Java - Herança e Interface

BCC 221 - Programação Orientada a Objectos(POO)

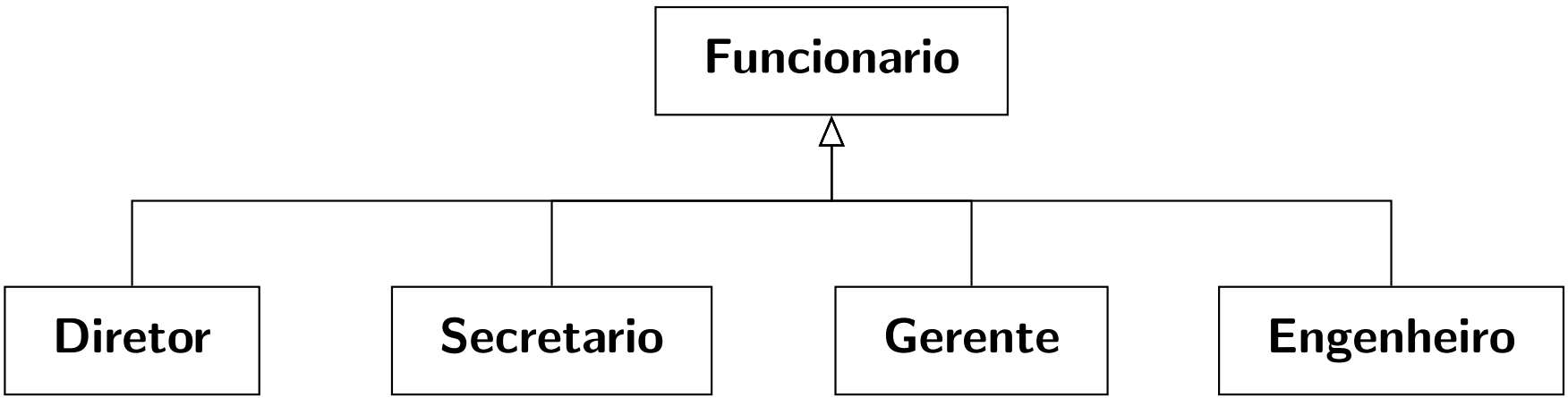
Guillermo Cámara-Chávez

Departamento de Computação - UFOP
Exemplo extraído de `www.caelum.com.br`

Polimorfismo

- Imagine que un sistema de controle de Banco pode ser acessado, além dos gerentes, também pelos Diretores, ambos precisam autenticar-se.

Polimorfismo (cont.)



Polimorfismo (cont.)

- Classe *Diretor*

```
public class Director extends Funcionario{
    public boolean autentica(int senha){
        // verifica se a senha confere
    }
}
```

- Classe *Gerente*

```
public class Gerente extends Funcionario{
    public boolean autentica(int senha){
        // verifica se a senha confere
    }
}
```

Polimorfismo (cont.)

- O método de autenticação de cada tipo de *Funcionario* pode variar muito
- Considere o *SistemaInterno* que recebe um *Diretor* ou *Gerente* como argumento e verificar se ele se autentica e colocá-lo dentro do sistema

```
public class SistemaInterno{

    public void login(Funcionario funcionario){
    // invocar o método autentica?
    // não da, nem todo funcionario tem autorização
    }
}
```

Polimorfismo (cont.)

- Problema:
  - aceita qualquer tipo de *Funcionario*, nem todo *Funcionario* possui o método *autentica*
  - Não é possível chamar ao método com a refêrencia a *Funcionario*

```
public class SistemaInterno{
    public void login(Funcionario funcionario){
        funcionario.autentica(...);
    }
}
```

Polimorfismo (cont.)

- Solução? : criar dois métodos *login* no *SistemaInterno*
  - um para *Gerente* e outro para *Diretor*

Polimorfismo (cont.)

```
public class SistemaInterno {

    // design problemático
    public void login(Diretor funcionario) {
        funcionario.autentica(...);
    }

    // design problemático
    public void login(Gerente funcionario) {
        funcionario.autentica(...);
    }

}
```

Polimorfismo (cont.)

- Se for criado um novo *Funcionario* autenticável?
  - Será necessário adicionar um novo método de login no *SistemaInterno*

Polimorfismo (cont.)

- Uma solução mais interessante seria criar uma classe no meio da hierarquia de herença *FuncionarioAutenticavel*

```
public class FuncionarioAutenticavel extends
    Funcionario {

    public boolean autentica(int senha) {
        // faz autenticacao padrão
    }

    // outros atributos e métodos

}
```

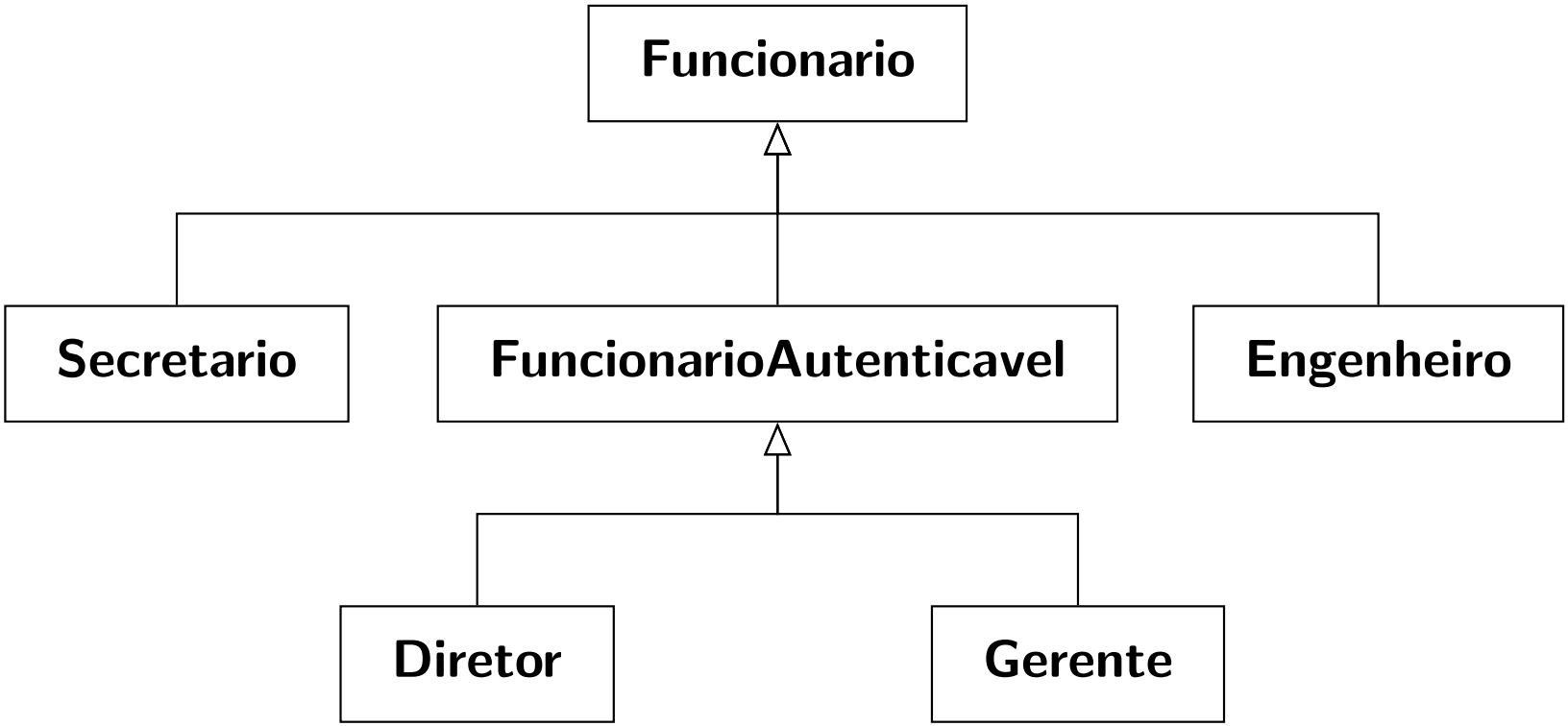
Polimorfismo (cont.)

- As classes *Diretor* e *Gerente* passariam a estender de *FuncionarioAutenticavel*
- O *SistemaInterno* receberia referências desse tipo

Polimorfismo (cont.)

```
public class SistemaInterno {

    public void login(FuncionarioAutenticavel fa) {

        int senha = //pega senha de um lugar, ou de um scanner de
            polegar

        // aqui eu posso chamar o autentica!
        // Pois todo FuncionarioAutenticavel tem
        boolean ok = fa.autentica(senha);

    }
}
```

Polimorfismo (cont.)



Polimorfismo (cont.)

- *FuncionarioAutenticavel* é uma forte candidata a classe abstrata
- O método *autentica* também pode ser um método abstrato
- O uso de herança resolve este caso em particular

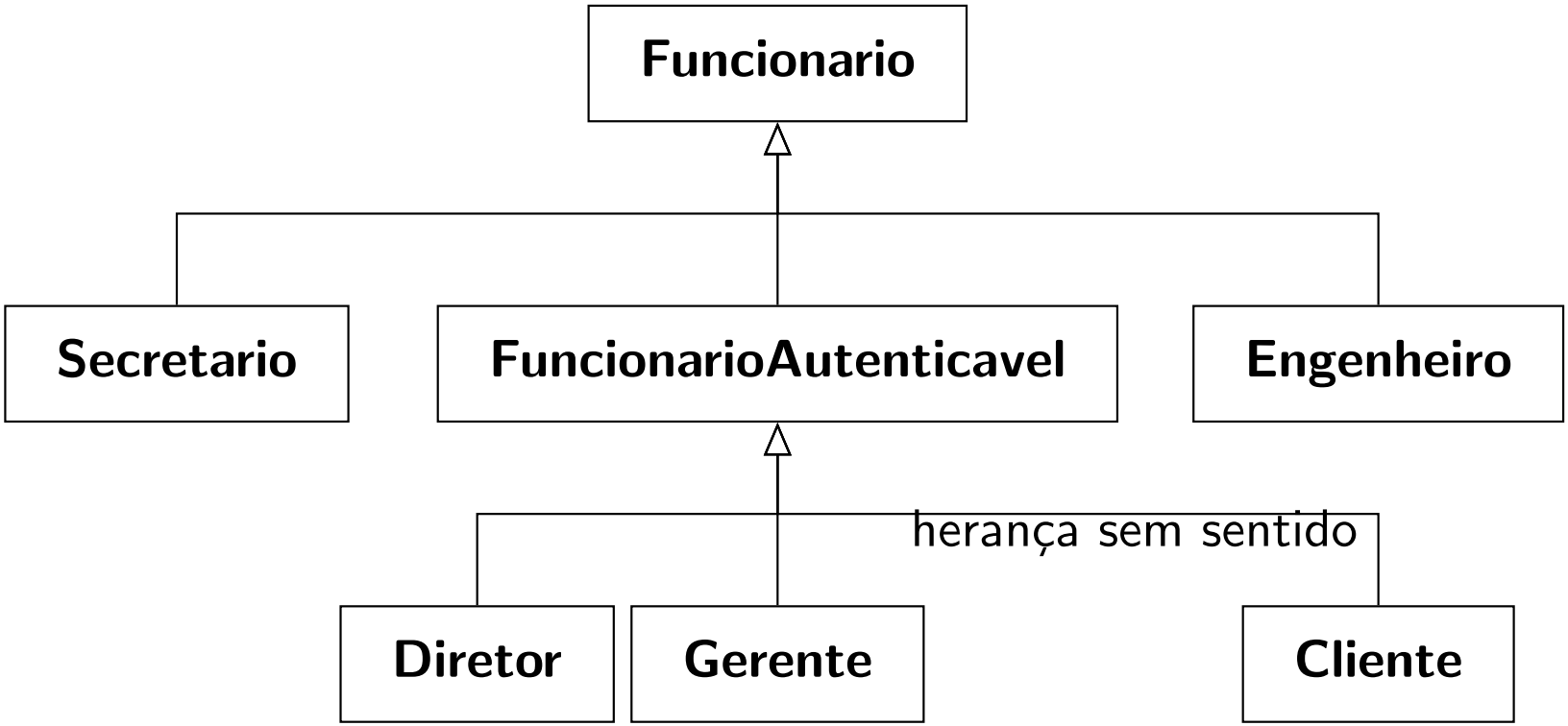
Polimorfismo (cont.)

- Vejamos uma outra situação mais complexa: precisa-se que todos os clientes também tenham acesso ao *SistemaInterno*
- Criamos outro método *login* em *SistemaInterno*?
  - Essa opção já foi descartada

Polimorfismo (cont.)

- Uma outro opção, por certo errada, seria *Cliente* herdar de *FuncionarioAutenticavel*
  - Resolve o problema parcialmente, já trará diversos outros
  - *Cliente* **não é** um *FuncionarioAutenticavel*, pode invocar métodos da classe *FuncionarioAutenticavel* que não lhe corresponde
- **Não faça** herança quando a relação não é estritamente "é um"

Polimorfismo (cont.)



Polimorfismo (cont.)

- Como resolver o problema?
- Arranjar uma forma de referenciar *Diretor*, *Gerente* e *Cliente* de uma mesma maneira
- Solução: estabelecer um tipo de "contrato" que defina o que uma classe deve fazer se quiser um deterinado status:

```
quem quiser ser Autenticavel precisar saber fazer:
    1. autenticar dada uma senha, devolvendo um
       booleano
```

Polimorfismo (cont.)

- Quem quiser pode "assinar" esse contrato
- A classe assinante deve explicar como será feita essa autenticação
- Pode-se criar esse contrato em Java

```
public interface Autenticavel {

    boolean autentica(int senha);

}
```

Polimorfismo (cont.)

- Interface: é a maneira através da qual conversamos com um objeto

Interface

"quem desejar ser autenticável precisa saber autenticar dado um inteiro e retornando um booleano"

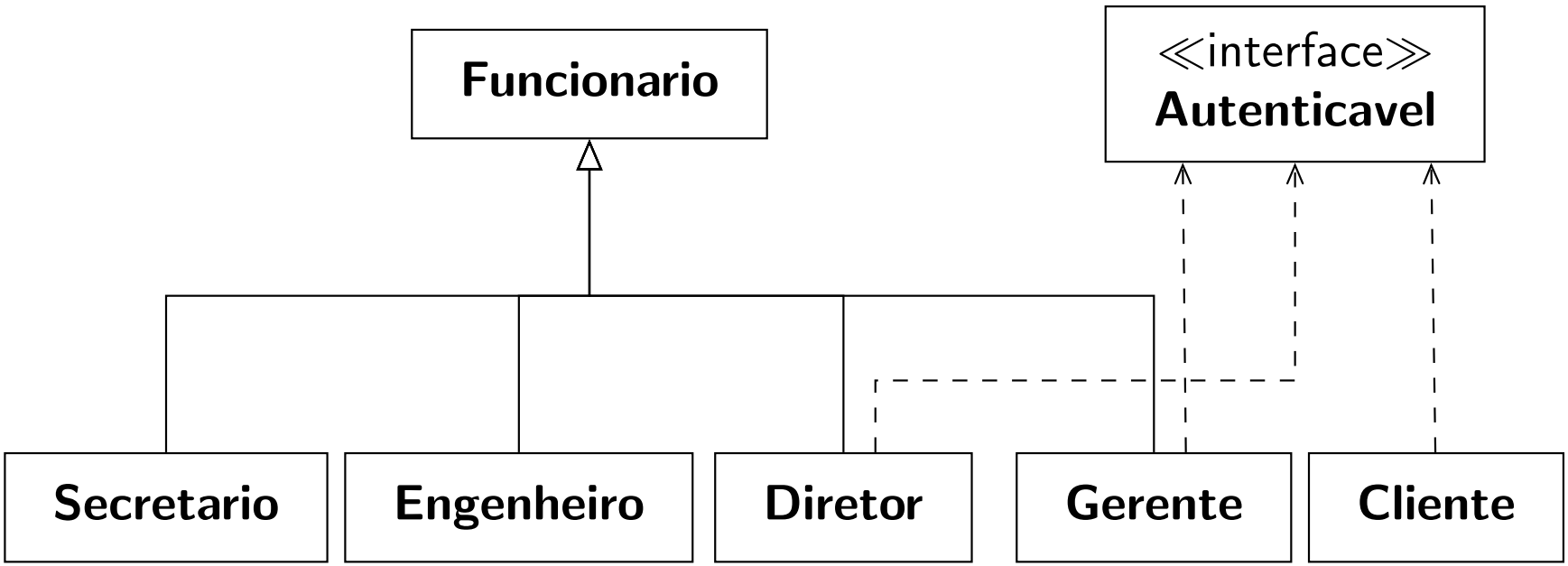
- Uma interface pode definir métodos, mas nunca conter implementaçao deles

Polimorfismo (cont.)

- Para o *Gerente* "assinar" o contrato, deve implementar a interface

```
public class Gerente extends Funcionario implements
    Autenticavel {

    private int senha;

    // outros atributos e métodos

    public boolean autentica(int senha) {
        if(this.senha != senha) {
            return false;
        }
        // pode fazer outras possíveis verificações, como saber se esse
        // departamento do gerente tem acesso ao Sistema
        return true;
    }
}
```

Polimorfismo (cont.)



Polimorfismo (cont.)

- Apartir de agora, pode-se tratar um *Gerente* como sendo *Autenticavel*
- Ganhamos mais polimorfismo, existe mais de uma forma de referencias a *Gerente*
- Quando se cria uma variável do tipo *Autenticavel*, está se criando uma referência para qualquer objeto que implemente *Autenticavel*

```
Autenticavel a = new Gerente();
// posso aqui chamar o método autentica!
```

Polimorfismo (cont.)

```
public class SistemaInterno {

    public void login(Autenticavel a) {
        int senha = // pega senha de um lugar, ou de um scanner de
            polegar
        boolean ok =     a.autentica(senha);

        // aqui eu posso chamar o autentica!
        // não necessariamente é um Funcionario!
        // Mais ainda, eu não sei que objeto a
        // referência .a.está apontando exatamente! Flexibilidade.
    }

}
```

- No dia que for necessário permitir que mais um funcionario tenha acesso ao sistema, basta implementar a interface

FIM