UML

- **UML** - **Unified Modeling Language** - é uma notação utilizada para representar uma aplicação através de um conjunto de diagramas.

- Os diagramas são criados e utilizados durante a fase de desenvolvimento da aplicação.

UML

- UML disponibiliza diferentes tipos de diagramas.
- No contexto da disciplina, vamos utilizar **diagramas de classes**.
- Um diagrama de classes permite descrever a estrutura de uma aplicação, mostrando as classes, atributos e métodos, e relações entre classes.

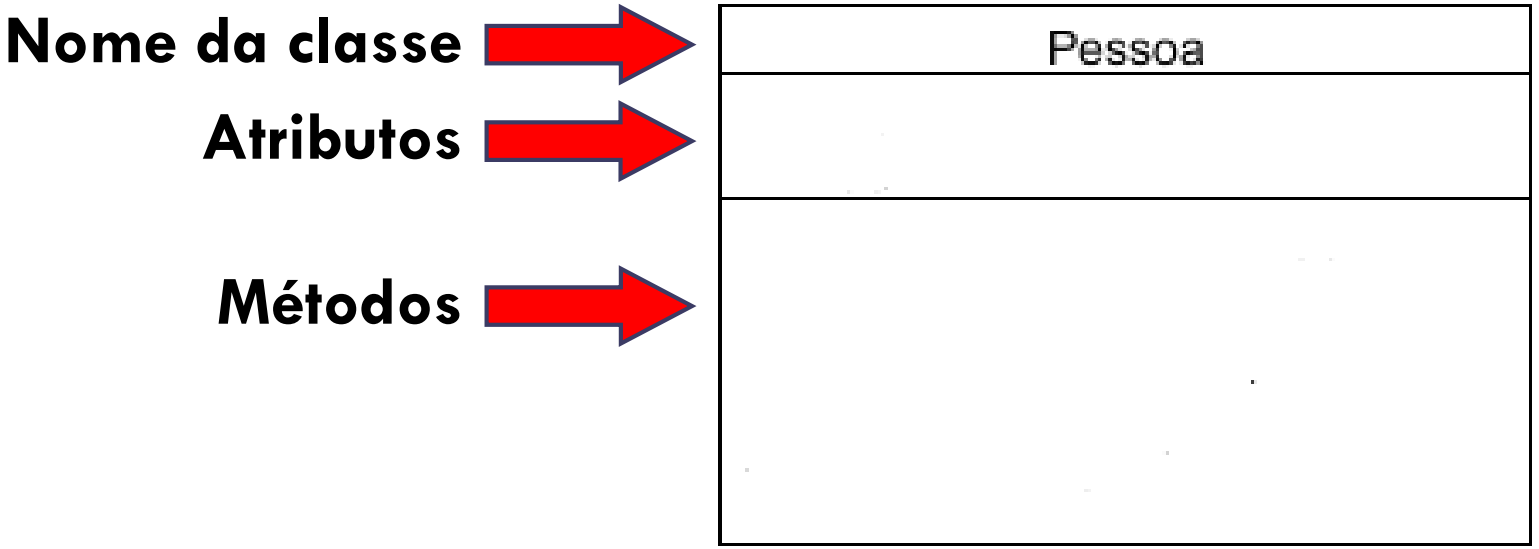
Diagramas de Classes

- Um diagrama de classes dá uma panorâmica geral do programa, mostrando as suas classes e as relações entre elas.

- Relações: que classes constituem outra, que classes uma dada classe conhece e usa, herança, etc.

Representação de uma Classe

- Uma classe é representada por um retângulo dividido em três secções:



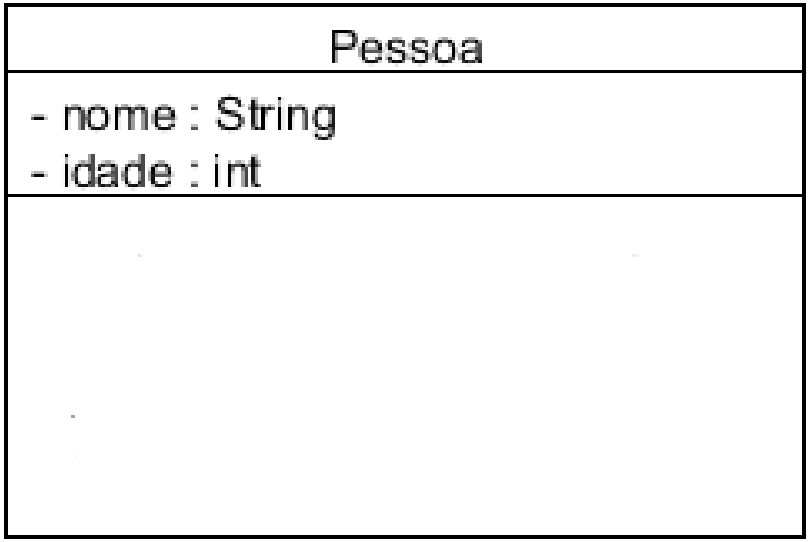
Visibilidade

- Simbologia para a visibilidade:
  - public +
  - private -
  - protected #

Representação de uma Classe

- Especificação dos atributos:

[visibilidade] **nome** : tipo

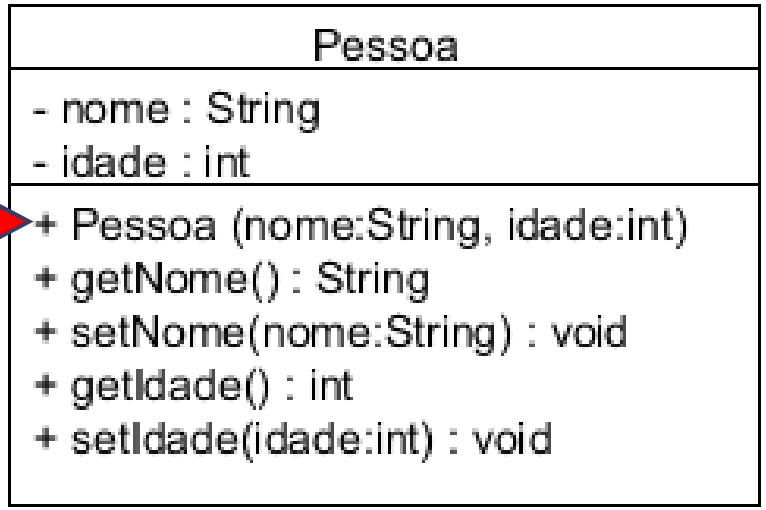


Representação de uma Classe

- Especificação dos métodos:

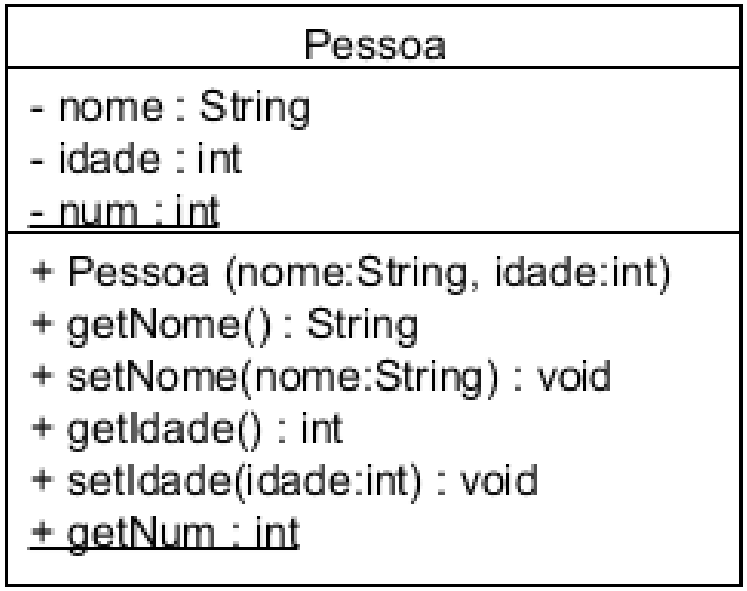
[visibilidade] **nome** (lista dos argumentos : tipo) : tipo de retorno

Construtor



Representação de uma Classe

- Atributos e métodos estáticos (static) aparecem sublinhados:



UML

- Os diagramas UML não são específicos para a linguagem Java → são independentes da linguagem.

- Por isso, a sintaxe não é necessariamente a mesma do Java.

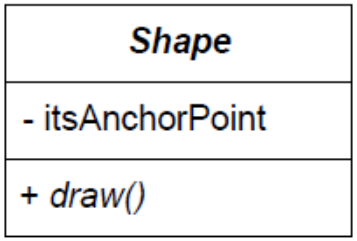
Representação de uma Classe

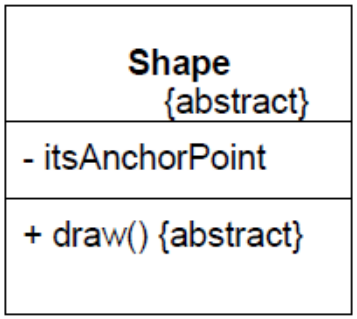
- Pode acontecer que o diagrama de uma classe não mostre:
  - Construtor
  - *Getters* e *setters*
  - Método *toString*

- **Isto não significa que não existam no código!**

Classe Abstrata

- Podemos representar uma classe e métodos abstratos escrevendo os seus nomes a itálico ou usando {abstract}.

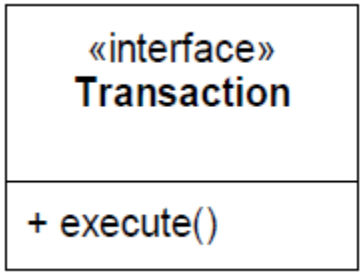





```
public abstract class Shape
{
  private Point itsAnchorPoint;
  public abstract void draw();
}
```

Interface

- Uma interface representa-se da seguinte forma:




```
interface Transaction
{
  public void execute();
}
```

Representação de uma Classe - Exemplo

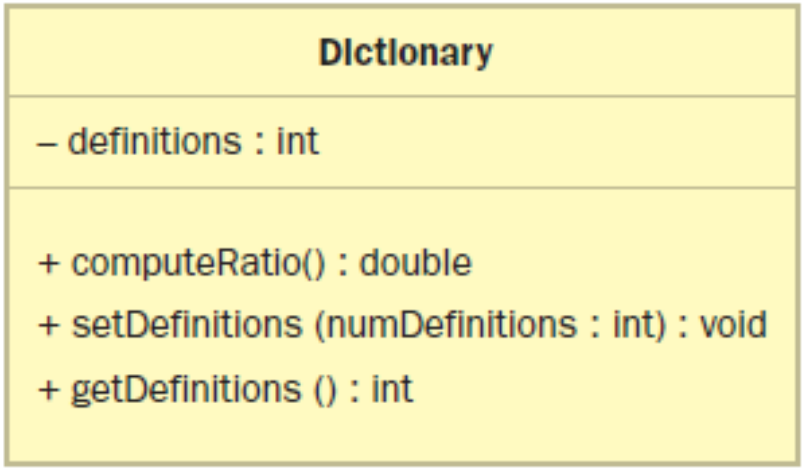
- Exemplo de representação da classe **Dictionary:**

```
public class Dictionary
{
   private int definitions = 52500;

   //-----------------------------------------------------------------
   //  Prints a message using both local and inherited values.
   //-----------------------------------------------------------------
   public double computeRatio ()
   {
      return (double) definitions/pages;
   }

   //-----------------------------------------------------------------
   //  Definitions mutator.
   //-----------------------------------------------------------------
   public void setDefinitions (int numDefinitions)
   {
      definitions = numDefinitions;
   }

   //-----------------------------------------------------------------
   //  Definitions accessor.
   //-----------------------------------------------------------------
   public int getDefinitions ()
   {
      return definitions;
   }
}
```



RELAÇÕES EM UML

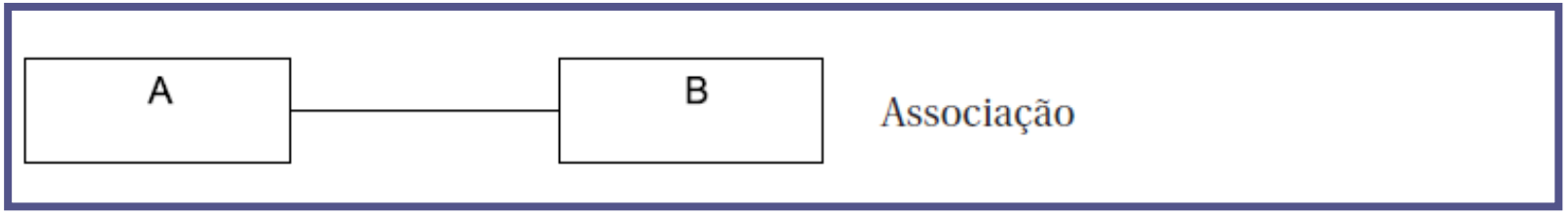
Relações em UML

- Uma relação é uma conexão entre elementos.

- Existem diferentes tipos de relações:
  - Associação
  - Agregação
  - Herança
  - Dependência

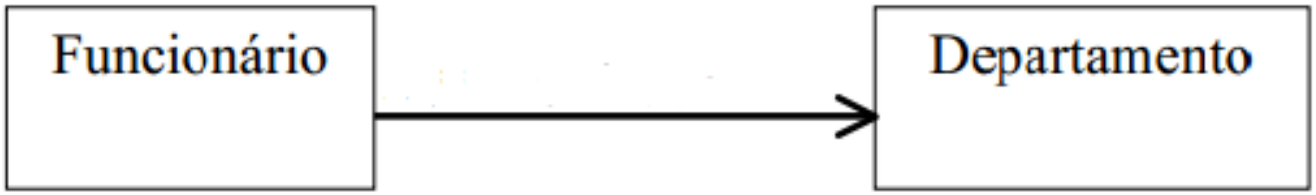
Associação

- Uma **associação** é uma relação entre instâncias de duas classes. Existe uma associação entre duas classes se uma instância de uma classe tem de **conhecer** uma instância da outra classe para poder realizar o seu trabalho.
- Uma associação representa-se por uma linha que une as duas classes.



Navegabilidade

- Nas associações é possível indicar a **navegabilidade** que descreve como a associação deve ser navegada.
- Usa-se como notação a seta aberta. Por exemplo, a classe **Funcionário** tem uma referência a **Departamento**.



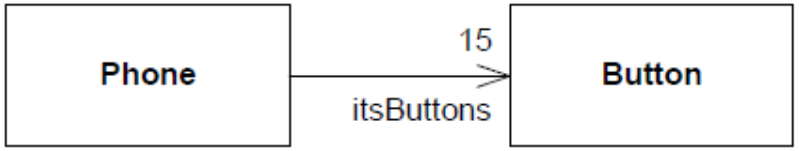
Multiplicidade (cardinalidade)

- A **multiplicidade** de uma associação traduz o nº de instâncias de uma classe que se podem associar com uma única instância da outra classe.
- Quando a multiplicidade é omitida, considera-se que é exatamente1.

Indicador	Multiplicidade
`0..1`	zero ou um
`1`	um
`0..*` ou apenas *	zero ou mais
`1..*`	um ou mais
`n..*`	n ou mais (com $n > 1$)
`n`	apenas n (com $n > 1$)
`0..n`	zero a n (com $n > 1$)
`1..n`	um a n (com $n > 1$)
`n..m`	n a m (com $n > 1$, $m > 1$ e $n < m$)

Exemplo

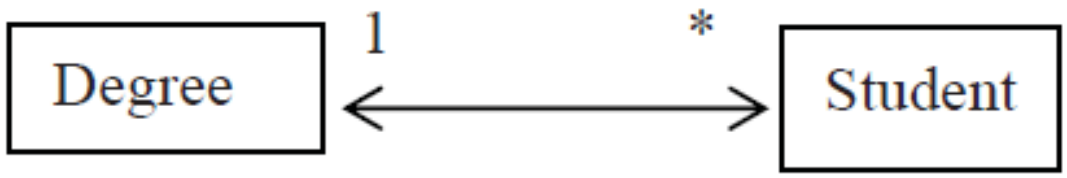
- Geralmente uma associação significa que um objeto de uma classe tem como atributo objetos de outra classe (muitas vezes uma coleção de objetos). Neste exemplo, Phone tem como atributo 1 array com 15 referências de Button.




```
public class Phone
{
  private Button itsButtons[15];
}
```

Associação Bidireccional

- Neste caso, objetos das duas classes têm de conhecer-se, isto é, têm referências um do outro:



Agregação

- Uma **agregação** é uma associação especial em que podemos dizer que um objeto é parte de outro. É uma relação "**tem um**".
- Denota uma relação do **todo** ser formado por **partes**.
- Podemos ter:
  - Agregação partilhada.
  - Agregação composta (ou composição).

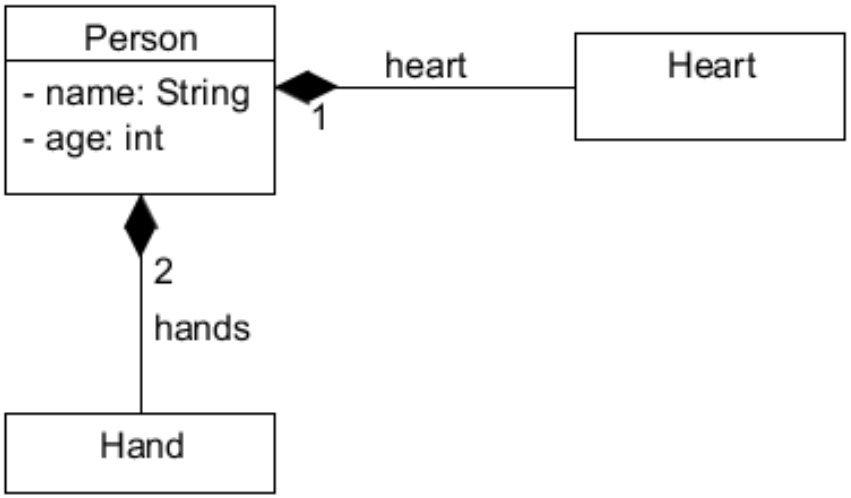
Agregação Partilhada

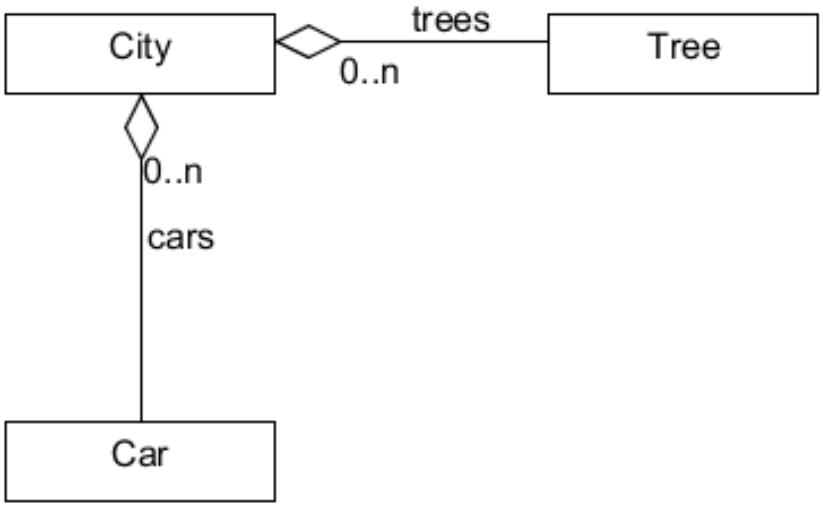
- Um objeto da classe B pertence ao objeto da classe A, mas sem exclusividade. Isto é, objetos da classe B têm uma existência independente da classe A.
- Representa-se colocando um losango vazio no lado do objeto que contém o outro.

Agregação Composta (Composição)

- Denota uma relaçao muito mais forte, em que a classe B é parte da classe A.
- Objetos de B são parte integral de A. Se A não existe, B não existe.
- Representa-se colocando um losango a cheio no lado do objeto (**composto**) que contém o outro (**componente**).

Exemplos


Person
- name: String
- age: int
heart
1
Heart
2
hands
Hand


City
trees
0..n
Tree
0..n
cars
Car

Associação/ Agregação

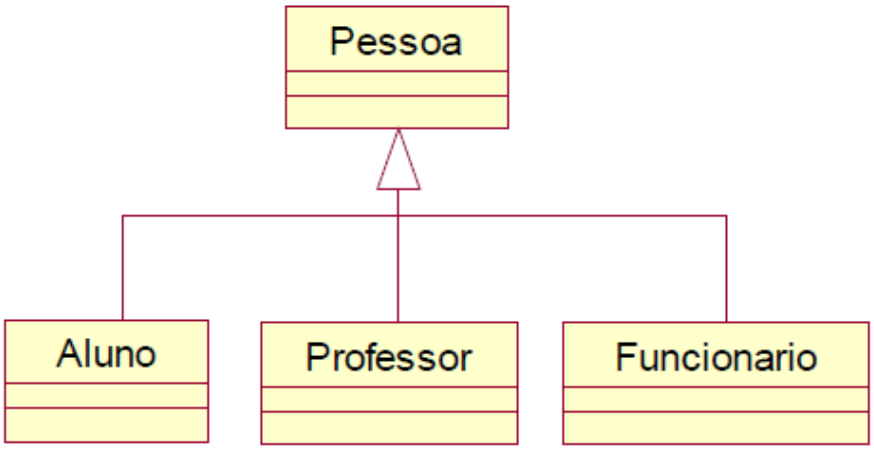
- Como regra, representamos como atributos os de tipo primitivo (por exemplo int, double) ou de classes de utilização comum como a classe **String**.
- Os atributos que correspondam a outras classes por nós desenvolvidas devem ser representados de forma implícita utilizando a notação gráfica para as associações.
- Só no código todos terão que surgir: os que já estavam dentro da classe no diagrama e os que resultam das especificações gráficas.

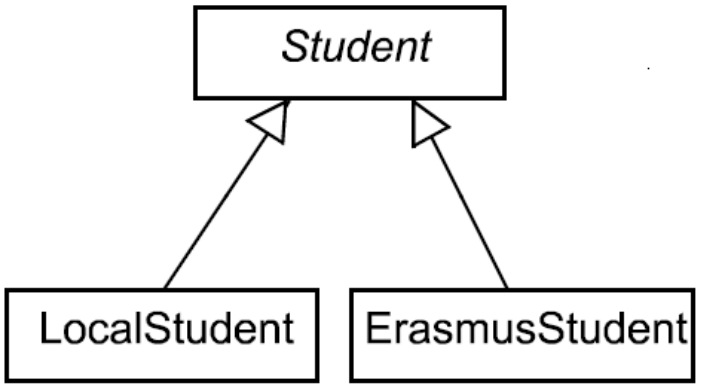
Herança

- A **herança** é um mecanismo em que a subclasse constitui uma especialização da superclasse. A superclasse pode ser vista como uma **generalização** das subclasses.
- A herança é representada por uma seta não preenchida das subclasses para a superclasse (A é uma subclasse de B).

Herança

- Exemplos:





Dependência

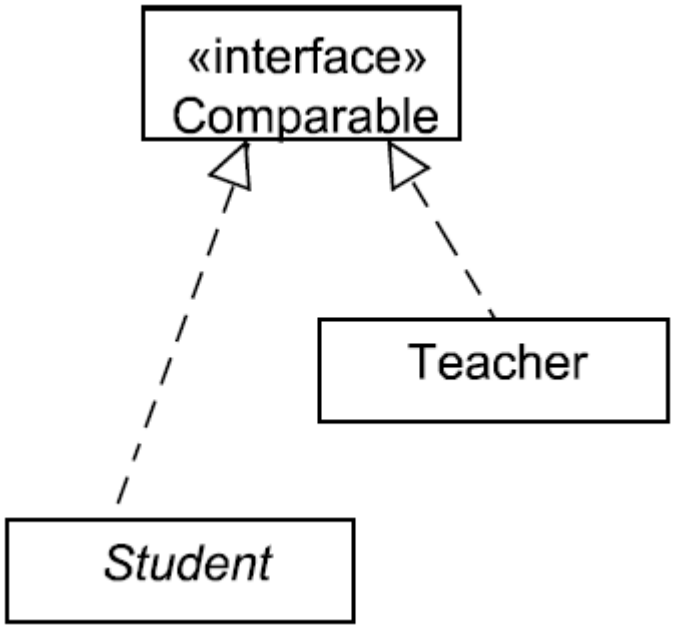
- Podemos ter relações de **dependência** do seguinte tipo:
  - Realização de interfaces
  - Utilização

Realização de Interfaces

- A **realização de interfaces** aplica-se no caso em que uma classe **implementa** (*implements*) uma interface.
- A realização é representada por uma linha a tracejado e uma seta não preenchida do lado da interface.

Realização de Interfaces

- Exemplo: **Student** e **Teacher** implementam a interface **Comparable**.



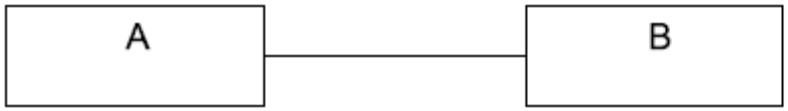
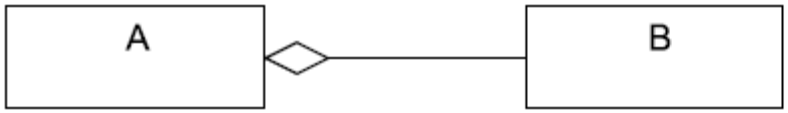
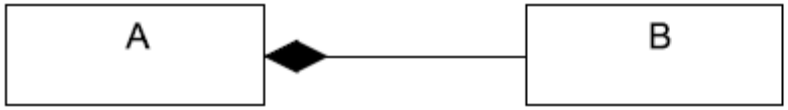
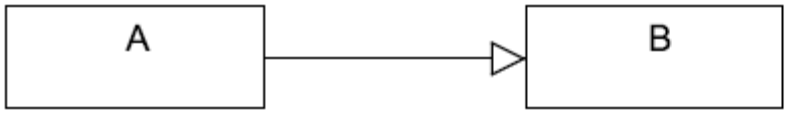
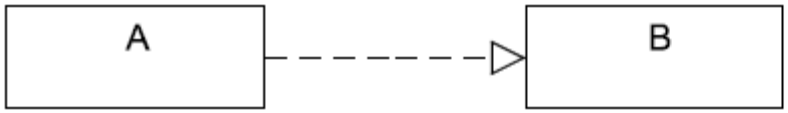
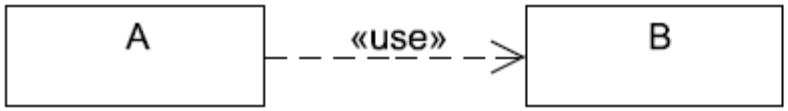
Utilização

- A **utilização** é uma forma de dependência no código.
- Dizemos que um objeto da classe A depende de outro da classe B quando necessita desse objeto para a sua definição.
- Note-se que outras relações anteriores também implicam dependência. A relação de utilização deve ser utilizada apenas quando nenhuma das outras se aplica.
- Notação:

Utilização

- Duas situações típicas em que existe **utilização** podendo não existir **associação** ou **herança**:
  - 1. Quando o objeto da classe A **utiliza** um objeto da classe B como variável local de um método, mas não o guarda como atributo.
  - 2. Quando o objeto da classe A **utiliza** um objeto da classe B que recebeu como parâmetro mas não o guarda como atributo.

Sumário de Notações

Notação	Relação
	Associação
	Agregação partilhada
	Agregação composta
	Generalização/herança (**`extends`**)
	Realização de interface (**`implements`**)
	Utilização

Vantagens do uso de UML

- Descrição e análise dos aspetos essenciais de uma aplicação a desenvolver.
- Representa um passo intermédio entre o enunciado do problema e a sua solução.
- Permite detetar omissões e inconsistências.
- Aumenta a legibilidade – menos informação que o código, permitindo visualizar a aplicação de uma forma global.
- Mostra a estrutura da aplicação, sem detalhes da implementação.

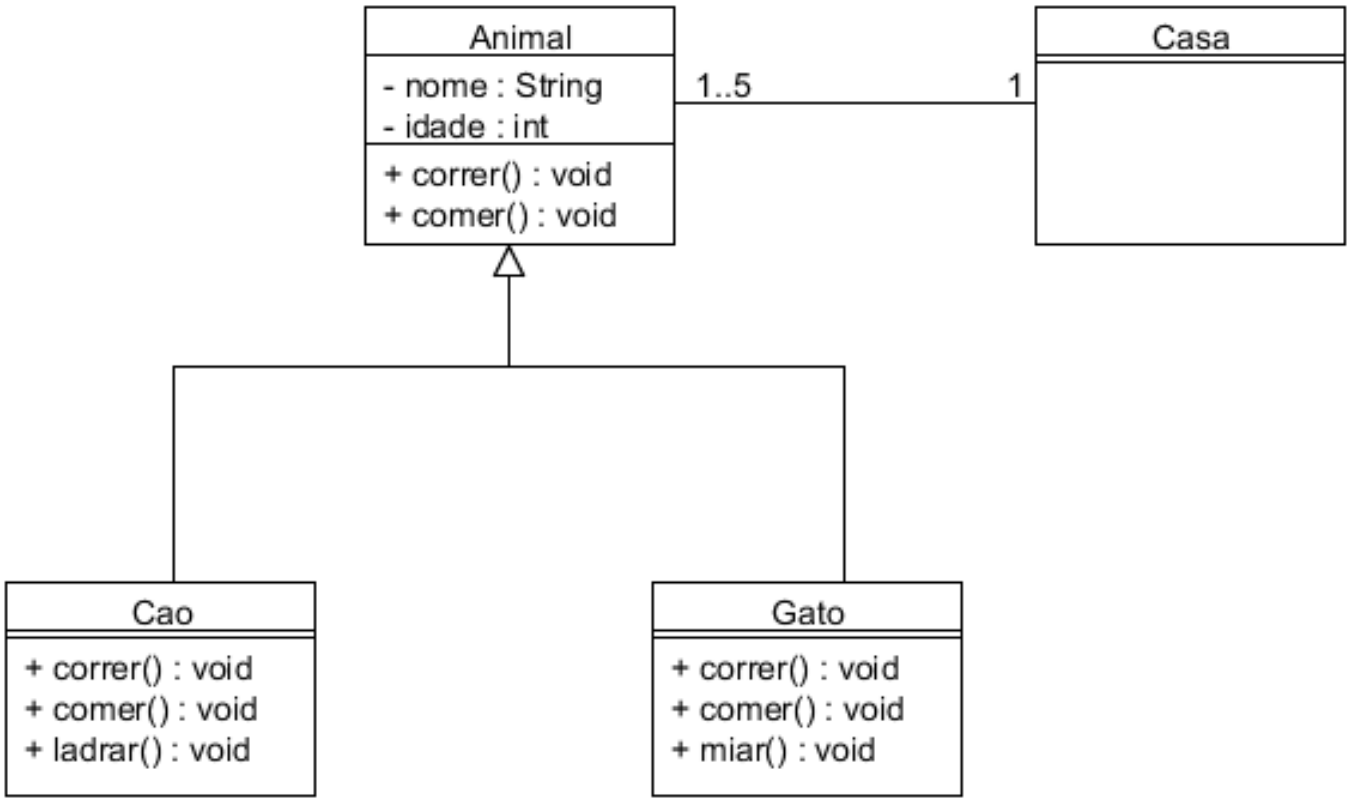
Ferramentas de desenho UML

- Podemos desenhar os diagramas manualmente.
- Mas existem várias ferramentas. Por exemplo:
  - **ArgoUML** (http://argouml.tigris.org/)
  - **UMLet** (http://www.umlet.com/)

Exercício

- Construa um diagrama de classes para a seguinte situação:
  - Numa casa podem viver no máximo 5 animais de estimação.
  - Um animal de estimação é caracterizado pelo nome e idade.
  - Os dois tipos de animal de estimação são o cão e o gato.
  - Os gatos e os cães correm e comem. Além disso, os cães ladram e os gatos miam.

Solução



O diagrama foi realizado com o UMLet

EXERCÍCIO

Elabore um diagrama de classes para a seguinte situação:

Considere uma empresa de fabrico/comercialização de produtos. A empresa é caraterizada por um nome, ano de fundação e um código postal (composto por número, extensão e zona). As pessoas relacionadas com esta empresa são caraterizadas pelo nome, contribuinte, idade, e código postal.

A empresa lida com três tipos de pessoas: fornecedores, empregados e clientes. Os empregados da empresa podem ser de três tipos: administrador, operário e vendedor. Um determinado vendedor possui um ou vários clientes.

A empresa fabrica produtos compostos por matérias-primas, que por sua vez são fornecidas por fornecedores. Um determinado fornecedor pode fornecer várias matérias-primas e uma determinada matéria-prima pode ser fornecida por vários fornecedores.

Primeira etapa

- Uma estratégia possível consiste em analisar o enunciado de forma a identificar palavras-chave que poderão indicar classes a utilizar na situação em causa.

Palavras Identificadas

Considere uma **empresa** de fabrico/comercialização de produtos. A empresa é caraterizada por um nome, ano de fundação e um código postal (composto por número, extensão e zona). As **pessoas** relacionadas com esta empresa são caraterizadas pelo nome, contribuinte, idade, e **código postal**.

A empresa lida com três tipos de pessoas: **fornecedores**, **empregados** e **clientes**. Os empregados da empresa podem ser de três tipos: **administrador**, **operário** e **vendedor**. Um determinado vendedor possui um ou vários clientes.

A empresa fabrica **produtos** compostos por **matérias-primas**, que por sua vez são fornecidas por fornecedores. Um determinado fornecedor pode fornecer várias matérias-primas e uma determinada matéria-prima pode ser fornecida por vários fornecedores.

Classes Identificadas

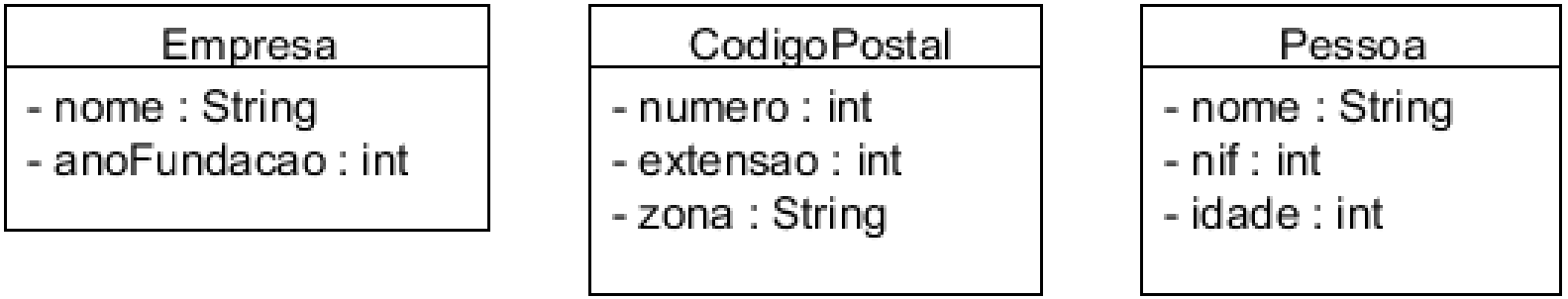
- Empresa
- Pessoa
- CódigoPostal
- Empregado
- Fornecedor
- Cliente
- Administrador
- Operario
- Vendedor
- Produto
- MateriaPrima

Segunda etapa

- Incluir as classes no diagrama e as suas relações de forma gradual. Uma estratégia é ir construindo/modificando frase a frase.

Construção do diagrama

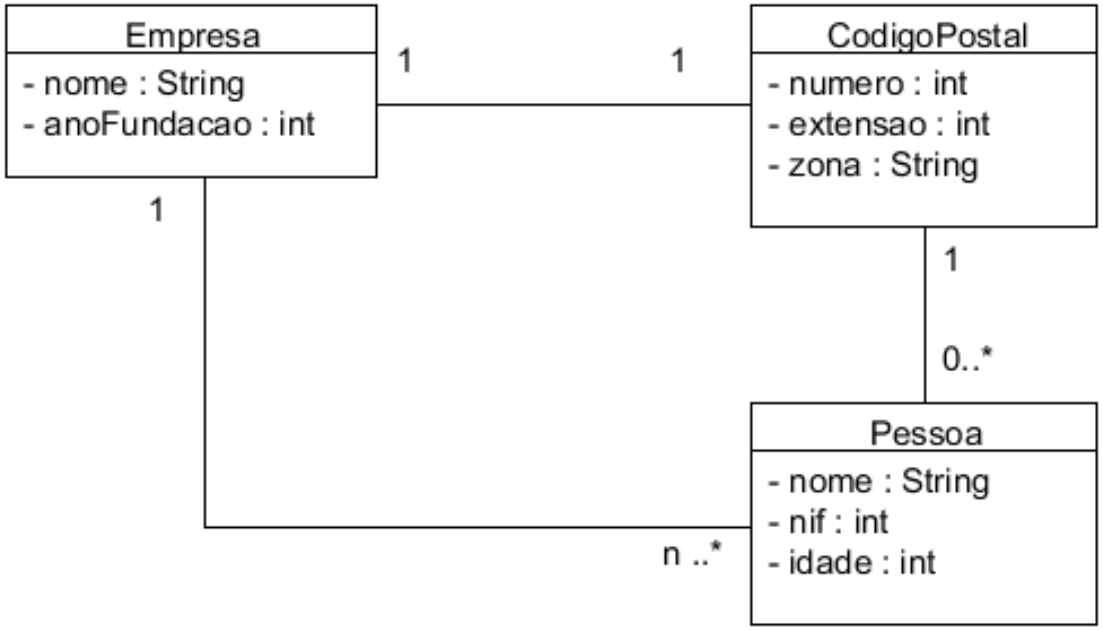
- A empresa é caraterizada por um nome, ano de fundação e um código postal (composto por número, extensão e zona). As pessoas relacionadas com a empresa são caraterizadas pelo nome, contribuinte, idade, e código postal.



Os diagramas foram realizados com o UMLet

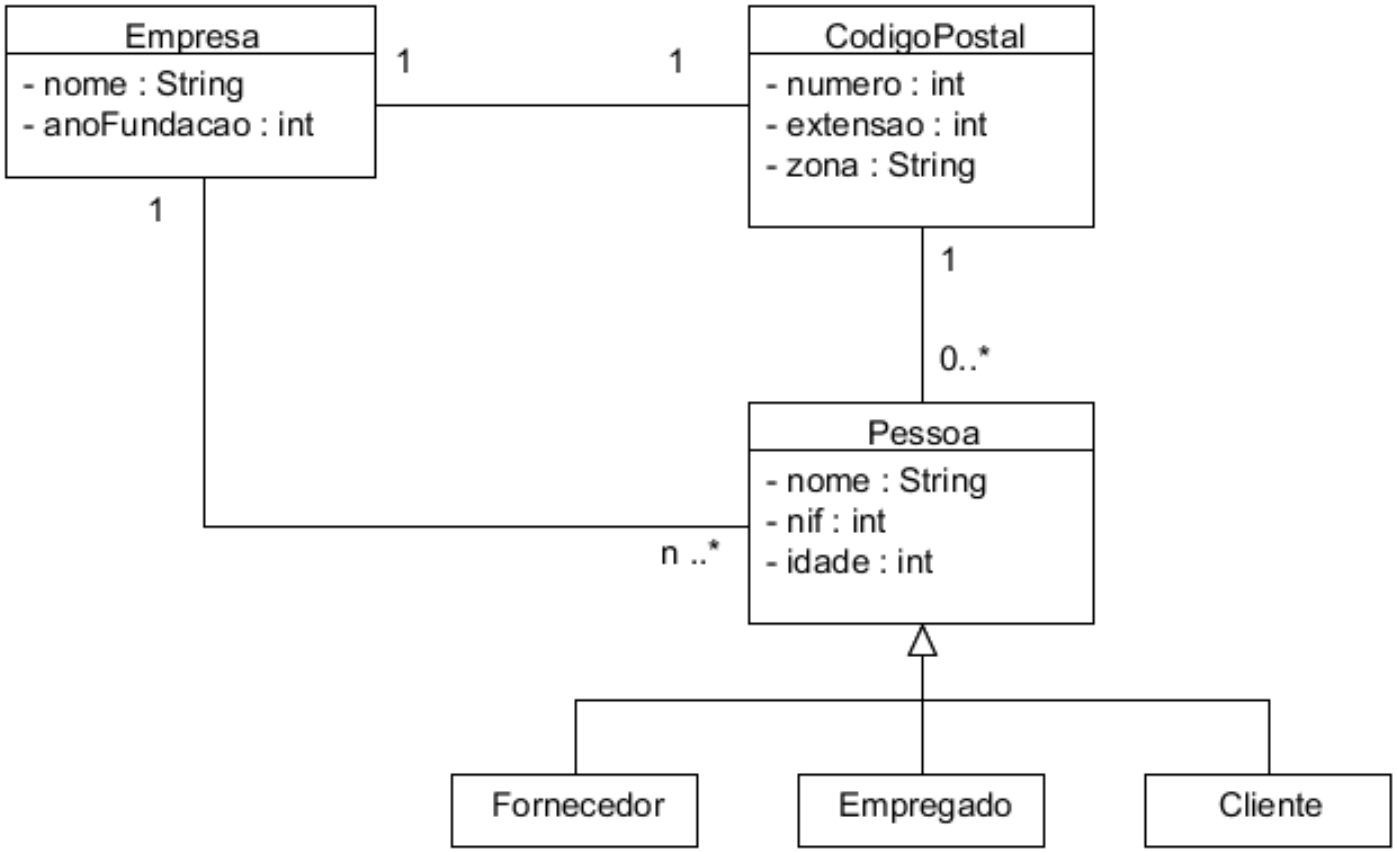
Construção do diagrama

- A empresa é caraterizada por um nome, ano de fundação e um código postal (composto por número, extensão e zona). As pessoas relacionadas com a empresa são caraterizadas pelo nome, contribuinte, idade, e código postal.



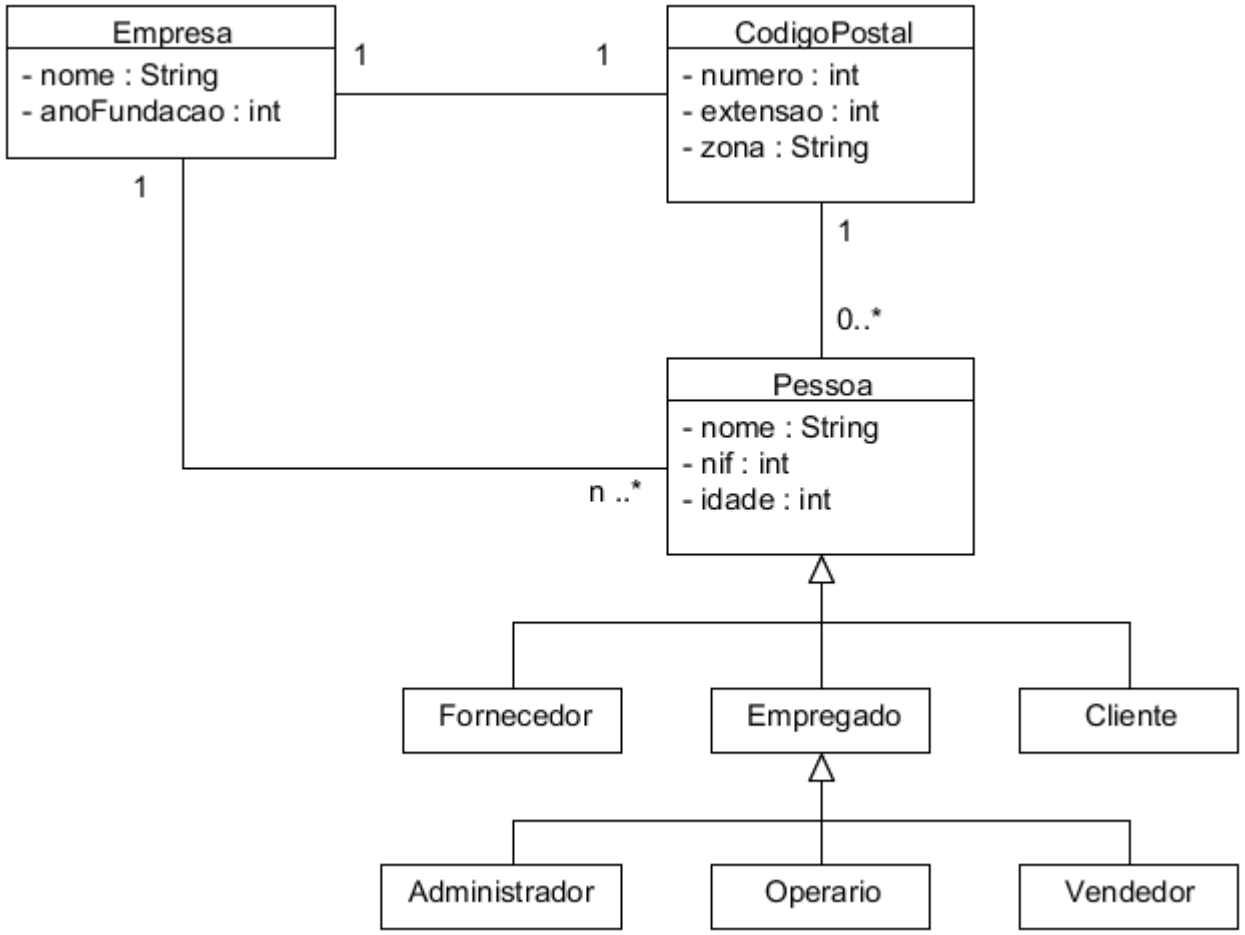
Construção do diagrama

A empresa lida com três tipos de pessoas: fornecedores, empregados e clientes.



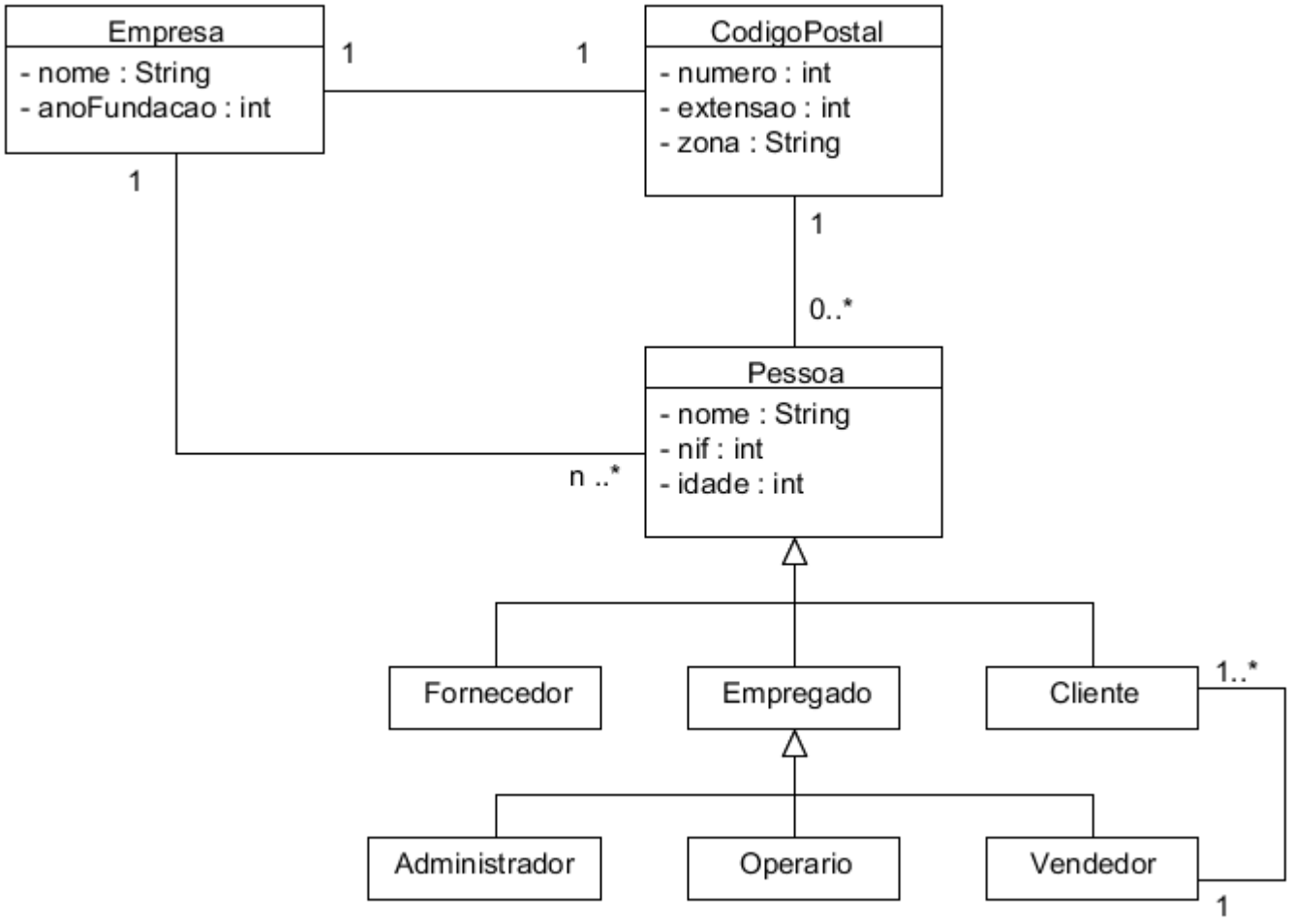
Construção do diagrama

Os empregados da empresa podem ser de três tipos: administrador, operário e vendedor.



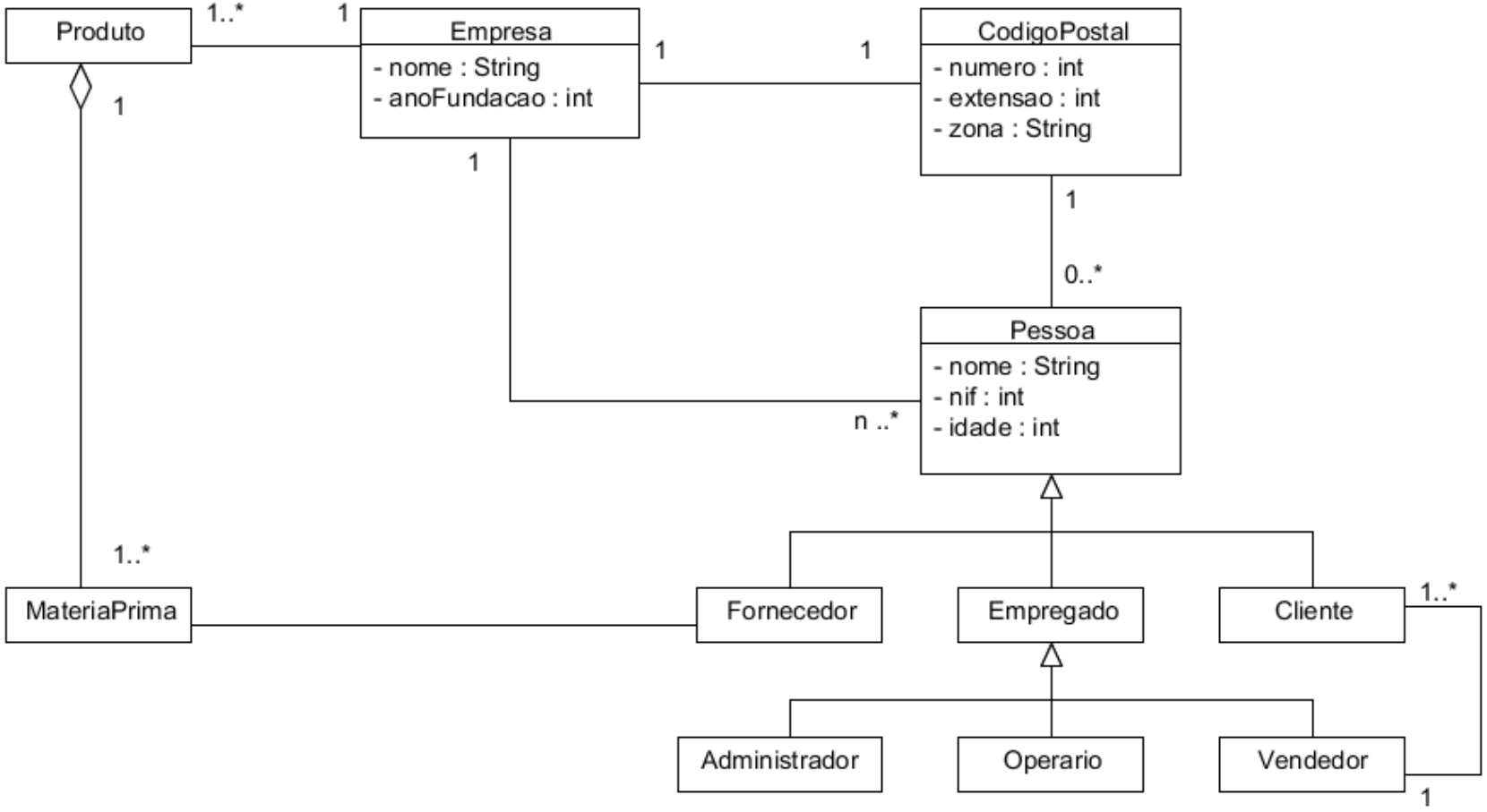
Construção do diagrama

Um determinado vendedor possui um ou vários clientes.



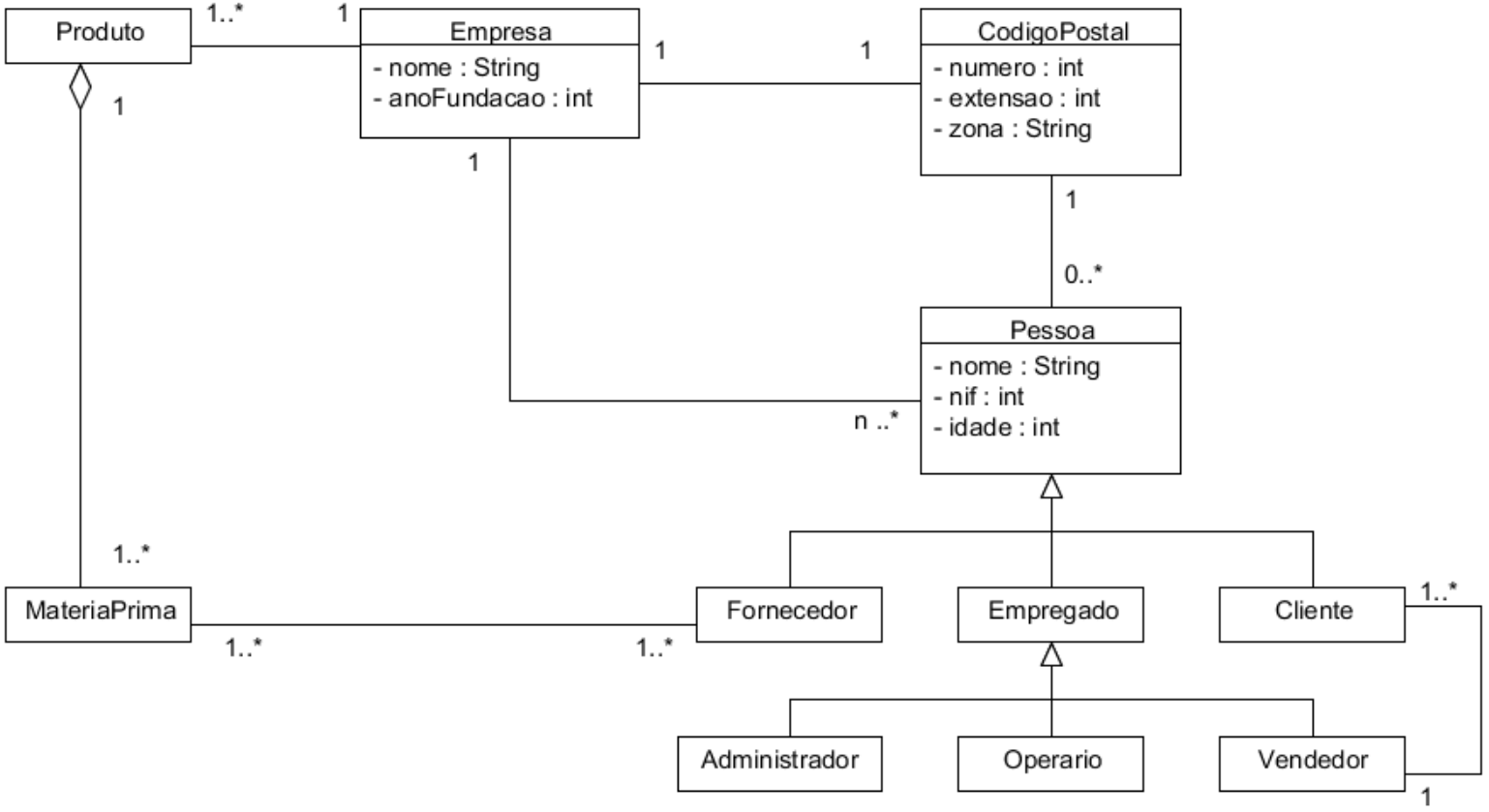
Construção do diagrama

A empresa fabrica produtos compostos por matérias-primas, que por sua vez são fornecidas por fornecedores.



Construção do diagrama

Um determinado fornecedor pode fornecer várias matérias-primas e uma determinada matéria-prima pode ser fornecida por vários fornecedores.



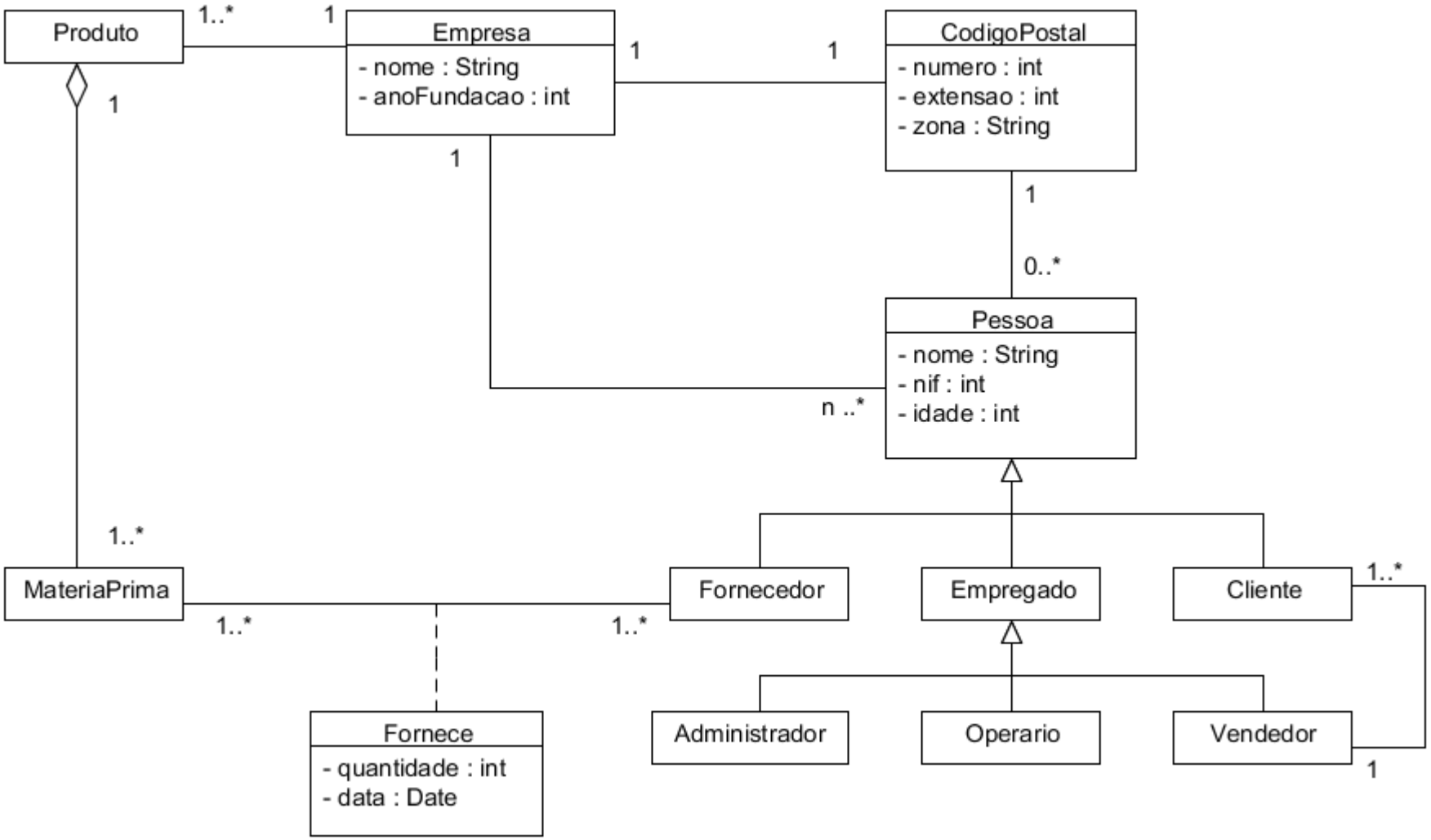
Construção do diagrama

- Considere agora que se adiciona a seguinte descrição ao nosso enunciado:

"*Cada fornecimento de matéria-prima é caraterizado por uma determinada quantidade e data de fornecimento*".

- O que podemos adicionar ao diagrama?

Construção do diagrama



Referências

- Java Software Solutions, Foundations of Program Design, John Lewis, William Loftus, Addison Wesley
- Um pouco sobre diagramas de classes em UML, João Paulo Barros, 2009.
- UML for Java Programmers, Robert Cecil Martin, Prentice-Hall, 2002