

# Fundamentos e Conceitos de POO

**Prof. Dr. Lucas C. Ribas**

**Disciplina:** Programação Orientada a Objetos

Departamento de Ciências de Computação e Estatística



UNIVERSIDADE ESTADUAL PAULISTA  
“JÚLIO DE MESQUITA FILHO”



IBILCE / UNESP - CÂMPUS DE SÃO JOSÉ DO RIO PRETO

# Agenda



- Paradigmas de Programação
- Os Pilares de POO
- Classes e Objetos
- Tipos de Acesso

# Paradigmas de Programação



- Paradigma: modelo ou padrão para abordar determinado problema segundo um conjunto de procedimentos, valores ou conceitos que direcionam o pensamento
- Visão sobre a estruturação e execução de um procedimento
- Técnicas, estruturas, conceitos utilizados para o desenvolvimento de determinado software

# Paradigmas de Programação



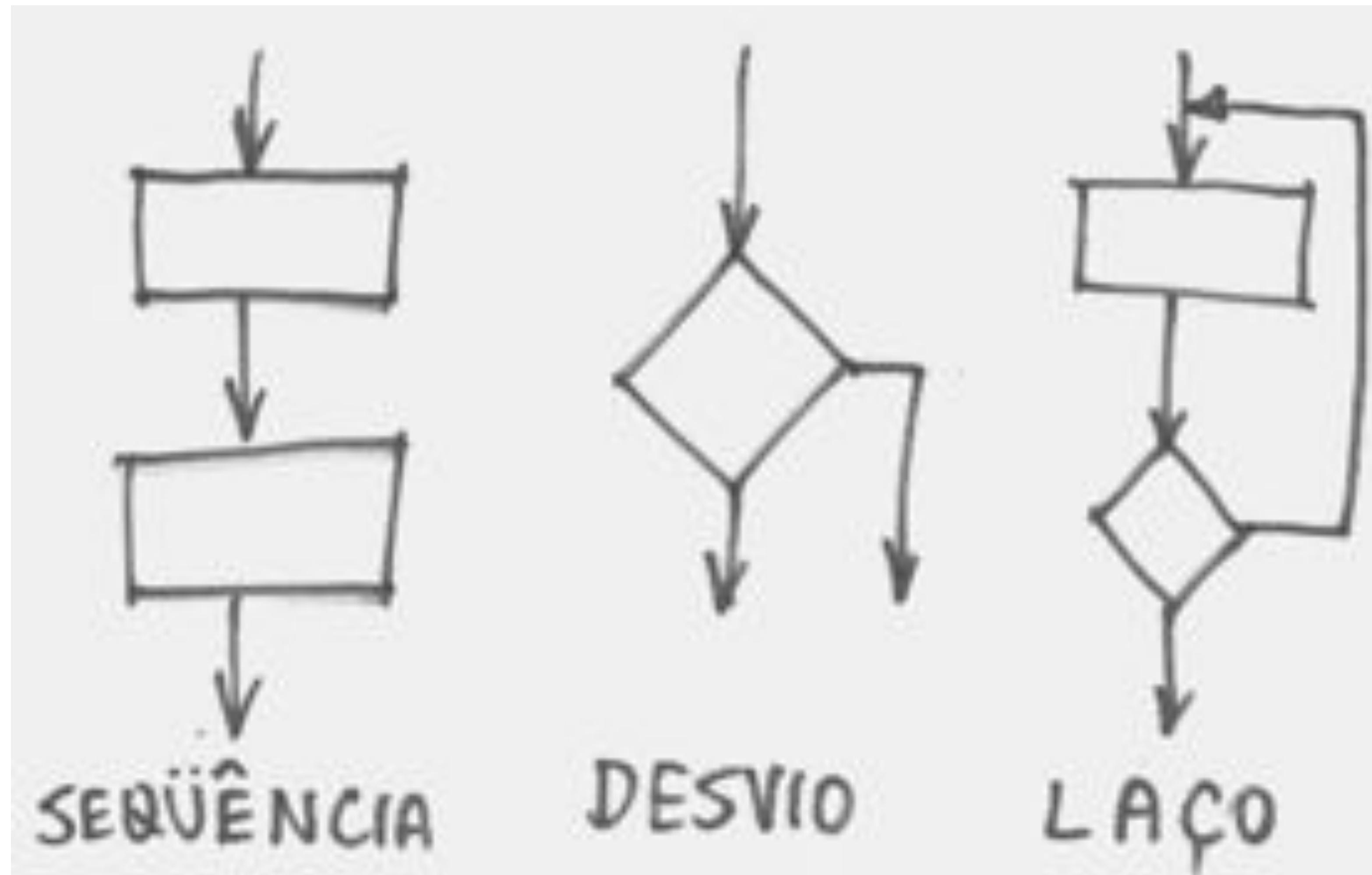
- Programação Estruturada
- Programação Orientada a Objetos



- Como você realmente escreve um grande *software*?
  - Quanto tempo levará?
  - Como o código será organizado?
  - Dá para reaproveitar algum código?
  - Como será testado?
  - Será fácil depurar os bugs?
  - Como se dividem as tarefas entre mais programadores?
  - Como juntar todos os códigos no final?
  - Funciona?

# Programação Estruturada

# Programação Estruturada



# Programação Estruturada



- Na **programação estruturada** os comandos de um programa são executadas sequencialmente
- Vários comandos permitem que essa sequência seja quebrada, causando o que é chamado de **transferência de controle**
  - Porém, de forma organizada

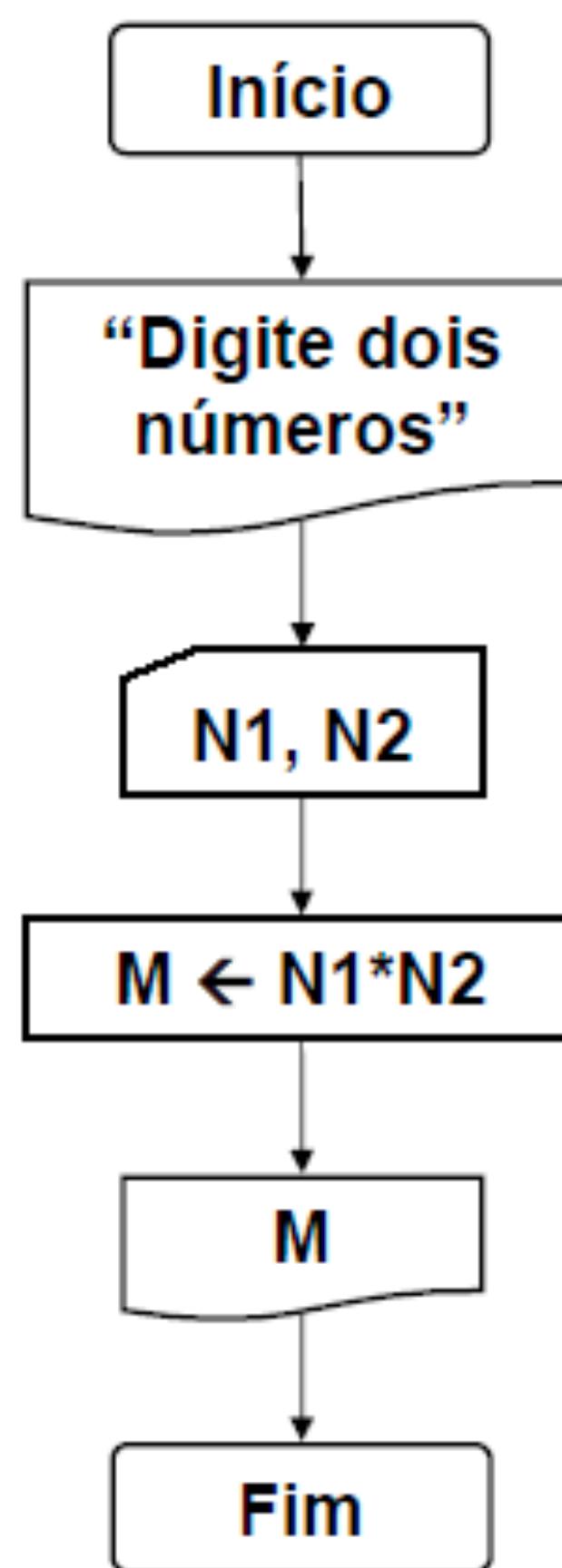


- Estabelece que um programa pode ser composto por blocos elementares de código que se interligam através de três mecanismos básicos
  - Sequência
  - Seleção
  - Repetição
- Cada uma destas construções tem um ponto de início (o topo do bloco) e um ponto de término (o fim do bloco) de execução.

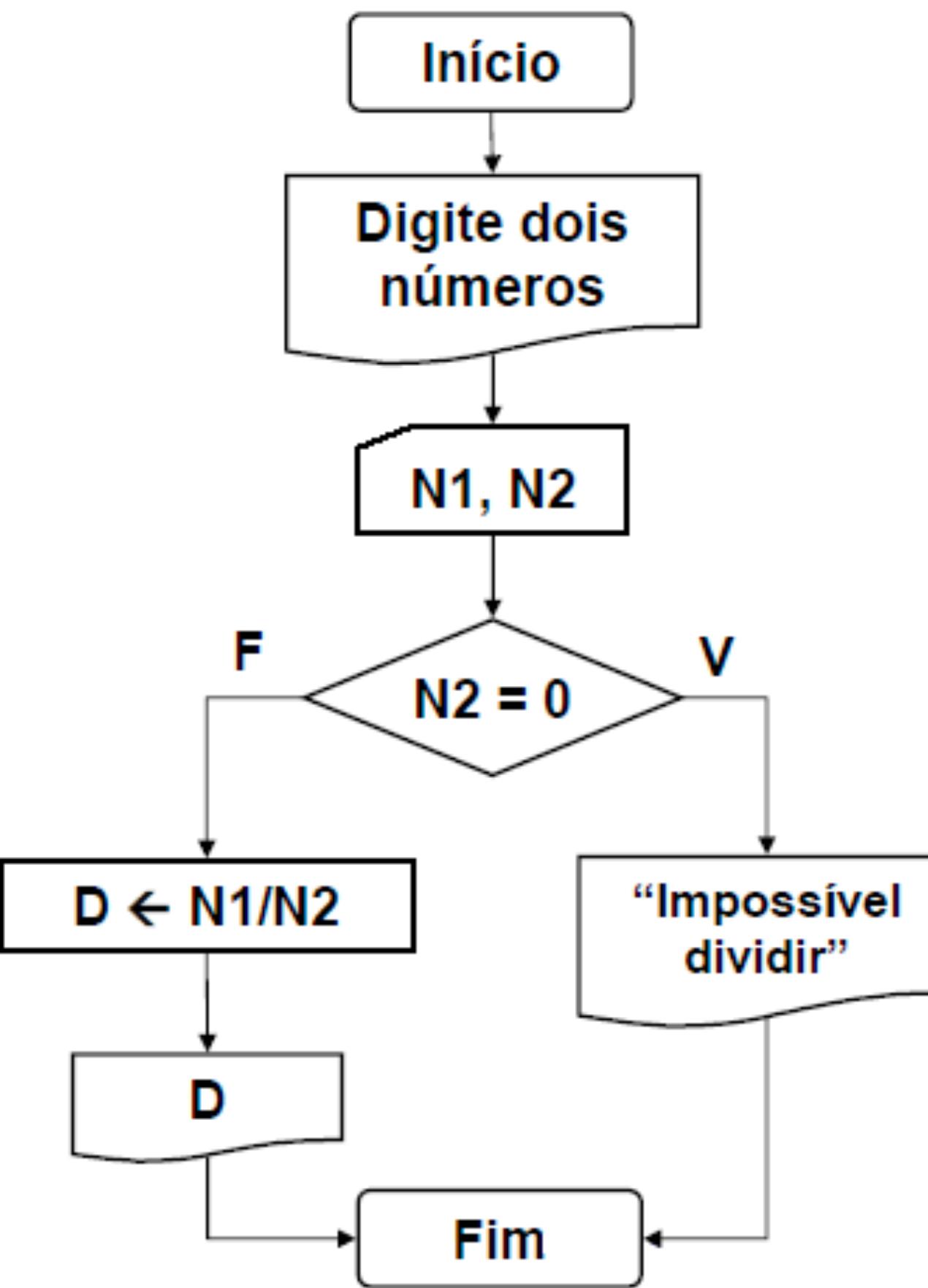
# Programação Estruturada



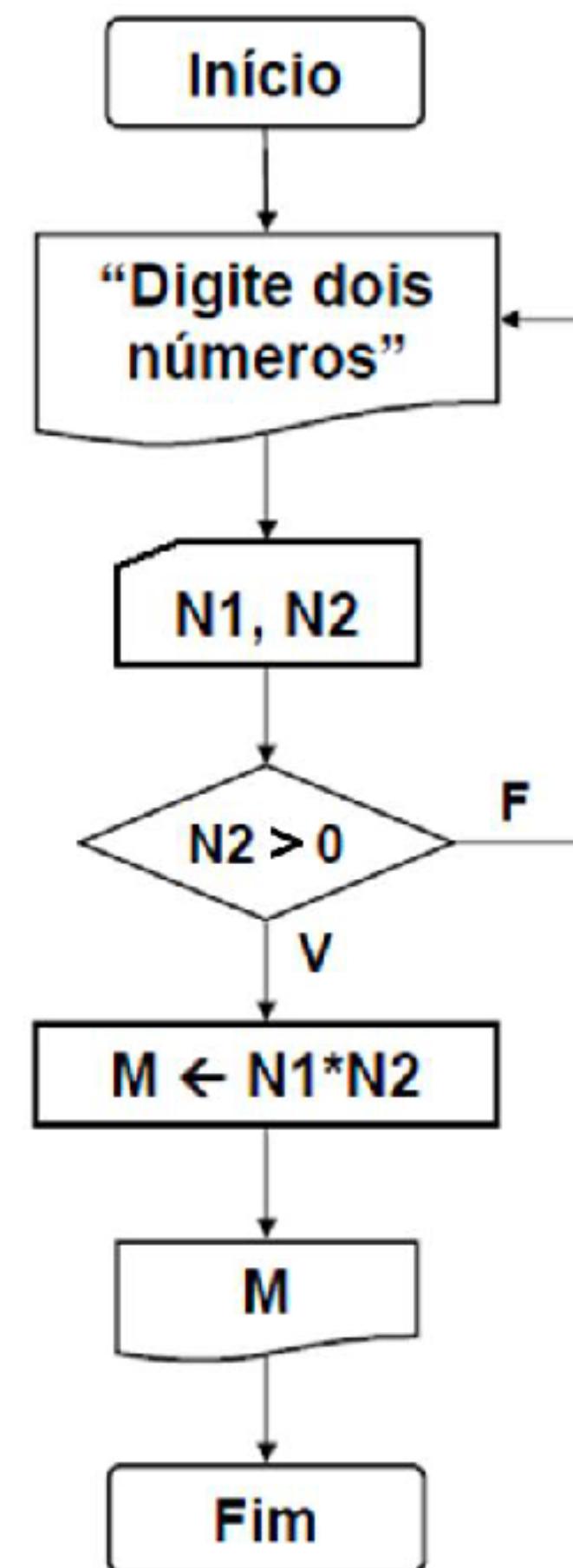
**Estruturas de sequência:**  
passos de processamento  
necessários para descrever  
qualquer programa



**Estruturas de seleção/decisão:**  
possibilidade de selecionar o fluxo de  
execução do processamento  
baseado em ocorrências lógicas (IF e  
SWITCH)



**Estruturas de repetição:**  
permite a execução repetitiva  
de segmentos do programa  
(FOR e WHILE)



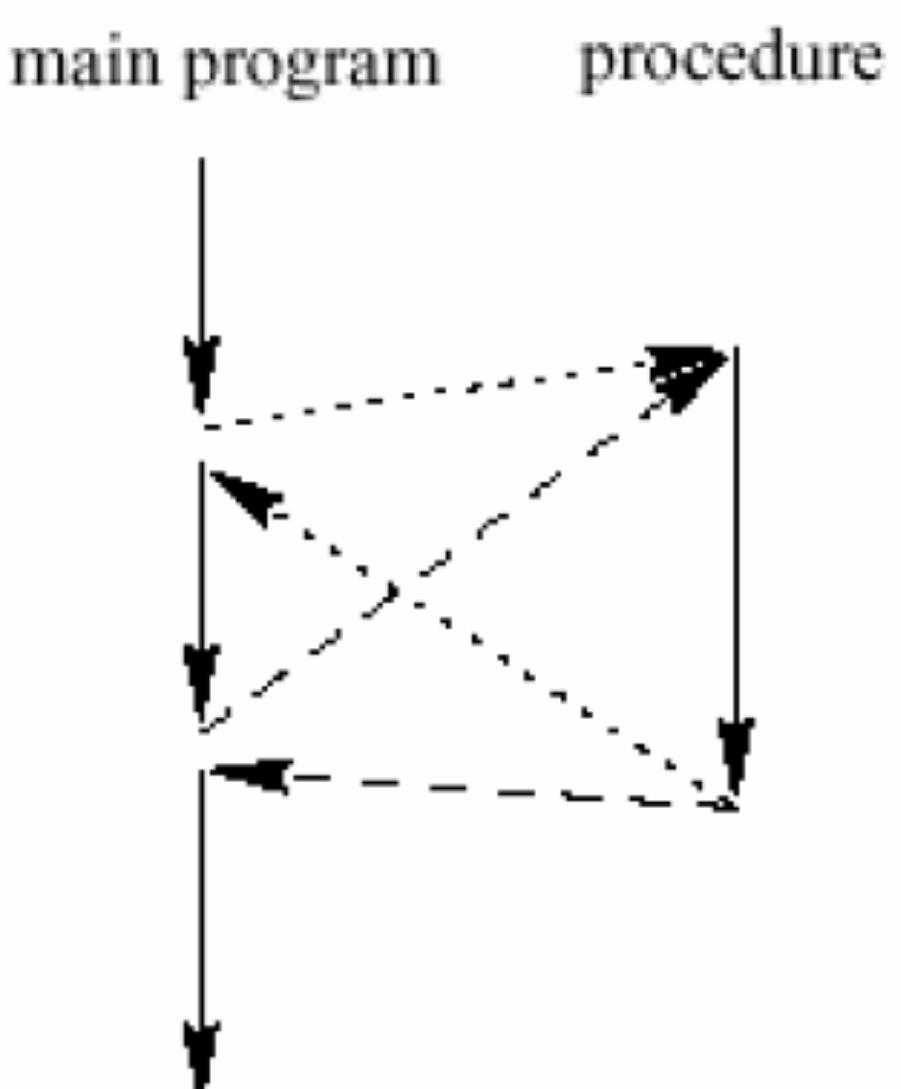
# Programação Estruturada



- A experiência tem mostrado que a melhor forma de se desenvolver programas de grande porte é dividí-los em pequenas partes
  - Dividir para conquistar
- Em programas estruturados essas partes são denominadas funções ou procedimentos

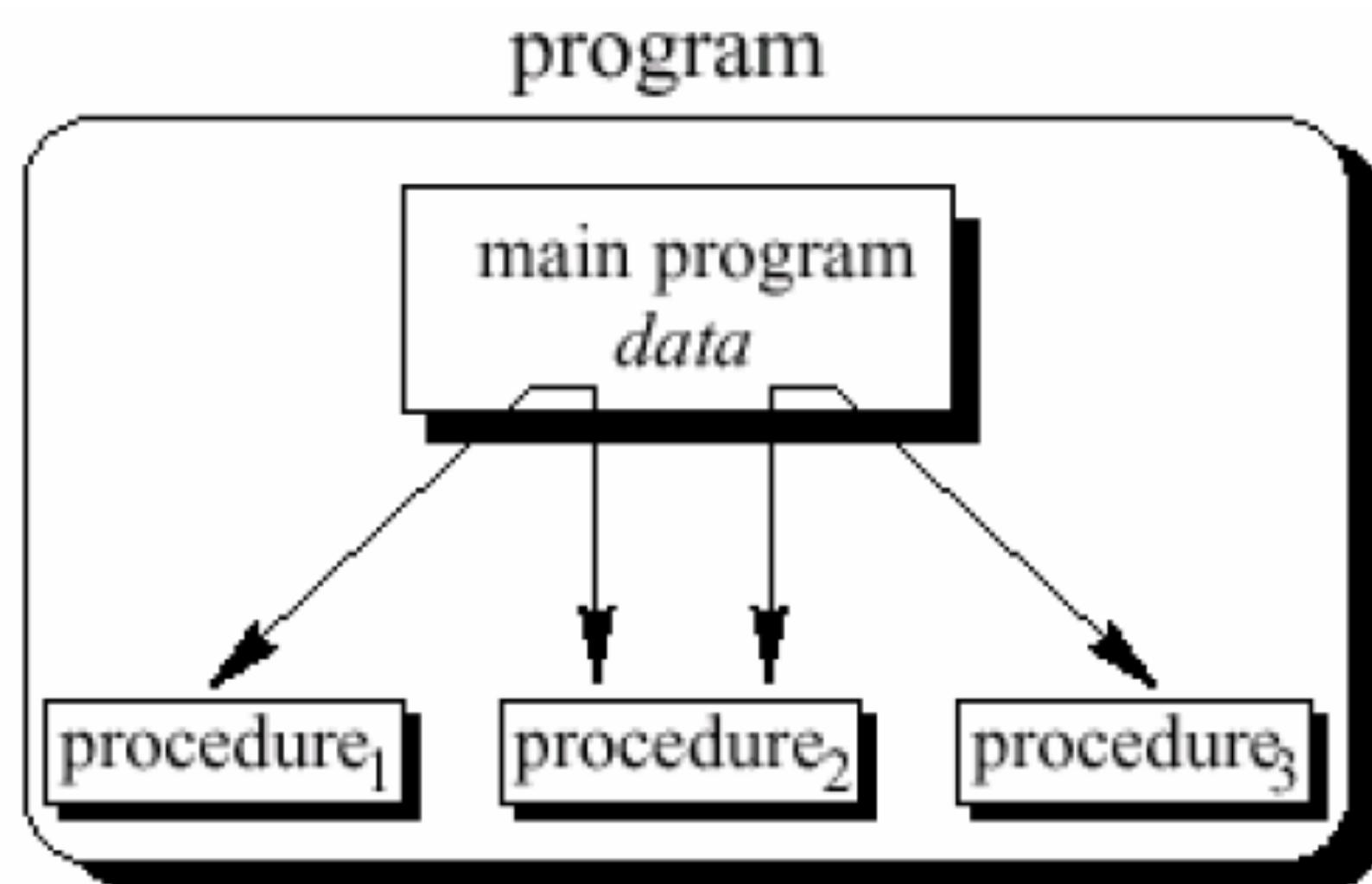


- A chamada de procedimento é utilizada para invocar o procedimento.
  - Transferência de controle
- Após a execução do procedimento, o fluxo de controle retorna para o ponto imediatamente após a chamada





- Os procedimentos são combinados para prover a funcionalidade desejada
- O programa pode ser visto como uma sequência de chamadas de procedimento. Programa principal é responsável por passar os dados para as chamadas individuais para serem processados



**Ex.: C, Pascal, Fortran**

# Programação Estruturada



- As linguagens estruturadas são de entendimento relativamente fácil
  - Por isso são utilizadas em cursos introdutórios
- No entanto, são focadas em **como** uma tarefa deve ser feita
  - E não em **o que** deve ser feito
- Mistura **tratamento de dados** e **comportamento** do programa

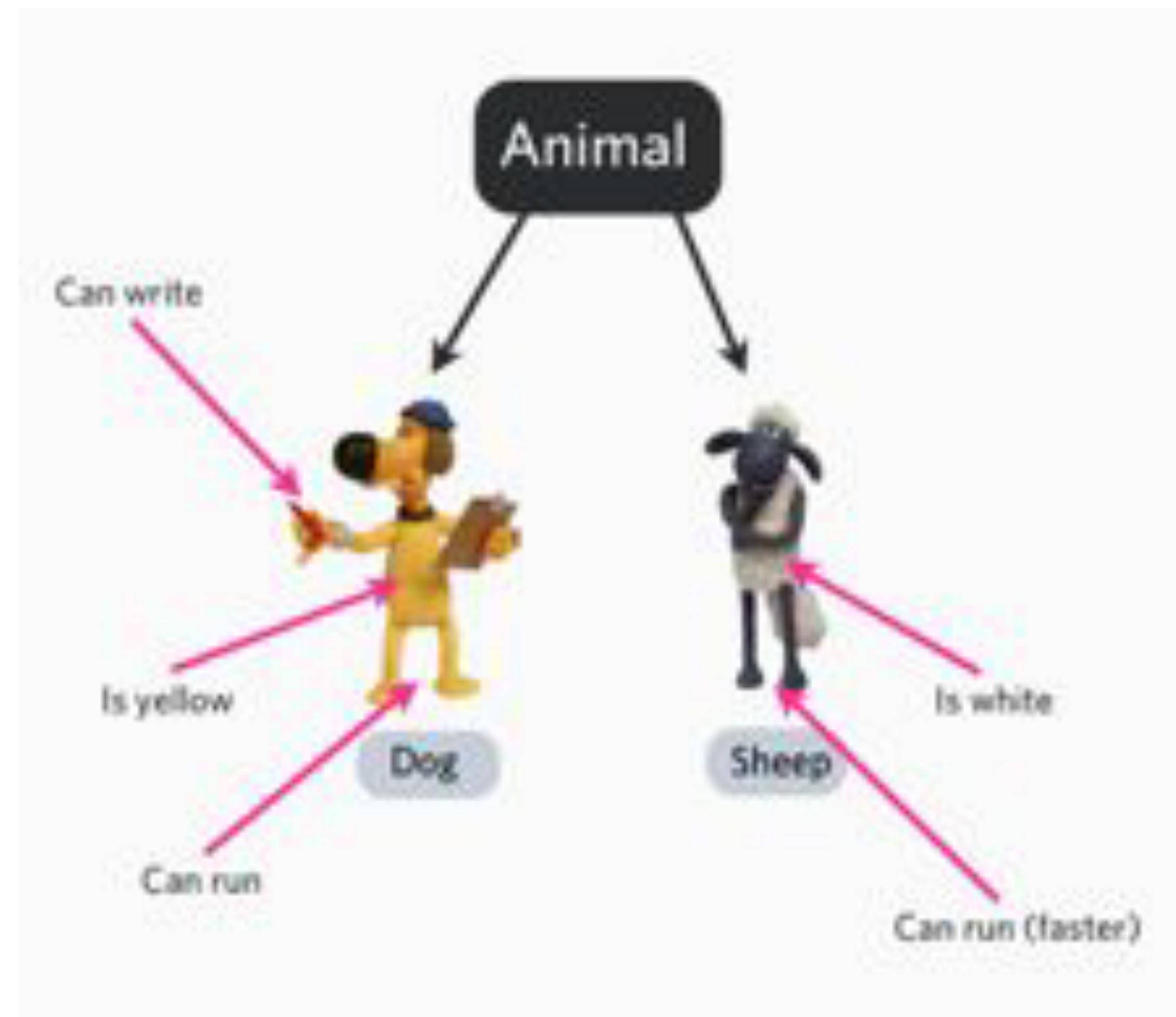
# Programação Estruturada



- A programação estruturada ainda é muito influente
  - Para cada situação uma ferramenta
- Para problemas simples e diretos, ainda é a melhor solução

# Orientação de Objetos

# Orientação a Objetos



# Orientação a Objetos: Histórico



- Crise do software -> dificuldades e consequentes frustrações que o desenvolvimento e a manutenção de softwares trouxeram para as grandes empresas
  - Expressão inicialmente utilizada no fim da década de 60, e início da década de 70
    - Simula 67 (60's);
    - Smalltalk (70's);
    - C++ (80's).
- Surgiu na tentativa de solucionar problemas existentes no desenvolvimento de Softwares **Complexos e Confiáveis** com baixo custo de desenvolvimento e manutenção



- Devido aos requisitos atuais, os softwares têm se tornado cada vez mais complexos e maiores
- Isso tem levado a busca de meios para tornar a tarefa de programação mais produtiva
- Ainda não existe uma resposta definitiva a essa busca, mas há um consenso de que a Programação Orientada a Objetos (POO) consegue produzir resultados mais competitivos do que as outras abordagens
- Os programa OO são mais fáceis de entender, corrigir e modificar

# Orientação a Objetos: Evolução



- Os modelos Orientados a Objetos evoluíram a partir da própria evolução das linguagens de programação
  - Primeiras linguagens de programação
    - Semelhantes à máquina
  - Programas extremamente dependentes de hardware, e de difícil manutenção
  - Idéia da O.O. -> ao invés de programar pensando como a máquina, pode-se programar pensando como humanos

# Orientação a Objetos



- Este paradigma de programação tenta ser a mais óvia, natural e exata possível;
- São conceitos essenciais:
  - Classes e objetos;
  - Atributos, Métodos e Mensagens;
  - Herança e Associação;
  - Encapsulamento;
  - Polimorfismo;
  - Interfaces

# Orientação a Objetos

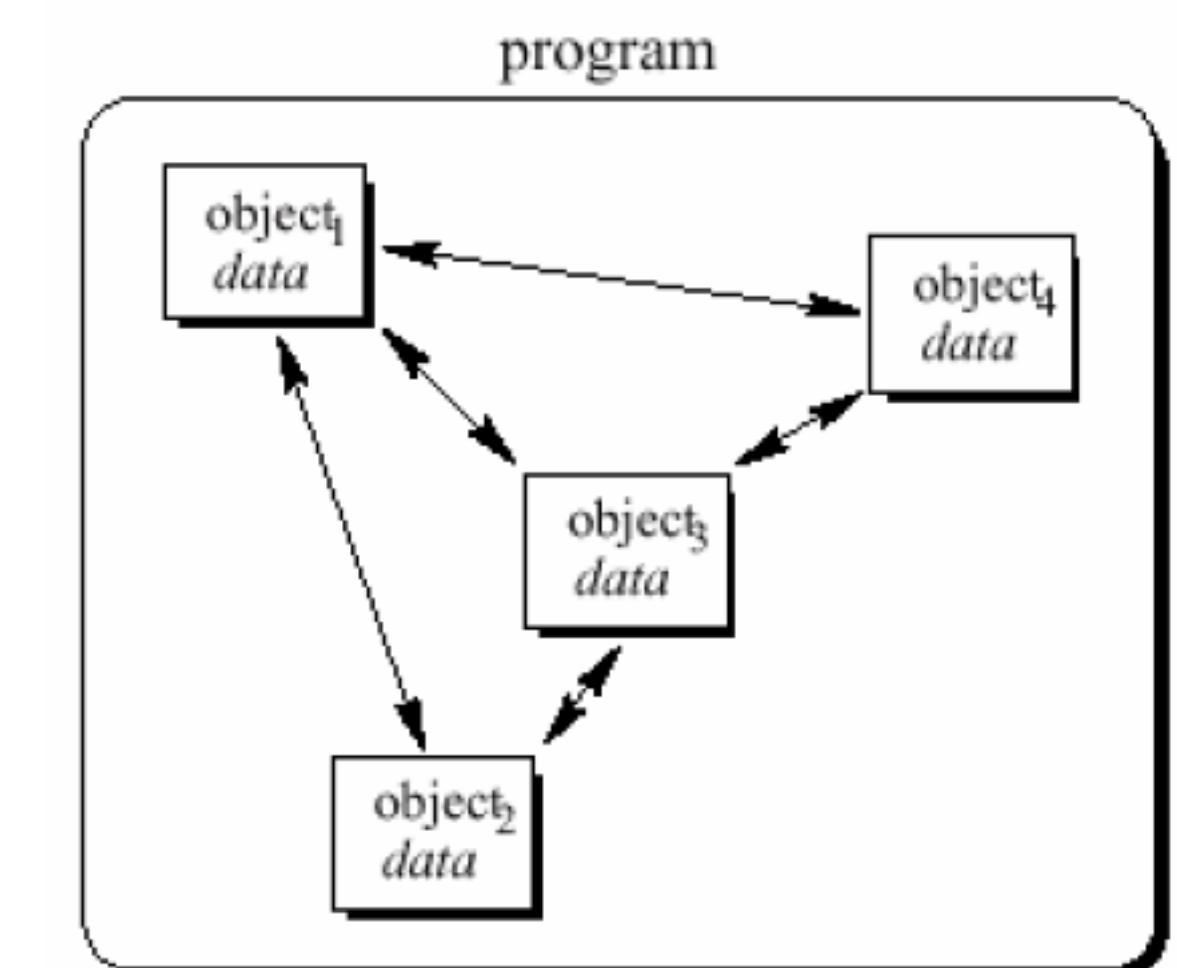


- Como melhor modelar o mundo real utilizando um conjunto de componentes de *software*?
- Considerando que nosso mundo é composto de **objetos**, por quê não utilizá-los?
- A ideia é modelar utilizando **objetos**, determinando como eles devem se comportar e como deve interagir entre si.

# Orientação a Objetos

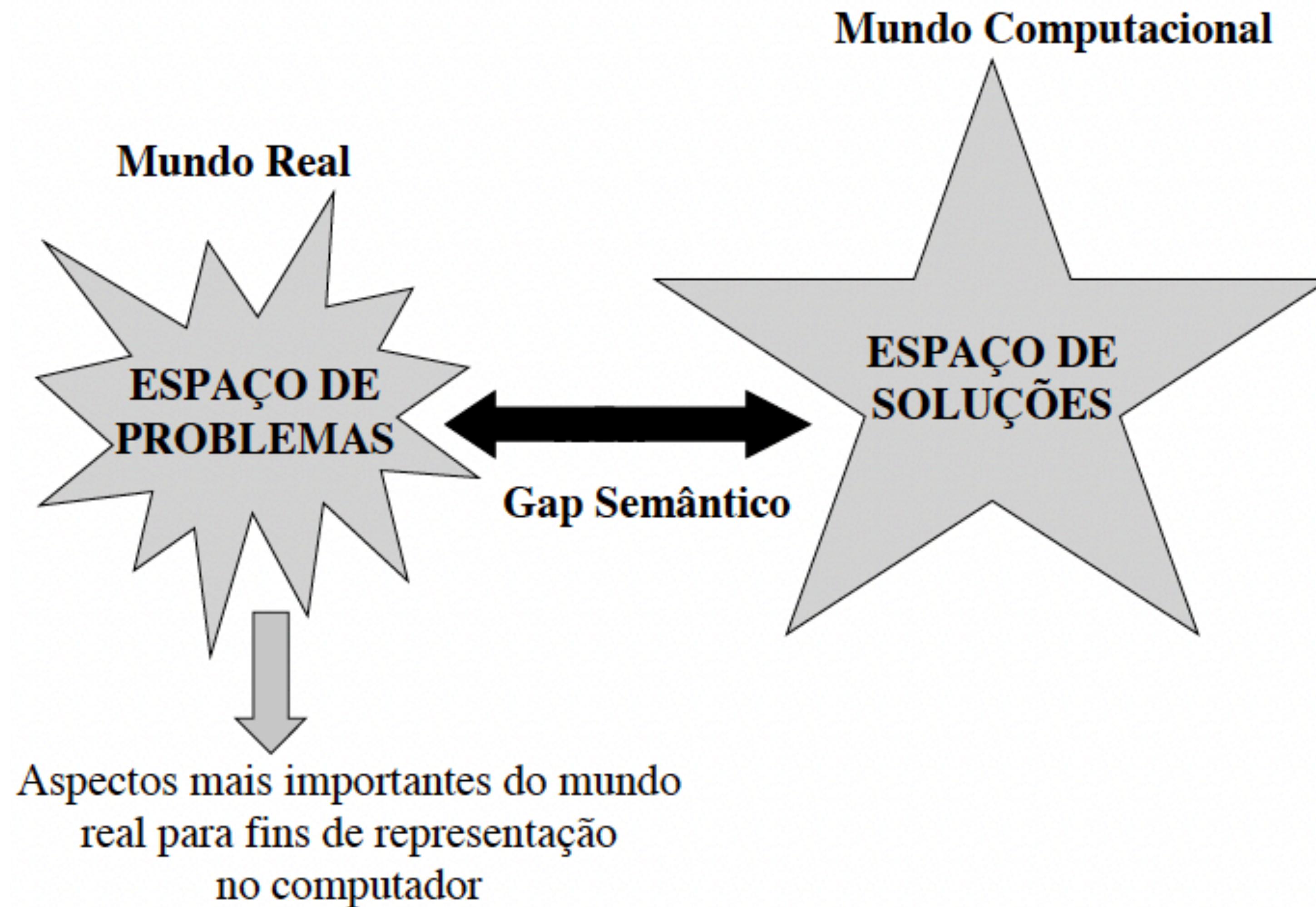


- POO é caracterizado pelo uso de um conjunto de objetos interagentes, cada qual responsável pelo gerenciamento de seu estado interno
  - Os objetos interagem uns com os outros através da *troca de mensagens*
  - Cada objeto é responsável pela inicialização e destruição de seus dados internos



Ex.: Eiffel, SmallTalk, C++, Java, C#

# Orientação a Objetos: Abstração



# Orientação a Objetos: Abstração

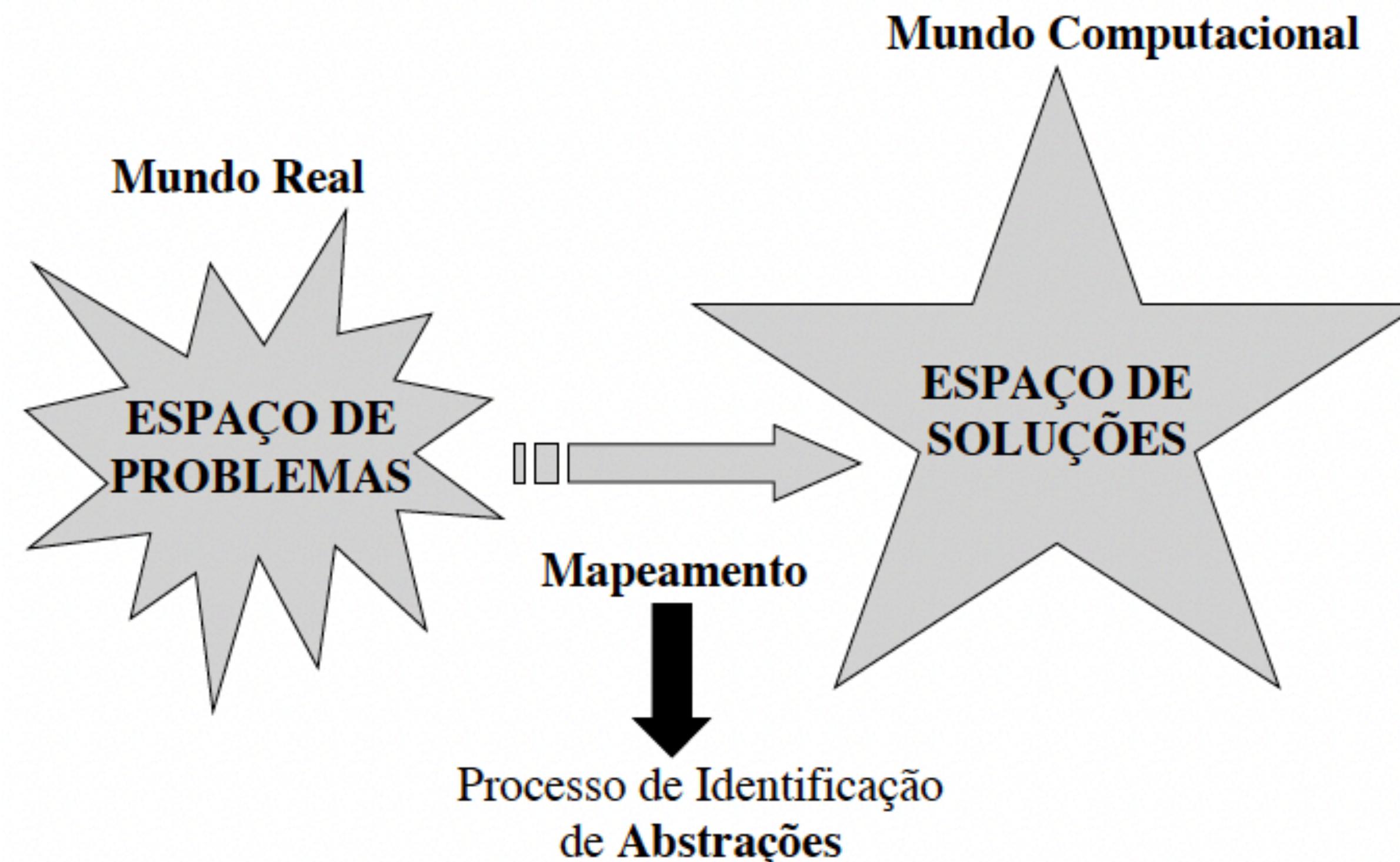


- Gap Semântico: diferença entre espaço de problemas e soluções
- Todo software representa um **Modelo** de um problema do mundo real (Espaço de Soluções)
- A construção de um software envolve um processo de mapeamento de **objetos** pertencentes ao espaço de problemas para o espaço de soluções, de tal maneira que **operações** sobre essas representações abstratas correspondam a operações do mundo real
- **EVIDENTE**: quanto mais próximo (conceitualmente) estiver o espaço de soluções do espaço de problemas, mais fácil será: desenvolvimento da aplicação

# Orientação a Objetos: Abstração

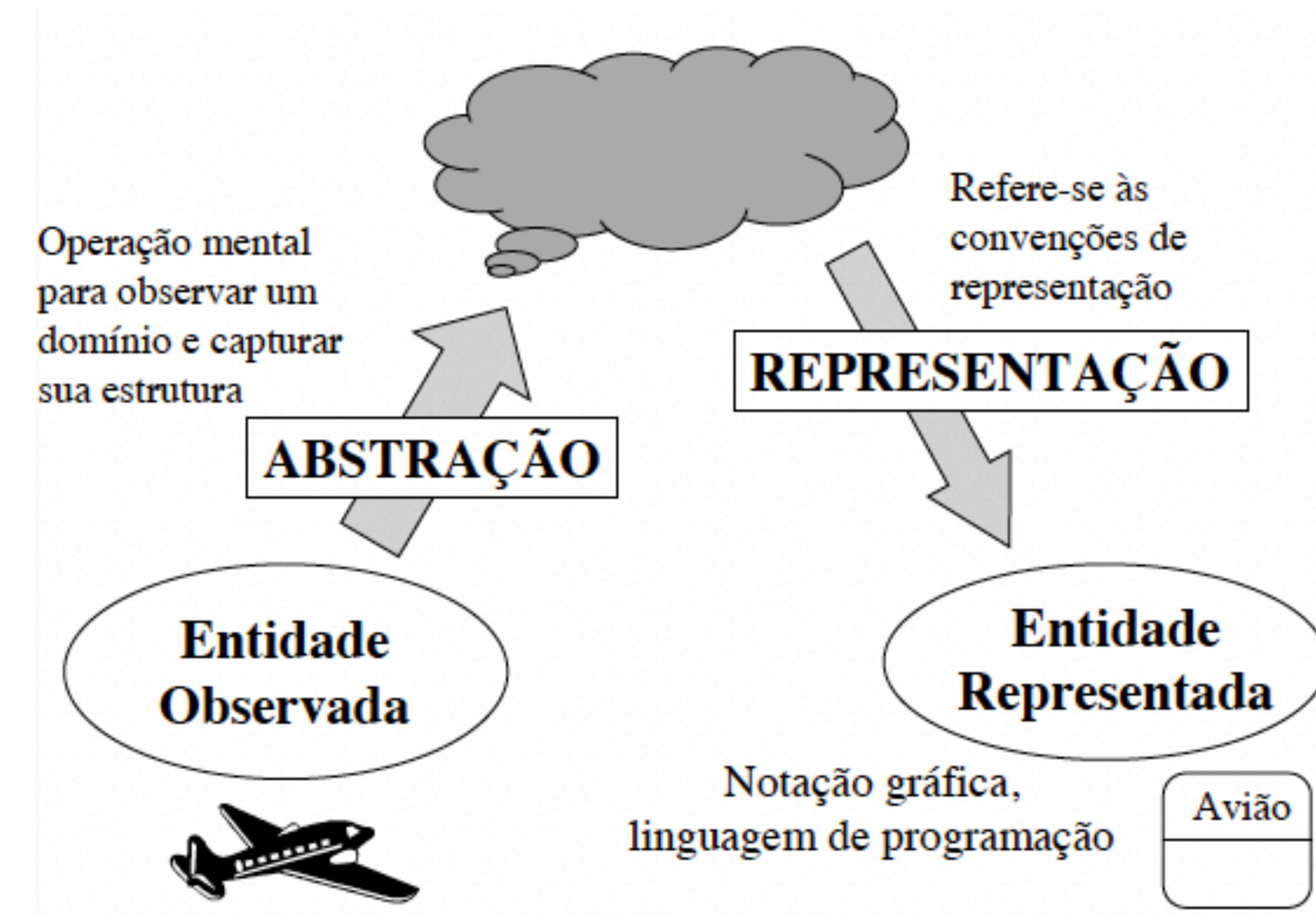


- Objetivo do Paradigma de Orientação a Objeto: **DIMINUIR O GAP SEMÂNTICO!**



- Se essas abstrações não tiverem uma expressão direta (ou próxima) do mundo computacional, a complexidade da solução será aumentada

# Orientação a Objetos: Abstração



# Orientação a Objetos: Abstração

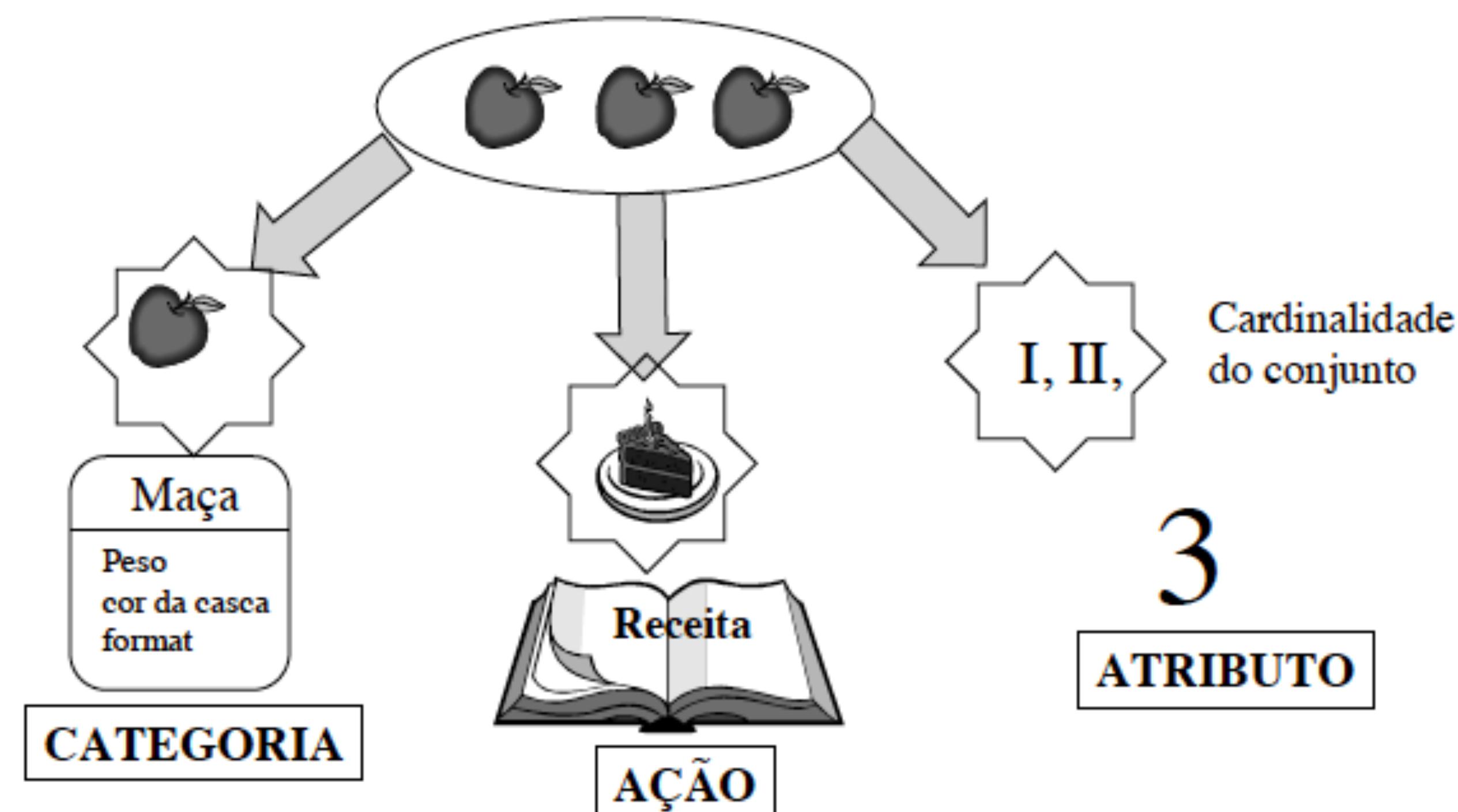


- Mecanismo utilizado na análise de um domínio
- Através dela, o indivíduo observa a realidade e dela abstrai entidades, ações, etc. consideradas essenciais para uma aplicação, excluindo todos os aspectos julgados irrelevantes
- Exemplo:
  - **Fotografia por satélite** (imagem da realidade), despida de alguns aspectos (por exemplo, cor, movimento)
  - Da foto, pode-se abstrair um **mapa**, que elimina diversas propriedades da foto (detalhes particulares de um edifício ou praça)
  - O mapa pode ser a base para abstrair um **grafo**

# Orientação a Objetos: Abstração



- “A beleza está nos olhos de quem a vê”
- Diferentes abstrações a partir de um mesmo objeto do mundo real





- Idéia básica:
  - “Nós percebemos o mundo como uma coleção de objetos que interagem entre si”
- Permite que objetos do mundo real sejam mapeados em **Objetos** no computador
- Visão do **DOMÍNIO** da aplicação é composto por **OBJETOS** que se comunicam através de **MENSAGENS**



## ○ Na programação estruturada

- Procedimentos são implementados em blocos e a comunicação entre eles se dá pela passagem de dados
- Um programa estruturado, quando em execução, é caracterizado pelo acionamento de procedimentos cuja tarefa é a manipulação de dados

## ○ Na programação orientada a objetos

- Dados e procedimentos são encapsulados em um só elemento denominado **objeto**
- O estabelecimento de comunicação entre objetos (envio e recebimento de mensagens) caracteriza a execução do programa



## ● Vantagens da POO em relação à programação estruturada

- Maior índice de reaproveitamento de código
- Maior facilidade de manutenção
- Menos código gerado
- Maior confiabilidade no código
- Maior facilidade de gerenciamento do código (reduz grandes problemas para problemas menores);
- ...



- **Programação estruturada** é baseada na definição de ações (funções)

- Verbos
- Dados e programa: entidades diferentes e separadas
- Duas partes: dados e funções

- **Programação orientada a objetos** é focada na definição de coisas ou objetos

- Substantivos
- Mais próximo da percepção que o programador tem do mundo real
- Dados e procedimentos: parte de um só elemento básico (Objeto)
- Duas partes (dados e funções) PARA CADA OBJETO



## Programação Estruturada

DADOS

FUNÇÕES

## Programação Orientada a Objetos

**Objeto**

Atributos  
(dados)

Métodos  
(Funções)

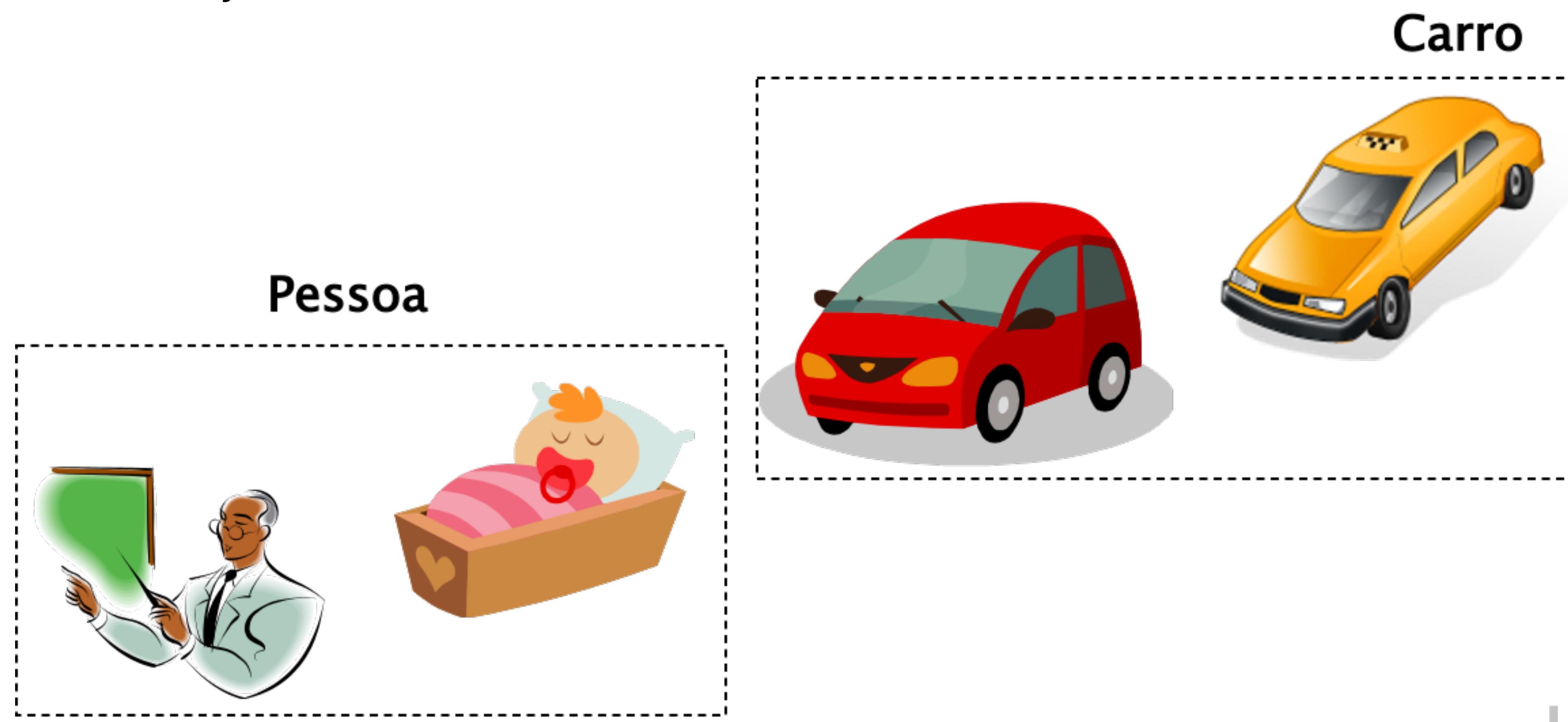


# Classes e Objetos

# Objetos e Classes



- Mundo real está repleto de objetos
- Agrupamos objetos semelhantes em classes





- Um **objeto** é uma entidade que formaliza o modo pelo qual compreendemos algo no domínio do problema
  - Reflete a capacidade do sistema de **guardar informações** sobre o elemento abstraído e **interagir** com ele
  - Entidade o mais próximo possível das entidades do mundo real - aquilo que é tangível ou visível
  - Dessa forma, os objetos são uma forma de diminuir o **gap semântico**
    - Diferença entre o domínio de problemas e soluções



- A um objeto sempre estarão associados

- *Estado*

- Definido pelas propriedades (**atributos**) que ele possui e pelos valores que elas estão assumindo

- *Comportamento*

- Definido pela forma como ele age e reage, em termos de mudança de seu estado e o relacionamento com os demais objetos do sistema (**métodos**). Comportamento através de métodos, que são correspondentes às funções em programação estruturada.

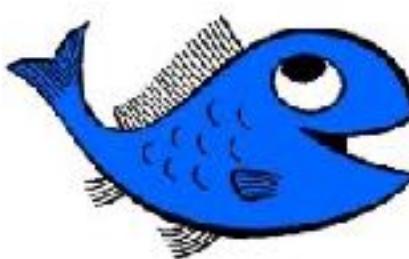
- *Identidade*

- Cada objeto é único



## ○ Exemplos de objetos no mundo real

- Cachorro, mesa, televisão, bicicleta, lâmpada, ...
- Lâmpada
  - Atributos: ligada, desligada
  - Métodos: ligar, desligar
- Rádio
  - Atributos: ligado, desligado, volume, estação, ...
  - Métodos: ligar, desligar, aumentar/abaixar volume, sintonizar,...
- Cachorro
  - Atributos: nome, cor, raça, peso, ...
  - Métodos: latir, morder, correr, dormir, ...
- Bicicleta - ??





- Bicicleta
  - Atributos: marcha, cadênciā do pedal, velocidade, ...
  - Métodos: mudar marcha, mudar cadênciā do pedal, frear, ...
- Objetos podem conter objetos como atributos?
  - Sim! Exemplo?
  - Objeto **Pessoa** pode possuir um objeto **Carro** (associação)



## ○ Atributos:

- Representam um conjunto de informações, ou seja, elementos de dados que caracterizam um objeto
- Descrevem as informações que ficam escondidas em um objeto para serem exclusivamente manipulado pelas operações daquele objeto
- São variáveis que definem o estado de um objeto, ou seja, são entidades que caracterizam os objetos
- Cada objeto possui seu próprio conjunto de atributos



- Resumindo:

- Pacote de informações (**atributos**) e a descrição de suas operações (**métodos**), de modo que elas são intrínsecas ao seu domínio e este é formado pelos elementos que o caracterizam

- Exemplo:

- **Objeto:** Pessoa
- **Atributos:** Nome, Data de Nascimento, Cor
- **Métodos:** Acordar, Comer, Beber, Dormir

Objeto
Nome
Data_Nasc
Cor
Idade
Acordar()
Comer()
Beber()
Dormir()



- Mais exemplos de Objeto:

- Uma fatura
- Uma organização
- Uma tela com a qual o usuário interage
- Um desenho de engenheiro
- Um avião
- Um vôo de avião
- Uma reserva num avião
- Um processo de atendimento de pedidos
- Um computador

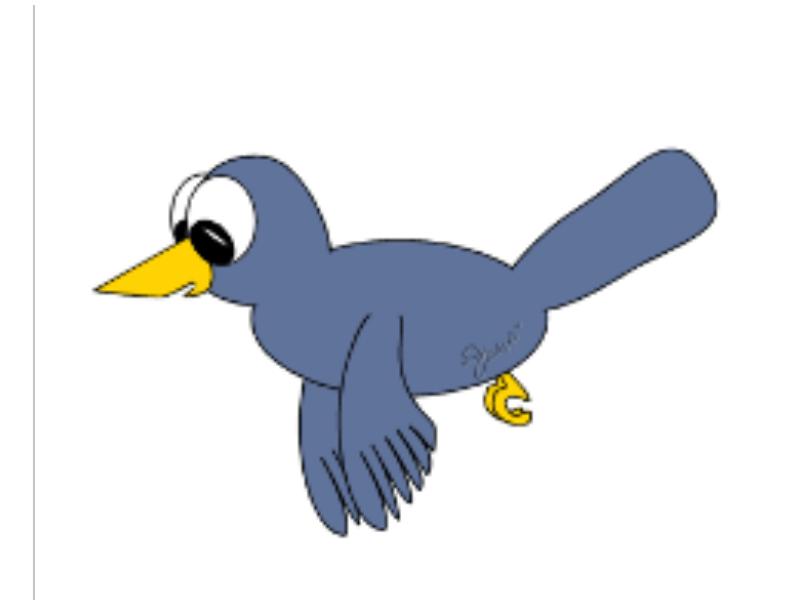


- Empacotar o código em objetos individuais fornece:
  - Modularidade
    - Objetos são independentes
  - Ocultação de informação
    - Os detalhes da implementação de um objeto permanecem ocultos
  - Reuso
    - Objetos podem ser reutilizados em diferentes programas
  - Plugabilidade
    - Objetos podem ser substituídos em um programa, como peças

# Classes de objetos



○ Quantas classes temos aqui?





- Uma classe descreve um conjunto de objetos semelhantes através de:
  - Atributos e métodos que resumem as características comuns de vários objetos
- Diferença entre classe e objeto
  - Objeto constitui uma entidade concreta com tempo e espaço de existência
  - Classe é tão-somente uma abstração
- Em termos de programação, definir uma classe significa formalizar um tipo de dado (TAD) e todas as operações associadas a esse tipo, enquanto declarar objetos significa criar variáveis do tipo definido



- Classe é um template (“forma ou receita”) para a criação de objetos
  - Uma classe especifica os tipos de dados (atributos) e operações (métodos) suportadas por um conjunto de objetos
- Todo objeto é uma *instância* de uma classe
  - Criação de um objeto a partir de uma classe é chamada de **instanciação**
  - Todas as instâncias de uma classe têm valores próprios para os atributos especificados na classe, assim os **objetos de uma classe** se **diferenciam** pelos **valores de seus atributos**
  - É muito comum que em um programa existam várias instâncias de uma mesma classe
    - O que diferencia cada uma?

# Classes vs Objetos



= **Objeto**

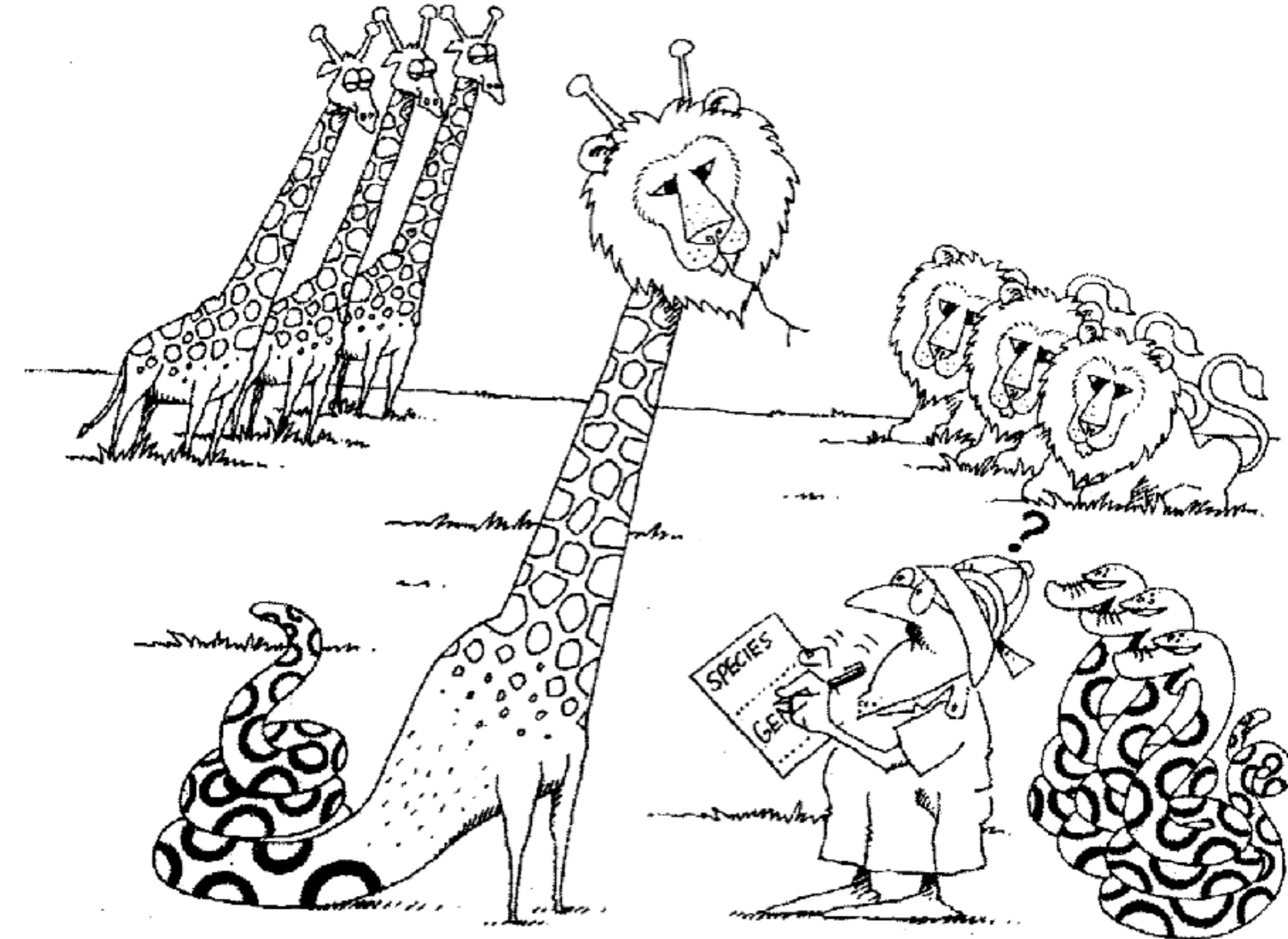


= **Classe**

# Classes vs Objetos



- A Classe é o tipo de Objeto



Fonte: livro “Object-Oriented Analysis and Design with Applications”

# Classes vs Objetos

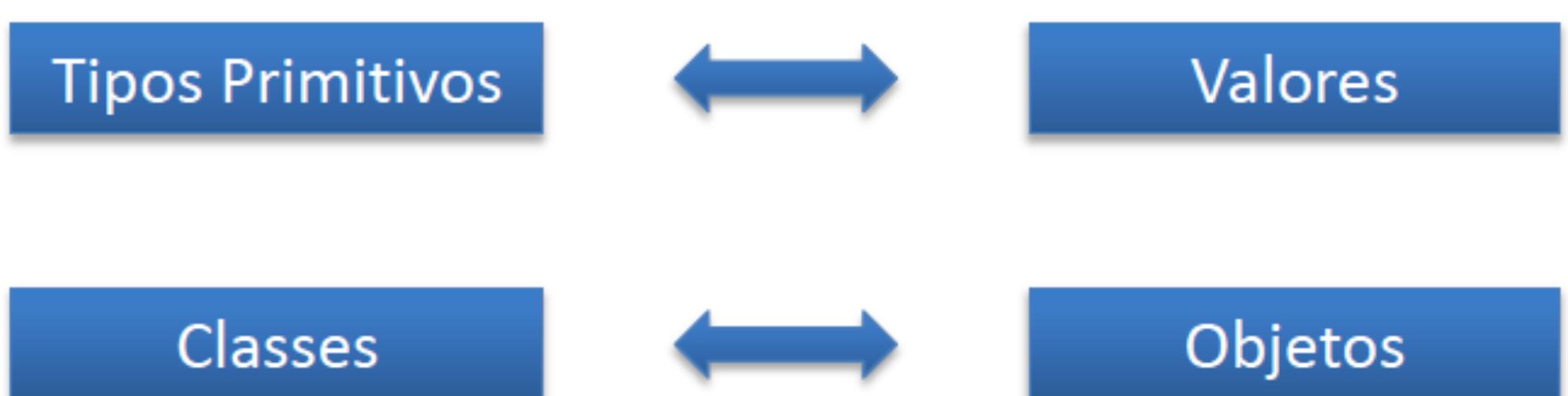


- Valores têm tipos primitivos

- 123 é um valor inteiro
- True é um valor booleano
- 3,1 é um valor real

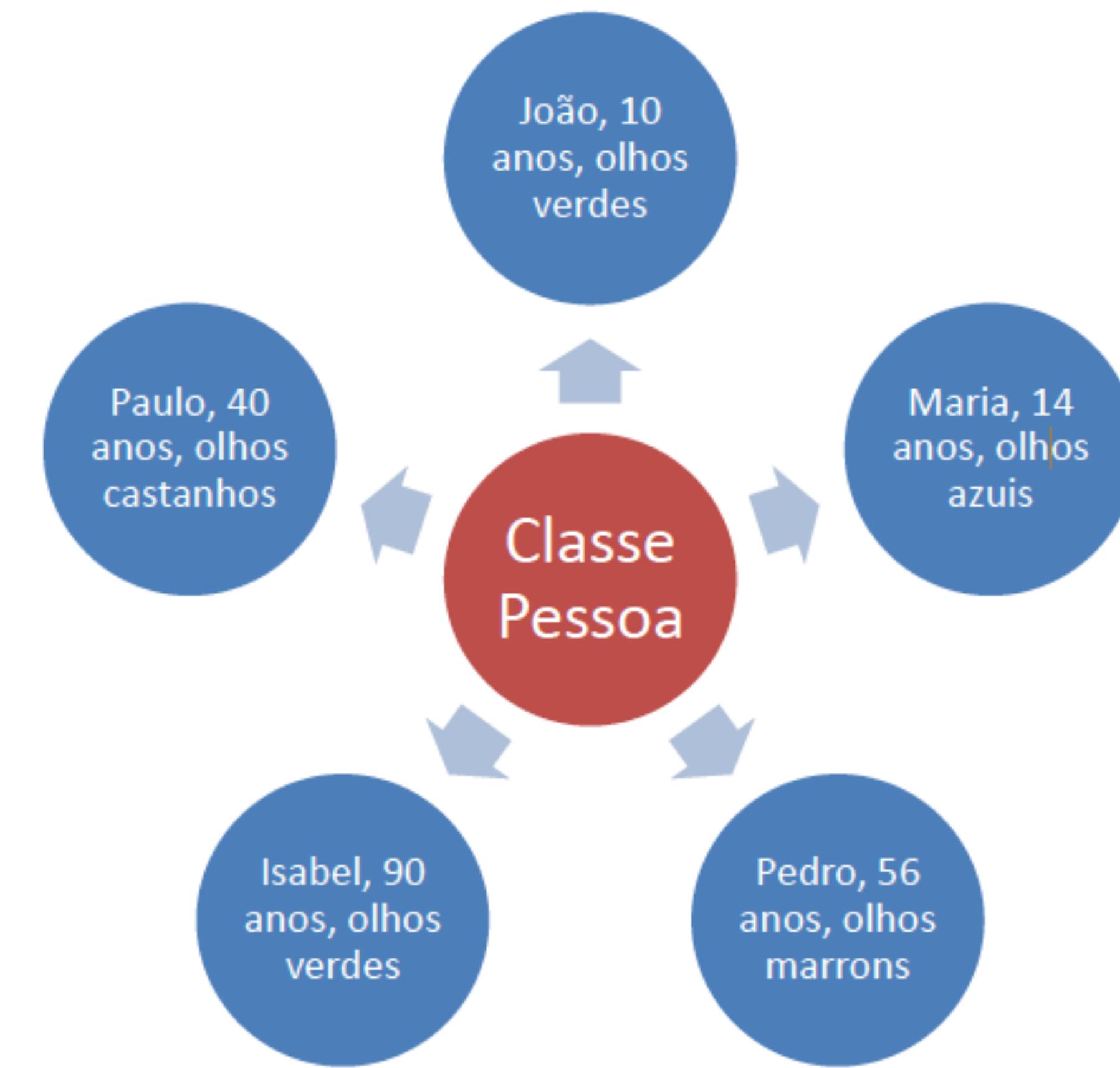
- Objetos pertencem a classes

- João, Pedro e Paulo são da classe Pessoa
- Fusca e Ferrari são da classe Carro
- São Paulo e Palmeiras são da classe Time





- Uma classe é uma forma, capaz de produzir objetos
- Os programadores criam classes, as classes instanciam objetos





- Criem uma classe do tipo Discente/Aluno



- Cada **instância** é formada por **valores de atributos únicos** e um comportamento comum definido pela classe
  - **Inúmeras instâncias** podem ser criadas a partir de uma classe
  - O estado de cada instância é representado pelos valores de seus atributos, que podem ser diferentes
  - Diferentes objetos de uma mesma classe possuem suas próprias cópias de cada atributo
    - A menos que isso seja desejado e explicitamente declarado
    - Neste caso, um único atributo pode ser compartilhado para todas as instâncias



- Os métodos são operações que podem ser executadas pelos objetos
  - Valores dos atributos são (normalmente) acessados através dos métodos definidos pela classe
  - *Information-hiding*
  - O serviço oferecido pelos método é um comportamento específico, residente no objeto, que define como ele deve agir quando exigido



- Uma classe pode definir o tipo de acesso à seus membros (atributos e métodos)

- **Público**

- Atributo ou método da classe pode ser acessado por todas as demais entidades do sistema

- **Protegido**

- Atributo ou método da classe pode ser acessado somente por classes da mesma hierarquia e mesmo pacote

- **Privado**

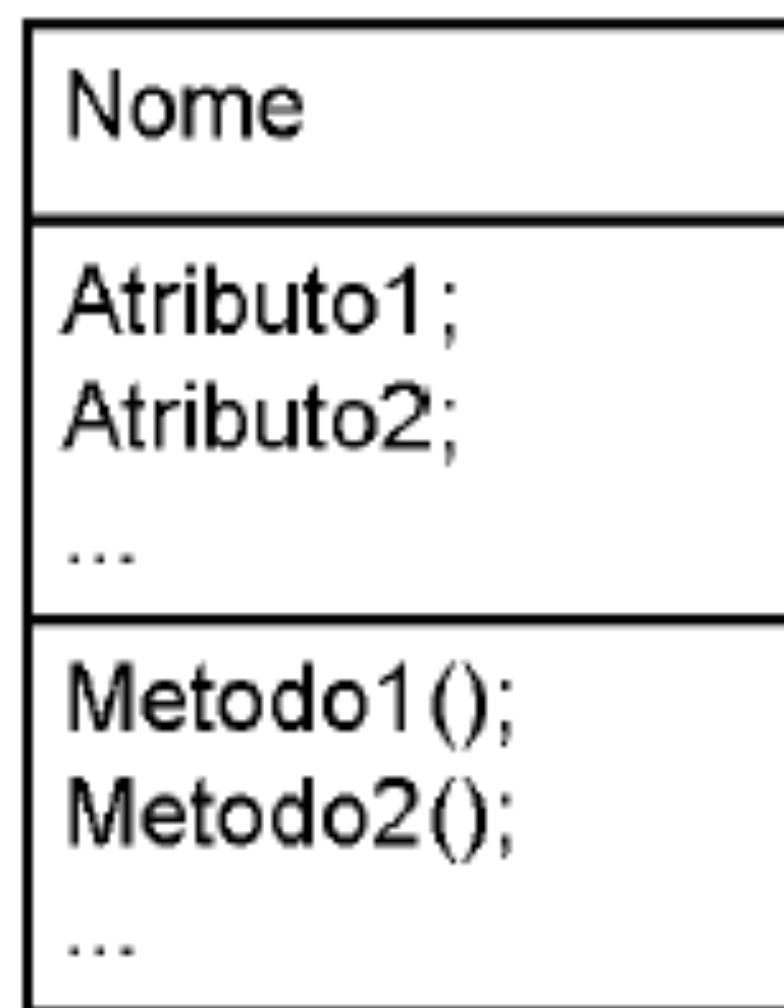
- Atributo ou método da classe pode ser acessado somente por métodos da própria classe



- A escolha dos tipos de acesso é muito importante na POO
  - Define o escopo dos atributos e métodos
- Em geral, atributos são declarados **privados**
  - Métodos da própria classe são responsáveis por modificar e recuperar o estado dos atributos
  - Tais métodos são públicos
  - Setters e getters
  - Garantem a estabilidade e segurança
  - *Information-hiding*



- A notação gráfica de uma classe permite visualizar uma abstração independente de qualquer linguagem de implementação específica, dando ênfase às partes mais importantes: seu nome, atributos e métodos (operações)
- Também é possível representar tipos de acesso



# Relembrando...



- Objetos
  - Atributos + Métodos
- Objetos x Classes
- Instâncias
- Acesso: público, protegido, privado



- O paradigma orientado a objetos define alguns princípios básicos que devem ser seguidos
  - Abstração
  - Encapsulamento
  - Herança
  - Polimorfismo
  - Mensagens

# Abstração



- Consiste em identificar os requisitos de software e modelá-los em classes
  - Ignorar aspectos não-relevantes, concentrando-se apenas nos aspectos principais do problema
- Classes são abstrações de conceitos
- Consiste basicamente no processo de retirar do domínio do problema os detalhes relevantes e representá-los na linguagem de solução (ex.: Java)
- Classes (objetos) podem ser qualquer entidade reconhecida como um elemento da solução
  - Objeto real ou não



- Exemplos

- Quadrado, Livro, Televisão, Prontuário, ...
- ContaCorrente, Conversor, Processo, ...

- Quais seriam os atributos e métodos dessas classes?

# Encapsulamento



# Encapsulamento



f1-blog.co.uk

# Encapsulamento



- A propriedade de implementar dados e procedimentos correlacionados em uma mesma entidade recebe o nome de encapsulamento
- A ideia por trás do encapsulamento é a de que um sistema orientado a objetos **não deve depender de sua implementação interna**, e sim de sua interface
  - *Information-hiding*
    - Visa esconder detalhes de implementação



# Encapsulamento



- Os métodos definem o estado interno de um objeto
  - E servem como mecanismo primário de comunicação entre objetos
- Esconder o estado interno e requerer que toda interação seja feita através de métodos é chamado de **encapsulamento de dados**
  - Princípio fundamental de OO





- Mantendo o estado e provendo métodos para alterar o estado, quem determina como o mundo pode interagir com o objeto é o próprio objeto
  - O objeto está no controle;
- Exemplo: objeto **Bicicleta**
  - Atributos (estados) **não** são alterados diretamente pelas outras entidades
  - Métodos da própria classe são definidos para fazê-lo
    - Permite controle total de como os atributos variam
    - Ex. Limite para número de marchas



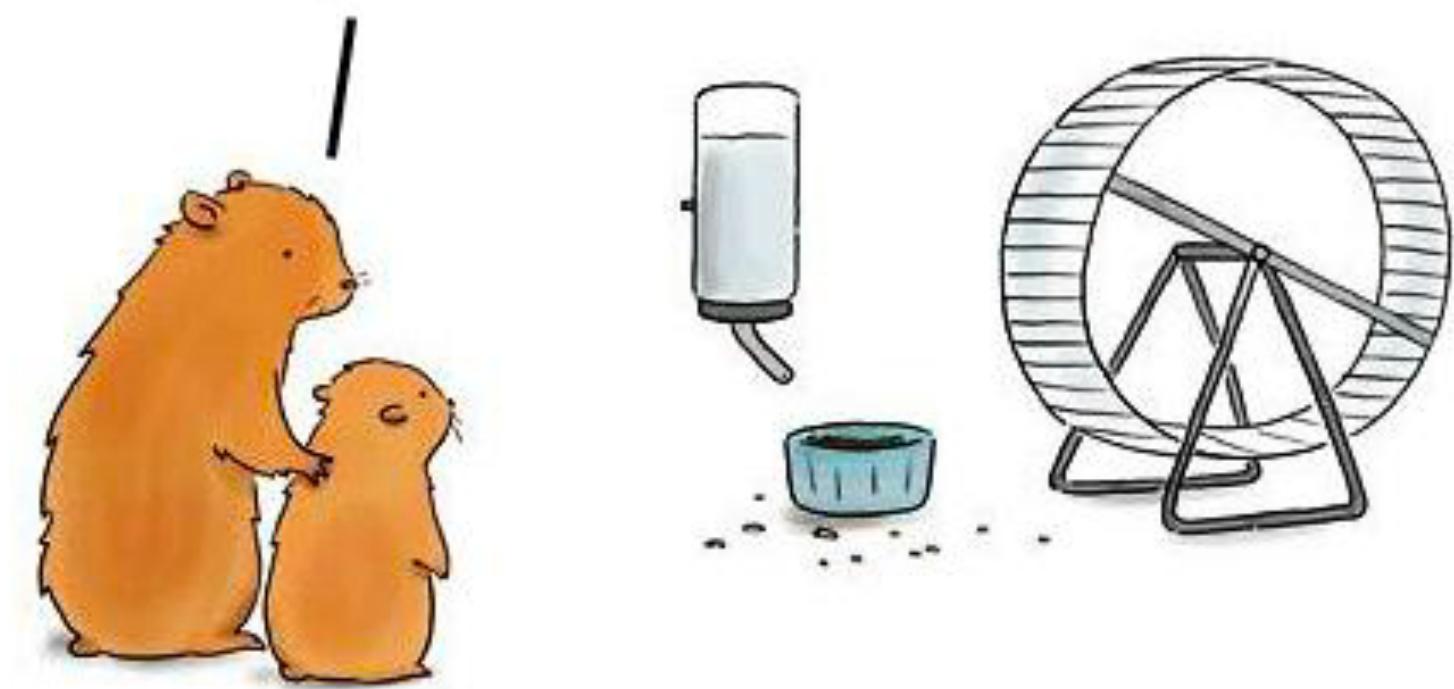


## ● Transparência

- Não importa como os métodos são implementados
- Para as outras entidades, o importante é saber como se comunicar com o objeto
  - Quais métodos estão disponíveis
  - Assinatura dos métodos
    - **Interface**
- Isso permite que a implementação de um método seja facilmente reescrita, sem prejuízo para as outras entidades
  - Ex: carros

# Herança

UM DIA TUDO ISSO  
SERÁ SEU.

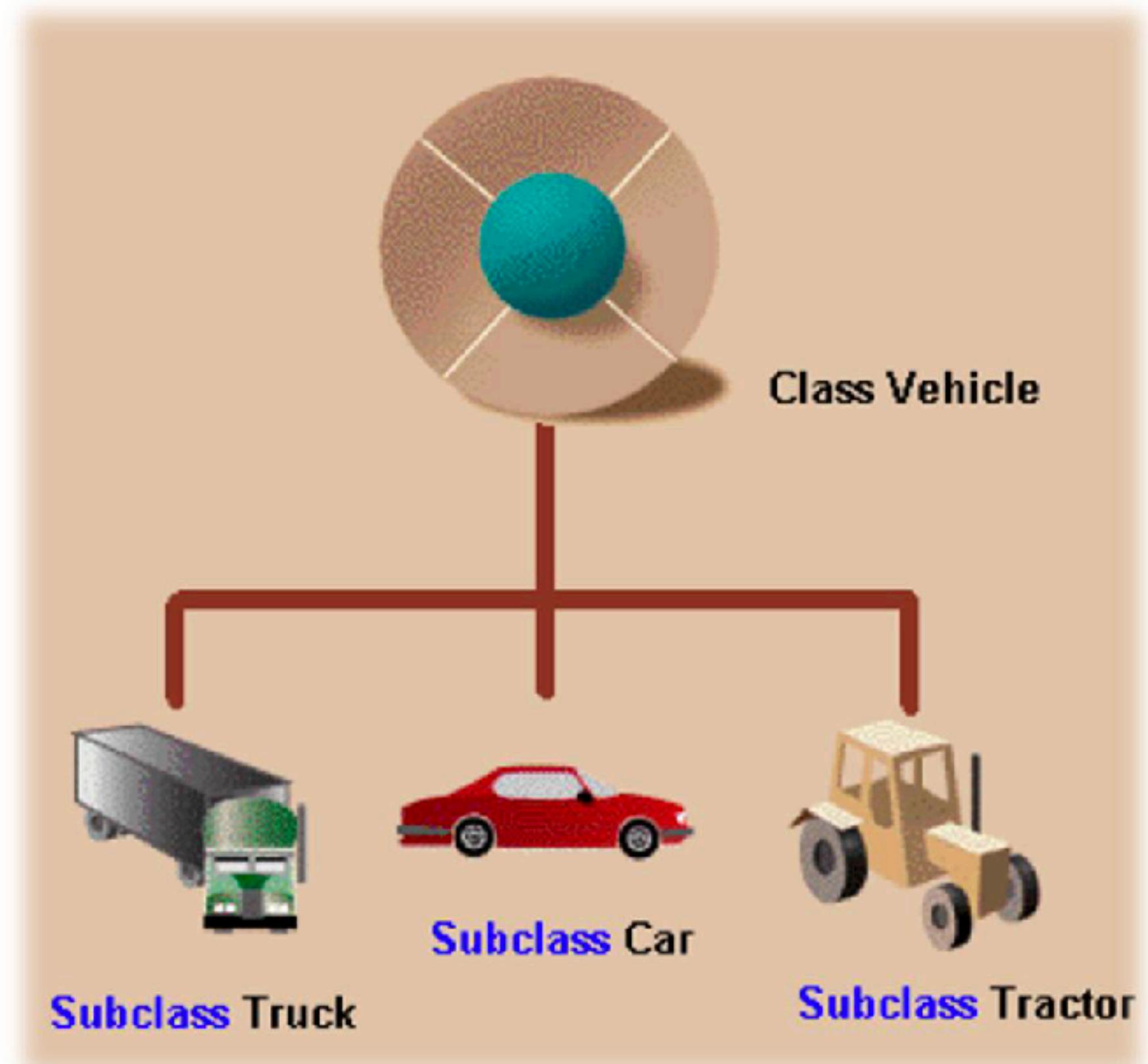




- Permite a hierarquização das classes em um sistema
- Uma classe mais especializada (sub-classe ou classe-derivada) herda as propriedades (**métodos e atributos**) de uma classe mais geral (super-classe ou classe-base)
- Uma sub-classe pode sobrescrever o comportamento de uma super-classe (polimorfismo)
- Promove reuso



- Novos atributos e métodos podem ser definidos nas sub-classes, além dos herdados



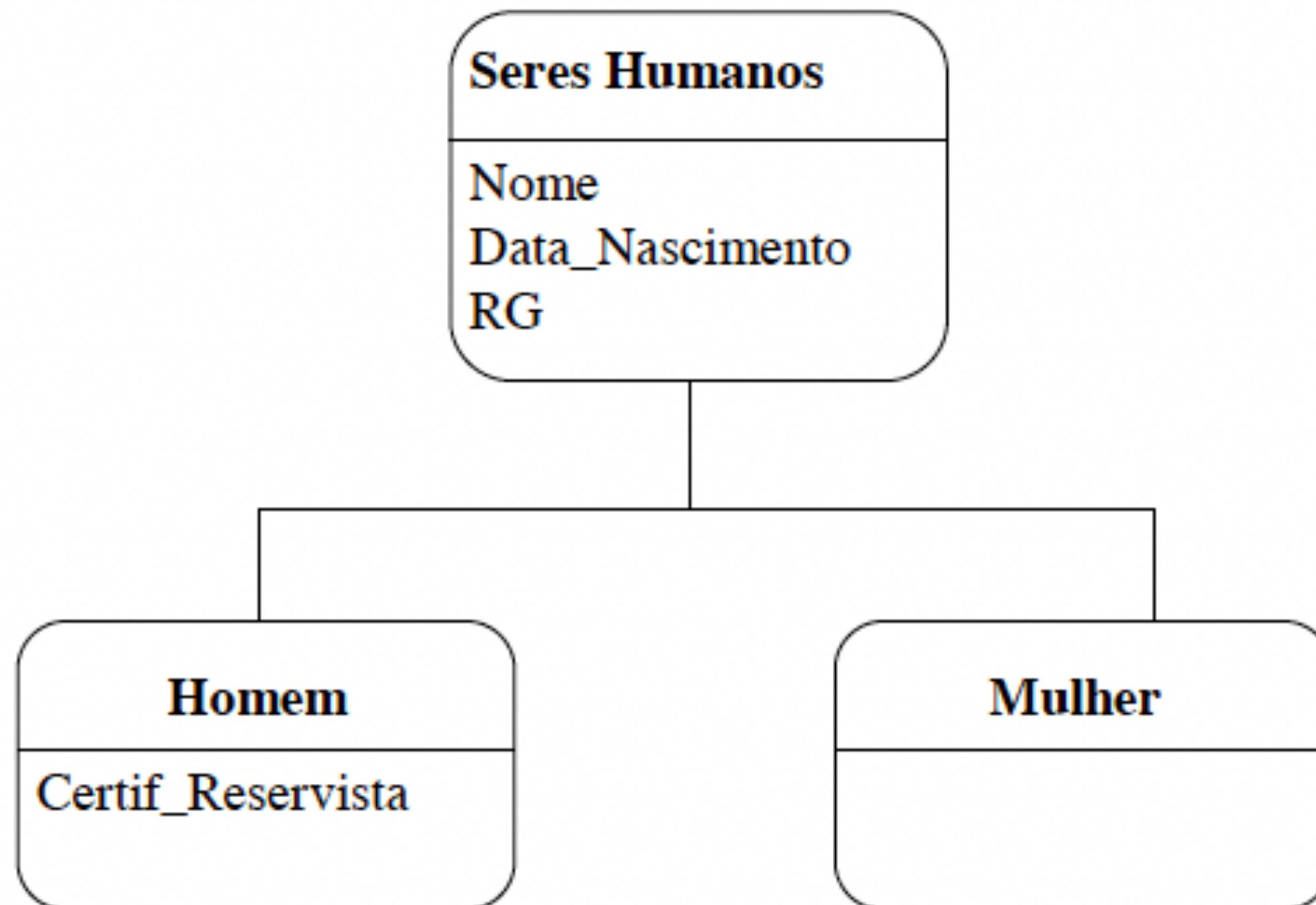


## ○ Exemplo com a classe **Bicicleta**

- Diferentes modelos de bicicleta possuem algumas características em comum
- Porém, cada uma terá características que as distinguem
  - **TandemBikes** possuem dois assentos e dois guidões
  - **RoadBikes** possuem guidão mais baixo
  - **MountainBikes** possuem uma roda dentada a mais
- Todos os campos e métodos de **Bicycle** serão herdados
- Apenas as partes específicas de cada um deverão ser codificadas

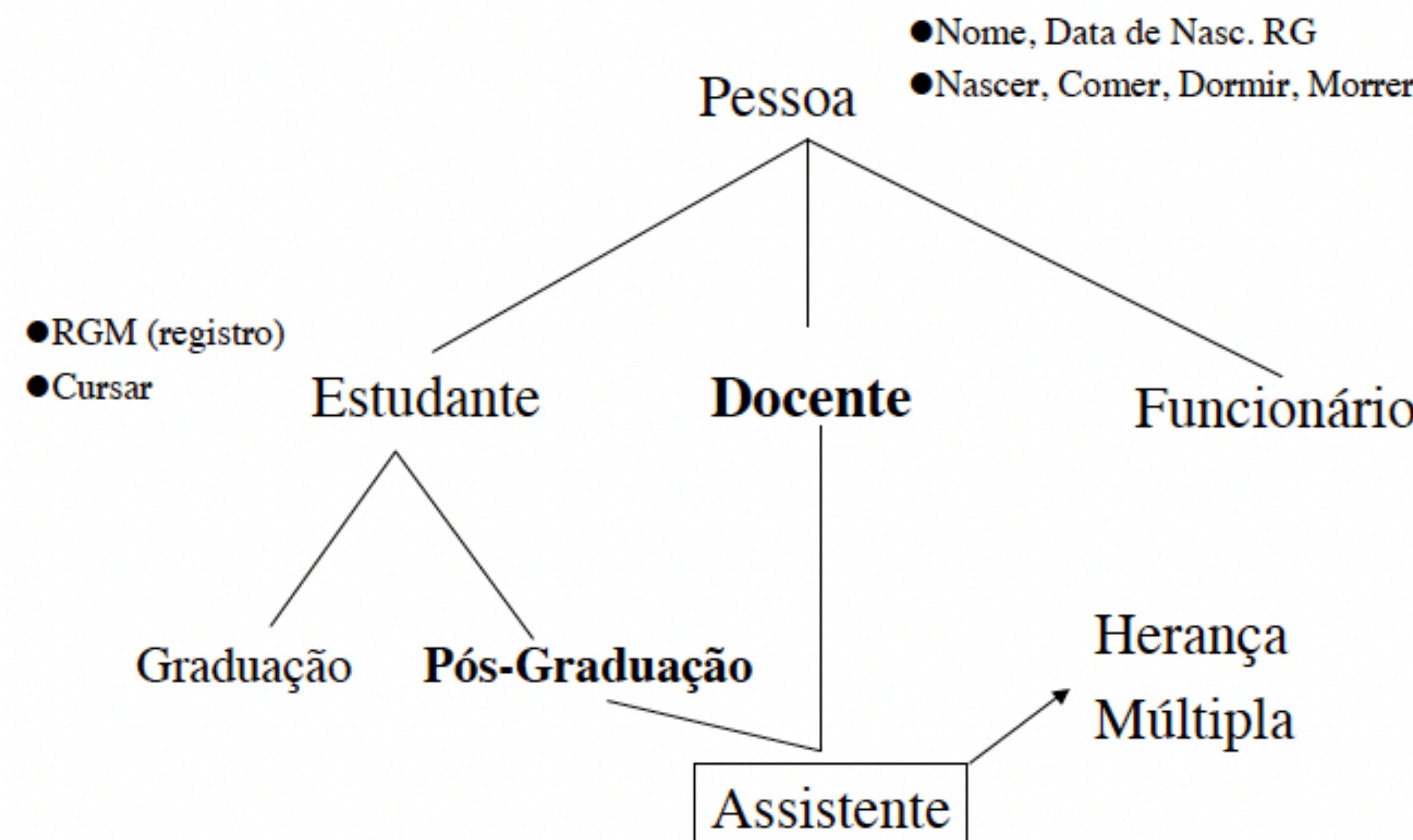


## ● Outros exemplos

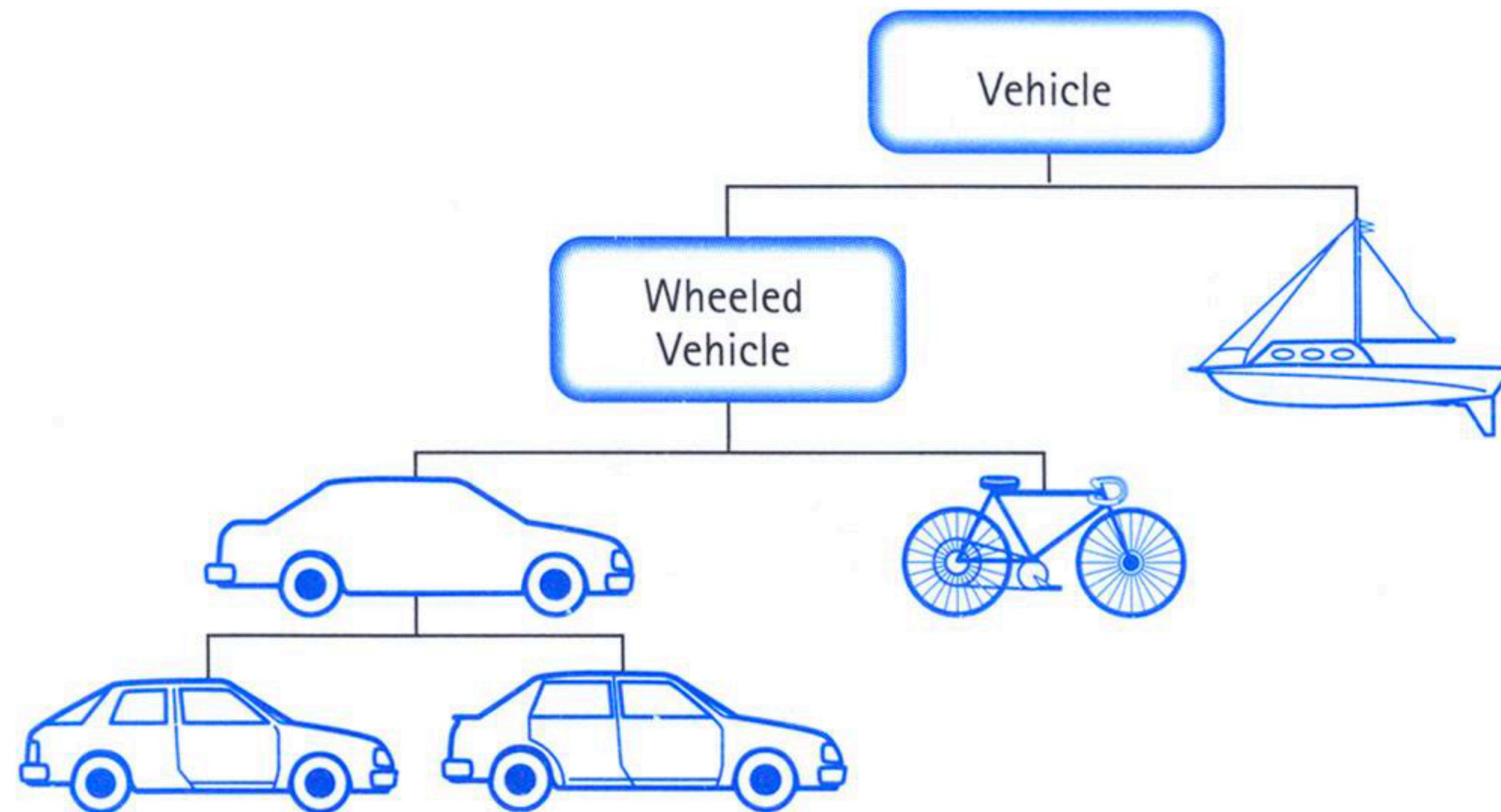




## ● Outros exemplos

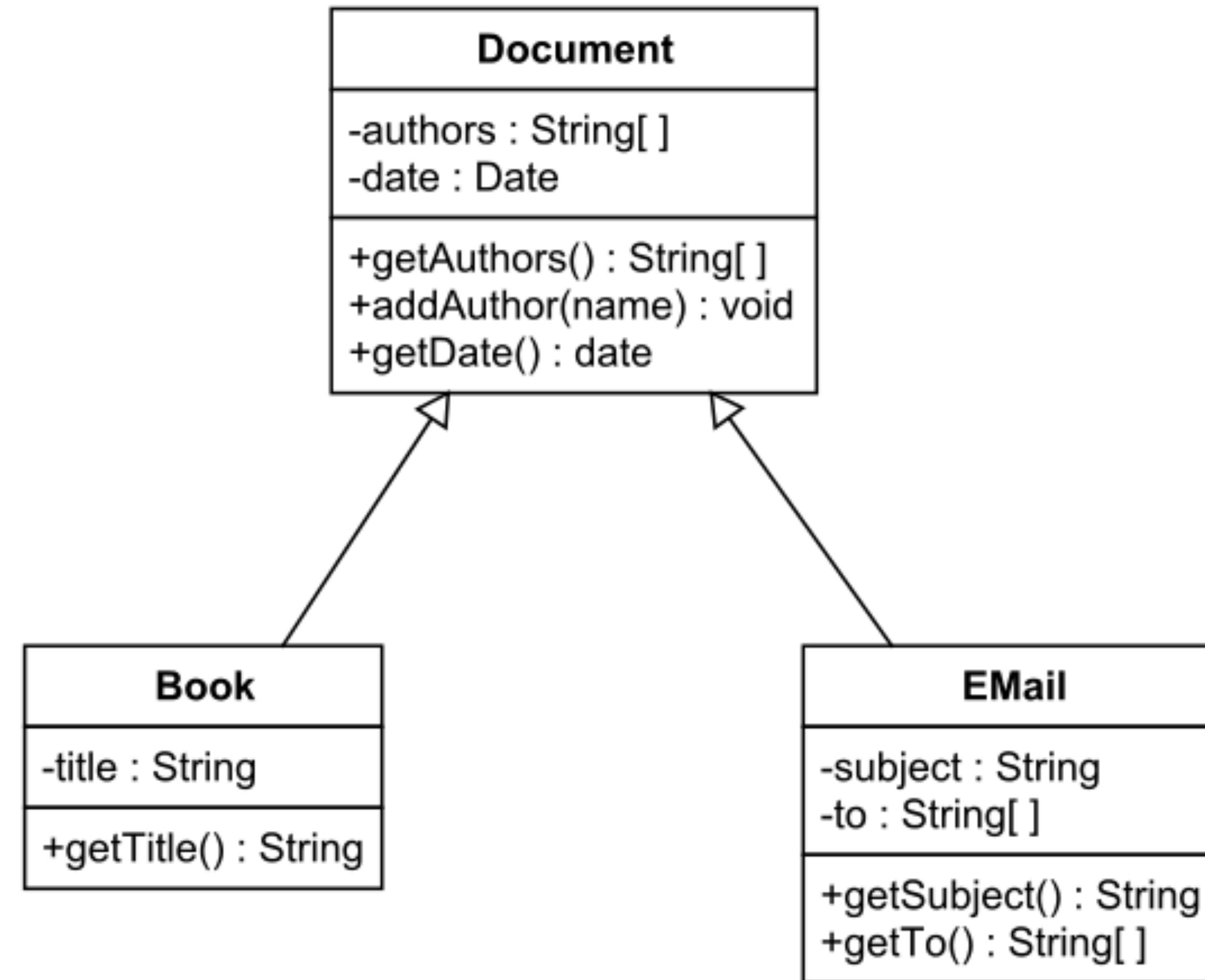


## ● Outros exemplos





## ○ Outros exemplos



# Mensagens



- É o mecanismo através do qual os objetos se comunicam, invocando as operações desejadas
- Especificação de uma operação do objeto
- Ao dirigir um carro, o ato de pressionar o acelerador envia uma **mensagem** para o veículo realizar uma tarefa – isto é, ir mais rápido.
- Em POO, mensagem é a transmissão de uma requisição por um objeto emissor para um objeto receptor para que este execute um comportamento (**método**) desejado
  - As mensagens enviadas devem corresponder aos métodos definidos pelo objeto receptor
  - Métodos respondem com um retorno à chamada



- É composta por:

- **Seletor:**

- nome simbólico que descreve o tipo da operação
    - descreve O QUE o objeto que envia quer que seja invocado, não como deveria ser invocado
    - o objeto recebedor da mensagem contém a descrição de COMO a operação deveria ser executada

- **Parâmetros:**

- argumentos que uma mensagem pode conter que faz parte da operação e requer uma ordem única



- Um objeto (**Emissor**) enviar uma mensagem a outro (**Receptor**) que executará o serviço
- Métodos são invocados por Mensagens
- Exemplo:
  - A chamada de um procedimento/função em LP é uma aproximação inicial de uma mensagem, como em **multi(10,20)** onde **multi** é o **seletor** e os valores **10** e **20** são os **parâmetros**
  - Diferença: a ação da mensagem a ser ativada depende essencialmente do objeto que receber a mensagem

# Interface

# Interface (Protocolo ou Assinatura)



- Objetos se comunicam através de **Mensagens**, que invocam os **Métodos** desejados
- Pontos de interação entre dois meios, objetos, etc.
- Ex: Televisão
  - As interfaces da TV são os botões de ligar/desligar, mudar canais, ...
  - Pontos de interação da entidade com o mundo externo
- Uma interface em Java representa uma declaração de um conjunto público de operações
  - Define um **contrato**
  - Não pode ser instanciada
  - Garante que os objetos terão esta forma de comunicação



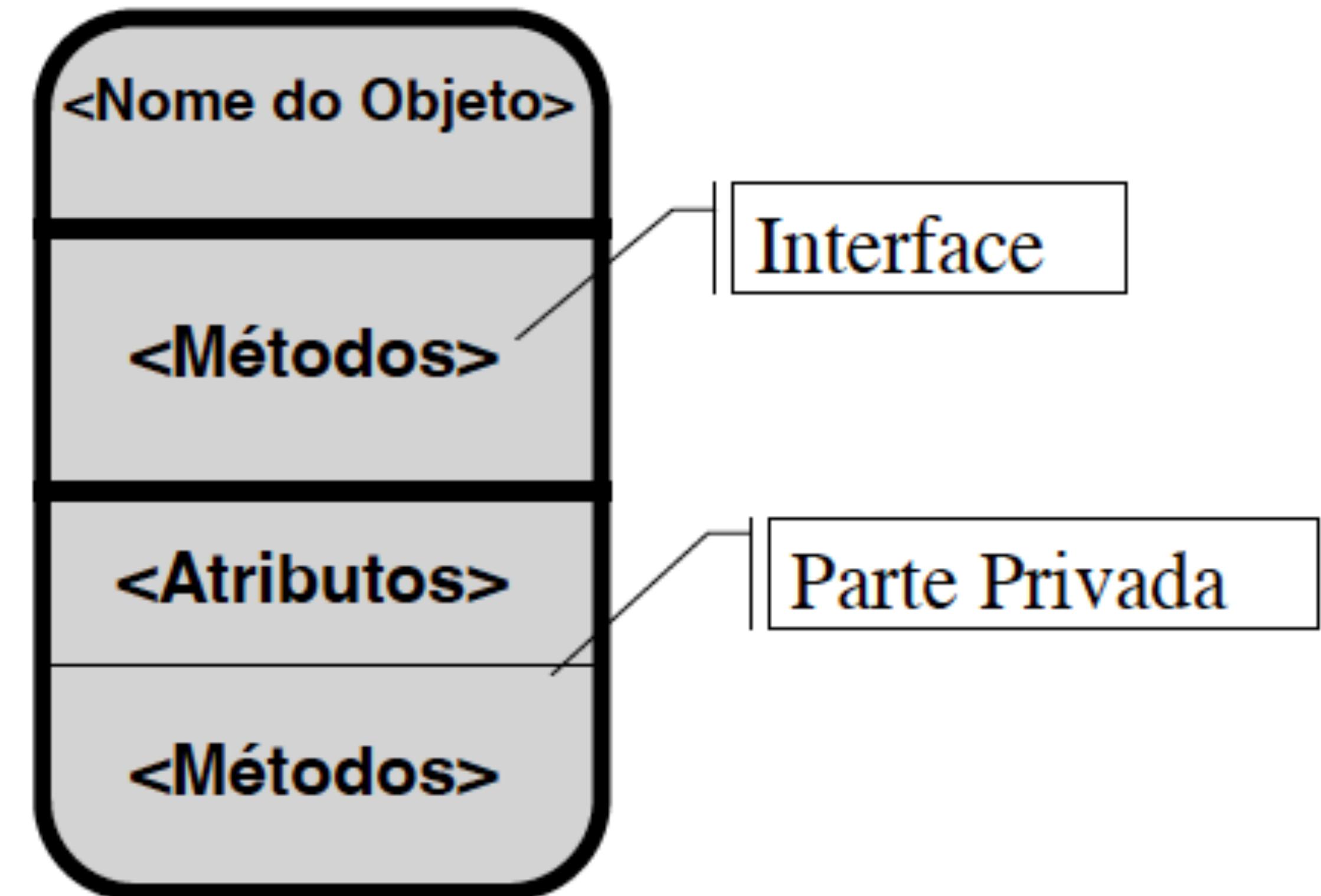
## ● **Parte Privada** do Objeto (Visão Interna)

- Métodos
- Atributos

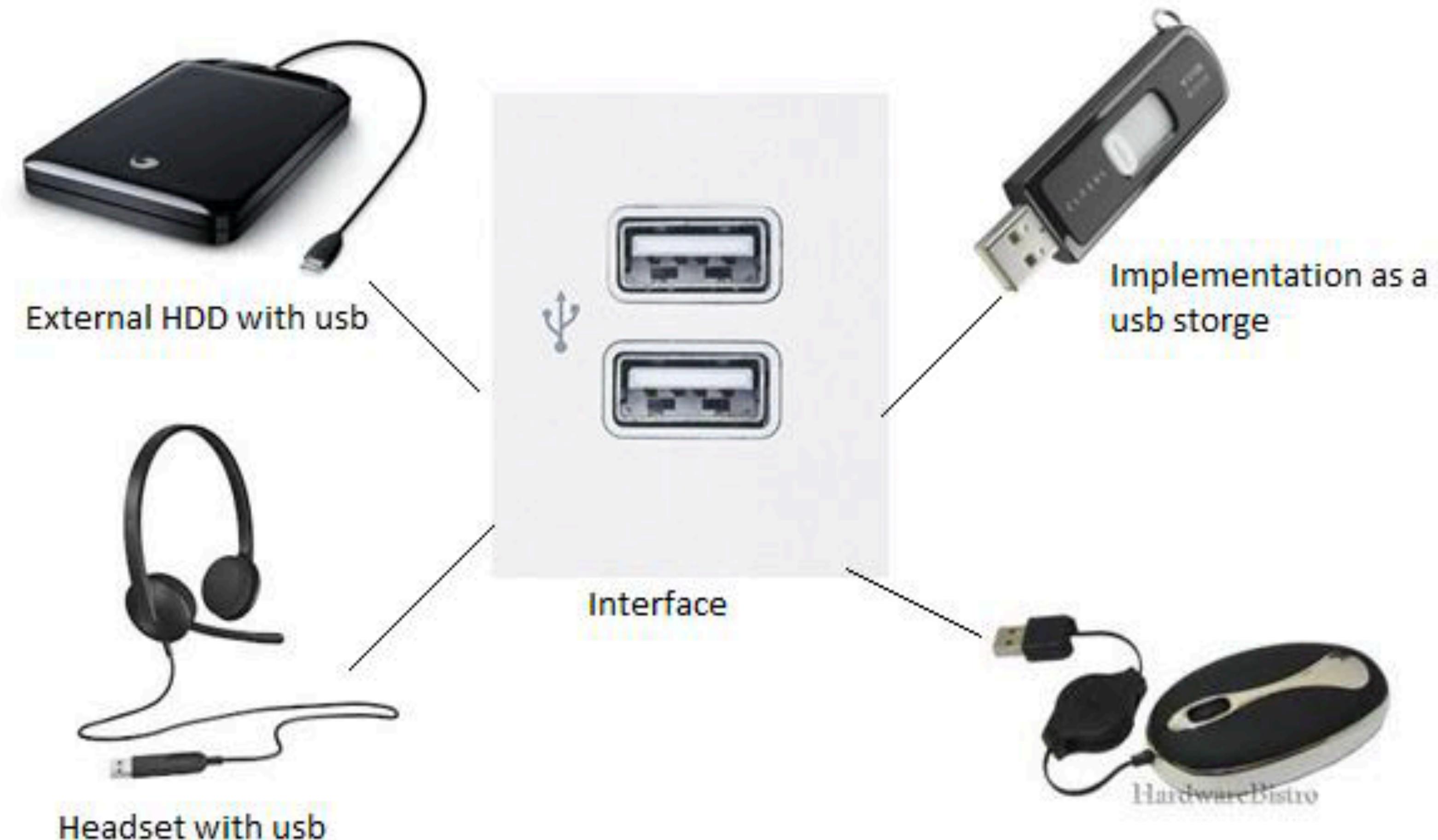
## ● **Parte Compartilhada** do Objeto ou Interface

- corresponde à parte externa do objeto, isto é, àquela que é vista por outros objetos com a finalidade de invocar os métodos que o objeto realiza
- agrupa um conjunto de mensagens que o objeto pode responder
- as mensagens especificam quais operações sobre o objeto podem ser realizadas, mas não como a operação será executada

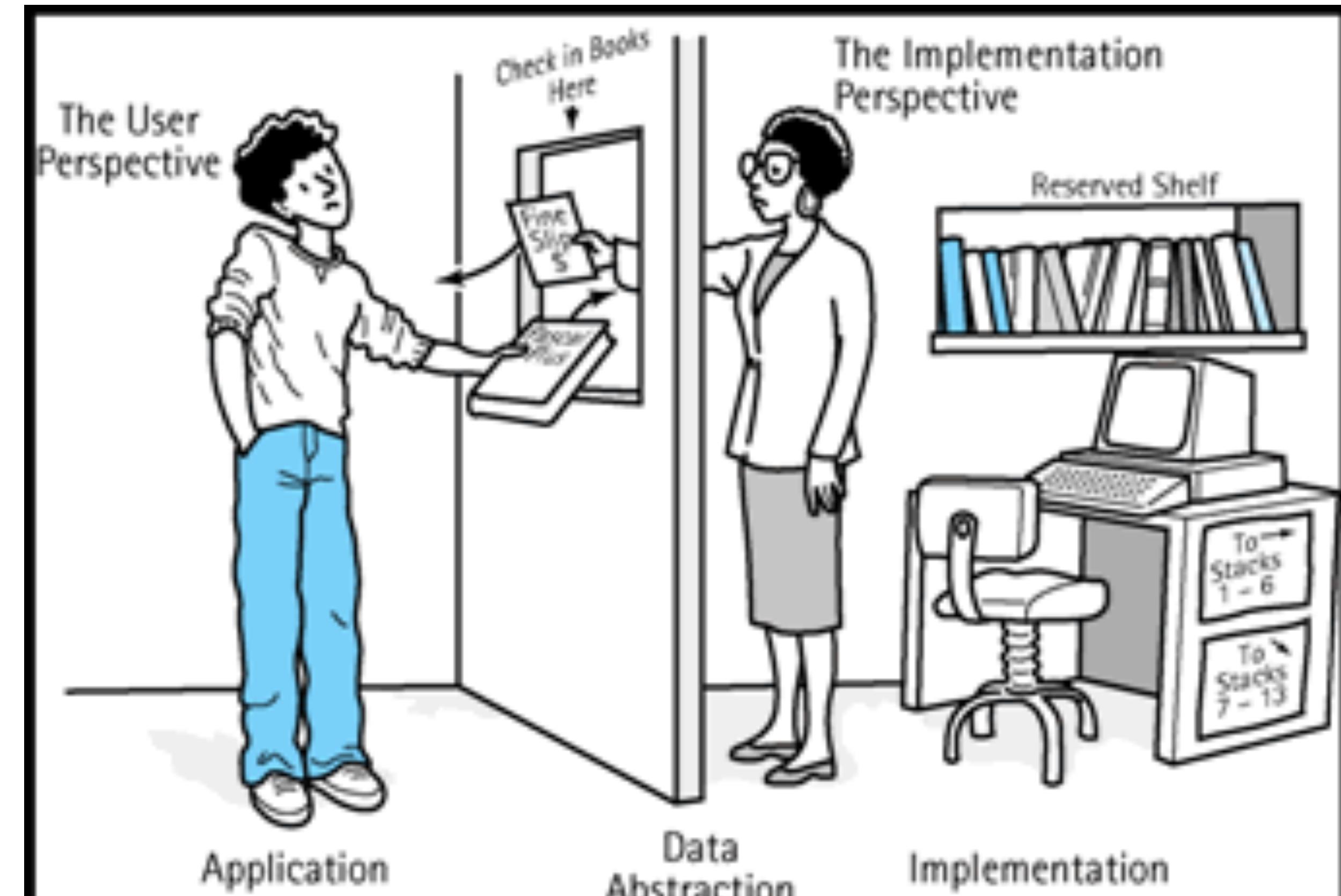
# Estrutura de um Objeto (classe)



# Interface



# Interface



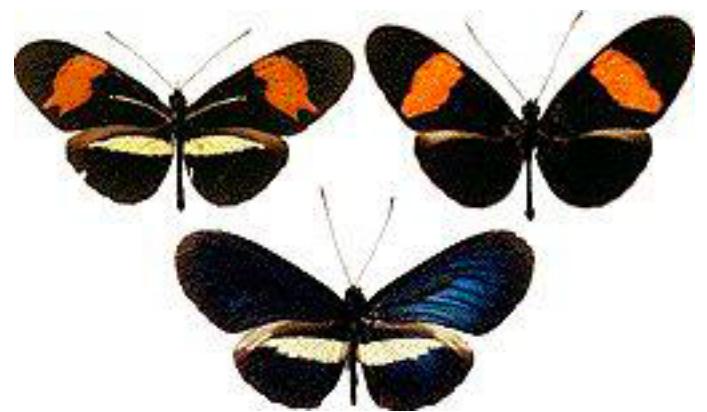
# Polimorfismo



# Polimorfismo



- Polimorfismo, em biologia, é um princípio no qual um organismo pode surgir de formas diferentes.
  - Indivíduos de uma mesma espécie possuem muitas características similares.
  - Contudo, algumas características são peculiares.





- O mesmo princípio se aplica no paradigma de POO
  - Herança permite que subclasses herdem as características de sua classe pai (mãe).
  - Contudo, propriedade particulares da subclasse podem ser redefinidas.
- Em POO, polimorfismo é a capacidade de uma mesma operação (método) comportar-se de maneira diferente nas diferentes classes de uma hierarquia
  - Método com a mesma assinatura, porém com serviços diferentes



- Ao receber uma mensagem para efetuar uma Operação, é o objeto quem determina como a operação deve ser efetuada, pois ele tem o comportamento próprio
- Como a responsabilidade é do **Receptor** e não do **Emissor**, pode acontecer que uma mesma mensagem ative métodos diferentes, dependendo da classe de objeto para onde é enviada a mensagem
- Permite a criação de várias classes com interfaces idênticas, porém objetos e implementações diferentes
- Capacidade de uma mensagem ser executada de acordo com as características do objeto que está recebendo o pedido de execução do serviço



## ○ Pacotes e bibliotecas

- Pacote é um conjunto de classes e interfaces relacionadas de alguma forma
- Biblioteca é o conjunto de pacotes
- API (Application Programming Interface)
  - Muito útil: reuso de código
  - API Java

## ○ Antes de definir uma nova classe, verifique se já não existe uma solução

- A validação do código é demorada



- **Linguagens Orientadas a Objetos**

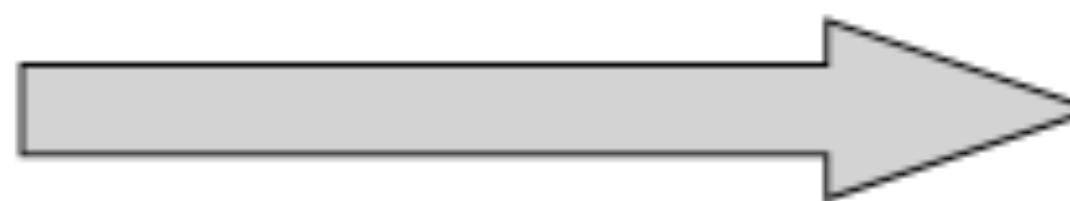
Objeto	→ Valor
Classe	→ Tipo (TAD)
Mensagem	→ Chamada de Procedimento
Método	→ Procedimento ou Função
Interface	→ Conjunto de nomes e funções para um fim específico

- **Linguagens Tradicionais**

- Exemplo de objeto do mundo real
  - “a coisa em que você está sentado agora”



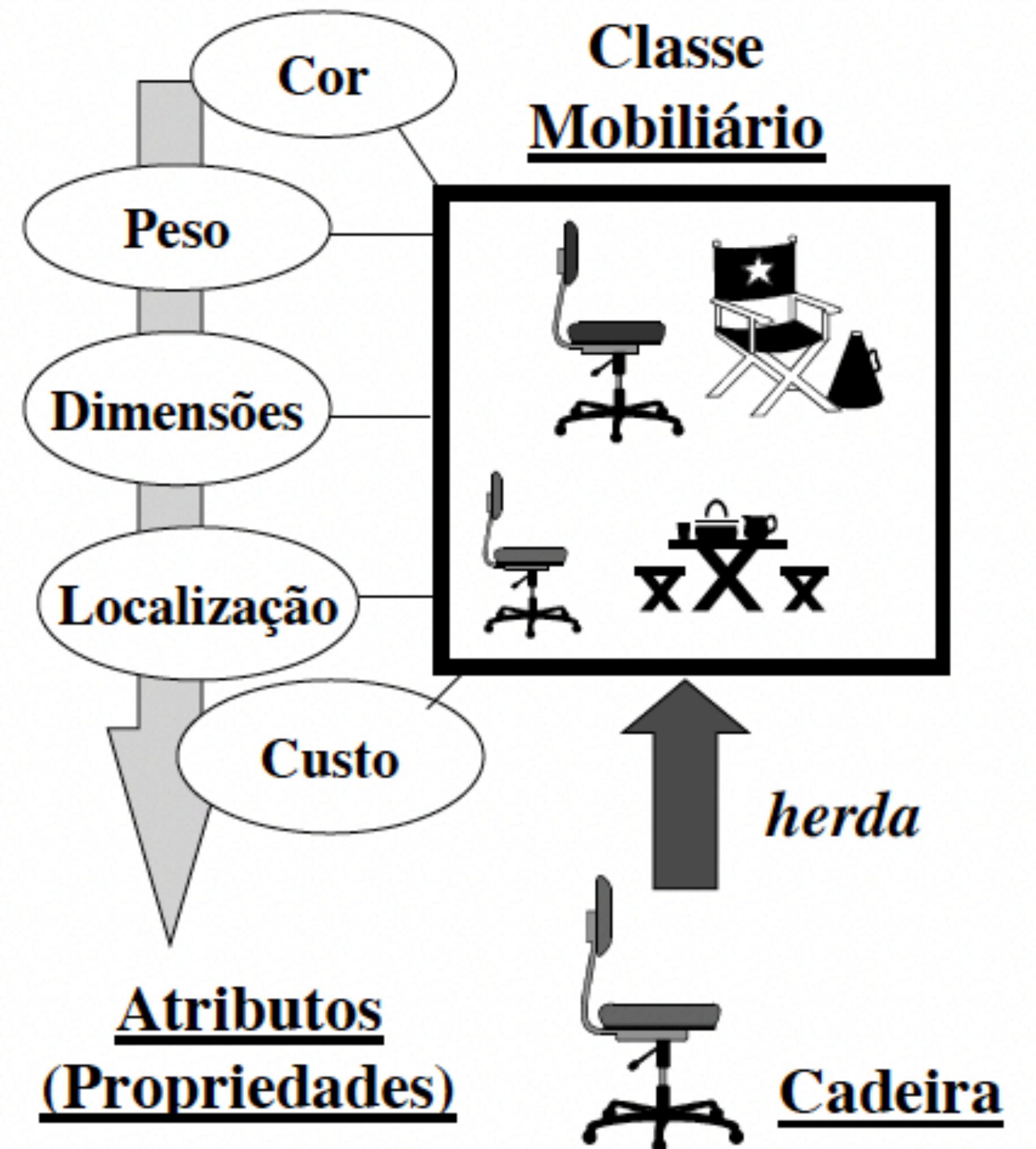
Cadeira

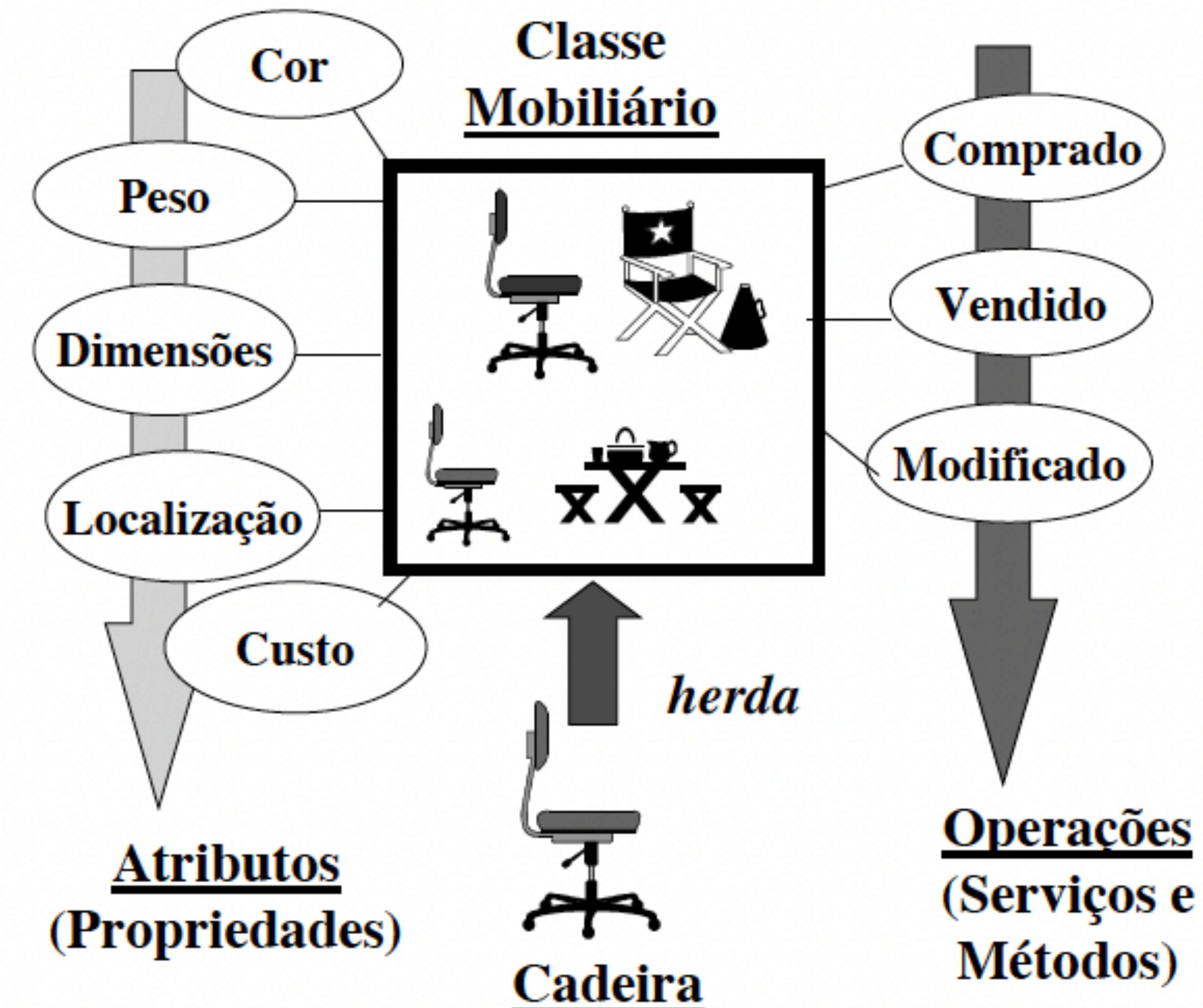


É membro - (instância)  
de uma classe maior de  
objetos



Mobiliário



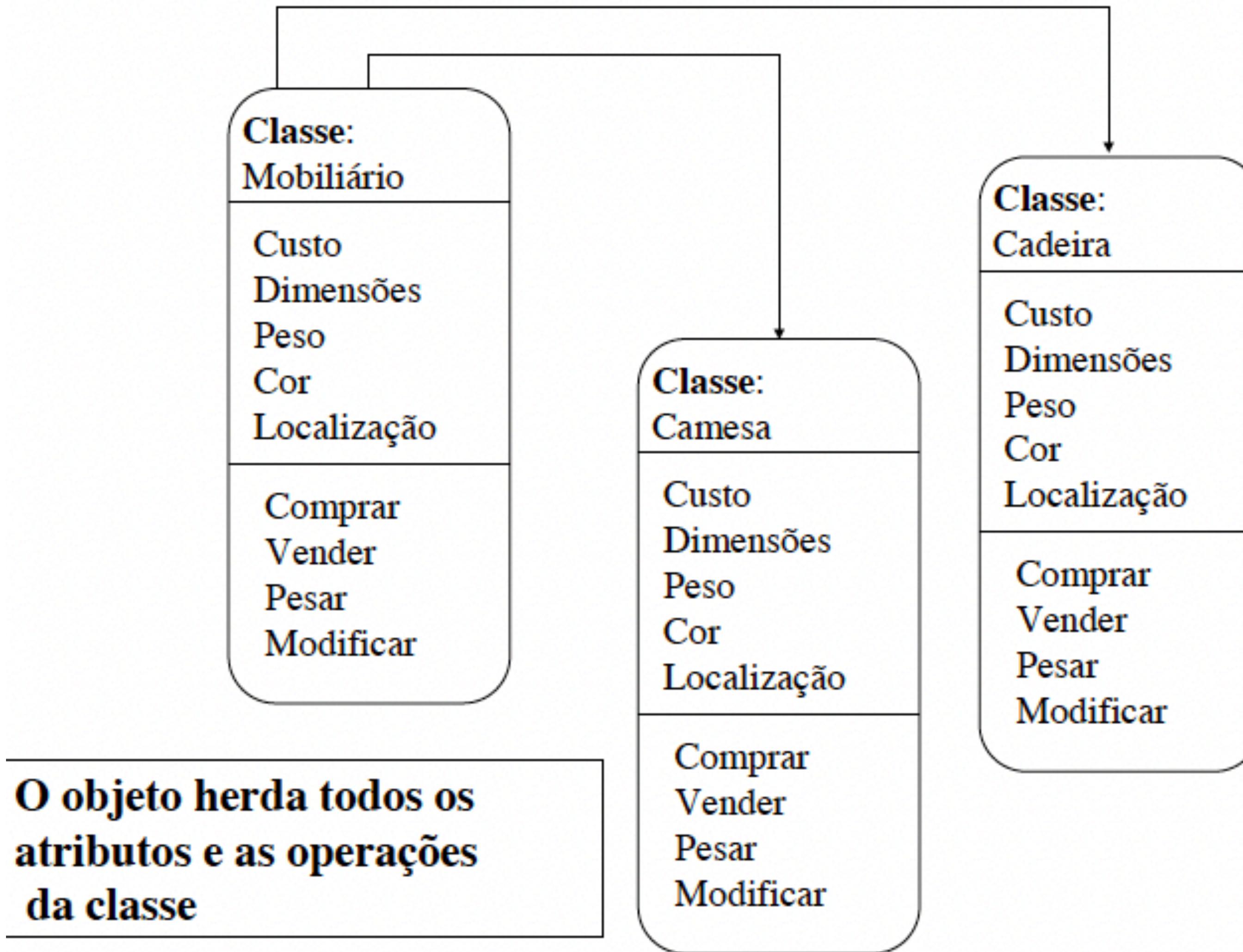




- O Objeto **Cadeira** é um membro (**INSTÂNCIA**) de uma **CLASSE** maior de objetos - **MOBILIÁRIO**
- **CLASSE** de objetos descreve um conjunto de objetos que possuem
  - propriedades semelhantes (**ATRIBUTOS**)
  - mesmo comportamento (**MÉTODOS**)
  - mesmos relacionamentos com outros objetos
  - mesma semântica
- Atributos **GENÉRICOS** da classe **MOBILIÁRIO**
  - custo, dimensões, peso, localização, cor
- Atributos se aplicam a todos os objetos da classe **MOBILIÁRIO**: mesa, cadeira, sofá, armário

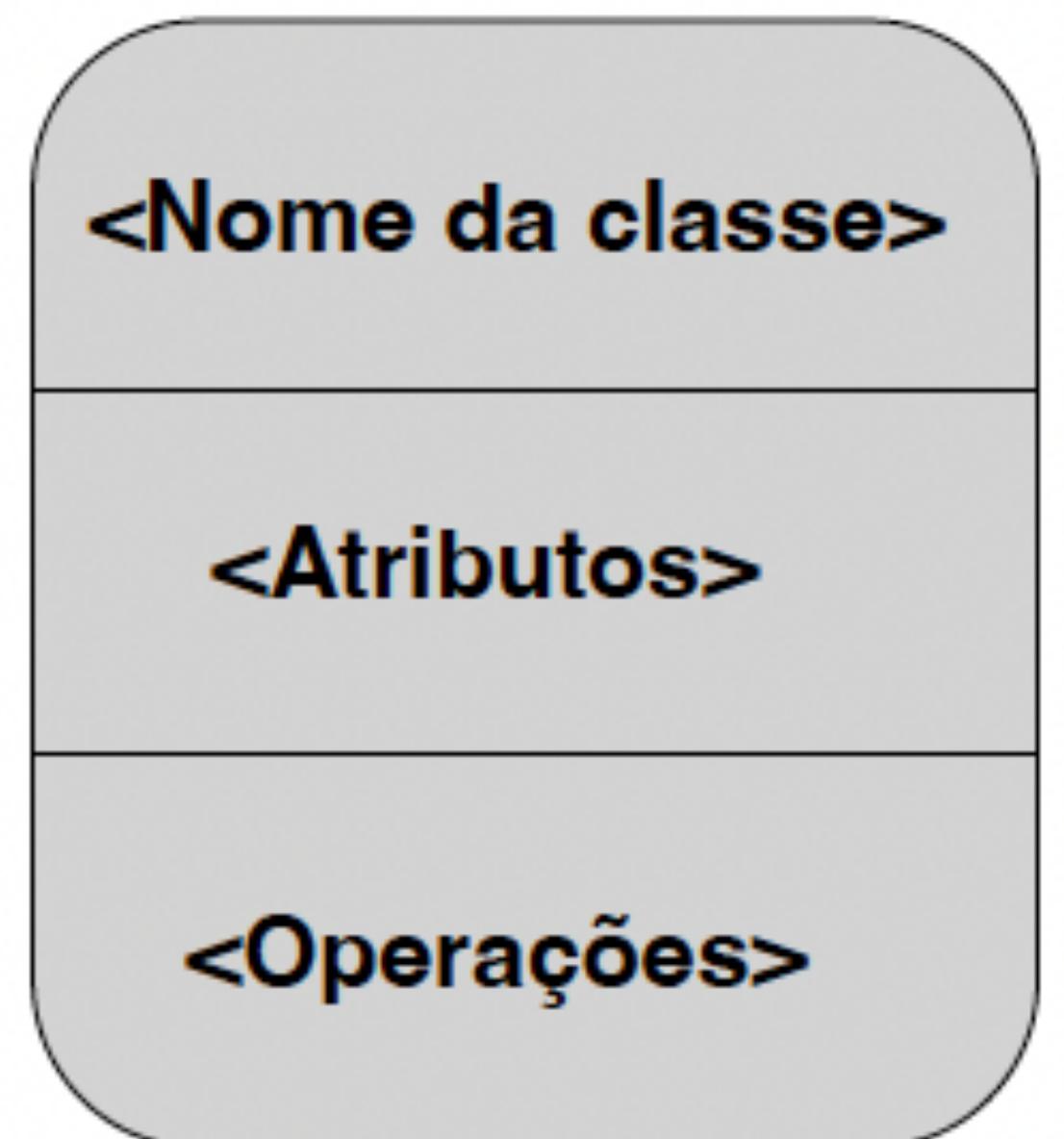


- Uma vez que o objeto CADEIRA é um membro da classe MOBILIÁRIO, HERDA todos os atributos definidos para a classe
- Todo objeto da classe MOBILIÁRIO pode ser manipulado de diversas maneiras (OPERAÇÕES)
  - comprado
  - vendido, pesado
  - modificado (pintar de roxo, serrar uma perna)
- Cada uma dessas operações (SERVIÇOS ou MÉTODOS) modificará um ou mais atributos do objeto





- O objeto CADEIRA e outros da mesma classe ENCAPSULA
  - dados (os valores dos atributos que definem a cadeira)
  - operações (as ações aplicadas para mudar os atributos da cadeira)





- Paradigmas de Programação
- Os pilares da POO
- Classes e Objetos
- Tipos de acessos
- Herança e Polimorfismo
- Mensagens
- Interfaces



- Definir as classes abaixo utilizando diagramas
  - Animal, Homem, Cachorro, Carro, Casa
- Defina atributos e métodos destas classes, pensando na interação entre objetos desses tipos
  - Lembre que atributos podem ser objetos
- Onde pode haver herança?
- Simule troca de mensagens entre os objetos



- Leitura do capítulo 2 do livro: **DEITEL, H. M. & DEITEL, P.J. "Java : como programar".**



- DEITEL, H. M. & DEITEL, P.J. "Java : como programar", Bookman, 2003.
- Material baseado nos slides:
  - Turine, M. A.; Minghim, R. Notas de Aula de Paradigma de Orientação a Objetos. (UNIC/USP)
  - Da Silva, L. E. Notas de Aula de Programação Orientada a Objetos (ICMC/USP).