

Unified Modeling Language (UML)

Luiz Eduardo Virgilio da Silva
ICMC, USP

Material baseado nos slides dos professores:
José Fernando Rodrigues Jr (ICMC/USP)



Sumário

- Histórico da UML
- Diagrama de classes
- Representação de classes
 - Atributos e métodos
 - Tipos de acesso e modificadores
- Relacionamentos entre classes
 - Herança, Implementação, Associação, Agregação e Composição

- UML (Linguagem de Modelagem Unificada) é uma linguagem visual
 - **Análise e projeto** de sistemas computacionais no paradigma de **Orientação a Objetos**
- Nos últimos anos, a UML se tornou a linguagem padrão de projeto de software, adotada internacionalmente pela indústria de Engenharia de Software

UML

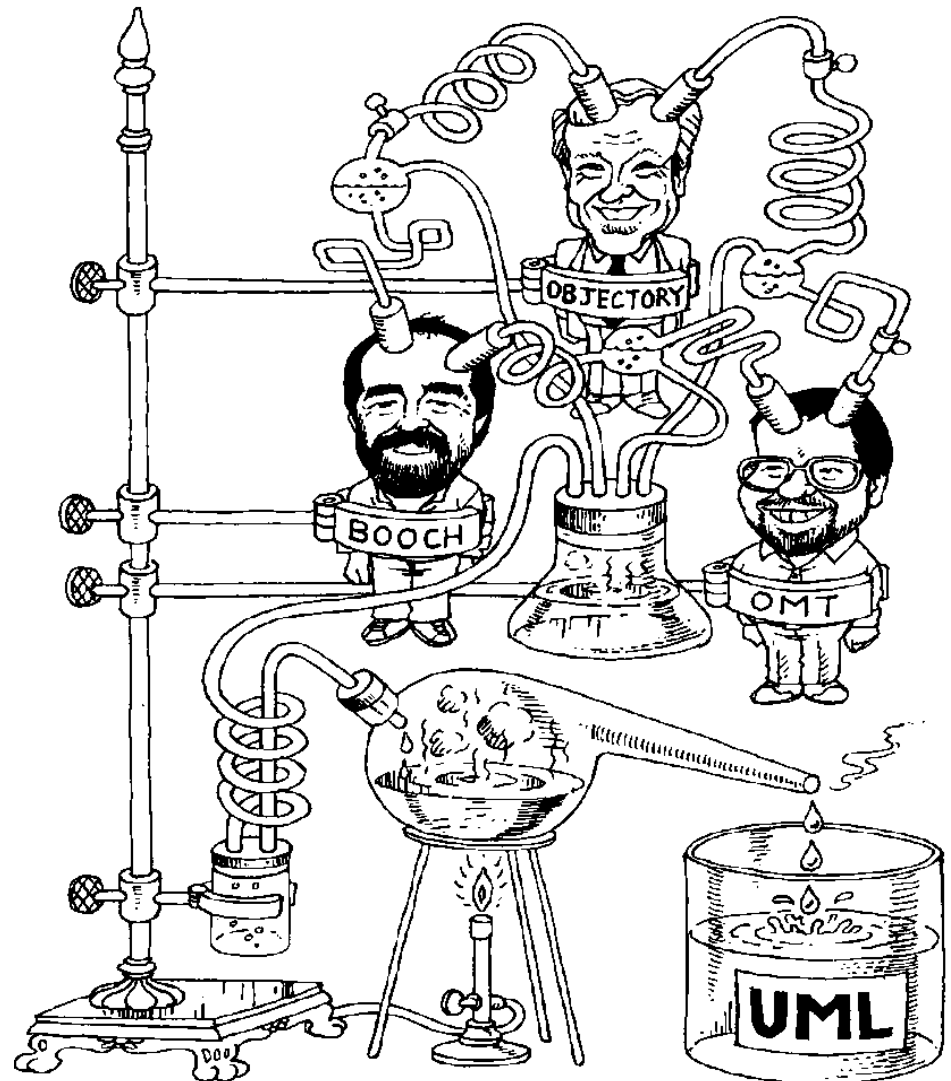
- UML não é uma linguagem de programação
- É uma linguagem de modelagem, utilizada para representar o sistema de software sob os seguintes aspectos:
 - Requisitos
 - Comportamento
 - Estrutura lógica
 - Dinâmica de processos
 - Comunicação/Interface com os usuários

- Por que modelar um sistema?
 - Um sistema computacional é, de modo geral, excessivamente complexo
 - Necessário decompô-lo em pedaços compreensíveis
 - Criação de **diagramas** auxiliam no entendimento do problema
 - **Linguagem única** que permite a todos os desenvolvedores entender quais objetos fazem parte do sistema e como eles se comunicam

- A UML surgiu da união de outras três linguagens de modelagem:
 - O método de Booch (**Grady Booch**, **Rational Software Corporation**)
 - O método OMT (Object Modeling Technique, **Ivar Jacobson**, **Objectory**)
 - Método OOSE (Object-Oriented Software Engineering, **James Rumbaugh**, **General Eletrics**)
- Até meados da década de 90, estas eram as três linguagens de modelagem mais populares entre os profissionais de ES.

UML

- Em meados da década de 90, os criadores destas três linguagens se reuniram para criar uma **linguagem unificada**, mais concreta e madura



Objetivo da UML

- O objetivo da UML é fornecer múltiplas visões do sistema que se deseja modelar
- Estas várias visões são representadas pelos diferentes diagramas UML
- Cada diagrama analisa o sistema sob um determinado aspecto
 - É possível ter enfoques mais amplos (externos) ou mais específicos (internos)

Vantagens e Desvantagens da UML

- Perdas

- Maior trabalho na modelagem
 - Mais tempo gasto



- Ganhos

- Menos trabalho na construção (implementação)
 - A solução está pronta
 - Menos tempo gasto
- Os problemas são encontrados em tempo hábil para sua solução
- As dúvidas são sanadas mais cedo e são levantadas em sua totalidade

Diagramas da UML

- **Diagrama de Classes**

- Diagrama de Objetos

- Diagrama de Componentes

- Diagrama de Casos de usos

- Diagrama de Sequências

- Diagrama de Colaboração

- Diagrama de Estado

- Diagrama de Atividades

Diagramas de
Estruturas

Diagramas de
Comportamento

Diagrama de classes

- O diagrama de classes é um dos mais importantes e mais utilizados da UML
- Representação das principais classes
 - Atributos e Métodos
- Relacionamento entre as classes
- Uma visão **estática** do sistema

Diagrama de classes

- Na UML, uma classe possui a notação de um retângulo dividido em três partes
 - Nome da classe
 - Atributos da classe
 - Métodos da classe

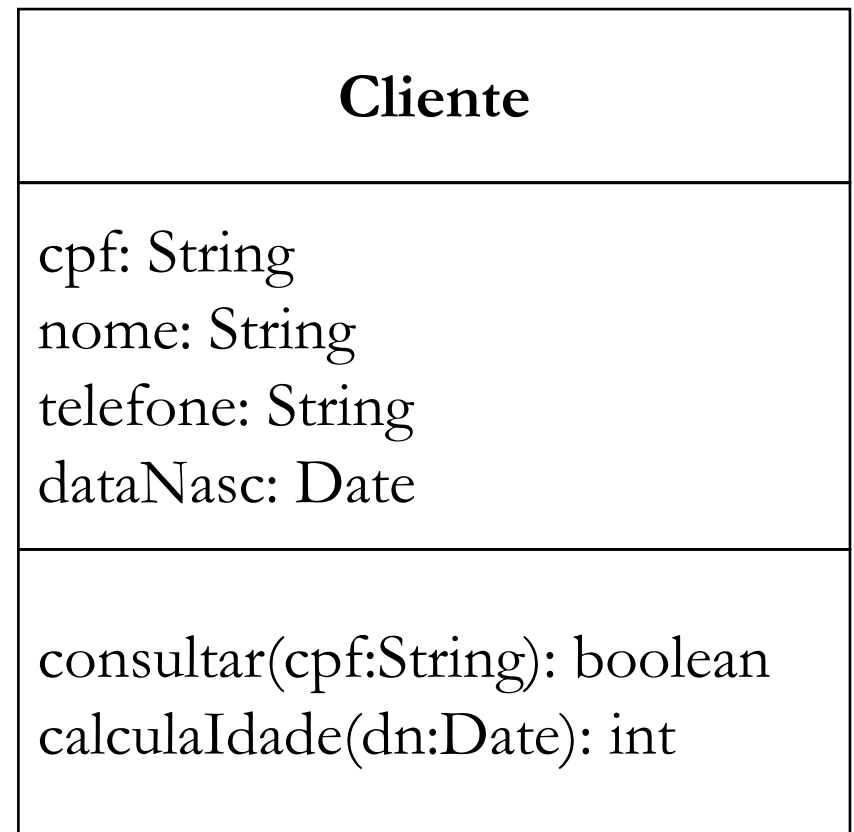


Diagrama de classes

- Representação de atributos

visibilidade nome : tipo = valor inicial {propriedades}

- **Visibilidade:** public (+), private (-), protected (#)
- **Tipo do atributo:** int, double, String, Date, ...
- **Valor inicial:** definido no momento da criação do objeto
- **Propriedades:** final, static, ...
- Exemplos:
 - nomeFunc:String = null
 - + PI:double = 3.1415 {final}

Diagrama de classes

- Representação de métodos

visibilidade nome(tipo) : tipo {propriedades}

- **Visibilidade:** public (+), private (-), protected (#)
- **Tipo do atributo:** int, double, String, Date, ...
- **Tipo de retorno:** int, double, String, Date, ...
- **Propriedades:** final, abstract, ...

- Exemplos:

```
+ getName() : String {abstract}  
+ calcArea(Shape) : double  
+ pow(double, double) : double {final}
```

Relacionamento Entre Classes

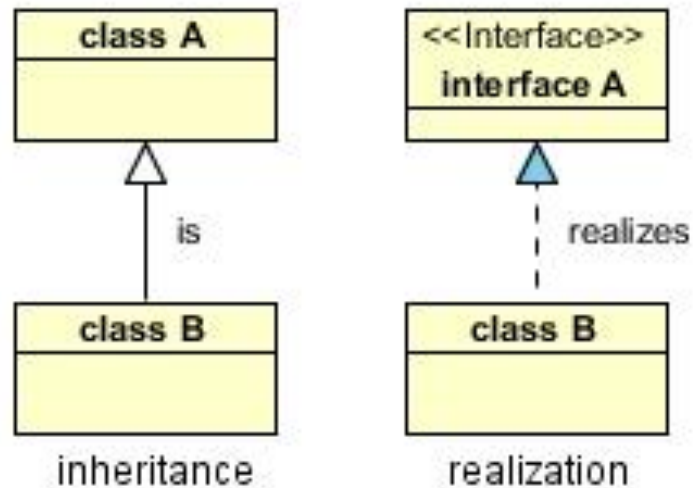
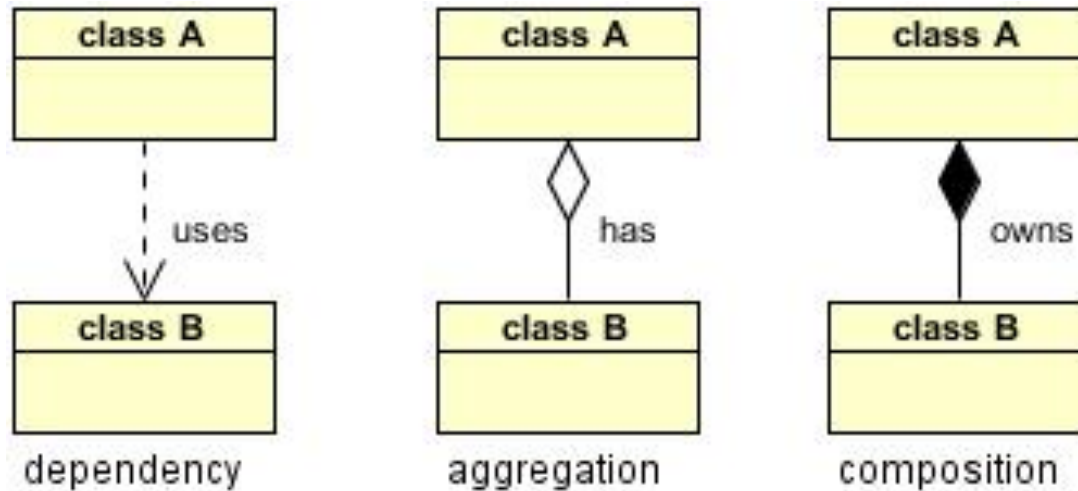
Relacionamento entre classes

- Em UML é possível representar o relacionamento entre as classes
- Vamos abordar as principais representações
 - Tipos de conexões
 - É uma parte do diagrama de classes

Relacionamento entre classes

- Generalização (herança)
 - “é um”
- Implementação (realização)
 - Aplicada para interfaces
- Associação (dependência)
 - “usa”
- Agregação
 - “é parte de” (possui)
 - Objeto ainda faz sentido mesmo sem a existência da agregação
- Composição
 - “é parte essencial de” (é dono de)
 - Objeto não faz sentido sem a composição

Relacionamiento entre classes

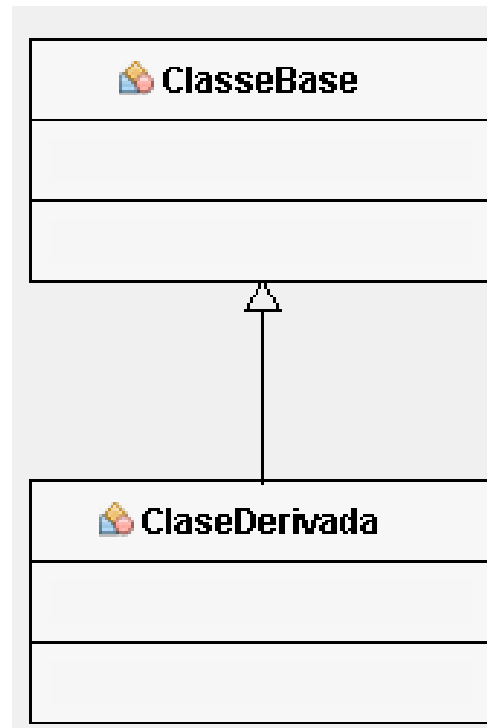


Generalização

- Representa relacionamentos do tipo “é um”
 - Herança
 - Ex: um cachorro é um mamífero
- Generalização/especialização
 - A partir de duas ou mais classes, abstrai-se uma classe mais genérica
 - De uma classe geral, deriva-se uma mais específica
 - Sub-classes possuem todas as propriedades das superclasses
 - Deve existir pelo menos uma propriedade que distingue duas classes especializadas
 - Caso contrário, não há necessidade

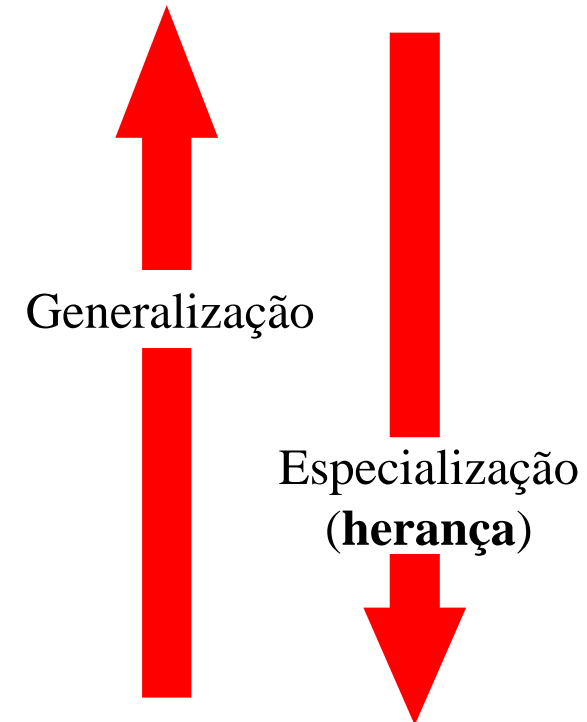
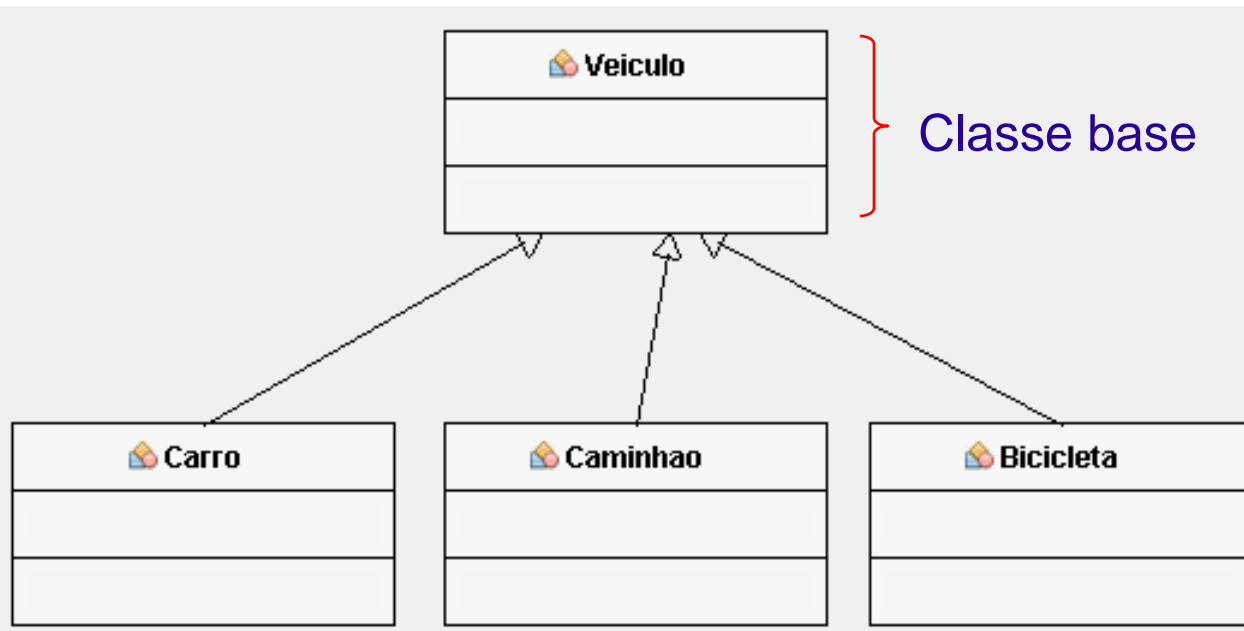
Generalização

- No diagrama de classes
 - A generalização é representada com uma seta do lado da classe mais geral (classe base)



Generalização

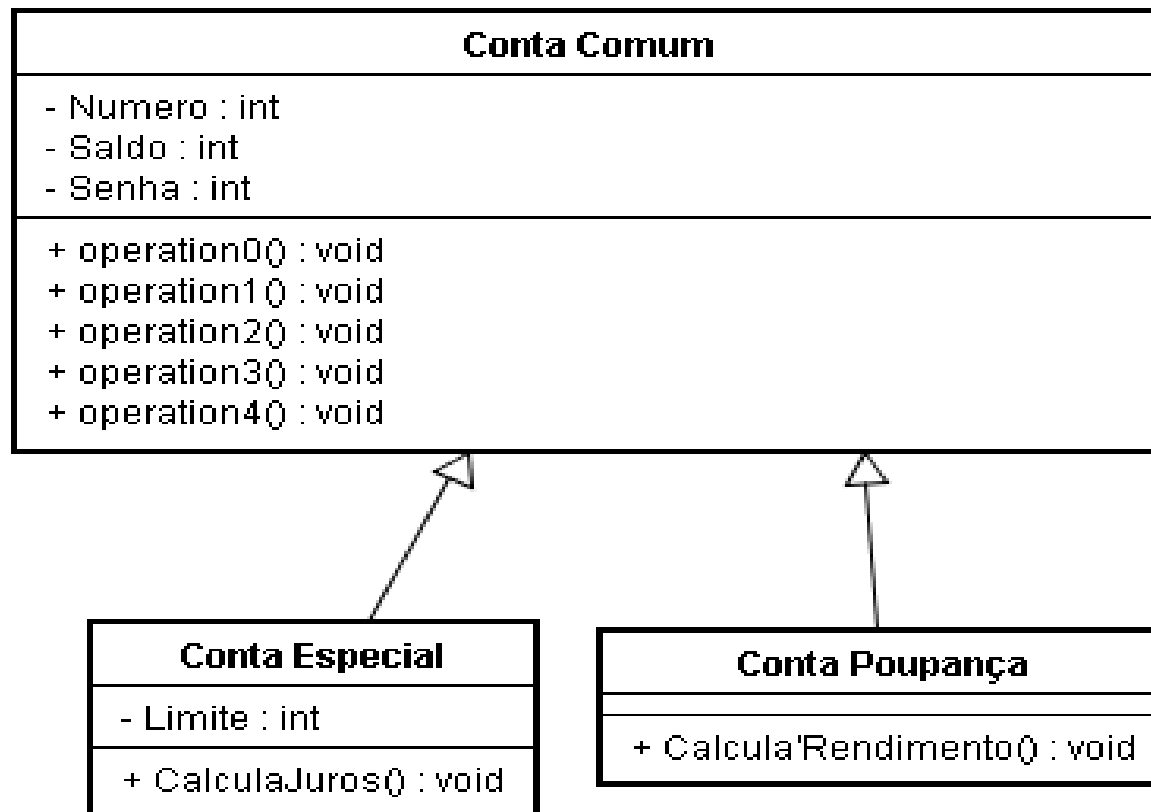
- Exemplo



Classes derivadas
Especializações

Generalização

- Exemplo

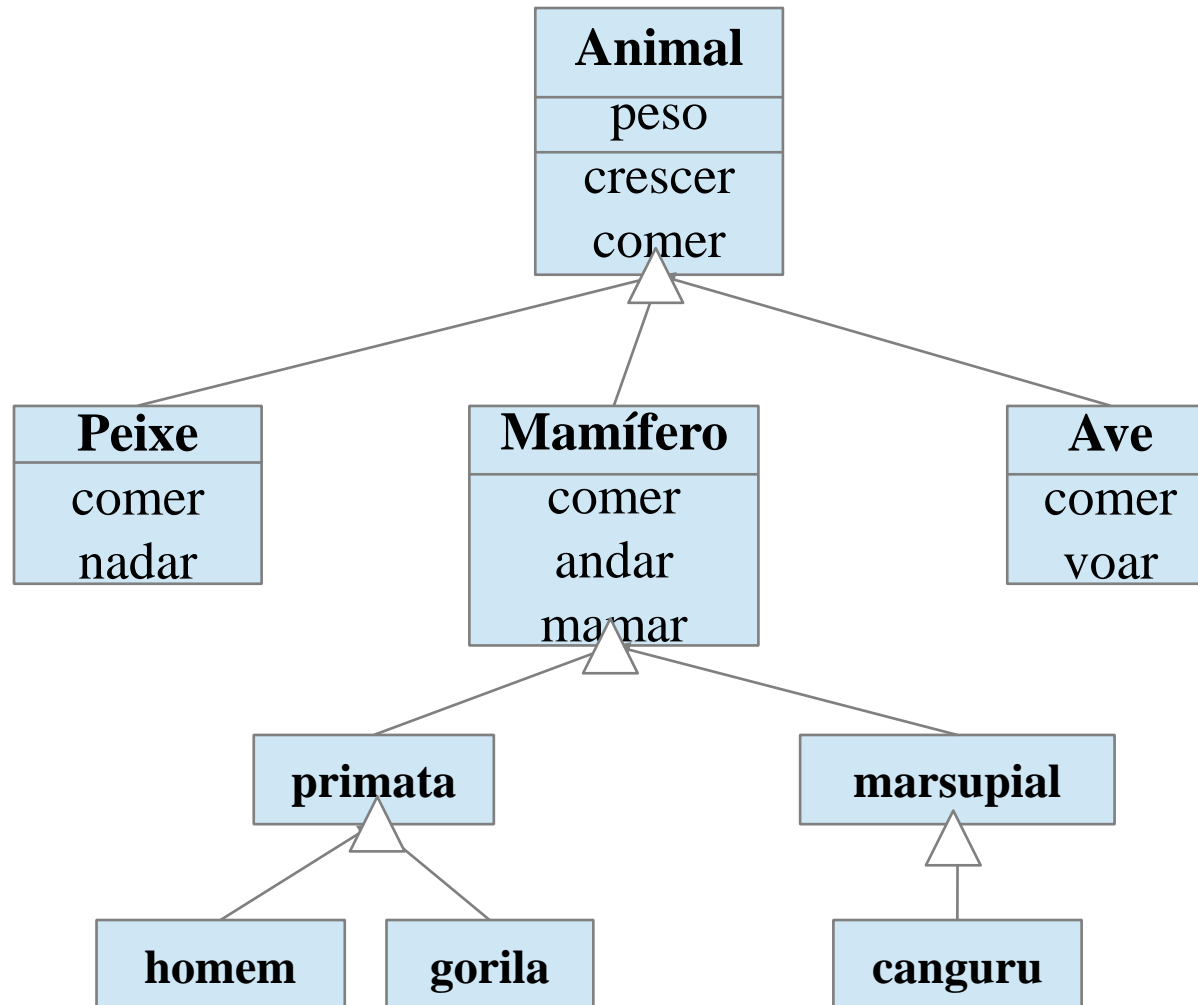


Generalização

- Permite organizar as classes hierarquicamente
- Técnica de reutilização de software
 - Novas classes são criadas a partir de classes existentes, absorvendo seus atributos e comportamentos (métodos)
 - Recebe novos recursos posteriormente

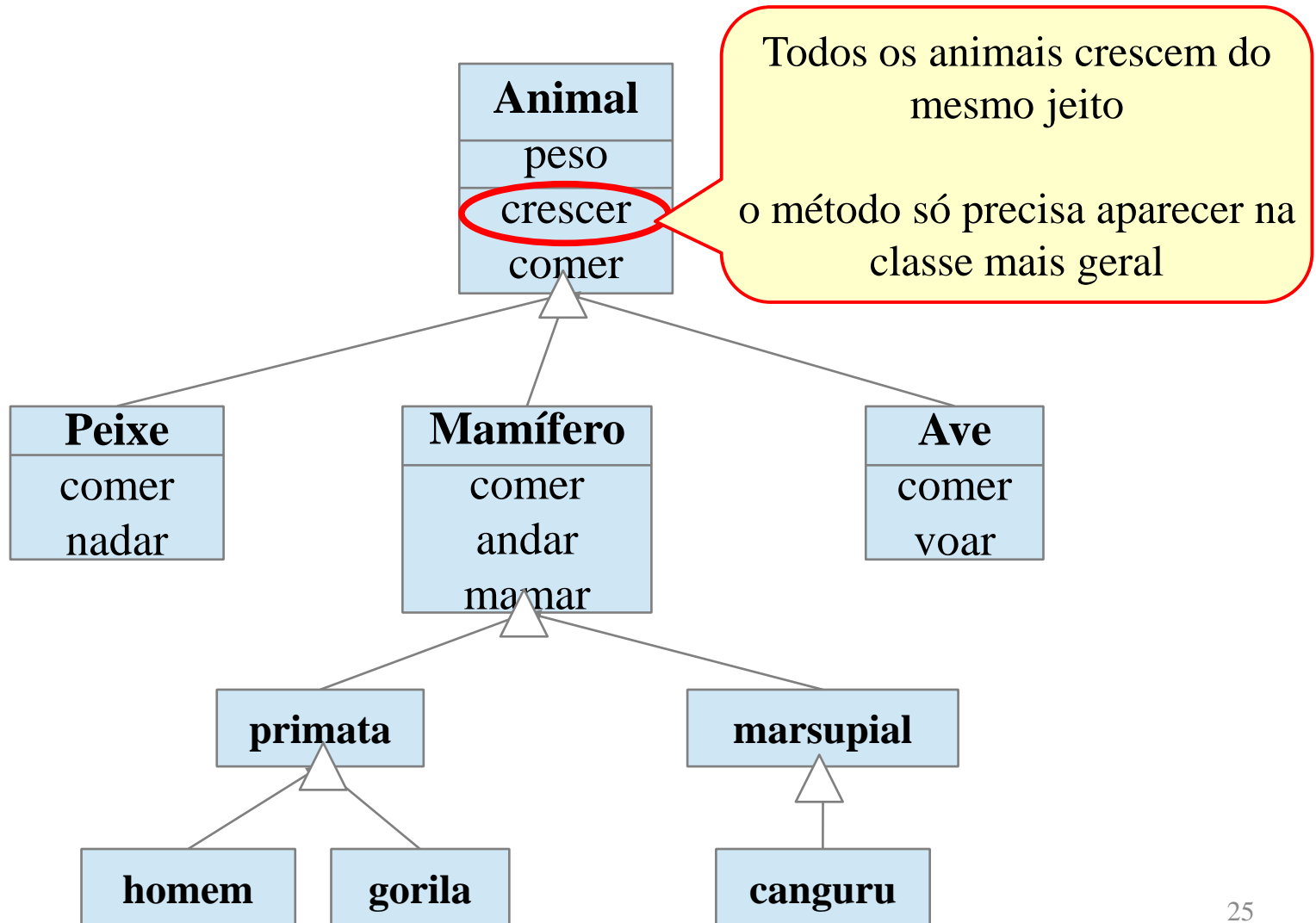
Generalização

- Exemplo de hierarquia de classes



Generalização

- Exemplo de hierarquia de classes

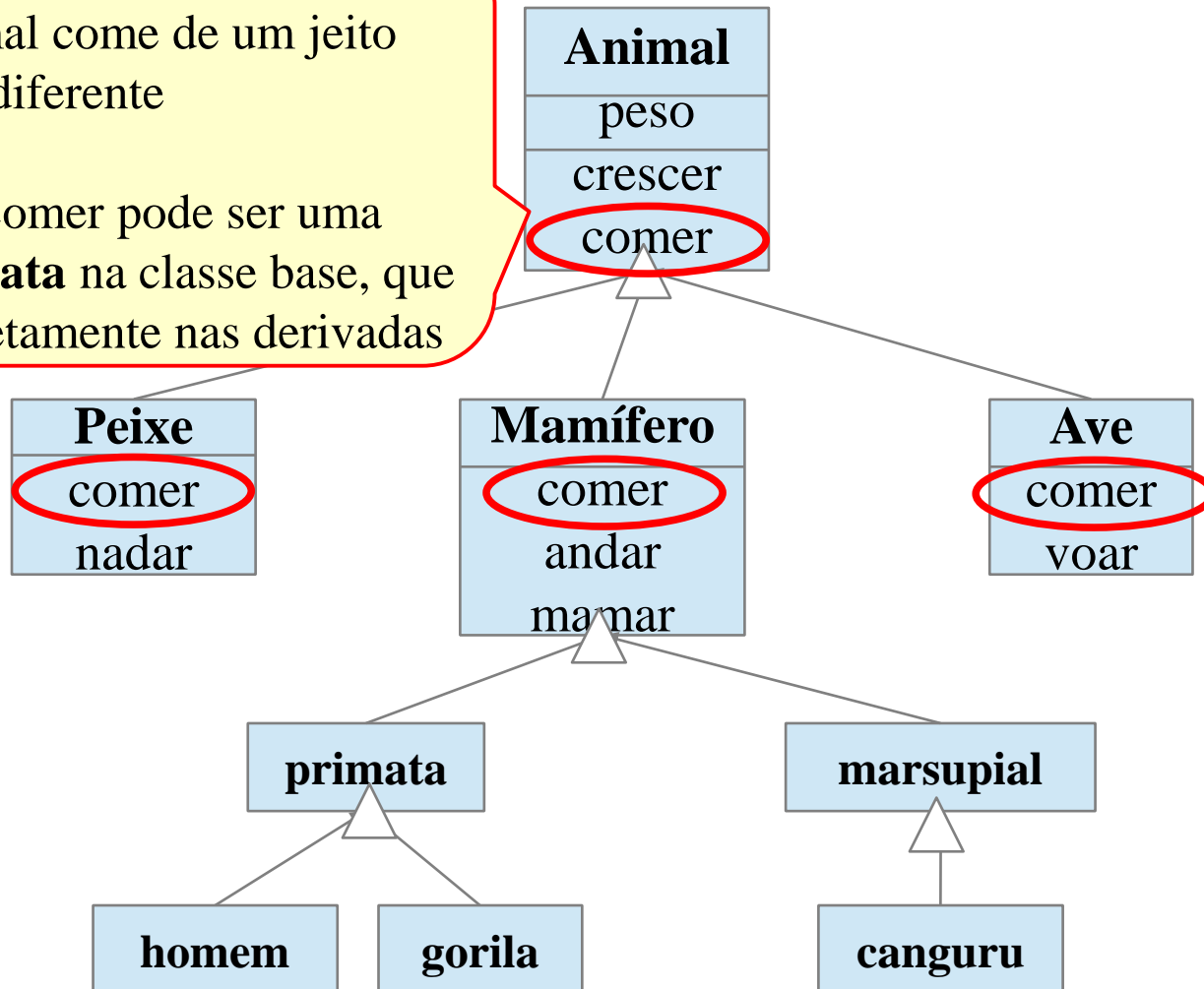


Generalização

- Exemplo de hierarquia de classes

Todos os animais comem, mas cada tipo de animal come de um jeito diferente

O método comer pode ser uma operação **abstrata** na classe base, que aparece concretamente nas derivadas

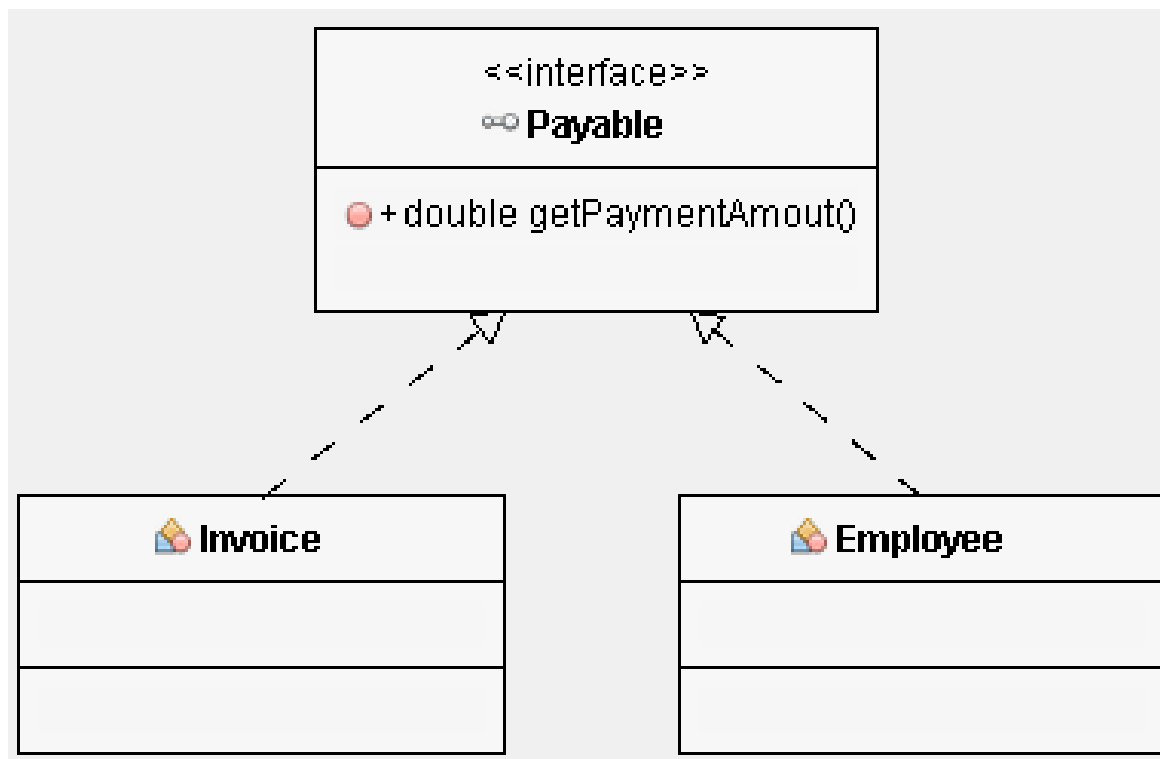


Implementação

- Interfaces estabelecem um contrato entre os objetos
 - Definição dos métodos pertencentes àquele contrato
- Interfaces não podem ser instanciadas
 - Não são classes comuns
- Em classes, podemos usar **herança**
- Em interfaces, utiliza-se a **implementação**

Implementação

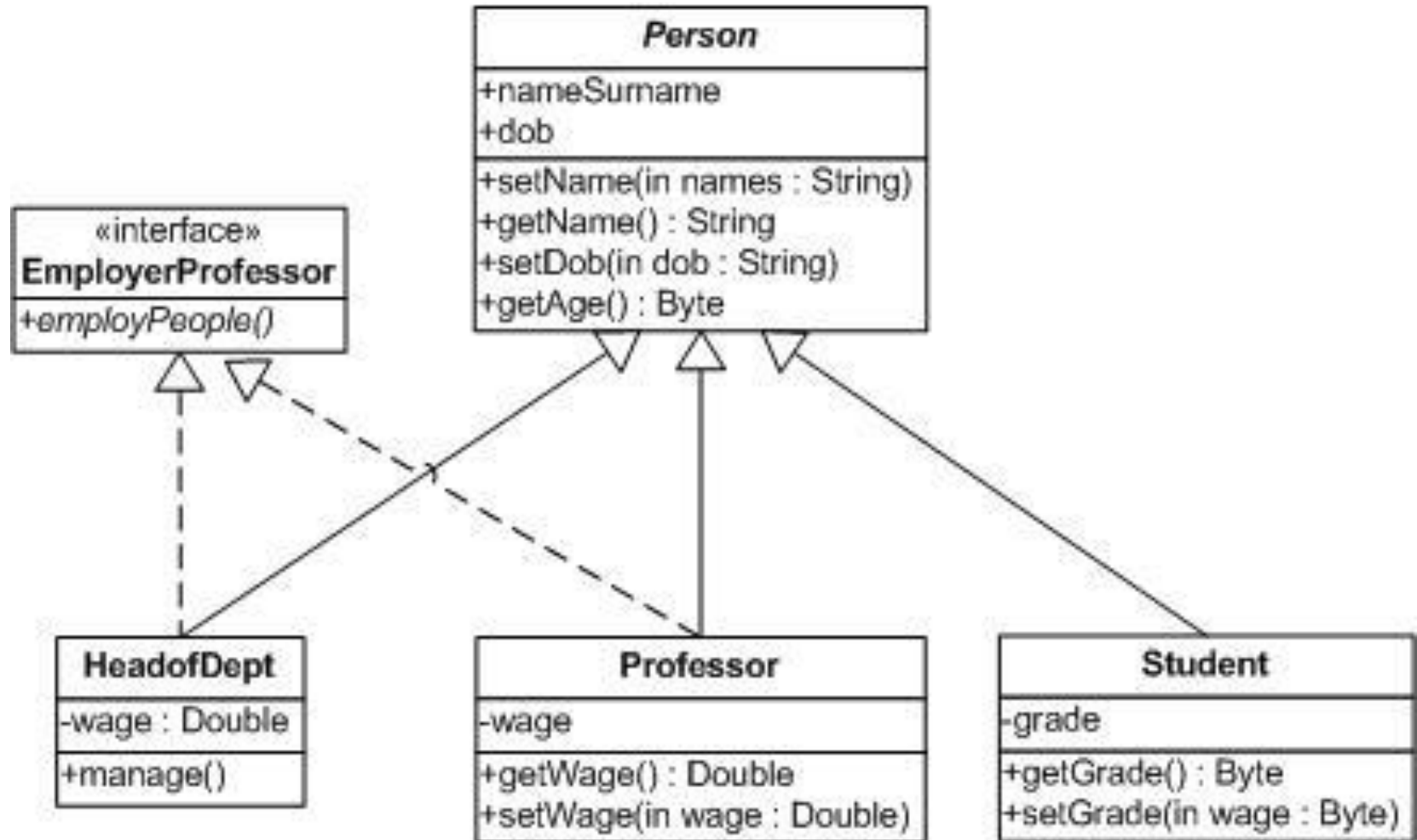
- Em UML, interfaces são definidas de forma similar às classes
 - Diferenciadas com uma marcação de *interface*
 - Implementação é parecida com a herança



Implementação

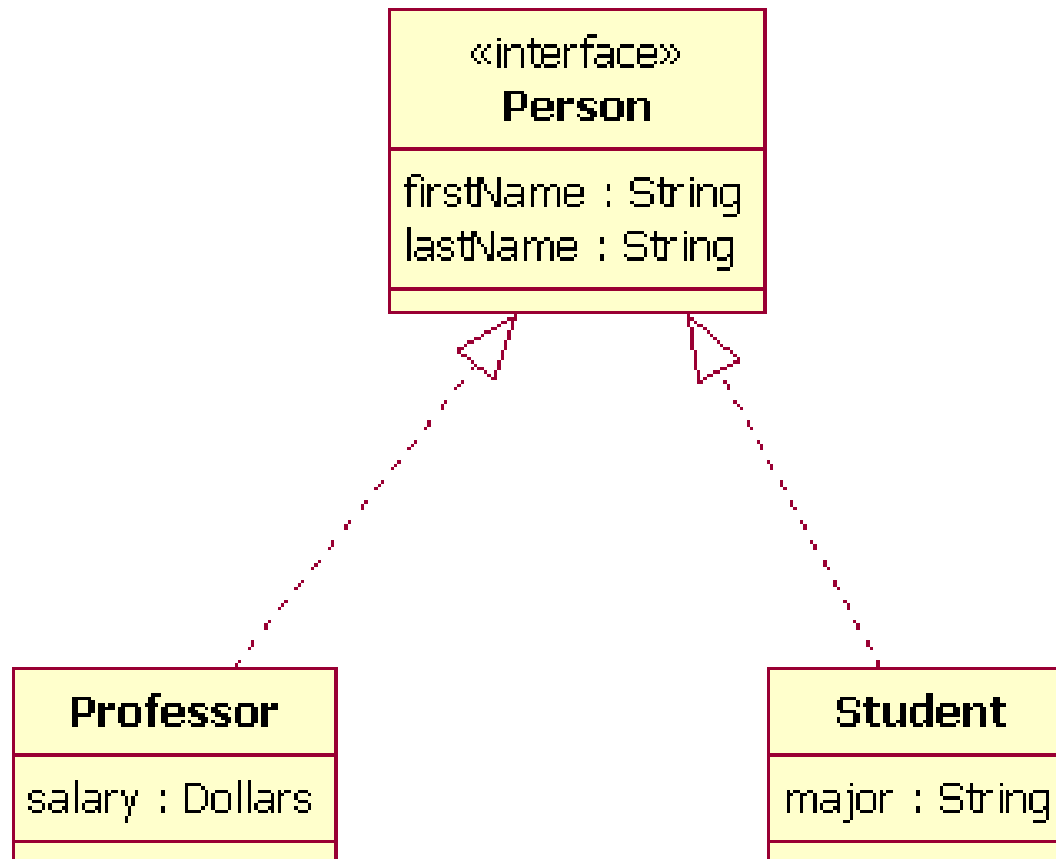
- Quando uma classe **herda** outra classe, a implementação dos métodos é herdada
- Quando uma classe **implementa** uma interface, os métodos definidos na interface precisam ser implementados
 - Em geral, não há implementação em uma interface, só definição
 - Todos os métodos da interface precisam necessariamente ser escritos pela classe que implementa a interface

Implementação



Implementação

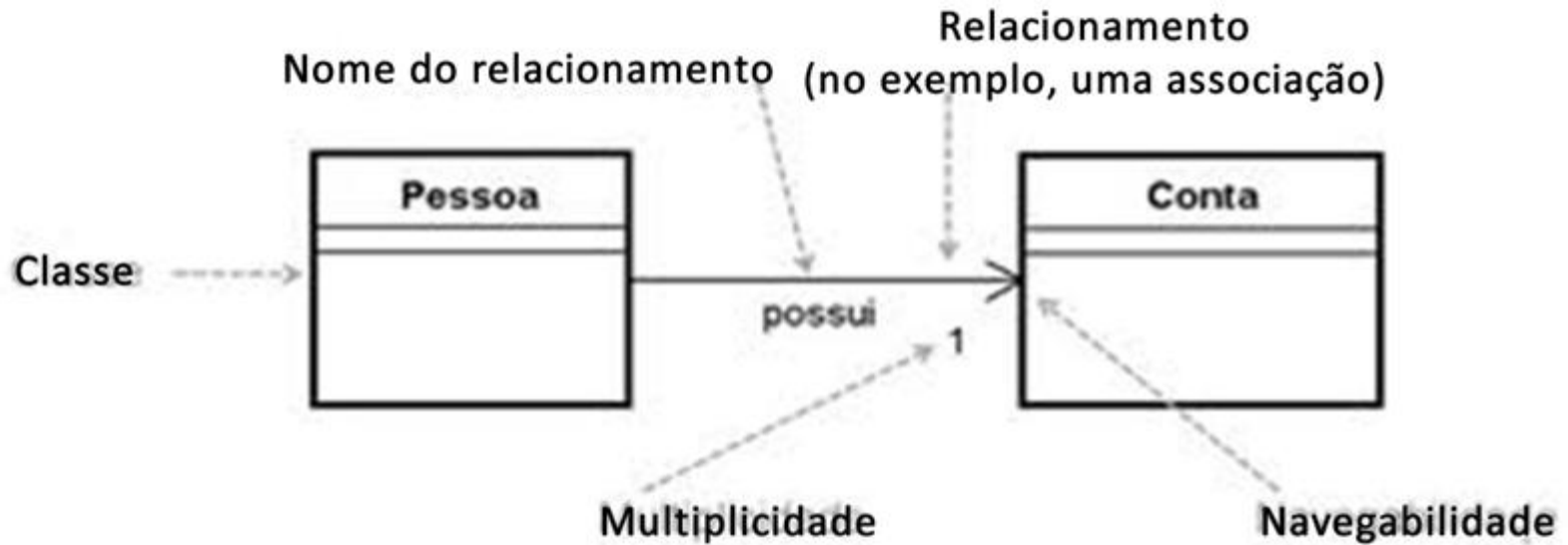
- Poderíamos usar herança? Qual a vantagem?
 - Qual a relação entre as classes?



Relacionamentos

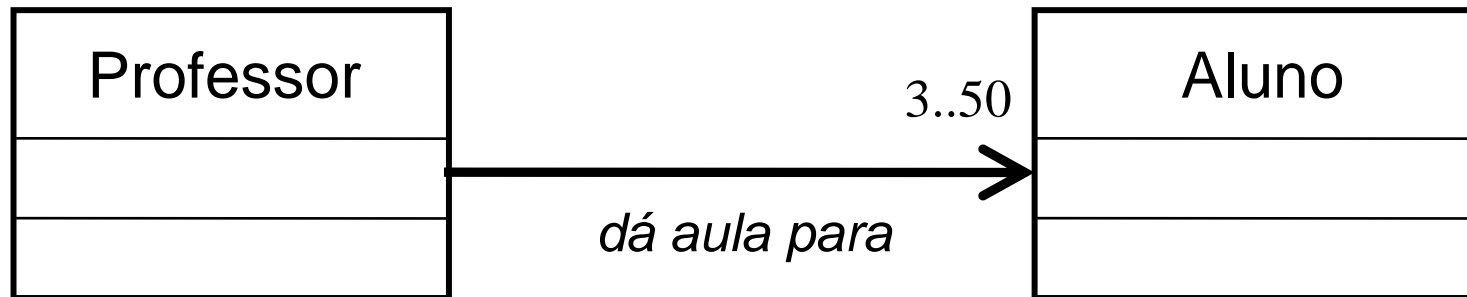
- Os relacionamentos são caracterizados por
 - Nome
 - Descrição do relacionamento
 - Em geral usa-se um verbo
 - Faz, tem, possui
 - Navegabilidade
 - Indicada por uma seta no fim do relacionamento
 - Uni (uma flecha) ou bidirecional (sem flechas/duas flechas)
 - Multiplicidade
 - Quantidade de elementos que cada relacionamento pode assumir
 - 0..1, 0..*, 1, 1..*, 2, 3..7

Relacionamentos

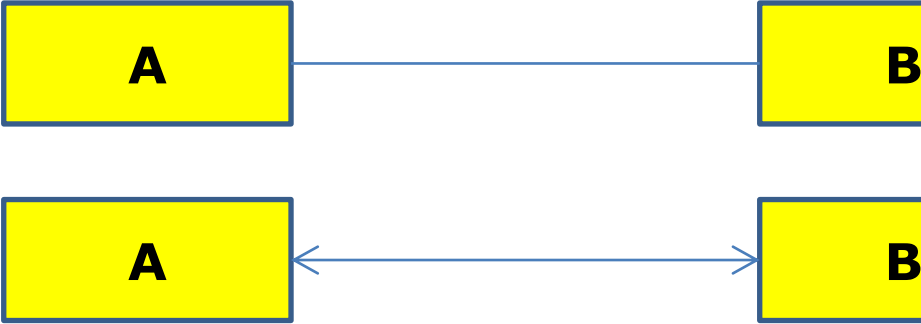



Nome do relacionamento


- Nomear um relacionamento facilita o entendimento
- Nome do relacionamento (rótulo) é colocado ao longo da linha de associação



Navegabilidade

- Navegabilidade indica a direcionalidade com que as classes se relacionam
- Ambas as classes se relacionam (sabem da existência uma da outra)

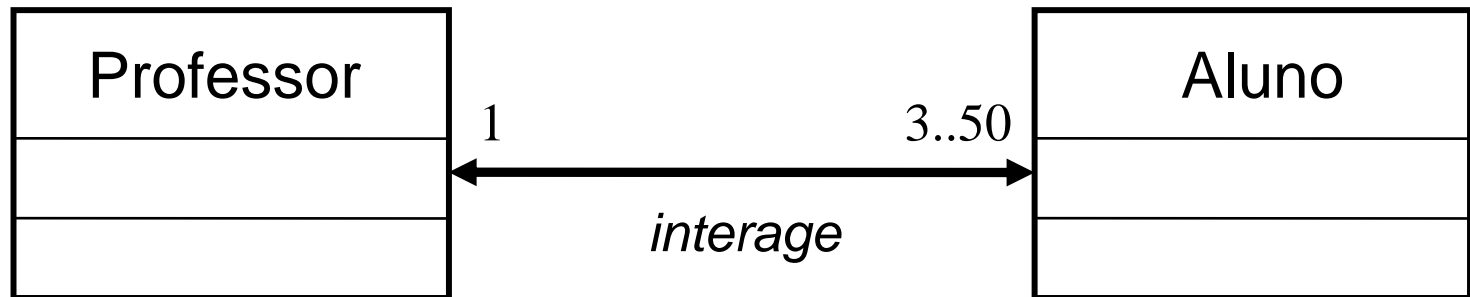
```
graph LR; A[A] <--> B[B];
```
- **B** não sabe da existência de **A**

```
graph LR; A[A] --> B[B];
```
- **A** não sabe da existência de **B**

```
graph LR; B[B] --> A[A];
```

Multiplicidade

- Multiplicidade é o **número de instâncias** de uma classe relacionada com uma ou mais instâncias de outra classe
- Exemplo: Professor e Aluno
 - Cada Professor pode interagir com 3 a 50 Alunos
 - Cada Aluno pode interagir com apenas um Professor
 - Pensando em um único curso



Multiplicidade

Muitos

*

Exatamente um

1

Zero ou mais

0..*

Um ou mais

1..*

Zero ou um

0..1

Faixa especificada

2..4

Multiplicidade

- Exemplos

- Uma mesa de restaurante pode ter vários ou nenhum pedido
 - $*..0$
- Uma cotação pode incluir no mínimo 1 e até muitos (*) itens cotados
 - $1..*$
- Uma casa pode ter de 0 a 3 funcionários
 - $0..3$

Associação simples

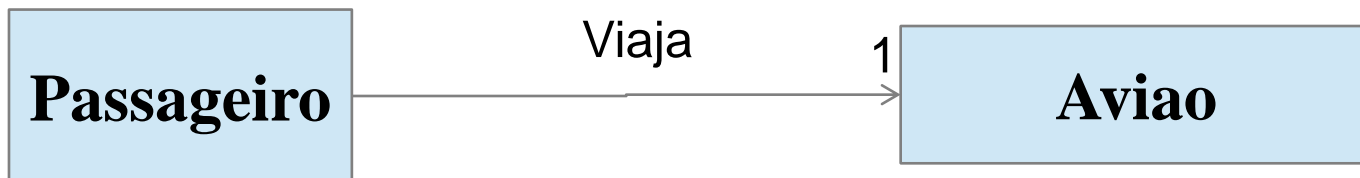
- É a forma mais fraca de relacionamento entre classes
 - As classes que participam desse relacionamento são **independentes**
 - São representadas como linhas conectando as classes participantes
 - Podem ter um nome identificando a associação
 - Podem ter uma seta junto ao nome indicando que a associação somente pode ser utilizada em uma única direção (o mais usual e adequado)
 - Representa relacionamentos “**usa um**”
 - Pessoa **usa um** Carro

Associação simples

- Na implementação
 - ObjetoA **usa** ObjetoB quando o ObjetoA **chama um método público** do ObjetoB
- Associação simples também é chamada de **dependência**
- Diagramas de dependência são os primeiros diagramas usado para compreender um código que não é seu

Associação simples

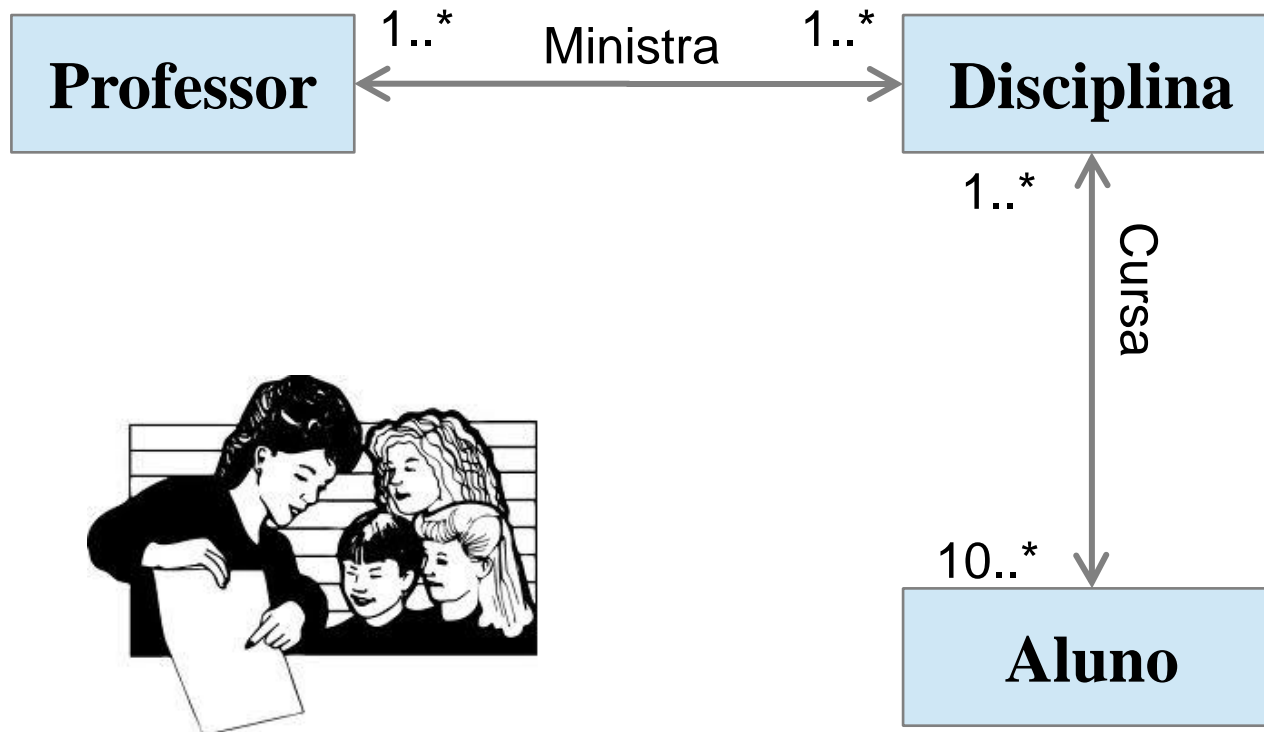
- Exemplo
 - Um **Passageiro** pode viajar para qualquer lugar, dependendo de qual **Avião** ele entrar
 - Para que um **Passageiro** viaje, ele precisa apenas de uma indicação de qual **Avião** ele deve entrar. Ele não precisa ter como parte de sua informação (atributo) a referência a um **Avião**.



- Leitura unidirecional
 - Um **Passageiro** viaja em um **Avião**

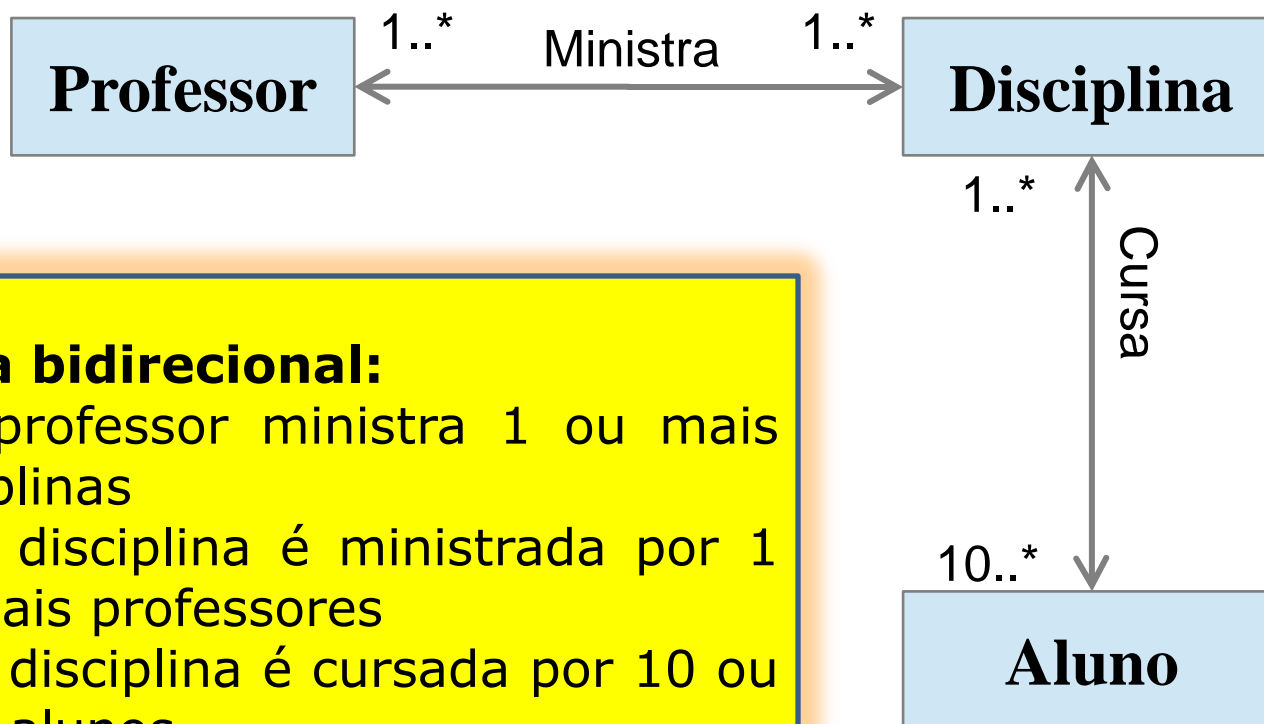
Associação simples

- Exemplo bidirecional



Associação simples

- Exemplo bidirecional



Leitura bidirecional:

- Um professor ministra 1 ou mais disciplinas
- Uma disciplina é ministrada por 1 ou mais professores
- Uma disciplina é cursada por 10 ou mais alunos
- Um aluno cursa 1 ou mais disciplinas

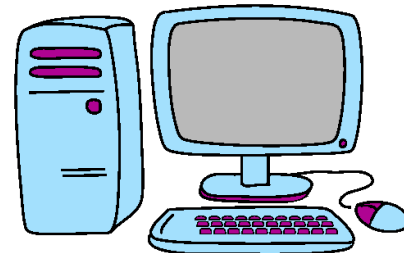
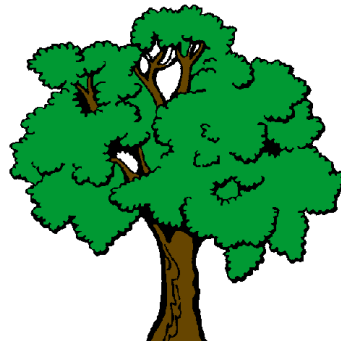
Associação simples

- Outro exemplo
 - Imagine um objeto gráfico que se auto-desenha.
 - O objeto sabe como se desenhar, mas precisa de acesso a funcionalidade gráficas exclusivas de componentes gráficos do sistema.
 - Para se desenhar, o objeto gráfico deve receber como parâmetro um componente gráfico em seu método *autoDesenho(CompGráfico comp)*.
 - Ele irá apenas usar a classe CompGráfico, sem contudo ser composto por ela.



Agregação/Composição

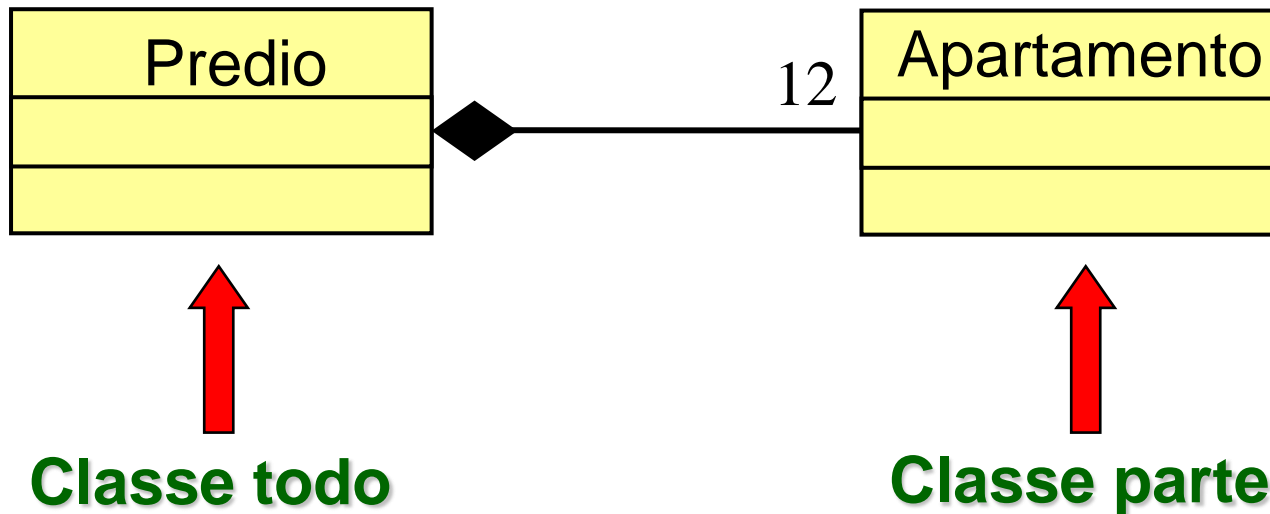
- São também formas de associação, mas representam relacionamentos do tipo “tem um”
 - Uma classe é **formada por** ou **contém** objetos de outras classes
 - Exemplos
 - Um carro possui rodas
 - Uma árvore é composta de folhas, tronco, raízes, ...
 - Um computador é composto de CPU, memória, teclado, mouse, monitor, ...



Agregação/Composição

- Classe todo
 - É a classe resultante da agregação/composição
- Classe parte
 - É a classe cujas instâncias formam a agregação/composição
- Exemplo de **composição**: **Predio** e **Apartamento**
 - Um prédio tem apartamentos
 - Classe **Predio**: todo ou agregada
 - Classe **Apartamento**: parte

Composição



- Prédio tem como atributo um conjunto (*array*) de apartamentos
- Se o prédio deixar de existir, os apartamentos também deixam de existir
- Segundo a cardinalidade, um prédio precisa ter obrigatoriamente 12 (exatos) apartamentos

Composição

- Na composição, o **todo** é responsável pelo ciclo de vida da **parte**.
- Também se diz que o **todo** é dono da **parte**, e não apenas “possui a **parte**”
- Assim, em composição, a criação da **parte** ocorre no **todo**.

Agregação

- Agregação é uma forma mais fraca de composição
- **Composição:** relacionamento todo-parte em que as partes não podem existir independentes do todo
 - Se o todo é destruído, as partes são destruídas também
 - Uma parte pode ser de um todo por vez
- **Agregação:** relacionamento todo-parte que não satisfaz um ou ambos os critérios
 - A destruição do objeto não implica a destruição do objeto parte
 - Um objeto pode ser parte componente de vários outros objetos

Agregação/Composição

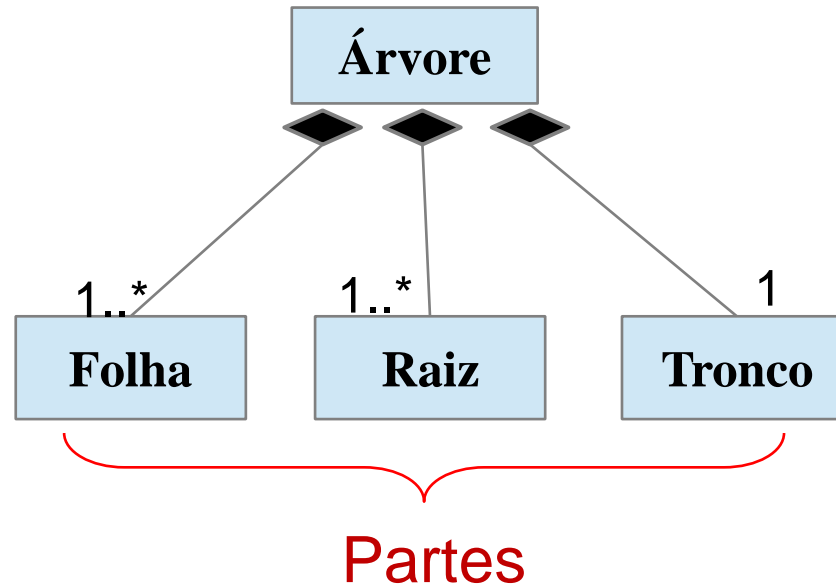
- No diagrama de classes
 - Composição
 - Associação representada com um losango **sólido** do lado **todo**



- Agregação
 - Associação representada com um losango **sem preenchimento** do lado **todo**



Composição

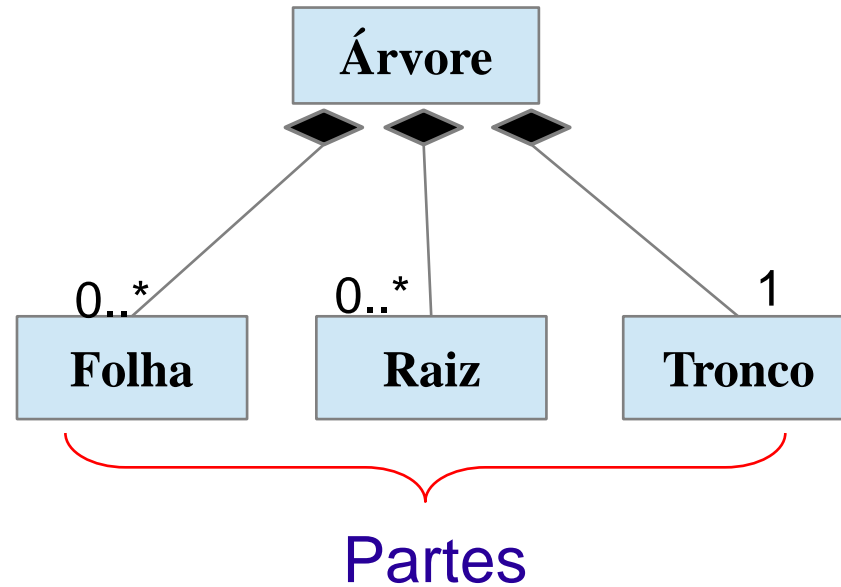
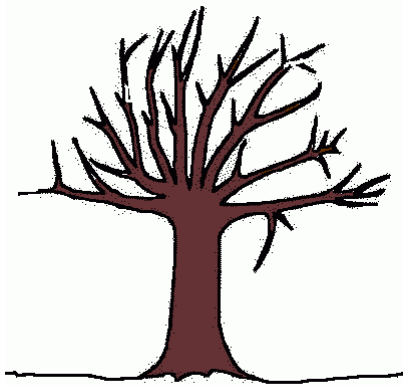


composição



- Não faz sentido que **Folha**, **Raiz** ou **Tronco** existam sem que sejam atributos de uma **Árvore**
 - Neste modelo em particular
- Ainda segundo o diagrama, não pode haver uma **Árvore** sem **Folha**, **Raiz** ou **Tronco** (cardinalidade)

Composição



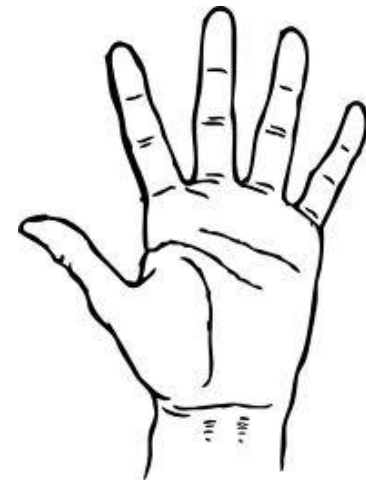
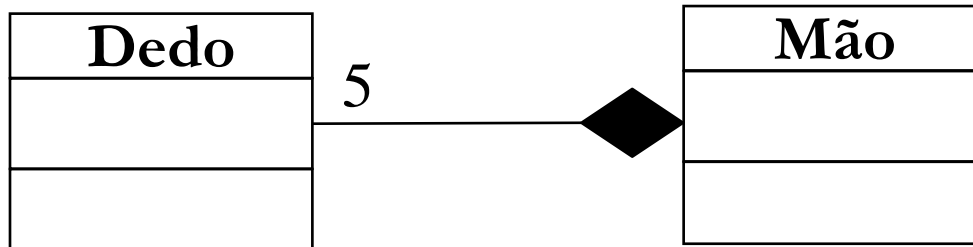
composição



- Agora pode haver uma **Árvore** sem **Folha** e sem **Raiz**, mas não sem **Tronco**

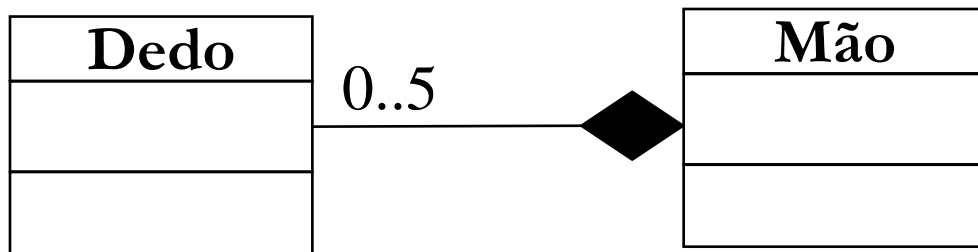
Composição

- Não faz sentido que um **Dedo** exista se não for parte de uma **Mão**
- Segundo a cardinalidade, não pode haver uma mão sem dedos
 - Todo mão tem exatos 5 Dedos



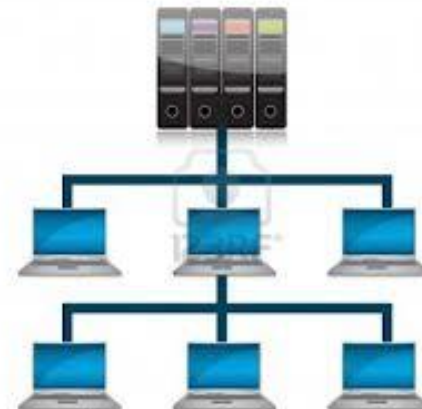
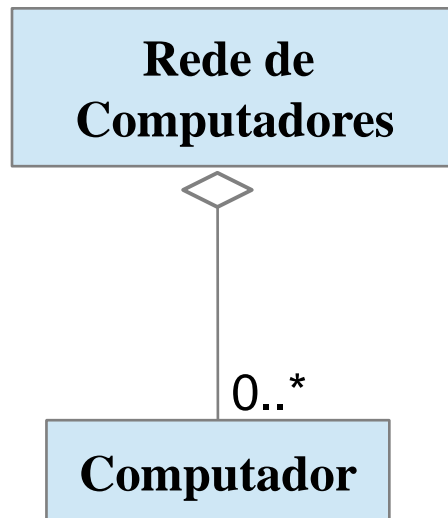
Composição

- Não faz sentido que um **Dedo** exista se não for parte de uma **Mão**
- Na definição de agora, uma mão pode não ter dedos
 - Cardinalidade mínima é 0 e máxima é 5



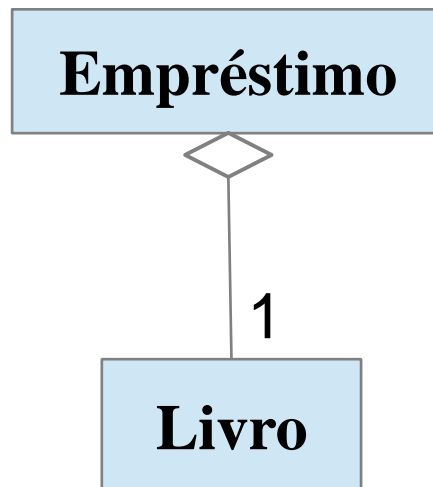
Agregação

- Uma **Rede** pode ter nenhum ou muitos **Computadores**
- Um computador existe independentemente de uma rede
- Um computador pode estar ligado a mais de uma rede ao mesmo tempo



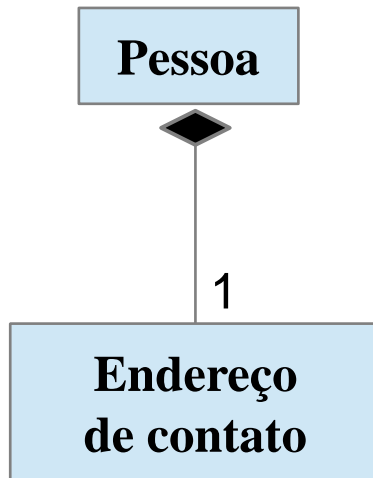
Agregação

- Um **Livro** existe independente de um **Empréstimo**
- Porém, um **Empréstimo** precisa ter pelo menos um **Livro**

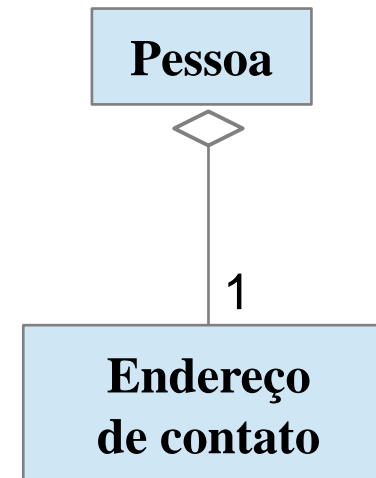


Composição/Agregação

- Quando usar composição ou agregação?



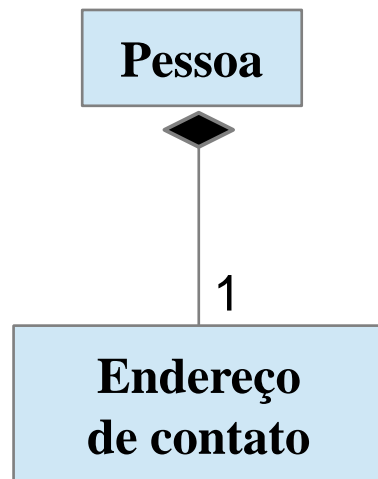
OU



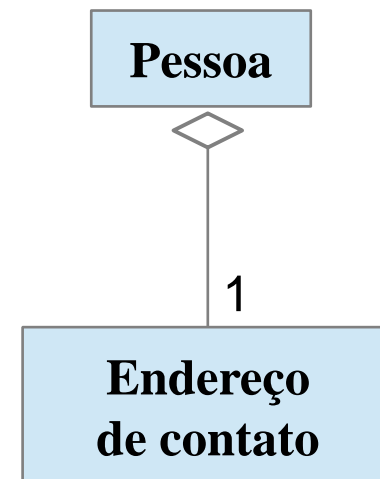
?

Composição/Agregação

- Quando usar composição ou agregação?
 - Depende dos requisitos do projeto
 - Deve-se interpretar o problema e justificar a escolha

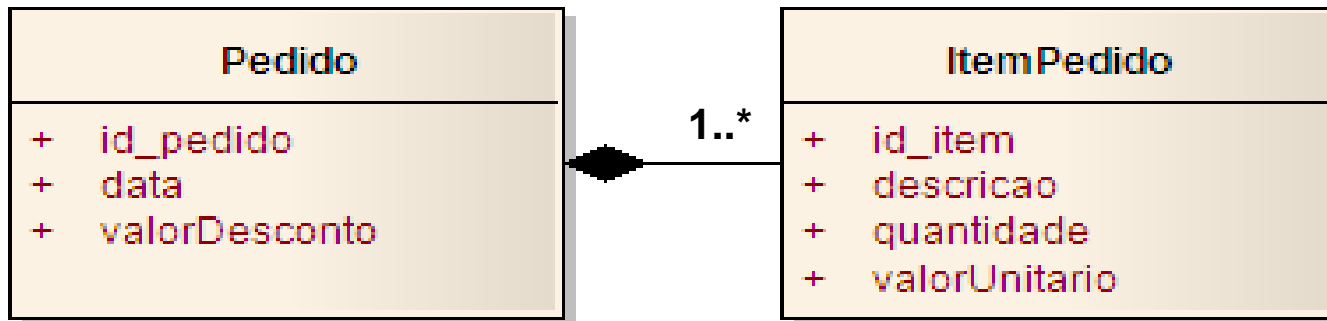


OU



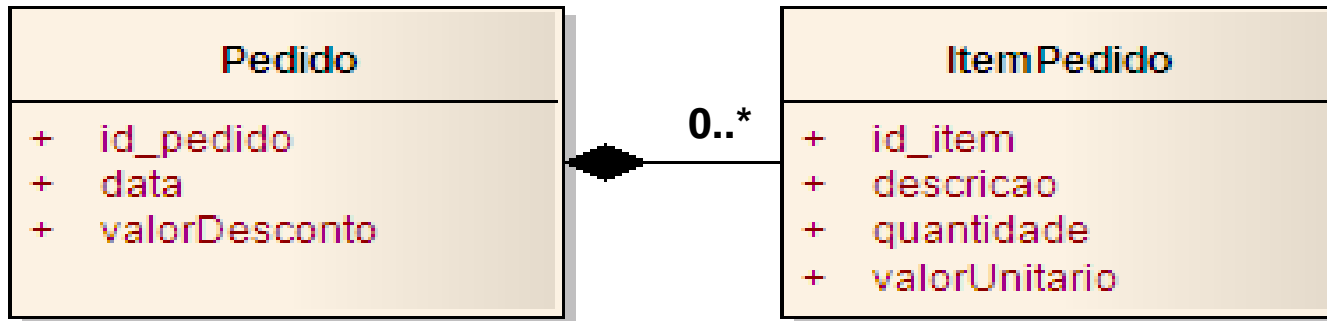
Composição

- Um **Pedido** é composto por um ou vários **ItemPedido**
 - Pela cardinalidade, um Pedido precisa ter ao menos um ItemPedido
 - Composição indica que ItemPedido só existe com Pedido



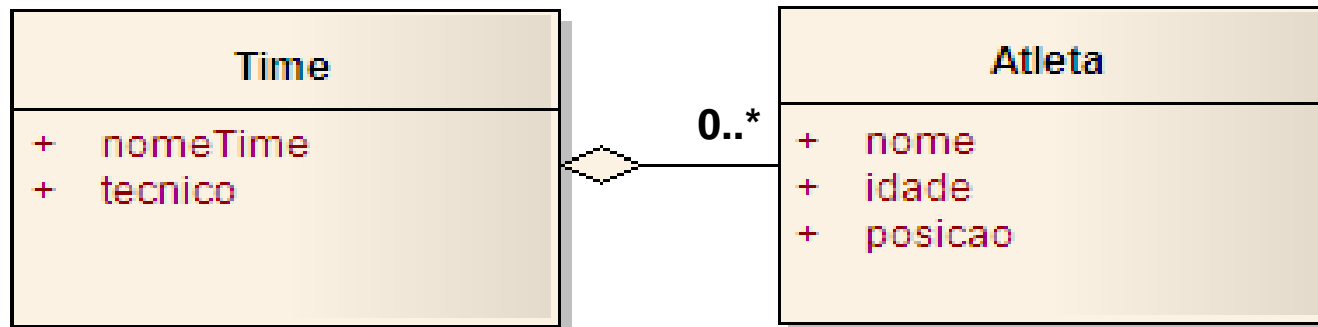
Composição

- Um **Pedido** é composto por um ou vários **ItemPedido**
 - **Agora**, um Pedido pode não ter ItemPedido
 - Contudo, a composição continua indicando que ItemPedido só existe com Pedido



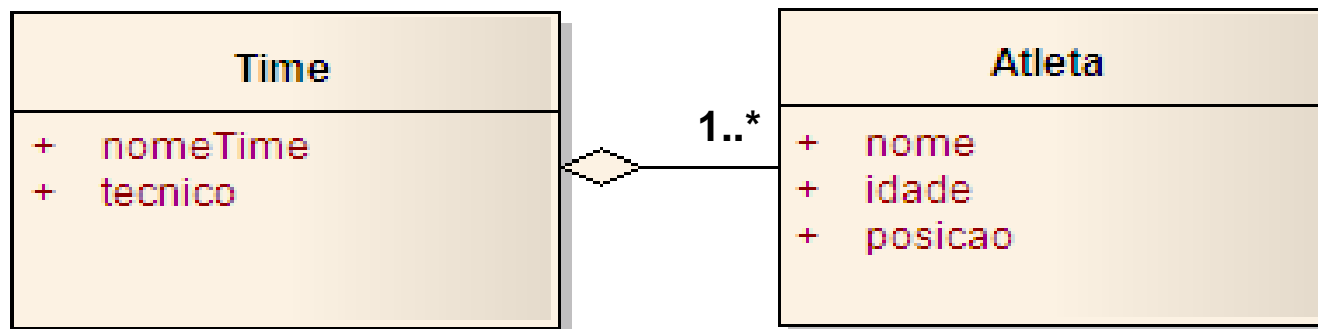
Agregação

- Um **Time** é formado por **Atletas**
 - Pela cardinalidade, um Time pode existir mesmo que não haja Atleta neste time
 - Agregação indica que Atleta existe independente da existência de um Time



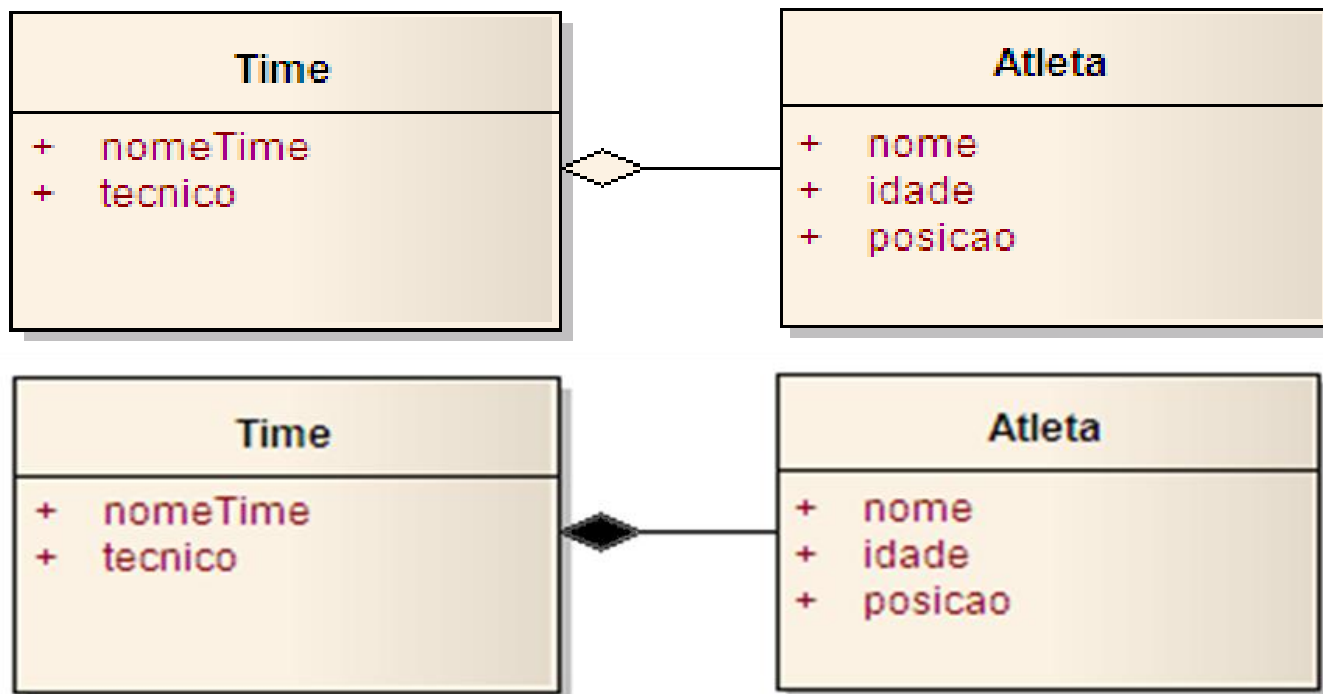
Agregação

- Um **Time** é formado por **Atletas**
 - **Agora**, um Time precisa conter pelo menos um Atleta
 - Contudo, a agregação continua indicando que Atleta existe independente da existência de um Time



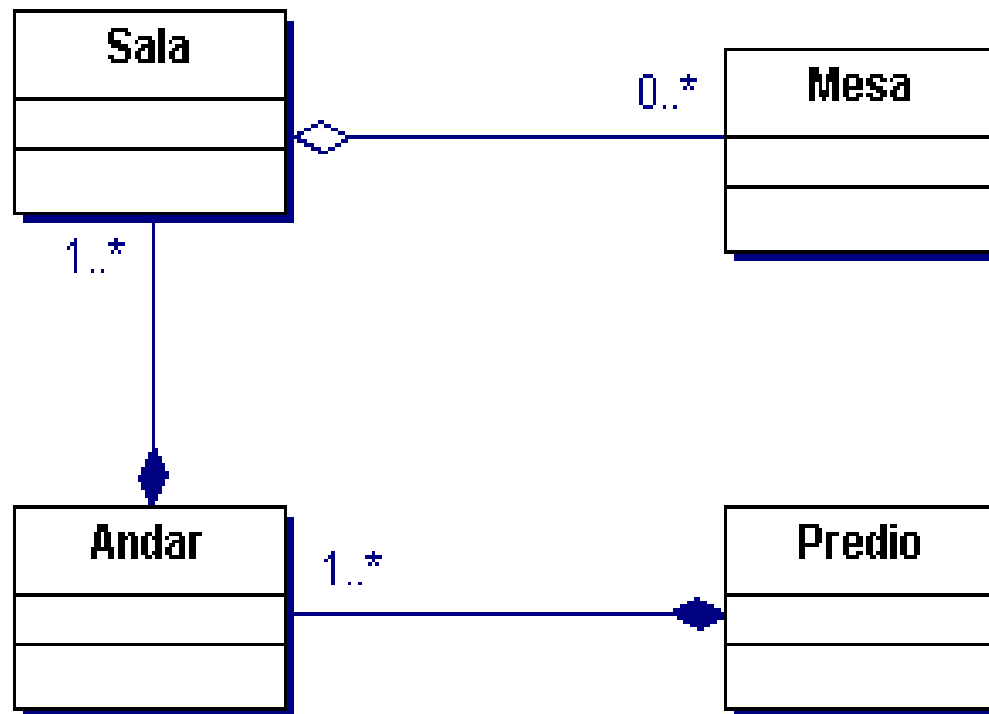
Agregação/Composição

- A semântica (significado) é interpretada pelo projetista
 - Modela o sistema de acordo com sua compreensão e conveniência
 - O mesmo problema pode ser interpretado como composição ou agregação



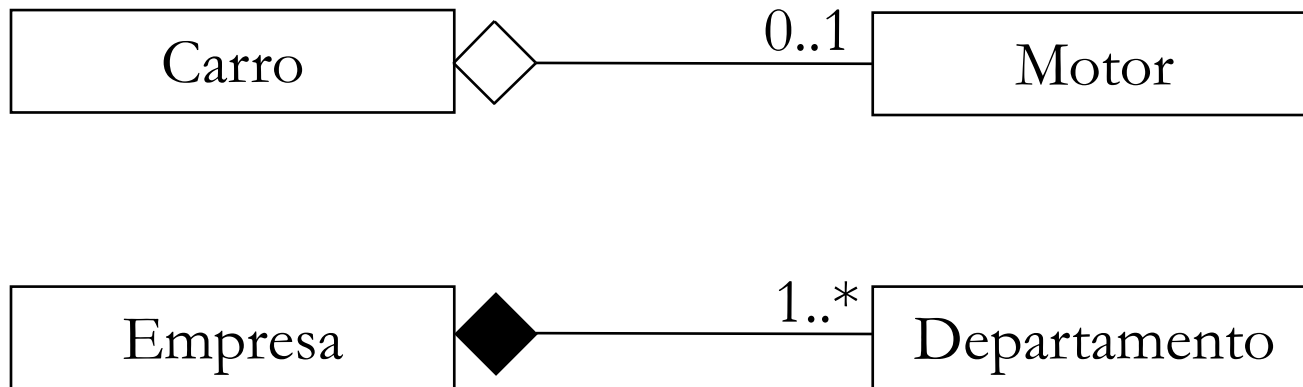
Agregação/Composição

- Outros exemplos



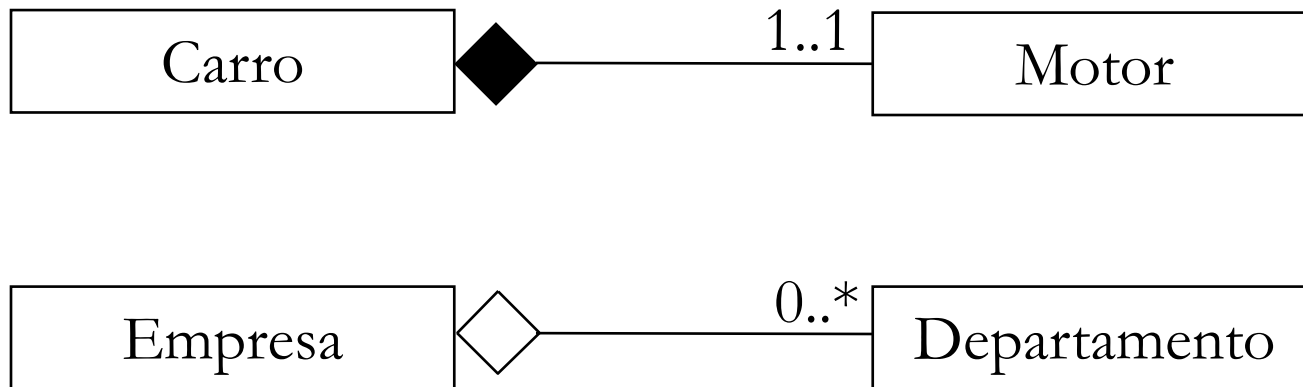
Agregação/Composição

- Outros exemplos



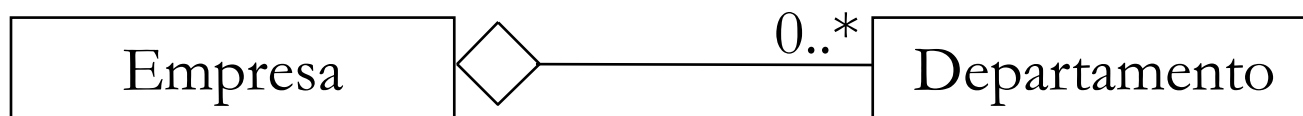
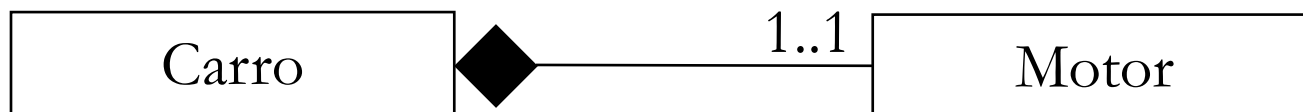
Agregação/Composição

- Outros exemplos
 - Se invertermos, fica correto?



Agregação/Composição

- Outros exemplos
 - Se invertermos, fica correto?
 - Sim! O projeto é seu!
 - Desde que satisfaça as características do problema



Agregação/Composição

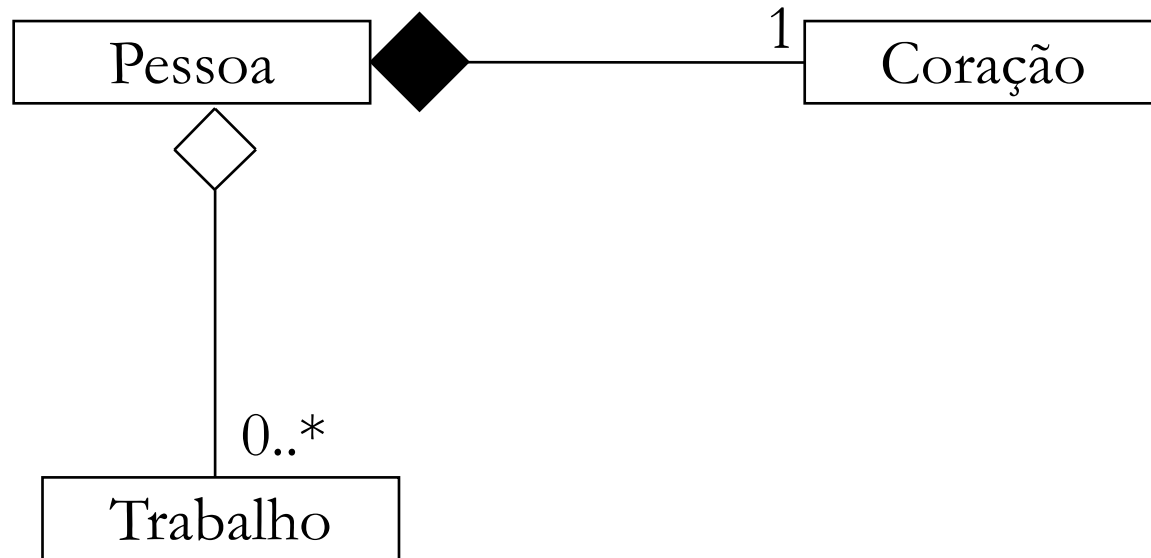
- Outros exemplos
 - Objeto **Pessoa** possui um atributo chamado **Coração**
 - **Coração** também possui atributos:
 - Conjunto to tipo **Arteria**
 - Conjunto do tipo **Cardiomiocito**
 - FC
 - ...
 - Um **Coração** só faz sentido se estiver vinculado a uma **Pessoa**
 - Atributo de **Pessoa**
- Então, a relação Pessoa – Coração é uma **composição**

Agregação/Composição

- Outros exemplos
 - Objeto **Pessoa** também pode ter um atributo chamado **Trabalho**
 - **Trabalho** pode ter seus próprios atributos: **Local**, CNPJ, ...
 - Em geral, uma **Pessoa** tem um **Trabalho** mas não precisa ter um para existir
 - Além disso, se você pedir demissão, o **Trabalho** que antes era seu não deixará de existir.
- Assim, a relação Pessoa – Trabalho é uma **agregação**

Agregação/Composição

- Outros exemplos

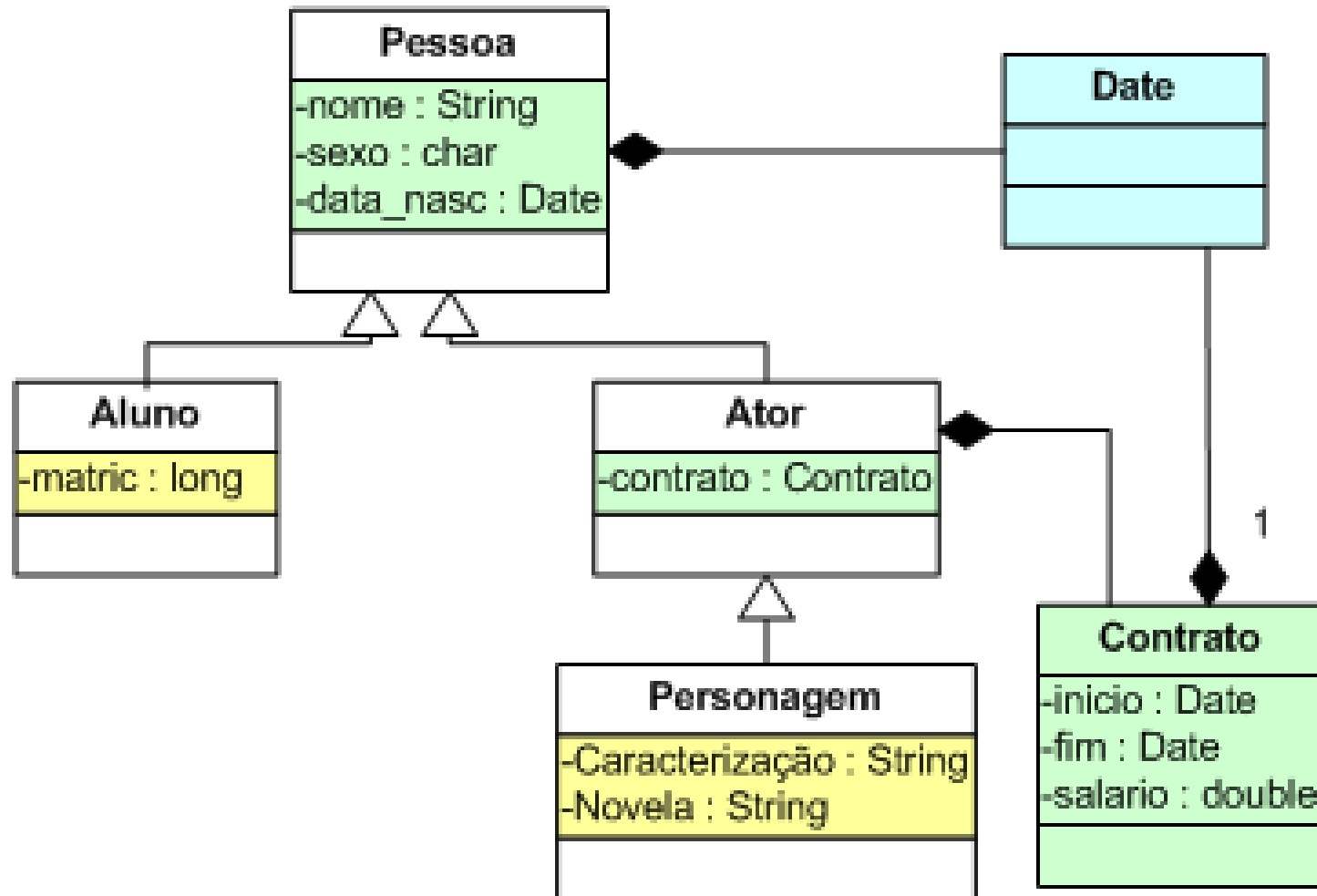


Relacionamentos

- Como saber que tipo de relacionamento deve ser utilizado?
 - Existem atributos ou métodos em comum entre as classes? Ou seja, uma classe “é do tipo” da outra?
 - **Sim**: Isso é **HERANÇA**
 - **Não**: Existe relação todo-parte?
 - **Não**: Isso é uma **ASSOCIAÇÃO SIMPLES**
 - **Sim**: A parte vive sem o todo?
 - » **Sim**: Isso é uma **AGREGAÇÃO**
 - » **Não**: Isso é uma **COMPOSIÇÃO**

Relacionamentos

- Exemplo



Resumo

- Histórico da UML
- Diagrama de classes
- Representação de classes
 - Atributos e métodos
 - Tipos de acesso e modificadores
- Nesta aula foram vistos os principais relacionamentos entre classes
 - Herança, Implementação, Associação, Agregação e Composição

Softwares para UML

- Plugin para o NetBeans
 - Ferramentas -> Plugins -> easyUML
- Outros
 - Rational Rose
 - Yed
 - StarUML
 - Dia
 - Argo UML
 - Microsoft Visio
 - Enterprise Architect

Dúvidas?

