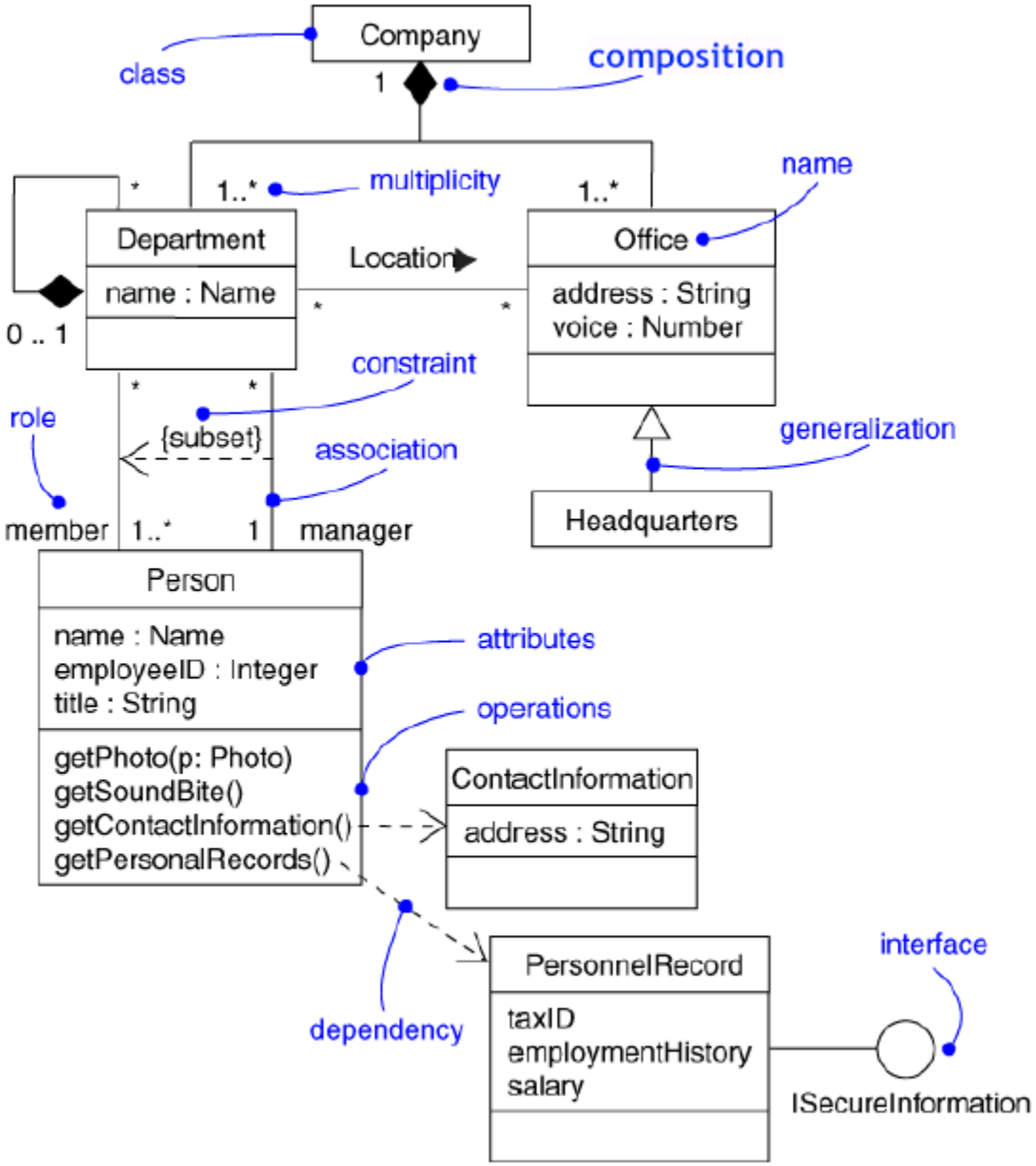
Diagrama de Classes de Objetos

(A maioria dos slides a seguir foram cedidos pelo Prof. Tarcísio Lima)

Parte 1 - Introdução

Exemplo de Diagrama de Classes



Objetos

- Um **objeto** é algo com fronteiras bem definidas,relevante para o problema em causa,
- com estado,
  - modelado por valores de atributos (tamanho, forma, peso etc.) e por ligações que num dado momento tem com outros objetos
- com comportamento
  - um objeto exibe comportamentos **invocáveis** (por resposta a chamadas de operações) ou **reativos** (por resposta a eventos)
- e identidade
  - **no espaço:** é possível distinguir dois objetos mesmo que tenham o mesmo estado
    - exemplo: podemos distinguir duas folhas de papel A4, mesmo que tenham os mesmos valores dos atributos
  - **no tempo:** é possível saber que se trata do mesmo objeto mesmo que o seu estado mude
    - exemplo: se pintarmos um folha de papel A4 de amarelo, continua a ser a mesma folha de papel

Objetos do mundo real e objetos computacionais

- No desenvolvimento de software orientado por objetos, procura-se imitar no computador o mundo real visto como um conjunto de objetos que interagem entre si
- Muitos objetos computacionais são imagens de objetos do mundo real
- Dependendo do contexto (análise ou projeto) podemos estar falando em objetos do mundo real, em objetos computacionais ou nas duas coisas simultaneamente
- Exemplos de objetos do mundo real:
  - o Sr. João
  - o curso de UML no dia 18/04/2011 às 19:00 horas
- Exemplos de objetos computacionais:
  - o registro que descreve o Sr. João **(imagem de objeto do mundo real)**
  - uma árvore de pesquisa binária **(objeto puramente computacional)**

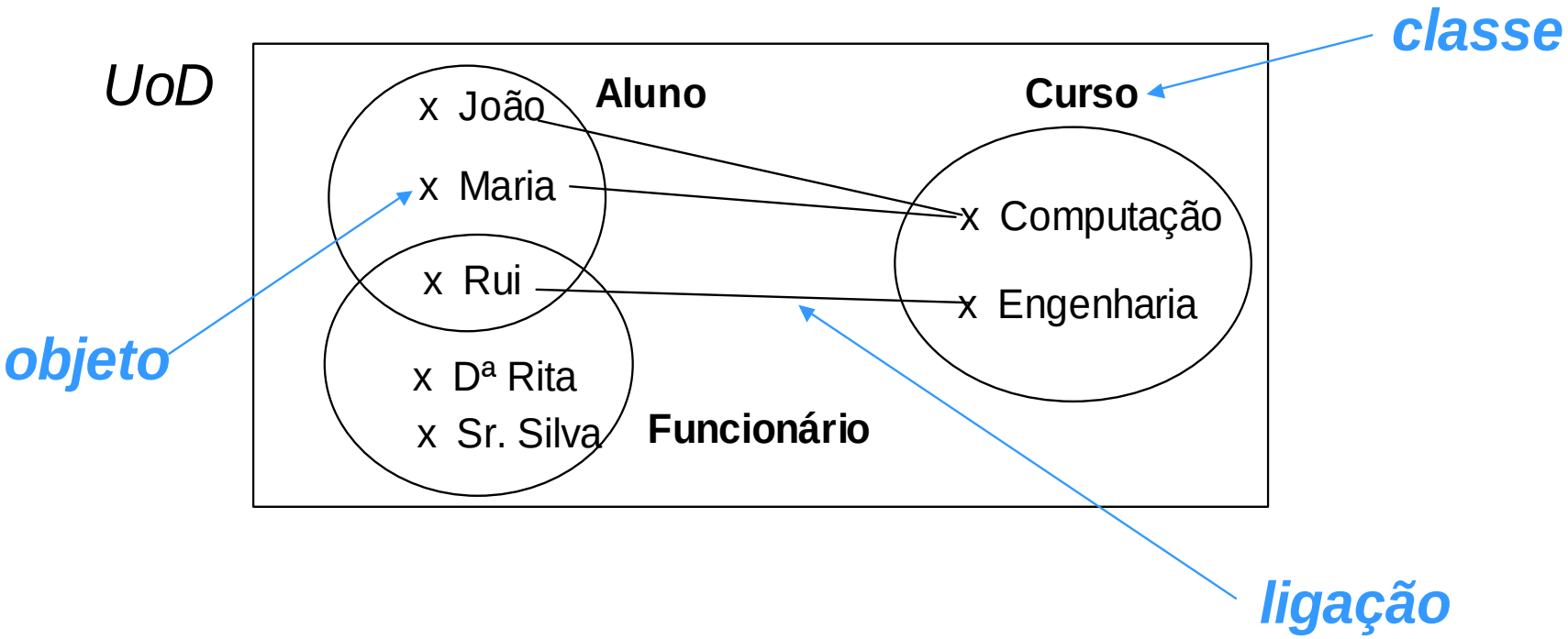
Classes (1)

- No desenvolvimento de software OO, não nos interessam tanto os objetos individuais mas sim as classes de objetos
- Uma **classe** é um descritor de um conjunto de objetos que partilham as mesmas propriedades (semântica, atributos, operações e relações)
  - Trata-se de uma noção de classe em **compreensão**, no sentido de **tipo de objeto**, por oposição a uma noção de classe em **extensão**, como **conjunto de objetos do mesmo tipo**
- Um objeto de uma classe é uma **instância** da classe
- A **extensão** de uma classe é o conjunto de instâncias da classe
- Em Matemática, uma classe é um conjunto de "objetos" com uma propriedade em comum, podendo ser definida indiferentemente em compreensão ou em extensão

$$C = \underbrace{\{x \in \mathbb{N} : x \bmod 3 = 2\}}_{\text{Definição em compreensão}} = \underbrace{\{2, 5, 8, 11, 14, ...\}}_{\text{Definição em extensão}}$$

Classes (2)

- O conjunto de todos os objetos num determinado contexto forma um universo (*UoD - Universe of Discourse*)
- As extensões das classes são subconjuntos desse universo

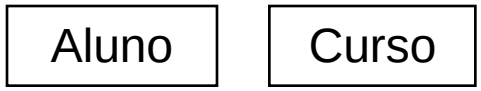


Classes (3)

- Classes podem representar:
  - **Coisas concretas**: Pessoa, Turma, Carro, Imóvel, Fatura, Livro
  - **Papéis que coisas concretas assumem**: Aluno, Professor, Piloto
  - **Eventos**: Curso, Aula, Acidente
  - **Tipos de dados**: Data, Intervalo de Tempo, Número Complexo, Vetor
- Decomposição orientada por objetos: começa identificando os tipos de objetos (classes) presentes num sistema
  - Em contraposição à decomposição funcional
  - Os tipos de objetos de um sistema são mais estáveis do que as funções, logo a decomposição orientada por objetos leva a arquiteturas mais estáveis

Classes (4)

- Em UML, uma classe é representada por um retângulo com o nome da classe



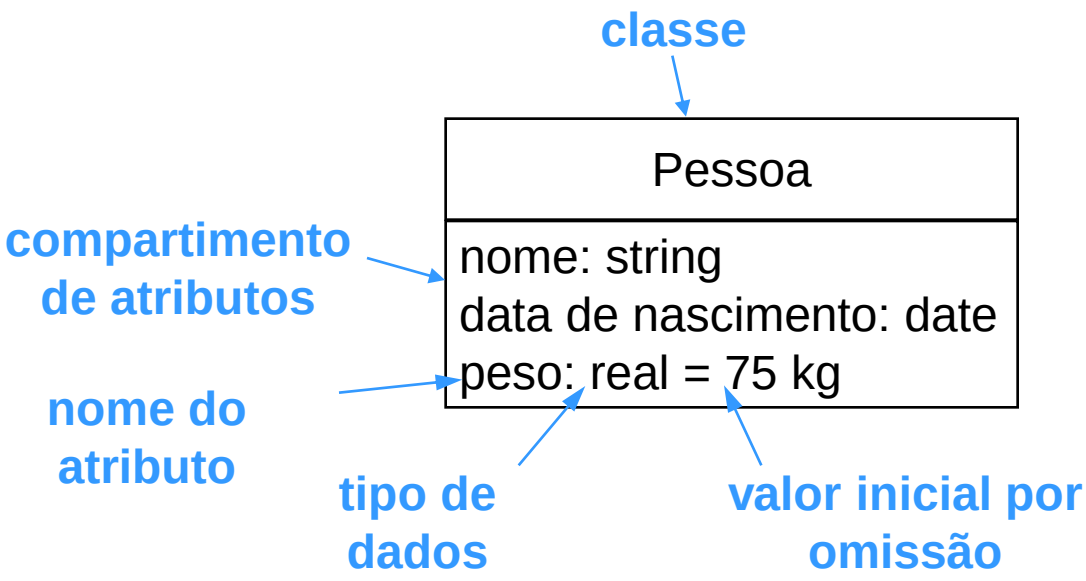
- Habitualmente escreve-se o nome da classe no singular (nome de uma instância), com a 1ª letra em maiúscula
- Para se precisar o significado pretendido para uma classe, deve-se explicar o que é (e não é ...) uma instância da classe
  - Exemplo: "Um aluno é uma pessoa que está inscrita num curso ministrado numa escola. Uma pessoa que esteve no passado inscrita num curso, mas não está presentemente inscrita em nenhum curso, não é um aluno."
  - Em geral, o nome da classe não é suficiente para se compreender o significado da classe

Atributos de instância

- O estado de um objeto é dados por valores de atributos (e por ligações que tem com outros objetos)
- Todos os objetos de uma classe são caracterizados pelos mesmos atributos (ou variáveis de instância)
  - o mesmo atributo pode ter valores diferentes de objeto para objeto
- **Atributos** são definidos ao nível da classe, enquanto que os **valores dos atributos** são definidos ao nível do objeto
- Exemplos:
  - uma pessoa (classe) tem os atributos nome, data de nascimento e peso
  - José (objeto) é uma pessoa com nome “José da Silva”, data de nascimento “18/3/1973” e peso “68kg”

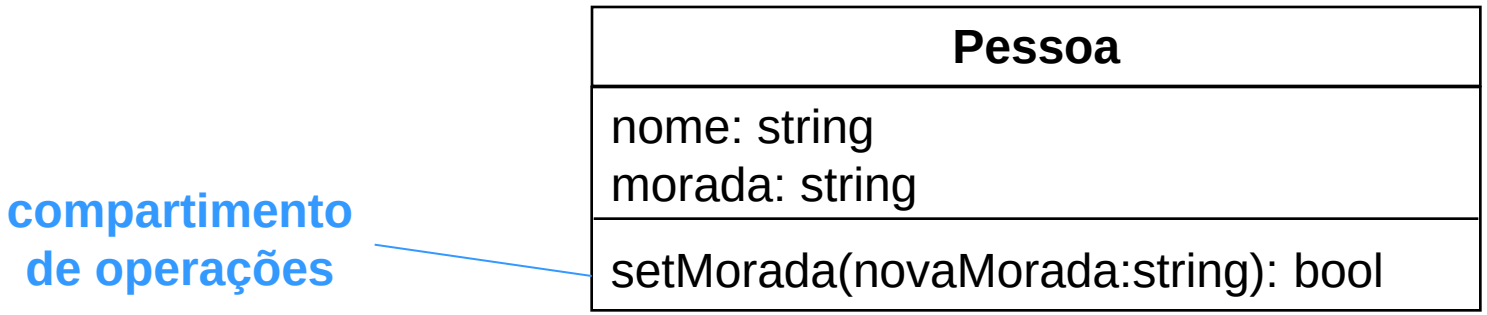
Atributos de instância

- Atributos são listados num compartimento de atributos (opcional) a seguir ao compartimento com o nome da classe
- Uma classe não deve ter dois atributos com o mesmo nome
- Os nomes dos tipos não estão pré-definidos em UML, podendo-se usar os da linguagem de implementação alvo



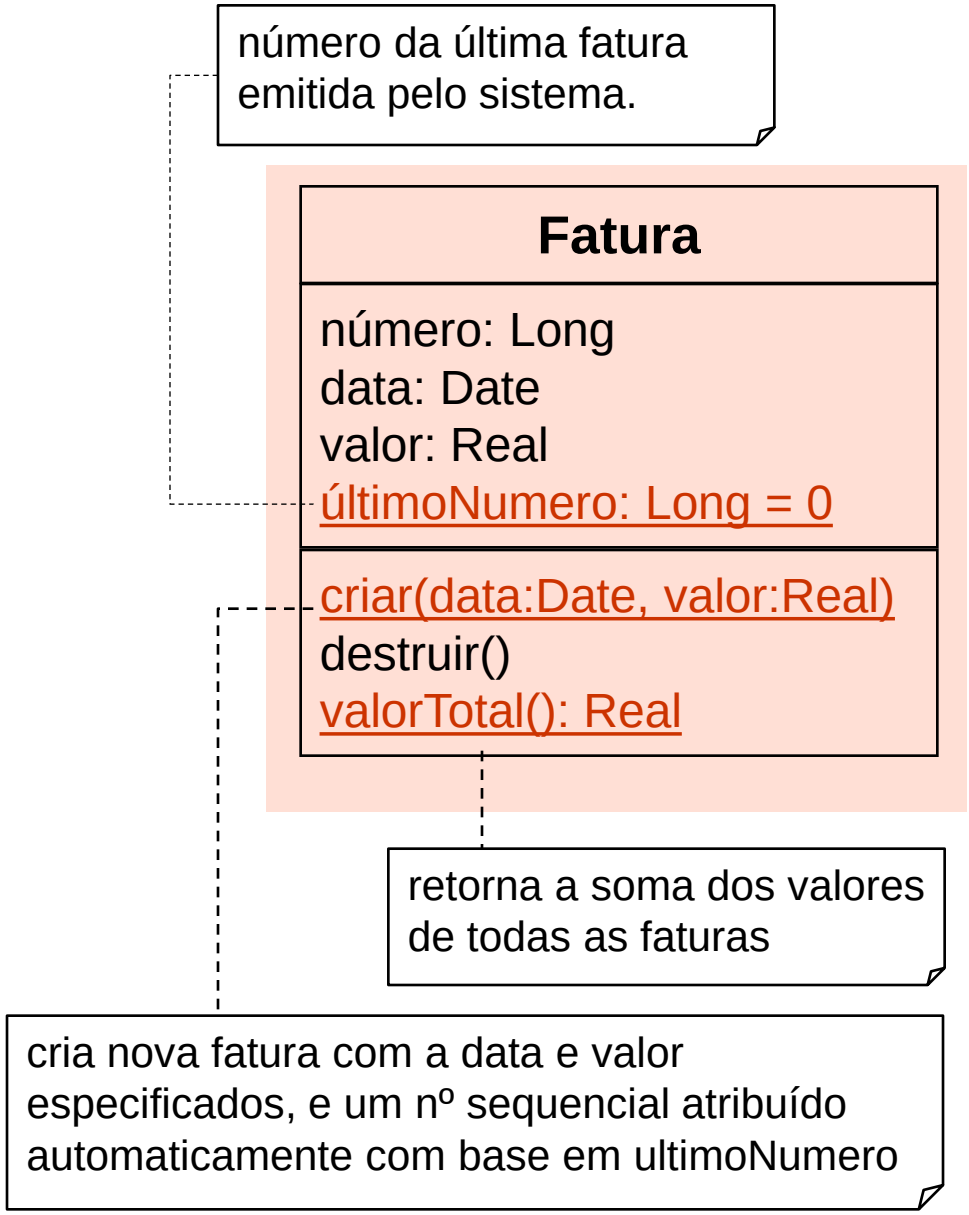
Operações de instância

- Comportamento invocável de objetos é modelado por operações
  - Uma operação é algo que se pode pedir que um objeto de uma classe faça
- Objetos da mesma classe têm as mesmas operações
- **Operações** são definidos ao nível da classe, enquanto que a **invocação de uma operação** é definida ao nível do objeto
- Princípio do **encapsulamento**: acesso e alteração do estado interno do objeto (valores de atributos e ligações) controlado por operações
- Nas classes que representam objetos do mundo real é mais comum definir **responsabilidades** em vez de operações



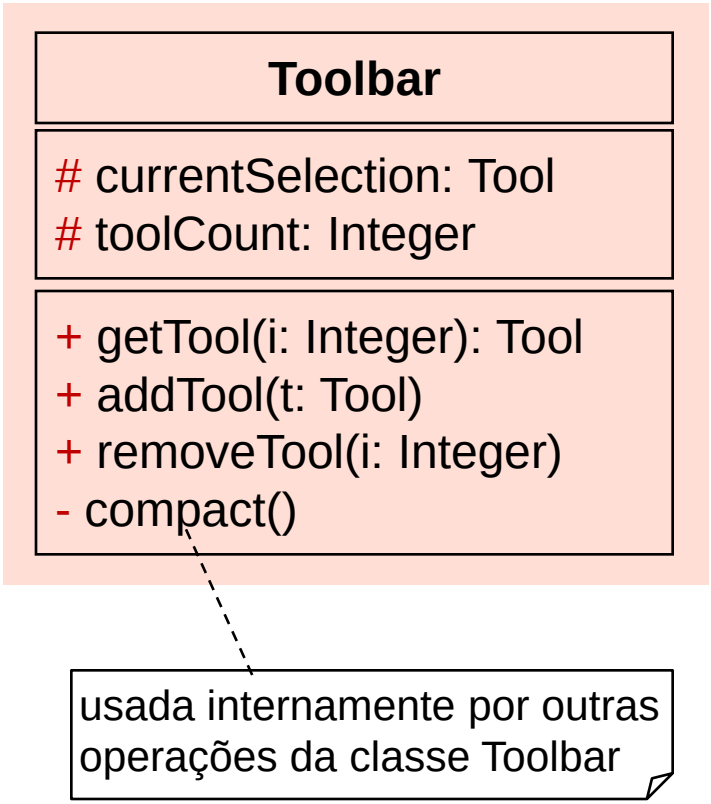
Atributos e operações estáticos (≠ de instância)

- **Atributo estático**: tem um único valor para todas as instâncias (objetos) da classe
  - O valor é definido ao nível da classe e não ao nível das instâncias
- **Operação estática**: não é invocada para um objeto específico da classe, mas sim para a classe como um todo.
- Notação: nome sublinhado
- Correspondem a métodos estáticos (*static*) em C++, C# e Java



Visibilidade de atributos e operações

- Visibilidade:
  - **+** (**public**) : visível por todas as operações, de qualquer classe
  - **-** (**private**) : visível só por operações da própria classe (do atributo ou operação)
  - **#** (**protected**): visível por operações da própria classe e descendentes (subclasses)
- Objetivo: permitir e**ncapsulamento**, ou seja, esconder todos os detalhes de implementação que não interessam aos clientes (usuários) da classe
  - Permite alterar a representação do estado (atributos) sem afetar os clientes
  - Permite validar alterações de estado

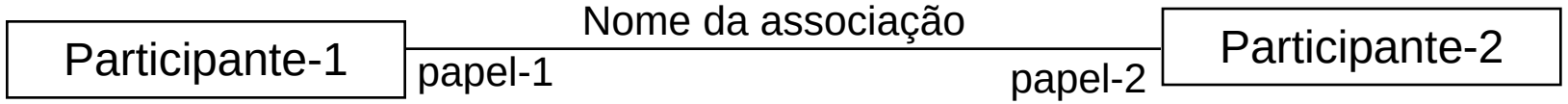


*Multiplicidade de classes e atributos

- Multiplicidade de classe: número de instâncias que podem existir
  - por omissão, é 0..*
- Multiplicidade de atributo: número de valores que o atributo pode tomar do tipo especificado
  - por omissão é 1
  - qual a diferença em relação a especificar a multiplicidade no próprio tipo de dados do atributo?

Relações entre classes: associação, agregação e composição

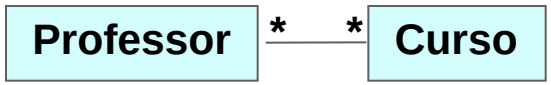
Associações binárias

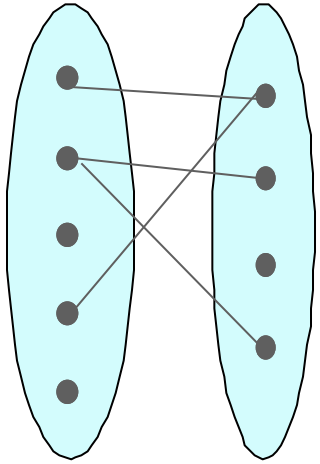


- Uma **associação** é uma relação binária entre duas classes
- Matematicamente, uma associação binária é uma relação binária entre duas classes: um subconjunto do produto cartesiano das extensões das classes participantes da associação
- Não gera novos objetos (mas sim, ligações)
- Assim como um objeto é uma instância de uma classe, uma **ligação** é uma instância de uma associação
- Pode haver mais do que uma associação (com nomes diferentes) entre o mesmo par de classes
- Papéis nos extremos da associação podem ter indicação de visibilidade (pública, privada etc.)

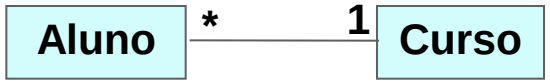
Multiplicidade de associações binárias

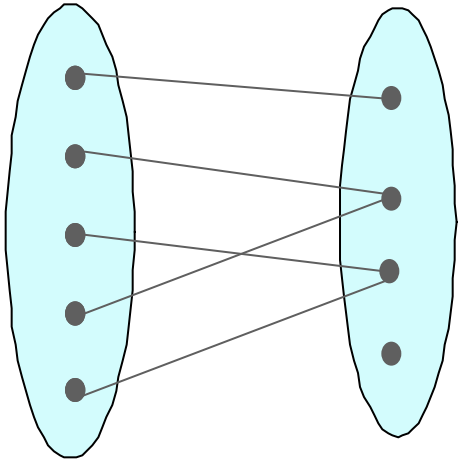
Muitos-para-Muitos



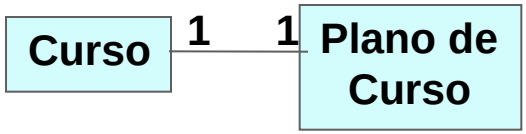


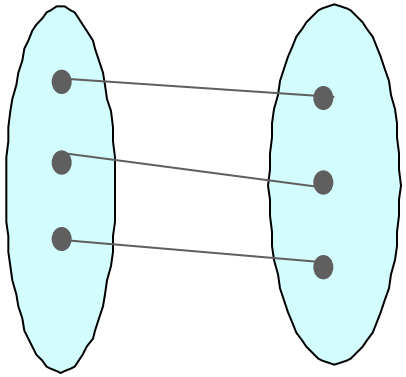
Muitos-para-1





1-para-1



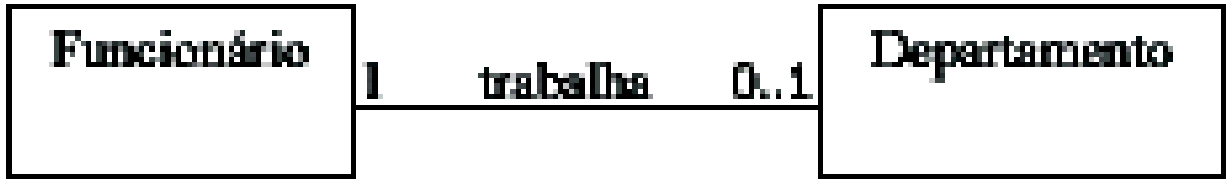


Notação para a multiplicidade

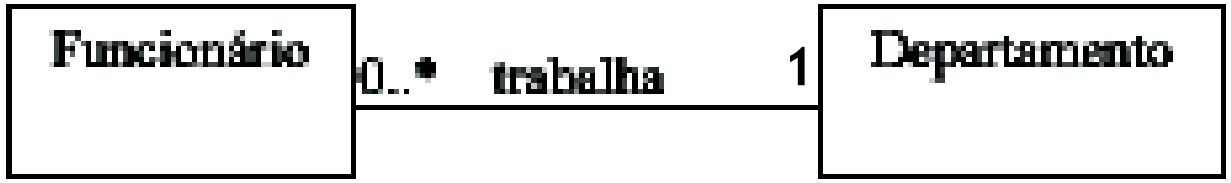
1	- exatamente um
0..1	- zero ou um (zero a 1)
*	- zero ou mais
0..*	- zero ou mais
1..*	- um ou mais
1, 3..5	- um ou três a 5

Qual é o correto?

1)



2)

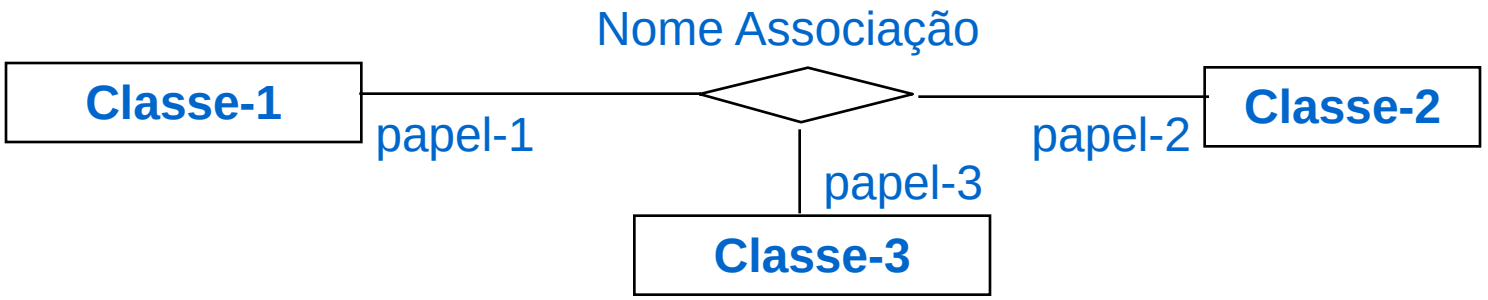


3)

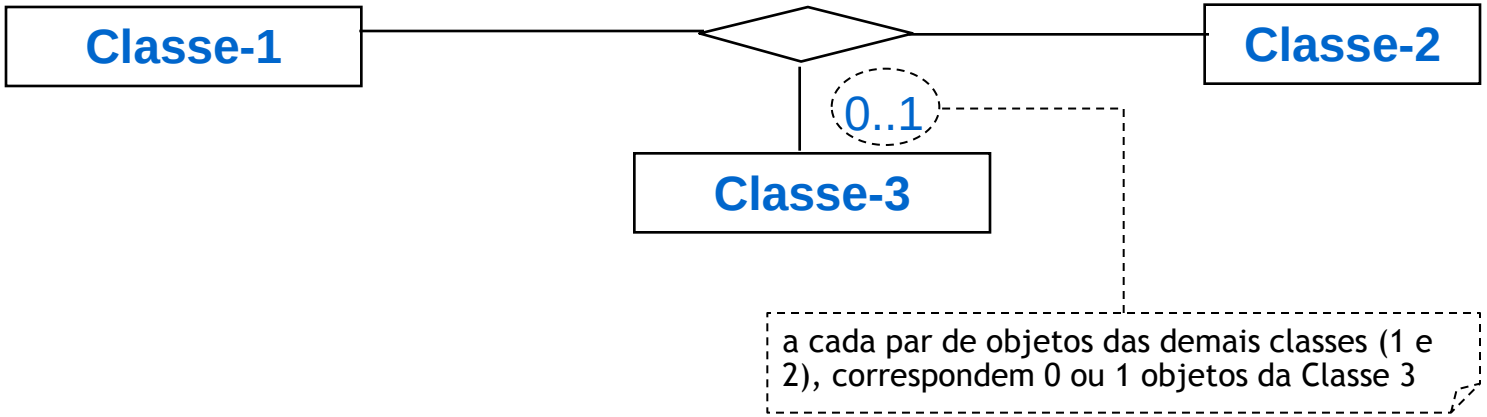
Resp: Depende da realidade que se está modelando!
Talvez a realidade que mais ocorra seja a opção 2

Associações n-árias

- Notação

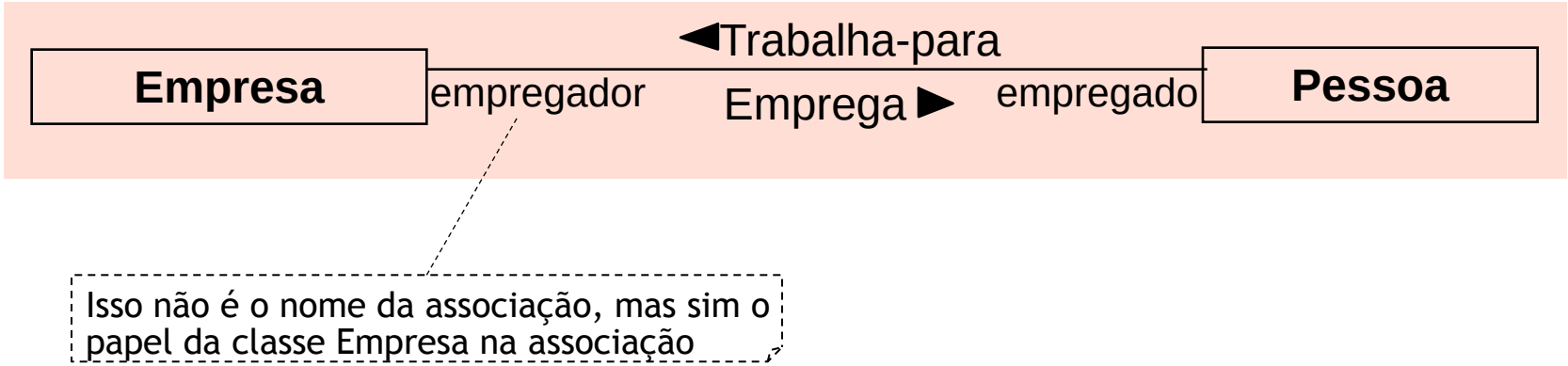


- Multiplicidade



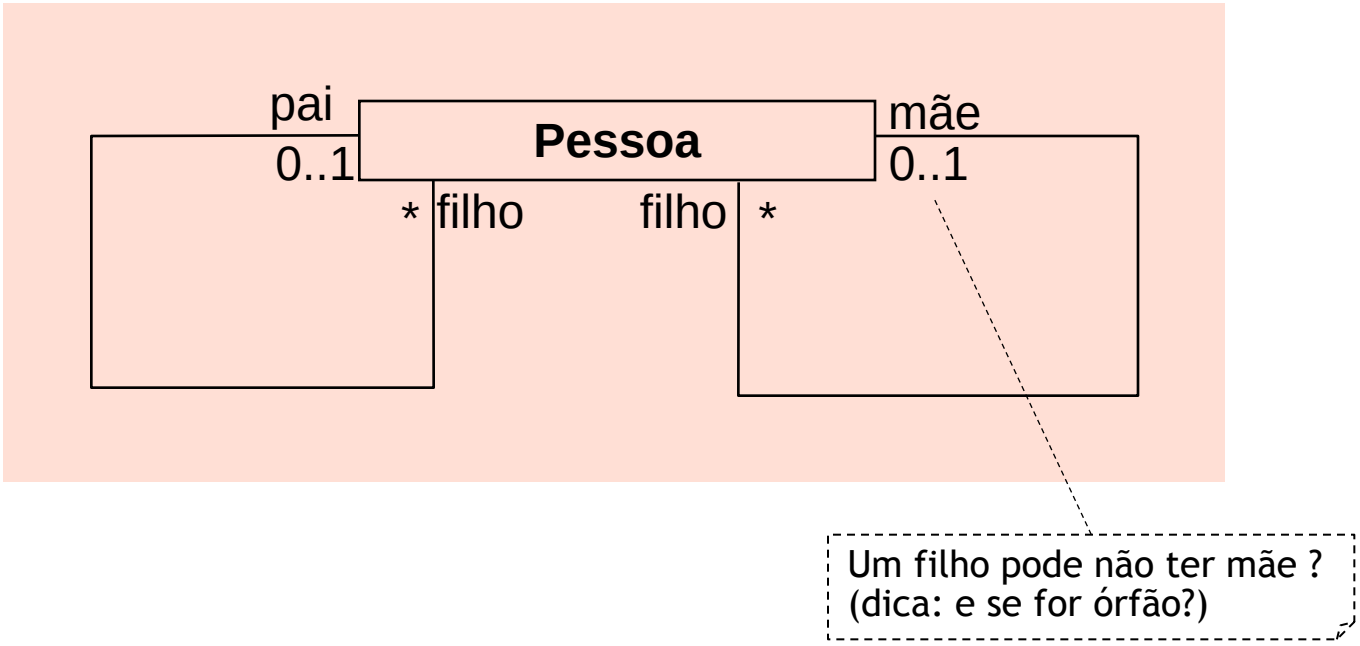
Nome de associação

- A indicação do nome é opcional
- O nome é indicado no meio da linha que une as classes participantes
- Pode-se indicar o sentido em que se lê o nome da associação



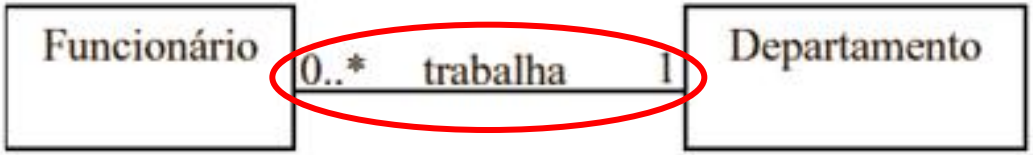
Associação reflexiva

- Pode-se associar uma classe com ela própria (em papéis diferentes)



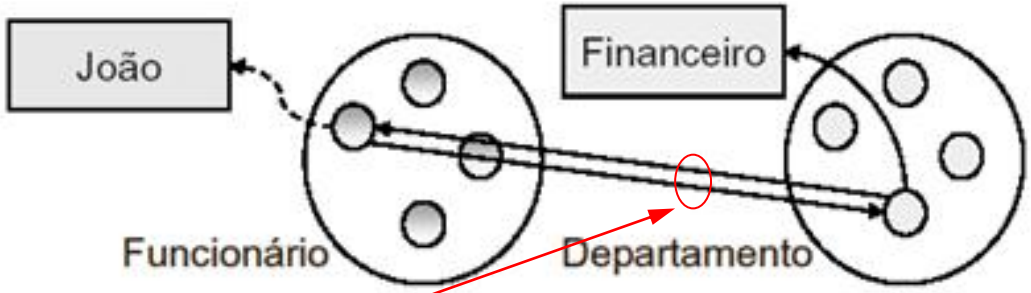
Navegabilidade de associações

Navegação ***bidirecional***



Associação Binária

Funcionário *trabalha* em Departamento



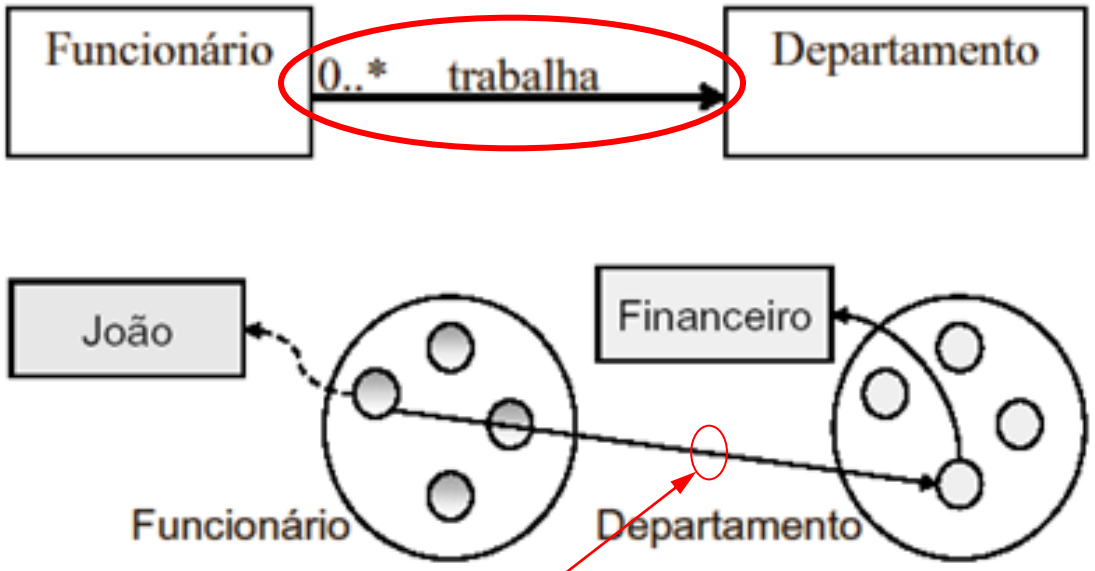
Navegação nos dois sentidos:
- **de funcionário para departamento e**
- **de departamento para funcionário**

A partir de um objeto da classe ***Funcionário (***por exemplo *João*), pode-se obter os objetos da classe ***Departamento*** a ele associados pela associação ***trabalha*** (no exemplo, apenas *Financeiro*).

Mas o contrário também é possível (no exemplo, obter *João* a partir de *Financeiro*)

Navegabilidade de associações

Navegação unidirecional

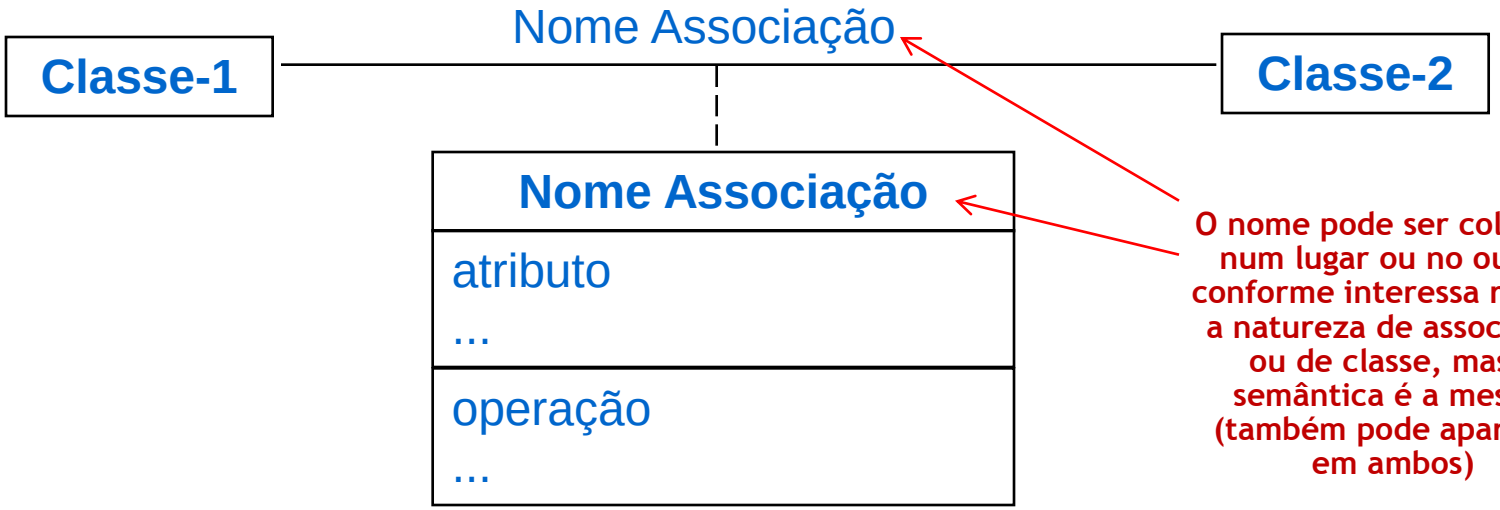


Navegação em um único sentido:
- **de funcionário para departamento**

A partir de um objeto da classe ***Funcionário (***por exemplo *João*), pode-se obter os objetos da classe ***Departamento*** a ele associados pela associação ***trabalha*** (no exemplo, apenas *Financeiro*).

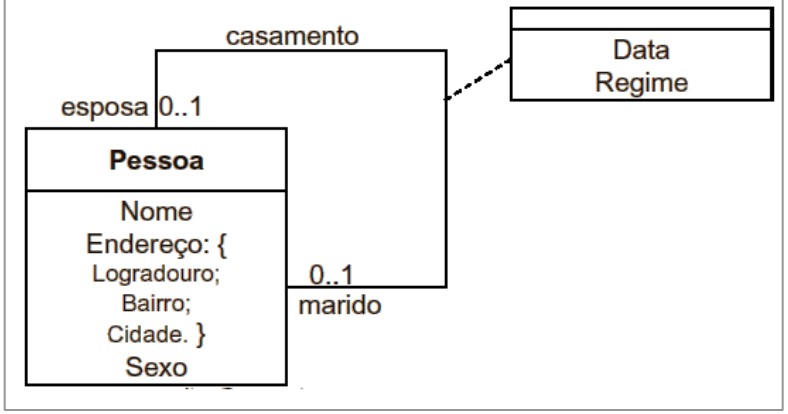
Mas NÃO o contrário (não é possível obter *João* a partir de *Financeiro*)

Classe de Associação



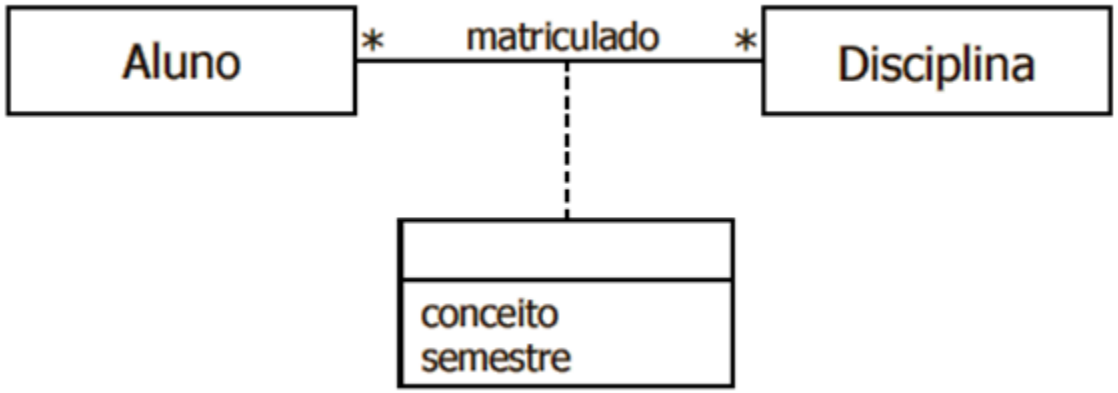
- Reúne as propriedades de associação e classe
- É necessária quando se precisa guardar atributos / operações relativas à associação
- Não é possível repetir combinações de objetos das classes participantes na associação

Exemplo:

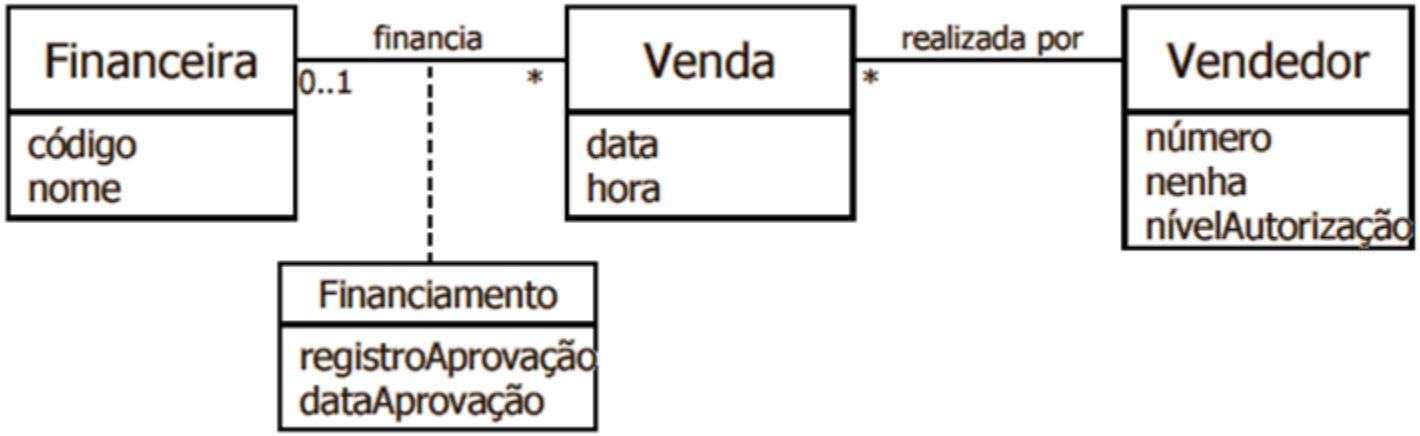


Classe de Associação - Outros Exemplos

a)

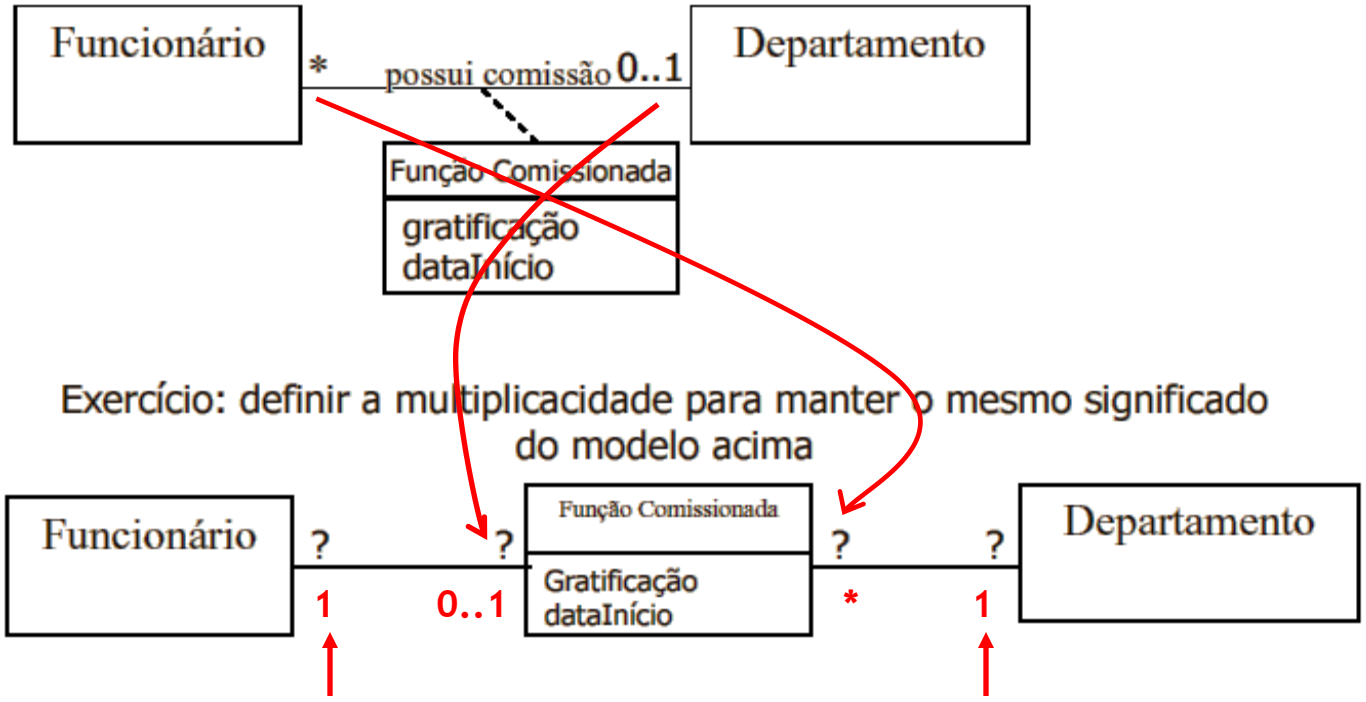


b)



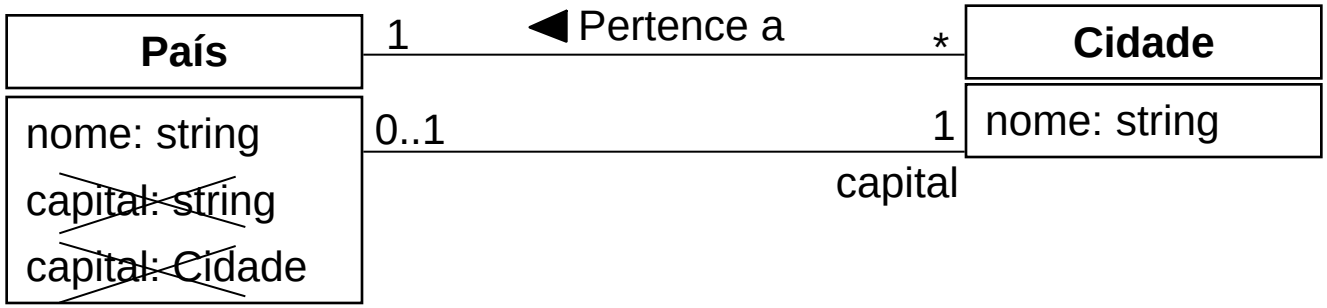
Classe de Associação – Eliminação

O conceito de classe de associação não é permitido em todas as linguagens de programação e sistemas de banco de dados OO. Por isso, é comum que as classes de associação, descobertas na etapa de análise do sistema, sejam transformadas em classes regulares na etapa de projeto.

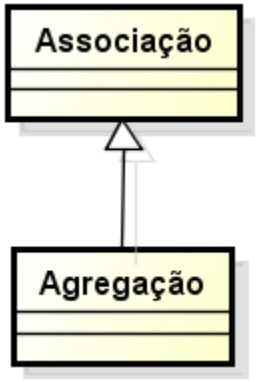


Atributos *versus* Associações

- Uma propriedade que designa um objeto de uma classe presente no modelo, deve ser modelada como uma associação e não como um atributo
- Exemplo:

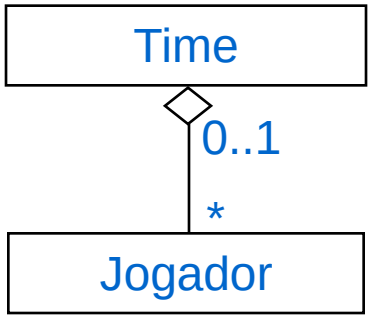


Agregação



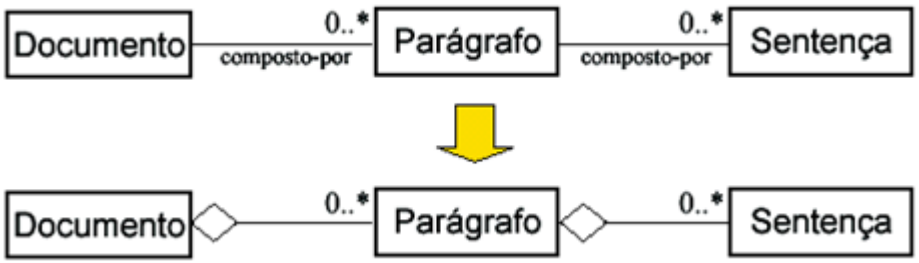
- Tipo especial de associação
- É uma associação que relaciona o *todo* às suas *partes*, com o significado de que o todo contém as (ou é constituído pelas) partes
- Semântica: ação realizada sobre o todo atinge as partes
- Exemplos:

a)



- Um time **contém** *0 ou mais* jogadores
- Um jogador **faz parte de** *um* time (num dado momento), mas também pode estar desempregado

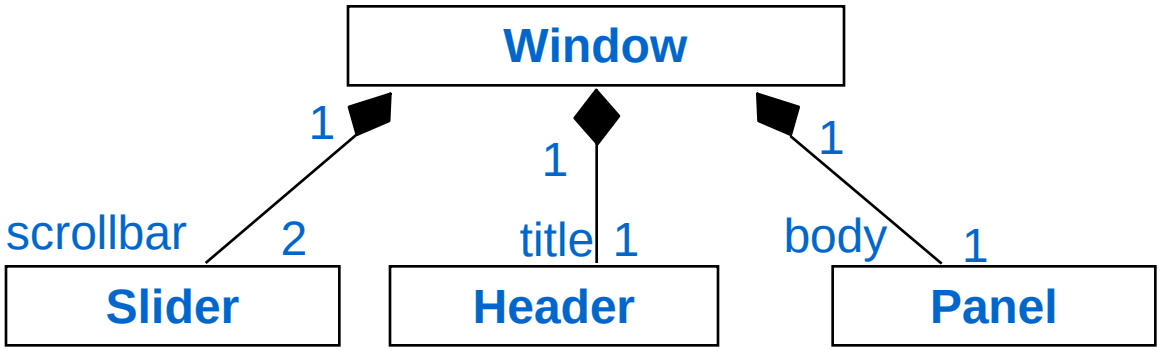
b)

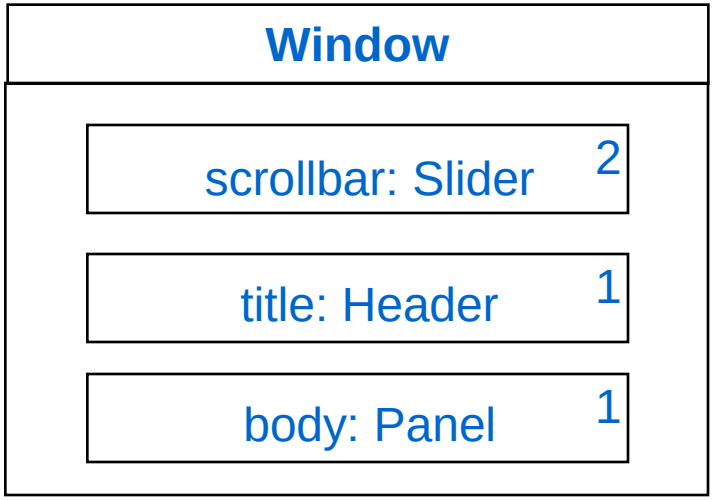


Composição

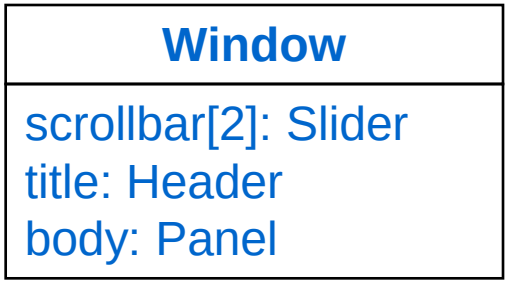
- É um tipo especial de agregação
- Forma mais forte de agregação aplicável quando existe um forte grau de pertença das partes ao todo, representado por:
  - Cada parte só pode fazer parte de um todo (i.e., a multiplicidade do lado do todo não excede 1)
  - O todo e as partes têm tempo de vida coincidente, ou, pelo menos, as partes nascem e morrem dentro de um todo
  - a eliminação do todo propaga-se para as partes, em cascata
- Notação: losango cheio (◆) ou notação encaixada (vide próximo slide)

Composição: notações alternativas





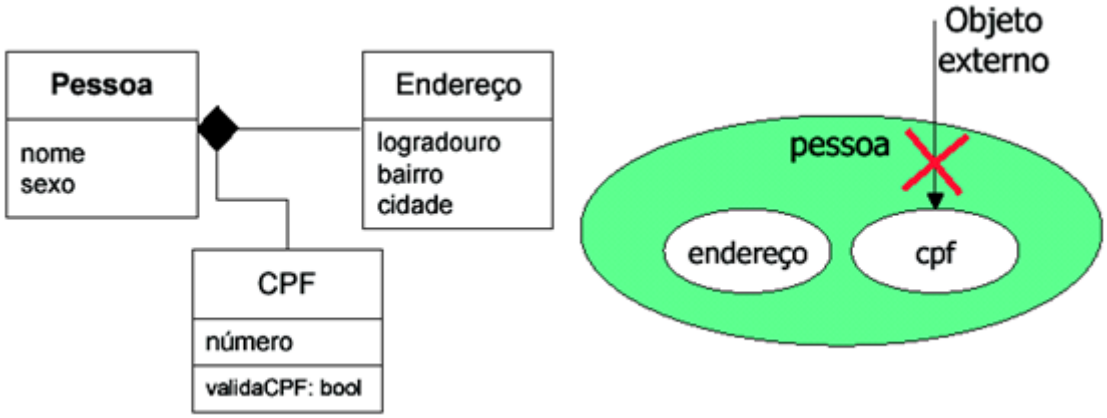
(sub-objetos no compartimento dos atributos)



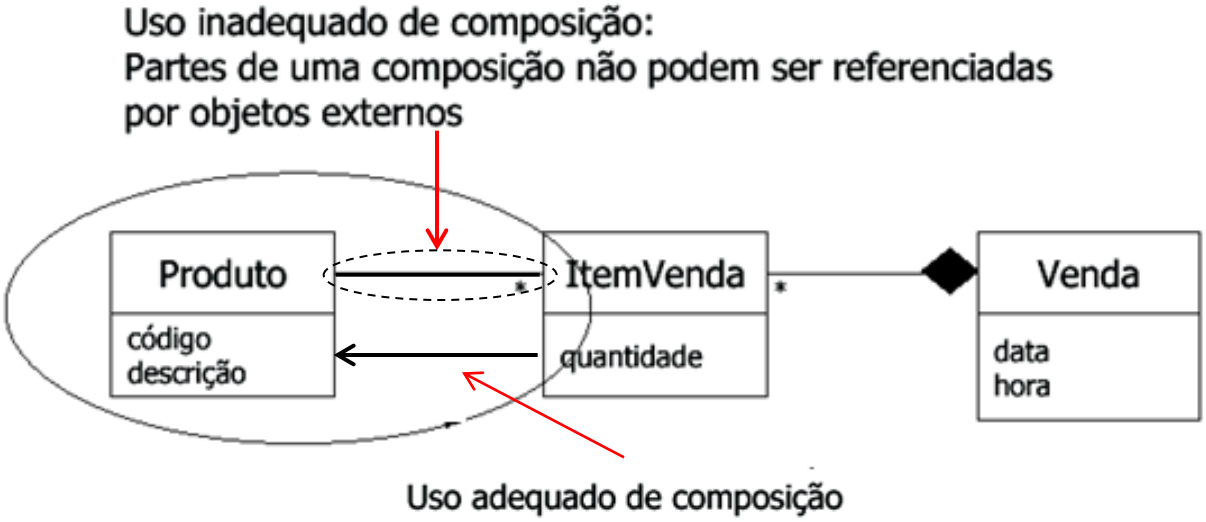
Composição (cont.)

Em geral, o acesso às *partes* é restrito ao *todo*. Exemplos:

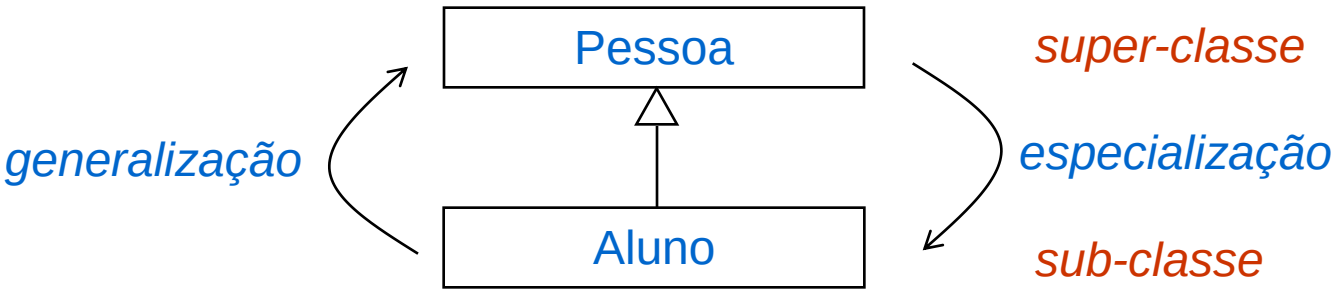
a)



b)

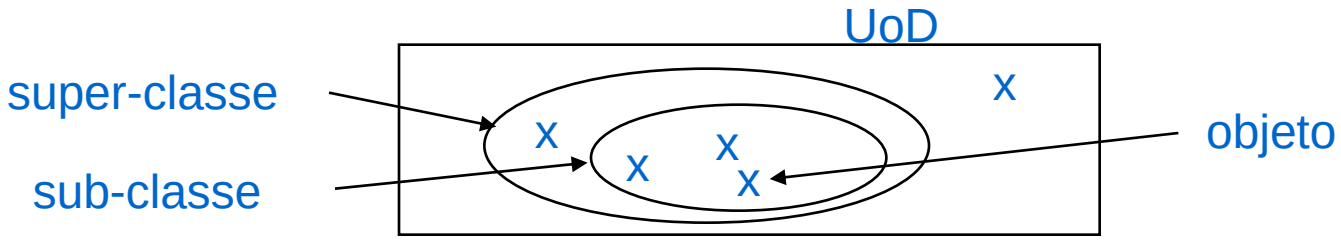


Generalização



- **Relação semântica "is a" ("é um" / "é uma") :** um aluno é uma pessoa

- **Relação de inclusão nas extensões das classes:**



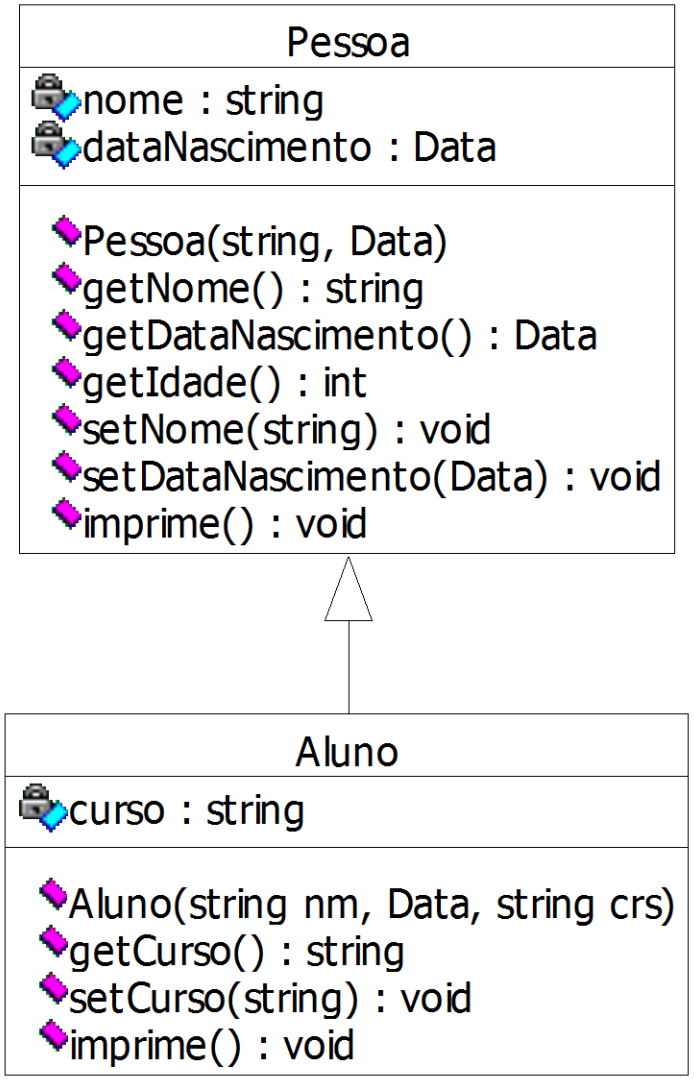
extensão (sub-classe) $\subset$ extensão (super-classe)

- **Relação de herança nas propriedades:** A sub-classe herda as propriedades (atributos, operações e relações) da super-classe, podendo acrescentar outras

As três facetas da generalização

- Substitutibilidade
  - onde se espera um objeto da super-classe pode-se passar um objeto de uma subclasse
- Herança de interface
  - a subclasse herda as assinaturas (e significados) das operações da super-classe
- Herança ou *overriding* de implementação
  - a subclasse pode herdar as implementações das operações da super-classe, mas também pode ter novas implementações de algumas dessas operações
  - quando em UML se repete numa subclasse a assinatura de uma operação da super-classe, quer dizer que tem uma nova implementação na subclasse

Exemplo em C++



Ferramenta: Microsoft Visual Modeler

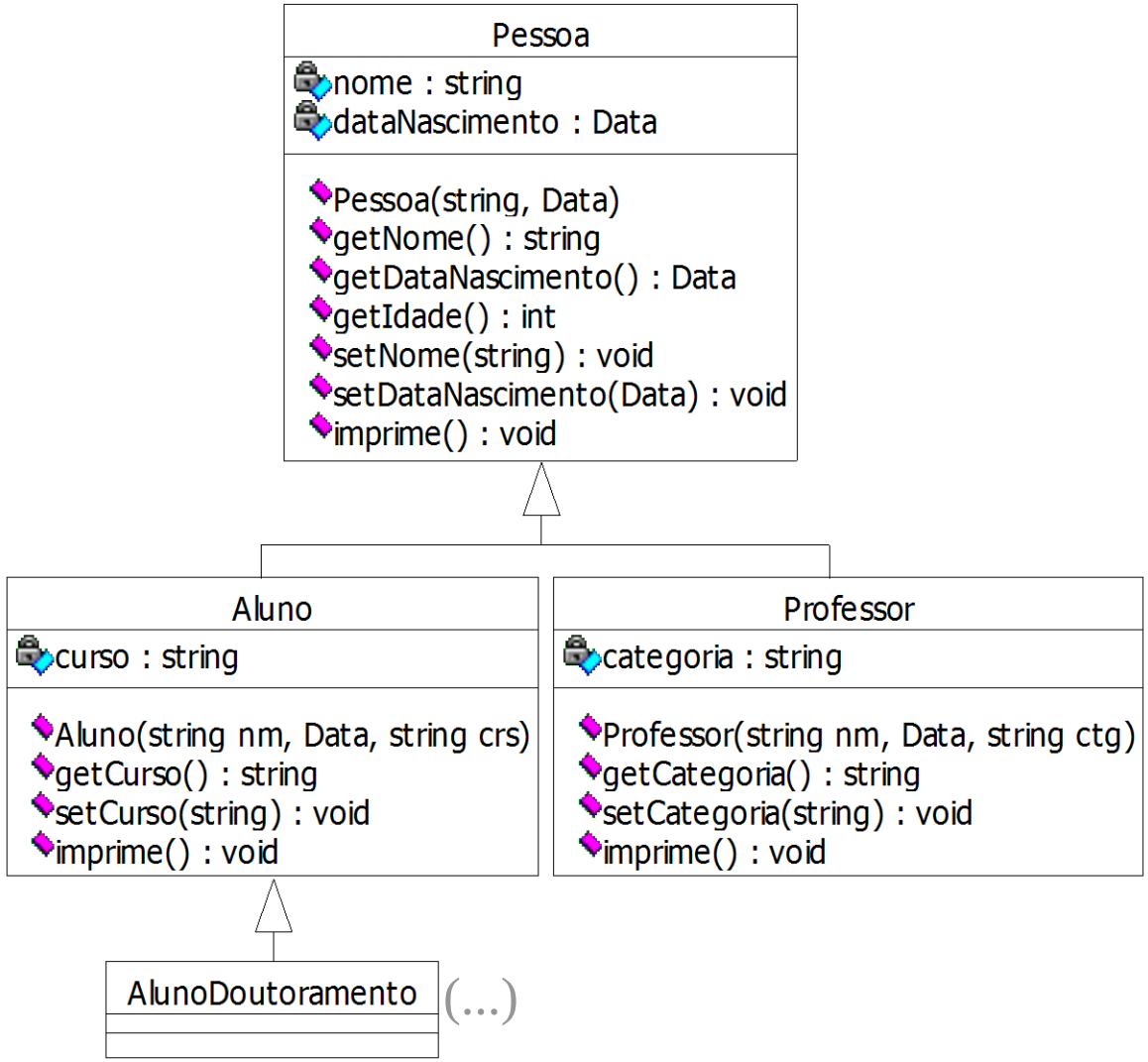
```
class Pessoa {
  private:
    string nome;
    Data dataNascimento;
   public:
     Pessoa(string, Data);
     string getNome() const;
     Data getDataNascimento() const;
     int getIdade() const;
     void setNome(string);
     void setDataNascimento(Data);
     virtual void imprime() const;
  };

class Aluno : public Pessoa {
  private:
    string curso;
  public:
    Aluno(string nm, Data, string
  crs);
    string getCurso() const;
    void setCurso(string);
    virtual void imprime() const;
  };
```

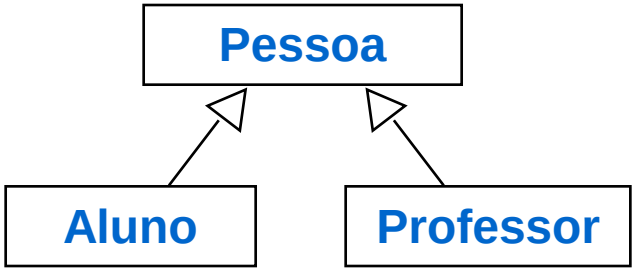
Hierarquias de classes

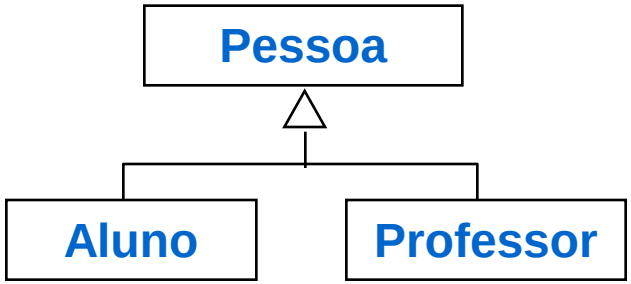
- Em geral, pode-se ter uma hierarquia de classes relacionadas por herança / generalização
  - Em cada classe da hierarquia colocam-se as propriedades que são comuns a todas as suas subclasses

  ⇒ **Evita-se redundância, promove-se reutilização!**

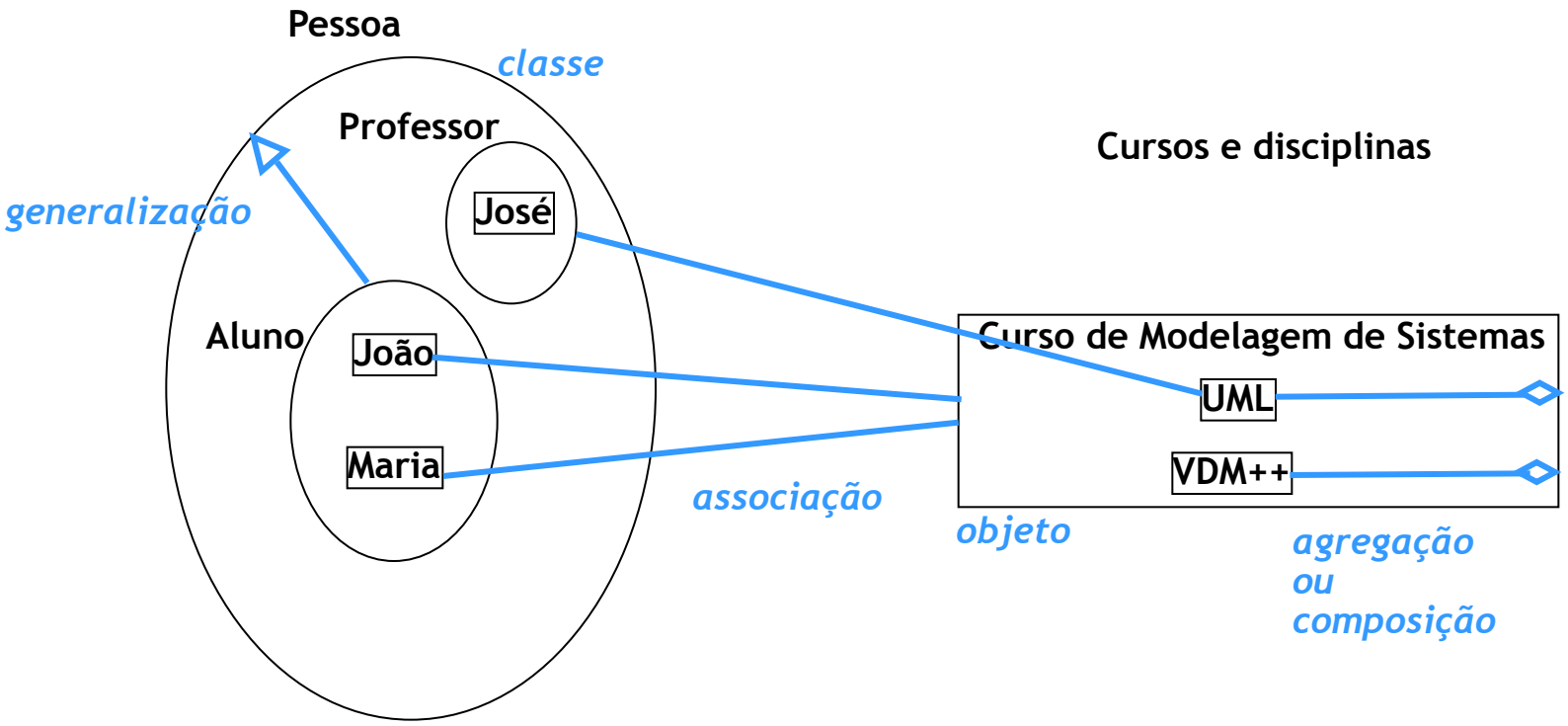


Notações alternativas para hierarquias de classes





Associação *versus* Agregação / Composição *versus* Generalização



Exemplo: meta-modelo parcial

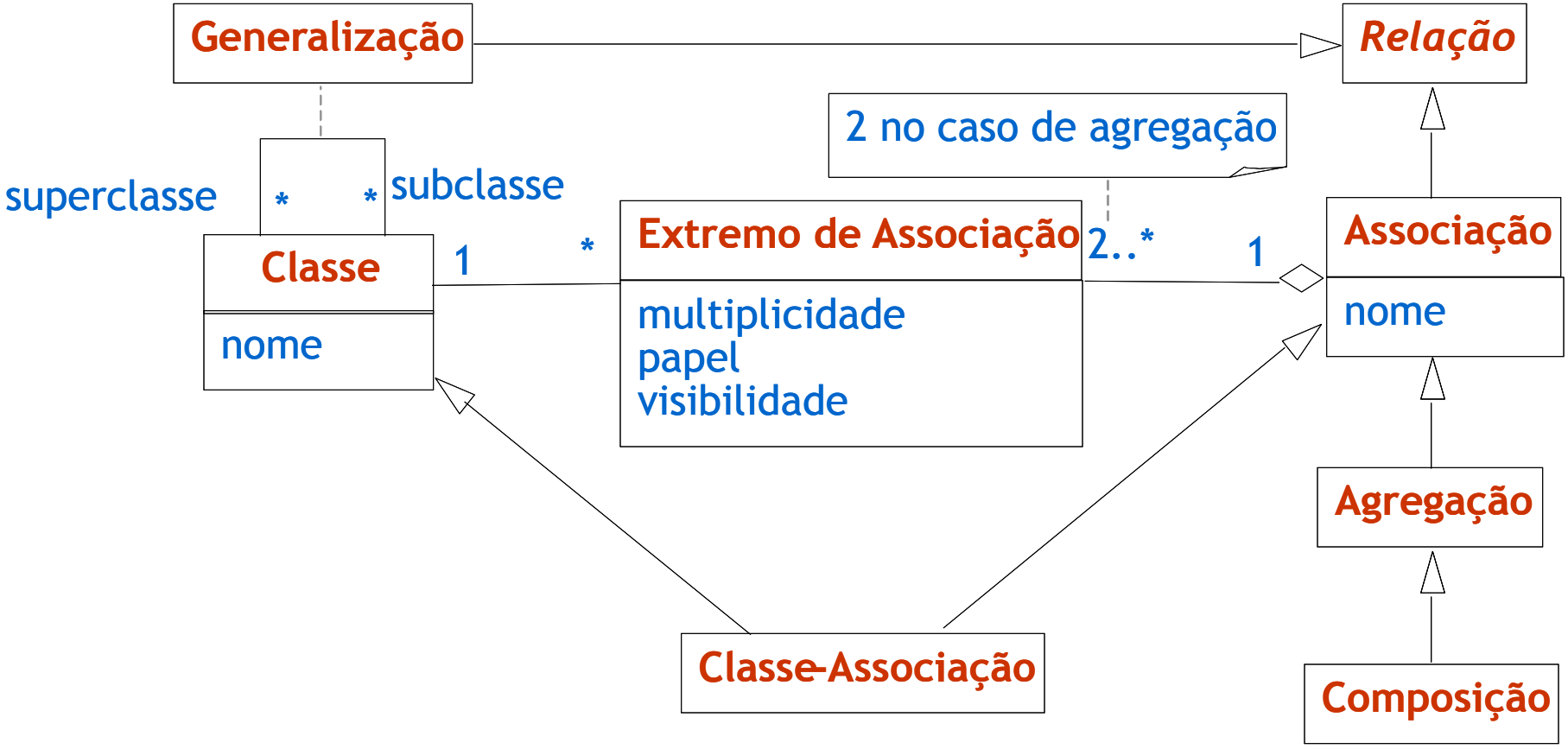
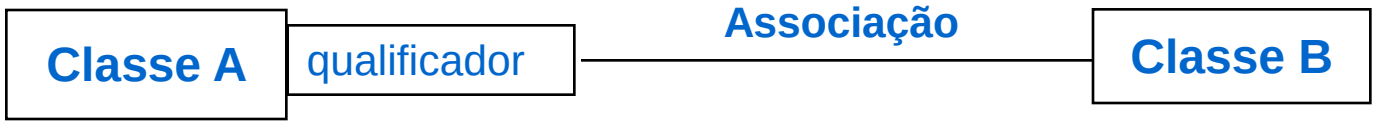


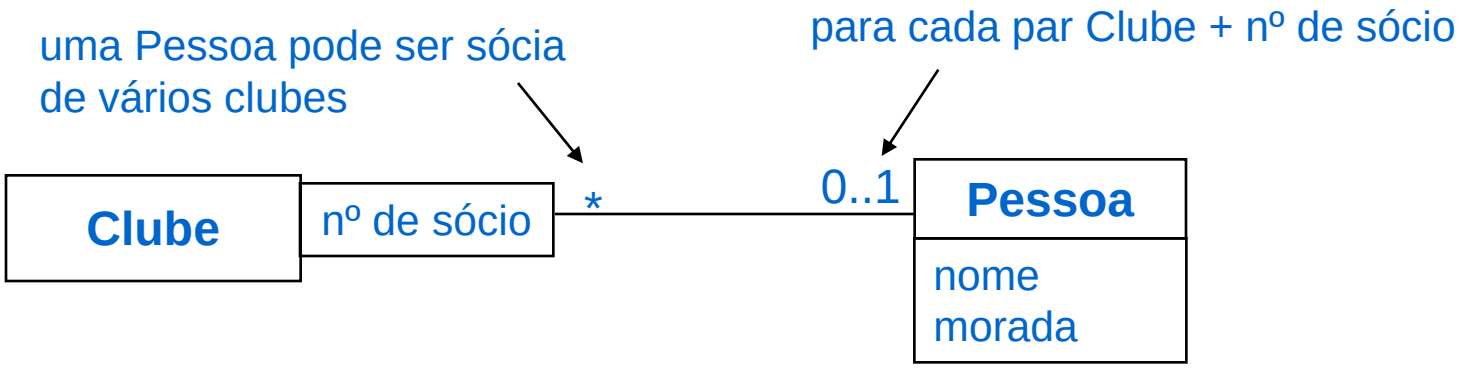
Diagrama de Classes de Objetos

Parte 2 - Conceitos Avançados

*Associações qualificadas

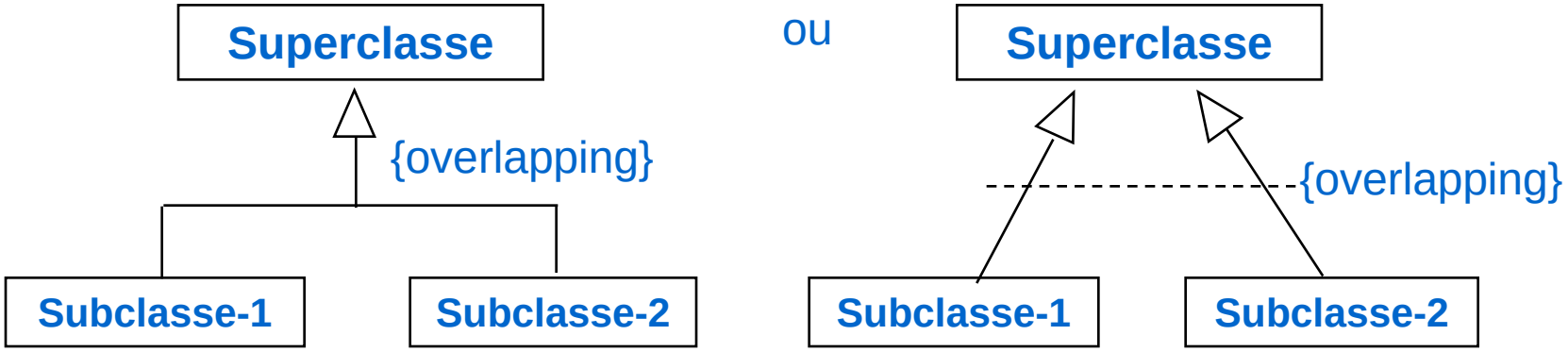


- Qualificador: lista de um ou mais atributos de uma associação utilizados para navegar de A para B
- "Chave de acesso" a B (acesso a um objeto ou conjunto de objetos) a partir de um objeto de A



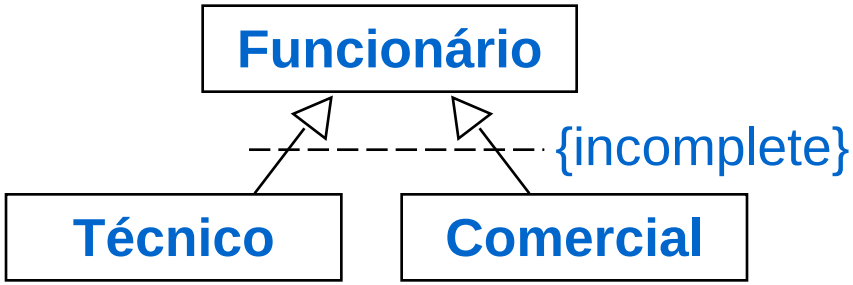
Subclasses sobrepostas (≠disjuntas)

- caso em que um objeto da superclasse pode pertencer simultaneamente a mais do que uma subclasse
- indicado por restrição {overlapping}
- o contrário é {disjoint} (situação por omissão?)



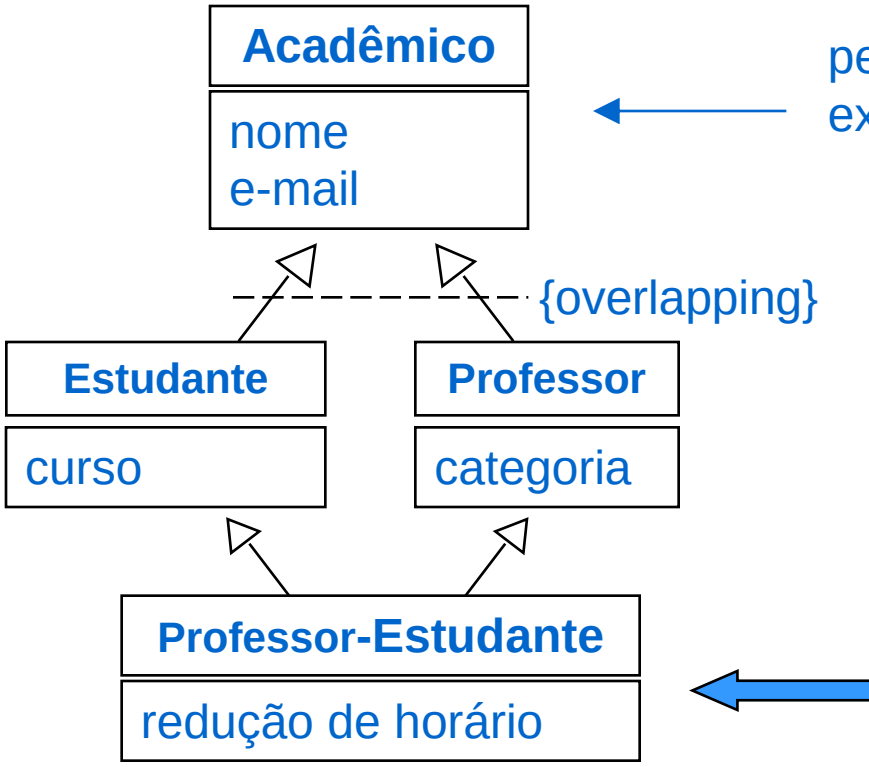
Subclasses incompletas (≠completas)

- caso em que um objeto da superclasse pode não pertencer a nenhuma das subclasses
- indicado por restrição {incomplete}
- o contrário é {complete} (situação por omissão?)



Herança múltipla (≠simples)

- ocorre numa subclasse com múltiplas super-classes
- geralmente suportada por linguagens de programação OO



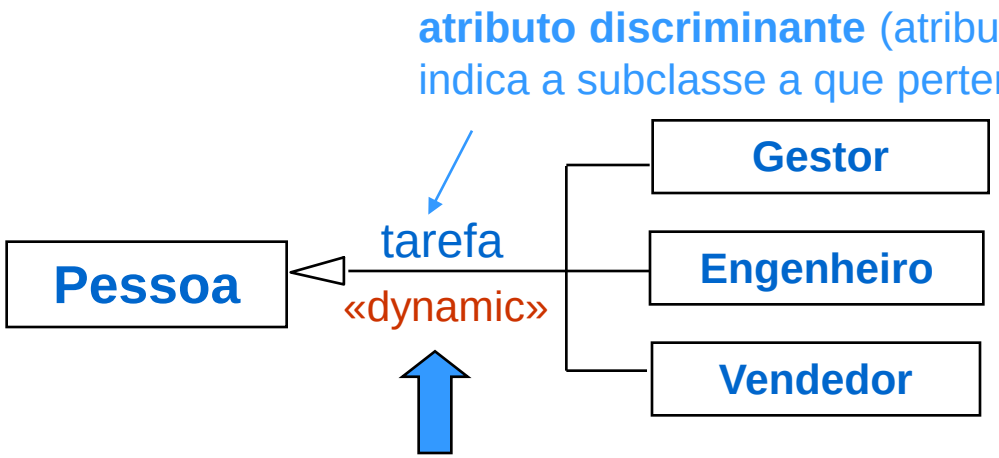
pelo menos conceitualmente, existe uma super-classe comum

Só faz sentido assim!

herda propriedades de Estudante e Professor e, indiretamente, de Acadêmico (uma única vez!)

Classificação dinâmica (≠estática)

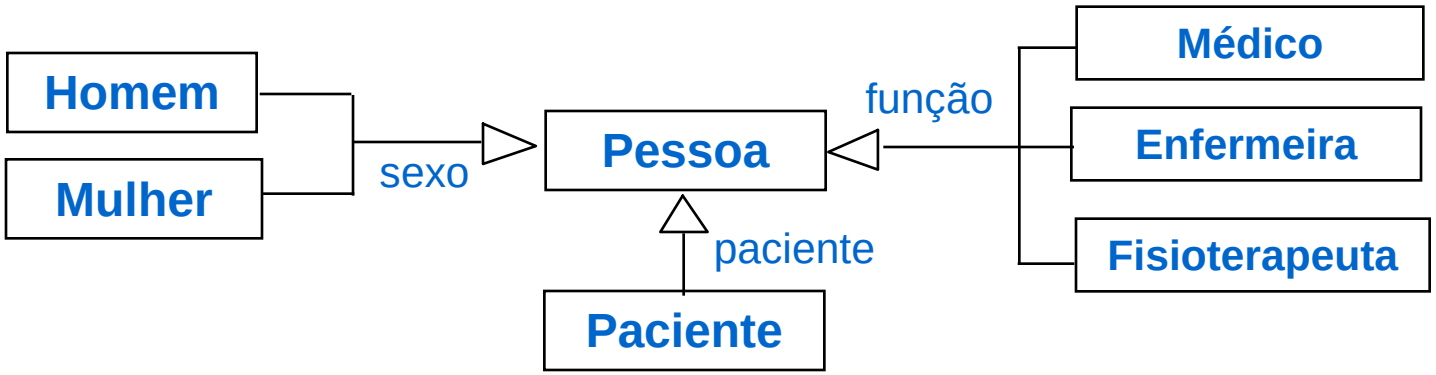
- classificar objeto: atribuir classe a objeto
- caso em que a(s) classe(s) a que um objeto pertence pode(m) variar ao longo da vida do objeto
- indicado por estereótipo «dynamic»
- geralmente não suportada por linguagens de programação OO



Uma pessoa que num dado momento tem a tarefa de gestor, pode em outro momento ter a tarefa de vendedor etc.

Classificação múltipla (≠simples)

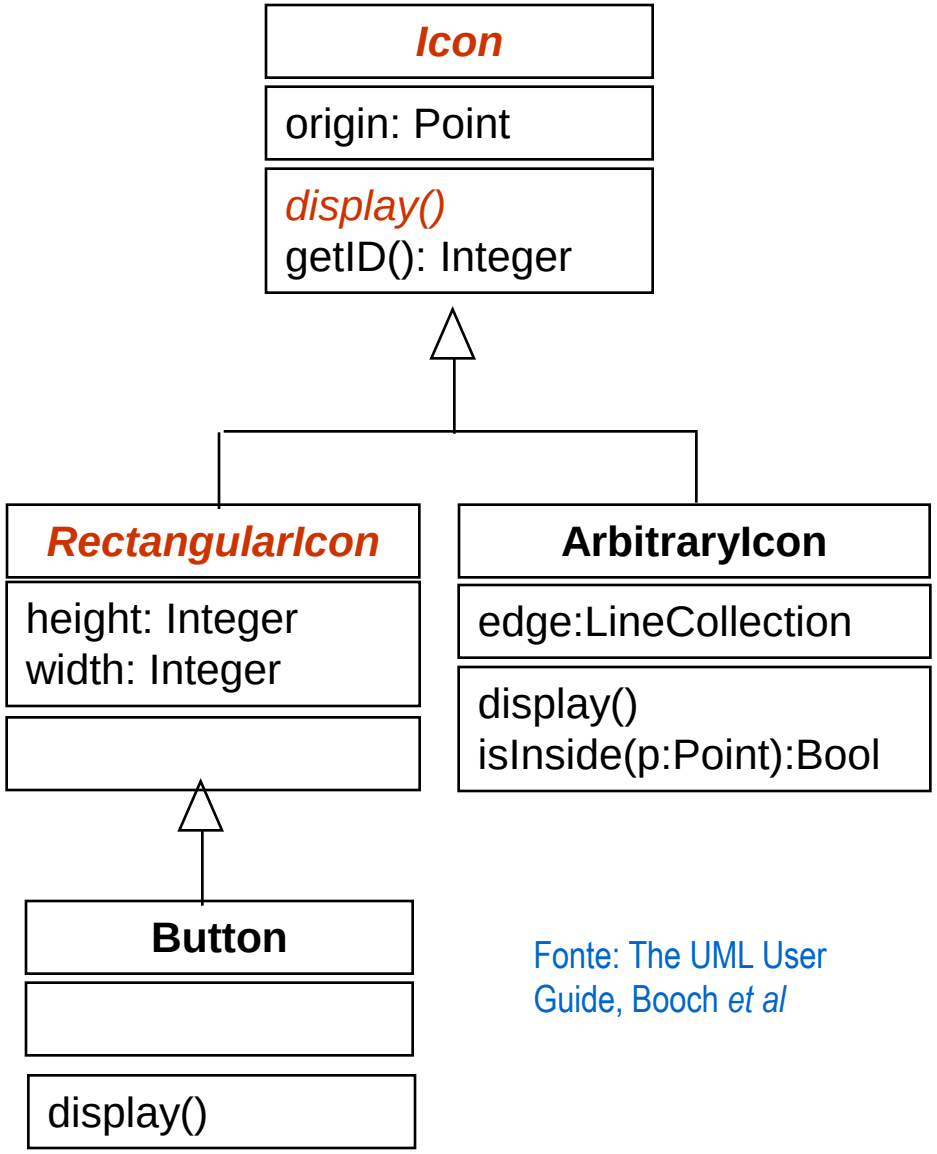
- caso em que um objeto pode pertencer num dado momento a várias classes, sem que exista uma subclasse que represente a intersecção dessas classes (com herança múltipla)
- geralmente não suportado pelas linguagens de programação OO (pode então ser simulada por agregação de papéis)



exemplo de combinação legal: {Mulher, Paciente, Enfermeira}

Classes e operações abstratas (≠concretas)

- Classe abstrata: classe que não pode ter instâncias diretas
  - pode ter instâncias indiretas pelas subclasses concretas
- Operação abstrata: operação com implementação a definir nas subclasses
  - uma classe com operações abstratas tem de ser abstrata
  - função virtual pura em C++
- Notação : nome em itálico ou propriedade {abstract}

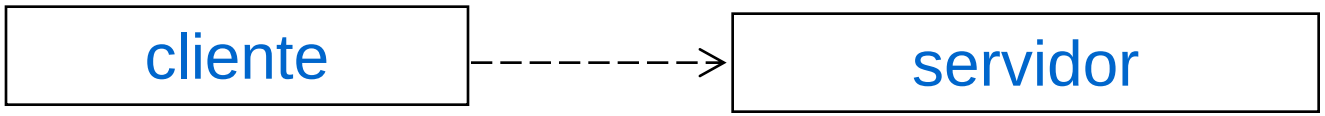


Fonte: The UML User Guide, Booch *et al*

Outras relações entre classes: dependência e concretização

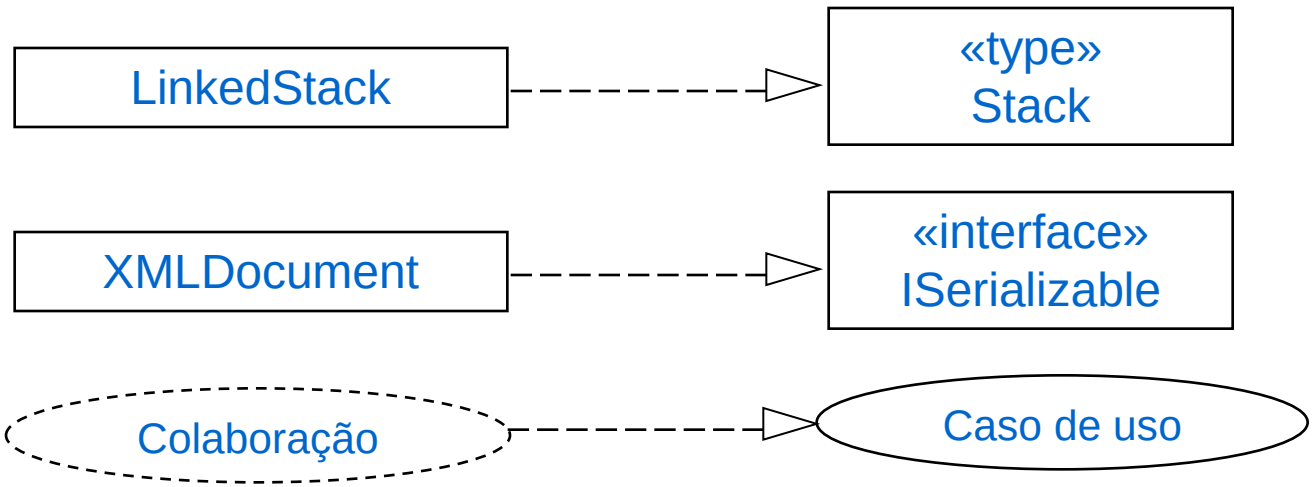
Relação de dependência

- Relação de uso entre dois elementos (classes, componentes etc.), em que uma mudança na especificação do elemento usado pode afetar o elemento utilizador
- Exemplo típico: classe-1 que depende de outra classe-2 porque usa operações ou definições da classe-2
- Úteis para gestão de dependências



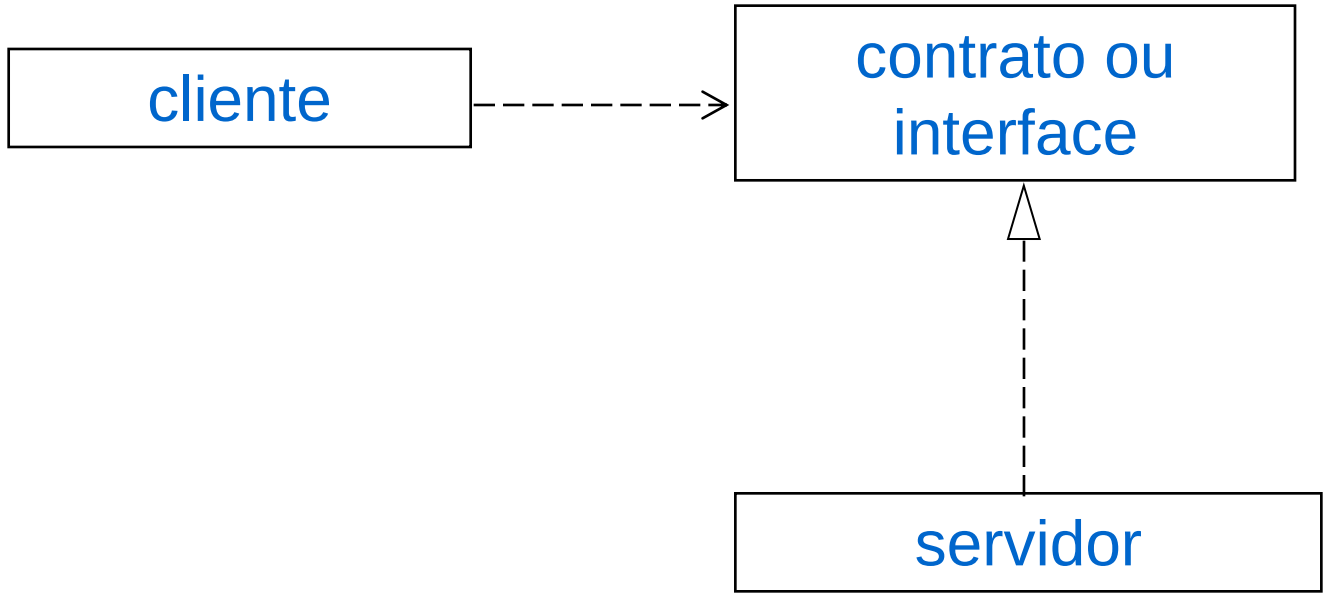
Relação de concretização (*realization*)

- Relação entre elementos de diferentes níveis de abstração, isto é, entre um elemento mais abstrato (que especifica uma interface ou um "contrato", entre clientes e implementadores) e um elemento correspondente mais concreto (que implementa esse contrato)
- Difere da generalização porque há apenas herança de interface e não herança de implementação



Dependência e concretização

- Aparecem freqüentemente combinados
- Cliente usa o servidor sem dele depender diretamente (depende apenas da interface ou contrato que o servidor implementa)

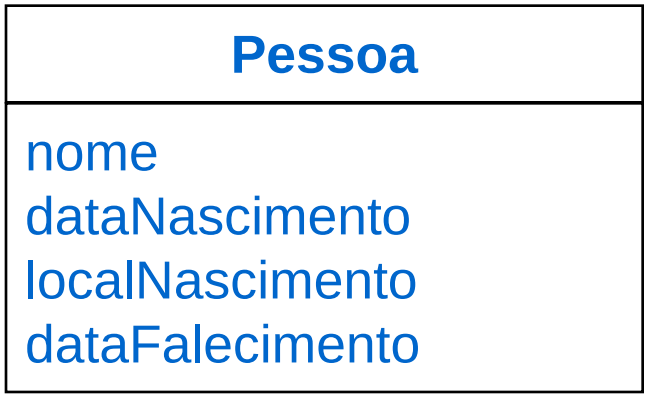


Restrições, elementos derivados, pré-condições e pós-condições

Restrições

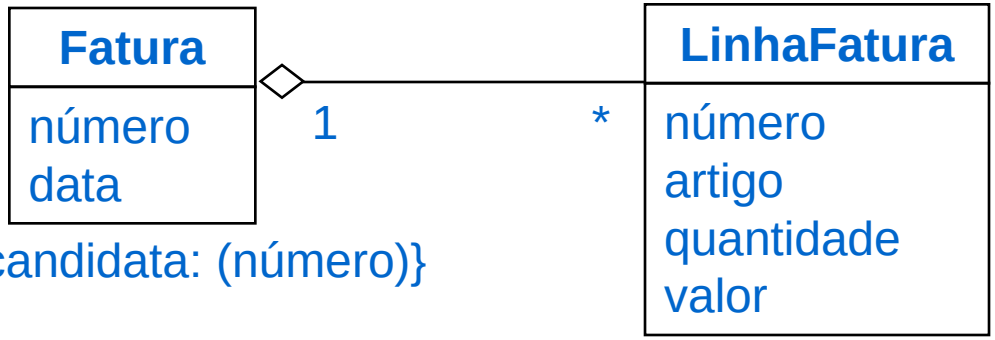
- Uma restrição especifica uma condição que tem de se verificar no estado do sistema (objetos e ligações)
- Uma restrição é indicada por uma expressão ou texto entre chaves ou por uma nota posicionada junto aos elementos a que diz respeito, ou a eles ligada por linhas a traço interrompido (sem setas, para não confundir com relação de dependência)
- Podem ser formalizadas em UML com a OCL - "*Object Constraint Language*"
- Também podem ser formalizadas (como invariantes) numa linguagem de especificação formal como VDM++

Restrições em classes



{chave candidata: (nome, dataNascimento, localNascimento)}

{dataFalecimento > dataNascimento}

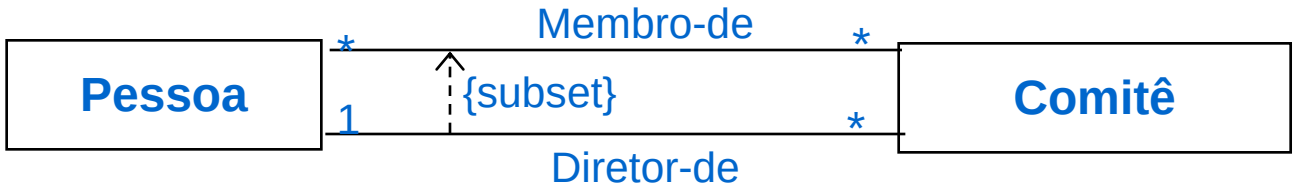


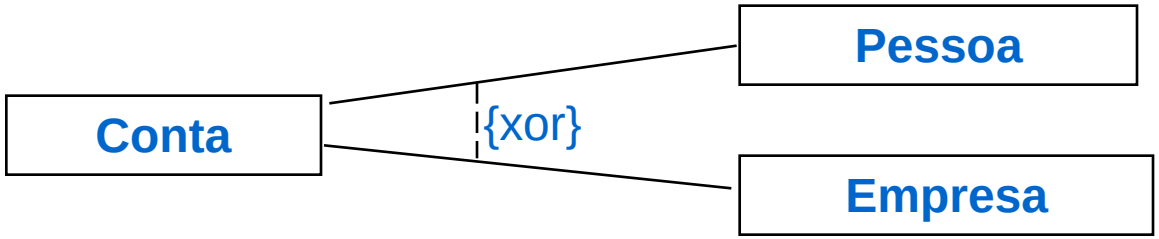
{chave candidata: (número)}

{chave candidata: (fatura, número)}

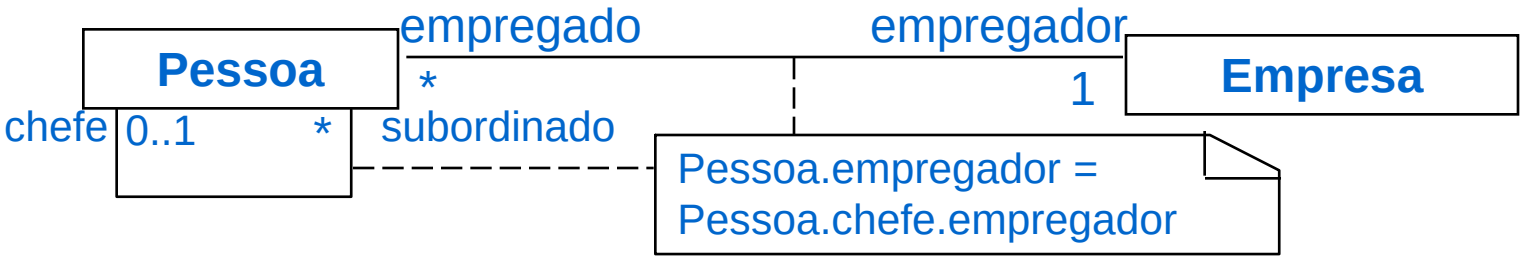
Restrições em associações

uma fatura é constituída por um conjunto ordenado de 0 ou mais linhas





associações mutuamente exclusivas



Exercício: Considerando o diagrama de classes abaixo, marque a alternativa correta:

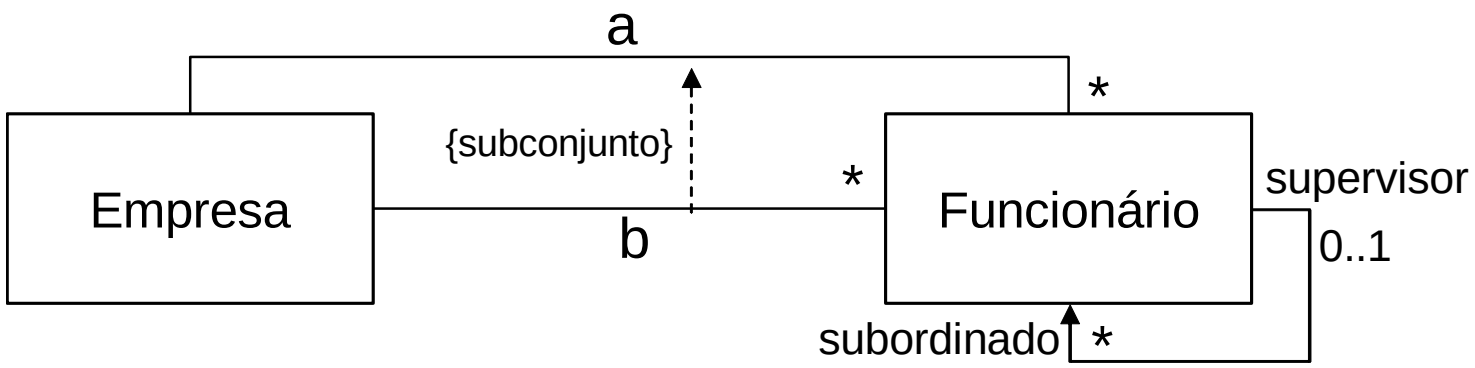
a) Para um objeto qualquer de Empresa, pode-se afirmar que a quantidade de objetos de Funcionário que participam da associação a é sempre inferior à quantidade de objetos que participam de b

b) Não existem objetos de Funcionário que participam simultaneamente da associação a e b com um objeto de Empresa

c) Pode existir um objeto de Empresa que não esteja associado a qualquer objeto de Funcionário, seja pela associação a ou b.

d) Um objeto de Funcionário pode estar associado com mais de um objeto de Empresa

e) A associação de subordinação garante que um supervisor está associado com subordinados que estão na mesma Empresa



Exercício: Considere o diagrama de classes abaixo e marque a alternativa correta:

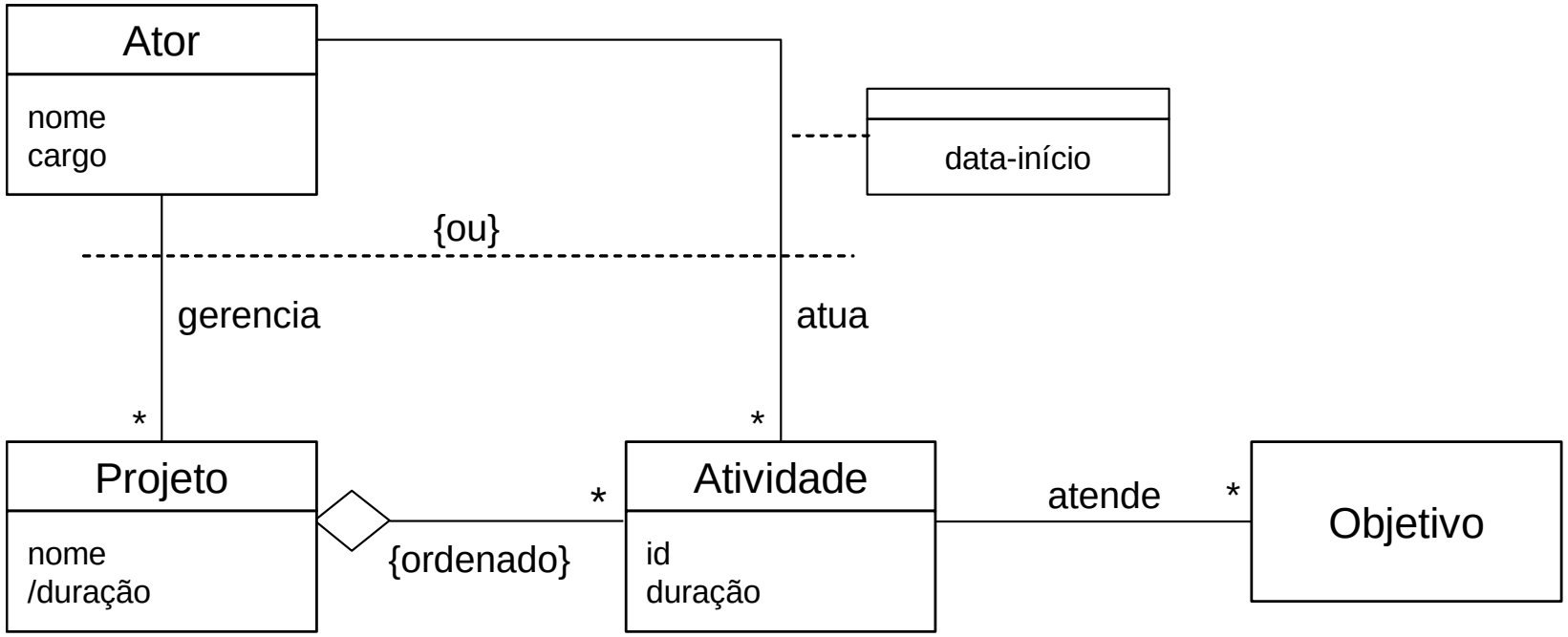
a) **Para cada (objeto de) Atividade existe somente um único valor de data-início.**

b) **Um (objeto de) Ator pode simultaneamente gerenciar vários projetos e atuar em várias atividades**

c) **Um Objetivo pode ser atendido por vários objetos de Atividade**

d) **Em um Projeto sua duração é determinada pela duração de suas atividades**

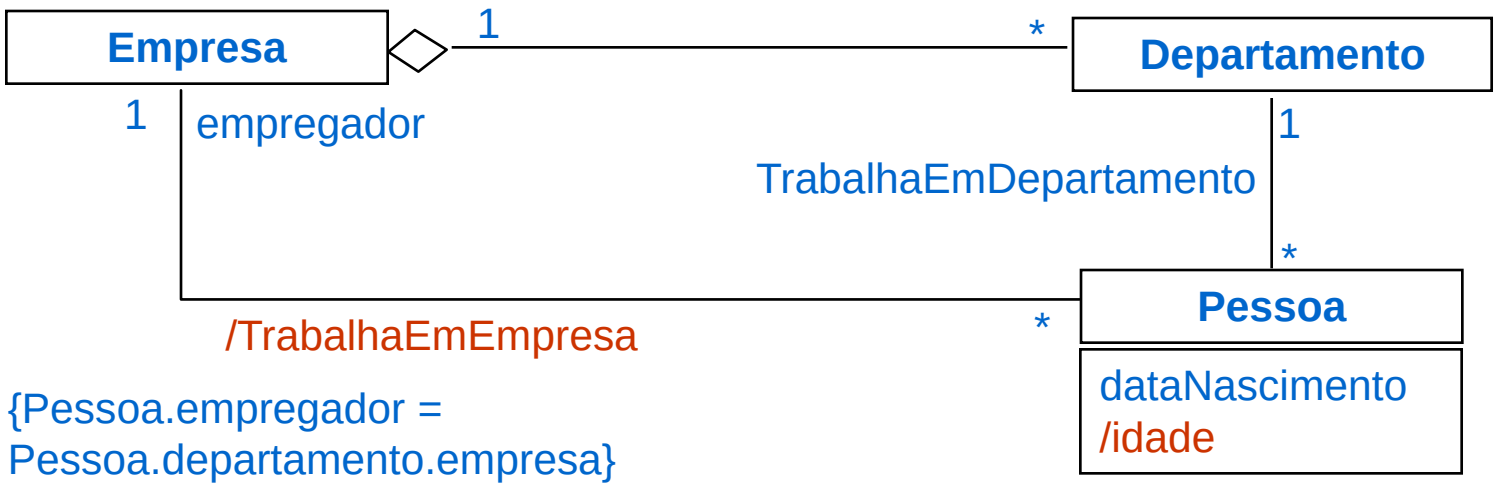
e) **Os objetos de Atividade são ordenados pelo valor do atributo id**

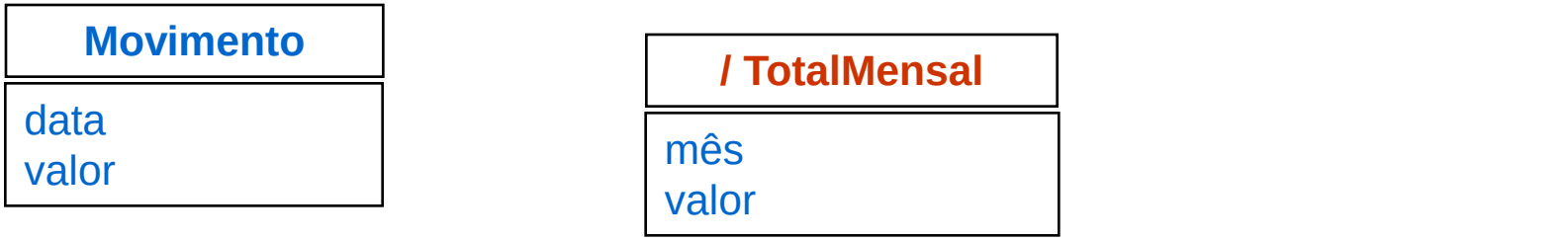


Elementos derivados

- Elemento derivado (atributo, associação ou classe): elemento calculado em função de outros elementos do modelo
- Notação: barra “/” antes do nome do elemento derivado
- Um elemento derivado tem normalmente associada uma restrição que o relaciona com os outros elementos

Exemplo de elementos derivados



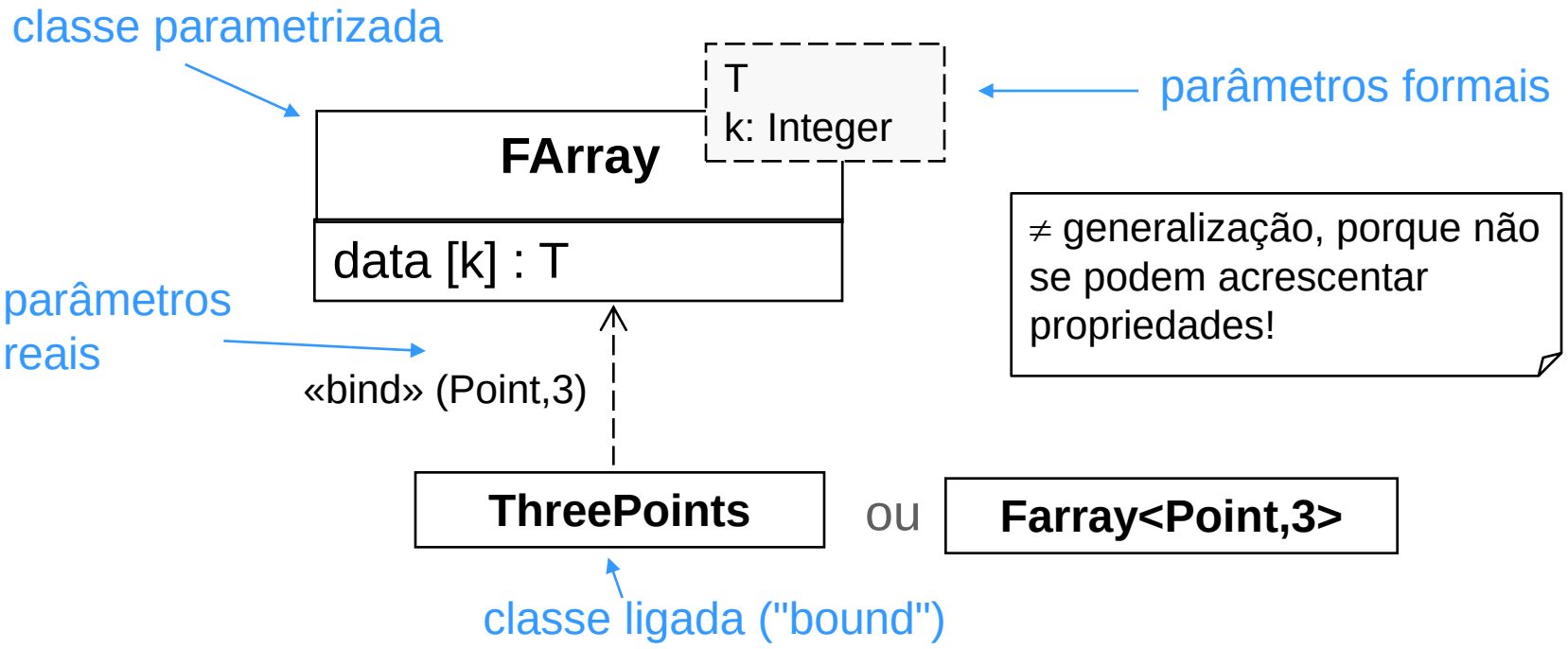


* Pré-condições e pós-condições

- Semântica de operações pode ser captada através de pré-condições e pós-condições
- Pré-condição: uma condição nos argumentos e estado inicial do objeto, verificada na chamada (início) da operação
- Pós-condição: uma condição nos argumentos, estado inicial do objeto, estado final do objeto e valor retornado pela operação, verificada no retorno (fim) da operação
- Podem ser expressas em UML através da OCL (*Object Constraint Language*)
- Também podem ser expressas numa linguagem de especificação formal, como VDM++
  - VDM++ tem a vantagem de permitir também implementar as operações e executar as pré-condições e pós-condições

Classes especiais: classes parametrizadas, interfaces, tipos, meta-classes, utilitários

* Classes parametrizadas (*templates*)



C++

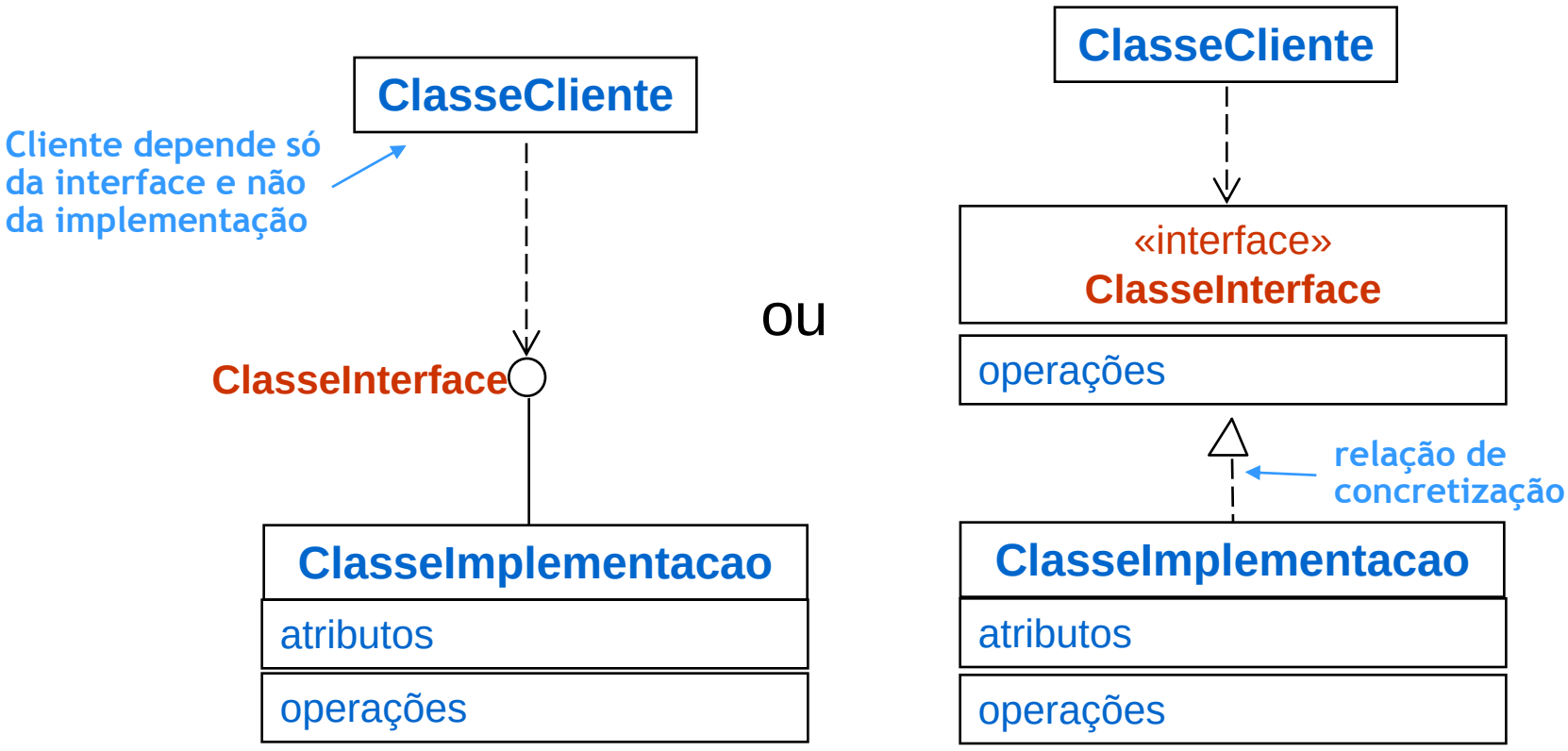
```
template <class T, int k>
class FArray { public: T[k] data; };

typedef Farray<Point,3> ThreePoints;
```

Interfaces

- Uma interface especifica um conjunto de operações (com sintaxe e semântica) externamente visíveis de uma classe de (ou componente, subsistema etc.)
  - semelhante a classe abstrata só com operações abstratas e sem atributos nem associações
  - separação mais explícita entre interface e (classes de) implementação
  - interfaces são mais importantes em linguagens como Java, C# e VB.NET que têm herança simples de implementação e herança múltipla de interface
- Relação de concretização de muitos parta muitos entre interfaces e classes de implementação
- Vantagem em separar interface de implementação: os clientes de uma classe podem ficar dependentes apenas da interface em vez da classe de implementação
- Notação: classe com estereótipo «interface» (ligada por relação de concretização à classe de implementação) ou círculo (ligado por linha simples à classe de implementação)

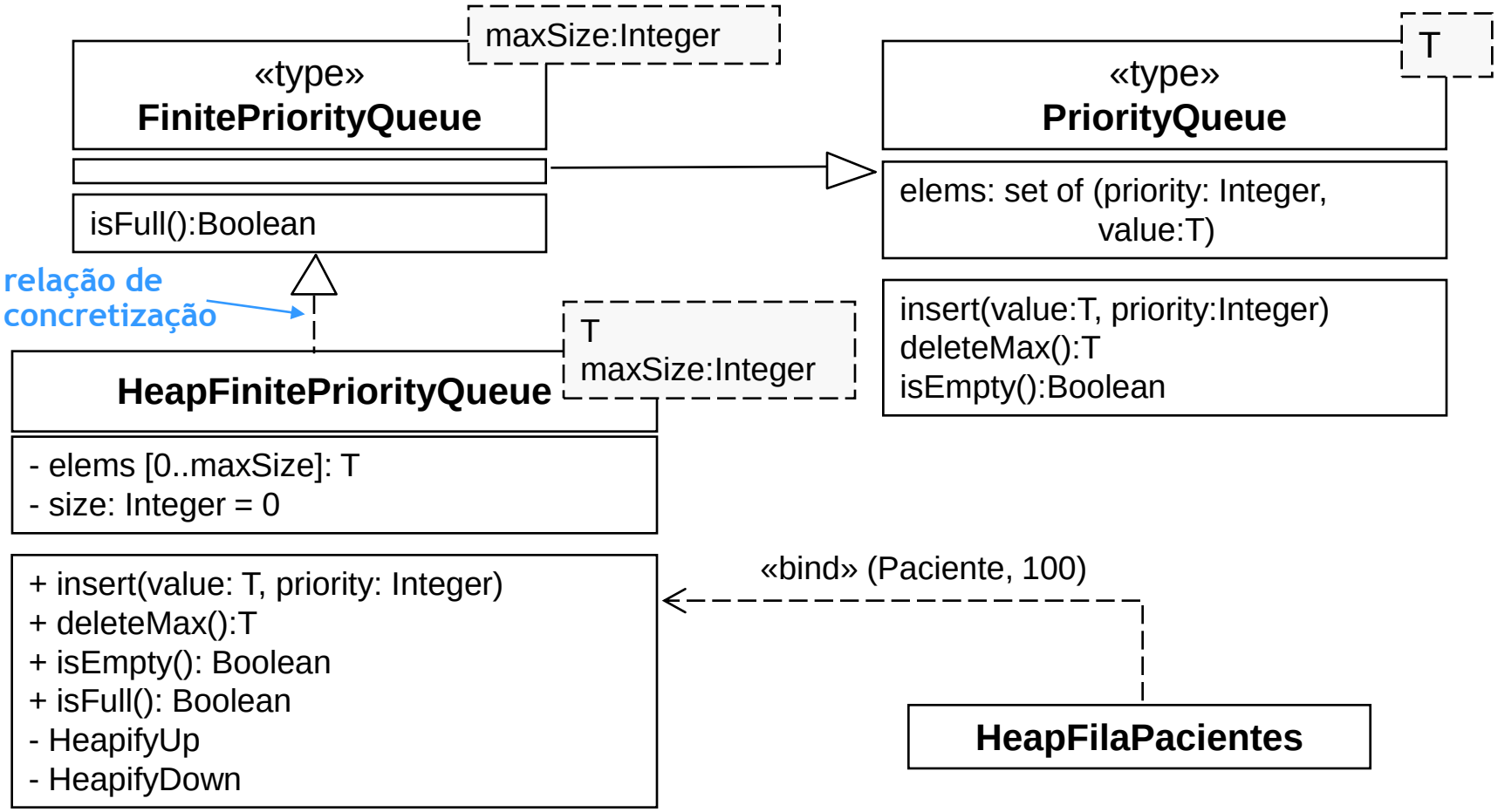
Interfaces: notações alternativas



* Tipos

- Um *tipo* é usado para especificar um domínio de objetos em conjunto com as operações aplicáveis a esses objetos, sem especificar a implementação física desses objetos
  - não pode incluir métodos (implementação de operações)
  - pode ter atributos e associações (abstratos!), mas apenas para especificar o comportamento das operações, sem compromisso de implementação
  - notação: classe com estereótipo «type»
  - semelhante a **tipo abstrato de dados**
- Uma *classe de implementação* (classe normal) define a estrutura física de dados (para atributos e associações) e métodos de um objeto tal como é implementado numa linguagem tradicional
  - notação: classe normal ou classe com estereótipo «implementationClass»
  - diz-se que uma classe de implementação concretiza ("realizes") um tipo se proporciona todas as operações definidos no tipo, com o mesmo comportamento especificado no tipo

* Tipos: exemplo

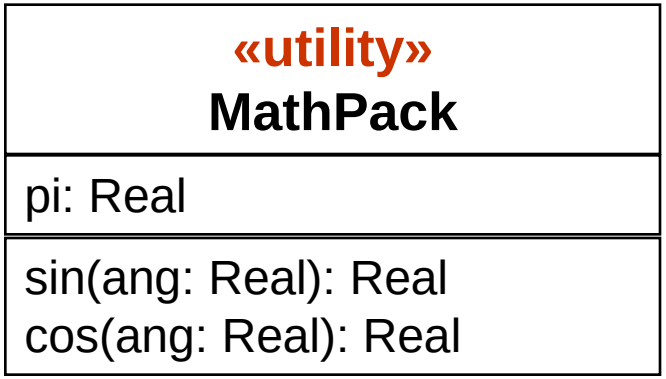


* Meta-classes

- Uma meta-classe é uma classe cujas instâncias são classes
- Notação: classe com estereótipo «metaclass»
- Usadas geralmente em meta-modelos
- Relação de instanciação (entre classe e meta-classe) pode ser indicada por dependência com estereótipo «instanceOf»

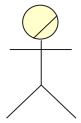
* Utilitários

- Um utilitário é um agrupamento de variáveis globais e procedimentos como classe
- Pode ser implementado por classe em que todos os atributos e operações são estáticos
- Notação: classe com estereótipo «utility»



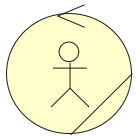
* Estereótipos de classes em modelos de negócio

Um modelo de negócio descreve a implementação dos processos de negócio (de fronteira) e de suporte (internos) através de colaborações entre objetos dos seguintes tipos:



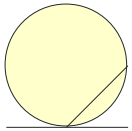
business actor

ator em relação ao negócio (cliente ou outra entidade ou sistema que interage com o negócio);
ator externo



business worker

perfil de trabalhador do negócio, interno (não interage com atores do negócio) ou de fronteira (interage com atores do negócio);
ator interno



business entity

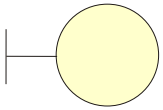
objeto passivo manipulado pelos trabalhadores e atores nas atividades dos processos de negócio

* Estereótipos de classes em modelos de análise

Um modelo de análise descreve implementações "ideais" dos casos de uso de um sistema de software, com vista a melhor compreender os seus requisitos, através de colaborações entre objetos dos seguintes tipos:

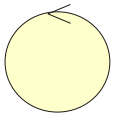
actor

ator em relação ao sistema de software (pode ser interno ou externo ao negócio)



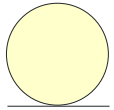
boundary

objeto de fronteira (formulário, janela etc.)



control

objeto de controle (controla sequência de funcionamento, transações etc.; estabelece ligação entre objetos de fronteira e entidades)



entity

objeto passivo que guarda estado mais ou menos persistente