INFINITY SCHOOL

VISUAL ART CREATIVE CENTER

AULA 12 – 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

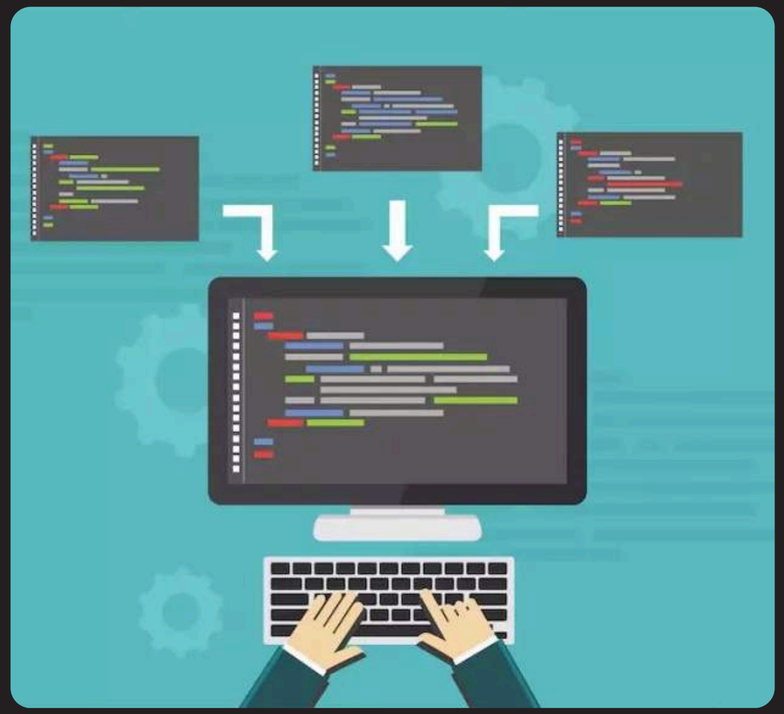
07 POLIMORFISMO E SUAS CLASSIFICAÇÕES

08 MÃOS NO CÓDIGO

Resumo da aula passada

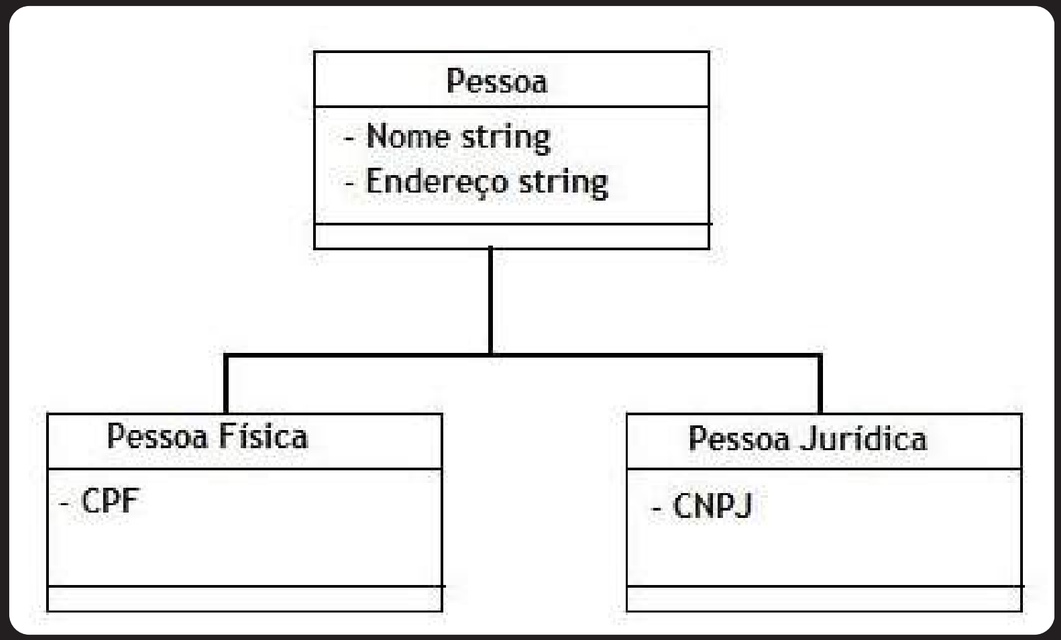
Programação Orientada a Objetos (POO) é um paradigma de programação que se baseia no conceito de "objetos" como a unidade fundamental de estruturação do código. Nesse paradigma, os objetos representam entidades do mundo real e possuem características (atributos) e ações (métodos) associadas a eles.

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

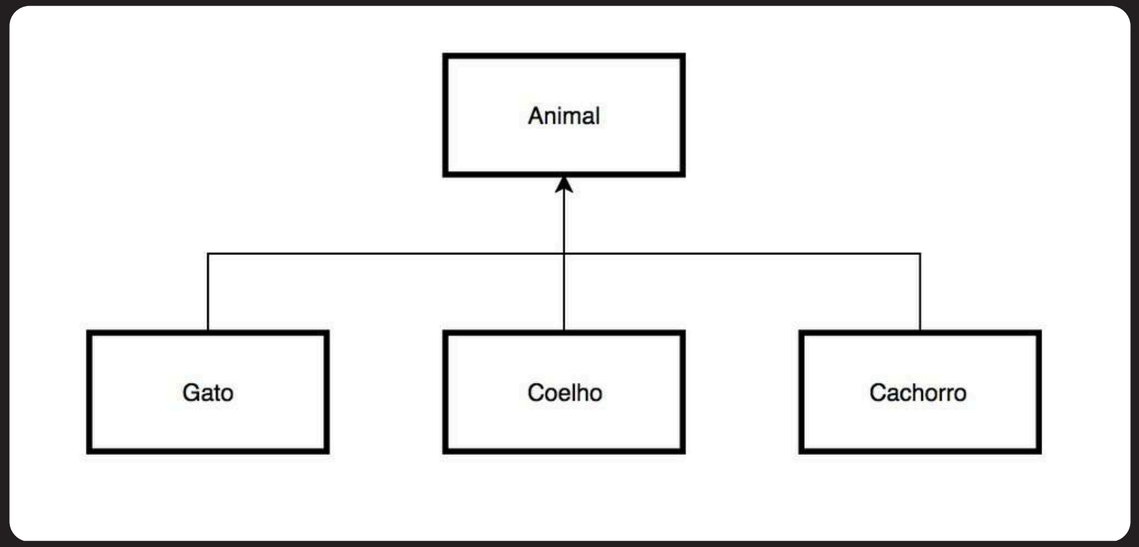
A herança é um princípio próprio à programação orientada a objetos (POO) que permite criar uma nova classe a partir de uma já existente. Herança, também chamada de subclasses, provém da classe recém-criada que contém atributos e métodos da qual deriva. A principal vantagem da herança é a capacidade para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



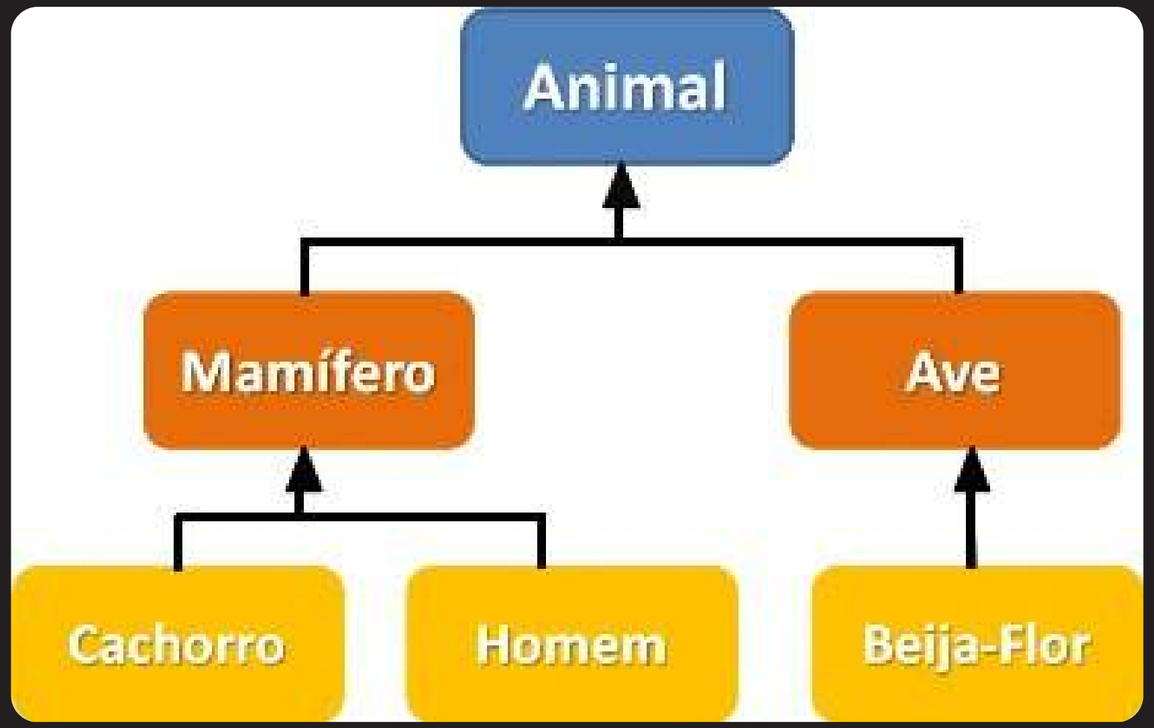
Tipos de Heranças

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.

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.

```python
class Animal:
    def __init__(self, nome):
        self.nome = nome
    def fazerSom(self):
        pass

class Cachorro(Animal):
    def fazerSom(self):
        return "Woof!"

class Gato(Animal):
    def fazerSom(self):
        return "Meow!"

rex = Cachorro("Rex")
whiskers = Gato("whiskers")

print(rex.nome, "faz", rex.fazerSom())
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.

```python
class Animal:
    def __init__(self, nome):
        self.nome = nome
    def fazerSom(self):
        pass

class Cachorro(Animal):
    def fazerSom(self):
        return "Woof!"

class Gato(Animal):
    def fazerSom(self):
        return "Meow!"

rex = Cachorro("Rex")
whiskers = Gato("whiskers")

print(rex.nome, "faz", rex.fazerSom())
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

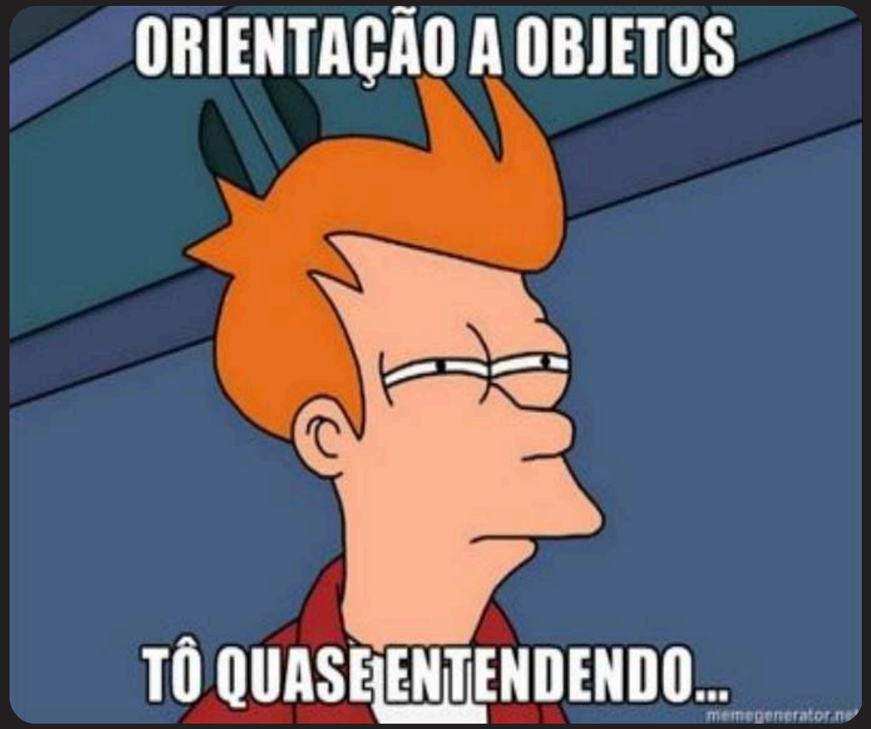
Ele pode apresentar duas classificações:

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, consequentemente, 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

```python
class Animal:
    def fazerSom(self):
        pass

class Cachorro(Animal):
    def fazerSom(self):
        return "Woof!"

class Gato(Animal):
    def fazerSom(self):
        return "Meow!"

class Pato(Animal):
    def fazerSom(self):
        return "Quack!"

# Função que usa polimorfismo
def fazerAnimalFalar(animal):
    return animal.fazerSom()

# Criando os objetos
rex = Cachorro()
whiskers = Gato()
donald = Pato()

# Chamando a função usando polimorfismo
animais = [rex, whiskers, donald]


for animal in animais:
    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.

PEDINDO AJUDA AO CHATGPT

You
PODE ME EXPLICAR POO(HERANÇA, PRINCIPAIS CLASSES, TIPOS DE HERANÇAS , POLIMORFISMO E SUAS CLASSIFICAÇÕES) EM PYTHON DE FORMA RESUMIDA?

ChatGPT
Claro! Aqui está uma explicação resumida sobre Programação Orientada a Objetos (POO) em Python, abordando os conceitos de herança, principais classes, tipos de herança, polimorfismo e suas classificações:

1. **Herança:**
   - A herança é um conceito em POO que permite que uma classe (subclasse) herde atributos e métodos de outra classe (superclasse).
   - Em Python, a herança é implementada colocando o nome da classe pai entre parênteses na definição da classe filha.
2. **Principais classes:**
   - As principais classes em POO incluem:
     - **Classe base (superclasse):** Uma classe que é usada como modelo para outras classes. Pode ter atributos e métodos que são compartilhados por suas subclasses.
     - **Classe derivada (subclasse):** Uma classe que herda atributos e métodos de outra classe. Pode adicionar novos atributos e métodos ou modificar os existentes.
3. **Tipos de herança:**
   - Existem diferentes tipos de herança em Python:
     - **Herança simples:** Uma subclasse herda de apenas uma superclasse.
     - **Herança múltipla:** Uma subclasse herda de várias superclasses.
     - **Herança multinível:** Uma subclasse herda de uma classe que por sua vez herda de outra classe.
     - **Herança híbrida:** Combinação de herança múltipla e multinível.
4. **Polimorfismo:**
   - Polimorfismo é a capacidade de objetos de diferentes classes responderem ao mesmo método de maneira diferente.
   - Em Python, o polimorfismo é frequentemente alcançado por meio do uso de métodos com o mesmo nome em classes diferentes, mas que têm comportamentos específicos para cada classe.
5. **Classificações do polimorfismo:**
   - Existem duas classificações principais de polimorfismo:
     - **Polimorfismo de sobrecarga:** Métodos com o mesmo nome, mas com assinaturas diferentes (número ou tipo de parâmetros).
     - **Polimorfismo de sobreposição:** Métodos com o mesmo nome e assinatura, mas que têm implementações diferentes em classes diferentes.

Em resumo, POO em Python oferece suporte a herança, polimorfismo e outros conceitos que permitem criar classes e objetos flexíveis e reutilizáveis. Esses conceitos são fundamentais para a modelagem de sistemas complexos e o desenvolvimento de programas orientados a objetos em Python.

PEDINDO AJUDA AO CHATGPT

You
me indique 4 exercícios sobre esse assunto

ChatGPT
Claro! Aqui estão quatro exercícios para praticar herança, polimorfismo e outros conceitos de Programação Orientada a Objetos (POO) em Python:

1. **Herança simples:**
   Crie uma classe `Animal` com métodos como `comer()` e `dormir()`. Em seguida, crie duas subclasses, como `Cachorro` e `Gato`, que herdam da classe `Animal`. Adicione métodos específicos para cada subclasse, como `latir()` para `Cachorro` e `miar()` para `Gato`.
2. **Herança múltipla:**
   Crie uma classe `Ave` com métodos como `voar()` e `chocar_ovos()`, e uma classe `Mamifero` com métodos como `amamentar()` e `respirar()`. Em seguida, crie uma subclasse chamada `Morcego` que herda de ambas as classes `Ave` e `Mamifero`. Adicione métodos específicos para `Morcego`, como `sonar()`.
3. **Polimorfismo de sobrecarga:**
   Crie uma classe chamada `Calculadora` com um método `calcular()`. Em seguida, crie subclasses como `Soma`, `Subtracao`, `Multiplicacao` e `Divisao`, cada uma sobrecarregando o método `calcular()` para realizar sua operação correspondente. Teste as subclasses para verificar se elas executam as operações corretamente.
4. **Polimorfismo de sobreposição:**
   Crie uma classe `Forma` com um método `area()`. Em seguida, crie subclasses como `Retangulo`, `Triangulo` e `Circulo`, cada uma sobrepondo o método `area()` para calcular a área correspondente à forma geométrica específica. Teste as subclasses para verificar se elas calculam as áreas corretamente.

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.

SE LIGA NO CONTEÚDO DA PRÓXIMA AULA!

AULA 13 DE PYTHON:
POO III - ASSOCIAÇÃO E ENCAPSULAMENTO

IN

INFINITY SCHOOL
VISUAL ART CREATIVE CENTER

Aula 13 - POO III - ASSOCIAÇÃO E ENCAPSULAMENTO

ASSOCIAÇÃO

TIPOS DE ASSOCIAÇÃO (UNIDIRECIONAL E BIDIRECIONAL)

ENCAPSULAMENTO

IMPORTÂNCIA DO ENCAPSULAMENTO

GETTERS E SETTERS

PROPERTY

INFINITY SCHOOL

VISUAL ART CREATIVE CENTER

ATÉ A PRÓXIMA...