CIÊNCIA DA COMPUTAÇÃO
UENF

Paradigma Orientado a Objetos
Conceitos Básicos

Prof. Ausberto S. Castro Vera
ascv@computer.org

Referências

Tecnologia Orientada a Objetos

- **Conceitos Básicos Objetc-Oriented (OO)**
  - **Fundamentos**
- **Paradigma Orientado a Objetos**
  - **Programação**
  - **Desenvolvimento**

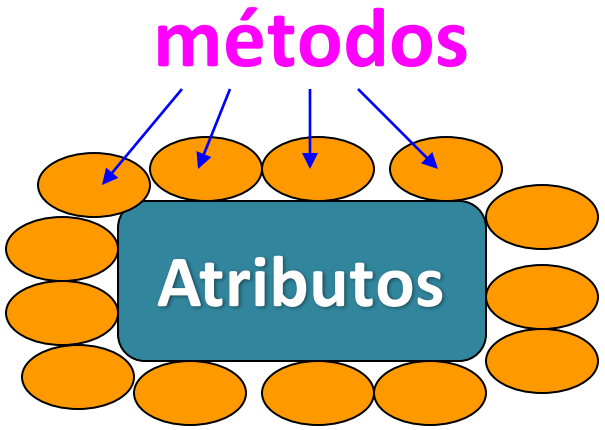
- **Desenvolvimento Orientado a Objetos**
  - **Análise Orientada a Objetos**
  - **Projeto Orientado a Objetos**
  - **Programação Orientada a Objetos**

Desenvolvimento OO

- **OOA - *Análise Orientado a Objetos*: desenvolvimento de um modelo orientado a objetos do domínio da aplicação. Os objetos podem ou não ser transformados em objetos do sistema.**
- **OOD - *Projeto Orientado a Objetos*: desenvolvimento de um modelo orientado a objetos de um sistema de software que implemente os requisitos identificados.**
- **OOP - *Programação Orientada a Objetos*: implementando um projeto de software usando uma linguagem de programação orientada a objetos.**
  - **OOA - Object-Oriented Analysis**
  - **OOD - Object-Oriented Design**
  - **OOP - Object-Oriented Programming**

Paradigma Orientado a Objetos

- **Paradigma OO: Baseado nos conceitos de *objeto* e *classe de objetos***
  - **Um *objeto* é uma variável (estrutura) junto com um conjunto de operações.**
- **Elementos do Paradigma**
  - **Objetos e classes**
  - **Métodos (mensagens, operações)**
  - **Herança**
  - **Polimorfismo**
  - **Encapsulamento**
  - **Tipos de Dados Abstratos (TDA)**



Linguagens O-O

- **Desenvolvidos para O-O**
  - **Smalltalk, Trellis, Eiffel, Oberon, Thetha, Self, Beta, Java**
- **Derivados/Híbridos**
  - **Modula-3, Modula-2 + OO extensões**
  - **Objective C, C++, C#**
  - **Ada 95,**
  - **Object Pascal, Turbo Pascal, Delphi**
  - **OOCobol**
- **Integrados com paradigma Funcional**
  - **Flavors, CLOS**
- **Integrados com Paradigma Concorrente**
  - **Pool, Actors, ABCL, Concurrent Smalltalk**

Historia O-O

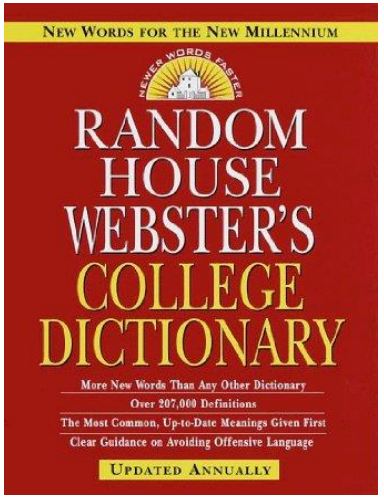
- **Pré-Objetos**
  - **Simula67 (Noruega, 1962-1968), TDAs**
  - **Criadores: Ole-Johan Dahl, Kristen Nygaard**
- **Primeira linguagem OO**
  - **Smalltalk (Xerox, 71,72,76, 80)**
  - **Criadores: Alan Kay, Dan Ingalls, Adele Goldberg**
- **C++**
  - **Bell Labs, 1983, ISO1998, 2003, C++11 (2011)**
  - **Bjarne Stroustrup**
- **Java**
  - **Sun Microsystems, James Gosling, 1991-92 (oak), 1995 (Java)**

Outros Conceitos O-O

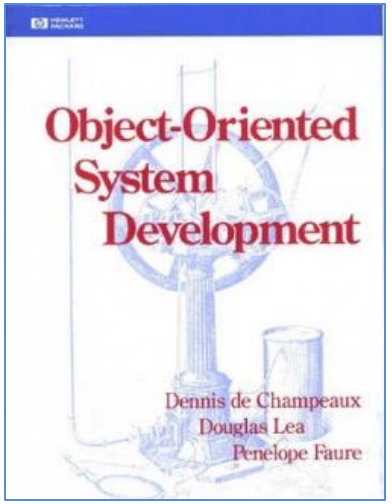
- **Vinculação estática e Dinâmica (Binding)**
- **Tipos Estáticos e Dinâmicos (Typing)**
- **Tipos e Encapsulamento**
  - **Tipos de Dados Abstratos (TDA)**
  - **Objetos encapsulados**
  - **Overloading/Polimorfismo**
  - **Coersões (conversão automática de um tipo para outro)**
  - **Genericidade**
  - **Subtipos**
- **Herança – Herança múltipla**
- **Garbage Collection**

Objeto

- **DEFINIÇÃO. Uma *coisa* visível ou tangível de forma estável e relativa; uma *coisa* que pode ser percebida intelectualmente; uma *coisa* para a qual o pensamento ou ação é direcionada**
  - Random House. *The Random House College Dictionary*. Random House, 1975



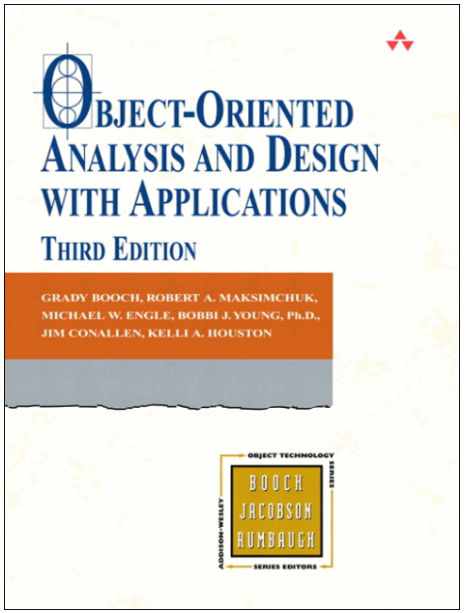
- **DEFINIÇÃO. Uma *entidade conceitual* que**
  - **é identificável**
  - **tem características espalhadas dentro de um espaço estado local**
  - **tem operações que podem mudar o status do sistema localmente**



Objeto

Um *objeto* tem identidade, estado e comportamento.
Um *objeto* é simplesmente uma entidade tangível que apresenta um comportamento bem definido. (Booch)

G. Booch et. al.
Object Oriented Design with Applications.
Pearson Education, 2007.

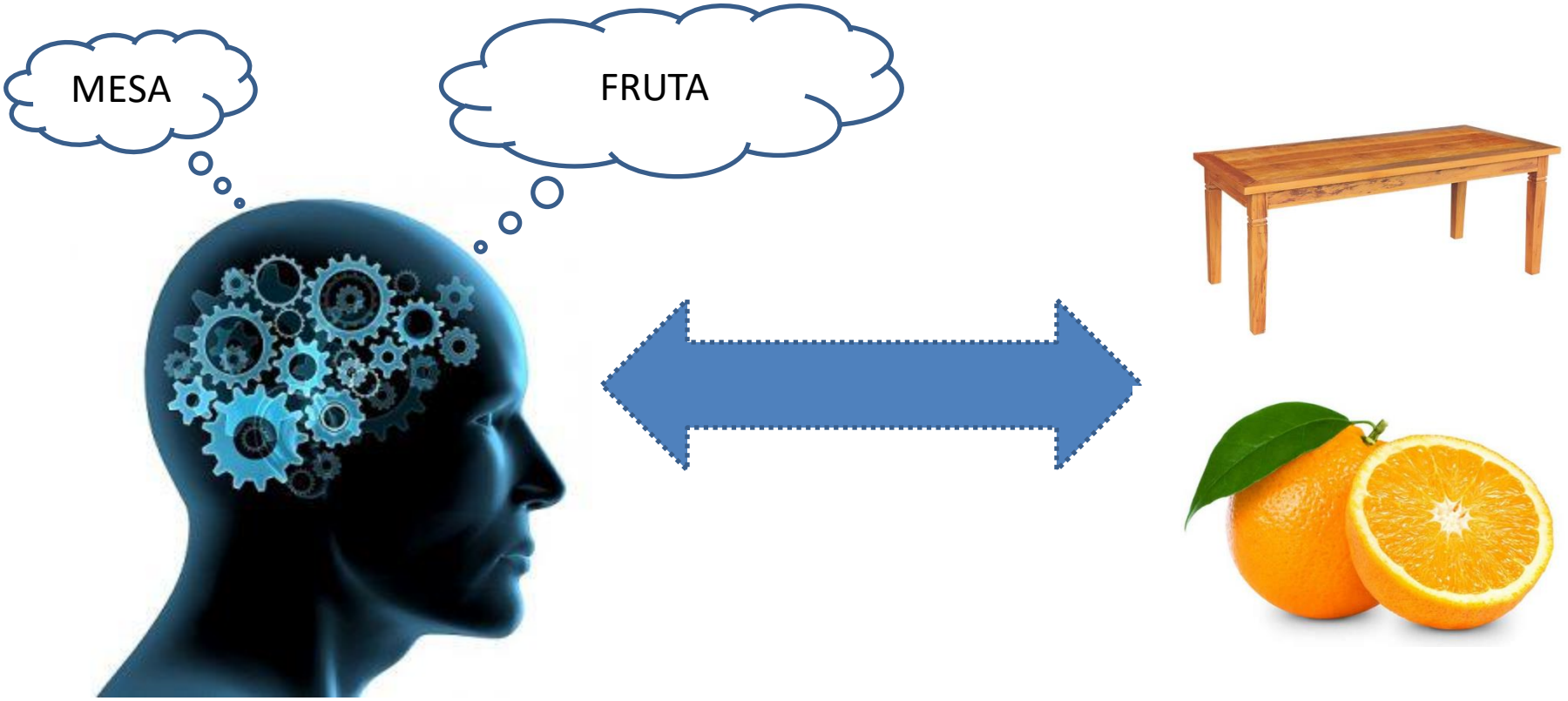


Um *objeto* é uma unidade de modularidade estrutural e comportamental que tem propriedades (Buhr)

R. Buhr. Machine charts for visual prototyping in system design. Technical Report 88-2, Carlton University, August 1988.

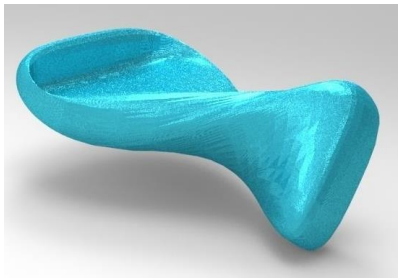
Objetos

- **Conceito**
  - ***Ideia* de algo que aplicamos às coisas**
    - **Laranja, mesa, arquivo, tabela, contador, tempo, salário, doutor**



Objetos

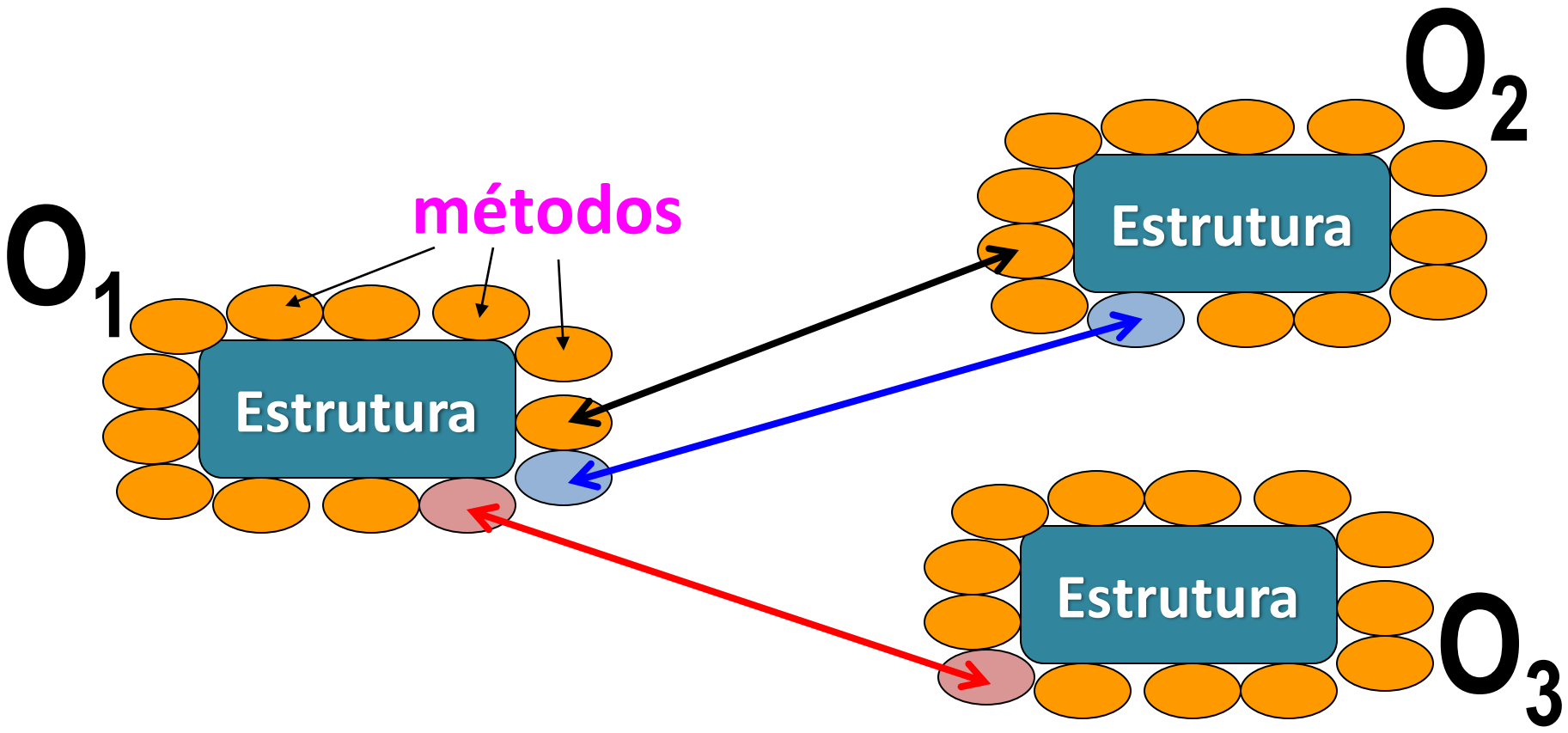
- **Objeto**
  - **É qualquer *entidade* a quem aplicamos um conceito**
    - **Laranja, mesa, arquivo, tabela, contador, tempo, salário, doutor**



FRUTA **MESA** **????**

Objetos

- **Objeto em Programação**
  - **Estrutura + comportamento**



Objetos: concretos e não concretos

Exemplos de objetos **CONCRETOS**

1) CANETA

Atributos de qualquer Caneta

- altura
- espessura
- cor

Nesse caso, o nosso objeto **Caneta** tem os seguintes dados em cada característica:

- altura: 10 cm
- espessura: 2 cm
- cor: rosa

Comportamentos de qualquer Caneta

- desenhar
- etc.

2) PESSOA

Atributos de qualquer Pessoa

- altura
- peso
- idade

Nesse caso, o nosso objeto **Pessoa** tem os seguintes dados em cada característica:

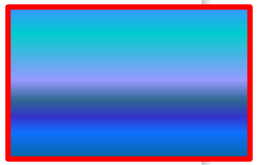
- altura: 1,80
- peso: 70
- idade: 25

Comportamentos de qualquer Pessoa

- andar
- sorrir
- ler
- etc.

Exemplos de objetos que **NÃO SÃO CONCRETOS**:

1) RETÂNGULO



Atributos de qualquer Retângulo

- base
- altura

Nesse caso, o nosso objeto **Retângulo** tem os seguintes dados em cada característica:

- base: 10
- altura: 5

Comportamentos de qualquer Retangulo

- calcular área
- calcular perímetro
- etc.

2) DISCIPLINA

Atributos de qualquer Disciplina

- código da disciplina
- nome da disciplina

Nesse caso, o nosso objeto **Disciplina** tem os seguintes dados em cada característica:

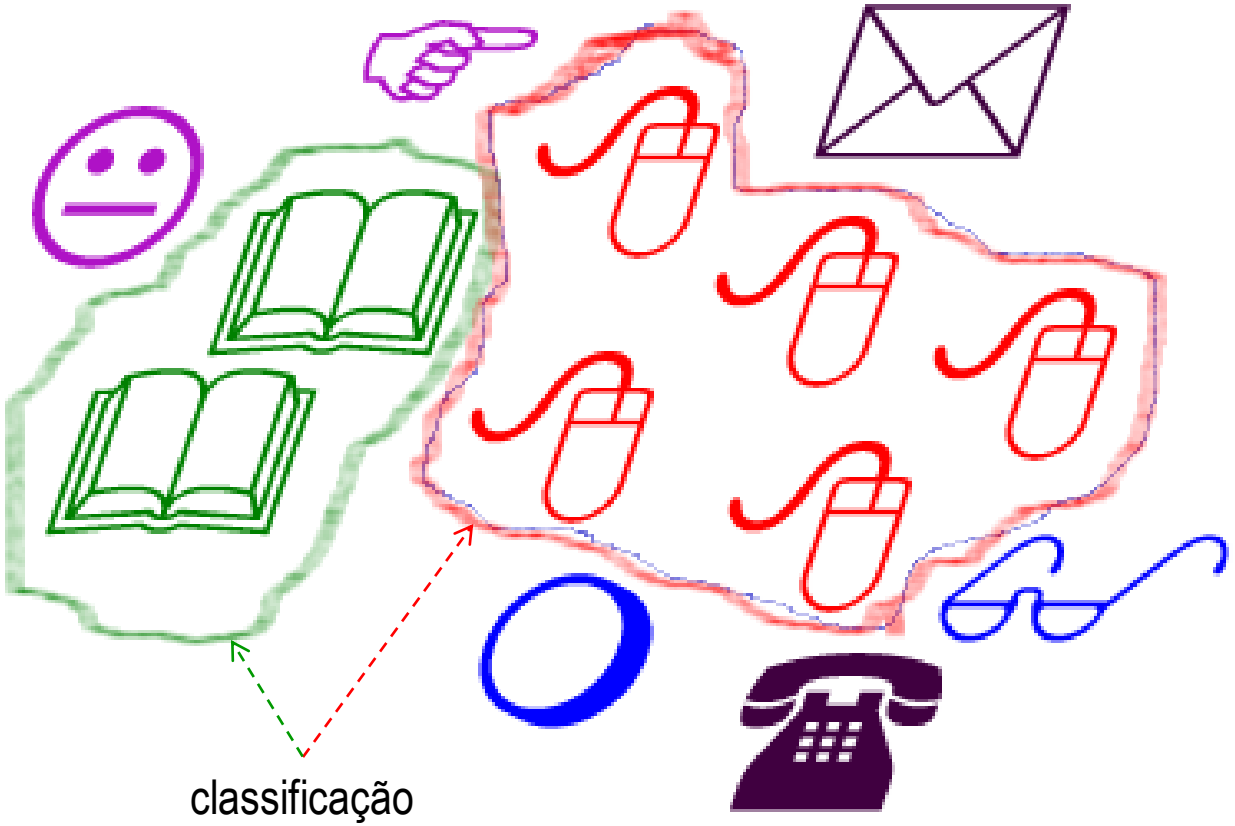
- código da disciplina: 1
- nome da disciplina: Álgebra Linear

Comportamentos de qualquer Disciplina

- listar o nome da disciplina
- etc.

Classe

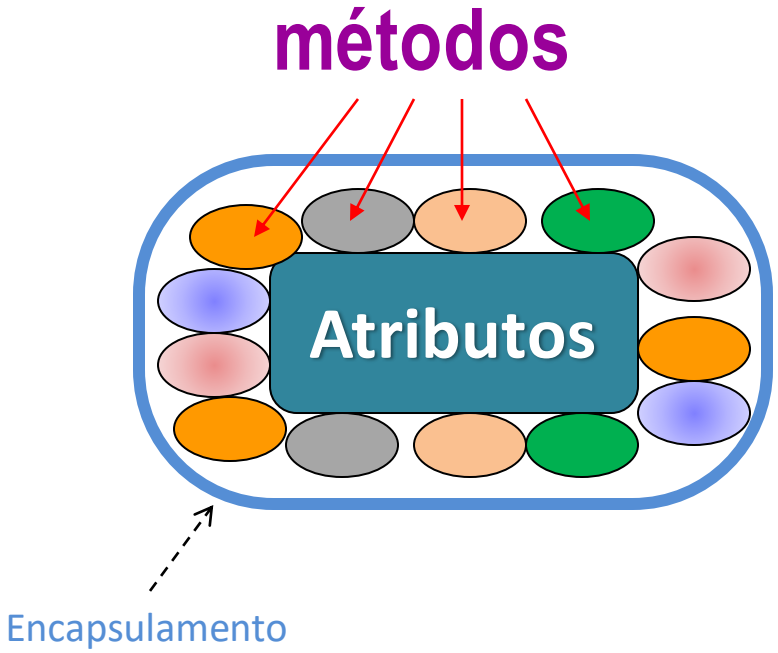
- ***Classe* é um termo denotando *classificação* de entidades (objetos)**



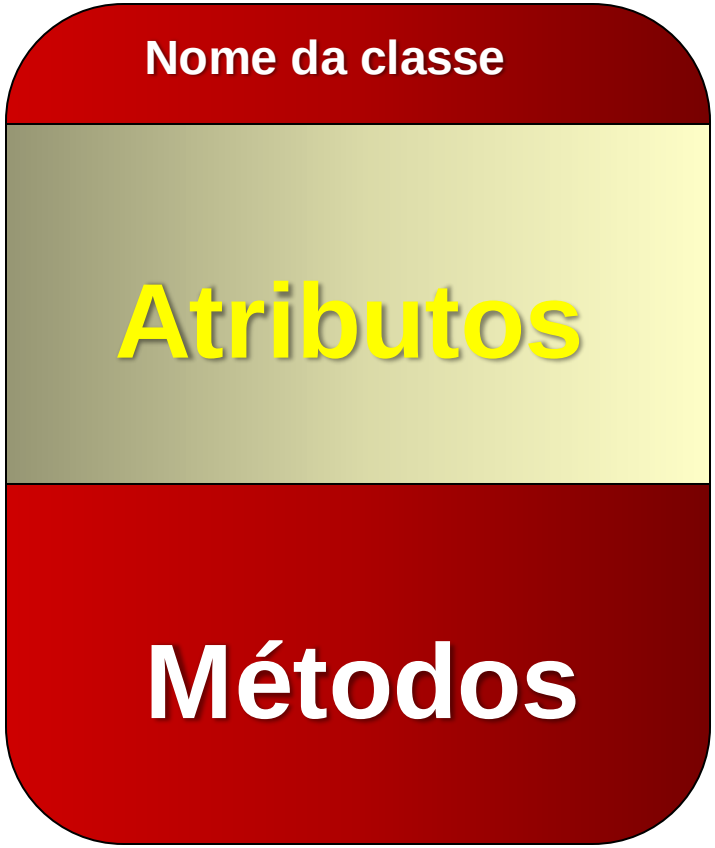
Classes

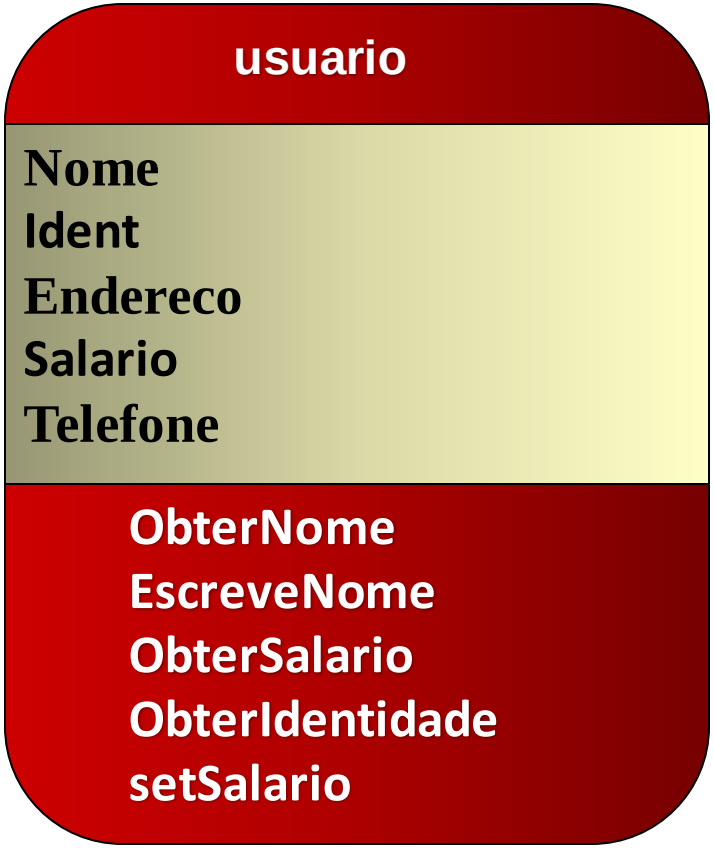
- **Abstrações para representar objetos com características e comportamentos comuns.**
- **Uma classe é formada por:**
  - **um nome**
  - **um conjunto de atributos (estrutura)**
    - **representar a características determinantes do estado interno dos objetos**
    - **São os moldes que conterão a informação (dados) dos objetos**
  - **um conjunto de métodos**
    - **implementa um comportamento específico dos objetos de uma classe**
    - **são as funções (operações) que permitirão acessar a informação dos objetos (o estado interno de um objeto)**

Classe = atributos + métodos



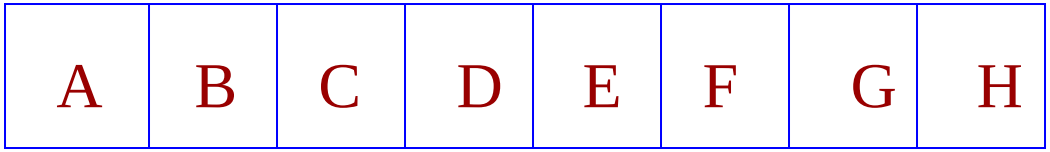
Classe = atributos + métodos





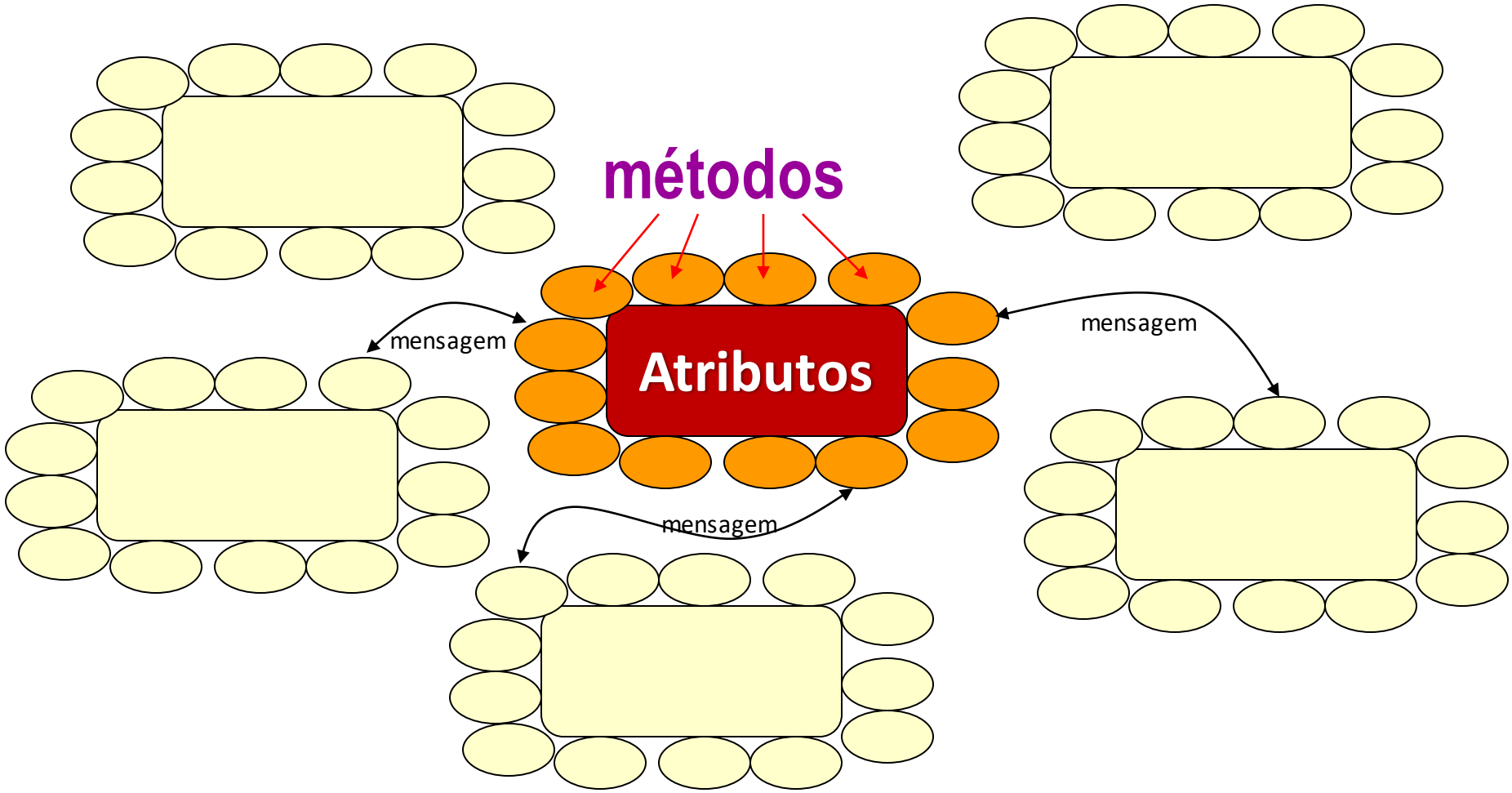
Classe = atributos + métodos

- atributos
  - **São os moldes (estrutura) que conterão a informação (dados) dos objetos**



- métodos:
  - **Operadores**
  - **São as funções (operações) que permitirão acessar a informação dos objetos (o estado interno de um objeto)**

Classe = atributos + métodos



Classes - Exemplo

classe “empregado”

nome	endereço	telefone	idade	salário

obterNome

atualizaEndereço

ObterEndereço

atualizaIdade

obterIdade

Atributos:

- nome
- endereço
- telefone
- idade
- salário

Métodos:

- *obterNome*
- *obterEndereço*
- *atualizaEndereço*
- *obterIdade*
- *atualizaIdade*
- .....

Classes - Exemplo

Classe **Pessoa**

Nome
Idade
CPF

→ **ATRIBUTOS**

```
Nasce(){
  ....
}

Estuda(){
  ....
}
```

→ **MÉTODOS**

Classe **Canario**

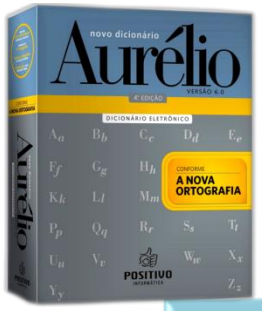
Espécie
Idade
Cor

```
Nasce(){
  ....
}

Morre(){
  ....
}
```

Classe Pilha - Java

```
import java.util.Stack;   // Importa definição da classe Stack

class pilha{
    public static void main(String args[ ]){
        Stack pilha=new Stack();     // Instancia pilha
        int el1;
        String el2;
        pilha.push(new Integer(1));          // Empilha referência de um objeto Integer
        pilha.push(new String("Java"));  // Empilha referência de um objeto String
        try{
            el2=(String) pilha.pop();      // Desempilha String
            el1=((Integer) pilha.pop()).intValue();   // Desempilha valor do inteiro
        }
        catch(EmptyStackException e){
            System.out.println(e);
        }
    }
}
```

Classes - Abstração



ABSTRAÇÃO:
Ato de separar mentalmente um ou mais elementos de uma totalidade complexa (coisa, representação, fato), os quais só mentalmente podem subsistir fora dessa totalidade

Objetos reais

Representação dos Objetos

Classes - Operações Abstratas

- Abstração
  - **mecanismo que permite se centrar nos *aspectos essenciais* de uma entidade**
  - **processo pelo qual se isolam *atributos comuns* de um conjunto de objetos**
  - **processo *seletivo* de certos aspectos de um problema**
  - **significa determinar o QUE de algo**

computador = I/O + memória + CPU

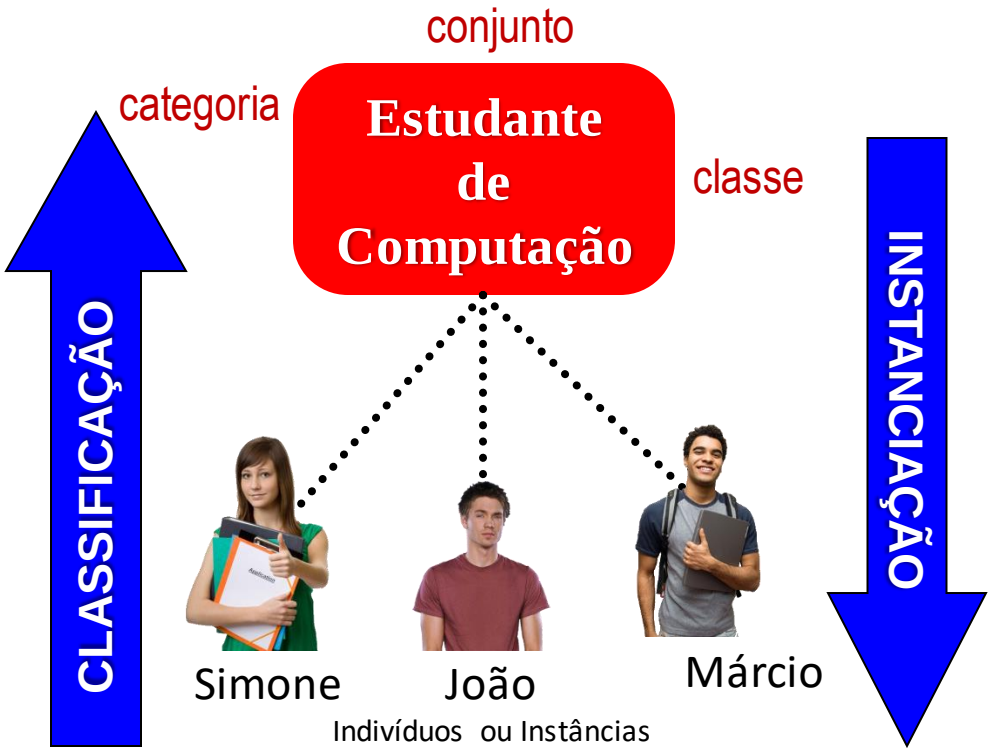
- Operações Abstratas

Cada operação abstrata tem uma operação inversa

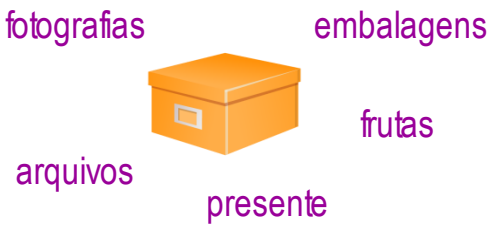
- **classificação (instanciação)**
- **generalização (especialização)**
- **agregação (decomposição)**

Classificação

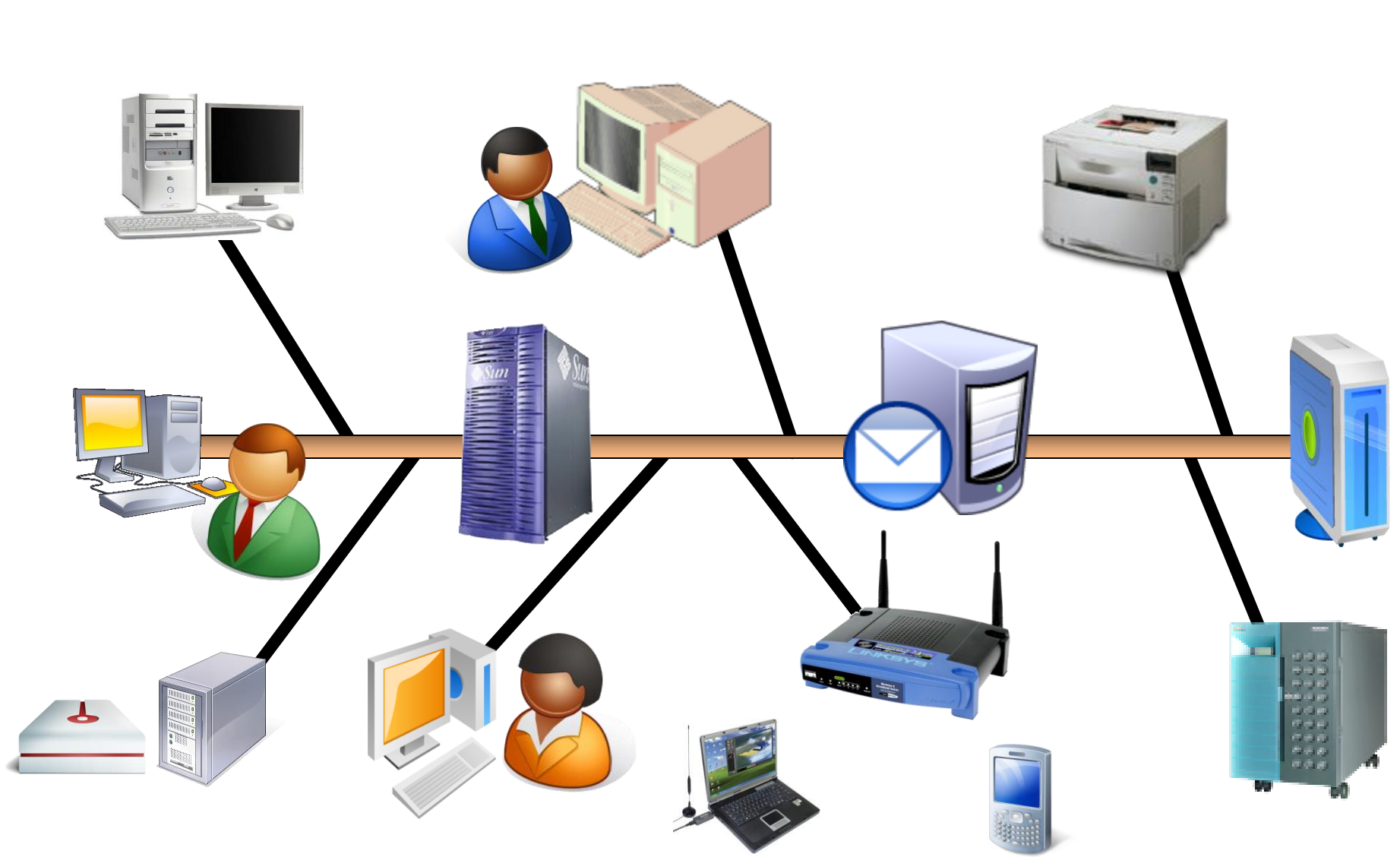
- **É o ato ou resultado de aplicar um conceito a uma entidade (objeto).**
- **É a categorização de entidades em grupos e classes com base em um conjunto de atributos ou propriedades comuns**
- **é a operação que divide em uma ou várias categorias (classes) os elementos de um domínio, de acordo com as suas características comuns.**



Uma entidade pode pertencer a mais de uma classe

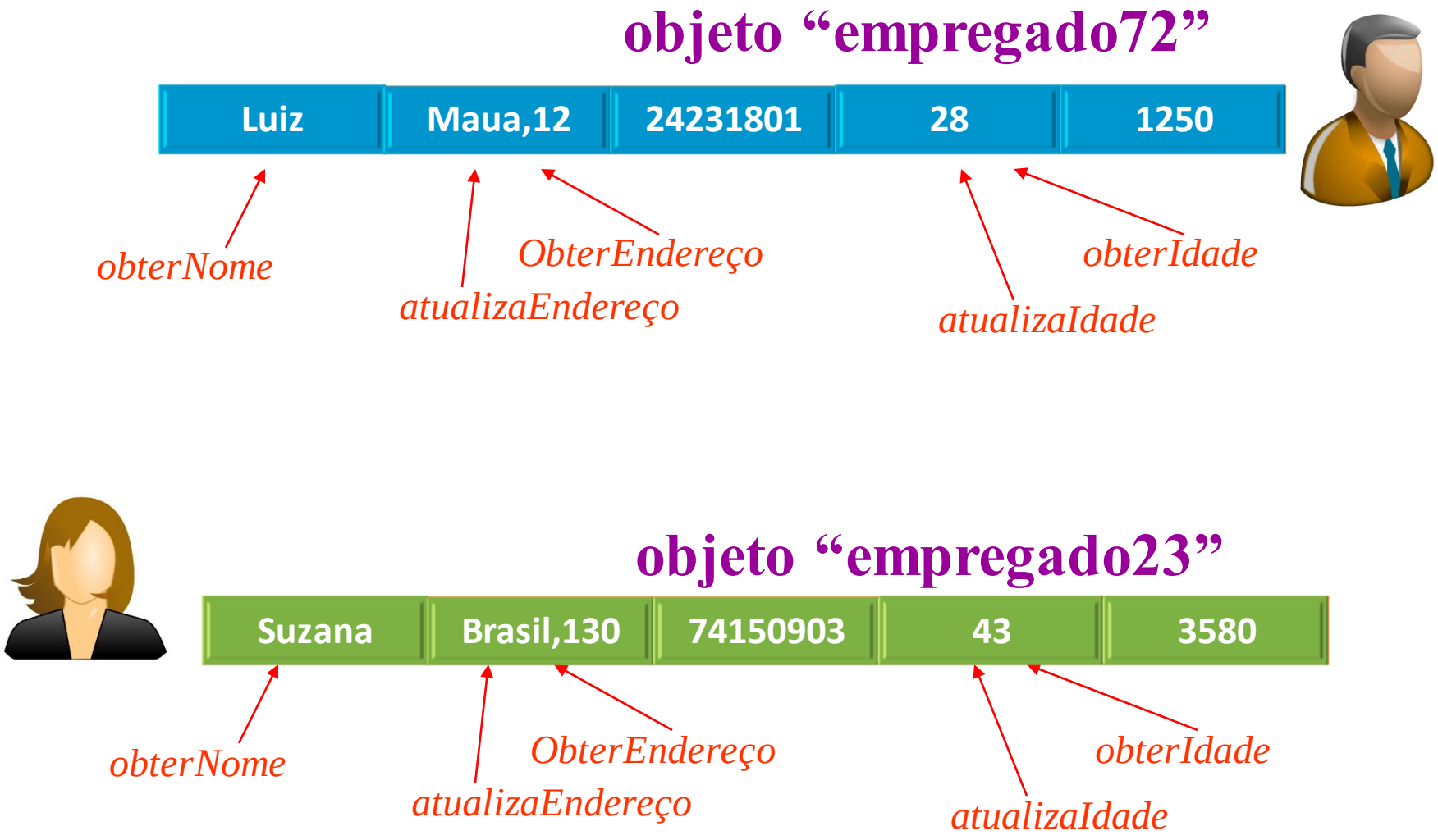


Classificação

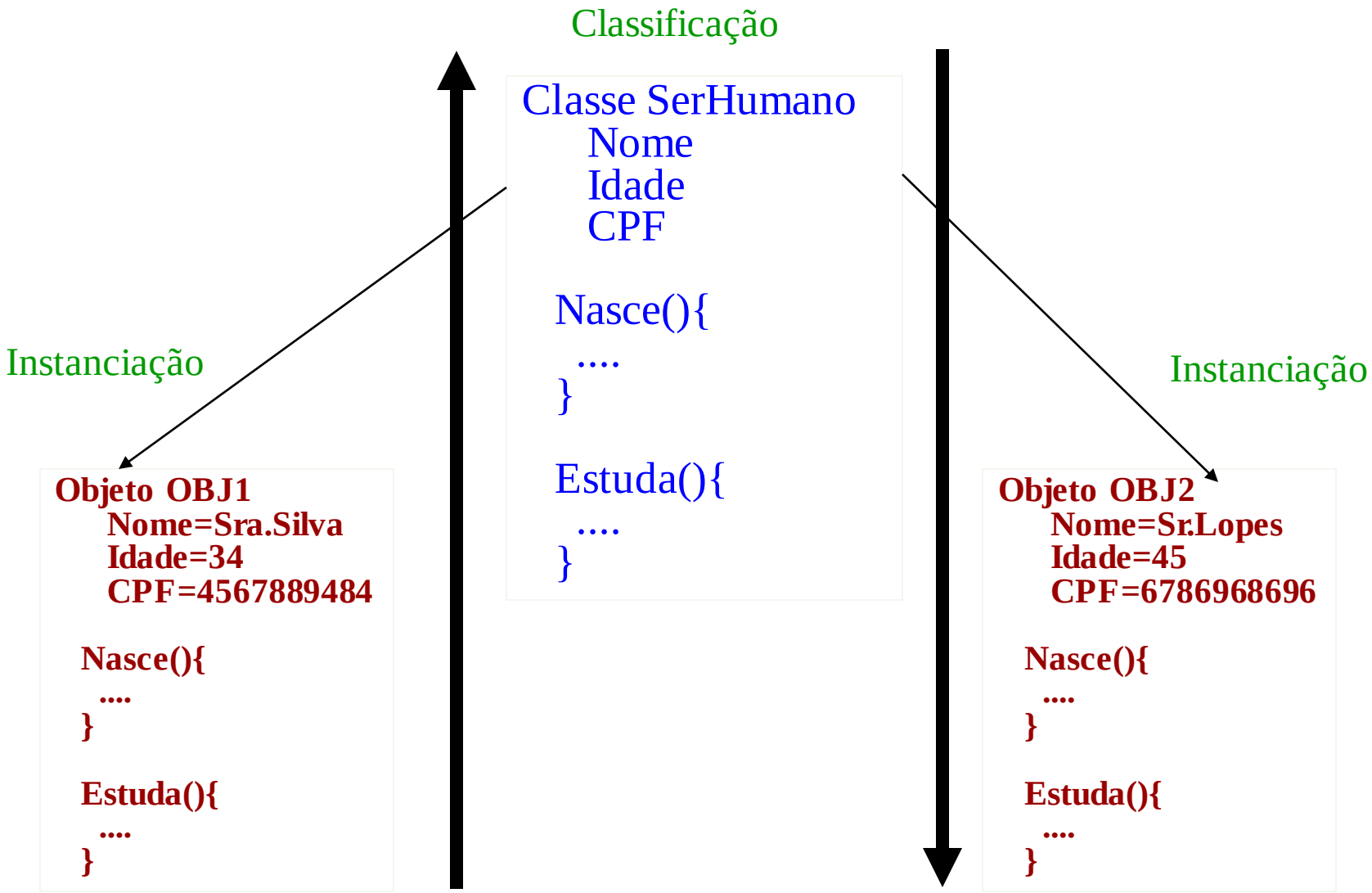


Classificação

Instanciação

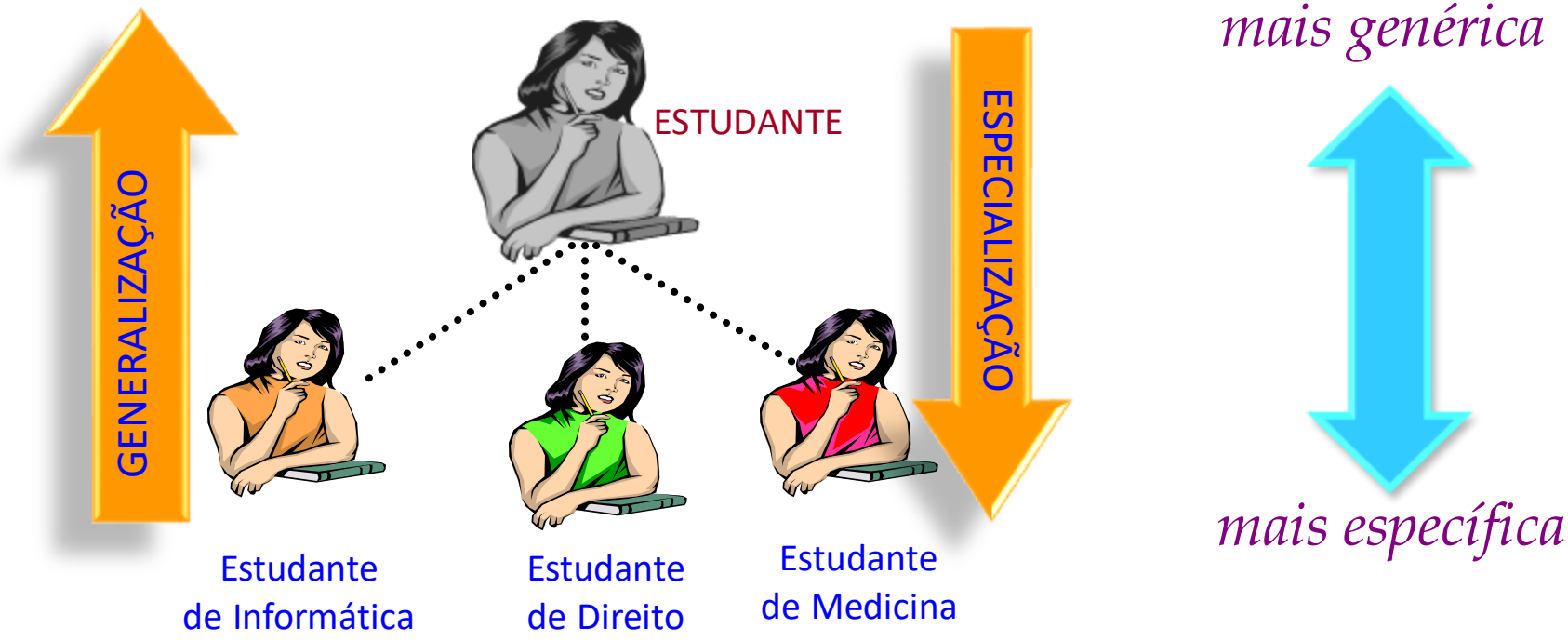


Classificação-Instanciação



Generalização

- **É o ato ou resultado de distinguir um conceito que inclui totalmente outro**
- **Relaciona *duas classes*: uma é a abstração da outra, pela fatoração de propriedades**



Generalização

Programador

- Programador em C++
- Programador Java
- Programador Delphi

Gerente

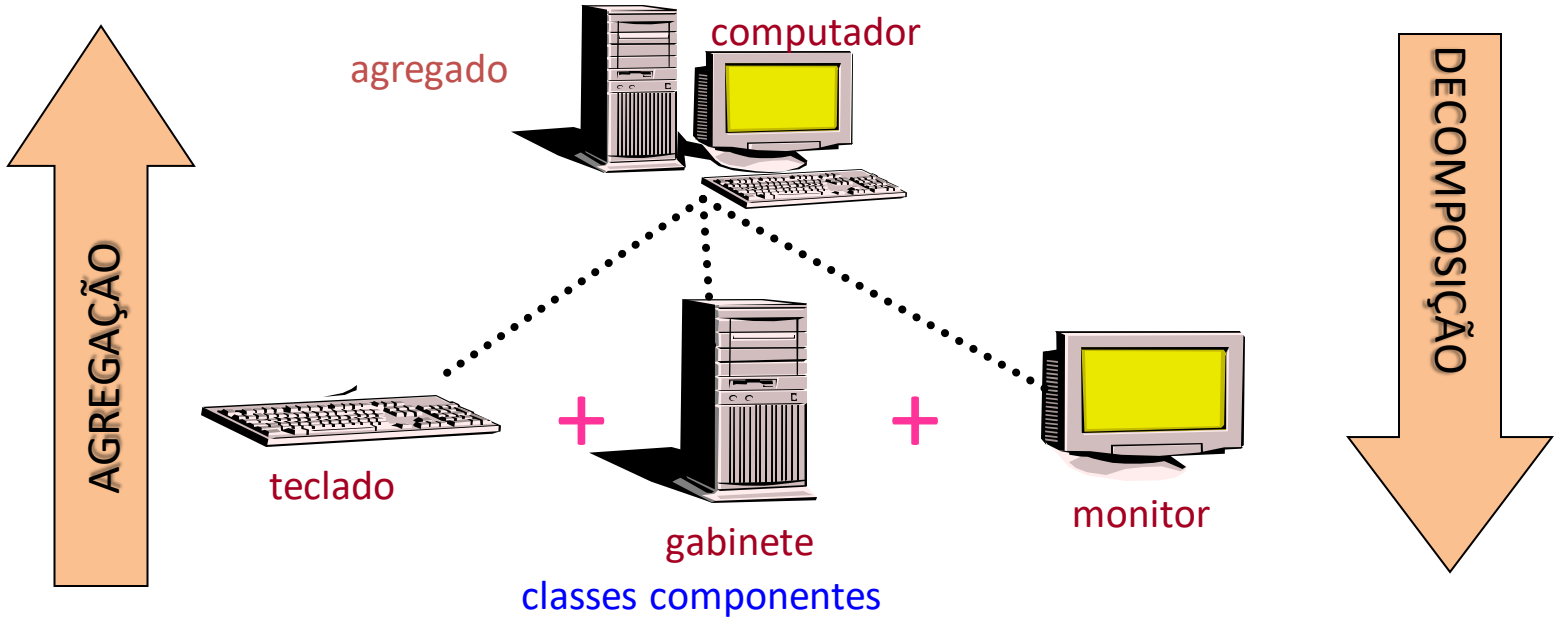
- Gerente de RH
- Gerente de TI
- Gerente de Finanças

Compilador

- Visual Fortran
- DevC++
- Borland Delphi
- MS VisualBasic

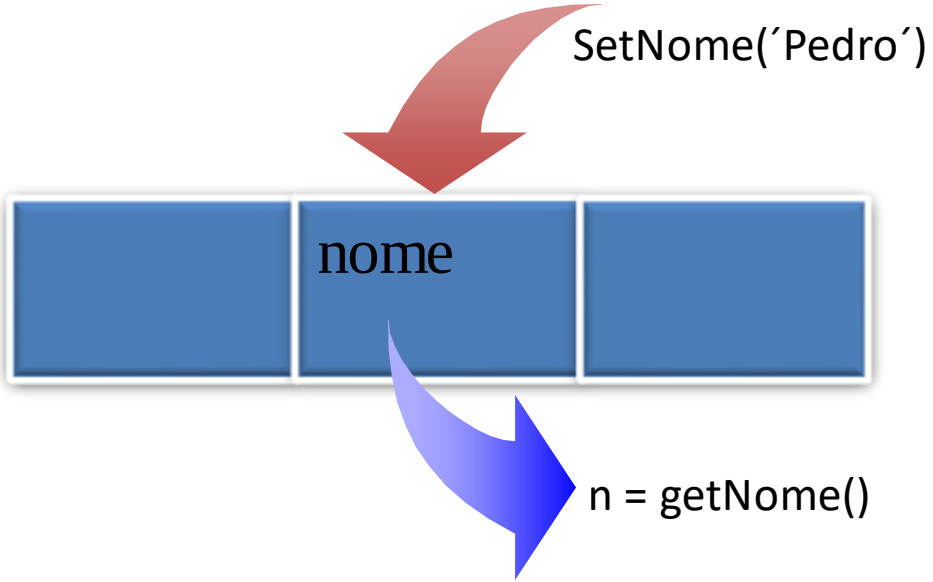
Agregação

- **É o ato ou resultado de formar um objeto (complexo) utilizando outros objetos (mais simples) chamados** componentes
- **é uma relação entre instâncias**
  - **agregação=*composição=síntese***
  - **refinamento=*decomposição=análise***



Métodos

- **Definição de método**
  - **É uma operação ou serviço executado sobre um objeto**
  - **Permitem acessar a informação dos objetos (o estado interno de um objeto)**
- **Tipos de métodos:**
  - **Construtores**
  - **Destrutores**
  - **Tipo SET (escrita)**
  - **Tipo GET (leitura)**
  - **Inicialização**
  - **Outros**



Construtores

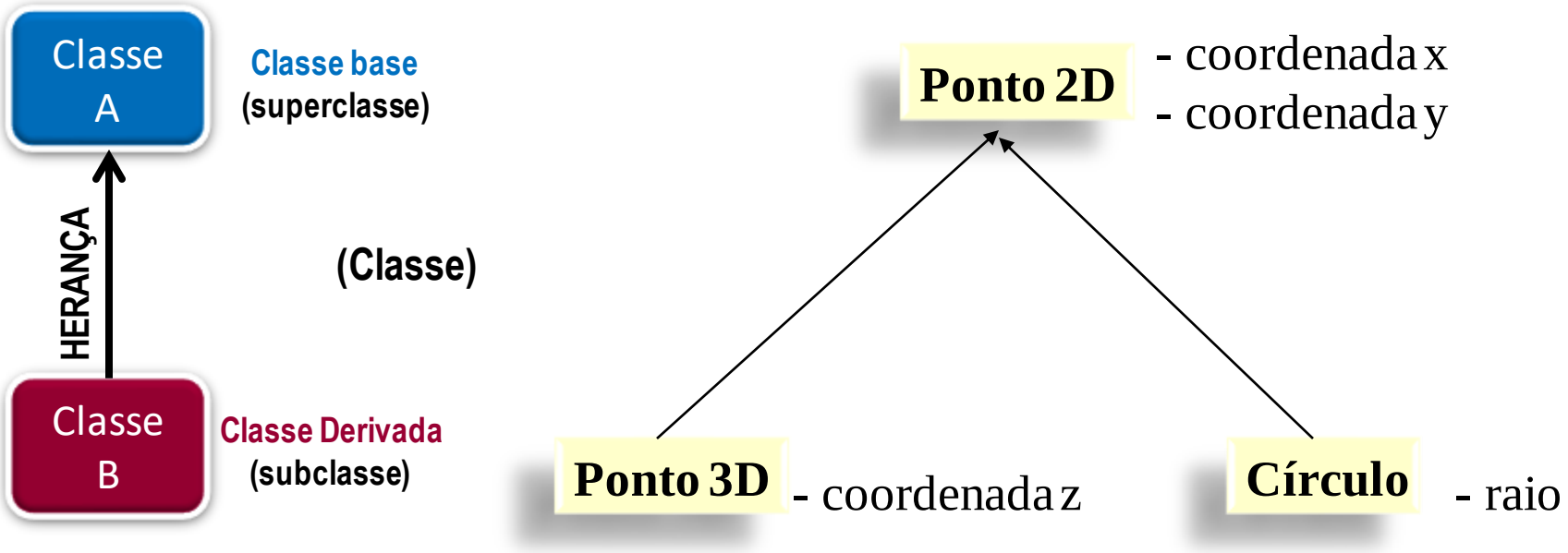
- São *métodos* utilizados para inicializar um objeto no momento da sua definição.
- Tem o mesmo nome da classe.
- Não tem valor de retorno.

Destrutores

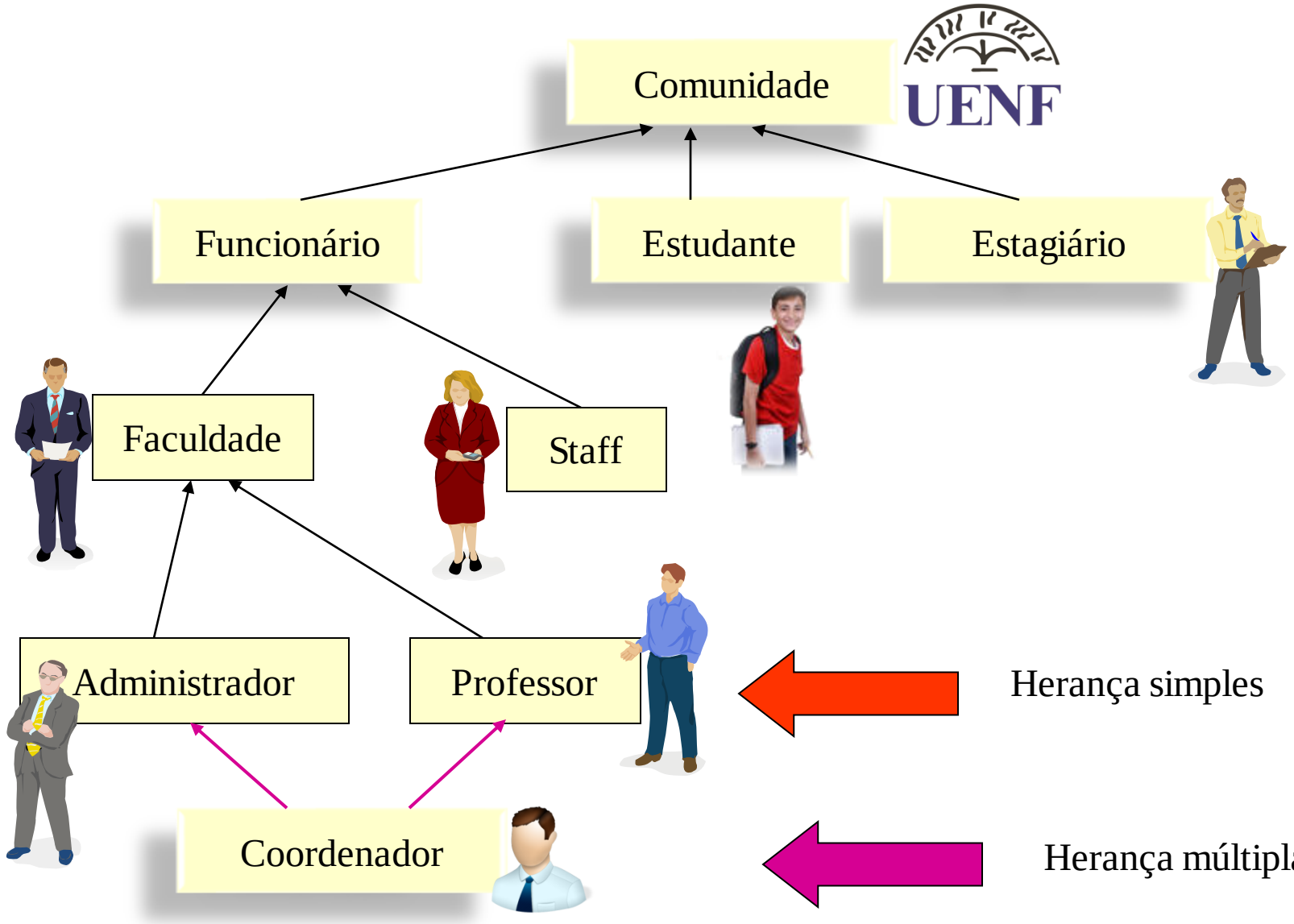
- **Destrutores são declarados similar aos construtores: utilizando o prefixo til ~**
  - **classe abcd**
  - **construtor abcd( ){ ... }**
  - **destrutor ~abcd( ){ }**
- **Servem para destruir objetos quando estes não são necessários**

Herança

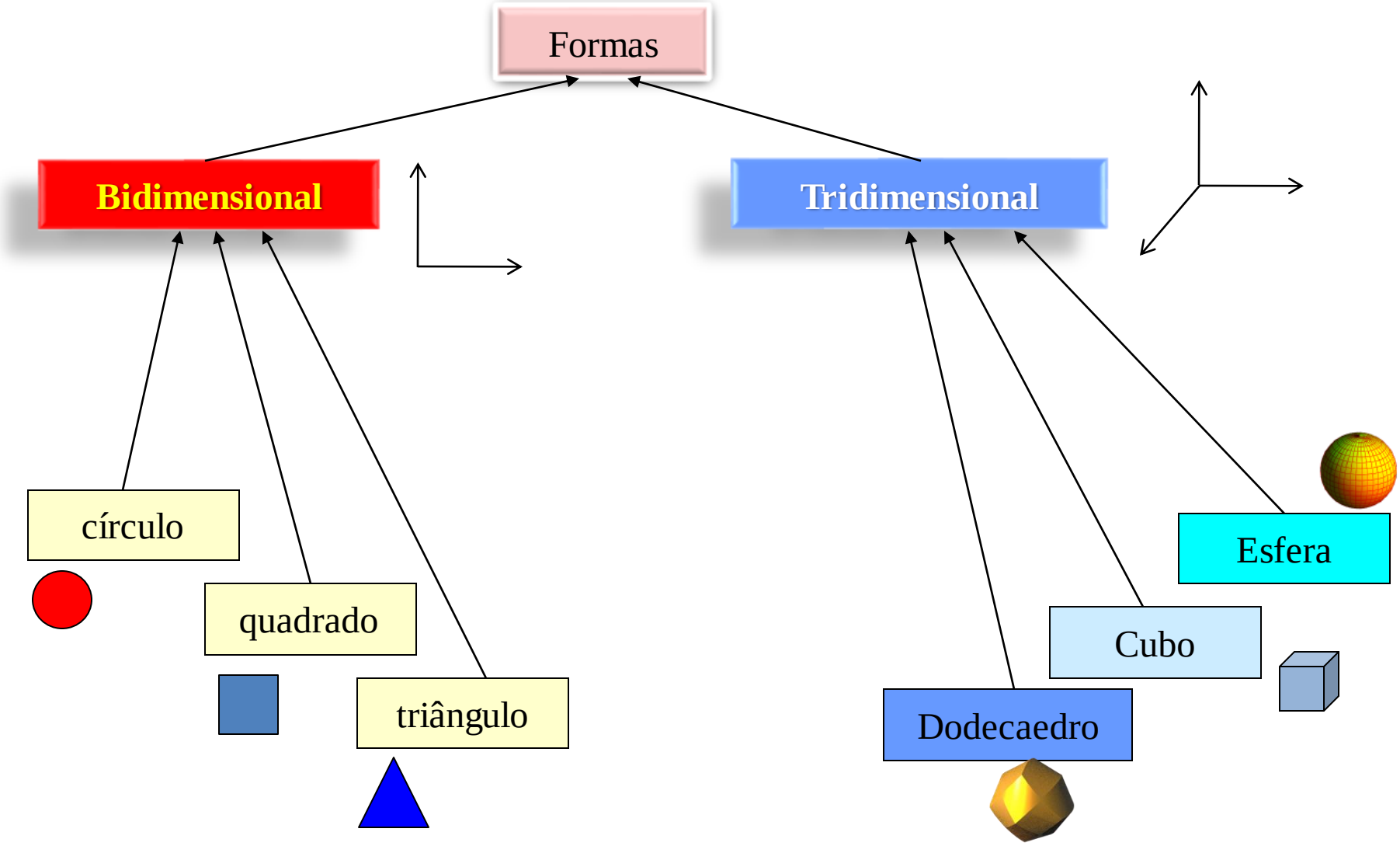
- **É o mecanismo por meio do qual uma classe B *herda* propriedades de outra classe A, i.é., objetos da classe B tem acesso aos atributos e métodos da classe A sem necessidade de re-definir.**



Herança

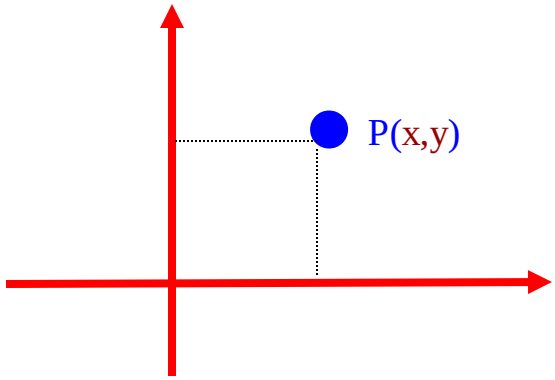

Comunidade
UENF
Funcionário
Estudante
Estagiário
Faculdade
Staff
Administrador
Professor
Coordenador
Herança simples
Herança múltipla

Herança


Formas
Bidimensional
Tridimensional
círculo
quadrado
triângulo
Esfera
Cubo
Dodecaedro

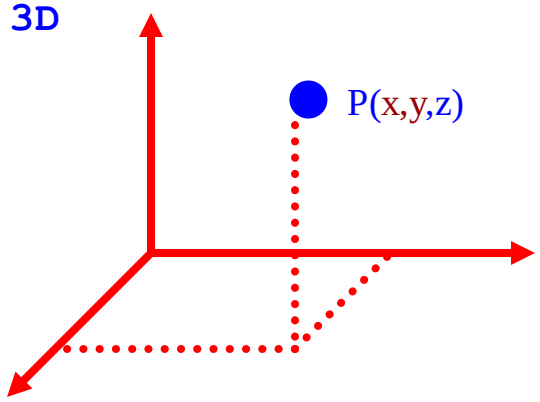
Herança: Ponto 2D

```
class Ponto2D {
      int x;
      int y;
public:
      Ponto2D(){
            x = y = 0;
      }
      Ponto2D(const int xx, const int yy){
            x = xx;
            y = yy;
      }
      ~Ponto2D() {    }   // nada a fazer
      void modificaX(int valx);
      void modificaY(int valY);
      int obterX( );
      int obterY( );
}  meuponto, teuponto;
```

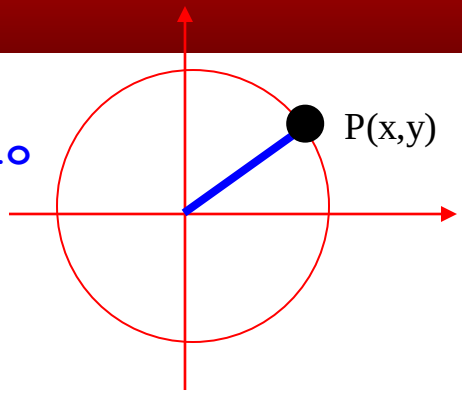


Herança: Ponto 3D

```
class Ponto3D : public Ponto2D {
                // atributo adicional para 3D
        int z;
public:
                //   construtores
        Ponto3D(){
                atualizaX(0);
                atualizaY(0);
                z = 0;
        }
        Ponto3D (const int xx, const int yy, const int zz){
                atualizaX(xx);
                atualizaY(yy);
                z = zz;
        }
                //  destrutor
        ~Ponto3D(){   }
                //  métodos somente para 3D
        int obterZ(){ return z; }
        void atualizaZ(const int valZ) { z = valZ; }
};
```



Herança: Círculo




```
class circulo : public Ponto2D {
                // atributo adicional para circulo
        float raio;
public:
                //   construtores
        circulo(){
                atualizaX(0);
                atualizaY(0);
                raio = 0.0;
        }
        circulo (const int xx, const int yy, const float r){
                atualizaX(xx);
                atualizaY(yy);
                z = r;
        }
                //  destrutor
        ~circulo(){  }
                //  métodos somente para 3D
        float obterRaio(){ return raio; }
        void atualizaRaio(const float valR) {raio=valR; }
};
```

Encapsulamento

- Associado ao conceito de abstração
- Permite ocultar detalhes de implementação
- Em C++ implementado através de cláusulas:
  - private
    - Nenhum acesso externo. Geralmente a estrutura da classe
  - protected
    - Acesso permitido somente via herança
  - public
    - Acesso livre. Geralmente os operadores

Classe Virtual

Uma **classe virtual** é uma *classe interna aninhada* cujas funções e variáveis de membros podem ser substituídas e redefinidas por subclasses de uma classe externa. Classes virtuais são análogas às funções virtuais.

```
#include <iostream>

class Machine {
public:
    void run() { }

    class Parts {
    public:
        virtual int get_wheels() = 0;

        virtual std::string get_fuel_type() = 0;
    };
};
```

As *classes virtuais* resolvem o problema de ***extensibilidade*** de estender a abstração de dados com novas funções e representações. Como funções virtuais, as classes virtuais seguem as mesmas regras de definição, substituição e referência

Classe Virtual

```cpp
#include <iostream>

class Machine {
public:
    void run() { }

    class Parts {
    public:
        virtual int get_wheels() = 0;

        virtual std::string get_fuel_type() = 0;
    };
};
```

```cpp
// The inner class "Parts" of the class "Machine" may return the number of wheels the machine has.
class Car: Machine {
public:
    void run() {
        std::cout << "The car is running." << std::endl;
    }

    class Parts: Machine::Parts {
    public:
        int get_wheels() override {
            std::cout << "A car has 4 wheels." << std::endl;
            return 4;
        }

        std::string get_fuel_type() override {
            std::cout << "A car uses gasoline for fuel." << std::endl;
            return "gasoline";
        }
    };
};
```

Java - Quicksort

```
//////////////////////////////////////////////
// Classe de Ordenação de Vetores de Nodes
// Implementação do algorítmo QuickSort
//
// Herval Freire de A. Júnior
// hervald@mailbr.com.br
//////////////////////////////////////////////

import java.util.Vector;


// classe de ordenação QuickSort de Vectors
public class QuickSorter extends Thread
{
  private Vector list;
  private GraphicNodePanel pan;
  private int wait;
  private Class classe;
            // a classe dos objetos da lista
```

```
 // lista de Objects e um painel para acompanhar
 // os resultados

public QuickSorter(Vector lista, GraphicNodePanel
      painel, int delay)
 {
      list = lista;
      pan = painel;
      wait = delay;

      if(lista.isEmpty() == false)
        classe = (lista.firstElement()).getClass();

      setPriority(4);
 }
```

Outros Tópicos OO

- **Polimorfismo**
- **Funções virtuais**
- **Funções amigas (friends)**
- **Classes abstratas e concretas**
- **Templates**
- **Tratamento de Exceções**
- **Herança Múltipla**

Outros Tópicos OO

- **Conceitos sobre Orientação a Objetos**
  - **Desenho modular**
  - **Encapsulamento**
  - **Coesão**
  - **Acoplamento**
  - **Modificabilidade**
  - **Testabilidade**
  - **Refatoração**

Prof. Dr. Ausberto S. Castro Vera
Ciência da Computação
UENF-CCT-LCMAT
Campos, RJ

ascv@uenf.br
ausberto.castro@gmail.com