Computação II - Python
Aula 3 - Trabalhando com classes e objetos - Herança

Carla A. D. M. Delgado

João C. P. da Silva

Dept. Ciência da Computação - UFRJ

Classes e Objetos na programação

Herança

- A orientação a objetos permite utilizar classes já definidas como base para a construção de outras classes.
- Dizemos que a nova classe *herda* da classe antiga propriedades e / ou comportamentos.
- Esse é um recurso para reutilizar código já desenvolvido.

Herança - Exemplo

Em nosso sistema de controle de contas bancárias, queremos agora oferecer a modalidade de conta especial, que permite aos correntistas sacar mais dinheiro do que a quantia disponível na conta, até um certo limite.

Classes e Objetos na programação

Conta bancária - Relembrando...

```
1     def __init__(self, correntistas, numero, saldo=0):
2         self.saldo =saldo
3         self.numero = numero
4         self.correntistas = correntistas
5         self.operacoes = [ ("DEPOSITO", saldo) ]
6
7     def resumo(self):
8         for c in self.correntistas:
9             print("%s , CPF: %10s \n" % (c.nome, c.CPF))
10        print("CC numero: %s Saldo: %10.2f" % (self.numero, self.saldo))
11
12    def saque(self, valor):
13        if self.saldo>=valor:
14            self.saldo -= valor
15            self.operacoes += [("SAQUE", valor)]
16        else:
17            print("Saldo insuficiente")
18
19    def deposito(self, valor):
20        self.saldo += valor
21        self.operacoes += [("DEPOSITO", valor)]
22
23    def extrato(self):
24        print("Extrato CC numero %s \n" % self.numero)
25        for op in self.operacoes:
26            print("%10s %10.2f\n" % (op[0], op[1]))
27        print("\n    Saldo: %10.2f\n" % self.saldo)
28
```

Herança - Exemplo

Em nosso sistema de controle de contas bancárias, queremos agora oferecer a modalidade de conta especial, que permite aos correntistas sacar mais dinheiro do que a quantia disponível na conta, até um certo limite.

- As operações de extrato, depósito e resumo continuam as mesmas de uma conta normal
- O limite da conta especial é instituido no momento da criação da conta, e possui valor default zero.

Herança - Exemplo

- A conta especial permite aos correntistas sacar mais dinheiro do que a quantia disponível na conta, até um certo limite.
- Usaremos o conceito de **herança** para construir a classe *conta especial* usando a classe *conta*, uma vez que:
  - A *conta especial* terá todos os atributos que *conta* já tem;
  - As operações de extrato, depósito e resumo continuam as mesmas de uma conta normal.
- O limite da conta especial será instituido no momento da criação da conta, e possui valor default zero.
- Precisamos de um novo construtor, para lidar com a nova propriedade *limite*, e de um novo método *saque*.

Herança - Exemplo

- Na definição da classe, ao lado do nome da classe colocamos dentro dos parântese o nome da classe de qual ela **herda** (caso haja).
- A classe *ContaEspecial* **herda** todos os atributos e métodos da classe *Conta*.
- Dizemos então que *ContaEspecial* é uma **subclasse** da classe *Conta*, e que *Conta* é uma **superclasse** de *ContaEspecial*.

```
1 class ContaEspecial(Conta):
2     def __init__(self, correntistas, numero, saldo=0, limite=0):
3         Conta.__init__(self, correntistas, numero, saldo)
4         self.limite = limite
5
6     def saque(self, valor):
7         if self.saldo + self.limite >= valor:
8             self.saldo -= valor
9             self.operacoes+= [("SAQUE", valor)]
```

Herança - Exemplo

- O método *construtor* da classe *ContaEspecial* inclui uma chamada para o construtor da classe *Conta*. Essa é uma boa prática para reutilizar a definição da superclasse, evitando ter que reescrever as atribuições das propriedades herdadas. Fazendo isso, caso algo seja mudado na superclasse, não teremos que mudar em cada subclasse individualmente.
- Dentro do *construtor* da subclasse, o construtor da superclasse é chamado antes de atribuirmos valor à propriedade *limite*, respeitando o processo de criação de super e subclasses.

```
1 class ContaEspecial(Conta):
2     def __init__(self, correntistas, numero, saldo=0, limite=0):
3         Conta.__init__(self, correntistas, numero, saldo)
4         self.limite = limite
```

Herança - Exemplo

- O método *saque* da *ContaEspecial* **não** chamou o método *saque* da classe *Conta*. Isso significa que estamos reescrevendo completamente esse método na subclasse. Esse é um recurso para sobrepor os métodos da superclasse na subclasse

```
1 class ContaEspecial(Conta):
2     def __init__(self, correntistas, numero, saldo=0, limite=0):
3         Conta.__init__(self, correntistas, numero, saldo)
4         self.limite = limite
5
6     def saque(self, valor):
7         if self.saldo + self.limite >= valor:
8             self.saldo -= valor
9             self.operacoes+= [("SAQUE", valor)]
```

Herança - Estudo Dirigido

Salve a classe *ContaEspecial* no mesmo arquivo que a classe *Conta*. Agora, faça um programa de testes que:

- Importe as classes Cliente, Conta e ContaEspecial;
- Crie dois clientes;
- Crie uma conta convencional para um destes clientes;
- Crie uma conta especial tendo ambos os clientes como correntistas;
- Faça operações de saque e depósito nestas contas de forma a exceder o saldo disponível no momento de algum saque, para ambas as contas;
- Peça o extrato de cada uma das contas e veja o que aconteceu.

Herança - Estudo Dirigido

- Modifique as classes *Conta* e *ContaEspecial* para que a operação de saque retorne verdadeiro caso o saque tenha sido efetuado, e falso caso contrário.
- Altere a classe *ContaEspecial* de forma que seu extrato exiba o limite e o total disponível para saque.
- Observe o método *saque* das classes *Conta* e *ContaEspecial*. Modifique o método na classe *Conta*, de forma que a verificação da possibilidade de saque seja feita por um novo método, substituindo a condição atual. Esse novo método retornará verdadeiro se o saque puder ser efetuado, e falso caso contrário. Modifique a classe *ContaEspecial* de forma a trabalhar com esse novo método. Verifique se você ainda precisa sobrescrever o método saque na subclasse.

Herança

- Ao utilizar *herança* aproveitamos funcionalidades que já tínhamos desenvolvido anteriormente, e podemos adicionar novos recursos.
- Uma subclasse aproveita tudo o que quiser da superclasse, tendo a opção de complementar os atributos e modificar apenas os comportamentos que forem diferentes.
- Uma boa maneira de trabalhar é criar classes de forma que o comportamento e as características comuns fiquem na superclasse. Isso permitirá a definição posterior de subclasses enxutas.
- O que for mudado na superclasse se refletirá nas subclasses.
- Não há obrigatoriedade em definir uma hierarquia de classes em seus programas, este é um recurso que está à seu serviço.

Referências

Para saber mais

1. Livro: **Introdução à Programação com Python - Algoritmos e lógica de programação para iniciantes**. Autor: Nilo Ney Coutinho Menezes

Computação II - Python
Aula 3 - Trabalhando com classes e objetos - Herança

Carla A. D. M. Delgado

João C. P. da Silva

Dept. Ciência da Computação - UFRJ