



# Diagrama de Classes

---

**Universidade Federal do Maranhão - UFMA**

**Departamento de Informática**

**Processo de Desenvolvimento de Software**

Prof<sup>a</sup>.MSc Simara Rocha

[simararocha@gmail.com](mailto:simararocha@gmail.com)/[simara@deinf.ufma.br](mailto:simara@deinf.ufma.br)

Referências: Booch, G. et al. The Unified Modeling Language User Guide  
Medeiros, E. Desenvolvendo Software com UML 2.0: Definitivo, Makron Books, 2006.



# Sumário

---

- Introdução
- Representação
- Visibilidade do Atributo
- Propósito
- Como Identificar
- Relacionamentos entre as Classes



# Introdução

---

- Uma classe é a descrição de um tipo de objeto.
- Todos os objetos são instâncias de classes, onde a classe descreve as propriedades e comportamentos daquele objeto.
- Identificar as classes de um sistema pode ser complicado, e deve ser feito por *especialistas* no domínio do problema a que o software modelado se baseia.



# Introdução

---

- Um diagrama de classes denota a estrutura estática de um sistema e as classes representam coisas que são manipuladas por esse sistema.
- O diagrama de classes é considerado estático já que a estrutura descrita é sempre válida durante o ciclo de vida do sistema.
- A notação utilizada para representar o diagrama de classes em UML é fortemente baseada na notação de Diagramas Entidade-Relacionamento [CHE90] e no Modelo de Objetos de OMT [RUM94].



# Propósito do Diagrama de Classes

---

- Fazer modelagem do vocabulário do sistema
  - Indica quais abstrações fazem parte do sistema e quais estão fora do limite do mesmo
- Fazer a modelagem e colaboração simples
  - Mostra como as classes trabalham em conjunto permitindo a compreensão de uma semântica maior do que se estas mesmas classes fossem analisadas individualmente.



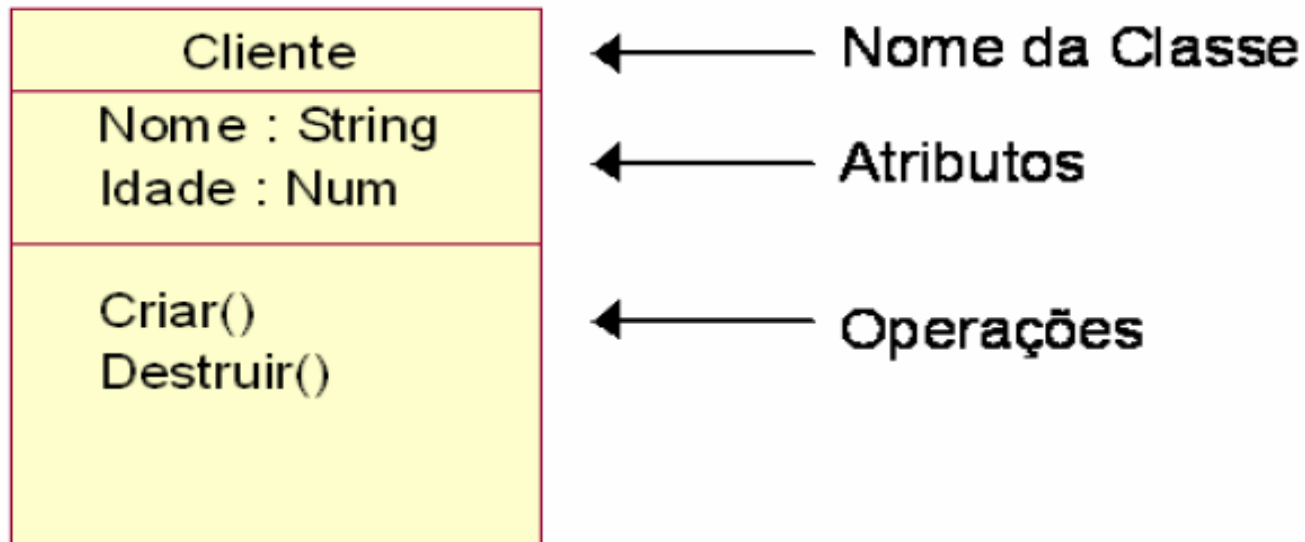
# Propósito do Diagrama de Classes

---

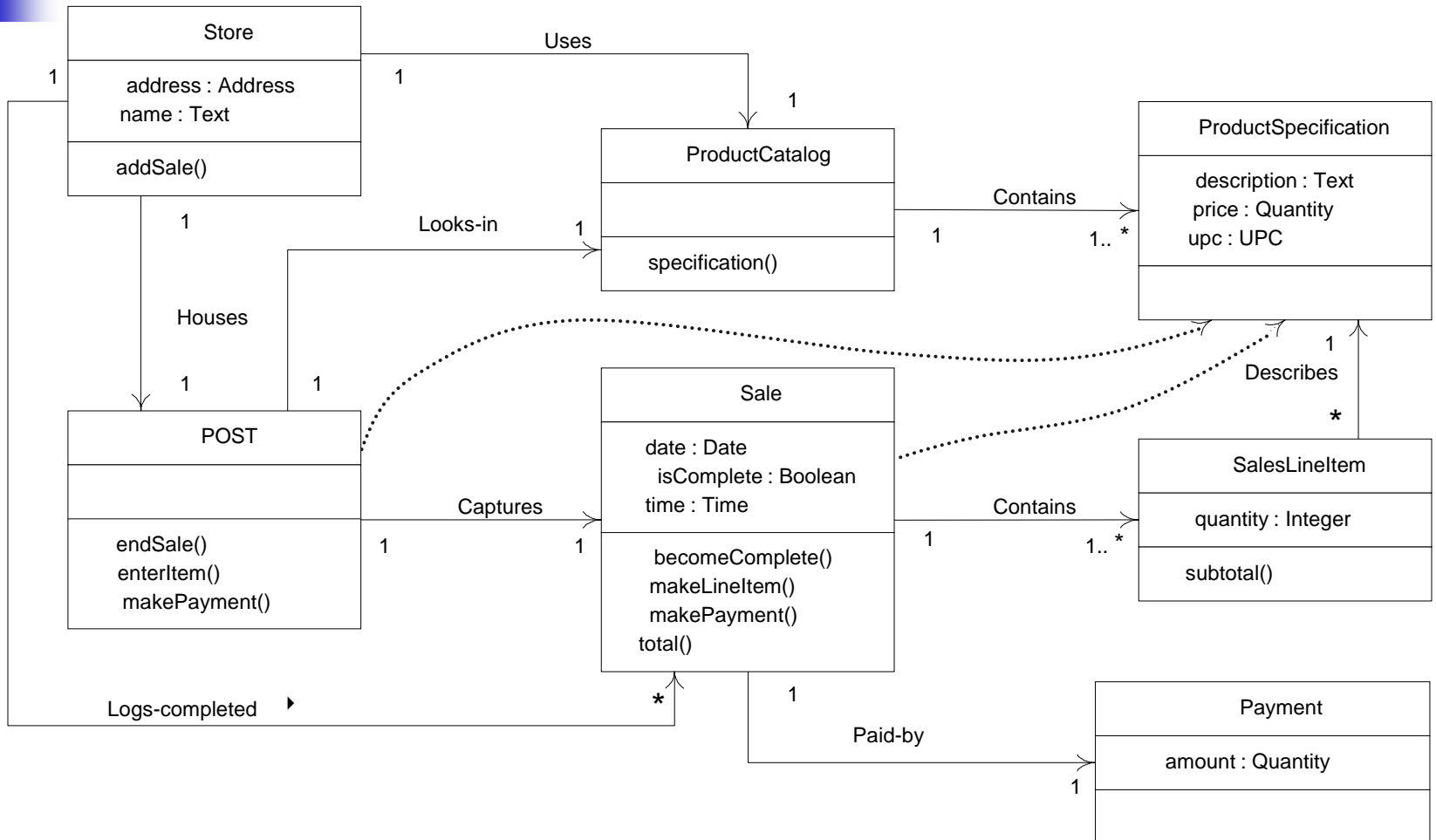
- Fazer a modelagem do esquema lógico de um banco de dados
  - Descreve, de forma orientada a objetos, o esquema lógico de um banco de dados que fisicamente pode ser orientado a objetos, relacional ou híbrido.

# Representação

- Uma classe é representada por um retângulo sólido com três partes:
  - uma para o nome da classe;
  - outra para os atributos da classe;
  - e a terceira para a declaração das operações definidas para a classe.



# Um exemplo







# Perspectivas de um Diagrama de Classes

---

- Conceitual: forma abstrata de se observar classes e objetos e independente da linguagem de programação (Modelo Conceitual)
- De Implementação: pensa-se em detalhes de implementação para definir as classes e objetos.



# Perspectiva Conceitual

---

- Conceitual: representa uma entidade, "coisa", processo ou conceito do mundo real e que possui:
  - Identidade – valor de uma característica que o identifica para reconhecimento
  - Atributos – qualidades, características
  - Comportamento – habilidades de processamento



# Perspectiva de Implementação

---

- De implementação: representa um módulo de sw que recebe e produz dados
  - Identidade – identificador em linguagem de implementação
  - Atributos – variáveis e seus tipos, que recebem diferentes valores e definem o estado do objeto
  - Comportamento – funções ou procedimentos, os resultados dessas funções determinam o comportamento do objeto

# Notação UML para Atributo - visibilidade

- Para poder representar a visibilidade dos atributos e operações em uma classe utiliza-se as seguintes marcas e significados:
  - + público: visível em qualquer classe
  - # protegido: qualquer descendente pode usar
  - - privado: visível somente dentro da classe





# Notação UML para Atributos - Multiplicidade

---

- Usada para especificar atributos que são arranjos
- Indica dimensão de vetores e matrizes
  - Exemplo: notas[10]
  - matrizDeValores[5,10]



# Notação UML para Atributos - Tipos

---

- Indicam o formato do valores que o atributo pode assumir
- Na visão conceitual o tipo é abstrato
  - Ex: dataDaVenda: tipoData
- Na visão de implementação utilizam-se os tipos da linguagem de programação
  - Ex: salario: float



# Notação UML para Atributos – Valor Inicial e Propriedades

---

- Pode-se indicar o valor ou conteúdo do atributo imediatamente após a sua criação, ou o seu valor default
  - Ex: resultado: int=0
- As propriedades descrevem comentários ou indicações sobre o atributo, podem mostrar se ele é ou não opcional
  - Ex: dataDaVenda {valor constante}



# Notação UML para Métodos

---

- A notação e uso vai depender do tipo de visão no qual estamos trabalhando e podem ser abstratos ou utilizar a notação de uma lg de programação

[Visibili/d]Nome(Parâmetros):[Retorno][{Proprie/ds}]



# Notação UML para Métodos - Parâmetros

- Os parâmetros – dados de entrada e/ou saída para o método são representados por:
  - Nome-do-Parâmetro:Tipo=Valor-Padrão
  - Ex:
    - Visão conceitual  
ImprimirData(data:TipoData)
    - Visão de implementação  
ArmazenarDados(nome:char[30],salario:float=0.0)

# Notação UML para Métodos – Tipo de Retorno

---

- O Valor-de-Retorno indica se o método retorna algum valor ao término de sua execução e qual o tipo de dado do valor retornado.
  - Ex:
    - Visão Conceitual  
CalcularValor(): TipoDinheiro
    - Visão de implementação  
ArmazenarDados(nome:char[30]): bool



# Identificação de Classes com Estereótipos

---

- Estereótipo é um classificador.
- Tipos:
  - Entidade: representam conceitos do mundo real e armazenam dados que os identificam
  - Controle: controlam a execução de processos e o fluxo de execução de todo ou de parte de casos de uso e comandam outras classes
  - Fronteira: realizam o interfaceamento com entidades externas (atores), contém o protocolo de comunicação com impressora, monitor, teclado, disco, porta serial, modem, etc.



# Exemplo de Uma Classe

## Visão Conceitual

Aluno
nome:TipoNome RA: TipoCódigo
calculaMédia():TipoNota

## Visão de Implementação

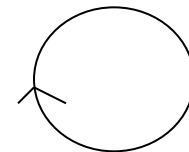
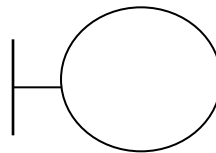
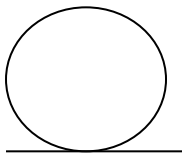
<<entidade>> Aluno DePacoteCadastro
-nome[30]:char +RA: int {valorconstante}
+calculaMédia():Double

# Exemplos no Sistema Posto Comercial

<<entidade>>  
Itens

<<fronteira>>  
InterfaceCliente

<<controle>>  
ControleComprarItens





# Relacionamentos

---

- Os relacionamentos ligam as classes/objetos entre si criando relações lógicas entre estas entidades.
- Os relacionamentos podem ser dos seguintes tipos:
  - Associação
  - Generalização
  - Dependência e Refinamentos



# Associação

---

- Uma associação representa que duas classes possuem uma ligação (link) entre elas, significando, por exemplo, que elas “conhecem uma a outra”, “estão conectadas com”, “para cada X existe um Y” e assim por diante.
- Podem ser:
  - Normal, Recursiva, Qualificada, Exclusiva, Ordenada, Classe, Ternária e Agregação.



# Associação Normal

---

- O tipo mais comum de associação é apenas uma conexão entre classes.
- É representada por uma linha sólida entre duas classes.
- A associação possui um nome (junto à linha que representa a associação), normalmente um verbo, mas substantivos também são permitidos.
  - Pode-se também colocar uma seta no final da associação indicando que esta só pode ser usada para o lado onde a seta aponta.
  - Mas associações também podem possuir dois nomes, significando um nome para cada sentido da associação.

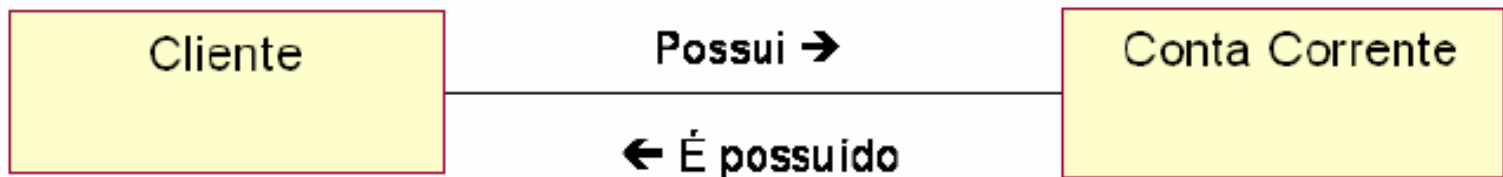




# Associação Normal - Exemplo

---

- No exemplo abaixo vemos um relacionamento entre as classes Cliente e Conta Corrente se relacionam por associação.



# Associações Normal-Cardinalidade (Multiplicidade)

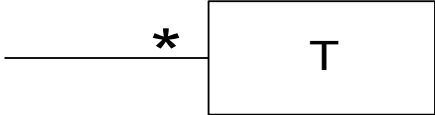
- Especifica o número de objetos de cada classe envolvidos com a associação
- A leitura da cardinalidade é diferente para os diferentes sentidos da associação



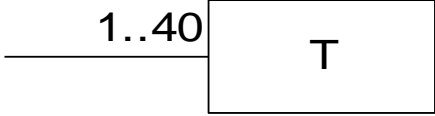


# Cardinalidade

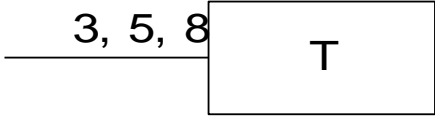
---

 \* T zero or more;  
"many"

 1..\* T one or more

 1..40 T one to forty

 5 T exactly five

 3, 5, 8 T exactly three,  
five or eight



# Associações - Navegabilidade

---

- Mostra a direção de navegação, ou seja do canal de comunicação entre os objetos e classes.
- Uma associação é navegável da classe A para a classe B, se, dado um objeto de A, consegue-se obter de forma direta os objetos relacionados da classe B.



# Associações - Navegabilidade

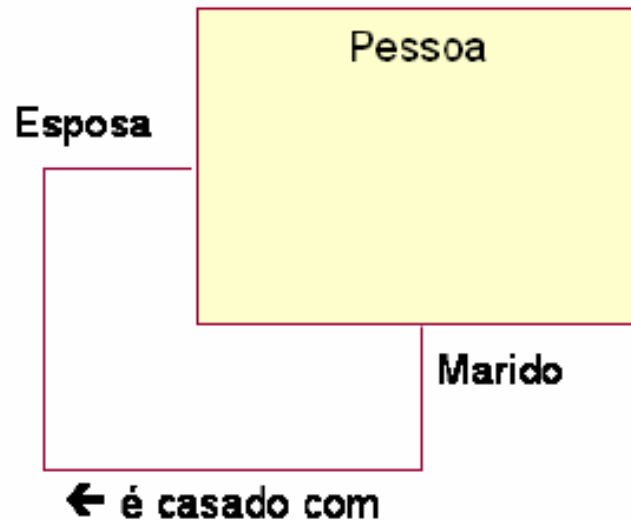
---

- Dada uma mercadoria pode-se identificar diretamente a empresa fornecedora, mas a volta não é verdadeira



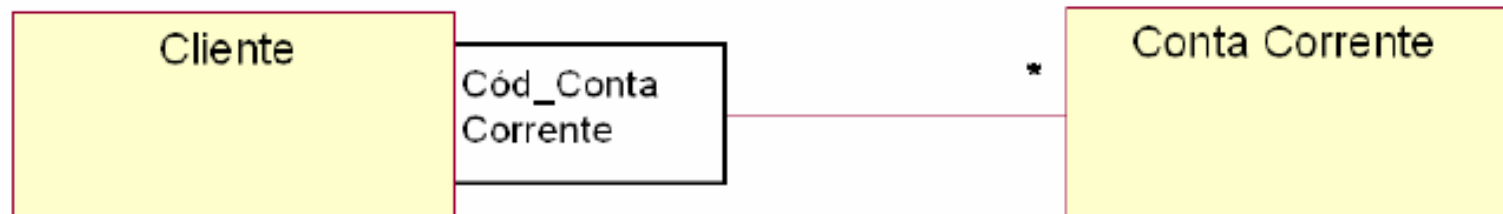
# Associação Recursiva

- É usada quando é necessário conectar uma classe a ela mesma através de uma associação e que ainda representa semanticamente a conexão entre dois objetos, mas os objetos conectados são da mesma classe.



# Associação Qualificada

- São usadas com associações de um para vários (1..\*) ou vários para vários (\*).
- O “qualificador” (identificador da associação qualificada) especifica como um determinado objeto no final da associação “n” é identificado, e pode ser visto como um tipo de chave para separar todos os objetos na associação.
- O identificador é desenhado como uma pequena caixa no final da associação junto à classe de onde a navegação deve ser feita.





# Associação Exclusiva

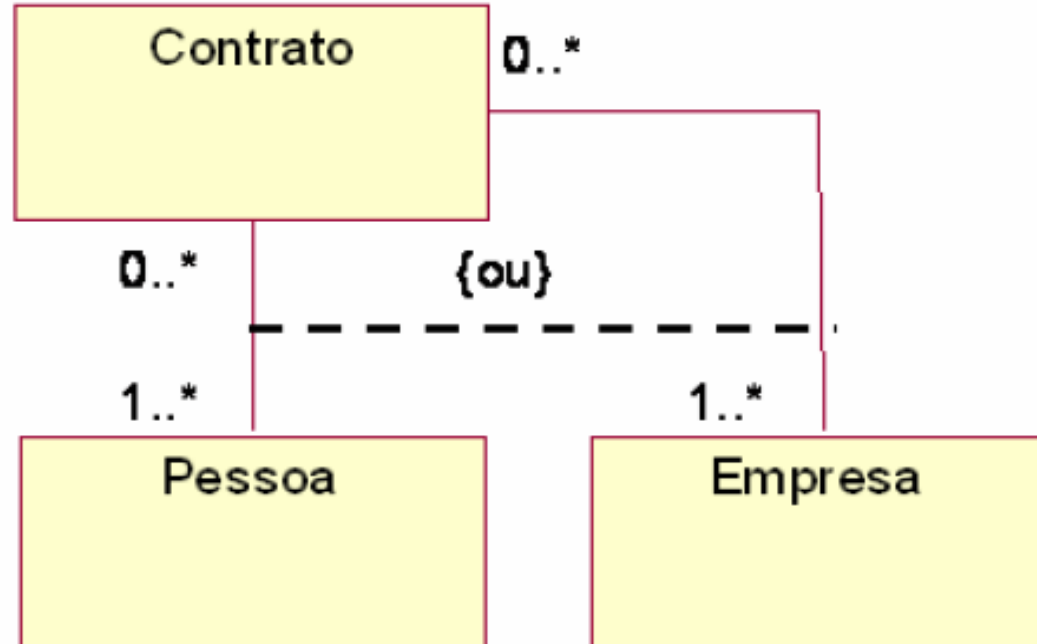
---

- Em alguns modelos nem todas as combinações são válidas, e isto pode causar problemas que devem ser tratados.
- Uma associação exclusiva é uma restrição em duas ou mais associações.
- Ela especifica que objetos de uma classe podem participar de no máximo uma das associações em um dado momento.
- Uma associação exclusiva é representada por uma linha tracejada entre as associações que são partes da associação exclusiva, com a especificação "{ou}" sobre a linha tracejada.



# Associação Exclusiva - Exemplo

- No exemplo abaixo um contrato não pode se referir a uma pessoa e a uma empresa ao mesmo tempo, significando que o relacionamento é exclusivo a somente uma das duas classes.





# Associação Ordenada

---

- As associações entre objetos podem ter uma ordem implícita.
- O padrão para uma associação é desordenado (ou sem nenhuma ordem específica). Mas uma ordem pode ser especificada através da associação ordenada.



# Associação Ordenada

---

- Este tipo de associação pode ser muito útil em casos como este:
  - janelas de um sistema têm que ser ordenadas na tela (uma está no topo, uma está no fundo e assim por diante).
- A associação ordenada pode ser escrita apenas colocando “{ordenada}” junto à linha de associação entre as duas classes.



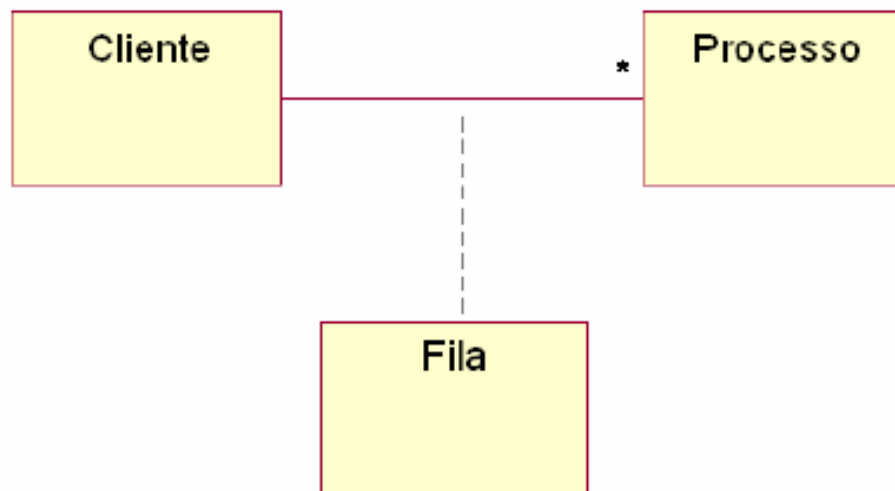
# Associação de Classe

---

- Uma classe pode ser associada a uma outra associação, não sendo conectada nenhuma das extremidades da associação já existente, mas na própria linha da associação.
- Esta associação serve para se adicionar informação extra a associação já existente.

# Associação de Classe - Exemplo

- A associação da classe Fila com a associação das classes Cliente e Processo pode ser estendida com operações de adicionar processos na fila, para ler e remover da fila e de ler o seu tamanho.
- Se operações ou atributos são adicionados a associação, ela deve ser mostrada como uma classe.





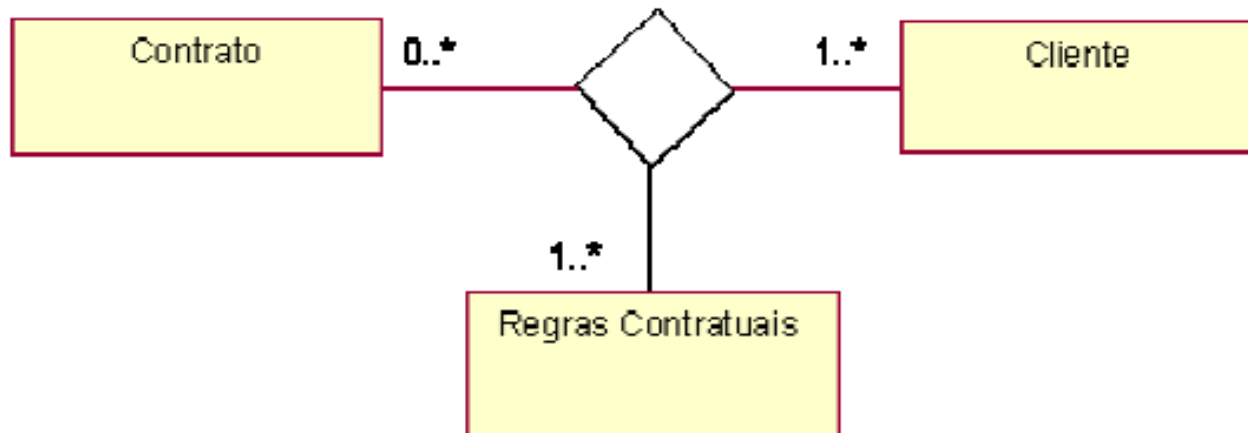
# Associação Ternária

---

- Mais de duas classes podem ser associadas entre si, a associação ternária associa três classes.
- Ela é mostrada como um grade losango e ainda suporta uma associação de classe ligada a ela, traçaria-se, então, uma linha a partir do losango para a classe onde seria feita a associação ternária.

# Associação Ternária - Exemplo

- O exemplo abaixo especifica que um cliente poderá possuir 1 ou mais contratos e cada contrato será composto de 1 ou várias regras contratuais.



# Associação de Agregação

- A agregação é um caso particular da associação.
- A agregação indica que uma das classes do relacionamento é uma parte, ou está contida em outra classe.
- As palavras chaves usadas para identificar uma agregação são: “consiste em”, “contém”, “é parte de”.
- Existem tipos especiais de agregação que são as agregações: compartilhadas e as compostas.





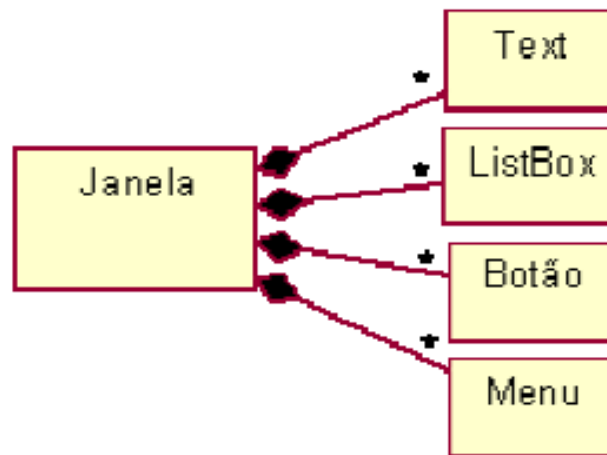
# Agregação Compartilhada

- A agregação é dita compartilhada quando uma das classes é uma parte, ou está contida na outra, mas esta parte pode estar contida na outras várias vezes em um mesmo momento.
- Exemplo: uma pessoa pode ser membro de um time ou vários times e em determinado momento.



# Agregação de Composição

- É uma agregação onde uma classe que está contida na outra "vive" e constitui a outra. Se o objeto da classe que contém for destruído, as classes da agregação de composição serão destruídas juntamente já que as mesmas fazem parte da outra.





# Generalização

---

- A generalização é um relacionamento entre um elemento geral e um outro mais específico.
- O elemento mais específico possui todas as características do elemento geral e contém ainda mais particularidades.
- Um objeto mais específico pode ser usado como uma instância do elemento mais geral.



# Generalização

---

- A generalização, também chamada de herança, permite a criação de elementos especializados em outros.
- Existem alguns tipos de generalizações que variam em sua utilização a partir da situação. São elas: normal, restrita.



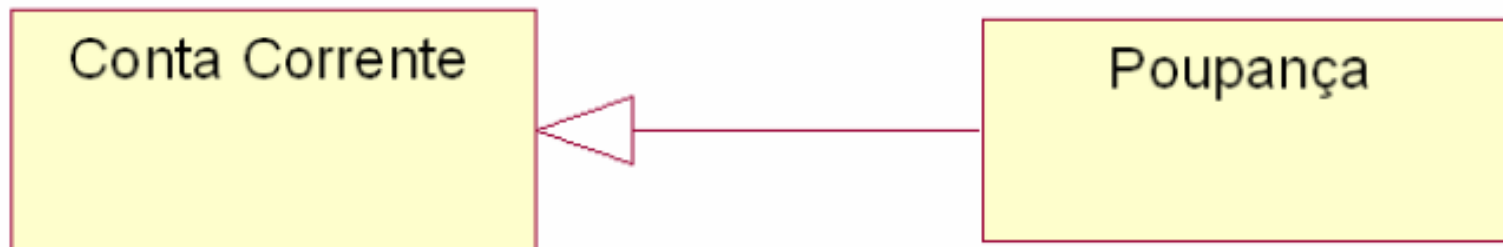
# Generalização Normal

---

- Na generalização normal a classe mais específica, chamada de subclasse, herda tudo da classe mais geral, chamada de superclasse. Os atributos, operações e todas as associações são herdados.
- Uma classe pode ser tanto uma subclasse quanto uma superclasse, se ela estiver numa hierarquia de classes que é um gráfico onde as classes estão ligadas através de generalizações.

# Generalização Normal - Exemplo

- A generalização normal é representada por uma linha entre as duas classes que fazem o relacionamento, sendo que se coloca uma seta no lado da linha onde se encontra a superclasse indicando a generalização.





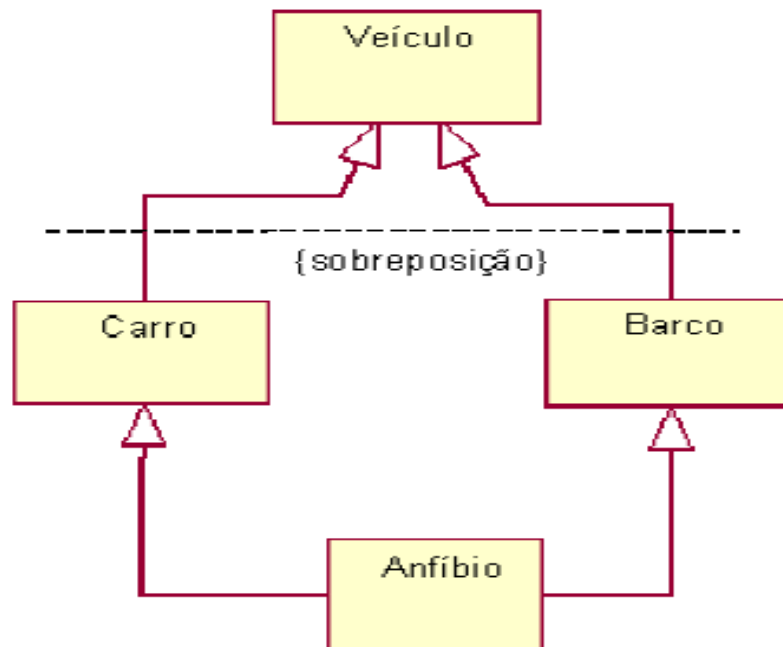
# Generalização Restritiva

---

- Uma restrição aplicada a uma generalização especifica informações mais precisas sobre como a generalização deve ser usada e estendida no futuro.
- Generalizações restritas com mais de uma subclasse:
  - Sobreposição e Disjuntiva
  - Completa e Incompleta

# Generalização Sobreposição e Disjuntiva

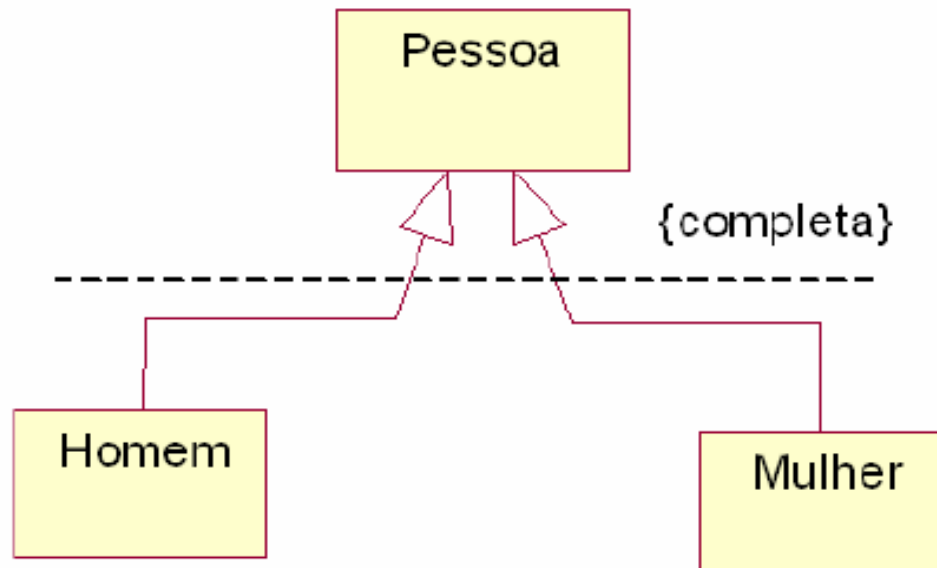
- Generalização de sobreposição ocorre quando subclasses herdam de uma superclasse por sobreposição e novas subclasses destas podem herdar de mais de uma subclasse.
- A generalização disjuntiva é exatamente o contrário da sobreposição e a generalização é utilizada como padrão.





# Generalização Completa e Incompleta

- Uma restrição simbolizando que uma generalização é completa significa que todas as subclasses já foram especificadas, e não existe mais possibilidade de outra generalização a partir daquele ponto.
- A generalização incompleta é exatamente o contrário da completa e é assumida como padrão da linguagem.





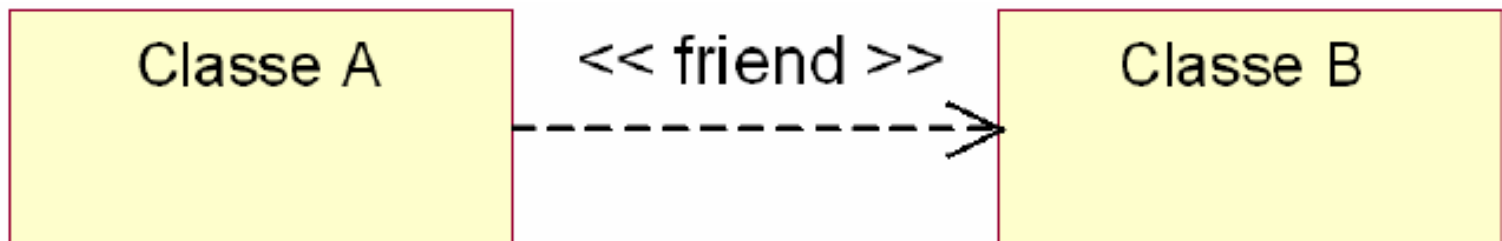
# Dependência e Refinamento

---

- Dependência é uma conexão semântica entre dois modelos de elementos, um independente e outro dependente.
- Uma mudança no elemento independente irá afetar o modelo dependente. Como no caso anterior com generalizações, os modelos de elementos podem ser uma classe, um pacote, um caso de uso e assim por diante.
- Quando uma classe recebe um objeto de outra classe como parâmetro, uma classe acessa o objeto global da outra. Nesse caso existe uma dependência entre estas duas classes, apesar de não ser explícita.

# Dependência e Refinamento

- Uma relação de dependência é simbolizada por uma linha tracejada com uma seta no final de um dos lados do relacionamento. E sobre essa linha o tipo de dependência que existe entre as duas classes.
- As classes “Amigas” provenientes do C++ são um exemplo de um relacionamento de dependência.





# Dependência e Refinamento

---

- Os Refinamentos são um tipo de relacionamento entre duas descrições de uma mesma coisa, mas em níveis de abstração diferentes, e podem ser usados para modelar diferentes implementações de uma mesma coisa (uma implementação simples e outra mais complexa, mas também mais eficiente).
- Os refinamentos são simbolizados por uma linha tracejada com um triângulo no final de um dos lados do relacionamento e são usados em modelos de coordenação.



# Dependência e Refinamento

---

- Em grandes projetos, todos os modelos que são feitos devem ser coordenados.
- Coordenação de modelos pode ser usada para mostrar modelos em diferentes níveis de abstração que se relacionam e mostram também como modelos em diferentes fases de desenvolvimento se relacionam.



# Identificando Classes Entidades – Regras Úteis



---

- É melhor especificar demais do que de menos
- Não exclua entidades simplesmente porque os requisitos não indicam a necessidade de guardar informações sobre eles (comum em projeto de BD)
- Comece fazendo uma lista de entidades candidatos a partir de uma lista
- Considere os substantivos e frases nominais nas descrições textuais do domínio do problema como possíveis candidatos a entidades ou atributos



# Como Identificar?

---

- Questões que podem ajudar a identificá-las:
  - Existem informações que devem ser armazenadas ou analisadas?
  - Existem sistemas externos ao modelado?
    - Se existir, eles deverão ser vistos como classes pelo sistema para que possa interagir com outros externos.
  - Qual o papel dos atores dentro do sistema?
    - Talvez o papel deles possa ser visto como classes, por exemplo, usuário, operador, cliente e daí por diante.



# Identificando Associações

---

- Regras úteis:
  - Focar nas associações cujo conhecimento deve ser preservado
  - Usar nomes baseados em expressões verbais que façam sentido quando lidas no contexto do modelo
  - Evitar mostrar associações deriváveis ou redundantes
  - É mais importante identificar conceitos do que associações
  - Associações demais tendem a confundir um modelo ao invés de iluminá-lo





# Identificando Generalizações

---

- Quando particionar em subclasses?
  - a subclasse tem atributos adicionais de interesse
  - a subclasse tem associações adicionais de interesse
  - a subclasse será manipulada ou usada de maneira diferente da super-classe ou das outras sub-classes
  - a subclasse se comporta diferente da super-classe ou das outras sub-classes



# Identificando Generalizações

---

- Quando criar uma superclasse?
  - as potenciais subclasses representam variações de um conceito similar
  - as subclasses satisfazem 100% a regra "is-a"
  - todas as subclasses possuem um atributo comum que poderá ser expresso na superclasse
  - todas as subclasses possuem uma associação comum que poderá ser relacionada à superclasse



# Identificando Agregações

---

- Verificar se algumas classes podem ser agrupadas em algum composto:
  - elas são geralmente criadas/destruídas no mesmo instante.
  - possuem relacionamentos comuns



# Identificando Agregações

---

- Verificar se alguma classe pode ser subdividida em partes.
  - as partes possuem tempo de vida limitado ao tempo de vida do composto
  - possuem relacionamentos particulares e de interesse



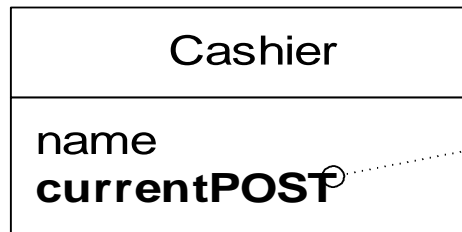
# Identificando Atributos

---

- Atributos devem preferencialmente representar tipos primitivos de dados ou de valores simples
  - Ex.: *Data, Número, Texto, Hora, Endereço*, etc.
- Atributos não devem ser usados para:
  - Representar um conceito complexo
  - Relacionar conceitos (atributo “chave-estrangeira”)

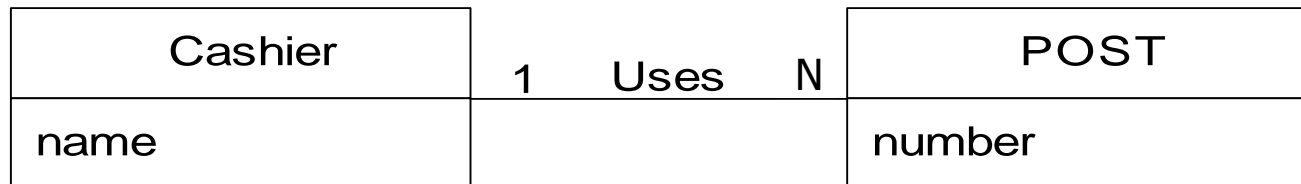
# Identificando Atributos

**Worse**



not a "simple" attribute

**Better**

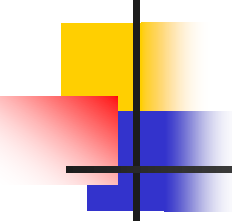




# Atributos de Tipo Não-Primitivo

---

- Um atributo deve ser de tipo não-primitivo quando:
  - É composto de seções separadas
    - Ex.: endereço, data
  - Precisa ser analisado ou validado
    - Ex.: CPF, número de matrícula
  - Possui outros atributos
    - Ex.: Um preço promocional com prazo de validade
  - É uma quantidade com uma unidade
    - Ex.: valores monetários, medidas

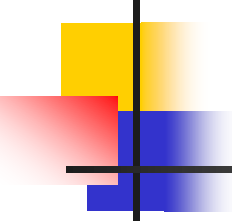


# Identificação dos Métodos

---

- Os métodos são acrescentados na perspectiva de implementação e são derivados a partir dos diagramas de interação: colaboração e sequências.
- É útil distinguir operações de métodos. Operações é algo que se evoca sobre um objeto (a chamada do procedimento). Para realizar uma operação a classe implementa um método (o corpo do procedimento)

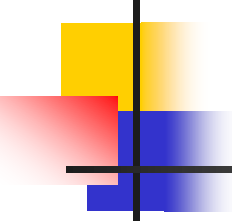




# Identificação dos Métodos

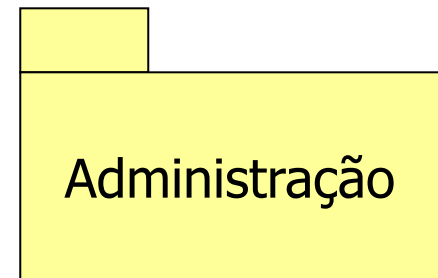
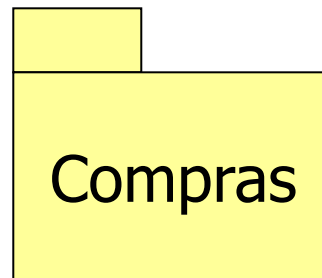
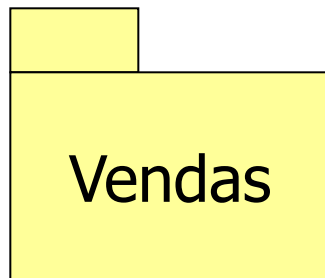
---

- É útil distinguir operações que alteram ou não o estado (atributos) de uma classe
- Não alteram: query, métodos de obtenção, getting methods
- Alteram: modificadores, métodos de atribuição ou fixação, setting methods

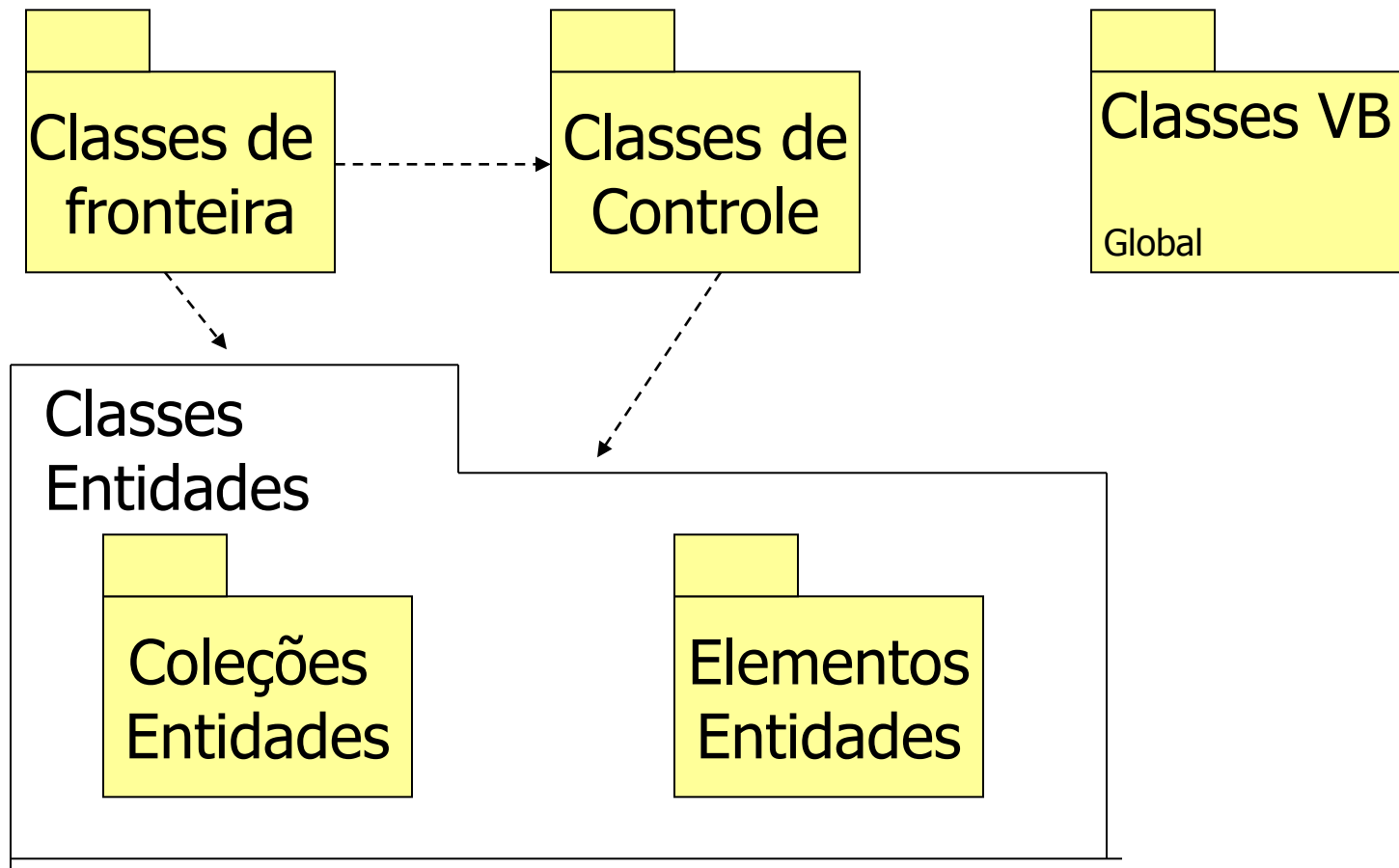


# Organização de Classes em Pacotes Lógicos

---



# Organização de Classes em Pacotes Lógicos





# Mecanismos Gerais

---

- A UML utiliza alguns mecanismos em seus diagramas para tratar informações adicionais, que são:
  - Ornamentos
  - Notas



# Ornamentos

---

- Ornamentos gráficos são anexados aos modelos de elementos em diagramas e adicionam semânticas ao elemento.
- Um exemplo de um ornamento é o da técnica de separar um tipo de uma instância.
  - Quando um elemento representa um tipo, seu nome é mostrado em **negrito**. Quando o mesmo elemento representa a instância de um tipo, seu nome é escrito sublinhado e pode significar tanto o nome da instância quanto o nome do tipo.



# Ornamentos

---

- Outros ornamentos são os de especificação de multiplicidade de relacionamentos, onde a multiplicidade é um número ou um intervalo que indica quantas instâncias de um tipo conectado pode estar envolvido na relação.



# Notas

---

- Nem tudo pode ser definido em uma linguagem de modelagem, sem importar o quanto extensa ela seja.
- Para permitir adicionar informações a um modelo não poderia ser representado de outra forma, UML provê a capacidade de adicionar Notas.
- Uma Nota pode ser colocada em qualquer lugar em um diagrama, e pode conter qualquer tipo de informação.

