Tecnologia de Objetos

Conceitos

3

Princípios

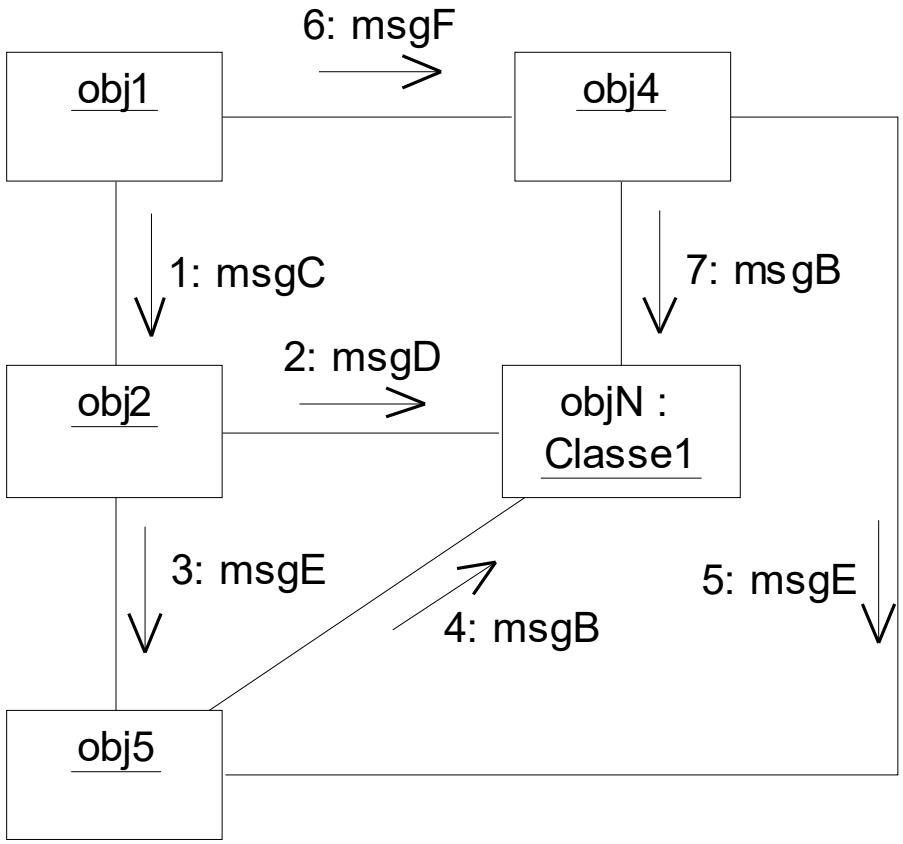
Modelagem

Fábio Nogueira de Lucena
fabio@inf.ufg.br

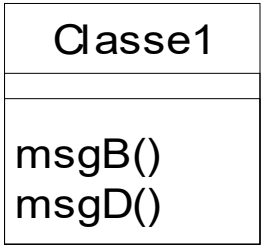
O universo é orientado a objetos.

Raio-X de software orientado a objetos

Coleção de objetos trocando mensagens entre eles



Cenário de troca de mensagens entre vários objetos



Classe do objeto **objN**

Detalhes de código Orientado a Objetos

Orientado a Objetos (OO)

- Abstração
- Classes
- Identidade de objeto
- Encapsulamento
- Herança
- Polimorfismo
- Persistência

Características geralmente presentes

OO (classes)

Molde através do qual objetos são criados

Cliente
- nome : String
+ Cliente() + Cliente() + perfil()

```
public class Cliente {

    private String nome; // Nome do cliente

    public Cliente() { nome = new String("Nome não fornecido"); }
    public Cliente(String sn) { nome = sn; }

    public void perfil() { System.out.println(nome); }
}
```

```
Cliente c1 = new Cliente("Euclides da Cunha");
Cliente c2 = new Cliente();

c1.perfil();
c2.perfil();
```

Mensagens enviadas aos objetos c1 e c2

OO (identidade de objeto)

Telefone

: Telefone

: Telefone

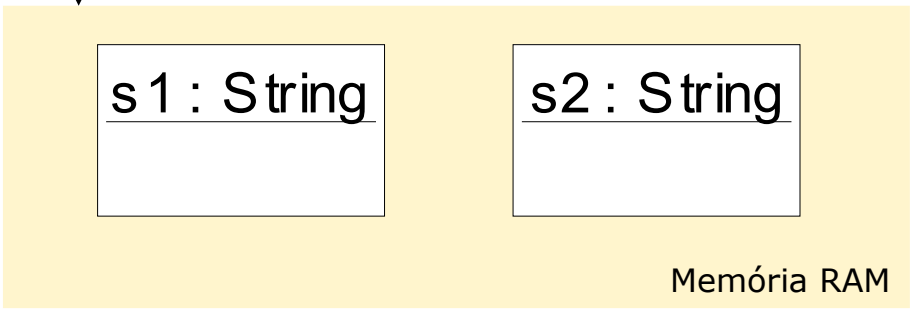
Instâncias distintas
Mesmo estado

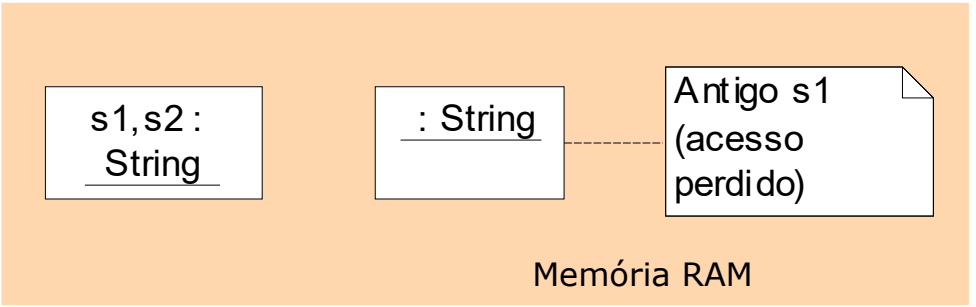
Suponha que telefone não possua um identificador único

OO (identidade de objeto)

- Todo objeto possui uma referência (*handle*)
  - Um único handle acompanha a vida do objeto
  - Dois objetos distintos não possuem a mesma referência

```
String s1;        // Referência para objeto String
String s2;        // Referência denominada de s2

s1 = new String("Programa");   // Cria instância de String
s2 = new String("Identidade"); // s2 referencia um objeto String

s1 = s2; // Acesso ao objeto referenciado por s1 é perdido
```





OO (encapsulamento)

- União de operações e atributos em objeto
- Um objeto é acessível apenas através da interface fornecida pelo encapsulamento

Exemplo

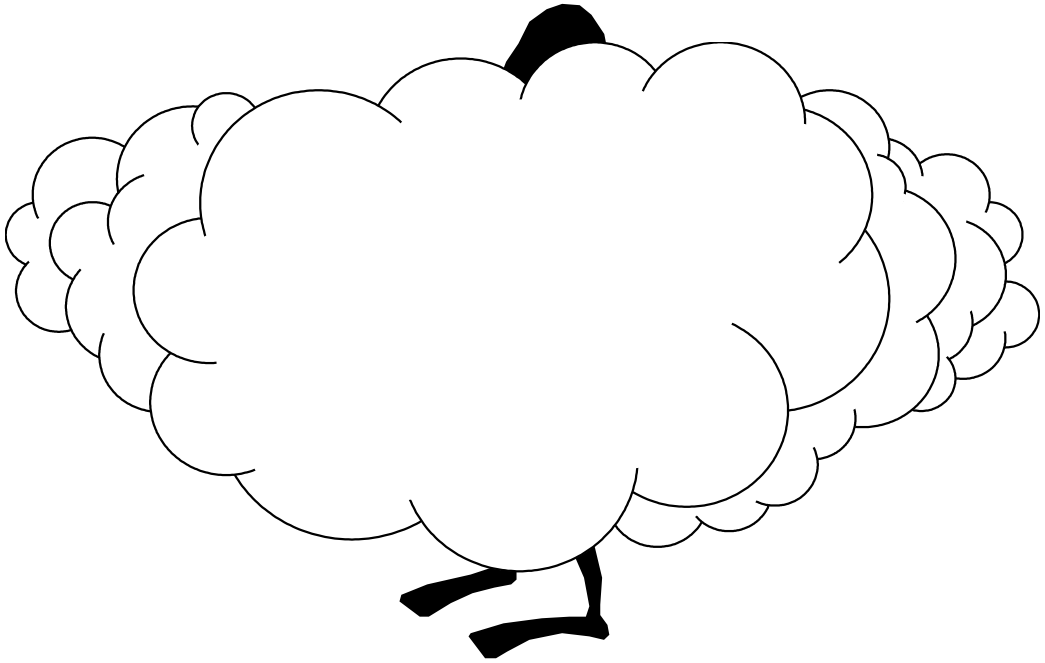
- Um telefone possui várias funções (realiza chamadas, toca, ...)
- Possui estado (p. ex.: ligado ou não)
- Permite o acesso às operações e estado através de interface bem definida

Encapsulamento

A composição é pública

Composição:
Cloridrato de Ranitidina (base), Celulose microcristalina, Fosfato de cálcio dibásico, glicolato de amido sódico, Dióxido de silício coloidal, Polietilenoglicol 6000 micronizado, Estearato de magnésio, Talco, Polímero do ácido acrílico, Dióxido de titânio, Polietilenoglicol 6000

Ocultamento de informação (censura)

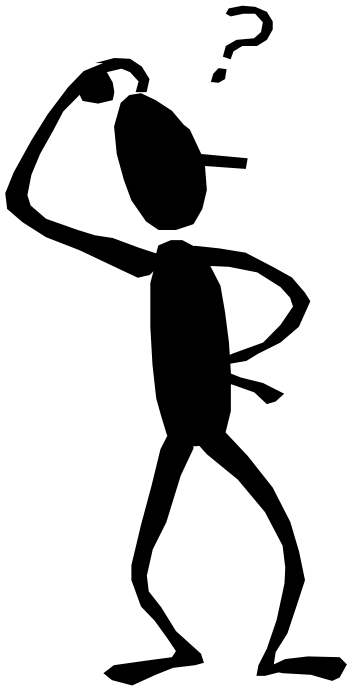


Não se sabe *quem*, *o que faz*, *o que veste, ...*
(protege ou oculta)

OO (ocultar informação)

- Permite esconder **como** um objeto realiza os serviços que oferece publicamente

Quem não sabe usar?



Quem sabe como funciona?

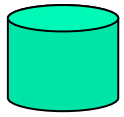
Interface

Access modifiers (Java):

- *Default (package)*
- `protected`
- `private`
- **`public`**

Evite quebrar o encapsulamento!

Encapsulamento e ocultamento



Estado

Comportamento

Comportamento

```
public class Pessoa {

    private String nome;
    private int anoNascimento;

    public Pessoa(String n, int an) {
        nome = n;
        anoNascimento = an;
    }

    public String toString() {
        return nome + " (" + anoNascimento + ")";
    }
}
class TestaPessoa {
    public static void main(String[] args) {
        Pessoa p = new Pessoa("Joao", 1975);
        System.out.println(p.toString());
    }
}
```

Information hiding

Interface pública

Código oculto!

Envio de mensagem

Implementação

Visível, pública, acessível

Invisível, privada, inacessível

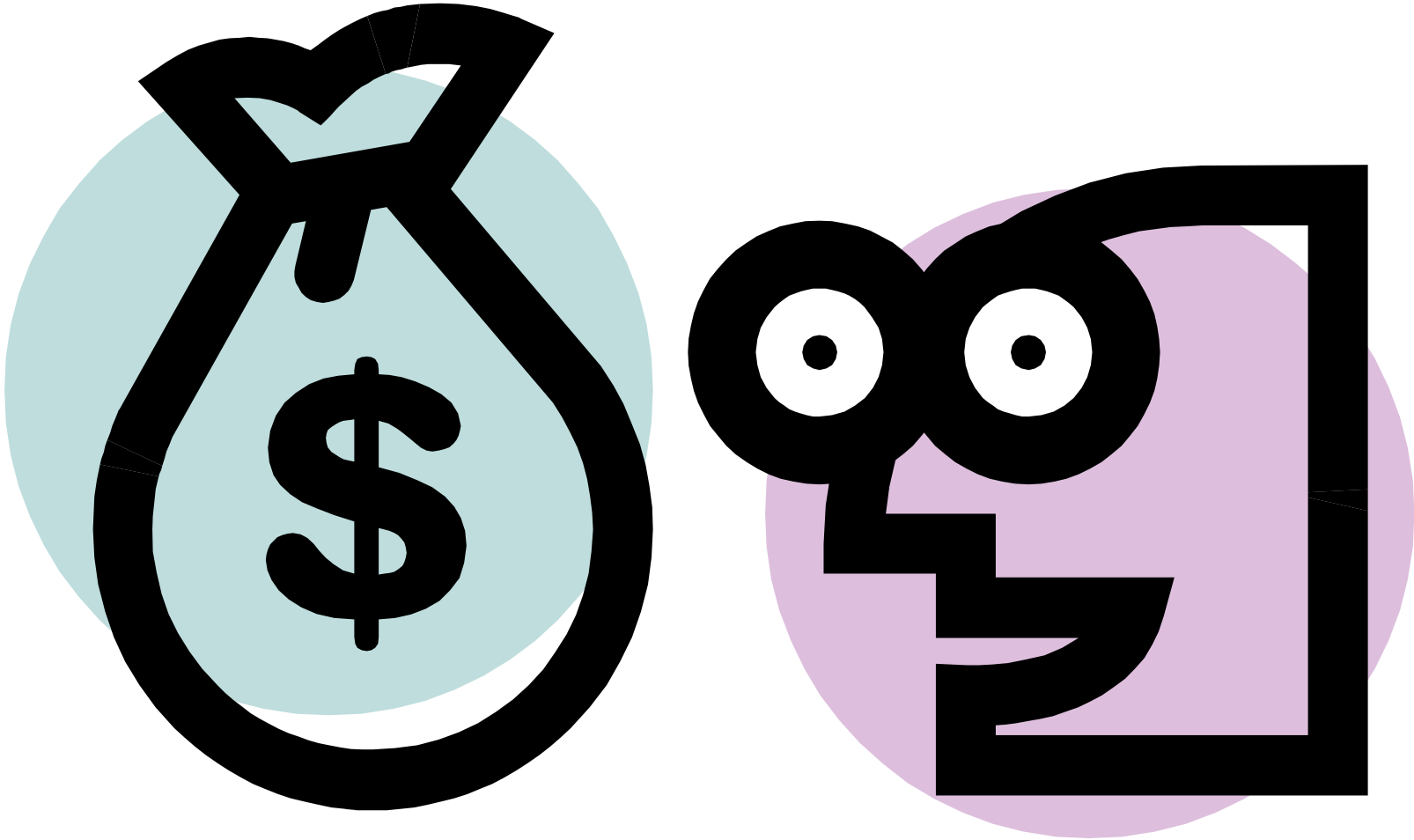
Interface

Cliente

Fronteira
(encapsulamento)

Implementação

Herança (uma interpretação)

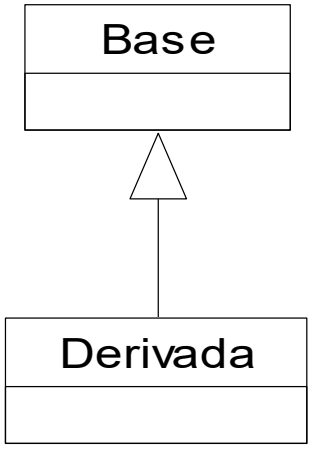


Herança (classes base e derivada)

- Uma nova classe é criada com base em uma classe existente.
- A classe que herda é chamada de derivada, a outra de base.

Superclasse, ascendente, ...

Subclasse, descendente, ...



INTERPRETAÇÃO

A classe `Derivada` herda da classe `Base`.

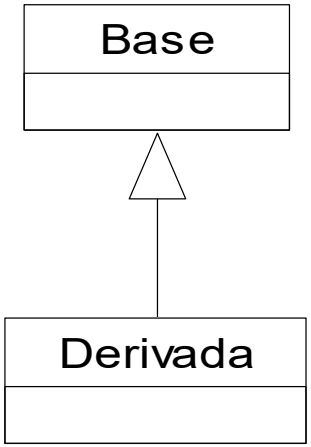

```
class Derivada extends Base {}
```

Visual Basic .NET

```
Class Derivada
      Inherits Base
End Class
```

Interpretação de herança

- Classe Derivada *herda* os atributos e comportamento de Base
- Classe Derivada pode estender e/ou refinar a classe Base



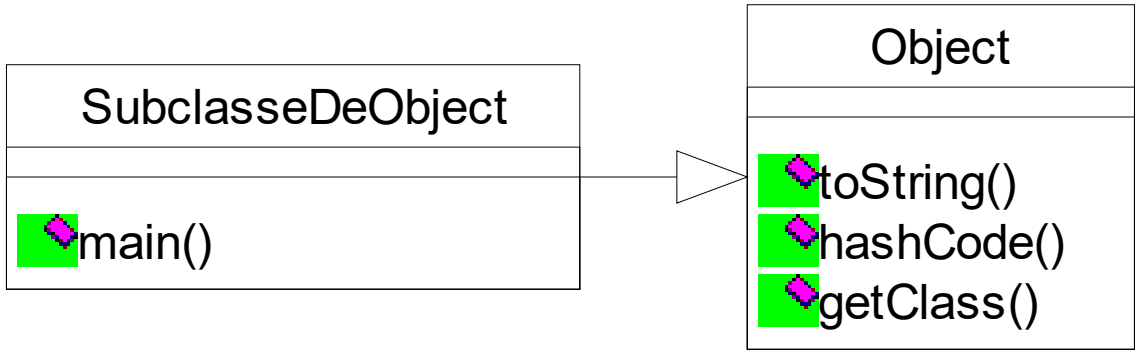
- *Atributos e comportamento são herdados.*
- *Novos atributos podem ser acrescentados.*
- *Novos comportamentos podem ser acrescentados.*
- *Comportamentos podem ser sobrepostos*

Não se esqueça!

Instância da classe **`Derivada`** comporta-se como instância da classe **`Base`**

Comportamentos são herdados

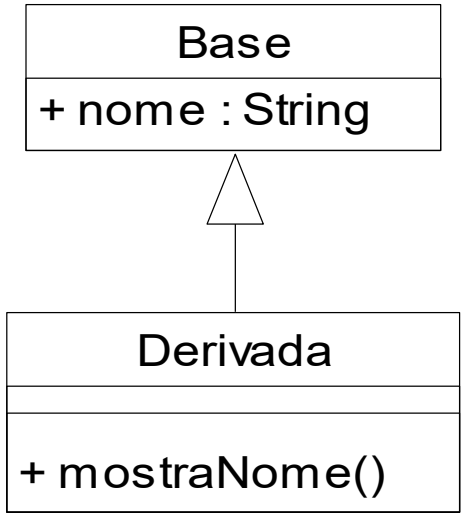
- Instância da subclasse comporta-se como instância da classe base




```
public class SubclasseDeObject extends Object {
    public static void main(String[] args) {
        SubclasseDeObject subObject = new SubclasseDeObject();
        Class c = subObject.getClass();
        int i = subObject.hashCode();
        String s = subObject.toString();
        System.out.println(c.getName() + " " + i + " " + s);
    }
}
```

Atributos são herdados e/ou criados

- A subclasse possui o atributo da classe base e acrescenta outro
- A subclasse adiciona um comportamento não presente na classe Base



Visual Basic .NET

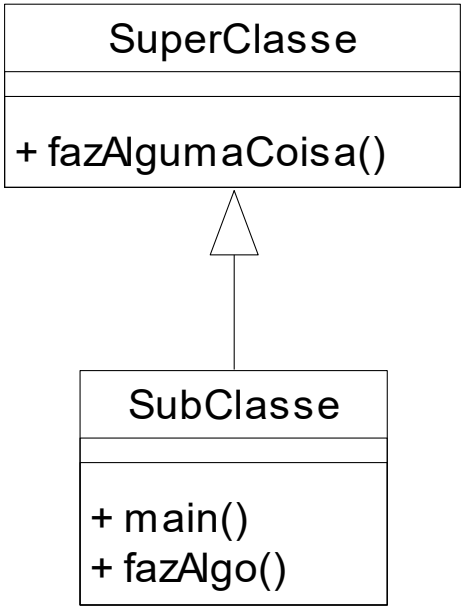
```
Public Class Base
    Public nome As String = "Base"
End Class

Public Class Derivada
    Inherits Base
    Private valor As Integer = 2

    Public Sub mostraNome()
        Console.WriteLine(nome)
    End Sub
End Class
```

Objeto de subclasse (comportamento)

- Instância de subclasse deve se comportar também como uma instância da superclasse.

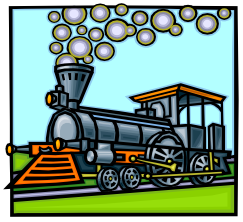
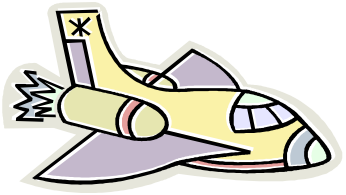
```
package heranca;

public class SuperClasse {
    public void fazAlgumaCoisa() {
        System.out.println("fazendo algo...");
    }
}
```

```
package heranca;

public class SubClasse extends SuperClasse {

    public static void fazAlgo(SuperClasse refSuperClasse) {
        refSuperClasse.fazAlgumaCoisa();
    }

    public static void main(String[] args) {
        SubClasse refSubClasse = new SubClasse();
        fazAlgo(refSubClasse);
        refSubClasse.fazAlgumaCoisa();
    }
}
```

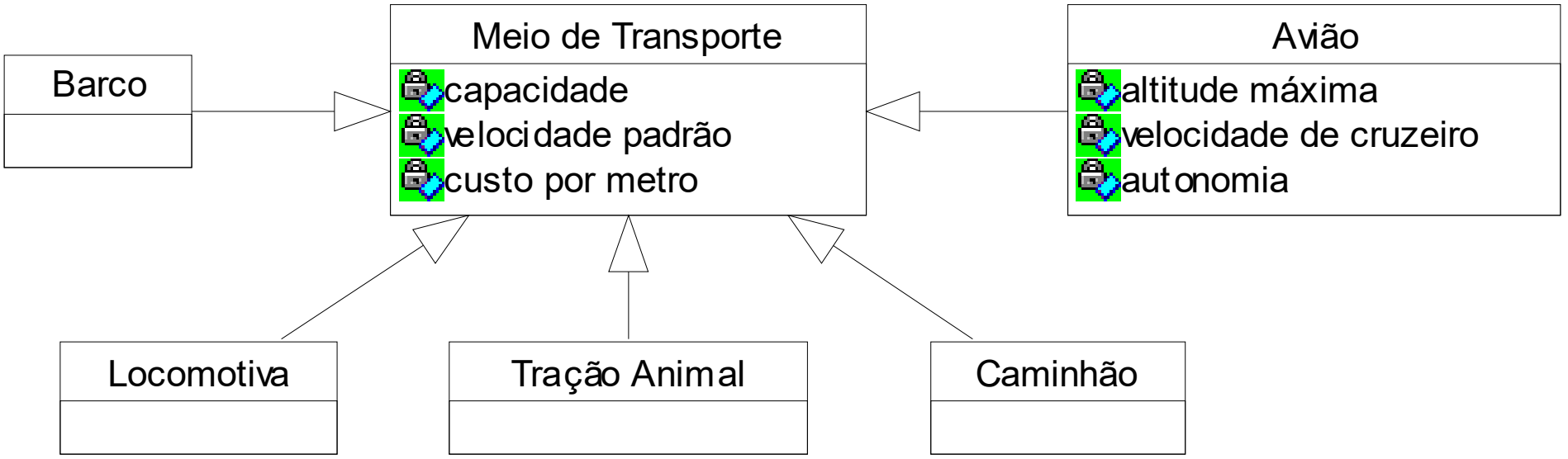


Herança

Meio de Transporte:
capacidade
velocidade padrão
custo por metro


Herança (UML)

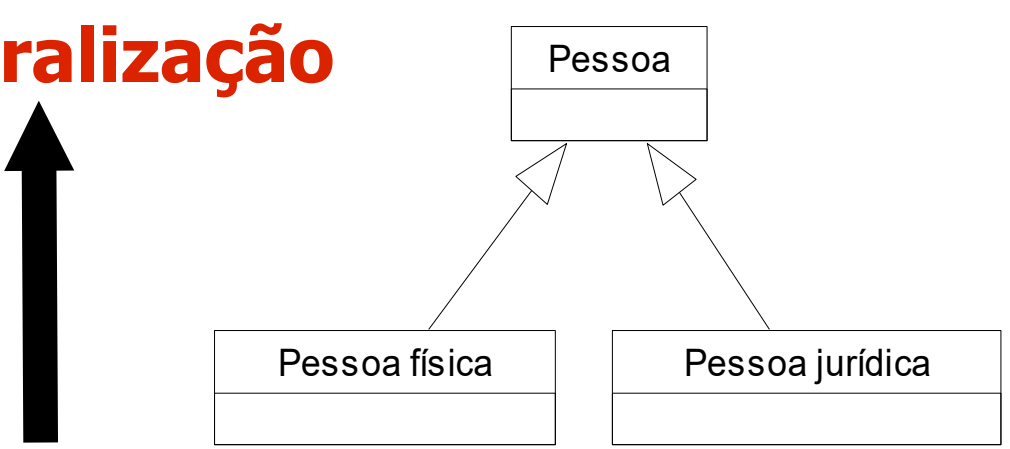



```
class Barco extends MeioDeTransporte {}
class Locomotiva extends MeioDeTransporte {}
class TracaoAnimal extends MeioDeTransporte {}
class Caminhao extends MeioDeTransporte {}
class Aviao extends MeioDeTransporte {}
```

OO (herança)

- `Pessoa` é generalização de `Pessoa física` e de `Pessoa jurídica`
- `Pessoa física` é uma especialização de `Pessoa`
- `Pessoa jurídica` é uma especialização de `Pessoa`

Generalização

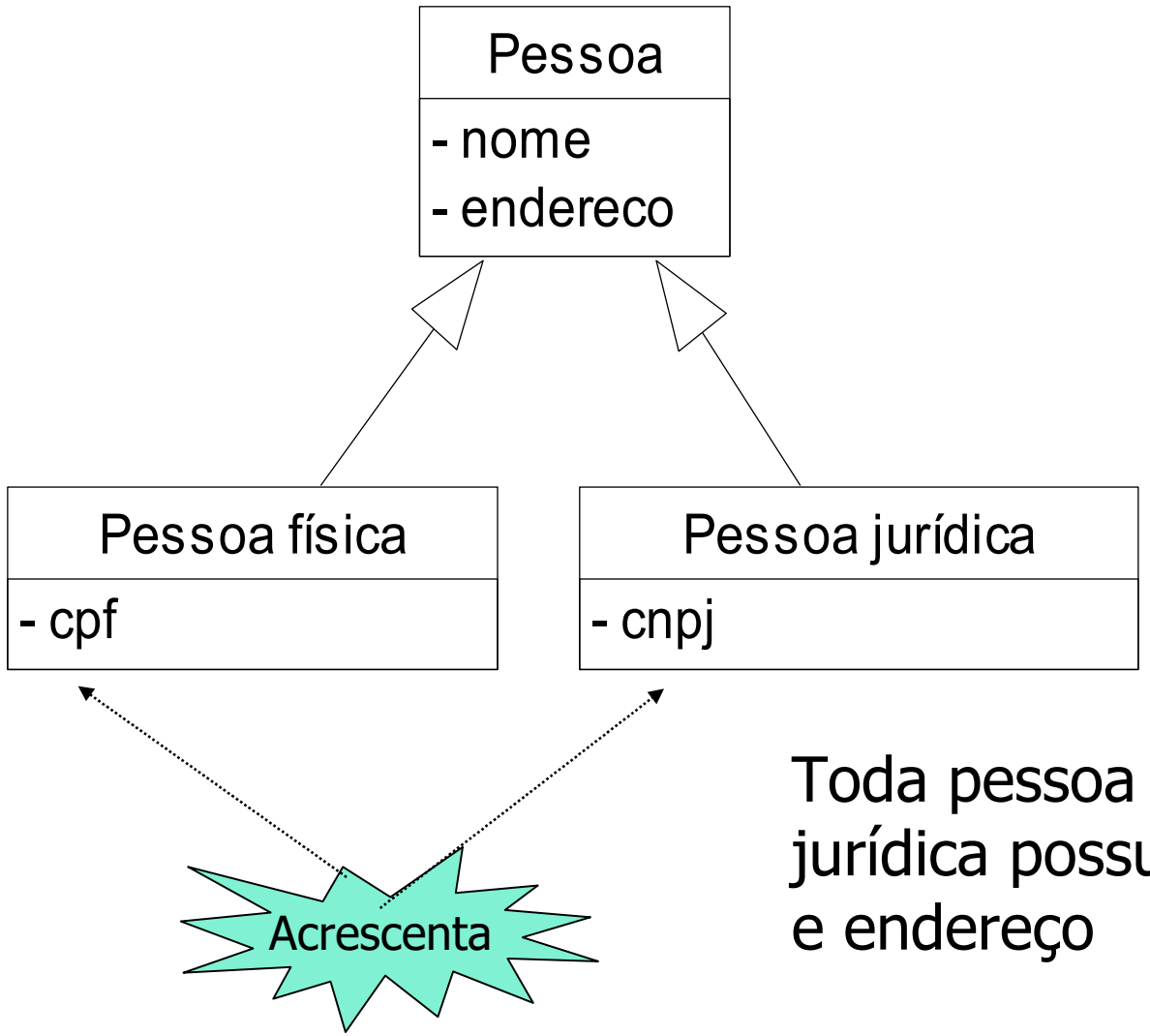


Especialização

Herança

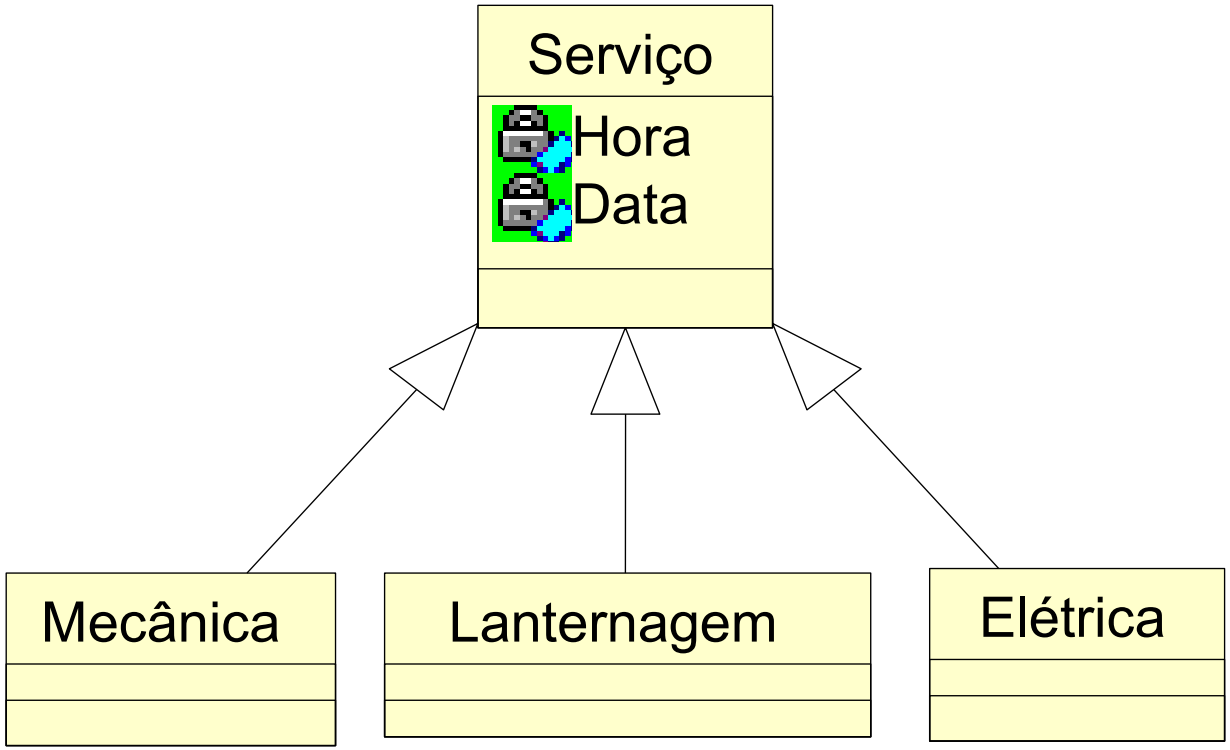
Superclasse (base)

Subclasses (derivada)



Toda pessoa física e jurídica possui nome e endereço

Herança (oficina mecânica)



Coleção de peças suspeitas e aquelas consertadas.

Componentes suspeitos e aqueles consertados.

Partes que receberão retoque.

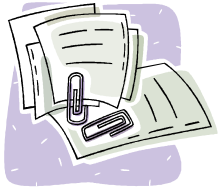
Heranças adequadas e inadequadas

- Polígono herda de ponto
- Quadrado herda de retângulo
- Pessoa herda de animal
- Computador herda de máquina
- Classe herda de aluno
- Voz herda de som
- Telefone herda de Comunicação
- Infância herda de FaseDaVida
- Maguila herda de Lutador

Outro elemento OO muito útil!

- Separe o que está bom e jogue fora o resto.
- Calculo os créditos e débitos destes documentos (Notas fiscais, promissórias, recibos, ...)
- Quais os compromissos noturnos que tenho?
- Toque de recolher afeta pessoas, negócios, ...
- Imposto de renda se aplica a PFs, PJs, PFs aposentadas, ...

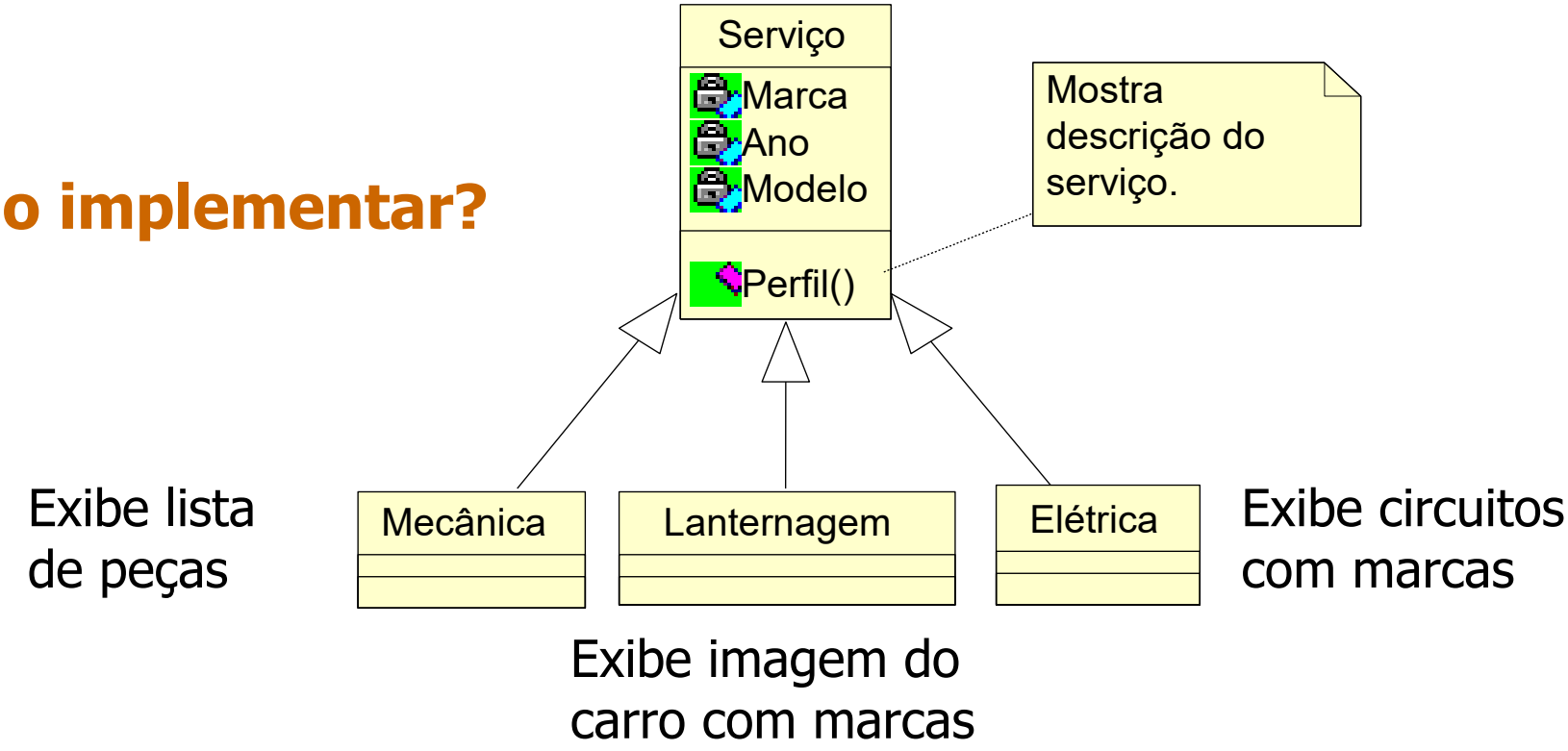
O que está subjacente?



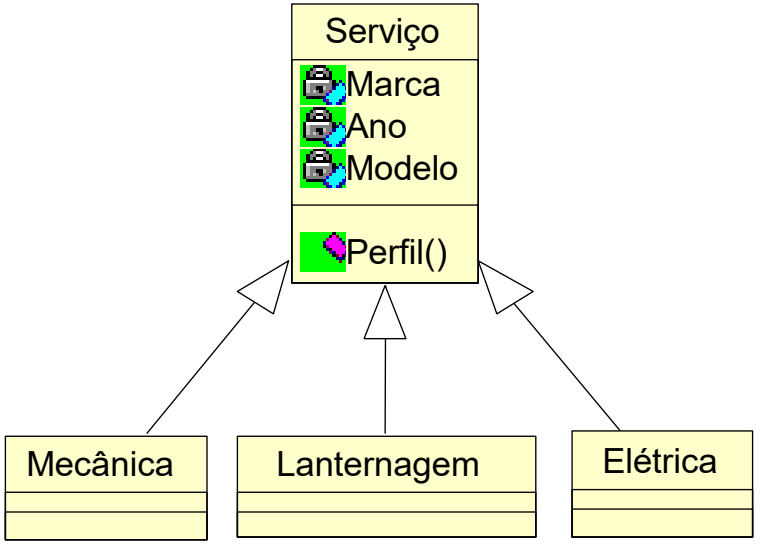
OO (polimorfismo)

- "Assume muitas formas"
- Mecanismo no qual uma operação possui implementações distintas em classes distintas

Como implementar?



OO (polimorfismo)




```
public class teste {

  public static void main (String[] args)
  {
      Mecanica m = new Mecanica();
      Lanternagem l = new Lanternagem();
      Eletrica e = new Eletrica();

      Servico s[] = new Servico[3];
      s[0] = m;
      s[1] = l;
      s[2] = e;

      for (int i = 0; i < s.length; i++)
          s[i].perfil();
  }
}
```

A mesma mensagem enviada para uma variedade de objetos terá "várias formas" de resultados – isto é *polimorfismo.* [*Java How to Program*]

OO (persistência)

- Um objeto retém seu estado

```
public class teste {
    public static void main (String[] args) {
        Persistencia p = new Persistencia();  // Cria objeto
        p.atribuiValor(-45);  // Altera valor através da interface

Δt > 0  // O valor irá persistir tanto quanto queiramos
        System.out.println(p.obtemValor());  // Imprime valor
    }
}
```

```
public class Persistencia {
    public int valor; // Variável que retém um valor inteiro

    public Persistencia() { valor = 0; } // Construtor

    // Apenas valores >=0 são permitidos
    public void atribuiValor (int nv) { valor = (nv < 0) ? 0 : nv; }

    public int obtemValor() { return valor; }
}
```

Interface

- Tipo abstrato que identifica serviços que uma classe deve implementar

```
public interface Runnable {
    public void run();
}
```

```
class FazAlgo implements Runnable {
        public void run() { System.out.println("Fazendo algo...");
}

public class Programa {
        public static void main(String[] args) {
                new Thread(new FazAlgo()).start();
                Runnable r = new FazAlgo();
                r.run();
        }
}
```

Tecnologia de Objetos

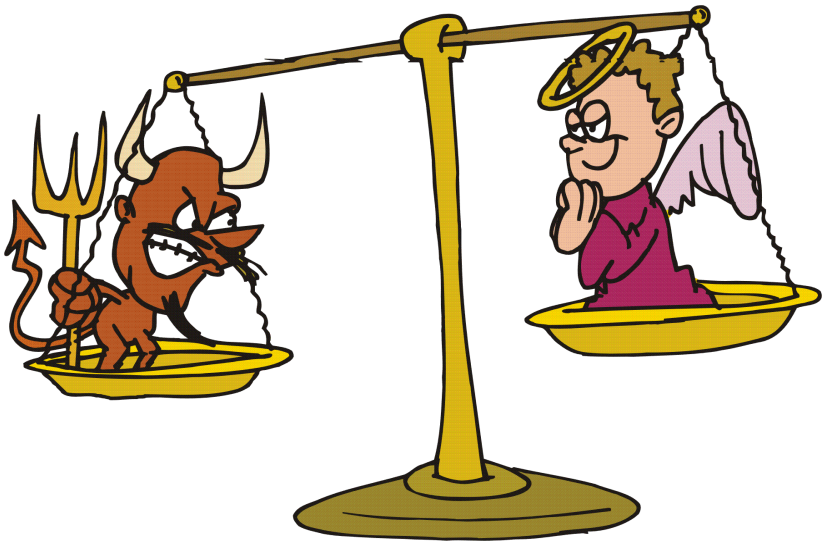
Como empregar?

Usar OO exige método

- Apropriado às necessidades
- Métodos OO não são mais fáceis

Alguns métodos OO:

- RUP
- Coad & Yourdon
- OMT
- Fusion
- Objectory
- OEP, Octopus, OOA/RD, OOBE, OOSE, OOSD, OOSC, OOram, OOHDM, ...

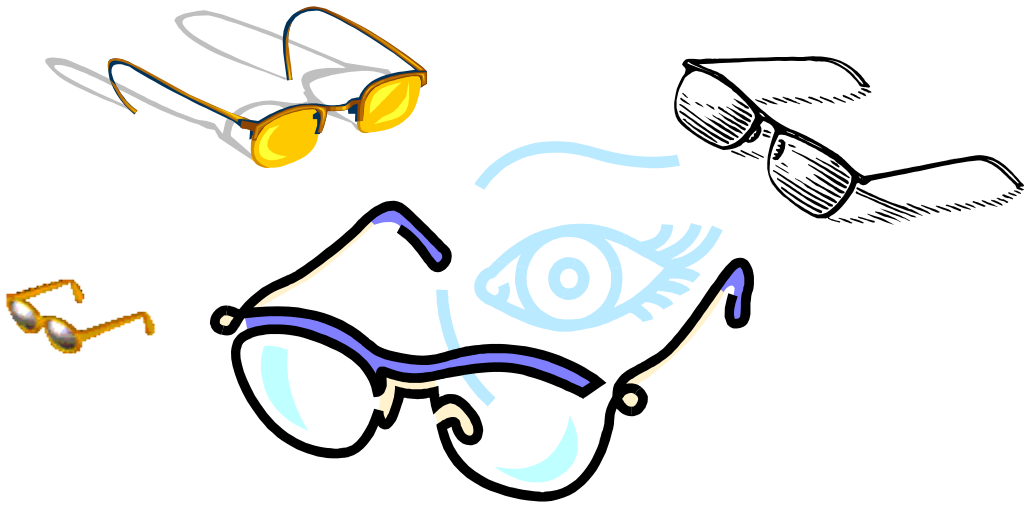


Nem sempre a escolha é fácil

Bibliografia – *Object-Oriented Analysis and Design: A Comparative Review,* Brinkkemper, S.

Como identificar objetos?

- Abstração
  relógio, lanterna, calculadora, ...
- Oferece serviços
  marca o tempo, ilumina, realiza cálculos, ...



Óculos OO
Ainda não é vendido!!!!

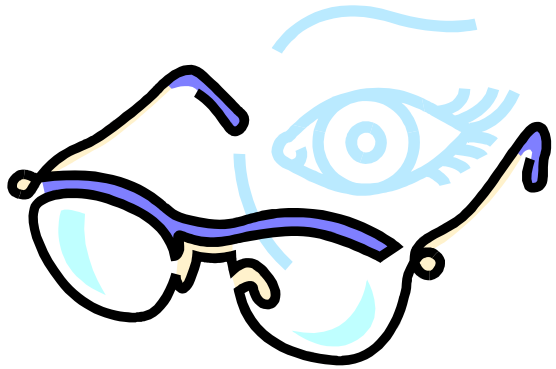
É preciso treinar a visão!

Exige pensar diferente!

- Para a orientação a objetos ser utilizada será necessário eliminar o fato de que a maioria dos praticantes não pensam em termos de objetos.

What it Takes to Make OO Work
Kozaczyinski, W. et al.
IEEE Software, jan/1993, pp 20-23.

É preciso "ver" o mundo recheado de objetos



Adquira o seu óculos OO!

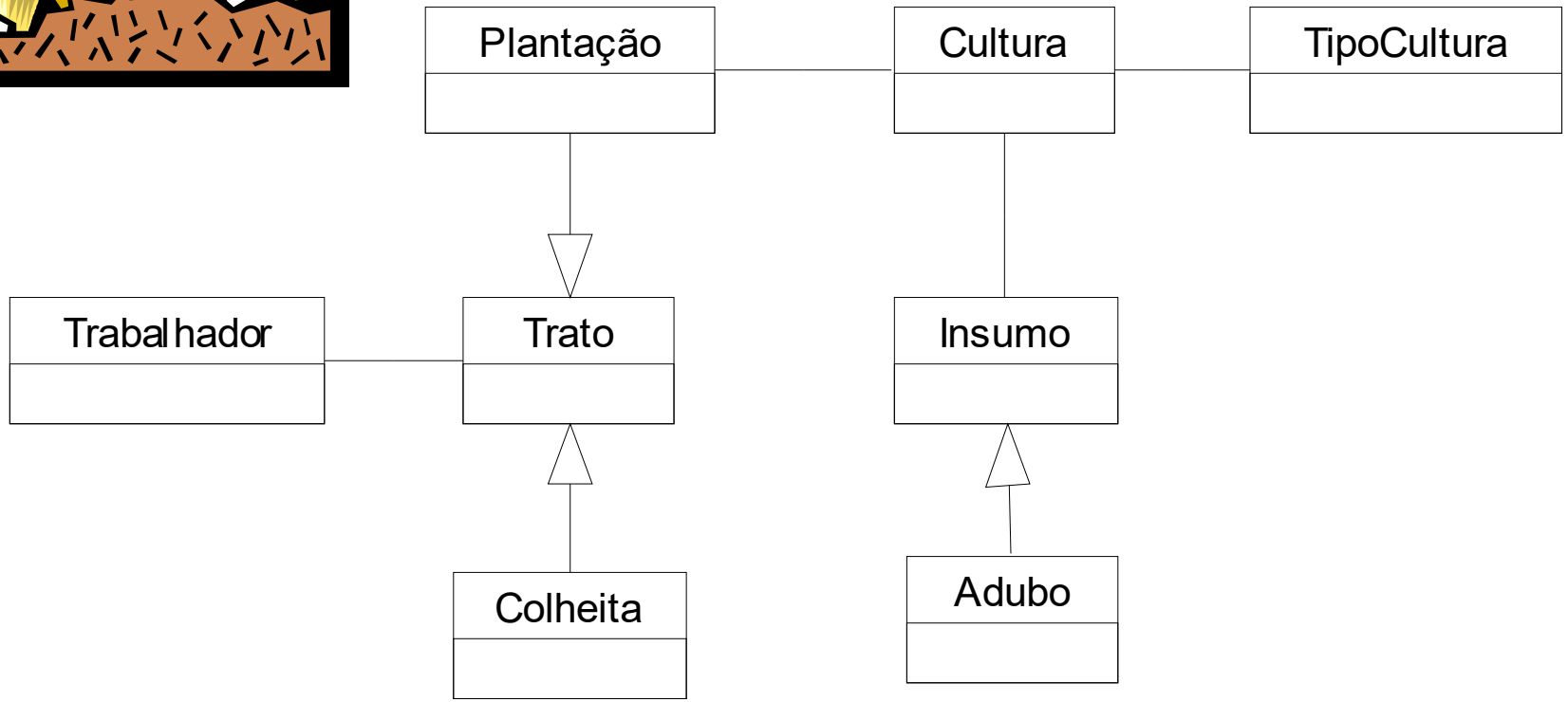
Visão "convencional"

Ênfase em processos

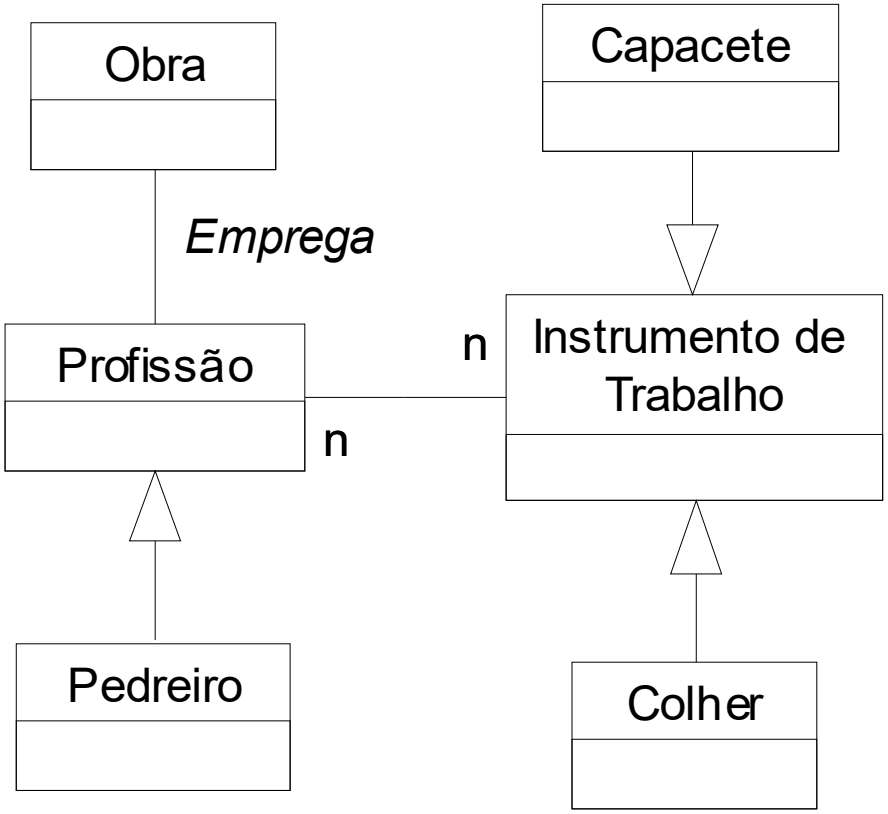
- `void plantar()`
- `void colher()`
- `void adubar()`
- `int nTrabalhadores()`
- `Time dataColheita()`
- `...`

Visão orientada a objetos

Ênfase em “entidades”

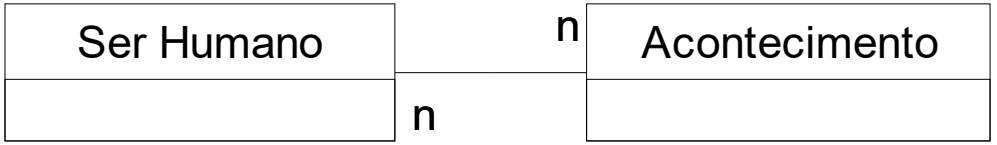


O que você vê? (I)

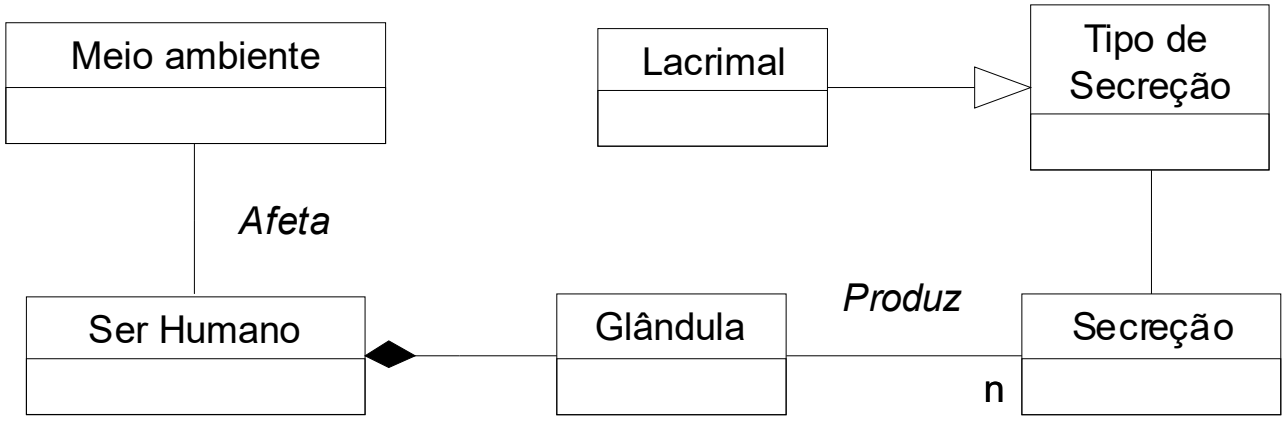


O que você vê? (II)

Especulação simples (evento desagradável)

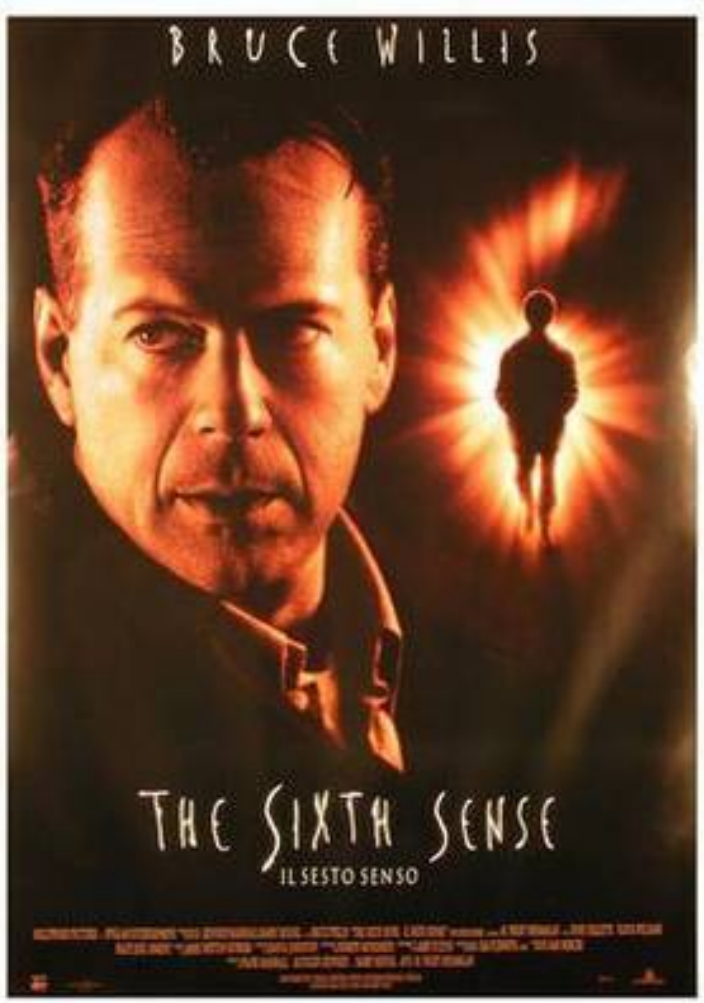


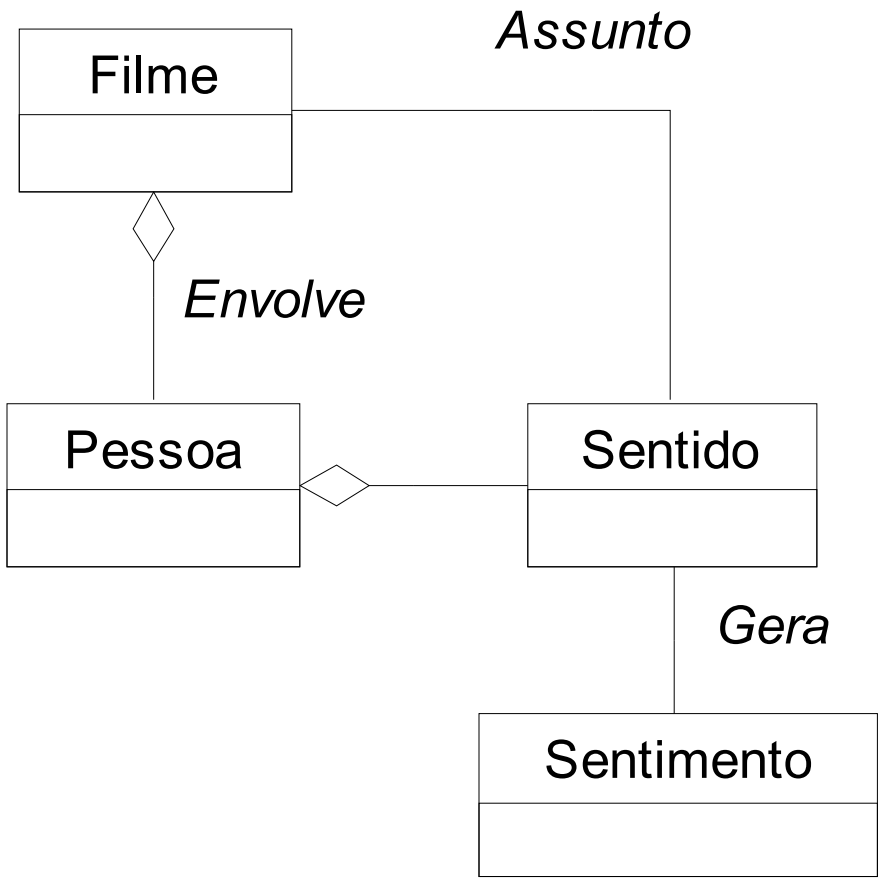
Divagação didática (vento conduz pó até os olhos deste desafortunado)



O que você vê? (III)

Fantasmas?

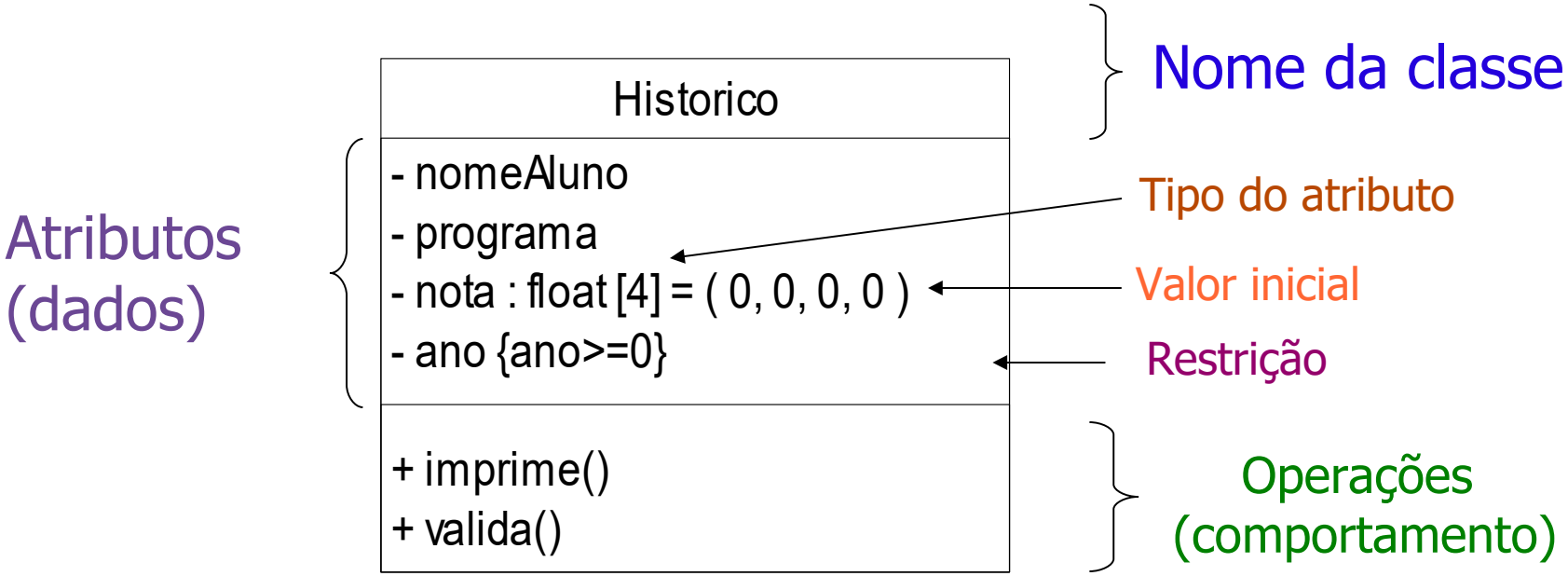




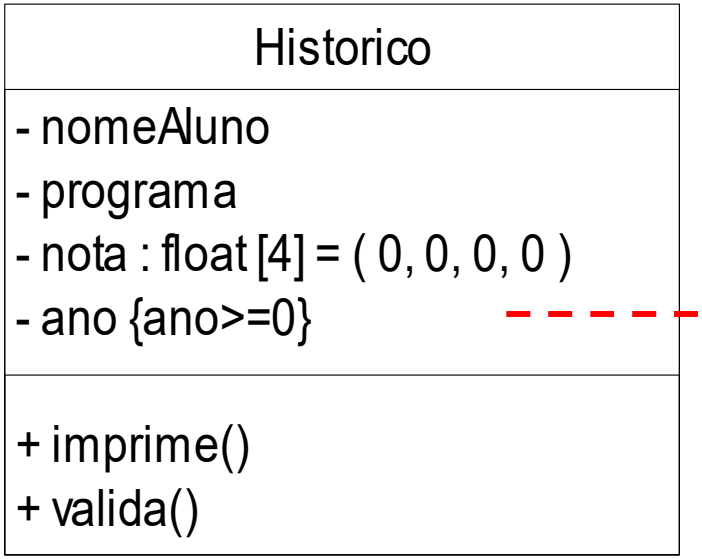
Classes, UML e Desenvolvimento Orientado a Objetos

Noções

Classe (detalhes segundo a UML)



Mapeamento entre UML e Java



?

```
public class Historico
{
   private String NomeAluno;
   private String Programa;
   private int[] Nota;
   private int ano;

   public void SetAno(int a) {
     ano = (a>=0) ? a : 0;
   }

   public void Imprime() {...}
   public void Valida() {...}
}
```

Projeto **Implementação**

Atividade de implementação

"Processo de Software"

- Para implementar é preciso

Implementação

  - Dominar programação e algoritmos
  - A estrutura do software

- Para organizar um software é preciso

Projeto

  - Dominar os princípios de projeto de software
  - Conhecer o que deverá ser oferecido ao cliente

- Para conhecer o que o cliente deseja
  - É preciso muita habilidade!
  - Envolver-se no negócio em questão

Análise

Análise, Projeto e Implementação OO

Análise (classificação)

Lâmpada

Projeto

Lampada
- ligada : Boolean
+ liga() + desliga() + estaLigada() : Boolean

Implementação

```
public class Lampada {
    private boolean ligada;
    public Lampada() { ligada = false; }
    public void liga() { ligada = true; }
    public void desliga() { ligada = false; }
    public boolean estaLigada() {
        return ligada;
    }

    public static void main (String[] args){
        Lampada l = new Lampada();
        l.liga();
        System.out.println(estaLigada()
                           ? "Ligada"
                           : "Desligada");
    }
}
```

Resumo

- "Principais recursos" da UML
  - Diagrama de classe e diagrama de seqüência
- Como tudo, OO tem vantagens e dificuldades
- Pensar no mundo orientado a objeto exige treino
- Características de OO
  - Identidade de objeto, encapsulamento, herança, ...
- Noção de desenvolvimento OO