



# Relacionamento entre classes

Prof. Renato Pimentel

2024/2



## Sumário



### 1 Relacionamento entre classes

Antes de apresentarmos relacionamentos entre classes na orientação a objetos, é importante falarmos sobre a representação visual, através do uso de UML.



## UML – Linguagem de modelagem unificada I



- **UML** (*Unified Modeling Language*) – linguagem visual muito utilizada nas etapas de **análise** e **projeto** de sistemas computacionais no paradigma de orientação a objetos
- A **UML** se tornou a linguagem padrão de projeto de software, adotada internacionalmente pela indústria de Engenharia de Software.



- UML não é uma linguagem de programação: é uma **linguagem de modelagem** utilizada para representar o sistema de software sob os seguintes parâmetros:
  - ▶ Requisitos
  - ▶ Comportamento
  - ▶ Estrutura lógica
  - ▶ Dinâmica de processos
  - ▶ Comunicação/interface com os usuários



- O objetivo da UML é fornecer múltiplas visões do sistema que se deseja modelar, representadas pelos **diagramas UML**;
- Cada diagrama analisa o sistema sob um determinado aspecto, sendo possível ter enfoques mais amplos (externos) do sistema, bem como mais específicos (internos).



- Diagrama de casos e usos
- **Diagrama de classes**
- Diagrama de objetos
- Diagrama de sequência
- Diagrama de colaboração
- Diagrama de estado
- Diagrama de atividades



## Softwares para UML



Algumas ferramentas para criação de diagramas UML:

- Rational Rose – a primeira ferramenta case a fornecer suporte UML
- Yed
- StarUML – Ferramenta OpenSource
- Dia
- Argo UML
- Microsoft Visio
- Enterprise Architect



- **Mais utilizado** da UML
- Objetivos:
  - ▶ Ilustrar as classes principais do sistema;
  - ▶ Ilustrar o relacionamento entre os objetos.
- Apresentação da estrutura lógica: classes, relacionamentos.



## Diagrama de classes II



Nos diagramas UML, cada classe é dada por um **retângulo** dividido em três partes:

- 1 O **nome** da classe;
- 2 Seus **atributos**;
- 3 Seus **métodos**.

Cliente
-cpf: String -nome: String -dtNasc: Date
+totalComprasMensal(): double



Atributos no diagrama de classe:

**visibilidade nome: tipo = valor inicial {propriedades}**

- **Visibilidade:** + para **public**, – para **private**, # para **protected**;
- **Tipo** do atributo: Ex: **int**, **double**, **String**, **Date**;
- **Valor inicial** definido no momento da criação do objeto;
- **Propriedades.** Ex.: {readonly, ordered}



Métodos no diagrama de classe:

**visibilidade nome (par1: tipo1, par2: tipo2, ...): tipo**

- **Visibilidade:** + para **public**, – para **private**, # para **protected**;
- **(par1: tipo1, par2: tipo2, ...):** Se método contém parâmetros formais (argumentos), o nome e o tipo de cada 1. Ex: (nome: **String**, endereço: **String**, código: **int**);
  - ▶ Se método não contém parâmetros, manter par de parênteses, vazio.
- **tipo:** tipo de retorno do método. Ex.: **void**, se não retorna nada; **int**, **Cliente** (nome de uma classe), etc.



Mais detalhes sobre UML:

<http://www.uml.org/what-is-uml.htm>



## Relacionamentos entre classes



- Como os objetos das diferentes classes se relacionam?
- Como **diferentes classes** se relacionam?
- Vamos identificar as principais formas de conexão entre classes:
  - ▶ E representar esses relacionamentos no **diagrama de classes**;
  - ▶ Relações fornecem um **caminho** para a comunicação entre os objetos.



Tipos mais comuns:

- Entre **objetos de diferentes classes**:
  - ▶ **Associação** – “usa”;
  - ▶ **Agregação** – “é parte de”;
  - ▶ **Composição** – “é parte essencial de”.
- Entre **classes**:
  - ▶ **Generalização** – “É um”



**Relacionamentos** são caracterizados por:

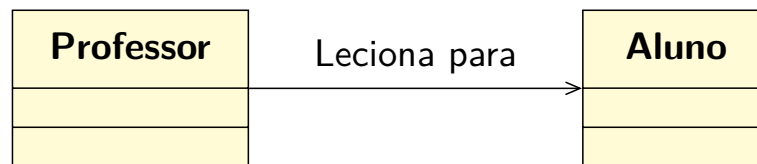
- **Nome**: descrição dada ao relacionamento (*faz, tem, possui,...*);
  - ▶ É usualmente um verbo.
- **Navegabilidade**: indicada por uma seta no fim do relacionamento;
  - ▶ Uni- ou bidirecional.
- **Multiplicidade**: 0..1, 0..\*, 1, 1..\*, 2, 3..7
- **Tipos de relacionamentos**: associação simples, agregação, composição, generalização.





Para facilitar seu entendimento, uma associação pode ser nomeada.

- O nome é representado como um “rótulo” colocado ao longo da linha de associação;
- Um nome de associação é usualmente um verbo ou uma frase verbal.



## Multiplicidade I



**Multiplicidade** refere-se ao número de instâncias de uma classe relacionada com uma instância de outra classe.

Para cada associação, há duas decisões a fazer, uma para cada lado da associação.

Por exemplo, na conexão entre Professor e Aluno:

- Para cada instância de Professor, podem ocorrer muitos (zero ou mais) Alunos;
- Por outro lado, para cada instância de Aluno, pode ocorrer exatamente um Professor (pensando em um curso isolado).



Tipos de multiplicidade:

Muitos (equivale a <i>0 ou mais</i> )	*
Zero ou mais	0..*
Exatamente um	1
Um ou mais	1..*
Zero ou um	0..1
Faixa especificada (ex.: entre 2 e 7)	2..7
<i>m ou n</i>	<i>m, n</i>



Exemplos:

- Um cliente pode ter apenas um nome;
- Uma turma deve ter no mínimo cinco e no máximo 100 alunos;
- Uma mesa de restaurante pode ter vários ou nenhum pedido;
- Uma cotação pode ter no mínimo um item.



É a forma **mais fraca** de relacionamento entre classes:

- As classes que participam desse relacionamento são independentes;
- Pode envolver duas ou mais classes.

Representa relacionamentos “usa”:

- Ex.: uma pessoa “usa um” carro
- **Na implementação:** um objeto A usa outro objeto B *chamando um método público de B*.



No diagrama de classes, as associações:

- São representadas como linhas conectando as classes participantes do relacionamento;
- 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.



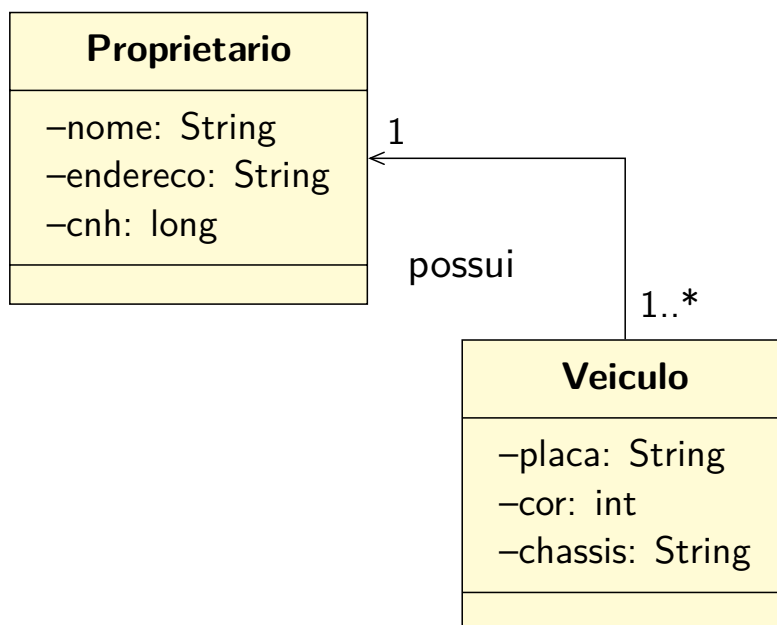
## Associação simples III



Exemplos:

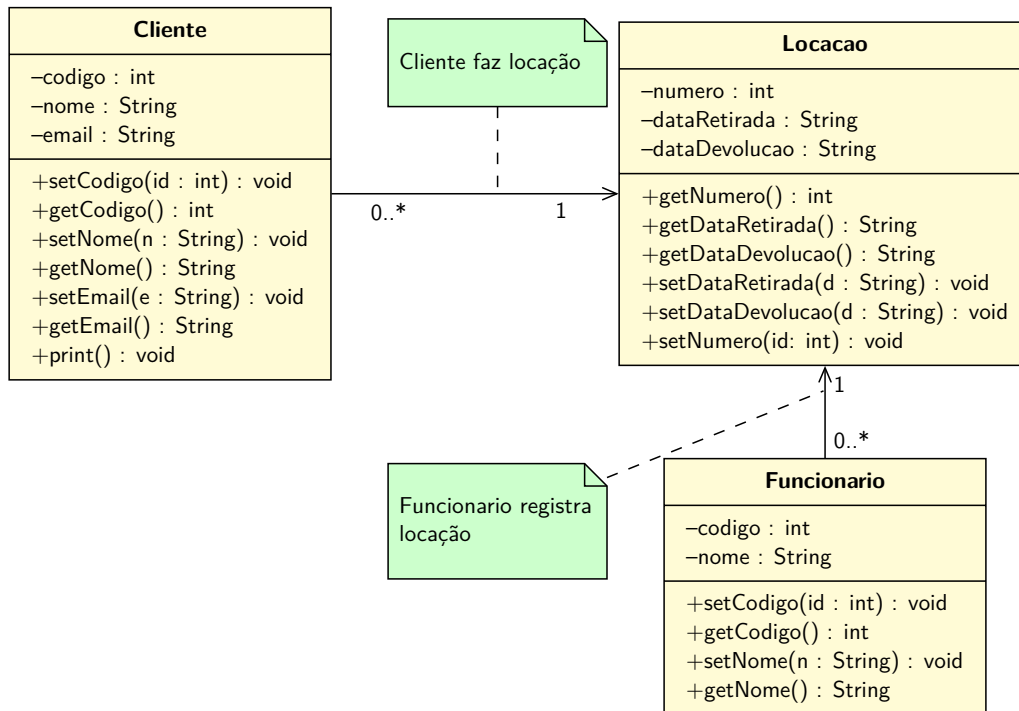


## Associação simples IV

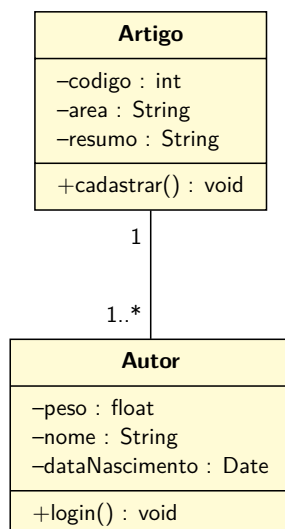




## Associação simples V



## Associação simples VI



Imagine um sistema de avaliação de artigos acadêmicos:

- Temos uma relação autor/artigo;
- Note que a classe **Autor** não compartilha atributos da classe **Artigo** e vice-versa;
- Nesse caso, não existe a relação *todo-parte*.



Essas duas formas de associação representam relacionamentos do tipo “**tem um**”

- Relacionamento **todo-parte**  $\Rightarrow$  Uma classe é formada por, ou contém objetos de outras classes

Exemplos:

- Um carro contém portas;
- Uma árvore é composta de folhas, tronco, raízes;
- Um computador é composto de CPU, teclado, mouse, monitor, ...



A **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 único *todo* por vez.
- **Agregação**: relacionamento todo-parte que não satisfaz um ou mais desses critérios:
  - ▶ A destruição do objeto *todo* não implica a destruição do objeto *parte*;
  - ▶ Um objeto pode ser *parte* componente de vários outros objetos.



Representação:

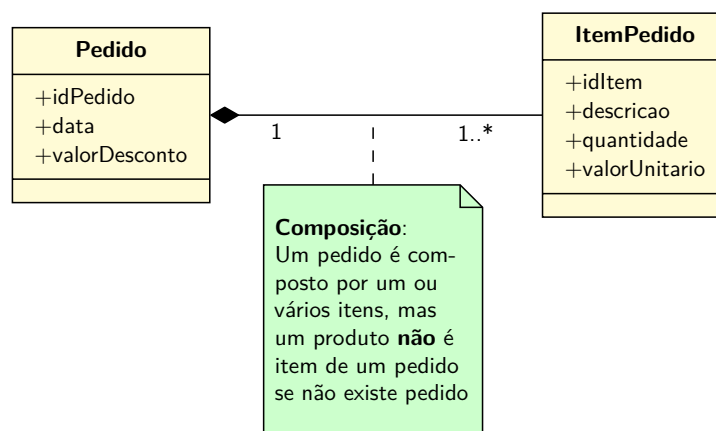
- **Composição**: associação representada com um losango **sólido** do lado *todo*.
  - ▶ O lado *todo* deve sempre ter multiplicidade 1.



- **Agregação**: associação representada com um losango **sem preenchimento** do lado *todo*.

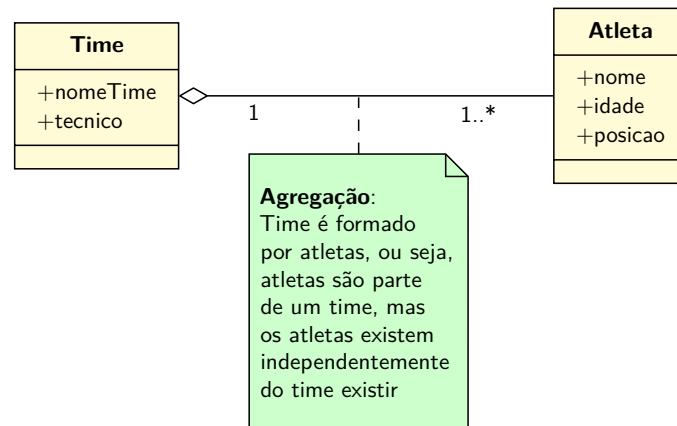


Exemplos:

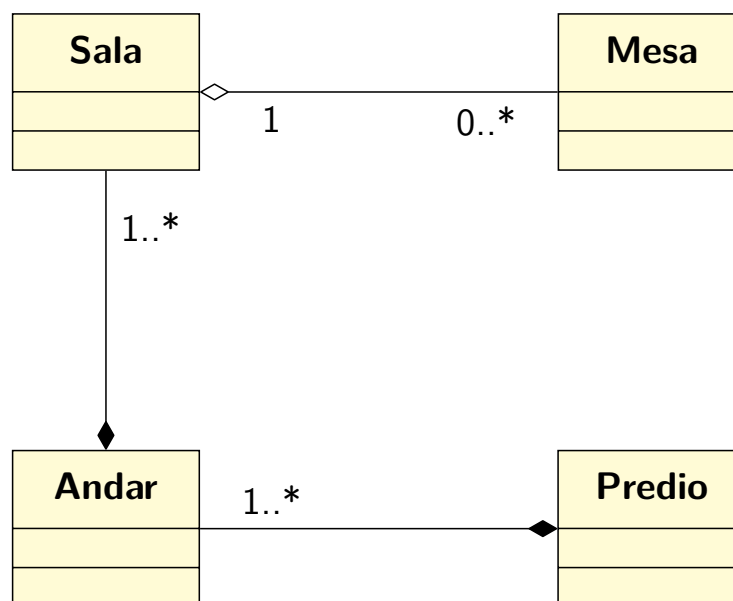




## Agregação e Composição V



## Agregação e Composição VI



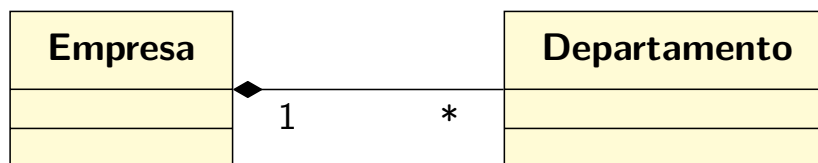




Agregação:



Composição:



**Classe todo:**

É a classe resultante da agregação ou composição

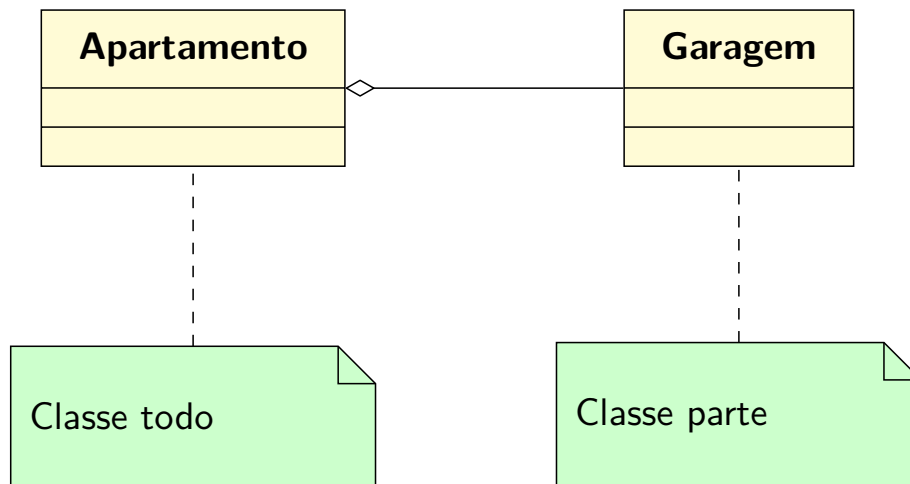
**Classe parte:**

É a classe cujas instâncias formam a agregação/composição

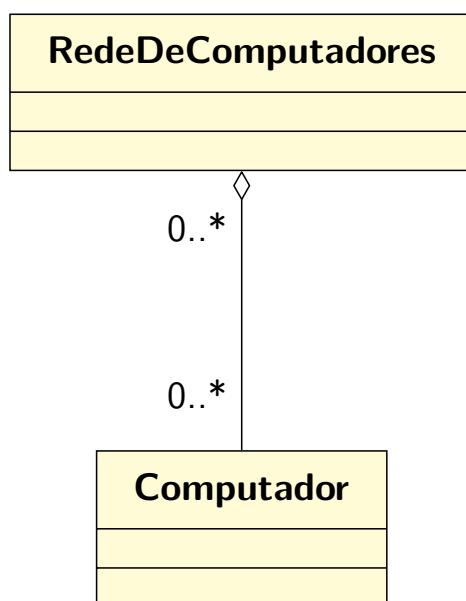
Exemplo:

Apartamento e Garagem: um apartamento pode ter garagem.

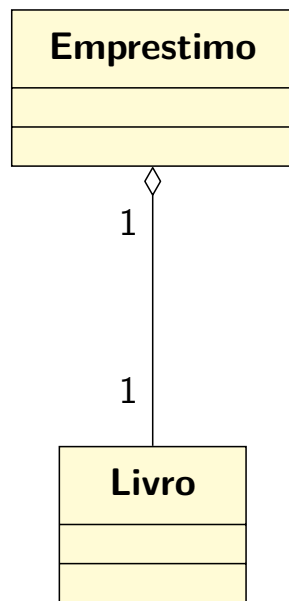
- Classe Apartamento: todo ou agregada
- Classe Garagem: parte



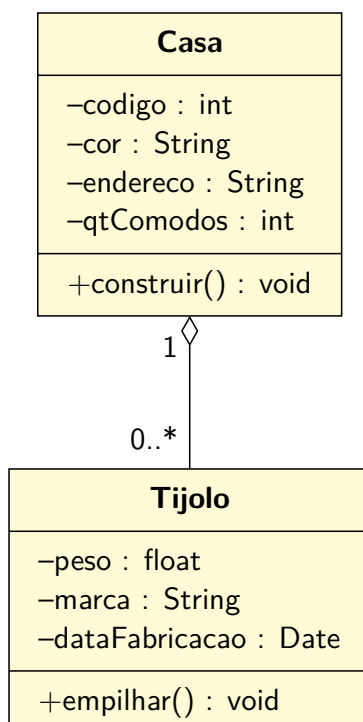
## Agregação: mais exemplos I



- Um computador existe independentemente de uma rede;
- Um computador pode estar ligado a mais de uma rede ao mesmo tempo.

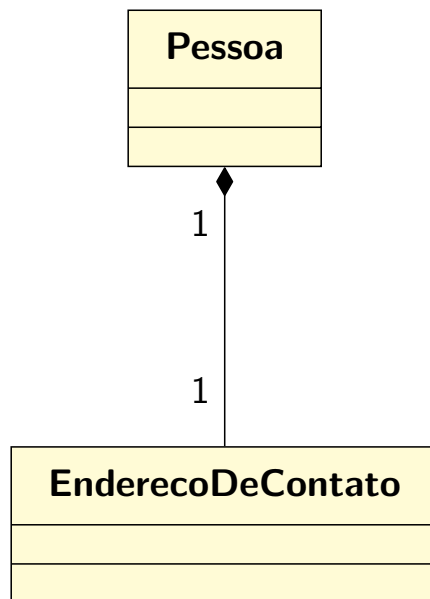


- Um empréstimo contém um livro, mas o livro não deixa de existir no sistema da biblioteca quando o empréstimo é concluído

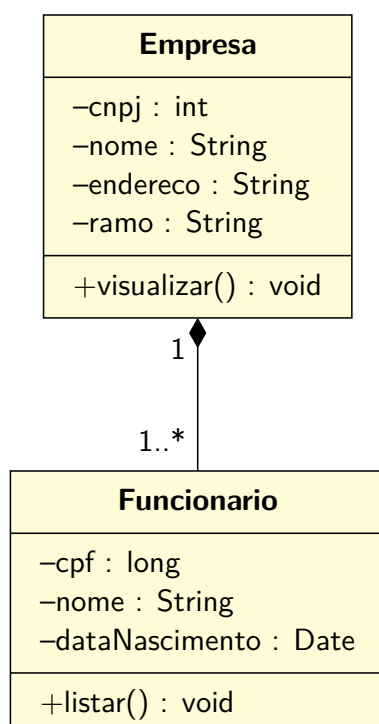


Imagine um sistema de gerenciamento de obras e suponha as classes Casa e Tijolo:

- Caso você deixe de construir uma casa, mesmo assim os tijolos poderão ser utilizados na construção de outro tipo de obra;
- Perceba que uma casa é feita de tijolos (relação **todo-parte**).



- O endereço de contato só faz sentido associado com uma pessoa;
- Se a pessoa é eliminada do sistema, não faz sentido manter o endereço de contato.



Imagine um sistema de Recursos Humanos e suponha as classes **Funcionário** e **Empresa**:

- Não faz sentido ter funcionários, se não existir uma empresa onde eles possam trabalhar;
- Se a empresa deixar de existir, automaticamente ela deixará de ter funcionários;
- Perceba que uma empresa é composta por funcionários (relação **todo-parte**).



```
class Pessoa {  
    String nome;  
    char sexo;  
    Data dataNasc; // Data: classe  
    ...  
}
```



```
class Data {  
    private int dia, mes, ano;  
    public void alteraData(int d, int m, int a){  
        dia = d;  
        mes = m;  
        ano = a;  
    }  
}
```



Representa relacionamentos entre classes do tipo “é um”.

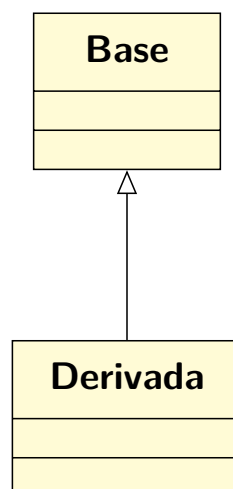
- Também chamado de **herança**.  
Exemplo: Um cachorro é um mamífero

Abstração de **Generalização/Especialização**:

- A partir de duas ou mais classes, abstrai-se uma classe mais genérica;
  - Ou de uma classe geral, deriva-se outra mais específica.
- Subclasses satisfazem todas as propriedades das classes as quais as mesmas constituem especializações;
- Deve haver ao menos uma propriedade que distingue duas classes especializadas.



No diagrama de classes, a generalização é representada por uma seta do lado da classe mais geral (**classe base**).

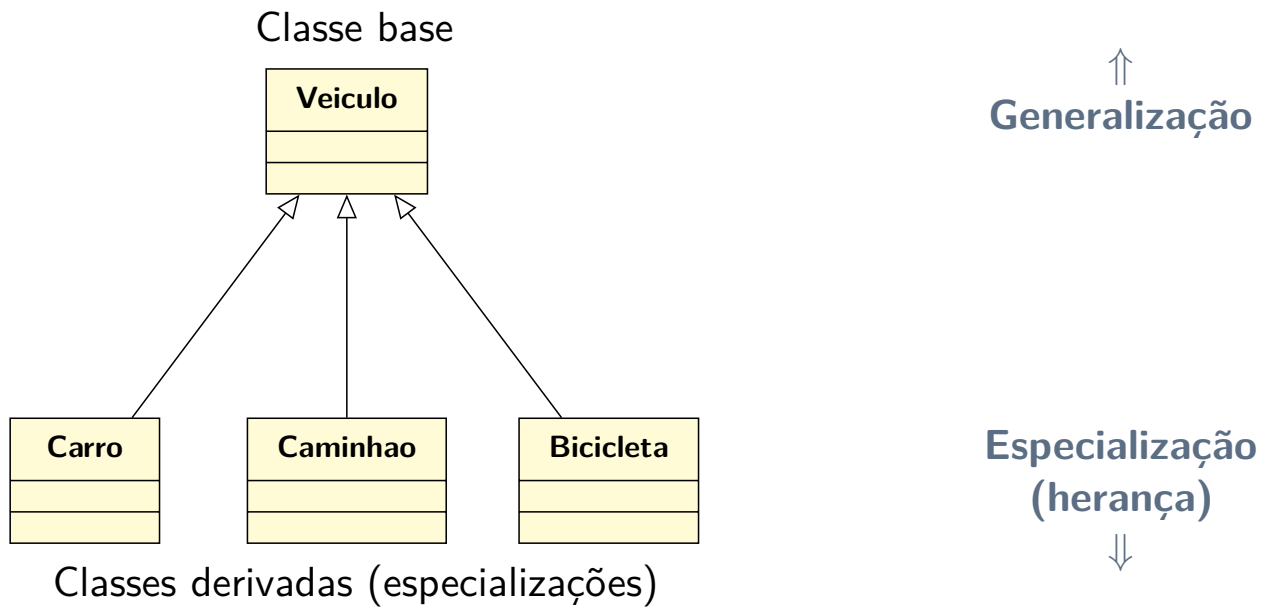




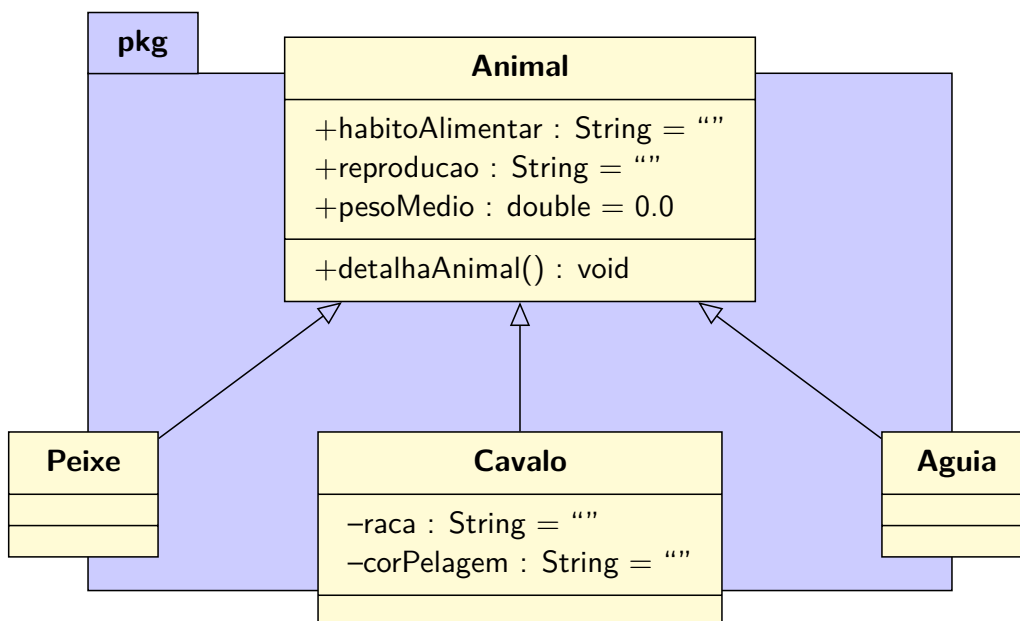
## Generalização ou herança III

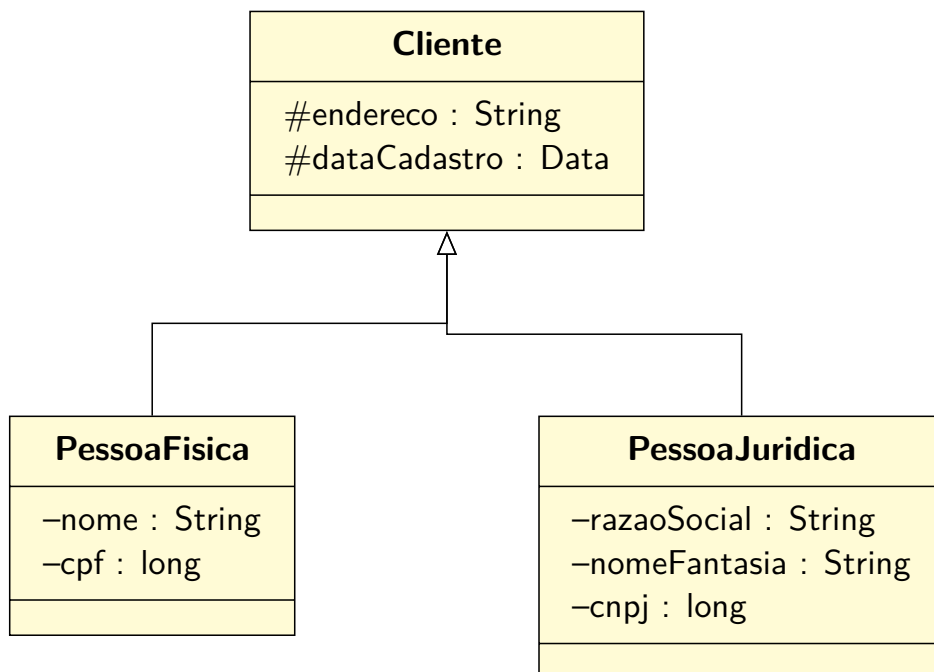
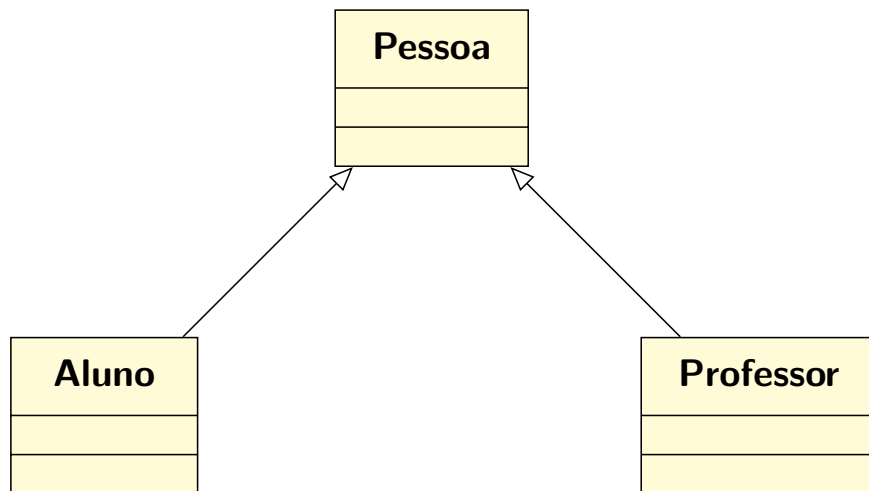


Exemplos:

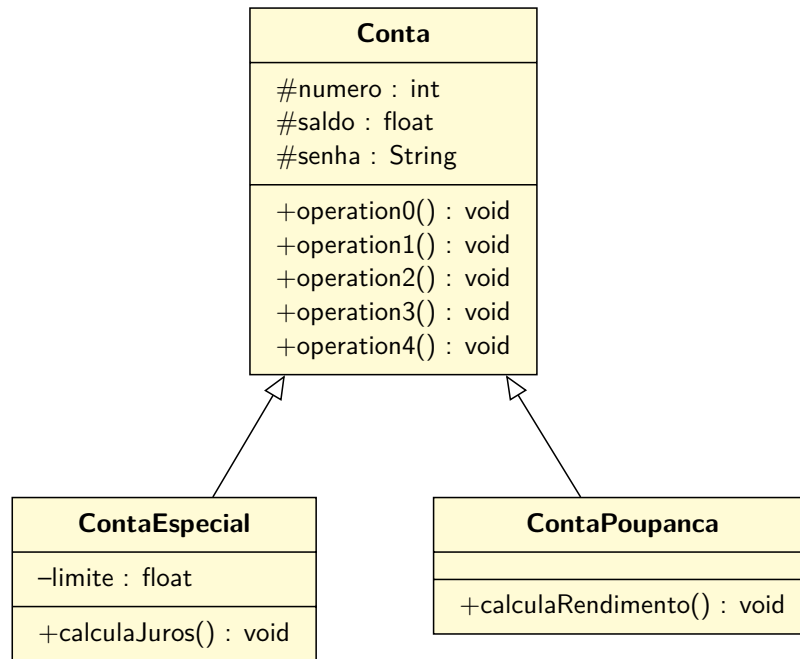


## Generalização ou herança IV









- A generalização permite organizar as classes de objetos hierarquicamente;
- Ainda, consiste numa forma de **reutilização** de software:
  - ▶ Novas classes são criadas a partir de existentes, absorvendo seus atributos e comportamentos, acrescentando recursos que necessitem



Como saber qual relacionamento deve ser utilizado?

- Existem atributos ou métodos sendo aproveitados por outras classes?  
A subclasse “é do tipo” da superclasse?
- **Sim**: Isso é herança
- **Não**: Existe todo-parte?
  - ▶ **Sim**: A parte vive sem o todo?
    - ★ **Sim**: Isso é agregação
    - ★ **Não**: Isso é uma composição
  - ▶ **Não**: Isso é associação



## Associação vs. agregação/composição I

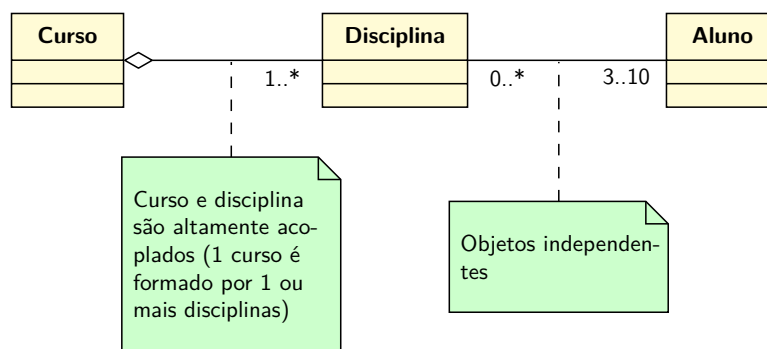


Se dois objetos são altamente acoplados por um relacionamento todo-parte:

- O relacionamento é uma **agregação** ou **composição**.

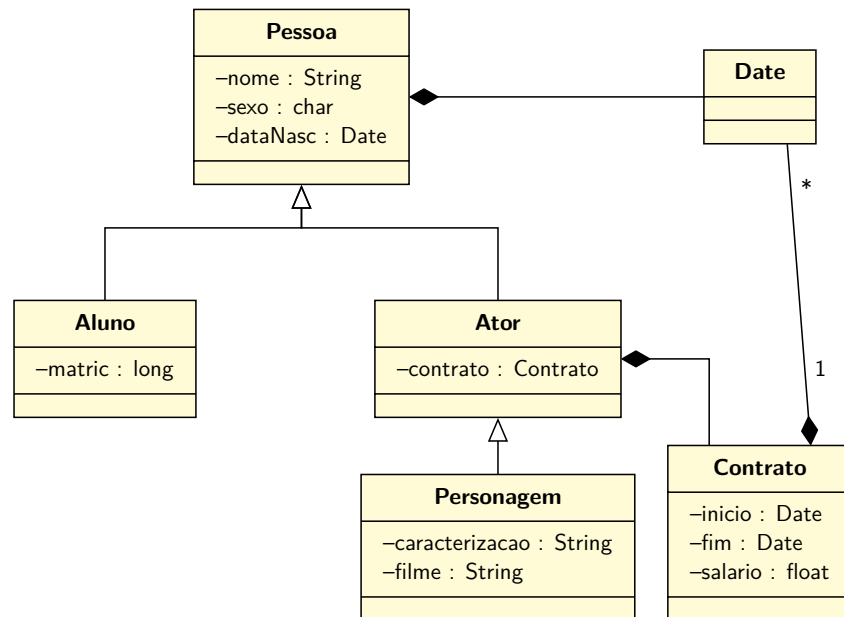
Se dois objetos são usualmente considerados como independentes, mesmo eles estejam frequentemente ligados:

- O relacionamento é uma **associação**.





Exemplo:

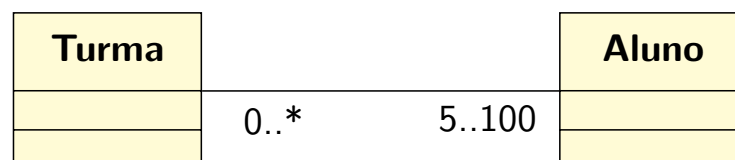


## Classes de associação I



As **classes de associação** são classes que fornecem um meio para adicionar atributos e operações a associações.

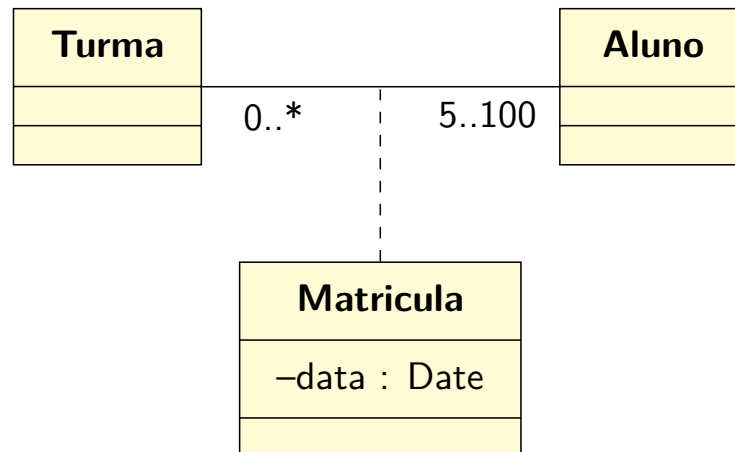
- Normalmente geradas entre ocorrências que possuem multiplicidade muitos nas extremidades
- **Exemplo**, considere o relacionamento a seguir:



- Deseja-se agora acrescentar a data em que cada aluno foi adicionado à turma;



- Obviamente, esta data não é uma propriedade *nem do aluno e nem da turma*.
- Sendo assim, criamos uma **classe associativa**, chamada, por exemplo, de Matricula:



## Exercícios I



- 1 Faça a modelagem em UML de um sistema bancário, relacionado à administração de contas bancárias (para cada classe defina, pelo menos, 4 atributos e 4 métodos). Em um banco há gerentes responsáveis por um grupo de clientes.
  - ▶ Um gerente poderá aceitar pedidos de abertura de conta, de empréstimo, de cartão de crédito, etc. Mas poderá decidir por oferecer os serviços, ou não.
  - ▶ Cada cliente poderá pedir serviços para um gerente: abertura de contas, empréstimo, cartão de crédito, etc. Ele também poderá ter acesso à sua conta bancária.
  - ▶ Cada conta bancária poderá oferecer serviços tais como: depositar, sacar, transferir dinheiro entre contas, pagar cartão de crédito, etc.
  - ▶ Após a modelagem, para cada classe coloque quais serviços pode solicitar das outras classes.



## Exercícios II



Gerente	Cliente	ContaBancaria
<ul style="list-style-type: none"><li>-nome : String</li><li>-funcao : String</li><li>-numeroClientes : int</li><li>-cpf : long</li></ul>	<ul style="list-style-type: none"><li>-nome : String</li><li>-cpf : long</li><li>-salario : double</li><li>-profissao : String</li></ul>	<ul style="list-style-type: none"><li>-nomeCliente : String</li><li>-tipoConta : String</li><li>-validade : String</li><li>-dataCriacao : String</li></ul>
<ul style="list-style-type: none"><li>+iniciarPedidoEmprestimo()</li><li>+iniciarPedidoCartao()</li><li>-liberarEmprestimo()</li><li>-liberarCartao()</li></ul>	<ul style="list-style-type: none"><li>+atualizarSenha()</li><li>+cadastrarComputador()</li><li>+pedirEmprestimo()</li><li>+pedirCartao()</li></ul>	<ul style="list-style-type: none"><li>+depositar()</li><li>+sacar()</li><li>+transferir()</li><li>+pagarCartao()</li></ul>



## Exercícios III



- ② Faça a modelagem em UML de um sistema de controle de cursos de informática equivalente ao módulo de matrícula de acordo com os seguintes fatos:
- ▶ o curso pode ter mais de uma turma, no entanto, uma turma se relaciona exclusivamente com um único curso.
  - ▶ uma turma pode ter diversos alunos matriculados, no entanto uma matrícula refere-se exclusivamente a uma determinada turma. Cada turma tem um número mínimo de matrículas para iniciar o curso.
  - ▶ um aluno pode realizar muitas matrículas, mas cada matrícula refere-se exclusivamente a uma turma específica e a um único aluno.



- ③ Faça a modelagem em UML de um sistema de reserva para uma empresa aérea (para cada classe defina, pelo menos, 4 atributos e 4 métodos).
- ▶ Cada voo deverá estar cadastrado no sistema, pois as reservas serão relacionadas a eles. Cada voo pode informar o número de assentos livres, sua tripulação, reservar acento, etc
  - ▶ Operadores são funcionários da empresa responsáveis pela operacionalização das reservas. Os operadores fazem as reservas, as cancelam, informam sobre possíveis atrasos, etc
  - ▶ Os clientes podem pedir reservas nos voos, podem cancelar reservas, podem pagá-las de forma adiantada, etc

Após a modelagem, para cada classe coloque quais serviços pode solicitar das outras classes.



## Referências



- ① Addison-BOOCH, G., RUMBAUGH, J., JACOBSON, I. *UML, Guia do Usuário*. Rio de Janeiro: Campus, 2000.
- ② FOWLER, M. *UML Essencial*, 2a Edição. Porto Alegre: Bookman, 2000.
- ③ LARMAN, C. *Utilizando UML e Padrões: Uma Introdução à Análise e ao Projeto Orientado a Objetos*. Porto Alegre: Bookman, 2001.

Os slides dessa apresentação foram cedidos por:

- Graça Marietto e Francisco Zampirolli, UFABC
- Profa Katti Faceli, UFSCar/Sorocaba
- Marcelo Z. do Nascimento, FACOM/UFU

LaTeXagem: Renato Pimentel, FACOM/UFU