ALGORITMOS E PROGRAMAÇÃO DE COMPUTADORES II

Programação Orientada a Objetos II

Sobrecarga de operadores

Operadores que são definidos diferentemente para múltiplas classes

Exemplo: operador +

```
>>> 2 + 4
6
>>> [2, 3, 4] + [5]
[2, 3, 4, 5]
>>> 'abc' + 'de'
'abcde'
```

Sobrecarga de operadores

Como a linguagem Python é orientada a objetos, os operadores aritméticos são, na essência, chamados a métodos definidos na respectiva classe do primeiro operando

Assim, x+y é o equivalente a x.__add__(y), sendo que o método __add__() é definido na classe de x

Sobrecarga de operadores

```
class Point():
   def __init__(self, x=0, y=0):
      self.x = x
      self.y = y

   def __repr__(self):
      return '({},{})'.format(self.x, self.y)

   def __add__(self, other):
      if type(other) == Point:
         return Point(self.x + other.x, self.y +
other.y)
      else:
         return Point(self.x + other, self.y + other)
```

Sobrecarga de operadores

Outros operadores

Operator	Method	Number	List and String
`x + y`	`x.__add__(y)`	Addition	Concatenation
`x - y`	`x.__sub__(y)`	Subtraction	—
`x * y`	`x.__mul__(y)`	Multiplication	Self-concatenation
`x / y`	`x.__truediv__(y)`	Division	—
`x // y`	`x.__floordiv__(y)`	Integer division	—
`x % y`	`x.__mod__(y)`	Modulus	—
`x == y`	`x.__eq__(y)`	Equal to	
`x != y`	`x.__ne__(y)`	Unequal to	
`x > y`	`x.__gt__(y)`	Greater than	
`x >= y`	`x.__ge__(y)`	Greater than or equal to	
`x < y`	`x.__lt__(y)`	Less than	
`x <= y`	`x.__le__(y)`	Less than or equal to	
`repr(x)`	`x.__repr__()`	Canonical string representation	
`str(x)`	`x.__str__()`	Informal string representation	
`len(x)`	`x.__len__()`	—	Collection size
`<type>(x)`	`<type>.__init__(x)`	Constructor	

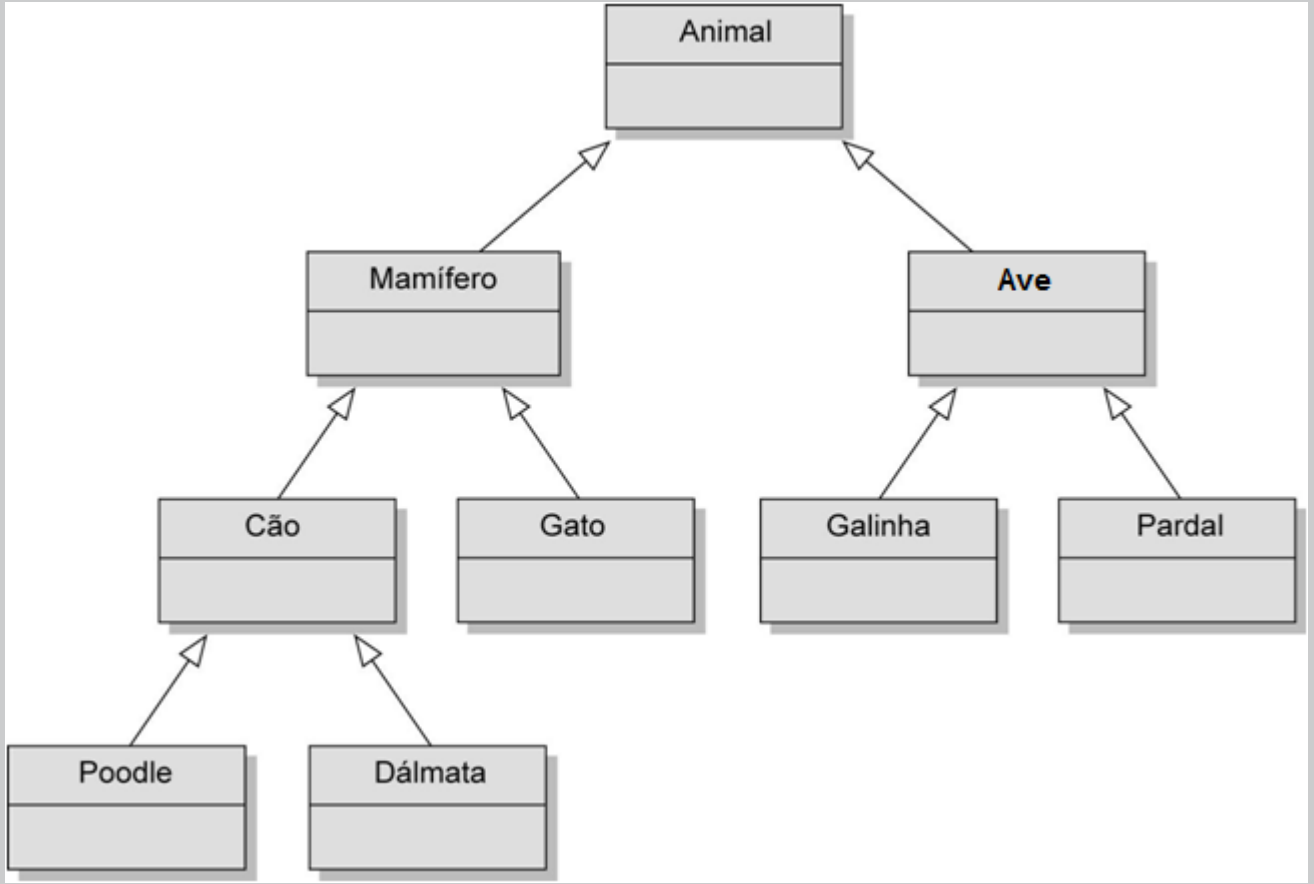
Fonte: Perkovic, 2015

Herança

Técnica fundamental em POO, usada para criar e organizar classes reutilizáveis.

Serve para reutilizar ou alterar os métodos de classes já existentes, bem como adicionar novos atributos e métodos a fim de adaptar as classes a novas situações.

Herança


Animal
Mamífero
Ave
Cão
Gato
Galinha
Pardal
Poodle
Dálmata

Herança

Exemplo: suponha que queremos definir uma classe Lista, que contenha um método adicional: choice(), que irá retornar aleatoriamente um elemento existente na lista.

Herança

Poderíamos definir uma classe MyList contendo todos os métodos necessários:

```
import random
class MyList:
    def __init__(self, initial = []):
        self.lst = initial
    def __len__(self):
        return len(self.lst)
    def append(self, item):
        self.lst.append(self, item)
```

Seria necessário, entretanto, redefinir mais de 30 métodos da classe list, a fim de usá-la da mesma forma que uma lista.

Herança

Ao invés, podemos fazer uma "extensão" da classe list, por meio de herança:

```
import random
class MyList(list):
    def choice(self):
        return random.choice(self)
```

Dizemos que MyList é uma subclasse da classe list, e list é uma superclasse de MyList.

Exercício

Considere uma entidade Funcionário, que possui nome, data de admissão e salário. Implemente sua classe, definindo também alguns métodos para manipulação dos atributos.

Em seguida, considere a entidade Gerente, que também é um funcionário. Além dos atributos de funcionário, um gerente também contém um bônus, que é uma porcentagem adicional aplicada no seu salário.

Implemente a classe Gerente como uma extensão de Funcionário.

ALGORITMOS E PROGRAMAÇÃO DE COMPUTADORES II

Programação Orientada a Objetos II