01

Métodos privados

Transcrição

Falamos sobre **propriedades**, como escrever *getters* e *setters* de maneira mais elegante. Na resolução dos exercícios, transformamos os dois *getters* em propriedades adicionando um `@property` em cada, evitando o uso do `get` e `set`.

```
@property
def saldo(self):
    return self.__saldo

@property
def titular(self):
    return self.__titular

@property
def limite(self):
    return self.__limite

@limite.setter
def limite(self, limite):
    self.__limite = limite
```

No console, executaremos um exemplo de como criamos uma nova `conta` e conseguimos executar o método `saldo()`:

```
>>> from conta import Conta
>>> conta = Conta(123, "Nico", 55.5, 1000.0)
Construindo objeto ... <conta.Conta object at 0x110839668>
>>> conta.saldo
55.5
```

A seguir, continuaremos falando sobre a classe `Conta`, mas focando no método `saca()`:

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

def transfere(self, valor, destino):
    self.saca(valor)
    destino.deposita(valor)
```

Se analisarmos o `saca()`, veremos que ele tem alguns problemas. A conta do `Nico` tem um saldo de `55.5` e um limite de `10000.0`. Qual o valor máximo de saque que podemos fazer? Teoricamente, só poderíamos sacar `1055.5`. Mas é possível fazer uma **malandragem** e sacar mais:

```
>>> conta. saca(1200.0)
>>> conta.saldo
-1144.5
```

O saldo negativo ultrapassou o limite de `1000.0`, ou seja, não existe uma verificação. É o que faremos a seguir.

Nós queremos verificar se existe dinheiro suficiente na conta para que seja realizado o saque, ou seja, a soma do saldo com o limite, deve ser maior do que o valor que sacaremos. Para isto, usaremos `if/else` para fazer isso no método.

```
def saca(self, valor):
    if(valor <= (self._saldo + self.__limite)):
        self.__saldo -= valor
    else:
        print("O valor {} passou o limite".format(valor))
```

No console, vamos testar o código criado, chamando o método `saca()`.

```
>>> from conta import Conta
>>> conta = Conta(123, "Nico", 55.5, 1000.0)
Construindo objeto ... <conta.Conta object at 0x10c662cc0>

>>> conta.saca(1200.0)
O valor 1200.0 passou o limite
```

Agora quando tentamos sacar `1200.0`, o Python nos informa que ultrapassamos o limite. Se verificarmos o `saldo`, veremos que ele ainda é o mesmo, porque o saque não foi realizado.

Nós executamos o método `saca()` e apenas realizamos a verificação se o valor que desejamos sacar é `<=` ao valor do `saldo`. Isto deveria ficar um pouco mais expressivo no nosso código. Nós criamos uma regra simples, mas o sistema ainda ficará mais complexo, por isso, nós queremos deixar o código mais expressivo.

O próximo passo será adicionar o novo método `pode_sacar()`. Em seguida, moveremos a expressão que está localizada atualmente no `if`, iremos movê-la para o `pode_sacar()`. E o `if` de `saca()` passará a ser o responsável por chamar `pode_sacar()`.

```
def pode_sacar(self):
    pass

def saca(self, valor):
    if(self.pode_sacar(valor)):
        self.__saldo -= valor
    else:
        print("O valor {} passou o limite".format(valor))
```

Com o nosso código mais expressivo, ele se torna mais fácil de entender. Continuaremos trabalhando no método `pode_sacar()`, passando como segundo parâmetro a variável `valor_a_sacar`. Já o retorno da função será a condição que antes estava no `if`.

```
def pode_sacar(self, valor_a_sacar):
    return valor_a_sacar <= (self.__saldo + self.__limite)
```

Para tornar o método mais expressivo, vamos colocar a condição dentro de outra variável:

```
def pode_sacar(self, valor_a_sacar):
    valor_disponivel_a_sacar = self.__saldo + self.__limite
    return valor_a_sacar <= valor_disponivel_a_sacar

def saca(self, valor):
    if(self.pode_sacar(valor)):
        self.__saldo -= valor
    else:
        print("O valor {} passou o limite".format(valor))
```

Testaremos se está tudo funcionando.

```
>>> from conta import Conta
>>> conta = Conta(123, "Nico", 55.5, 1000.0)
Construindo objeto ... <conta.Conta object at 0x110091358>

>>> conta.saca(1200.0)
O valor 1200.0 passou o limite
>>> conta.saldo
55.5
```

Se tentarmos sacar `1200.0`, o valor não passará pela condição e ele retornará uma mensagem avisando isso. Para confirmarmos, pedimos o saldo e vimos que continua `55.5`. Podemos testar fazer um saque de um valor que esteja dentro da condição.

```
>>> conta.saca(100.0)
>>> conta.saldo
-44.5
```

Tudo continua funcionando da mesma forma, mas agora tivemos um retorno negativo. Em seguida, vamos testar `pode_sacar()`, que tem o retorno `true` ou `false`.

```
>>> conta.pode_sacar(100.0)
True
```

O método `pode_sacar()` facilita a compreensão do `if`. Porém, ele não deve ser usado desta forma. O `pode_sacar()` deve ter um aviso explícito de que o mesmo só poderá estar disponível dentro da classe. Precisamos alertar o desenvolvedor de que o método é privado e isso feito adicionando `__`.

```
def __pode_sacar(self, valor_a_sacar):
    valor_disponivel_a_sacar = self.__saldo + self.__limite
    return valor_a_sacar <= valor_disponivel_a_sacar

def saca(self, valor):
    if(self.__pode_sacar(valor)):
        self.__saldo -= valor
    else:
        print("O valor {} passou o limite".format(valor))
```

O método `__pode_sacar()` foi criado para ser executado apenas dentro da classe, por isso, o caractere *underscore* foi adicionado dentro do `if` também.

Se tentarmos executar `pode_sacar()` diretamente no console, receberemos uma mensagem de erro.

```
>>> conta = Conta(123, "Nico", 55.5, 1000.0)
Construindo objeto ... <conta.Conta object at 0x10159c898>
>>> conta.pode_sacar(100.0)
Traceback (most recent call last):
  File "<input>", line 1, in <module>
AttributeError: 'Conta' object has no attribute 'pode_sacar'
```

O método `pode_sacar` já não existe mais. Ele mudou de nome, passando a se chamar `_Conta__pode_sacar()` . Temos o nome da classe `Conta` , assim como os atributos privados. O Python permite que o método seja invocado, mas recomenda ao desenvolvedor que evite o `__pode_sacar()` , que é privado.

Assim como existem métodos privados, temos atributos que seguem a nomenclatura especial.