Tipos de ligações entre classes

terça-feira, 11 de outubro de 2022 19:46

Tipo de relacionamento entre classes

Associação

Na associação uma classe não depende da outra.

Na associação devemos criar um atributo de classe com o valor none para receber algo, a partir disso devemos fazer um getter e um setter

Exemplo:

```python
class Amor:

    def __init__(self,paixao):
        self.__paixao = paixao
        self.__at1 = None
    @property
    def paixao(self):
        return self.__paixao

    @property
    def at1(self):
        return self.__at1

    @at1.setter
    def at1(self,valor):
        self.__at1 = valor


class Novo:

    def __init__(self,escreve):
        self.__escreve = escreve

    def valor2(self):
        print("Eu estou aqui")


instancia1 = Amor('gero')
instancia2 = Novo('valor')
instancia1.at1 = instancia2

instancia1.at1.valor2()
```

Com isso a instância e o atributo vazio vai ser associado a outra instância pegando os valores de outra classe.

Da seguinte forma

Instância1 = classe1
Instância 2 = classe2
Instância 3 = classe 3

Então fica assim:
Instância1.atributovazio = instância 2

Logo podemos usar assim

Instância.atributovaziodainstancia1.atributoclasse2()

Agregação

Agregação em classes é basicamente criar uma relação entre classes onde elas dependem uma da outra.

Ex: é como se fosse um carro e as rodas, você não consegue dirigir um carro sem as rodas direito.

é basicamente 2 classes que uma funciona sem o outro porém so funciona corretamente uma com a outra.

Ex pratico:

```python
class Carrinho:
    def __init__(self):
        self.produtos = []

    def inserir_produto(self,produto):
        self.produtos.append(produto)

    def lista_produtos(self):
        for produto in self.produtos:
            print(produto.nome,produto.valor)

    def soma_total(self):
        total = 0
        for produto in self.produtos:
            total += produto.valor

class Produto:
    def __init__(self,nome,valor):
        self.nome = nome
        self.valor = valor
```

a classe carrinho precisa dos produtos da classe Produto para inserir algo na lista da classe carrinho, porém para isso é preciso instanciar o objeto que vai ser inserido na lista da classe.

```python
from teste1 import Produto,Carrinho

compras = Carrinho()

produto1 = Produto('camisa',50)
produto2 = Produto('Iphone',5000)
produto3 = Produto('roupa',500)

compras.inserir_produto(produto1)
compras.inserir_produto(produto2)
compras.inserir_produto(produto3)

compras.lista_produtos()
```

compras é a instância da classe Carrinho()

os produtos são a instância da classe Produto e com isso está inserindo o produto dentro da lista quando chamamos a classe e jogamos os parâmetros dentro.

Composição

na composição as classes estão ligadas entre si, os objetos de uma classe estão diretamente ligados a outra classe, ou seja quando apagamos os objetos de uma classe apaga junto os objetos da outra classe que estão ligados diretamente.

Ex:

```python
class Carrinho:
    def __init__(self):
        self.produtos = []

    def inserir_produto(self,nome,valor):
        self.produtos.append(Produto(nome,valor))

    def lista_produtos(self):
        for produto in self.produtos:
            print(produto.nome,produto.valor)

    def soma_total(self):
        total = 0
        for produto in self.produtos:
            total += produto.valor

class Produto:
    def __init__(self,nome,valor):
        self.nome = nome
        self.valor = valor
```

Os objetos da classe Produto estão sendo instânciados como parametros para a função da classe Produto, logo quando forem apagados a instância da classe carrinho também serão excluidos os objetos que estão sendo inseridos pelo inserir_produtos.

Herança

Quando falamos de herança estamos dizendo que uma classe irá receber outra como se fosse um parâmetro e logo recebe todos os metodos, objetos que estão dentro da outra classe.

Ex:

```python
class Carrinho:
    def __init__(self,nome,idade):
        self.nome = nome
        self.idade = idade
        self.produtos = []

    def inserir_produto(self,nome,valor):
        self.produtos.append(Produto(nome,valor))

    def lista_produtos(self):
        for produto in self.produtos:
            print(produto.nome,produto.valor)

    def soma_total(self):
        total = 0
        for produto in self.produtos:
            total += produto.valor

class Produto(Carrinho):
    pass
```

O que acontece é exatamente isso, você pode usar as mesmas coisas da outra classe para a que voce usou recebendo a classe como parâmetro.

```python
from teste1 import Produto,Carrinho

p1 = Carrinho('gero',22)
p2 = Produto('amor',12)

p2.inserir_produto(p1)
p2.lista_produtos()

print(p1.nome,p1.idade)
print(p2.nome,p2.idade)
```

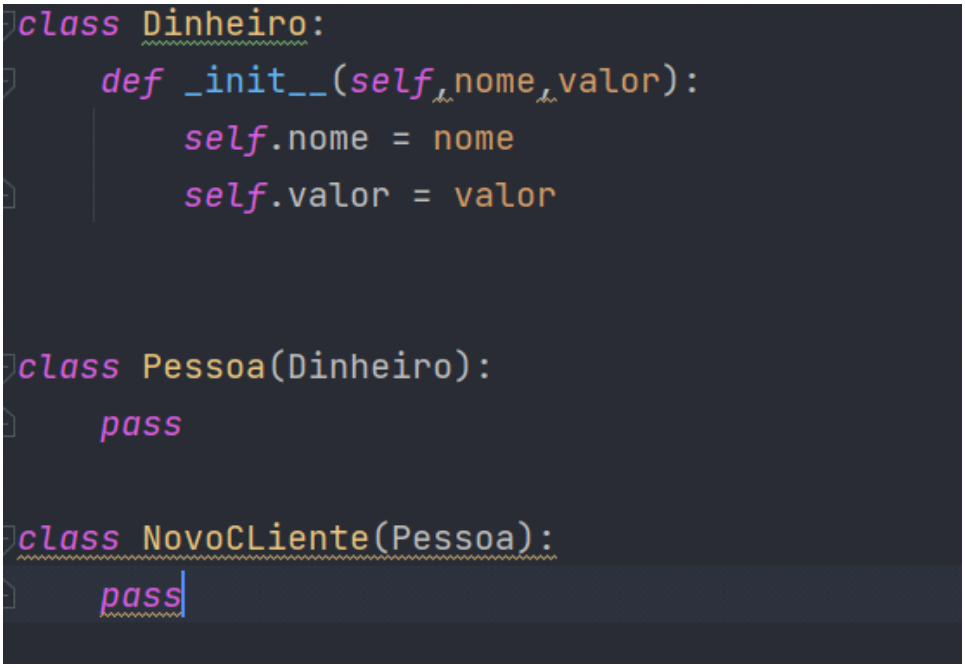
a classe carrinho é a super classe e as outras próximas serão chamadas de

subclasses, o que acontece é que as subclasses recebem herança da super classe, porém a super classe não recebe nada das sub classes e as sub classes não recebem nenhum metodo da outra sub classe.

Sobreposição de membros

Quando criamos uma herança multipla o que acontece é que uma classe é relacionada com herança a outra e criamos mais uma que recebe a segunda e também a primeira automaticamente.

Ex:



O que acontece é que a classe Dinheiro está sendo relacionada com a classe Pessoa, e a classe NovoCliente está relacionada a Pessoa logo NovoCliente também recebe tudo da classe Dinheiro.

o que acontece é o seguinte na sobreposição, existem metodos na classe Dinheiro que eu quero que sejam alterados para a classe que eu estiver chamando ao inves do metodo da classe Dinheiro então deve-se criar um metodo com o mesmo nome na classe que quer chamar e colocar o que quer fazer.

Ex:

```python
class Dinheiro:

    def __init__(self, nome, valor):
        self.nome = nome
        self.valor = valor

    def falar(self):
        print("Estou falando primeiro.")

    def valor(self):
        print(self.nome)


class Pessoa(Dinheiro):
    pass


class NovoCLiente(Pessoa):
    def falar(self):
        print('Estou falando...')
```

Ou seja o metodo falar está na class Dinheiro e está sendo sobrescrita na classe NovoCliente, logo vai ser chamada na classe NovoCliente de forma diferente.

Ex:

```python
from teste2 import Dinheiro,Pessoa,NovoCLiente

c = Dinheiro('ronaldo','22')
c1 = NovoCLiente('rony','20')
c1.falar()
```

```
C:\Users\geron\PycharmProjects\pythonPr
Estou falando...

Process finished with exit code 0
```

se eu quiser usar o metodo falar() da classe que estiver acima dessa NovoCLiente usamos a super().nomedafunção()

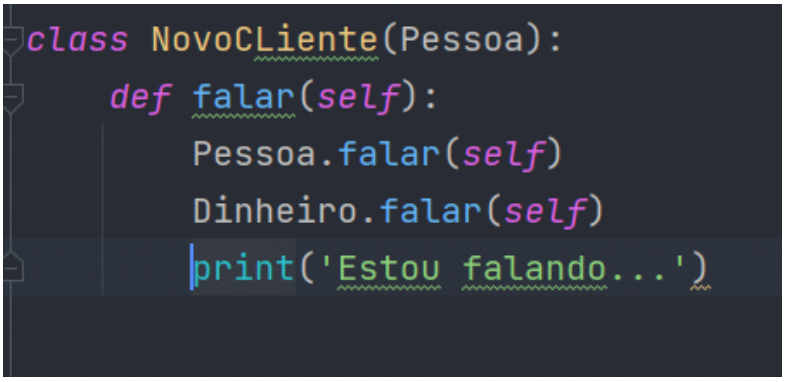
Ex:

```python
class Dinheiro:

    def __init__(self, nome, valor):
        self.nome = nome
        self.valor = valor

    def falar(self):
        print("Estou falando primeiro.")

    def valor(self):
        print(f'{self.nome}')


class Pessoa(Dinheiro):
    pass


class NovoCLiente(Pessoa):
    def falar(self):
        super().falar()
        print('Estou falando...')
```

Então o que acontece é que quando chamamos NovoCliente.falar() o metodo falar() vai buscar o falar() da super que no caso é qualquer metodo de uma classe que estiver acima dessae depois executa o resto que estiver no metodo da classe atual.

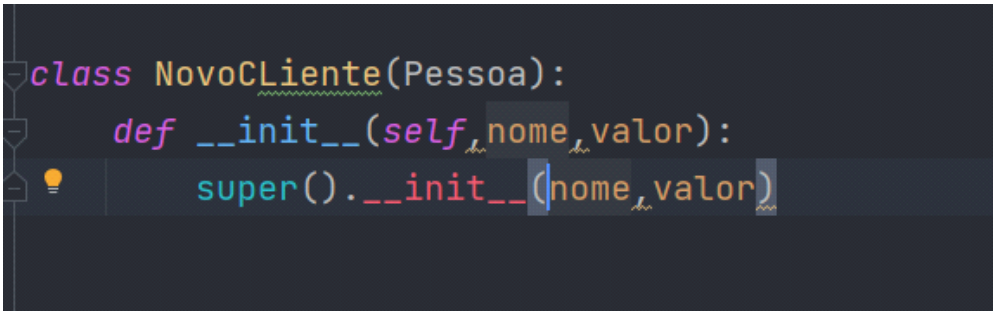
ou então podemos usar a classe + metodo + argumento do metodo

Ex:

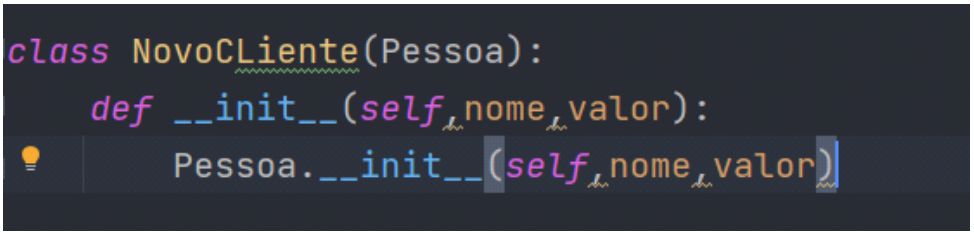


com isso podemos então sobrescrever o construtor da classe principal.

Ex:



Podendo ser dessa forma acima ou dessa forma:



Podemos também criar mais atributos de instancia dentro desse construtor.

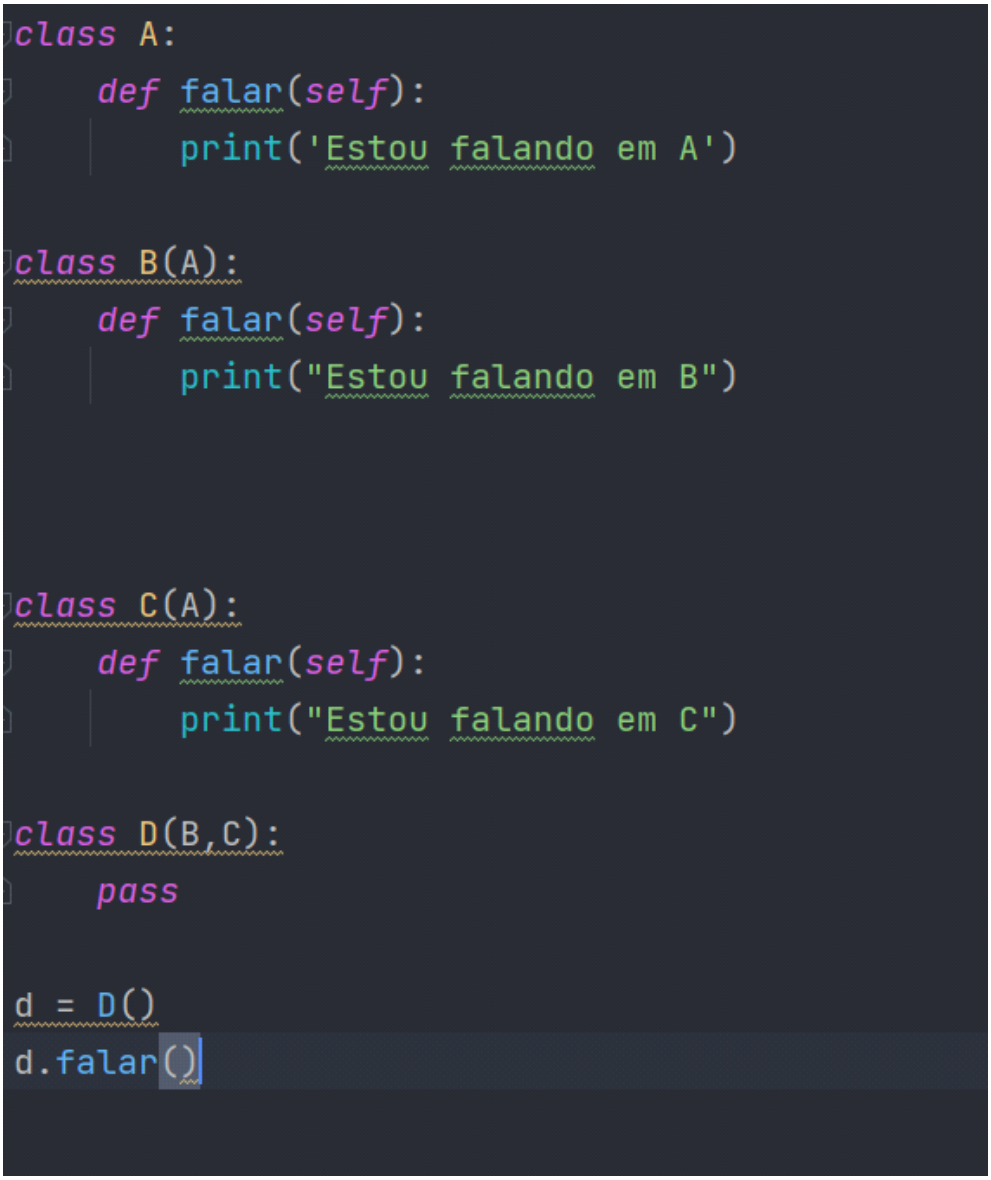
ex:

```python
class NovoCLiente(Pessoa):
    def __init__(self,nome,valor,sobrenome):
        Pessoa.__init__(self,nome,valor)
        self.sobrenome = sobrenome

    def falar(self):
        Dinheiro.falar(self)
        print(f'{self.nome} {self.sobrenome} está falando')
```

Quando for chamado a classe NovoCliente e o metodo falar self.nome será criado no seu construtor principal que é o do construtor da classe Dinheiro.

Herança Multipla

Quando se fala em Herança multipla é quando relacionamos uma classe com várias outras, ou seja mais de 1.



```python
class A:
    def falar(self):
        print('Estou falando em A')

class B(A):
    def falar(self):
        print("Estou falando em B")



class C(A):
    def falar(self):
        print("Estou falando em C")

class D(B,C):
    pass

d = D()
d.falar()
```

A classe D está Herdando duas classes.

Quando solicitamos um metodo que tem em várias das classes ele vai buscar em qual a primeira que foi relacionada, ou seja buscar na Classe B, Depois na Classe C.

Executando a primeira que encontrar.