

# ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

# P.PORTO

**CTeSP DWDM**

Análise e Arquitetura de Sistemas

UML: Diagramas de Classes e Objetos

Catarina Félix | AAS | CTeSP DWDM | 1º Semestre | 2018/19

# Objetivos do diagrama de classes

- Descrição formal da estrutura dos objetos/entidades do sistema
  - Uma entidade tem
    - Identidade (nome)
    - Estrutura (atributos)
    - Comportamento (operações/métodos)
    - Relacionamentos com outras entidades
- **Uma Classe é um tipo de entidade**, ou seja, uma família de objetos que tem uma estrutura e comportamento semelhantes.

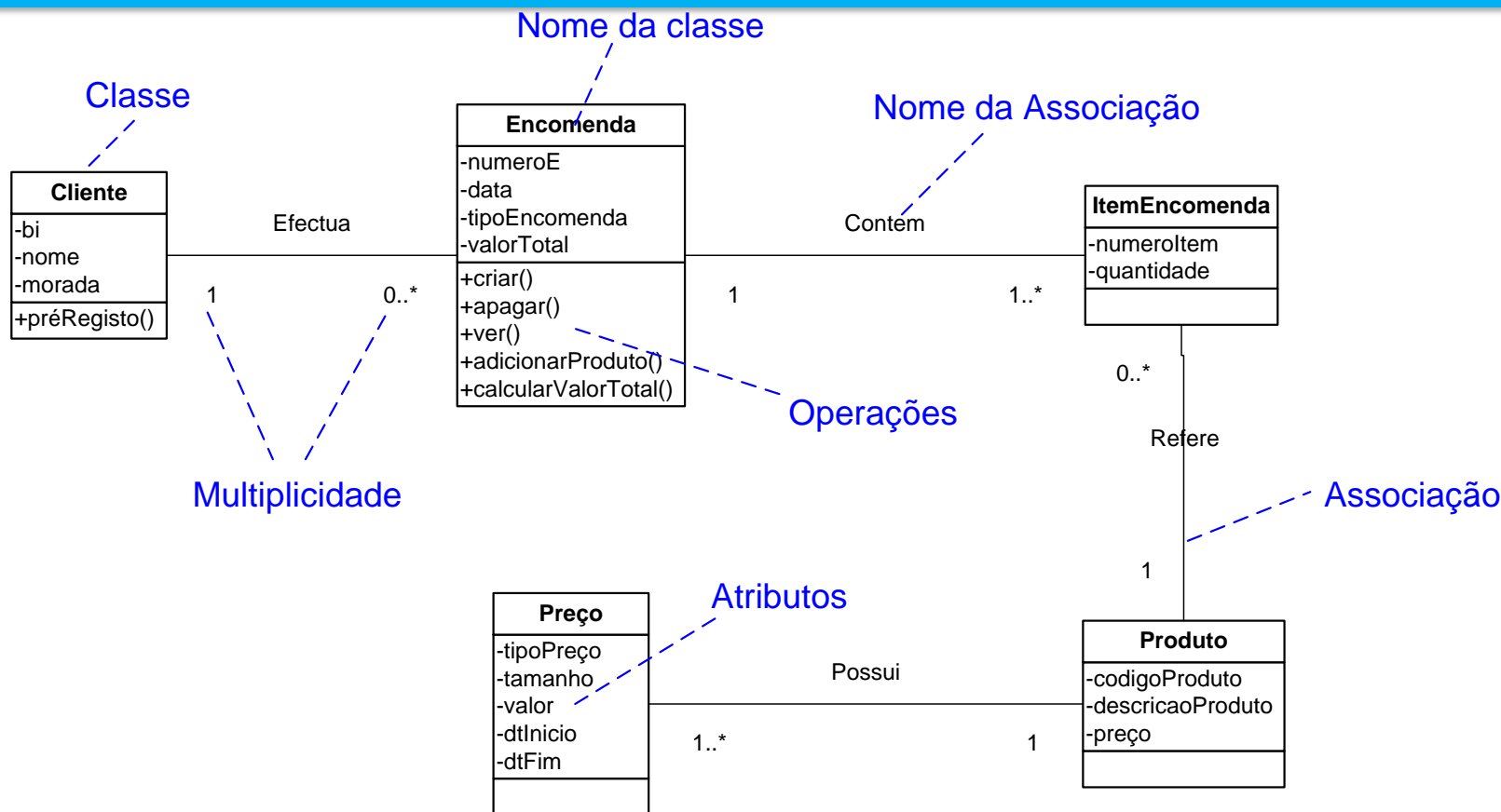
# Objetivo

- Criar diagramas de classes é:
  - Identificar objetos (**entidades/conceitos**) relevantes no contexto que se pretende modelar e onde se procuram descrever características comuns em termos de propriedades (**atributos**) e de comportamento (**operações**)
  - O modelo de classes descreve o modelo geral de informação do sistema!
  - Os diagramas de classes pretendem suportar os requisitos funcionais do sistema, “levantados” previamente.
- O diagrama de classes é construído e refinado ao longo das várias fases do desenvolvimento do software, por analistas, projetistas (designers) e programadores

## Exemplo

Aproveitando o exemplo apresentado aquando do estudo dos diagramas de casos de utilização, e acrescentando a seguinte descrição:

“Um cliente pode efetuar muitas encomendas, contendo cada encomenda diversos itens, numerados sequencialmente, que se referem a um determinado produto e respetiva quantidade encomendada. Os produtos vendidos pela PhonePizza abrangem pizzas com diversos tamanhos (pequena, média, grande), bebidas e saladas. O preço pode variar conforme o tamanho do produto bem como com as promoções existentes que têm uma data de início e de fim.”



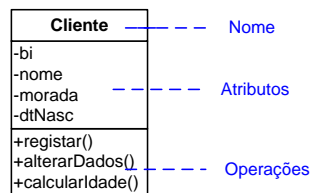
Restrições:  
 Preço.tipoPreço={Normal, Produção}  
 Preço.tamanho={Único, Pequeno, Médio, Grande}

# Objetos: reais vs funcionais

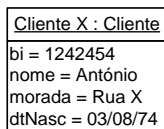
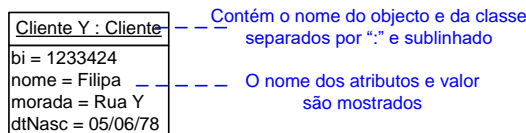
- Tipicamente há uma correspondência direta entre os objetos de Software e os objetos do domínio que representam. Mas no desenho e implementação do sistema segundo o paradigma O.O, existem (quase sempre) objetos adicionais de software, onde não se identifica uma correspondência com o negócio, no entanto são objetos cruciais para o funcionamento do sistema.
- **Exemplos de objetos do mundo real:**
  - o Sr. João
  - a aula de AAS no dia 04/11/2015
- **Exemplos de objetos computacionais:**
  - o registo que descreve o Sr. João (imagem de objeto do mundo real)
  - uma árvore de pesquisa binária (objeto puramente computacional)

# Classes e objetos

- No desenvolvimento de software OO, não nos interessam tanto os objetos individuais mas sim as classes de objetos
- Uma classe representa uma abstração sobre um conjunto de objetos que partilham a mesma estrutura e comportamento
- Um objeto é um caso particular de uma classe, ou seja, uma instância da classe



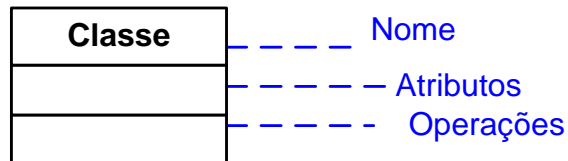
Classe



Objetos

# Classe

- A representação gráfica de uma classe é um retângulo com uma, duas ou três secções.
- Pode ainda ter uma quarta secção (opcional) onde se poderá especificar outra informação
  - (e.g., a lista de responsabilidades que a classe assume)



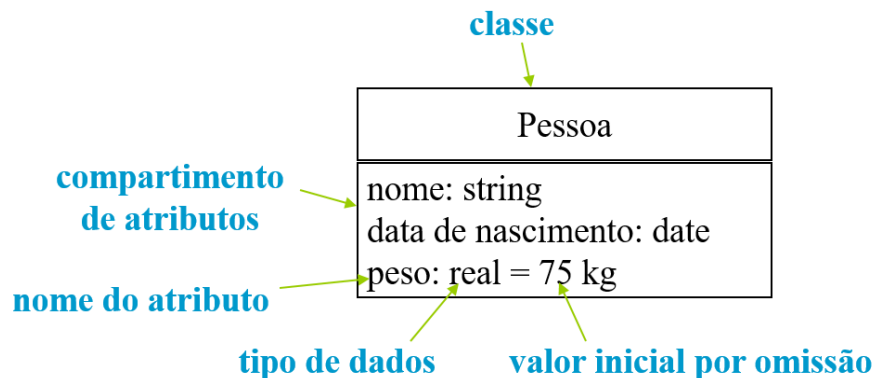


# Atributos

- Um atributo é uma característica que os objetos possuem e que é representada por um valor de dados
  - Exemplo: o atributo Cor poderá ser igual a “Vermelho” ou “Azul”
- O estado de um objeto é dado por valores de atributos
- Atributos são definidos ao nível da classe, enquanto que os valores dos atributos são definidos ao nível do objeto
- Cada atributo pode incluir:
  - nome do atributo
  - tipo de dados
  - valor por defeito (valor atribuído se não for indicado nenhum)
  - multiplicidade (número de elementos representados por esse atributo)

# Atributos

- Os nomes dos atributos são únicos numa classe, não podendo existir na mesma classe dois atributos com o mesmo nome
- Em classes diferentes podem existir atributos com nomes iguais
- Atributos são listados num compartimento de atributos (opcional) a seguir ao compartimento com o nome da classe
- Os nomes dos tipos não estão pré-definidos em UML, podendo-se usar os da linguagem de implementação alvo



## Tipos de dados UML

- UML reconhece um conjunto de tipos de dados intrínsecos
  - string, integer, boolean, unlimited natural
- também assume a existência de tipos de dado primitivos que são definidos fora de UML, nas linguagens de programação
  - date, double, float
- ou definidos pelo utilizador:
  - numeroTelefone, endereço
- adicionalmente, existe o tipo de dados enumeration, útil para atributos com listas pequenas de valores fixos e discretas
  - Exemplos: gênero (masculino, feminino), cor (branco, amarelo, vermelho)

# Operações

- Os objetos apenas comunicam entre si por mensagens, o que na prática resulta na invocação de operações
- As operações são a representação lógica do comportamento de um objeto, consistindo em ações efetuadas por ou sobre um objeto
- Também se pode definir operações como serviços disponibilizados por um objeto (estes serviços serão invocados por outros objetos como parte integrante de uma colaboração)
  - Objetos da mesma classe têm as mesmas operações
- Operações são definidas ao nível da classe, enquanto que a invocação de uma operação é definida ao nível do objeto

# Operações

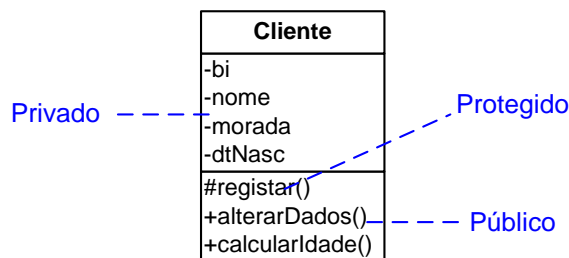
- **Princípio do encapsulamento:** acesso e alteração do estado interno do objeto (valores de atributos e ligações) controlado por operações
- Nas classes que representam objetos do mundo real é mais comum definir responsabilidades em vez de operações

## Visibilidade de operações

Símbolo	Visibilidade	Significado
+	Pública	Qualquer objeto pode invocar a operação
-	Privada	Só o objeto pode invocar a operação
#	Protegido	Só os objetos e descendentes podem invocar a operação
~	Pacote	Só os objetos no mesmo pacote podem invocar a operação

## Visibilidade de atributos

- Os atributos são normalmente privados.
- Contudo, há 2 formas de os fazer públicos:
  - Através de operações “accessor” tipicamente produzidas automaticamente pelas ferramentas de UML. Estas operações podem ter qualquer dos tipos de visibilidade na tabela anterior.
  - Através dos atributos estáticos. Estes atributos são partilhados por todas as instâncias duma tabela.



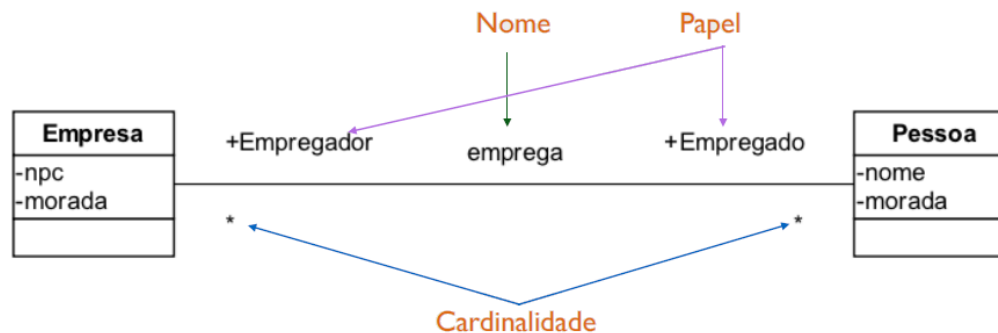
## Identificar classes e objetos

- Sublinhar nomes frequentes, a partir de descrições textuais do funcionamento do sistema, documentos de requisitos ou especificação de use cases.
- para cada nome verificar:
  - Se (*nome = família\_de\_coisas*) Então **Classe** [ex: *Livro*]
  - Se (*nome = nome\_próprio*) Então **Objeto** [ex: *LivroUML*]
  - Se (*nome = uma\_característica*) Então **Atributo** [ex: *autor*]
  - Se (*nome = valor*) Então **valor\_de\_atributo** [ex: *A.Silva*]
  - Se (*nome = situação\_ou\_estado*) Então **Estado** [ex: *disponível*]
  - Se (*nome = ocorrência/evento*) Então Evento, Ação ou **Método** [ex: *requisitar*]



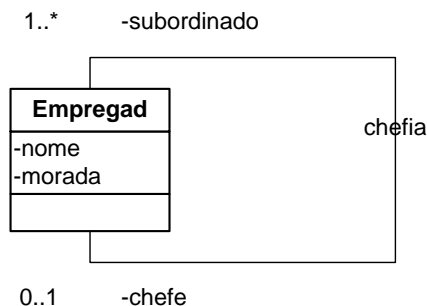
## Relações entre classes

- As associações representam as relações entre objetos.
- São caracterizadas por possuir um nome e quando necessário podem também incluir o papel que os objetos têm na relação.
- O nome das associações corresponde normalmente a verbos e possui um tamanho reduzido.



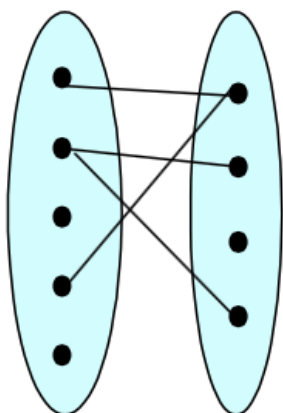
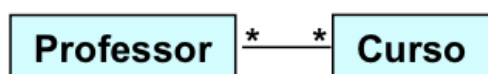
## Relações entre as classes

- Uma classe pode possuir uma associação consigo própria, significando neste caso que um objeto da classe se relaciona com um ou vários objetos da mesma classe (Associação reflexiva)

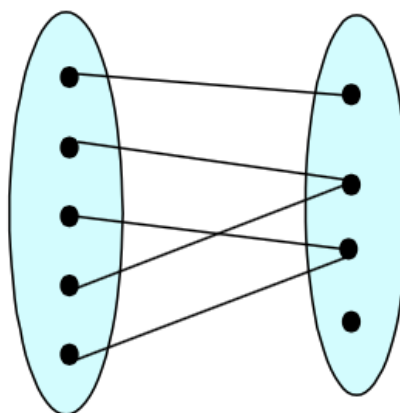


- Pode haver mais do que uma associação (com nomes diferentes) entre o mesmo par de classes
- Papéis nos extremos da associação podem ter indicação de visibilidade (pública, privada, etc.)

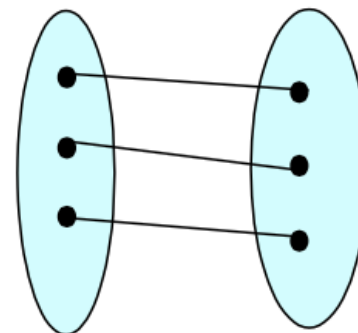
## Multiplicidade das relações



Muitos-para-Muitos



Muitos-para-1



1-para-1

# Cardinalidade

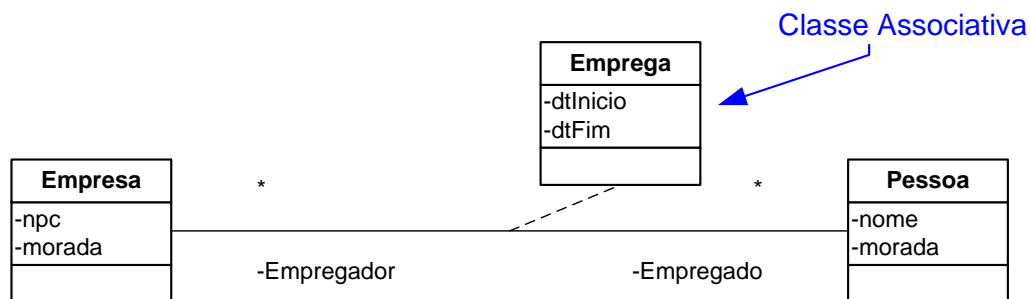
- Considerando uma associação qualquer entre duas classes C1 e C2, na perspectiva da classe C1.
- 1 → Cada instância da classe C1 deve ser ligada com **exatamente 1** instância da classe C2
- \* → Cada instância da classe C1 pode ser ligada com **0 ou mais** instâncias da classe C2
- 0..\* → Cada instância da classe C1 pode ser ligada com **0 ou mais** instâncias da classe C2
- 0..1 → Cada instância da classe C1 pode ser ligada com **0 ou 1** instância da classe C2
- 1..\* → Cada instância da classe C1 pode ser ligada com **1 ou mais** instâncias da classe C2
- m..n → Cada instância da classe C1 pode ser ligada com pelo menos **m ou até n** instâncias da classe C2 ( $m < n$ )
- m.n.p → Cada instância da classe C1 deve ser ligada com **m, n ou p** instâncias da classe C2

## Classes associativas

- surge da necessidade de reforçar o detalhe de informação de uma associação
- reúne as propriedades de associação e classe
- o nome pode ser colocado num sítio ou noutro, conforme interessa realçar a natureza de associação ou de classe, mas a semântica é a mesma
- o nome também pode ser colocado nos dois sítios
- não é possível repetir combinações de objetos das classes participantes na associação

Normalmente, as classes associativas surgem nas relações de “Muitos para Muitos” e o nome da classe é dado pelo nome da associação

**Exemplo:**



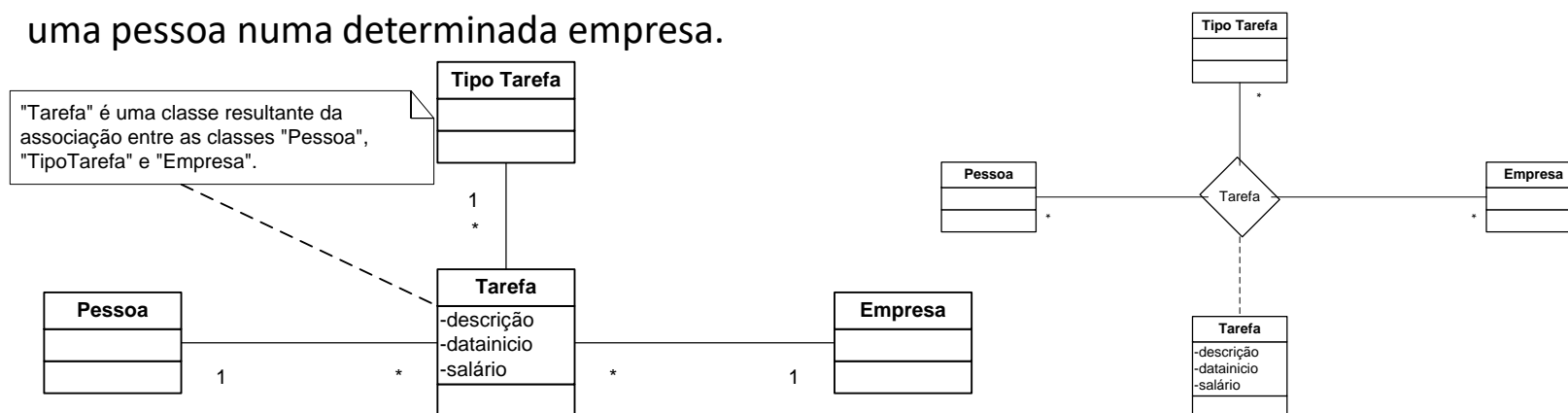
## Relações n-árias

- Há associações que envolvem mais de duas classes.
- Uma associação entre três classes denomina-se ternária.
- Uma associação entre quatro classes chama-se quaternária.
- Generalizando, a este tipo de associações chama-se N-árias.
- Um exemplo de uma relação ternária, é um contrato entre um arquiteto, um cliente e um projeto. Esta associação envolve três classes
  1. Arquiteto
  2. Cliente
  3. Projeto



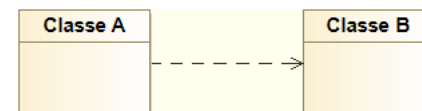
## Exemplo n=4 (quaternária)

- As associações N-árias podem geralmente ser transformadas em várias relações binárias entre a classe-associação e as restantes classes participantes.
- No entanto, se esta for a estratégia adotada dever ser assinalado esse facto (por exemplo, através de um estereótipo ou de uma anotação) junto à classe- associação.
- Um exemplo de uma relação quaternária, é uma tarefa de um determinado tipo realizada por uma pessoa numa determinada empresa.



## Relações entre classes: dependência

- A relação menos formal entre classes
  - **Genericamente** significa que a classe origem (A) depende de algum modo da classe destino (B)
  - **Especificamente** pode significar:
    - A uses B
    - se o interface de B mudar, poderá haver um impacto na Classe A
- Em termos programáticos significa que:
  - A recebe uma instância de B como parâmetro em pelo menos um dos seu métodos
  - A cria uma instância de B local para um dos seus métodos
- Não se deve usar para indicar que A declara uma variável de instância de B




Exemplo:

A: Window


B: WindowCloseEvent

```
import B;
public class A {
    public void method1(B b) {
        // ...
    }

    public void method2() {
        B tempB = new B();
        //...
    }
}
```

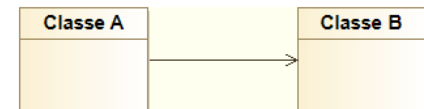


```
import B;
public class A {
    private B b; // Errado!
    public B getB() {
        return b;
    }
    // ...
}
```





## Relações entre classes: associação



Exemplo:  
A: Window  
B: Cursor

- Dependência mais forte e a “seta”, neste caso, indica uma relação unidirecional
- A classe A está associada com B, ou seja a classe A usa ou contém uma instância de B, mas B desconhece nem contém uma instância de A

```
import B1;
public class A1 {
    private B1 b1;
    public B1 getB1() {
        return b1;
    }
}
```

# Relações entre classes: agregação e composição

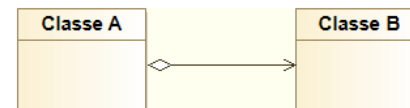
- Quando há a necessidade de definir o tempo de vida das instâncias que compõem uma associação e quantas instâncias estão envolvidas
- Utilização um de dois tipos de associações específicas: **Agregação** ou **Composição**

```
import B1;
public class A1 {
    private B1[] b1;
    // ...
    public B1 getB1(int anIndex)
    {
        return b1[anIndex];
    }
}
```

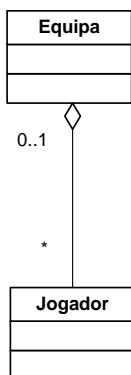
## Relações entre classes: agregação

- **Agregação**

- pretende demonstrar o facto de que um todo (A) é composto por partes (B)
- Associação com o significado **contém** (é constituído por) / **faz parte de** (part of)
- Relação de inclusão nas instâncias das classes
- Hierarquias de objetos
- A contém referência a uma instância de B como parte do seu estado

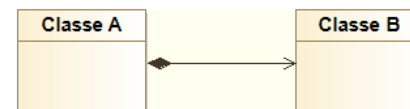


Exemplo:  
A: Equipa  
B: Jogador



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

## Relações entre classes: Composição



Exemplo:

A: Encomenda

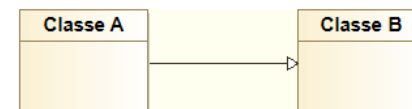
B: ItemEncomenda

- A composição é uma agregação com um significado mais forte existindo uma dependência direta entre as duas classes (**se a parte deixar de existir, o todo também deixa e vice-versa**).
  - 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 distinção entre a composição e a agregação é ténue, ficando por vezes ao critério do analista.**



Utiliza-se a composição porque não faz sentido possuir uma encomenda sem itens ou itens sem uma encomenda

## Relações entre classes: Generalização

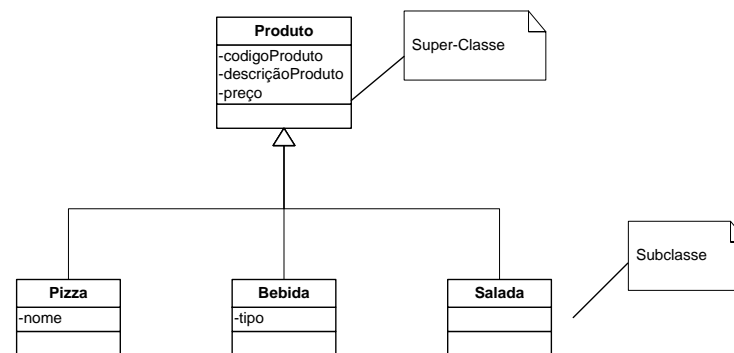
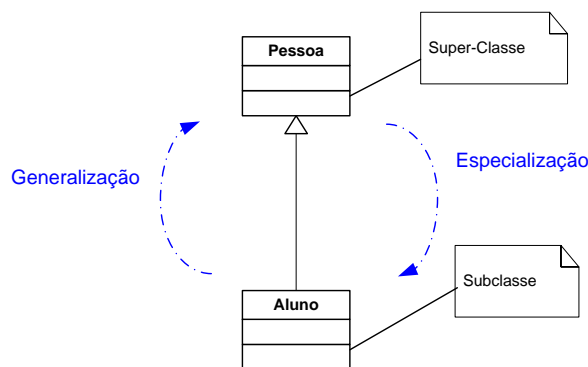


Exemplo:

A: Aluno

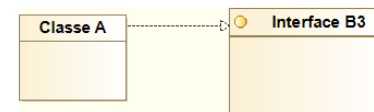
B: Pessoa

- Relação semântica “is a” (“é um” / “é uma”) : um aluno é uma pessoa
- Relação de herança nas propriedades: A sub-classe herda as propriedades (atributos, operações e relações) da super-classe, podendo acrescentar outras



```
import B2;  
public class A2 extends B2 { // ...}
```

## Relações entre classes: Realização



- Relação parecida com a generalização
- No desenvolvimento OO a relação de realização representa a implementação de um interface por uma classe!
- O interface representa a forma como algumas características de uma classe são definidas sem “dizer” nada acerca dos detalhes de implementação.

Exemplo:

A: Printer

B: PrinterSetup

### Importante no desenho de subsistemas

```
import B3;
public class A3 implements B3 {
    // ...
}
```

# Restrições

- Para além das restrições impostas pelas associações no diagrama de classes, é também possível restringir o valor dos atributos das classes.
- As restrições devem ser representadas utilizando as chavetas “{ }”
- Para além do nome do atributo, também se pode adicionar o nome da respetiva classe

Restrições:

Preço.tipoPreço= {Normal, Promoção}

Preço.tamanho= {Único, Pequeno, Médio, Grande}

Esta é uma visão simplificada das restrições. Algumas ferramentas case usadas para criação de modelo UML permitem usar a linguagem OCL (Object Constraint Language) para definir formalmente as restrições.

## Diagrama de classes PhonePizza revisto

