Herança e Polimorfismo

CIÊNCIA DA COMPUTAÇÃO

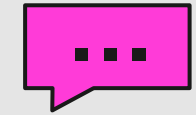
ATITUS EDUCAÇÃO

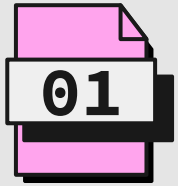
- Aula 06 -
Organização e Abstração

Prof. Me. Lucas R. C. Pessutto

ATITUS EDUCAÇÃO

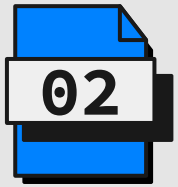
Na aula de hoje...



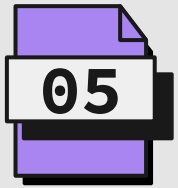


Herança

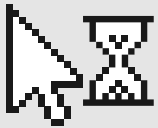
Sobrescrita de Métodos



Polimorfismo

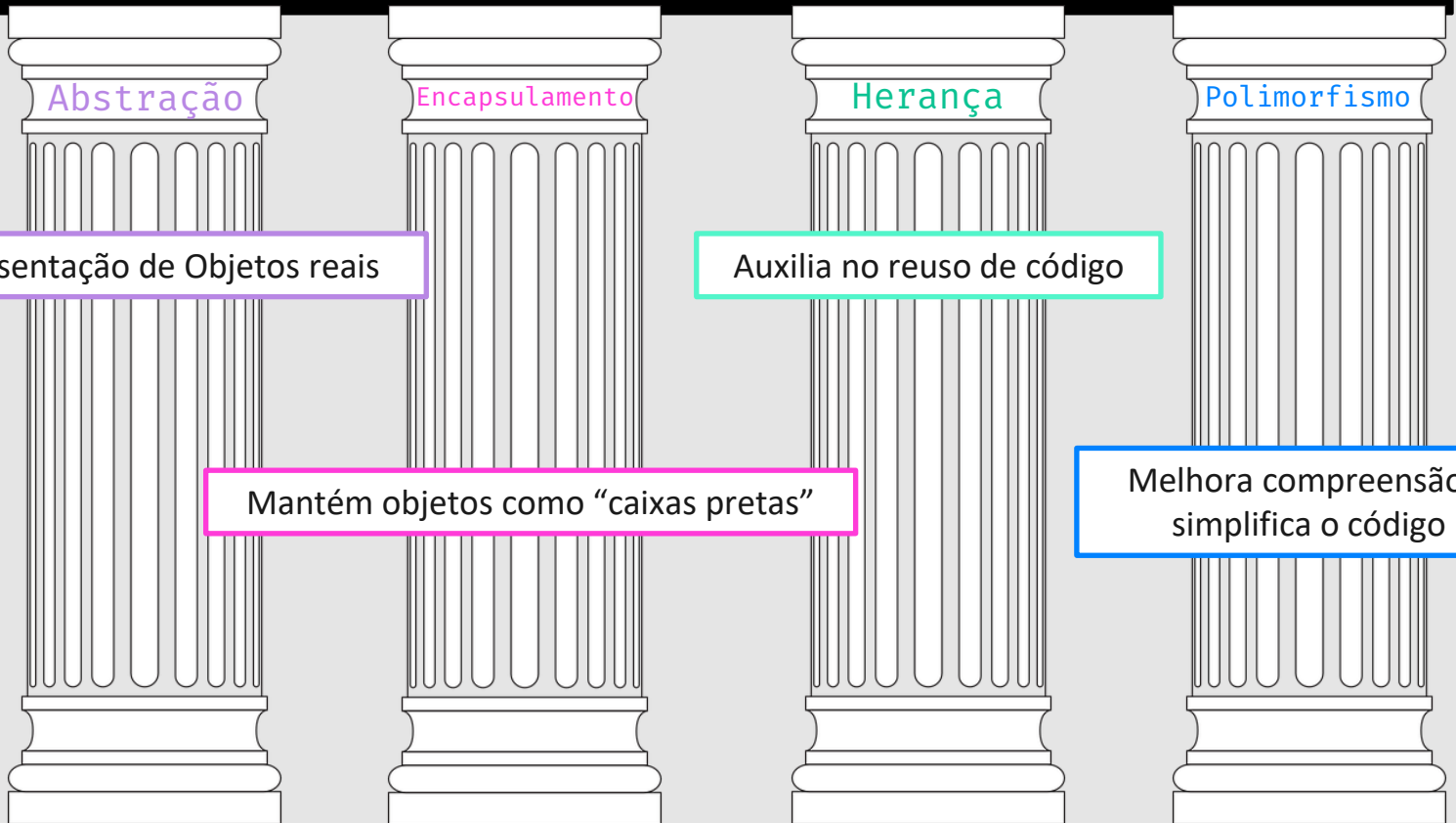


Herança Múltipla



BOOCH, 1994

Pilares Orientação a Objetos



Representação de Objetos reais

Mantém objetos como “caixas pretas”

Auxilia no reuso de código

Melhora compreensão e simplifica o código

Problema: A rede social

* Imagine uma rede social que permita aos seus usuários realizar postagens

* Nesse tipo de rede social é comum haver um feed de notícias, que exibe as postagens realizadas na plataforma

* Vamos construir um protótipo desse mecanismo de exibição das postagens na rede social

Problema: A rede social

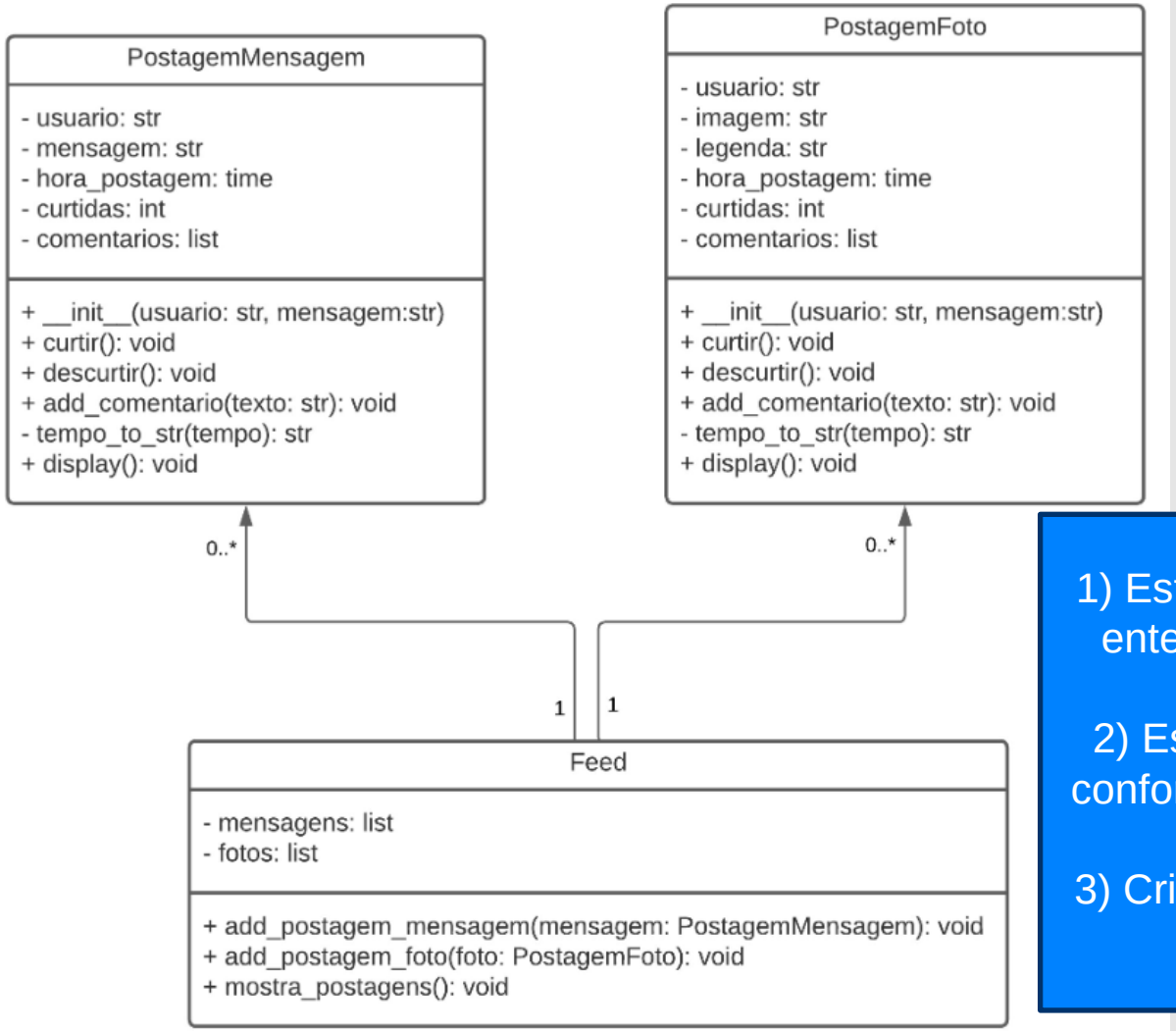
* Existem dois tipos de postagens nessa rede social:

Posts textuais: uma mensagem contendo texto, possivelmente ocupando múltiplas linhas.

* As seguintes informações devem ser armazenadas para esse tipo de postagem:
  - o username do autor
  - a mensagem em si
  - um time stamp (horário da postagem)
  - quantas pessoas curtiram
  - uma lista de comentários

Posts com Imagem: consistem em uma imagem e sua legenda

* As seguintes informações devem ser armazenadas para esse tipo de postagem:
  - o username do autor
  - o nome do arquivo da imagem
  - legenda da foto
  - um time stamp (horário da postagem)
  - quantas pessoas curtiram
  - uma lista de comentários



1) Estude o código fornecido para entender o seu funcionamento.

2) Escreva o programa principal conforme as instruções fornecidas

3) Crie um novo tipo de postagem para essa rede social

Problemas na solução apresentada

* Duplicação de código: O código das classes PostagemMensagem e PostagemFoto são praticamente idênticos

* Na classe Feed tudo precisa ser feito duas vezes, uma vez para PostagemMensagem e outra para PostagemFoto

* O que teve que ser feito para adicionar um novo tipo de postagem?
  - → Criar uma nova classe, copiar (e adaptar) o código de PostagemMensagem
  - → Adicionar uma nova coleção de objetos em Feed
  - → Criar laços de repetição para o novo tipo de postagem em Feed

* Duplicação de código dificulta a manutenção do código! Quanto mais código duplicado, mais difícil será realizar mudanças no código

BOOCH, 1994

Herança

Herança

* A herança é uma forma de diminuir a duplicação no código

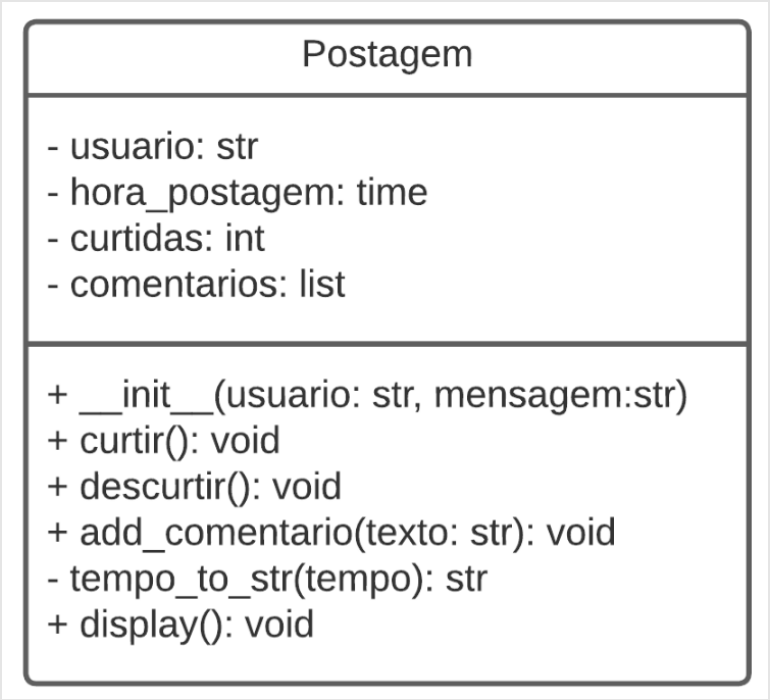
* Seu funcionamento é simples: define-se uma classe que possui os atributos e métodos que as classes com código duplicado possuem em comum. Em nosso exemplo, chamaremos essa classe de *Postagem*

* Em seguida, diremos que PostagemMensagem e PostagemFoto **são** uma Postagem e podemos remover todo o código repetido destas classes, mantendo somente os atributos e métodos exclusivos de cada postagem particular

* Toda classe na linguagem Python possui uma superclasse. Se a herança não for especificada, a classe pai será a classe `object`.

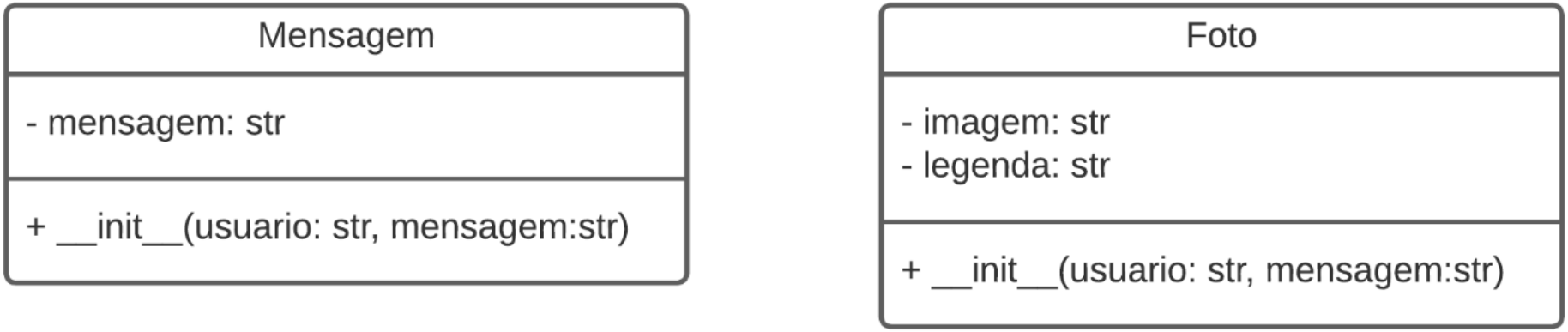
Na prática: Rede Social 2.0

* Passo 1: Criação da Classe Postagem



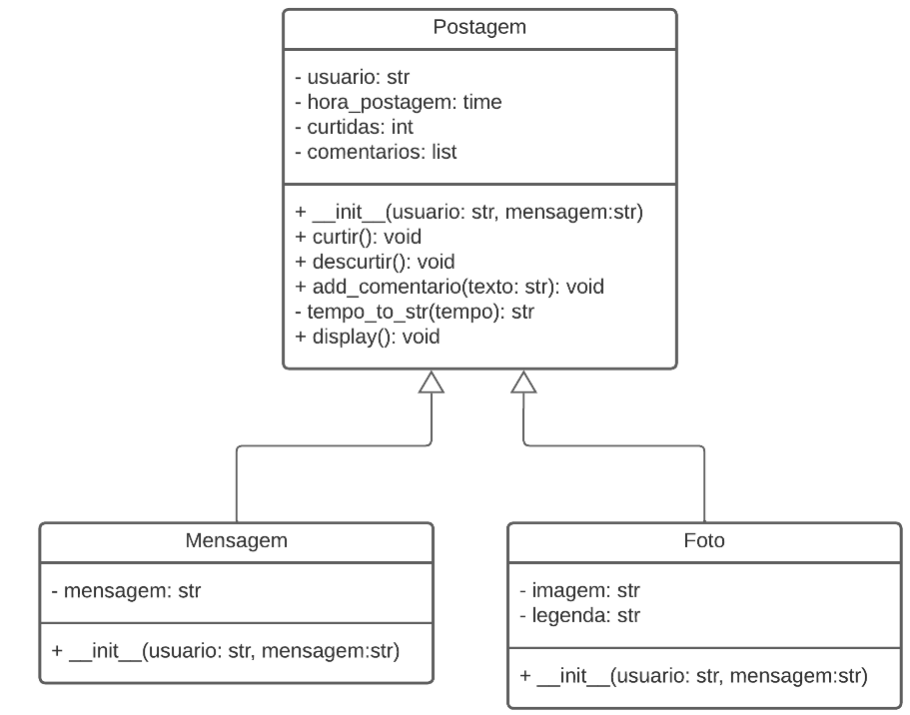
Na prática: Rede Social 2.0

* Passo 2: Remover os atributos duplicados das classes PostagemMensagem e PostagemFoto



Na prática: Rede Social 2.0

* Passo 3: Dizer que Mensagem e Foto herdam as características de Postagem



Herança - Jargão

* Dizemos que "*Mensagem herda as características de Postagem*". Também pode-se dizer que a classe "*Foto estende Postagem*"

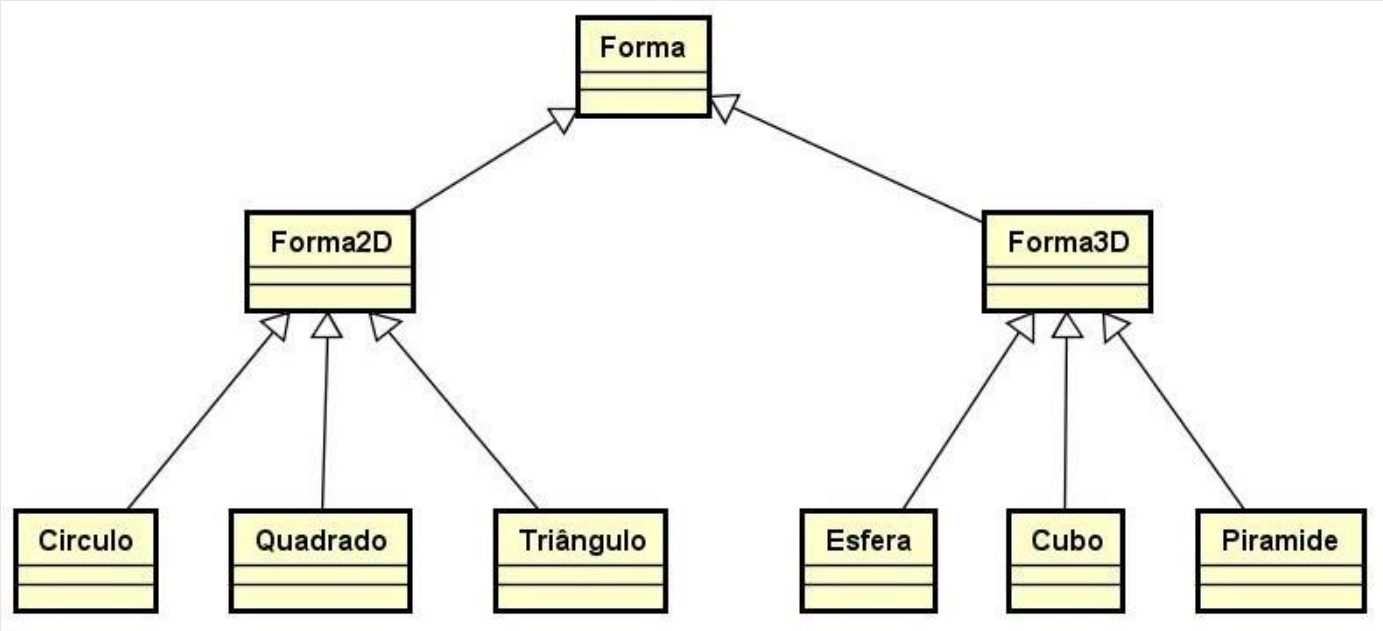
Postagem => Classe pai ou superclasse
Mensagem e Foto => Classes filhas ou subclasses

* A herança define um relacionamento "**é um**"
  - → Mensagem é uma Postagem
  - → Foto é uma Postagem

* Um objeto de uma classe filha pode ser tratado como um objeto da classe pai

Herança: Hierarquia de Classes

* Podemos estabelecer uma hierarquia de classes: onde existem vários níveis de especialização.



Herança: Implementação

* Classe Postagem (Pai da Hierarquia): implementação “normal”

```
class Postagem:
    def __init__(self, usuario):
        self._usuario = usuario
        self._hora_postagem = time.time()
        self._curtidas = 0
        self._comentarios = []

    # Getters e Setters

    def curtir(self):
        # Implementação do método

    def descurtir(self):
        # Implementação do método
```

```
def add_comentario(self, texto):
    # Implementação do método

def _tempo_to_str(self, tempo):
    # Implementação do método

def display(self):
    # Implementação do método
```

Herança: Implementação

* Classe Mensagem:

```
class Mensagem (Postagem):
    # Construtor e Métodos omitidos por enquanto
```

* Acrescentamos o nome da classe pai, junto da declaração da classe filha, entre parênteses para indicar uma herança

* Ela indica que a classe Mensagem estende o comportamento da classe Postagem

* Note que Mensagem só possuirá os atributos que são únicos dessa classe. Os demais atributos estarão na classe Postagem

Herança e Inicialização

* Tanto classes pai como classes filhas podem ter seus próprios construtores. Mas, então, qual deles é responsável por construir um objeto?
  A resposta é: cada construtor constrói sua parte do objeto

* Note que o construtor de Mensagem e Foto possuem o atributo usuario, que é utilizado no construtor de Postagem

* Podemos chamar o construtor da classe pai no construtor da classe filha e adicionando os parâmetros correspondentes ao construtor que queremos chamar

* A chamada ao construtor super deve sempre ser a primeira dentro do construtor da subclasse.

Herança e Inicialização

* Construtor na classe pai: (Normal)

```
class Postagem:
    def __init__(self, usuario):
        self._usuario = usuario
        self._hora_postagem = time.time()
        self._curtidas = 0
        self._comentarios = []
```

* Construtor nas classes filhas:

```
class Mensagem (Postagem):
    def __init__(self, usuario, mensagem):
        Postagem.__init__(usuario)
        self._mensagem = mensagem

class Foto (Postagem):
    def __init__(self, usuario, imagem, legenda):
        Postagem.__init__(usuario)
        self._imagem = imagem
        self._legenda = legenda
```

Vantagens de usar Herança

* Evita duplicação de código: não é mais necessário copiar o código de uma classe para outra

* Reuso: O código existente pode ser reutilizado, ao invés de ser totalmente reimplementado

* Facilitar Manutenção: Se algo na classe pai for modificado, todas as classes filhas perceberão automaticamente esta modificação

* Extensibilidade: Adicionar novas classes à hierarquia torna-se muito simples. A nova classe é capaz de se integrar à aplicação mais facilmente

Classe Feed

Como fica a implementação da classe Feed na segunda versão da nossa rede social?

```
class Feed:
    def __init__(self):
        self._postagens = []

    def add_postagem(self, postagem: Postagem):
        self._postagens.append(postagem)

    def mostra_postagens(self):
        for postagem in self._postagens:
            postagem.display()
            print("-" * 40)
```

Método Inserir

```
mensagem1 = Mensagem("rei_do_python", "print('Hello World!')")

mensagem2 = Mensagem("profLucas", "Trabalho de Java para hoje!")

foto = Foto("lucasrafaelc", "praia.jpg", "Saudades das férias...")

feed = Feed()

feed.add_postagem(mensagem1)
feed.add_postagem(mensagem2)
feed.add_postagem(foto)
```

Repare que podemos passar um objeto do tipo Mensagem e Foto para um método que recebe uma Postagem como argumento
Isso acontece porque **toda Mensagem é uma Postagem** e **toda Foto também é uma postagem**

Polimorfismo

* Polimorfismo (poli = muitas, morphos = formas) refere-se a capacidade de um objeto em ser referenciado de formas diversas

* Assim, utilizamos a herança para programar no geral e não no específico

* A classe Feed continua funcionando se criarmos outros tipos de postagens na nossa rede social

Problema do Método display

* Você deve ter notado que o método display não consegue mostrar as informações específicas de uma `Mensagem` ou `Foto`

* Isso ocorre pois este método foi implementado em Postagem e não em Mensagem ou Foto

* O que acontece se movermos esse método para as classes filhas e removermos a implementação da classe pai? A classe Feed vai parar de funcionar!

```
ReiDoJava
0 segundos atrás  -  1 pessoas curtiram isso.
          Sem comentários.
-------------------
lucasrafaelc
0 segundos atrás
          1comentários.
-------------------
```

Sobrescrita

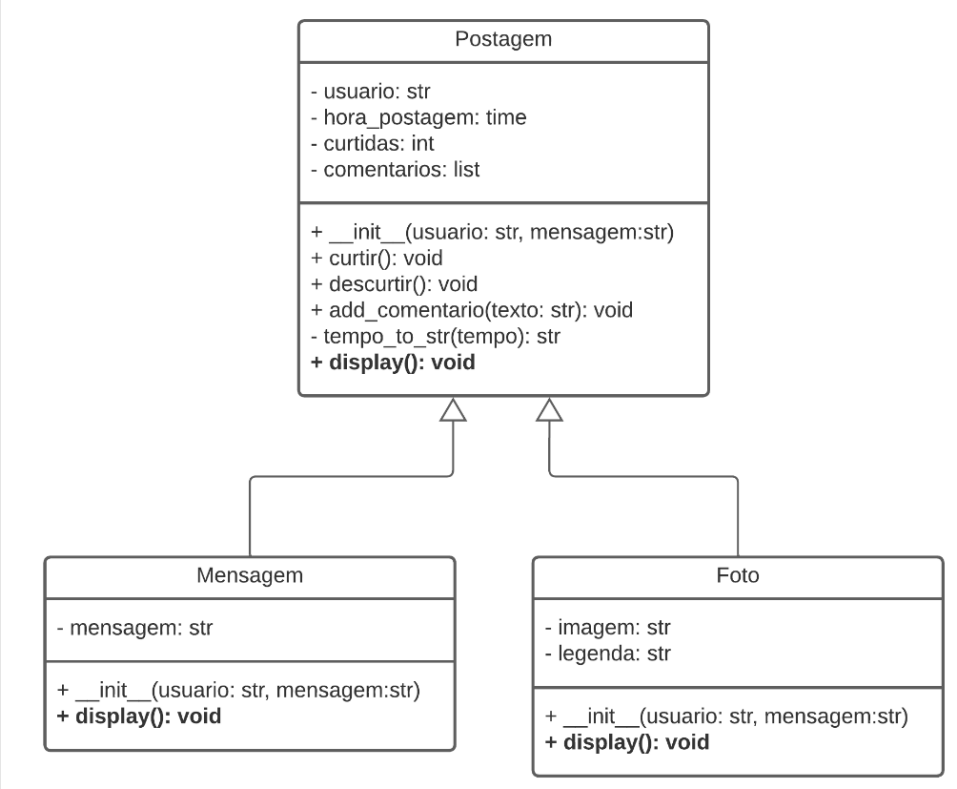
* A solução para o problema do método display é utilizar sobrescrita de métodos

* Sobrescrita ocorre quando um método da classe pai é definido novamente em uma classe filha, com a mesma assinatura: mesmo nome, mesmos argumentos e mesmo tipo de retorno

* No momento de escrever as postagens, a classe Feed invocará o método display correspondente ao tipo do objeto (Foto ou Mensagem)

* Não importa como referenciamos um objeto, o método invocado é sempre o que é da sua classe!

Sobrescrita



Sobrescrita

classe Postagem

```
        def display(self):
                # Imprime a postagem
                print(self._tempo_to_str(self._hora_postagem))

                # Imprime a quantidade de likes
                if self._curtidas > 0:
                    print(f"{self._curtidas} pessoas curtiram isso")
                else:
                    print()

                # Imprime os comentários
                if len(self._comentarios) == 0:
                    print("\tSem comentários.")
                else:
                    print(f"\t{len(self._comentarios)} comentários")
```

Sobrescrita

classe Mensagem

```
    def display(self):
            print(self._usuario + "diz:")
            print(f'"{self._mensagem}"')
            # Chama o método display na classe pai
            super().display()
```

classe Foto

```
    def display(self):
            # Imprime a postagem
            print(self._usuario, "fotografou:")
            print(f'[{self._imagem}]')
            print(self._legenda)
            # Chama o método display na classe pai
            super().display()
```

Sobrescrita

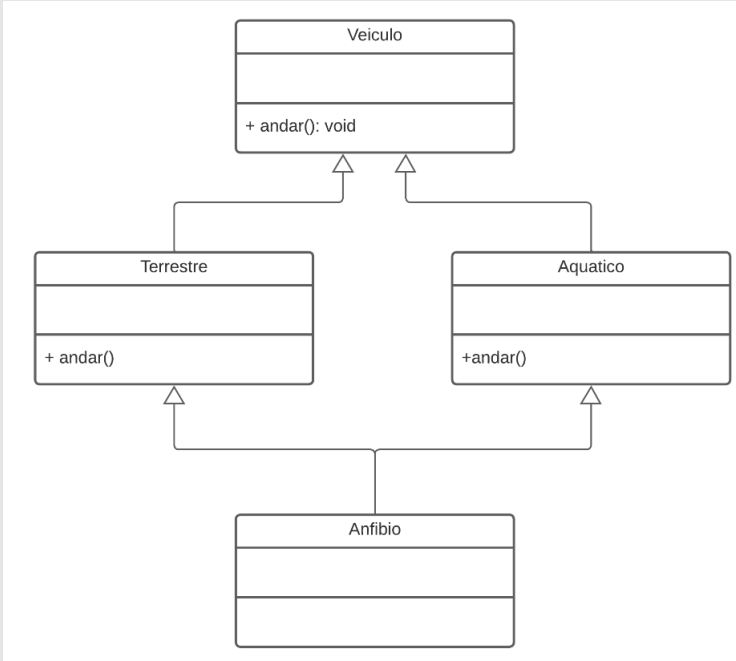
* O método display precisa ser implementado nas três classes da hierarquia, mesmo que o método executado sempre seja das classes filhas

* Métodos sobrescritos nas subclasses tem preferência sobre os métodos das superclasses. A busca pelo método que será executado inicia na base da hierarquia de classes e vai subindo por ela

* Quando um método é sobrescrito, somente a última versão desse método é executada. Se for necessário chamar o método da classe pai ele precisa ser invocado com `super().nome_método()`

Herança Múltipla

* Python admite herança múltipla, ou seja, uma classe pode derivar de mais do que uma classe pai.




```
class Veiculo:
    pass

class Terrestre(Veiculo):
    pass

class Aquatico(Veiculo):
    pass

class Anfibio(Terrestre, Aquatico):
    pass
```

Problema do Diamante

Quando o método andar for chamado em um objeto do tipo Anfíbio, qual dos métodos andar será de fato executado?

(A) Terrestre

(B) Aquatico

(C) Veiculo

```python
class Veiculo:
    def andar(self):
        print("Andar na classe Veiculo")

class Terrestre(Veiculo):
    def andar(self):
        print("Andar na classe Terrestre")

class Aquatico(Veiculo):
    def andar(self):
        print("Andar na classe Aquatico")

class Anfibio(Terrestre, Aquatico):
    pass

anfibio = Anfibio()
anfibio.andar()
```

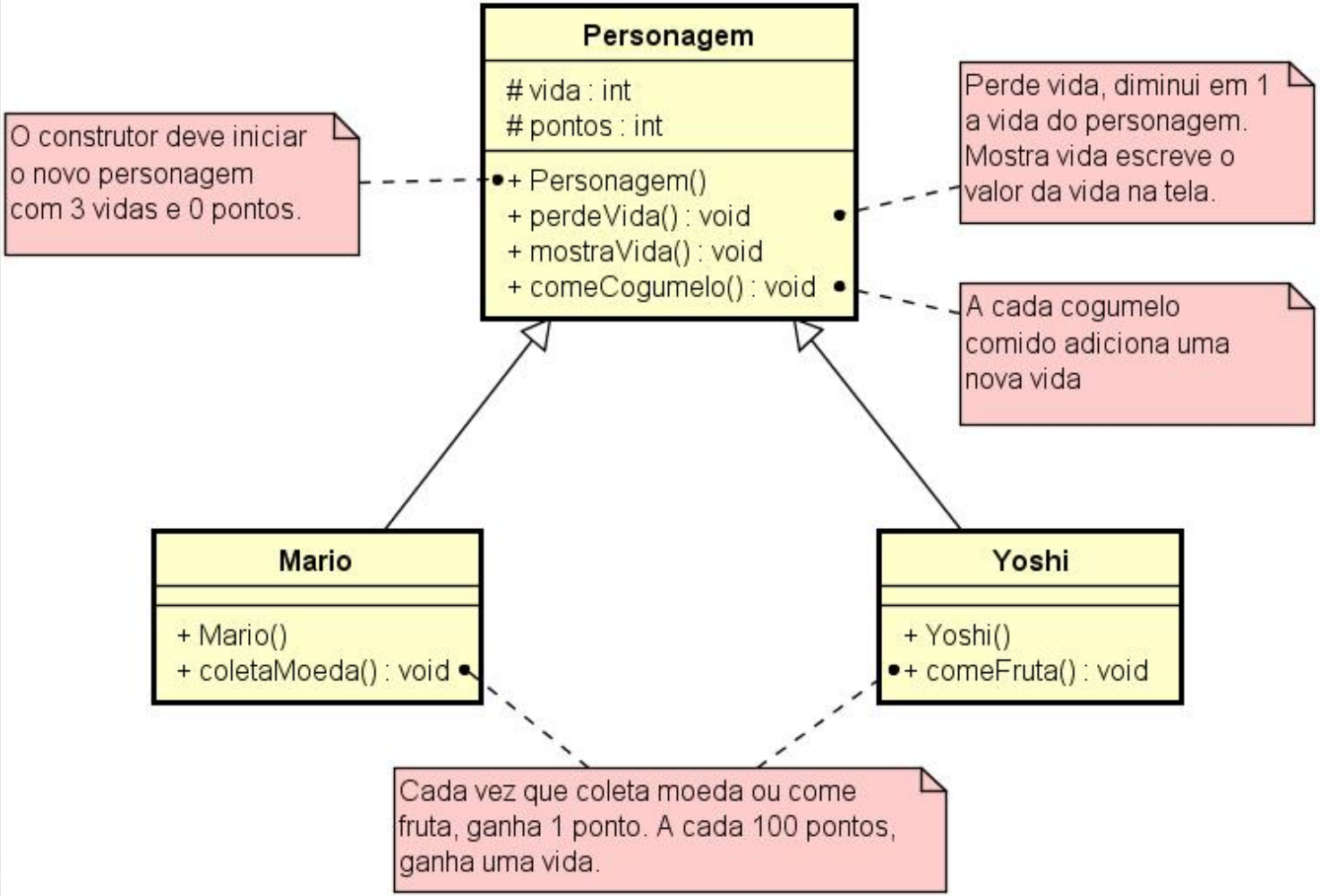
Problema do Diamante

* O Python resolve esse problema seguindo uma ordem específica para percorrer a hierarquia de classes

* Essa ordem é chamada MRO (Method Resolution Order). Toda classe tem um atributo `__mro__` que mostra essa ordem

```
print(Anfibio.mro())
```

```
[<class '__main__.Anfibio'>, <class '__main__.Terrestre'>, <class
'__main__.Aquatico'>, <class '__main__.Veiculo'>, <class 'object'>]
```


Personagem
vida : int
pontos : int
+ Personagem()
+ perdeVida() : void
+ mostraVida() : void
+ comeCogumelo() : void
O construtor deve iniciar o novo personagem com 3 vidas e 0 pontos.
Perde vida, diminui em 1 a vida do personagem. Mostra vida escreve o valor da vida na tela.
A cada cogumelo comido adiciona uma nova vida
Mario
+ Mario()
+ coletaMoeda() : void
Yoshi
+ Yoshi()
+ comeFruta() : void
Cada vez que coleta moeda ou come fruta, ganha 1 ponto. A cada 100 pontos, ganha uma vida.

Tamagotchi

Implemente um programa que permita a criação de amigos virtuais. Um amigo imita as ações cotidianas de seres humanos e/ou animais, como: dormir, comer, tomar banho, realizar atividades, etc. Para cada uma dessas ações, o amigo possui uma escala, o qual pode aumentar ou diminuir. Por exemplo, a escala “energia” varia conforme ele dormiu (ex: aumenta energia) ou joga futebol (ex: diminui energia).

Neste trabalho, você deverá implementar um sistema que permita criar amigos virtuais. Você deverá estabelecer 2 categorias de amigos (animais, personagens, etc.) e cada uma deve conter pelo menos 2 tipos de amigos (ex: cachorro, garoto, passarinho, monstro). Independentemente do tipo de amigo, este deverá oferecer, ao menos, 3 escalas (ex: carinho, energia, fome) e 6 atividades.