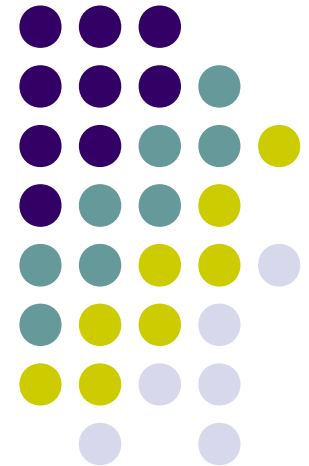


**Introdução aos Sistemas de Informação (ISI) (025607)
e
Engenharia de Software 1 (ES1) (1001530)**



**Aula 6:
Diagrama Conceitual
(ou Modelo de Classes de Domínio)**



Prof. Fabiano Cutigi Ferrari
2º semestre de 2020

ENPE – Ensino não presencial emergencial

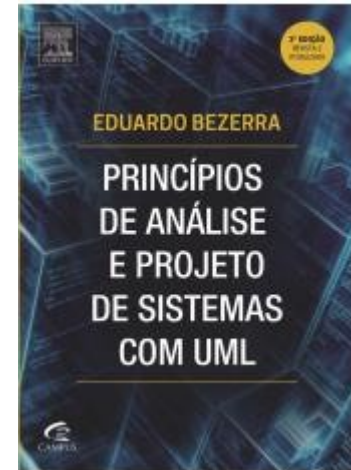
Recados Iniciais



- **Mantenham seus e-mails atualizados no Moodle.**
- Como está a execução do projeto?

Notas Iniciais

- Preparado com base nos materiais a seguir*:
 - Slides disponibilizados em conjunto com o livro
 - Eduardo BEZERRA: Princípios de Análise e Projeto de Sistemas com UML, 3ª ed., Campus/Elsevier (2015).
 - Notas de aula e slides elaborados pelo professor, e outros materiais disponíveis na Web

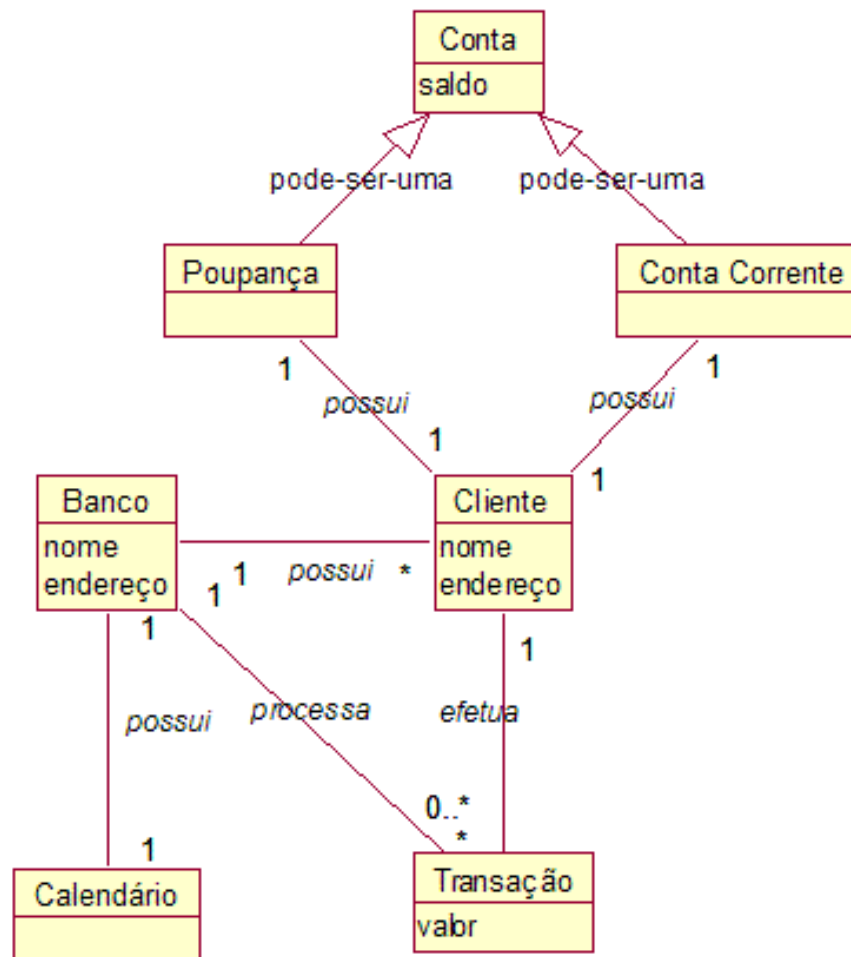


* Notas de rodapé ajudam a identificar os slides produzidos por Bezerra (2015).

- Introdução
- Diagrama de classes
- Técnicas para identificação de classes
- Modelo de classes no processo de desenvolvimento

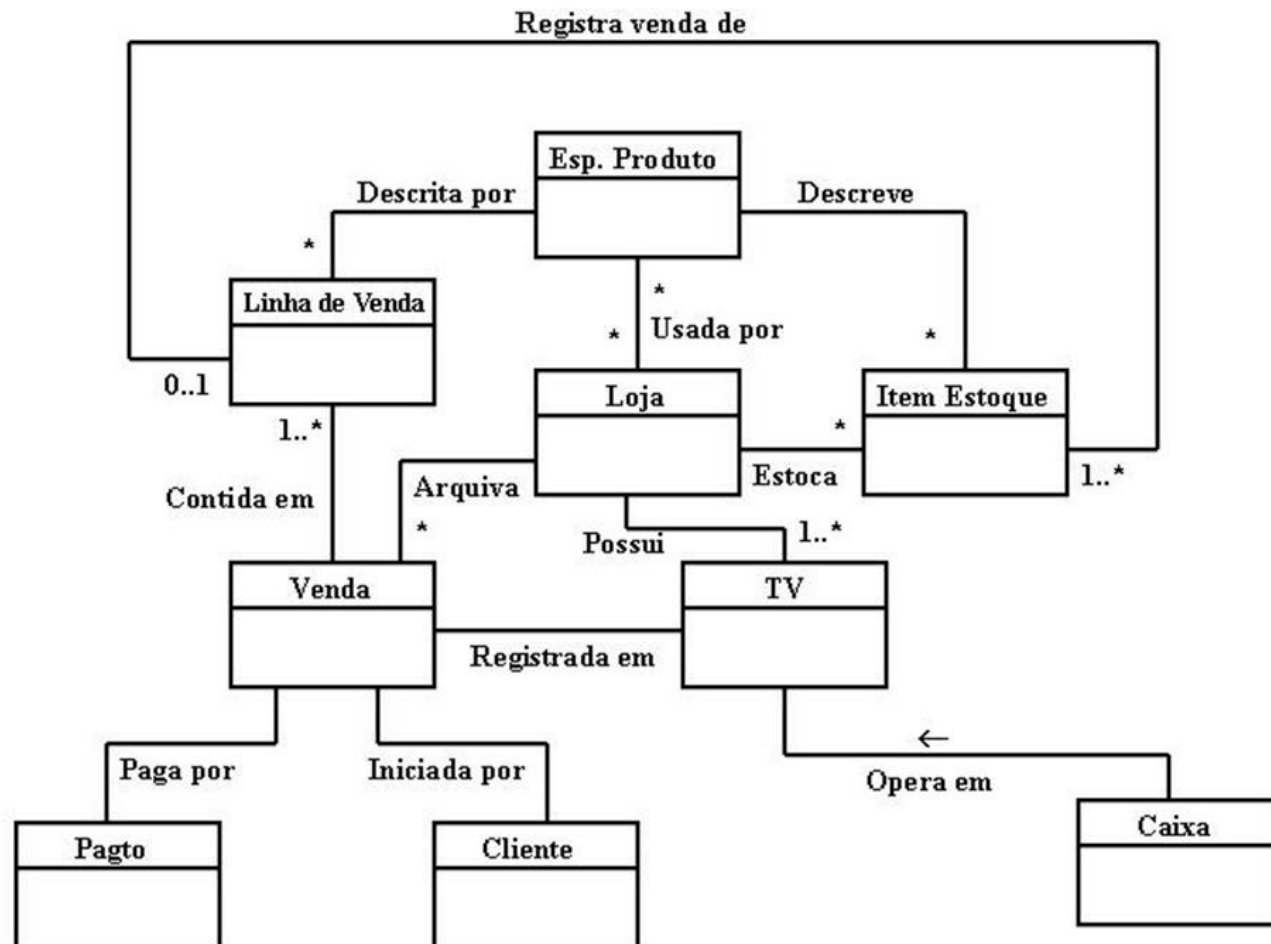
Exemplos Iniciais

O que queremos produzir?



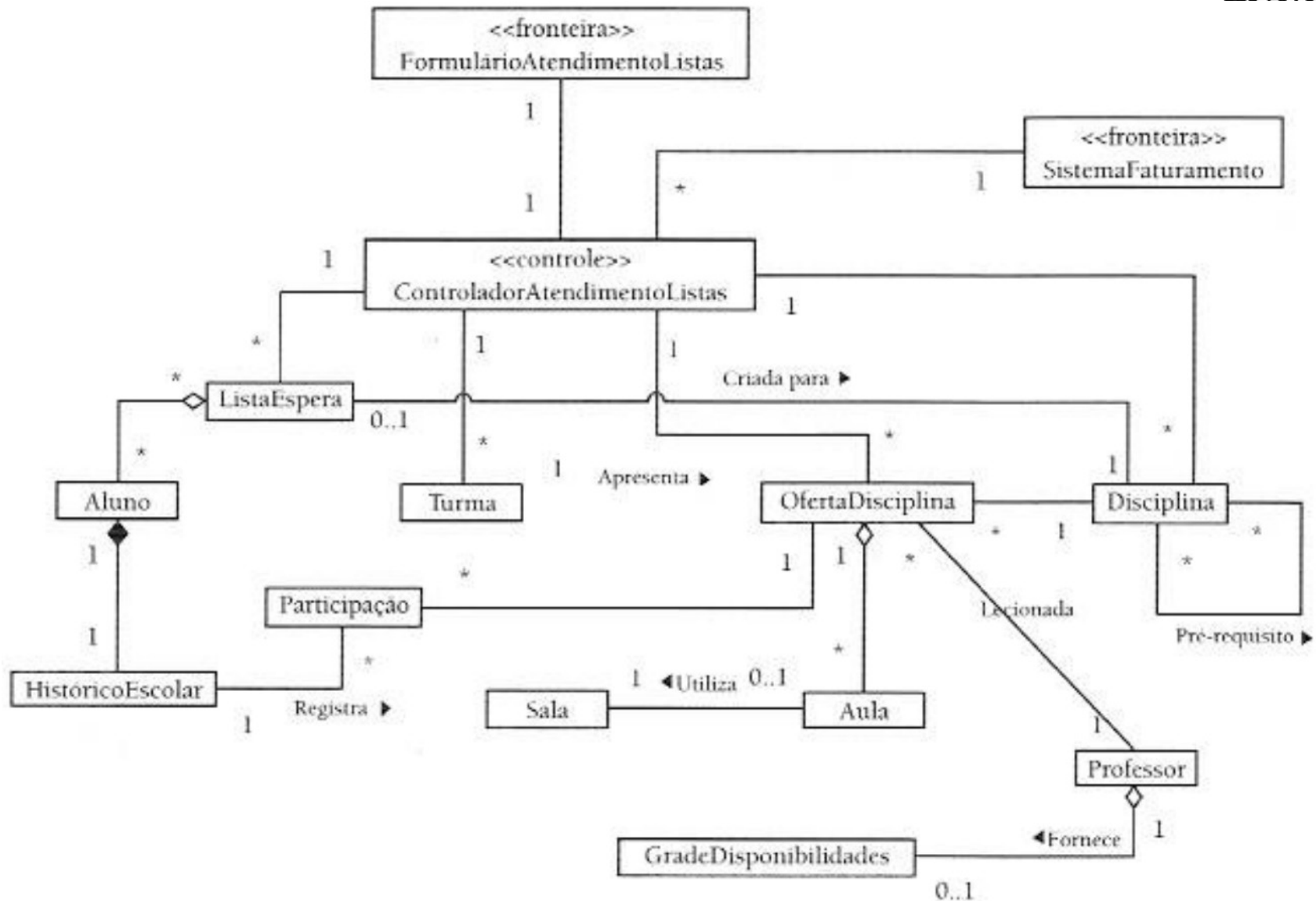
Exemplos Iniciais

O que queremos produzir?



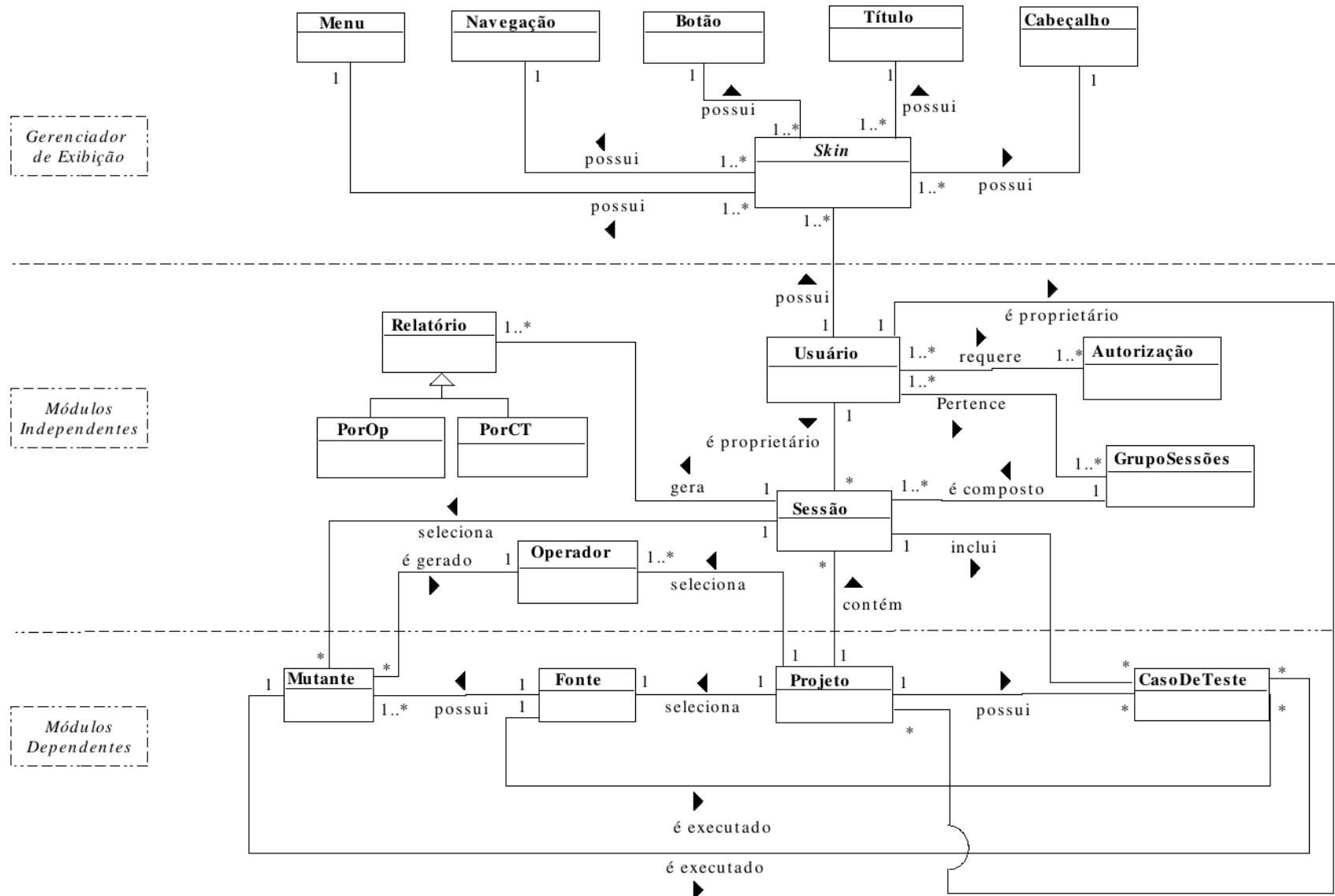
Exemplos Iniciais

O que queremos produzir?



Exemplos Iniciais

O que queremos produzir?



- As funcionalidades de um software OO são realizadas internamente através de **colaborações** entre objetos.
 - Atores externos visualizam resultados, relatórios etc..
 - Internamente, os objetos colaboram para produzir os resultados.
- Essa colaboração pode ser vista sob o **aspecto dinâmico** e sob o **aspecto estrutural estático**.
- O **diagrama de classes** representa o aspecto estrutural e estático que compõe a solução encapsulada em um software OO.

Objetivo da Modelagem de Classes

- O objetivo da modelagem de classes de **análise** é prover respostas para as seguintes perguntas:
 - Em um nível alto de abstração, que objetos constituem o sistema em questão?
 - Quais são as classes candidatas?
 - Como as classes se relacionam?
 - Quais são as responsabilidades de cada classe?

Modelo de Classes de Análise

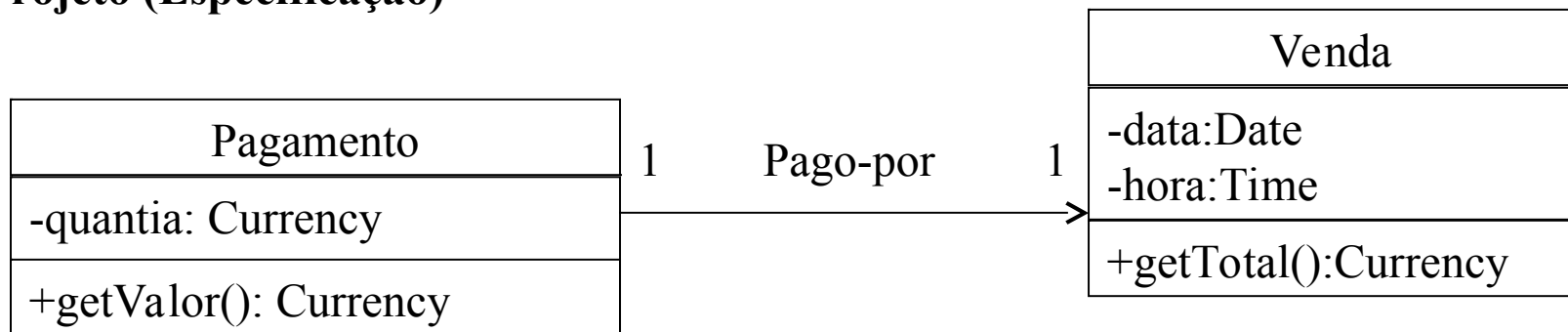
- Representa termos do domínio do negócio.
 - idéias, coisas, e conceitos no mundo real.
- Descreve o problema representado pelo sistema a ser desenvolvido, sem considerar características da solução final.
- É um dicionário “visual” de conceitos e informações relevantes ao sistema sendo desenvolvido.
- Elementos (da notação UML para Diagrama de Classes):
 - classes e atributos; associações, composições e agregações (com seus adornos); classes de associação; generalizações (herança).

Modelo de Análise: Foco no Problema

- O modelo de análise não representa detalhes da solução do problema.
 - Mas serve como ponto de partida para um projeto detalhado.



Projeto (Especificação)



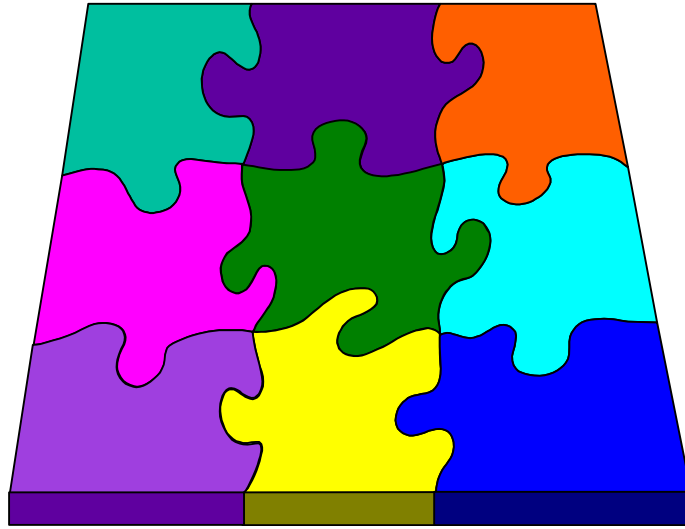


Diagrama de classes

- Uma classe descreve esses objetos através de *atributos* e *operações*.
 - Atributos correspondem às informações que um objeto armazena.
 - Operações correspondem às ações que um objeto sabe realizar.
- Notação na UML: “caixa” com no máximo três compartimentos exibidos.
 - Detalhamento utilizado depende do estágio de desenvolvimento e do nível de abstração desejado.

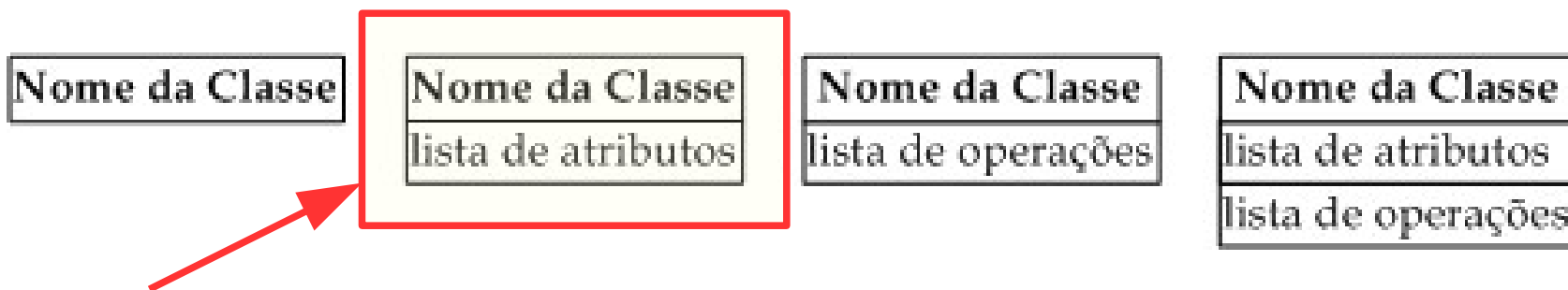
Nome da Classe

Nome da Classe
lista de atributos

Nome da Classe
lista de operações

Nome da Classe
lista de atributos
lista de operações

- Uma classe descreve esses objetos através de *atributos* e *operações*.
 - Atributos correspondem às informações que um objeto armazena.
 - Operações correspondem às ações que um objeto sabe realizar.
- Notação na UML: “caixa” com no máximo três compartimentos exibidos.
 - Detalhamento utilizado depende do estágio de desenvolvimento e do nível de abstração desejado.



Exemplo (classe ContaBancária)

ContaBancária

ContaBancária
 número
 saldo
 dataAbertura

ContaBancária
 criar()
 bloquear()
 desbloquear()
 creditar()
 debitar()

ContaBancária
 número
 saldo
 dataAbertura
 criar()
 bloquear()
 desbloquear()
 creditar()
 debitar()

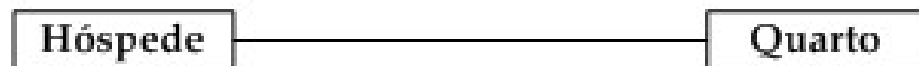
ContaBancária
 -número : String
 -saldo : Quantia
 -dataAbertura : Date
 +criar()
 +bloquear()
 +desbloquear()
 +creditar(in valor : Quantia)
 +debitar(in valor : Quantia)

Associações

- Representam relacionamentos (ligações) que são formados entre objetos durante a execução do sistema.
- Note que, embora as associações sejam representadas entre classes do diagrama, tais associações representam ligações possíveis entre os objetos das classes envolvidas.

Notação para Associações

- Na UML, associações são representadas por uma linha que liga as classes cujos objetos se relacionam.
- Exemplos:



Multiplicidades

- Representam a informação dos limites inferior e superior da quantidade de objetos aos quais outro objeto pode se associar.
- Cada associação em um diagrama de classes possui duas multiplicidades, uma em cada extremo da linha de associação.

Nome	Simbologia na UML
Apenas Um	1..1 (ou 1)
Zero ou Muitos	0..* (ou *)
Um ou Muitos	1..*
Zero ou Um	0..1
Intervalo Específico	$L_i \dots L_s$

Limites inferior e superior

Exemplos (multiplicidade)



- Exemplo
 - Pode haver um cliente que esteja associado a vários pedidos.
 - Pode haver um cliente que não esteja associado a pedido algum.
 - Um pedido está associado a um, e somente um, cliente.



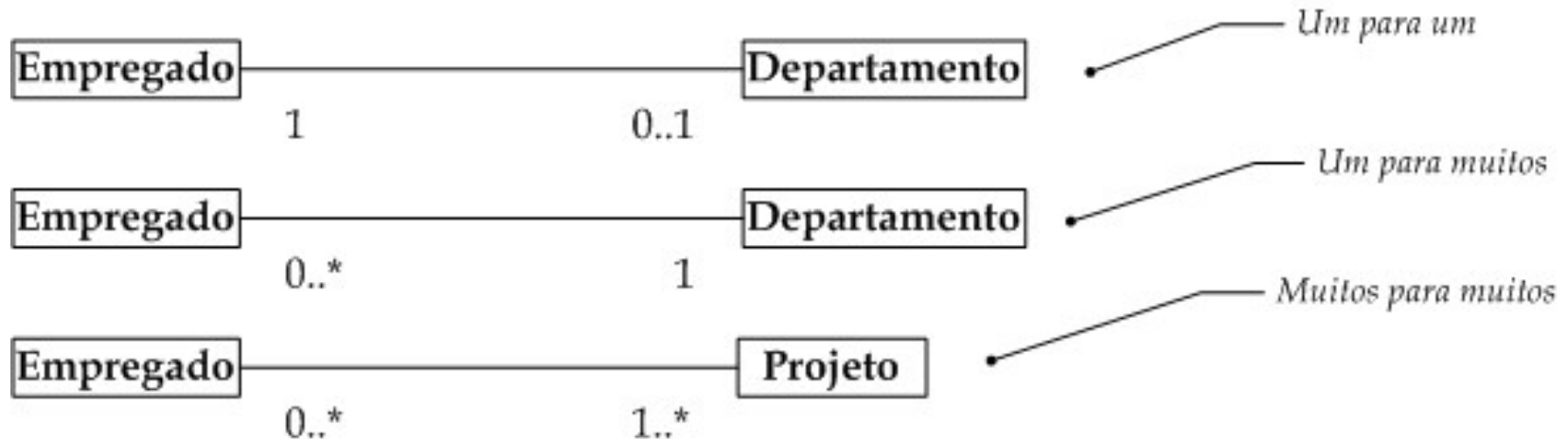
- Exemplo
 - Uma corrida está associada a, no mínimo, dois velocistas
 - Uma corrida está associada a, no máximo, seis velocistas.
 - Um velocista *pode* estar associado a várias corridas.

Conectividade

- A **conectividade** corresponde ao tipo de associação entre duas classes: "*muitos para muitos*", "*um para muitos*" e "*um para um*".
- A conectividade da associação entre duas classes depende dos símbolos de multiplicidade que são utilizados na associação.

Conectividade	Em um extremo	No outro extremo
Um para um	0..1 1	0..1 1
Um para muitos	0..1 1	* 1..* 0..*
Muitos para muitos	* 1..* 0..*	* 1..* 0..*

Exemplo (conectividade)

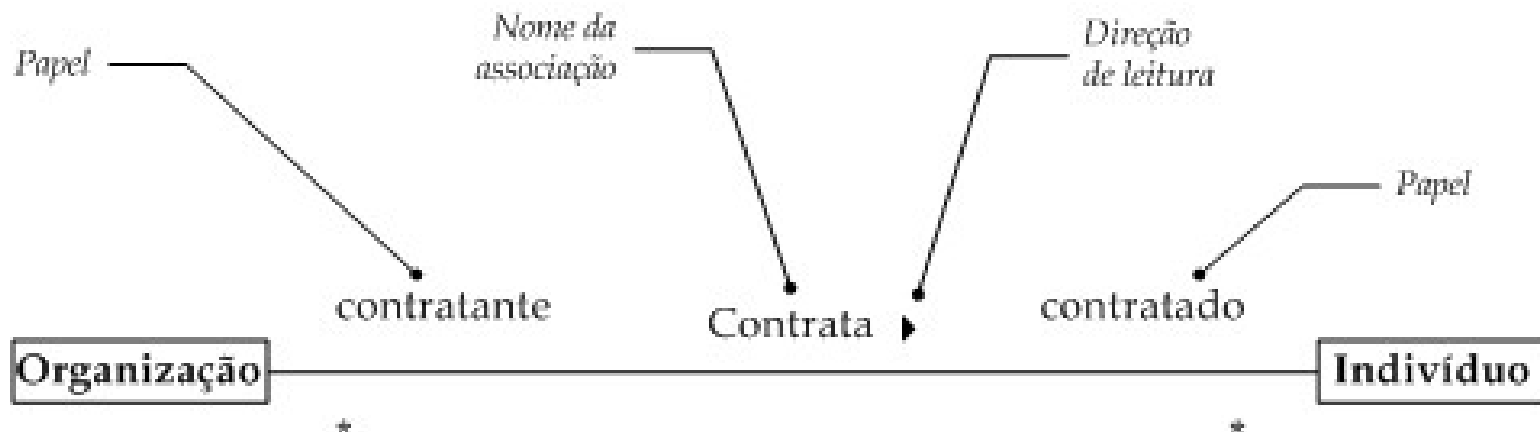


Participação

- É uma característica de uma associação que indica a necessidade (ou não) da existência desta associação entre objetos.
- A participação pode ser *obrigatória* ou *opcional*.
 - Se o valor mínimo da multiplicidade de uma associação é igual a 1 (um), significa que a participação é obrigatória
 - Caso contrário, a participação é opcional.

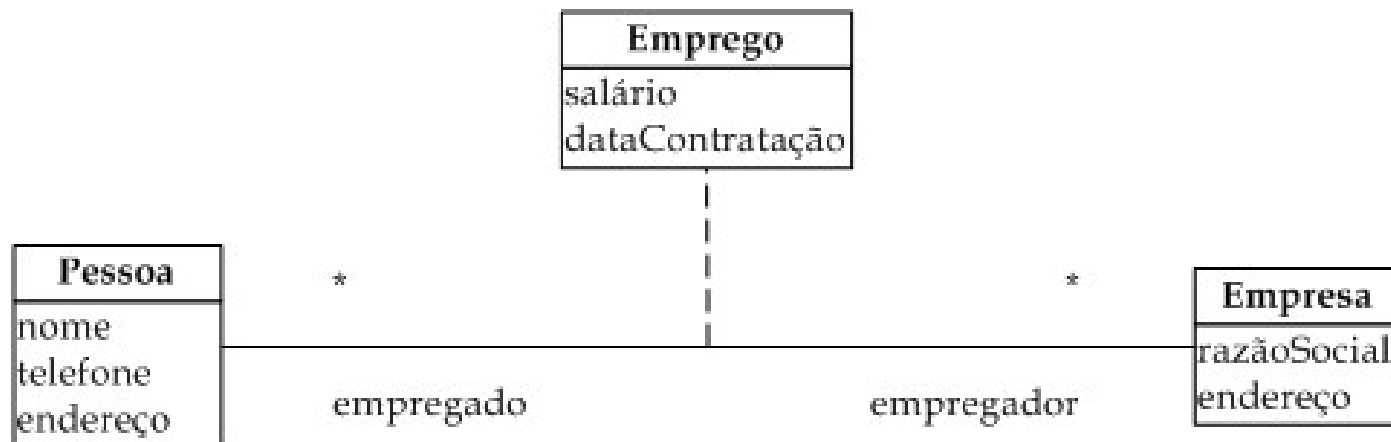
Acessórios para Associações

- Para melhor esclarecer o significado de uma associação no diagrama de classes, a UML define três recursos de notação:
 - **Nome da associação:** fornece a ela algum significado semântico.
 - **Direção de leitura:** indica como a associação deve ser lida
 - **Papel:** para representar um papel específico em uma associação.



Classe Associativa

- É uma classe que está ligada a uma associação, em vez de estar ligada a outras classes.
 - A notação é semelhante à utilizada para classes ordinárias. A diferença é que esta classe é ligada a uma associação por uma linha tracejada.
 - Exemplo: para cada par de objetos [pessoa, empresa], há duas informações associadas: salário e data de contratação.



Classe Associativa

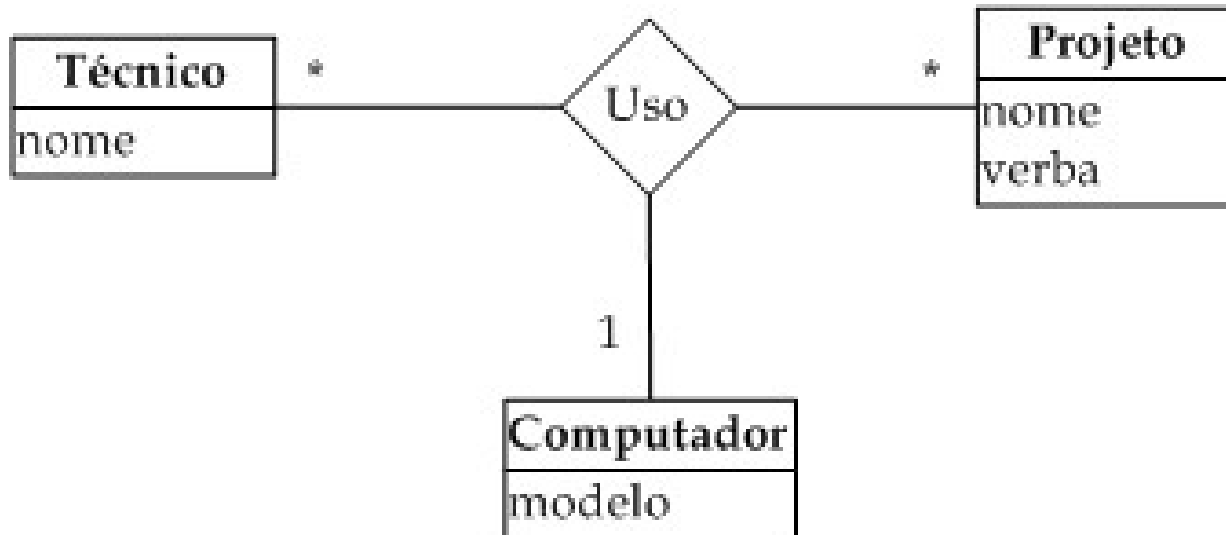
- É normalmente necessária quando duas ou mais classes estão associadas, e é necessário manter informações sobre esta associação.
- Uma classe associativa pode estar ligada a associações de qualquer tipo de conectividade.
- Sinônimo: *classe de associação*

Associações n-árias

- Define-se o *grau* de uma associação como a quantidade de classes envolvidas na mesma.
- Na notação da UML, as linhas de uma *associação n-ária* se interceptam em um losango.
- Na grande maioria dos casos práticos de modelagem, as associações normalmente são *binárias*.
- Quando o grau de uma associação é igual a três, dizemos que a mesma é *ternária*.
 - Uma associação ternária é o caso mais comum (menos raro) de associação n-ária ($n = 3$).

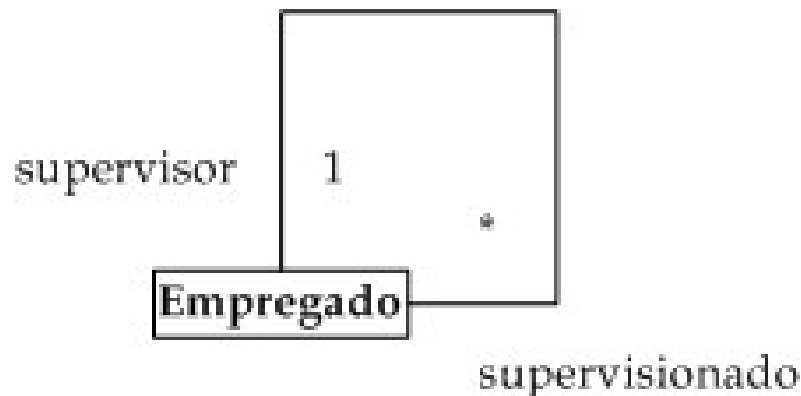
Exemplo (Associação Ternária)

- Na notação da UML, as linhas de uma associação n-ária se interceptam em um losango nomeado.
 - Notação similar ao do Modelo de Entidades e Relacionamentos



Associação Reflexiva

- É um tipo especial de associação que representa ligações entre objetos que pertencem a uma mesma classe.
 - Não indica que um objeto se associa a ele próprio.
- Quando se usa associações reflexivas, a definição de papéis é importante para evitar ambiguidades de leitura.
 - Cada objeto tem um papel distinto na associação.



Agregações e Composições (1/4)

- A *semântica* de uma associação corresponde ao seu significado, ou seja, à natureza conceitual da relação que existe entre os objetos que participam daquela associação.
- De todos os significados diferentes que uma associação pode ter, há uma categoria especial de significados, que representa *relações todo-parte*.

Agregações e Composições (2/4)

- Uma relação todo-parte entre dois objetos indica que um dos objetos está contido no outro. Podemos também dizer que um objeto contém o outro.
- A UML define dois tipos de relacionamentos todo-parte, a *agregação* e a *composição*.

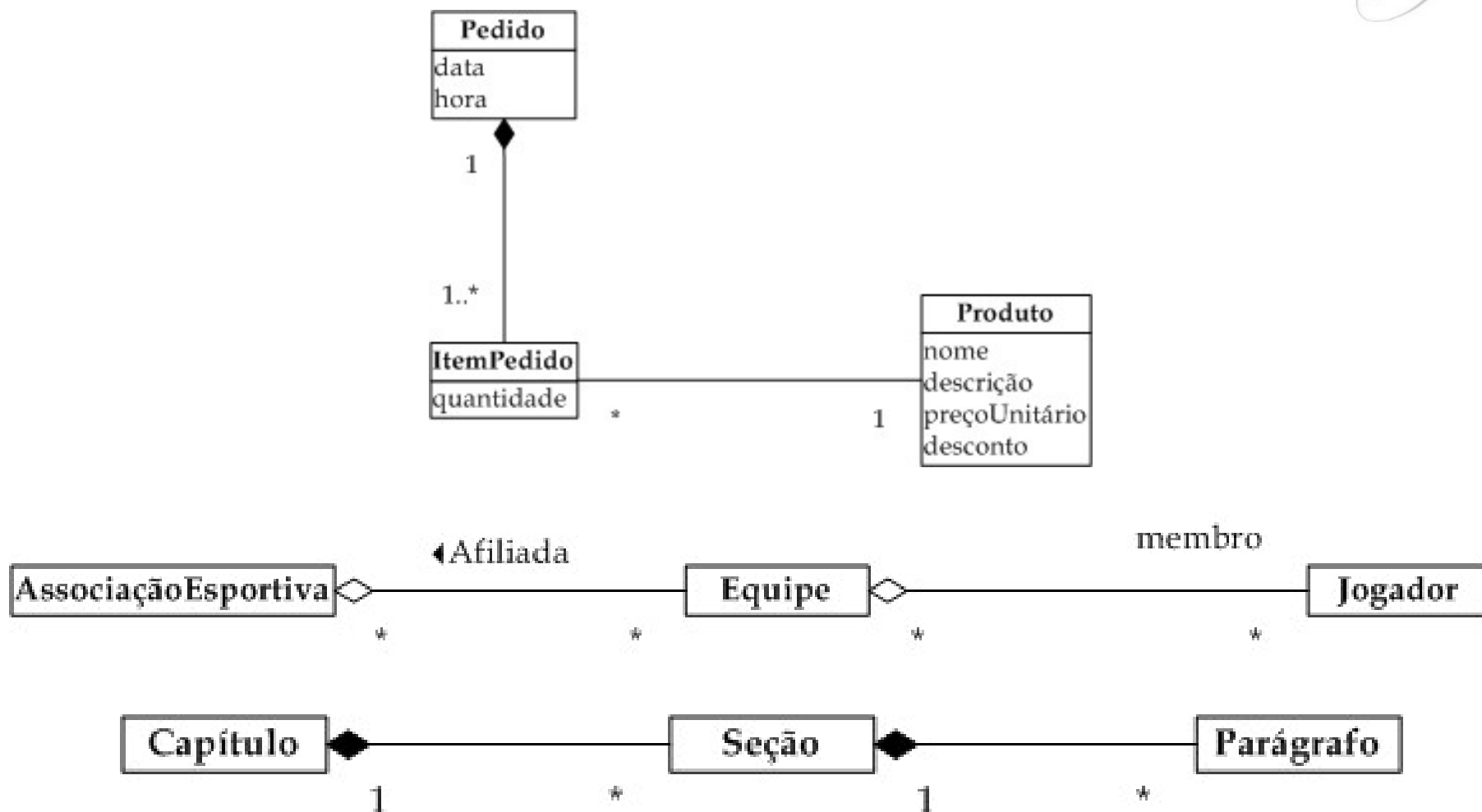
Agregações e Composições (3/4)

- Algumas particularidades das agregações/composições:
 - são assimétricas, no sentido de que, se um objeto A é parte de um objeto B, o objeto B não pode ser parte do objeto A.
 - propagam comportamento, no sentido de que um comportamento que se aplica a um todo automaticamente se aplica às suas partes.
 - as partes são normalmente criadas e destruídas pelo todo. Na classe do objeto todo, são definidas operações para adicionar e remover as partes.

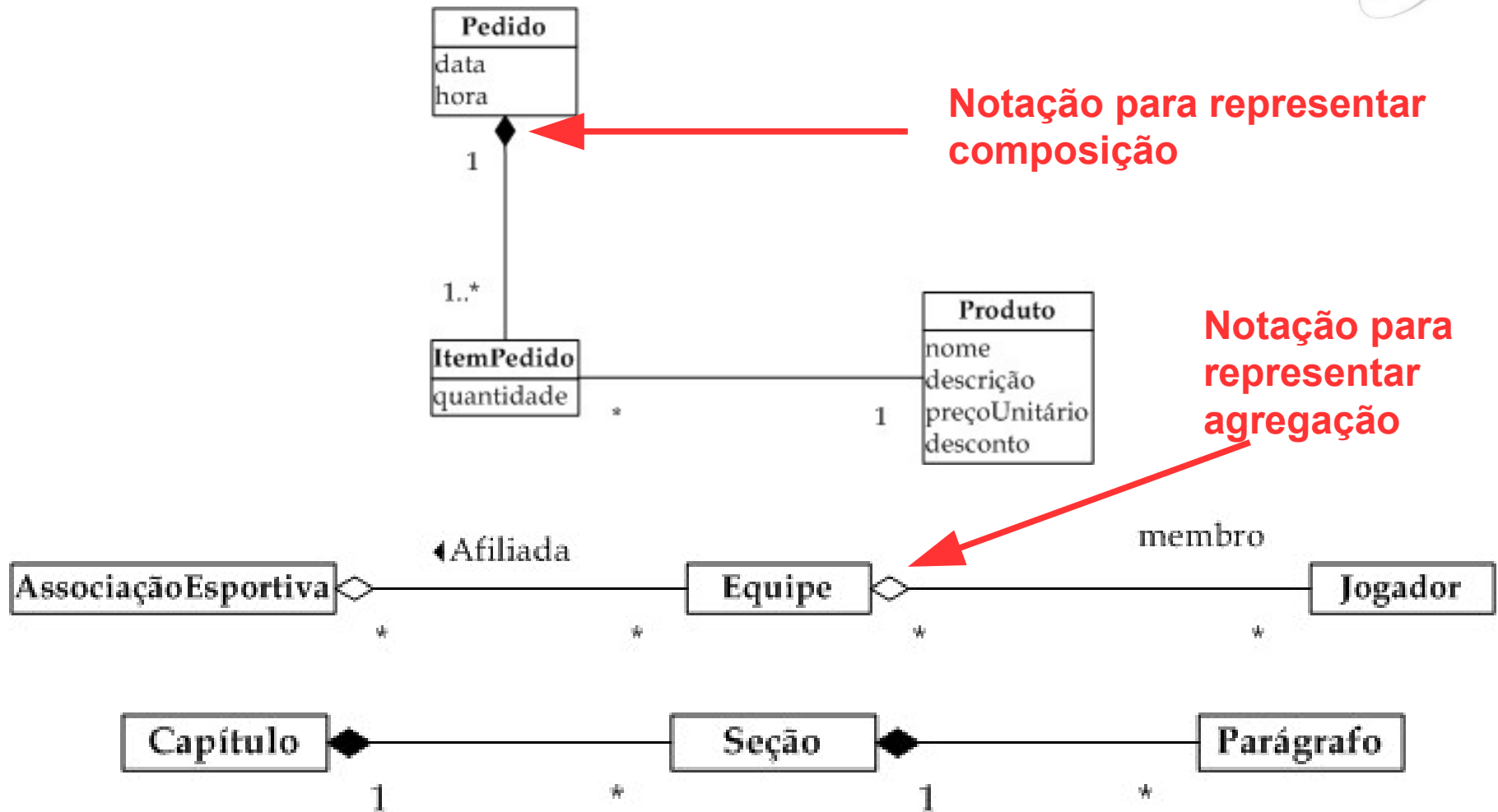
Agregações e Composições (4/4)

- As diferenças mais marcantes entre elas são:
 - Destruição de objetos
 - Na **agregação**, a destruição de um objeto todo não implica necessariamente na destruição do objeto parte.
 - Pertinência
 - Na **composição**, os objetos parte pertencem a um único todo.
 - Por essa razão, a composição é também denominada agregação não-compartilhada.
 - Em uma agregação, pode ser que um mesmo objeto participe como componente de vários outros objetos.
 - Por essa razão, a agregação é também denominada agregação compartilhada.

Exemplos

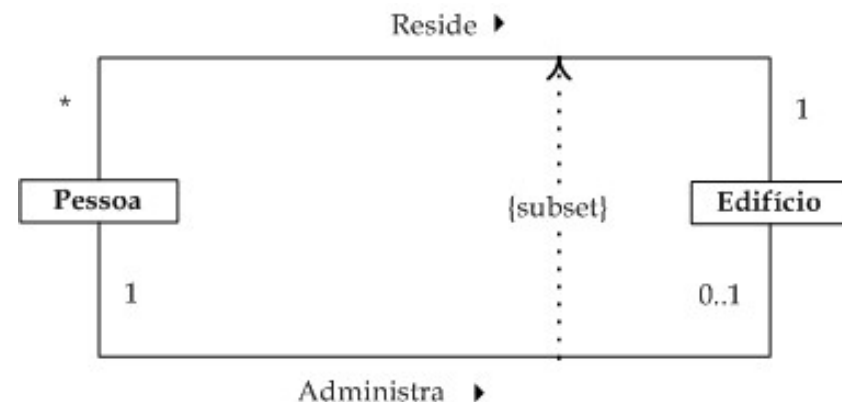
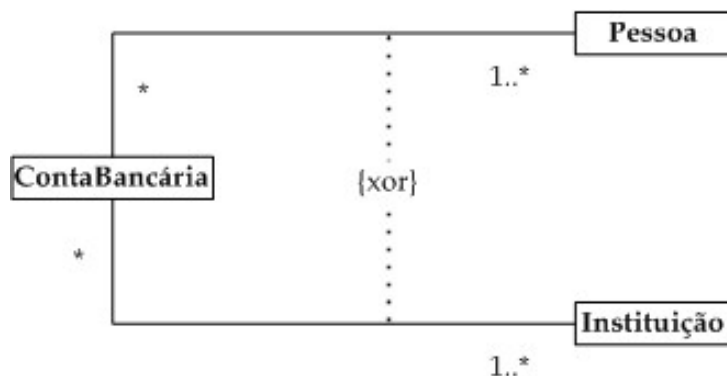


Exemplos

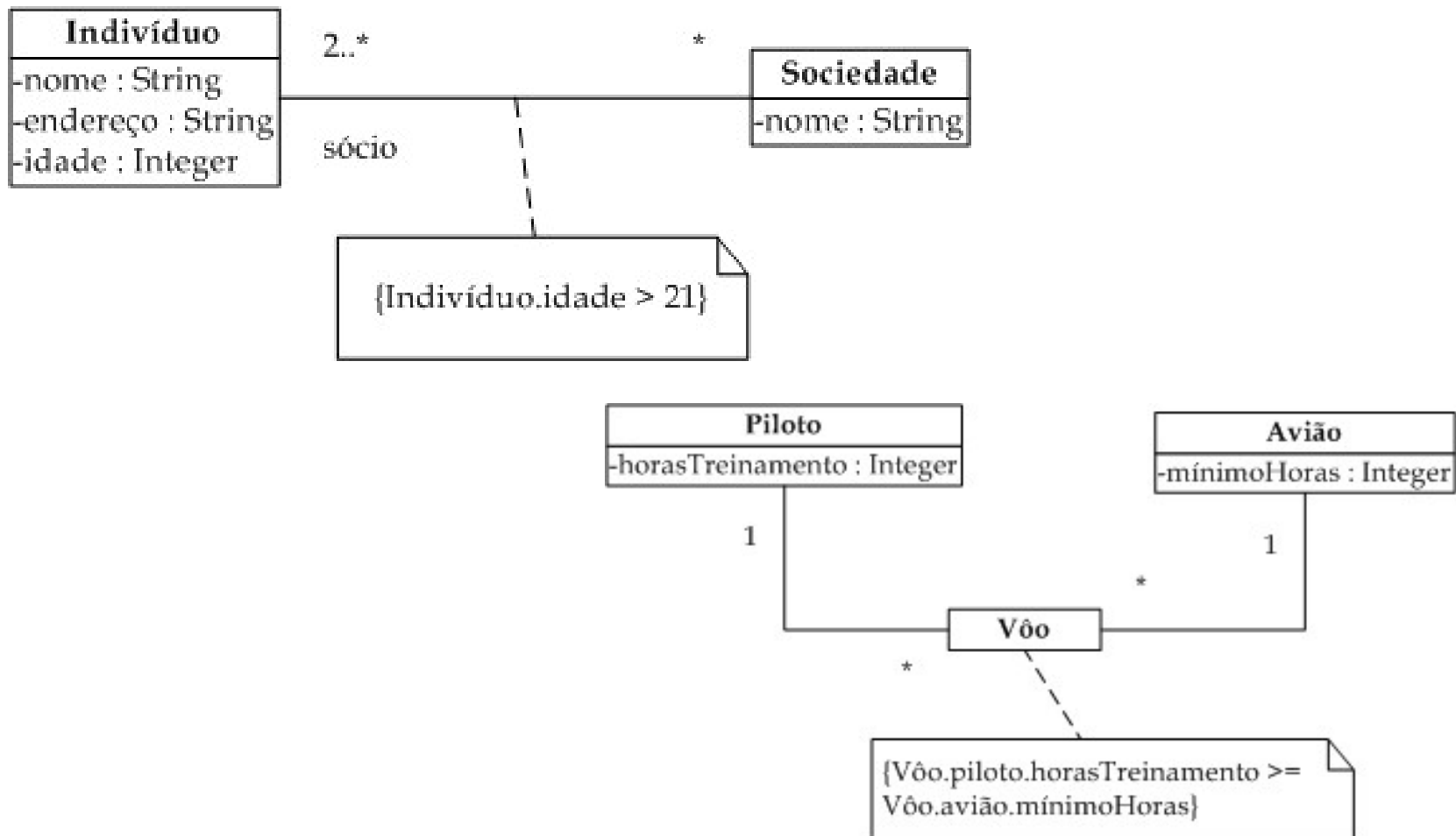


Restrições sobre associações

- Restrições OCL (Object Constraint Language) podem ser adicionadas sobre uma associação para adicionar a ela mais semântica.
 - Duas das restrições sobre associações predefinidas pela UML são **subset** e **xor**.
 - O modelador também pode definir suas próprias restrições em OCL.



Restrições sobre associações (cont)

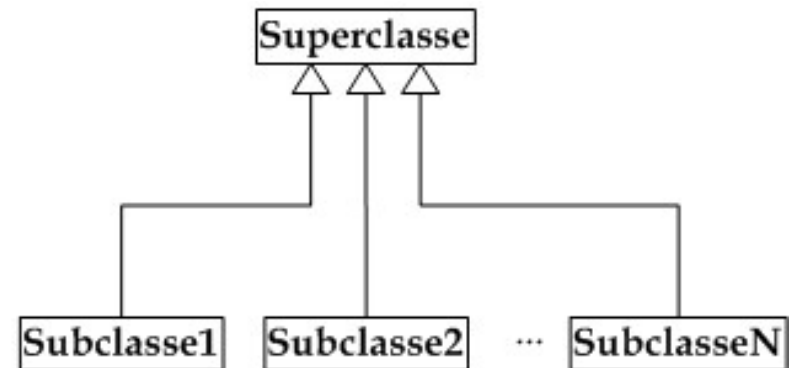
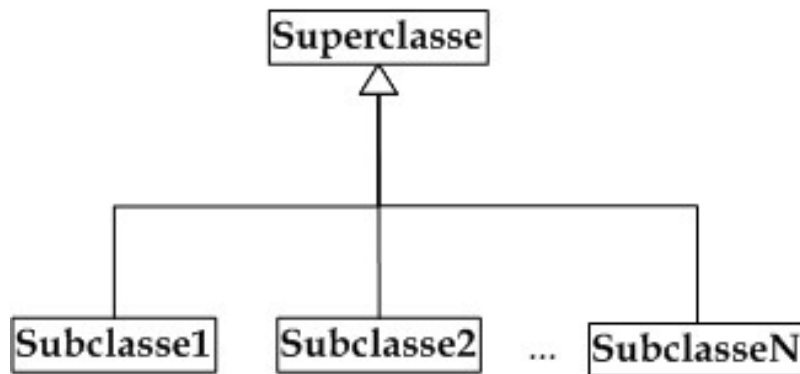


Generalizações e Especializações

- Relacionamentos entre classes denotam relações de generalidade ou especificidade entre as classes envolvidas.
 - Exemplos:
 - o conceito *mamífero* é mais genérico que o conceito *ser humano*.
 - o conceito *carro* é mais específico que o conceito *veículo*.
- Esse é o chamado ***relacionamento de herança***.
 - relacionamento de generalização/especialização

Generalizações e Especializações

- Terminologia
 - *superclasse X subclasse*
 - *supertipo X subtipo*
 - *classe base X classe herdeira*
 - *classe de generalização X classe de especialização*
 - *ancestral e descendente* (herança em vários níveis)
- Notação definida pela UML:

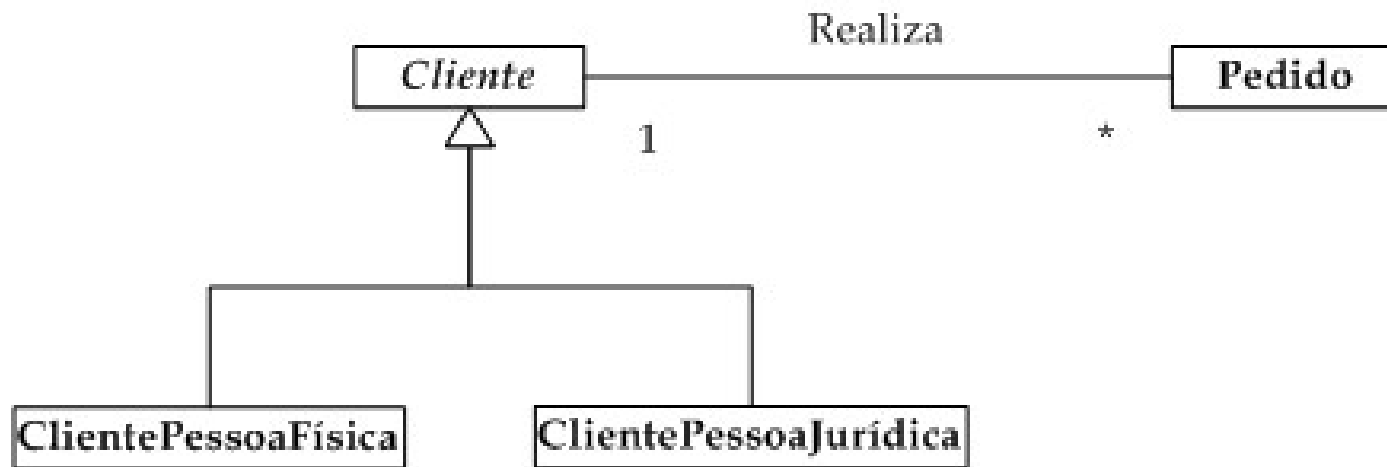


Semântica da Herança

- Subclasses herdam as características de sua superclasse.
 - É como se as características da superclasse estivessem definidas também nas suas subclasses.
 - Além disso, essa herança é transitiva e assimétrica.

Herança de Associações

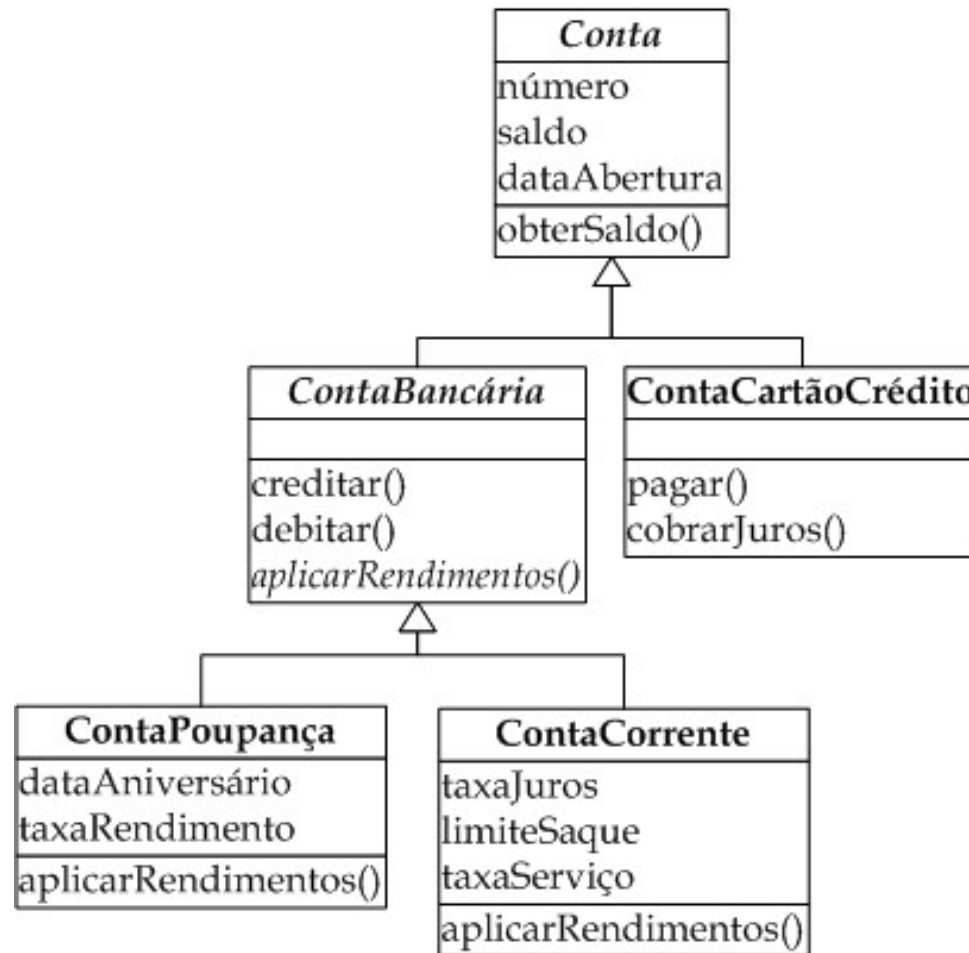
- Não somente atributos e operações, mas também associações são herdadas pelas subclasses.
- No exemplo abaixo, por herança, cada subclasse está associada a Pedido.



Propriedades da Herança

- **Transitividade:** uma classe em uma hierarquia herda propriedades e relacionamentos de todos os seus ancestrais.
 - Ou seja, a herança pode ser aplicada em vários níveis, dando origem a *hierarquia de generalização*.
 - Uma classe que herda propriedades de uma outra classe pode ela própria servir como superclasse.
- **Assimetria:** dadas duas classes A e B, se A for uma generalização de B, então B não pode ser uma generalização de A.
 - Não pode haver ciclos na hierarquia.

Exemplo de Hierarquia de Generalização



Classes Abstratas (1/2)

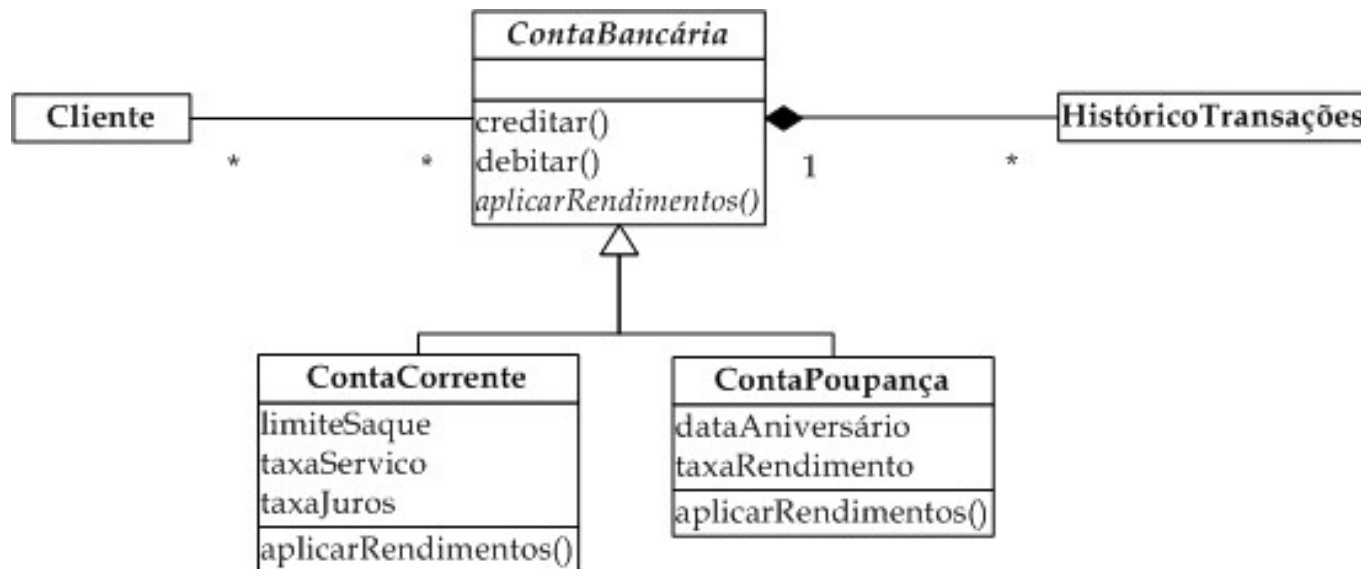
- Usualmente, a existência de uma classe se justifica pelo fato de haver a possibilidade de gerar instâncias da mesma.
 - Essas são as *classes concretas*.
- No entanto, podem existir classes que não geram instâncias diretas.
 - Essas são as *classes abstratas*.

Classes Abstratas (2/2)

- Classes abstratas são utilizadas para organizar e simplificar uma hierarquia de generalização.
 - Propriedades comuns a diversas classes podem ser organizadas e definidas em uma classe abstrata, a partir da qual as primeiras herdam.
- Subclasses de uma classe abstrata também podem ser abstratas, mas a hierarquia deve terminar em uma ou mais classes concretas.

Notação para classes abstratas

- Na UML, uma classe abstrata é representada com o seu nome em *itálico*.
- No exemplo a seguir, *ContaBancária* é uma classe abstrata.



Refinamento do Modelo com Herança

- Critérios a avaliar na criação de subclasses:
 - A subclasse tem atributos adicionais.
 - A subclasse tem associações.
 - A subclasse é manipulada (ou reage) de forma diferente da superclasse.
- Sempre se assegure de que se trata de um relacionamento do tipo "é-um":
 - *"X é um tipo de Y?"*

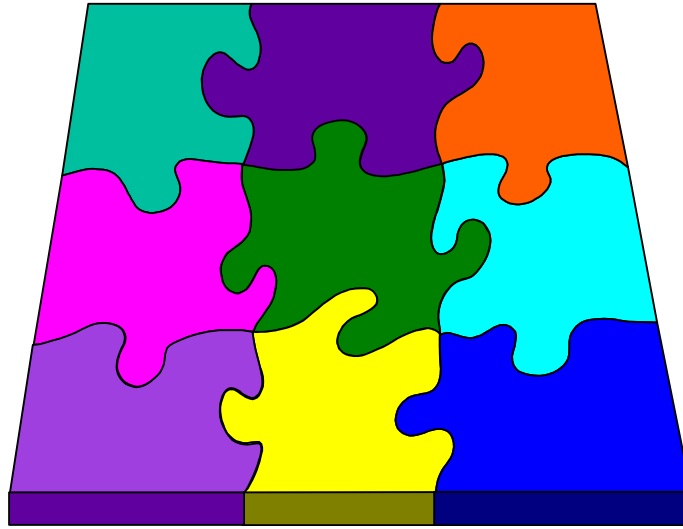
Refinamento do Modelo com Herança

- A regra "é-um" é mais formalmente conhecida como regra da substituição ou princípio de Liskov.

Regra da Substituição: sejam duas classes A e B, onde A é uma generalização de B. Não pode haver diferenças entre utilizar instâncias de B ou de A, do ponto de vista dos clientes de A.

Barbara Liskov (<http://www.pmg.csail.mit.edu/~liskov/>)





Técnicas para identificação de classes

Qual é o Desafio?

Apesar de todas as vantagens que a OO pode trazer ao desenvolvimento de software, um problema fundamental ainda persiste: identificar corretamente e completamente objetos (classes), atributos e operações.

Técnicas de Identificação

- Várias técnicas (de uso não exclusivo) são usadas para identificar classes:
 1. Categorias de Conceitos
 2. Análise Textual de Abbott (*Abbot Textual Analysis*)
 3. Análise de Casos de Uso
 4. Padrões de Análise (Analysys Patterns)
 5. Identificação Dirigida a Responsabilidades

Técnicas de Identificação

- Várias técnicas (de uso não exclusivo) são usadas para identificar classes:
 1. Categorias de Conceitos
 2. Análise Textual de Abbott (*Abbot Textual Analysis*)
 3. Análise de Casos de Uso
 4. Padrões de Análise (*Analysys Patterns*)
 5. Identificação Dirigida a Responsabilidades

Análise Textual de Abbott (1983)

- **Estratégia:** identificar termos da narrativa de casos de uso e documento de requisitos que podem sugerir classes, atributos, operações.
 - São utilizadas diversos artefatos sobre o sistema: documento e requisitos, modelos do negócio, glossários, conhecimento sobre o domínio, etc.
 - Para cada um desses artefatos, os nomes (substantivos e locuções equivalentes a substantivos) que aparecem no mesmo são destacados.
 - Após isso, os sinônimos são removidos (permanecem os nomes mais significativos para o domínio do negócio em questão).

Análise Textual de Abbott (cont.)

- Cada termo remanescente se encaixa em uma das situações a seguir:
 - O termo se torna uma classe (ou seja, são classes candidatas)
 - O termo se torna um atributo
 - O termo não tem relevância alguma com software pretendido

Análise Textual de Abbott (cont.)

- Abbott também preconiza o uso de sua técnica na identificação de operações e de associações.
 - Para isso, ele sugere que sejam destacados os verbos no texto.
 - Verbos de ação (e.g., calcular, confirmar, cancelar, comprar, fechar, estimar, depositar, sacar, etc.) são operações em potencial.
 - Verbos com sentido de "ter" são potenciais agregações ou composições.
 - Verbos com sentido de "ser" são generalizações em potencial.
 - Demais verbos são associações em potencial.

Análise Textual de Abbott (cont.)

- Algumas observações:
 - Dependendo do estilo que foi utilizado para escrever os documentos utilizados como base, essa técnica pode levar à identificação de diversas classes candidatas que não gerarão classes.
 - A análise do texto de um documento pode não deixar explícita uma classe importante para o sistema.
 - Em linguagem natural, as variações linguísticas e as formas de expressar uma mesma idéia são bastante numerosas.

Análise de Casos de Uso

- Essa técnica é também chamada de identificação dirigida por casos de uso, sendo um caso particular da técnica de Abbott.
- Técnica preconizada pelo Processo Unificado.

Análise de Casos de Uso (cont.)

- O MCU é utilizado como ponto de partida.
 - Premissa: um caso de uso corresponde a um comportamento específico do software. Esse comportamento somente pode ser produzido por objetos que compõem o sistema.
 - Com base nisso, o modelador aplica a técnica de análise dos casos de uso para identificar as classes necessárias à produção do comportamento que está documentado na descrição do caso de uso.

Análise de Casos de Uso

- Procedimento de aplicação:
 - O modelador estuda a descrição textual de cada caso de uso para identificar classes candidatas.
 - Para cada caso de uso, se texto (fluxos principal, alternativos e de exceção, pós-condições e pré-condições, etc.) é analisado.
 - Na análise de certo caso de uso, o modelador tenta identificar classes que possam fornecer o comportamento do mesmo.
 - Na medida em que os casos de uso são analisados um a um, as classes do software são identificadas.
 - Quando todos os casos de uso tiverem sido analisados, todas as classes (ou pelo menos a grande maioria delas) terão sido identificadas.

Leitura Complementar:

Categorização BCE

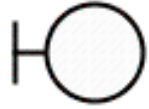
- Na aplicação deste procedimento, podemos utilizar as categorização BCE, na qual os objetos são agrupados de acordo com o tipo de responsabilidade a eles atribuída.
 - objetos de **entidade**: usualmente objetos do domínio do problema
 - objetos de **fronteira**: atores interagem com esses objetos
 - objetos de **controle**: servem como intermediários entre objetos de fronteira e de entidade, definindo o comportamento de um caso de uso específico.
- Categorização proposta por Jacobson (1992).
 - Possui correspondência com o padrão *model-view-controller* (MVC)
- Estereótipos na UML: «boundary», «entity», «control»



Leitura Complementar:

Objetos de Entidade

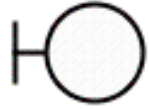
- Repositório para *informações* e as *regras de negócio* manipuladas pelo sistema.
 - Representam conceitos do domínio do negócio.
- Características
 - Normalmente armazenam informações persistentes.
 - Várias instâncias da mesma entidade existindo no sistema.
 - Participam de vários casos de uso e têm ciclo de vida longo.
- Exemplo:
 - Um objeto *Pedido* participa dos casos de uso *Realizar Pedido* e *Atualizar Estoque*. Este objeto pode existir por diversos anos ou mesmo tanto quanto o próprio sistema.



Leitura Complementar:

Objetos de Fronteira

- Realizam a comunicação do sistema com os atores.
 - traduzem os eventos gerados por um ator em eventos relevantes ao sistema → eventos de sistema.
 - também são responsáveis por apresentar os resultados de uma interação dos objetos em algo inteligível pelo ator.
- Existem para que o sistema se comunique com o mundo exterior.
 - Por consequência, são altamente dependentes do ambiente.



Leitura Complementar:

Objetos de Fronteira



- Há dois tipos principais de objetos de fronteira:
 - Os que se comunicam com o usuário (atores humanos): relatórios, páginas HTML, interfaces gráfica desktop, etc.
 - Os que se comunicam com atores não-humanos (outros sistemas ou dispositivos): protocolos de comunicação.



Leitura Complementar:

Objetos de Controle



- São a “ponte de comunicação” entre objetos de fronteira e objetos de entidade.
- Responsáveis por controlar a lógica de execução correspondente a um caso de uso.
- Decidem o que o sistema deve fazer quando um evento de sistema ocorre.
 - Eles realizam o controle do processamento
 - Agem como **gerentes** (coordenadores, controladores) dos outros objetos para a realização de um caso de uso.
- Traduzem eventos de sistema em operações que devem ser realizadas pelos demais objetos.

Leitura Complementar:

Importância da Categorização BCE



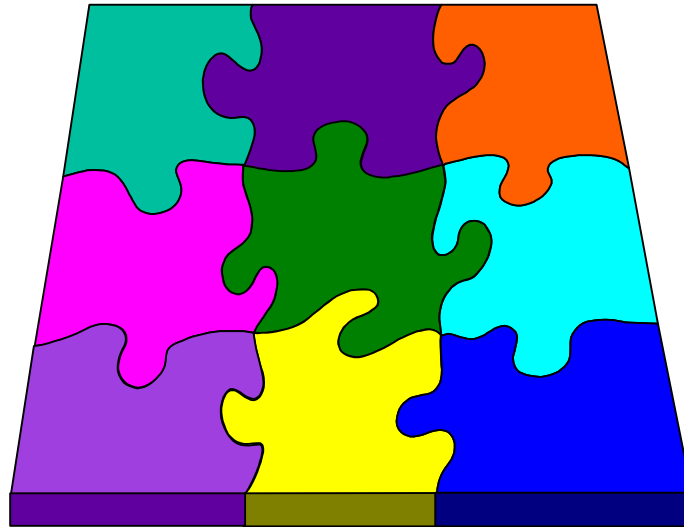
- A categorização BCE parte do princípio de que cada objeto em software OO é especialista em realizar um de três tipos de tarefa, a saber:
 - se comunicar com atores (**fronteira**),
 - manter as informações (**entidade**) ou
 - coordenar a realização de um caso de uso (**controle**).
- A categorização BCE é uma "receita de bolo" para identificar objetos participantes da realização de um caso de uso.

Leitura Complementar:

Importância da Categorização BCE (cont.)



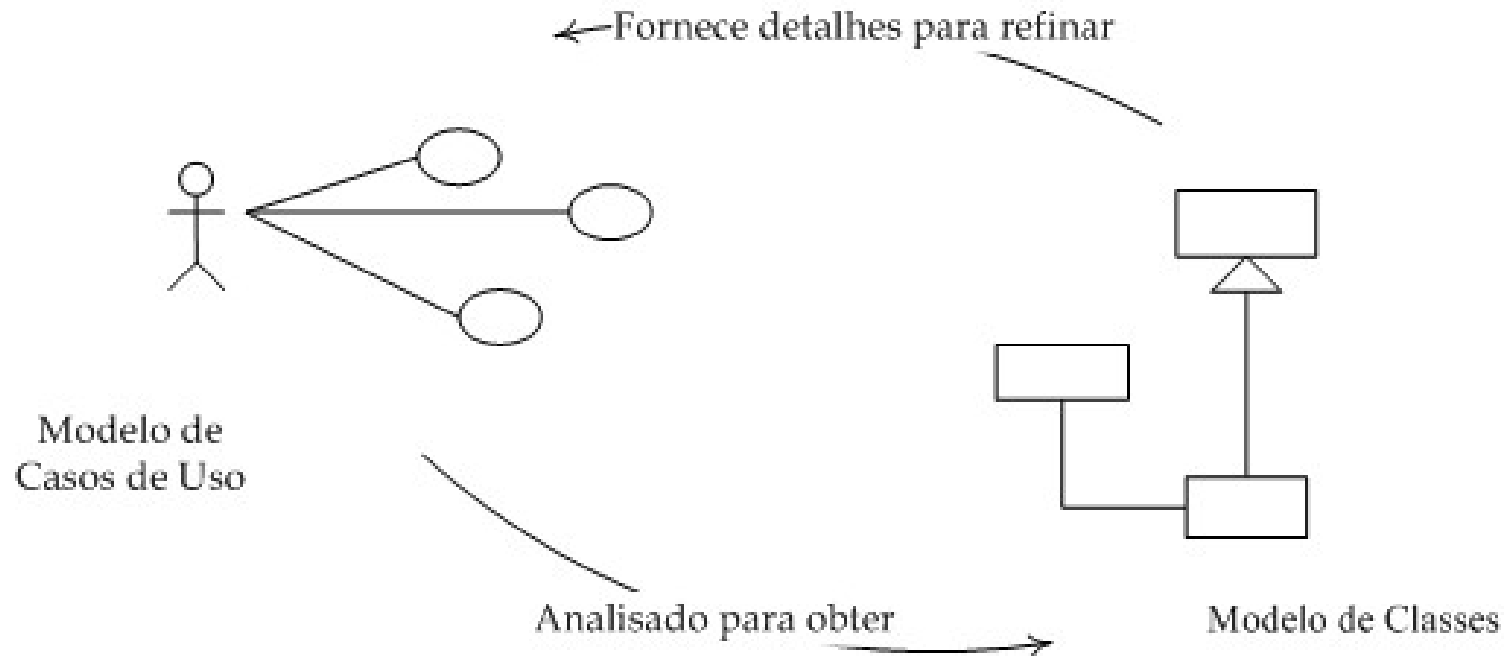
- A importância dessa categorização está relacionada à capacidade de adaptação a eventuais mudanças.
 - Se cada objeto tem atribuições específicas dentro do sistema, mudanças podem ser menos complexas e mais localizadas.
 - Uma modificação em uma parte do sistema tem menos possibilidades de resultar em mudanças em outras partes.



**Modelo de classes no processo
de desenvolvimento**

Modelo de classes no processo de desenvolvimento

- Interdependência entre o modelo de casos de uso e o modelo de classes.



Referências

- ABBOTT, R.: Program design by informal English descriptions. In: Communications of the ACM, 26 (11), 1983.
- BEZERRA, E.: Princípios de Análise e Projeto de Sistemas com UML, 3ª edição, Campus - Elsevier, 2015.
- JACOBSON, I.: Object-oriented software engineering: a use case driven approach. ACM Press, 1993.