Orientação a Objetos na Linguagem Python

1. Sobrecarga de operadores

Quando declaramos 2 variáveis com número do tipo inteiro, podemos por exemplo, realizar a soma entre eles. Se tratando da programação orientada a objetos quando criamos uma classe se quisermos "somar" dois objetos desta classe precisamos implementar o método `__add__` dentro da classe. Veja o exemplo:

```
class Cachorro:
      def __init__(self, nome, raca, idade):
          self.nome = nome
          self.raca = raca
          self.idade = idade

      def __add__(self, outro_dog):
           return self.idade+outro_dog.idade
```

Agora, se quisermos "somar" dois cachorros, podemos usar o operador "+"

```
dog1 = Cachorro("Rex", "bulldog", 3)
dog2 = Cachorro("Lili", "poodle", 1)
print(dog1+dog2)
```

O resultado da execução acima vai ser a soma das idades dos dois cachorros.

Segue uma lista com os operadores mais utilizados e seus métodos correspondentes:

Operador	Método	Operação
+	`__add__`	Adição
-	`__sub__`	Subtração
*	`__mul__`	Multiplicação
/	`__div__`	Divisão
%	`__mod__`	Módulo
**	`__pow__`	Potência
<	`__lt__`	Menor que
>	`__gt__`	Maior que

2. Métodos Estáticos x Métodos de Classes

Já falamos anteriormente sobre os métodos estáticos. Um método estático é definido com o decorador `@staticmethod`. Ele não recebe implicitamente uma referência para a instância da classe (self). Portanto, um método estático não pode acessar ou modificar atributos de instância ou da classe. Um exemplo seria um método em uma classe Calculadora que realiza cálculos independentes de instâncias específicas:

```
class Calculadora:

    @staticmethod
    def soma(x, y):
        return x + y

print(Calculadora.soma(3, 5))
```

Por outro lado, um método de classe é definido com o decorador `@classmethod`. Ele recebe implicitamente uma referência para a própria classe (cls) em vez da instância da classe (self). Isso permite que o método acesse ou modifique atributos da classe, em vez de atributos de instância. Um exemplo seria um método em uma classe Usuarios que retona o nome do último usuário instanciado

```
class Usuario:
    ultimo_nome = "ninguém"

    def __init__(self, nome):
        self.nome = nome
        Usuario.ultimo_nome = nome

    @classmethod
    def getLastName(cls):
        return cls.ultimo_nome

# Criando instâncias de Usuario
print(Usuario.getLastName())
u1 = Usuario("Alice")
print(Usuario.getLastName())
u2 = Usuario("Bob")
print(Usuario.getLastName())
```

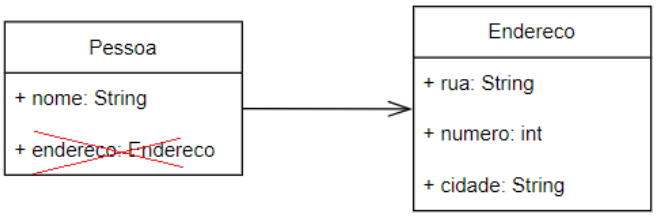
3. Associação

A associação é uma relação entre objetos onde um objeto faz referência ao outro, mas não possui uma dependência forte. Ou seja, os objetos estão relacionados, mas um pode existir sem o outro.

```
class Endereco:
    def __init__(self, rua, numero, cidade):
        self.rua = rua
        self.numero = numero
        self.cidade = cidade

class Pessoa:
    def __init__(self, nome, endereco):
        self.nome = nome
        self.endereco = endereco

endereco = Endereco("Rua A", 123, "Cidade X")
pessoa = Pessoa("João", endereco)

print(pessoa.nome)
print(pessoa.endereco.rua)
```

Neste exemplo, temos uma classe Endereco e uma classe Pessoa. A classe Pessoa possui uma referência a um objeto Endereco. No entanto, ambos os objetos podem existir independentemente um do outro. Se a pessoa não tiver um endereço, isso não afetará a existência da pessoa ou do endereço.

A representação em UML da associação é a seguinte:



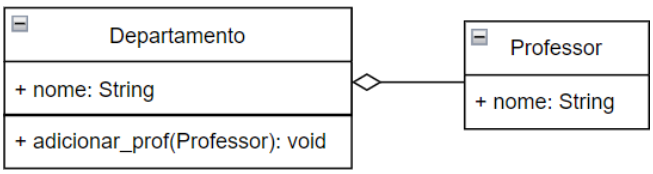
4. Agregação

A agregação não deixa de ser uma associação, mas existe uma exclusividade e determinados objetos só podem se relacionar a um objeto específico. É uma relação de um para muitos. Um objeto é proprietário de outros, mas não há dependência, então ambos podem existir mesmo que a relação não se estabeleça.

Um exemplo é a relação entre os professores e os departamentos. Departamentos podem ter vários professores. E o professor só pode estar vinculado a um único departamento. Mas eles são independentes. Um professor pode existir sem vínculo com um departamento e este não depende de professores para existir.

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

class Departamento:
    def __init__(self, nome):
        self.nome = nome
        self.professores = []

    def adicionar_prof(self, professor):
        self.professores.append(professor)

depto = Departamento("FGA")
prof1 = Professor("João")
prof2 = Professor("Maria")

depto.adicionar_prof(prof1)
depto.adicionar_prof(prof2)

print(depto.nome)
for prof in depto.professores:
    print(prof.nome)
```

A representação em UML do exemplo acima é a seguinte:



5. Composição

A composição é uma agregação que possui dependência entre os objetos, ou seja, se o objeto principal for destruído, os objetos que o compõe não podem existir mais. Há a chamada relação de morte.
Um exemplo é a relação entre uma a universidade e os departamentos. Além da universidade possuir vários departamentos, eles só podem existir se a universidade existir. Há uma dependência. Se a universidade deixar de existir, então os departamentos não existirão mais.

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

class Departamento:
    def __init__(self, nome):
        self.nome = nome
        self.professores = []

    def adicionar_prof(self, professor):
        self.professores.append(professor)

    def mostrar_professores(self):
      for prof in self.professores:
        print(prof.nome)

class Universidade:
    def __init__(self, nome):
        self.nome = nome
        self.deptos = []

    def adicionar_depto(self, depto):
        novo_depto = Departamento(depto)
        self.deptos.append(novo_depto)

uni = Universidade("UnB")
uni.adicionar_depto("FGA")
uni.adicionar_depto("CIC")

prof1 = Professor("Glauco")
prof2 = Professor("Maria")

uni.deptos[0].adicionar_prof(prof1)
uni.deptos[1].adicionar_prof(prof2)

print(uni.nome)
for dp in uni.deptos:
    print("Os profs do Depto "+ dp.nome + "
são: ")
    dp.mostrar_professores()
```

No exemplo acima, a classe Departamento foi instanciada dentro do construtor da classe Universidade. Assim, se a Universidade deixar de existir, os objetos da classe Departamento também serão extintos, mas os professores continuarão existindo, pois eles foram instanciados fora.

A representação em UML do exemplo acima é a seguinte: