

JAVA Review

⑧ Programas de computador ⑨

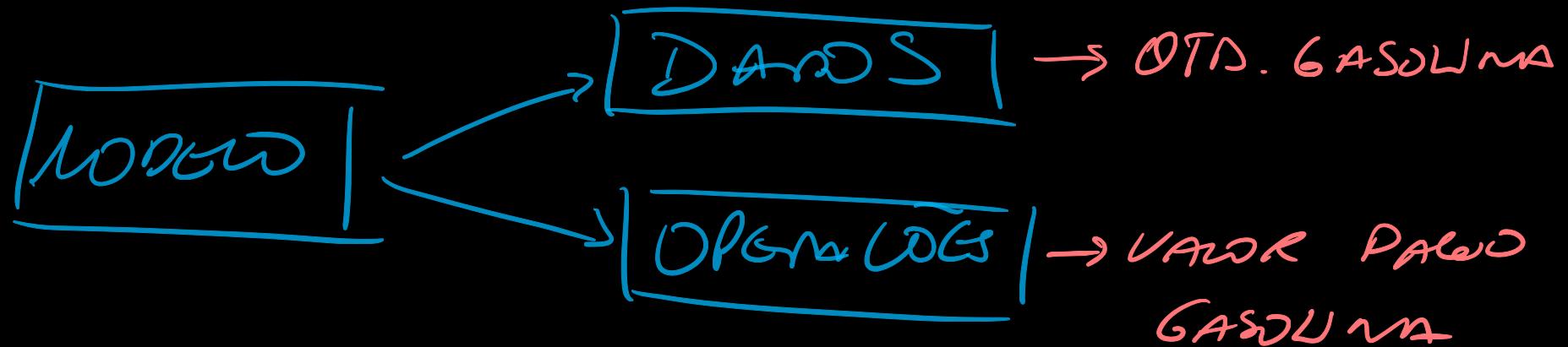
↳ Condensar de códigos e regras em um programa na forma de texto contendo passos para manipulação dos dados de um computador.

↳ São feitos em uma linguagem de programação.

↳ Pode-se em um programa possuir seus dados definidos na memória para que as linguagens em possam ser comandadas pelo computador através de um comando.

↳ Programas processam dados.

- ↳ O Paradigma de Programação Orientada a OBJ.
concentra nos OS DADOS a serem processados
E OS MECANISMOS DE PROCESSAMENTO desses dados
decom for transformar os dados.
- ↳ OS nomes representam consumidores DADOS
E OPERAÇÕES sobre esses dados.



④ MODELOS ④

↳ MODELOS SÃO REPRESENTAÇÕES SIMPLIFICADAS DE OBJETOS, PESSOAS, ITENS, MÉTODOS, PROCESSOS, CONCEITOS, IDÉIAS, ETC. SÃO USADOS COMUMENTE POR PESSOAS NO SETOR DRA-A-PLA, INDEPENDENTEMENTE DO USO DE COMPUTADORES.

④ MODELOS & DADOS ④

GO QUANDO BOMBEIRO É UM MODELO DE RESTAURANTE, REPRESENTADO DA FORMA SIMPLIFICADA. AS INFORMAÇÕES DO RESTAURANTE Necessárias P/ COMANDAR OS PESSOAS.

↳ O MODELO RESponde ENTOS DADOS OU INFORMAÇÕES. OS DADOS CONTAM NO MODELO SÓ Sobre REQUERIMENTOS ABSTRATOS DO MUNDO PRA FORMA.

RESTAURANTE QASEMD
MOTEL

Mesa 1

Kg Refrig
Sobre mesa
Refrigerante
Cerveja

Mesa 2

Kg Refrig
Sobre mesa
Refrigerante
Cerveja

Mesa 3

Kg Refrig
Sobre mesa
Refrigerante
Cerveja

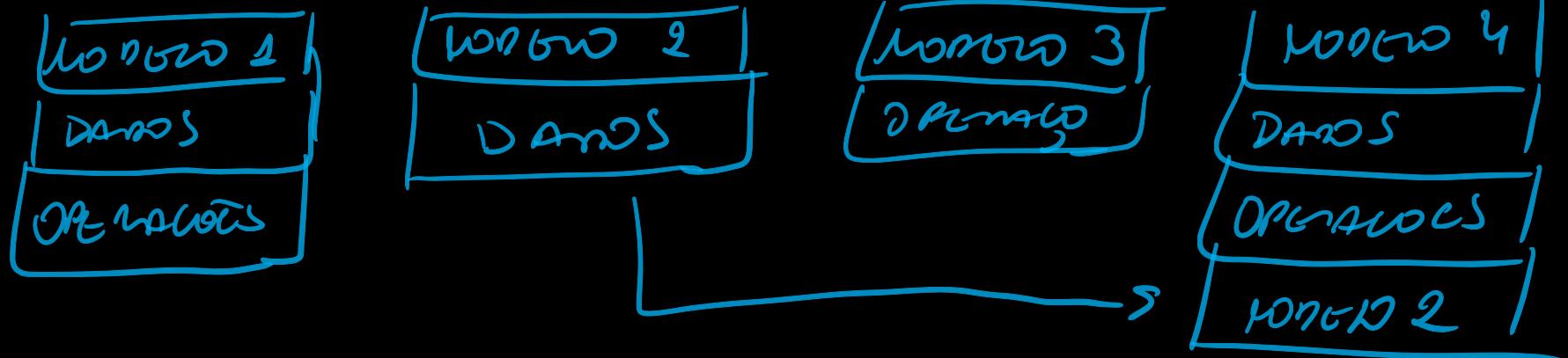
④ MODELOS E OPERAÇÕES

↳ Um novo ambiente contém operações ou procedimen-
tos associados à ele, por exemplo:

- ① Inclusões de um Produto p/ uma mesa.
- ② Marcação do status de um Produto.
- ③ Entregamento dos Produtos de uma mesa.

↳ Também é possível a criação de novos eut possuem
semente DADOS ou somente OPERAÇÕES.

↳ Modelos podem conter submodelos e ser parte
de outros modelos.



* MARCOS E ORGANIZAÇÃO DE OBJETOS *

↳ A diferença entre os marcadores é, em muitos casos, necessária; dependendo do **Contexto**, algumas informações devem ser obtidas em diferentes momentos.

- ① Pessoa que emprega de empresa
- ② Pessoa que faz parte de uma unidade médica.
- ③ Pessoa que contactou com a polícia.

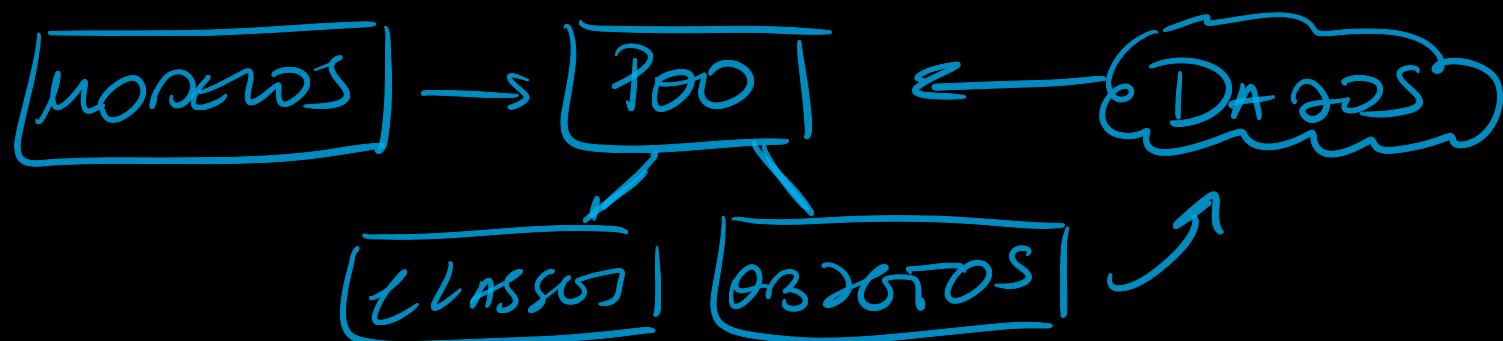
↳ A base e uso de marcadores é uma técnica natural e a extensão dessa abordagem à programação de orden do **PARADIGMA DA PROGRAMAÇÃO ORGANIZADA DE OBJS**. TOP

Programas Dinâmicos & Objetos

↳ VANTAGENS NOVOS BASEADOS EM OBETOS:

- ④ UTEIS P/ COMPUTAÇÕES DE PROBLEMAS.
- ④ Comunhão com os usuários das aplicações.
- ④ Novas empresas
- ④ Produtivos programas.

↳ POO É UM PARADIGMA DE PROGRAMAÇÃO DOS COMPUTADORES ONDE SE USAM CLASSES E OBETOS, ENQUANTOS A PARTIR DOS NOVOS, PARA REPRESENTAR E PROCESSAR DADOS USANDO PROGRAMAS DE COMPUTADOR.



④ ENCAPSULAMENTO

- ↳ A encapsulação é o ocultar dados dentro dos módulos, permitindo que somente operações especializadas ou declarações manipulem os dados ocultos.
- ↳ **Encapsulamento** é um dos benefícios mais principais da programação orientada a objetos POO/ODP.
- ↳ Módulos têm encapsulamento de dados POSSIBILITANDO a execução de programas com ERROS E MAIS ESTÁVEIS.
- ↳ Módulos permitem controlar:
 - ① Dados de representação das estruturas de dados relativos ao que se deseja manter;

② OpenSIS PI manterá os dados.

↳ Em muitos casos, será necessário que os dados
não possam ser acessados por outras pessoas,
mas somente amigos dos usuários.

Ex: Manas Fotografia

Mecanismo da economia
oculta os dados
é a manutenção das
processadoras

↳ em muitos novos sistemas manterá em uso
mecanismos de ocultação de dados: sempre que existir uma
manutenção de dados ao nível da comunicação e respon-
sabilidade para com o usuário ou em de seus dados, deve-
mos ter um openSIS PI fazendo

Ex: banco que representa conta bancária e
dados de rotina.

④ MAS TECNOLOGIAS DE MONITORIOS ④

Ex 1 → Lampada Incandescente

Lampada
EstadoDaLampada
Aceende()
Apaga()
MudaEstado()

④ ATENÇÃO PARA OS NOME DE DADOS E DAS OPERAÇÕES.

④ A ABORDAGEM DO MONITOR DEFINE OS NOMES E OPERAÇÕES.
~ OP

Programa 0260 → Novas Lâmpadas

Início

Dans estandarLampada;

OpenS asconde(); → ACESSO A LÂMPADA

Início

↑EstandarLampada = ACESSO;

Fim

OpenS abre(); → ABRE A LÂMPADA

Início

↑EstandarLampada = ABERTA;

Fim

OpenS novasLâmpada(); → NOVA GRANDE LÂMPADA

Início

↑SE (EstandarLampada == ACESSO)
Impres "Lâmpada Acessa";

SENTO

Impres "Lâmpada Programada";

Fim

(Fim mundo)

Ex 2 → Conta Bancária Simples

Conta Bancária Simples	
Nome do Correntista	
Saldo	Var
Conta - É - ESPECIAIS	
ABRIR CONTA (Nome, XERATO, É ESPECIAL)	
ABRIR CONTA SIMPLES (Nome)	
Deposito (Valor)	
Retirada (Valor)	Obs
MOSTRAR BALANÇO()	

↳ ASPECTOS PRINCIPAIS DE CONTAIS REAIS (SALDOS, TAXAS)
FORMA ORGANIZADA DE LARO.

Pseudobalto → Monetaria Contabilidade Simples

início mês

Dados nome - ex: LIMA, JOSÉ, CONTA-GASTOS;
Nome Abreviado (nome, sobrenome, esquerda) → Produtos
Início → Numeros p/
Dicas S

|
NOME-AD-CONSOADA = NOME
Sobrenome = SOBRENOME
CONTA-GASTOS = GASTOS;

} Incluir na
SIMULTANEAMENTE
TODOS OS DADOS
DO NOME

Exm

opções nome conta simples (novo)

Início

|
NOME-CONSOADA = NOME;
Sobrenome = S. O.;
CONTA-GASTOS = GASTOS;

Exm

✓ Incluir todos os
dados no nome,
usando o novo
Passo com abertura
toda e os outros
dados em lower
aberto

Drops debtors from (new) → Debtors in new
Invoicing system

Inco

$$SPR2 = \text{spr}_2 + (n\omega);$$

Elm

$\text{SPR} = \text{SPR}_{\text{old}} + \Delta \text{SPR}$

Old Retina (SPR_{old}) \longrightarrow Retina now na ONTA
Now $\sim \epsilon$
- contralateral

Turbo

Se l'onta è (sporca = farsa)

Início

$\text{SE}(\text{Var}(w) \leq \text{Var}(z))$

$$s_{\text{var}} = s_{\text{var2}} - \text{var} ;$$

Sams -

$$g_{\mu\nu} = g_{\mu\nu} - \lambda g_{\mu\nu} \delta$$

von mir.

1

From most and () → most more information on address

Ink'c

Influence "more:" + more no commentist"

Imp. m " down : " + son. (2)

INMATE SE (WATA É GROWL) FIMOU "ESTRADA"

$\tilde{W}^n \rightarrow W^n$ as $n \rightarrow \infty$

Ex3 → MOSTRAR DATA

DATA
DIA
MES
ANO
INCLUIR DATA(D,M,A)
DATA E' IGUAL (D,M,A)
MOSTRAR DATA()

④ Observamos que o valor do mês é em Intervalo.

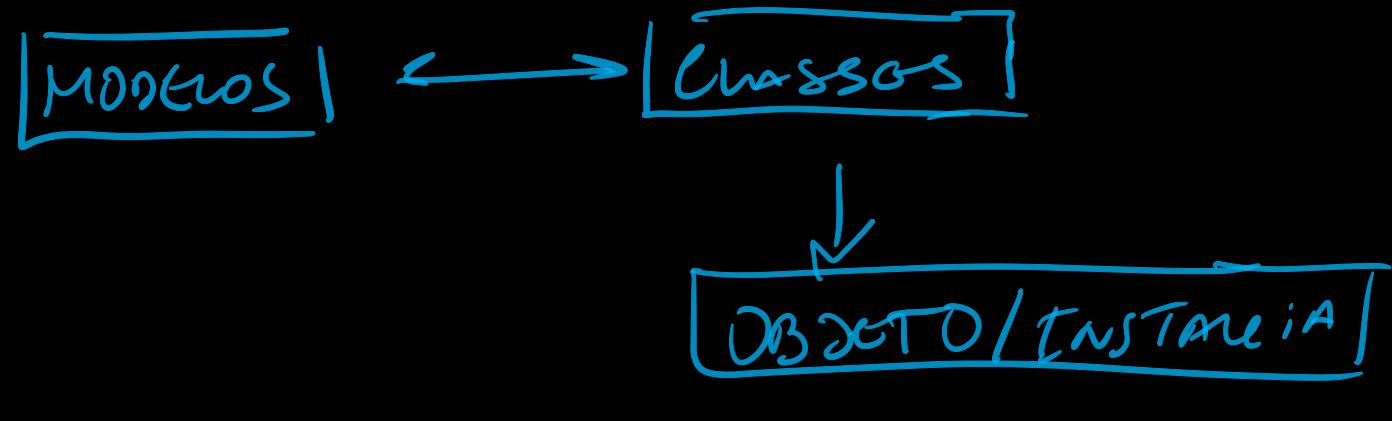
⑤ Observamos também que os meses estão unidos e não separados.

Classe e Objetos

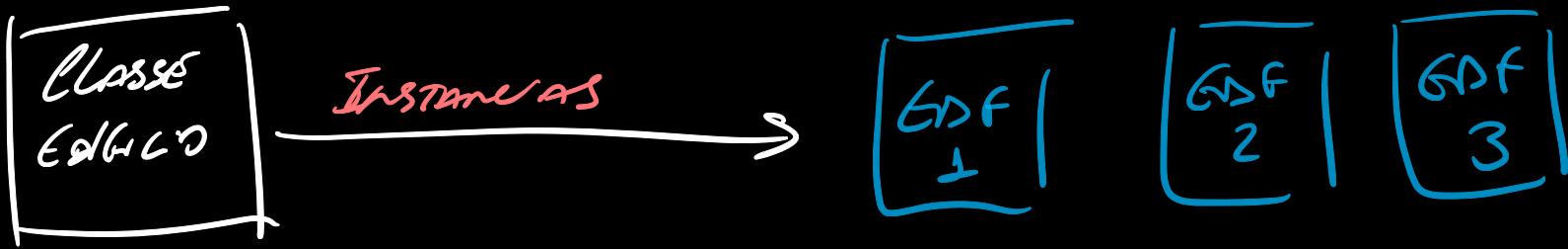
- ↳ Programadores que utilizam POO criam e usam **OBJETOS** a partir de **CLASSES**, que são **recomendadas** dinamicamente com os nomes descritivos.
- ↳ Classes são estruturas organizadas para descrever os **atributos**, ou seja, contém a descrição dos **dados** (**ATRIBUTOS**) e suas **operações** (**MÉTODOS**) que podem ser realizadas sobre elas.

Nome da classe
Atributo 1
Atributo 2
Atributo 3
...
Método 1
Método 2
...

- ↳ Um **Objeto** ou **Instância** é a **realização** da classe.



- ↳ OS DADOS CONTAMOS COM UMA CLASSE ESSA CONCORRENTE
ESSA CLASSE TEM OS CAMPOS OU ATRIBUTOS DAQUELA CLASSE.
- ↳ CADA CAMPO TEM UM NOME E SÓ DE UM TIPO DE
DADO PRIMARIO OU DE UMA CLASSE FALSTEIRA.
- ↳ VARIÁVEIS DENTRO DAS CLASSESS PODEM SER VARIÁVEIS.
- ↳ AS OPERAÇÕES CONTAMOS SÃO OS MÉTODOS.
- ↳ MÉTODOS PODEM RECEBER PARÂMETROS (PARAMETROS).
- ↳ O MELHOR MÉTODO DE RECEBER OS PARÂMETROS É USAR
DE ASSIGNATURA.
- ↳ PARA QUE OS OBJETOS DA INSTRÂNCIA POSSAM SER MANIPULADOS,
É NECESSÁRIO FAZER ALGUMAS REFERÊNCIAS A ESTES OBJETOS,
QUE SÃO MANTIDOS DO "TIPO" DA CLASSE.



④ **Classe**: Parte do Gráfico, que descreve o Edifício, mas não consegue responder a ele.

④ **Instância/Object**: Edifício Concreto

④ **Referência**: Nome do Objeto (Item do Sol)

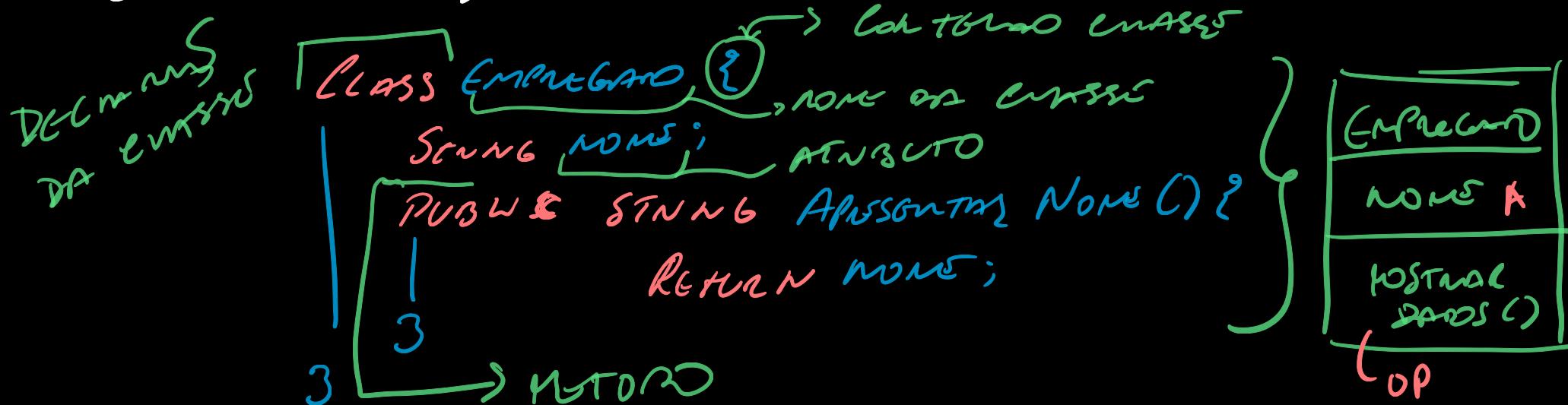
↳ Para manipular a Instância

Classe → Instância → Referência

Classe em Java Sintaxe Básica

↳ Uma classe em Java tem sua declaração com a palavra-chave CLASS seguida do nome da classe.

- ① O nome é Pôrter George.
 - ② Aos amigos que temos.
 - ③ Peço que devolvam as nossas assinaturas.
 - ④ Encorajo todos os amigos a devolverem.
Saudos.
 - ⑤ Considero os amigos unidos para elas { }



Keywords Roserams

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

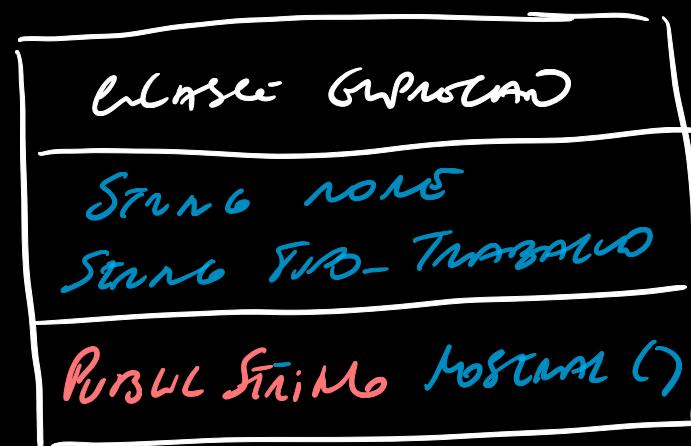
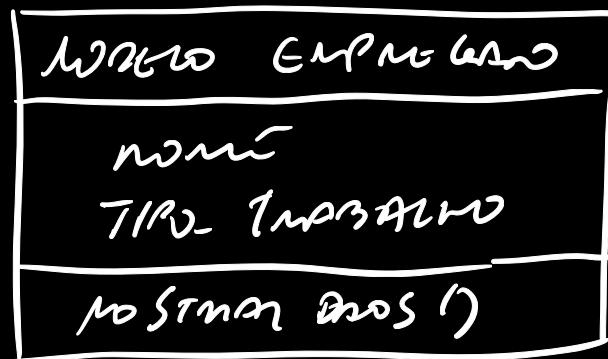
* Clássico em Java → Campos DA CLASSE ⚡

- ↳ Os campos das classes (ATRIBUTOS) servem para declarar variáveis no corpo da classe.
- ↳ Cada campo deve ser representado por um determinado tipo de dado.
- ↳ Em linguagens OO, é possível declarar campos com REGRAS e instâncias de outros classes existentes (OBJETOS).
- ↳ Também é possível declarar campos de TIPOS PRIMITIVOS da prova linguagem (inteiros)

④ Classes em Java → Declaração dos Campos ④

- ↳ A DECLARAÇÃO SIMPLIFICADA, BASTA DECLARAR O TIPO DE DADO, SEGUINDO OS NOMES DOS CAMPOS QUE SÃO DAQUELE TIPO.
- ↳ Podemos observar que, para cada dado no modelo existente um campo correspondente na classe.
→ manipula a instância
- ↳ Campos BEM-SIMILARES REFERENCIAM A UMA CLASSE.

[Ex] → class DataSimple



Classe em Java → TIPOS PRIMITIVOS (MÁTRICAS)

<i>Tipo</i>	<i>Faixa de Valores</i>	<i>Armazenamento</i>
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

• A classe `String` é usada para representar sequências de caracteres. (não são matrizes, somos instâncias da classe `String`)

* Classes em Java	→ Operações Aritméticas	(R)
[Soma] → $+$	[Multiplicação] → \times	Razão Divisão Inteira
[Subtração] → $-$	[Divisão] → \div	$\%$

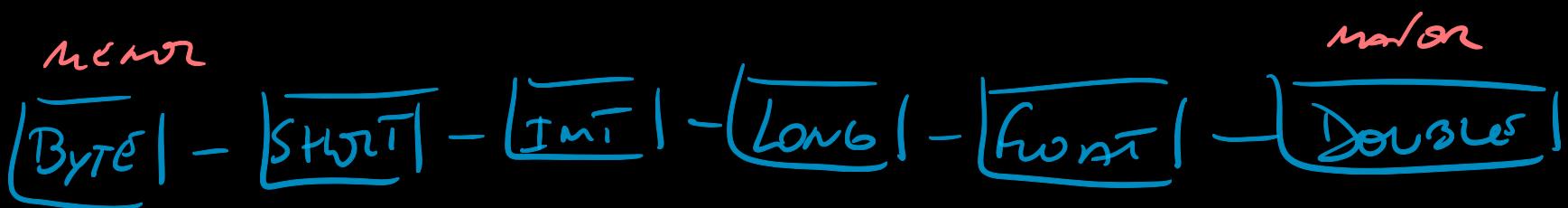
Parte Inteira de uma Divisão → $\text{Int}(A) / \text{Int}(B)$

↳ Operações com Instâncias/Objetos da classe String:

concatenar → $+$

* Classes em Java → Conversões entre TIPOS NÚMEROS

↳ A conversão entre tipos numéricos: sempre deve ser numerais
usar operações matemáticas (+, -, *, /) entre dois tipos diferentes.
Só se conseguimos como setar os operadores básicos no mesmo
tipo. Para isso, o operador de maior **EXPLICAÇÃO** deve ser o
TIPO A ser convertido.



↳ Conversões EXPLÍCITAS: para fazer conversão entre os tipos
A REGRA GERAL é necessária regras de conversão EXPLÍCITAS,
não o contrário:

Double x = 9.997; → Sintaxe para conversão explícita
Int nx = (int) x;

Resultado → Nx = 9

* Clases em Java → Métodos de Classes

- ↳ A maneira das classes representar valores em termos de Dados e operações que manipulam esses dados.
- ↳ As operações são chamadas de métodos.
Operações = métodos
- ↳ Métodos não podem ser criados dentro de outros métodos, nem fora da classe à qual pertencem.

[Ex] → Método Data Simples:

DataSimples
DIA
MES
Ano
InserirDataSimple()
DATAELEMNTA()
ELEMNT2()
mostreData()

PSICO C226 → DATA SIMPLUS

CLASS Datasimplus {

 BYTE DA, MS;] TIPOS PRIMITIVOS DEFINIDOS
 SHORT AWD;] COMO ATAVELHOS (VARIÁVEIS)

 → METODOS DA CLASSE

 VOLU INIDATASIMPLUS (BYTE D, BYTE M, SHORT A) {

 IF (DATA ∈ VARIÁVEL (D, M, A) == TRUE) {

 | DA = D ; MS = M ; AWD = A ;

 } ELSE {

 | DA = 0 ; MS = 0 ; AWD = 0 ;

 } } → METODOS DA CLASSE

 Boolean DATAEVARIÁVEL (BYTE D, BYTE M, SHORT A) {

 IF ((D >= 1) && (D <= 31) && (M >= 1) && (M <= 12) {

 RETURN TRUE ;

 } ELSE {

 RETURN FALSE ;

 } }

PARÂMETROS
COM TIPOS
PRIMITIVOS

PRIMITIVOS

Boolean return (DataSimus or a Data) {

```
If((DVA == OUTDATA.DVA) &&
   (MES == OUTDATA.MES) &&
   (ANO == OUTDATA.ANO) )
{
    return true;
}
else {
    return false;
}
```

Parametros basados en un
desarrollo/estacionamiento pro-
prio en una
clase.

Voir mes autres Data ! ?

```
↑ | System.out.println(DA + "/" + mes + "/" + ADO);
```

3 3
Am Poco Loco

* CLASSE DATA → ESCOPO DE VARIÁVEIS E OBjetos *

- ↳ O ESCOPO DOS CAMPOS E VARIÁVEIS DENTRO DE UMA CLASSE DETERMINA A SUA VISIBILIDADE.
- ↳ CAMPOS DECLARADOS EM UMA CLASSE SÃO VÍRIOES PARA Toda A CLASSE, MAS NAO SÃO OS CAMPOS SÓ SÓ DECLARADOS ATRAVÉS DOS MÉTODOS QUE OS USAM.
- ↳ VARIÁVEIS E INSTÂNCIAS DE OBjetos DECLARADAS DENTRO DE MÉTODOS SÓ SÃO VÍRIOES DENTRO DOSSOS MÉTODOS.
- ↳ DENTRO DE MÉTODOS E BLOQUES DE COMANDOS, AS VARIÁVEIS E OBjetos PODEM SER DECLARADAS ANTES DE SEREM UTILIZADAS.
- ↳ VARIÁVEIS PASSADAS COM ARGUMENTOS PARA MÉTODOS SÓ SÃO VARIÁVEIS DENTRO Desses MÉTODOS.

Ex → class DATA

Classe em Java → Mecanismos de Acesso

- ↳ Mod. Acesso Permite que os usuários tenha os campos como em métodos de uma classe.
- ↳ O objetivo é proteger a integridade e a consistência dos dados e operações que uma determinada classe manipula.

Exemplo de Proteção
aos atributos (dados)

Por exemplo, na classe **DataSimple** anotam-se observações que só se encapsula um **DATA**, os campos **dia**, **mes**, **ano** passam por um processo de **consistência**, só se entram o método **DATA** e **VERIFICA**.
Entretanto, como só temos membros, só podemos garantir que sempre as **DATAS** sejam usadas, mas os **CAMPOS** da classe podem ser acessados diretamente, sem a utilização dos métodos da classe.

Brando de Prova aos Micos (Aves)

Arro bronquio, sans un ^{PAGE Num39(1)} ~~metodo~~
de una clase ~~restante~~.

End class now o metd *Pathogens()*

En Somos una asociación sin orientación política ni religiosa.

Só a interface a exige o PrintPageEventArgs descrevendo os
metodo PrintPageOn.

(S) No know excep^t, was encounter was in
methylase P1 protease of genes, after gene editing
was at least some some missense.

↳ no `Second` campo, seria conveniente usar um `Method`
ou `PI RESTRICTION` o acesso ao método `PrintPageNumber()`, da
í aí que só nesse caso estaria o método `PrintPage()`.

TIPOS MODIFICADORES

Public → Gerante do o campo ou método da classe declarado em esse modificador permitirá seu acesso em qualquer parte da programação ou dentro de outras classes.

Private → só podem ser acessados, modificados ou executados por métodos da mesma classe, sendo outros para o resto. maior uso é para usar instâncias dessa classe ou suas classes filhas na implementação.

Protected → funções como o Private, exceto que classes herdeiras em diferentes namespaces também tem acesso ao campo ou método.

↳ **Final**, campos e métodos podem ser declarados sem modificadores. Neste caso, seu valor permanecerá constante dentro de sua classe à categoria PACKAGE, significando que seus campos e métodos serão visíveis para todas as classes de um mesmo package.

(5) Ao Criar uma Classe, o Programador deve Implementar
uma POUPIA de ocultação ou de ACESSO A DADOS E A MÉTODOS
INTEROS.

*Regras Básicas p/ Implementação de Poupias p/ Classes Simples

- 1) Tudo campo deve ser DECLARADO como PRIVATE ou PROTECTED;
- 2) MÉTODOS que devem ser ACESSADOS DENTRO DA CLASSE SÓ COM O MÉTODO PUBLIC. CASO CLASSES NÃO VENHAM A SER AGREGADAS EM PACOTES, A OMISÃO NÃO CAUSA PROBLEMAS.
- 3) MÉTODOS PARA CONTROLE DOS CAMPOS DEVEM SER ESCRITOS, E ESTES MÉTODOS DEVEREM TER O MÉTODO PUBLIC.
- 4) SE FOR DESGRARAR, MÉTODOS PODEM SER DECLARADOS como PRIVATE.

Ex 1 -> ESCOPO E MODIFICAÇÕES

Public class DATA {

Private byte dia, mes;
Private short ano;

Public void Inicializa (byte [D], byte [M], short [A]) {

If (DATAÉValida (d, m, a) == true) {

dia = d; mes = m; ano = a;

else {

dia = 0; mes = 0; ano = 0;

}

↳ Inicia esse o método para
ser acessado sem restrições

Detetina que essa classe
pode ser instanciada dentro
de outras outras classes.

Detetina que os campos
so podem ser manipulados
dentro da propria classe
como "dia", "mes" e "ano" são deca-
rados fora dos métodos, são
variáveis globais (atributos).

As variáveis "d", "m" e "a" são
parâmetros (argumentos) e só
podem ser manipulados dentro
desse método.

→ Isso é que esse método só pode ser acessado dentro
de outros métodos da própria classe.

PRIVATE Boolean DataValida (Byte D, Byte M, Short A) {

Boolean VARNAME = false;

IF ((D >= 1) && (D <= 31) && (M >= 1) && (M <= 12) {

varname = true;

}

RETURN varname;

}

Varáveis só podem ser manipuladas dentro do
método.

PROTECTED void MostraData() {

System.out.println(DIA + "/" + MES + "/" + ANO);

}

↑
Isso é que esse método só pode ser acessado
dentro de outros métodos desse mesmo ou de classes
herdadoras ou derivadas.

Programas Baseados em Orientação a Objetos

↳ Programas Orientados a Objetos são compostos por:

- 1) **Conjunto de Objetos** que representam os objetos existentes no mundo real.
- 2) **Conjunto de Objetos** que representam estruturas adequadas para manipulação de dados.
- 3) Os Objetos que compõem o programa comunicam-se por troca de mensagens
- 4) O Programa precisa ter um Ponto de Entrada que insere à máquina virtual (JVM) onde iniciar a execução.

Public class AppWard 2

```
|  
| public static void main(String[] args) {  
| | //Programa  
3 | }  
3
```

* APUCAIGÓES EM JAVA *

PUBLIC STATIC VOID MAIN (String[] args)

PUBLIC → É visível para outras classes.

STATIC → Dispõe a cada objetos.

VOID → Nenhum retorno esperado.

MAIN → Consegue passar informações para o Jvm
desde o ponto de entrada da aplicação.

String[] args → Parâmetros passados para através via linha de comando. Args é o IDENTIFICADOR.

Ex: → APLICAGÃO

PUBLIC CLASS APLICAGÃO {
 PRIVATE PONTO P1, P2;
 PUBLIC VOID MOVERPONTOS() {
 P1.MOVER(4F, 2F);
 P2.MOVER(-2F, -4F);
 }
 PUBLIC VOID LINIRPONTOS() {
 P1 = new PONTO();
 P2 = new PONTO();
 }
 PUBLIC VOID LERAR() {
 LINIRPONTOS();
 MOVERPONTOS();
 }
 PUBLIC STATIC VOID MAIN(STRING[] ARGS) {
 APLICAGÃO AP = new APLICAGÃO();
 AP.LERAR();
 }
 }

Declarado os objetos da classe (Pontos) com ATIBUTOS.

Utilizado do método mover da classe (Ponto), através do objeto (P1 e P2).

Instância de um OBSTETO da classe (Ponto).

MÉTODO PRINCIPAL, que será executado imediatamente pelo (JVM).

Java Virtual Machine

Criação de um OBSTETO DA PROPRIA CLASSE.

PUBLIC CLASS PONTO {

```
PRIVATE float x;  
PRIVATE float y; } DeclaroS de variáveis do tipo  
                    Rmadas float como ATRIBUTO
```

```
PUBLIC void mover (float novoX, float novoY) {
```

```
    x = novoX;  
    y = novoY;
```

3

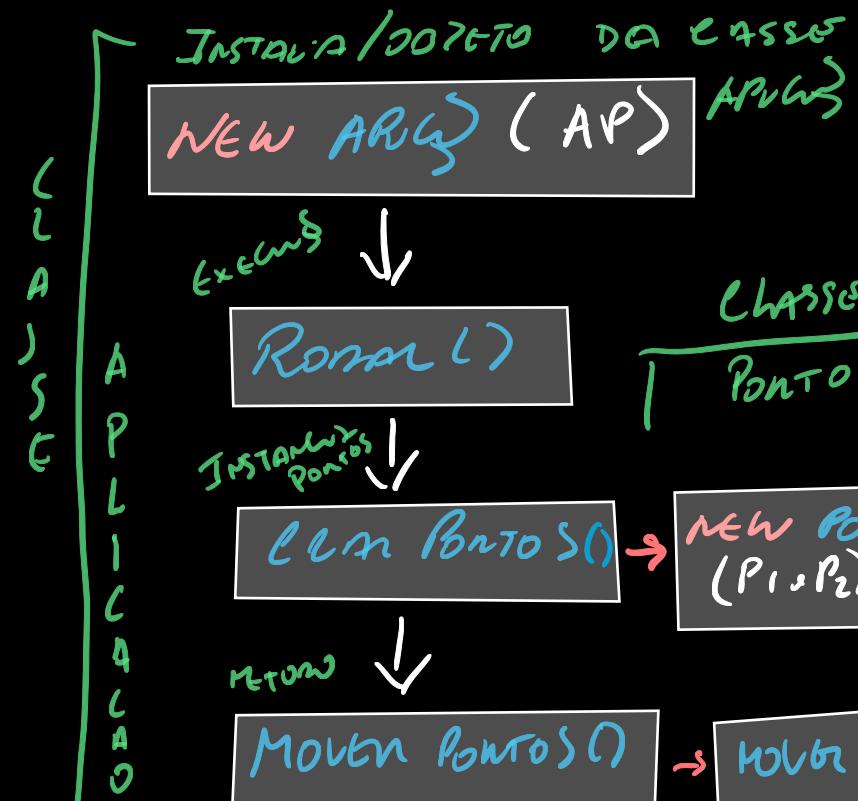
classe filha

PONTO_PUB	
X	PR
Y	PR
mover () PUBLIC	

classe pai

APLICA ^R IS	
PONTO_PU	
moverPontos () PR	
emPontos () PR	
rodar () PR	

```
PUBLIC STATIC void main()
```



* OPERADOR NEW *

- ↳ NEW CRIA UMA NOVA **INSTÂNCIA** DE UM **OBJETO**.
- ↳ NEW TAMBÉM CRIA OS **MEMÓRIAS** NECESSÁRIOS PARA MANTER O ESTADO DO OBJETO E UM **TABLEAU** P/ INSERIR O ENDERECO NO COÉRICO NECESSÁRIO PARA EFETUAR SUAS **OPERAÇÕES**.
- ↳ NEW EXECUTA AS TAREFAS DE INICIARISMO DO OBJETO CONFORME O SEU **CONSTRUTOR**.
- ↳ NEW "RETORNA" O IDENTIFICADOR DO OBJETO CRIADO E O OPERADOR DE ATIVAÇÃO É UTILIZADO P/ FAZER A REFERÊNCIA ENTRE O **VARAVEL**, QUE SERÁ O NOME DA VARIÁVEL, E AS **REFERENCIAS** AO OBJETO.
- ↳ O COMBINACIONAR É A JVM CONHECER O TIPO DE VARIAVEIS E O TIPO DA REFERÊNCIA RETORNADA.

④ Garbage Collection ⑤

↳ Gerenciador de memória para Aplicações Java.

↳ THREAD CHAMA O AUTOMATIC GARBAGE COLLECTOR.

- 1) Reforma a memoria quando os objetos que não estão mais sendo usados.
- 2) Monitora a evolução dos objetos através do new.
- 3) Faz um contador de referência para cada objeto.
- 4) Libera a memória quando o contador chegar a zero.

Referencia NULL

- ↳ Objetos Almacenan una Referencia null que serán **Indisponibles**.
- ↳ A Referência Null é representada para programar resolvendo **null**.
- ↳ Pode-se desfazer uma referência null para un **objeto** atribuindo a ele a palavra **null**. 

* Primitivas e Referencias *

- ↳ Variáveis de TIPOS PRIMITIVOS armazenam um valor.
 - ④ Na declaração mostra-se expresso no membro SUCCESSIONE para o armazenamento da variável.
- ↳ Variáveis DE TIPOS REFERENCIAIS armazenam IDENTIFICAÇÕES PARA OBJETOS.
 - ④ Na Declaração mostra-se expresso para a referência ao OBJETO.
 - ④ A AÇÃO DO OBJETO é realizada através do comando NEW.

④ CONSTRUTORES ④

- ↳ JÁ Vimos que podemos criar APlicações que UTILIZAM Instâncias de CLASSES definidas Pelo USUARIO ou JÁ EXISTENTES.
- ↳ Após Criar 4 Instâncias da CLASSE com a PRIMEIRA - CHAMADA NEW, Vemos METODOS para INIALIZAR OS CAMPOS da INSTÂNCIA.
- ④ na DECLARAÇÃO SUCCEDE ESPAÇO PARA A REFERÊNCIA AO OBJETO.
- ↳ Por Exemplo, um Programa que CRIA 4 INSTÂNCIAS de uma CLASSE MAS AS INICIALIZA COM DADOS

Construtor vazio | — Construtor c/ Parâmetros |

↳ Inicializar os Dados

- ↳ CONSTRUTOR É um tipo especial de membro de classe em que automaticamente cria instâncias de classes através da palavra-chave new.
- ↳ Construtores são usados para:
 - 1) Inicializar atributos de uma classe.
 - 2) Remover rotinas complexas de funções.
 - 3) Remover interações entre os parâmetros passados no momento da criação do objeto.

↳ NA DECADA DE UM CONSTRUIR SEU SE CONSTRUTOR?

- 1) construtores novos só extensão
o mesmo move os classes.
- 2) construtores não possuem tipo de retorno, nem
mimam o valor.
- 3) construtores são, na maioria, públicos.
- 4) programar é pôr seus construtores
diretamente

- ↳ Toda Classe Possui um construtor.
- ↳ ISSO é Programador não tem que usar VERSÃO EXPLÍCITA DO CONSTRUTOR, DÁS UTILIZA A VERSÃO DE CONSTRUTOR IMPLICITA QUE NÃO REQUEIRE NENHUM PARÂMETRO.
- ↳ SE O PROGRAMADOR FOR NECESSARIA UM CONSTRUTOR A UMA CLASSE, A INTERAÇÃO É FEITA INCLUINDO O CONSTRUTOR NOS IMPLENTOS.

[Ex] → Construtores

```
public class EMPREGADO {
```

```
    String nome, endereço, CPF;
```

```
    float salário;
```

```
    int IDADE;
```

```
    public EMPREGADO(String N, String E, String CPF, int I) {
```

```
        nome = nome;
```

```
        endereço = end;
```

```
        CPF = CPF;
```

```
        IDADE = IDA;
```

```
}
```

Constructor da classe EMPREGADO.
Observe que é foi definido um
(tipo de retorno) e possui o retorno
(nome da classe).

```
    public String getCPF() {
```

```
        return CPF;
```

```
}
```

```
    public String getNome() {
```

```
        return nome;
```

```
}
```

→ Programa / código



Class (na Expressão) ?

Public static void

Expressão [obj]

→ URLs → constructor da classe (Expressão)
Para Instanciar um Objeto.

main(String[] args) {

 Expressão obj = new Expressão("manaus", "Rua X moy");

 System.out.println("Nome é = " + obj.getNome());

 System.out.println("Cpf é = " + obj.getCpf());

) nome do objeto.

Novo exemplo

3

⑧ Saída de Dados Párm S ⑧

- ↳ Representando por um ATributo Estático da classe System.
- System.out.println(...)
- ↳ Tipos Pármetros para concatenar & strings ainda
sempre do operador (+).

[Ex] → Saída PármS

Class SaídaDadoS {

```
| public static void main(String[] args) {  
|     | System.out.println("Saída de Dados");  
| }  
| }
```

Entrada de Dados Padrão

↳ Tom classe `FileInputStream` de Entrada de Dados via Tela
para obter o `PACKAGE Java.IO`. Esse pacote con-
tém as classes que fazem de Entrada de Dados.

Import Java.IO.

↳ Para obter valores da tela em sua classe Java é
necessário criar um objeto da classe `BufferedReader`.

`BufferedReader OB = new BufferedReader(new InputStreamReader(`
`System.in));`

↳ Para entrada de dados é só solicitar a `readLine()`, que
devolve a string da digitação do usuário para final. A função
muito simples de se tratar essa execução é sorridente ou
a execução deixa tratando locamente strings da UNHA DE
letras → `throws Java.IO.IOException`.

Conversos der String Batas TIPOS MUY RICOS

↳ O Jardim só permite a entrada de estrangeiros, dessa forma, para
que pessoas internas na rede musical é necessário fazer
uma licença de trabalho para o Tito music designer, com
permisão por A Seguir:

String -> Int → int i = Integer.valueOf(line).intValue();
↳ A variable (line) contains String.

String -> Long | \rightarrow Long L = Long.valueOf(line).longValue();

String -> float → float f = float.parseFloat(str).floatValue();

String to Double | \rightarrow Double D = Double.parseDouble(str);

→ Entradas e Saídas de Dados
→ Pode se com classes ser tratado a entradas de 124205.

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

↓

Entradas e saídas de dados
→ Entradas e saídas de dados de 124205.
Import java.io.*;
class soma {
 ↓
 public static void main(String[] args) throws java.io.IOException {
 String aux;
 int a, b;
 BufferedReader obj = new BufferedReader(new InputStreamReader(
 System.in));
 System.out.println("Digite o primeiro:");
 aux = obj.readLine();
 a = Integer.valueOf(aux).intValue();
 System.out.println("Digite o segundo:");
 aux = obj.readLine();
 b = Integer.valueOf(aux).intValue();
 a = a + b;
 System.out.println("O resultado é: " + a);
 }
}

↓

Entradas e saídas de dados
→ Entradas e saídas de dados de 124205.
Import java.io.*;
class soma {
 ↓
 public static void main(String[] args) throws java.io.IOException {
 String aux;
 int a, b;
 BufferedReader obj = new BufferedReader(new InputStreamReader(
 System.in));
 System.out.println("Digite o primeiro:");
 aux = obj.readLine();
 a = Integer.valueOf(aux).intValue();
 System.out.println("Digite o segundo:");
 aux = obj.readLine();
 b = Integer.valueOf(aux).intValue();
 a = a + b;
 System.out.println("O resultado é: " + a);
 }
}

* Entrada de Dados com Scanner *

```
import java.util.Scanner;           → Precisamos importar a classe SCANNER  
da BIBLIOTECA  
public class MenProgJava {  
    public static void main (String[] args) {  
        int i;  
        Scanner [SCAN] = new Scanner (System.in); → Precisamos usar um Objeto,  
        i = scan.nextInt(); → Instances da classe SCANNER  
        System.out.println ("Valor digitado = " + i); → Utilizamos o objeto [SCAN] para  
        }                                              Enviar a Informação do Teclado. Neste  
        }                                              caso, [SCAN] GETA LENDO UM NUMERO  
                                                INTEIRO.
```

Outras Entradas
do Scanner

↳

next	BYTE() SHORT() INT() LONG()	Float() Double() Boolean()
------	--------------------------------------	----------------------------------

OPERADORES RELACIONAIS LÓGICOS

Op. Relacionais

- < → menor
- > → maior
- <= → menor ou igual
- >= → maior ou igual
- = = → igual
- != → diferente

Op. Aritméticos

- && → E lógico
- || → ou lógico
- ! → não lógico

Op. Relacionais p/ A Classe String

equals → IF (nome.equals("maia"));
!equals → IF (!nome.equals("maia"));

ESTRUTURA DE DECISÃO E CONTROLE - COMANDOS

IF - ELSE

Diagrama
estrutura
do
comando

```
if (expressão - booleana) {  
    | Bloco de Comandos IF  
    } (ELSE) ? opçional  
    | Bloco de Comandos  
    } do ELSE
```

- OBS
- 1) O comando ELSE é opcional.
 - 2) Na estrutura de IF'S ANINHADOS, o ELSE refere-se SEMPRE ao IF mais PROXIMO. Procure usar CHAVES {} para DELIMITAR OS BLOCOS.
 - 3) O operador de comparação IGUAL A é representado por `[==]` e NÃO por `[=]`
 - 4) Bloco no comando PODE SER UM ÚNICO COMANDO (separado por Ponto-e-Vírgula `[;]`) ou VÁRIOS COMANDOS DESEJADAS PELA CLASSE {}.
 - 5) É INVISITABILIZAR O USO DE PARENTESES () NA Expressão-Booleana

Switch

switch (VariávelDeControle) {

 case constante1 :

 Bloco1;

 Break;

 case constante2 :

 opçional Bloco 2

 Break;

[Default] :

 Bloco3;

 Break;

3

↓
OBS

- 1) Usar para selecionar Ações ~~Ações~~
de um número de alternativas.
- 2) As Variáveis de controle só pode ser
DOS TIPOS PRIMITIVOS:
`int, char, String`
- 3) O caso(case) define o ponto
de Entrada da Execução. Se você
quiser que só um Bloco de DE-
clarações seja executado use
Break.
- 4) O caso default é opicional.

WHILE

↓
Estutura do
loop

While (expressão Booleana) {
| Bloco de comandos
3

OBS

- 1) É importante o uso de parênteses () na expressão Booleana.
- 2) O loop permanece em execução enquanto a expressão Booleana for verdadeira.

3) Um erro comum é não atualizar as variáveis de controle do loop, o que acarreta em um loop só.

4) Use sempre { } para delimitar o bloco de comandos.

DO - WHILE

Estrutura do comando

```
DO {  
    | Bloco de Comandos  
} WHILE ( expressão Booleana ) ;
```



- 1) É semelhante ao comando while, sendo que a condição de parada do laço é testada após o bloco de comandos.
- 2) Para todos uma vez o bloco de comandos será executado
- 3) Obs: as mesmas considerações do comando while.

FOR

strutura do comando

```
for (inicializações; tempo de; Iterações) {  
    |  
    Bloco de comandos;  
    }  
}
```

In
Obs →

- 1) Num Loop FOR é exibido as 4 partes de uma ITERAÇÃO.
- 2) Um End comum é coloca o ponto-e-vagalo(;) depois o FOR, ficando o Bloco de comandos bem no longo. O resultado é um Loop que não realiza.

Ex → Ex 8. Decs & Repetition

```
public class exLoopRep {
    public static void main(String[] args) {
        double value = 2;
        char letter = 'A';
        int container;
        while (value <= 20) {
            System.out.println(value);
            value = value * 2;
            if (value >= 20) {
                System.out.println("Nm da Galu");
            }
        }
        for (container = 0; container < 10; container++) {
            System.out.println("Nm da Galu");
        }
    }
}
```

do {

 switch (letra) {

 case 'A':

 System.out.println(letra + " é uma vogal");

 case 'E':

 System.out.println(letra + " — ");

 break;

 case 'I':

 System.out.println(letra + " — ");

 break;

 case 'O':

 System.out.println(letra + " — ");

 break;

 case 'U':

 System.out.println(letra + " — ");

 break;

 } FECHAMENTO DO SWITCH

3

CONTINUAR

FECHAMENTO DO

letra++;

if (letra != 'Z');

System.out.println("fim da S. D. W.");

SobreCARGA

- ↳ UM MÉTODO PODE TER O MESMO NOME COM OUTRO MÉTODO NA MESMA CLASSE.
- ↳ ISso É USAR VÁRIOS FUSOS 2 OU MAIS GRUPOS DIFERENTES DE SE RETORNAR A MESMA TAREFA;
- ↳ MODO CERTO DE SE DIZER QUE O MÉTODO ESTA SobreCARGADA.
- ↳ JAVA PODERÁ A CRIAÇÃO DE MÉTODOS COM NOMES IGUAIS, CONTAJANDO AS ASSINATURAS SEJAM DIFERENTES.
- ↳ A ASSINATURA É' COMPOSTA DO NOSSO MARS OS TIPOS DE PARÂMETROS. O TIPO DE RETORNO NÃO FAZ PARTE DA ASSINATURA.
- ↳ DIFERENÇAS DOS TIPOS DOS PARÂMETROS É' DAS DIFERENTES ASSINATURAS DIFERENTES.

- ↳ JÁ NA ARVORE TAMBÉM, AINDA SE Sobreencanvare os construtores de uma classe
- ↳ AS RESTRIÇÕES SÓS SIMBOLIZAM ALGUMAS ARGUMENTOS DOS MÉTODOS Sobreencanvados.
- ↳ Dizer-se referir DE Dentro do um construtor Sobreencanvado para OUTRO, ATENHA'S NO USO DA PALAVRA RESERVA THIS.

Ex 1 → Sobrecarga

```
Class Empregado {  
    ↓ Public static void main(String[] args) {  
        Empregado EMP1 = new Empregado("Ana", "123", "R. Figueira", 25, 500);  
        Sys.out.println("Nome" + EMP1.DevolveTome());  
        Sys.out.println("Idade + 10 anos = " + EMP1.DevolveTome(10));  
        Empregado EMP2 = new Empregado("Tarcisio", 700);  
        S.O.Println("Salario Tarcisio = " + EMP2.Salario());  
    }  
}
```

① Characteristics dos métodos sobrecarregados -

→ Empregado: 2 maneiras de instanciar com parâmetros constantes.

→ DevolveTome: Sem parâmetro e com + 10 anos (método)

public class Empeno {

String nome, sobrenome, CPF;
float salario;
int idade;

public Empeno (String n, String s, String c, int id, float sal) {

THIS(n, sal);

 nome = n;

 CPF = c;

 idade = id;

}

 Use da referência THIS para
 chamar o seu constructor.

}

public Empeno (String n, float sal) {



 nome = n;

 salario = sal;

}

public int Devolucao () {

 return idade;

}

Subclasses

public int Devolucao (int anos) {

 return idade + anos;

}

④ A Referência THIS

- ↳ NA CHAMADA DOS MÉTODOS DE UMA CLASSE, A MÉTODA MÍTICA (SVM) PASSA IMPLICIMENTE COMO PARÂMETRO UMA REFERÊNCIA AO OBJETO NO USAR A MENSAGEM FOR CURRENT.
- ↳ Quando se deseja RECUPERAR essa REFERÊNCIA DE DENTRO DO CORPO DA CLASSE, USA-SE A PALAVRA RESERVADA THIS.
- ↳ PODE-SE ATÉ UTILIZAR A PALAVRA-CHAVE THIS PARA SE RECUPERAR UM CONSTRUTOR SOBRE O MESMO OBSTO, COM VENOS ANTES MESTRE. ENTRETANTO, NESTE CASO, A PALAVRA-CHAVE THIS NÃO REFERIRÁ À REFERÊNCIA AO OBJETO, VISTO QUE ESTA SERIA CITA SOMA LIVRE.

↳ RESUMO USO THIS;

① Referência ao proprio Objeto;

② Pode dados e métodos na Instância de um Objeto.

③ Passar o objeto como参照 (referência).

public class ExemploThis {

 String msg[]

 public static void main(String[] args) {

 String name; int age; a classe

 funcionario func1 = new Funcionario("maria", 500);

 funcionario func2 = new Funcionario("daniel", 700);

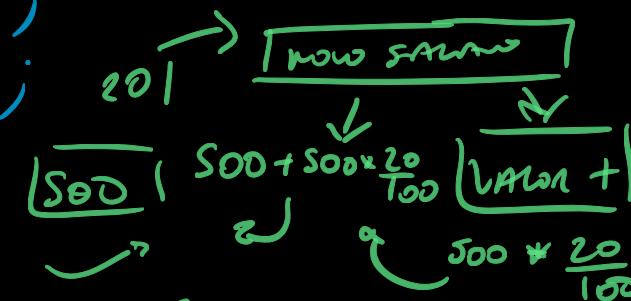
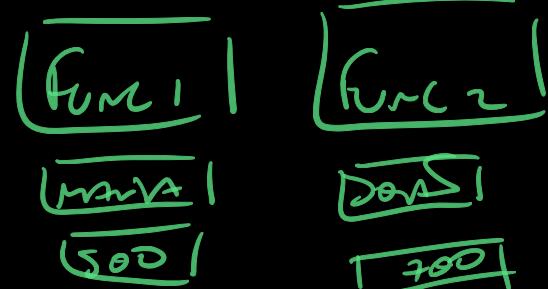
 retornos da classe

 func1.salario = func1.AJUSTA_SALARIO(20);

 func2.salario = func2.AJUSTA_SALARIO(30);

 System.out.println("Novo Salário de "+func1.nome+" é "+func1.salario);

 System.out.println("Novo Salário de "+func2.nome+" é "+func2.salario);



↑
3
}

↓ Public class Funcionario {

| String nome;
| float salario;

Public Funcionario (String n, float s) {

| nome = n;
| salario = s;
| } Objeto = func1 [500]

Public float aumentosalario (float porcent) {

| float novoSalario;
| novoSalario = this.salario ⁵⁰⁰ + calculo (porcent, this);

| return novoSalario; } } (uso de THIS como Referencia)

| } (uso de THIS p/ Acessar
| em Attributo do Objeto) → Passa os
| Dados no Objeto
| Chamado no
| Programa P/ a
| Referencia.

Private float calculo (float percent, Funcionario obj) { } → Função

| float valorAcrecido;

| valorAcrecido = obj.salario * percent / 100;

| return valorAcrecido; } } ^{500 * 20 / 500}

↑ 3

* VETORES *

- ↳ Um VETOR (ARRAY) É uma SEQUÊNCIA DE OBJETOS ou variáveis de tipos diferentes, todos no mesmo tipo e combinados sobre um único IDENTIFICADOR.
- ↳ VETORES São ESTRUTURAS. O seu TAMAÑO É DEFINIDO NO MOMENTO DA SUA CRIAÇÃO.
- ↳ Em Java, VETORES São OBJETOS. Na prática, GRESS HERDA DE OBJECT.
CÓDIGO PROPOSTO:
- ↳ ARRAYS POSSUEM UM ATRIBUTO PÚBLICO QUE INFORMA O SEU TAMAÑO: LENGTH
- ↳ ARRAYS EM JAVA INICIAM NA POSIÇÃO (INDEX) 0.

Declaración → Tipo[] identificador;

[Ex] → int[] VET; → Declaración de un vector de tipo Primitivo.

 → Button() b; → Declaración de un vector de Objeto.

Construcción → Identificador = new Tipo[tamaño];

[Ex] → VET = new int[12];
b = new Button[10];

Inicialización → int VET = new int{0,1,2,3,4,5,6,7,8,9,10};

Obsérvese que un vector posee su

Declaración
Construcción
Inicialización

→ A los mismo tiempo

[Ex] → Uitkomst de vertrekken
van de dobbelstenen?

④ Esse exemplo mostra os mesmos
de um problema de DAS em
que não é possível.

public String vanMês(Stairf() args) {
 int[] maxDiasMes = new int[12]; → arrays + constant
 Stairf() meses = {"JAN", ..., "DEZ"};
 (→ Decs + invars)
 for (int i = 0; i < maxDiasMes.length; i++) {
 if (((i + 1) < 8) && ((i + 1) % 2 == 1)) || ((i + 1) > 3) && ((i + 1) % 2 == 0)))
 {
 maxDiasMes[i] = 31;
 } else {
 maxDiasMes[i] = 30;
 }
 }
 maxDiasMes[1] = 28;
 for (int i = 0; i < 12; i++) {
 S.O..println(meses[i] + ":" + maxDiasMes[i]);
 }
}

④ MATRICES ④

- ↳ OS ARRAYS MULTIDIMENSIONALES FUNCIONAN DE FORMA SIMILAR AOS ARRAYS DIMENSIONAIS.
- ↳ LAS DIMENSOES E REPRESENTACIONES RECLAMAN PAR DE LLAVES [].

Ex) MATRICES

```
char matriona matriz {  
    p. s. v. n (Strg() of.) { 2 (c DECLARAC  $\in$  const)  $\rightarrow$  un  
    int MAT[]{} = new int[5][2];
```

```
    for (int i = 0; i < MAT.LENGTH; i++) {  $\longrightarrow$  LINHAS
```

```
        for (int j = 0; j < MAT[0].LENGTH; j++) {  $\longrightarrow$  COLUNAS
```

```
            MAT[i][j] = (i * 2) + 1;
```

```
            S.O.Println("MAT[" + i + "][" + j + "] = " + MAT[i][j]);
```

```
3
```

```
S.O.Println(" ");
```

```
339
```

④ Rectificação de classes ④

- ↳ Um dos motivos BENEFICIOSOS para a P.O.O nos proporciona é a rectificação de código.
- ↳ A rectificação de características para aproximamento de classes e sous métodos que na estrutura escuros e que não conhecem o seu funtionamento restam e comprovação.
- ↳ A rectificação de codes diminui a necessidade de escrever novos métodos e classes gerando economia de tempo e segurança.
- ↳ Existem 2 maneiras de conseguir esse reajustamento:
Através de Composição ou Atributos de herança.

Composition

- ↳ A composição é o conceito mais usado no desenvolvimento de aplicações baseadas em orientação a objetos.
- ↳ Parte do pressuposto que uma determinada classe usa instâncias de outras classes, utilizando-se deles como partes para compor as funcionalidades da nova classe.
- ↳ Esse conceito pode ser visto em exemplos anteriores (classe CriarEmpregado e classe Exemplares), que instanciam objetos da classe Empregado e Funcionario respectivamente.

CriarEmpregado



Empregado

Exemplares



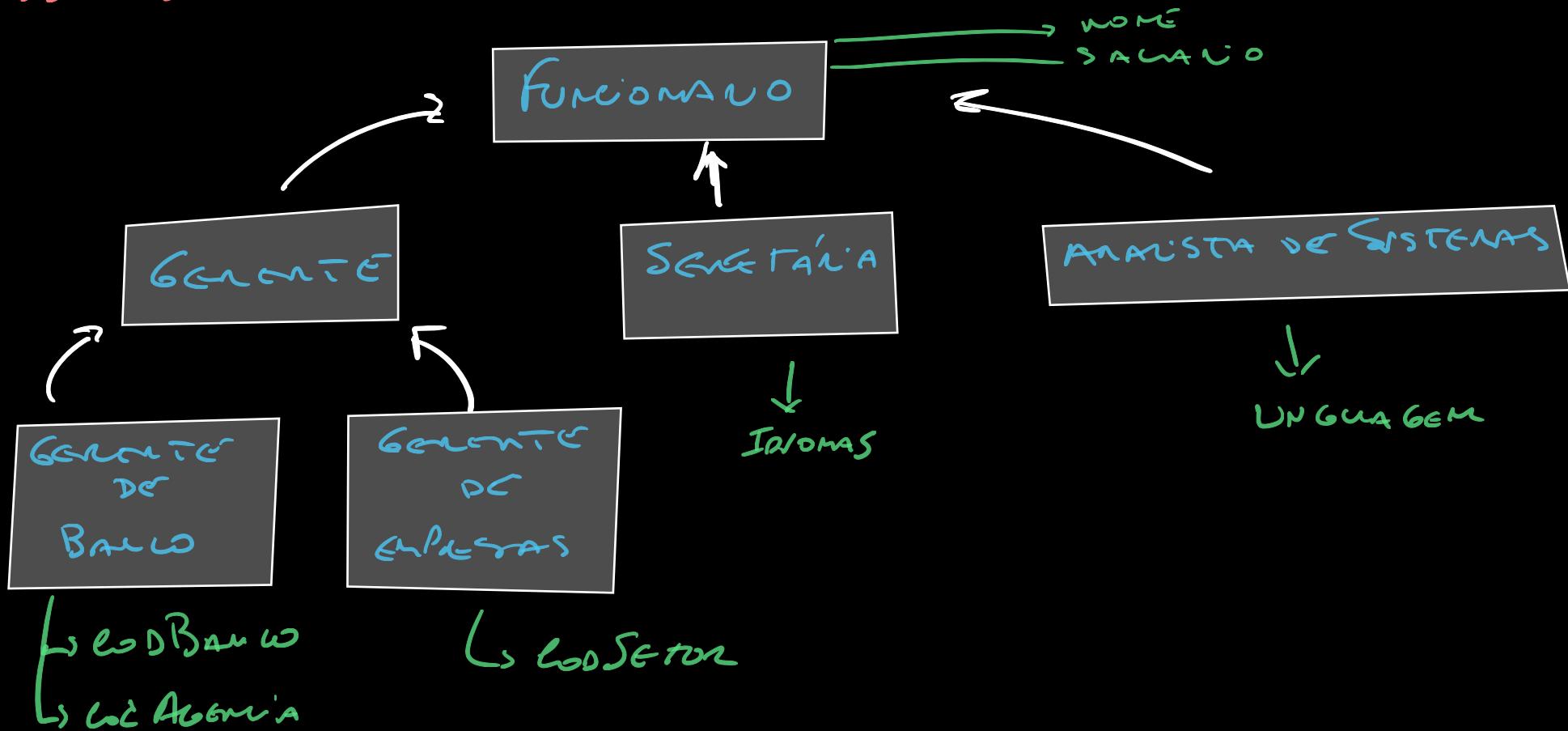
Funcionario

* Hierarquia

- ↳ Hierarquia é a capacidade de uma **classe** definir o seu **comportamento** e sua **estrutura** **em termos** de outras **classes**, normalmente conhecida como **classe Filha** ou **classe Pai**.
- ↳ A Hierarquia não precisa ser imediatamente no decorrer de uma hierarquia de classes. A coleção de todos as classes que se estendem de uma classe Pai é chamada de **Hierarquia de Hierarquia**.
- ↳ A palavra Reservada **extends** é usada para definir que uma classe herda - herda de outra.



Os Atividades do mecanismo de herança é possuir degrau.
Mas classes genéricas que agregam um conjunto de
diferenças comuns a um grande número de objetos (generalizações).
A partir destas generalizações podemos construir
novas classes, mais específicas, que acrescentam novas caracte-
rísticas e comportamentos aos já existentes (especializações).



④ Herança e a Recomendação Super ④

↳ A herança recomenda super, nos possuímos:

- 1) Referência à classe Base (par.) do objeto.
- 2) Acesso a variáveis econômicas (státuos), ou seja, variáveis de uma classe que tem o mesmo nome de uma variável declarada em sua subclasse.
- 3) Acesso a métodos que possuem a mesma assinatura na classe Base e na classe Derivada.
- 4) Utilização para classe Derivada do construtor da classe Par.

→ flexíveis

Public Class Aeronave 2 → Classe Base

↓
public static final byte numeroAGasolina = 5;

public " " " " " " numeroAAleco = 2;

public " " " " " " numeroADvisor = 3;

private static final byte numeroMaximoDePrestimos = 24;

declaracōes

de

CONSTANTES

private String nome;

private " " cor;

private byte combustivel;

} ATTRIBUTOS

public Aeronave (String n, String c, byte comb) {

CONSTRUTOR

 | modelo = n;

 | cor = c;

 | combustivel = comb

3

 | public byte numeroPrestimos() {

 | } return numeroMaximoDePrestimos;

```
public float quantoLustro() {
```

```
float preco = 0;
```

```
switch (consumo) {
```

```
case menorAGasolina:
```

```
    Preco = 12000.0f;
```

```
    Break;
```

```
case menorADiesel:
```

```
    Preco = 10500.0f;
```

```
    Break;
```

```
case menorAOleod:
```

```
    Preco = 11000.0f;
```

```
    Break;
```

```
} return Preco;
```

```
public String toString() {
```

```
String resultado;
```

```
resultado = nome + " " + lata + "\n";
```

```
switch (consumo) {
```

```
case menorAGasolina:
```

```
    resultado += " Gasolina \n";
```

```
    Break;
```

```
case menorAOleod:
```

```
    resultado += " Oleod \n";
```

```
    Break;
```

```
case menorADiesel:
```

```
    resultado += " Diesel \n";
```

```
    Break;
```

```
} return resultado;
```

Public Classe AutomovelBaseo | Extends | Automovel {
 Private Boolean RetornoDoLadoDoPassageiro;
 " " " UnidadeDeLuzTraseira;
 " " " RadioAMFM; ATRIBUTOS

A Parent EXTENDS a
des determina que
AUTOMOVEL BASEO herda
de Automovel.

Public AutomovelBaseo (String m, String c, Syte com, Boolean r, Boolean l,
 Boolean at) {
 Super(m, c, com);

}
 A Parenta SUPER, indica que deve
ser usado o CONSTRUTOR DA CLASSE
 Super(m, c, com);
 retornoDeLadoDoPassageiro = r;
 UnidadeDeLuzTraseira = l;
 radioAMFM = sf;

Public AutomovelBaseo (String m, String c, Syte com) {
 Super(m, c, com);

}
 retornoDeLadoDoPassageiro = true;
 UnidadeDeLuzTraseira = true;
 radioAMFM = true;

}

```

Public float getPrecoCusto() {
    float preco = super.getPrecoCusto();
    if (retornarIndicadorDesconto == true) {
        preco = preco + 280;
    }
    if (limparIndicadorDesconto == true) {
        preco = preco + 650;
    }
    if (rodarAnfim == true) {
        preco = preco + 190;
    }
    return preco;
}

```

```

Public String toString() {
    String resultado = super.toString();
    if (retornarIndicadorDesconto == true) {
        resultado += " / Retornar Desconto ";
    }
    if (limparIndicadorDesconto == true) {
        resultado += " / Limpar Desconto ";
    }
    if (rodarAnfim == true) {
        resultado += " / Rodar Anfim ";
    }
    return resultado;
}

```

Retur preco; Super String ao só se chamado o método
Quando⁽¹⁾ a toString() da classe Base

→ OS DOIS MÉTODOS APRESENTADOS POSSUEM A MESMA ASSINATURA
DA CLASSE Autovetor, D. SÓ SE CARACTERIZA UMA REDEFINIÇÃO
DE MÉTODOS DA CLASSE RASE.

Public Class AppAutomovel {

P.S.V.M (3 tipos) ongs) ? Gastar em ou não com
gasolina ou etanol ou gaso
Automovel a = new Automovel("Fusca", "Vermelho", Automovel.rodasAAlto);

S.O.PLN(a.toString());

S.O.PLN(a.getGralista());

Automovel Basico ab = new Automovel.Basico ("longa", "larja", Automovel.rolinsAlGasolina, true, true, false);

S.O.PLN(ab.getGralista()); Instancia de um OBGETO da classe Automovel Basico
S.O.PLN(ab.getLargura());

OBS O metodo getLargura() esta sendo acessado através de
um OBGETO DA CLASSE AutomovelBasico. ISSO NÃO É POSSIVEL
ACESSAR A CLASSE AutomovelBasico FORDA DA CLASSE AUTOMOVEL.
Assim todos os ATRIBUTOS E METODOS DA CLASSE BASE PODEM
SER USADOS DENTRO CLASSE DERIVADA.

* Polimorfismo *

- ↳ O mecanismo de herança permite a criação de classes a partir de outras já existentes com regras E-Um-Tipo-De, de forma que, a partir de uma classe genérica, classes mais especializadas podem ser criadas.
- ↳ A regra E-Um-Tipo-De, entre classes, permite a existência de outras estruturas hierárquicas de classes que englobam a obectos, o Polimorfismo.
- ↳ Polimorfismo ("muitas formas") permite a manipulação de instâncias de classes da mesma classe-mãe através de forma uniforme.
- ↳ Polimorfismo é um mecanismo que necessita instâncias de uma classe C, e os mesmos métodos serão escritos se processar instâncias de diferentes classes filhas de C, ou seja, diferentes classes que herdam de C, E-Um-Tipo-De-C.

VANTAGENS

- ① Permite o Envio de mensagens a um OBJETO SEM A DETERMINADA CARA DO SEU TIPO.
- ② Permite a EXTENSIBILIDADE DO SISTEMA com pouco a nenhuma IMPACTO nos classes existentes.
- ③ Permite a construção de classes GERAIS para EFETUAR Tarefas BONS comuns aos Softwares, como as Estruturas DE DADOS.

Ex → Polimorfismo

Public Class CarrinhosDeAutomovel {

P. S. V. M (String args[]) {

Automovel A1 = new Automovel("Fiat", "Bras", Automovel.novaAPalco),

AutomovelA2 = new AutomovelBravo("Linha", "Ego", Automovel.novaAGraxina);

Impresso (A1); } O método IMPRESSO recebe um OBJETO da classe AUTOMOVEL
Impresso (A2); } que possui. Dessa forma nesse exemplo, chamamos o
método PASSANDO [A1] e [A2], ou seja, OBJETOS de CLASSES
diferentes mas na mesma HIERARQUIA de classes, caracte-
} rizando dessa forma a
3

Public String VaiImpresso (Automovel a) { CONCEPÇÃO de POLOMORFISMO.

S. O. Pn ("DANOS AUTO");

S. O. Pn (a. ToString());

S. O. Pn ("Vai:" + a. quantoCustou());

S. O. Pn (a. quantoPrestog() + " Preço de " + (a. quantoTotal) / a. quanto_
} Preço ());

Classe Abstrata

- ↳ São classes compostas por implementações genéricas e especificações de procedimentos. Como não temos totalmente implementados, não podemos instanciar, ou seja, não podemos criar objetos dessas classes. Elas apresentam características e comportamentos que servem de modelos para outras classes e fornecem regras de comportamento que servem de moldes para subclasses.
- ↳ A classe serve a servir como molde genérico para as classes que servem de suas subclasses.
- ↳ Para se definir classe abstrata usa-se a palavra-chave **ABSTRACT**.
- ↳ Um método abstrato somente aponta o molde de uma implementação a ser provista pelas classes filhas concretas.

- ↳ Cada classe filha herda todos os métodos de sua implementação de uma forma particular.
- ↳ Métodos abstractos só podem ser usados em classes abstractas.
- ↳ Métodos não-abstractos podem ser usados tanto diretamente como através dos objetos das classes que devem ser subclasses abstractas.
- ↳ As classes que derivarem de uma classe abstracta devem implementar todos os métodos abstractos definidos na classe base.

(E) → Classes Abstractas

↓
abstract class Figura {
 int x;
 int y;
}

DEFINIÇÃO DA CLASSE ABSTRATA, ATUAIS
DA CLASSE ABSTRACT

public Figura(int x1, int y1) {
 x = x1;
 y = y1;
}
} A classe deve ser um CONSTRUTOR,
mesmo que não possa INSTANCIAR
OBJETOS.

ABSTRACT
public abstract void desenha();
public Abstract void apaga();
public void move(int x1, int y1) {
 apaga();
 x = x1;
 y = y1;
 Desenha();
}
} DECLARAÇÃO DOS MÉTODOS ABSTRATOS,
esses métodos devem obter
implementação por parte das classes
filhas.

↑
MÉTHODOS
3

DECLARAÇÃO DE UM MÉTODO NORMAL
que fornece a implementação
de classes filhas.

Classe Quadrado Gato é filha da classe Abstract
 ↓ public Quadrado(int x1, int y1) {
 | super(x1, y1); → constante da classe Base.
 }

 public void desenha() {
 | S.O..Println("Desenhando quadrado (" + x + ", " + y + ")");
 }

 public void move() {
 | S.O..Println("Aproximo quadrado (" + x + ", " + y + ")");
 }

 ↑ Classes de todos os tipos

Classe Quadrado Abstract
 P.S.V.M (String args()) {
 Quadrado q = new Quadrado(10, 10);
 q.desenha();
 q.move(50, 50);
 q.Abraço();

⑧ Interfaces ⑧

- ↳ Se uma classe é definida por apenas métodos abstratos então é melhor usar a herança - classe abstrata. Para estes casos o Java oferece uma técnica chamada de Interfaces que podem ser usadas para definir um protocolo de comportamento.
- ↳ As Interfaces definem os métodos abstratos no fundo de seus membros métodos da interface para todo o corpo.
- ↳ As Interfaces são contrainte hierárquicas.
- ↳ As Interfaces não podem ser instâncias.
- ↳ Uma classe pode implementar várias interfaces mas apenas herda (extende) de uma única classe.
- ↳ As Interfaces são comportamentos da mesma forma em várias classes.

- ↳ Uma Interface é uma coleção de definições de métodos.
- ↳ Uma Interface Pode Também Declarar Constantes.
- ↳ Elas Possuem a mesma regras para interfaces para indicar a definição de uma interface.
- ↳ Sintaticamente é considerada a sua entidade compundida MESTRADA.
 - ④ São sem retornos sem assinaturas.
 - ④ Temos as suas variações com PUBLIC STATIC FINAL;
- ↳ Semanticamente entretanto, as duas diferem, pois uma Interface guarda uma IDÉIA DE PROTOCOLO (COMPOSISSOS) ENTRE CLASSES.

ER → Interfaces

```
Class Linha Implements figuraGeometrica {  
    int x, y;  
  
    public void desenhar() {  
        System.out.println("Aparece Linha (" + x + ", " + y + ")");  
    }  
  
    public void area() {  
        System.out.println("Araçau Linha (" + x + ", " + y + ")");  
    }  
  
    public void mover(int x1, int y1) {  
        this.desenhar();  
        x = x1;  
        y = y1;  
        this.desenhar();  
    }  
}
```

```
Interface figuraGeometrica {  
    public void desenhar();  
    public void area();  
    public void mover(int x1, int y1);  
}
```

Nesse exemplo a classe Linha implementa a interface figuraGeometrica. Ela precisa desenvolver todos os METODOS definidos para interface.

Packages

- ↳ Um PACKAGE é um agrupamento de classes e interfaces que tem localização reservada.
- ↳ No Java é um conjunto de classes e interfaces que estão dentro de um mesmo diretório de arquivos.
- ↳ O Java utiliza Packages para:
 - ① Garantir a unicidade do nome da classe, ou seja, não permitir que existam nomes de classes iguais em um mesmo pacote.
 - ② Restringir localizações de acesso.
- ↳ Membros de uma classe têm a especificação de controle de acesso que consigna quem pode ter acesso a essas classes dentro do mesmo package.

↳ Para colocar una clase en un determinado paquete, deve-se coloca-lá dentro de um pacote existente ou criar um novo do mesmo nome (`.java`).

`package joga.lanç;`

↳ Para utilizar classes de outras em outros pacotes deve-se importar as classes através da declaração `import`.

`import joga.lanç.String`] Importa strings e
classe String

ou ainda

`import joga.lanç;`] Importa todos as classes do
Pacote Lanç

⑧ Principais Pacotes da Linguagem Java

java.awt

→ Classes para a criação de Aplicativos.

java.awt

→ Classes para criar as Interfaces Gráficas.

java.io

→ Classes para Gerenciamento de Entradas e Saídas.

java.lang

→ Classes Básicas da Linguagem (inclui os AUTOMATICALLY, mas não necessita o uso do IMPORT).

java.net

→ Classes para trânsito em rede.

java.util

→ Classes de UTILIDADES (Data, ArrayList, Pilha, Vector, Random, etc).

java.sql

→ Classes para manipulação de Banco de Dados.

④ Componentes Gráficos da Biblioteca Swing

JLabel

→ São utilizados para INVENTAR ALGUNS SOBRE OUTROS
COMPONENTES EM UMA INTERFACE.

④ Principais métodos: setVisible(boolean);

Ex → JLabel.setVisible(true);

JTextField

→ São usados para POSSIBILITAR O USUÁRIO DIGI-
RAR INFORMAÇÕES PARA A APPL.

④ Principais métodos: getText();

setText(String);

Ex →

JTextField.getText()

JTextField.setText(" ");

JButton

→ São usados para DAR O INÍCIO DA EXECUÇÃO DE
ALGUMA Tarefa PODE ABERTURA.

④ Principais métodos: setLabel(String);

Ex →

JButton.setText("Entrar");

list

→ São comandos para permitir que os dados sejam mostrados na tela em forma de lista.

④ Princípios metodos: add(String);

Ex →

lists.add("") + numero);

JOptionPane.showMessageDialog

↳ Este componente permite gerenciar mensagens para o usuário.

④ A classe deve importar a seguinte BIBLIOTECA:

import javax.swing.JOptionPane;

④ A SINTAXE PARA O USO DO COMANDO É:

JOptionPane.showMessageDialog(null, "Mensagem");

TRATAMENTO DE EXCEÇÕES

- ↳ Exceções são erros inesperados, que podem ocorrer em programas. Por exemplo, quando se divide zero em um usuário entre com um número e este divide um zero.
- ↳ As exceções na linguagem Java são objetos normais, instâncias de classes que herdam da classe THROWSABLE. Um objeto da classe THROWSABLE é inválido, quando uma exceção é gerada.
- ↳ A linguagem Java permite que se tratem exceções através dos palavras TRY, CATCH e FINALLY.

TRY → No bloco try você protege o código que pode gerar uma exceção.

CATCH → No bloco CATCH você verifica se é tratada uma exceção gerada

FINALLY → No bloco finally você possivelmente seu programa execute algumas coisas quando o bloco finalmente termina.

↳ Existe en varios tipos de excepciones en Java. Para que no se para todos los algoritmos es necesario descubrir cuáles son las excepciones que son errores (errores) en un motor-viajero metido.

① Algunas causadas por un error de Java:

IOException

NameNotFoundException

EOFException

SocketException

MalformedURLException

```
import javax.swing.JOptionPane;
```

```
public class PlayerPanel extends JFrame {
```

```
    public PlayerPanel() {
```

```
        initComponents();
```

```
}  
private void initComponents() {
```

```
    // CREAMOS E DECLARAMOS NUEVOS NETBEANS
```

```
    // EN EL PUNTO DE ARRANQUE.
```

```
}  
private void jButton1MouseClick(jav.awt.event.MouseEvent evt) {
```

```
    int m;  
    try {
```

```
        m = Integer.parseInt(jTextField1.getText()).intValue();
```

```
    } catch (Exception e) {
```

```
        JOptionPane.showMessageDialog(null, "Ponga numeros enteros");
```

```
        return;
```

Finalmente?

| JTextField.JTextField(" "));

3

List1.add(" "+m);

3

public static void main(String args[]) {

// Este método é definido pelo NetBeans e não deve ser alterado.

java.awt.EventQueue.invokeLater(new Runnable()) {

| public void run() {

| | new Principal().setVisible(true);

o

```
public static void main(String args[])
{
    // Este método é definido pelo NetBeans e não deve ser alterado
    java.awt.EventQueue.invokeLater(new Runnable()
    {
        public void run()
        {
            new Principal().setVisible(true);
        }
    });
}
```

```
// Estes atributos são criados pelo NetBeans e não devem ser alterados
private javax.swing.JButton jButton1;
private javax.swing.JLabel jLabel1;
private javax.swing.JTextField jTextField1;
private java.awt.List list1;
}
```

Nota de Caixa

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.

