01

Getters e Setters

Transcrição

Nós avançamos nos conteúdos apresentados sobre a linguagem e vimos conceitos fundamentais como encapsulamento, coesão e referências. Mostramos como é a criação de objetos, o funcionamento das classes, métodos e atributos.

No entanto, quando falamos de **atributos privados** — que não são verdadeiramente privados —, ao adotarmos a nomenclatura especial adicionando `__` , geramos um pequeno problema.

Vamos recriar no console, a conta do Nico:

```
>>> from conta import Conta
>>> conta = Conta(123, "Nico", 55.5, 1000.0)
Construindo objeto ... <conta.Conta object at 0x10f6f5630>
```

Explicamos que não deveríamos poder modificar o valor do saldo diretamente. Apesar de podermos usar `_Conta__saldo` , o desenvolvedor é avisado de que não deveria fazer isso, porque o atributo é privado. Porém, como imprimiremos o `extrato()` ? Quando executamos `conta.extrato()` , o retorno será uma string formatada. No entanto, como nosso objetivo é acessar apenas o saldo, a solução será escrever um método que imprima unicamente o saldo, abaixo de `transfere()` .

```
def pega_saldo(self):
    return self.__saldo
```

Criamos um método com **uma** responsabilidade, que retorna o saldo. Poderíamos criar um método semelhante para retornar `__titular` .

```
def devolve_titular(self):
    return self.__titular
```

Ou para identificarmos `__limite` .

```
def retorna_limite(self):
    return self.__limite
```

Com o uso do `return` , sempre nos será retornado o valor de um atributo. Porém, o design do código está cheirando mal. Vamos testar se o código funciona.

```
>>> from conta import Conta
>>> conta = Conta(123, "Nico", 55.5, 1000.0)
Construindo objeto ... <conta.Conta object at 0x10adfd2b0>
>>> conta.pega_saldo()
55.5
>>> conta.devolve_titular()
'Nico'
```

```
>>> conta.retorna_limite()
1000.0
```

Escrevemos métodos específicos que nos devolvem os dados solicitados. É comum utilizarmos funcionalidades como estas para gerar relatórios, que nos mostre os dados principais da conta. Por serem recorrentes, existe uma nomenclatura padrão para esses métodos: **getters** (que nos **dão** um dado). Ou seja, a forma mais apropriada de nomear os métodos seria usando o nome `get`.

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

def get_saldo(self):
    return self.__saldo

def get_titular(self):
    return self.__titular
```

O uso do getters é um dos primeiros conceitos aprendidos pelos desenvolvedores Java. Além desses métodos usados apenas para retornar, existem aqueles que **modificam**. No caso, falamos dos **setters**.

Nós já temos métodos para acessar `saldo`, mas ainda temos que criar as formas de trabalhar com `limite`. O objetivo é podermos aumentar o limite por meio de `set_limite()`.

```
conta.set_limite(10000.0)
```

Este é o método com que definiremos um novo limite. A seguir, vamos definir o método `set_limite()`, para o qual, além do `self`, passaremos `limite` como parâmetro:

```
def set_limite(self, limite):
    self.__limite = limite
```

Lembrem-se que com `set` nunca retornaremos um valor, nós iremos `modificar` um atributo. Agora, colocaremos um novo `limite` no atributo `__limite` e testaremos no console.

```
>>> from conta import Conta
>>> conta = Conta(123, "Nico", 55.5, 1000.0)
Construindo objeto ... <conta.Conta object at 0x10d92c160>
>>> conta.get_limite()
1000.0
>>> conta.get_saldo()
55.5
>>> conta.get_titular()
'Nico'
>>> conta.set_limite(1000.0)
```

Como explicamos, o `set_limite()` apenas altera, então, `get_limite` nos informará se o saldo foi atualizado.

```
>>> conta.get_titular()
'Nico'
>>> conta.set_limite(1000.0)
>>> conta.get_limite()
1000.0
```

Importante: Usem os *getters* e *setters* com parcimônia. Eles devem ser criados apenas quando forem necessários.

O método `set_limite()` é útil, porque o limite pode ser alterado dentro do contexto de negócio. Mas, por exemplo, o número da conta de um cliente não deve mudar. Neste caso, é inapropriado implementarmos `set_numero`. Se o cliente encerrar uma conta e, depois, quiser abrir uma outra, ele receberá um novo número. Mas trata-se de um número fixo.

Temos que ficar atentos para evitar criar funcionalidades inutilmente. Mais adiante, conheceremos uma alternativa para esses métodos *getters* e *setters*. Aproveite para praticar com os exercícios.