Universidade Federal do Piauí - UFPI

Sistemas de Informação

Programação Orientada a Objetos I - POO

Herança múltipla e Interfaces

Prof. Flávio Araújo - UFPI - Picos PI

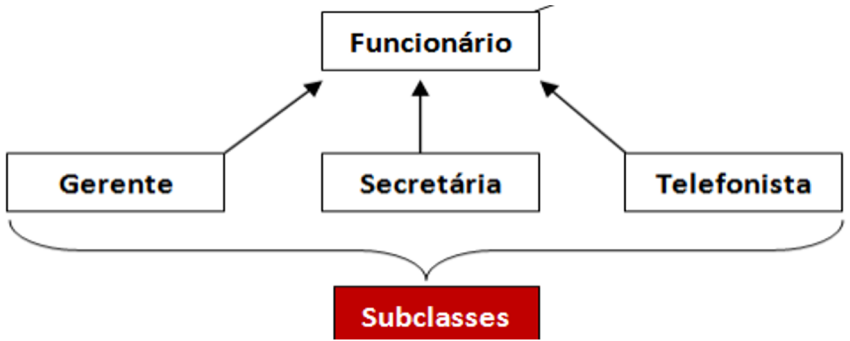
Problema de acesso

- Imagine que será necessário implementar um sistema em que os Gerentes e Diretores tenham acesso ao sistema, entretanto os outros funcionários não.

```
class Diretor(Funcionario):

    def autentica(self, senha):
        # verifica se a senah confere
```

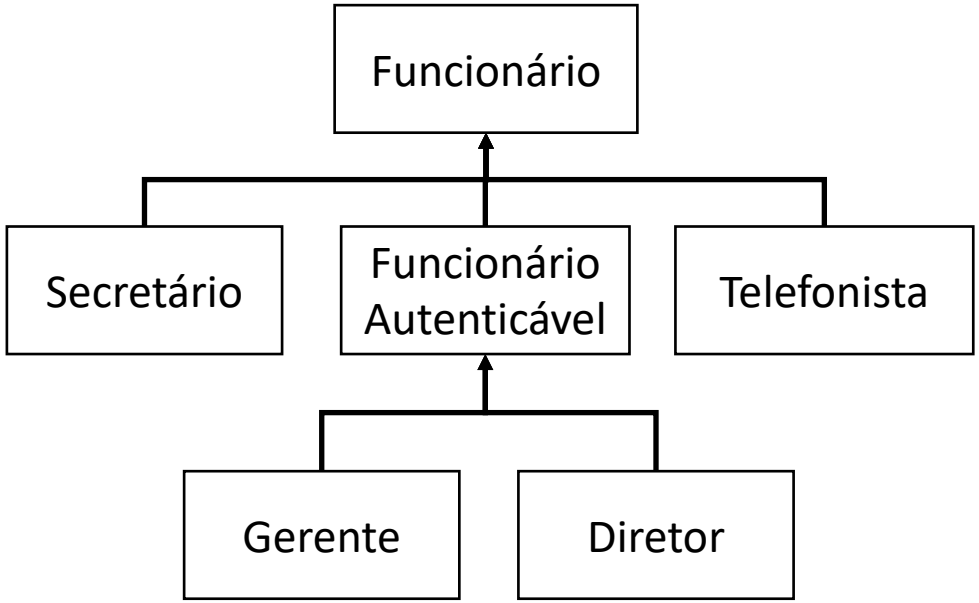
```
class Gerente(Funcionario):

    def autentica(self, senha):
        # verifica se a senha confere e também se o seu departamento tem acesso
```

```
class SistemaInterno:

    def login(self, funcionario):
        if(hasattr(obj, 'autentica')):
            # chama método autentica
        else:
            # imprime mensagem de ação inválida
```



Problema de acesso

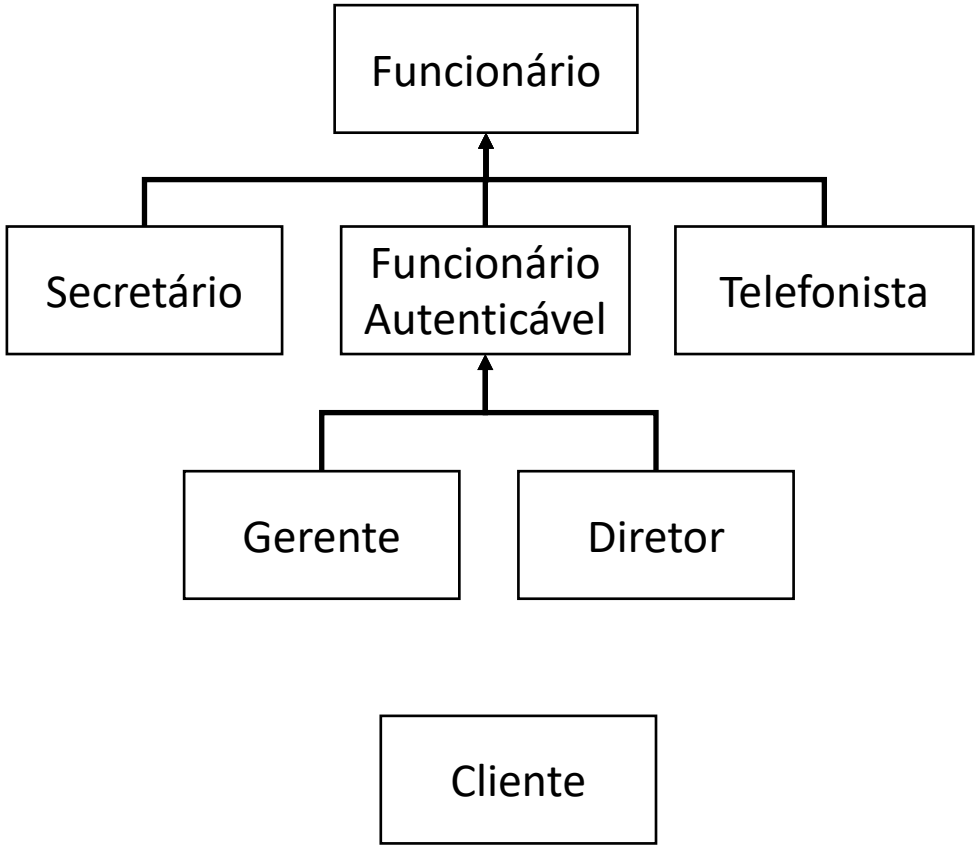
- Uma forma mais interessante seria criar uma classe no meio da árvore de herança, a FuncionárioAutenticável().
- Todas as classes que tivessem acesso ao sistema herdariam dessa classe e implementariam o método autentica(). Ou seja, essa classe e método deveriam ser abstratos.




```
class FuncionarioAutenticavel(Funcionario):

    def autentica(self, senha):
        # verifica se a senha confere
```

Herança simples

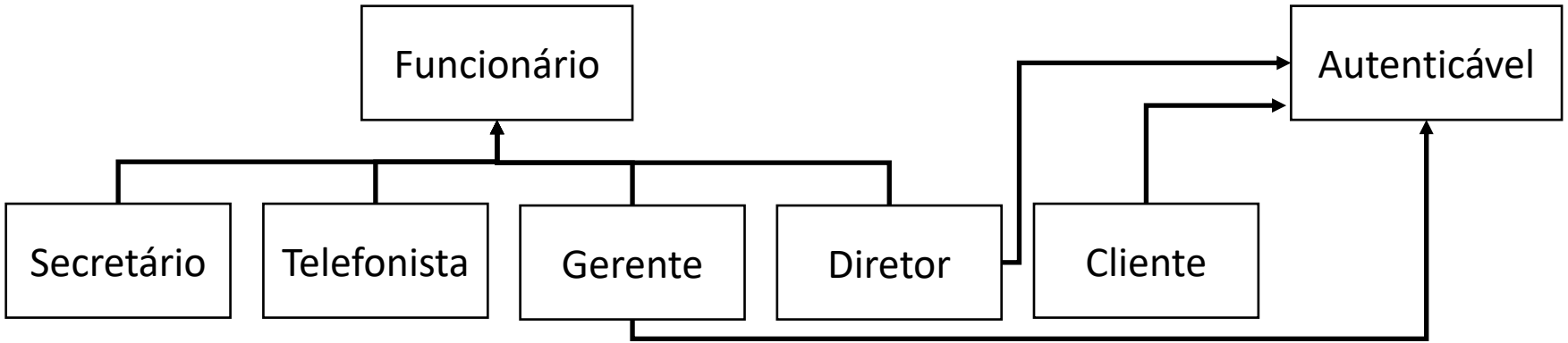
- Herança simples resolveu o problema, mas surgiu a classe Cliente que precisa acessar o sistema. O que fazer?
- Uma solução sem sentido seria fazer Cliente estender de FuncionárioAutenticável.
- Para resolver esse problema precisaríamos de uma forma de referenciar Diretor, Gerente e Cliente ao método autentica().



Herança Múltipla

```
class Autenticavel:

    def autentica(self, senha):
        # verifica se a senha confere
```

```
class Gerente(Funcionario, Autenticavel):
    # código omitido

class Diretor(Funcionario, Autenticavel):
    # código omitido

class Cliente(Autenticavel):
    # código omitido
```



Herança Múltipla

```
class SistemaInterno:

    def login(self, obj):
        if(hasattr(obj, 'autentica')):
            obj.autentica()
            return True
        else:
            print('{} não é autenticável'.format(self.__class__.__name__))
            return False

if __name__ == '__main__':
    diretor = Diretor('João', '111111111-11', 3000.0, '1234')
    gerente = Gerente('José', '222222222-22', 5000.0, '1235')
    cliente = Cliente('Maria', '333333333-33', '1236')

    sistema = SistemaInterno()
    sistema.login(diretor)
    sistema.login(gerente)
    sistema.login(cliente)
```

Interfaces

- Podemos definir uma interface como um contrato entre a classe e o mundo exterior. Quando uma classe implementa uma interface, se compromete a fornecer o comportamento publicado por essa interface.

- As interfaces são formadas pela declaração de um ou mais métodos, os quais obrigatoriamente não possuem corpo.

- De um modo geral, podemos dizer que as interfaces definem certas funcionalidades, as quais dependem das classes que implementam as interfaces para que os métodos existam.

Interfaces

```
import abc

class Autenticavel(abc.ABC):
    """Classe abstrata que contém operações de um objeto autenticável.

    As subclasses concretas devem sobrescrever o método autentica
    """

    @abc.abstractmethod
    def autentica(self, senha):
        """ Método abstrato que faz verificação da senha
        return True se a senha confere, e False caso contrário.
        """
```

Interfaces

```python
class Funcionario():

    def __init__(self, nome):
        self._nome = nome

class Gerente(Funcionario):

    def __init__(self, nome, senha):
        super().__init__(nome)
        self._senha = senha

    def autentica(self, senha):
        if self._senha == senha:
            print('Logado')
            return True
        else:
            return False
```

```python
from autenticavel import Autenticavel
class SistemaInterno():

    def login(self, obj):
        if isinstance(obj, Autenticavel):
            return obj.autentica('123')
        else:
            print('Método autentica não implementado')
            return False
```

```python
import abc

class Autenticavel(abc.ABC):

    @abc.abstractmethod
    def autentica(self, senha):
        pass
```

```python
from funcionario import Funcionario, Gerente
from autenticavel import Autenticavel
from sistemaInterno import SistemaInterno

g = Gerente('flavio', '123')
Autenticavel.register(Gerente)
SistemaInterno().login(g)
```

Exercício

- Façam as questões 1, 2, 3, 4, 5, 6, 7, 8 , 9, 10, 11 e 12 nas páginas 165, 166, 167 e 168 da apostila da Caelum.