Linguagens Imperativas

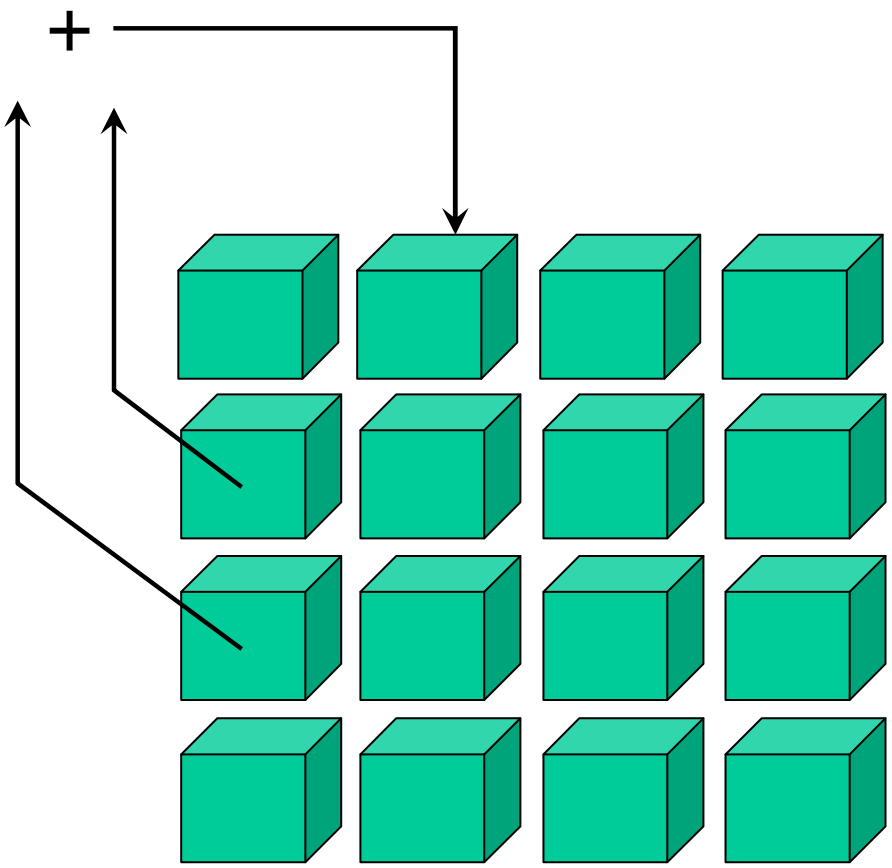
Linguagens Imperativas (i)

Linguagens imperativas ou procedurais são linguagens orientadas à comandos ou declarações. Elas baseiam-se na idéia de que a memória mantém uma máquina de estados que pode ser modificada a cada execução das declarações da linguagem.

A Figura a seguir ilustra este processo.

Linguagens Imperativas (ii)

Geralmente tem uma sintaxe do tipo:



declaração 1;
declaração 2;
declaração 3;
.....

A memória é vista como um conjunto de caixas modificadas a cada declaração do programa.

Linguagens Imperativas (iii)

Desenvolvimento de programas consiste na modificação consecutiva dos estado da memória até se chegue a uma solução desejada.

Este paradigma representa geralmente a primeira abordagem que alguém aprende em programação (no MIT usava-se programação funcional como primeiro curso). Algumas das mais usadas linguagens de programação usam este modelo de programação:

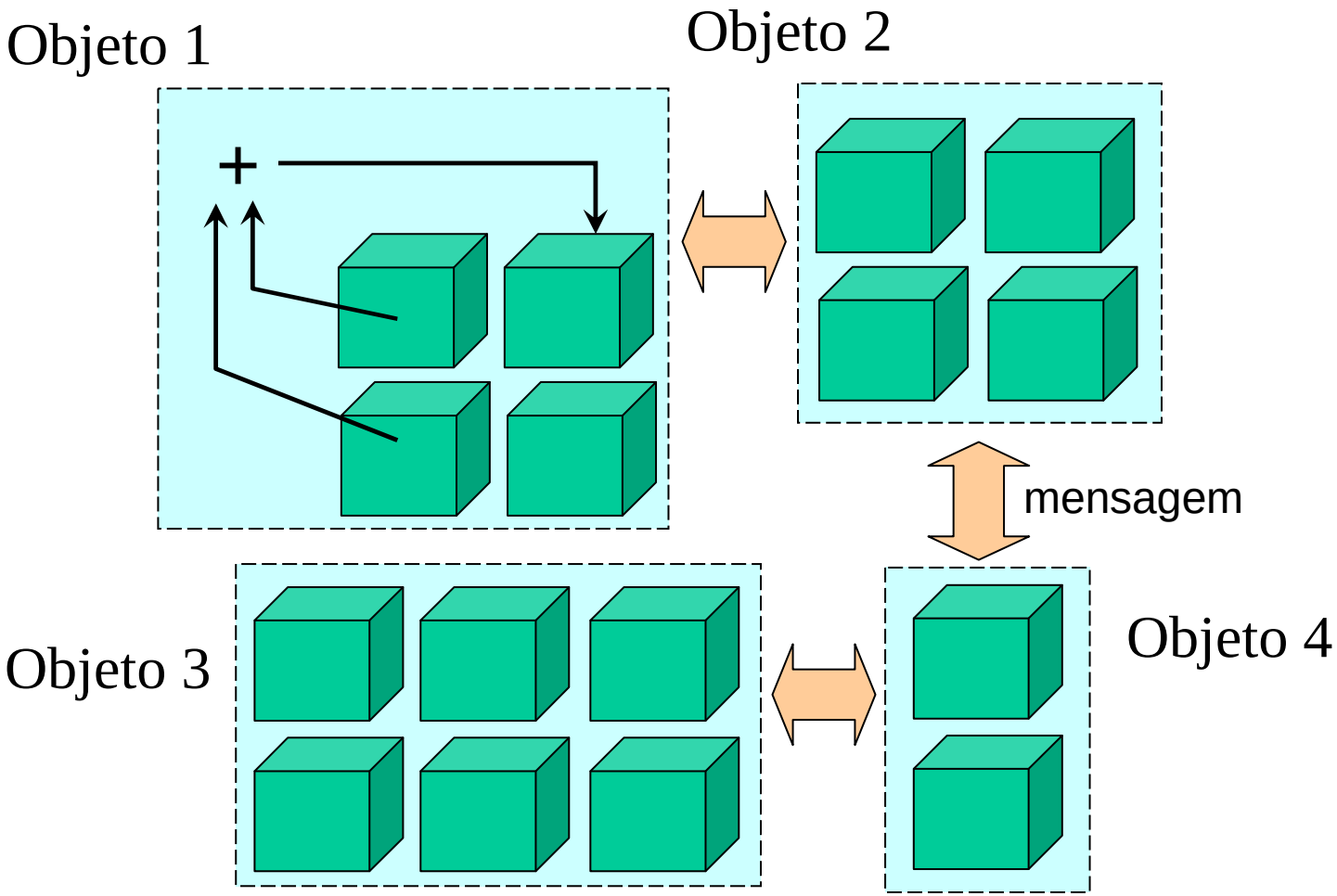
C, FORTRAN, Algol, Pascal, COBOL, “C++”, “Java”, etc.

Linguagens Orientadas a Objetos

Linguagens OO (i)

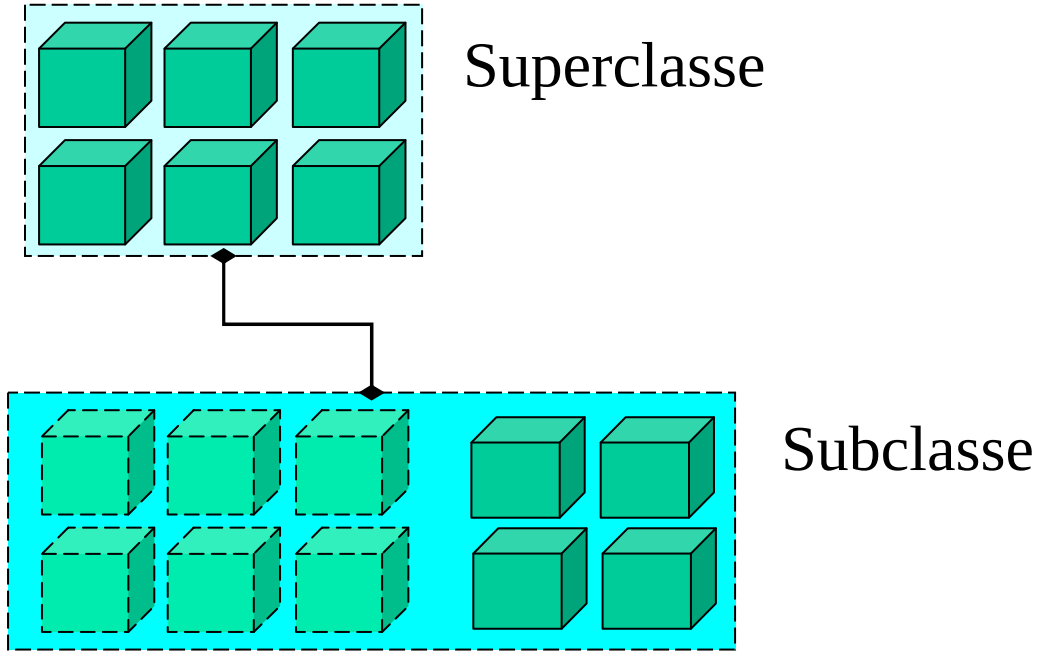
Linguagens orientadas à objeto são linguagens que estendem a linguagens imperativas com a introdução do conceitos de objetos. Objetos são conjuntos de dados geralmente complexos com funções bem definidas que podem ser aplicadas eles. Funções e dados são **encapsulados** pelo objeto de forma a expor somente o que se quer a outros objetos. Objetos se comunicam por **mensagens**, que chamam as funções expostas. Para facilitar esta prática, funções similares podem ter o mesmo nome deste que tenham estruturas diversas (polimorfismo em tempo de compilação), ou a associação entre uma mensagem e um objeto (e a função chamada) pode ser feita dinamicamente em tempo de execução (**polimorfismo** real).

Linguagens OO (ii)



Linguagens OO (iii)

Objetos são definidos a partir de outros objetos mais genéricos através de mecanismos de **herança**.



Linguagens OO (iv)

Ao construir objetos concretos na memória, OO usa a eficiência da programação imperativa. Ao construir classes de funções que aplicam-se a um conjunto restrito de objetos, OO ganha através de *encapsulamento* e *polimorfismo* a flexibilidade e confiabilidade da programação funcional (nossa próxima transparência).

São exemplos de linguagens OO as linguagens Smalltalk, Ada (não tem polimorfismo real), C++, e Java.

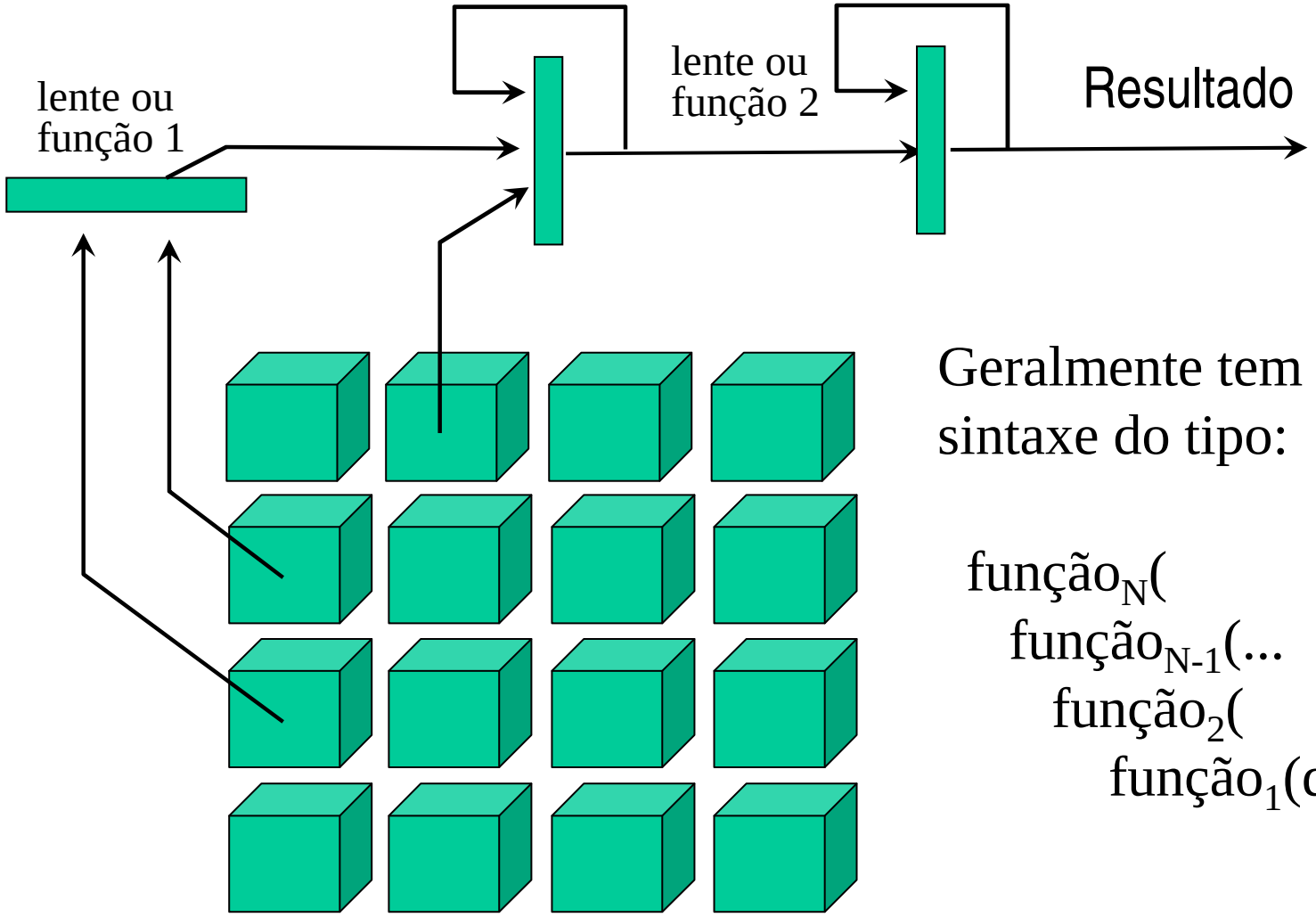
Linguagens Funcionais

Linguagens Funcionais (i)

Linguagens funcionais ou aplicativas são linguagens orientadas à função que um programa representa. Isto é conseguido pensando-se na função que deve ser aplicada num estado de máquina inicial para transformá-lo em um estado de máquina final desejado como resposta.

Pode-se ver este processo como a construção de uma lente que pega o estado inicial da memória e o "transforma" em um estado final desejado. A Figura a seguir ilustra este processo.

Linguagens Funcionais (ii)



Geralmente tem uma sintaxe do tipo:

$função_N($
  $função_{N-1}(...$
    $função_2($
      $função_1(dados))$ ...
))

Linguagens Funcionais (iii)

Desenvolvimento de programas consiste em sucessivamente desenvolver funções a partir de funções previamente existentes (bottom-up), até chegar uma função final (geralmente complexa) que consegue computar a resposta desejada a partir do conjunto inicial de dados.

Em vez de olhar para sucessivas máquinas de estados de uma computação, a programação aplicativa considera sucessivas transformações funcionais. LISP, ML e HASKELL são três da mais conhecidas linguagens funcionais.

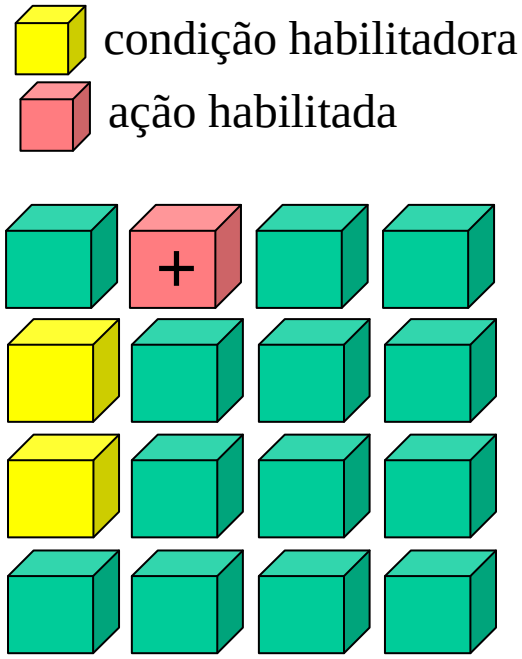
Linguagens Lógicas

Linguagens Lógicas (i)

Linguagens lógicas ou baseadas em regras são linguagens baseada na lógica de predicados, onde uma série de regras são definidas para que o programa tome ações apropriadas para cada estado habilitador na memória do computador.

Pode-se ver este processo como a construção de uma série de filtros (as regras) que sucessivamente habilitam mudança de estados. A figura a seguir ilustra este processo.

Linguagens Lógicas (2)



Geralmente tem uma sintaxe do tipo:

$condição_1$ -> $ação_1$
$condição_2$ -> $ação_2$
....
$condição_n$ -> $ação_n$

Linguagens Lógicas (3)

Desenvolvimento de programas consiste em construir-se um conjunto de regras que trate (tenha condições habilitadoras e tome ações adequadas) todos os possíveis estados de iniciais do programa.

Prolog (linguagem de programação lógica) é o exemplo mais conhecido desta família de linguagens, mas qualquer qualquer linguagem ou ferramenta baseada em tabelas de decisões podem ser vistas como linguagens baseadas em regras. Exemplos incluem ferramentas de *parsing* tais como YACC (Yet Another Compiler Compiler) ou JavaCC.

Introdução à POO

Programação OO

Programação imperativa separa (ou tende a separar) dados dos procedimentos usados para manipular estes dados.

Programação OO concentra-se definir os objetos dentro de um certo domínio com um conjunto “estado” e “comportamento”.

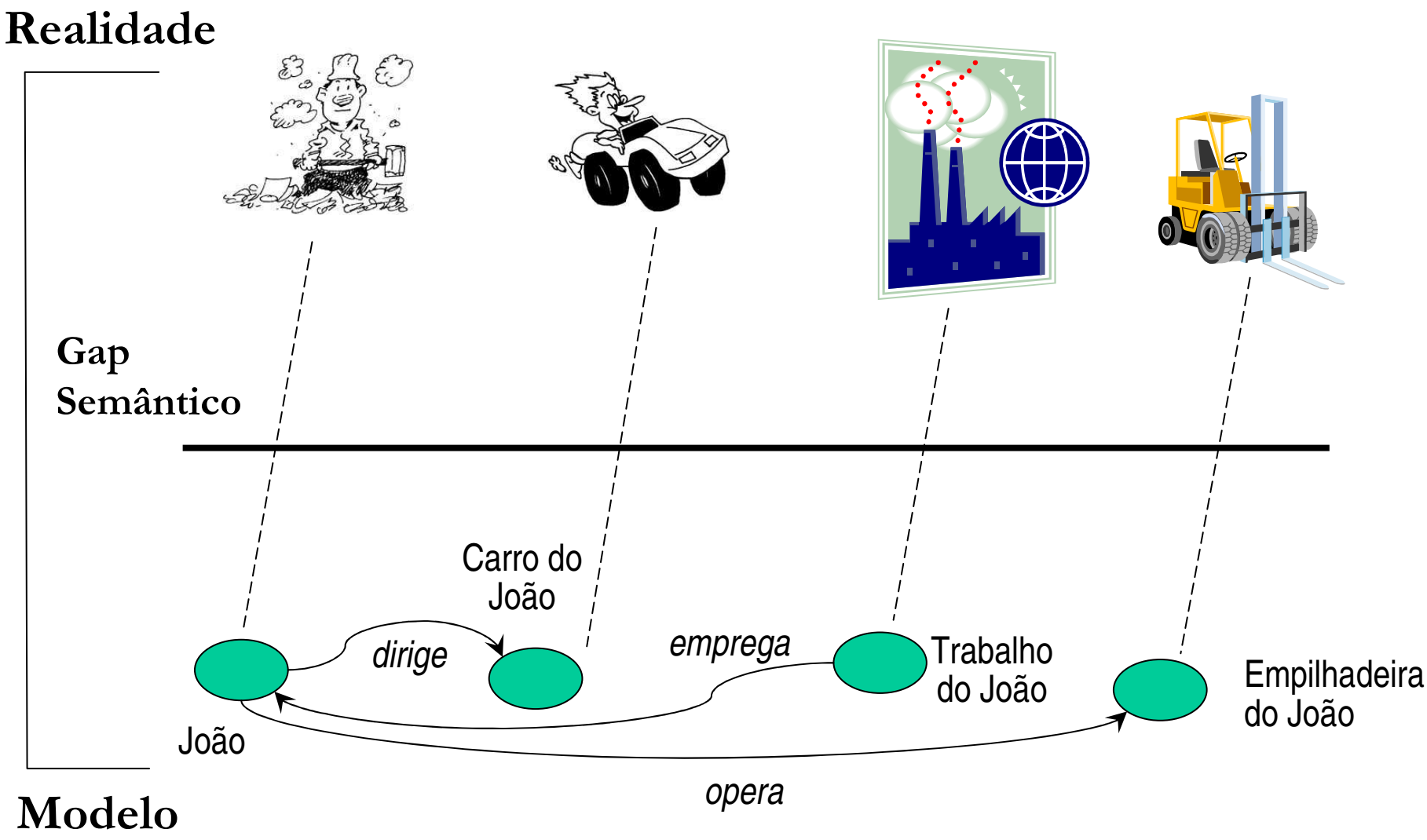
Conceitos Básicos em OO

• **Encapsulamento:** os detalhes de implementação dos objetos são “escondidos” dos usuários deste objeto (só a sua interface e comportamento é de interesse real).

• **Herança:** novas classes de objetos podem ser criadas a partir de outras classes de objetos mais abstratos. Estes novos objetos herdam e estendem as propriedades dos objetos mais abstratos.

• **Polimorfismo** (um nome muita formas): o remetente da mensagem não precisa se preocupar em saber os detalhes dos objetos destinatários desta mensagem. A mesma mensagem pode ser dinamicamente enviada para muitos tipos de objetos.

Gap Semântico



POO e o Gap Semântico

• **O Gap Semântico** é a diferença entre a forma como o modelo representa a realidade, e a realidade propriamente dita.

• Quanto menor o intervalo, mais fácil será a compreensão do sistema e a forma de alterá-lo. As alterações serão na maioria das vezes locais, afetando um ou poucos “indivíduos” que são representados por códigos contidos em objetos.

POO tenta reduzir o GAP Semântico !

Objetos

• Objeto: é uma entidade habilitada a ter um estado (informação) e oferece um determinado número de operações (comportamentos) que podem examinar ou afetar o estado do objeto.

• Modelo Orientado a Objeto: refere-se a modelos cujos componentes são representados por objetos.

Características da POO (1)

• **Acesso a Informações:** a única forma de acesso externo ao objeto será através dos métodos, suas particularidades internas devem ser protegidas do mundo externo.

• **Encapsulamento:** variáveis e métodos associados a um objeto são encapsulados. Esta é uma idéia simples e poderosa que proporciona duas vantagens:

Proteção dos Dados e **Modularidade**.

• **Comunicação:** a comunicação entre os objetos ocorre por

Passagem de mensagens (chamada a um Método)

Características da POO (2)

Tipo Abstrato de Dados **encapsulam** **implementação** e **interface** em uma **classe de objetos**.

Uma ou mais **instâncias** de uma **classe** podem então ser **instanciadas**.

Uma **instância** de uma **classe** é conhecida como um **objeto**.

Todo **objeto** tem um **estado** e **comportamento**. Aonde o **estado** é determinado pelos valores atuais armazenados nas **variáveis de instância** e o **comportamento** é determinado pelos **métodos de instância** da classe da qual o **objeto** foi **instanciado**.

Classes e Instâncias

• **Classe:** é uma definição, um modelo existente para a criação de novos objetos. Pode ser considerada como uma abstração que descreve todas as características comuns dos objetos criados a partir dela.

• Usando o conceito de classe, características podem ser associadas a um grupo inteiro de objetos.

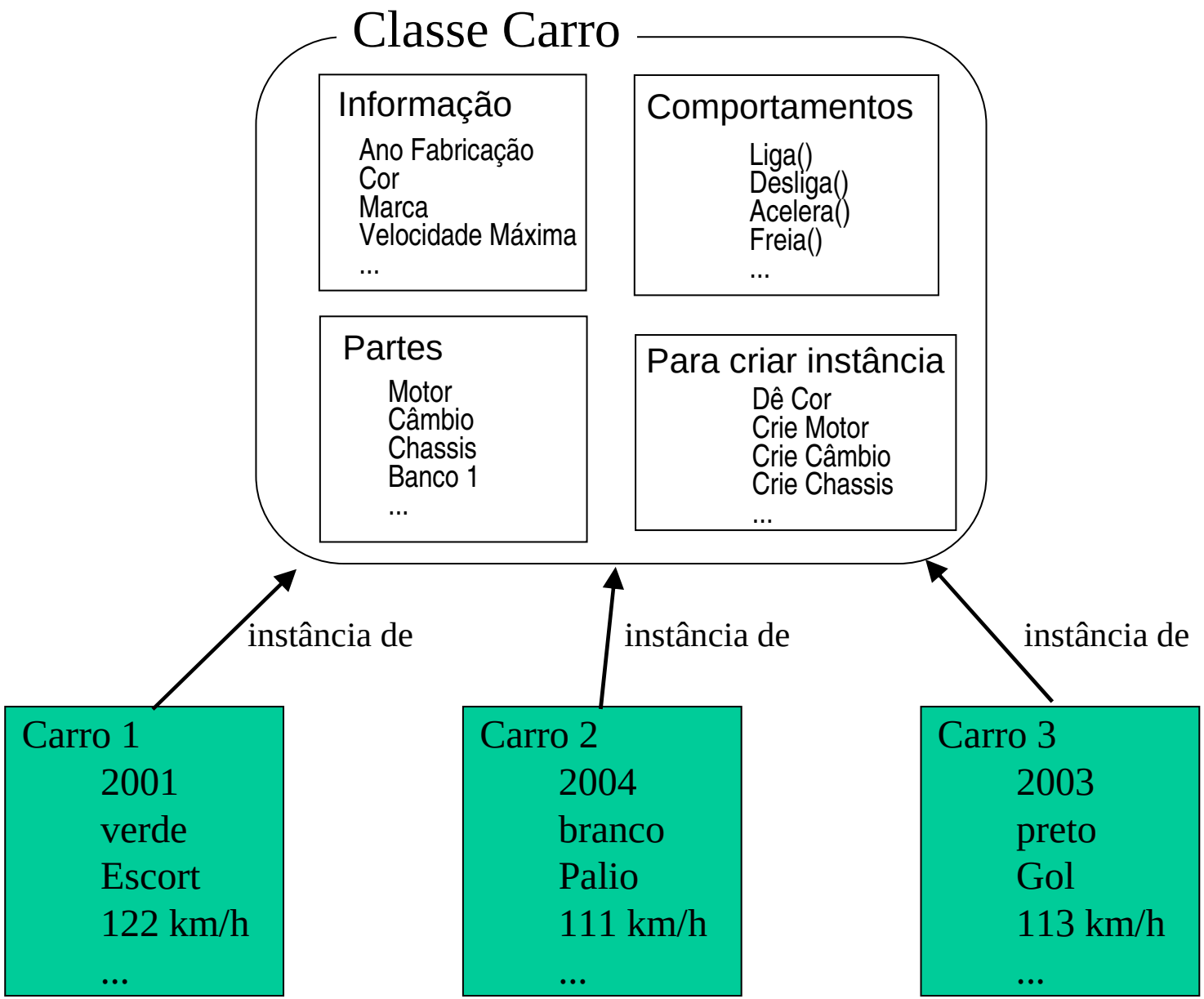
• **Instância**: um objeto que pertença a uma classe é chamado de instância desta classe.

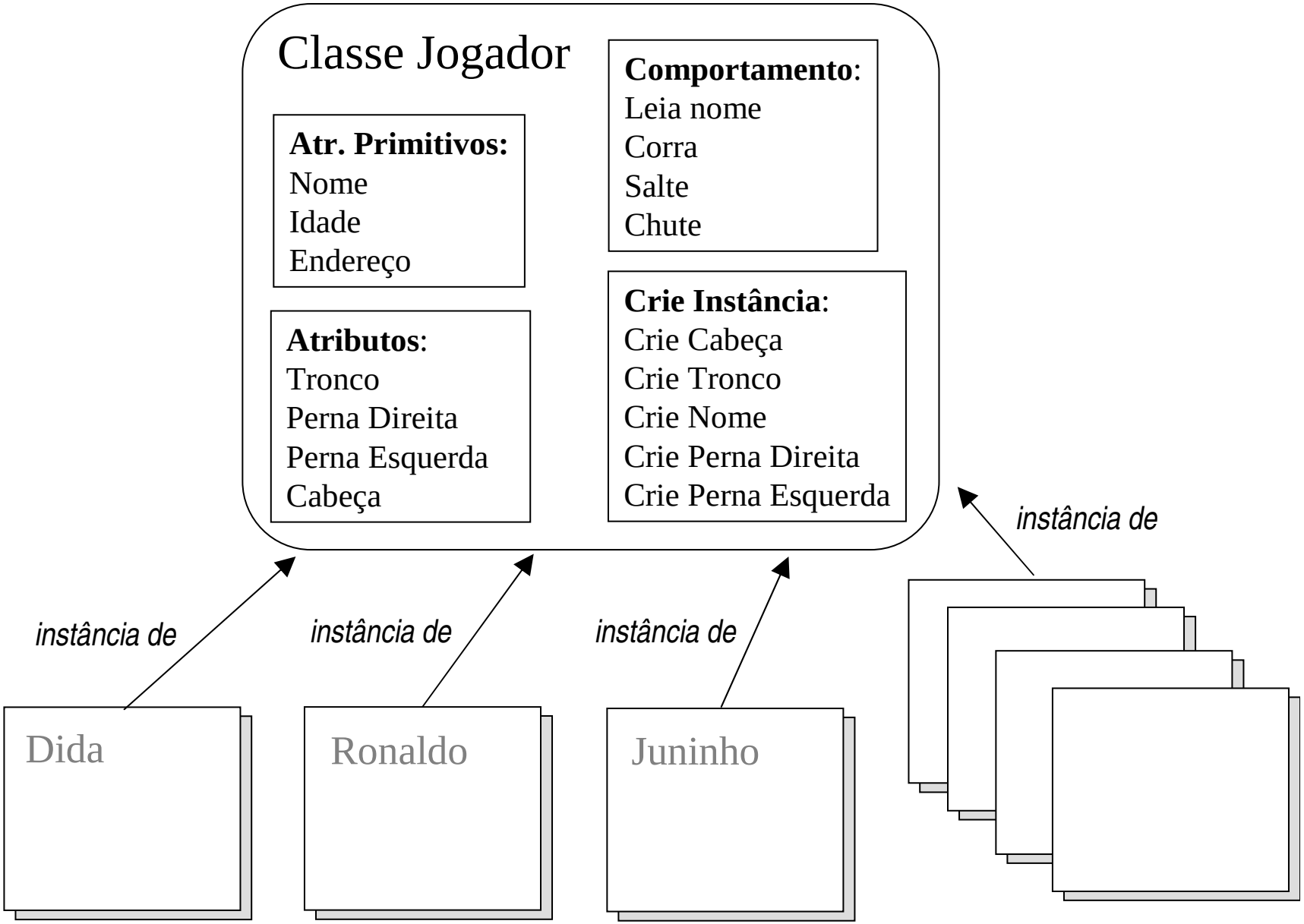
Mais Classes e Instâncias

• Uma **classe** é uma fôrma que descreve de forma genérica grupos de objetos com características similares.

• Uma **instância** de uma classe é um objeto real.

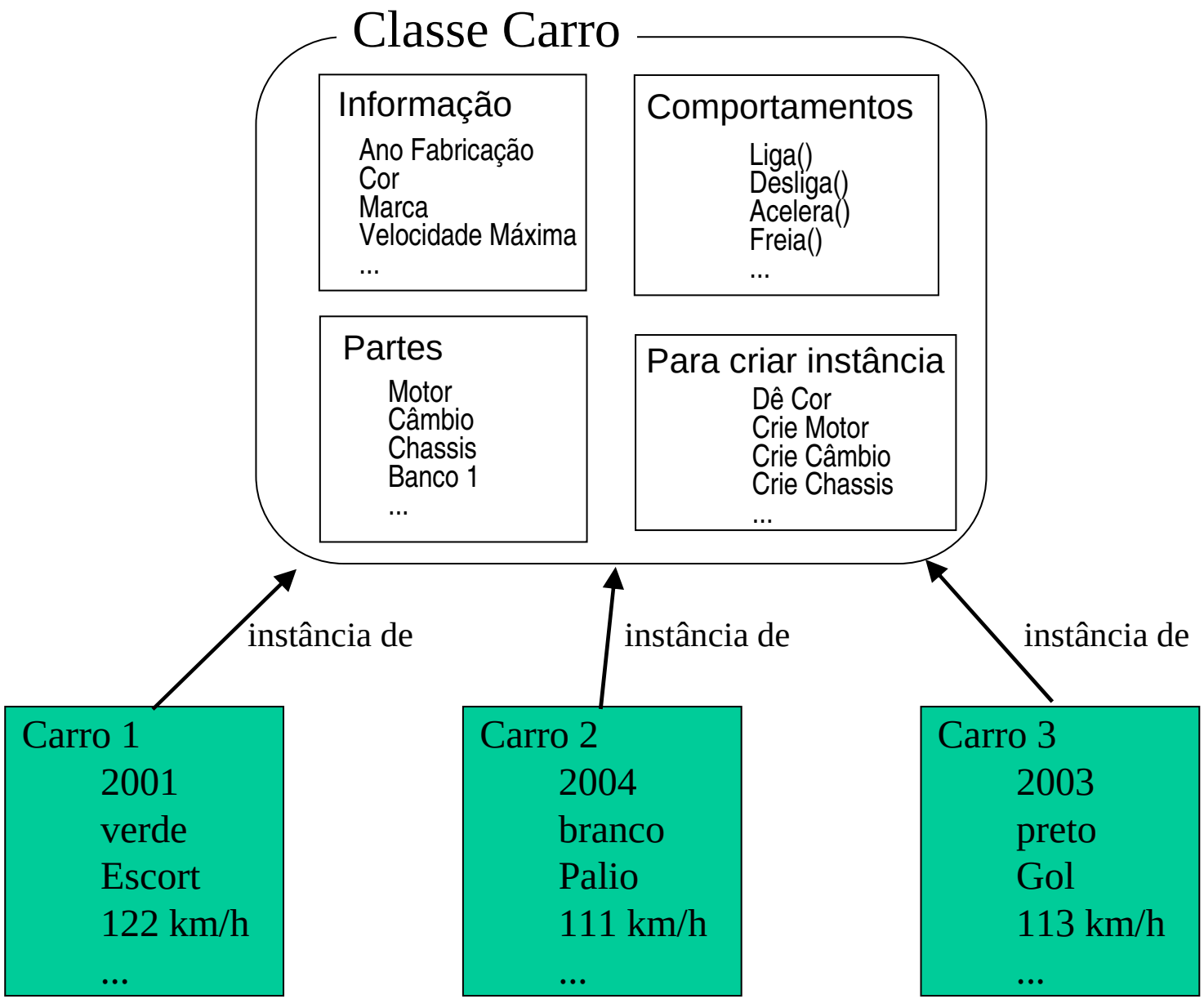
A classe representa a descrição do que seria um objeto (ex., um Carro que deve ter modelo, cor, ano, e dono) enquanto uma instância é uma representação concreta de um destes objetos na memória (ex., o Escort verde, ano 2001, do Prof. Manoel).


Classe Carro
Informação
Ano Fabricação
Cor
Marca
Velocidade Máxima
...
Comportamentos
Liga()
Desliga()
Acelera()
Freia()
...
Partes
Motor
Câmbio
Chassis
Banco 1
...
Para criar instância
Dê Cor
Crie Motor
Crie Câmbio
Crie Chassis
...
instância de
instância de
instância de
Carro 1
2001
verde
Escort
122 km/h
...
Carro 2
2004
branco
Palio
111 km/h
...
Carro 3
2003
preto
Gol
113 km/h
...


Classe Jogador
Atr. Primitivos:
Nome
Idade
Endereço
Comportamento:
Leia nome
Corra
Salte
Chute
Atributos:
Tronco
Perna Direita
Perna Esquerda
Cabeça
Crie Instância:
Crie Cabeça
Crie Tronco
Crie Nome
Crie Perna Direita
Crie Perna Esquerda
instância de
instância de
instância de
instância de
Dida
Ronaldo
Juninho

Atributos e Métodos

- Atributos
  - As informações e partes que compõem o objeto
  - Os valores dos atributos definem o estado do objeto
- Métodos
  - Funções e Procedimentos que alteram o estado do objeto e modela o seu comportamento


Classe Carro
Informação
Ano Fabricação
Cor
Marca
Velocidade Máxima
...
Comportamentos
Liga()
Desliga()
Acelera()
Freia()
...
Partes
Motor
Câmbio
Chassis
Banco 1
...
Para criar instância
Dê Cor
Crie Motor
Crie Câmbio
Crie Chassis
...
instância de
instância de
instância de
Carro 1
2001
verde
Escort
122 km/h
...
Carro 2
2004
branco
Palio
111 km/h
...
Carro 3
2003
preto
Gol
113 km/h
...

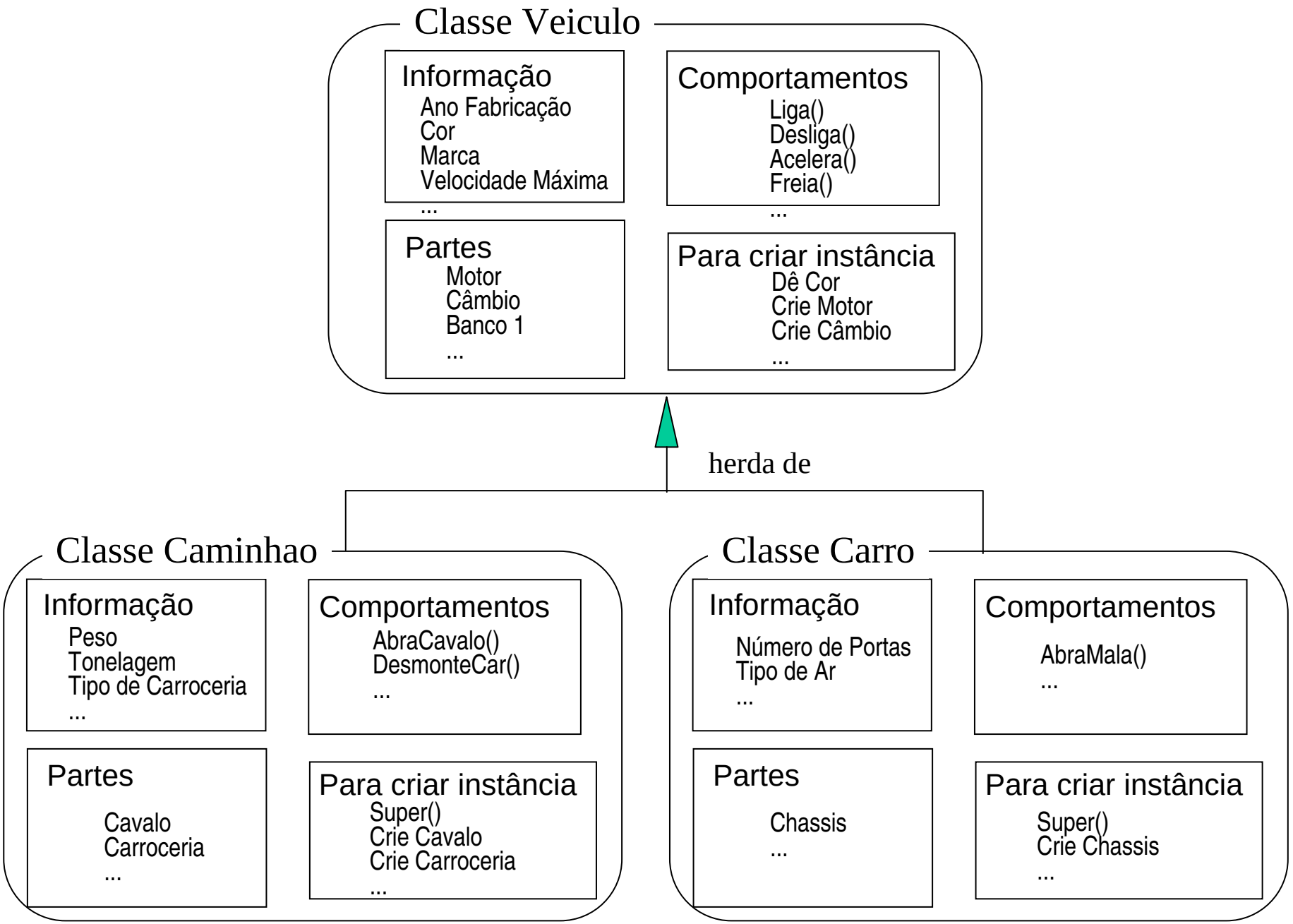
Herança (1)

- **Herança:**
  - Quando várias classes têm informações e comportamentos em comum, estes podem ser abstraídos para evitar redundância de implementação
  - As informações e comportamentos comuns podem ser colocados em uma classe mais genérica, chamada superclasse
  - As classes mais específicas podem então **herdar** estas informações e procedimentos da superclasse

Considere como exemplo a necessidade de se construir Caminhões e Ônibus, além de Carros. Os pontos em comum destas classes poderiam ser abstraídos para uma superclasse “Veículo”.

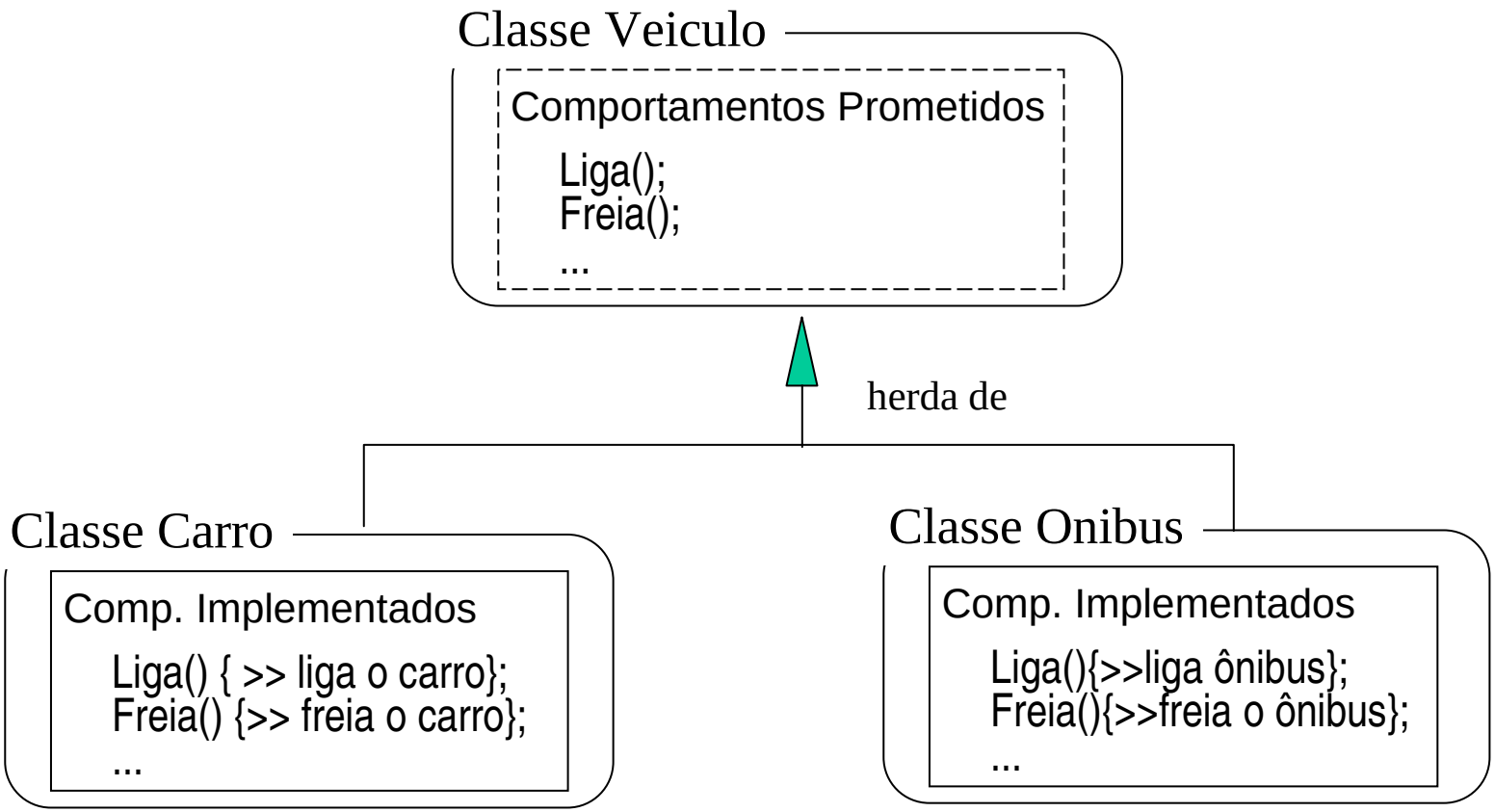
Herança (2)

- Através da herança, descrições comuns podem ser re-utilizadas, promovendo o conceito de código re-utilizável.
- Herança elimina a redundância pelo fato de classes descendentes apenas implementarem informações e comportamentos adicionais, que as diferenciam das demais. Isto conduz a sistemas menores e mais fáceis de compreender.
- Quando modificações são implementadas nas partes comuns (superclasses), todas as classes descendentes automaticamente herdam estas modificações. Isto possibilita a criação de modelos mais fáceis de se modificar.


Classe Veiculo
Informação
Ano Fabricação
Cor
Marca
Velocidade Máxima
...
Comportamentos
Liga()
Desliga()
Acelera()
Freia()
...
Partes
Motor
Câmbio
Banco 1
...
Para criar instância
Dê Cor
Crie Motor
Crie Câmbio
...
herda de
Classe Caminhao
Informação
Peso
Tonelagem
Tipo de Carroceria
...
Comportamentos
AbraCavalo()
DesmonteCar()
...
Partes
Cavalo
Carroceria
...
Para criar instância
Super()
Crie Cavalo
Crie Carroceria
...
Classe Carro
Informação
Número de Portas
Tipo de Ar
...
Comportamentos
AbraMala()
...
Partes
Chassis
...
Para criar instância
Super()
Crie Chassis
...

Polimorfismo

- superclasses também podem ser usadas para "prometer" um comportamento que as subclasses devem implementar
- desta maneira o remetente de uma mensagem só precisa saber a um nível abstrato que o objeto tem aquele comportamento
- isto permite criar situações em que o comportamento real de um objeto é definido em tempo de execução



Ao saber que um objeto é um *Veiculo*, outros objetos sabem que podem enviar para ele a mensagem *Liga()*, independente de sua classe real. Devido as diferentes implementações, esta mensagem gerará comportamentos diferentes para diferentes classes de objetos.

Exemplo de Polimorfismo

Considere que a variável “v” é definida com um veículo.

```
função ligueVeiculo( Veiculo v) {
   v.liga();
}
```

Qualquer carro ou ônibus pode então ser associada a “v”:

```
Carro c1 = CrieCarro();
Onibus o3 = CrieOnibus();
ligueVeiculo(c1);
ligueVeiculo(o3);
```

Ambos veículos são ligados através da mensagem v.liga(), mas se as classes carro e ônibus têm implementações próprias para a rotina liga(). A mensagem “liga()” chamará rotinas diferentes quando “v” for um carro ou um caminhão.