Aula 02 – Classes e objetos

TECNOLOGIA em **Gestão de Dados**

Programação II

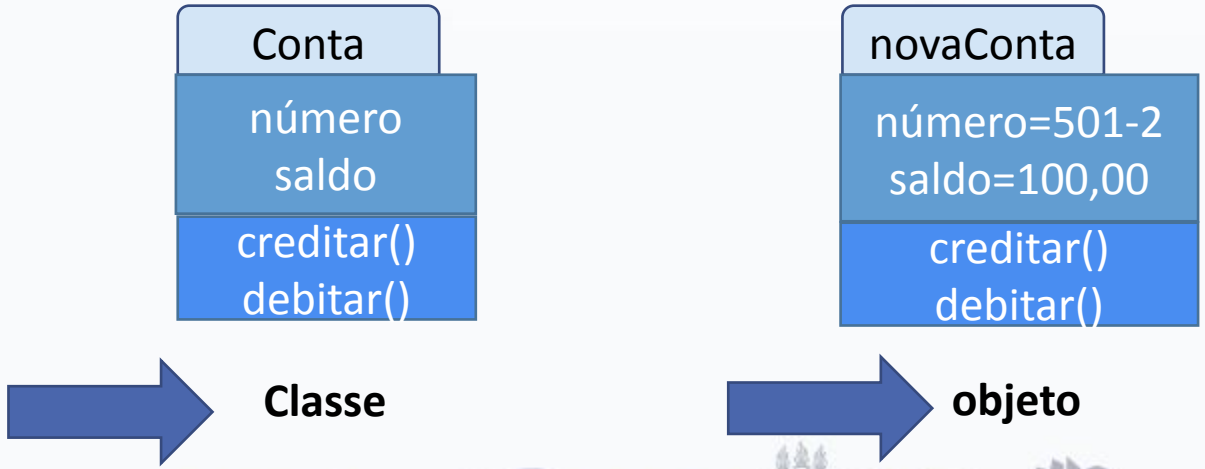
Prof. Dr. Wener Sampaio

ReUni DIGITAL · EAD Centro de Educação Aberta e a Distância · UFPI Universidade Federal do Piauí · MEC

Classes x objetos

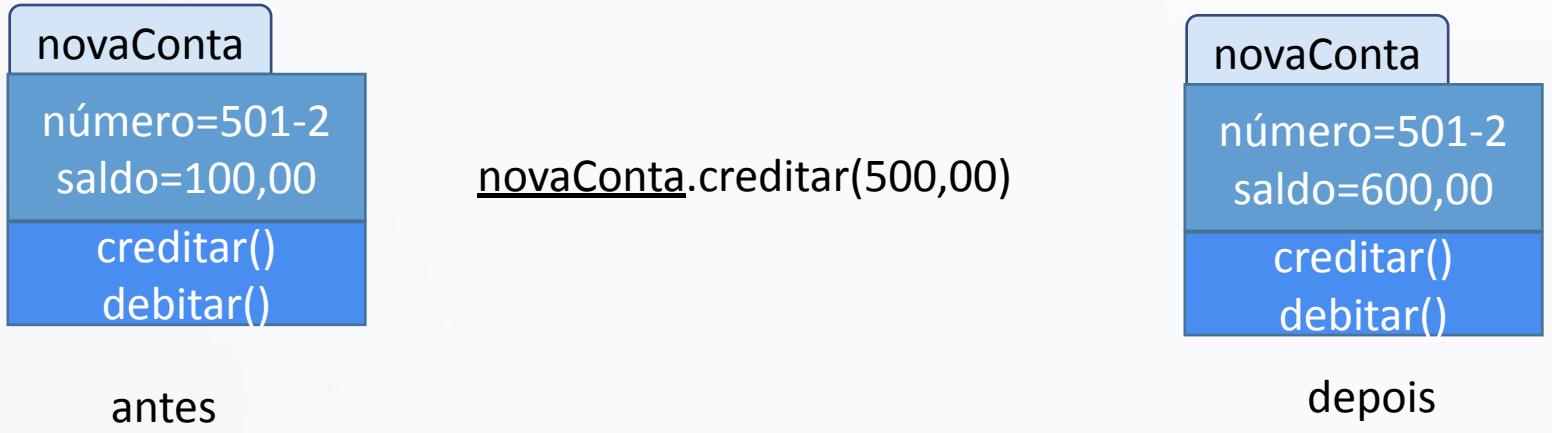
- Revisando:
  - Atributos e métodos fazem parte da mesma entidade.
  - Classes são definições (de atributos e métodos), mas não armazenam dados.
  - Objetos são instâncias (‘materialização’) de uma Classe, que podem armazenar e manipular dados.

novaConta = Conta(‘501-2’, 100,00)



Classes x objetos

- Revisando:
  - Classes criam os objetos.
  - Objetos armazenam dados.
  - Os métodos transformam os dados do próprio objeto. (encapsulamento)



Classes x objetos

- Definindo Classes:

```
def nomeDaClasse:
        corpoDaClasse
```

- Corpo da Classe:
  - Atributos
  - Métodos
    - Construtores (inicializadores)
    - Destruidores
    - Outras classes

Classes x objetos

- Exemplo 1:

moduloConta.py > ...

```python
class Conta:
    def __init__(self, numero, saldo=0):
        self.numero=numero
        self.saldo=saldo
    def setNumero(self, novoNumero):
        self.numero = novoNumero
    def creditar(self,valor):
        self.saldo+=valor
    def debitar(self,valor):
        self.saldo-=valor
    def getNumero(self):
        return self.numero
    def getSaldo(self):
        return self.saldo
    def transferir(self, valor, contaDst):
        self.debitar(valor)
        contaDst.creditar(valor)
    def __str__(self):
        txt=f'Número = {self.numero}\nSaldo = {self.saldo}\n'
        return txt
```

Classes x objetos

- Exemplo 1: Classe Conta.
  - Construtor __init__ cria uma instância da Classe Conta (opcional).
  - Quantidade de parâmetros variável.
  - Parâmetros podem ter valor pré-definidos (opcionais).
  - O primeiro parâmetro deve ser ‘self’. (mentira)
  - Parâmetros são variáveis locais (só existem apenas na função).
    - Mas cuidado com os apelidos (veremos adiante)
  - ‘self.atributo = ’ cria atributos do objeto (não locais, do objeto).
    - self.saldo é diferente saldo
  - Demais métodos manipulam os atributos.

Classes x objetos

- Exemplo 2: objeto da Classe Conta

```
3    def exemplo2():
4        novaConta = Conta('201-2',100.0)
5        print(novaConta)
6        novaConta.creditar(500.0)
7        novaConta.saldo-=300
8        print(novaConta)
```

Classes x objetos

- Exemplo 2:
  - Cria um objeto da Classe Conta.
  - self não precisa ser informado, pois referencia ao próprio objeto.
    - É como dar uma ordem a si mesmo.
    - Operador ‘.’ acessa atributos e métodos do objeto.
  - Deve-se:
    - Dar preferência a manipulação dos atributos através dos métodos. (linha 6)
    - Evitar manipular atributos diretamente. (linha 7)
    - Veremos futuramente formas de ‘proteger’ os atributos desta manipulação direta e de demais usos indevidos.
  - Por quê o método creditar não necessita da informação do número?

Comunicação entre objetos

- Objetos podem se comunicar com outros objetos por chamada de métodos (exemplo 3):

moduloConta.py > ...

```
19            return self.saldo
20        def transferir(self, valor, contaDst):
21            self.debitar(valor)
22            contaDst.creditar(valor)
```

```
def exemplo3():
    novaConta = Conta('201-2',100.0)
    contaBB = Conta('16.222-4',1000.0)
    contaBB.transferir(500,novaConta)
    print(novaConta)
    print(contaBB)
```


wener.sampaio@ufpi.edu.br

Construtores

- Criam objetos:
  - Inicializando atributos.
  - Recomenda-se não executar outros códigos complexos.
  - Objetos criados por atribuição.
- def __init__(self, <parâmetros>):
  - Método padrão para criar objetos.
  - <parâmetros> pode ser vazia.
  - Uma classe pode não ter __init__
  - Geralmente é o primeiro método
  - Um objeto pode ser não definido
    - contaCEF = None

Passagem de parâmetros

- Objetos são mutáveis:
  - Passagem por referência (apelido);
  - Lembre=se: “O que acontece com um, acontece com o outro”!
  - Temos no exemplo 3:
    - contaDst é um apelido para novaConta.
    - Alterações em contaDst também ocorrerão em novaConta.
    - Ou seja, elas estão no mesmo endereço de memória.
    - Cuidado:
      - Atribuição gera um novo dado, ou seja, cria um novo dado em endereço diferente, assim as alterações não se aplicarão no objeto passado por parâmetro.
      - Exemplo4()

Passagem de parâmetros

- Exemplo 4:

```
18    def exemplo4():
19        novaConta = Conta('201-2',100.0)
20        print(novaConta)
21        atribuicao(novaConta)
22        print(novaConta)
23
24    def atribuicao(conta):
25        print(conta)
26        conta = Conta('0001-x',0.0)
27        print(conta)
```

```
Número = 201-2
Saldo = 100.0

Número = 201-2
Saldo = 100.0

Número = 0001-x
Saldo = 0.0

Número = 201-2
Saldo = 100.0
```

Liberando memória

- Remoção de objetos:
  - Objetos criados ocupam um endereço e uma quantidade de memória.
  - Isso se torna um problema para grandes volumes de dados.

- É possível remover:
  - Função del: remoção manual
    - del(objeto)
  - Garbage collection: remoção automática / manual
    - Import gc
    - gc.collect()

Atributos estáticos

- São atributos da Classe:
  - É um único atributo para a Classe inteira.
  - Compartilhada com todos os objetos desta Classe.
  - São criadas no corpo da função, sem o self.
  - São acessadas usando o nome da Classe.
    - ‘Classe.atributo’
  - Não são excluídas com o fim do método ou objeto que as usou.

Atributos estáticos

```
23  class ContaAutomatica:
24      proximaConta=1
25      def __init__(self, saldo=0):
26          self.numero=ContaAutomatica.proximaConta
27          ContaAutomatica.proximaConta+=1
28          self.saldo=saldo
29      def setNumero(self, novoNumero):
30          self.numero = novoNumero
31      def creditar(self,valor):
32          self.saldo+=valor
33      def debitar(self,valor):
34          self.saldo-=valor
35      def getNumero(self):
36          return self.numero
37      def getSaldo(self):
38          return self.saldo
39      def transferir(self, valor, contaDst):
40          self.debitar(valor)
41          contaDst.creditar(valor)
42      def __str__(self):
43          return f'Número = {self.numero}\nSaldo = {self.saldo}\n'
```

```
def exemplo5():
    cAutA = ContaAutomatica()
    cAutB = ContaAutomatica(100.0)
    cAutC = ContaAutomatica(1000.0)
    print(cAutA)
    print(cAutB)
    print(cAutC)
```

```
Número = 1
Saldo = 0

Número = 2
Saldo = 100.0

Número = 3
Saldo = 1000.0
```

Métodos estáticos

- São semelhantes aos atributos estáticos
- Definidos com o decorador: @staticmethod:
  - Usados diretamente na classe
    - Classe.método()
  - Não necessitam de um objeto para existir
  - Ou seja, não necessitam do parâmetro ‘self’
    - Isso implica que métodos estáticos **só acessam atributos e métodos também estáticos.**
    - Útil quando se quer realizar uma ação que não necessita de um objeto específico.
    - Evita-se criar um objeto específico cujos atributos não terão a menor importância na chamada do método.

Métodos estáticos

Classe ContaAutomatica:

moduloConta.py ● moduloPrincipal.py

moduloConta.py > ...

```
40          contaDst.creditar(valor)
41      def __str__(self):
42          return f'Número = {self.numero}\nSaldo = {self.saldo}\n'
43      @staticmethod
44      def validaSaldo(valor):
45          return valor >= 0
```

ModuloPrincipal

```
37  def exemplo6():
38      valor=-1000.0
39      if ContaAutomatica.validaSaldo(valor):
40          c_a = ContaAutomatica(valor)
41      else:
42          print("Erro ao criar a conta.")
```

Métodos da Classe

- São semelhantes aos métodos estáticos
- Definidos com o decorador: @classmethod:
  - São acessados pela Classe
    - Classe.metodo()
  - Porém, também são acessados pelos objetos
  - A definição usa ‘cls’ -> referencia a Classe
  - Atenção
    - Métodos do objeto usam self
    - Métodos da classe usam cls

Métodos da Classe

ModuloPrincipal

```
44  def exemplo7():
45      if ContaAutomatica.validaNumero():
46          c_a = ContaAutomatica(100.0)
47          print('Conta criada.')
48      else:
49          print("Erro! Número de contas excedido.")
50
51      c_a = ContaAutomatica(100.0)
52      if c_a.validaNumero():
53          print('Agência ainda disponível.')
54      else:
55          print("Erro! Número de contas excedido.")
```

Na classe

No objeto

Classe ContaAutomatica:

```
46      @classmethod
47      def validaNumero(cls):
48          return  cls.proximaConta<100
```