

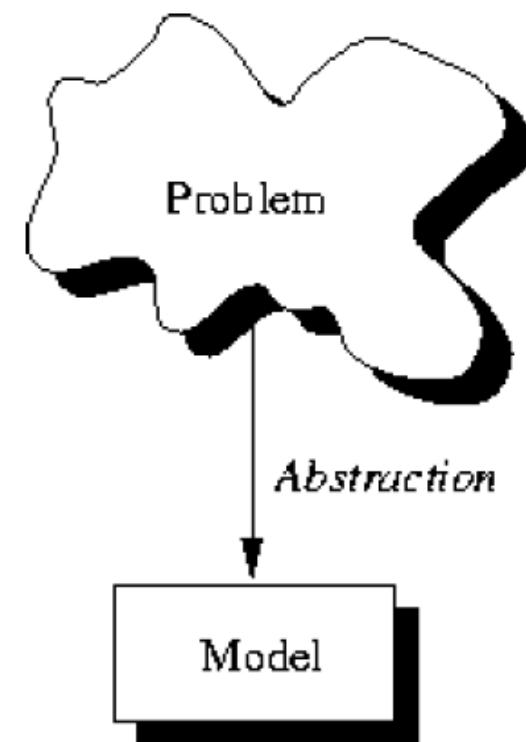
# Programação Orientada a Objetos

Introdução à Orientação e Objetos:

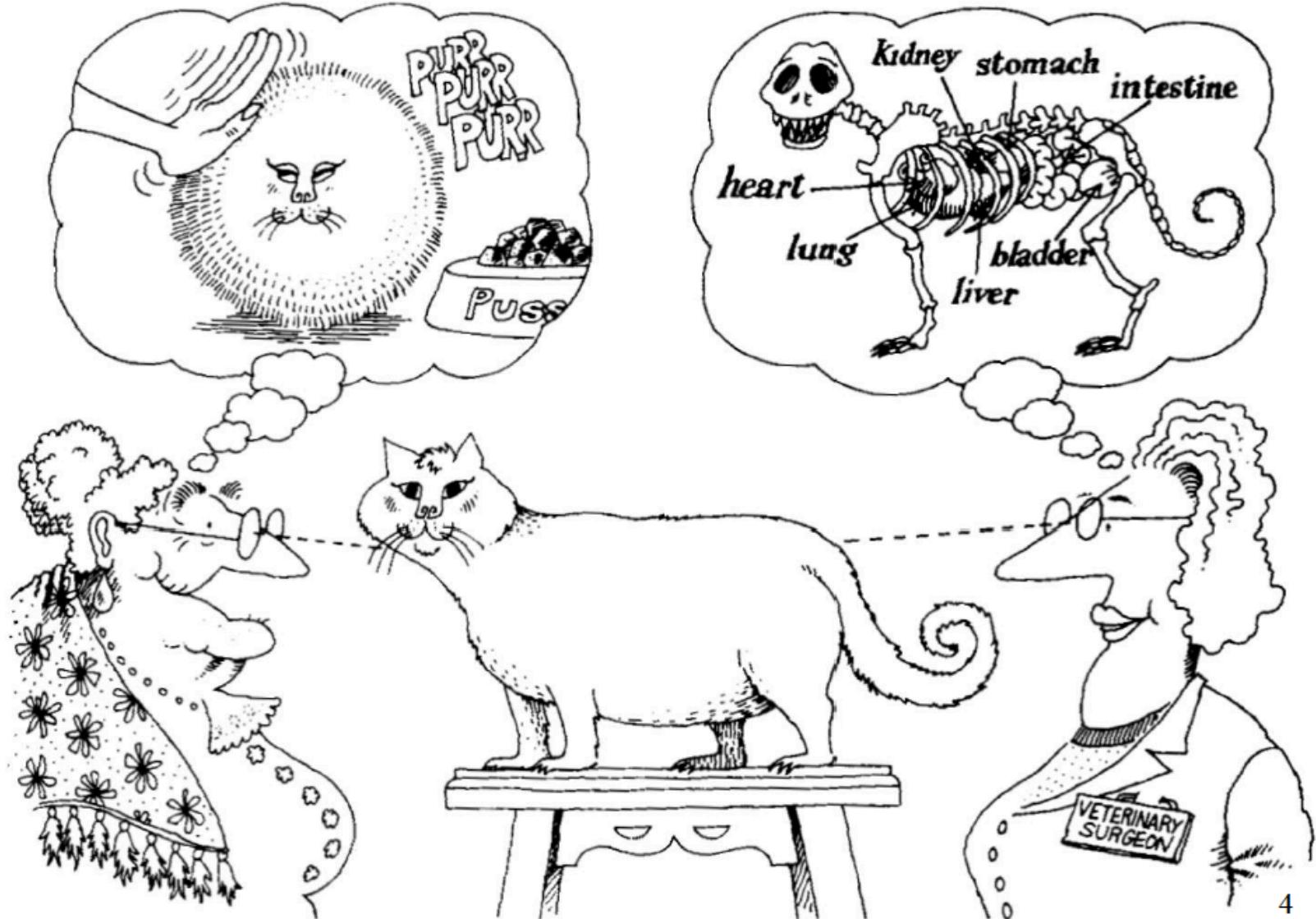
- Classes e Objetos

# A Evolução dos Modelos OO

- A evolução do processo de abstração:
  - abstrair consiste no processo de retirar do domínio do problema os detalhes relevantes e representá-los não mais na linguagem do domínio, e sim na linguagem da solução.
- ➡ para a Engenharia, abstrair é criar modelos a serem usados para solucionar o problema



# Diferentes Abstrações

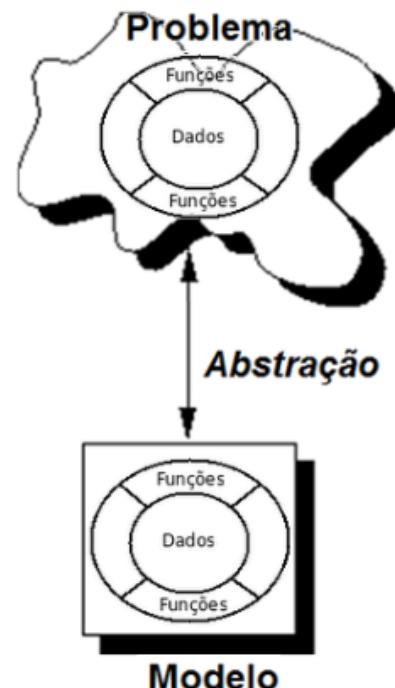


# A Evolução dos Modelos OO

- A evolução do processo de abstração:
  - Todas as linguagens de programação fornecem **abstrações**.
    - a linguagem *Assembly* é uma **abstração** da máquina na qual os programas são executados;
  - As linguagens estruturadas (FORTRAN, BASIC e C) são **abstrações** da linguagem *Assembly*.
  - Nestas linguagens, é necessário pensar em termos da **estrutura do computador**, ao invés de se pensar em termos da **estrutura do problema** a ser resolvido
  - O programador deve estabelecer uma associação entre o modelo da máquina (no **espaço da solução**) e o modelo do problema que está sendo resolvido (no **espaço do problema**).
    - **O esforço para fazer este mapeamento pode ser gigantesco !!!**

# A Evolução dos Modelos OO

- A Programação Orientada a Objetos tenta **trazer o espaço da solução para o espaço do problema**
  - ambos são representados como objetos!
- Permite adaptação ao problema
  - Adiciona novos tipos ao espaço da solução que mapeiam os tipos existentes no espaço do problema
  - Assim, descreve-se o problema em termos do problema e não em termos da solução!
- Cada objeto funciona como uma pequena parte do problema
  - possui seu estado e suas operações, que podemos pedir que eles executem
    - isso é semelhante ao comportamento dos objetos no mundo real, que também possuem características e comportamentos!



# A Abstração de Objetos

- O que são objetos?
  - Um objeto é uma variável ... ele armazena dados. Então uma struct (em C) é um objeto?
    - Sim, uma struct é um objeto .. mas um objeto pode ser mais
      - Podemos pedir que algumas operações sejam feitas sobre os objetos.
  - Um objeto possui então,
    - atributos (dados) e comportamentos (métodos, procedimentos, funções) que atuam sobre ele
  - Exemplos de objetos:
    - cachorros, carros, telefones celulares, edifícios, funcionários, indústrias ...

# Programação Estruturada x POO

// Uma janela : Implementação clássica em C ...

```
typedef struct tag_Window {  
    int x, y;      // Posição na tela  
    int cx, cy ;   // Largura e altura  
    Canvas* my_canvas; // Estrutura que contém atributos  
                        // a serem utilizados para desenho  
} CWindow;
```

```
int initialize(CWindow* w, int xp, int yp, int cx, cy); // inicializa  
int show(CWindow* w); // mostra na tela  
int cleanup(CWindow* s); // libera  
int move(CWindow* s ,int newx, int newy); //Move  
int resize(CWindow* s , int newcx, int newcy); // Redimensiona  
int setTextColor(CWindow* s , COLORREF cor); // Define cor  
int textOut(CWindow* s , int x, int y, char* text); // Texto em x,y
```

# Programação Estruturada x POO

```
#include "CWindow .h" // página anterior (outros includes não mostrados)
int main() {
    CWindow janela1, janela2; // Duas janelas
    initialize(&janela1, 1, 1, 100, 200);
    initialize(&janela2, 110, 220, 120, 100);

    ...
    show(&janela1);
    show(&janela2);

    ...
    COLORREF c1(255, 0 , 0 ); // RGB ==> vermelho
    setTextColor(&janela1, c1);
    textOut(&janela1, 3, 5, "Alo Mundo com Janelas");

    ...
    cleanup(&janela1);
    cleanup(&janela2);
}
```

# Programação Estruturada x POO

- Quais os problemas com esta abordagem?
  - Toda vez que trabalhamos com a estrutura, temos que passar o endereço dela para a função
  - Apesar de as funções terem sido criadas para manipular a estrutura, elas são separadas dela
  - Este é o velho problema da separação entre dados e funções!
  - Outro problema que pode ocorrer é o de “briga de nomes” entre bibliotecas: **initialize** e **cleanup**, por exemplo, são muito comuns ...

# Programação Estruturada x POO

- A programação orientada a objetos nos permite uma solução bem mais elegante: criar classes!

```
class CWindow {  
    int x, y;      // Posição na tela  
    int cx, cy ;  // Largura e altura  
    Canvas* my_canvas; // Estrutura que contém atributos  
                        // a serem utilizados para desenho  
  
public:  
    int initialize(int xp, int yp, int cx, cy); // +- (ver construtor)  
    int show(); // mostra na tela  
    int cleanup(); // +- (ver destrutor)  
    int move(int newx, int newy); //Move  
    int resize(int newcx, int newcy); // Redimensiona  
    int setTextColor(COLORREF cor); // Define cor do texto  
    int textOut(int x, int y, char* text); // Texto em x,y  
};
```

# Programação Estruturada x POO

- Uma vez que a classe seja implementada, podemos instanciar objetos e chamar seus métodos:

```
CWindow janela1;  
janela1.initialize(1,1,100,200);  
janela1.move(6,7);
```

- O tamanho do objeto janela1 em memória é equivalente ao da *struct* correspondente!

# Programação Estruturada x POO

```
#include "CWindow.h" // Declaração da classe com includes necessários
int main() {
    CWindow janela1, janela2; // Duas janelas
    janela1.initialize(1, 1, 100, 200);
    janela2.initialize( 110, 220, 120, 100);

    ...
    janela1.show();
    janela2.show();

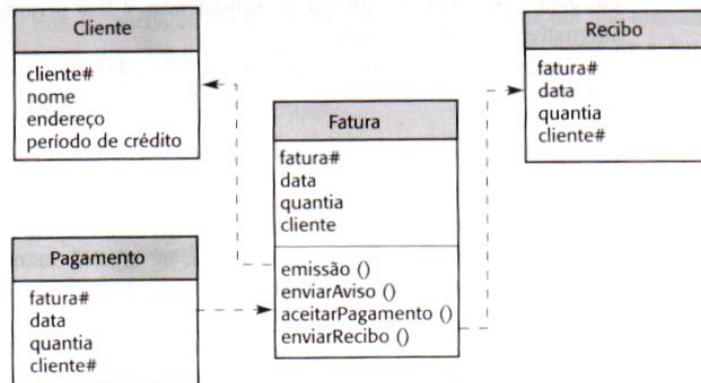
    ...
    COLORREF c1(255, 0 , 0 ); // RGB ==> vermelho
    janela1.setTextColor( c1);
    janela1.textOut(3, 5, "Alo Mundo com Janelas Orientadas a Objetos");

    ...
    janela1.cleanup();
    janela2.cleanup();
}
```

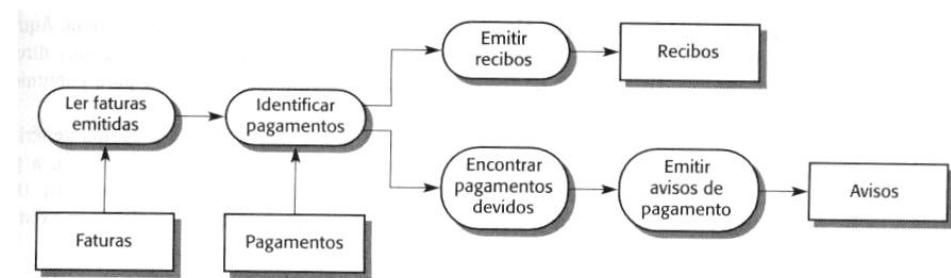
# Analise Estruturada x Analise OO

- A Análise e o Projeto Orientado a Objetos é uma metodologia que nos leva a uma decomposição orientada a objetos de um sistema.
- os modelos resultantes serão **focados** nas “coisas”, “entidades” => **objetos** que compõem o sistema.
- totalmente distinto da Análise Estruturada que é focada nos **procedimentos**.

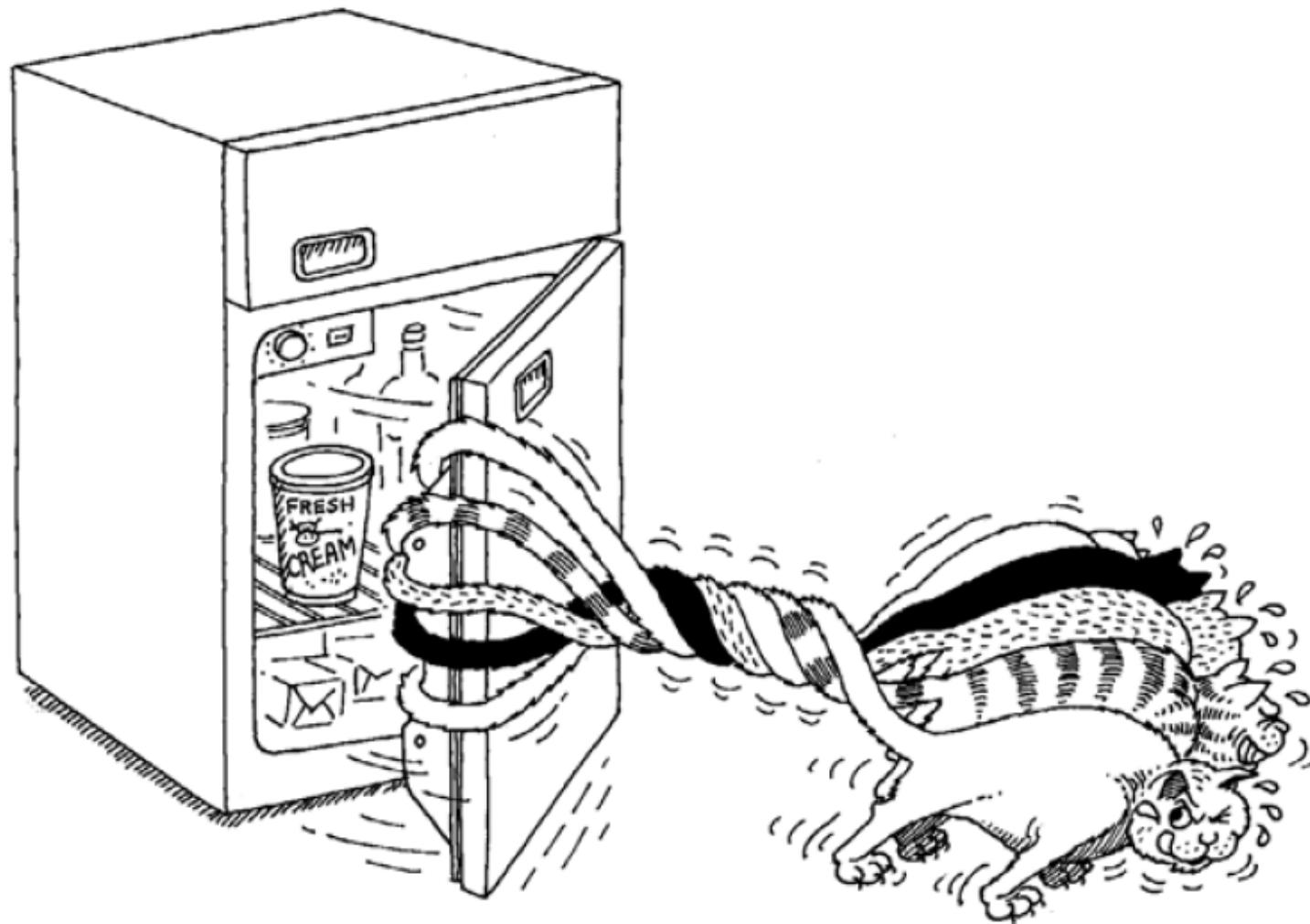
OO



Estruturada

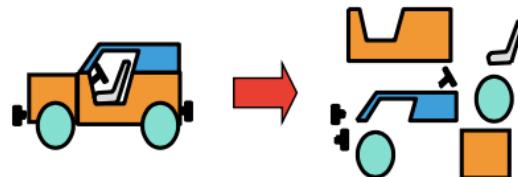


# A abstração de objetos



# A abstração de objetos

- O que é um programa em linguagem OO?
  - Conjunto de objetos dizendo uns para os outros o que fazer através do envio de mensagens.
  - Concretamente, pode-se pensar nas mensagens como sendo chamadas a funções que pertencem a um objeto em particular.
  - Cada objeto tem a sua própria região de memória, que pode ser composta por outros objetos, também.
    - Exemplo: o objeto carro pode ser composto pelos objetos lataria, rodas, motor, etc.
  - Cada objeto tem um **tipo**: isto é, pertence a uma **classe**.



# Objetos e Classes

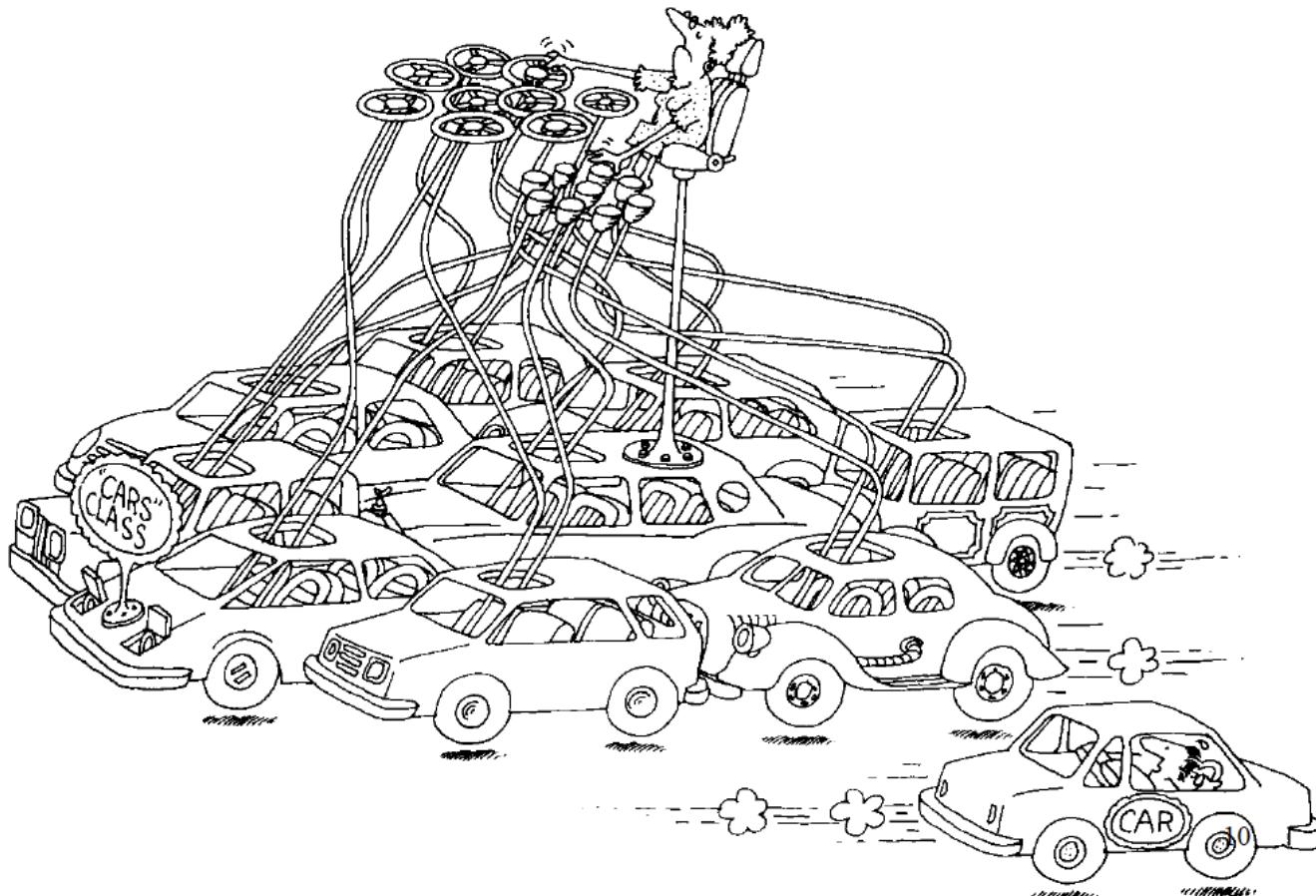
Aristóteles já identificava a idéia de **tipos**:

- Classe das aves e classe dos peixes;
- Os objetos, apesar de serem únicos, fazem parte de uma **classe de objetos** que possuem características comuns
- O tipo (classe) a que pertence um objeto **identifica o que se consegue fazer com ele**:
  - O pombo da praça 7 (objeto) consegue voar porque é uma ave (classe a que pertence);
  - O peixe dourado do aquário (objeto) consegue nadar porque é um peixe (classe a que pertence).



# Classes

- Uma classe define a estrutura e o comportamento de um conjunto de objetos.



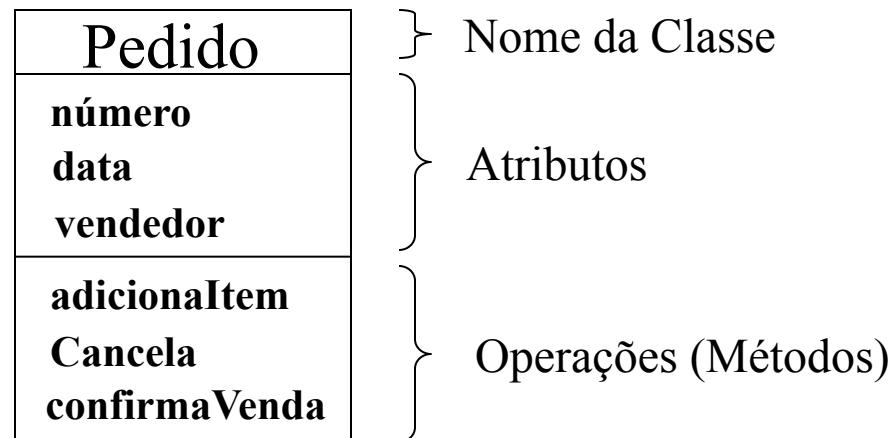
# Classes

- Uma classe é a descrição de um grupo de objetos com propriedades similares (atributos) e comportamento comum (operações).
  - Um objeto é uma instância de uma classe.
- Exemplo:

Classe	Atributos	Operações
Pedido	Número	Adiciona item
	Data	Cancela
	Vendedor	Confirma venda

# Classes

- Em UML, uma classe é representada utilizando-se um retângulo dividido em três seções:
  - A primeira seção contém o nome da classe.
  - A segunda seção mostra a estrutura (atributos).
  - A terceira seção mostra o comportamento (operações).
- Exemplo:



# Classes - Exemplos:

Conta
<b>número</b>
<b>titular</b>
<b>saldo</b>
<b>saca</b>
<b>deposita</b>
<b>transferepara</b>

Carro
<b>cor</b>
<b>modelo</b>
<b>velocidadeAtual</b>
<b>liga</b>
<b>acelera</b>

Cliente
<b>nome</b>
<b>endereço</b>
<b>cpf</b>
<b>alteraEndereco</b>

# Atributos

- Os atributos definem o conjunto de propriedades de uma classe.
- Cada atributo deve ter uma definição clara e concisa.
- Um atributo é definido por:
  - **nome**: um identificador para o atributo.
  - **tipo**: o tipo do atributo (inteiro, real, caractere etc.)
  - **valor default**: opcionalmente, pode-se especificar um valor inicial para o atributo.
  - **visibilidade**: opcionalmente, pode-se especificar o quanto acessível é um atributo de um objeto a partir de outros objetos. Valores possíveis são:
    - privativo - nenhuma visibilidade externa;
    - público - visibilidade externa total;
    - protegido - visibilidade externa limitada.

# Atributos

```
class CWindow {  
  
private:  
    int x, y;      // Posição na tela  
    int cx, cy ; // Largura e altura  
    Canvas* my_canvas; // Estrutura que contém atributos  
                        // a serem utilizados para desenho  
public:  
};
```

# Operações

- Uma classe incorpora um conjunto de responsabilidades que definem o comportamento dos objetos na classe.
- Uma operação é um serviço que pode ser requisitado por um objeto para obter um dado comportamento.
- Operações também são chamadas de métodos (Java) ou funções-membro (C++).

# Definição de Operações

- Uma operação (função membro) é definida por:
  - **nome**: um identificador para o método.
  - **tipo**: quando o método tem um valor de retorno, o tipo desse valor.
  - **lista de argumentos**: quando o método recebe parâmetros para sua execução, o tipo e um identificador para cada parâmetro.
  - **visibilidade**: como para atributos, define o quanto visível é um método a partir de objetos de outras classes.

```
int CWindow::move(int newx, int newy){  
    x = newx;  
    y = newy;  
    return 0;  
}
```

# Classes em C++

```
class NomeDaClasse {  
  
    private:  
  
        // atributos  
        int variavel_privada;  
  
        // outros atributos ou funções membro  
        ....  
  
    public:  
  
        // funções membro  
        void metodo_publico();  
  
        // outras funções membro  
        ...  
};
```

# Classes em C++

- OBS: como organizamos o código em C++?
  - Definição das classes **no “.h”** e das funções **no “.cpp”**

```
// Código defensivo para prevenir a redefinição em inclusões múltiplas
#ifndef SIMPLE_H
#define SIMPLE_H

class Simple {
    int i, j, k;
public:
    initialize() { i = j = k = 0; }
};

#endif // SIMPLE_H
```

# Exemplo – class Rectangle

```
// rectangle.h

#ifndef RECTANGLE_H
#define RECTANGLE_H

class Rectangle {
private:
    int height;
    int width;

public:
    void initialize(int initH, int initW)
    int area();
    void setHeight(int newHeighth);

}; // class definition

#endif
```

```
// rectangle.cpp
#include "rectangle.h"

void Rectangle::initialize(int initH, int
initW){
    height = initH;
    width = initW;
}

int Rectangle::area() {
    return height * width;
}

void Rectangle::setHeight(int
newHeighth){
    height = newHeight;
}
```

# Exemplo – class Rectangle

```
// main.cpp
```

```
#include "rectangle.h"
```

```
#include <iostream>
```

```
using namespace std;
```

```
int main() {
```

```
    Rectangle box, square; // instanciando objetos
```

```
    box.initialize(12, 10);
```

```
    square.initialize(8, 8);
```

```
    cout << "The area of the box is " << box.area() << "\n";
```

```
    cout << "The area of the square is " << square.area() << "\n";
```

```
    square.setHeigth(5);
```

```
    cout << "The area of the square is " << square.area() << "\n";
```

```
}
```

# Exercícios

- Qual a diferença entre o uso de estruturas e o uso de classes em C++?
- Escreva o código que implemente uma classe chamada Point em C++ para manipular pontos no espaço bi-dimensional. Além da inicialização de objetos ponto, a sua classe deve fornecer funções membro para:
  - acessar e mudar atributos dos objetos;
  - mover o ponto para uma nova posição;
  - calcular a distância do ponto a origem;
  - calcular a distância entre dois pontos p1 e p2, onde p1 é o objeto que chama a função.
  - escrever as coordenadas do ponto na tela;