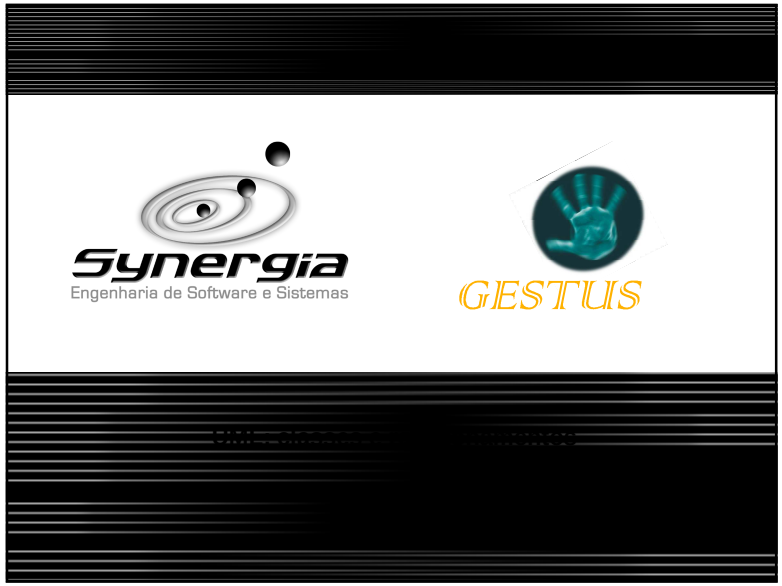


UML: classe e relacionamentos

Prof.: Clarindo Isaías Pereira da Silva e Pádua

Synergia / Gestus

Departamento de Ciência da Computação - UFMG

Bibliografia

- ***Rumbaugh, J.; Jacobson, I.; Booch, G., The Unified Modeling Language Reference Manual, Addison Wesley, 2nd edition, 2004.***
- ***Booch, G.; Rumbaugh, J.; Jacobson, I., "Unified Modeling Language User Guide", 2nd Edition, Addison Wesley, 2005.***
- ***Eriksson, H-E, Penker, M. Lyons, B., Fado, D. UML 2 Toolkit, Wiley, 2004***

UML: visão geral

- **Classe**
- **Regras**
- **Mecanismos comuns**
- **Arquitetura**

Classe

- Classe é o descritor de um conjunto de objetos que compartilham os mesmos atributos, operações, métodos e comportamento.
  - Uma classe e outros elementos da UML que descrevem conjuntos de instâncias são considerados descritores.

- Representam um conceito dentro do sistema que está sendo modelado.
  - Formam o vocabulário do sistema.
  - Dependendo do modelo, o conceito pode ser relativo a coisas do mundo real ou envolver algoritmos ou implementação em computador.
  - Exemplo: Parede, Janela, Porta,etc podem ser exemplos classes de um modelo que representa um edíficio.

Reificação

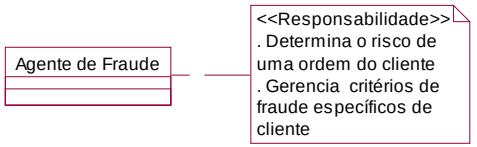
- Uma classe pode também representar conceitos abstratos (não material). Neste caso, usa-se reificação, isto é, tratar o conceito abstrato como coisa.
  - A reificação é uma instrumento muito utilizado pelo ser humano, por exemplo, na literatura e filosofia.
    - Ex.: Tupã, deus do trovão em tupi.
    - Um nome é a reificação de uma coisa e um verbo é a reificação de uma ação
  - A reificação é também muito utilizada em modelagem.
    - Por exemplo, pode-se criar uma classe Sonho que vai descrever as propriedades de um sonho.

- O nome de uma classe é um *string* textual, constituidos de letras, números e outros caracteres especiais exceto alguns utilizados na linguagem como " :"
  - Na prática, devem ser nomes ou frases curtas que funcionam como nomes, trazidos do vocabulário do domínio.
  - As letras iniciais de cada palavra devem ser maiúsculas.
  - Ex. SensorTemperatura
- O nome pode ser um caminho (*path*).
  - java::awt::Rectangle

Responsabilidade

- Uma *responsabilidade* é um contrato ou obrigação de uma classe.
  - Responsabilidades representam os conhecimentos e as ações que possibilitam a uma classe cumprir seu papel.
  - Os atributos e operações podem ser vistos como aspectos ou caracteríticas através das quais as responsabilidades são cumpridas.
- Um bom ponto de partida para a definição de uma classe é a definição de suas responsabilidades.

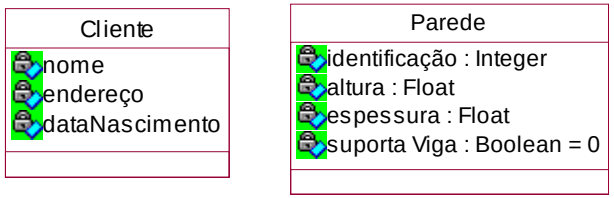
- Responsabilidades são descritas como frases ou parágrafos curtos em formato textual livre.
- Podem ser documentadas em um compartimento específico, como parte da descrição da classe ou como uma nota estereotipada como <<responsabilidade>> ligada a classe.
- Exemplo:



Atributo

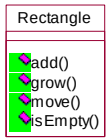
- Atributo é um propriedade, com nome, de uma classe que descreve um conjunto de valores que instâncias da propriedade podem ter.
- Uma classe pode ter nenhum ou vários atributos.
- A cada momento, os objetos de uma classe vão ter valores específicos para cada um de seus atributos.

- Os nomes de atributos são semelhantes aos nomes de classe, só que a primeira letra costuma ser minúscula.
- Pode-se especificar a classe (tipo) a que pertence um atributo e um valor inicial.
- Exemplo:



Operação

- Uma operação é a especificação de uma transformação ou consulta que um objeto pode ser solicitado a executar.
- Em geral, a invocação de uma operação de um objeto resulta na alteração de seu estado.
- O nome de uma operação costuma ser verbo ou frase curta que representa um comportamento.
- Exemplo:classe de java::awt



Técnicas de modelagem

- Classes são usadas para modelar abstrações oriundas do problema a resolver ou da solução a ser desenhada.
  - Essas abstrações são parte do vocabulário do domínio;
  - Juntas, representam as coisas que são importantes para usuários e/ou implementadores.

- Uma classe bem estruturada:
  - provê uma abstração clara de alguma coisa trazida do domínio do problema ou da solução;
  - compreende um conjunto pequeno, bem definido de responsabilidades e as realiza bem;
  - permite uma clara separação entre sua especificação e sua implementação.
  - é simples e inteligível mas ao mesmo tempo extensível e adaptável.

- Quando mostrar uma classe em um diagrama,
  - mostre somente as propriedades importantes para o entendimento da abstração em seu contexto;
  - organize listas longas de atributos e operações usando agrupamentos;
  - mostre classes relacionadas entre si no mesmo diagrama de classe.

- **Para modelar o vocabulário de um sistema:**
  - **Identifique as coisas (substantivos) que os usuários ou implementadores usam para descrever o problema ou uma solução.**
    - **Essas coisas são classes ou candidatas a classes.**
  - **Para cada abstração, identifique um conjunto de responsabilidades.**
    - **As responsabilidades precisam estar bem definidas e entendidas por todos.**

- **Para modelar o vocabulário de um sistema:**
  - **Estabeleça uma distribuição balanceada de responsabilidades entre as classes.**
  - **Identifique as classes que colaboram com as classes identificadas.**
  - **Proveja os atributos e operações necessários para executar as responsabilidades das classes.**

Cartões CRC

- **No trabalho de modelagem, cartões CRC (Classes, Responsabilidade, Colaborações) podem ser usados em na etapa inicial de identificação de classes ou de candidatos a classe.**
- **Cartões CRC são cartões (fichas de papel), cada ficha corresponde a uma classe. Cada ficha contém o nome da classe e 2 colunas com descrição de suas responsabilidades e colaborações.**
- **Colaborações apresentam outras classes que interagem com a classe descrita para o cumprimento de suas responsabilidades.**

- **Exemplo de cartão CRC:**

Nome da classe	Venda
Responsabilidades	Colaborações
Inserir/excluir item de venda	Item de venda
Editar item de venda	Estoque
Calcular impostos	Mercadoria
Totalizar venda	
Registrar data e hora	

Exemplo de modelagem de vocabulário

- Modelar o vocabulário de uma empresa que fornece equipamentos para clientes.
- Cenário: um cliente faz pedidos de compras de produtos. Uma transação comercial é realizada. A empresa localiza os produtos e faz um carregamento que é enviado ao cliente.

- Solução:
  - Descrever tarefas realizadas no negócio.
  - Identificar os substantivos ou locuções (conjunto de duas ou mais palavras que funcionam como uma unidade) que aparecem nas descrições.
  - Os substantivos (ou locuções) são candidatos a classe.
    - Mas podem também ser atributos, operações ou conceitos fora do domínio de interesse da modelagem.

- Solução:
  - A empresa armazena seus produtos em vários armazéns.
  - O cliente envia uma solicitação de orçamento.
  - O orçamento é enviado ao cliente.
  - O cliente envia um pedido de compra.
  - Os itens do pedido são localizados nos armazéns e embarcados.

Nome da classe	Cliente
Responsabilidades	Colaborações
Descrever dados de um cliente	Fatura
Criar um novo cliente	Pedido
Excluir cliente	
Alterar dados do cliente	

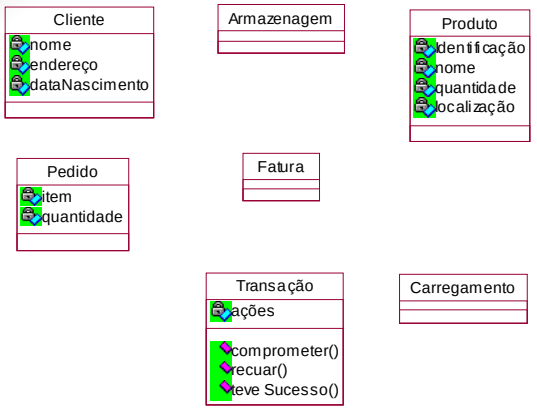
Nome da classe	Armazenagem
Responsabilidades	**Colaborações**
Descrever a localização de um produto antes do carregamento	Produto
	Fatura
	Carregamento

Nome da classe	Produto
Responsabilidades	**Colaborações**
Descrever um produto	Armazenagem
Criar, editar e excluir um produto do cadastro da empresa	Fatura
Localizar um produto	Pedido

Nome da classe	Pedido
Responsabilidades	**Colaborações**
Identificar um pedido de fornecimento de um cliente	Produto
	Fatura
	Carregamento
	Cliente

Nome da classe	Transação
Responsabilidades	**Colaborações**
Descrever o estado da transação comercial envolvendo o fornecimento.	Pedido
	Carregamento
	Cliente
	Fatura

Nome da classe	Carregamento
Responsabilidades	**Colaborações**
Manter informação sobre produtos embarcados em comparação com pedidos	**Pedido**
	Cliente
	Armazenagem
	Produto

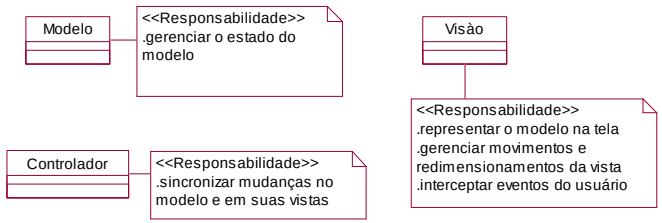


Exemplo de distribuição de responsabilidades

- **Em uma modelagem, não é interessante ter classes com poucas responsabilidades (muito simples) nem classes muito complexas.**
- **Classes com muitas responsabilidades são**
  - **difíceis de usar**
  - **não muito re-utilizáveis.**
- **Classes muito simplificadas:**
  - **levam a uma fragmentação do modelo**
  - **modelo fica difícil de gerenciar e de se entender.**

- **Para se distribuir responsabilidades:**
  - **identifique as classes que trabalham em conjunto para executar um comportamento;**
  - **identifique o conjunto de responsabilidades de cada classe;**
  - **analise o conjunto de classes como um todo. Parta classes muito complexas em classes menores, junte classes muito simples em classes maiores e re-aloque suas responsabilidades.**
  - **Considere o modo como as classes interagem e redistribua suas responsabilidades de modo que nenhuma classe faça muito ou muito pouco.**

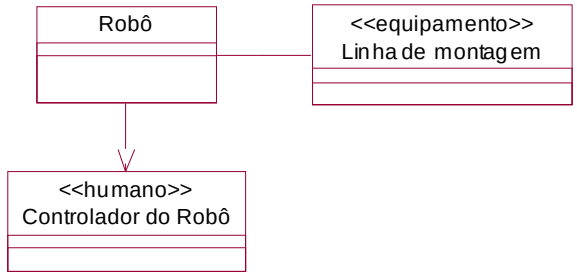
- **Exemplo de distribuição balanceada de responsabilidades (do Smalltalk):**



Modelagem em outros domínios

- **Para utilizar a UML em modelagem de processos de negócio ou outras aplicações que não envolvam software:**
  - **Modele suas abstrações como classes.**
  - **Pode-se usar mecanismos de extensão da UML para criar estereótipos e outros elementos de modelagem específicos do domínio que lhe interessa.**
  - **Pode-se modelar coisas envolvendo hardware que podem conter software como nodos, de modo a permitir detalhar sua estrutura.**

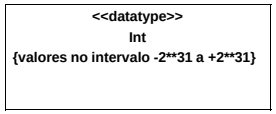
- **Exemplo.**

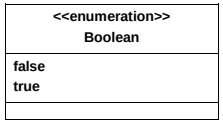


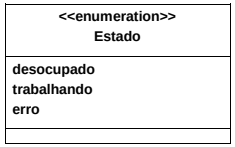
Modelagem de tipos primitivos

- **No outro extremo, as vezes é interessante modelar tipos primitivos que vêm de linguagens de programação, ex.: inteiros, enumeração, etc.**
- **Para isso:**
  - **modele a abstração como *datatype* ou *enumeration*, usando classes estereotipadas.**
  - **Use restrições da UML (*constraints*) se for necessário especificar um intervalo de valores.**

- **Exemplos:**







Visibilidade

- **É uma enumeração cujos valores {public, protected, private} indicam se o elemento de modelagem ao qual se referem podem ser vistos fora de seu espaço de nome.**
  - **Espaço de nome é a parte do modelo na qual o nome pode ser definido ou usado. Dentro de um espaço de nome, o nome tem um significado único.**

- **A visibilidade de um elemento de modelo define se ele poderá ser referenciado por um elemento fora de seu espaço de nome.**
  - **Visto de outra maneira, a visibilidade é parte da relação entre um elemento e seu Contêiner.**
  - **O Contêiner de um elemento pode ser uma classe, um pacote ou algum outro espaço de nome.**
- **Visibilidade pode ser aplicada a atributos e operações em relação a uma classe ou entre classes e o pacote onde ela foi definida (seu Contêiner), por exemplo.**

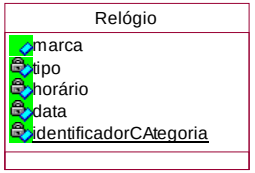
- **A UML define 3 níveis de visibilidade:**
  - **Público: um elemento que possa ver o contêiner de um elemento indicado na visibilidade pode também vê-lo.**
  - **Protegido: somente um elemento dentro de um contêiner ou o descendente de um contêiner pode ver o elemento indicado.**
  - **Privado: somente um elemento dentro do contêiner pode ver o elemento.**

- **A visibilidade é usada para realizar o encapsulamento de uma abstração, ou seja, expor somente as *features* que são necessárias para a cumprimento das responsabilidades de um elemento de modelagem.**
  - **Isso é essencial para se construir sistemas sólidos e resilientes, facilitando o entendimento, a modularização, a manutenção, etc.**

Escopo

- **Feature: nome que se dá a uma propriedade como atributo ou operação que é encapsulada em um elemento de modelagem como um classificador.**
- **O escopo indica o contexto que dá significado a um nome. Define se uma *feature* de uma classe representa um valor em cada instância da classe ou se representa um valor compartilhado por todas as instâncias da classe.**

- **Há 2 tipos de escopo na UML:**
  - **instance: cada instância do classificador tem seu próprio valor para a *feature*;**
  - **classe ou classifier (estático): só há um único valor para a *feature* para todas as instâncias do classificador.**
- **Notação: um escopo de classificador é denotado por sublinhado no nome do atributo ou operação**



Classe abstrata

- **Uma classe abstrata é aquela usada somente na construção de uma hierarquia de relacionamentos de generalização – não pode ter nenhuma instância direta.**
  - **É utilizada para definir operações e atributos comuns a um conjunto de classes filho.**
  - **O oposto, aquela que gera objetos como instâncias, é chamado classe concreta.**

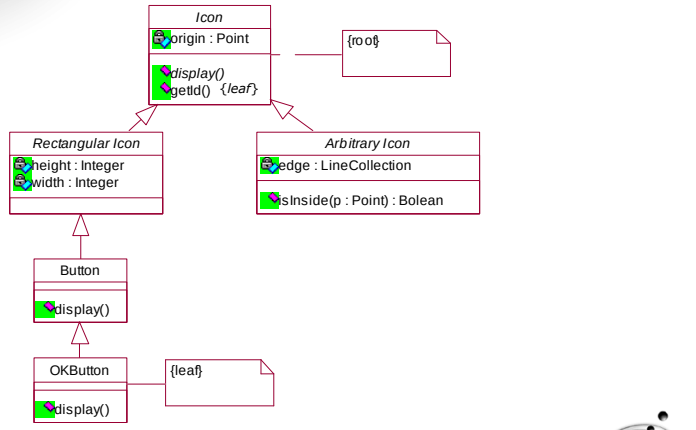
- Uma classe folha é aquela que não pode gerar um filho
  - Em UML, pode-se usar a propriedade {leaf} para especificar uma classe folha.
- Uma classe raiz é aquela que não pode ter pais.
  - Em UML, pode-se usar a propriedade {root} para especificar uma classe raiz.

Polimorfismo

- Operações têm algumas propriedades similares a classes. Uma operação abstrata é aquela que não tem um método definido.
  - Uma implementação para a operação tem que ser suprida em algum de seus descendentes concretos.
- Um elemento abstrato é denotado pelo nome em itálico.
- O oposto de abstrato é concreto

- Uma operação normalmente é considerada polimórfica, isto é, sua implementação (seu método) pode ser suprida ou sobrescrita por uma classe descendente.
  - Um descendente pode definir ou sobrescrever um método de uma operação polimórfica.
- O oposto de polimórfica é uma operação dita folha (*leaf*).
  - Neste caso, um método tem que ser definido para a operação.
- Uma operação não declarada folha é considerada polimórfica.

- Exemplo a seguir:
  - as classes *Icon*, *Rectangular Icon* e *Arbitrary Icon* são abstratas;
  - display e isInside são operações polimórficas;
  - *Icon::display()* é uma operação abstrata;
  - Icon::getID() é uma operação folha e concreta.



Atributo

- **Atributo é uma propriedade de uma classe que descreve um conjunto de valores que instâncias da propriedade podem ter.**
- **A sintaxe completa da definição de um atributo é:**

  [*visibility*] nome [*mutiplicity*] [:*type*] [= *initial-value*] [{*property-string*}].

  Ex.: apelido [0:5]: string

  + id : Integer {frozen}

- **Há 3 propriedades de atributos definidas na UML:**
  - ***Changeable*: não há restrições quanto a se modificar o valor do elemento (default).**
  - ***addOnly*: para atributos com multiplicidade maior que 1, novos valores podem ser adicionados ao conjunto de valores para um atributo mas, uma vez criados, um valor não pode ser removido ou alterado.**
  - ***frozen*: o valor do atributo não pode ser alterado depois que o elemento é inicializado.**

Operação

- **Operação é a especificação de uma transformação ou consulta que um objeto pode ser solicitado a executar.**
- **A sintaxe completa de uma operação é:**

  [*visibility*] nome [(*parameter-list*)] [:return *type*] [{*property-string*}].

  Ex.: - rearranja

  conjunto (n : Name, s : string)

  restart () {guarded}

- O nome da operação mais seus parâmetros e tipo de retorno é chamado de Assinatura da operação.
- Cada parâmetro é da forma:
  [*direction*] name : *type* [= *default-value*]
  - *Direction* pode ter um dos valores*:*

    in parâmetro de entrada, não pode ser modificado;

    out parâmetro de saída, pode ser alterado para comunicar informação ao chamador;

    inout combinação dos anteriores, pode ser alterado.

Propriedades de operações

- Há 5 propriedades de operações definidas na UML:

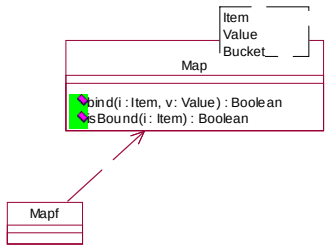
  leaf folha, indica que a operação não é polimórfica e não se pode sobrescrever.

  isQuery a execução da operação não altera o estado do sistema, isto é, é uma função de consulta sem efeitos colaterais.

- *sequential:* os chamadores devem coordenar a execução da operação de fora, de modo que somente um fluxo de execução exista no objeto em um tempo. Caso contrário, a semântica e integridade do objeto não pode ser garantida.
- *guarded:* a semântica e integridade do objeto é garantida na presença de múltiplos fluxos de controle através da sequencialização das chamadas para todas as operações guardadas do objeto.
- *concurrent :* a semântica e integridade da operação é garantida na presença de múltiplos fluxos de controle tratando a operação como atômica, ou seja, ou seja, vários fluxos de controle podem se ativados concorrentemente sem problemas.

Classe template

- Classe *template*, também chamada classe parametrizada, é um elemento parametrizado, usado para a geração de novas classes a partir de um gabarito.
- Exemplo:



Modelagem de semântica de uma classe

- A UML permite várias possibilidades para especificação da semântica de uma classe, variando de mais informal, como responsabilidade, a mais formal, como OCL (Object constraint language).
  - Menos formal significa menos completo ou detalhado, não significa menos preciso.

Possibilidades para especificação da semântica

- Especificar as responsabilidades da classe.
- Especificar a semântica da classe como um todo usando um texto estruturado em uma Nota ligada à classe.
- Especificar o corpo de cada método usando um texto estruturado ou linguagem de programação.
  - Para isso pode ser utilizado uma Nota associada à operação.

- Especificar pré e pós condições de cada operação e o invariante de uma classe, utilizando Notas associadas às operações ou a classe.
- Especificar uma Máquina de Estados para a classe.
  - A máquina de estados é um comportamento que especifica a seqüencia de estados que um objeto assume durante seu ciclo de vida em resposta a eventos.

- Especificar uma Colaboração que representa a classe.
  - Uma Colaboração dá nome a uma sociedade de papéis e outros elementos que trabalham juntos para prover um comportamento cooperativo que é maior que a soma de todos os elementos.
    - Uma Colaboração tem uma parte estrutural e uma parte dinâmica, pode sr usada para especificar todas as dimensões da semântica de uma classe.
- Especificar pré e pós condições de cada operação e o invariante de uma classe utilizando uma linguagem formal como OCL.

Observações

- Pode-se combinar as várias possibilidades apresentadas.
- Ter em mente os objetivos da especificação da semântica.
  - Pode-se especificar o que uma classe faz (visão externa) ou como ela faz (visão interna), ou ambas.
  - A visão externa é mais voltada para os usuários/clientes, a visão interna é mais voltada para os implementadores.

Interface

- Interface é uma declaração de um conjunto coerente de características e obrigações, como um contrato que um classificador se propõe a cumprir.
- Descreve o comportamento visível externo de um elemento
- Pode representar um comportamento completo ou parte dele.
- Uma interface descreve a especificação das operações (assinaturas) mas não as implementações das operações.
- Uma interface pode ser associada a uma classe ou a um componente que a realiza.

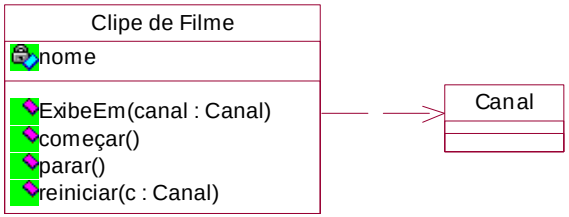
Relacionamento

- Relacionamentos são denotados por linhas conectando elementos de modelagem.
  - Modelam conexões semânticas entre elementos.
- Um modelo forma uma teia de relacionamentos. Cuidado ao modelar:
  - o abuso da utilização de relacionamentos em um modelo dificulta sua compreensão – vira uma bagunça;
  - O oposto, pode significar uma perda da riqueza que está associada ao modo como as coisas colaboram entre si em um sistema.

Dependência

- Dependência é uma conexão semântica entre duas Coisas na qual uma alteração em uma delas (a dependente) pode afetar a semântica da outra Coisa.
  - Pode ser considerada um relacionamento de uso, onde uma Coisa necessita utilizar outra Coisa.
- Graficamente, é representada como uma seta com linha pontilhada, dirigida no sentido da Coisa de que se depende.
- Pode-se dar um nome a uma dependência, mas mais comum é se usar estereótipos para definir tipo de dependência.

- **Exemplo: dependência é utilizada com freqüência para indicar que uma classe usa outra classe como argumento na assinatura de uma operação.**

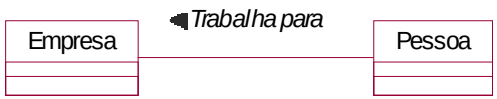


Generalização

- **É uma relação taxonômica (de classificação) entre um elemento mais geral e um elemento mais específico.**
- **Um relacionamento de especialização/generalização indica que objetos do elemento especializado (ou filho) podem substituir os objetos do elemento generalizado (ou pai).**
  - **O filho tem todos os atributos e operações dos pais mas pode ter outros atributos e operações**
- **Uma generalização pode ter nome mas raramente é utilizado.**

Associação

- **Descreve um conjunto de ligações, sendo uma ligação uma conexão semântica entre objetos.**
- **Muito utilizada para indicar possibilidade de navegação entre objetos.**
- **Uma associação pode ter nome e indicação de direção de leitura:**

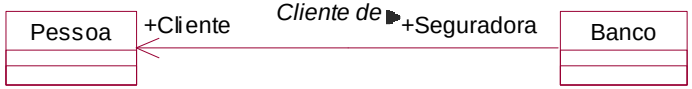


Navegação

- **Uma associação, por *default*, indica uma navegação bidirecional.**
- **No entanto, pode-se indicar uma sentido de navegação:**
  - **uma seta em um dos lados da associação indica o sentido de navegação;**
  - **isso indica o sentido em que a navegação é mais eficiente mas não exclui navegação no outro sentido.**

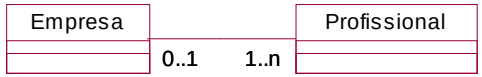
Nome de papel

- Oficialmente, na UML 2.0 chama-se nome de extremidade de associação.
- É utilizado em um dos lados de uma associação para indicar o papel com que a classe a seu lado apresenta-se para a classe do lado oposto.



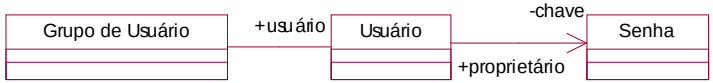
Multiplicidade

- Multiplicidade pode ser associado a um lado de uma associação para indicar quantos objetos da classe pode ser associado a objetos da classe do lado oposto.
- Multiplicidade pode ser considerada como uma restrição à cardinalidade de um conjunto.



Visibilidade

- Em uma associação, a visibilidade, usada em um de seus lados, indica se objetos de uma classe podem ver ou navegar para objetos do outro lado.
- Exemplo: no exemplo, a senha somente pode ser acessado por objetos da classe usuário.



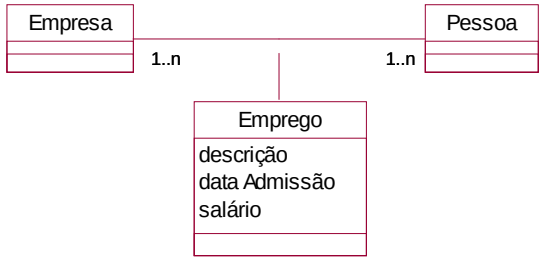
- A visibilidade pública é *default*.
- A visibilidade privada indica que objetos de um lado não são acessíveis a nenhum objeto fora da associação.
- A visibilidade protegida indica que objetos de um lado não são acessíveis a nenhum objeto fora da associação, exceto para filhos no outro lado.

Qualificador

- Descreve um atributo ou lista de atributos de uma associação, associado a um de seus lados, onde os valores desses atributos selecionam um sub-conjunto (geralmente um) de objetos da classe do outro lado.
- Exemplo:

Classe de associação

- É utilizada para se definir propriedades de uma associação.
- Exemplo:

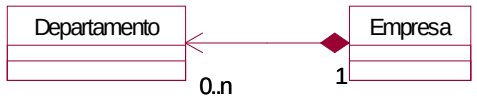


Restrições

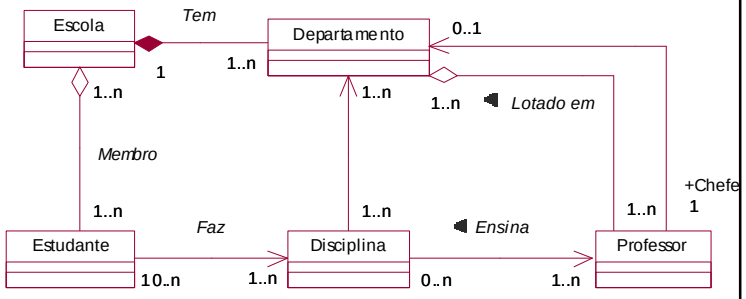
- Além das propriedades mostradas acima, outras restrições (*constraints*) podem ser aplicadas a associações.
- Algumas restrições são definidas na UML:
  - *ordered:* especifica que o conjunto de objetos naquele lado da associação é ordenado.
  - *xor*: especifica que, em um conjunto de associações, somente uma se manifesta para cada objeto associado.

Agregação e composição

- Agregação é um tipo especial de associação que indica uma relação todo/parte.
- Composição é uma forma de agregação com posse forte e tempo de vida idêntico.
  - Um objeto só pode participar de uma composição.
- A relação de agregação ou composição é independente da direção de navegação.



Exemplo de modelo



Realização

- **É uma relação semântica entre classificadores onde um classificador especifica um contrato que o outro classificador garante cumprir.**
- **A realização é utilizada entre interfaces e as classes ou componentes que as realizam e entre casos de uso e as colaborações que os realizam.**