIgnoreCase** = ignora diferenças decaídas de minúsculas em maiúsculas.
- ↳ Ambos também retornam a quantidade de diferenças, sendo que nos importa saber; no momento, que quando o returno = 0 as duas strings são iguais.

String valor = "DEVMEDIA - Java";

S.O.PLN(valor. componente("DEVMEDIA - Java")) == 0? true : false);

S.O.PLN(valor. componente("DEVMEDIA - Java")) == 0? true : false);

S.O.PLN(valor. componenteIgnoreCase("DEVMEDIA - Java")) == 0?

true : false);

, true

is false

, true

## 8) Método endsWith e startsWith

↳ **ENDSWITH** = Verifica se a string termina com o valor especificado

↳ **STARTSWITH** = Verifica se a string começa com o valor especificado.

④ Possui duas vantagens:

↳ Parâmetro "Int Offset" = onde deve começar a verificação do inicio da string.

String valor = "DEVMEDIA - Java";

S.O.PLN(valor. endsWith("Java"));

S.O.PLN(valor. startsWith("DEV")));

S.O.PLN(valor. startsWith("ME", 3)));

5  
true  
true  
true.

## 9) Método ToCharArray

↳ CONVIRTE UMA STRING EM UM ARRAY DE CHAR, OU SEJA, UMA STRING DE 10 POSIÇÕES IRÁ SER CONVERTIDA EM UM VETOR char[10] DE 10 POSIÇÕES.

```
String valor = "DEVMARIA - ATUA";  
for (char c : valor.toCharArray()) {  
    S.O.PN("char: " + c);  
}
```

DOS: OS ESPAÇOS TAMBÉM SÃO PASSADOS P/ O VETOR.

D  
E  
M  
E  
D  
I  
A  
-  
J  
A  
V  
A

Espaço  
Espaço

## 10) Método GetBytes

↳ CONVIRTE A STRING EM UM VETOR DE BYTE[ ].

↳ USANDO VENEMOS PRENSAROS SÓ UNICAMENTE NO BANCO DESCONSIDERANDO A COORDENADA ATUAL.

↳ NO PostgreSQL, P/ EXEMPLO, PODEMOS UTILIZAR O TIPO "BYTEA" QUE É ANÁLOGO AO "BYTE" EM JAVA.

↳ Assim como você pode converter ?/ BYTE, podemos voltar p/ String em char.

```
String valor = "DEVMARIA - ATUA";  
for (byte b : valor.getBytes()) {  
    S.O.PN("Byte: " + b);  
}
```

68 68 32  
69 73 74  
86 65 97  
77 32 118  
69 45 97

## 11) Método isEmpty

→ VERIFICA SE UMA STRING ESTA VAZIA OU NÃO.

→ VERIFICADO BEM-SA-SE SE SUA STRING POSSUI TAMANHO=0, OU SEJA, O String.LENGTH()=0,

→ SE USAMOS O MÉTODO EM UMA STRING COM VALOR NULL, RECEBEREIS UM ERRO.

## Formato Entrada

↳ Strg var = null;  
 S.O.PLN(nvl.isNullOrEmpty());

→ ENTRADA.

## 12) Método SPLIT()

↳ Cria um **Array** de **Strings** com base no "REGEX" passado via parâmetro, ou seja, ele divide a **String** em várias outras **Strings** com base no seu **REGEX**.

↳ UTIL P/ SEPARAR TAGS EM UMA STRING COMPLEXA.

"Software, engenharia, computing" → 3 strings

1 string

↳ Método count() = Parâmetro "int limit", onde você IDENTIFICA quantas vezes o **REGEX** sera' aplicado em toda **String**.

Strg var = "DEVMARIA-JAVA";

Strg[] var com SPLIT = nvl.split("-");

for (Strg s : varComSplit) {

| S.O.PLN(s);

3  
Sen Unite

→ DEV MARIA

JAVA

Strg var = "DEVMARIA - Java - Engenharia - Software";

Strg[] var com SPLIT = nvl.split("-", 2);

for (Strg s : varComSplit) {

| S.O.PLN(s);

9  
Com Unite

→ DEV MARIA ①

Java - Engenharia - Software ②

### 13) METODOS *substring* e *SubSequence*

- ↳ RETORNA UMA PARTE ESPECIFICA DE UM DETERMINADO STRING.
- ↳ substring = RETORNA NOVA STRING.
- ↳ SubSequence = RETORNA UM CharSequence.
  - ( $\hookrightarrow$  INTERFACE  $\rightarrow$  STRING É A IMPLEMENTADA INTERFACE).

Exemplo: "DEVMENIA - DATA";

S.O.PLN(nahr. SubSequence(0,5));

S.O.PLN(nahr. substring(0,5));

DEVME  
DATA

### 14) METODOS toLowerCase, toUpperCase e trim

- ↳ toLowerCase = Torna a string p/ caixa Baixa.
- ↳ toUpperCase = Inverso ^
- ↳ trim = Remove espaço em Branco no Início e no Fim da string.

Exemplo: "DEVMENIA - DATA";

S.O.PLN(nahr. toLowerCase());

S.O.PLN(nahr. toUpperCase());

S.O.PLN(nahr. trim());

DEVMEIA - DATA  
DEVMENIA - DATA  
DEVMENIA - JAYA

### 15) Método valueOf()

- ↳ CONverte DIVERSOS TIPOS (Boolean, Int, Char, Double, Float, Long, Object...) PARA STRING.

Boolean myBoolean = true;

S.O.PLN(Strg. valueOf(myBoolean));

Float myFloat = -10;

S.O.PLN(Strg. valueOf(myFloat));

int myInt = 9;

S.O.PLN(Strg. valueOf(myInt));

Double myDouble = 10.3d;

S.O.PLN(Strg. valueOf(myDouble));

true

-10.0

9

10.3

## 16) Método format

↳ Formata o **String** de acordo com as especificações passadas.

```
String result = String.format("Hoje temos preços %.2f reais", 100000);
S.O..Println(result);
```

```
Result = String.format("10/3 = %.2f", 10.0/3.0);
S.O..Println(result);
```

```
public class CharVector {
    private char[] vetchar;
    //Constructor for objects of class CharVector
    public CharVector(String frase) {
        this.vetchar = frase.toCharArray();
    }
    // An example of a method - replace this comment with your own
    public void showCharVector(){
        for(int i = 0; i < vetchar.length; i++){
            System.out.println(vetchar[i]);
        }
    }
    public int vogals(String frase){
        int contVogals = 0;
        frase.toLowerCase();
        for(int i = 0; i < frase.length(); i++){
            char charac = frase.charAt(i);
            if(charac == 'a' || charac=='e' || charac=='i' || charac=='o' || charac=='u'){
                contVogals++;
            }
        }
        return contVogals;
    }
    //aplicação teste.
    public static void main(String args[]){
        String frase = "test String to charVector";
        int qtd;
        CharVector charvet = new CharVector(frase);
        charvet.showCharVector();
        qtd = charvet.vogals(frase);
        System.out.println("Qtd Vogals = " + qtd);
    }
}
```

Questão 6 → Programa, receber uma matriz, a matriz é encadeada no vetor.

**Sintaxe**: `int MAT[][] = new int[s][c];`

```
public class OrderMatrix {
    private int line, column;

    public OrderMatrix(int l, int c){
        this.line = l;
        this.column = c;
    }
    public OrderMatrix(){
    }
    public void showOrder(){
        System.out.println("Matrix Order = [" + line + "][" + column + "]");
    }
    //setters and getters
    public void setLine(int l){
        this.line = l;
    }
    public void setColumn(int c){
        this.column = c;
    }
    public int getLine(){
        return line;
    }
    public int getColumn(){
        return column;
    }
}
```

```

import java.util.Scanner;
public class MatrixApp
{
    public static void main(String args[]){
        int bigger = 0;
        OrderMatrix order = new OrderMatrix();
        int[][] matrix;
        Scanner kb = new Scanner(System.in);

        System.out.println("Type Matrix order[LxC]\n");
        System.out.println("Type Line : ");
        order.setLine(kb.nextInt());
        System.out.println("Type Column : ");
        order.setColumn(kb.nextInt());
        order.showOrder();
        matrix = new int[order.getLine()][order.getColumn()];

        for(int i = 0; i < order.getLine(); i++){
            for(int j = 0; j < order.getColumn(); j++){
                System.out.println("Type matrix value [" + i + "][" + j + "] :");
                matrix[i][j] = kb.nextInt();
            }
        }

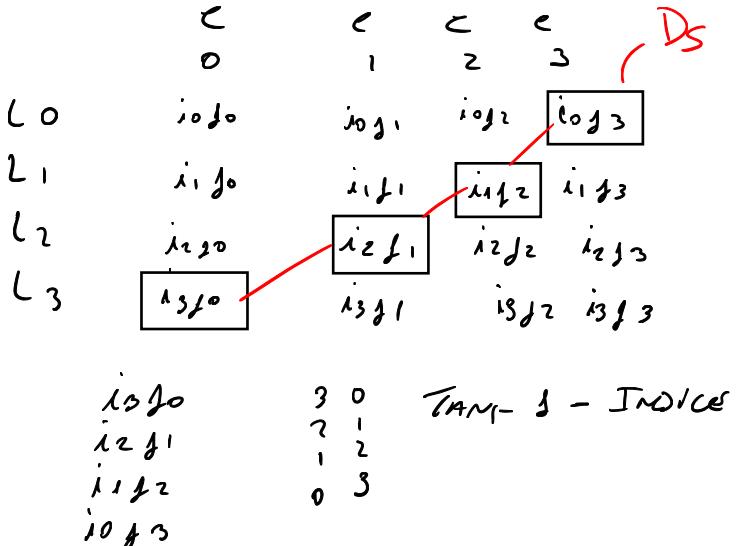
        for(int i = 0; i < order.getLine(); i++){
            for(int j = 0; j < order.getColumn(); j++){
                System.out.println("Matrix [" + i + "][" + j + "] = " + matrix[i][j]);
                if(matrix[i][j] > bigger){
                    bigger = matrix[i][j];
                }
            }
        }

        System.out.println("\n Biggest Number = " + bigger);
    }
}

```

Questão 7 → Orden de uma matriz quadrada, elementos e soma da diagonal secundária.

$M(n \times n)$



```

import java.util.Scanner;
public class MatrixSquareApp
{
    public static void main(String args[]){
        OrderMatrix order;
        int[][] matrix;
        int soma = 0;
        order = new OrderMatrix();
        Scanner kb = new Scanner(System.in);

        System.out.println("Type Matrix order[LxC]\n");

        do{
            System.out.println("Type Line : ");
            order.setLine(kb.nextInt());
            System.out.println("Type Column : ");
            order.setColumn(kb.nextInt());
            if(order.getLine() != order.getColumn()){
                System.out.println("Needs Square Matrix! Type Again..");
            }
        }while(order.getLine() != order.getColumn());
        order.showOrder();
        matrix = new int[order.getLine()][order.getColumn()];

        for(int i = 0; i < order.getLine(); i++){
            for(int j = 0; j < order.getColumn(); j++){
                System.out.println("Type matrix value [" + i + "][" + j + "] :");
                matrix[i][j] = kb.nextInt();
            }
        }

        System.out.println("Diagonal Secundaria");
        for(int i = 0; i < order.getLine(); i++){
            System.out.println("Matrix [" + i + "][" + (order.getLine() - 1 - i) + "] = " + matrix[i][order.getLine() - 1 - i]);
            soma = soma + matrix[i][order.getLine() - 1 - i];
        }

        System.out.println("Soma Diagonal Secundaria = " + soma);
    }
}

```

# Questão 8 → Classe Matrix De Inteiros

## Classe MatrixInt

### MATINT

Constructor (Order)

↳ Instance  
↳  $line = 0$   
 $column = 0$

ADD (l, c, v)

↳ l < n & c < m = true

IsSquare () → T/F

↳  $l = c$

TotalMatrix()

ValueLine()

↳ NOT found = -1

```
public class MatInt
{
    // instance variables - replace the example below with your own
    private int[][] matrixInt;
    private int line, column;
    //Constructor for objects of class MatInt
    public MatInt(OrderMatrix o){
        line = o.getLine();
        column = o.getColumn();
        this.matrixInt = new int[line][column];
        for(int i = 0; i < o.getLine(); i++){
            for(int j = 0; j < o.getColumn(); j++){
                this.matrixInt[i][j] = 0;
            }
        }
        // int value = 0;
        // for( int l : matrixInt){
        //     for( int el : l){
        //         // value = el;
        //         // System.out.println(value);
        //     }
        // }
    }
    public void addMatrixInt(int l, int c, int v){
        if(l > matrixInt.length || c > matrixInt.length){
            System.out.println("Out of Matrix Order");
        }
        matrixInt[l][c] = v;
    }
    public boolean isSquare(MatInt m){
        if(line > column){
            return false;
        }else{
            return true;
        }
    }
    public int MatrixTotalValue(){
        int total = 0;
        for(int[] line : matrixInt){
            for(int el : line){
                total = total + el;
            }
        }
        return total;
    }
    public int SearchValueInLine(int val){
        int line = 0;
        for(int i = 0; i < matrixInt.length; i++){
            for(int j = 0; j < matrixInt.length; j++){
                if( val == matrixInt[i][j]){
                    System.out.println("Value: " + val + " is in Line: " + i );
                    line = i;
                    return line;
                }else{
                    line = -1;
                }
            }
        }
        return line;
    }
}
```

# Quest 9 → Classe funzionale

## Classe funzionale

MATRICA (int)  
Nome (String)  
DEPARTMENT (int)  
Salvo (float)  
FOTO (String)

## Constructor

```
public class Employee{  
    // instance variables - replace the example below with your own  
    private int registration, department;  
    private float salary;  
    private String name, role;  
  
    // Constructor for objects of class Employee  
  
    public Employee(int reg,int d, float s, String n, String rol){  
        // initialise instance variables  
        this.registration = reg;  
        this.department = d;  
        this.salary = s;  
        this.name = n;  
        this.role = rol;  
    }  
    // setters and getters  
    public void setRegistration(int reg){  
        this.registration = reg;  
    }  
    public void setDepartment(int d){  
        this.department = d;  
    }  
    public void setSalary(float s){  
        this.salary = s;  
    }  
    public void setName(String n){  
        this.name = n;  
    }  
    public void setRole(String rol){  
        this.role = rol;  
    }  
    public int getRegistration(){  
        return registration;  
    }  
    public int getDepartment(){  
        return department;  
    }  
    public float getSalary(){  
        return salary;  
    }  
    public String getName(){  
        return name;  
    }  
    public String getRole(){  
        return role;  
    }  
}
```

## (Questão 10) → Classe SGN PESSOAL

Classe SGN PESSOAL
Função VET[] ↳ INVERSE (int)
CONSTRUTOR (P) ↳ NÚMERO DE FUNCIONÁRIOS INICIALIZAÇÃO = 0
ADD FUNC(); IMPRIMIR TABELA PAGAMENTO()
TOTAL FOLHA PAGAMENTO() MÉTODOS DESENTRALIZAR (P) ↳ NUN DEP
FUNCIONÁRIOS FUND (P) ↳ FUND
IMPRIMIR FOLHA PAGAMENTO DESP (P)

### ④ METODOS DE ORDENAÇÃO

① → for (i = 0; i < Array.length; i++) {  
    for (j = 0; j < Array.length; j++) {  
        IF (array[i] < array[j]) {  
            int temp = array[i];  
            array[i] = array[j];  
            array[j] = temp;  
        }  
    }  
}

Obs → Ver outros métodos!

```

import java.util.Arrays;
public class PersonalSector{
    // instance variables - replace the example below with your own
    private int index;
    private Employee[] vetPS;

    // Constructor for objects of class PersonalSector

    public PersonalSector(int numEmp){
        this.vetPS = new Employee[numEmp];
        this.index = 0;
    }
    public void AddEmp(Employee emp){
        vetPS[index] = emp;
        index++;
    }
    public void showPayroll(){
        for(Employee emp : vetPS){
            System.out.println("Nome: " + emp.getName() + "Salary: R$" + emp.getSalary());
        }
    }
    public void showTotalPayroll(){
        float total = 0;
        for(Employee emp : vetPS){
            total = total + emp.getSalary();
        }
        System.out.println("Total Payroll = R$" + total);
    }
    public void ShowBiggerSal(){
        float bgSal = 0;
        String name = "";
        for(Employee emp : vetPS){
            if(emp.getSalary() > bgSal){
                bgSal = emp.getSalary();
                name = emp.getName();
            }
        }
        System.out.println("Name: " + name + "Salary: R$" + bgSal);
    }
    public void showDepEmp(int num){
        for(Employee emp : vetPS){
            if(emp.getDepartment() == num){
                System.out.println("Name: " + emp.getName() + "Role: " + emp.getRole());
            }
        }
    }
    public void showRoleEmp(String rol){
        for(Employee emp : vetPS){
            if(emp.getRole() == rol){
                System.out.println("Name: " + emp.getName());
            }
        }
    }
    public void showRoleEmpOrd(){
        Employee[] vetAux = this.vetPS; // perguntar ao professor sobre esse uso do this;
        for(int i = 0; i < vetAux.length; i++){
            for(int j = 0; j < vetAux.length; j++){
                if(vetAux[i].getSalary() < vetAux[j].getSalary()){
                    Employee temp = vetAux[i];
                    vetAux[i] = vetAux[j];
                    vetAux[j] = temp;
                }
            }
        }
        for(Employee emp: vetAux){
            System.out.println("Name: " + emp.getName() + "Salary: R$" + emp.getSalary());
        }
    }
}

```

Question 22 → Programs Analysis =  $\ell$  wanna

- ④ Recuerda los tipos de instrumentos
  - ④ Instancia los instrumentos
  - ④ Coloca en cuál tipo de clase forman parte
  - ④ Viven más o menos en cuáles son posibles.