Python

Orientação a Objetos

Prof. Dr. Dieval Guizelini

Analista e Desenvolvedor de Sistemas

Mestre em Bioinformática e Doutor em Ciências-Bioquímica

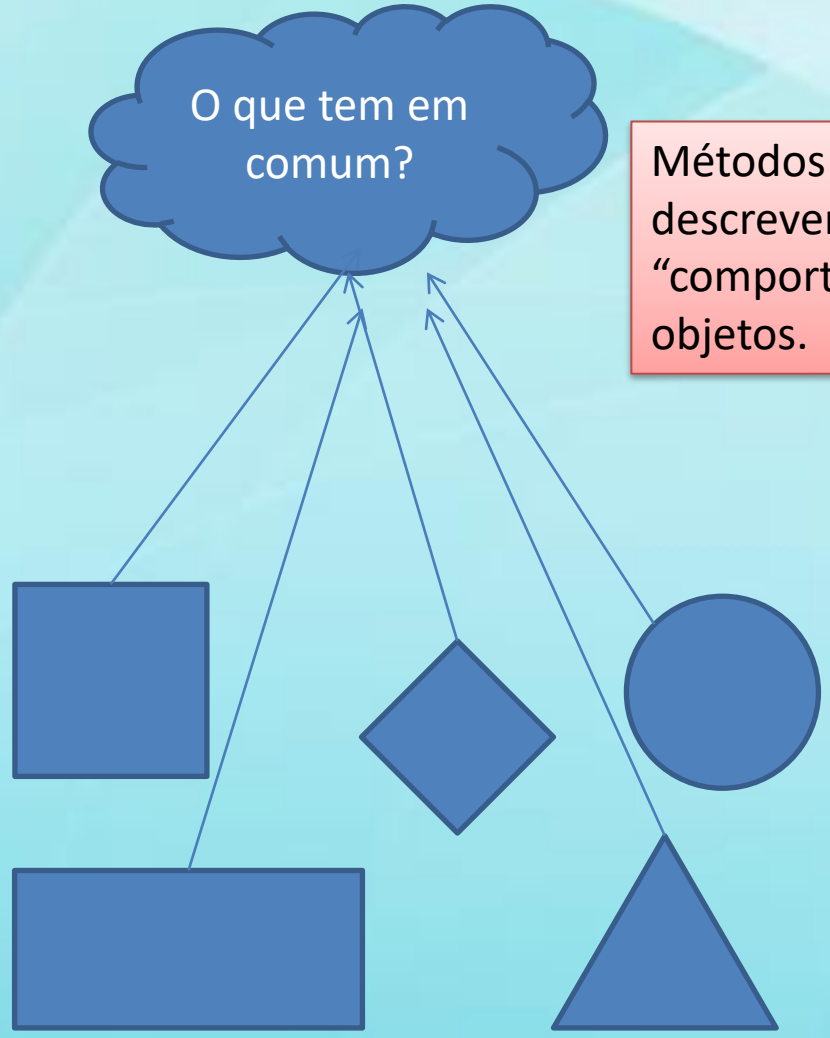
dieval at ufpr.br / dievalg at gmail.com

Paradigma Orientado a Objetos

Estruturado / Imperativo	Orientado a Objetos
• Foco: código • Descrição do procedimento para resolver um problema • O escopo dos dados se dividem em global/local • As “funções” são “invocadas”. • Os programas possuem áreas de código e de dados distintos	• Foco: nos dados • Como a informação é manipulada para obter a solução • Os objetos são criados e destruídos continuamente • Os objetos trocam mensagens entre si (através dos métodos) • Os dados possuem referências aos códigos.

A ordem de execução das instruções são determinadas pelas instruções, pelos blocos de comandos, seja eles “encapsulados” por funções ou métodos. Porém a concepção, a forma de modelar o problema, o ponto de partida da solução é completamente diferente.

Represente o “mundo” da solução em classes e objetos...



Métodos descrevem/realizam o “comportamento” dos objetos.

Classe: Figura	
atributos	N.Lados TamLados[]
métodos	calcPerimetro() calcArea()

Objetos são instâncias de classes...

Obj: Quadrado
N.lados: 4 TamLados = {2}

Obj: Retângulo
N.lados: 4 TamLados = {4,2}

Obj: Triangulo
N.lados: 3 TamLados = {1,2,3}

Em Python...

Descrevendo uma classe

```
class NomeClasse :
        pass
```

Instanciando uma classe

```
NomeClasse()

x = NomeClasse()
```

A principio, uma classe serve apenas para descrever um objeto, ou seja, ela não tem “atividade” efetiva durante a execução do programa. A execução do programa ocorre através da instanciação dos objetos e da comunicação entre eles.

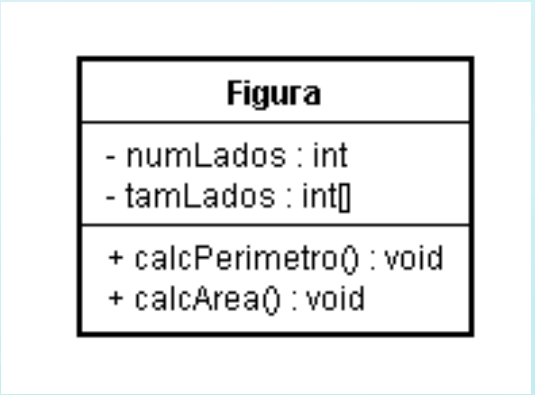
```
>>> dir(NomeClasse)
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__',
'__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__init_subclass__', '
__le__', '__lt__', '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '
__repr__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__']
>>>
```

Instâncias e igualdade...

- x = NomeClasse()
- y = NomeClasse()
- x == y
  false
- id(x) == id(y) # id retorna a “identidade” do objeto
  False

Voltando ao nosso modelo...

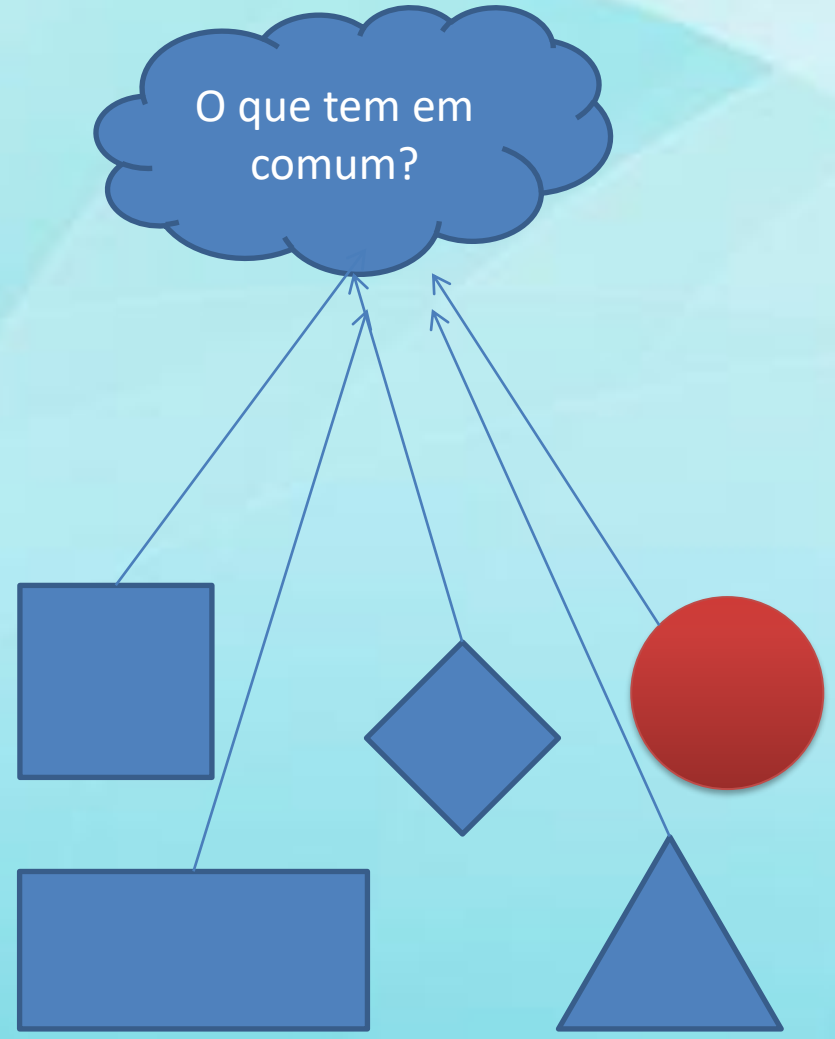
```
class Figura:
        numLados=0
        tamLados=tuple()
        def calcPerimetro(self):
                if self.numLados>0:
                        soma = 0
                        for i in range(0,self.numLados):
                                soma = soma + self.tamLados[i]
                        return soma
                return 0
```



No IDLE

```
x = Figura()
x.numLados = 4
x.tamLados = (2,3,2,3)
x.calcPerimetro()
```

Atende todas as situações?



Classe: Figura	
atributos	N.Lados
	TamLados[]
métodos	calcPerimetro()
	calcArea()

Número de lados?
Perímetro?

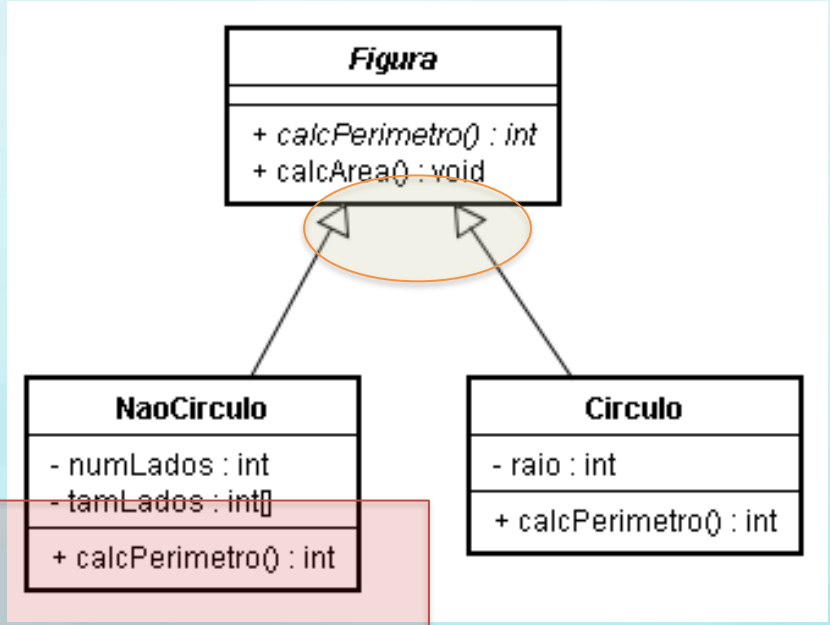
Uma possível solução

```
class Figura:
        def calcPerimetro(self):
                pass

class NaoCirculo(Figura):
        numLados=0
        tamLados=tuple()
        def calcPerimetro(self):
                if self.numLados>0:
                        soma = 0
                        for i in range(0,self.numLados):
                                soma = soma + self.tamLados[i]
                        return soma
                return 0

class Circulo(Figura):
        raio=0
        def calcPerimetro(self):
                return self.raio * math.pi * 2
```

Sobreposição

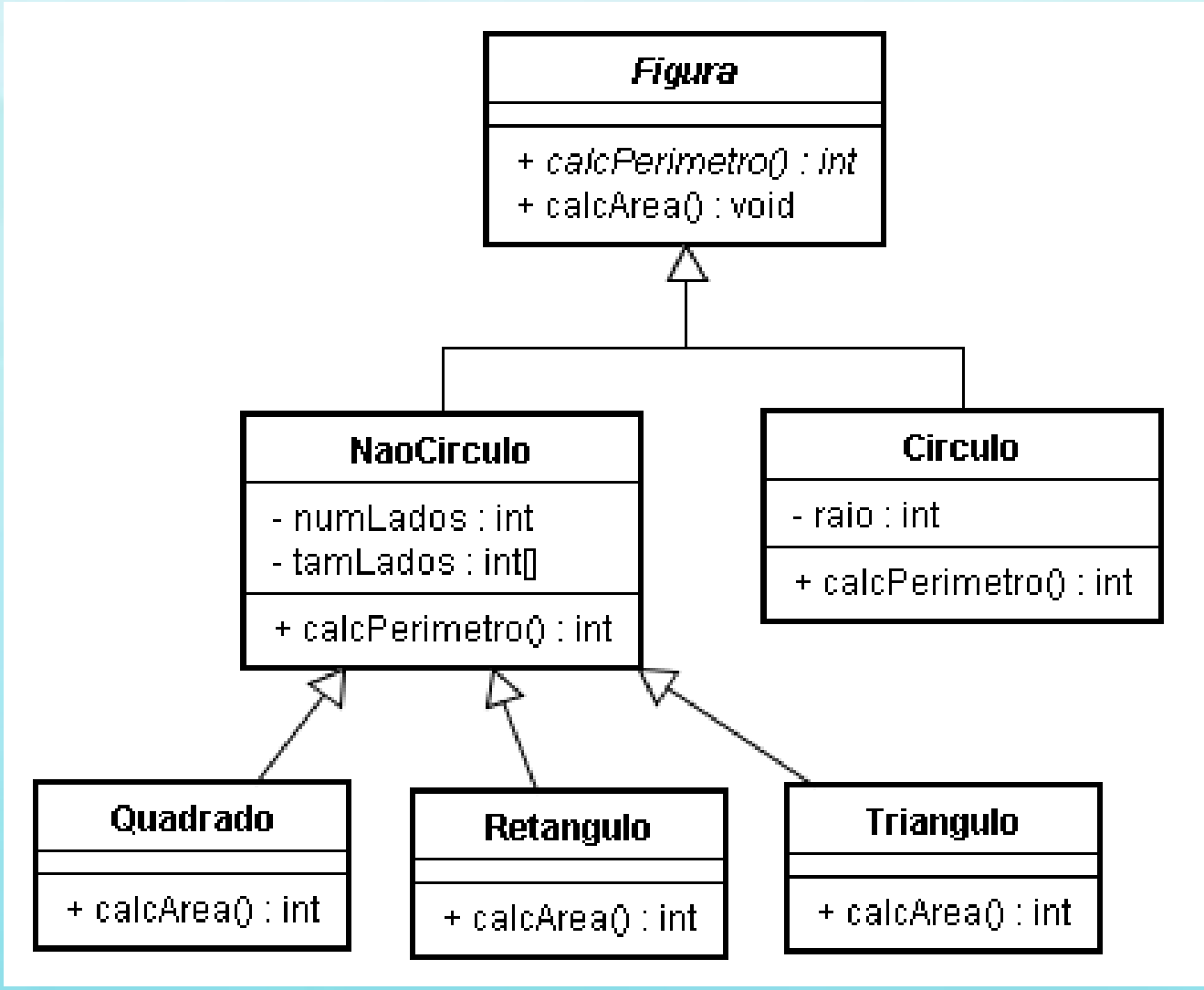


HERANÇA

Uma classe herda todos os elementos da classe ancestral.

Exercício 1

- Escrevam em Python a implementação do modelo ao lado.



Métodos especiais

- Construtor
  _ _init_ _(self,param...)

- Sobrecarga de operadores
  object.__lt__(*self*, *other*)
  object.__le__(*self*, *other*)
  object.__eq__(*self*, *other*)
  object.__ne__(*self*, *other*)
  object.__gt__(*self*, *other*)
  object.__ge__(*self*, *other*)

- Método hash
  object.__hash__(*self*)

https://docs.python.org/3/reference/datamodel.html?highlight=class#object.__init__

__doc__

```
>>> class Fausto:
      """Fausto é um romance de Goethe
que Beethoven transformou em Ópera."""
      def review(self):
            """
            Este método responde com a avaliação dos críticos
            """
            print 'Um romance excepcional'


>>> print Fausto.__doc__
Fausto é um romance de Goethe
que Beethoven transformou em Ópera.
```

```
>>> print Fausto().__doc__
Fausto é um romance de Goethe
que Beethoven transformou em Ópera.


>>> print Fausto().review.__doc__
Este método responde com a avaliação dos críticos
```

Recurso da Classe ou do Objeto?

```
class Dog:

    kind = 'canine'         # class variable shared by all instances

    def __init__(self, name):
        self.name = name    # instance variable unique to each instance

>>> d = Dog('Fido')
>>> e = Dog('Buddy')
>>> d.kind                  # shared by all dogs
'canine'
>>> e.kind                  # shared by all dogs
'canine'
>>> d.name                  # unique to d
'Fido'
>>> e.name                  # unique to e
'Buddy'
```

https://docs.python.org/2/tutorial/classes.html#class-objects

isinstance

- Dado:
  - x = Figura()
  - c = Circulo()
- O que acontece com:
  - isinstance(x,Figura)
  - isinstance(x,Circulo)
  - isinstance(c,Circulo)
  - isinstance(c,Figura)

Exercício

```
from datetime import datetime

class Bico:

    def __init__(self, data, hora, posicao, n2o, co2, ch4, h2o, nh3):
        self.data = datetime.strptime(data,'%Y-%m-%d')
        self.hora = datetime.strptime(hora,'%H:%M:%S.%f')
        # https://www.tutorialspoint.com/python/time_strptime.htm
        self.posicao = posicao
        self.n2o = n2o
        self.co2 = co2
        self.ch4 = ch4
        self.h2o = h2o
        self.nh3 = nh3

    def __str__(self):
        return
'{self.data}\t{self.hora}\t{self.posicao}\t{self.n2o}\t{self.co2}\t{self.ch4}\t{self.h2o}
\t{self.nh3}'.format(self=self)
```

```
class ColecaoBico:
    dados = []

    # https://docs.python.org/2/tutorial/errors.html
    def load(self,arq):
      try:
          f = open(arq)
          buf = f.read()
          linhas = buf.split('\n')
          self.dados = []
          for i in range(0, len(linhas)):
              cols = linhas[i].split(' ')
              if len(cols) == 8:
                  self.dados.append(Bico(*cols))
              else:
                  print('Warn: cols nao tem 8 colunas cols={} lin={} - {} '.format(len(cols),i,linhas[i]) )
      finally:
          f.close()


dados = ColecaoBico()
dados.load('C:\\bioinfo\\python\\2018-07-16.csv')
```

Referências

- Funções de IO
  https://docs.python.org/3/tutorial/inputoutput.html
  https://docs.python.org/3/library/io.html