02

Reaproveitando código através de Herança

DOWNLOAD

Segue o link (https://s3.amazonaws.com/caelum-online-public/python/10-python.zip) com os arquivos desta aula.

Introdução

Aprendemos conceitos de orientação a objetos, inclusive aprendemos a criar classes e entendemos como o Python lida com encapsulamento de atributos. Neste capítulo avançaremos ainda mais nos conceitos de orientação a objetos.

Vamos revisitar nossa classe Perfil:

```
# -*- coding: UTF-8 -*-
class Perfil(object):
    'Classe padrão para perfis de usuários'

    def __init__(self, nome, telefone, empresa):
        self.nome = nome
        self.telefone = telefone
        self.empresa = empresa
        self.__curtidas = 0

    def imprimir(self):
        print "Nome : %s, Telefone: %s, Empresa %s" % (self.nome, self.telefone, self.empresa)

    def curtir(self):
        self.__curtidas+=1

    def obter_curtidas(self):
        return self.__curtidas
```

Beleza, inclusive vamos realizar algumas curtidas:

```
>>> perfil = Perfil('Flávio Almeida', 'não informado', 'Caelum')
>>> perfil.curtir()
>>> perfil.curtir()
>>> perfil.obter_curtidas()
2
```

Excelente, mas agora pintou um novo requisito: perfis VIP's possuem um benefício especial que perfis padrões não têm: para cada curtida, eles ganham R$ 10,00 em crédito em compras virtuais. Já temos a classe Perfil criada, vamos implementar essa lógica através do método `obter_creditos` :

```
# -*- coding: UTF-8 -*-
class Perfil(object):
    'Classe padrão para perfis de usuários'

    def __init__(self, nome, telefone, empresa):
```

```
        self.nome = nome
        self.telefone = telefone
        self.empresa = empresa
        self.__curtidas = 0

    def imprimir(self):
        print "Nome : %s, Telefone: %s, Empresa %s" % (self.nome, self.telefone, self.empresa)

    def curtir(self):
        self.__curtidas+=1

    def obter_curtidas(self):
        return self.__curtidas

    def obter_creditos(self):
        return self.__curtidas * 10.0
```

Agora, vamos testá-lo:

```
>>> perfil = Perfil('Flávio Almeida', 'não informado', 'Caelum')
>>> perfil.curtir()
>>> perfil.curtir()
>>> perfi.obter_creditos()
20.0
```

Porém, qual tipo de perfil criamos? É padrão, é VIP? Precisamos arrumar uma maneira de identificarmos que tipo de perfil estamos criando, e garantirmos que o método `obter_creditos` só poderá ser chamado para perfis VIP. Que tal adicionarmos um atributo privado chamado `tipo` e convencionarmos que 0 e 1 correspondem aos tipos padrão e VIP respectivamente?

Precisamos alterar o construtor da nossa classe para receber o tipo. Inclusive, criaremos um método que retornará seu tipo:

```
# -*- coding: UTF-8 -*-
class Perfil(object):
    'Classe padrão para perfis de usuários'

    def __init__(self, nome, telefone, empresa, tipo):
        self.nome = nome
        self.telefone = telefone
        self.empresa = empresa
        self.__curtidas = 0
        self.__tipo = tipo

    def imprimir(self):
        print "Nome : %s, Telefone: %s, Empresa %s" % (self.nome, self.telefone, self.empresa)

    def curtir(self):
        self.__curtidas+=1

    def obter_curtidas(self):
        return self.__curtidas

    def obter_creditos(self):
        return self.__curtidas * 10.0
```

```
    def obter_tipo(self):
        return self.__tipo
```

Agora, vamos testar criando dois tipos:

```
>>> padrao = Perfil('Nico Steppat', 'não informado', 'Caelum', 0)
>>> vip = Perfil('Flávio Almeida', 'não informado', 'Caelum', 1)
>>> padrao.obter_tipo()
0
>>> vip.obter_tipo()
1
>>> vip.curtir()
>>> vip.obter_creditos()
10.0
>>> padrao.curtir()
>>> padrao.obter_creditos()
10.0
```

Criamos dois tipos de perfis, porém conseguimos chamar o método `obter_credito` no perfil padrão, o que fere nossa regra. Podemos resolver isso realizando um `if` dentro de nosso método `obter_creditos` :

```
def obter_creditos(self):
    if self.__tipo == 1:
        return self.__curtidas * 10.0
```

```
>>> padrao = Perfil('Nico Steppat', 'não informado', 'Caelum', 0)
>>> padrao.curtir()
>>> padrao.obter_creditos()
```

Muito bem, quando chamamos o método `obter_creditos` para um perfil padrão, nada acontece. Resolvemos nosso problema? Sim, mas independente do tipo de perfil, teremos sempre o método `obter_creditos` disponível. Se o método está disponível, nada impede do programador desavisado chamá-lo através de um perfil padrão. Com certeza ele gastará um bom tempo para entender que ele chamou um método que não faz nada!

Será que temos uma maneira de disponibilizarmos este método apenas para perfis VIP? Bem, aprendemos a criar especificações, tanto isso é verdade que criamos a especificação de um Perfil. Será que não podemos fazer a mesma coisa, criando uma classe chamada Perfil_Vip? Ela tem tudo o que a classe Perfil tem, a única novidade é que ela tem apenas o método 'obter_creditos'.

Primeiro vamos remover o atributo tipo da classe Perfil, ajustar seu construtor e inclusive remover o método 'obter_creditos':

```
# -*- coding: UTF-8 -*-
class Perfil(object):
    'Classe padrão para perfis de usuários'

    def __init__(self, nome, telefone, empresa):
        self.nome = nome
        self.telefone = telefone
        self.empresa = empresa
        self.__curtidas = 0
```

```
    def imprimir(self):
        print "Nome : %s, Telefone: %s, Empresa %s" % (self.nome, self.telefone, self.empresa)

    def curtir(self):
        self.__curtidas+=1

    def obter_curtidas(self):
        return self.__curtidas
```

Agora, criaremos a classe Perfil_Vip no mesmo arquivo da nossa classe Perfil. Ela é um Perfil, sendo assim, podemos dar CONTROL + C e CONTROL + V na nova classe.

```
# -*- coding: UTF-8 -*-
class Perfil_Vip(object):
    'Classe padrão para perfis de usuários VIPs'

    def __init__(self, nome, telefone, empresa):
        self.nome = nome
        self.telefone = telefone
        self.empresa = empresa
        self.__curtidas = 0

    def imprimir(self):
        print "Nome : %s, Telefone: %s, Empresa %s" % (self.nome, self.telefone, self.empresa)

    def curtir(self):
        self.__curtidas+=1

    def obter_curtidas(self):
        return self.__curtidas

    def obter_creditos(self):
        return self.__curtidas * 10.0
```

Agora, vamos importar as duas classes em nosso console instanciando um perfil padrão e um perfil VIP:

```
>>> from models import *
>>> vip = Perfil_Vip('Flávio Almeida', 'não informado', 'Caelum')
>>> vip.curtir()
>>> vip.curtir()
>>> vip.obter_creditos()
20.0
```

Excelente, temos um instância da classe `Perfil_Vip` que acabamos de obter seus créditos. Agora, será que conseguimos fazer a mesma coisa com o Perfil padrão? Não, pois ele não possui o método `obter_creditos` .

Agora, deixa eu te perguntar: se aparecer um novo atributo em `Perfil` , faz sentido ele ser adicionado em `Perfil_Vip` ? E se o nome de um atributo mudar em `Perfil` , precisaremos alterá-lo em `Perfil_Vip` ? Se o método `imprimir` mudar, precisaremos alterar o método imprimir em `Perfil_Vip` ? Está evidente que nosso copia e cola nos trará problemas de manutenção, justamente por um Perfil_Vip também ser um Perfil.

Será que em Python existe alguma maneira de representarmos uma relação `é um`? Será que podemos fazer com que a classe `Perfil_Vip` herde os atributos e métodos da classe `Perfil`? Sim! E já usamos herança sem você perceber.

Lembra da discussão sobre declaração de classes *old style* vs *new style*? Você deve lembrar que no *new style* a classe recebe como parâmetro 'object'. Lá estamos indicando que a classe 'Perfil' é um objeto. Porém, herdamos de `object` para resolvermos a incongruência entre a função `type` e a propriedade **class** de instâncias.

Agora, em nossa classe `Perfil_Vip` podemos apagar todo aquele código duplicado e indicar que nossa classe herda de 'Perfil':

```
# -*- coding: UTF-8 -*-
class Perfil_Vip(Perfil):
   'Classe padrão para perfis de usuários VIPs'

   def obter_creditos(self):
      return self.__curtidas * 10.0
```

Não é possível, só essa quantidade de código? Será que nosso código funciona?

```
>>> vip = Perfil_Vip('Flávio Almeida', 'não informado', 'Caelum')
>>> vip.nome
'Flávio Almeida'
>>> vip.telefone
'não informado'
>> vip.empresa
'Caelum'
```

Incrível! Nem declaramos o construtor, muito menos os atributos e lá estão eles! Disponíveis em nossa classe `Perfil_Vip`. Agora, só nos resta testar o método `obter_creditos`:

```
>>> vip.curtir()
>>> vip.curtir()
>>> vip.obter_creditos()
AttributeError: 'Perfil_Vip' object has no attribute '_Perfil_Vip__curtidas'
```

Temos um erro! Isso acontece porque estamos referenciando no método `obter_creditos` o self para a partir dele acessarmos o atributo `curtidas`. Porém este self é do `Perfil_Vip` e seu self não possui essa propriedade. A propriedade está no self de `Perfil`. E agora? Como fazemos para acessar o self de uma classe pai a partir de sua classe filha? Para isso utilizamos a função `super`, que recebe como primeiro parâmetro o nome da classe filha e como segundo parâmetro seu self:

```
# -*- coding: UTF-8 -*-
class Perfil_Vip(Perfil):
   'Classe padrão para perfis de usuários VIPs'

   def obter_creditos(self):
      return super(Perfil_Vip, self).obter_curtidas() * 10.0
```

Usamos o método `obter_curtidas`, caso contrário, não seria possível acessar o atributo `__curtidas` da classe pai, devido ao prefixo `__` que muda o nome da variável escondendo-a. Agora, recarregando nossa classe e testando mais uma vez:

```
>>> vip.curtir()
>>> vip.curtir()
>>> vip.obter_creditos()
20.0
```

Excelente, agora nosso código funciona como esperado. Pode parecer um pouco redundante passarmos o nome da nossa classe, inclusive seu self, não? Mas esse é o jeito Python 2.7! Será que ainda há tempo para adicionarmos mais uma mudança em nosso código? Um perfil do tipo VIP além dos atributos nome, telefone e empresa, também possui um apelido. Algo que o perfil padrão não tem. Queremos construir um perfil VIP passando para seu construtor este parâmetro:

```
# -*- coding: UTF-8 -*-
class Perfil_Vip(Perfil):
   'Classe padrão para perfis de usuários VIPs'

   def __init__(self, nome, telefone, empresa, apelido):

      self.apelido = apelido

   def obter_creditos(self):
      return super(Perfil_Vip, self).obter_curtidas() * 10.0
```

E agora? Bem, a classe Perfil já tem seu próprio construtor que recebe nome, telefone e empresa. Que tal chamarmos esse construtor da classe pai?

```
# -*- coding: UTF-8 -*-
class Perfil_Vip(Perfil):
   'Classe padrão para perfis de usuários VIPs'

   def __init__(self, nome, telefone, empresa, apelido):

      super(Perfil_Vip, self).__init__(nome, telefone, empresa)
      self.apelido = apelido

   def obter_creditos(self):
      return super(Perfil_Vip, self).obter_curtidas() * 10.0
```

Agora, pela milésima vez, vamos importar novamente nosso arquivo `models` e em seguida criar um perfil do tipo VIP:

```
>>> vip = Perfil_Vip('Flávio Almeida', 'não informado', 'Caelum', 'Almeida')
>>> vip.nome
'Flávio Almeida'
>>> vip.apelido
'Almeida'
>>> vip.curtir()
>>> vip.curtir()
>>> vip.obter_creditos()
20.0
```

Excelente. Acabamos de aprender como o Python 2.X lida com herança. Aliás, a herança que você ganhará agora são os exercícios do capítulo.