

# Conceitos da Orientação à Objetos

Análise de Sistemas e  
Requisitos de Software II

Aula 2

Allan Rodrigo Leite

# Motivações da Orientação a Objetos

- Avanços tecnológicos das arquiteturas de computadores
  - Suporte à ambientes de programação e interfaces homem-máquina sofisticados
- Avanços das linguagens de programação
  - Modularização, privacidade de informação, etc.
- Crise do Software: termo utilizado para descrever problemas associados ao modo como o software é desenvolvido
  - Como são realizadas as manutenções e evolução do software

# Orientação a Objetos

- Mas o que é orientação a objetos?
  - Paradigma para desenvolvimento de sistemas baseados em objetos
  - Oferece uma melhor abstração de conceitos do mundo real
  - O funcionamento de um sistema orientado a objetos ocorre através da interação entre os objetos
- Um paradigma é um conjunto de regras que estabelecem fronteiras e descrevem como resolver problemas dentro desta fronteira
  - Organiza e coordena a maneira como o mundo é observado

# Paradigmas de Desenvolvimento

## Procedural

Tipo de dados

Variável

Função  
(procedimento)

Chamada de função

## Orientação a Objetos

Classe

Objeto  
(instância)

Método  
(comportamento)

Envio de mensagem

# Vantagens da Orientação a Objetos

- A orientação a objetos reflete o mundo real de maneira mais aproximada
  - Descrevem os dados de maneira mais precisa
  - A decomposição é baseada em uma separação natural de conceitos
  - Mais fáceis de entender e manter
- Facilita a reutilização de código
  - Pequenas mudanças nos requisitos não implicam em alterações massivas no desenvolvimento do sistema
  - Implementação de tipos abstratos de dados
  - A orientação a objetos não garante a reutilização de código, apenas fornece mecanismos para que isto ocorra

# Linguagens Orientadas a Objetos

- Linguagem baseada em objetos
  - Fornece apoio somente ao conceito de objetos
  - Exemplo: Visual-Basic, JavaScript
- Linguagem orientada a objetos
  - Fornecem todos os conceitos da orientação a objetos
  - Exemplo: Smalltalk e Java

# Linguagens Orientadas a Objetos

- Linguagem orientada a objetos híbrida
  - Linguagens que originalmente não foram projetadas para ser orientada a objetos, mas passaram a incorporar os conceitos deste paradigma
  - Exemplo: C++ e Object Pascal
- Linguagem essencialmente orientada a objetos
  - Linguagens projetadas originalmente orientadas a objetos
  - Exemplo: Smalltalk e Java

# Conceitos da Orientação a Objetos

Classe

Objeto

Encapsulamento

Abstração

Herança

Classe abstrata

Interface

Acoplamento

Polimorfismo

Delegação



# Classe

- Refere-se a descrição estrutural e comportamental de um conceito
  - Atributos e comportamento
- Atributos representam as característica de uma classe
  - Define a estrutura da classe
- Operações caracterizam o comportamento de um conceito
  - Única maneira para acessar, manipular e modificar o conteúdo dos atributos de um objeto

# Classe

- Exemplo de uma classe

Pessoa
nome dataNascimento
calculaIdade()

# Classe

- Exemplo de uma classe

Nome  
da classe

<b>Pessoa</b>
nome dataNascimento
calculaIdade()

# Classe

- Exemplo de uma classe

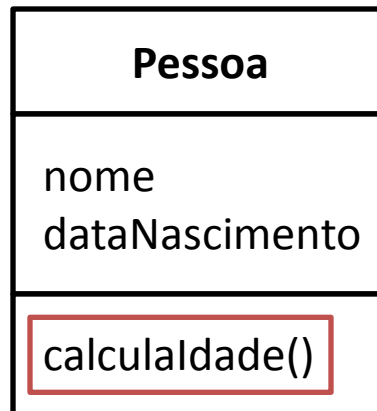
Lista de  
atributos

Pessoa
nome dataNascimento
calculaIdade()

# Classe

- Exemplo de uma classe

Lista de  
métodos



# Objeto

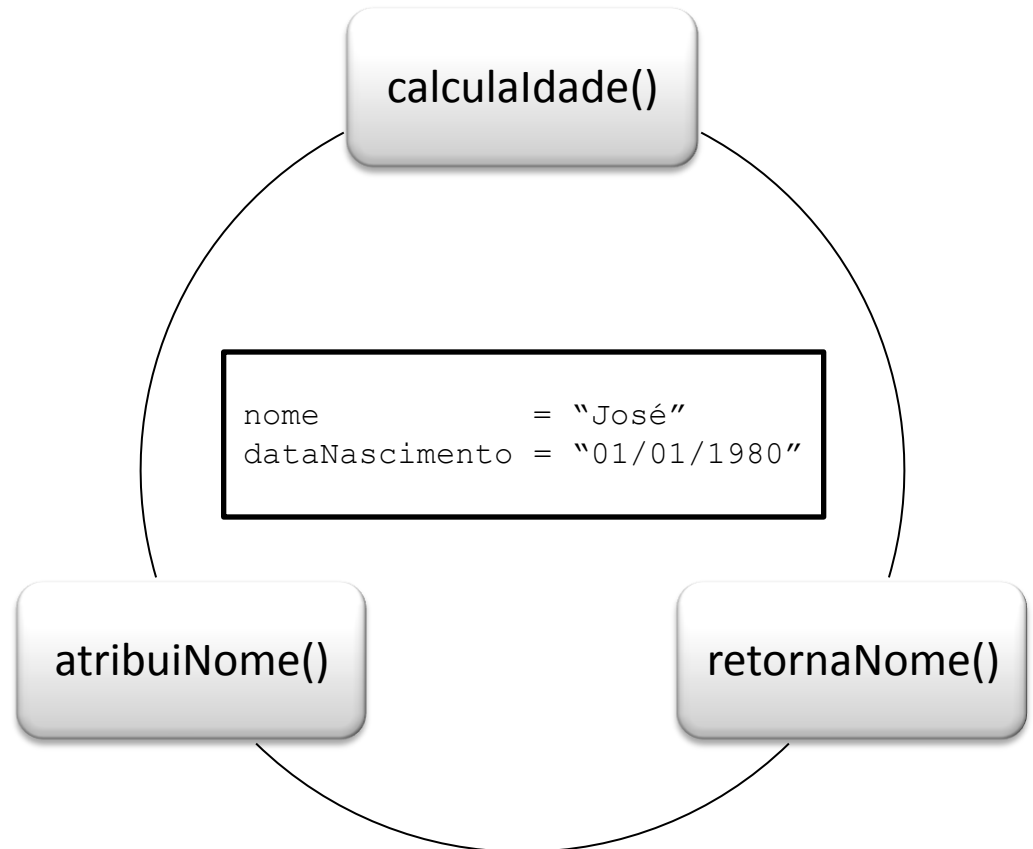
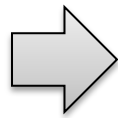
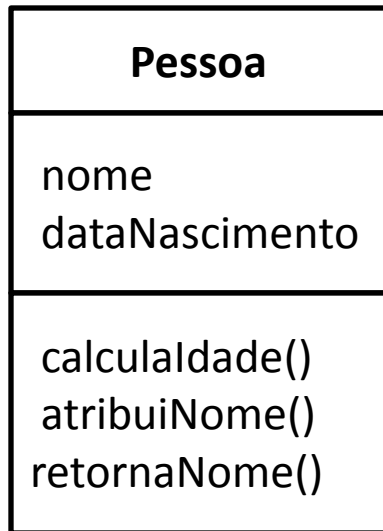
- Objeto refere-se a instância de uma classe
- Um objeto detém o controle de seu estado e reage a mensagens enviadas a ele
  - O estado de um objeto é caracterizado como o conjunto de valores assumidos pelos seus atributos
- Vários objetos podem ser criados a partir de uma única classe
  - A classe funciona como uma espécie de “molde” para o objeto, definindo suas características e seu comportamento

# Objeto

- Formalmente, um objeto deve possuir:
  - Um estado, que é normalmente implementado através de atributos e as ligações que o objeto pode ter com outros objetos
  - Uma identidade única capaz de distinguir um objeto de todos os outros objetos
  - Um comportamento, que define como um objeto reage às requisições de outros objetos

# Objeto

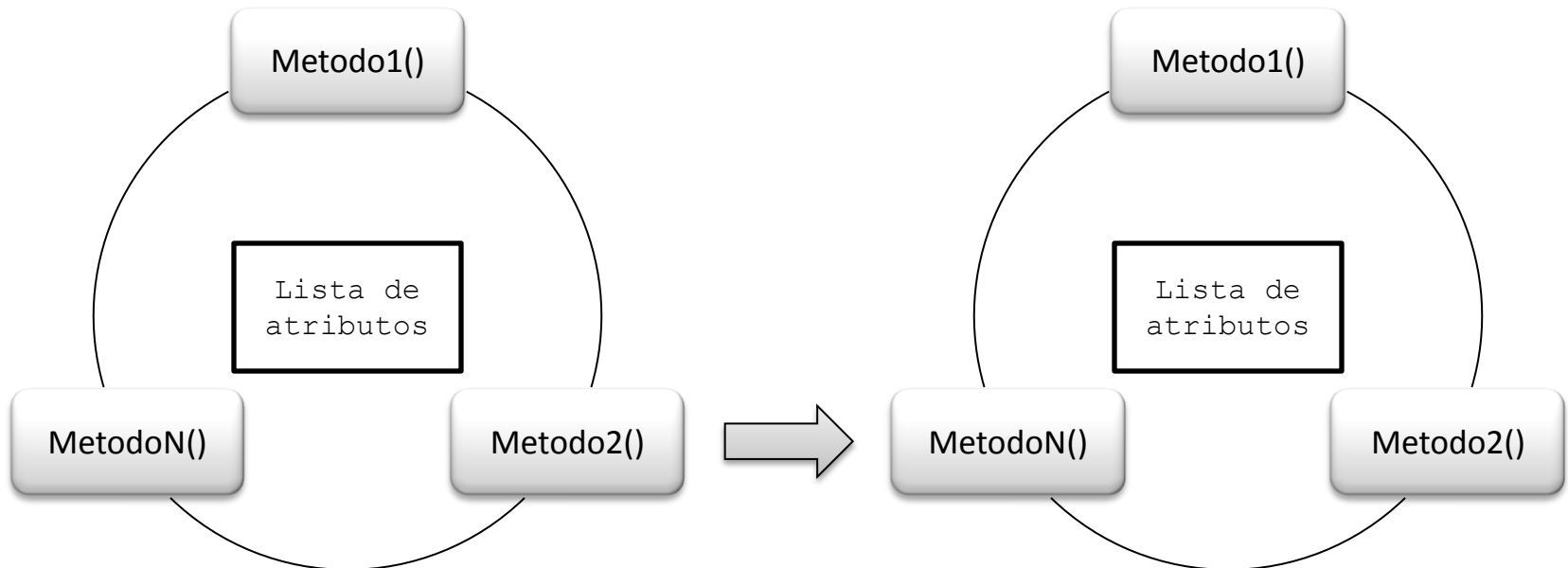
- Exemplo de um objeto





# Objeto

- Os objetos se comunicam através de mensagens
  - Cada mensagem contém a operação que será invocada no objeto de destino



# Encapsulamento

- Os dados de um objeto devem ser totalmente escondidos e protegidos de outros objetos
- Um objeto pode apresentar seus atributos privados, acessível somente através dos métodos definidos na sua interface pública
  - Não se deve permitir o acesso direto aos atributos de uma classe

# Encapsulamento

- Vantagens
  - Não é possível alterar diretamente o estado de um objeto
    - Apenas através de uma operação
  - A implementação do objeto pode sofrer alterações, sem necessidade de modificar o código do objeto solicitante
  - A manutenção é mais fácil e menos custosa
    - O software torna-se mais legível e bem estruturado

# Abstração

- Termo utilizado para definir a capacidade de redução da complexidade do software
  - Cada objeto deve ser visto como uma caixa preta, não é necessário que se conheça detalhes de sua implementação para utilizá-lo
  - Basta conhecer suas operações e os parâmetros de cada operação
- Para isto, cada objeto deve conhecer seu “papel” dentro do sistema

# Abstração



**A abstração...**

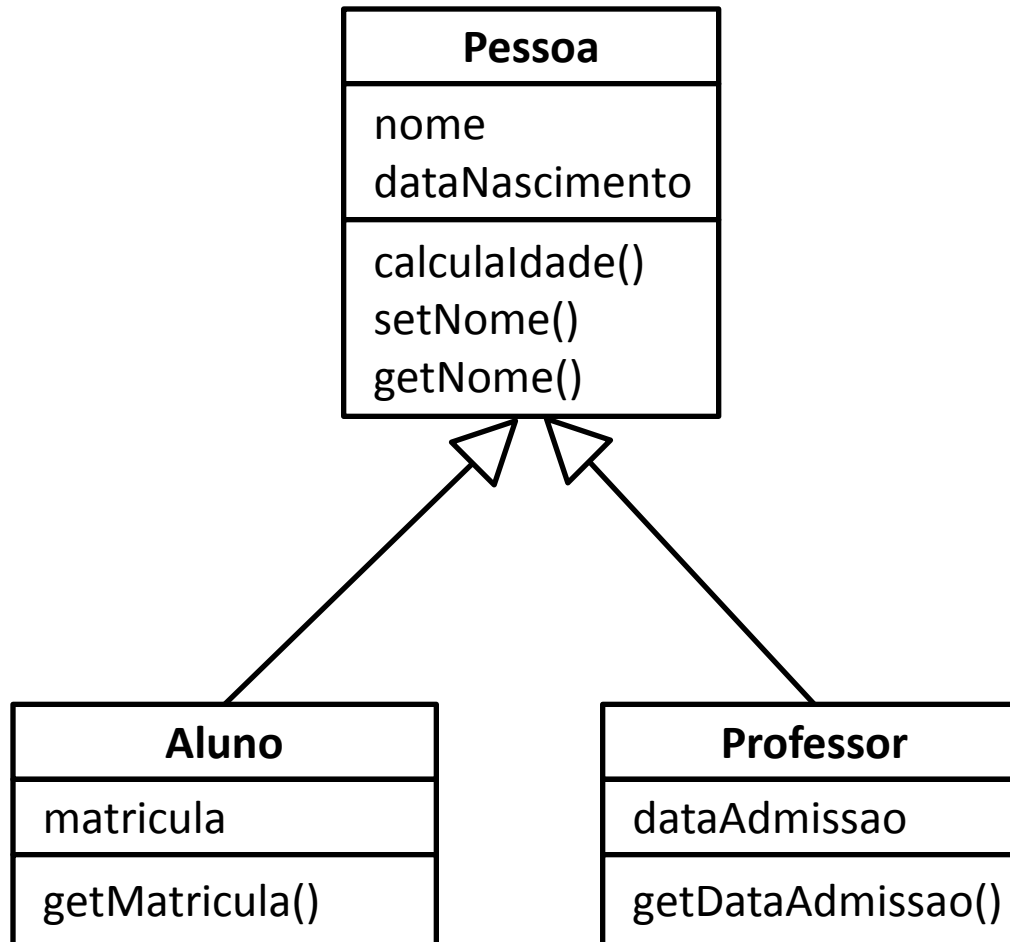


**... da implementação concreta**

# Herança

- Permite a reutilização da estrutura e do comportamento de uma classe ao ser definida uma nova classe
  - Conhecida como relacionamento “é um”
  - A classe que herda o comportamento é chamada de subclasse e a que definiu o comportamento superclasse

# Herança

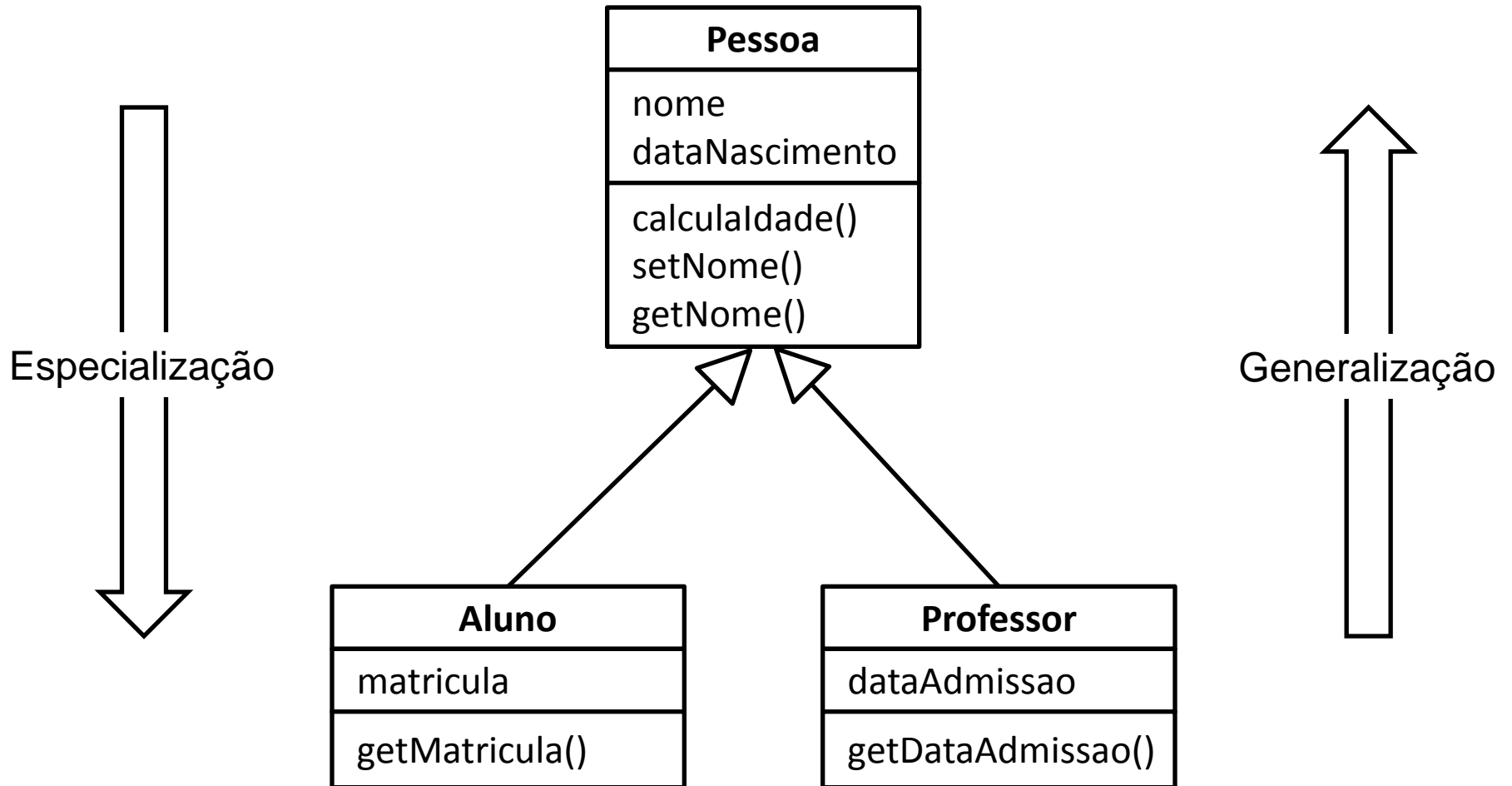


# Herança

- Através da herança é possível representar uma relação de generalização e especialização entre duas classes:
  - A superclasse é uma generalização da subclasse
  - A subclasse é uma especialização da superclasse



# Herança



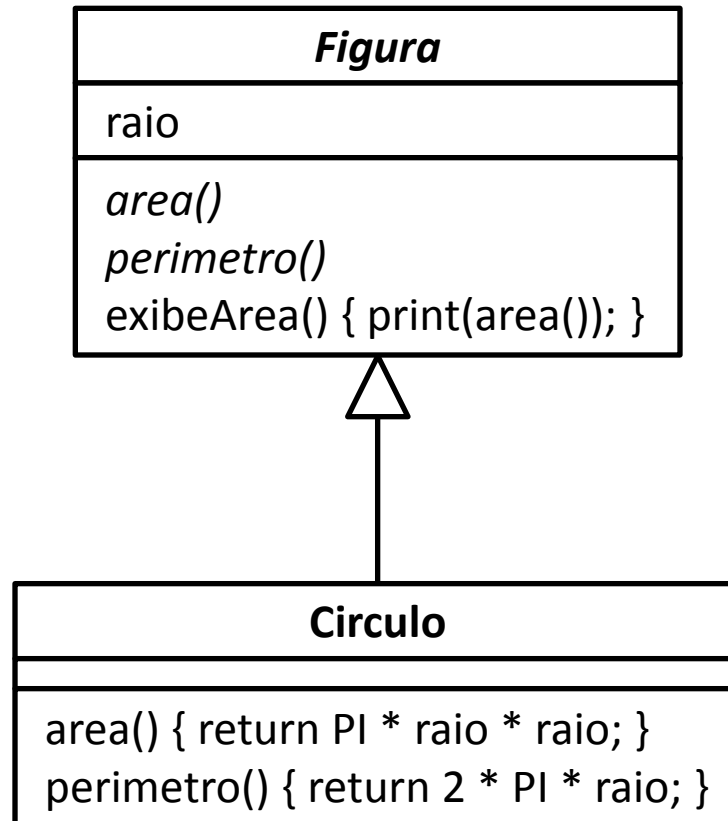
# Herança

- Uma subclasse pode:
  - Adicionar novos atributos e métodos
  - Redefinir um método existente
  - Remover um método (pouco comum)
- A ação de redefinir um método ou atributo é chamada de sobrecarga (overloading)

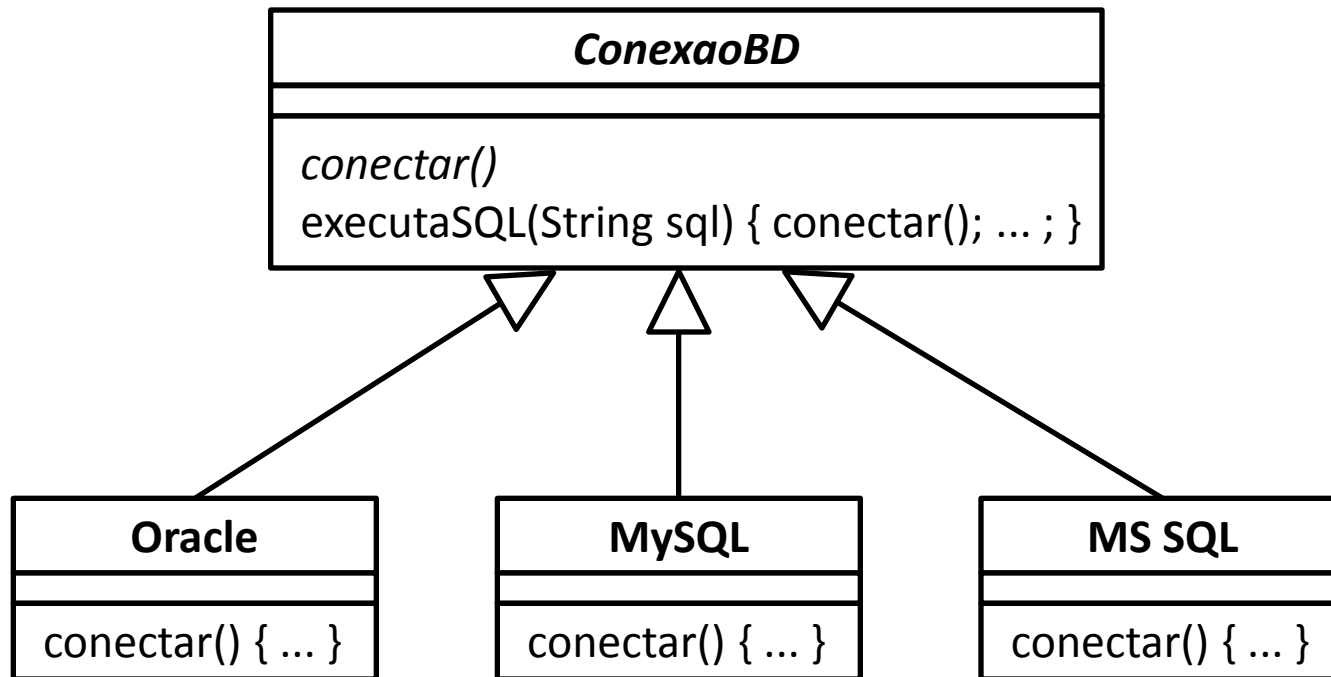
# Classe Abstrata

- Classe que possui métodos abstratos
  - Não apresenta instâncias diretas
  - Os métodos abstratos devem ser implementados pelas subclasses da classe abstrata
- Por que usar classes abstratas?
  - Em algumas situações faz sentido mover o máximo de funcionalidade possível para uma superclasse
  - O objetivo ao criar classes abstratas é encapsular outras classes com comportamento comum, promovendo a redução de código

# Classe Abstrata



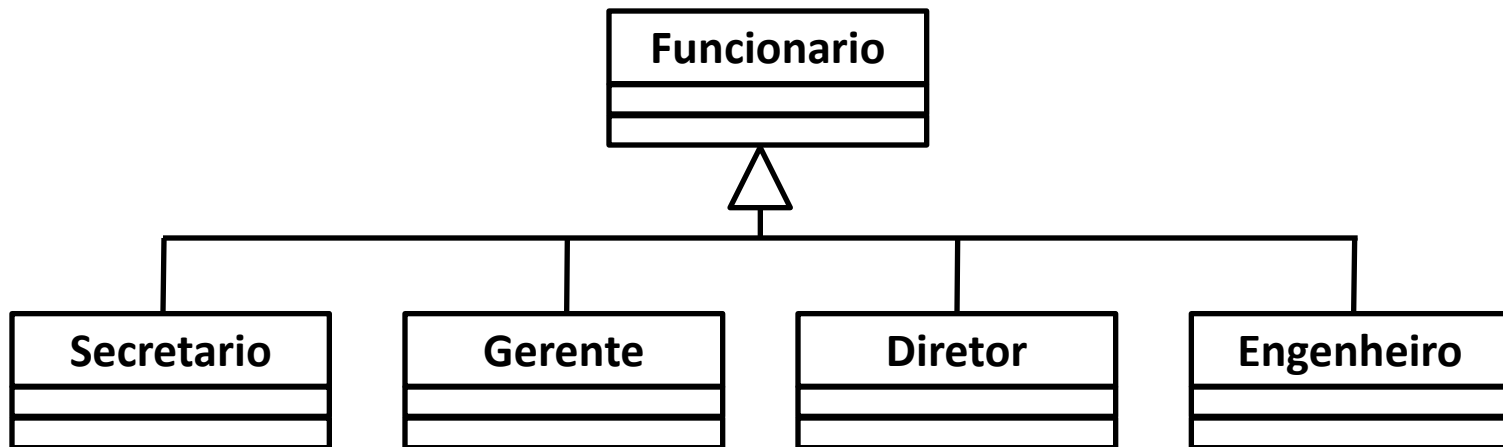
# Classe Abstrata



# Interface

- Uma interface especifica um conjunto de métodos que devem ser oferecidos por objetos que a implementarem
- Interfaces permitem a herança de tipos, não a herança de comportamento
  - Portanto, não permitem a reutilização de código
- Uma interface pode definir uma série de métodos, mas nunca conter implementação deles
  - Somente expõe o que o objeto deve fazer, e não como ele faz, nem o que ele tem
  - Como ele faz vai ser definido em uma implementação dessa interface

# Interface



- Imagine um sistema de controle de obras que possui quatro tipos de funcionários distintos
- Apenas o gerente e o diretor podem aprovar solicitações para compra de materiais de construção
  - O gerente pode aprovar se o preço for abaixo de R\$ 1000,00
  - O diretor pode solicitar de qualquer valor, desde que exista saldo suficiente em caixa

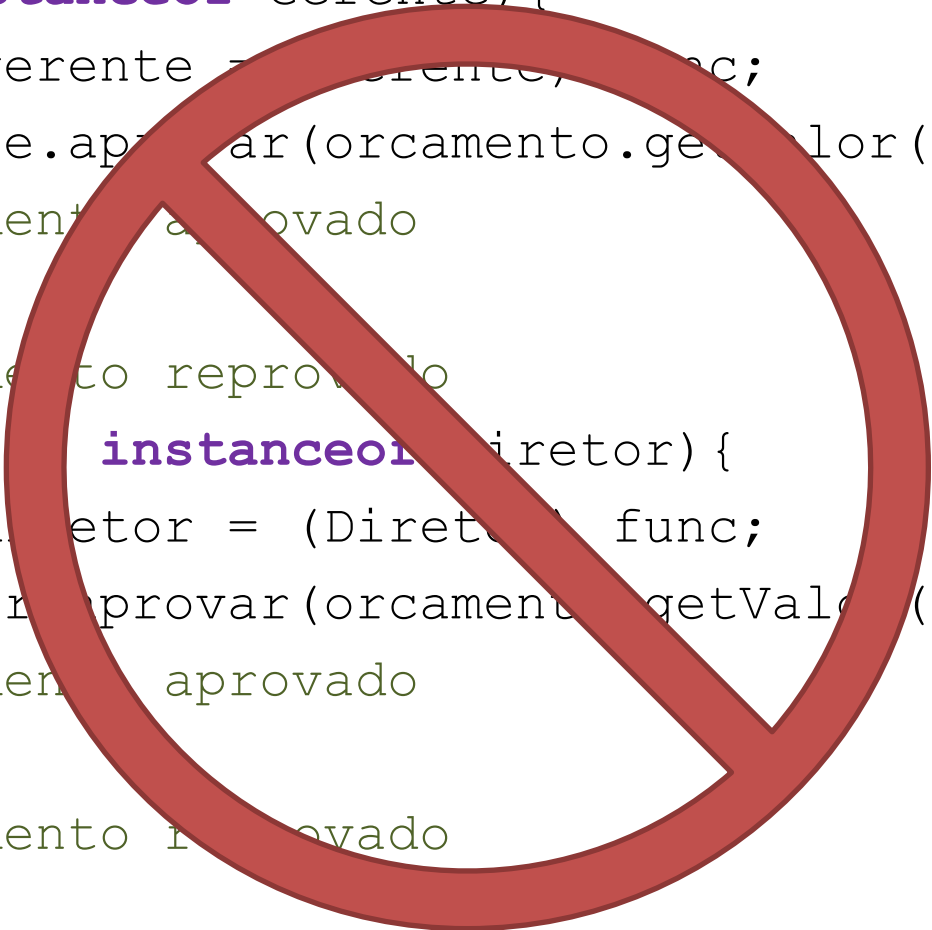
# Interface

```
void solicitaAprovacao(Orcamento orc,Funcionario func){  
    if(func instanceof Gerente){  
        Gerente gerente = (Gerente) func;  
        if(gerente.aprovar(orcamento.getValor())){  
            //Orcamento aprovado  
        }  
        else  
            //Orcamento reprovado  
    }  
    else if(func instanceof Diretor){  
        Diretor diretor = (Diretor) func;  
        if(diretor.aprovar(orcamento.getValor())){  
            //Orcamento aprovado  
        }  
        else  
            //Orcamento reprovado  
    }  
}
```

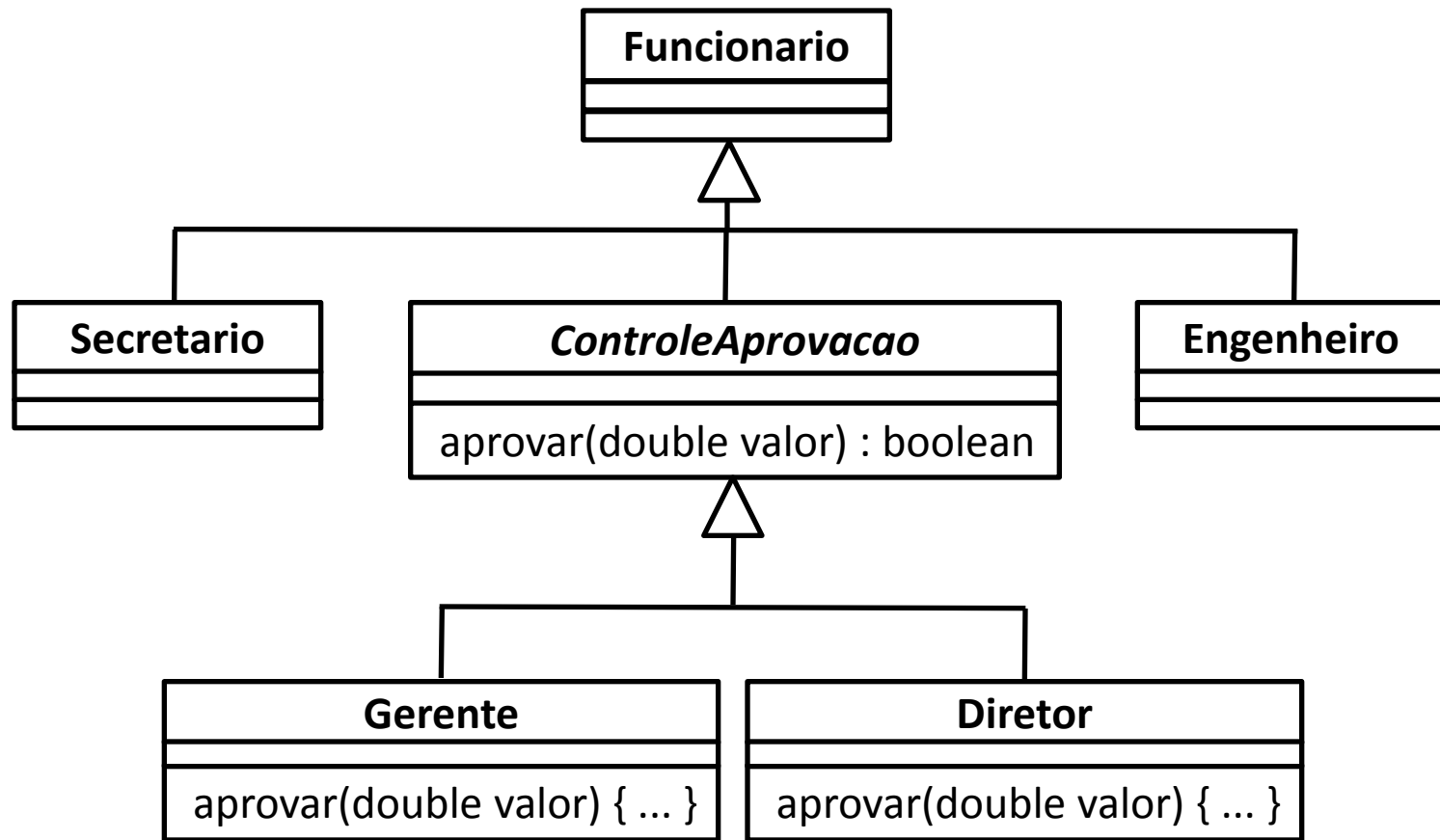


# Interface

```
void solicitaAprovacao(Orcamento orc, Funcionario func) {  
    if(func instanceof Gerente) {  
        Gerente gerente = (Gerente) func;  
        if(gerente.aprovar(orcamento.getValor())) {  
            //Orcamento aprovado  
        }  
        else  
            //Orcamento reprovado  
    } else if(func instanceof Diretor) {  
        Diretor diretor = (Diretor) func;  
        if(diretor.aprovar(orcamento.getValor())) {  
            //Orcamento aprovado  
        }  
        else  
            //Orcamento reprovado  
    }  
}
```



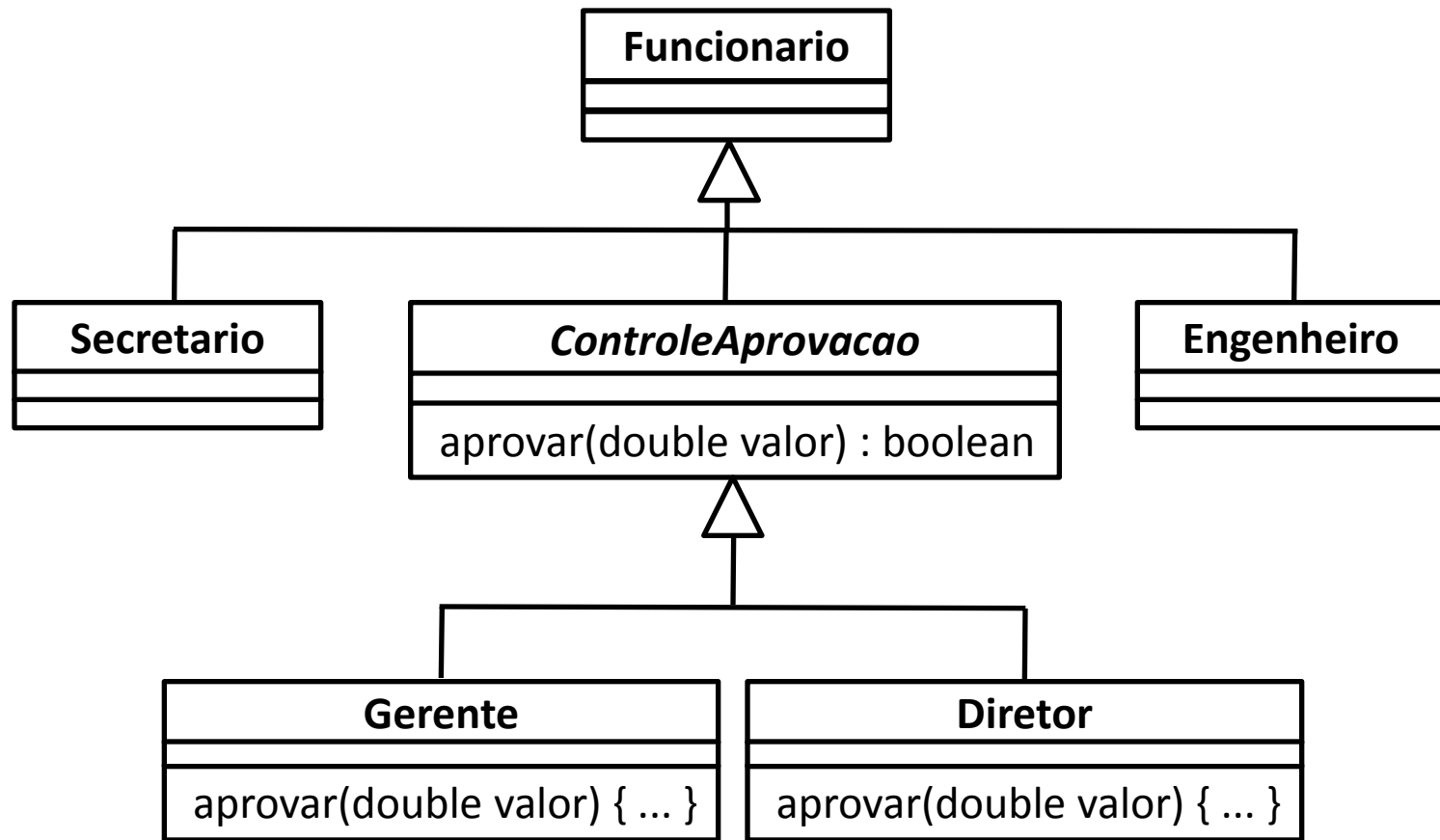
# Interface



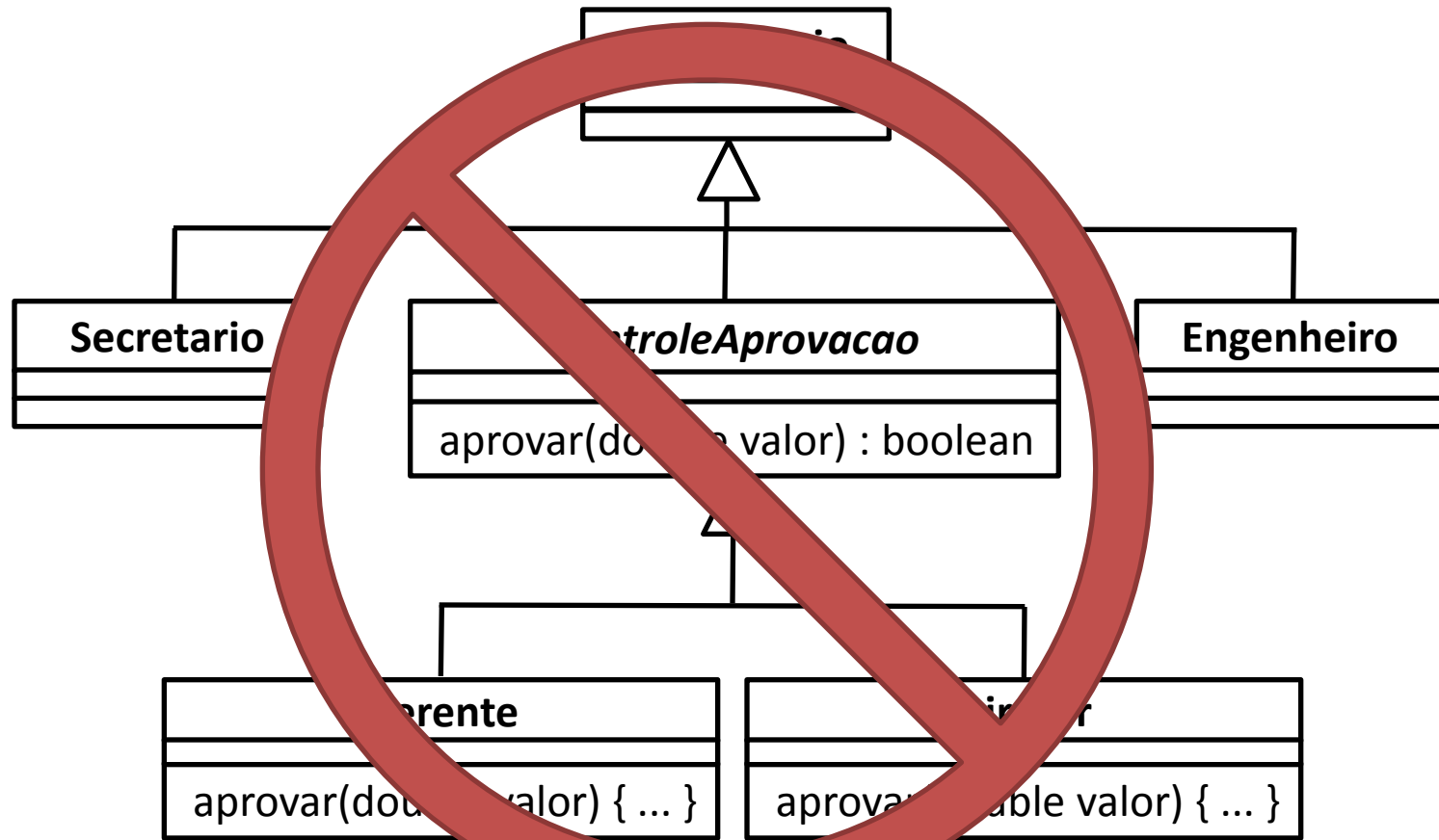
# Interface

```
void solicitaAprovacao(Orcamento orc,  
                        ControleAprovacao ctrl) {  
    if(ctrl.aprovar(orcamento.getValor())) {  
        //Orcamento aprovado  
    else  
        //Orcamento reprovado  
}
```

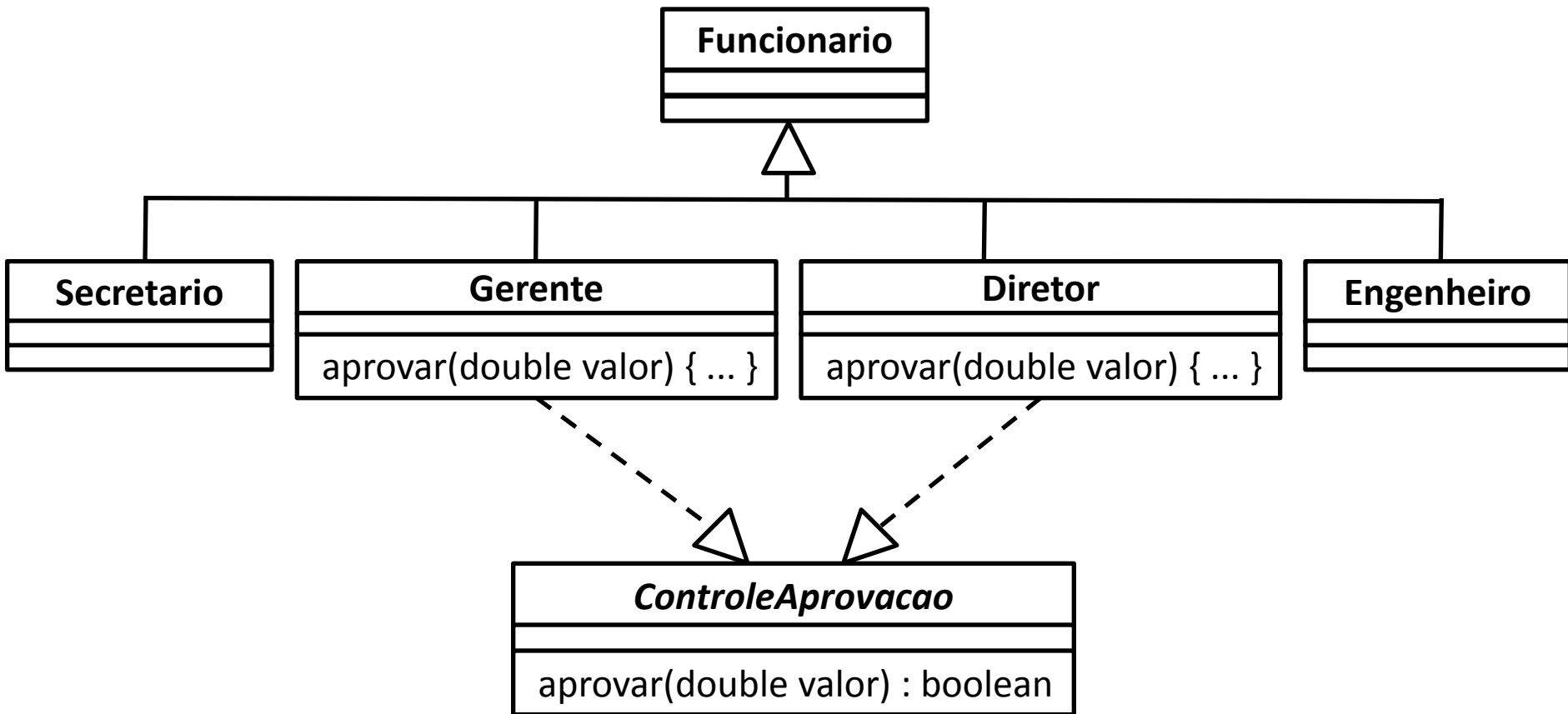
# Interface



# Interface



# Interface

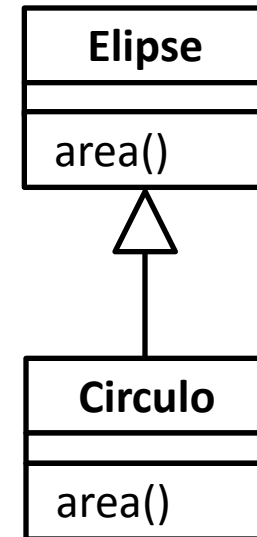


# Acoplamento

- Consiste na associação um atributo ou variável e sua respectiva classe (tipo)
- O acoplamento pode ser:
  - Estático: se ocorre antes do tempo de execução e permanece inalterado durante a execução do programa
  - Dinâmico: se ocorre durante o tempo de execução ou muda durante a execução do programa

# Acoplamento

```
Ellipse e = new Ellipse();  
Circulo c = new Circulo();  
  
e = c;  
e.area();
```



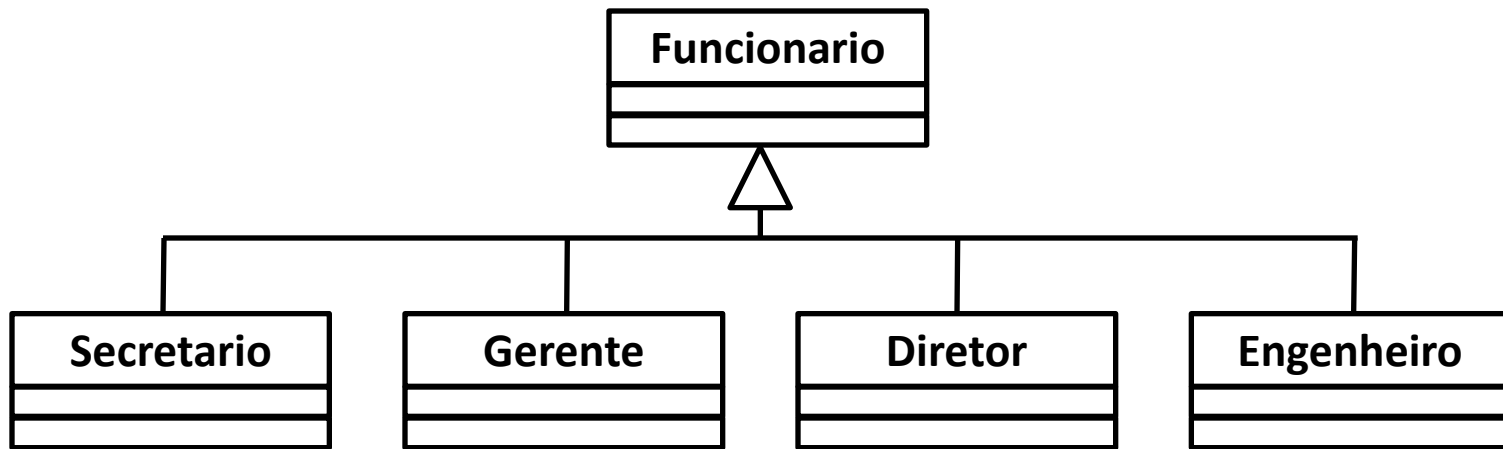
Qual será o método executado? Ellipse ou Circulo?



# Polimorfismo

- É a habilidade de variáveis assumirem “mais de uma forma ou tipo”
- Isto é possível para as superclasses de uma hierarquia de classes
- Requer o uso de acoplamento dinâmico

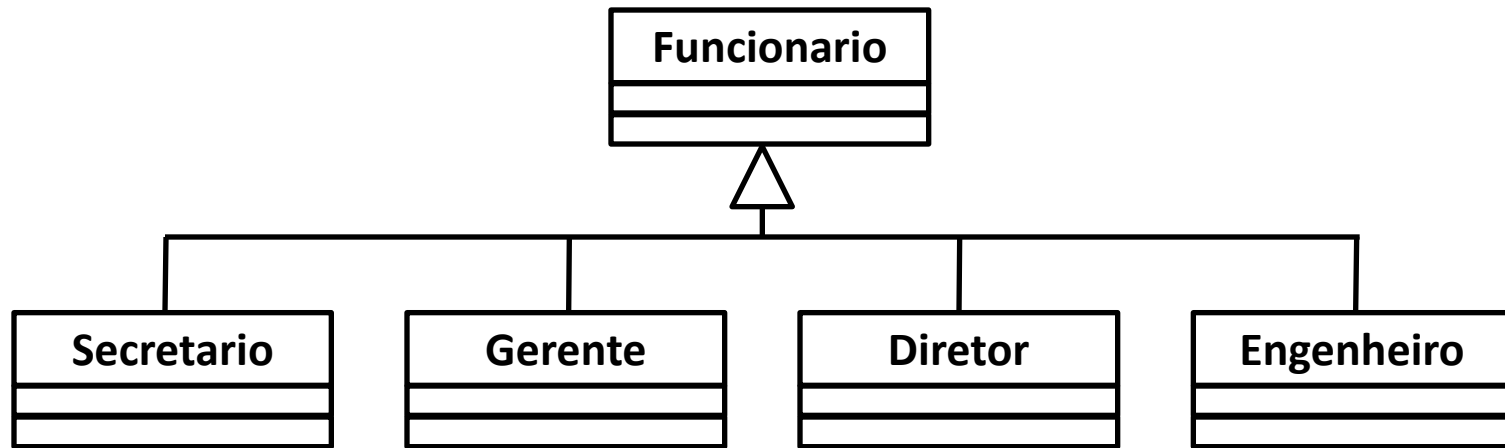
# Polimorfismo



```
Funcionario f = new Secretario();  
f.getNome();
```

```
Engenheiro e = new Funcionario();  
e.getCRC();
```

# Polimorfismo



```
Funcionario f = new Secretario();  
f.getNome();
```

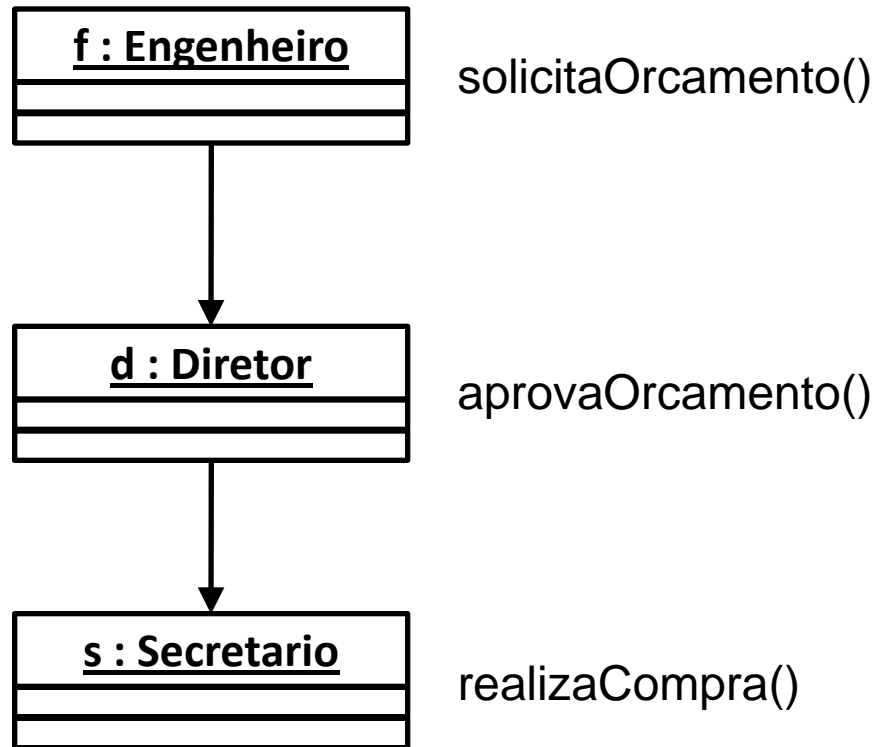
```
Engenheiro e = new Funcionario();  
e.getCRC();
```



# Delegação

- Cada objeto deve conhecer suas responsabilidades dentro do sistema
  - O comportamento do sistema emerge da interação dos objetos
  - Deve-se evitar classes centralizadoras
  - O compartilhamento dos comportamentos das classes é a maneira de interagir entre os objetos
- Herança:
  - Permite o compartilhamento de comportamento baseado em classes
- Delegação:
  - Permite o compartilhamento baseado em objetos

# Delegação



# Conceitos da Orientação à Objetos

Análise de Sistemas e  
Requisitos de Software II

Aula 2

Allan Rodrigo Leite