04-POO-tipos-de-metodos

March 3, 2020

1 Programação Orientada aos Objetos (POO) - parte IV

Pedro Cardoso
(ISE/UAlg - pcardoso@ualg.pt)

1.0.1 "Métodos especiais"

- Métodos especiais são identificados por nomes no padrão `__metodo__()` e definem como os objetos derivados da classe se comportarão em situações particulares, como em sobrecarga de operadores.
- Métodos de objeto podem usar atributos e outros métodos do objeto.
- Os métodos ("normais") têm sempre uma primeira variável, por convenção `self`, que *representa o objeto*. Que "não conta na chamada" ao método.
- Alguns métodos de classe são decorados com `@classmethod sendo especiais pois como primeiro argumento passam uma *referência à classe*, por convençãocls`'
- Os métodos de classe apenas podem usar atributos e outros métodos de classe.
- Os métodos de classe podem funcionar como *factories*
- Métodos estáticos são aqueles que não tem ligação com atributos do objeto ou da classe. Funcionam como as funções comuns.

```
class Classe(supcl1, supcl2):
    """ Isto é uma classe """
    clsvar = []

    def __init__(self, args):
        """Inicializador da classe"""
        <bloco de código>

    def metodo(self, params):
        """Método de objeto"""
        <bloco de código>

    @classmethod
```

```
    def cls_metodo(cls, params):
        """Método de classe"""
        <bloco de código>

    @staticmethod
    def est_metodo(params):
        """Método estático"""
        <bloco de código>
```

Vejamos um exemplo

```
In [1]: class Pizza:

            # area de pizza por pessoa
            area_por_pessoa = 750.

            def __init__(self, ingredientes):
                self.ingredientes = ingredientes

            def __repr__(self):
                return f'Pizza({self.ingredientes})'

            @classmethod
            def margherita(cls):
                """devolve um objeto, instancia de Pizza, com os ingredientes
                da Pizza margherita"""
                return cls(['mozzarela', 'tomate'])

            @classmethod
            def prosciutto(cls):
                """devolve um objeto, instancia de Pizza, com os ingredientes
                da Pizza prosciutto"""
                return cls(['mozzarela', 'tomate', 'fiambre'])

            @staticmethod
            def para_quantas_pessoas(raio):
                """metodo (estatico) que estima e devolve para quantas pessoas
                é uma pizza, sabendo o seu raio devolve area_pizza / area_por_pessoa
                """
                area_pizza = 3.14 * raio ** 2
                return area_pizza / Pizza.area_por_pessoa

            @staticmethod
            def qual_o_raio(numero_pessoas):
                """metodo (estatico) que estima o raio que a pizza deve ter dado
                    o número de pessoas devolve
                """
                area_total = numero_pessoas * Pizza.area_por_pessoa
                return (area_total / 3.14) ** .5
```

```
In [2]: quatro_queijos = Pizza(['mozzarela', 'gorgonzola', 'requeijão', 'parmesão'])
        quatro_queijos

Out[2]: Pizza(['mozzarela', 'gorgonzola', 'requeijão', 'parmesão'])

In [3]: margherita = Pizza.margherita()
        margherita

Out[3]: Pizza(['mozzarela', 'tomate'])

In [4]: prosciutto = Pizza.prosciutto()
        prosciutto

Out[4]: Pizza(['mozzarela', 'tomate', 'fiambre'])

In [5]: r = 30
        f'uma pizza com {r}cmš dá para {Pizza.para_quantas_pessoas(r)} pessoas'

Out[5]: 'uma pizza com 30cmš dá para 3.768 pessoas'

In [6]: p = 4
        f'para {p} pessoas deve encomendar uma pizza com {Pizza.qual_o_raio(p)} cm de raio'

Out[6]: 'para 4 pessoas deve encomendar uma pizza com 30.909772123696634 cm de raio'
```