

*# 06# → Número de série celular



Câm → 192.168.254.5

[O que significa a sigla IDE?]
Integrat Development Environment
Ambiente de desenvolvimento integrado

[O que é um deadlock?]

Um deadlock acontece quando duas ou mais tarefas bloqueiam permanentemente uma à outra.

Comentário de bloco

```
/* Aplicação HelloWorld */  
public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello, world!");  
    }  
}
```

Nome da classe

Nome do método

Declaração de argumento

variável local: args tipo: String[]

Ponto-e-vírgula é obrigatório no final de toda instrução

Atribuição de argumento para o método println()

Definição de classe HelloWorld

Chamada de método println() via objeto out acessível através da classe System

Abstração = Generalizar

ex: Se vou pegar o que importo pro mim de um obj deixando de lado o que não vou usar

Abstrato = é o que eu não posso tocar como uma avião por exemplo

comportamento
características
ou estados,
então é obj!!!.

Private X Só
Public X Modificadores
Protect X De Acesso.

Linguagens que utilizam POO

1. **Java:** Amplamente usado em aplicações empresariais, desenvolvimento web e desenvolvimento de aplicativos Android.
2. **C#:** Desenvolvido pela Microsoft, comumente usado para aplicativos Windows, desenvolvimento web com ASP.NET e desenvolvimento de jogos com Unity.
3. **Python:** Uma linguagem versátil usada para desenvolvimento web, computação científica, análise de dados, inteligência artificial, entre outros.
4. **C++:** Uma extensão da linguagem de programação C, frequentemente usada em desenvolvimento de sistemas/software, desenvolvimento de jogos e aplicações de alto desempenho.
5. **Ruby:** Conhecido por sua simplicidade e legibilidade, comumente usado no desenvolvimento web com o framework Ruby on Rails.
6. **PHP:** Principalmente usado para desenvolvimento web no lado do servidor, frequentemente integrado a bancos de dados.
7. **Swift:** Desenvolvido pela Apple para desenvolvimento de aplicativos iOS, macOS, watchOS e tvOS.
8. **Kotlin:** Uma linguagem moderna que interoperabiliza com o Java e é oficialmente suportada para o desenvolvimento de aplicativos Android.
9. **Smalltalk:** Uma das primeiras linguagens de programação orientadas a objetos, conhecida por sua simplicidade e influência em outras linguagens.
10. **Objective-C:** Tradicionalmente usado para desenvolvimento de aplicativos iOS e macOS, sendo gradualmente substituído pelo Swift.

classe = Molde do obj

Metodo → coisas que eu tenho

Atributo → coisas que eu faço

Estado → como estou agora

→ Método = função

instanciar é criar um objeto com as características da classe

c1 = Nova caneta

c1.cor = "Azul"

c1.ponta = 0.5

c1.tampada = False

→ c1.rabiscar() → método

O "this" serve para referenciar um atributo, dentro da classe, fora do método.

Dica: O ideal é que os métodos não escrevam

O que é uma instância em POO

?

R: É um obj cujo comportamento e estado são definidos pela classe.

Uma boa analogia é pensar em uma classe como se fosse uma forma de biscoitos e os biscoitos como se fossem os objetos criados a partir da classe

Para criar um constutor em Java basta criar um método com o mesmo nome da classe

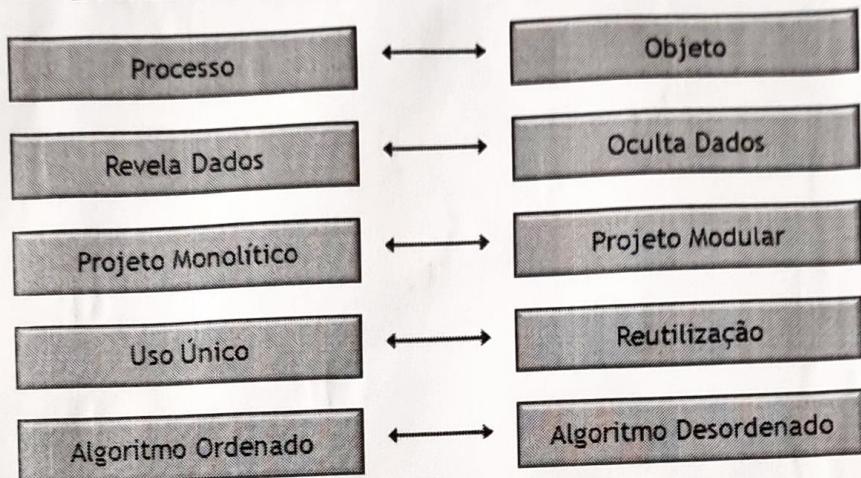
Interface é o contato com o mundo externo

O que é uma classe abstrata?

É um tipo de classe especial que não pode ser instanciada, apenas herdada

Programação Estruturada

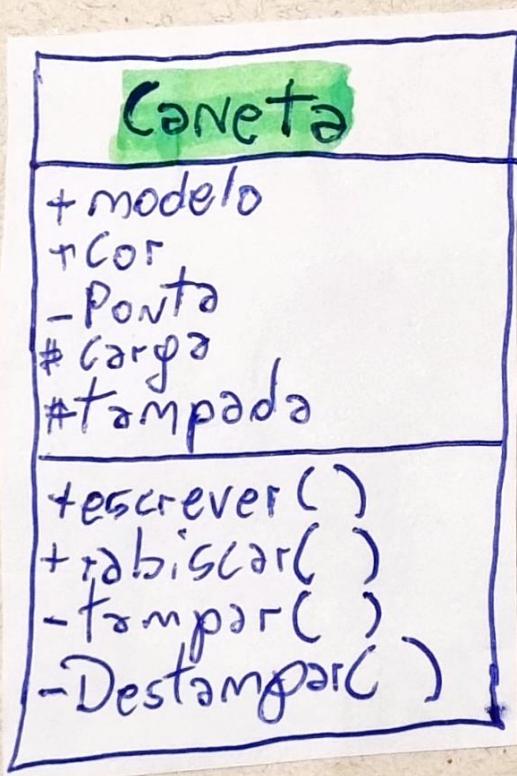
Orientação a Objeto



+ Público → A classe atual e todos as outras classes tem acesso

- Privado → Somente a classe atual vai ter acesso.

Protegido → A classe atual e todas as subclasses têm acesso



A função do `toString`
é retornar um `obj String`

Classe

- Representação de um conjunto de objetos com características afins. Definição do comportamento dos objetos (métodos) e seus atributos (atributos).

Objeto

- Uma instância de uma classe.
- Armazenamento de estados através de seus atributos e reação a mensagens enviadas por outros objetos.

Herança

- Mecanismo pela qual uma classe (sub-classe) pode estender outra classe (super-classe), estendendo seus comportamentos e atributos.

Polimorfismo

- Princípio pelo qual as instâncias de duas classes ou mais classes derivadas de uma mesma super-classe podem invocar métodos com a mesma assinatura, mas com comportamentos distintos.

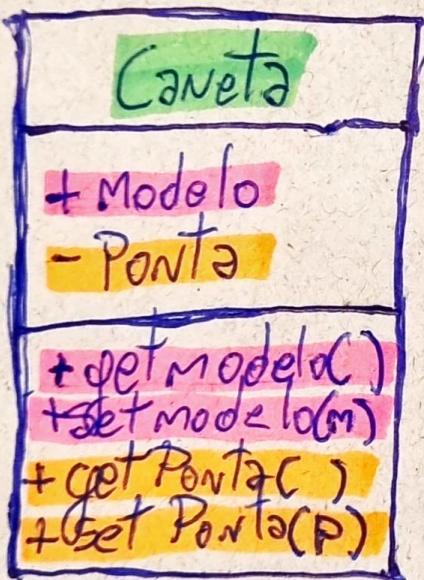
Encapsulamento

- Proibição do acesso direto ao estado de um objeto, disponibilizando apenas métodos que alterem esses estados na interface pública.

Métodos Acessores

Getters \Rightarrow Pegar Algo
Coisa Na Classe.

Setters \Rightarrow Para Modificar
alguma coisa na
classe.



Método construtor: Sempre que se
estanciar um obj ele vai executar
tudo que tem no método construtor

Classe

Agrupamento de objetos similares que apresentam os mesmos atributos e métodos.

Ex: Individuo, caracterizando as pessoas do mundo.

Método

Lógica contida em uma classe para designar-lhe um comportamento.

Ex: Cálculo da idade de uma pessoa em uma classe.

Estado

Situação de um objeto em um dado instante de tempo.

Ex: Informando a idade da pessoa.

Generalização

Atributos e operações comuns compartilhados por classes.

Ex: Superclasse [Parte] ou [Pessoa] como generalização das subclasses [Organização] e Individuo.

Objeto

Elemento do mundo real; sinônimo de instância de classe.

Ex: Uma pessoa "Fulano de Tal".

Atributo

Característica particular de uma ocorrência da classe.

Ex: Dados do Individuo (nome, sexo, data de nascimento).

Mensagem

Uma solicitação entre objetos para invocar certo método.

Ex: Informar a idade da pessoa.

Evento

Uma ocorrência significativa no mundo real que deve ser tratada.

Ex: Idade informada, entrega efetuada, etc.

classe Caneta

publico modelo: caractere

privado ponta: real

publico metodo setModelo

(m:caractere)

modelo = m

Fim metodo

publico metodo getModelo

retorne modelo

Fim metodo

publico metodo setPonta

(P:real)

Ponta = P

Fim Modelo

publico metodo getPonta

retorne: Ponta

Fim metodo

Fim classe

Herança

Polimorfismo

POO



Encapsulamento

Abstração

ABSTRAÇÃO

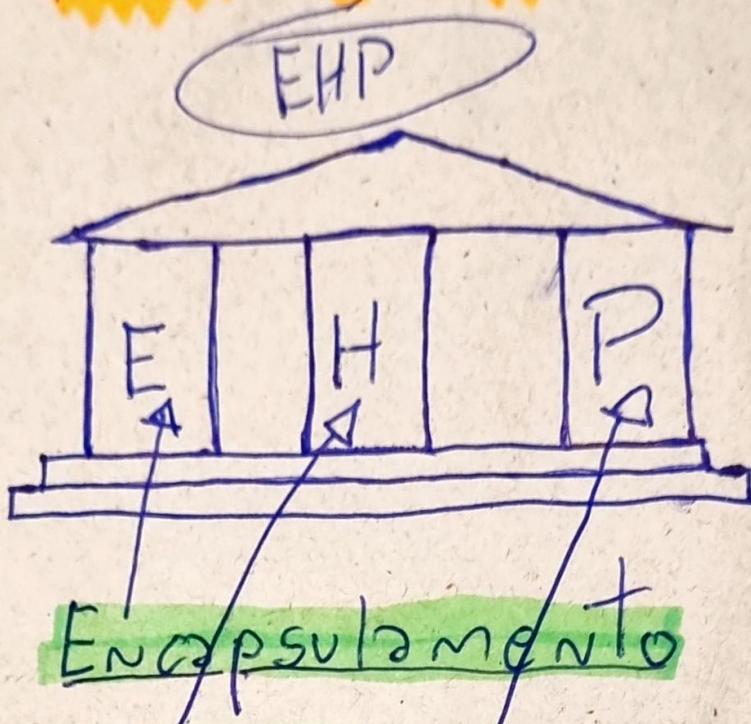
Abstração é o processo de extrair as características essenciais de um objeto real. A abstração é necessária para se ter um modelo fiel da realidade sobre a qual se possa operar.

<http://www.slideshare.net/danielrpgj30/curso-de-programao-orientada-a-objetos>

O conjunto de características resultante da abstração forma um tipo de dado abstrato com informações sobre seu estado e comportamento.

<http://www.slideshare.net/danielrpgj30/curso-de-programao-orientada-a-objetos>

Pilares Da POO

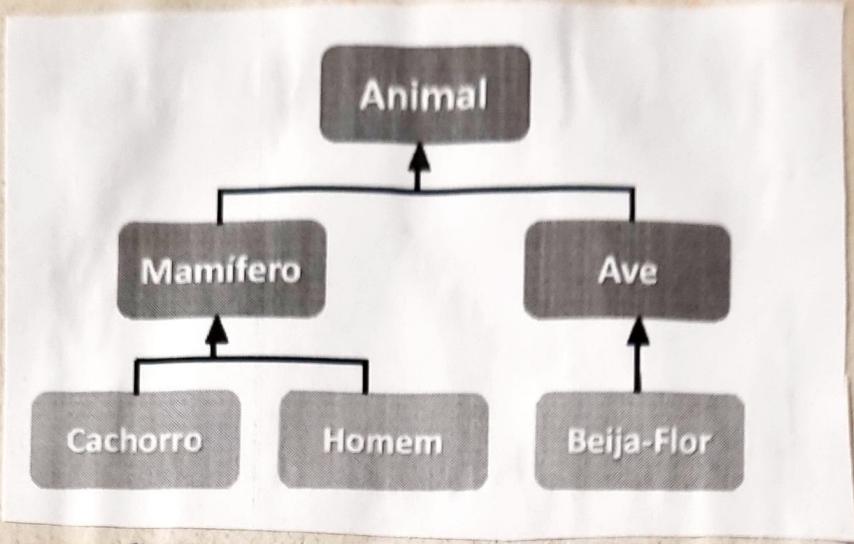


Herança
Polimorfismo

Algumas bibliografias falam sobre quatro pilares e incluem a Abstração

IMPORTANTE ↴

! encapsular Não é obrigatório, mas é uma boa prática para produzir classes mais eficientes.



Encapsular: Ocultar partes independentes da implementação, permitindo construir partes invisíveis ao Mundo exterior



Ex: quando va compra uma pilha não precisa se preocupar com o que tem dentro dela

Interface: Lista de serviços fornecidos por um componente. é o contato com o Mundo exterior, que define o que pode ser feito com um obj dessa classe.



ex: e como se fosse os pdos + e - de uma pilha.

Vantagens:

1 - Fornecer mudanças invisíveis.

2 - Facilitar reutilização de código.

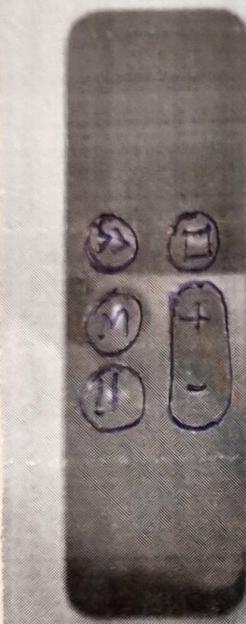
3 - Reduzir efeitos colaterais.

[outro programador não precisa mecher dentro da capsula]

<<interface>>
Controlador

- + ligar()
- + desligar()
- + abrirMenu()
- + fecharMenu()
- + maisVolume()
- + menosVolume()
- + ligarMudo()
- + desligarMudo()
- + play()
- + pause()

Potops do
Controle



implementa
tudo que tem
na interface.

ControleRemoto

- volume
- ligado
- tocando
- + ligar()
- + desligar()
- + abrirMenu()
- + fecharMenu()
- + maisVolume()
- + menosVolume()
- + ligarMudo()
- + desligarMudo()
- + play()
- + pause()

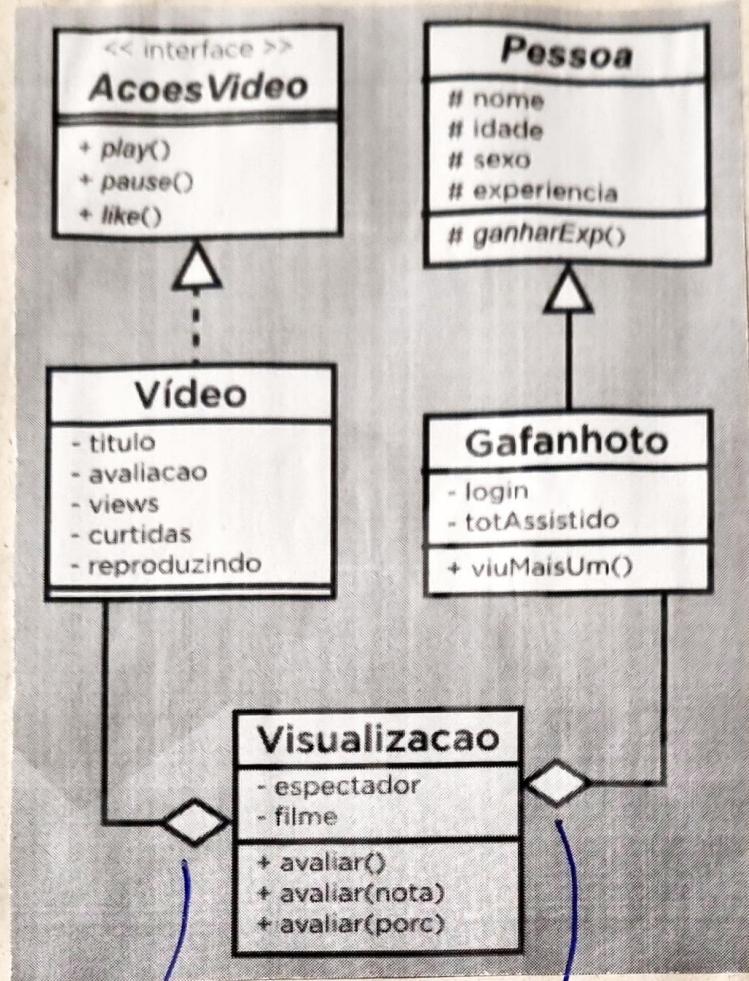
Parte interna
do Controle,
onde o user
não precisa
ter acesso.

Uma interface é uma maneira de declarar o comportamento de uma classe. Nessa declaração não especificamos exatamente como acontece internamente cada comportamento.

indica que a classe
implementa a interface



A Main implementa
a classe e classe
implementa a interface.

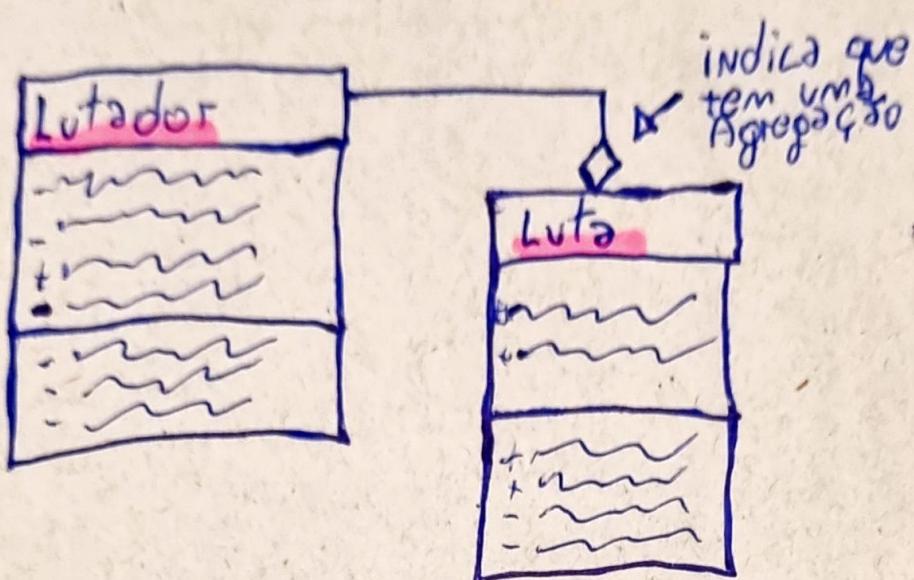


Public class Visualizacao {

 Private Gafanhoto espectador;
 Private Vídeo filme;

}

Relacionamento de Agregação.



Agregação é uma forma especial de associação utilizada para mostrar que um tipo de obj é composto, pelo menos em parte, de outro em uma relação todo/Parte

(Interface utiliza "implements")
Herança utiliza "extends"

O que é Composição?

A composição é simplesmente um obj que conta com outros objetos para compor sua estrutura

```
const Cozinha = estilo => `Cozinha ${estilo}`;
const Banheiro = tipo => `Banheiro do tipo ${tipo}`;
const Quarto = tipo => `Quarto ${tipo}`;

class Imovel {
    constructor(endereco) {
        this.endereco = endereco;
    }
}

const minhaCasa = new Imovel("rua abc");
minhaCasa.cozinha = Cozinha('Moderna');
minhaCasa.lavabo = Banheiro('Lavabo');
minhaCasa.banheiro = Banheiro('Completo');
minhaCasa.suite = Quarto('Suite');

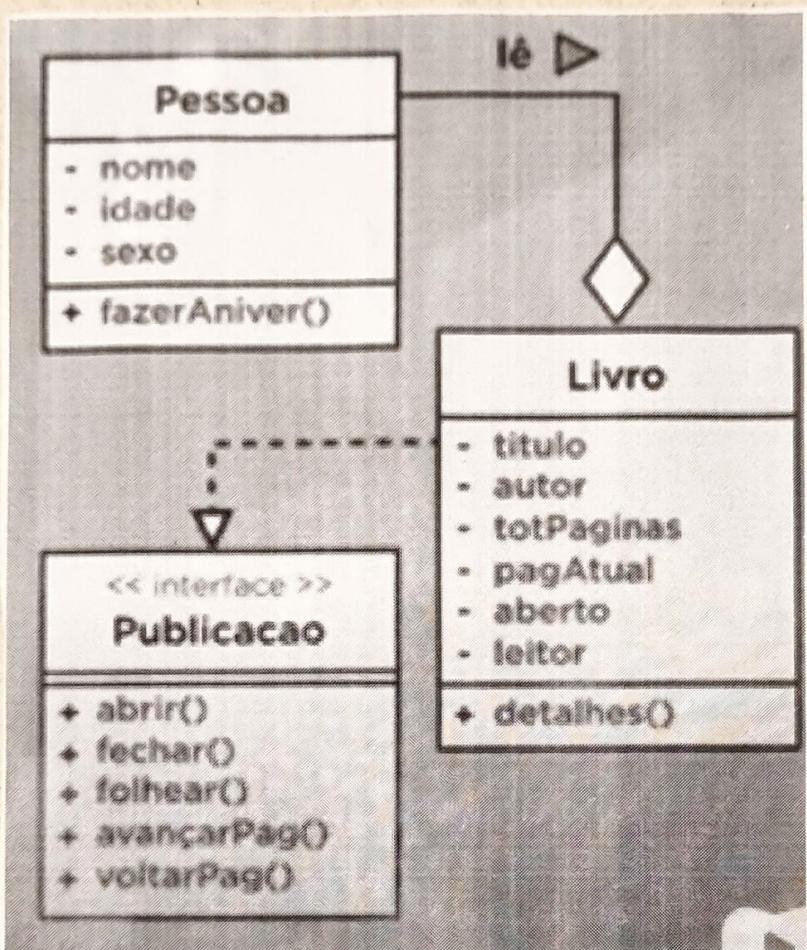
console.log(minhaCasa.suite);
// Quarto Suite
console.log(minhaCasa.lavabo);
// Banheiro do tipo Lavabo
```

O que é agregação e composição?

A agregação faz parte de um relacionamento de associação. A composição faz parte de uma relação de associação. A agregação é considerada um tipo fraco de associação. A composição é considerada um tipo forte de associação.

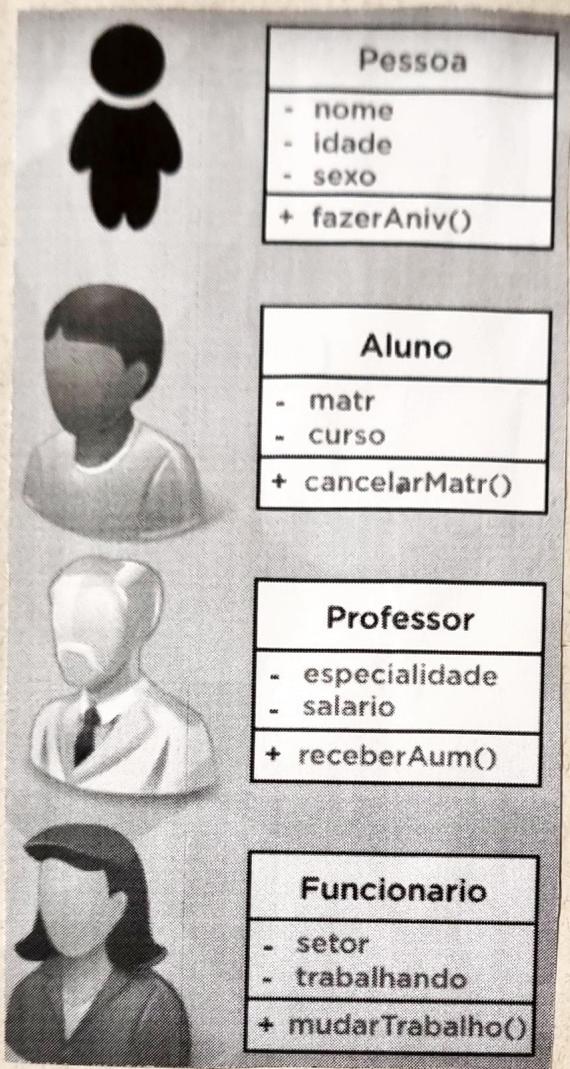
A POO se preocupa em produzir software que tenham as seguintes características

Natural / Confiável / Reutilizável
Manutencível / Extensível / Oportuno.



No Exemplo de uma empresa podemos dizer que a empresa é uma composição de departamentos e os departamentos são uma agregação de funcionários

Um funcionário pode trabalhar em mais de um departamento, porém um departamento não pode fazer parte de outra empresa.

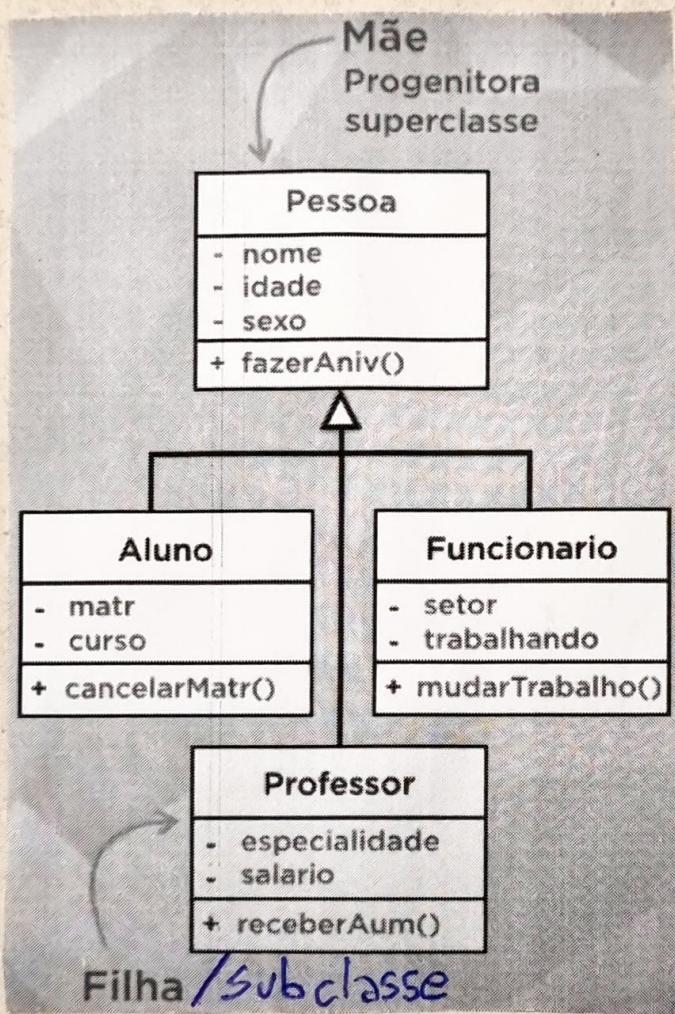


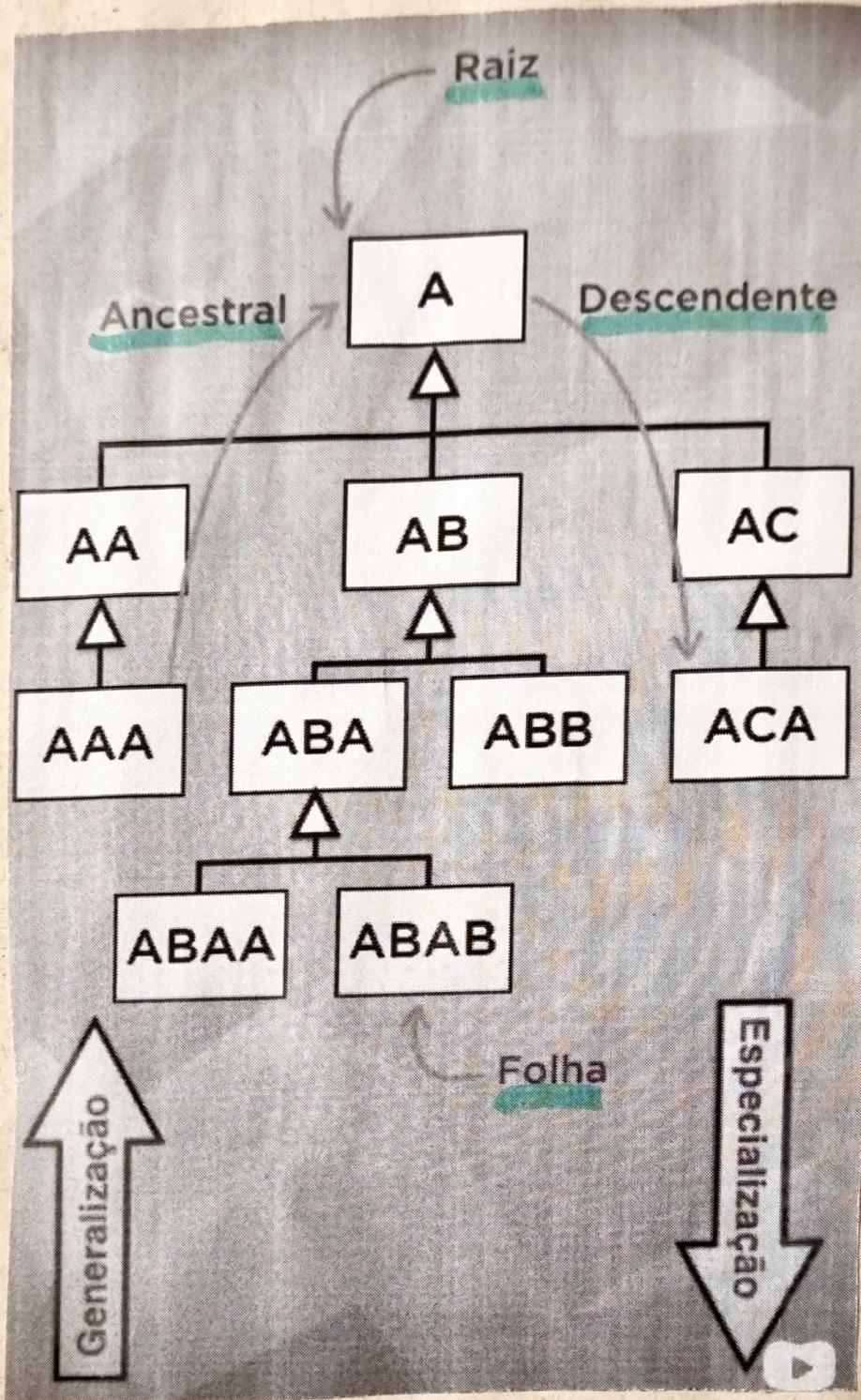
Herança

Permite basear uma nova classe na definição de uma outra classe previamente existente.

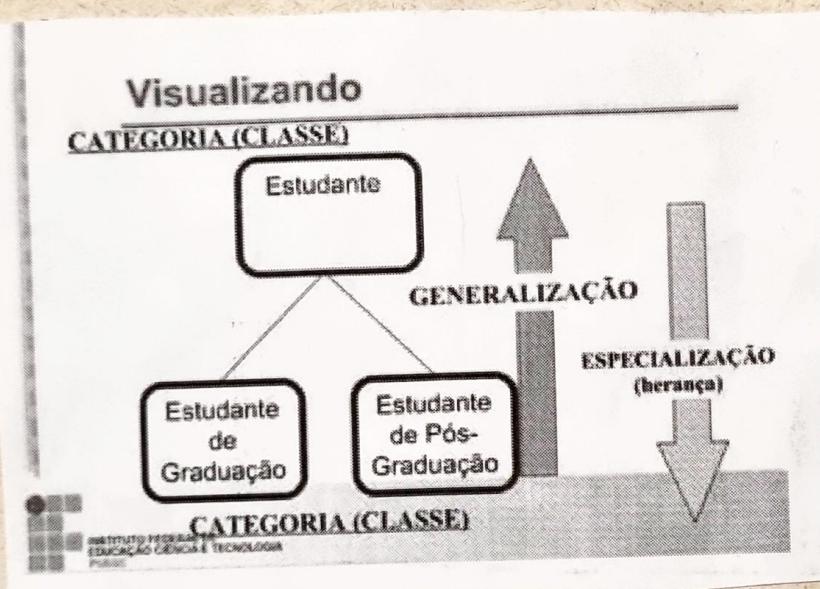


A herança será aplicada tanto para as características quanto para os comportamentos.

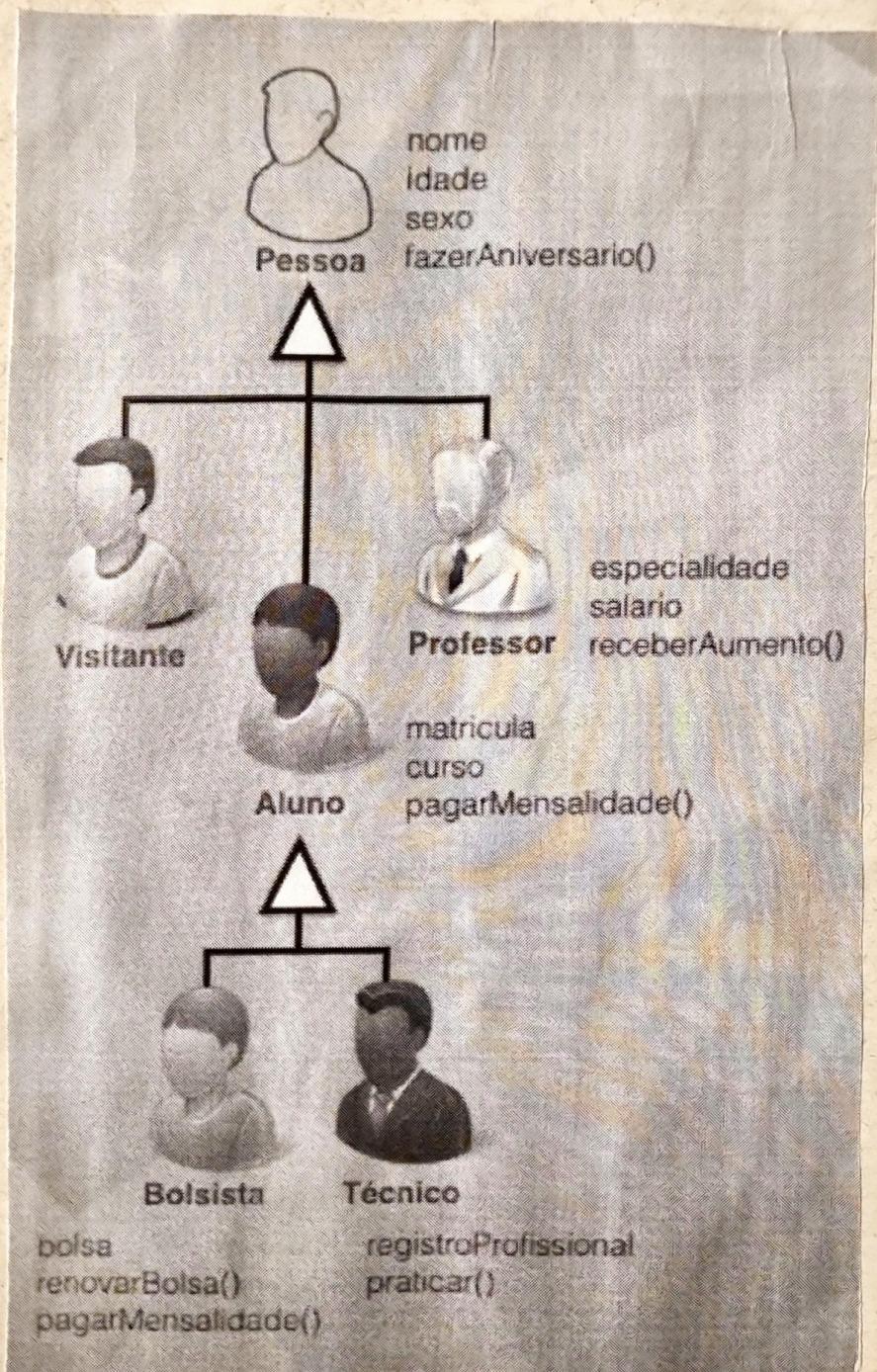




- Uma classe Progenitora/superclasse também pode ser uma subclasse dependendo do ponto de vista.
- Uma classe que não é subclasse é chamada de raiz.
- Uma classe que não tem subclasse é chamada de folha.
- Para definir um Ancestral ou descendente eu não conto a mãe
- percorrer a árvore de cima para baixo é chamado Especialização.
- Percorrer a árvore de baixo para cima é chamado de Generalização.



Obs: Alguns livros falam sobre
varios tipos de heranca
mas os dois principais
sao esses.



Herança de Implementação:

- Uma classe pode herdar o código (métodos e propriedades) de outra classe.
- Isso ajuda a reutilizar o código e compartilhar funcionalidades.

Herança para Diferença:

- Uma classe base define uma interface (métodos a serem implementados) que as subclasses devem seguir.
- Cada subclass fornece sua própria implementação para esses métodos.

Resumindo, herança de implementação é sobre reutilizar código, enquanto herança para diferença é sobre garantir que as classes filhas sigam uma interface específica, implementando métodos definidos pela classe base.

Classe Abstrata

Não pode ser instanciada.
Só pode servir como
progenitora.

Método Abstrato

Declarado, mas não
implementado na
progenitora.

Classe Final

Não pode ser herdada por
outra classe.
Obrigatoriamente folha.

Método Final

Não pode ser sobrescrito
pelas suas sub-classes.
Obrigatoriamente herdado.



```
classe abstrata Pessoa
    privado nome: Caractere
    privado idade: Inteiro
    privado sexo: Caractere
    publico metodo final fazerAniv()
        ...
    fimMetodo

FimClasse

classe Visitante estende Pessoa
    ...
FimClasse

classe Aluno estende Pessoa
    privado matricula: Inteiro
    privado curso: Caractere
    publico metodo PagarMensalidade()
        ...
    fimMetodo

FimClasse

classe Bolsista estende Aluno
    privado bolsa: Inteiro
    publico metodo RenovarBolsa()
        ...
    fimMetodo
    @Sobrepor
    publico metodo PagarMensalidade()
        ...
    fimMetodo

FimClasse

// Programa Principal
p1 = novo Pessoa()
v1 = novo Visitante()
a1 = novo Aluno()
b1 = novo Bolsista()
```

• classe abstrata não pode ser estanciada apenas herdada



A classe abstrata não pode ter um obj criado apartir da sua instanciação.

O que diferencia uma classe abstrata de uma interface?

Uma **interface** não pode conter campos, construtores, ou destrutores. Pode possuir apenas a propriedade da assinatura, mas não a implementação. Uma **classe abstrata** não suporta múltiplas herança. Assim, uma classe pode implementar várias **interfaces**, mas apenas herdar de uma **classe abstrata**.

A **classe abstrata** pode fornecer código completo, código padrão ou ter apenas a declaração de seu esqueleto para ser posteriormente sobrescrita.

ASSINATURA DO MÉTODO

Quantidade e os tipos dos parâmetros

```
publico metodo calcMedia(n1: Real,  
                         n2: Real): Real
```

```
publico metodo calcMedia(v1: Real,  
                         v2: Real): Inteiro
```

```
publico metodo calcMedia(bim: Inteiro,  
                         n1: Real,  
                         n2: Real): Real
```

```
publico metodo calcMedia(n1: Real,  
                         n2: Real,  
                         n3: Real,  
                         n4: Real): Real
```

```
publico metodo calcMedia(medMin: Real,  
                         medMax: Real,  
                         sit: Caractere,  
                         bim: Inteiro)  
                         :Caractere
```

• Método final não pode ser sobreescrito

Herança pobre = Não implementa nenhum método, utiliza todos os métodos da classe Mãe.

Método final não pode ser sobreposto (Override)

Polimorfismo

POLI = MUITAS

MORFO = FORMAS



Muitas formas de fazer
alguma coisa.



Quando um método têm
Muitas formas de Acontecer

Polimorfismo

Permite que um mesmo
Nome represente vários
comportamentos
Diferentes.

Assinatura do Método

Quantidade e os tipos dos
parâmetros

1

Polimorfismo de Sobreposição

- Mesma Assinatura
- Classes Diferentes

2

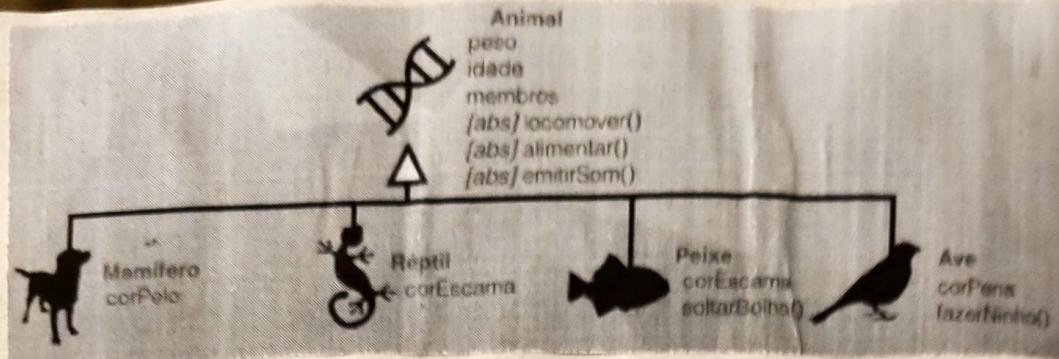
Polimorfismo de Sobrecarga

Assinaturas Diferentes
Mesma Classe

1

Polimorfismo de Sobreposição

Acontece quando substituímos um método de uma superclasse na sua subclasse, usando a mesma assinatura.



// Programa Principal

```

m = novo Animal()
m = novo Mamifero()
r = novo Reptil()
p = novo Peixe()
a = novo Ave()

```

```

m.setPeso(85.3)
m.setIdade(2)
m.setMembros(4)
m.locomover() // Correndo
m.alimentar() // Mamando
m.emitirSom() // Som de Mamifero

p.setPeso(0.35)
p.setIdade(1)
p.setMembros(0)
p.locomover() // Nadando
p.alimentar() // Comendo substâncias
p.emitirSom() // Peixe não faz som
p.soltarBolha()

a.setPeso(0.89)
a.setIdade(2)
a.setMembros(2)
a.locomover() // Voando
a.alimentar() // Comendo frutas
a.emitirSom() // Som de Ave
a.fazerNinho()

```

Este é um exemplo de polimorfismo de sobreposição.

Dicas de POO 3

Encapsulamento:

Utilize encapsulamento para ocultar os detalhes internos de uma classe e expor apenas as interfaces necessárias.

Defina atributos como privados (ou protegidos) e forneça métodos públicos para acessá-los. Isso ajuda a manter a integridade dos dados e facilita futuras modificações na implementação interna da classe sem afetar o restante do programa.

Herança:

Use herança de forma consciente, evitando hierarquias profundas e excessivamente complexas. Prefira a composição quando for mais apropriado.

Evite herdar de classes apenas para reutilizar código. A herança deve ser usada quando há uma relação lógica de "é um" entre as classes.

Polimorfismo:

Aproveite o polimorfismo para criar interfaces mais flexíveis e genéricas.

Faça uso de interfaces e classes abstratas para definir contratos que as classes derivadas devem seguir. Isso facilita a substituição de objetos e a escrita de código mais modular.

Abstração:

Abstraia conceitos complexos para criar modelos mais simples e compreensíveis. Concentre-se nos aspectos essenciais e ignore detalhes desnecessários.

Utilize interfaces e classes abstratas para representar abstrações, permitindo que diferentes implementações possam ser usadas de maneira intercambiável.

Composição sobre Herança:

Prefira a composição em vez da herança sempre que possível. Isso promove uma estrutura mais flexível e menos acoplada.

Utilize interfaces e injeção de dependência para compor funcionalidades, permitindo que diferentes partes do sistema possam ser modificadas independentemente uma da outra.

Polimorfismo de Sobreposição



Sobrecarga é o mesmo método com assinaturas diferentes na mesma classe.

```
classe abstrata Animal  
    protegido peso: Real  
    protegido idade: Inteiro  
    protegido membros: Inteiro  
    publico metodo abstrato emitirSom()  
FimClasse
```

```
classe Mamifero estende Animal  
    protegido corPelo: Caractere  
    @Sobrepor  
    publico metodo emitirSom()  
        Escreva("som de Mamífero")  
    fimMetodo  
FimClasse
```

```
classe Lobo estende Mamifero  
    @Sobrepor  
    publico metodo emitirSom()  
        Escreva("Auuuuuuuuuuuuuu!")  
    fimMetodo  
FimClasse
```

```
classe Cachorro estende Lobo  
    @Sobrepor  
    publico metodo emitirSom()  
        Escreva("Au!Au!Au!")  
    fimMetodo  
FimClasse
```

Mesma Assinatura
Classes Diferentes
É SOBREPOSIÇÃO!



reagir()

falar frase

agradável: abanar e latir
agressiva: rosnar

horário do dia

manhã: abanar
tarde: abanar e latir
noite: ignorar

dono

é dono: abanar
não é: rosnar e latir

idade e peso

novo e leve: abanar
novo e pesado: latir
velho e leve: rosnar
velho e pesado: ignorar

```
classe Cachorro estende Lobo
publico metodo reagir(frase: Caractere)
fimMetodo
publico metodo reagir(hora, min: Inteiro)
fimMetodo
publico metodo reagir(dono: Logico)
fimMetodo
publico metodo reagir(idade: Inteiro,
                      peso: Real)
fimMetodo
FimClasse
```

Assinaturas diferentes
Mesma classe
É SOBRECARGA!