1 October 2019

Orientação a Objetos em Python

SIDIA - Setembro/2019

Jailson P. Januário
jpj.ads@uea.edu.br
Github:@jailsonpj

Apresentação

- **Jailson Pereira Januário**

- Graduando em Sistemas de Informação (EST/UEA)
- Pesquisador no Laboratório de Sistemas Inteligentes
- Buritech

- *Machine Learning*
- *Deep Learning*
- Entusiasta Python
- Coordenador do PyData Manaus

O que veremos?

- O que é Orientação a Objetos?
- Como utilizar?
- Entendendo as definições de classes, objetos, atributos e métodos
- Como fica a memória quando rodamos um programa orientado a objetos?

Orientação a Objetos - OO

- Em pyhton, não existe variáveis de tipos primitivos. Tudo é objeto
- Diferente da programação estruturada, em OO juntamos dados e operações em um mesmo lugar (objeto)
- As definições de quais dados e operações um objeto possui são feitas em classes

Benefícios da abordagem OO

- Modularidade: Uma vez criado, um objeto pode ser passado por todo o sistema
- Encapsulamento: Detalhes de implementação ficam ocultos externamente ao objeto
- Reuso: Uma vez criado, um objeto pode ser utilizado em outros programas
- Manutenibilidade: Manutenção é realizada em pontos específicos do seu programa

Objetos

- São 'Coisas' do mundo real, que possuem dados (atributos) e realizam ações (métodos) relevantes
- O estado de um objeto pode variar de acordo com o tempo de execução de um programa
- São oriundos (instâncias) de classes

Classes

- A classe é o conjunto de especificações de atributos e métodos que um objeto possui
- Todo objeto é criado (ou instanciado) a partir de uma classe, respeitando suas regras
- Um objeto é uma instância de uma classe

Pensando em sistema

- Imagine que você é programador de uma instituição financeira e precisa criar um sistema para cadastro de contas bancárias de clientes
- Consegues perceber que a conta bancária é uma entidade importante para este contexto?

Pensando na conta bancária

- Quais características da conta bancária são importantes para o nosso projeto?
  - número da conta, agência, nome do cliente e saldo
- Quais operações poderíamos realizar em nossas contas bancárias?
  - sacar, depositar, transferir, ver saldo

Pensando na modelagem

ContaBancaria
+ numero: String - agencia: String - nome_cliente: String - saldo: double
+ sacar(valor: double) : boolean + depositar(valor: double): void + transferir(valor: double, destino : ContaBancaria): boolean

Vamos criar nosso 1o Projeto

- Crie um projeto chamado **Financeiro**
- Crie arquivo model.py
  - Em python, é comum que as classes que representam entidades fiquem em um arquivo chamado model.py

Exercício 1 - Conta bancária

- Agora que criamos nosso projeto financeiro, chegou a hora de transformar o diagrama de classes em código fonte
- Abra o arquivo model.py e crie a classe ContaBancaria

Exercício 1 - Conta bancária

```
1  class ContaBancaria(object):
2      def __init__(self):
3          self.agencia = None
4          self.numero = None
5          self.nome_cliente = None
6          self.saldo = 0.0
```

Por dentro do código

- Acabamos de criar nossa 1º classe Python
- Por ora, precisamos saber apenas que:
  - **class** - indica a criação de uma classe
  - **__init__** - método para inicialização de objeto
  - **self** - objeto recebido como parâmetro
  - self.**variavel** - declaração de atributos

Criação e uso de Objetos

- Agora que definimos nossa classe, podemos criar objetos do tipo **ContaBancaria**, usando a sintaxe:

```
novo_objeto = ContaBancaria()
```

- Para acesso aos **atributos** da conta, podemos usar o operador '.' (ponto)

Exercício 02 - Usando a conta

- Crie um novo arquivo teste_conta.py
- Para usar a definição de conta bancária, precisamos **importar** o arquivo model.py
- Crie um novo objeto
- Informe os dados do objeto
- Imprima os valores informados
- Execute o arquivo teste_conta.py

Exercício 02 - Usando a conta

```
1  from model import ContaBancaria
2
3  conta = ContaBancaria()
4
5  conta.agencia = '02333'
6  conta.numero = '1234-5'
7  conta.nome_cliente = 'Maria Jose'
8  conta.saldo = 1500.0
9
10 print('Cliente ' + conta.nome_cliente)
11 print('Agencia %s e conta %s' % (conta.agencia, conta.numero))
12 print('Saldo ' + str(conta.saldo))
```

Executando o script de teste

- Diferente do JAVA, não precisamos de uma classe para realizar os testes
- Criamos um script python simples, para a execução dos testes
- Basta executar o arquivo teste_conta.py

Métodos

- Representam ações que objetos podem realizar sobre seus atributos
- São equivalentes às funções e procedimentos da programação estruturada
- Costumam ser definidos após a declaração de atributos

Métodos

- Vamos pensar nos métodos da nossa classe ContaBancaria
- Qual é a lógica necessária para a realização de depósito e saque da conta?
- Altere a classe ContaBancaria, do arquivo model.py, e implemente os métodos sacar(valor) e depositar()

Novos métodos, novos testes

- Vamos testar nossos novos métodos?
- Atualize o arquivo teste_conta.py
- Faça um depósito de R$ 500,00
- Exiba mensagens de feedback para o usuário, informando o resultado da operação de saque

Trabalhando com referências

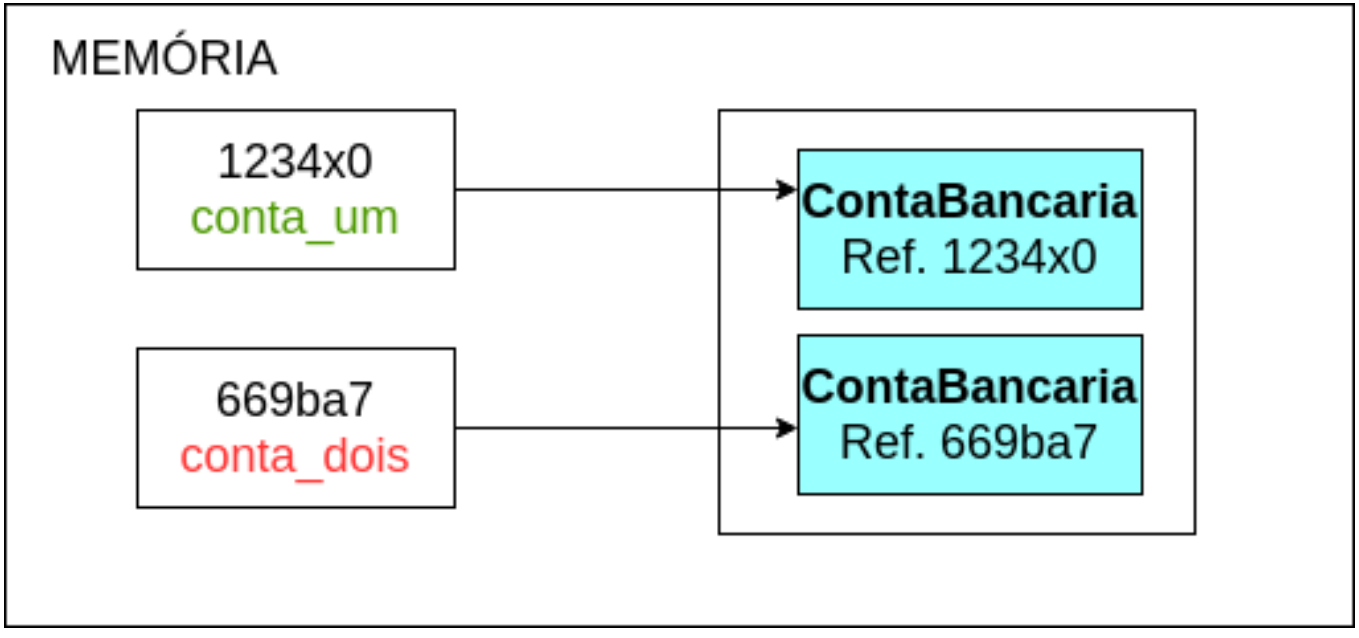
- A partir de uma instrução:
  - `conta = ContaBancaria()`
- É comum escutarmos a afirmação:
  - "A variável conta é um objeto"
- Contudo, a variável conta NÂO é um objeto
- A conta é uma referência para um objeto

Trabalhando com referências

- Variáveis de referências guardam os endereços de memória onde foram criados os objetos
- Em determinados pontos do sistema, é possível que mais de uma variável de referência aponte para um determinado objeto

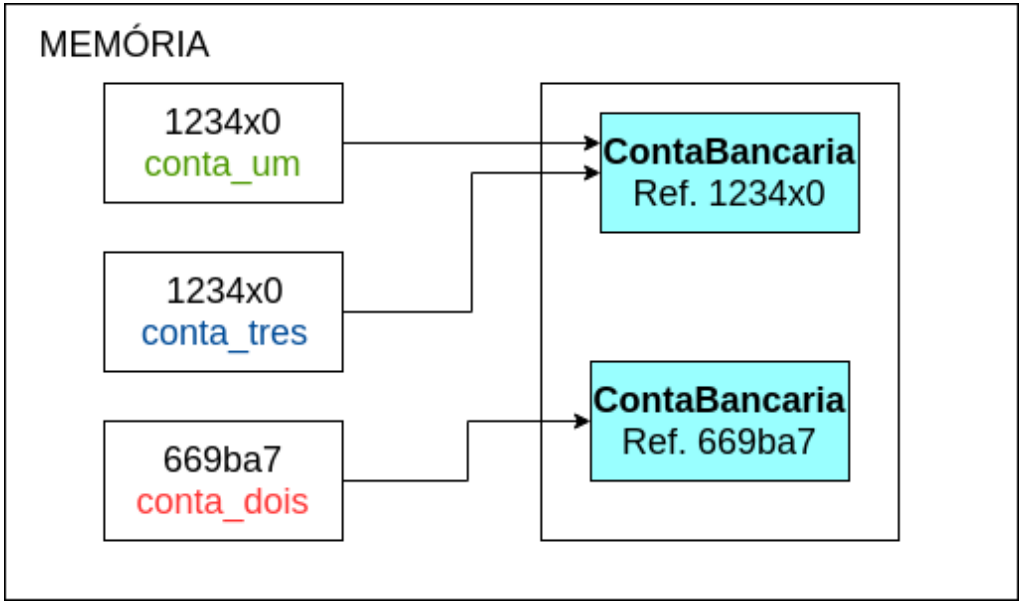
Trabalhando com referências

- Como fica a memória, após a execução de:
- `conta_um = ContaBancaria()`
- `conta_dois = ContaBancaria()`



Um objeto, duas referências

- `conta_um = ContaBancaria()`
- `conta_dois = ContaBancaria()`
- `conta_tres = conta_um`



Transferência entre contas

- Agora, precisamos implementar o método para transferência de valores entre contas bancárias
- Podemos assumir que o método transferir de uma conta vai receber o valor de transferir e a conta de destino da transferência

Exercício 03 - Novo método

- Altere a classe **ContaBancaria** criando um método **transferir**
- Crie duas contas bancarias: **origem**, com saldo inicial = 1200 e **destino**, sem saldo
- Transfira 500 da conta **origem** para **destino**

Refletindo a respeito

- O que aconteceu com a conta que foi passada como parâmetro para o método **transferir**?
- O objeto foi clonado?
- Negativo
- Passamos a **referência** para a conta de destino

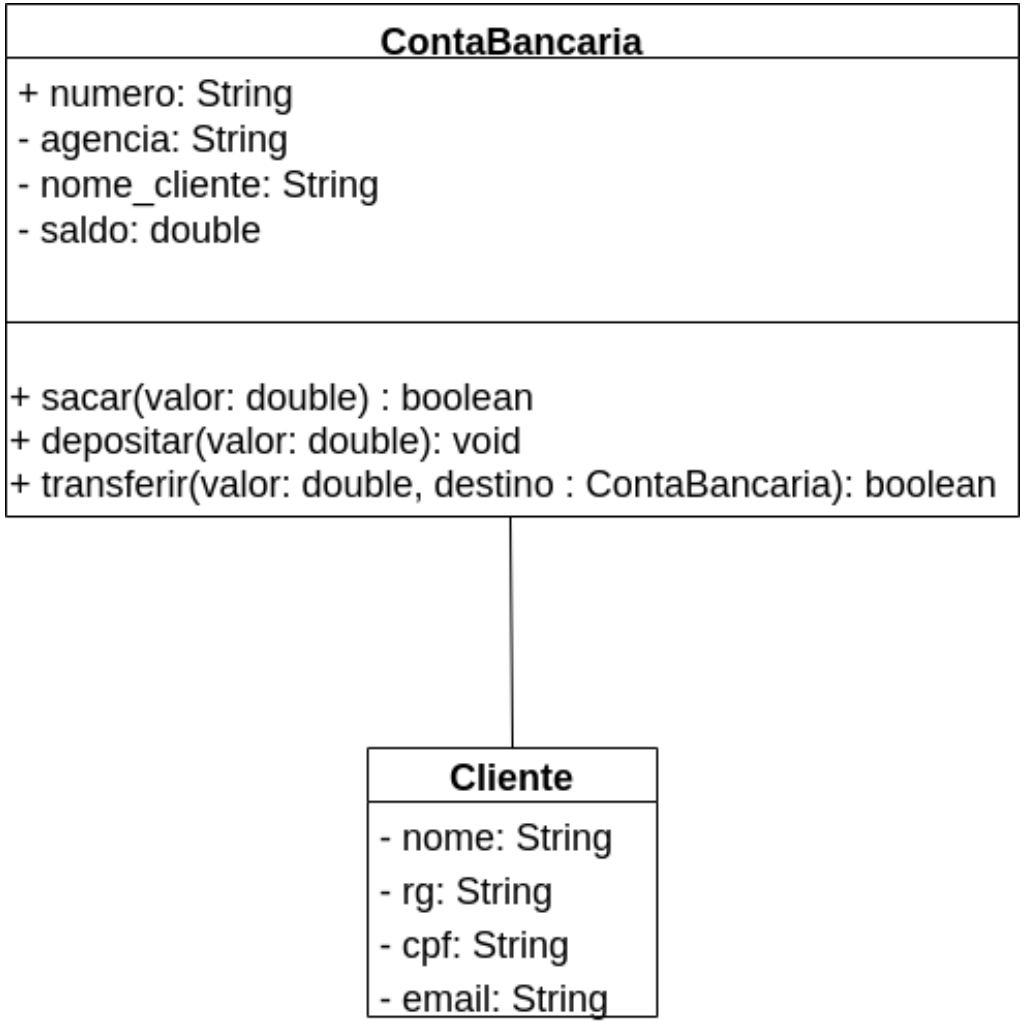
E o tempo passa

- O tempo passa e surge a necessidade de armazenamento de novos dados
- Agora, precisamos armazenar RG, CPF e email do **cliente**
- Vamos criar novos atributos na classe **ContaBancaria**?
- Negativo

Preocupados com a coesão

- Em OO, uma classe possui **alta coesão** quando tem um papel bem definido
- Os novos dados pertencem ao cliente
- Vamos atualizar a modelagem

Diagrama de classes atualizado



Exercício 04

- No arquivo model.py, crie a classe **Cliente**, abaixo do método **transferir()** da classe ContaBancaria
- Na **ContaBancaria**, altere o atributo **nome_cliente** para **cliente**

Teste da classe Cliente

- Vamos testar a nossa classe Cliente
- Instancie um cliente e uma conta
- **Atribua** o cliente à conta criada
- Imprima os dados do cliente e da conta

Referências

- LIVRO: Apress - Beginning Python From Novice to Professional
- LIVRO: O'Relly - Learning Python
- hqp://wiki.python.org.br/ListaDeExercicios
- hqps://docs.python.org/2/