**Curso de Java POO**

**Aula 1:**

Vantagens POO:

“COMERNada”

Confiável -> O isolamento entre as partes gera software seguro. Ao alterar uma parte, nenhuma outra é afetada.

Oportuno -> Ao dividir tudo em partes, várias delas podem ser desenvolvidas em paralelo.

Manutenível -> Atualizar um software é mais fácil. Uma pequena modificação vai beneficiar todas as partes que usarem o objeto.

Extensível -> O software não é estático. Ele deve crescer para permanecer útil.

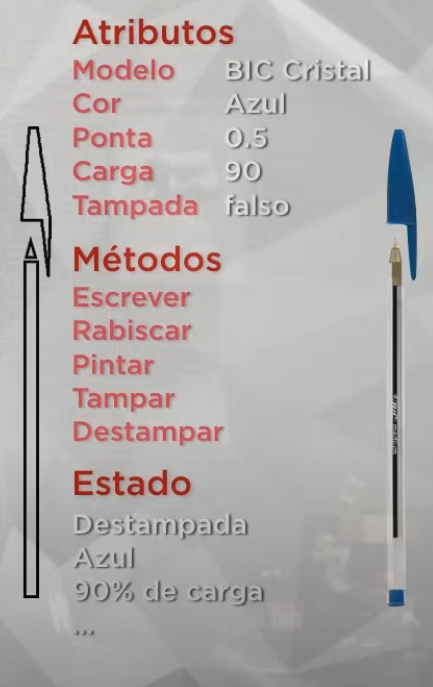
Reutilizável -> Podemos usar objetos de um sistema que criamos em outro sistema futuro.

Natural -> Mais fácil de entender. Você se preocupa mais na funcionalidade do que nos detalhes de implementação.

**Aula 2:**

OBJETO:

Definição - > Coisa material ou abstrata que pode ser percebida pelos sentidos e descrita por meio das suas características, comportamentos e estado atual.



Todo objeto vem de uma classe (molde).

Classe Caneta

modelo: Caractere

cor: Caractere

ponta: Real

carga: Inteiro

tampada: Lógico

Metodo rabiscar()

Se (tampada) entao

Escreva(“ERRO”)

senao

Escreva(“Rabisco”)

FimSe

FimMetodo

Metodo tampar()

tampada = verdadeiro

FimMetodo

FimClasse

Para criar um objeto é necessário instanciar uma classe. Gerando o objeto a partir da classe.



c1 e c2 são novos objetos criados a partir do instanciamento da classe Caneta.

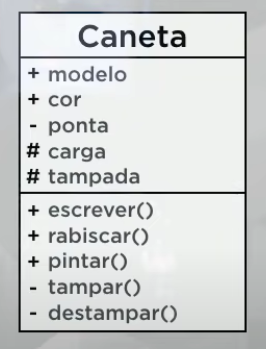
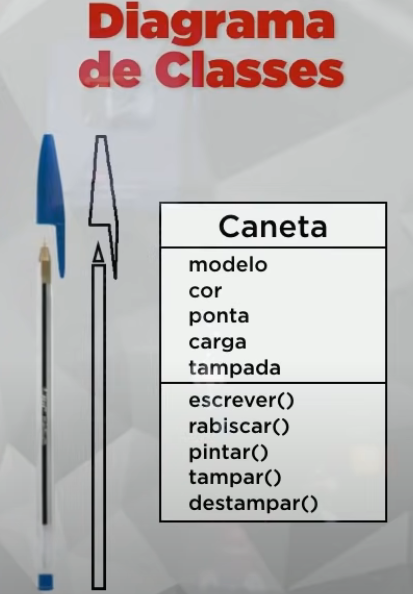
Depois de gerado ele herda os atributos e métodos da classe que podem ser alterados especificamente para ele como visto na imagem acima.

A classe define os atributos e métodos comuns que serão compartilhados por um objeto.

O objeto é a instância de uma classe.

\*\*Abstração é o princípio de ocultar os detalhes internos de implementação de um objeto e expor apenas as funcionalidades essenciais. Isso permite focar no "o que" um objeto faz, em vez de "como" ele faz.\*\*

**Aula 3:**



Modificadores de Visibilidade:

**+** -> public -> classe atual e todas as outras classes.

**-** -> private -> somente a classe atual.

**#** -> protected - > classe atual e todas as suas subclasses.

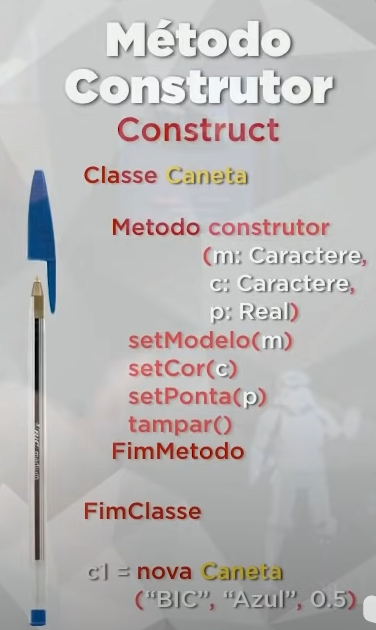
**Aula 4:**

Métodos Getters -> Acessam um atributo mantendo a segurança de acesso a ele.

Métodos Setters -> Modificam um atributo mantendo a segurança de acesso a ele.

Método Construct -> “Determina um padrão de criação do novo objeto.” Inicializa os atributos (ou propriedades) do objeto. Realiza qualquer configuração inicial necessária.

É executado automaticamente no momento em que o objeto é instanciado (chamando a classe como se fosse uma função).



**Aula 5:**

Montamos um banco como exemplo aplicando o que foi aprendido nas aulas anteriores.

**Aula 6:**

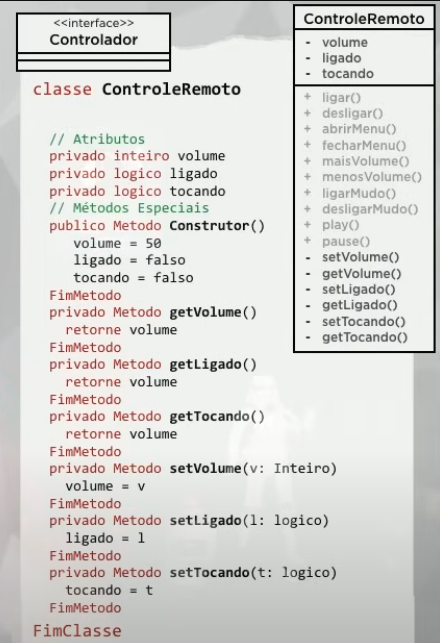
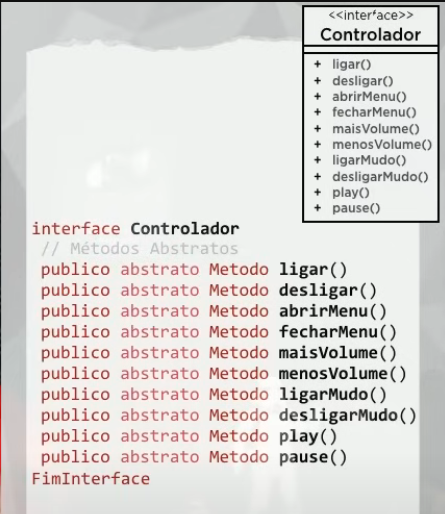
Pilares da POO: Encapsulamento

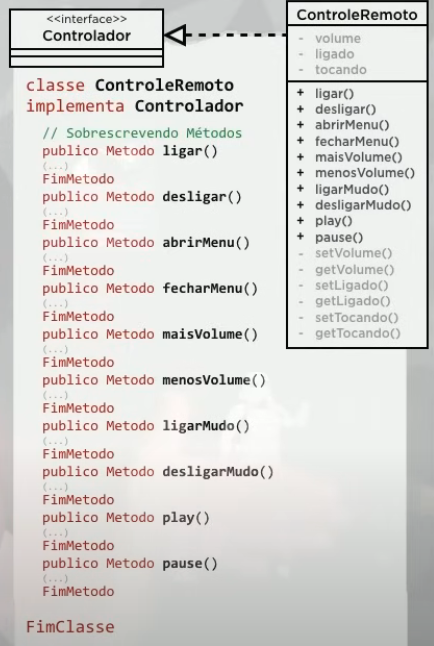
Conceito -> Ocultar partes independentes da implementação, permitindo construir partes invisíveis ao mundo exterior.

Princípio que controla o acesso aos dados e métodos de uma classe. Ele protege os atributos internos de alterações diretas, permitindo que sejam acessados ou modificados apenas por meio de métodos específicos (getters e setters). Assim, a classe mantém o controle sobre como seus dados são usados, garantindo maior segurança e organização.

Vantagens : Tornar mudanças invisíveis. Facilitar reutilização do código. Reduzir efeitos colaterais.

**Interface - >** É como um contrato que define um conjunto de métodos que uma classe precisa implementar. Ela não contém a implementação dos métodos, apenas a assinatura (o "o que" deve ser feito, não "como"). **É um guia para garantir que diferentes classes sigam um mesmo padrão**.





**Aula 7:**

Relacionamento entre Classes.

### **1. Associação**

A associação é um relacionamento básico onde uma classe **usa** outra, podendo ser unidirecional (apenas uma classe conhece a outra) ou bidirecional (ambas se conhecem).

* **Exemplo:** Um Pedido está associado a um Cliente.

**Código Exemplo:**

class Cliente {

private String nome;

public Cliente(String nome) {

this.nome = nome;

}

public String getNome() {

return nome;

}

}

class Pedido {

private Cliente cliente; // Associação unidirecional

public Pedido(Cliente cliente) {

this.cliente = cliente;

}

public void mostrarCliente() {

System.out.println("Pedido do cliente: " + cliente.getNome());

}

}

public class Main {

public static void main(String[] args) {

Cliente cliente = new Cliente("João");

Pedido pedido = new Pedido(cliente);

pedido.mostrarCliente();

}

}

### **2. Agregação**

A agregação é uma relação "parte-todo" onde as partes podem **existir separadamente** do todo.

* **Exemplo:** Um Departamento contém Funcionarios, mas os funcionários podem existir sem o departamento.

**Código Exemplo:**

import java.util.ArrayList;

import java.util.List;

class Funcionario {

private String nome;

public Funcionario(String nome) {

this.nome = nome;

}

public String getNome() {

return nome;

}

}

class Departamento {

private List<Funcionario> funcionarios = new ArrayList<>();

public void adicionarFuncionario(Funcionario funcionario) {

funcionarios.add(funcionario);

}

public void listarFuncionarios() {

for (Funcionario funcionario : funcionarios) {

System.out.println("Funcionário: " + funcionario.getNome());

}

}

}

public class Main {

public static void main(String[] args) {

Funcionario f1 = new Funcionario("Maria");

Funcionario f2 = new Funcionario("José");

Departamento departamento = new Departamento();

departamento.adicionarFuncionario(f1);

departamento.adicionarFuncionario(f2);

departamento.listarFuncionarios();

}

}

### **3. Composição**

A composição é um tipo de relação "parte-todo" mais forte, onde a **parte não pode existir sem o todo**.

* **Exemplo:** Um Carro possui um Motor. Se o carro for destruído, o motor também será.

**Código Exemplo:**

class Motor {

private String tipo;

public Motor(String tipo) {

this.tipo = tipo;

}

public String getTipo() {

return tipo;

}

}

class Carro {

private Motor motor;

public Carro(String tipoMotor) {

this.motor = new Motor(tipoMotor); // Composição

}

public void mostrarMotor() {

System.out.println("Tipo de motor: " + motor.getTipo());

}

}

public class Main {

public static void main(String[] args) {

Carro carro = new Carro("V8");

carro.mostrarMotor();

}

}

### **4. Herança**

A herança descreve uma relação **"é um"**, onde uma classe filha herda os atributos e métodos de uma classe pai.

* **Exemplo:** Um Cachorro é um tipo de Animal.

**Código Exemplo:**

class Animal {

public void comer() {

System.out.println("Animal está comendo");

}

}

class Cachorro extends Animal {

public void latir() {

System.out.println("Cachorro está latindo");

}

}

public class Main {

public static void main(String[] args) {

Cachorro cachorro = new Cachorro();

cachorro.comer(); // Método herdado

cachorro.latir(); // Método próprio

}

}

### **5. Dependência**

A dependência é um relacionamento transitório onde uma classe **usa outra apenas temporariamente** para realizar uma tarefa.

* **Exemplo:** Uma Impressora depende de um Documento para imprimir, mas não mantém referência ao documento.

**Código Exemplo:**

class Documento {

private String texto;

public Documento(String texto) {

this.texto = texto;

}

public String getTexto() {

return texto;

}

}

class Impressora {

public void imprimir(Documento doc) { // Dependência

System.out.println("Imprimindo documento: " + doc.getTexto());

}

}

public class Main {

public static void main(String[] args) {

Documento documento = new Documento("Contrato de aluguel");

Impressora impressora = new Impressora();

impressora.imprimir(documento);

}

}

### **Resumo das Diferenças**

| **Tipo** | **Ciclo de Vida das Classes** | **Relacionamento** | **Exemplo** |
| --- | --- | --- | --- |
| Associação | Independente | Usa | Pedido e Cliente |
| Agregação | Independente | Parte-todo (fraco) | Departamento e Funcionarios |
| Composição | Dependente | Parte-todo (forte) | Carro e Motor |
| Herança | Dependente (hierárquico) | É um | Animal e Cachorro |
| Dependência | Transitório | Precisa de | Impressora e Documento |

**Aula 8:**

Uma **instância de classe** pode ser usada como um tipo abstrato para um atributo de outra classe. Isso é feito criando um relacionamento entre as classes, conhecido como **composição** ou **agregação**.

### **Exemplo:**

class Endereco {

String rua;

String cidade;

String estado;

public Endereco(String rua, String cidade, String estado) {

this.rua = rua;

this.cidade = cidade;

this.estado = estado;

}

}

class Pessoa {

String nome;

int idade;

Endereco endereco; // Atributo de tipo abstrato (composição)

public Pessoa(String nome, int idade, Endereco endereco) {

this.nome = nome;

this.idade = idade;

this.endereco = endereco;

}

}

### **Explicação:**

1. **Classe Endereco** é um tipo abstrato que encapsula dados relacionados ao endereço.
2. **Classe Pessoa** tem um atributo endereco que é uma instância da classe Endereco.
3. O atributo endereco em Pessoa permite que a relação "Pessoa tem um Endereço" seja representada de maneira clara e modular.

### **Uso:**

public class Main {

public static void main(String[] args) {

Endereco endereco = new Endereco("Rua A", "Niterói", "RJ");

Pessoa pessoa = new Pessoa("João", 30, endereco);

System.out.println(pessoa.nome + " mora em " + pessoa.endereco.cidade);

}

}

### **Vantagens:**

* **Modularidade:** A lógica do endereço está separada em sua própria classe.
* **Reutilização:** A classe Endereco pode ser usada em outras classes.
* **Flexibilidade:** Facilita a manutenção e a extensão do código.

Essa abordagem é fundamental para projetar sistemas bem organizados e escaláveis em POO.

Outra forma de gerar um número aleatório:



**Aula 9:**

Exercícios POO:

01. Uma casa está para uma planta arquitetônica assim como um objeto está para uma classe.

02. A POO se preocupa em produzir software que tenha as seguintes características:

Natural, Confiável, Reutilizável, Manutenível, Extensível e Oportuno.

03. Um objeto é a instância de uma classe.

Um objeto é uma construção de software que encapsula estado e comportamento.

Uma classe define os atributos e comportamentos compartilhados por um tipo de objeto.

Em um linguagem POO pura, tudo é um objeto, desde os tipos mais básicos.

04. Atributo > Característica de uma classe que é visível.

Classe > Define os atributos e comportamentos comuns.

Comportamento > Ação executada por um objeto.

Domínio > Espaço onde o problema reside.

Objeto > Construção de software que encapsula estado e comportamento.

05. Uma variável interna é um valor mantido dentro do objeto.

Atributos são características de uma classe visíveis externamente.

Comportamento são funcionalidades internas de uma classe.

O estado de um objeto é o significado combinado das variáveis internas do objeto.

06. Construtor > métodos usados para inicializar objetos durante a instanciação.

Acessor > métodos que dão acesso aos dados internos.

Mutante > métodos que permitem que se altere o estado de um objeto.

Tipos > define as diferentes espécies de valores que podem ser usados.

07. Quais são os três pilares da POO?

Encapsulamento, herança e polimorfismo.

08. Classe > define os atributos e comportamentos compartilhados.

Interface > define o que uma entidade pode fazer com o objeto.

Construtor > utilizado para inicializar objetos.

Objeto > instanciação de uma classe.

09. A maioria das POO suporta quais níveis de acesso?

public, private, protected.

10. protected > acesso ao objeto e suas subclasses.

11. Encapsulamento > A proteção de atributos e operações das classes, fazendo com que estas se comuniquem com o meio externo por meio de suas interfaces.

12. Quando se utiliza o conceito de encapsulamento da POO, enquanto a IMPLEMENTAÇÃO define os detalhes internos do componente, a INTERFACE lista os serviços fornecidos por ele.

13. Encapsulamento é a característica da POO que permite separar o programa em várias partes menores e independentes cada parte possui sua implementação isolada e realiza seu trabalho de forma autônoma. Com essa característica é possível ocultar os detalhes internos de cada parte através de uma interface.

14. Abstração é o conceito de POO que significa representar uma entidade, incluindo apenas seus atributos mais relevantes.

15. Classes são tipos abstratos de dados.

Objetos são instâncias de uma classe.

Subclasse é uma classe definida por meio de outra classe.

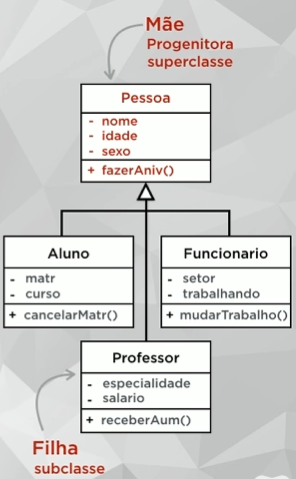
Métodos são subprogramas que definem as operações em objetos de uma classe.

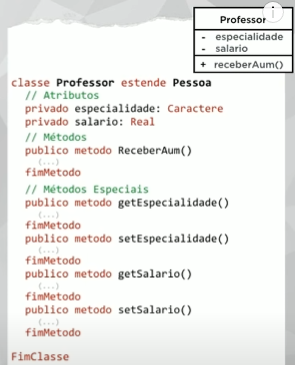
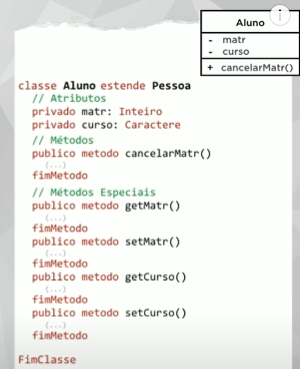
**Aula 10:**

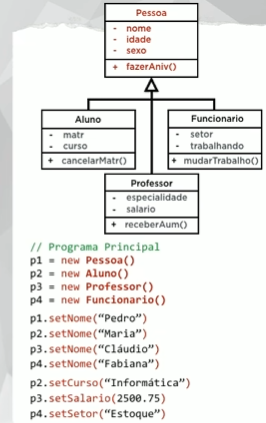
Herança

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

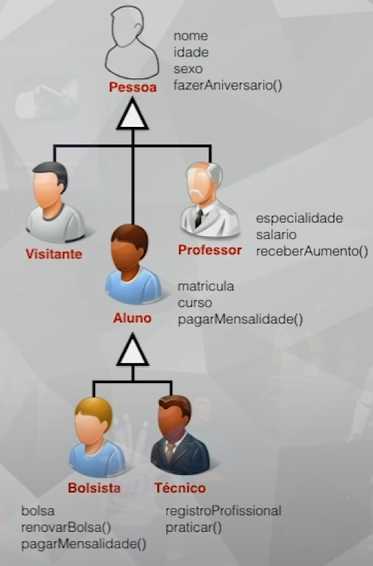
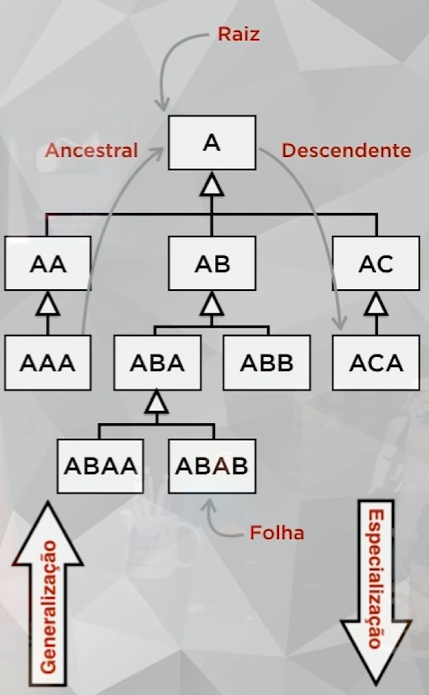
A herança será aplicada tanto para as características quanto para o comportamento.

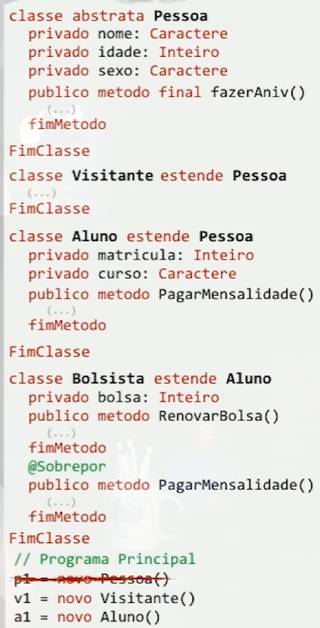
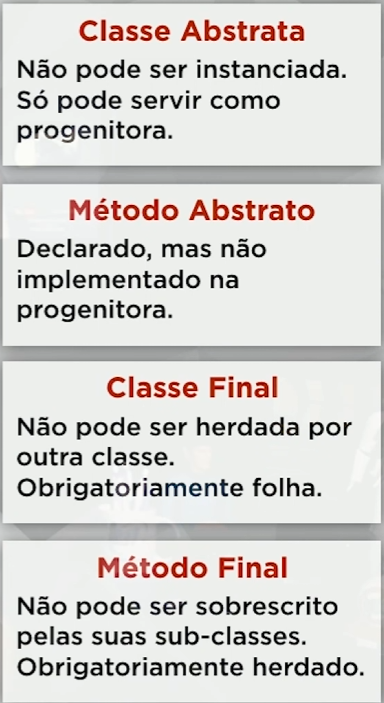






**Aula 11:**

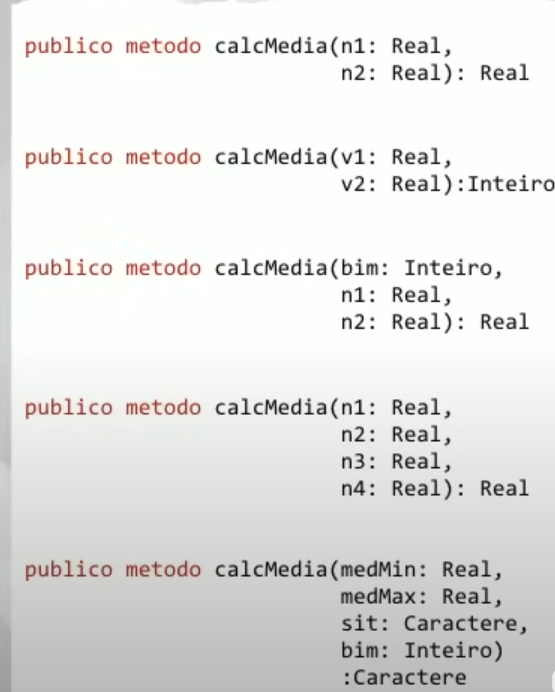




**Aula 12:**

Polimorfismo

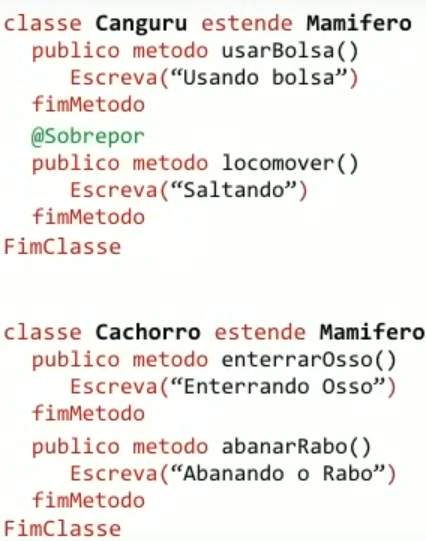
Assinatura do método: Quantidade e tipo dos parâmetros.



Os dois primeiros possuem a mesma assinatura. E todos os outros são diferentes.

Polimorfismo de Sobreposição:

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



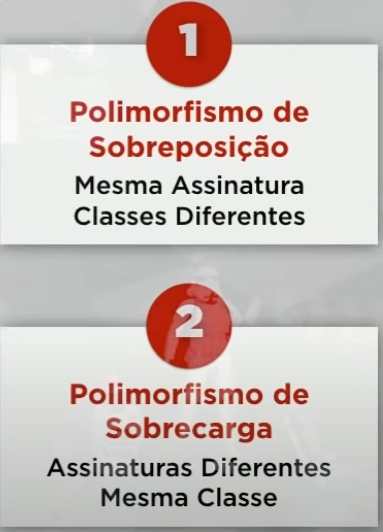
@Sobrepor = @Override

**Aula 13:**

Polimorfismo de Sobrecarga



Comparando sobreposição com sobrecarga.



**Aula 14:**

Questão 01:

Segundo Deitel (2010) com relação à linguagem de programação Java, é correto afirmar que:

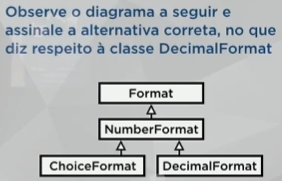
O compilador fornece um construtor padrão sem parâmetros em qualquer classe que não inclui explicitamente um construtor.

Questão 02:

Uma das formas de reutilização de software é a criação de uma classe que absorve membros de outra classe já existente. A nova classe é aprimorada com capacidades novas ou modificadoras.

Herança.

Questão 03:

 DecimalFormat é descendente de Format

Questão 04:

Segundo Deitel (2010), fornecer uma superclasse apropriada a partir da qual outras classes podem herdar e, assim, compartilhar um design comum, é o propósito de…

Uma classe Abstrata.

Questão 05:

Entende-se por método abstrato um método…

Declarado, mas não implementado.

Questão 06:

Sobre herança em POO:

A herança permite basear uma nova classe na definição de outra classe.

Pode-se sobrepor um comportamento de uma superclasse em uma subclasse.

Com o mecanismo de herança, é possível estabelecer relacionamentos do tipo “é um” entre classes.

Questão 07:

Ao programar em POO e declarar uma classe com a palavra-chave “final" pode se afirmar que…

A classe não poderá ter subclasses.

Questão 08:

É possível transformar um método private para protected ou public, mas não ao contrário.

Questão 09:

Para programar em Java utiliza-se o símbolo “.” para poder acessar as funções definidas em uma classe. Integer.parseInt().

Questão 10:

Segundo Deitel (2010), em POO, o conceito que permite escrever programas que processam objetos que compartilham a mesma superclasse (direta e indiretamente) como se todos fossem objetos dessa superclasse, é conhecido como POLIMORFISMO PARAMÉTRICO.

OBS: Estudar mais tipos de polimorfismo.

**Aula 15:**

Questão 01:

“de inclusão”, “paramétrico”, “sobrecarga” e “sobreposição” são quatro tipos de…

Polimorfismo.

Questão 02:

Quando se utiliza, dentro de uma mesma classe, um mesmo nome de método para muitos métodos diferentes, estamos utilizando o conceito de…

Sobrecarga.

Questão 03:

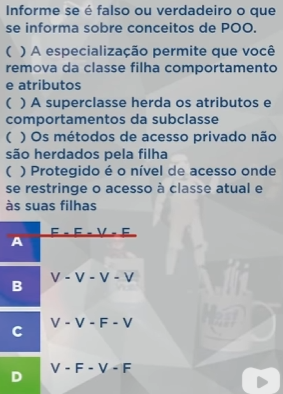
Classe Abstrata > nunca será instanciada.

Classe Final > não pode ser superclasse.

Método Abstrato > deve ter implementação na subclasse.

Método Final > não pode ser sobrescrito.

Questão 04:



Questão 05:

O polimorfismo permite que um único NOME expresse muitos comportamentos diferentes. Um tipo de polimorfismo é a SOBRECARGA, que permite que possa ser declarado o mesmo MÉTODO várias vezes, sendo que cada declaração difere simplesmente no(a) NÚMERO e TIPO de argumentos.

Questão 06:

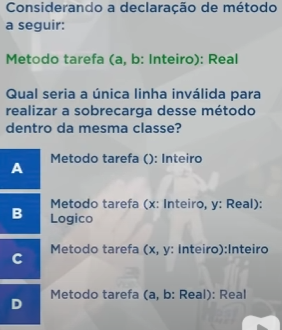
Faz parte da assinatura de um método:

1 - Seu nome

2 - A quantidade de parâmetros

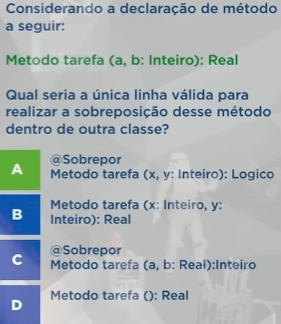
5 - Os tipos dos seus parâmetros

Questão 07:



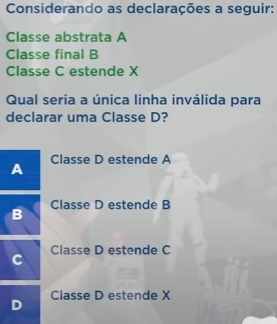
A Opção C seria a única inválida, pois possui a mesma assinatura. Ambos recebem 2 parâmetros e do mesmo tipo.

Questão 08:



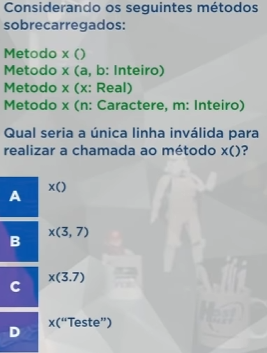
A Opção A seria a única válida, pois possui a mesma assinatura. Ambos recebem 2 parâmetros e do mesmo tipo.

Questão 09:



A Opção B seria a única inválida, pois uma Classe Final não pode ter subclasse.

Questão 10:



A opção D seria a única inválida, pois nenhum método recebe somente um String.