10

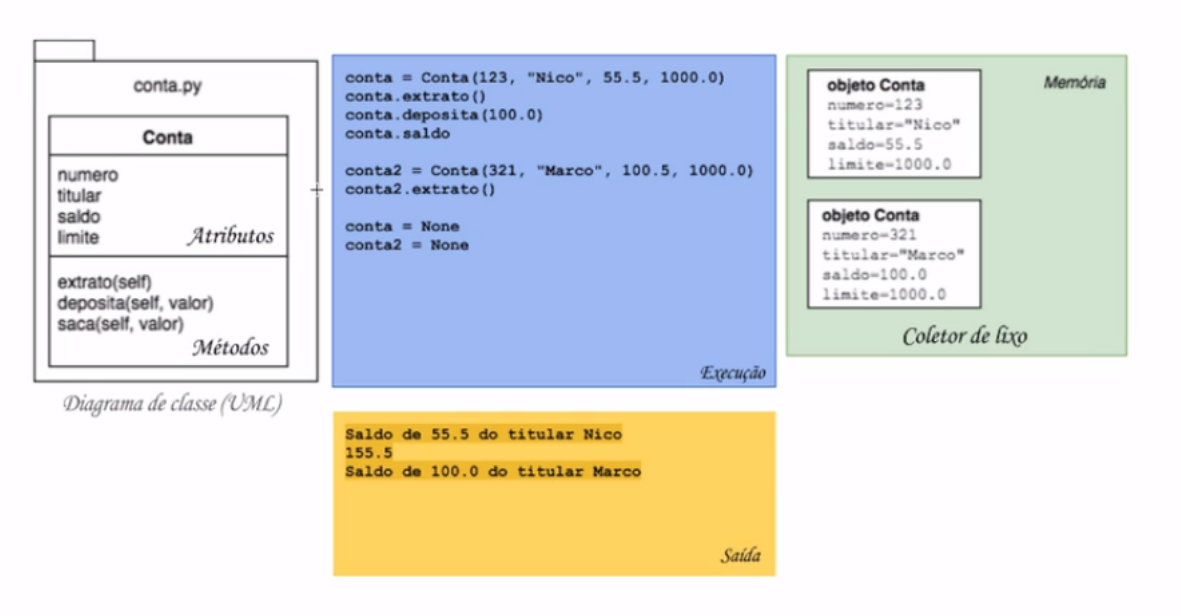
Revisão e Conclusão do curso

Transcrição

A seguir, faremos uma revisão. Mesmo que ache estranho fazer tantas revisões, é importante que você tenha uma boa base sobre Orientações a Objeto. Falaremos em outros cursos sobre outros tópicos, como herança, composição, relacionamentos e conceitos mais avançados.

Mas com a base que desenvolvemos, aprender coisas novas será mais fácil. Vamos relembrar os pontos principais vistos no curso.

Nossa motivação inicial era unir os dados e o comportamento, ou seja, juntarmos os atributos e os métodos. Tudo foi agrupado dentro de uma classe.



A classe é a menor unidade de organização no mundo OO. Colocamos os elementos dentro, com os referentes de atributos e os diferentes métodos. Colocamos dentro da classe, tudo o que está relacionado a ela. No caso, os **atributos** relacionados com a classe `Conta` são:

- numero
- titular
- saldo
- limite

Os **métodos** relacionados são:

- extrato(self)
- deposita(self, valor)
- saca(self, valor)

Isto significa que evitaremos colocar na `Conta` o código relacionado com impostos da nota fiscal, porque as duas coisas não estão relacionadas. Seria mais apropriado criar a classe `notaFiscal` e adicionar o código relacionado.

Surge a pergunta: onde os programadores procuram as funcionalidades no seu projeto? A resposta é: na classe relacionada. Desta forma, mantemos o código organizado e separamos as responsabilidades.

Vimos como escrever uma classe no mundo Python, adicionamos um construtor e, além disso, mostramos como ele é executado. Usamos o nome da classe, depois, passamos os parâmetros como vemos no conteúdo do quadro azul do diagrama:

```
conta = Conta(123, "Nico", 55.5, 1000.0)
conta.extrato()
conta.deposita(100.0)
conta.saldo
```

E o resultado será um objeto:

```
objeto Conta

numero = 123
titular = "Nico"
saldo = 55.5
limite = 1000.0
```

A responsabilidade de criar o objeto fica por conta do Python. Sabemos que um espaço é alocado para representar os atributos e o resultado da execução devolve a referência.

Essa referência sabe onde está guardado o objeto. Tendo esse endereço, podemos interagir com a classe, trabalhando com o objeto. Ou seja, quando escrevemos `conta.extrato()` vai executar o elemento do `extrato` . Com ele, acessaremos o objeto por meio do `self` .

Falamos também que o `self` é uma referência que sempre assume o valor da referência que fez a chamada. Por exemplo, se a referência é `conta` , o `self` será um equivalente na linha em que for utilizado: `deposita(self, valor)` .

No arquivo `conta.py` , implementamos vários métodos, como `extrato()` e `deposita()` , mostramos ainda como tornar um método privado. Ao adicionarmos os dois *uderscores* (`__`) como em `__pode_sacar()` , o desenvolvedor é alertado que só deve utilizá-lo dentro da classe `Conta` .

Criamos atributos privados usando a mesma nomenclatura, como `__numero` e `__titular` .

Os métodos podem crescer e ficar ainda maiores e mais complexos, mas para quem faz as chamadas do `deposita()` , a quantidade de linhas de um método é irrelevante. Isto ocorre, porque o código está encapsulado:

```
def deposita(self, valor):
    self._saldo += valor
```

Em uma conta da vida real, provavelmente, o método seria mais complexo e seria necessário adicionar verificações antes da realização do depósito.

Quem usa a classe `Conta` e chama `deposita()`, usa de alto nível, sem a preocupação com os detalhes da implementação.

No curso, falamos sobre propriedades (`properties`). Quando digitávamos no console `conta.saldo`, parecia que acessávamos simplesmente um atributo, porque não usamos parênteses dos métodos. Mas por baixos dos panos, o método anotado com `@property`. Também temos *properties* que podem alterar os ***setters***.

Uma classe pode ter diversos objetos. Usando a analogia da receita, ela pode ter diversos elementos. Basta repetir a linha que constrói o objeto, passando os novos valores.

Se temos um novo objeto de memória, teremos uma nova referência que guardará o valor do endereço do objeto.

Mostramos que é possível zerar uma referência, com o uso do `None`.

```
conta = None
conta2 = None
```

Nós podemos falar que um referência não pode apontar para um objeto e se ela não aponta, guardará o valor `None`. Isto significa que o objeto criado ficou abandonado, porque a conta estava apontando para o objeto, porém ela foi zerada. Por isso, o objeto ficou perdido. Para casos como esse, o Python tem **coletor de lixo**, responsável por procurar os objetos que foram criados há muito tempo mas não são mais utilizados no projeto.

Criamos uma classe com vários objetos que podem reaproveitar os métodos com as funcionalidades encapsuladas.

Trabalhamos com um diagrama de classes bem simples, que utiliza a linguagem de notação UML. Vimos que em alguns casos, os métodos não estão relacionados com o objeto. Mostramos um exemplo em que gostaríamos de usar um método antes de ter o objeto, que recebem o nome de ***static method*** (método estático).

No entanto, eles devem ser usados com parcimônia. O objetivo da Orientação a objetos é a criação de objetos. Se trabalhamos apenas com métodos estáticos isso não acontece. Mas é um recurso oferecido pelo Python e que pode fazer sentido.

Esclarecemos conceitos de método, atributo, como acessar os métodos e como funciona o uso do `self` e os atributos privados. Falamos sobre encapsulamento e coesão. Vimos conceitos fundamentais do Paradigma Orientado a Objeto.

No próximo curso, iremos além. Serão apresentadas as peças que faltam do mundo OO. Falaremos sobre **associações** entre classes e mostraremos como funciona a **herança** no Python. Abordaremos também conceitos como agregação e composição. Entenderemos quando é mais apropriado utilizar cada tipo de relacionamento.

Outro ponto forte será o **tratamento de erro**, veremos como nossa aplicação deve se comportar quando acontece algo inesperado, como devem se comportar as ações do código. Mostraremos como descobrir onde está o problema e depurar a aplicação. Tudo isso será apresentado passo a passo.

Temos um prato cheio para o próximo curso, além de tópicos que podemos incrementar no mundo Orientado a Objetos, inclusive, indo além dele.

Você está convidado a continuar na nossa jornada nesta viagem pela Orientação a Objeto. Agradeço que tenha chegado até aqui e te espero no próximo curso.