ACH2002

Orientação a Objetos

Professores:
Delano Medeiros Beder
Fátima L. S. Nunes

EACH – USP

O que é abstração ?

Abstração

s.f. Operação do espírito, que isola de uma noção um elemento, negligenciando os outros.

Resultado desta operação: a brancura considerada em geral, sem ser aplicada a um objeto, é uma abstração.

Fazer abstração de uma coisa, não levá-la em consideração.

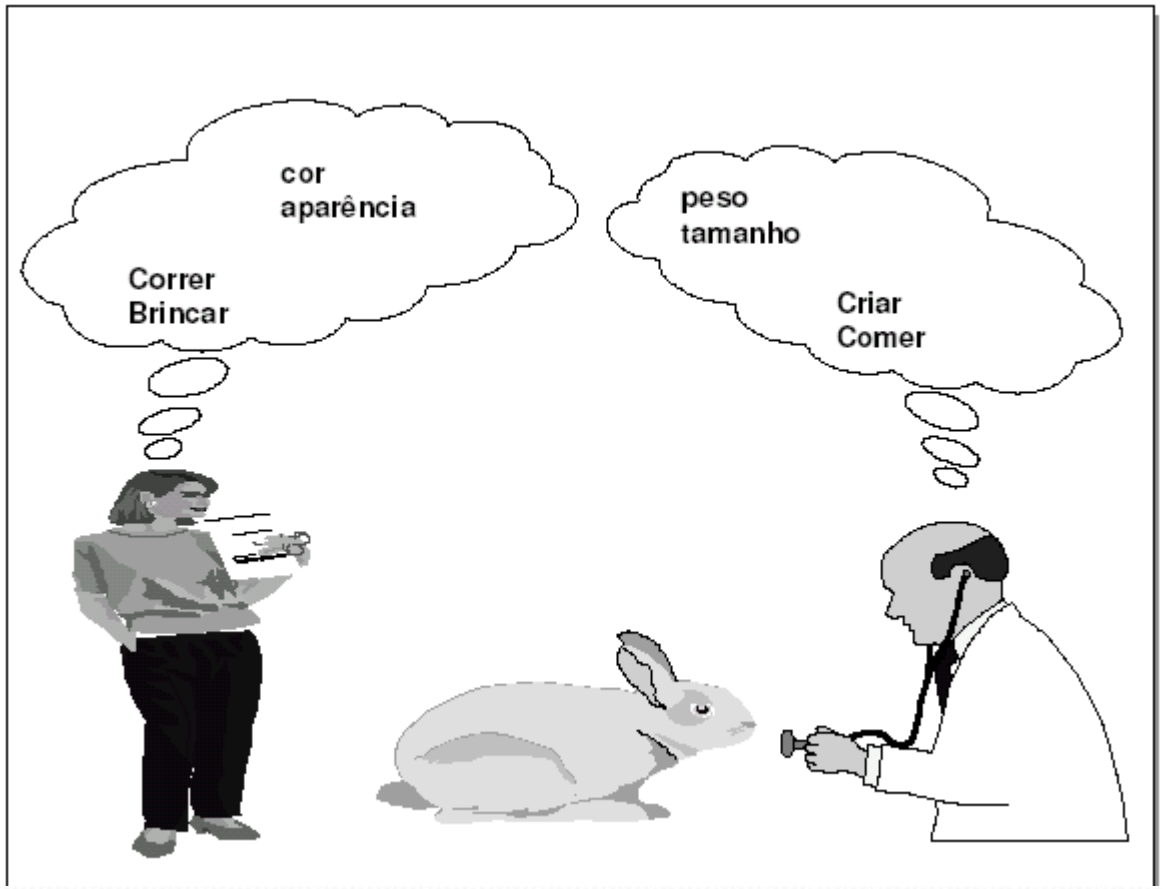
S.f.pl. Idéias quiméricas, desvinculadas da realidade: perder-se em abstrações.

Abstração

- Focar em aspectos inerentes e essenciais de uma entidade
  - em desenvolvimento de software:
    - O QUE É e o QUE FAZ um objeto, e não como faz
  - evita comprometimento prematuro com detalhes

Abstração

- Depende do ponto de vista



Tipos Abstratos de Dados

- Objeto é definido a partir de um tipo abstrato de dados
- Combinação de dados e operações (sobre eles) em um elemento único
- Ocultamento da informação e implementação
  - Representação de objetos de um determinado tipo não é visível pelos clientes que usam o usam
  - Únicas operações diretas sobre objetos são aquelas oferecidas na definição do tipo.

Tipos Abstratos de Dados

- Ocultamento da informação e implementação
  - Representação de objetos de um determinado tipo não é visível pelos clientes que usam
  - Únicas operações diretas sobre objetos são aquelas oferecidas na definição do tipo.
- Qual a vantagem do ocultamento?

Tipos Abstratos de Dados

- Ocultamento da informação e implementação
  - implementação do tipo pode ser alterada sem afetar as unidades de programa que fazem uso dele (contanto que mantenha as mesmas operações)
  - Aumenta a confiabilidade, pois nenhuma outra unidade de programa pode mudar, acidentalmente ou intencionalmente, as representações do tipo, aumentando a integridade de tais objetos

Paradigma OO

- Objeto:
  - abstração de uma entidade real, cujas características e comportamento são conhecidos
  - se apresenta a outras entidades por meio de uma interface bem definida
  - Exemplo:
    - **carro** é um objeto
      - **características:** modelo, marca, ano, cor
      - **comportamento:** acelera, estaciona, liga

Paradigma OO

- Visão Interna: define a estrutura e o comportamento do objeto, ou seja, define características e comportamentos

- Visão Externa: interface que define como o objeto é visto por outros objetos

- Mensagem: comunicação entre objetos

Paradigma OO

- POO é um paradigma de programação
- A “unidade” de programação é a classe de objetos
- Quais os componentes de uma classe?

Paradigma OO

- POO é um paradigma de programação
- A “unidade” de programação é a classe de objetos
- Quais os componentes de uma classe?
  - atributos (características dos objetos)
  - métodos (comportamento dos objetos)

Orientação a Objetos

- Extensão do conceito de Tipos Abstratos de Dados
  - Combinação de dados e operações (sobre eles) em um elemento único
- Classe: definição do Tipo Abstrato de Dados
- Objeto: cada instância derivada da classe
- Representa em software entidades que encontramos no mundo real

Orientação a Objetos

- Classe: definição do Tipo Abstrato de Dados
- Objeto: cada instância derivada da classe

- Exemplo:
  - Classe Carro (define um Tipo Abstrato de Dados denominado Carro)
  - Objetos instanciados a partir da classe Carro:
    - bmw, mercedes, corsa, fiesta

Orientação a objetos

- Classes/Objetos
- Encapsulamento
- Ocultamento da informação e implementação (Abstração)
- Retenção de estado
- Identidade de objeto
- Mensagens
- Herança
- Polimorfismo/Vinculação tardia (*late binding*)
- Relacionamento entre classes/objetos

Orientação a objetos

- Classes/Objetos
- Encapsulamento
- Ocultamento da informação e implementação (Abstração)
- Retenção de estado
- Identidade de objeto
- Mensagens
- Herança
- Polimorfismo/Vinculação tardia (*late binding*)
- Relacionamento entre classes/objetos

Classes / Objetos

- Classe: define um conjunto de objetos com a mesma estrutura e o mesmo comportamento
  - Carro, Pessoa, Aluno são exemplos de classes
  - **BMW** e **Ferrari** são dois objetos da classe Carro
  - **João** e **Maria** são dois objetos da classe Pessoa

- A classe é o que você projeta e programa
- O objeto é o que você cria (a partir de uma classe) em tempo de execução

Classes / Objetos

- Todo objeto é criado a partir (é uma instância) de uma classe
- Dados (atributos) são associados a cada objeto
- Classes podem conter atributos gerais (e operações) independentes das diversas instâncias
  - Métodos e atributos estáticos (C++ e Java)

Objetos

- Exemplo: um Estudante e uma Disciplina podem ser considerados objetos, pois qualquer um deles pode ser definido em termos de um conjunto de atributos e operações
- Objetos podem se relacionar um com o outro
  - um Estudante pode Cursar uma Disciplina
  - o relacionamento Cursar define uma conexão específica entre Estudante e Disciplina

Objetos

Objeto = Atributos + Métodos + Encapsulamento

dados — funcionalidade (comportamento)

- Outros exemplos de objetos:
  - um computador
  - uma tela de um aplicativo do computador
  - uma reserva de livro
  - um livro
  - um processo de identificação de acessos ilegais numa rede de computadores
  - um acesso ilegal

Atributos

- Representam um conjunto de informações, ou seja, elementos de dados que caracterizam um objeto
- Exemplos:
  - objeto Estudante
    - atributos: nome, nro USP, ano de ingresso, data de nascimento, ...
  - objeto Disciplina
    - atributos: nome, código, nro de créditos, pré-requisitos, equivalências, ...
  - objeto Reserva de Vôo
    - atributos: número da reserva, número do vôo, nome passageiro, data validade da reserva, prioridade, ...

Métodos

- Quando um objeto é mapeado dentro do domínio do software, os processos que podem alterar seus atributos (dados) são denominados Operações ou Métodos
- Métodos são invocados por Mensagens
- Cada objeto possui seu próprio conjunto de métodos
- Definições:
  - procedimentos definidos e declarados que atuam sobre o objeto
  - descrição de uma sequência de ações a serem executadas pelo objeto
  - especificação de COMO o objeto deve FAZER alguma coisa
  - são intrínsecos ao objeto e não podem ser separados dele

Modificadores de Visibilidade

- Podem estar presentes tanto para atributos como para métodos
- Em princípio, três categorias de visibilidade podem ser definidas:
  - público: o atributo ou método de um objeto dessa classe pode ser acessado por qualquer outro objeto (visibilidade externa total)
  - privativo: o atributo ou método de um objeto dessa classe não pode ser acessado por nenhum outro objeto (nenhuma visibilidade externa)
  - protegido: o atributo ou método de um objeto dessa classe pode ser acessado apenas por objetos de classes que sejam derivadas dessa por meio do mecanismo de herança

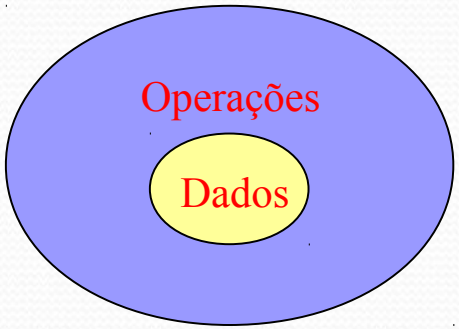
Orientação a objetos

- Classes/Objetos
- Encapsulamento
- Ocultamento da informação e implementação (Abstração)
- Retenção de estado
- Identidade de objeto
- Mensagens
- Herança
- Polimorfismo/Vinculação tardia (*late binding*)
- Relacionamento entre classes/objetos

Encapsulamento

- Outros paradigmas convencionais (estrutural, por exemplo) separam dados e operações sobre eles

- O objeto contém tantos os dados quanto as operações:
  - Dados: atributos
  - Implementação das operações: métodos



Encapsulamento

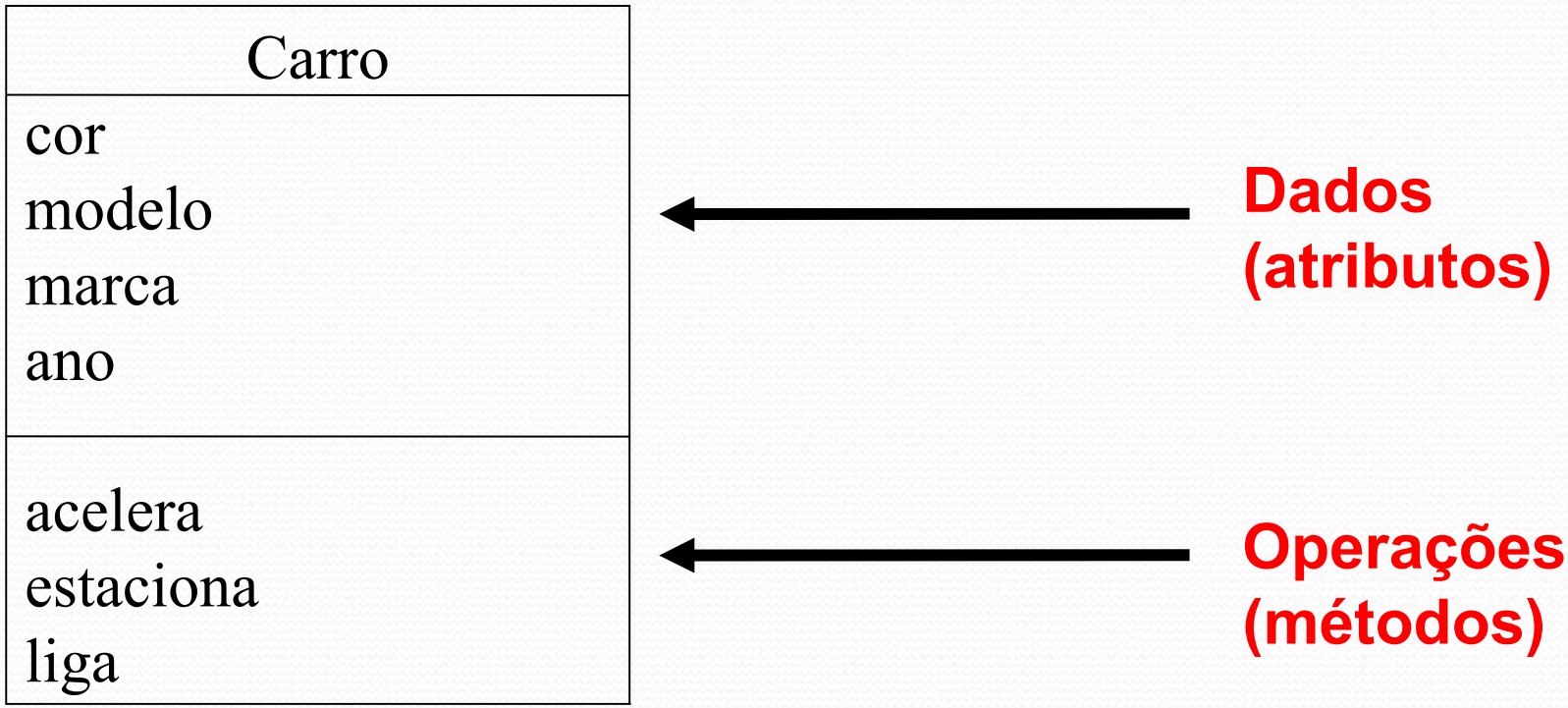
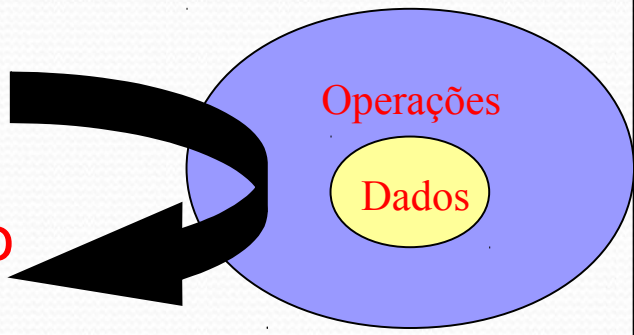


Diagrama de Classes UML

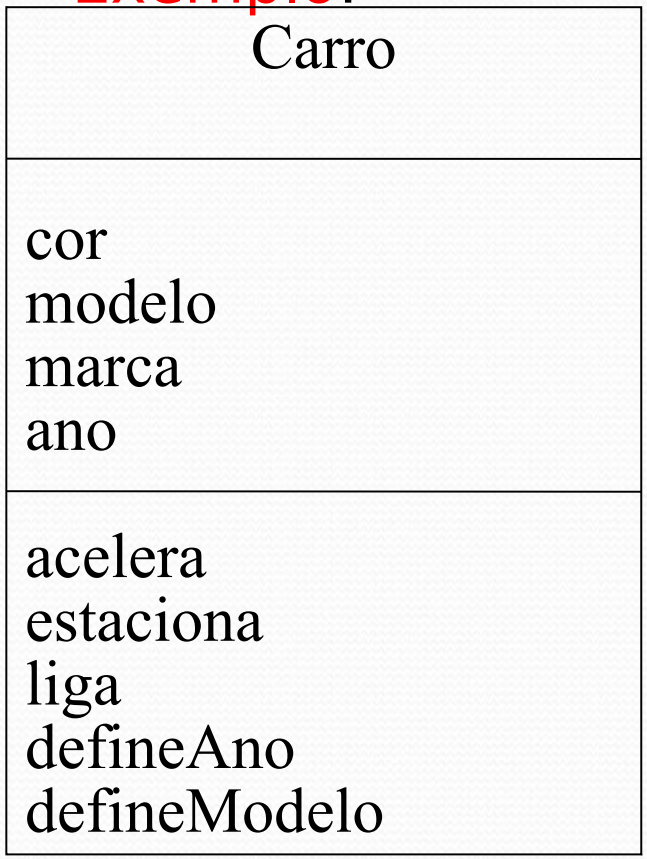
Encapsulamento e Ocultamento de Informações

- Utilização de encapsulamento para restringir a visibilidade externa de detalhes de informações (dados) ou implementações (operações), os quais são internos à estrutura de encapsulamento
- Não é possível chegar aos dados diretamente
- Cliente não tem conhecimento acerca de como as operações são implementadas



Encapsulamento e Ocultamento de Informações

- Exemplo:



- Classe Carro encapsula atributos e métodos referentes a um tipo abstrato
- Cada cliente (programa) que usar esta classe para definir objetos não saberá como os métodos foram implementados e não poderá alterar os atributos diretamente.
- Cliente pode usar somente os métodos que lhe serão permitidos.

Orientação a objetos

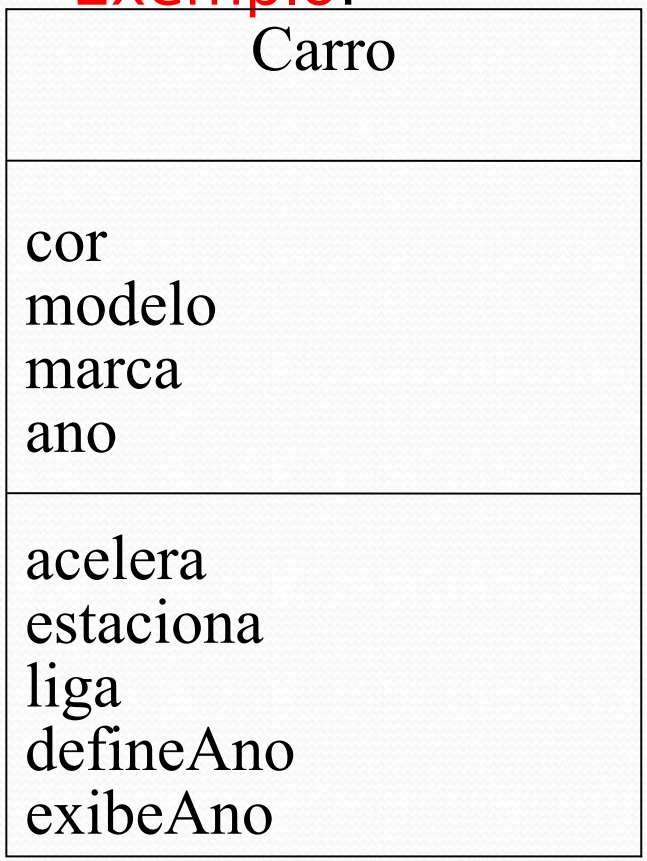
- Classes/Objetos
- Encapsulamento
- Ocultamento da informação e implementação (Abstração)
- Retenção de estado
- Identidade de objeto
- Mensagens
- Herança
- Polimorfismo/Vinculação tardia (*late binding*)
- Relacionamento entre classes/objetos

Retenção de Estado

- Habilidade de um objeto reter seu estado
  - estado ≅ conjunto de valores de seus atributos
- Um objeto é ciente de seu passado (operações que foram executadas previamente)
- O estado influencia o comportamento do objeto

Retenção de Estado

- Exemplo:



- Suponha que foram definidos dois objetos: ***bmw*** e ***corsa***

  Carro bmw = new Carro()

  Carro corsa = new Carro()

- se eu usar o método *defineAno* para *bmw* e também usá-lo para *corsa,* cada um dos objetos guardará o seu próprio ano.
  - bmw.defineAno(2008)
  - corsa.defineAno(2001)
  - bmw.exibeAno() mostrará 2008
  - corsa.exibeAno() mostrará 2001

Orientação a objetos

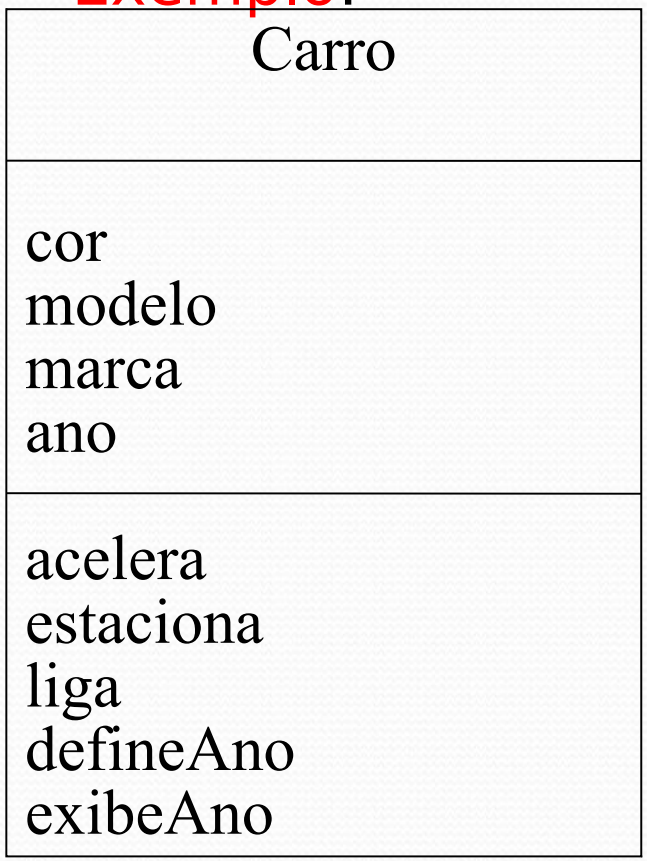
- Classes/Objetos
- Encapsulamento
- Ocultamento da informação e implementação (Abstração)
- Retenção de estado
- Identidade de objeto
- Mensagens
- Herança
- Polimorfismo/Vinculação tardia (*late binding*)
- Relacionamento entre classes/objetos

Identidade de Objeto

- Propriedade pela qual cada objeto (independentemente de sua classe ou seu estado) pode ser identificado e tratado como uma entidade distinta de software
  - O mesmo identificador permanece com o objeto por toda sua vida
  - Dois objetos nunca podem ter o mesmo identificador
  - Dois objetos podem ter o mesmo “estado”, porém suas identidades são distintas

Identidade de Objeto

- Exemplo:



- Suponha que foram definidos dois objetos distintos: ***bmw*** e ***corsa***
  - *bmw* será identificador do primeiro objeto durante sua existência
  - *corsa* será identificador do segundo objeto durante sua existência
- mesmo que aos atributos dos dois objetos sejam atribuídos os mesmos valores, eles serão objetos distintos

Orientação a objetos

- Classes/Objetos
- Encapsulamento
- Ocultamento da informação e implementação (Abstração)
- Retenção de estado
- Identidade de objeto
- Mensagens
- Herança
- Polimorfismo/Vinculação tardia (*late binding*)
- Relacionamento entre classes/objetos

Mensagens

- Um objeto pode se comunicar com outros através da troca de mensagens

- Uma mensagem é o veículo pelo qual um objeto remetente **obj1** transmite a um objeto destinatário **obj2** um pedido para o **obj2** aplicar um de seus métodos

- Mensagens ocorrem quando ocorre invocações (chamadas a métodos)

Mensagens

- Exemplo:

Estacionamento
quantidadeCarros
adicionaCarro()

- Suponha que o método adicionaCarro faça as seguintes ações:
  - instancia um carro

    Carro carro1 = new Carro()
  - define ano e modelo do carro instanciado

    carro1.defineAno(2001)

    carro1.defineModelo("ferrari")

- Ao invocar os métodos ***defineAno*** e ***defineModelo,*** está ocorrendo uma troca de mensagens entre um objeto da Classe **Estacionamento** e um objeto da Classe **Carro**

Orientação a objetos

- Classes/Objetos
- Encapsulamento
- Ocultamento da informação e implementação (Abstração)
- Retenção de estado
- Identidade de objeto
- Mensagens
- Herança
- Polimorfismo/Vinculação tardia (*late binding*)
- Relacionamento entre classes/objetos

Herança

- Qual é o conceito de herança no nosso dia-a-dia ???

Herança

- O que torna a computação orientada a objetos única é o conceito de herança

- Mecanismo que permite definir uma nova classe (subclasse) a partir de uma classe já existente (superclasse)

Herança

- A subclasse herda as características da superclasse:
  - os atributos e os métodos da superclasse passam a ser também atributos e métodos da subclasse
  - a subclasse pode adicionar novos atributos e métodos, e reescrever métodos herdados

- Portanto: herança é a habilidade de um objeto derivar seus atributos (dados) e métodos (funcionalidade) automaticamente de outro objeto

Herança

- Permite modelar uma hierarquia entre classes: classes mais especializadas (subclasses) herdam propriedades da classe mais geral (superclasse)

- Cria uma nova classe inserindo somente as diferenças desta para sua superclasse

- Identifica-se a possibilidade de herança por meio da seguinte expressão típica: “é um tipo de”

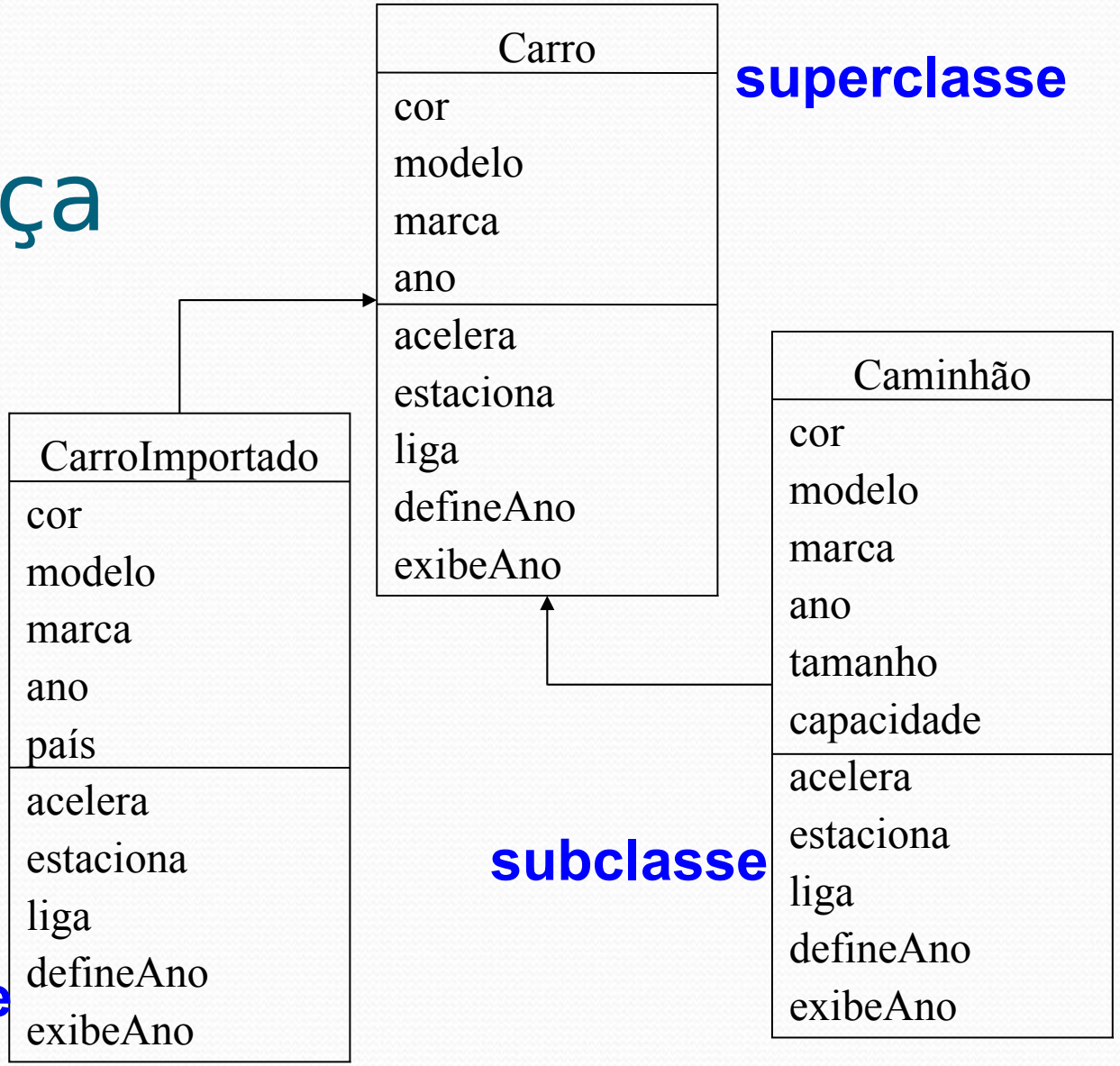
Herança

- Herança: mecanismo para derivar novas classes a partir das classes já existentes
  - Exemplo: Classe Caminhão é derivada da Classe Carro
  - Caminhão tem os atributos tamanho e capacidade, além dos atributos herdados da Classe Carro

- Uma classe derivada herda a representação dos atributos e operações públicas da classe base, podendo:
  - adicionar novas operações
  - estender a representação dos atributos
  - sobrepor a implementação de operações já herdadas

Herança

Diagrama de Classes UML (Herança)



Herança

- Construção de forma incrementada de software: (Reutilização de Software)

- Boa prática:
  - Primeiro, construir classes para lidar com o caso mais geral
  - Em seguida, a fim de tratar os casos especiais, definir classes especializadas - herdadas da primeira classe

Herança

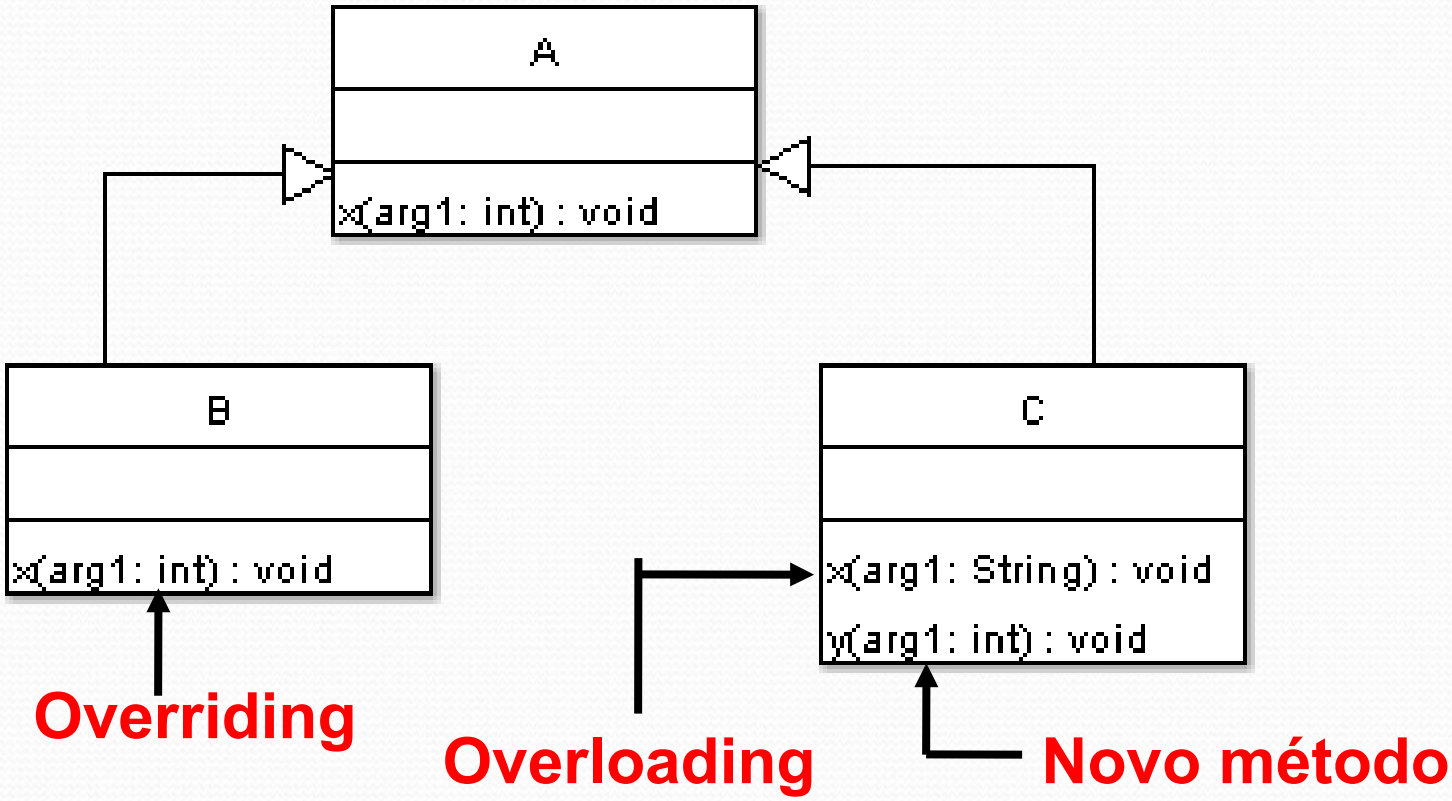
- Capacidade de Substituição
  - Deve ser capaz de substituir uma classe **Derivada** dentro de qualquer programa que exija uma classe **Base** e tudo deve funcionar bem
  - Basicamente, isso significa que, se você escrever um programa supondo que tem um **Carro**, então pode usar livremente qualquer classe herdada de **Carro (Caminhão, CarroImportado)**.

Sobrecarga *(overloading)*
Redefinição *(overriding)*

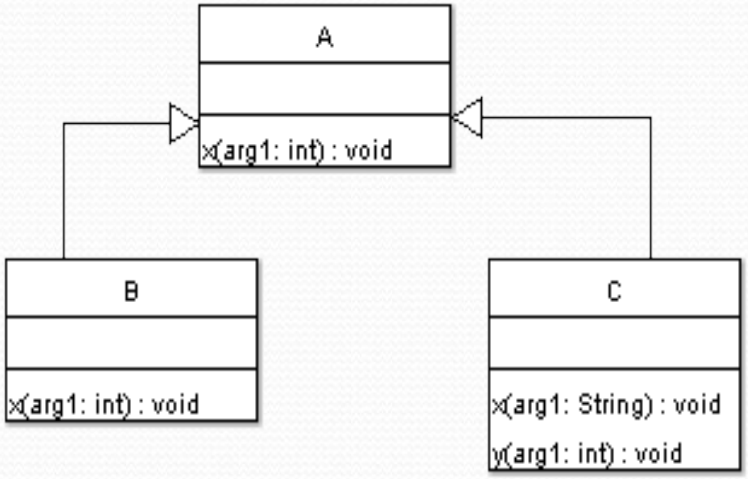
- **Sobrecarga :** dar um mesmo nome a mais de um método
  - Desde que a assinatura seja **diferente**
  - **assinatura ↔ parâmetros + retorno**

- A redefinição de um método em classes diferentes, dentro de uma estrutura de herança é conhecida como ***overriding***
  - Necessita ter a mesma **assinatura**

Overloading & Overriding



Overloading



A a; // define objeto do tipo A

...

if usuário diz **OK**

then a = new B(); // Capacidade de Substituição

else a = new C(); // Capacidade de Substituição

...

a.x(10); // OK.

a.x("Aula"); // **Erro de compilação**: x(String) não está
// definida na classe A

Orientação a objetos

- Classes/Objetos
- Encapsulamento
- Ocultamento da informação e implementação (Abstração)
- Retenção de estado
- Identidade de objeto
- Mensagens
- Herança
- Polimorfismo/Vinculação tardia (*late binding*)
- Relacionamento entre classes/objetos

Polimorfismo

- “O que possui várias formas”
- Propriedade de se usar o mesmo nome para métodos diferentes, implementados em diferentes níveis de uma hierarquia de classes
- Para cada classe, tem-se um comportamento específico para o método

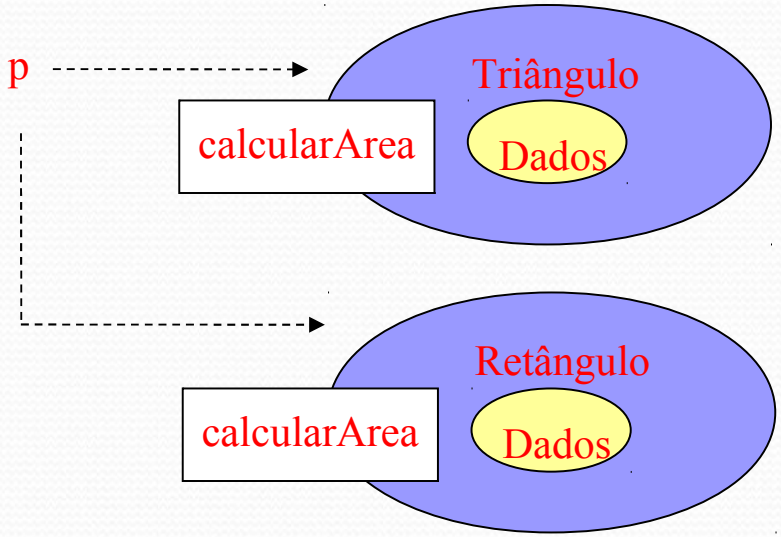
Polimorfismo / Vinculação Tardia

- Habilidade pela qual uma única operação pode ser definida em mais de uma classe e assumir implementações diferentes em cada uma dessas classes
  - *Overriding* de operações

- Vinculação tardia (*late binding*) é a técnica pela qual a operação a ser executada é determinada somente em tempo de execução
  - Java (implementado diretamente)
  - Palavra chave **virtual** (Linguagem C++)

Polimorfismo / Vinculação Tardia

```
Polígono p;
...
if usuário diz OK
then p = new Triângulo();
else p = new Retângulo();
...
p.calcularArea(); // aqui p pode referir-se a um objeto
                  // Triângulo ou a um objeto Retângulo
                  // Capacidade de Substituição
```



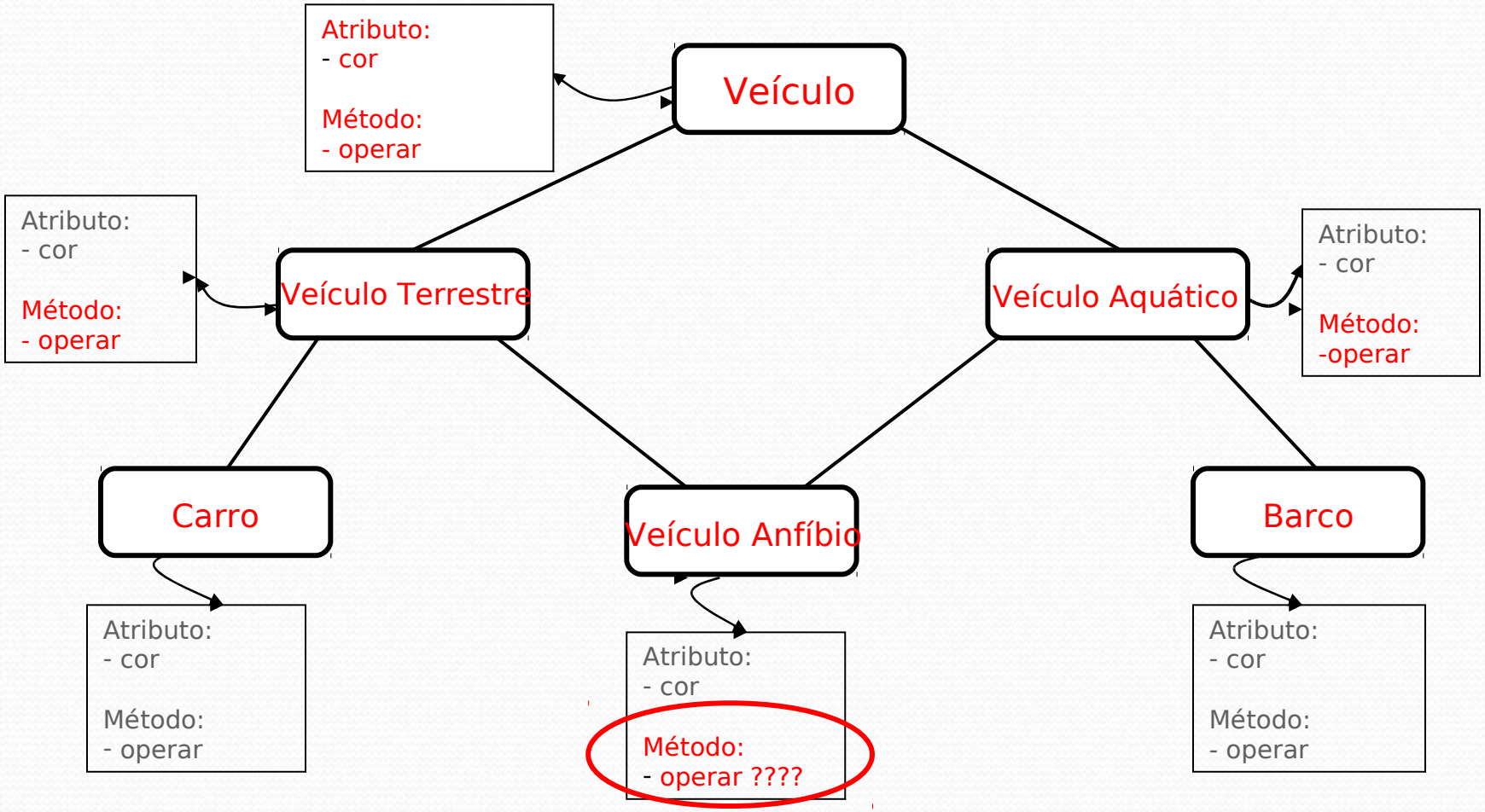
Polimorfismo

- Permite a cada objeto responder a um formato de mensagem da maneira apropriada à classe (ou subclasse) da qual foi instanciado
- Uma mesma operação pode apresentar comportamentos diferentes em classes (ou subclasses) distintas
- Uma operação pode ter diferentes implementações, isto é, mais de um método pode implementá-la

Herança Múltipla

- Uma classe pode herdar características de mais de uma classe, ou seja, pode ter mais de uma superclasse
  - a subclasse herda todos os atributos e métodos de todas as suas superclasses
  - atributos/métodos de um mesmo ancestral que “alcancem” a subclasse por mais de um caminho na hierarquia são herdados apenas uma vez (são o mesmo atributo/método). Ex. a seguir: atributo cor na hierarquia de Veículo
  - conflitos em definições paralelas na hierarquia podem gerar ambiguidades (ex. a seguir: método operar na hierarquia de Veículo)

Herança Múltipla

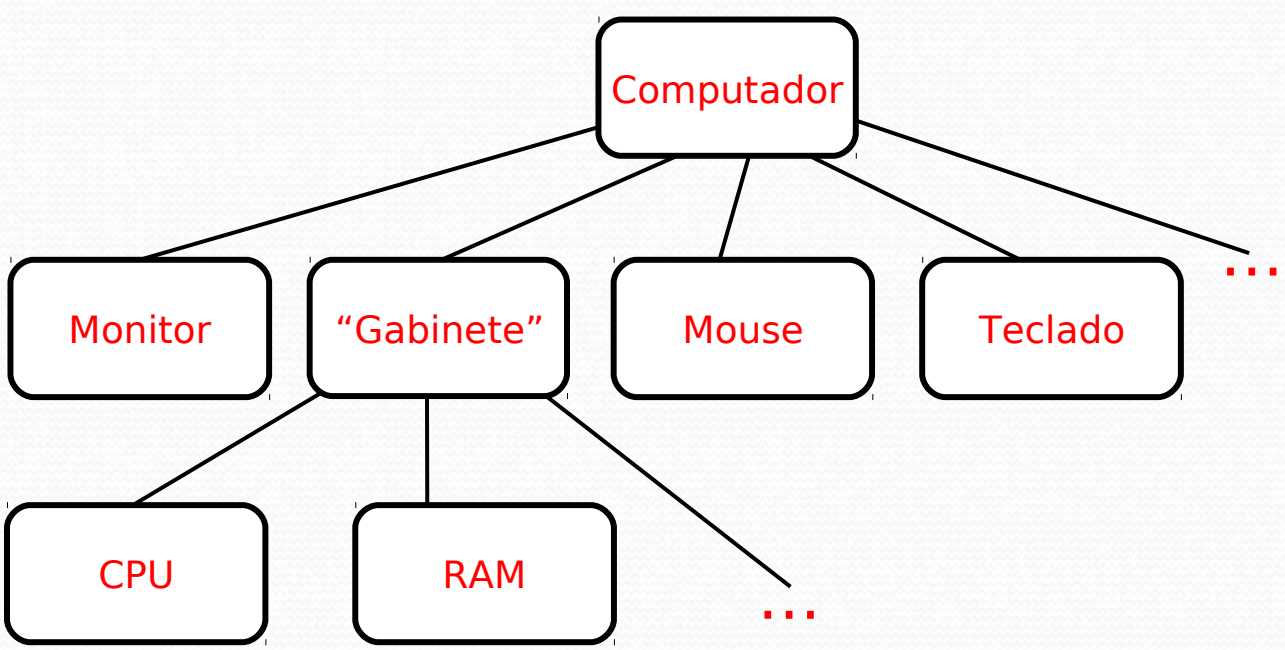


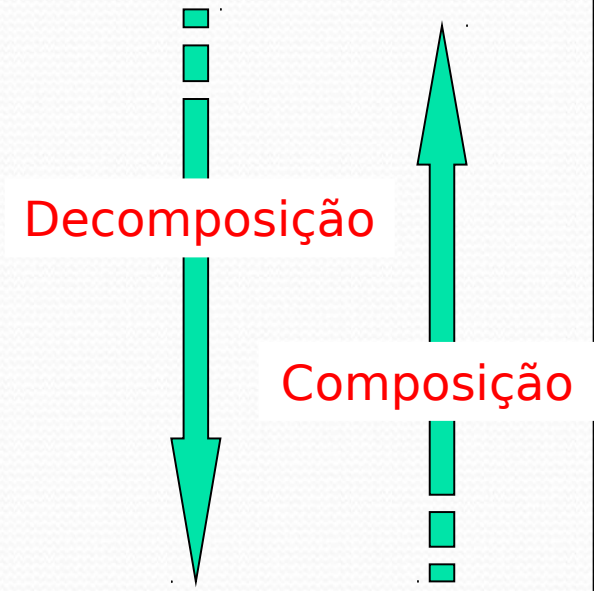
Orientação a objetos

- Encapsulamento
- Ocultamento da informação e implementação (Abstração)
- Retenção de estado
- Identidade de objeto
- Mensagens
- Classes/Objetos
- Herança
- Polimorfismo/Vinculação tardia (*late binding*)
- Relacionamento entre classes/objetos

Agregação e Composição

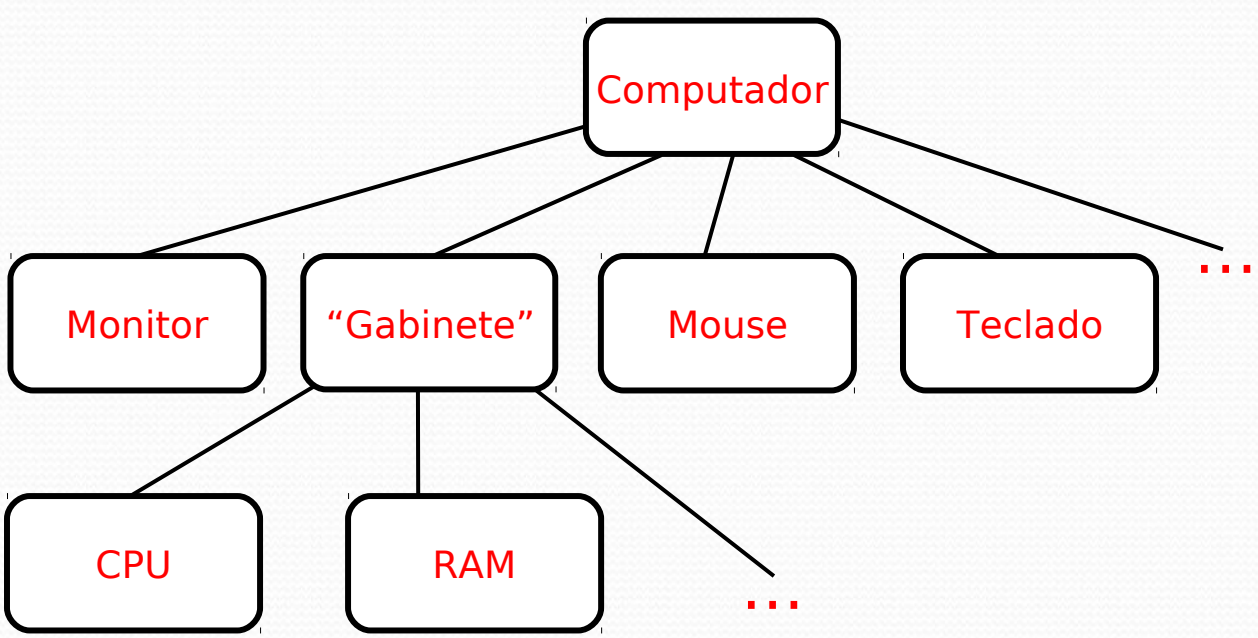
- Uma classe pode ser <u>composta</u> por outras classes, consideradas <u>partes</u> ou <u>componentes</u>

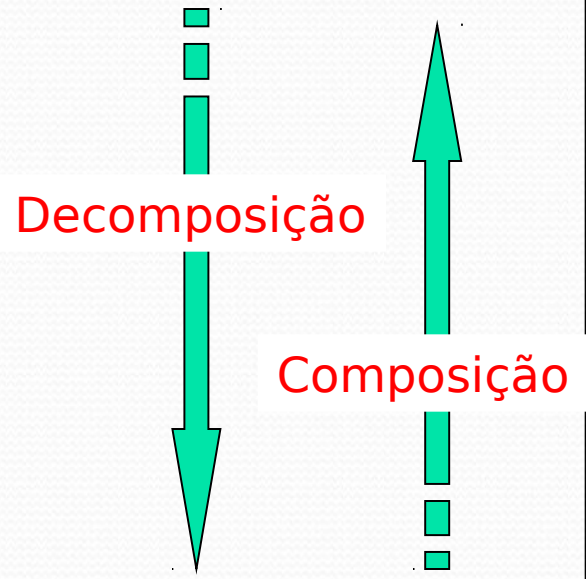




Agregação e Composição

- A composição é frequentemente referida por relacionamento “todo-parte”, no qual o agregado (“todo”) é composto de partes





Relacionamento: Agregação

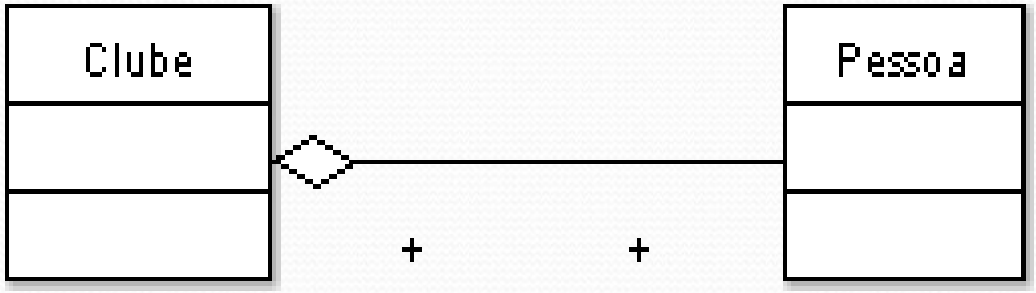
- Capturam relacionamentos do tipo “todo-parte” entre objetos
- Não existe herança entre objetos participando de uma agregação
- Agregações reduzem a complexidade ao permitir tratar vários objetos como um único

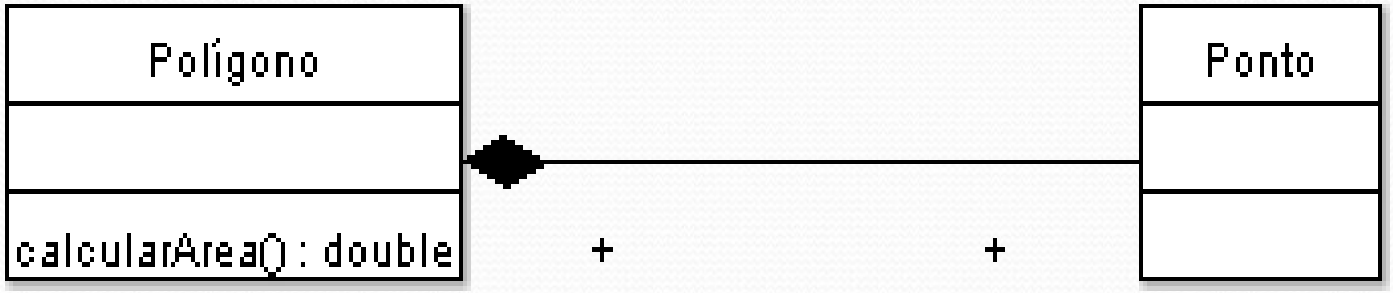
Relacionamento: Composição

- Nesse tipo de agregação, os objetos da classe “parte” não podem viver quando o objeto “todo” é destruído

- Não compartilhamento: os objetos da classe “parte” são componentes de apenas um objeto “todo”

Agregação X Composição





Diagramas de Classes UML (Agregação e Composição)

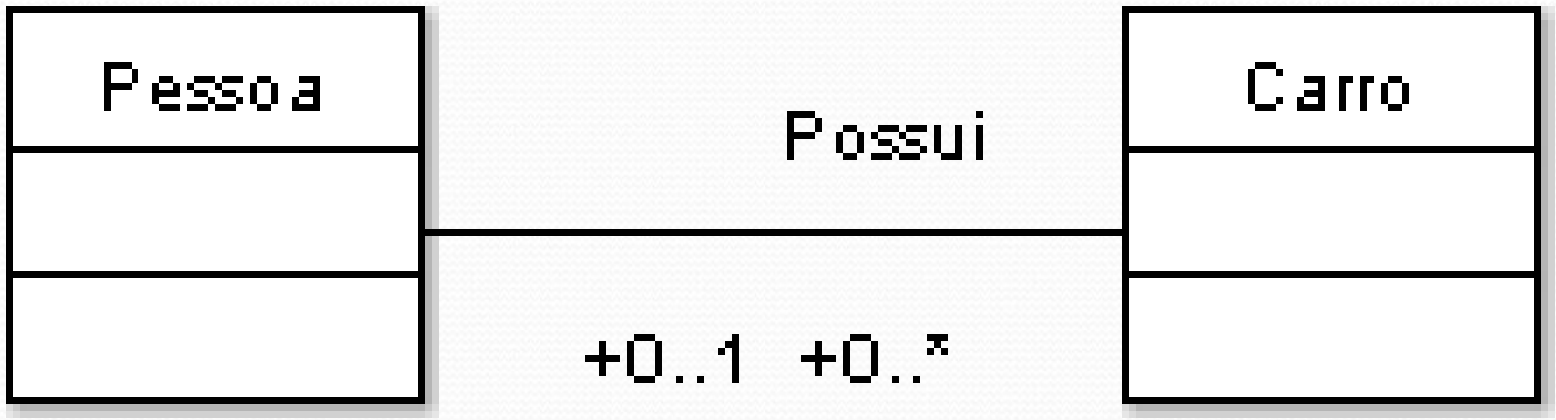
Herança e Agregação/Composição

- Herança é uma relação “é um”
  - Carro **é um** Veículo

- Agregação/Composição é uma relação “é parte de um”
  - Pneu **é parte de um** Carro
  - Motor **é parte de um** Carro

Associação

- Uma associação nos permite capturar relacionamentos básicos que existem entre conjuntos de objetos



Diagramas de Classes UML (Associação)

ACH2002

Orientação a Objetos

Professores:
Delano Medeiros Beder
Fátima L. S. Nunes

EACH – USP