Paradigmas e Linguagens de Programação

Prof. Josino Rodrigues Neto
Email: josinon@gmail.com

Linguagens de Programação: Módulo 1

Introdução às Linguagens de Programação

Sobre o Curso

Objetivos do Curso

- Apresentar os principais paradigmas de programação.
- Introduzir programação imperativa (usando Java ou C).
- Introduzir programação orientada a objetos (usando Java).
- Introduzir programação funcional (usando Lisp).
- Introduzir programação Lógica(usando Prolog).
- Introduzir conceitos básicos de implementação de linguagens de programação.

Sobre Este Curso

Qualquer notação para descrição de algoritmos e estruturas de dados que é “implementada” em uma computador pode ser chamada de uma **Linguagem de Programação**.

Este curso se discutirá justamente o que é uma “implementação” de uma LP, quais os paradigmas ou tipos de implementação de LPs, e algumas das LPs mais significativas dentro do universo destes paradigmas.

Como este é um curso relativamente curto nós discutiremos um paradigma e uma linguagem de programação em um pouco mais de profundidade: a *programação orientada a objetos* e a linguagem *Java*.

Ementa

- Introdução às LPs
- Programação Funcional e LISP
- Programação Lógica e Prolog
- Programação OO e Java
- Linguagens Dinâmicas e de Script
- Organização de LPs

Bibliografia

Qualquer bom livro!

Recomendados:

- Programming Language Pragmatics, Michael L. Scott
- Practical Common Lisp, Peter Seibel , Apress © 2005
  - Disponível em http://www.gigamonkeys.com/book/
- The Art of Prolog Programming, Sterling and Shapiro, MIT Press 1999.

- Java How to Program (7th Edition) Harvey & Paul Deitel
- David Flanagan, Java in a Nutshell, O´Reilly and Associates, 4a edição, 2002, *ou , Java - O Guia Essencial, Editora Campus*.
- Abelson, Jerry Sussman and Sussman, Structure and Interpretation of Computer Programs (MIT Press, 1984; ISBN 0-262-01077-1) http://mitpress.mit.edu/sicp/
- Friedman, Wnad, Haynes, Fundamentos de Linguagens de Programação, Editora Berkley
- Conceitos de Linguagens de Programação, por Robert Sebesta, Bookman, 4a edição.
- Steele, G. L., Jr. Common Lisp: The Language. Digital Press, Bedford, Massachusetts, 1990
- Programming in Prolog, W. F. Clocksin e C. S. Mellish, Springer-Verlag, 4a edição, 1994.
- Programming Languagens Design and Implementation, Terrence W. Pratt e Marvin V. Zelkowitz, Prentice Hall, 4a edição.

Assumido e Não Assumido

Assumido

- Conhecimento de uma linguagem de programação de alto nível (ex. C ou Pascal)

Não Assumido

- Prévio conhecimento de outros paradigmas ou linguagens de programação.

Parte 1 - Introdução às Linguagens de Programação

Motivação

Porque Estudar LPs ? (1)

Já em 1969, SAMMET listou 120 LPs que eram relativamente bem difundidas. Desde então muitas centenas de outras LPs surgiram (e desapareceram).

A maioria dos programadores, todavia, nunca chega a usar mais do que umas poucas linguagens. Muitos não usam mais que uma ou duas em sua inteira vida profissional.

Por que então estudar linguagens de programação?

Porque Estudar LPs (2)

- Melhorar sua habilidade de desenvolver algoritmos (ex., uso eficiente de recursividade)

- Melhorar sua habilidade de usar a LP (ex., uso eficiente dos recursos de gerenciamento de memória)

- Aumentar o seu vocabulário de construção úteis de programação (ex., declarações repita até <*repeat-until*> e chaveamento <*case*> ou <*switch*>)

Porque Estudar LPs (3)

• Melhor escolher uma linguagem de programação por conhecer seus pontos fortes e fracos (++ depende muito do conhecimento dos programadores).

• Torna mais fácil aprender novas LPs e acompanhar a tecnologia (ex., quanto mais LPs você souber, mais fácil se torna aprender uma nova LP).

• Torna mais fácil projetar novas linguagens de programação (ex., algumas interfaces de usuário podem adquirir contornos de uma "linguagem de programação").

• E por último, abrangência de formação.

Leituras Para Próxima Aula:

A vingança dos nerds

http://www.paulgraham.com/icad.html

A linguagem de cem anos

http://www.paulgraham.com/hundred.html

Riscos das escolas de Java

http://www.joelonsoftware.com/articles/ThePerilsofJavaSchools.html

Comparação de Sintaxe

http://merd.sourceforge.net/pixel/language-study/syntax-across-languages/

História

Um Pouco de História (i)

1951-55

Hardware: computadores a válvula e memórias com linhas de retardo de mercúrio

Métodos: Linguagens de máquina, primórdios do uso de sub-rotinas e estruturas de dados.

PLs: uso experimental de compiladores de expressões.

1956-60

Hardware: armazenamento em fitas magnéticas, memórias de núcleo de ferrite, circuitos transistorizados.

Métodos: início do uso de compiladores, otimização de código, interpretadores, processamento de listas, e métodos dinâmicos de armazenamento.

PLs: FORTRAN, ALGOL 58, ALGOL 60, COBOL, LISP.

Um Pouco de História (ii)

1961-65

Hardware: famílias de arquiteturas compatíveis, discos magnéticos
Métodos: SO multi-programados, compiladores direcionados à sintaxe.
PLs: COBOL61, novo ALGOL 60, SNOBOL, JOVIAL, notação APL.

1966-70

Hardware: crescimento explosivo de capacidade e velocidade e decréscimo de custo, minicomputadores, micro-programação, e circuitos integrados.
Métodos: sistemas interativo de tempo compartilhado, compiladores com otimização, sistemas de escrita de tradução.
PLs: APL, FORTRAN 66, COBOL 65, ALGOL 68, SNOBOL 4, BASIC, PL/I, SIMULA 67, ALGOL-W.

Um Pouco de História (iii)

1971-75

Hardware: primeiros microprocessadores/microcomputadores, auge dos minicomputadores, sistema de armazenamento de massa, memórias de semicondutores.

Métodos: Verificação de programas, programação estruturada, início da engenharia de software como disciplina.

PLs: Pascal, COBOL 74, PL/I padrão, C, Scheme, Prolog.

1976-80

Hardware: microcomputadores comerciais, grandes sistemas de armazenamento de massa, programação distribuída.

Métodos: tipos abstratos de dados, semânticas formais, programação de tempo real e sistemas embarcados.

PLs: Smalltalk, Ada, FORTRAN 77, ML.

Um Pouco de História (iv)

1981-85

Hardware: computadores pessoais, primeiras estações de trabalho, vídeo-games, redes locais, Arpanet.

Métodos: Programação orientada à objetos, sistemas interativos de programação, editores baseados em sintaxe.

PLs: Turbo Pascal, Smalltalk-80, uso de Prolog, Ada 83, Postscript.

1986-90

Hardware: idade do microcomputador, estações de trabalho de preço médio, arquitetura RISC, comunicação global de dados, Internet.

Métodos: computação cliente-servidor (e-mail, ftp, terminais remotos)

PLs: FORTRAN 90, C++, SML (Standard ML).

Um Pouco de História (v)

1991-95

Hardware: estações de trabalho rápidas e baratas, arquiteturas maciçamente paralelas, imagem, voz e fax sobre a Internet.

Métodos: sistemas abertos, ambientes de desenvolvimento, infraestruturas nacionais de comunicação de dados, surgimento da WWW.

PLs: Ada95, linguagens de processos (TCL, PERL), HTML.

1996-00

Hardware: popularização de computadores e estação de trabalho, surgimento dos "equipamentos de informação" (celulares e PDAs), vídeo sobre a Internet.

Métodos: infraestruturas mundiais de comunicação de dados, sistemas Web, arquiteturas 3-tiers e n-tiers, máquinas virtuais independentes de plataforma.

PLs: Java, XML, etc ... .

E o Momento Atual

2000-5
Hardware: sistemas de informação domésticos, computadores de baixíssimo custo, e celulares; disseminação de redes wireless.
Métodos: multicasting, identidade digital, sistemas Web semânticos, sistemas de auxílio à GC, computação social.
PLs: .Net, Java, XML, J2ME, Python, PHP

2005-10
Hardware: smartphones, equipamentos de informação convergentes TV digital; fazenda de servidores em nuvem.
Métodos: multicasting a nível pessoal (Móvel), computação em nuvem, sistemas convergentes, redes sociais.
PLs: Ruby on Rails, Python, Groovy

Parte 2 - Paradigmas de Programação

Introdução

- **O que é uma linguagem?**
  - "Conjunto de regras que estabelecem normas de comunicação".
  - Caso as partes envolvidas na comunicação falem línguas diferentes, surge a necessidade de um tradutor (intermediário).
- **O que é uma linguagem de Programação?**
  - Também é um conjunto de regras que estabelecem normas de comunicação entre o programador e o computador.
  - Uma LP deve ser extremamente formal e exata.
  - Uma linguagem ambígua torna-se difícil de ser traduzida para uma linguagem de máquina.

Introdução

- **Evolução das LPs**
  - **Primeira Geração: Linguagem de máquina**
    - Código de Máquina (0s e 1s).
  - **Segunda Geração:**
    - Linguagem de Montagem - Assembler
  - **Terceira Geração**
    - Imperativas: FORTRAN, Cobol, Basic, Algol, ADA, Pascal, C
    - Lógicas e Funcionais: LISP, ML, Prolog
  - **Quarta Geração**
    - Geradores de Relatórios, Linguagens de Consultas: SQL, CSP
  - **Quinta Geração**
    - LOO : Smalltalk, Java, Eiffel, Simula 67
  - **Sexta Geração ?**
    - Web e Linguagens Dinâmicas : Python, JavaScript, Ruby

Ranking de Popularidade 2006

Position Feb 2012	Position Feb 2011	Delta in Position	Programming Language	Ratings Feb 2012	Delta Feb 2011	Status
1	1	=	Java	17.050%	-1.43%	A
2	2	=	C	16.523%	+1.54%	A
3	6	↑↑↑	C#	8.653%	+1.84%	A
4	3	↓	C++	7.853%	-0.33%	A
5	8	↑↑↑	Objective-C	7.062%	+4.49%	A
6	5	↓	PHP	5.641%	-1.33%	A
7	7	=	(Visual) Basic	4.315%	-0.61%	A
8	4	↓↓↓↓	Python	3.148%	-3.89%	A
9	10	↑	Perl	2.931%	+1.02%	A
10	9	↓	JavaScript	2.465%	-0.09%	A
11	13	↑↑	Delphi/Object Pascal	1.964%	+0.90%	A
12	11	↓	Ruby	1.558%	-0.06%	A
13	14	↑	Lisp	0.905%	-0.05%	A
14	26	↑↑↑↑↑↑↑↑↑↑	Transact-SQL	0.846%	+0.29%	A
15	17	↑↑	Pascal	0.813%	+0.08%	A
16	22	↑↑↑↑↑↑	Visual Basic .NET	0.796%	+0.21%	A--
17	32	↑↑↑↑↑↑↑↑↑↑	PL/SQL	0.792%	+0.38%	A
18	24	↑↑↑↑↑↑	Logo	0.677%	+0.10%	B
19	16	↓↓↓	Ada	0.632%	-0.17%	B
20	25	↑↑↑↑↑	R	0.623%	+0.06%	B

Linguagens Imperativas

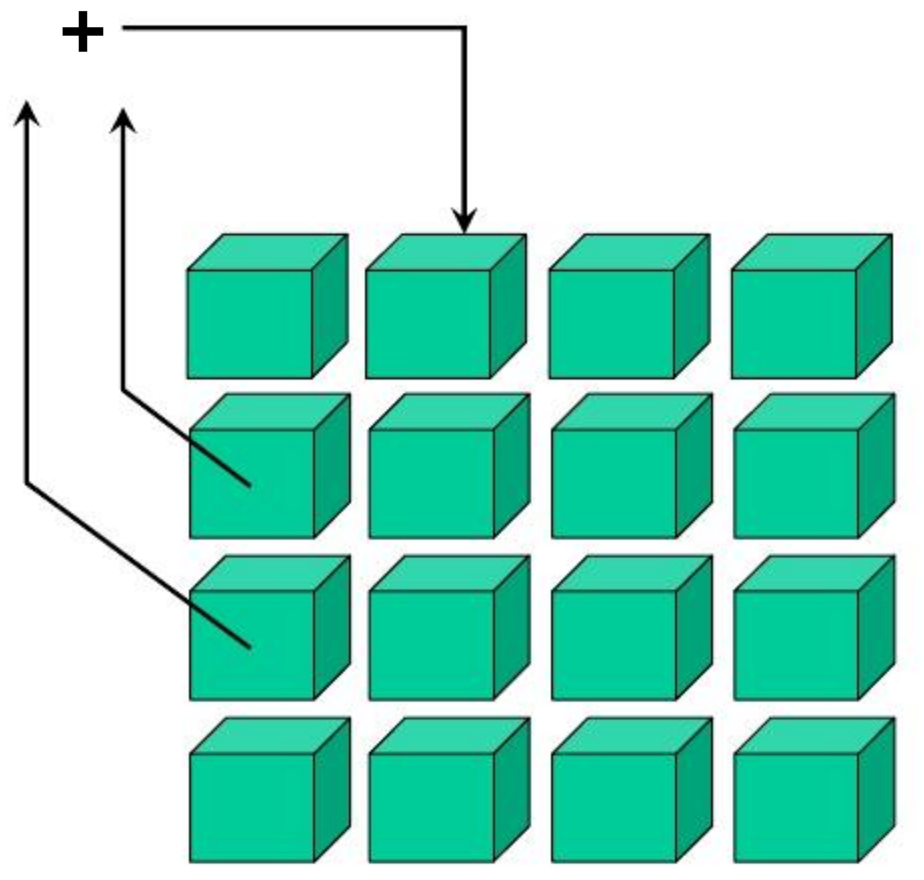
Linguagens Imperativas (i)

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:

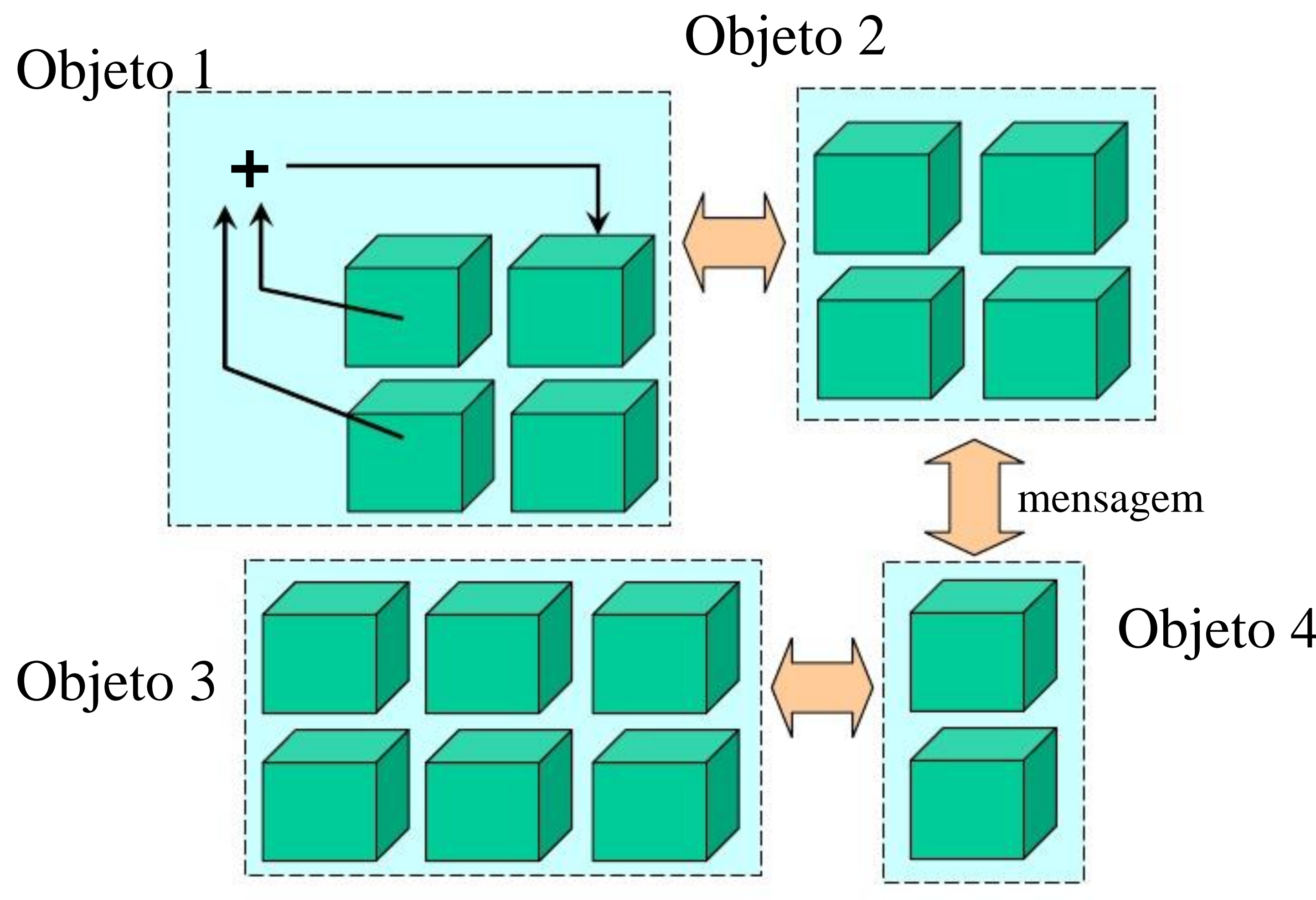
C, FORTRAN, Algol, Pascal, COBOL, “C++”, “Java”, etc.

Linguagens Orientadas a Objetos

Linguagens OO (i)

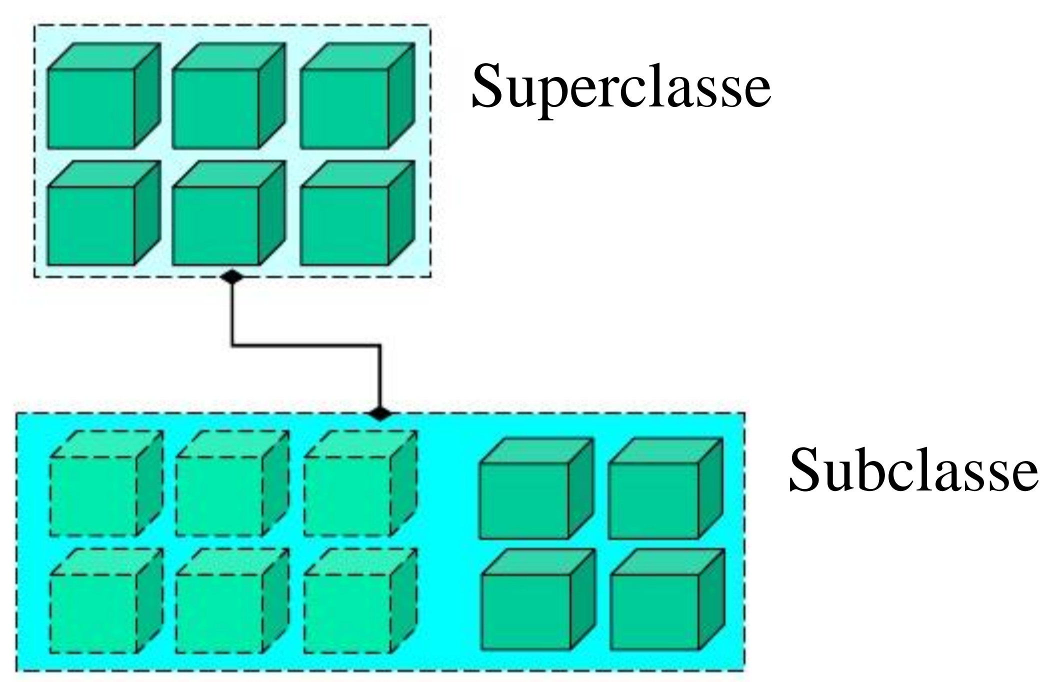
Linguagens orientadas à objeto são linguagens que estendem a linguagens imperativas com a introdução do conceitos de objetos. Objetos são conjuntos de dados geralmente complexos com funções bem definidas que podem ser aplicadas eles. Funções e dados são **encapsulados** pelo objeto de forma a expor somente 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.

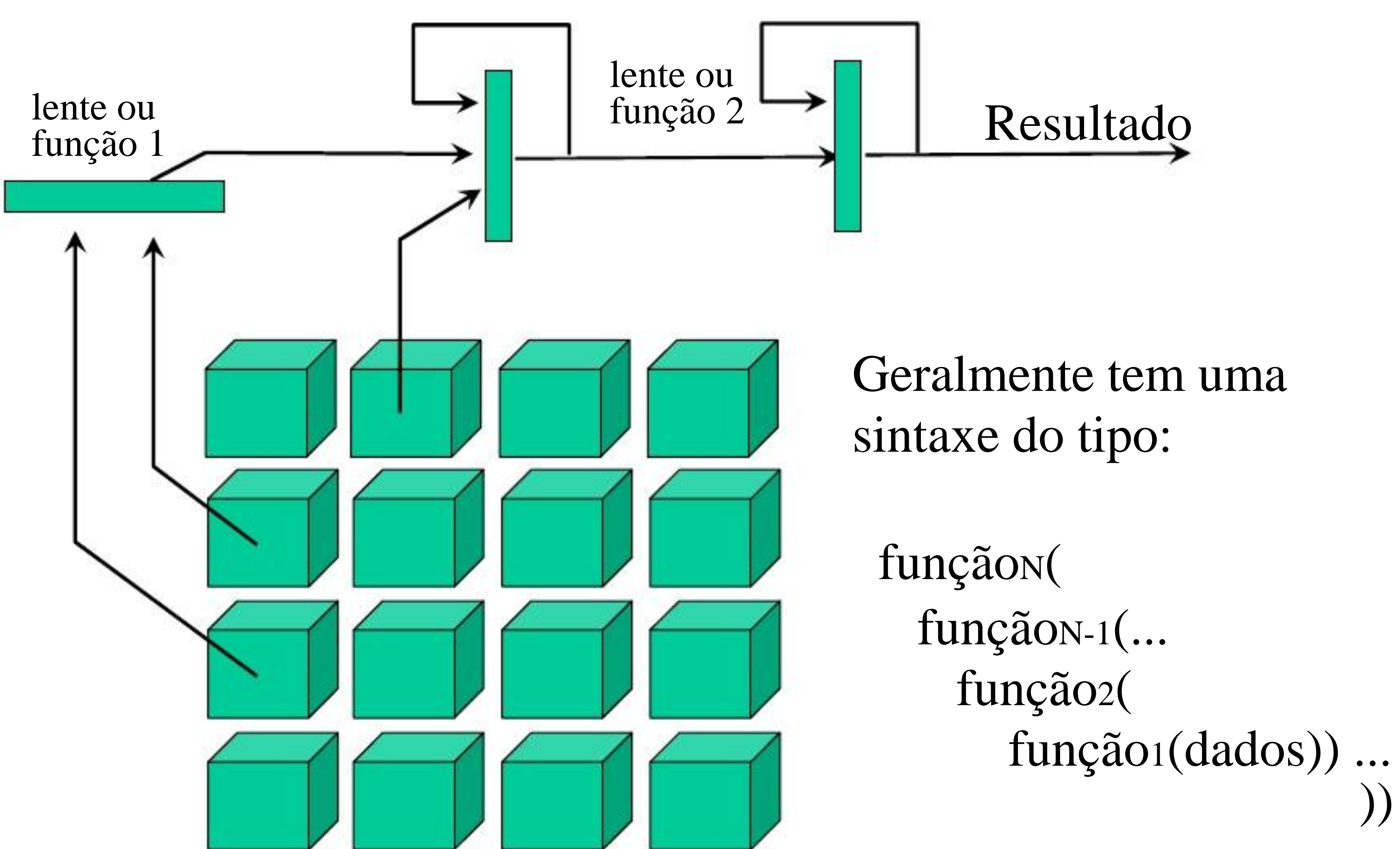
Linguagens Funcionais

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.

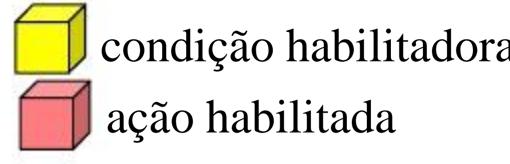
Linguagens Lógicas

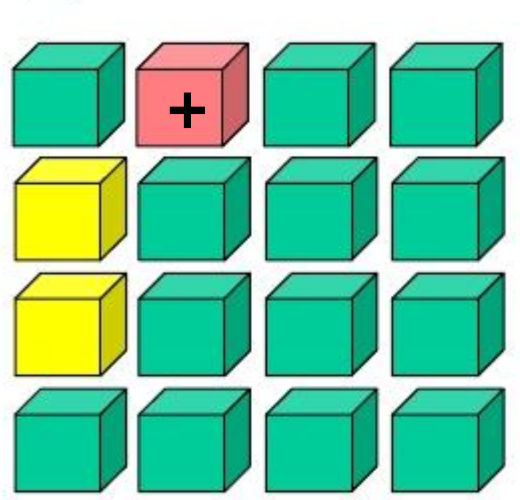
Linguagens Lógicas (i)

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

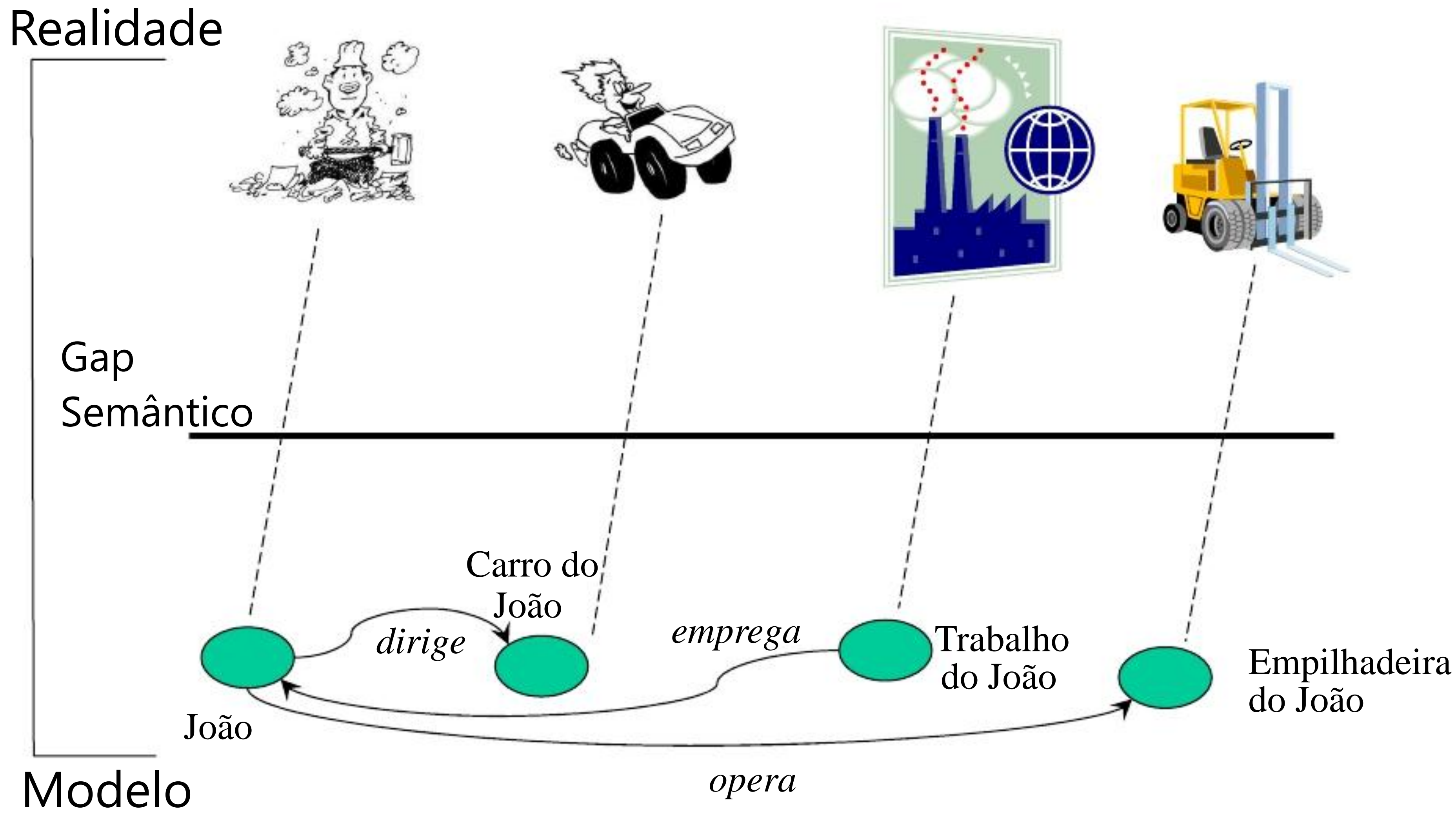
Programação imperativa separa (ou tende a separar) dados dos procedimentos usados para manipular estes dados.

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.

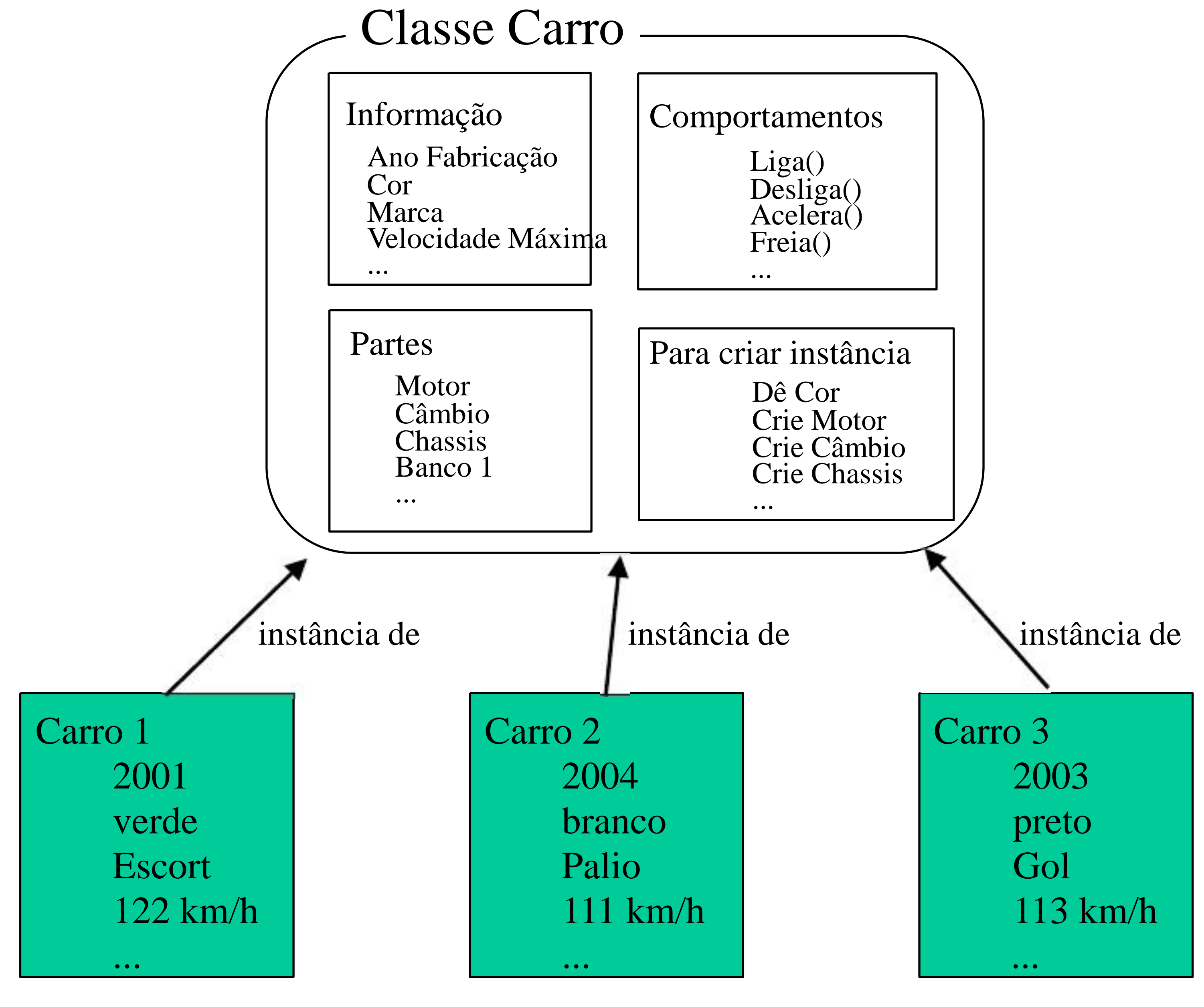
- **Instância**: um objeto que pertença a uma classe é chamado de instância desta classe.

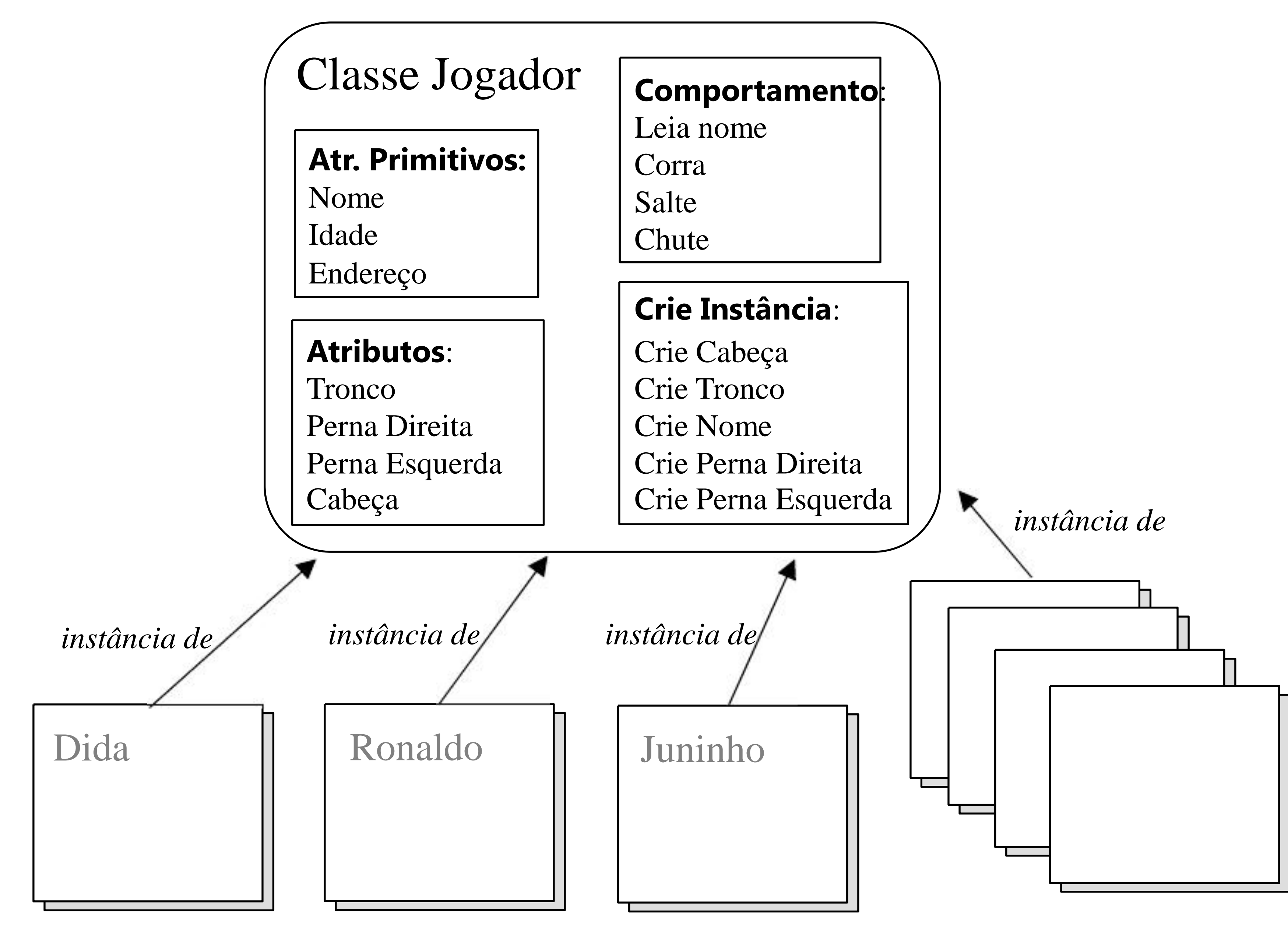
Mais Classes e Instâncias

- 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).

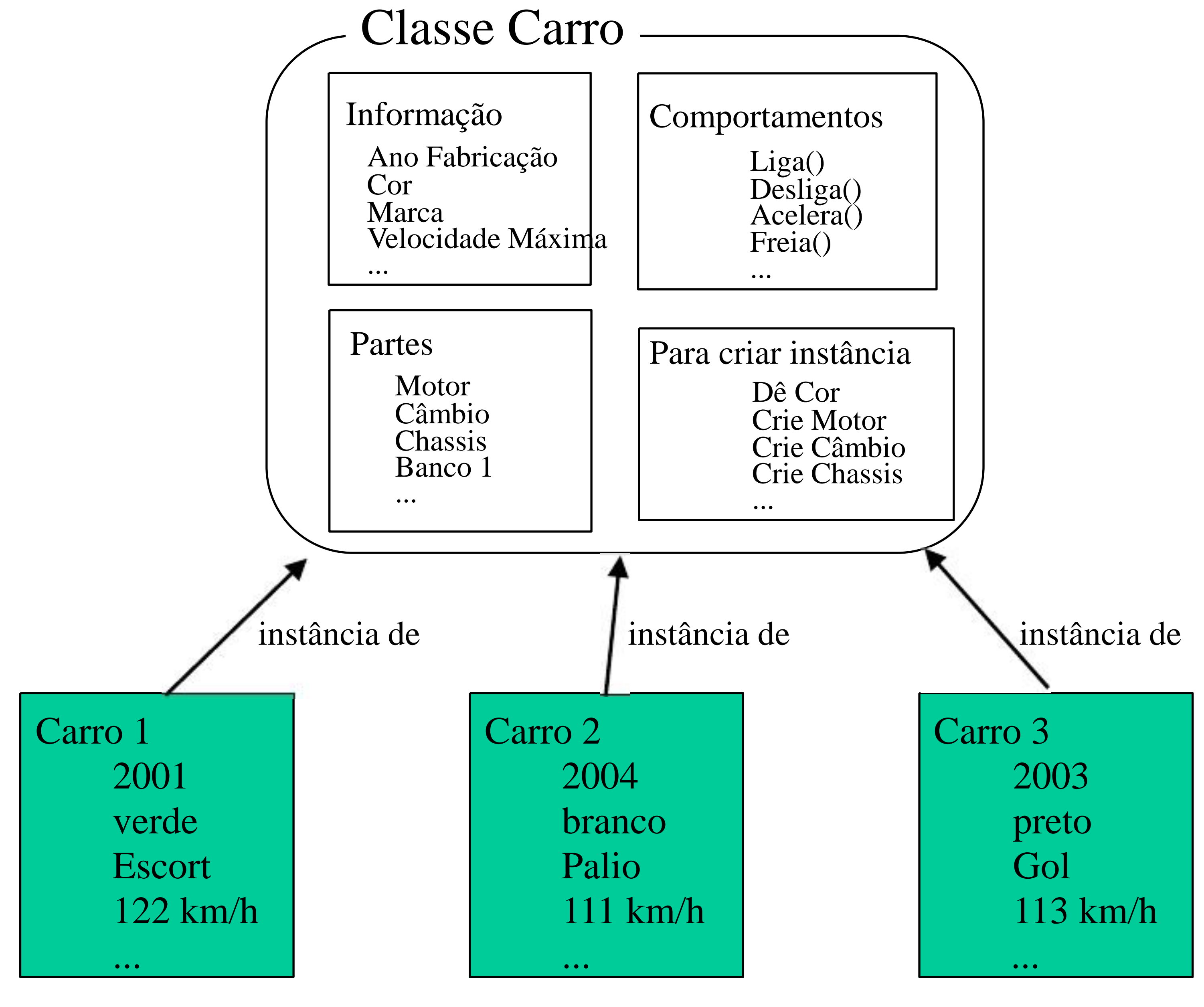

Classe Carro
Informação
Ano Fabricação
Cor
Marca
Velocidade Máxima
...
Comportamentos
Liga()
Desliga()
Acelera()
Freia()
...
Partes
Motor
Câmbio
Chassis
Banco 1
...
Para criar instância
Dê Cor
Crie Motor
Crie Câmbio
Crie Chassis
...
instância de
instância de
instância de
Carro 1
2001
verde
Escort
122 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
Atributos:
Tronco
Perna Direita
Perna Esquerda
Cabeça
Comportamento:
Leia nome
Corra
Salte
Chute
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
...

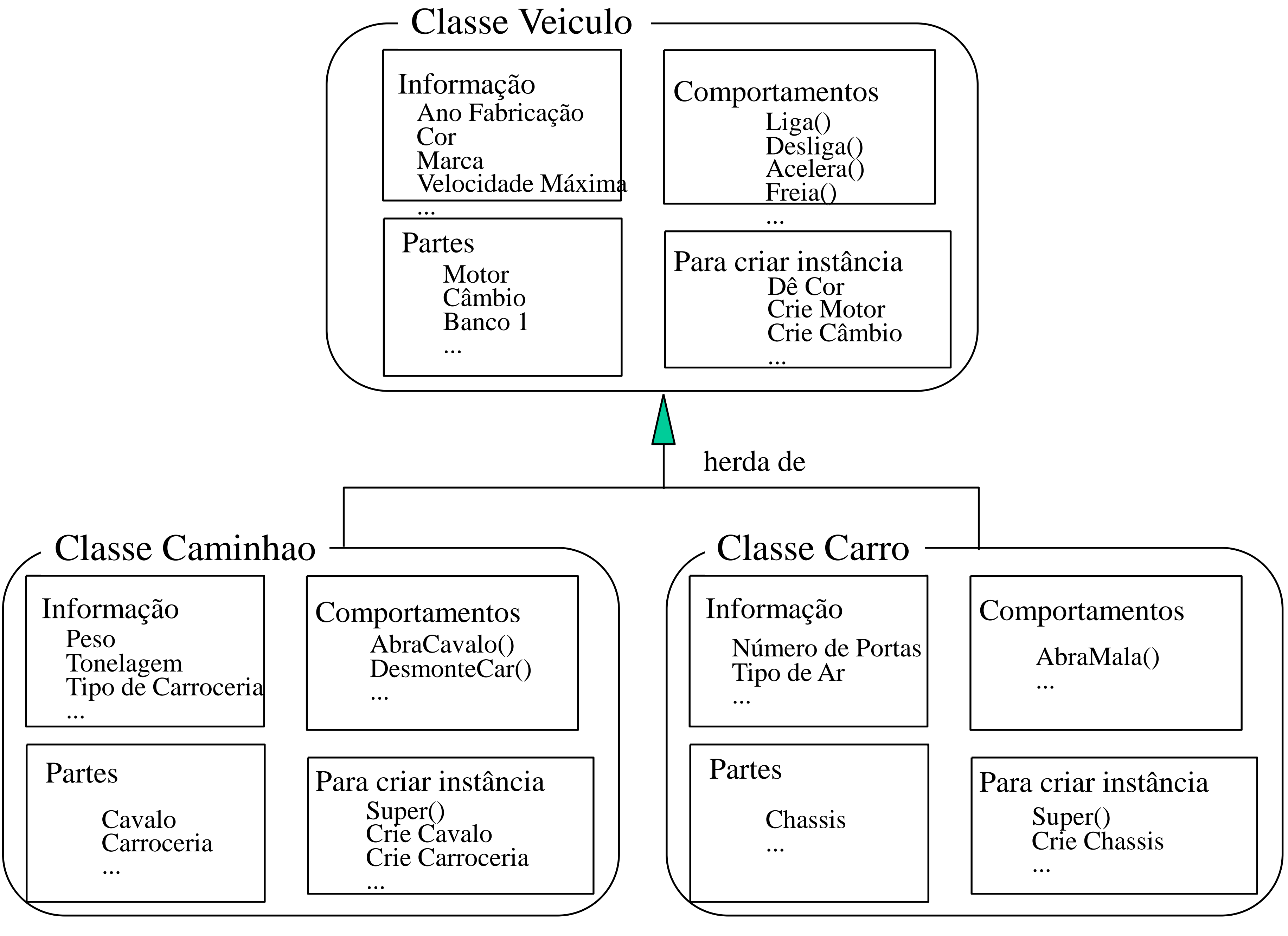
Herança (1)

- **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.

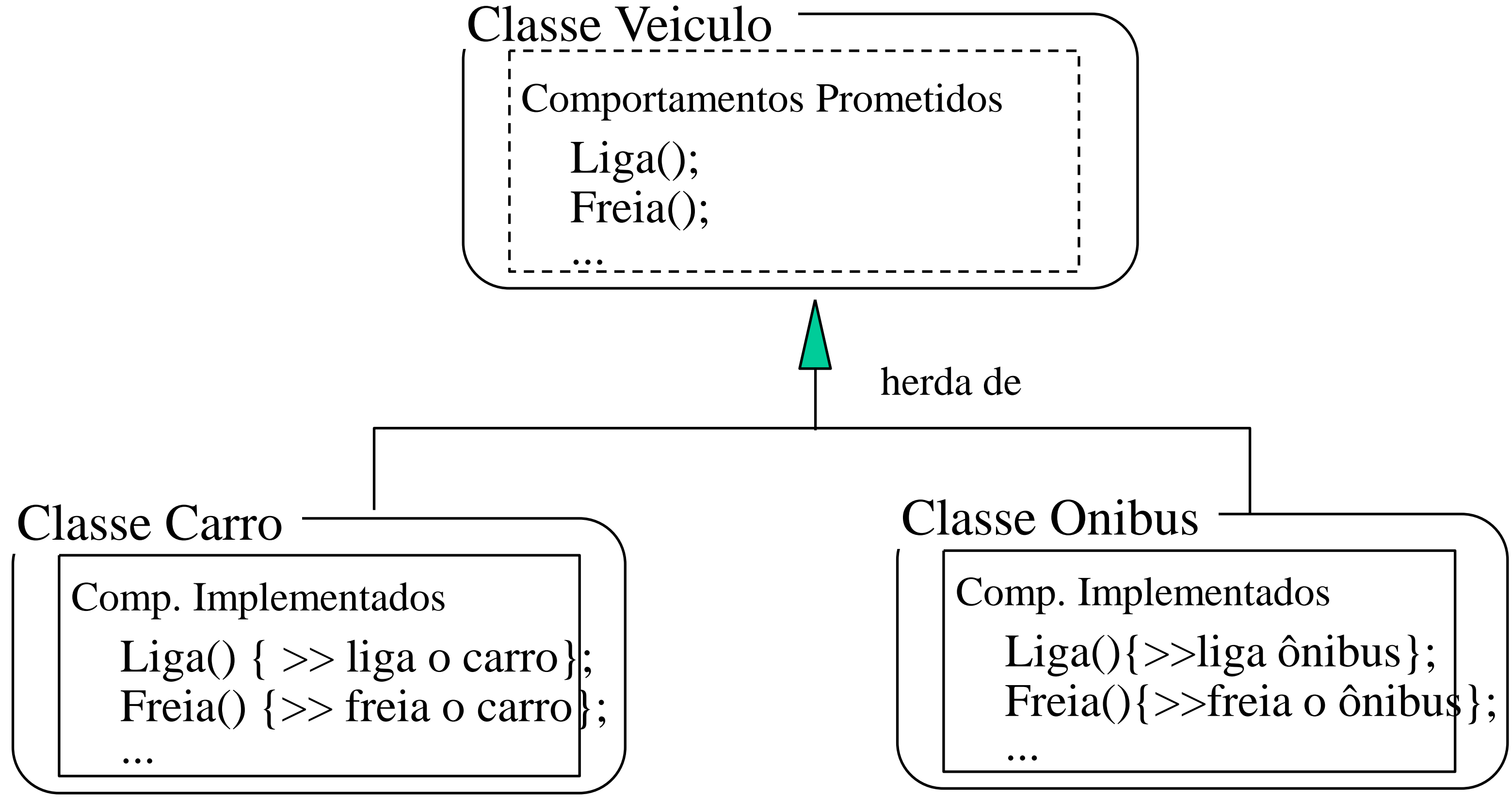

Classe Veiculo
Informação
Ano Fabricação
Cor
Marca
Velocidade Máxima
...
Comportamentos
Liga()
Desliga()
Acelera()
Freia()
...
Partes
Motor
Câmbio
Banco 1
...
Para criar instância
Dê Cor
Crie Motor
Crie Câmbio
...
herda de
Classe Caminhao
Informação
Peso
Tonelagem
Tipo de Carroceria
...
Comportamentos
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.