01

Usando métodos

Transcrição

Nós avançamos bastante no conteúdo e já definimos a função construtora `__init__` . Em algumas linguagens (como Java), esta função recebe o nome de **construtor**. O Python não se enquadra neste caso, porque uma função construtora não é um equivalente exato do método construtor.

É importante ter definido os atributos e suas principais características, o que mostramos como fazer. Agora falta adicionar o que um objeto `Conta` pode fazer.

De acordo com `teste.py` , definimos que podemos: depositar, sacar e tirar extrato.

```
def cria_conta(numero, titular, saldo, limite):
    conta = {"numero": numero, "titular": titular, "saldo": saldo, "limite": limite}

def saca(conta,valor):
    conta["saldo"] -= valor

def extrato(conta):
    print("Saldo é {}".format(conta["saldo"]))
```

Estas são as funções relacionadas com uma conta, que nas linguagens Orientadas a Objetas são nomeadas como métodos. Ou seja, os métodos são referentes às ações que um objeto sabe fazer. Onde eles serão colocados? Na **classe**, desta forma, sempre adicionaremos elementos relacionados à conta na classe `Conta` .

Primeiramente, adicionaremos uma nova função na classe. Neste caso, temos liberdade de usar o termo "função", mas ela não tem uma utilidade específica como `__init__` . Incluiremos a função `extrato` , que receberá a referência `self` do próprio editor porque iremos usá-lo para imprimir o saldo do titular, incluindo sempre a função `format()` dentro do `print()` .

```
class Conta:

    def __init__(self, numero, titular, saldo, limite):
        print("Construindo objeto ... {}".format(self))
        self.numero = numero
        self.titular = titular
        self.saldo = saldo
        self.limite = limite

    def extrato(self):
        print("Saldo {} do titular {}".format(self.saldo, self.titular))
```

O saldo está no objeto, para alcançá-lo, usamos a referência que sabe onde ele está. No caso, usaremos o `self` como referência.

Em seguida, criaremos a conta do titular `Nico` no console.

```
>>> from conta import Conta
>>> conta = Conta(123, "Nico", 55.5, 1000.0)
Construindo objeto ... <conta.Conta object at 0x105c24cf8>
```

Porém, se executarmos apenas `extrato()` no console, receberemos uma mensagem de erro, informando que `extrato` não existe. Isso aconteceu, porque o Python não sabe quais objetos queremos utilizar.

O próximo passo será criar o objeto `conta2`, que receberá novos atributos. Ou seja, temos dois objetos em uma mesma classe.

```
>>> from conta import Conta
>>> conta2 = Conta(321, "Marco", 100.0, 1000.0)
Construindo objeto ... <conta.Conta object at 0x109fdff60>
```

Para chamarmos `extrato()`, especificaremos de qual é o objeto que queremos os dados referentes ao extrato. Esta é uma diferença da abordagem procedural para OO.

Dentro da função `__init__()`, usamos `self` seguido do `.` para acessar o objeto, como em `self.numero = numero`. Faremos o mesmo para acessar `extrato`, que será antecedido pela referência.

```
>>> conta.extrato()
Saldo 55.5 do titular Nico
```

Ao executarmos esta linha, o Python entenderá que a referência `conta` aponta para o objeto `Conta`, baseado na classe homônima. O retorno será a mensagem `Saldo 55.5 do titular Nico`. Como usamos a referência, ele encontrou a referência e imprimiu o valor do saldo e o titular. Podemos fazer o mesmo com o outro objeto.

```
>>> conta2.extrato()
Saldo 100.0 do titular Marco
```

O Python nos retornou o valor da referência. Conseguimos adicionar o primeiro método dentro da classe `Conta`. No `teste.py`, onde seguimos a programação procedural, criamos `deposita` e `saca`. Iremos adicionar os dois métodos em `conta.py`, que receberam automaticamente a variável `self`, lembrando que `deposita()` deve agregar um valor ao saldo, por isso, além de `self`, receberemos `valor` como parâmetro.

Usaremos o valor para modificar o saldo do objeto, utilizando a referência que sabe onde está o objeto, e acessando o saldo deste, no qual incrementamos um valor. O método `saca()` é bastante semelhante, com a diferença que ele vai subtrair um valor.

```
def extrato(self):
    print("Saldo de {} do titular {}".format(self.saldo, self.titular))

def deposita(self, valor):
    self.saldo += valor

def saca(self, valor):
    self.saldo -= valor
```

A seguir criaremos um novo objeto no console e chamaremos o método `extrato()`:

```
>>> from conta import Conta
>>> conta = Conta(123, "Nico", 55.5, 1000.0)
Construindo objeto ... <conta.Conta object at 0x10db29e80>
>>> conta.extrato()
Saldo de 55.5 do titular Nico
```

Em seguida, será a vez de invocar o método `deposita()`. No caso, depositaremos `15` reais.

```
>>> conta.deposita(15.0)
```

Se executarmos, não receberemos uma mensagem de erro. Para testar se tudo saiu bem, vamos pedir o extrato novamente:

```
>>> conta.deposita(15.0)
>>> conta.extrato()
Saldo de 70.5 do titular Nico
```

Agora o saldo da conta tem `70.5` reais. Vamos experimentar usar o método `saca()`.

```
>>> conta.saca(10.0)
>>> conta.extrato()
Saldo de 60.5 do titular Nico
```

O saldo tem o valor `60.5`. Conseguimos sacar e manipular o saldo, além de imprimirmos o valor atualizado do objeto, com o método `extrato()`.

Observe que quem utiliza o objeto se comunica com ele por meio da referência `conta`, mas não sabemos como funciona o método `extrato()`, por exemplo, porque a função está encapsulada.

Em uma aplicação do mundo real, os métodos serão mais complexos. A função `saca()` deveria, por exemplo, verificar o limite. Mas ao executarmos `saca()`, é irrelevante a complexidade da funcionalidade, porque ela ficou encapsulada dentro do método. O uso de encapsulamentos é uma característica da Orientação a Objetos.

A classe possui todas as informações, como os atributos e funcionalidades da conta que, certamente em um sistema real, serão numerosas.

Vimos por que criamos a classe, mostramos como criar o objeto por meio da função construtora `__init__()` e definir os atributos dentro dela, além dos métodos e funcionalidades que o objeto deve ter. Aprendemos também como chamar os métodos usando a referência.

Nós já temos uma base de conhecimento para praticar com os exercícios.