



INFINITY SCHOOL

VISUAL ART CREATIVE CENTER

AULA 09 – POO II HERANÇA

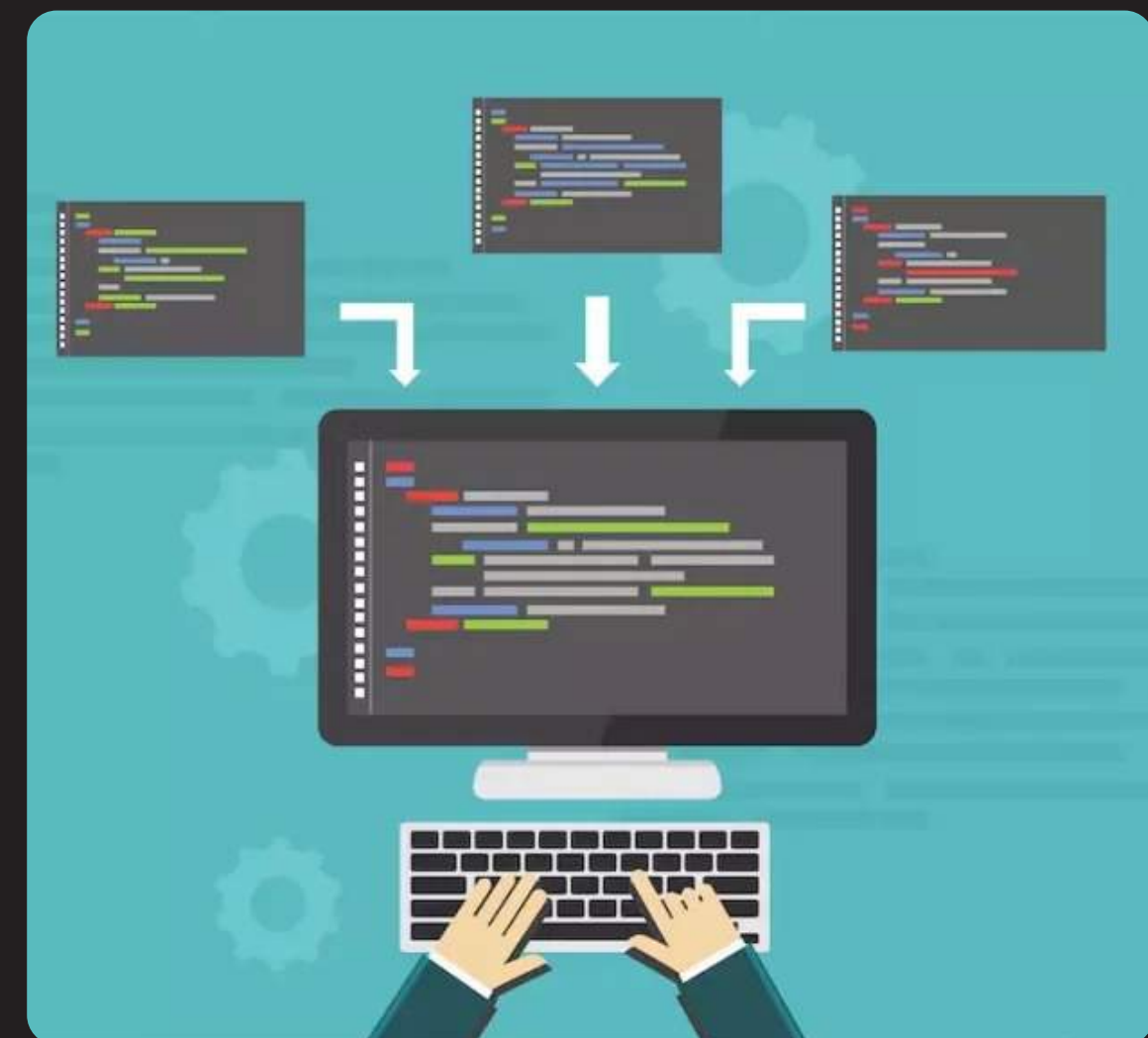
O QUE IREMOS APRENDER

- 01** RESUMO DA AULA PASSADA
- 02** CONTEXTUALIZAÇÃO DA AULA DE HOJE
- 03** HERANÇA
- 04** PRINCIPAIS CLASSES
- 05** TIPOS DE HERANÇAS
- 06** POLIMORFISMO
- 07** POLIMORFISMO E SUAS CLASSIFICAÇÕES
- 08** MÃOS NO CÓDIGO
- 09** PROJETO

Resumo da aula passada

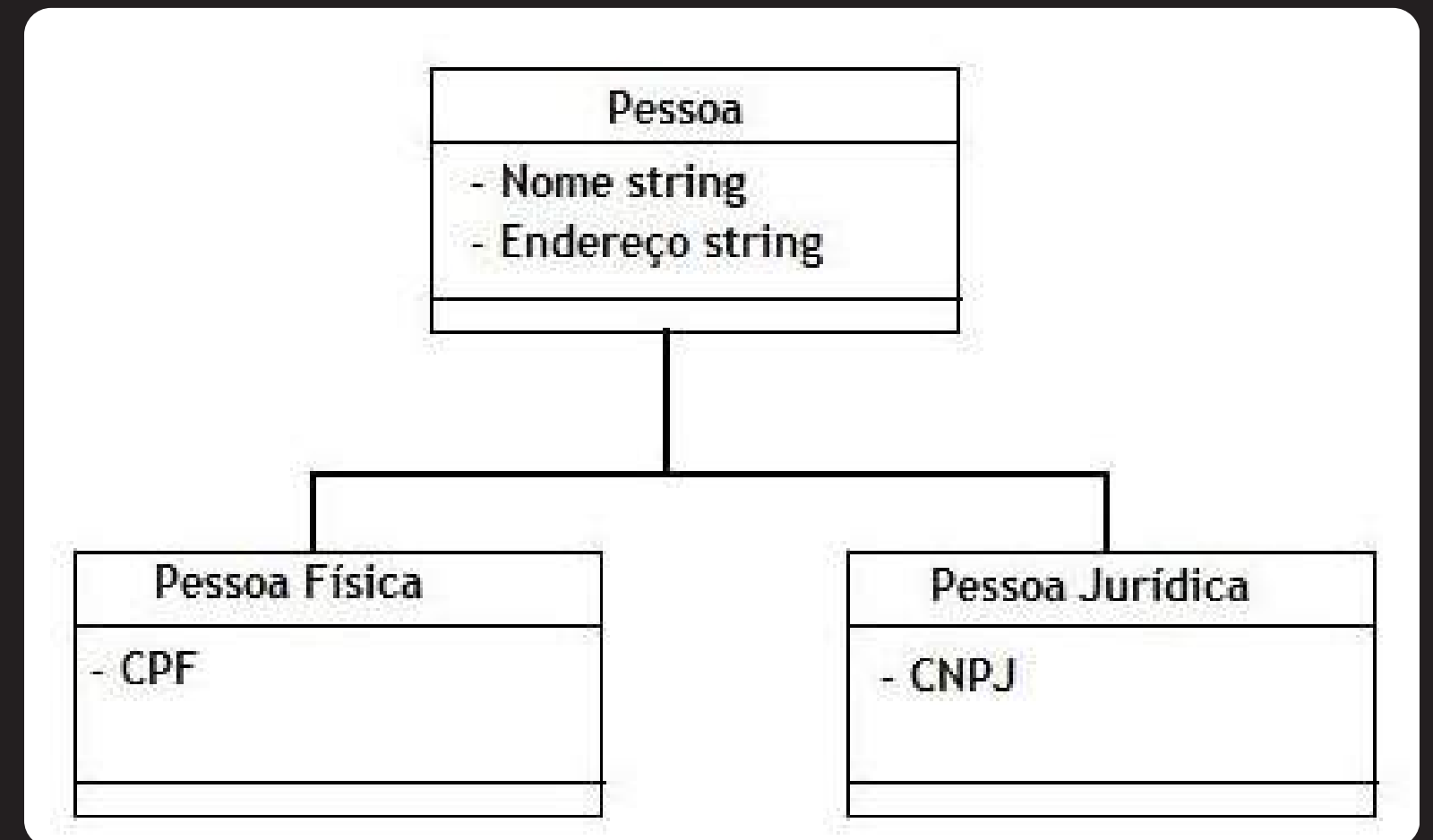
Programação Orientada a Objetos (POO) é um paradigma de programação que se baseia no conceito de "objetos" como a unidade fundamental de estruturação do código. Nesse paradigma, os objetos representam entidades do mundo real e possuem características (atributos) e ações (métodos) associadas a eles.

Abstração, por outro lado, é um conceito relacionado à POO que envolve a identificação das características e comportamentos essenciais de um objeto, enquanto ignora detalhes irrelevantes.



Herança

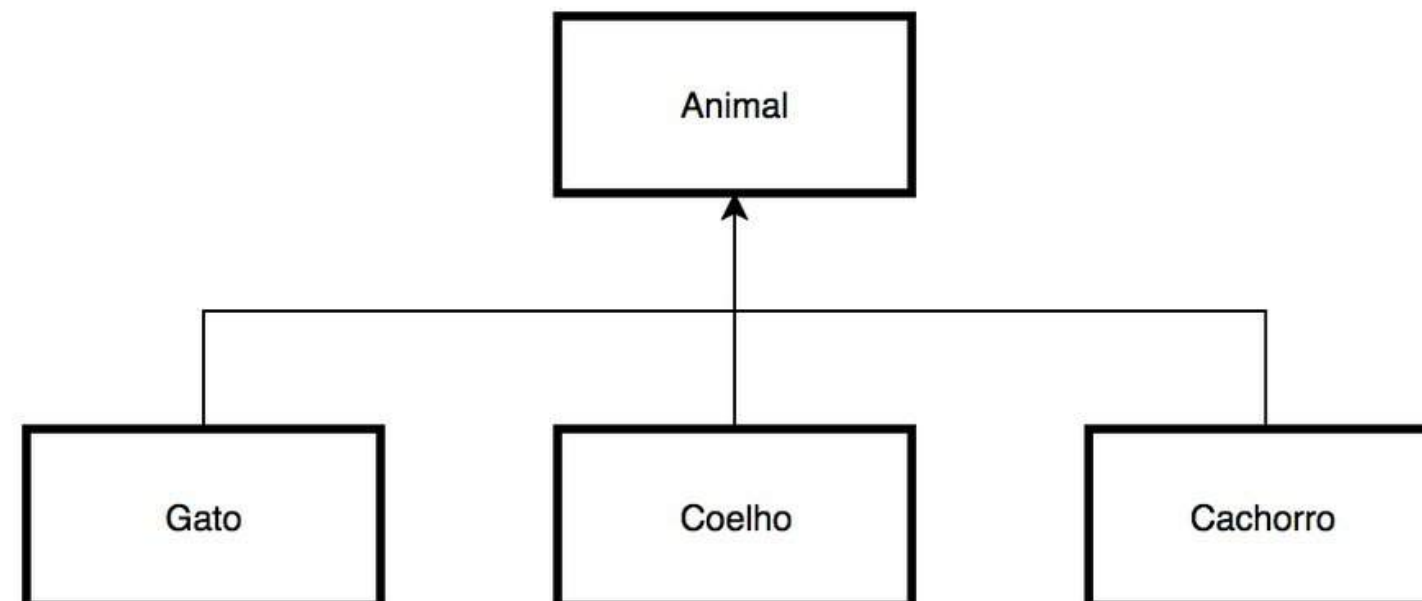
A **herança** é um princípio próprio à programação orientada a objetos (POO) que permite criar uma nova classe a partir de uma já existente. Herança, também chamada de subclasses, provém da classe recém-criada que contém atributos e métodos da qual deriva. A principal vantagem da herança é a capacidade para definir novos atributos e métodos para a subclasse, que se somam aos atributos e métodos herdados.



Principais Classes

Na Herança temos dois tipos principais de classe:

Classe Base: A classe que **concede** as características a uma outra classe. Classe Derivada: A classe que **herda** as características da classe base.



Tipos de Heranças

Assim, ela pode se dividir em 2 tipos:

Herança simples: classe herda atributos e métodos de apenas uma classe.

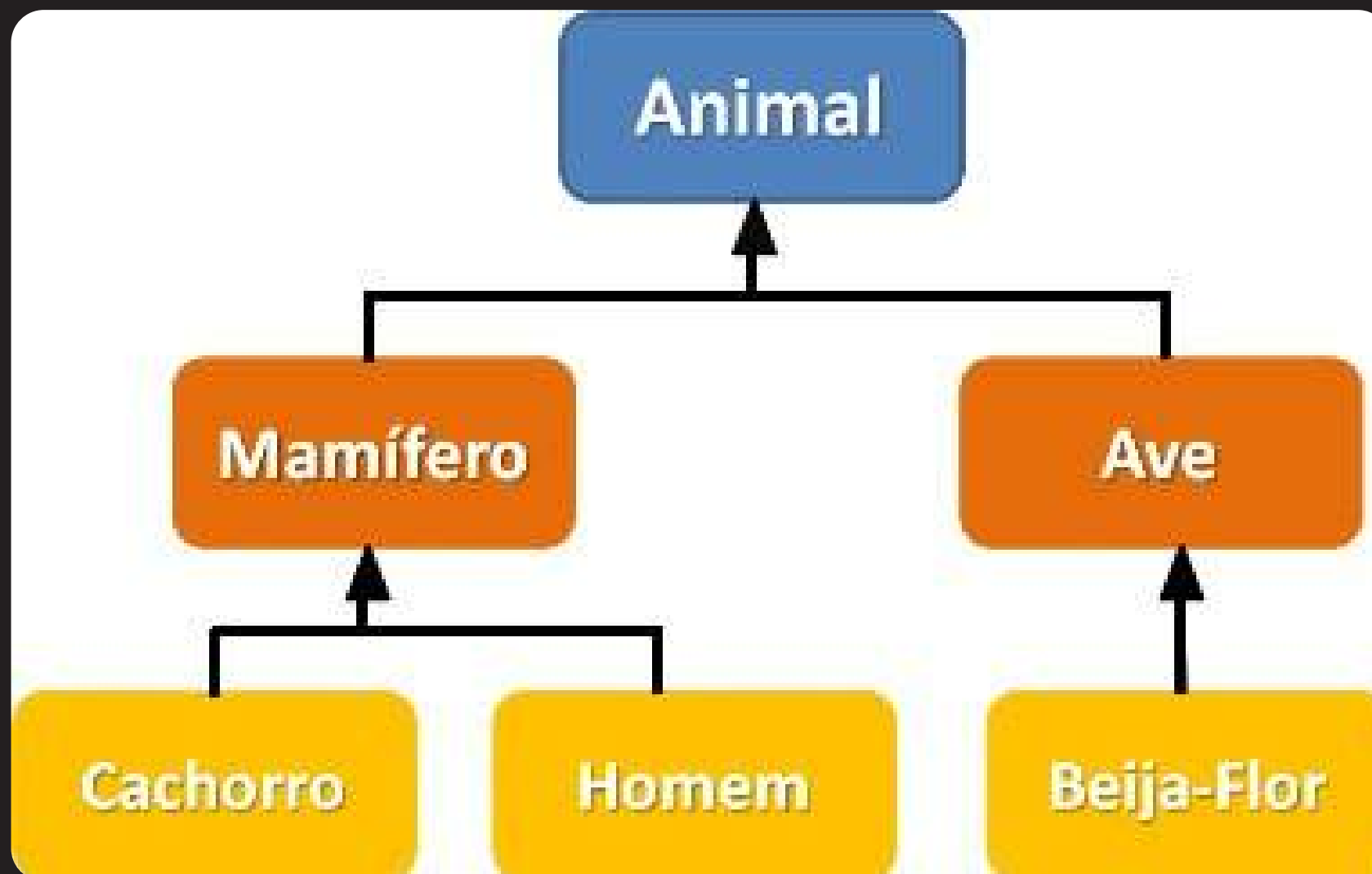
Herança múltipla: classe herda atributos e métodos de duas ou mais classes.

```
if path:
    self.file = open(os-
    self.file.seek(0)
    self.fingerprints.up

@classmethod
def from_settings(cls, sett
    debug = settings.getbo
    return cls(job_dir(set

def request_seen(self, req
    fn = self.request_fing
```


Herança - Você sabia?



Nem todas as linguagens que trabalham com POO suportam a herança múltipla. Um exemplo frequente em provas é a linguagem Java, que apenas lida com herança simples.

Exemplo - Herança

Neste exemplo, temos uma classe base chamada `Animal` com um construtor que recebe um nome e um método `fazerSom` que é declarado, mas não implementado (usamos `pass` para indicar isso). Em seguida, criamos duas classes derivadas, `Cachorro` e `Gato`, que herdam de `Animal`, o método `fazerSom` para fornecer um comportamento específico para seus respectivos animais.

```
1 class Animal:
2     def __init__(self, nome):
3         self.nome = nome
4     def fazerSom(self):
5         pass
6
7 class Cachorro(Animal):
8     def fazerSom(self):
9         return "Woof!"
10
11 class Gato(Animal):
12     def fazerSom(self):
13         return "Meow!"
14
15 rex = Cachorro("Rex")
16 whiskers = Gato("whiskers")
17
18 print(rex.nome, "faz", rex.fazerSom())
19 print(whiskers.nome, "faz", whiskers.fazerSom())
```


Exemplo - Herança

Ao criar objetos Cachorro e Gato e chamar o método fazerSom, obtemos o comportamento apropriado para cada tipo de animal. Isso demonstra como a herança permite compartilhar e especializar funcionalidades em classes relacionadas.

```
1 class Animal:
2     def __init__(self, nome):
3         self.nome = nome
4     def fazerSom(self):
5         pass
6
7 class Cachorro(Animal):
8     def fazerSom(self):
9         return "Woof!"
10
11 class Gato(Animal):
12     def fazerSom(self):
13         return "Meow!"
14
15 rex = Cachorro("Rex")
16 whiskers = Gato("whiskers")
17
18 print(rex.nome, "faz", rex.fazerSom())
19 print(whiskers.nome, "faz", whiskers.fazerSom())
```

Polimorfismo

O **polimorfismo** é um conceito fundamental na programação orientada a objetos que se refere à capacidade de objetos de classes diferentes responderem a chamadas de métodos com o mesmo nome de maneira uniforme. Isso permite tratar objetos de maneira genérica, independentemente de sua classe específica.



Polimorfismo e suas Classificações

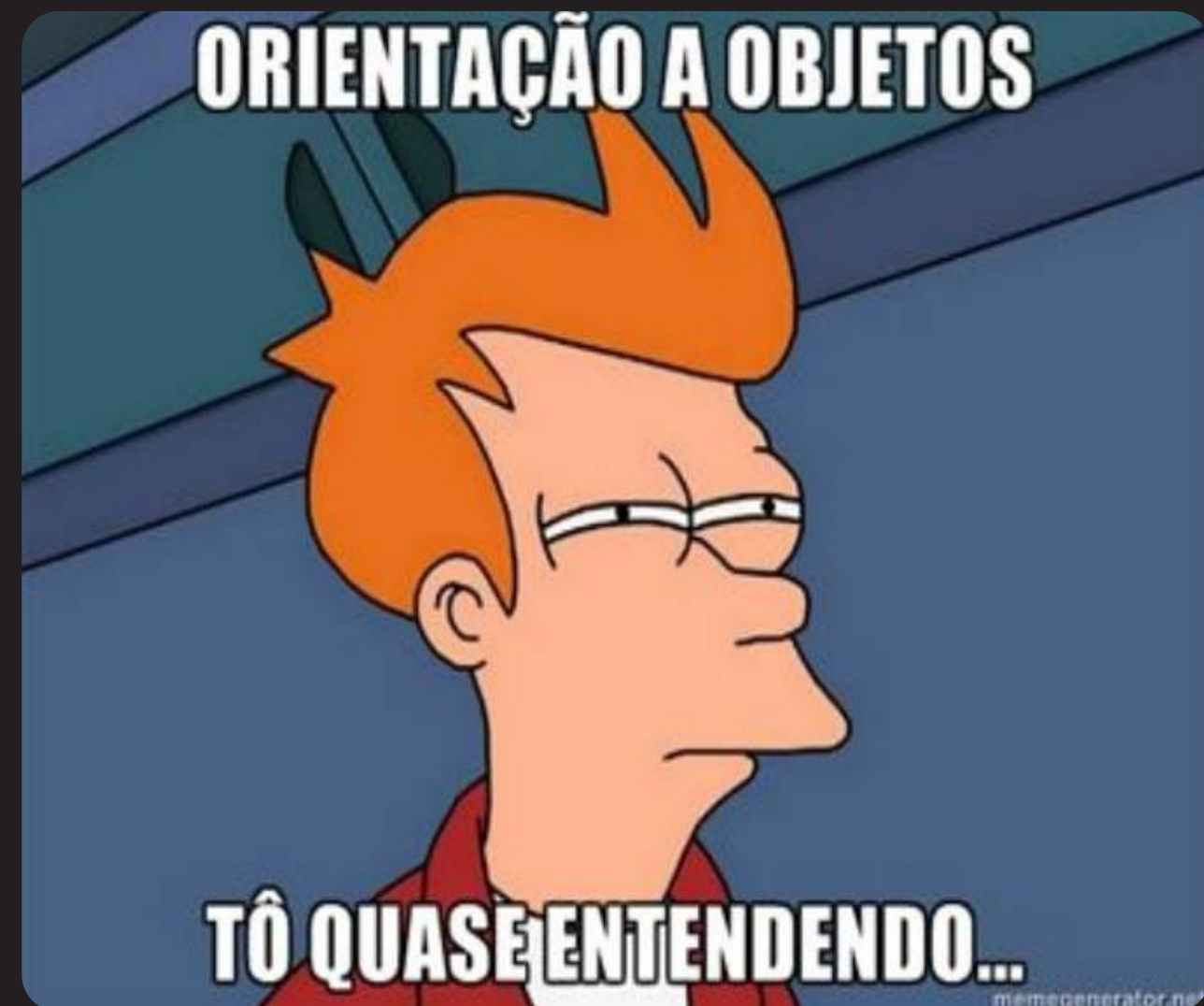
Ele pode apresentar duas classificações:

Polimorfismo estático: ocorre em momento de compilação. O mesmo método é implementado várias vezes na mesma classe, com parâmetros diferentes. A escolha do método a ser chamado vai variar de acordo com o parâmetro passado.

Polimorfismo dinâmico: ocorre em momento de execução. O mesmo método é implementado várias vezes nas subclasses derivadas, com os mesmos parâmetros. A escolha do método depende do objeto que o chama (e, conseqüentemente, da classe que o implementa).

Polimorfismo - Você sabia?

O polimorfismo estático também pode ser chamado de sobrecarga, enquanto o polimorfismo dinâmico também pode ser chamado de sobreposição.



Classe POO

```
1 class Animal:
2     def fazerSom(self):
3         pass
4
5 class Cachorro(Animal):
6     def fazerSom(self):
7         return "Woof!"
8
9 class Gato(Animal):
10    def fazerSom(self):
11        return "Meow!"
12
13 class Pato(Animal):
14    def fazerSom(self):
15        return "Quack!"
16
17 # Função que usa polimorfismo
18 def fazerAnimalFalar(animais):
19     return animais.fazerSom()
20
21 # Criando os objetos
22 rex = Cachorro()
23 whiskers = Gato()
24 donald = Pato()
25
26 # Chamando a função usando polimorfismo
27 animais = [rex, whiskers, donald]
28
29
30 for animal in animais:
31     print(animal.__class__.__name__, "faz", fazerAnimalFalar(animal))
```

A função `fazerAnimalFalar` demonstra o polimorfismo, já que a função pode tratar objetos de diferentes classes derivadas da mesma forma. No final, colocamos esses objetos em uma lista e usamos um loop para chamar a função `fazerAnimalFalar` para cada um.

ATIVIDADE PRÁTICA 1

Crie uma hierarquia de classes representando formas geométricas. Comece com uma classe base chamada "Forma" e, em seguida, crie classes derivadas como "Círculo" e "Retângulo" que herdem da classe base. Adicione métodos para calcular área e perímetro em cada classe derivada.

ATIVIDADE PRÁTICA 2

Crie uma hierarquia de classes que represente veículos. Comece com uma classe base "Veículo" e, em seguida, crie classes derivadas como "Carro" e "Bicicleta." Adicione métodos para definir atributos, como "cor" e "modelo," e permita a chamada de métodos em cadeia para configurar esses atributos.

ATIVIDADE PRÁTICA 3

Crie uma classe chamada "Calculadora" com um método "somar" que pode somar dois números inteiros ou duas strings. Use o polimorfismo para implementar a sobrecarga do método "somar" para que ele funcione com diferentes tipos de entrada (números inteiros e strings). Crie exemplos de uso para demonstrar como a mesma função pode se comportar de maneira diferente com base nos tipos de entrada.

ATIVIDADE PRÁTICA 4

Crie uma interface chamada "Veículo" com métodos "acelerar" e "frear." Em seguida, crie classes concretas como "Carro" e "Bicicleta" que implementem a interface "Veículo" e forneçam suas próprias implementações dos métodos "acelerar" e "frear." Demonstre como o polimorfismo pode ser usado para tratar diferentes tipos de veículos de maneira uniforme, chamando os métodos da interface.

ATIVIDADE PRÁTICA 5

Crie uma classe base chamada "Animal" com um método "emitirSom." Em seguida, crie classes derivadas como "Cachorro," "Gato" e "Pássaro" que herdem de "Animal" e sobrescrevam o método "emitirSom" para cada tipo de animal. Crie uma lista de animais e percorra-a, chamando o método "emitirSom" para cada animal. Demonstre como o polimorfismo permite que diferentes tipos de animais emitam seus sons de maneira uniforme.

DESAFIO PRÁTICO

sistema de gerenciamento de contas bancárias em Python

Crie um sistema de gerenciamento de contas bancárias em Python usando herança e polimorfismo. O sistema deve incluir as seguintes classes:

DESAFIO PRÁTICO

Classe Conta:

- A classe base "Conta" deve ter atributos para o número da conta, o titular da conta e o saldo.
- Ela deve incluir métodos para depósitos, saques e exibição do saldo atual.

DESAFIO PRÁTICO

Classe ContaCorrente:

- A classe "ContaCorrente" herda de "Conta" e inclui atributos específicos, como taxa de manutenção e limite de cheque especial.
- Deve sobrescrever o método de saque para considerar o limite de cheque especial, se necessário.

DESAFIO PRÁTICO

Classe ContaPoupanca:

- A classe "ContaPoupanca" também herda de "Conta" e inclui atributos específicos, como taxa de juros.
- Ela deve ter um método para calcular e adicionar juros ao saldo.
- Crie um método chamado resumo que pode ser chamado em qualquer objeto de conta (ContaCorrente ou ContaPoupanca).

DESAFIO PRÁTICO

Esse método resumo irá exibir um resumo das informações da conta, incluindo o tipo de conta (corrente ou poupança), o número da conta, o titular da conta e o saldo atual.

Teste de Funcionalidade:

Crie um programa principal que demonstre o uso dessas classes. Crie instâncias de contas correntes e poupanças, realize depósitos, saques, adicione juros e chame o método resumo para cada conta.

Material Complementar

- Exploração: Não tenha medo de explorar e testar diferentes códigos. A experimentação é uma grande aliada da aprendizagem.
 - Perguntas: Faça perguntas, seja curioso! Entender o "porquê" das coisas ajuda a consolidar o conhecimento.
 - Revisão: Revise o que aprendeu, tente explicar para si mesmo ou para outras pessoas. Ensinar é uma ótima forma de aprender.
- Prática: A prática leva à perfeição. Quanto mais
- exercícios fizer, mais fácil será lembrar e entender os conceitos.



PROJETO

Criar uma **aplicação de gerenciamento de biblioteca** com classes e POO para representar livros, membros e a biblioteca.

Passos do projeto:

Criação de Classes:

Crie as seguintes classes: Livro, Membro, Biblioteca.

Classe Livro:

A classe Livro deve conter atributos como título, autor, ID, status de empréstimo (disponível ou emprestado).

PROJETO

Classe Membro:

- A classe Membro deve conter atributos como nome, número de membro e histórico de livros emprestados.

Classe Biblioteca:

- A classe Biblioteca deve conter um catálogo de livros disponíveis, um registro de membros e métodos para empréstimo, devolução e pesquisa de livros.

PROJETO

Operações da Biblioteca:

Implemente **métodos** na **classe Biblioteca** para:

• Adicionar livros ao catálogo.

• Adicionar membros à biblioteca.

• Permitir empréstimo de livros por membros.

• Registrar a devolução de livros.

- Pesquisar livros por título, autor ou ID.

Material Complementar

- Exploração: Não tenha medo de explorar e testar diferentes códigos. A experimentação é uma grande aliada da aprendizagem.
 - Perguntas: Faça perguntas, seja curioso! Entender o "porquê" das coisas ajuda a consolidar o conhecimento.
 - Revisão: Revise o que aprendeu, tente explicar para si mesmo ou para outras pessoas. Ensinar é uma ótima forma de aprender.
- Prática: A prática leva à perfeição. Quanto mais
- exercícios fizer, mais fácil será lembrar e entender os conceitos.



SE LIGA NO CONTEÚDO DA PRÓXIMA AULA!

AULA 10 DE PYTHON:
SQL I



INFINITY SCHOOL
VISUAL ART CREATIVE CENTER

SQL

Usar um banco de dados é essencial por várias razões em ambientes onde é necessário **armazenar** e **gerenciar** dados de forma eficiente. Os bancos de dados desempenham um papel crítico em organizações e aplicativos modernos, pois fornecem uma estrutura robusta para armazenar, acessar e proteger os dados.

Eles facilitam a tomada de decisões informadas, o compartilhamento de informações, a segurança dos dados e a eficiência operacional.

```

    ... }
    ...event('on' + type, callback);

    action decorate(event) {
    event = event || window.event;
    var target = event.target || event.srcElement;
    if (target && (target.getAttribute('action') || target.
        ga(function (tracker) {
            var linkerParam = tracker.get('linkerParam');
            document.cookie = '_shopify_ga=' + linkerParam + ';
        }));

    stener(window, 'load', function(){
        i=0; i < document.forms.length; i++) {
            document.forms[i].getAttribute('action');
            if ('/cart') >= 0) {
                submit', decorate);
            }
        }
    }
}

```



INFINITY SCHOOL

VISUAL ART CREATIVE CENTER

AULA 09 – POO II HERANÇA