01-POO-introducao

March 3, 2020

1 Programação Orientada aos Objetos (POO) - parte I

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

1.1 Introdução

A orientação a objetos tenta gerir a complexidade inerente aos problemas do mundo real abstraindo o conhecimento relevante e **encapsulando-o** dentro de objetos.

Na POO um programa de computador é conceptualizado como um **conjunto de objetos** que trabalham juntos para realizar uma tarefa.

Cada objeto é uma parte do programa, interagindo com as outras partes de maneira específica e totalmente controlada.

Na **visão da orientação a objetos, o mundo é composto por diversos objetos** que possuem um conjunto de **características e um comportamento bem definido**.

No paradigma POO definimos **abstrações dos objetos** reais existentes.

Todo objeto possui as seguintes características: - **Estado**: conjunto de propriedades de um objeto (valores dos atributos). - **Comportamento**: conjunto de ações possíveis sobre o objeto (métodos da classe). - **Unicidade**: todo objeto é único (possui um endereço de memória).

1.2 Classe

Quando escrevemos um programa numa linguagem OO, não definimos objetos individuais. Em vez disso definimos as **classes** utilizadas para criar esses objetos.

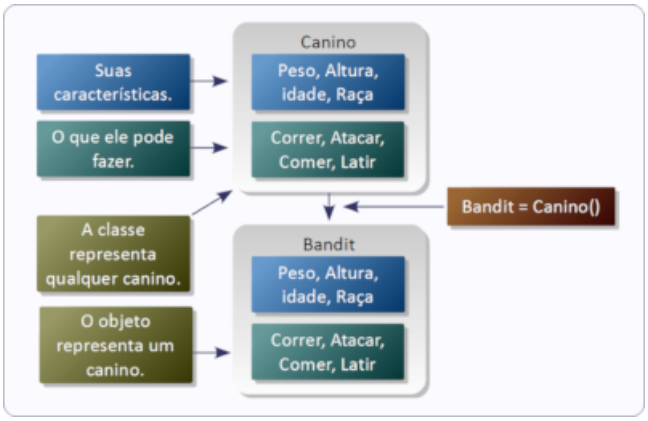
Podemos entender uma **classe** como um modelo ou como uma especificação para um conjunto de objetos, ou seja, a **descrição genérica dos objetos individuais** pertencentes a um dado conjunto.

A partir de uma classe é possível criar quantos objetos forem desejados.

Uma classe define as características e o comportamento de um conjunto de objetos}.

A criação de uma classe implica definir um **tipo de objeto** em termos de seus atributos (variáveis que conterão os dados) e seus métodos (funções que manipulam tais dados).

Definição: Uma classe é uma componente de um programa que descreve a *estrutura* e o *comportamento* de um grupo de objetos semelhantes - isto é, as informações que caracterizam o estado desses objetos e as ações (ou operações) que eles podem realizar



alt text

1.3 Declaração de uma classe

A declaração de uma classe segue a estrutura

```
class nome_da_classe:
    declaracao dos atributos da classe
    declaracao dos metodos da classe
```

A classe mais simples será

```
In [1]: class Classe:
            pass
```

Que podemos instanciar fazendo

```
In [2]: c = Classe()
```

`c` é um objeto do tipo `Classe`

```
In [3]: c

Out[3]: <__main__.Classe at 0x7fd12c512e10>
```

Que tem um conjunto de atributos e métodos que herdou da classe (`Object`). Mais à frente veremos o que é isto de "herdar"...

```
In [4]: dir(c)

Out[4]: ['__class__',
         '__delattr__',
         '__dict__',
         '__dir__',
         '__doc__',
         '__eq__',
         '__format__',
```

```
'__ge__',
'__getattribute__',
'__gt__',
'__hash__',
'__init__',
'__init_subclass__',
'__le__',
'__lt__',
'__module__',
'__ne__',
'__new__',
'__reduce__',
'__reduce_ex__',
'__repr__',
'__setattr__',
'__sizeof__',
'__str__',
'__subclasshook__',
'__weakref__']
```

1.4 Inicialização dos objetos

Quando um novo objeto é criado, o construtor da classe é executado. Em Python, o construtor é um método especial, chamado `__new__()` Após a chamada ao construtor, o método `__init__()` é chamado para inicializar a nova instância.

```
In [5]: class Classe:
            def __init__(self):  # self é uma referência ao próprio objeto, mais tarde veremos
                pass
```

1.5 Exemplo: a classe Carro

Para uma classe carro devemos considerar a "funcionalidades" comuns a todas...

O que tem um carro de importante? * Nome do dono * Marca * Modelo * Consumo * Cor * Kms (percorridos)

O que todo o carro faz e é importante para nós? Isto é, o que gostaríamos de "pedir a um carro"? * Definir Dono, Marca, Modelo, Consumo, Cor, Kms * Adicionar Kms percorrigos * Imprime o nome do dono, a cor, ... * Devolve o nome do dono, os Kms, o consumo, ... * ...

```
In [6]: class Carro:
            def __init__(self, cor, marca, modelo, dono, consumo, kms):
                self.cor = cor
                self.marca = marca
                self.modelo = modelo
                self.dono = dono
                self.consumo = consumo
                self.kms = kms
```

Podemos instanciar um Carro

```
In [7]: carro_1 = Carro('Branca', 'Fiat', '500', 'Claudia', 6, 20000)
        carro_1

Out[7]: <__main__.Carro at 0x7fd12c4c9b00>
```

Podemos criar várias instâncias/objetos com o mesmo modelo/*template* (classe)

```
In [8]: carro_2 = Carro('Vermelha', 'Seat', 'Ibiza', 'Margarida', 5.4, 12000)
        carro_2

Out[8]: <__main__.Carro at 0x7fd12c4c9e80>
```

Para aceder aos atributos e métodos usamos "." (ponto): * `objeto.atributo` * `objeto.metodo` * `classe.atributo` * `classe.metodo`

E podemos podemos criar métodos que recebem instancia de Carro como parâmetro... etc.

```
In [9]: def print_info(carro):
            print('A {} tem um {} {} de cor {} que gasta {}l/100Km e tem {}kms. Logo gastou {}l
                carro.dono, carro.marca, carro.modelo, carro.cor, carro.consumo, carro.kms, ca

        print_info(carro_1)
        print_info(carro_2)
```

```
A Claudia tem um Fiat 500 de cor Branca que gasta 6l/100Km e tem 20000kms. Logo gastou 1200.0l
A Margarida tem um Seat Ibiza de cor Vermelha que gasta 5.4l/100Km e tem 12000kms. Logo gastou
```

Em resumo: * Em Python os novos objetos são criados a partir das classes através de atribuição (de uma instanciação). * O objeto é uma instância da classe, que possui características próprias.

1.6 Referências a objetos

Um objeto existe em memória enquanto existir pelo menos uma referência a ele.

O interpretador de Python possui um recurso chamado coletor de lixo (*Garbage Collector*) que limpa da memória objetos sem referências.

Quando o objeto é apagado, o método especial `__done__()` é evocado.

Com isto, podemos ter mais do que uma referência para o mesmo objeto

```
In [10]: carro_a = Carro('Branca', 'Fiat', '500', 'Claudia', 6, 20000)
         carro_b = carro_a
```

`carro_a` e `carro_b` referênciam o mesmo objeto (têm o mesmo id)

```
In [11]: id(carro_a) == id(carro_b)

Out[11]: True
```

Logo, se alterarmos um objeto...

```
In [12]: carro_a.cor = 'Vermelha'
```

as duas referências ficam alteradas...

```
In [13]: carro_b.cor

Out[13]: 'Vermelha'
```

1.7 O que é um objeto em Python?

Afinal o que é um objeto em Python: * "Tudo" é um objeto, mesmo os tipos básicos, como números inteiros. * Tipos e classes são unificados. * Os operadores (e.g., +, -, ...) são na verdade chamadas para métodos especiais. * As classes são abertas (menos para os tipos *builtins*).

```
In [14]: a = 10
         type(a)
```

```
Out[14]: int
```

Quais são os métodos de um inteiro?

```
In [15]: print(dir(a))
```

```
['__abs__', '__add__', '__and__', '__bool__', '__ceil__', '__class__', '__delattr__', '__dir__
```

O que significa que os podemos chamar...

```
In [16]: a.__pow__(3)
```

```
Out[16]: 1000
```

1.8 Métodos

Mais do que atributos podemos definir comportamentos comuns aos objetos da classe `Carro` declarando "funções" a que chamamos *métodos* * Os métodos recebem argumentos/parâmetros * Podemos ainda definir variáveis locais dentro do método. * Tanto as variáveis definidas nos métodos como os argumentos têm âmbito/*scope* que se restringe ao método.

```
In [17]: class Carro:
             def __init__(self, cor, marca, modelo, dono, consumo, kms):
                 self.cor = cor
                 self.marca = marca
                 self.modelo = modelo
                 self.dono = dono
                 self.consumo = consumo
                 self.kms = kms

             def print_info(self):
                 print('A {} tem um {} {} que gasta {}l/100Km e tem {}kms.'.format(
                     self.dono, self.marca, self.modelo, self.consumo, self.kms))
```

```
In [18]: carro_a = Carro('Branca', 'Fiat', '500', 'Claudia', 6, 20000)
```

Para invocar o método usamos o "." (ponto)

```
In [19]: carro_a.print_info()
```

```
A Claudia tem um Fiat 500 que gasta 6l/100Km e tem 20000kms.
```

Exercício

Implemente na classe Carro um método que calcula e devolve o consumo total do Carro desde que foi comprado (tendo em conta a média e os kms percorridos)

```
In [20]: class Carro:
             def __init__(self, cor, marca, modelo, dono, consumo, kms):
                 self.cor = cor
                 self.marca = marca
                 self.modelo = modelo
                 self.dono = dono
                 self.consumo = consumo
                 self.kms = kms

             def print_info(self):
                 print('A {} tem um {} {} que gasta {}l/100Km e tem {}kms.'.format(
                     self.dono, self.marca, self.modelo, self.consumo, self.kms))

             def consumo_total(self):
                 return self.consumo * self.kms / 100

         carro_a = Carro('Branca', 'Fiat', '500', 'Claudia', 6, 20000)
         carro_a.consumo_total()

Out[20]: 1200.0
```