

Princípios de Análise e Projeto de Sistemas com UML

2ª edição

Eduardo Bezerra

Editora Campus/Elsevier

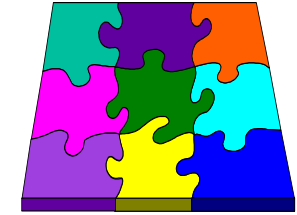


Capítulo 5

Modelagem de Classes de Análise

“O engenheiro de software amador está sempre à procura da mágica, de algum método sensacional ou ferramenta cuja aplicação promete tornar trivial o desenvolvimento de software. É uma característica do engenheiro de software profissional saber que tal panacéia não existe” -Grady Booch

Tópicos



- Introdução
- Diagrama de classes
- Diagrama de objetos
- Técnicas para identificação de classes
- Construção do modelo de classes
- Modelo de classes no processo de desenvolvimento

Introdução

- As funcionalidades de um SSOO é são realizadas internamente através de *colaborações* entre objetos.
 - Externamente, os atores visualizam resultados de cálculos, relatórios produzidos, confirmações de requisições realizadas, etc.
 - Internamente, os objetos colaboram uns com os outros para produzir os resultados.
- Essa colaboração pode ser vista sob o *aspecto dinâmico* e sob o *aspecto estrutural estático*.
- O **modelo de objetos** representa o aspecto estrutural e estático dos objetos que compõem um SSOO.
- Dois diagramas da UML são usados na construção do modelo de objetos:
 - **diagrama de classes**
 - **diagrama de objetos**

Introdução

- Na prática o diagrama de classes é bem mais utilizado que o diagrama de objetos.
 - Tanto que o modelo de objetos é também conhecido como modelo de classes.
- Esse modelo *evolui* durante o desenvolvimento do SSOO.
 - À medida que o SSOO é desenvolvido, o modelo de objetos é incrementado com novos detalhes.
- Há três níveis sucessivos de detalhamento:
 - *Análise → Especificação (Projeto) → Implementação.*

Objetivo da Modelagem de Classes

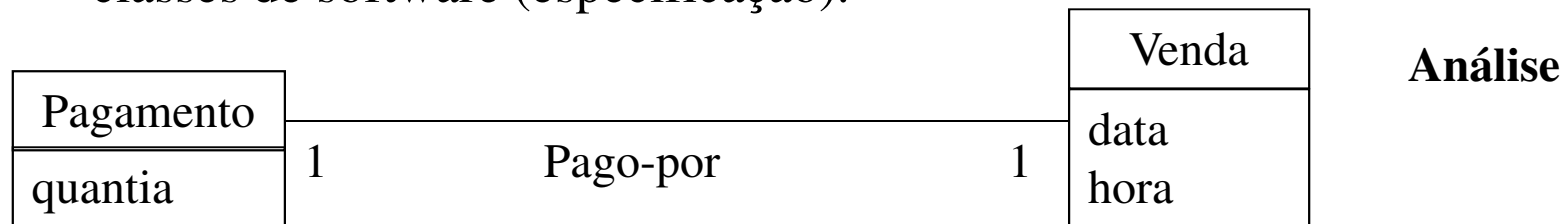
- O objetivo da modelagem de classes de análise é prover respostas para as seguintes perguntas:
 - Por definição um sistema OO é composto de objetos...em um nível alto de abstração, **que objetos constituem o sistema em questão?**
 - Quais são as classes candidatas?
 - Como as classes do sistema estão relacionadas entre si?
 - Quais são as responsabilidades de cada classe?

Modelo de Classes de Análise

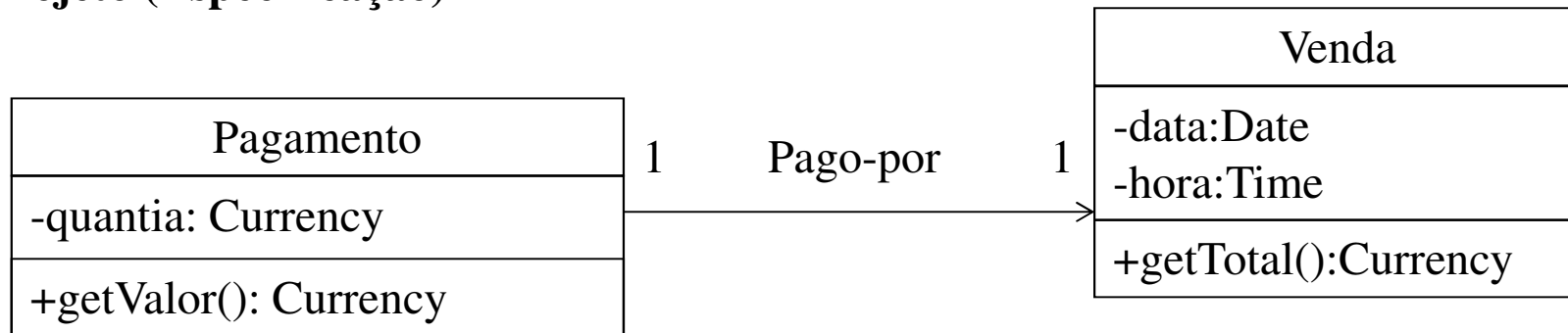
- Representa termos do domínio do negócio.
 - idéias, coisas, e conceitos no mundo real.
- Objetivo: descrever o problema representado pelo sistema a ser desenvolvido, sem considerar características da solução a ser utilizada.
- É um dicionário “visual” de conceitos e informações relevantes ao sistema sendo desenvolvido.
- Duas etapas:
 - modelo conceitual (modelo de domínio).
 - modelo da aplicação.
- Elementos de notação do diagrama de classes normalmente usados na construção do modelo de análise:
 - classes e atributos; associações, composições e agregações (com seus adornos); classes de associação; generalizações (herança).

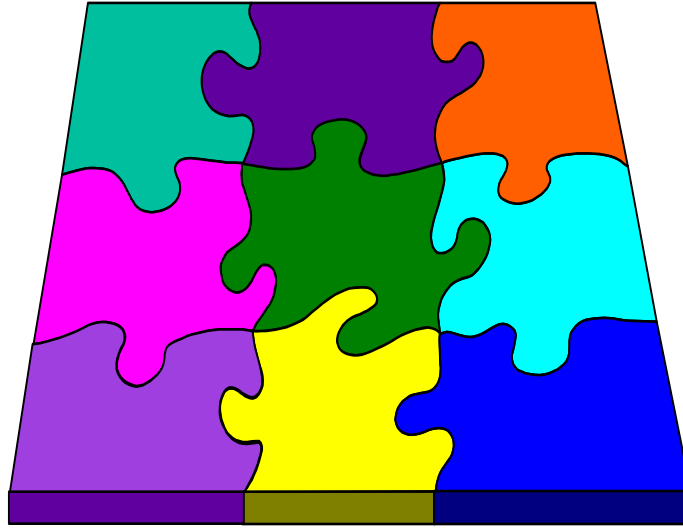
Modelo de Análise: Foco no Problema

- O modelo de análise não representa detalhes da solução do problema.
 - Embora este sirva de ponto de partida para uma posterior definição das classes de software (especificação).



Projeto (Especificação)





5.2 Diagrama de classes

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 |

Exemplo (classe ContaBancária)

ContaBancária

| ContaBancária |
|---------------|
| número |
| saldo |
| dataAbertura |

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

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

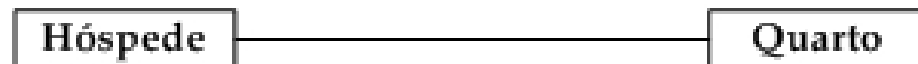
| ContaBancária |
|-------------------------------|
| -número : String |
| -saldo : Quantia |
| -dataAbertura : Date |
| +criar() |
| +bloquear() |
| +desbloquear() |
| +creditar(in valor : Quantia) |
| +debitar(in valor : Quantia) |

Associações

- Para representar o fato de que objetos podem se relacionar uns com os outros, utilizamos **associações**.
- Uma associação representa 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 | $l_i..l_s$ |

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.

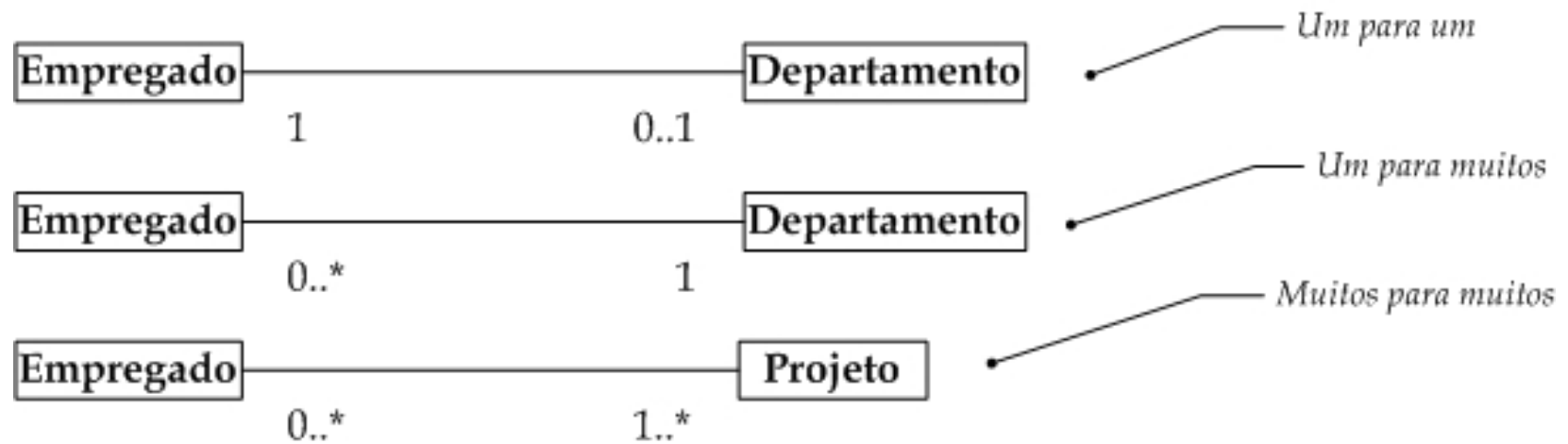
- Exemplo 
 - Uma corrida está associada a, no mínimo, dois velocistas
 - Uma corrida está associada a, no máximo, seis velocistas.
 - Um velocista *pode* estar associado a várias corridas.

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)

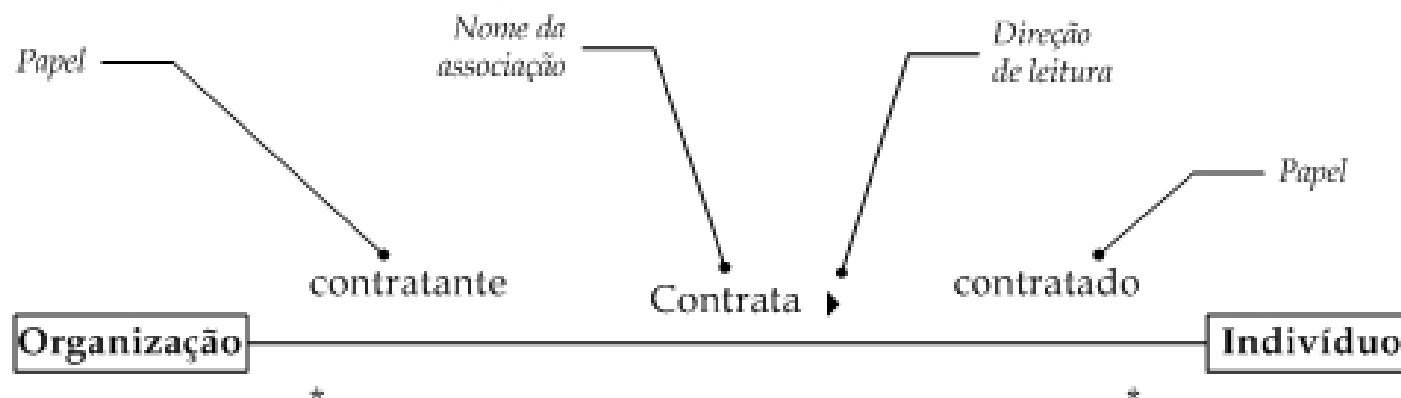


Participação

- 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 algum significado semântico a mesma.
 - *Direção de leitura*: indica como a associação deve ser lida
 - *Papel*: para representar um papel específico em uma associação.

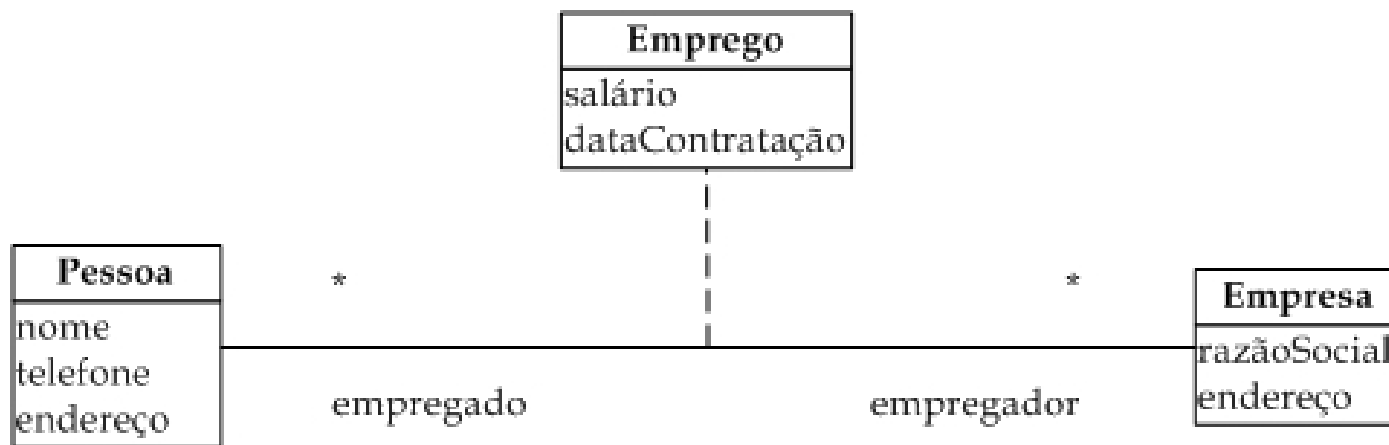


Classe associativa

- É uma classe que está ligada a uma associação, em vez de estar ligada a outras classes.
- 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*

Notação para Classes Associativas

- 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.

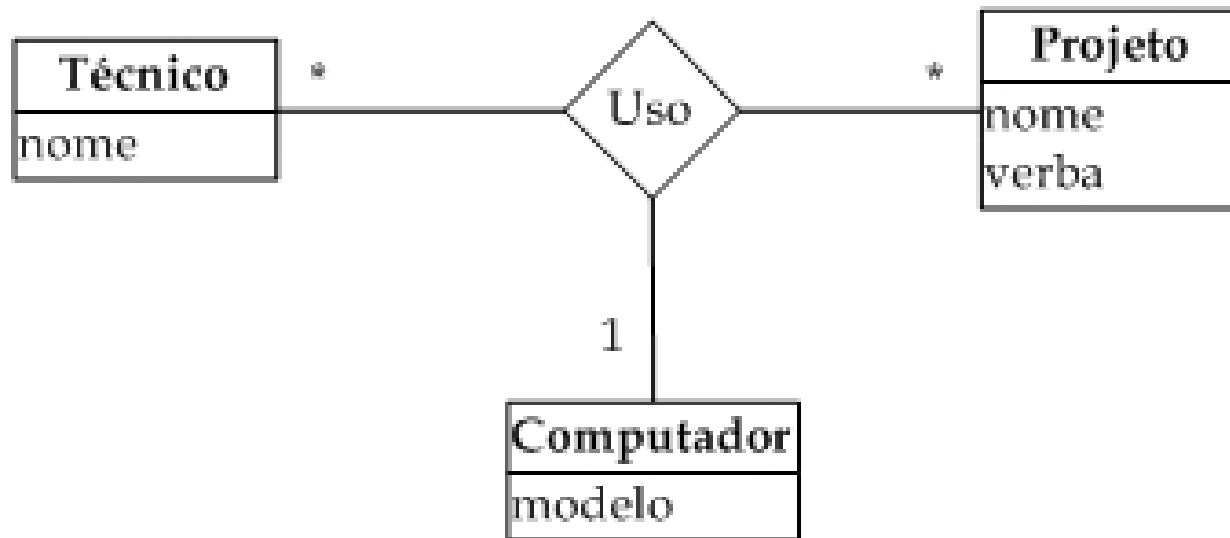


Associações n-árias

- 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 são uma caso mais comum (menos raro) de associação n-ária ($n = 3$).

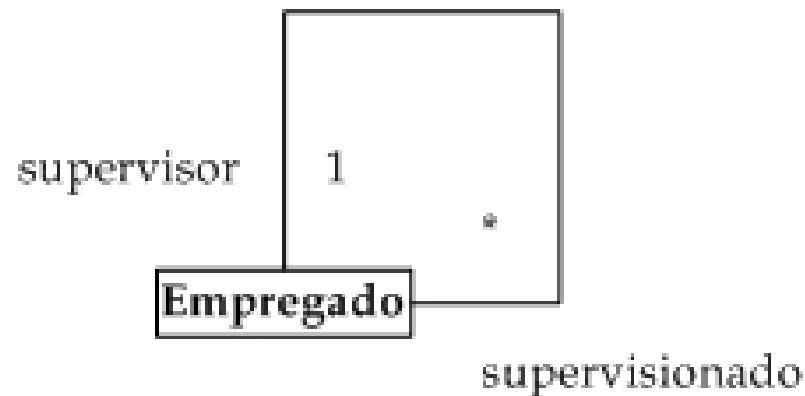
Exemplo (associação ternária)

- 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



Associações reflexivas

- 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 ambigüidades na leitura da associação.
 - Cada objeto tem um papel distinto na associação.



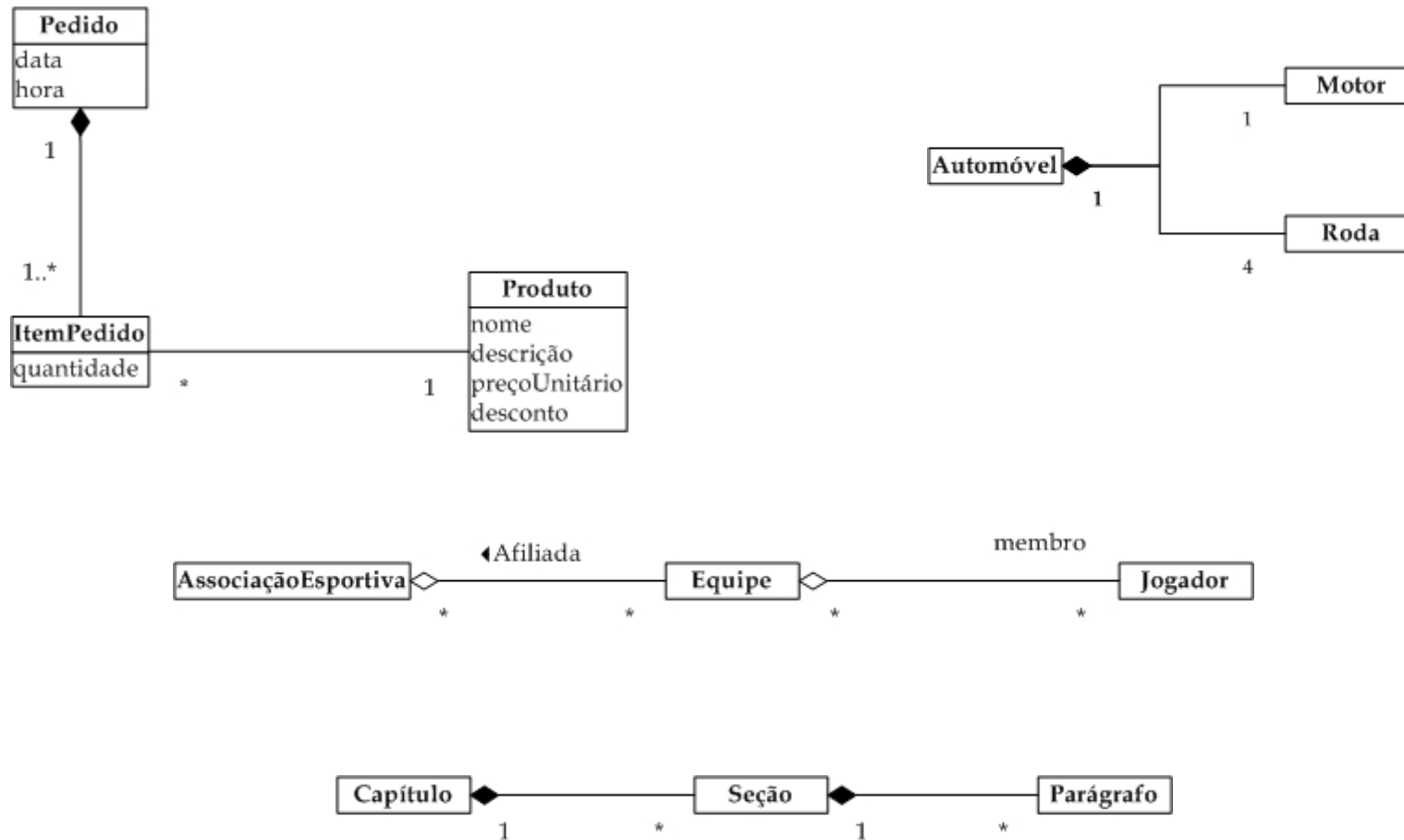
Agregações e Composições

- 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*.
- 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

- 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.
- Se uma das perguntas a seguir for respondida com um sim, provavelmente há uma agregação onde X é todo e Y é parte.
 - *X tem um ou mais Y?*
 - *Y é parte de X?*

Exemplos

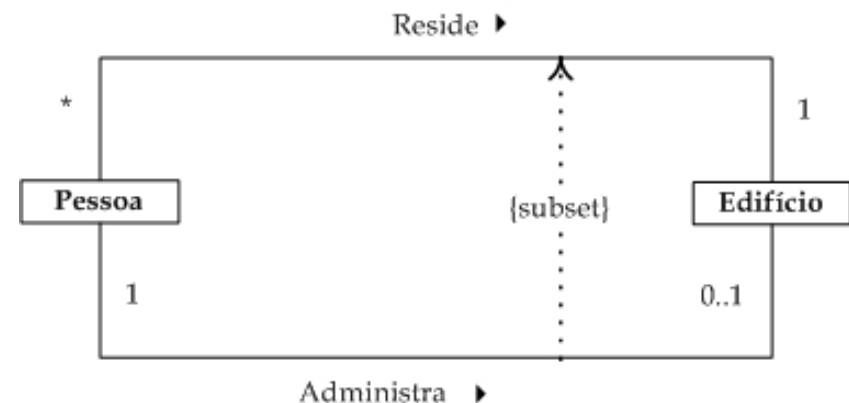
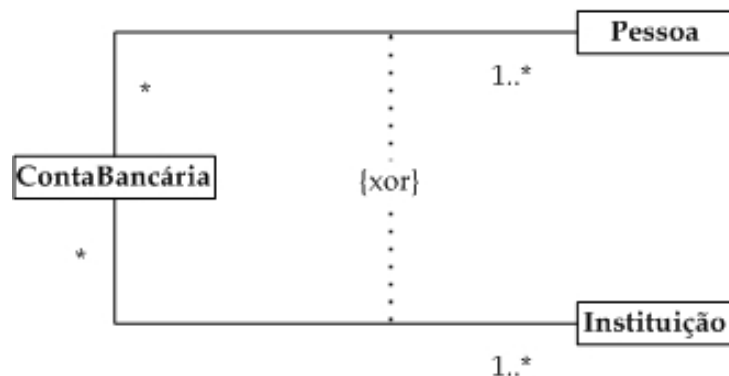


Agregações e Composições

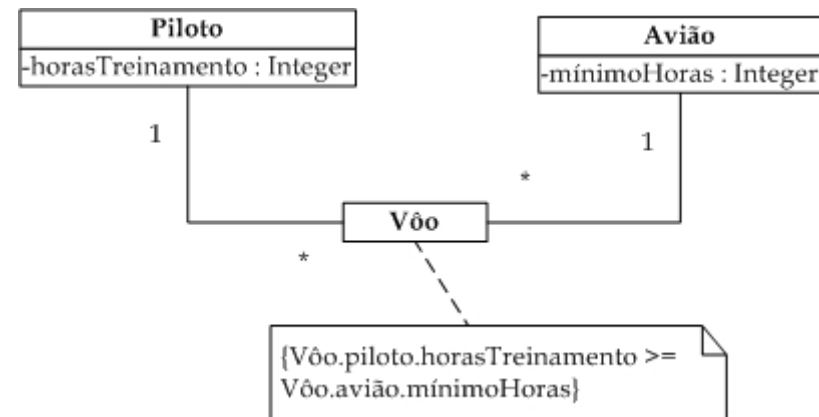
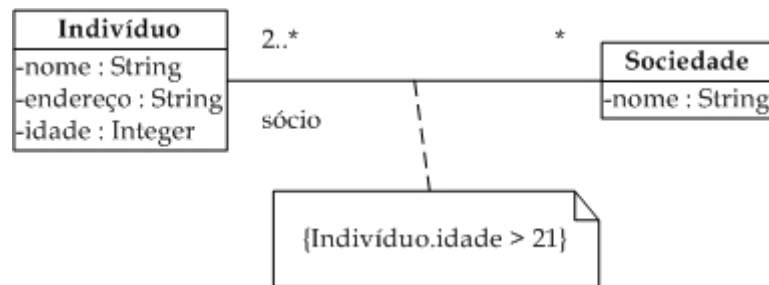
- As diferenças entre a agregação e composição não são bem definidas. A seguir, as diferenças mais marcantes entre elas.
- Destruição de objetos
 - Na agregação, a destruição de um objeto todo não implica necessariamente na destruição do objeto parte.
- Pertinência
 - Na composição, os objetos parte 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.

Restrições sobre associações

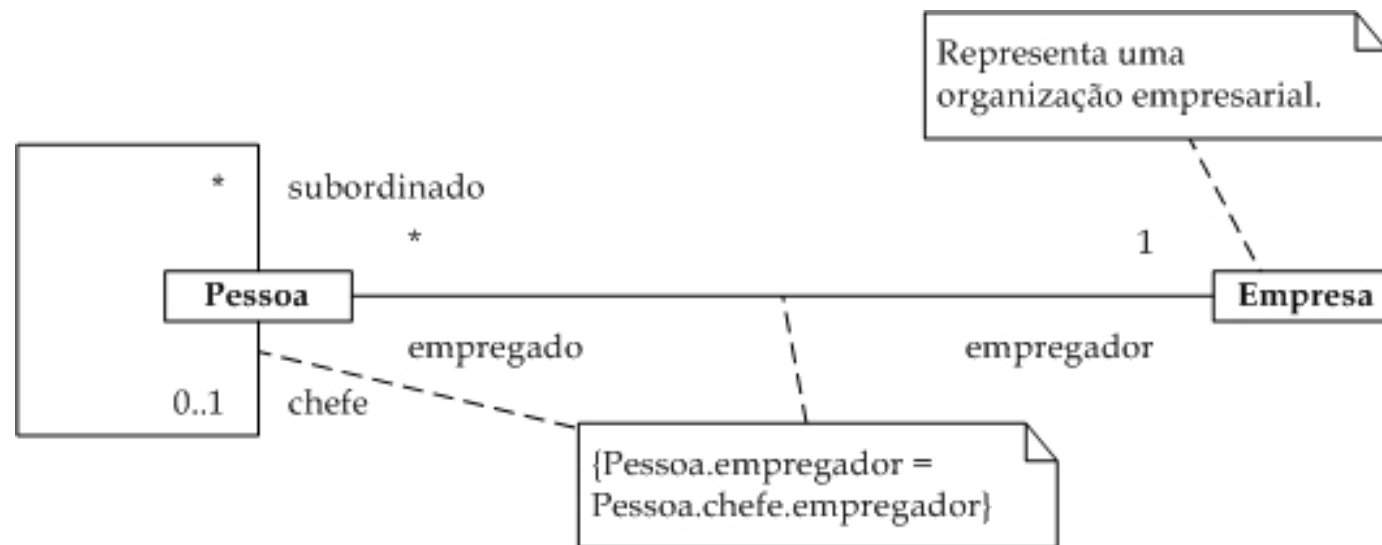
- Restrições OCL podem ser adicionadas sobre uma associação para adicionar a ela mais semântica.
 - Duas das restrições sobre associações predefinidas pela UML são **subset** e **xor**.
 - O modelador também pode definir suas próprias restrições em OCL.



Restrições sobre associações (cont)



Restrições sobre associações (cont)

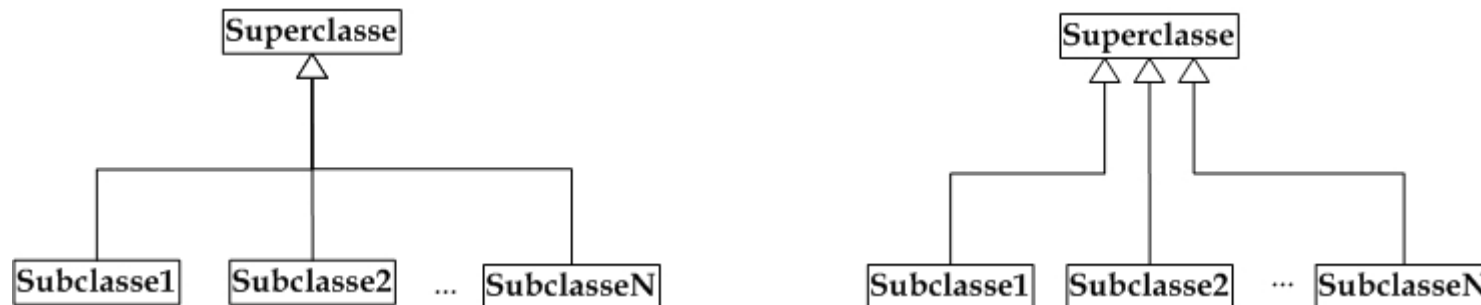


Generalizações e Especializações

- O modelador também pode representar relacionamentos entre classes.
 - Esses denotam relações de generalidade ou especificidade entre as classes envolvidas.
 - Exemplo: o conceito *mamífero* é mais genérico que o conceito *ser humano*.
 - Exemplo: o conceito *carro* é mais específico que o conceito *veículo*.
- Esse é o chamado ***relacionamento de herança***.
 - relacionamento de generalização/especialização
 - relacionamento de gen/espec

Generalizações e Especializações

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

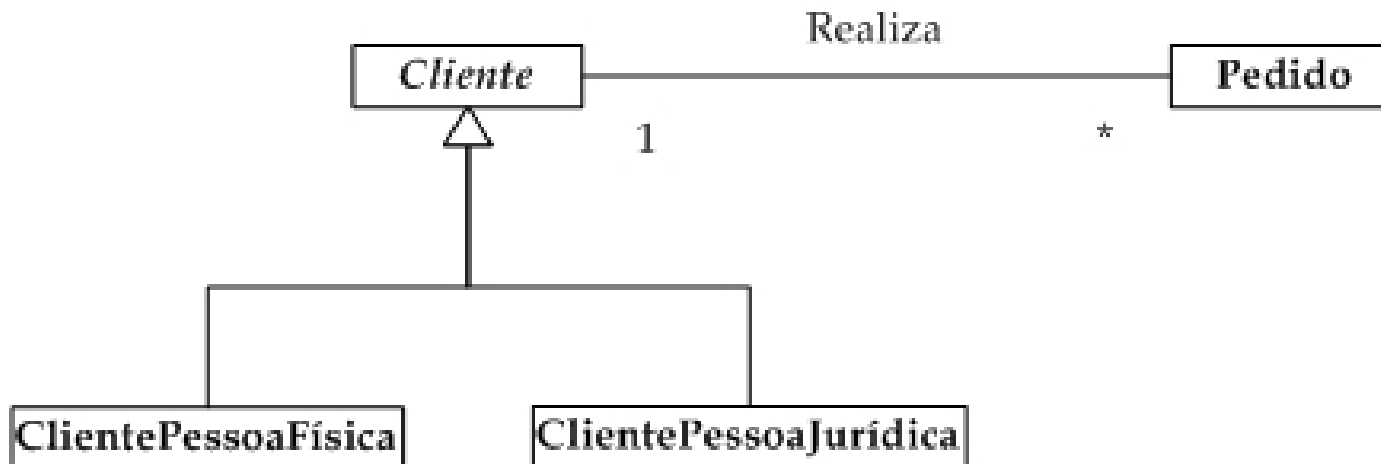


Semântica da Herança

- 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 anti-simétrica
- Note a diferença semântica entre a herança e a associação.
 - A primeira trata de um relacionamento entre classes, enquanto que a segunda representa relacionamentos entre instâncias de classes.
 - Na associação, objetos específicos de uma classe se associam entre si ou com objetos específicos de outras classes.
 - Exemplo:
 - Herança: “Gerentes são tipos especiais de funcionários”.
 - Associação: “Gerentes chefiam departamentos”.

Herança de Associações

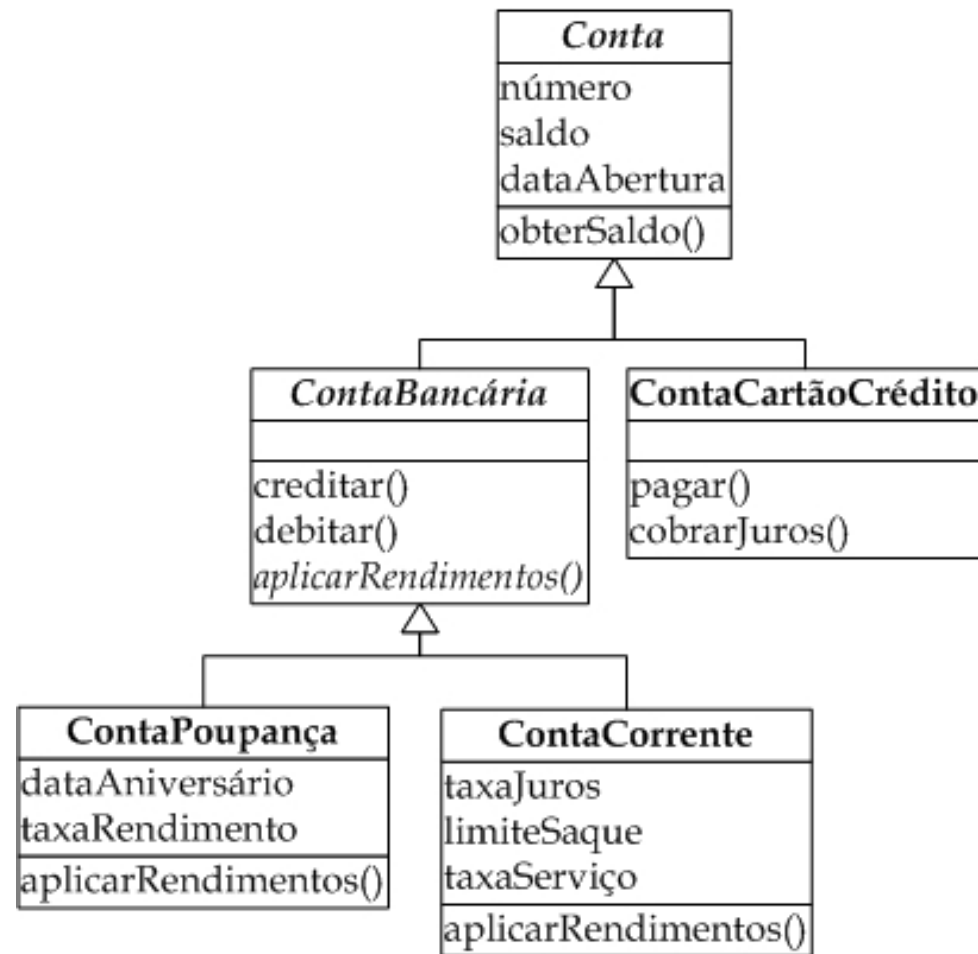
- Não somente atributos e operações, mas também associações são herdadas pelas subclasses.
- No exemplo abaixo, cada subclasse está associada a Pedido, por herança.



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.
 - Ou seja, *não* pode haver ciclos em uma hierarquia de generalização.

Propriedades da Herança

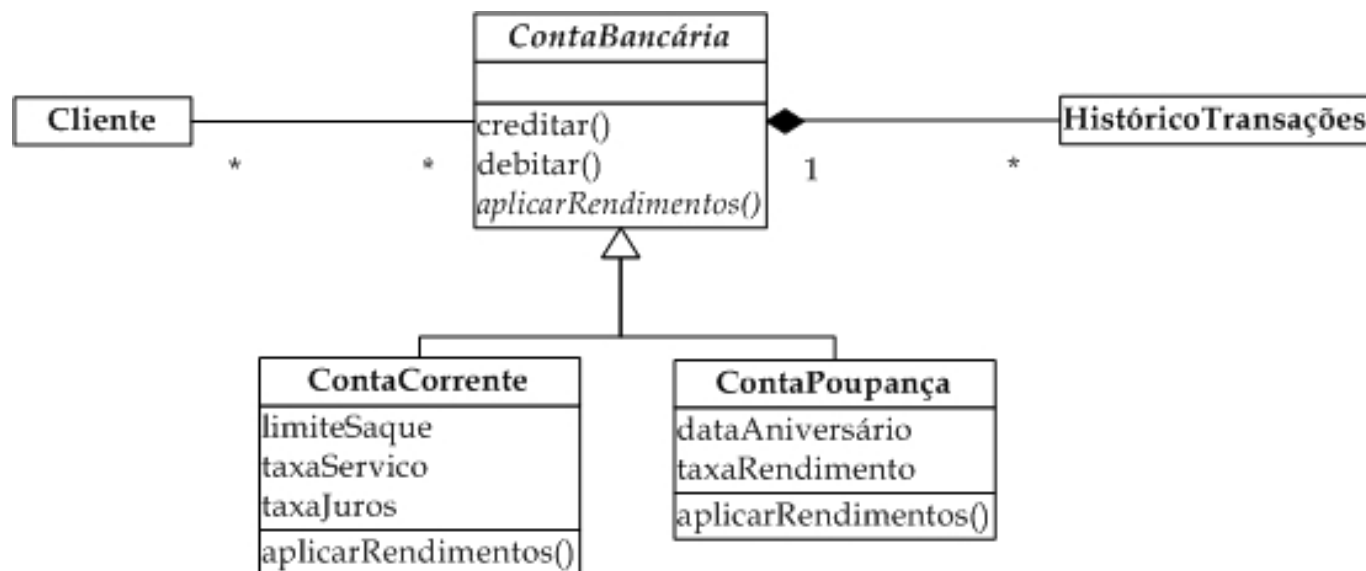


Classes Abstratas

- 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 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.
- Se algum “subconceito” (subconjunto de objetos) atenda a tem algum(ns) das critérios acima, a criação de uma subclasses deve ser considerada.
- 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.

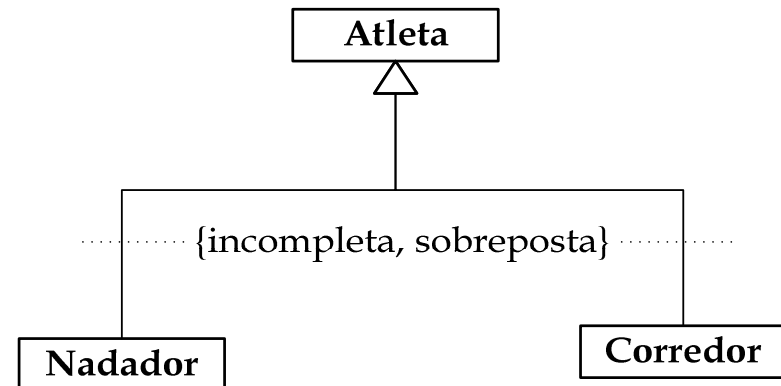
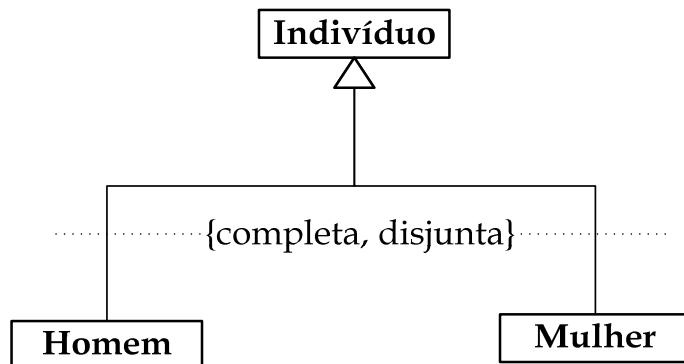
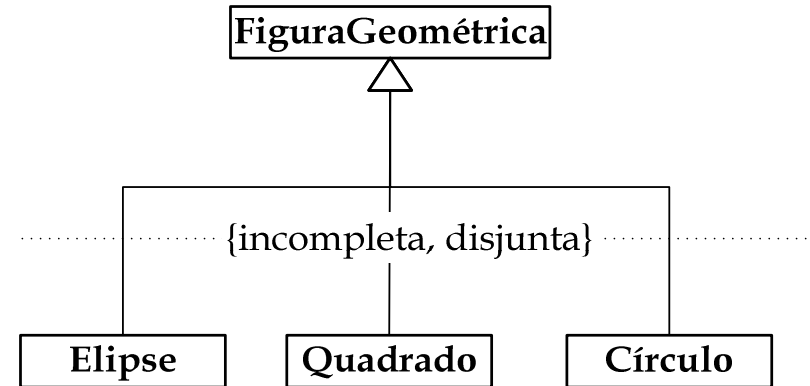
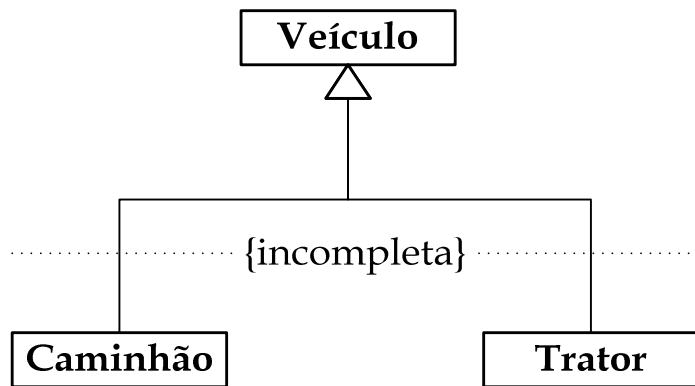


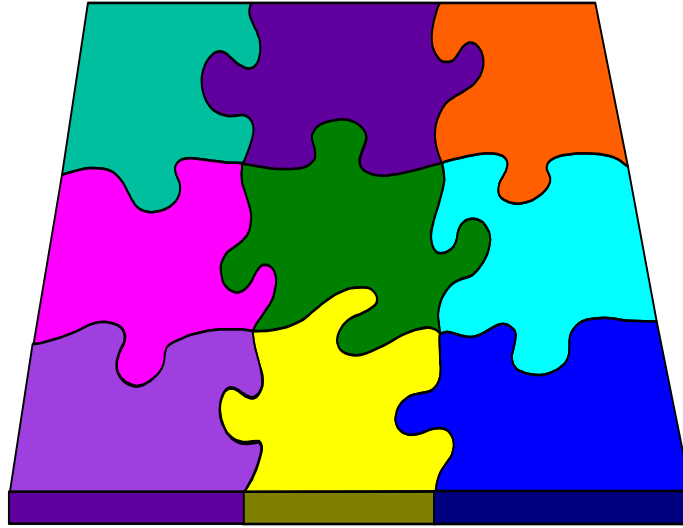
Barbara Liskov (<http://www.pmg.csail.mit.edu/~liskov/>)

Restrições sobre gen/espec

- Restrições OCL sobre relacionamentos de herança podem ser representadas no diagrama de classes, também com o objetivo de esclarecer seu significado.
- Restrições predefinidas pela UML:
 - Sobreposta X Disjunta
 - Completa X Incompleta

Restrições sobre gen/espec





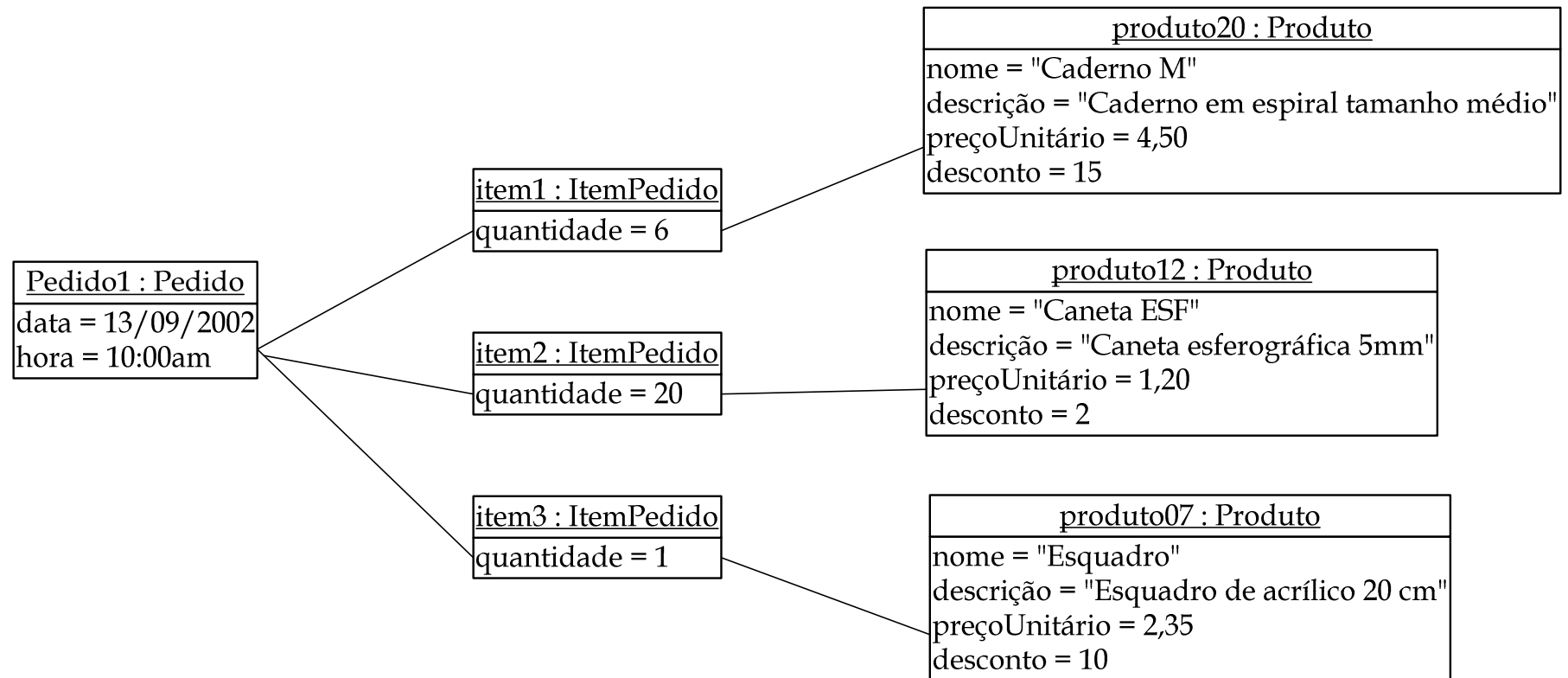
5.3 Diagrama de objetos

Diagrama de objetos

- Além do diagrama de classes, A UML define um segundo tipo de diagrama estrutural, o diagrama de objetos.
- Pode ser visto com uma instância de diagramas de classes
- Representa uma “fotografia” do sistema em um certo momento.
 - exibe as ligações formadas entre objetos conforme estes interagem e os valores dos seus atributos.

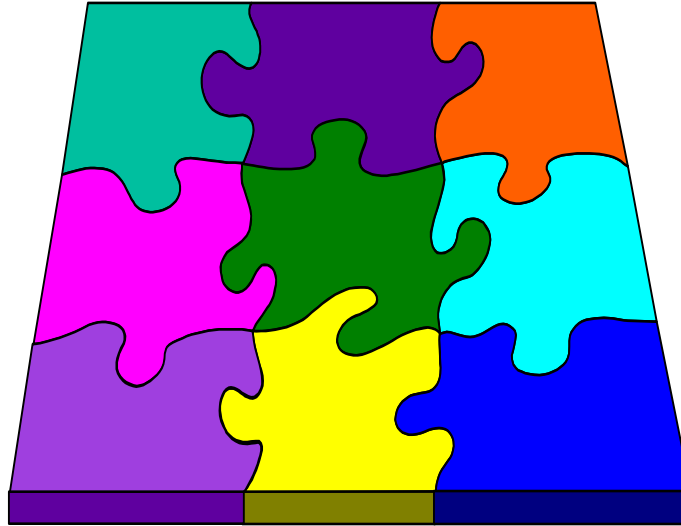
| Formato | Exemplo |
|-------------------------------|-------------------------|
| <u>nomeClasse</u> | <u>Pedido</u> |
| <u>nomeObjeto: NomeClasse</u> | <u>umPedido: Pedido</u> |

Exemplo (Diagrama de objetos)



Exemplo (Diagrama de objetos)





5.4 Técnicas para identificação de classes

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
 - Categorização BCE
 4. Padrões de Análise (Analysys Patterns)
 5. Identificação Dirigida a Responsabilidades

Categorias de Conceitos

- Estratégia: usar uma lista de conceitos comuns.
 - **Conceitos concretos.** Por exemplo, edifícios, carros, salas de aula, etc.
 - **Papéis** desempenhados por seres humanos. Por exemplo, professores, alunos, empregados, clientes, etc.
 - **Eventos**, ou seja, ocorrências em uma data e em uma hora particulares. Por exemplo, reuniões, pedidos, aterrisagens, aulas, etc.
 - **Lugares:** áreas reservadas para pessoas ou coisas. Por exemplo: escritórios, filiais, locais de pouso, salas de aula, etc.
 - **Organizações:** coleções de pessoas ou de recursos. Por exemplo: departamentos, projetos, campanhas, turmas, etc.
 - **Conceitos abstratos:** princípios ou idéias não tangíveis. Por exemplo: reservas, vendas, inscrições, etc.

Análise Textual de Abbott

- Estratégia: identificar termos da narrativa de casos de uso e documento de requisitos que podem sugerir classes, atributos, operações.
- Neste técnica, são utilizadas diversas fontes de informação sobre o sistema: documento e requisitos, modelos do negócio, glossários, conhecimento sobre o domínio, etc.
- Para cada um desses documentos, **os nomes (substantivos e adjetivos) que aparecem no mesmo são destacados.** (São também consideradas locuções equivalentes a substantivos.)
- 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 ao SSOO.
- Abbott também preconiza o uso de sua técnica na identificação de operações e de associações.
 - Para isso, ele sugere que destaquemos 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.)

- A ATA é de aplicação bastante simples.
- No entanto, uma desvantagem é que seu resultado (as classes candidatas identificadas) depende de os documentos utilizados como fonte serem completos.
 - Dependendo do estilo que foi utilizado para escrever esse documento, 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 lingüísticas 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, e é um caso particular da ATA.
- Técnica preconizada pelo Processo Unificado.
- Nesta técnica, o MCU é utilizado como ponto de partida.
 - Premissa: um caso de uso corresponde a um comportamento específico do SSOO. Esse comportamento somente pode ser produzido por objetos que compõem o sistema.
 - Em outras palavras, a realização de um caso de uso é responsabilidade de um conjunto de objetos que devem colaborar para produzir o resultado daquele caso de uso.
 - 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 SSOO são identificadas.
 - Quando todos os casos de uso tiverem sido analisados, todas as classes (ou pelo menos a grande maioria delas) terão sido identificadas.
- Na aplicação deste procedimento, podemos utilizar as **categorização BCE...**

Categorização BCE

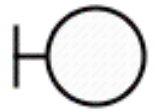
- Na categorização BCE, os objetos de um SSOO são agrupados de acordo com o tipo de responsabilidade a eles atribuída.
 - objetos de **entidade**: usualmente objetos do domínio do problema
 - objetos de **fronteira**: atores interagem com esses objetos
 - objetos de **controle**: servem como intermediários entre objetos de fronteira e de entidade, definindo o comportamento de um caso de uso específico.
- Categorização proposta por Ivar Jacobson em 1992.
 - Possui correspondência (mas não equivalência!) com o framework *model-view-controller* (MVC)
 - Ligação entre análise (o que; problema) e projeto (como; solução)
- Estereótipos na UML: «boundary», «entity», «control»



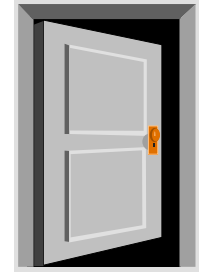
Objetos de Entidade



- 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.



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.
- 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.



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.

Importância da Categorização BCE

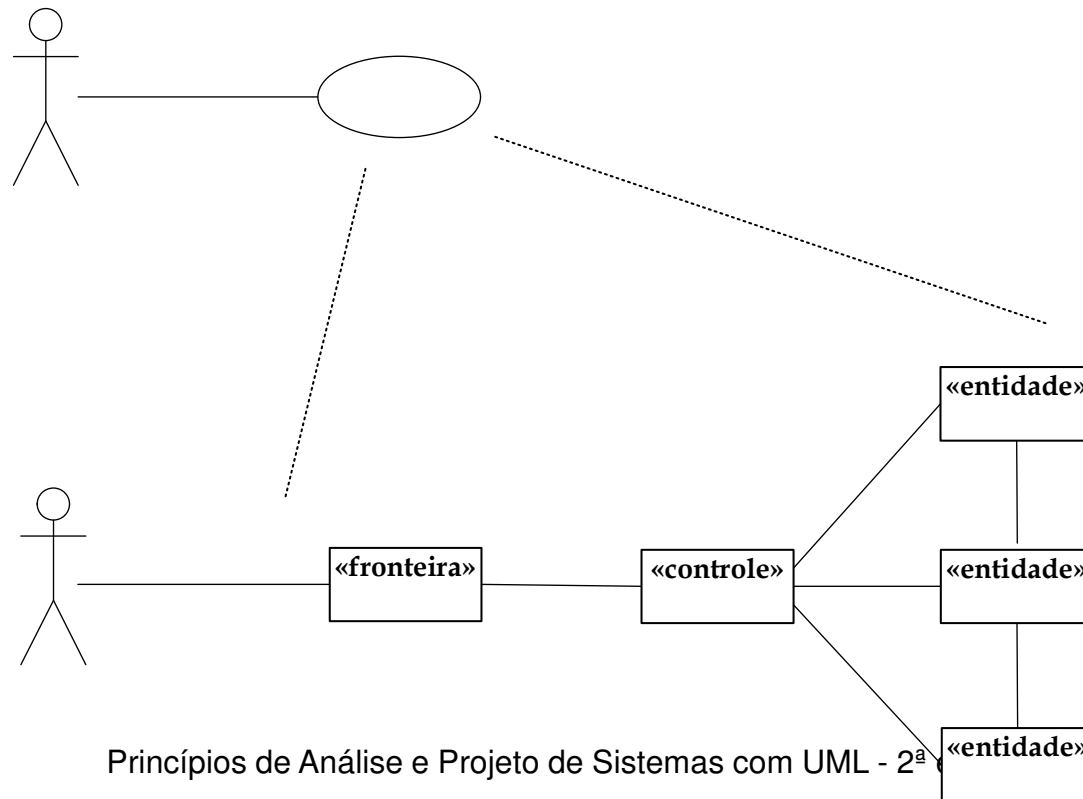
- A categorização BCE parte do princípio de que cada objeto em um SSOO é especialista em realizar um de três tipos de tarefa, a saber:
 - se comunicar com atores (**fronteira**),
 - manter as informações (**entidade**) ou
 - coordenar a realização de um caso de uso (**controle**).
- A categorização BCE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

Visões de Classes Participantes

- Uma **Visão de Classes Participantes** (VCP) é um diagrama das classes cujos objetos participam da realização de determinado caso de uso.
 - É uma recomendação do UP (Unified Process). UP: “definir uma VCP por caso de uso”
 - Termo original: *View Of Participating Classes* (VOPC).
- Em uma VCP, são representados objetos de fronteira, de entidade e de controle para um caso de uso particular.
- Uma VCP é definida através da utilização da categorização BCE previamente descrita...vide próximo slide.

Estrutura de uma VCP

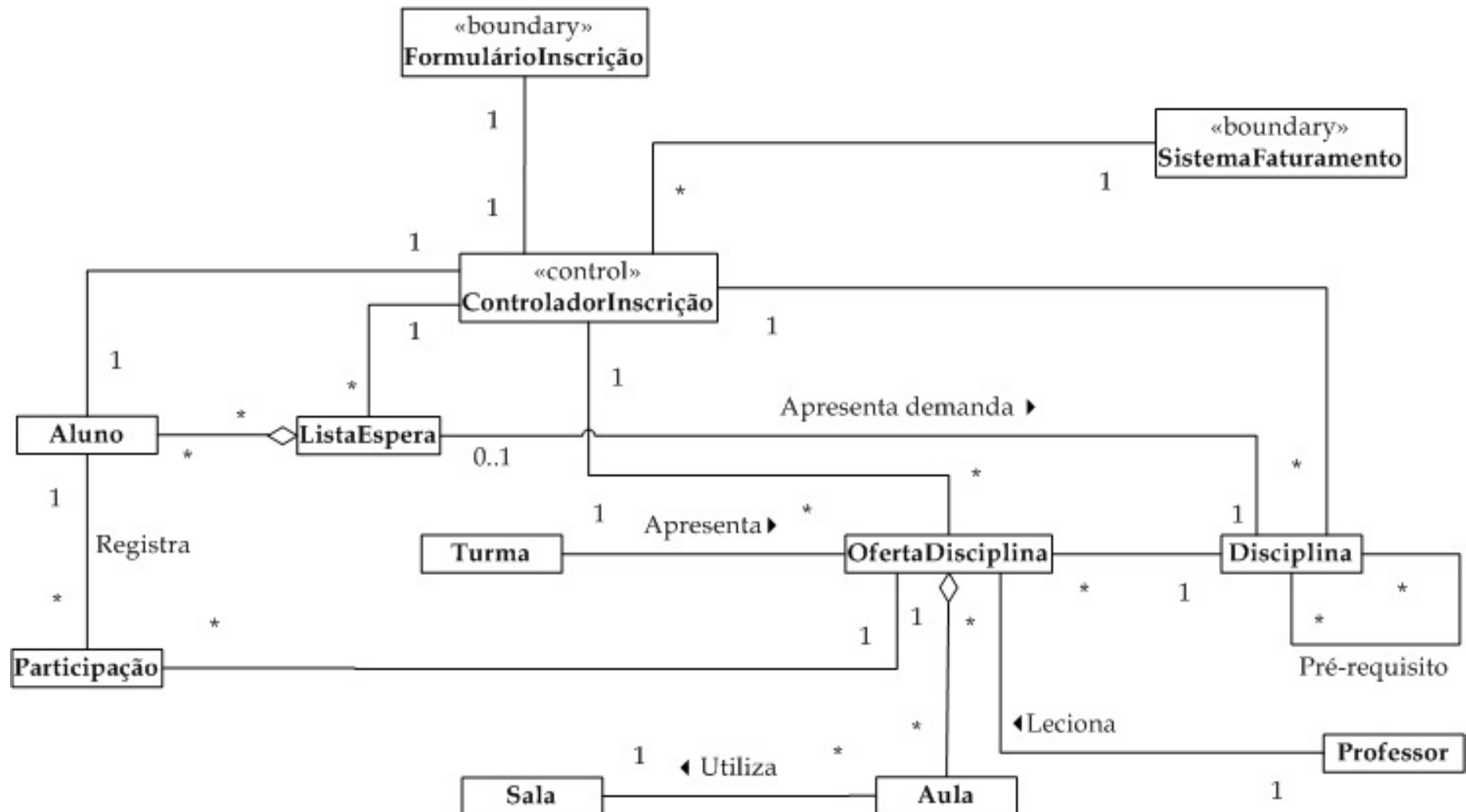
- Uma VCP representa a estrutura das classes que participam da realização de um caso de uso em particular.



Construção de uma VCP

- Para cada caso de uso:
 - Adicione uma fronteira para cada elemento de interface gráfica principal, tais com uma tela (formulário) ou relatório.
 - Adicione uma fronteira para cada ator não-humano (por exemplo, outro sistema).
 - Adicione um ou mais controladores para gerenciar o processo de realização do caso de uso.
 - Adicione uma entidade para cada conceito do negócio.
 - Esses objetos são originários do modelo conceitual.
- Os estereótipos gráficos definidos pela UML podem ser utilizados.

VCP (exemplo) – Realizar Inscrição



Regras Estruturais em uma VCP

- Durante a fase de análise, use as regras a seguir para definir a VCP para um caso de uso.
 - Atores somente podem interagir com objetos de fronteira.
 - Objetos de fronteira somente podem interagir com controladores e atores.
 - Objetos de entidade somente podem interagir (receber requisições) com controladores.
 - Controladores somente podem interagir com objetos de fronteira e objetos de entidade, e com (eventuais) outros controladores.

Padrões de Análise

- Após produzir diversos modelos para um mesmo domínio, é natural que um modelador comece a identificar características comuns entre esses modelos.
- Em particular, um mesmo conjunto de classes ou colaborações entre objetos costuma recorrer, com algumas pequenas diferenças, em todos os sistemas desenvolvidos para o domínio em questão.
 - Quantos modelos de classes já foram construídos que possuem os conceitos Cliente, Produto, Fornecedor, Departamento, etc?
 - Quantos outros já foram construídos que possuíam os conceitos Ordem de Compra, Ordem de Venda, Orçamento, Contrato, etc?

Padrões de Análise

- A identificação de características comuns acontece em consequência do ganho de experiência do modelador em determinado domínio de problema.
- Ao reconhecer processos e estruturas comuns em um domínio, o modelador pode descrever (catalogar) a essência dos mesmos, dando origem a ***padrões de software***.
- Quando o problema correspondente ao descrito em um padrão de software acontecer novamente, um modelador pode utilizar a solução descrita no catálogo.
- A aplicação desse processo permite que o desenvolvimento de determinado aspecto de um SSOO seja feito de forma mais rápida e menos suscetível a erros.

Padrões de Análise

- Um padrão de software pode então ser definido como uma descrição essencial de um problema recorrente no desenvolvimento de software.

Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice

<http://www.patternlanguage.com/leveltwo/ca.htm>



“Cada padrão descreve um problema que ocorre frequentemente no nosso ambiente, e então descreve o núcleo de uma solução para tal problema. Esse núcleo pode ser utilizado um milhão de vezes, sem que haja duas formas de utilização iguais.”

Christopher Alexander

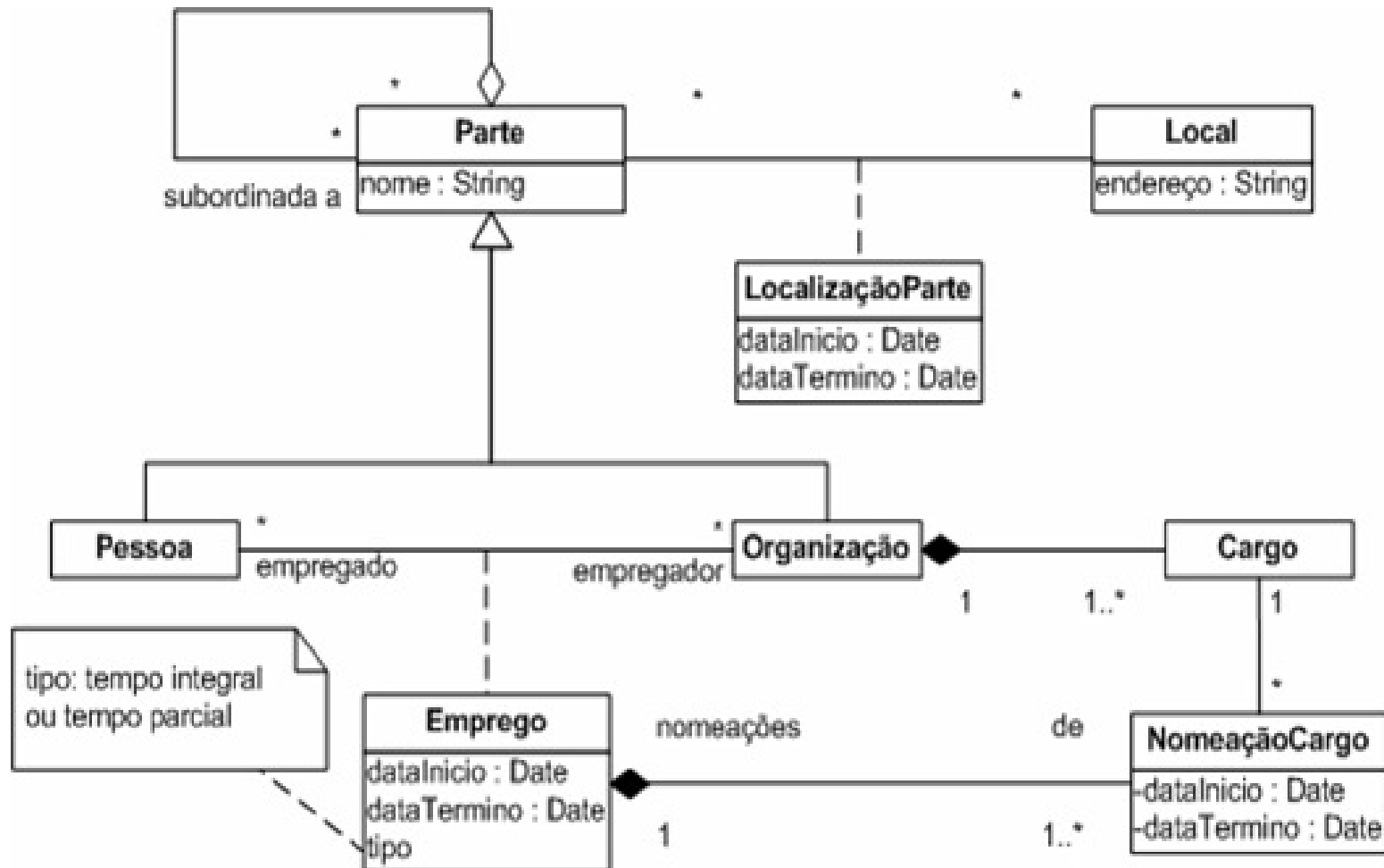
Padrões de Análise

- Padrões de software podem ser utilizados em diversas atividades e existem em diversos níveis de abstração.
 - *Padrões de Análise* (Analysis Patterns)
 - *Padrões de Projeto* (Design Patterns)
 - *Padrões Arquiteturais* (Architectural Patterns)
 - *Idiomas* (Idioms)
 - *Anti-padrões* (Anti-patterns)

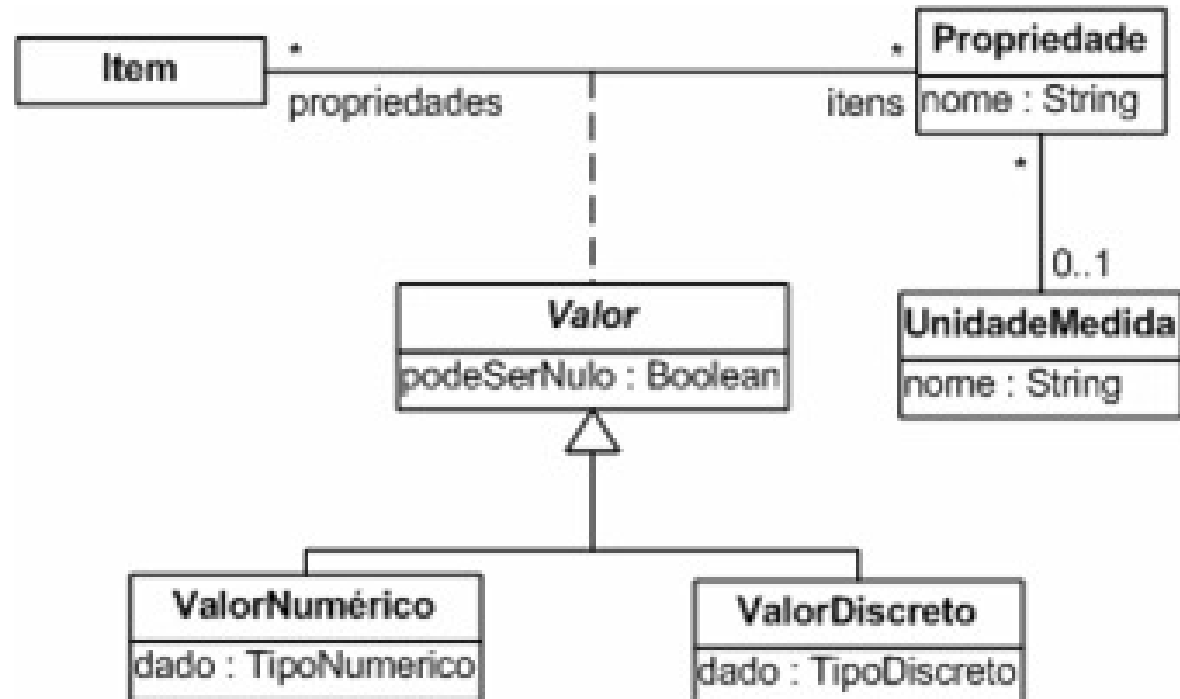
Padrões de Análise

- Um padrão de software pode então ser definido como uma descrição essencial de um problema recorrente no desenvolvimento de software.
- Padrões de software vêm sendo estudados por anos e são atualmente utilizados em diversas fases do desenvolvimento.
- Padrões de software utilizados na fase de análise de um SSOO são chamados *padrões de análise*.
- Um padrão de análise normalmente é composto, dentre outras partes, de um fragmento de diagrama de classes que pode ser customizado para uma situação de modelagem em particular.

Exemplo 1 – Padrão Party



Exemplo 2 – Padrão Metamodel



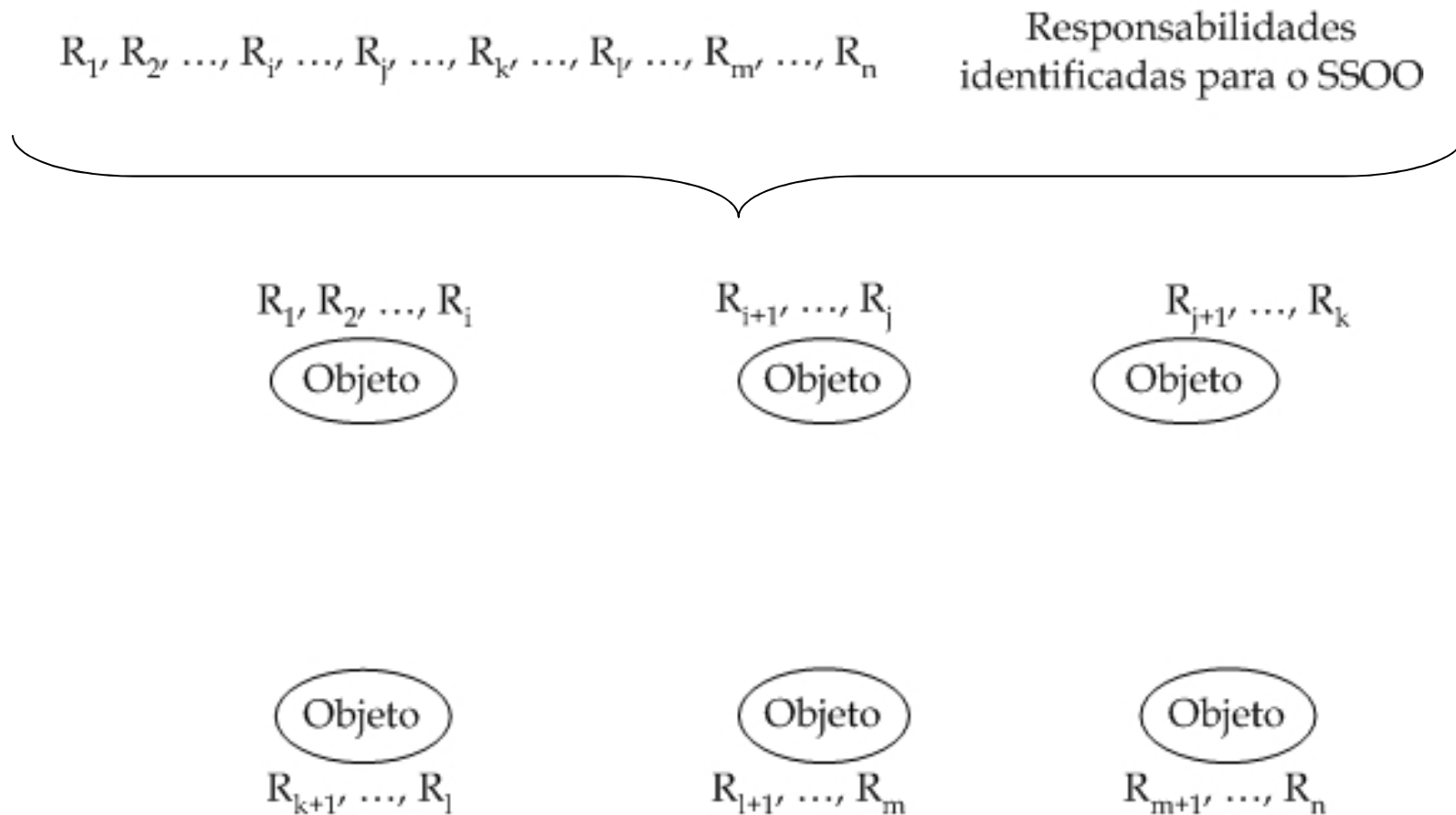
Identificação Dirigida a Responsabilidades

- Nesta técnica, a ênfase está na identificação de classes a partir de seus comportamentos relevantes para o sistema.
 - O esforço do modelador recai sobre a identificação das *responsabilidades* que cada classe deve ter dentro do sistema.
- Método foi proposto por Rebecca Wirfs-Brock et al (*RDD*).
 - “*O método dirigido a responsabilidades enfatiza o encapsulamento da estrutura e do comportamento dos objetos.*”
- Essa técnica enfatiza o **princípio do encapsulamento**:
 - A ênfase está na identificação das responsabilidades de uma classe que são úteis externamente à mesma.

Os detalhes internos à classe (*como* ela faz para cumprir com suas responsabilidades) devem ser abstraídos.



As responsabilidades de um SSOO devem ser alocadas aos objetos (classes) componentes do mesmo.



Responsabilidades de uma Classe

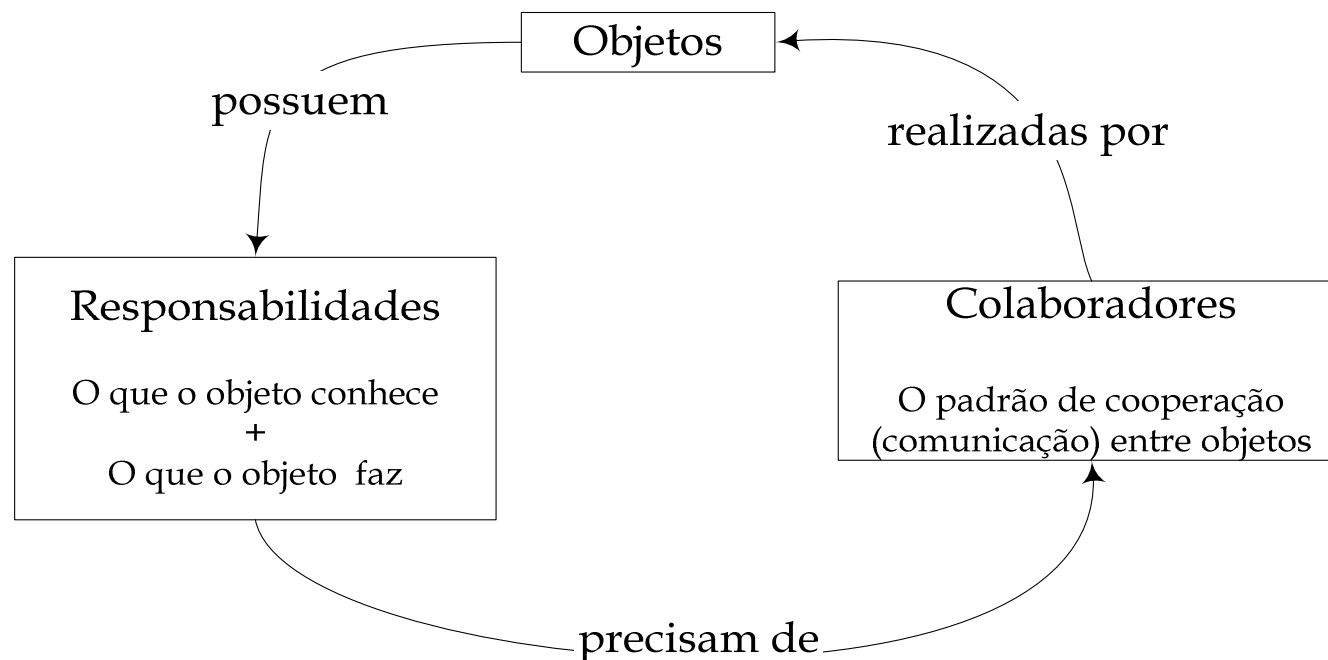
- Em um SSOO, objetos encapsulam comportamento.
 - O comportamento de um objeto é definido de tal forma que ele possa cumprir com suas *responsabilidades*.
- Uma responsabilidade **é uma obrigação que um objeto tem para com o sistema no qual ele está inserido**.
 - Através delas, um objeto colabora (ajuda) com outros para que os objetivos do sistema sejam alcançados.
- Na prática, uma responsabilidade é alguma coisa que um objeto **conhece** ou **sabe fazer** (sozinho ou “pedindo ajuda”).
- Se um objeto tem uma responsabilidade com a qual não pode cumprir sozinho, ele deve requisitar **colaborações** de outros objetos.

Responsabilidades e Colaboradores

- Exemplo: considere clientes e seus pedidos:
 - Um objeto Cliente *conhece* seu nome, seu endereço, seu telefone, etc.
 - Um objeto Pedido *conhece* sua data de realização, *conhece* o seu cliente, *conhece* os seus itens componentes e *sabe fazer* o cálculo do seu total.
- Exemplo: quando a impressão de uma fatura é requisitada em um sistema de vendas, vários objetos precisam colaborar:
 - um objeto Pedido pode ter a responsabilidade de fornecer o seu valor total
 - um objeto Cliente fornece seu nome
 - cada ItemPedido informa a quantidade correspondente e o valor de seu subtotal
 - os objetos Produto também colaboraram fornecendo seu nome e preço unitário.

Responsabilidades e Colaborações

Um objeto cumpre com suas responsabilidades através das informações que ele possui ou das informações que ele pode derivar a partir de colaborações com outros objetos.



Pense em um SSOO como uma sociedade onde os cidadãos (colaboradores) são objetos.

Modelagem CRC

- A identificação dirigida a responsabilidades normalmente utiliza uma técnica de modelagem que facilita a participação de especialistas do domínio e analistas.
- Essa técnica é denominada *modelagem CRC* (CRC modeling).
- CRC: Class, Responsibility, Collaboration
- A modelagem CRC foi proposta em 1989 por Kent Beck e Ward Cunningham.
 - A princípio, essa técnica de modelagem foi proposta como uma forma de ensinar o paradigma OO a iniciantes.
 - Contudo, sua simplicidade de notação a tornou particularmente interessante para ser utilizada na identificação de classes.

Sessão CRC

- Para aplicar essa técnica, esses profissionais se reúnem em uma sala, onde tem início uma *sessão CRC*.
- Uma sessão CRC é uma reunião que envolve cerca de meia dúzia de pessoas.
- Entre os participantes estão especialistas de domínio, projetistas, analistas e o moderador da sessão.
- A cada pessoa é entregue um cartão de papel que mede aproximadamente 10cm x 15cm.
- Uma vez distribuídos os cartões pelos participantes, um conjunto de cenários de determinado caso de uso é selecionado.

Sessão CRC

- Então, para cada cenário desse conjunto, uma sessão CRC é iniciada.
 - (Se o caso de uso não for tão complexo, ele pode ser analisado em uma única sessão.)
- Um cartão CRC é dividido em várias partes.
 - Na parte superior do cartão, aparece o nome de uma classe.
 - A parte inferior do cartão é dividida em duas colunas.
 - Na coluna da esquerda, o indivíduo ao qual foi entregue o cartão deve listar as responsabilidades da classe.
 - Na coluna direita, o indivíduo deve listar os nomes de outras classes que colaboram com a classe em questão para que ela cumpra com suas responsabilidades.

Modelagem CRC

- Normalmente já existem algumas classes candidatas para determinado cenário.
 - Identificadas através de outras técnicas.
- A sessão CRC começa com algum dos participantes simulando o ator primário que dispara a realização do caso de uso.
- Na medida em que esse participante simula a interação do ator com o sistema, os demais participantes encenam a colaboração entre objetos que ocorre internamente ao sistema.
- Através dessa encenação dos participantes da sessão CRC, as classes, responsabilidades e colaborações são identificadas.

Estrutura de um Cartão CRC

| Nome da classe | |
|---|--|
| Responsabilidades | Colaboradores |
| 1ª responsabilidade 2ª responsabilidade ... n-ésima responsabilidade | 1º colaborador 2º colaborador ... m-ésimo colaborador |

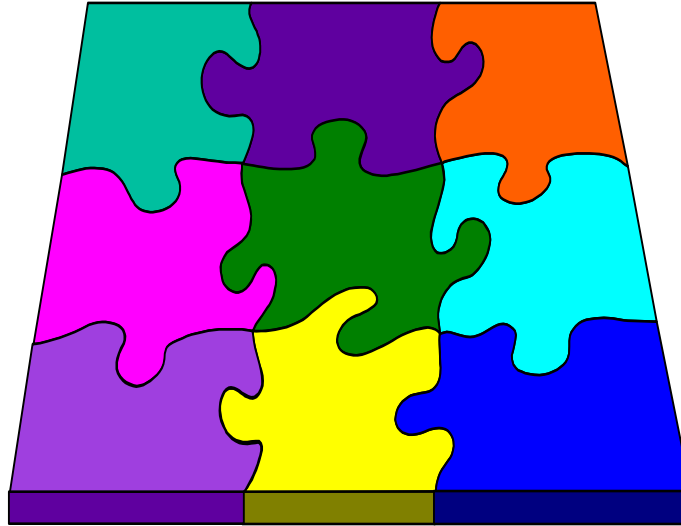
Exemplos de cartões CRC

| Aluno | |
|--|---------------|
| Responsabilidades | Colaboradores |
| 1. Conhecer seu número de registro 2. Conhecer seu nome 3. Conhecer as disciplinas que já cursou | Participação |

| Disciplina | |
|---|---------------|
| Responsabilidades | Colaboradores |
| 1. Conhecer seus pré-requisitos 2. Conhecer seu código 3. Conhecer seu nome 4. Conhecer sua quantidade de créditos | Disciplina |

Criação de Novos Cartões

- Durante uma sessão CRC, para cada responsabilidade atribuída a uma classe, o seu proprietário deve questionar se tal classe é capaz de cumprir com a responsabilidade sozinha.
- Se ela precisar de ajuda, essa ajuda é dada por um colaborador.
- Os participantes da sessão, então, decidem que outra classe pode fornecer tal ajuda.
 - Se essa classe existir, ela recebe uma nova responsabilidade necessária para que ela forneça ajuda.
 - Caso contrário, um novo cartão (ou seja, uma nova classe) é criado para cumprir com tal responsabilidade de ajuda.



5.5 Construção do modelo de classes

Construção do modelo de classes

- Após a identificação de classes, o modelador deve verificar a consistência entre as classes para eliminar incoerências e redundâncias.
 - Como dica, o modelador deve estar apto a declarar as razões de existência de cada classe identificada.
- Depois disso, os analistas devem começar a definir o mapeamento das responsabilidades e colaboradores de cada classe para os elementos do diagrama de classes.
 - Esse mapeamento resulta em um diagrama de classes que apresenta uma estrutura estática relativa a todas as classes identificadas como participantes da realização de um ou mais casos de uso.

Definição de propriedades

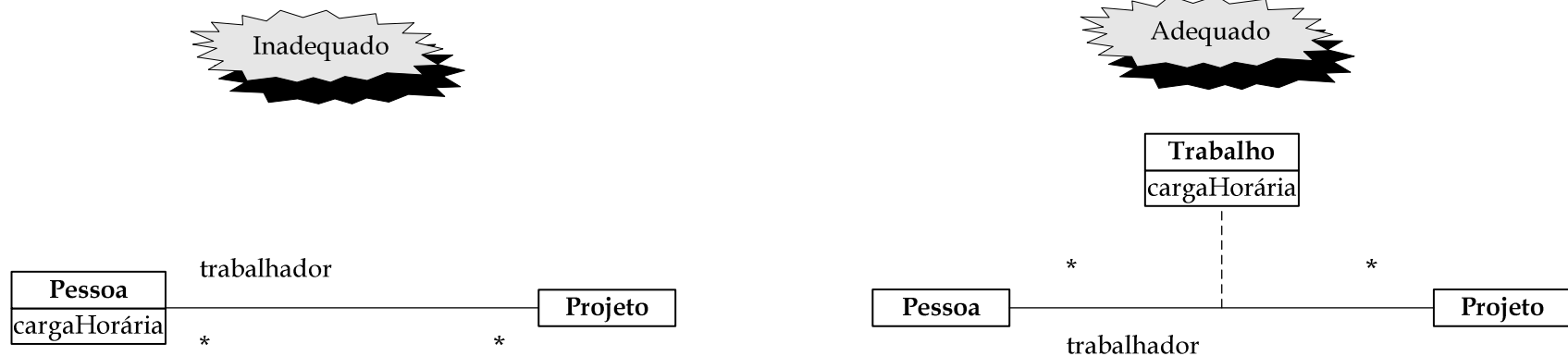
- Uma responsabilidade de conhecer é mapeada para um ou mais atributos.
- Operações de uma classe são um modo mais detalhado de explicitar as responsabilidades de fazer.
 - Uma operação pode ser vista como uma contribuição da classe para uma tarefa mais complexa representada por um caso de uso.
 - Uma definição mais completa das operações de uma classe só pode ser feita após a construção dos **diagramas de interação**.

Definição de associações

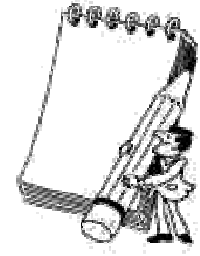
- O fato de uma classe possuir colaboradores indica que devem existir relacionamentos entre estes últimos e a classe.
 - Isto porque um objeto precisa conhecer o outro para poder lhe fazer requisições.
 - Portanto, para criar associações, verifique os colaboradores de uma classe.
- O raciocínio para definir associações reflexivas, ternárias e agregações é o mesmo.

Definição de classes associativas

- Surgem a partir de responsabilidades de conhecer que o modelador não conseguiu atribuir a alguma classe.
 - (mais raramente, de responsabilidades *de fazer*)



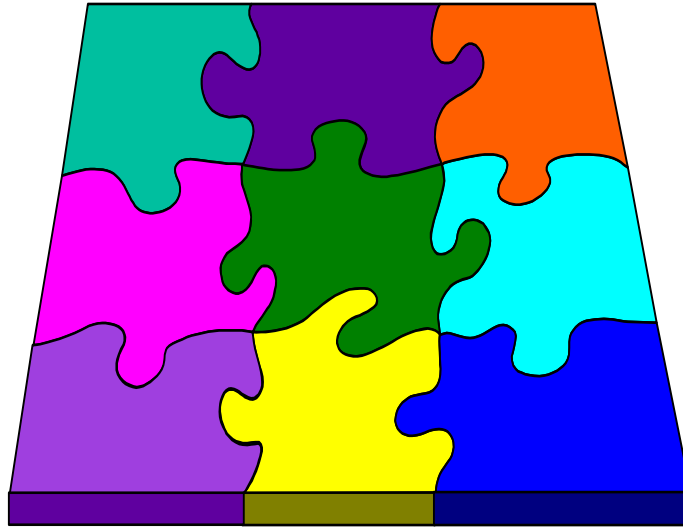
Organização da documentação



- As responsabilidades e colaboradores mapeados para elementos do modelo de classes devem ser organizados em um diagrama de classes e documentados, resultando no *modelo de classes de domínio*.
- Podem ser associados estereótipos predefinidos às classes identificadas:
 - <<fronteira>>
 - <<entidade>>
 - <<controle>>

Organização da documentação

- A construção de um único diagrama de classes para todo o sistema pode resultar em um diagrama bastante complexo. Um alternativa é criar uma *visão de classes participantes* (VCP) para cada caso de uso.
- Em uma VCP, são exibidos os objetos que participam de um caso de uso.
- As VCPs podem ser reunidas para formar um único diagrama de classes para o sistema como um todo.



5.6 Modelo de classes no processo de desenvolvimento

Modelo de classes no processo de desenvolvimento

- Em um desenvolvimento dirigido a casos de uso, após a descrição dos casos de uso, é possível iniciar a identificação de classes.
- As classes identificadas são refinadas para retirar inconsistências e redundâncias.
- As classes são documentadas e o diagrama de classes inicial é construído, resultando no modelo de classes de domínio.

Modelo de classes no processo de desenvolvimento

- Inconsistências nos modelos devem ser verificadas e corrigidas.
- As construções do modelo de casos de uso e do modelo de classes são retroativas uma sobre a outra.
 - Durante a aplicação de alguma técnica de identificação, novos casos de uso podem ser identificados
 - Pode-se identificar a necessidade de modificação de casos de uso preexistentes.
- Depois que a primeira versão do modelo de classes de análise está completa, o modelador deve retornar ao modelo de casos de uso e verificar a consistência entre os dois modelos.

Modelo de classes no processo de desenvolvimento

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

