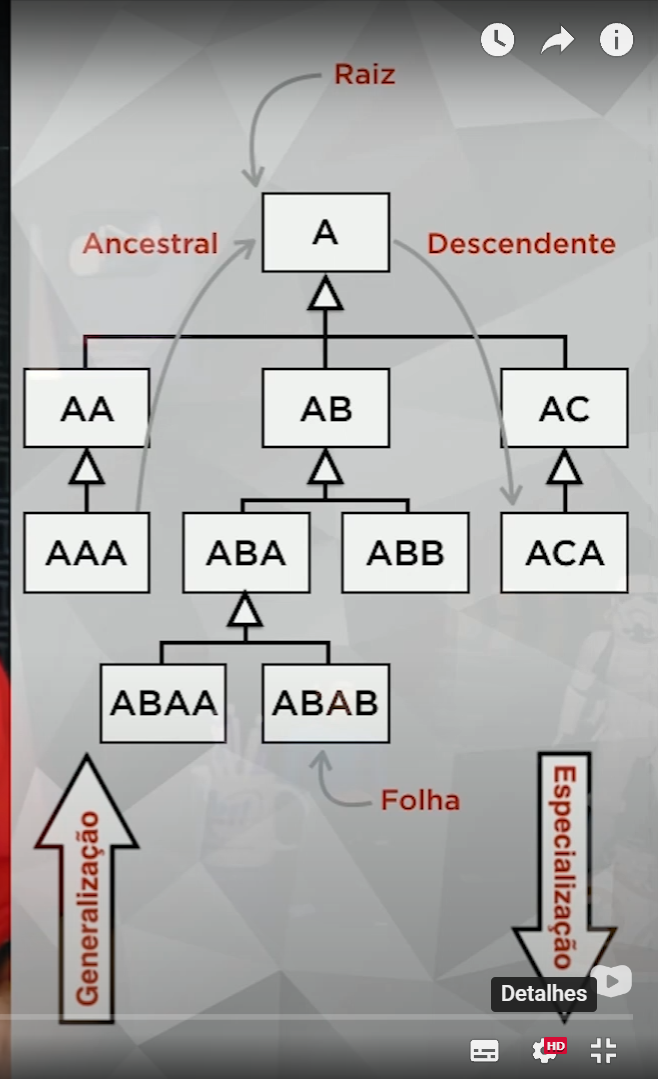
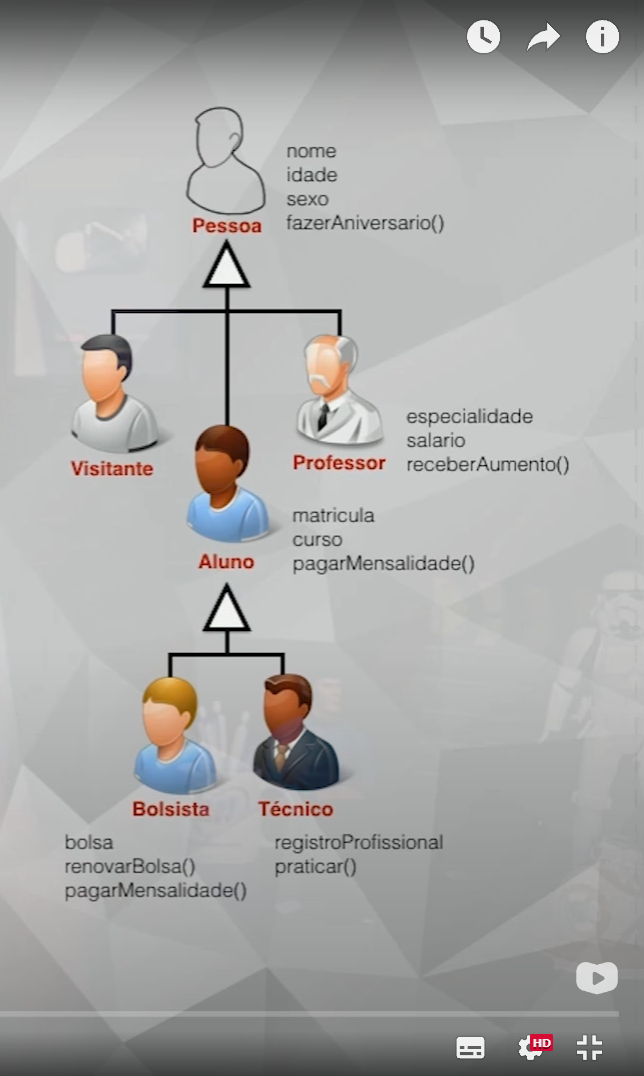
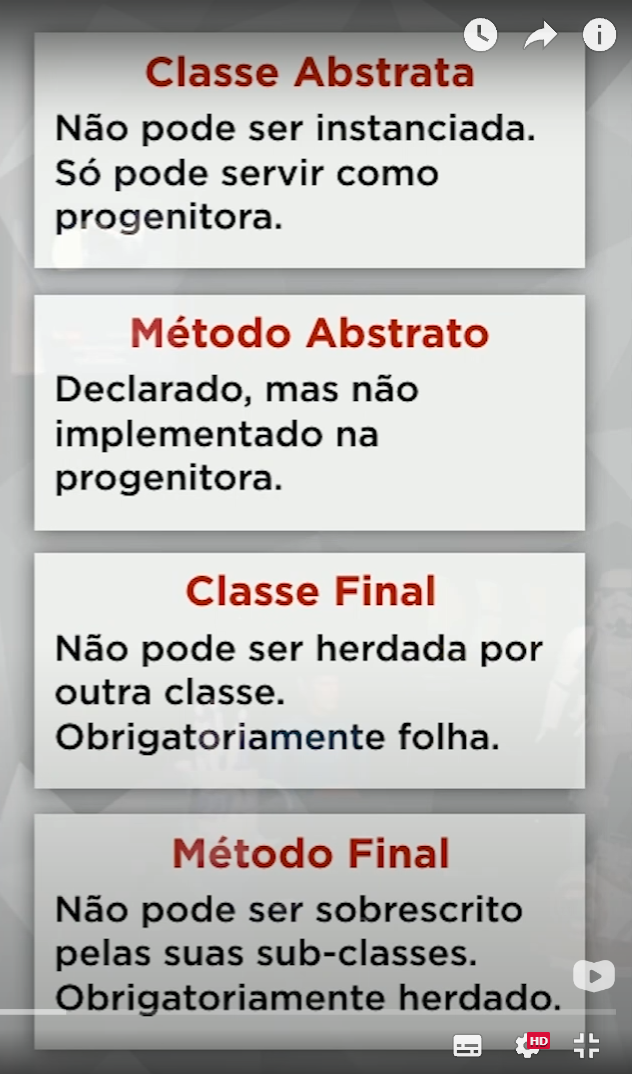
**Curso POO Teoria**[**#11a**](https://www.youtube.com/hashtag/11a)**- Herança (Parte 2) – Guanabara: Resumo**

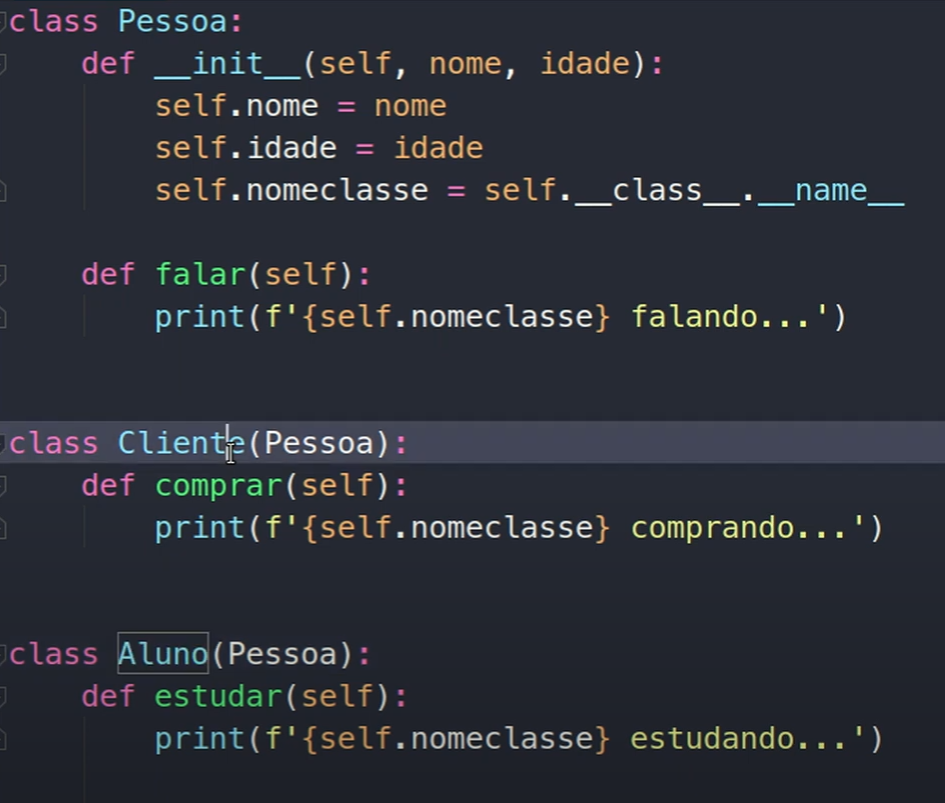
* **Herança de Implementação** e **Herança para Diferença**:
  + *Herança de Implementação* permite que uma subclasse herde e reutilize diretamente o código de uma superclasse.
  + *Herança para Diferença* é usada quando a subclasse redefine ou estende o comportamento da superclasse.
* **Nomenclaturas na Herança**:
  + **Superclasse** e **Subclasse**: A superclasse é a classe "mãe" que fornece atributos e métodos, enquanto a subclasse herda esses elementos.
  + **Ancestral** e **Descendente**: Ancestral é qualquer classe na cadeia de herança superior, e descendente é qualquer classe abaixo na cadeia.
  + **Classe Raiz** e **Classe Folha**: A classe raiz está no topo da hierarquia de herança, enquanto a classe folha é aquela que não é mais herdada.
* **Conceitos de Generalização e Especialização**:
  + *Generalização* é o processo de criar uma superclasse a partir de subclasses comuns.
  + *Especialização* é o oposto, onde subclasses derivam de uma superclasse para adicionar comportamentos específicos.
* **Classes e Métodos Específicos**:
  + **Classe Abstrata**: Não pode ser instanciada diretamente e geralmente serve como modelo para outras classes.
  + **Classe Final**: Não pode ser herdada por outras classes.
  + **Método Abstrato**: Definido na classe abstrata sem implementação, sendo obrigatório que as subclasses o implementem.
  + **Método Final**: Método que não pode ser sobrescrito pelas subclasses.

Esses conceitos formam a base para estruturar e organizar código em POO de forma eficiente e flexível.

**Herança**

Herança é um relacionamento onde uma classe (a subclasse) deriva de outra classe (a superclasse), herdando seus atributos e métodos. Esse relacionamento reflete uma relação "é um tipo de", onde a subclasse é um tipo mais específico da superclasse.



Neste código, o conceito de **herança** em POO está sendo utilizado para criar uma hierarquia entre as classes Pessoa, Cliente e Aluno. Vamos entender o que está acontecendo em cada parte:

### 1. Classe Pessoa

A classe Pessoa é a **superclasse** ou **classe base**. Ela contém um construtor (\_\_init\_\_) que inicializa os atributos nome e idade para cada instância de Pessoa. Além disso, ela define o atributo nomeclasse usando self.\_\_class\_\_.\_\_name\_\_, que armazena o nome da classe de cada instância criada.

A classe Pessoa também tem um método chamado falar, que exibe uma mensagem indicando que uma instância da classe está "falando".

### 2. Classe Cliente

A classe Cliente é uma **subclasse** de Pessoa, indicada pela sintaxe class Cliente(Pessoa):. Isso significa que Cliente herda todos os atributos e métodos de Pessoa.

Além dos atributos e métodos herdados, Cliente possui um método adicional chamado comprar, que imprime uma mensagem indicando que a instância Cliente está "comprando". Como Cliente herda de Pessoa, ela também pode acessar o método falar, além do método comprar.

### 3. Classe Aluno

A classe Aluno também é uma subclasse de Pessoa, definida como class Aluno(Pessoa):. Isso significa que Aluno herda tudo o que Pessoa possui.

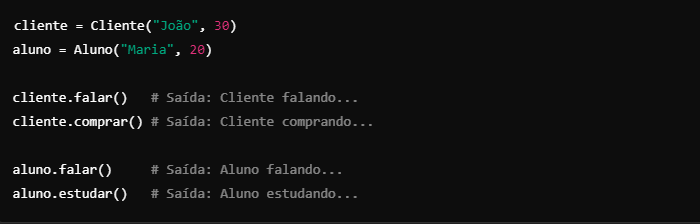
Aluno tem um método específico chamado estudar, que exibe uma mensagem indicando que a instância Aluno está "estudando". Assim como Cliente, Aluno também pode acessar o método falar de Pessoa, além de seu próprio método estudar.

### Como funciona a herança neste exemplo?

A herança permite que Cliente e Aluno reutilizem o código da classe Pessoa, evitando a duplicação. Ambos herdam o construtor \_\_init\_\_, o atributo nomeclasse, e o método falar. Cada subclasse (Cliente e Aluno) pode definir métodos específicos para si (comprar para Cliente e estudar para Aluno), mantendo suas funcionalidades exclusivas.

### Exemplo de Uso

Imagine que você cria instâncias das classes Cliente e Aluno:



Neste exemplo, vemos como a herança facilita o uso dos métodos falar em Cliente e Aluno, além de permitir que cada classe tenha métodos específicos para suas responsabilidades.

Se atentar ao metodo super().\_\_init\_\_() esse metodo cama o init da classe mãe. Logo os parametros requeridos no init da mae devem ser passados na construcao da classe filha.

