
BY

IME-USP

Tec Prog 1

Orientação a Objetos

Fabio Kon
IME-USP

Principal característica de um bom Programador OO

- Escolha de bons nomes para
  - variáveis
  - atributos/propriedades
  - métodos e seus argumentos
  - classes

- *Intention-revealing names*
- Classes geralmente são substantivos
- Métodos geralmente são verbos

Característica importante do projeto de Classes

- Em um sistema orientado a objetos, as classes devem possuir
  - uma única responsabildade.
  - *Single Responsibility Principle*
- Ou seja, uma classe não deve tratar de assuntos diferentes.
- Se uma classe trata de 2 ou 3 assuntos, veja se não dá para quebrá-la em 2 ou 3 classes menores.

Herança

- Subclasses em Simula-67
  (a primeira linguagem OO da história)
- **Relação “é-um”**
- Bom exemplos de Herança
  - Pessoas numa Universidade
  - Animal
  - Arquivos em um Sistema de Arquivos
- Exemplos de mau-uso de Herança
  - Carro e suas partes

Herança terminologia

- Superclasse `Pessoa`
- Subclasses `Aluno`, `Professor`, `Funcionário`

- O ato de descer da superclasse para as subclasses é chamado de **Especialização**

- O ato de subir das subclasses para a superclasse é chamado de **Generalização**

Quando usar Herança?

1 Para organizar as abstrações

2 Para acrescentar comportamento novo

- Novos métodos
- Novos atributos

3 Para alterar comportamento

- Novas implementações de métodos mais especializados

Exemplo de Herança

```
class Poligono:
    def __init__(self, numero_de_lados):
        self.n = numero_de_lados
        self.lados = [0 for i in range(numero_de_lados)]

    def le_lados(self):
        self.lados = [float(input("Digite tamanho do lado " + str(i+1) + ": "))
                      for i in range(self.n)]

    def mostra_lados(self):
        for i in range(self.n):
            print("Lado", i+1, "is", self.lados[i])



class Triangulo(Poligono):
    def __init__(self):
        Poligono.__init__(self,3)

    def area(self):
        a, b, c = self.lados
        # calcula o semi-perímetro
        s = (a + b + c) / 2
        area = (s*(s-a)*(s-b)*(s-c)) ** 0.5
        print('A área do triângulo é %0.2f' %area)
```

Polimorfismo

- Um objeto que adquire várias formas
- É o que torna a OO tão poderosa
- Exemplos:
  - Pássaros
  - Visualizador de Arquivos
  - Figuras Geométricas
  - Cálculo de pagamento

```python
class Empregado:
    def __init__(self, nome, CPF, RG):
        self.nome = nome
        self.CPF = CPF
        self.RG = RG


class EmpregadoHorista(Empregado):
    def __init__(self, nome, CPF, RG, horasTrabalhadas, pagamentoPorHora):
        Empregado.__init__(self, nome, CPF, RG)
        self.horasTrabalhadas = horasTrabalhadas
        self.pagamentoPorHora = pagamentoPorHora

    def pagamento(self):
        return self.horasTrabalhadas * self.pagamentoPorHora

class EmpregadoCLT(Empregado):
    def __init__(self, nome, CPF, RG, salario):
        Empregado.__init__(self, nome, CPF, RG)
        self.salario = salario

    def pagamento(self):
        return 13.3 * self.salario


class PrestadorDeServico(Empregado):
    def __init__(self, nome, CPF, RG, pagamentoAvulso):
        Empregado.__init__(self, nome, CPF, RG)
        self.pagamentoAvulso = pagamentoAvulso

    def pagamento(self):
        return self.pagamentoAvulso
```

Polimorfismo

- Graças ao polimorfismo, um código bastante enxuto pode ser muito poderoso.
- Sem OO, precisamos de vários comandos **if** ou **switch** para ter o mesmo efeito
  - Imagine como seria o código abaixo sem OO:

```
for e in empregados:
    custoTotal += e.pagamento()
```

Conceitos Fundamentais de OO

- Objetos
- Classes
- Abstração
- variáveis de instância vs. variáveis de classe
- Troca de mensagens/chamada de método
- Nomes que revelam intenção
- Princípio da Responsabilidade Única
- Herança
- Polimorfismo

IME-USP


BY