Introdução à Programação Orientada a Objetos usando Java

Helio Henrique L. C. Monte-Alto

Disciplina: Paradigma de Programação Imperativa e Orientada a Objetos

2012

Tópicos

Introdução

- Orientação a objetos (OO) é uma aplicação de tipos abstratos de dados
- O programa é estruturado de forma a representar os objetos e as relações entre eles no mundo real
- Exemplos de linguagens: Java, C++, Python, Ruby, Smalltalk, Javascript, C#, Scala, etc.

Discussão

Orientação a objetos é um paradigma de programação? Ou seria apenas um estilo de programação?

Classes e objetos

Classe

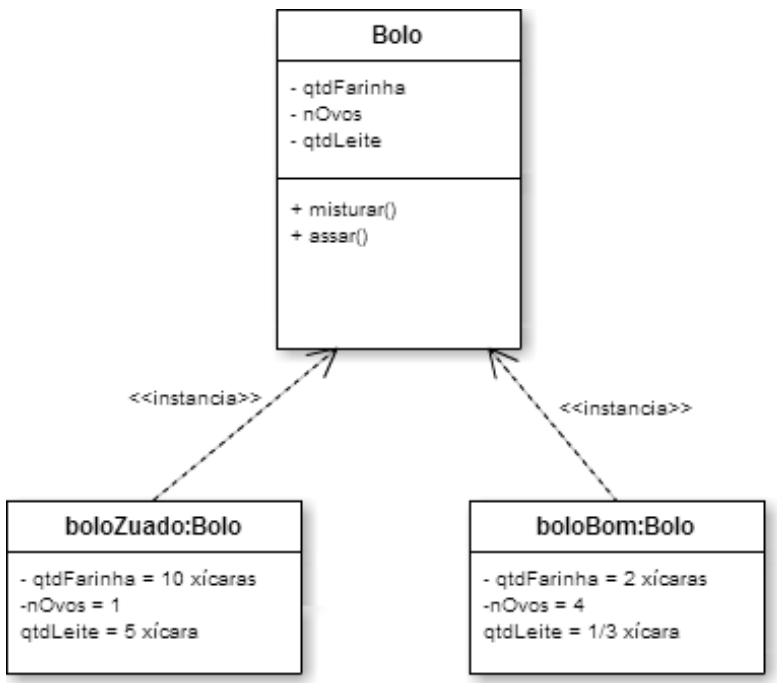
- Conjunto de indivíduos (**objetos**) com **atributos** e comportamentos (**métodos**) em comum
- Ex: classe dos animais mamíferos
  - Comportamentos: respirar, amamentar, locomover-se, etc.
  - Atributos: tempo de vida, tamanho, etc.
- Além disso, em LP, uma classe define um tipo de dados, assim como sua interface (operações / métodos)

Objeto

- Um objeto é uma instância (ou concretização) de uma classe

Classes e objetos - Exemplos

- A definição de uma classe pode ser vista como uma receita de bolo
- Os bolos criados a partir dessa receita são instâncias dessa classe



Atributos e Métodos

Atributos

- Valores dos atributos constituem o **estado** de um **objeto**
- Estado diferencia objetos da mesma classe

Métodos

- **Métodos públicos** constituem a **interface** (ou protocolo) de mensagens dos objetos de uma classe
- Diferenças na interface (métodos a mais) e na implementação (polimorfismo) diferenciam objetos de classes pais, filhas e irmãs

Exemplo Lampada 1

```
class Lampada {

    //Estado
    private boolean acesa = false;

    //Interface
    public void acender(){
        acesa = true;
    }

    public void apagar(){
        acesa = false;
    }

}
```

Questão

Quantos objetos distintos pertencem à classe Lampada?

Exemplo 1.3

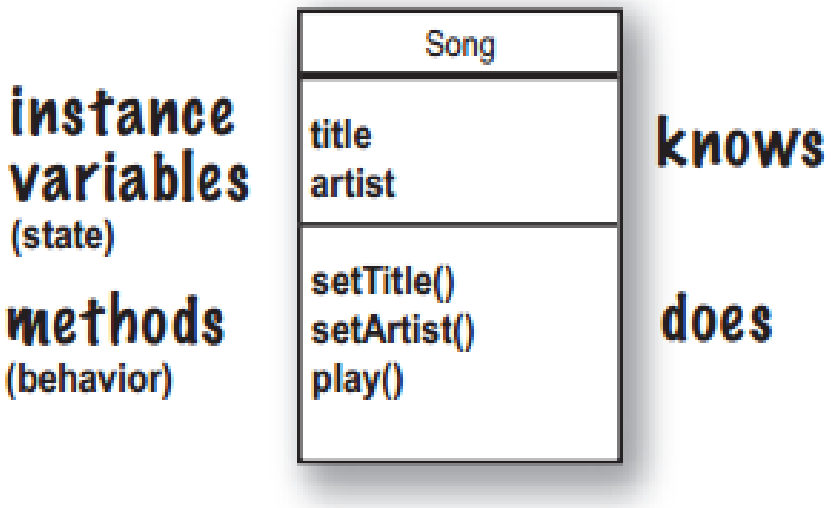
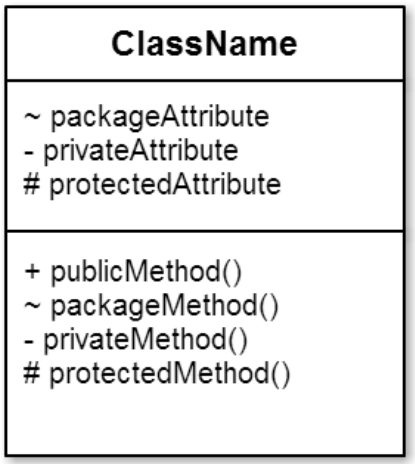


Figura : Atributos (estado): o que objeto **conhece**. Métodos: o que o objeto **faz** (Sierra e Bates, 2005)

Modificadores de acesso / visibilidade

Em Java

- `private`
- `public`
- `package`
- `protected`



Encapsulamento

Encapsulamento

- Consiste em esconder os membros da classe;
- Utilização do objeto deve ser feita apenas por meio de sua interface;
- Exemplos:
  - Carro: volante e pedais (interface); motor, rodas (implementação)
  - Estrutura de pilha: empilhar, desempilhar e ver o topo (interface); lista estática, lista ligada dinâmica (implementação)

Métodos de acesso: *getters* e *setters*

- Servem para recuperar os dados privados e fazer atribuição de maneira confiável, por meio de métodos

```
public class Conta {
    private double saldo;
    //... outros atributos omitidos

    public double getSaldo() {
        return this.saldo;
    }
    public void setSaldo(double saldo) {
        this.saldo = saldo;
    }
    //... outros metodos omitidos
}
```

Questão

É uma boa prática fazer *getters* e *setters* para todos os atributos? Por quê?

Construtores

- Servem para inicializar objetos de uma classe

```
public class Conta {
    private int numero;
    private Cliente titular;
    private double saldo;

    public Conta(int numero, Cliente titular) {
        this.numero = numero;
        this.saldo = 0;
        this.titular = titular;
    }
    public Conta(int numero, Cliente titular, double
        saldoinicial) {
        this.numero = numero;
        this.titular = titular;
        this.saldo = saldoinicial;
    }
```

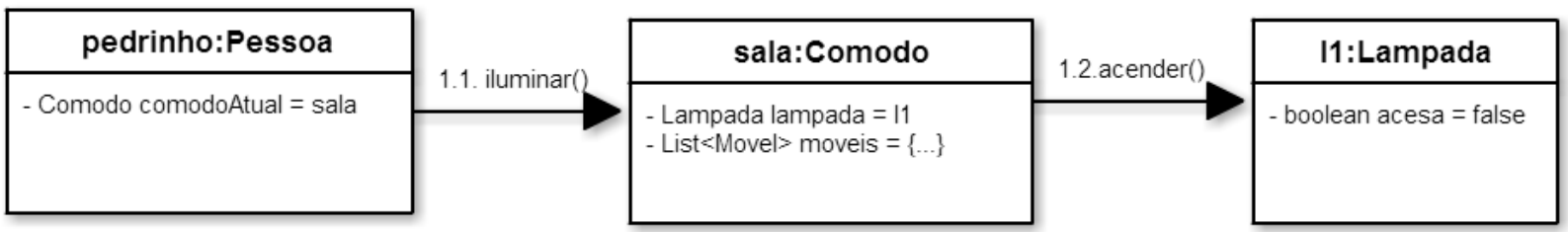
Membros de classe

- São atributos e métodos que não pertencem às instâncias, mas às classes
- Também chamados de membros estáticos

```
class Conta {
    private static int totalDeContas;
    //...
    public Conta() {    //construtor
        Conta.totalDeContas += 1;
    }
    public static int getTotalDeContas() {
        return Conta.totalDeContas;
    }
}
```

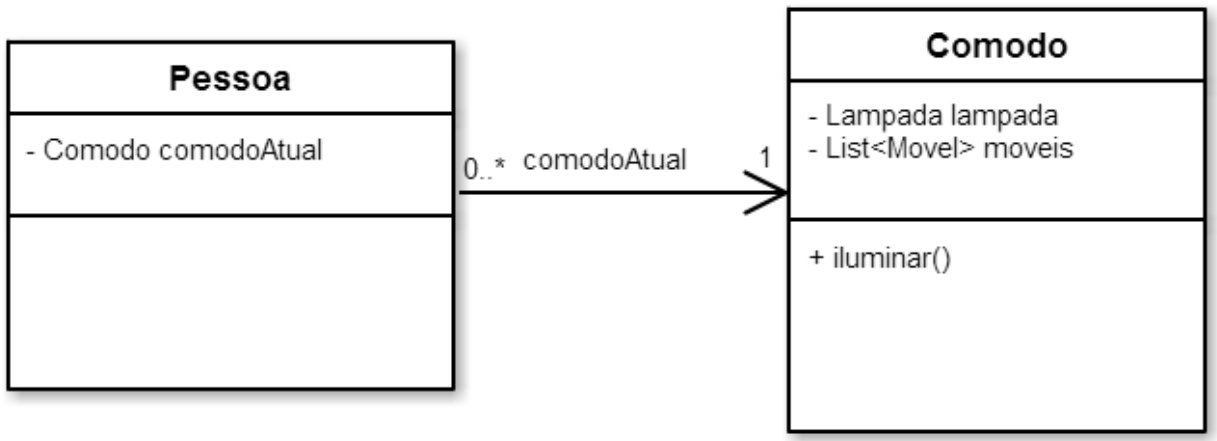
Trocas de mensagens

- Em OO, enviar uma mensagem síncrona a um objeto corresponde a chamar um método público desse objeto (lembre-se que os métodos públicos constituem o protocolo da classe)



Associações

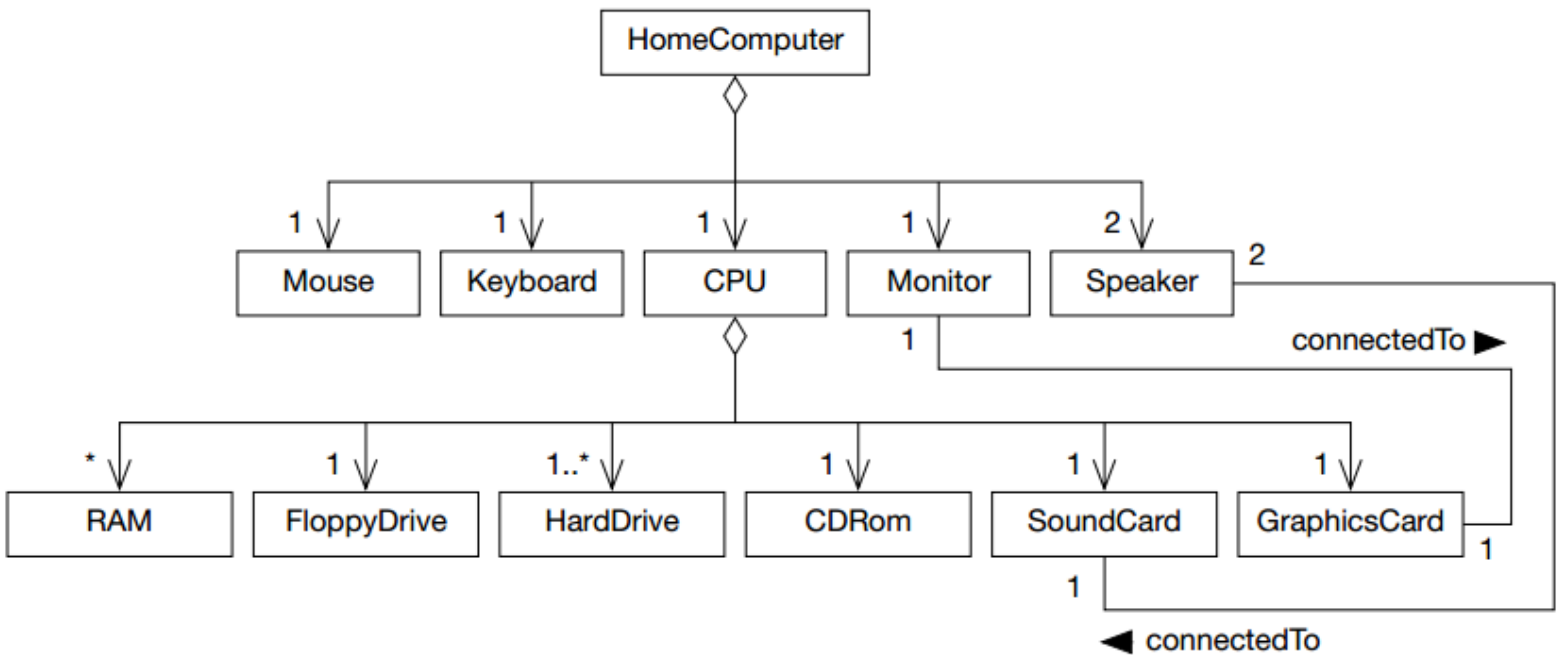
- Definem ligações entre objetos de diferentes classes



Agregação e composição (associações do tipo parte-de)

Agregação

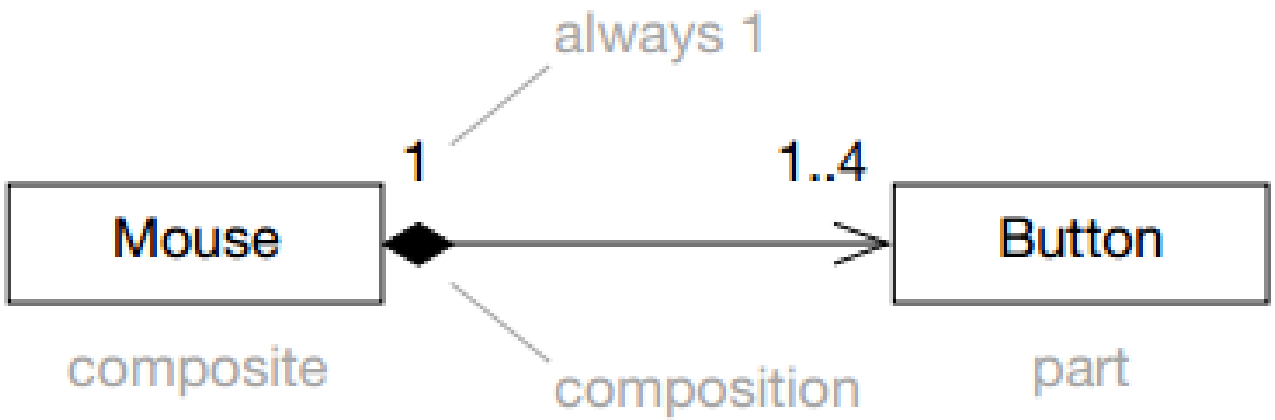
- Agregação: relação "parte-de" fraca entre objetos.
- Ex: um computador e seus periféricos. Os periféricos PODEM existir sem o computador (ligados a outro computador, por exemplo).



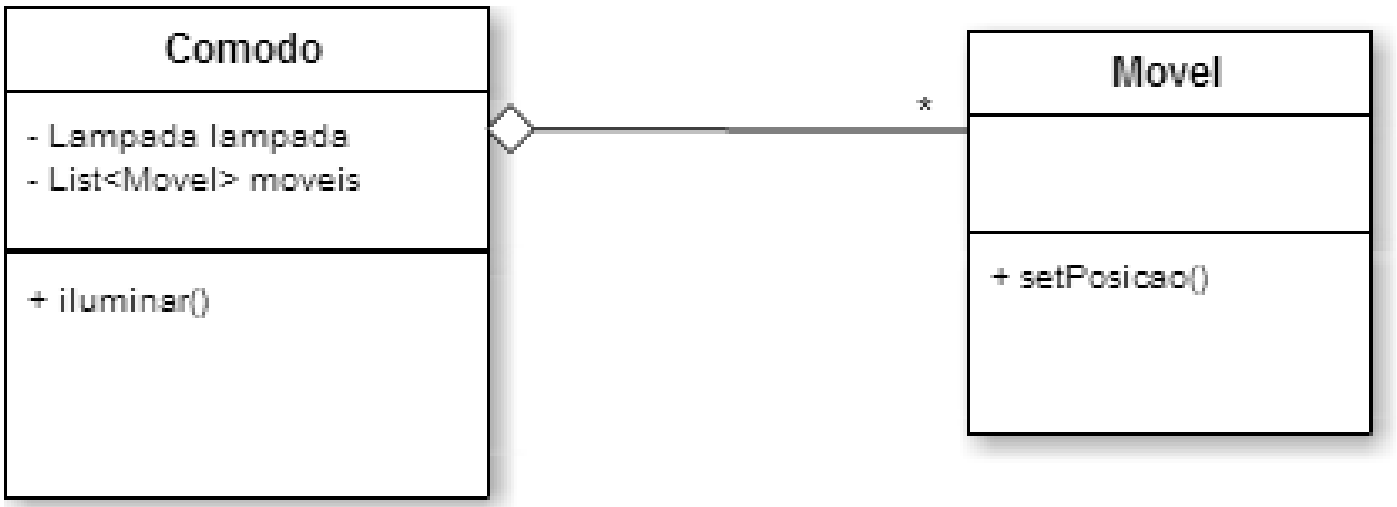
Agregação e composição (associações do tipo parte-de)

Composição

- Composição: relação forte entre objetos. Ex: um mouse e seus botões. Os botões NÃO PODEM existir independentes do mouse para o qual foram fabricados.



Agregação - Exemplo



Herança - Introdução

Motivação

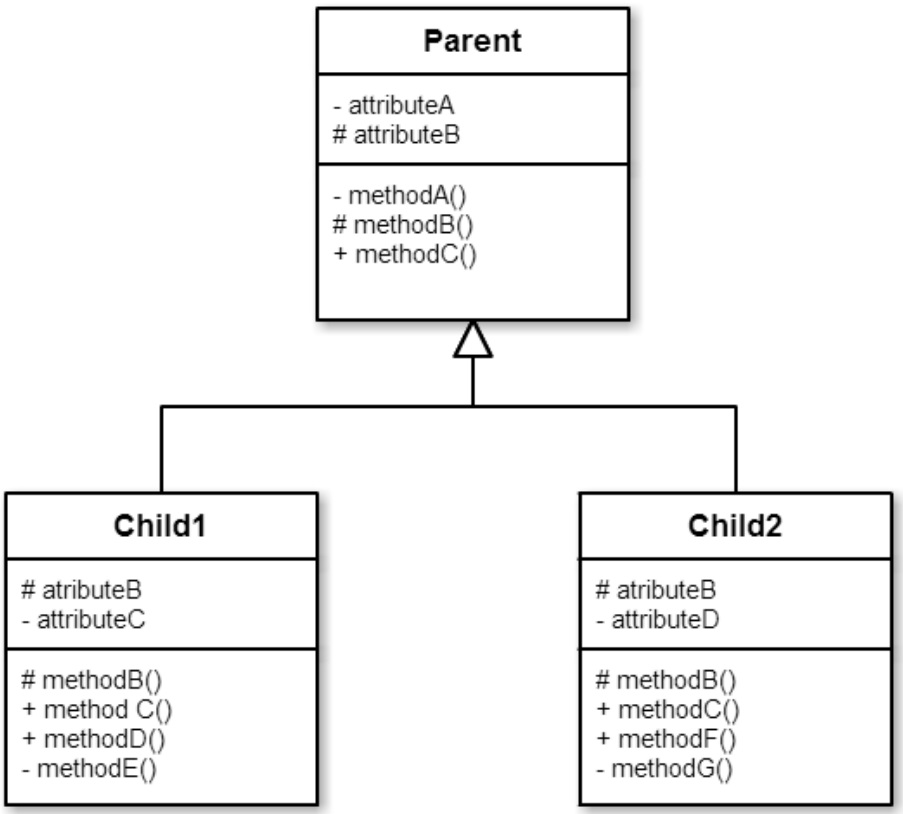
- Tipos abstratos de dados facilitam o reuso
- **Problemas:**
  - Ao criar uma nova aplicação, em quase todos os casos são requeridas modificações nos TAD já existentes
  - Tipos sem forma alguma de hierarquia não condizem com a modelagem de muitos espaços de problema
- **Solução:** herança (especialização / generalização)

Herança

Permite que novas classes herdem atributos e métodos de outra classe, podendo modificar alguns desses membros e adicionar novos membros específicos da nova classe.

Conceitos

- Uma classe `Child` que herda / estende uma classe `Parent` é uma **especialização** de `Parent`, e é chamada classe derivada, **classe filha** ou **subclasse** de `Parent`;
- Uma classe `Parent` pode ser estendida por uma classe `Child` e é uma **generalização** de `Child`, sendo chamada **classe mãe** ou **superclasse** de `Child`.
- Uma subclasse pode ou não ser subtipo de sua superclasse



Exemplo Lampada 2

```
class PiscaPisca extends Lampada{

    //Estado
    private boolean piscando = false;

    //Interface
    public void ativarPisca() {
        acender();
        piscando = true;
    }

    public void desativarPisca() {
        piscando = false;
    }

}
```

Questão

E agora? Quantos objetos distintos pertencem à classe Lampada? E à classe PiscaPisca?

Modificador de acesso `Protected`

- Um membro protegido é visível nas classes filhas
- Em Java, a cláusula `protected` também implica que o membro é visível a todas as classes do mesmo pacote (*package*)

Polimorfismo

- Métodos públicos e protegidos da classe mãe podem ser sobrescritos pelas classe filha
- A vinculação dos objetos com os métodos é feita dinamicamente, em tempo de execução

Vantagem

Permite que outras partes do programa referenciem os objetos pertencentes a uma hierarquia de classes de maneira mais transparente, facilitando o reuso.

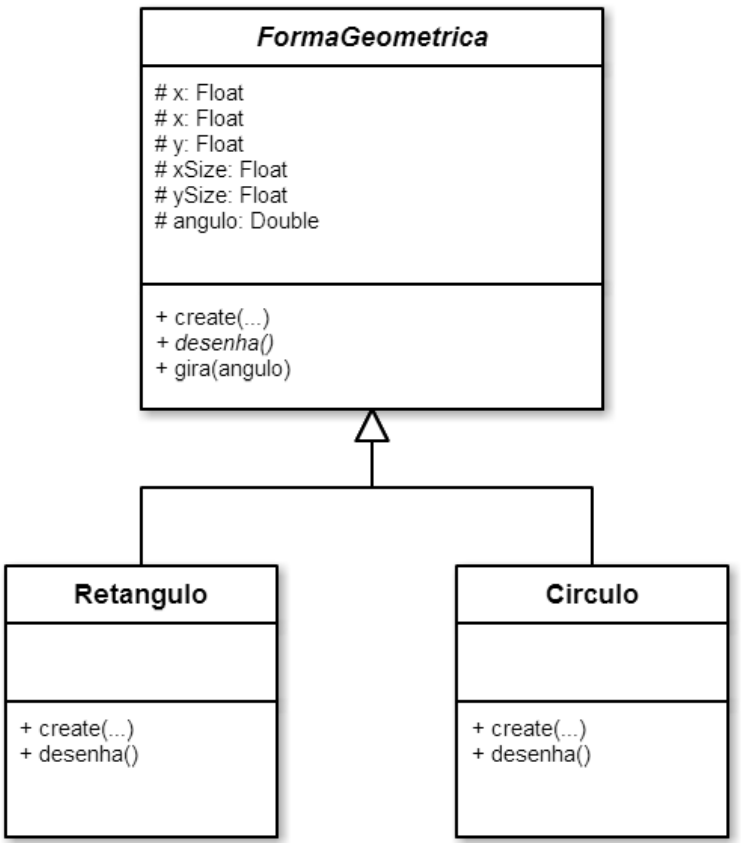
Classes abstratas

- São classes que não podem ser instanciadas diretamente
- As classes filhas implementam os **métodos abstratos** da classe mãe

Exemplos

- **Classe abstrata:** Forma geométrica; **Classes concretas**: Retângulo, Circulo, etc
- **Classe abstrata:** Funcionário; **Classes concretas:** Gerente, Secretário, etc

Exemplo Forma Geométrica



Será que é herança mesmo?

- O uso de herança **aumenta** o acoplamento entre as classes
- Usar apenas quando necessário, procurando alternativas como **composição** e **Interfaces**

Regra básica

- Se uma classe B estende A, então classe B **É-UMA** classe A;
- Exemplos:

Será que é herança mesmo?

- O uso de herança **aumenta** o acoplamento entre as classes
- Usar apenas quando necessário, procurando alternativas como **composição** e **Interfaces**

Regra básica

- Se uma classe B estende A, então classe B **É-UMA** classe A;
- Exemplos:
  - Quadrado é uma Forma Geometrica?

Será que é herança mesmo?

- O uso de herança **aumenta** o acoplamento entre as classes
- Usar apenas quando necessário, procurando alternativas como **composição** e **Interfaces**

Regra básica

- Se uma classe B estende A, então classe B **É-UMA** classe A;
- Exemplos:
  - Quadrado é uma Forma Geometrica? SIM!

Será que é herança mesmo?

- O uso de herança **aumenta** o acoplamento entre as classes
- Usar apenas quando necessário, procurando alternativas como **composição** e **Interfaces**

Regra básica

- Se uma classe B estende A, então classe B **É-UMA** classe A;
- Exemplos:
  - Quadrado é uma Forma Geometrica? SIM!
  - Lobo é um Animal?

Será que é herança mesmo?

- O uso de herança **aumenta** o acoplamento entre as classes
- Usar apenas quando necessário, procurando alternativas como **composição** e **Interfaces**

Regra básica

- Se uma classe B estende A, então classe B **É-UMA** classe A;
- Exemplos:
  - Quadrado é uma Forma Geometrica? SIM!
  - Lobo é um Animal? SIM!

Será que é herança mesmo?

- O uso de herança **aumenta** o acoplamento entre as classes
- Usar apenas quando necessário, procurando alternativas como **composição** e **Interfaces**

Regra básica

- Se uma classe B estende A, então classe B **É-UMA** classe A;
- Exemplos:
  - Quadrado é uma Forma Geometrica? SIM!
  - Lobo é um Animal? SIM!
  - Banheira é um Banheiro?

Será que é herança mesmo?

- O uso de herança **aumenta** o acoplamento entre as classes
- Usar apenas quando necessário, procurando alternativas como **composição** e **Interfaces**

Regra básica

- Se uma classe B estende A, então classe B **É-UMA** classe A;
- Exemplos:
  - Quadrado é uma Forma Geometrica? SIM!
  - Lobo é um Animal? SIM!
  - Banheira é um Banheiro? NÃO!

Será que é herança mesmo?

- O uso de herança **aumenta** o acoplamento entre as classes
- Usar apenas quando necessário, procurando alternativas como **composição** e **Interfaces**

Regra básica

- Se uma classe B estende A, então classe B **É-UMA** classe A;
- Exemplos:
  - Quadrado é uma Forma Geometrica? SIM!
  - Lobo é um Animal? SIM!
  - Banheira é um Banheiro? NÃO! Banheiro TEM-UMA Banheira (use agregação / composição)

Será que é herança mesmo?

- O uso de herança **aumenta** o acoplamento entre as classes
- Usar apenas quando necessário, procurando alternativas como **composição** e **Interfaces**

Regra básica

- Se uma classe B estende A, então classe B **É-UMA** classe A;
- Exemplos:
  - Quadrado é uma Forma Geometrica? SIM!
  - Lobo é um Animal? SIM!
  - Banheira é um Banheiro? NÃO! Banheiro TEM-UMA Banheira (use agregação / composição)
  - Pilha é uma Lista?

Será que é herança mesmo?

- O uso de herança **aumenta** o acoplamento entre as classes
- Usar apenas quando necessário, procurando alternativas como **composição** e **Interfaces**

Regra básica

- Se uma classe B estende A, então classe B **É-UMA** classe A;
- Exemplos:
  - Quadrado é uma Forma Geometrica? SIM!
  - Lobo é um Animal? SIM!
  - Banheira é um Banheiro? NÃO! Banheiro TEM-UMA Banheira (use agregação / composição)
  - Pilha é uma Lista? NÃO!

Será que é herança mesmo?

- O uso de herança **aumenta** o acoplamento entre as classes
- Usar apenas quando necessário, procurando alternativas como **composição** e **Interfaces**

Regra básica

- Se uma classe B estende A, então classe B **É-UMA** classe A;
- Exemplos:
  - Quadrado é uma Forma Geometrica? SIM!
  - Lobo é um Animal? SIM!
  - Banheira é um Banheiro? NÃO! Banheiro TEM-UMA Banheira (use agregação / composição)
  - Pilha é uma Lista? NÃO! Pilha não tem métodos add() nem remove() (use composição)

Interfaces

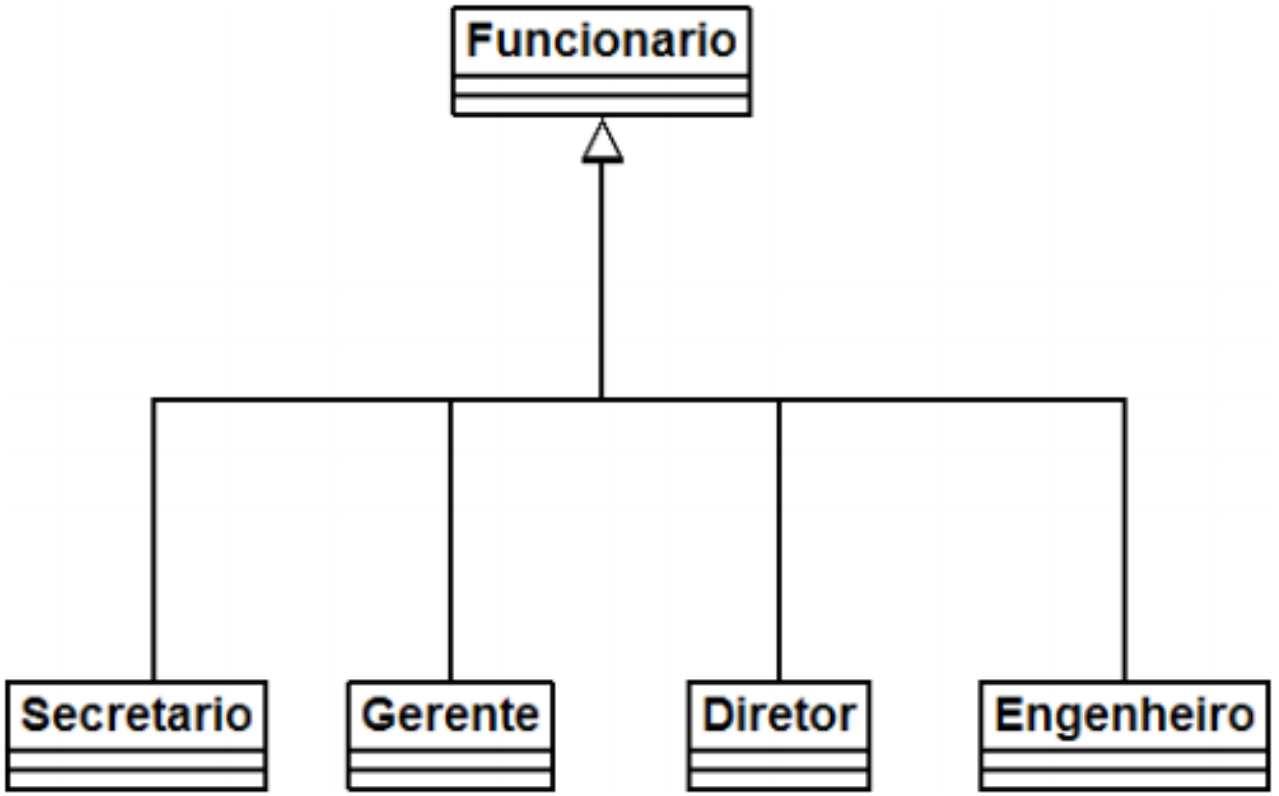
- Uma interface em Java define uma coleção de métodos públicos, sem definir a implementação
- Apenas expõe **o que o objeto deve fazer**, e não **como ele faz** ou **o que ele tem**

Exemplo

- Em Java, para ordenar uma lista usando `Collection.sort(lista)`, é necessário que os membros da lista implementem a interface `Comparable`.

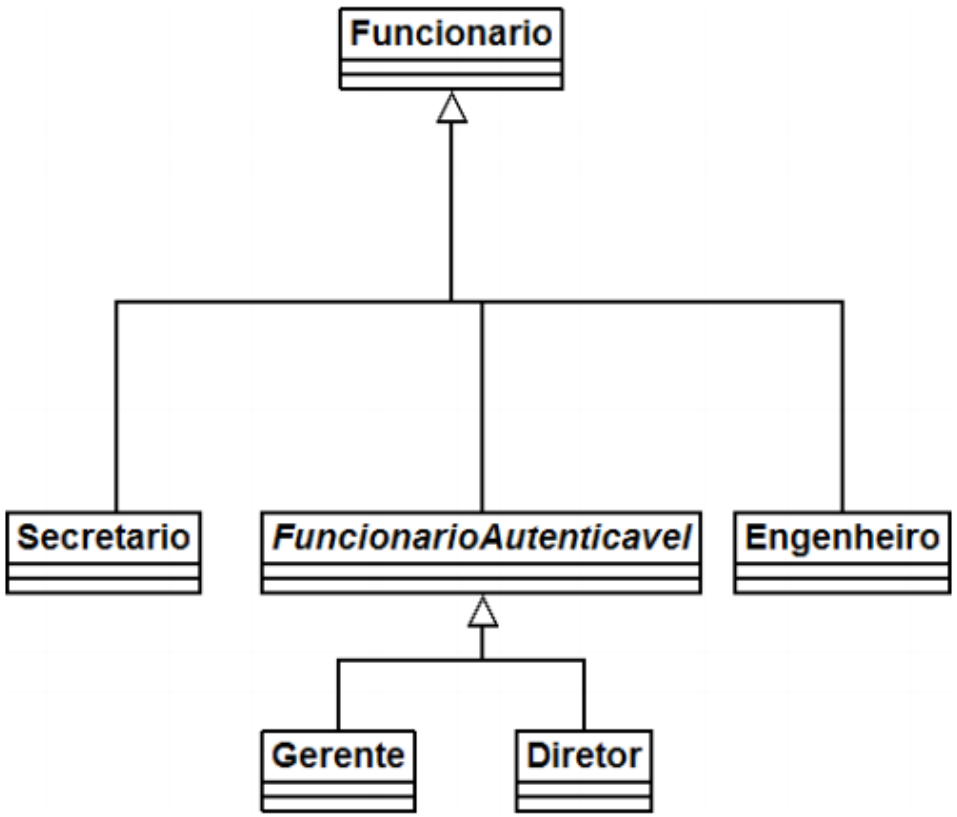
```
public interface Comparable<T> {
    int compareTo(T outro);
}
```

Exemplo Funcionários



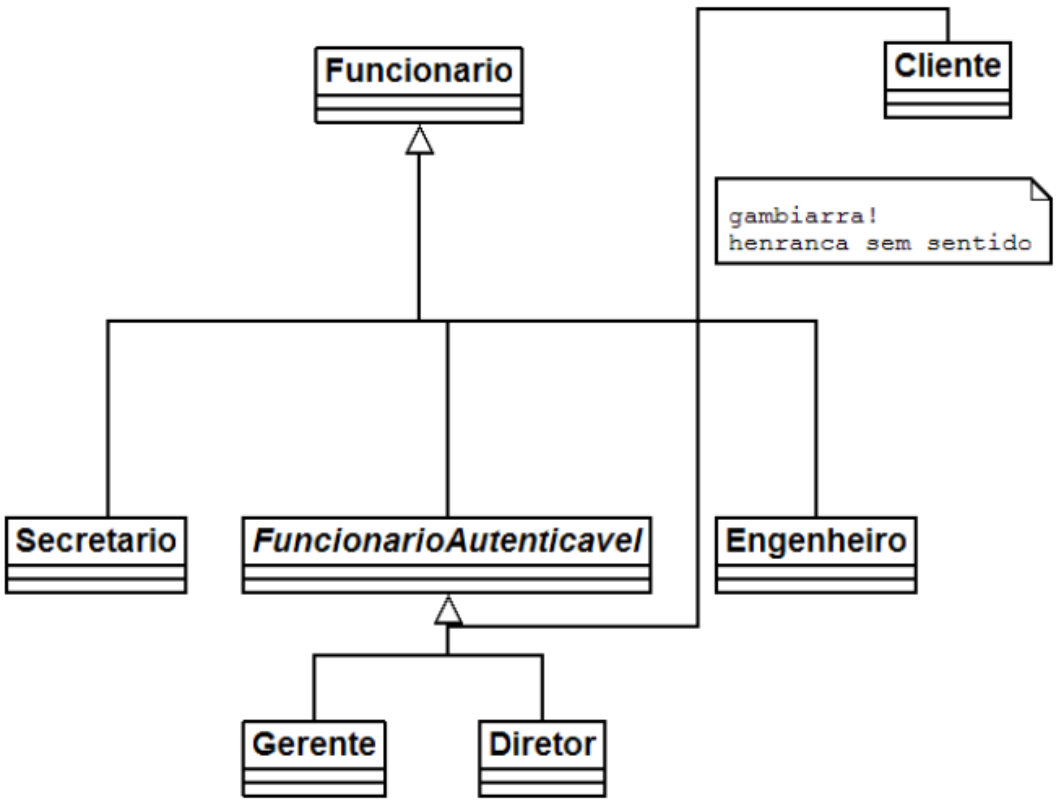
Exemplo Funcionários

Quero que alguns funcionários (Diretor e Gerente) consigam se autenticar no sistema interno.



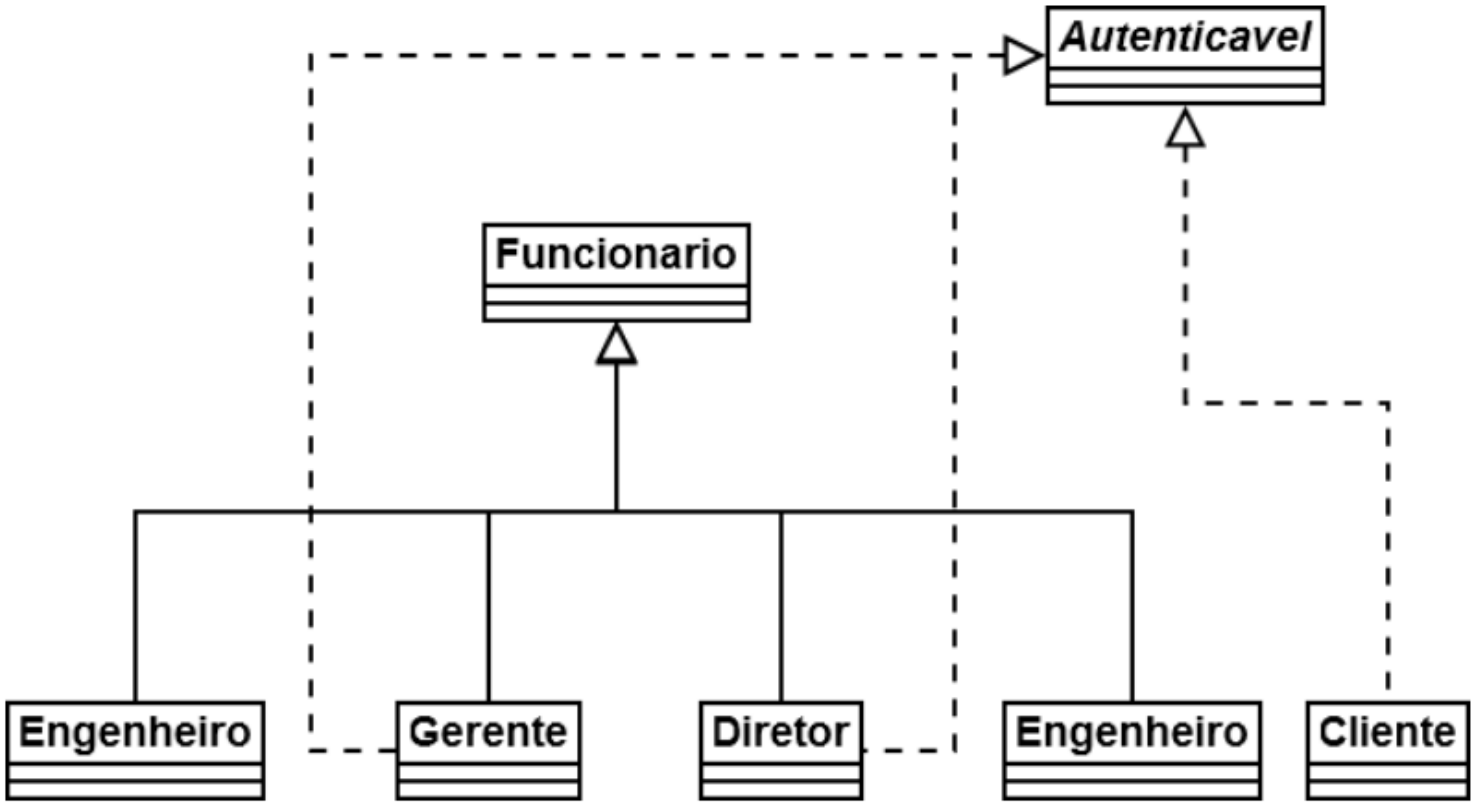
Exemplo Funcionários

Depois de um tempo, descobri que preciso que o cliente também possa se autenticar no sistema interno, então fiz a caca abaixo:



Exemplo Funcionários

Depois de refatorar meu sistema usando interfaces, eliminei as gambiarras e deixei o sistema menos acoplado e mais manutenível. (Um protótipo do código estará nos exemplos.)



Referências Bibliográficas

- Arlow J., Neustadt I. UML and the Unified Process - Practical Object-Oriented Analysis and Design. Pearson. 2002
- Caelum. Java e Orientação a Objetos. Disponível em < http://www.caelum.com.br/curso/fj-11-java-orientacao-objetos>
- Sierra K., Bates B. Use a Cabeça! - Java. 2ª Edição. O'Reilly Media. 2005