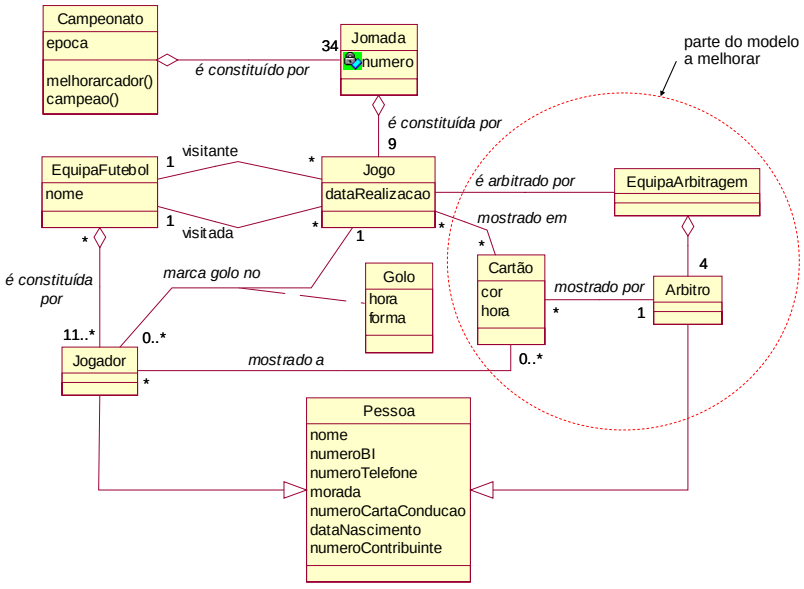
Exemplo: Campeonato de futebol

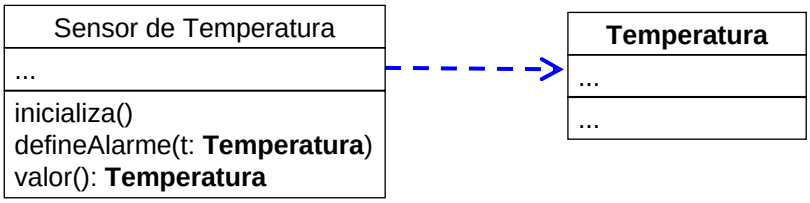
- Um campeonato é constituído por 34 jornadas, é identificado por uma época e possui um campeão e um melhor marcador.
- Cada jornada é identificada por um número e é constituída por 9 jogos.
- Cada jogo é efectuado por uma equipa de futebol visitada, outra visitante e é arbitrado por uma equipa de arbitragem.
- Cada equipa de futebol tem 11 ou mais jogadores e cada equipa de arbitragem possui 4 árbitros.
- Cada jogador pode marcar golos, a determinado instante do jogo e de uma de diversas formas: de grande penalidade, de livre, de pontapé de fora da área, de cabeça, etc.
- O árbitro de um jogo pode mostrar um ou mais cartões, amarelos ou vermelhos.

Campeonato de futebol: possível solução...
(mas que pode ser melhorada)



Relação de Dependência

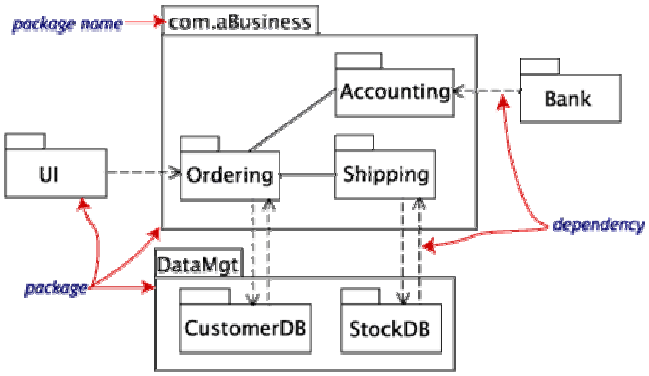
- As dependências são desenhadas como linhas dirigidas a tracejado



- Uma **dependência** é uma relação de uso entre dois elementos (não necessariamente classes), em que uma mudança na especificação de um dos elementos pode forçar uma alteração no outro elemento
- Por motivos de simplicidade, em geral não se explicita este tipo de relações nos diagramas de classes – são implícitas!
  - as dependências são muito mais usadas entre outro tipo de elementos do UML, nomeadamente Pacotes

Relação de Dependência: Exemplo

- Exemplo de um modelo de negócio, em que as classes estão agrupadas em pacotes



- Um pacote depende de outro se uma alteração do pacote de destino afectar o pacote de origem (dependente)

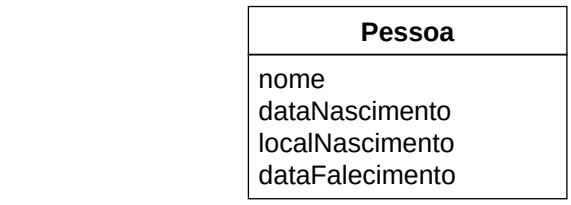
Restrições (*constraints*)

- Uma restrição
  - especifica uma condição que tem de se verificar
  - é indicada por uma expressão ou texto entre chavetas...

  ... ou por uma nota posicionada junto aos elementos a que diz respeito, ou a eles ligada por linhas a traço interrompido (sem setas, para não confundir com relação de dependência)

- Alguns exemplos:
  - o salário de um empregado não pode exceder o salário do seu chefe
    `{empregado.salario < chefe.salario}`
  - as janelas não podem ter uma relação de aspecto inferior a 0.8 e superior a 1.5
    `{0.8 <= largura/altura <= 1.5}`

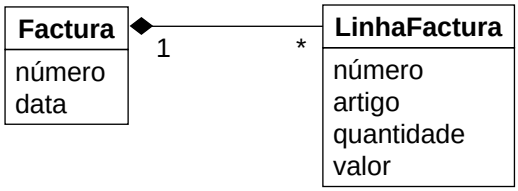
Restrições em Classes



{ dataFalecimento > dataNascimento }

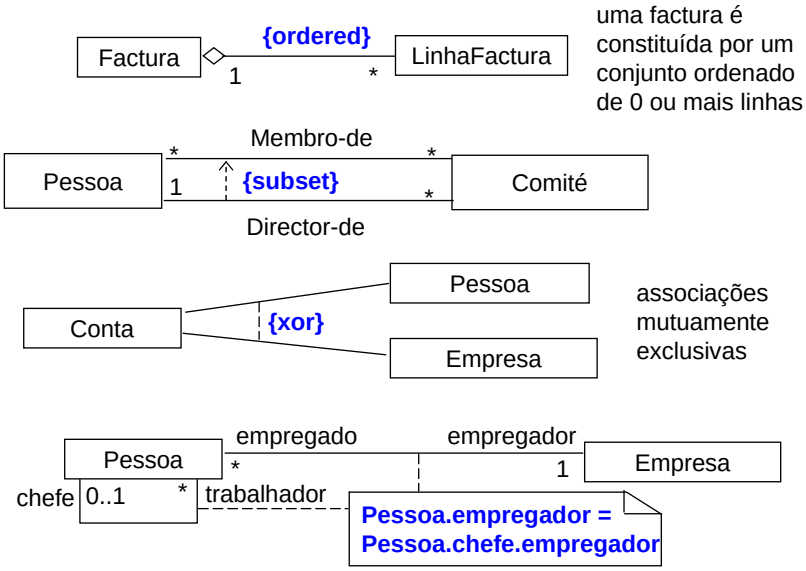
Restrições simples, como estas, podem ser colocadas no modelo...

... mas restrições mais complexas devem ser especificadas no modelo funcional

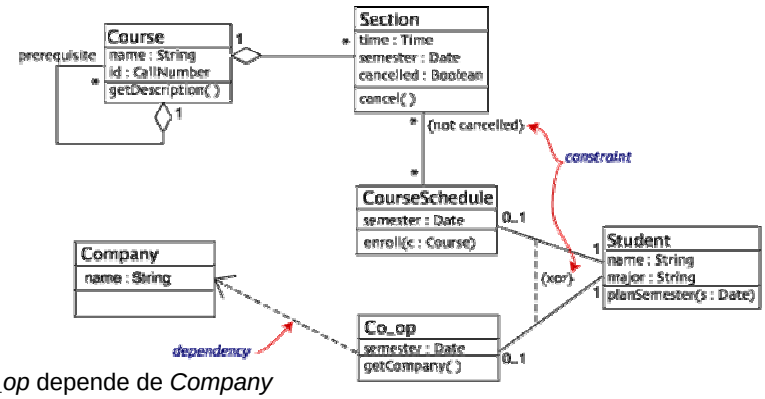


{ chave candidata: (factura.número, número) }

Restrições em Associações



Dependências e Restrições: Exemplo



- *Co_op* depende de *Company*
  - se modificarmos *Company*, podemos também ter de modificar *Co_op*
- O diagrama tem duas restrições:
  - uma *Section* apenas pode ser parte de um *CourseSchedule* se ele não tiver sido cancelado
  - a restrição `{xor}` indica que um *Student* tem de ter um *CourseSchedule* ou uma posição *Co_op*, mas não ambos

Extensão e Restrição na Generalização

- Conforme já vimos, instâncias de uma classe são também instâncias de todas as classes ascendentes

- Consequentemente, todas as características das classes ascendentes são aplicadas às instâncias das sub-classes

  - uma classe descendente não pode omitir ou suprimir determinada característica de uma classe ascendente

  - no entanto, as sub-classes podem acrescentar novas características (atributos e operações) à sua super-classe
    - a sub-classe passa a ser uma **extensão** da super-classe

Extensão e Restrição na Generalização (cont.)

- Os atributos de uma super-classe podem ter de obedecer a determinadas **restrições** quando aplicadas às subclasses
  - os valores assumidos pelas subclasses passam a ficar restritos a determinadas gamas de valores
    - por exemplo, a classe `círculo` definido com base na classe `elipse` tem de obedecer à restrição “eixos menor e maior são iguais”

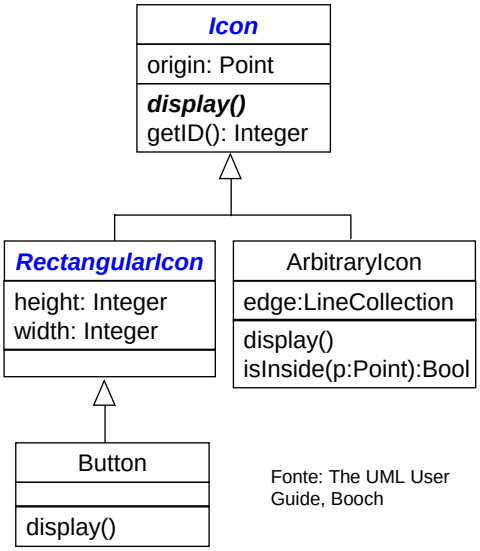
- Contudo, alterações arbitrárias dos atributos da super-classe podem violar as restrições relativas à subclasse
  - por exemplo, se os eixos menor e maior de uma elipse forem alterados por uma das operações da classe elipse e o objecto em causa for um circulo, a restrição “eixos menor e maior são iguais” é certamente violada

Extensão e Restrição na Generalização
(cont.)

- Para casos deste tipo:
  - as propriedades herdadas podem ser renomeadas
    - por exemplo, eixos → diâmetro
  - a subclasse pode não herdar todas as operações da super-classe
    - por exemplo, operações de alteração de escala desigual segundo os eixos X e Y devem ser suprimidas de forma adequada na classe círculo
- A restrição implica que nem todas as operações da super-classe possam ser herdadas pela subclasse

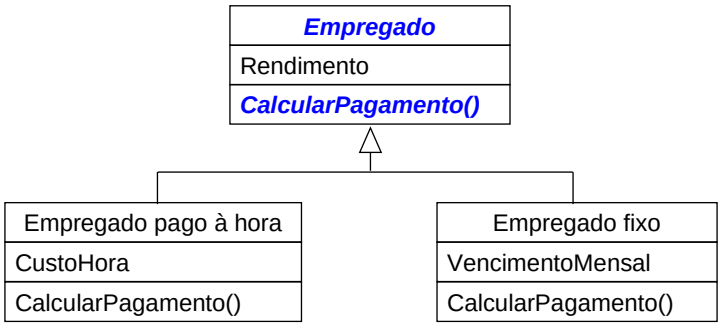
Classes e Operações abstractas (≠concretas)

- **Classe abstracta**: classe que não pode ter instâncias directas
  - pode ter instâncias indirectas pelas subclasses concretas
- **Operação abstracta**: operação com implementação a definir nas subclasses
  - uma classe com operações abstractas é obrigatoriamente uma classe abstracta
- **Notação**: nome em *itálico* ou propriedade {**abstract**}



Fonte: The UML User Guide, Booch

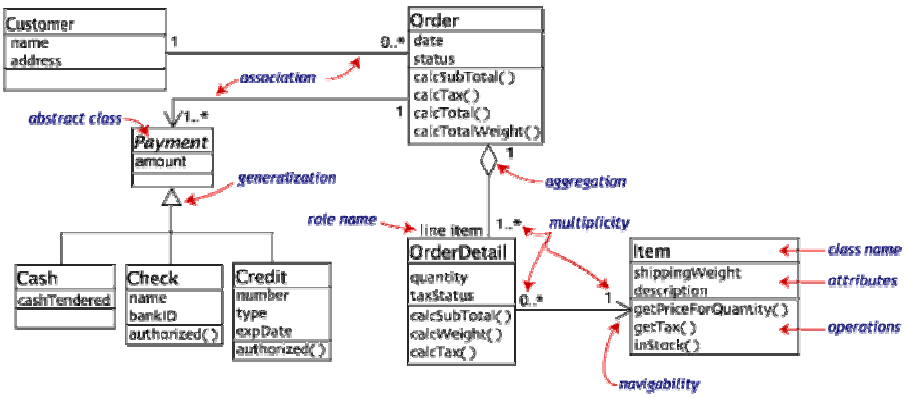
Classes abstractas (cont.)



- Uma **classe é abstracta** quando só os seus descendentes é que podem ser instanciados de forma directa
  - a classe Empregado não pode ser directamente instanciada
- Uma **classe é concreta** quando pode ser directamente instanciada
  - as subclasses de Empregado podem ser instanciadas

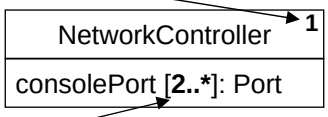
Exemplo: Encomendas

Modelização das encomendas (Orders) de um cliente (Customer), cujo pagamento (Payment) é feito em dinheiro (Cash), cheque (Check) ou através de cartão de crédito (Credit Card). As encomendas contêm linhas (OrderDetail), cada uma das quais com o respectivo item (Item) associado.



Multiplicidade de classes e atributos

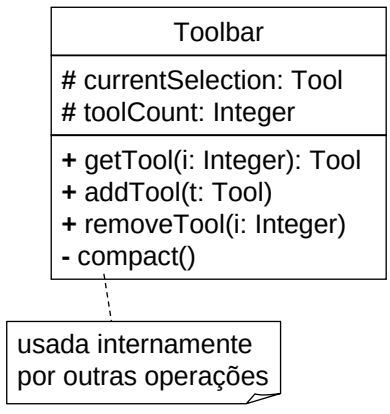
- **Multiplicidade de classe**: número de instâncias que podem existir
  - por omissão é **0..***



- **Multiplicidade de atributo**: número de valores que o atributo pode tomar do tipo especificado
  - por omissão é **1**

Visibilidade de atributos e operações

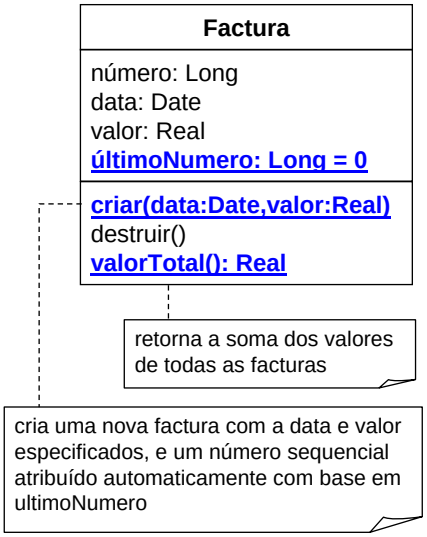
- **Visibilidade**
  - **-** (private): visível só por operações da própria classe
  - **#** (protected): visível por operações da própria classe e descendentes (subclasses)
  - **+** (public): visível por todos



- Recordando o encapsulamento...
  - devem-se esconder todos os detalhes de implementação que não interessam aos clientes (utilizadores) da classe

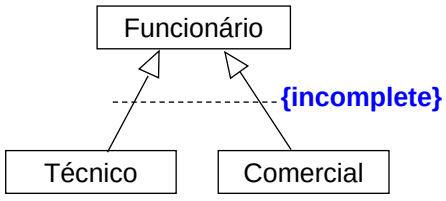
Atributos e operações do âmbito da classe (≠instância)

- **Atributo do âmbito da classe**: tem um único valor para todas as instâncias (objectos) da classe, o qual está definido mesmo que não exista nenhuma instância
- **Operação do âmbito da classe**: não tem como argumento implícito um objecto da classe
- Notação: nome sublinhado
- Correspondem a membros estáticos (*static*) em C++ e Java



Subclasses incompletas (≠ completas)

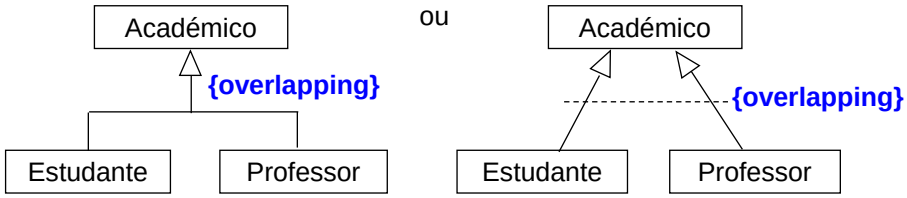
- Caso em que um objecto da super-classe pode não pertencer a nenhuma das subclasses
- Indicado por **restrição** ***{incomplete}***
- O contrário é ***{complete}*** (situação por omissão)



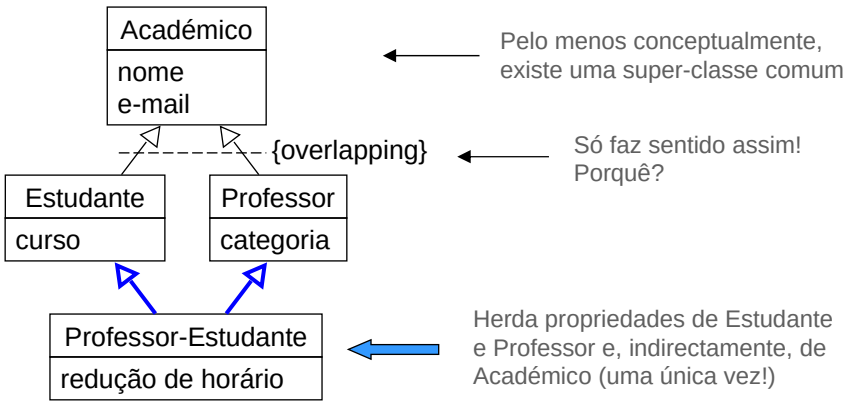
- outras possibilidades: Administrativo, Director, etc.

Subclasses sobrepostas (≠ disjuntas)

- Caso em que <u>um objecto da super-classe pode pertencer, simultaneamente, a mais do que uma subclasse</u>
- Indicado por **restrição {*overlapping*}**
- O contrário é **{*disjoint*}** (situação por omissão)



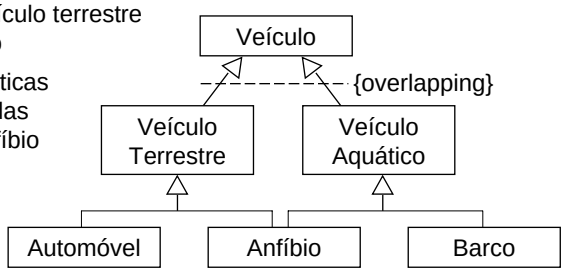
Herança múltipla (≠ simples)



- Uma subclasse com duas ou mais super-classes é denominada **classe de junção**

Herança múltipla (cont.)

- As propriedades de uma classe ascendente, que são obtidas por caminhos diferentes, só são herdadas uma vez
  - um veículo anfíbio herda todas as características de um veículo terrestre e de um veículo aquático
  - no entanto, as características de veículo só são herdadas uma vez pelo veículo anfíbio



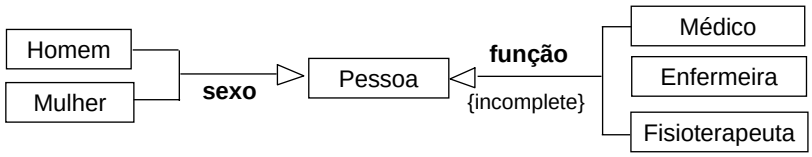
- Definições paralelas em classes ascendentes produzem conflitos
  - a definição de características idênticas para os veículos terrestres e aquáticos torna a definição de um veículo anfíbio ambígua e inconsistente

Herança múltipla (cont.)

- As subclasses de uma dada generalização podem-se sobrepor ou podem ser disjuntas
  - por exemplo, um determinado veículo pode ser terrestre e aquático ao mesmo tempo, ou seja, assumir ambas as formas em simultâneo – a generalização é sobreposta
  - na generalização relativa aos empregados (pagos à hora, assalariados ou fixos), um determinado empregado ou pertence a uma das subclasses ou a outra, mas nunca às duas em simultâneo – generalização disjunta
- A herança múltipla permite misturar informação de duas ou mais fontes
- Cada generalização deve ser feita sobre uma das propriedades da classe
- A generalização múltipla deve ser usada quando uma classe necessitar de refinamentos em dimensões distintas e independentes

Classificação múltipla (≠ simples)

- Caso em que um objecto pode pertencer num dado momento a várias classes, sem que exista uma subclasse que represente a intersecção dessas classes (com herança múltipla)



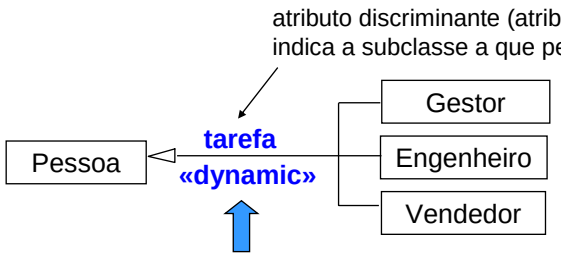
combinações legais: {Mulher, Enfermeira}, {Homem, Fisioterapeuta}, ...

- A classificação múltipla pode ser simulada por agregação de papéis

Classificação dinâmica (≠ estática)

- Caso em que a(s) classe(s) a que um objecto pertence pode(m) variar ao longo da vida do objecto

- Indicado por **estereótipo «dynamic»**

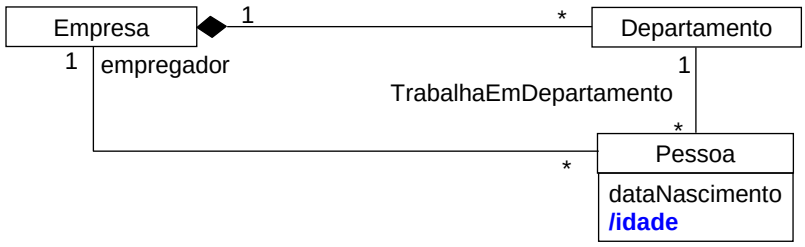


uma pessoa que num dado momento tem a tarefa de gestor, pode noutro momento ter a tarefa de vendedor, etc.

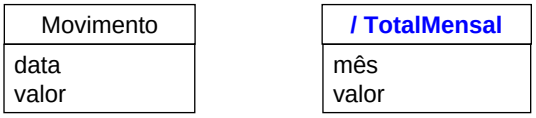
Elementos derivados

- **Elemento derivado** (atributo, associação ou classe) - elemento calculado em função doutros elementos do modelo
- Notação: **barra "/"** antes do nome do elemento derivado
- Um elemento derivado tem normalmente associada uma restrição que o relaciona com os outros elementos

Elementos derivados: Exemplos



{ idade = dataActual() – dataNascimento }



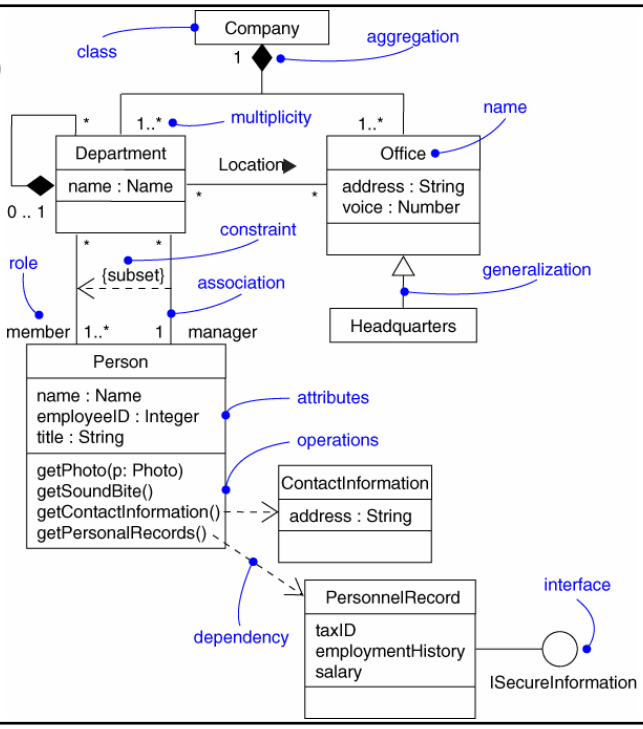
{ valor = (select sum(valor) from Movimento where month(data) = mês) }

Perspectivas a usar
(num diagrama de classes)

- Um diagrama de classes é construído de acordo com determinada perspectiva (situada num dado nível de abstracção)
- Tipos comuns de perspectivas:
  - Conceptual: descrição dos conceitos do domínio do problema
    - as classes e respectivas associações representam conceitos
    - em alguns casos, descrição das principais responsabilidades do objecto
  - Especificação: descrição das interfaces das classes
    - as classes já representam elementos de software
    - as associações representam as responsabilidades dos objectos

  - Implementação: a descrição das interfaces é complementada com a descrição das respectivas implementações

Resumo



Resumo

- Um diagrama de classes é construído e refinado ao longo das várias fases do desenvolvimento do software, por analistas, projectistas (*designers*) e implementadores

- Elementos da UML presentes nos diagramas de classes:
  - **Classes**, com as respectivas estruturas internas e comportamento
  - **Associações**, **agregações**, **dependências** e **relações de herança**
  - **Multiplicidade** e indicadores de **navegação**
  - Nomes de **papéis** (o que uma classe representa para outra)

Referências

Estes apontamentos foram baseados em:

- "UML – Unified Modeling Language", Curso em Tecnologia de Objectos, FEUP, Novembro 2000
  Ademar Aguiar, Gabriel David, João Pascoal Faria
- "Desenvolvimento OO – Construção do modelo de objectos através da linguagem UML", ISPGaya, Novembro 1998
  César Toscano
- UML, Capítulo 2, "Objectos e Classes", ISPGaya
  José Maria Bonnet
- "Practical UML: A Hands-On Introduction for Developers", TogetherSoft Corporation
- "UML Metodologias e Ferramentas CASE", Centro Atlântico, 2001
  Alberto Silva, Carlos Videira