

PRINCÍPIOS DE ANÁLISE E PROJETO DE SISTEMAS COM UML

2^a EDIÇÃO

Diagrama de Classes de Análise

- **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

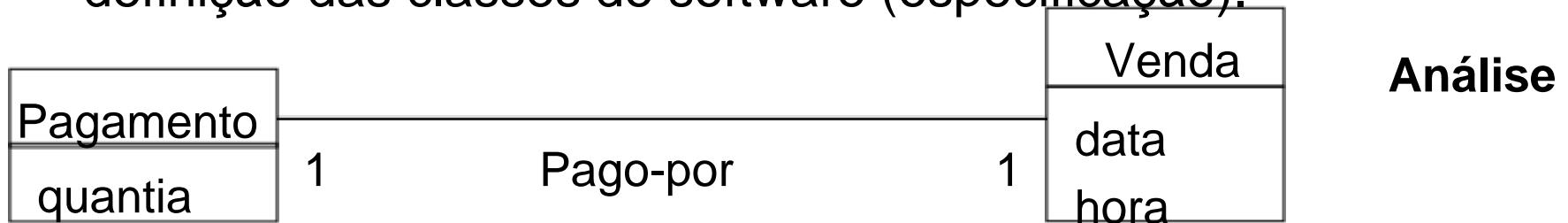
- O objetivo da modelagem de classes de análise é prover respostas para as seguintes perguntas:
 - Por definição um sistema OO é composto de objetos...em um nível alto de abstração, que objetos constituem o sistema em questão?
 - Quais são as classes candidatas?
 - Como as classes do sistema estão relacionadas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

MODELO DE CLASSES DE ANÁLISE

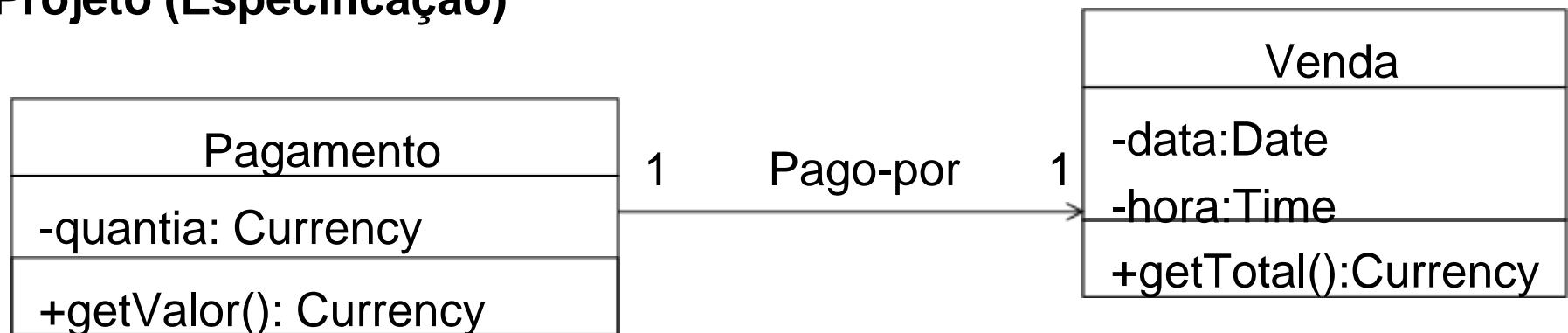
- Representa termos do domínio do negócio.
 - idéias, coisas, e conceitos no mundo real.
- Objetivo: descrever o problema representado pelo sistema a ser desenvolvido, sem considerar características da solução a ser utilizada.
- É um dicionário “visual” de conceitos e informações relevantes ao sistema 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).

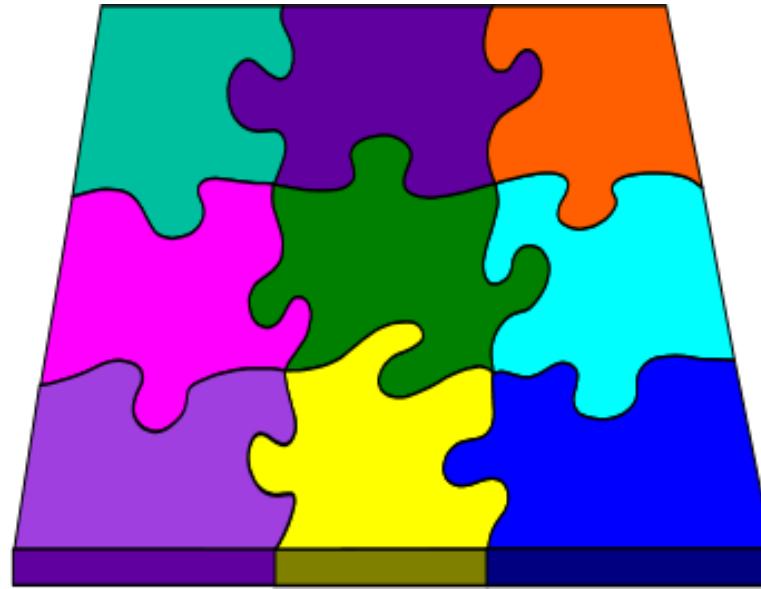
MODELO DE ANÁLISE: FOCO NO PROBLEMA

- 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	$I_i \dots I_s$

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

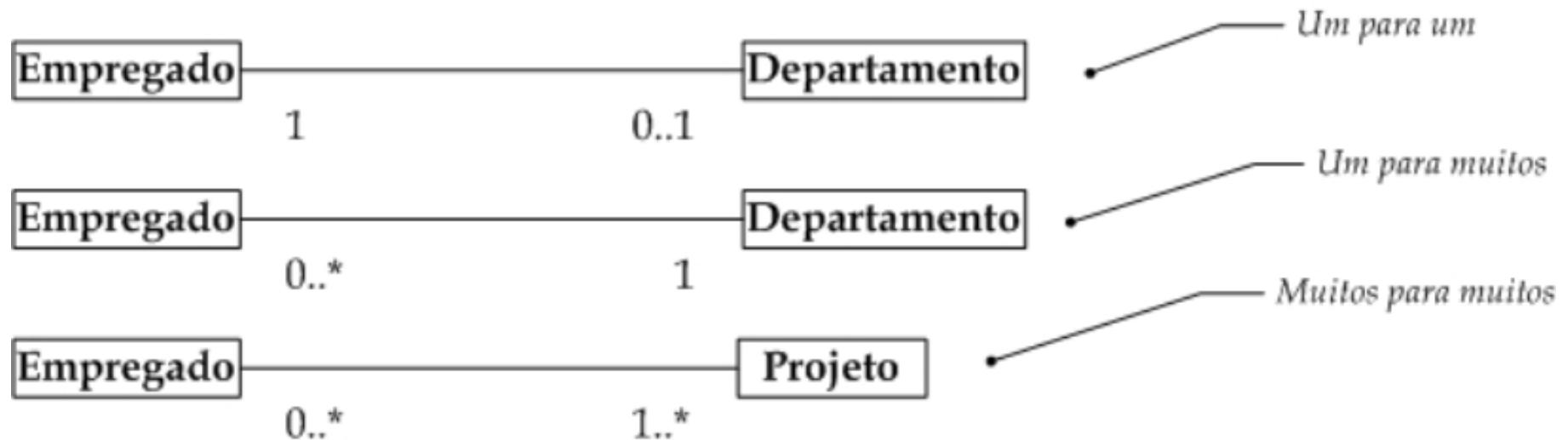
- Uma corrida está associada a, no mínimo, dois velocistas
- Uma corrida está associada a, no máximo, seis velocistas.
- Um velocista pode estar associado a várias corridas.

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	0..1
	1	1
Um para muitos	0..1	*
	1	1..*
		0..*
Muitos para muitos	*	*
	1..*	1..*
	0..*	0..*

EXEMPLO (CONECTIVIDADE)

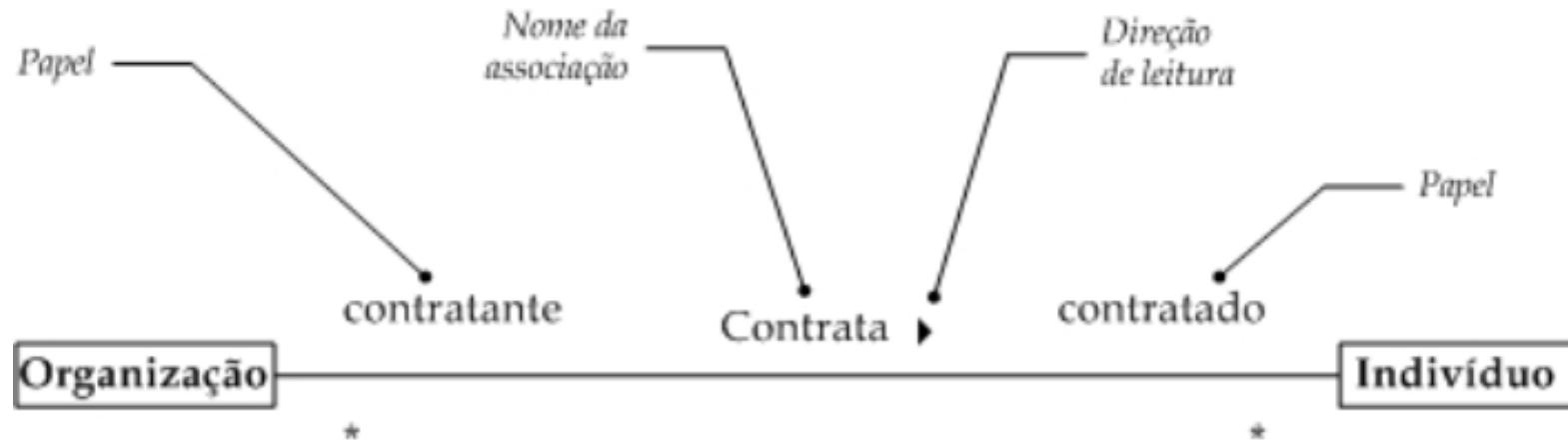


PARTICIPAÇÃO

- 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.

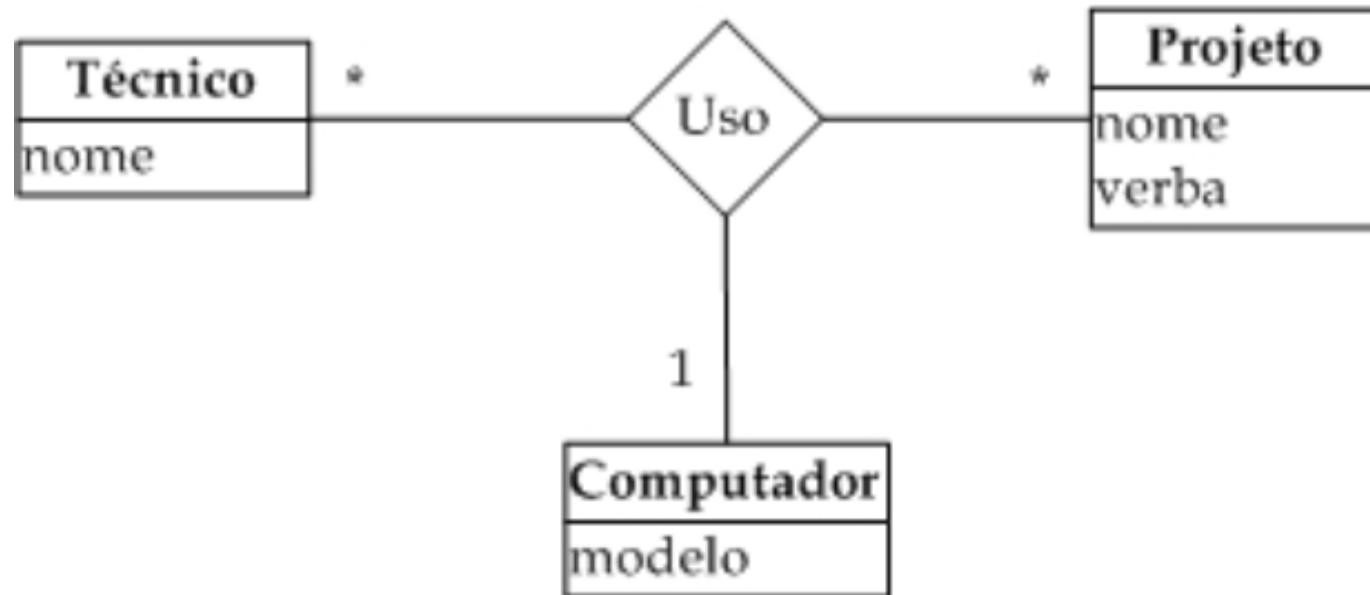


ASSOCIAÇÕES N-ÁRIAS

- 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 ($n = 3$).

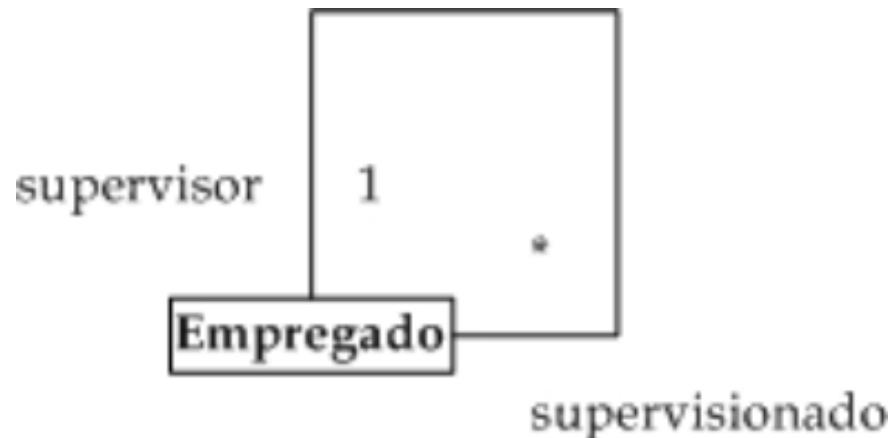
EXEMPLO (ASSOCIAÇÃO TERNÁRIA)

- 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.



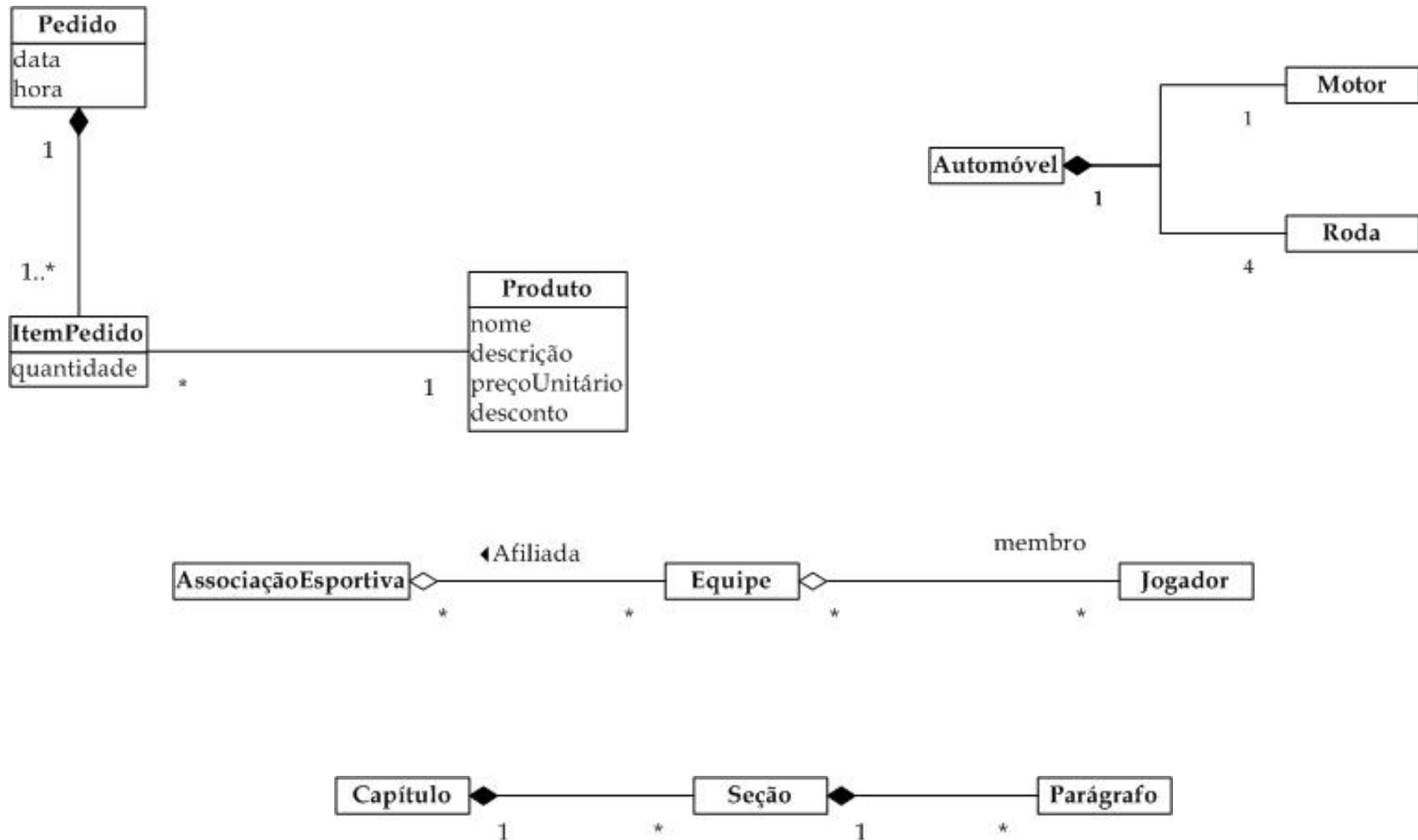
AGREGAÇÕES E COMPOSIÇÕES

- 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.
- Destrução 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.

GENERALIZAÇÕES E ESPECIALIZAÇÕES

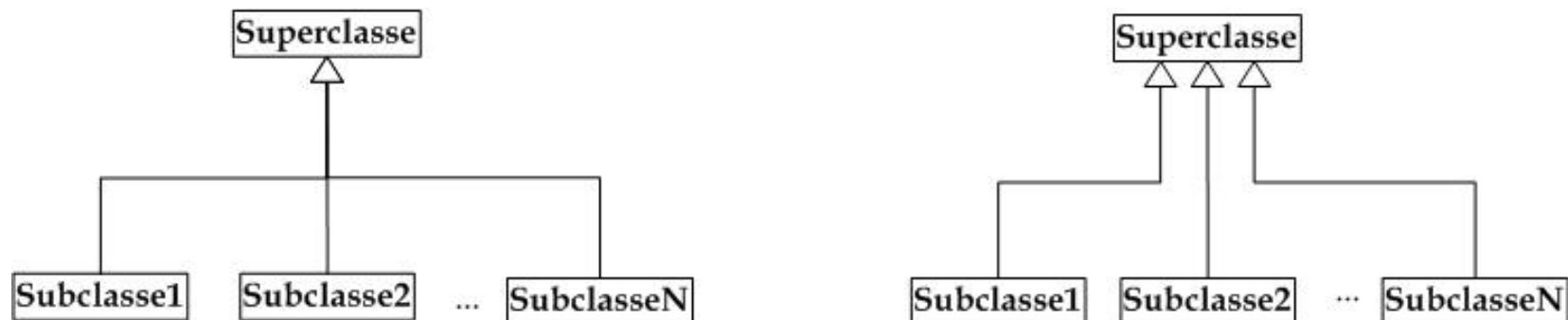
- O modelador também pode representar relacionamentos entre classes.
 - Esse denotam relações de generalidade ou especificidade entre as classes envolvidas.
 - Exemplo: o conceito *mamífero* é mais genérico que o conceito *ser humano*.
 - Exemplo: o conceito *carro* é mais específico que o conceito *veículo*.
- 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

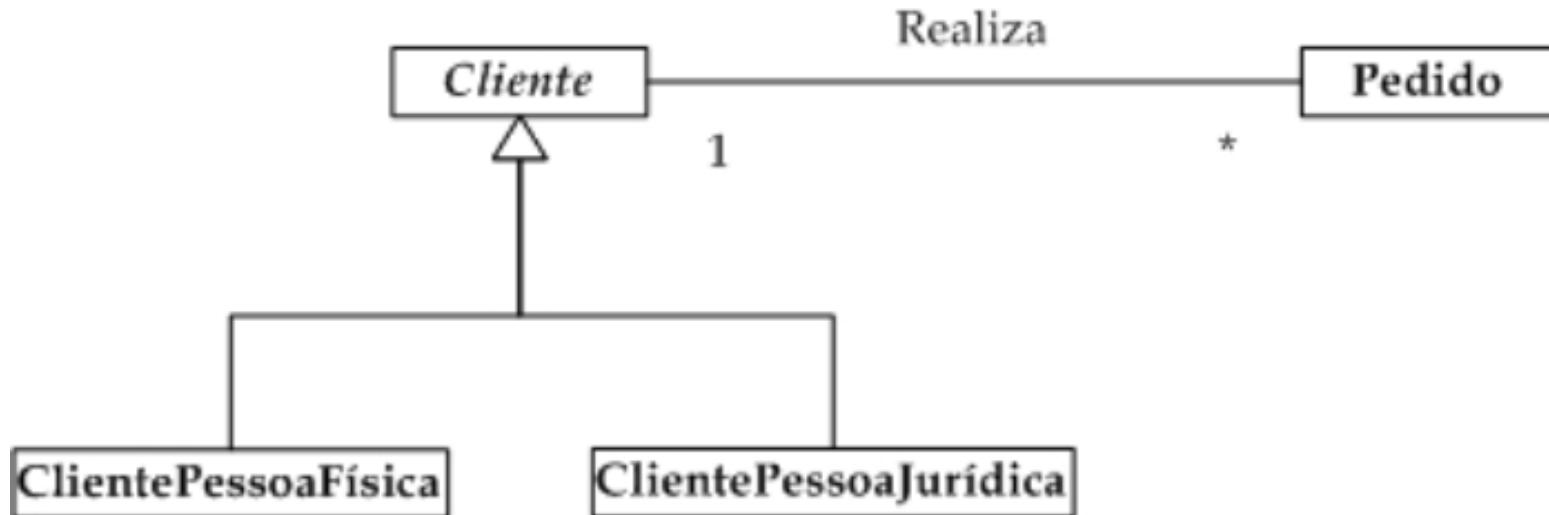


SEMÂNTICA DA HERANÇA

- 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

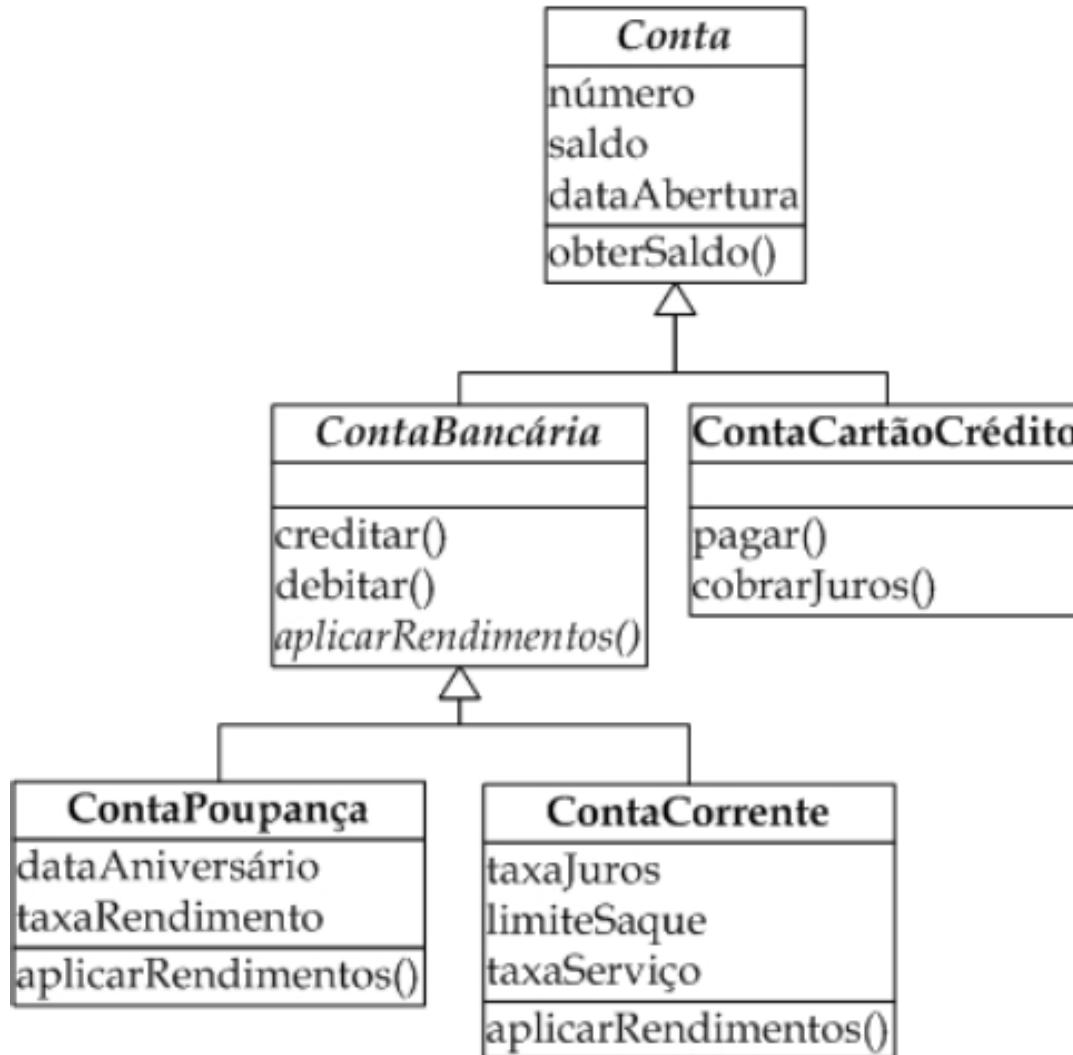
- Não somente atributos e operações, mas também associações são herdadas pelas subclasses.
- No exemplo abaixo, cada subclass está associada a Pedido, por herança.



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

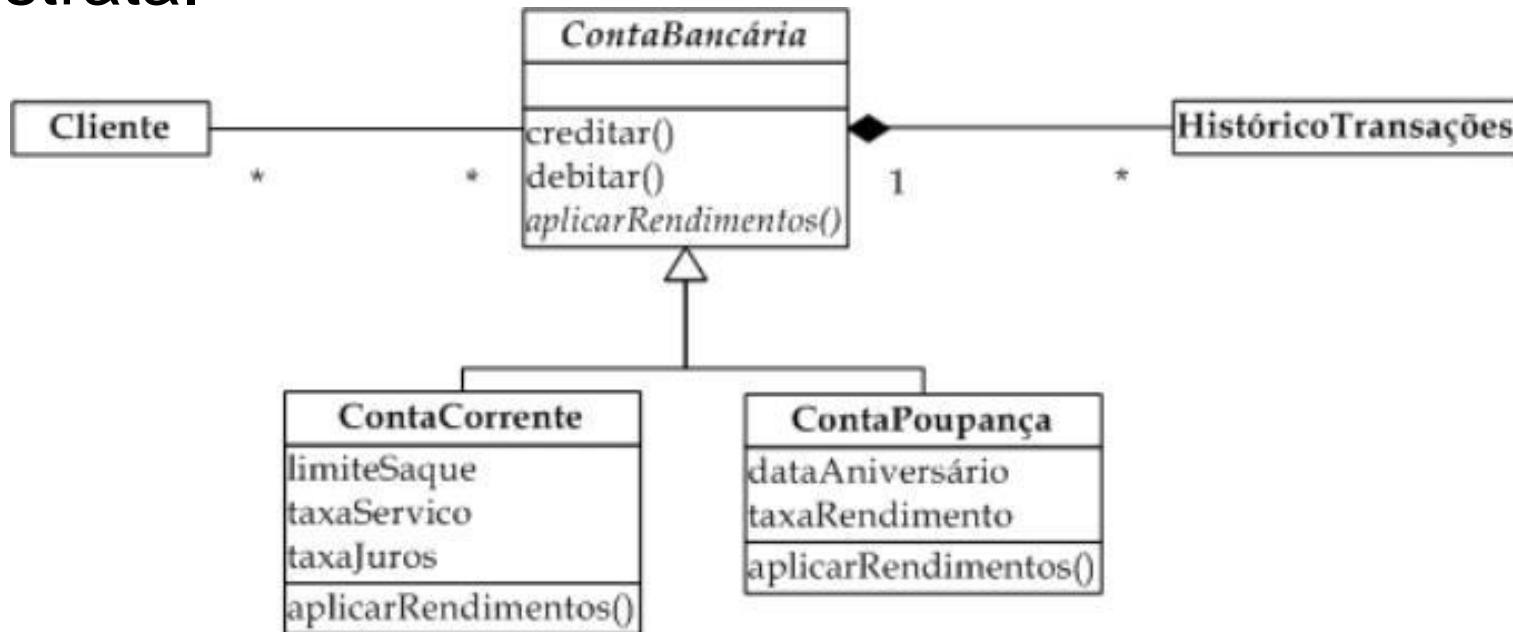


CLASSE ABSTRATAS

- 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 CLASSESS 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

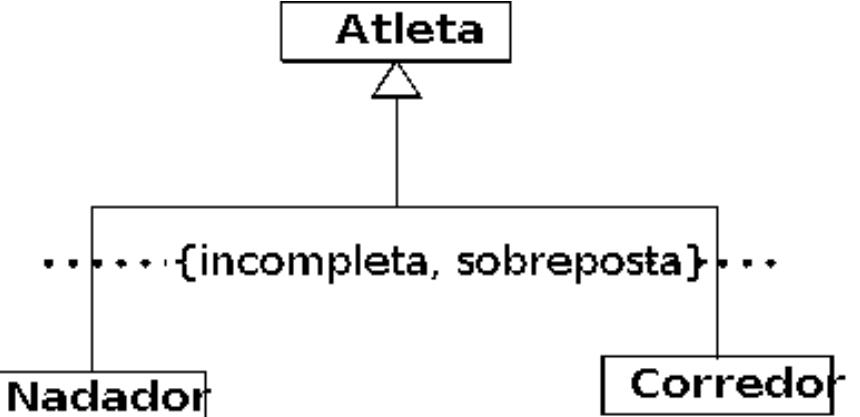
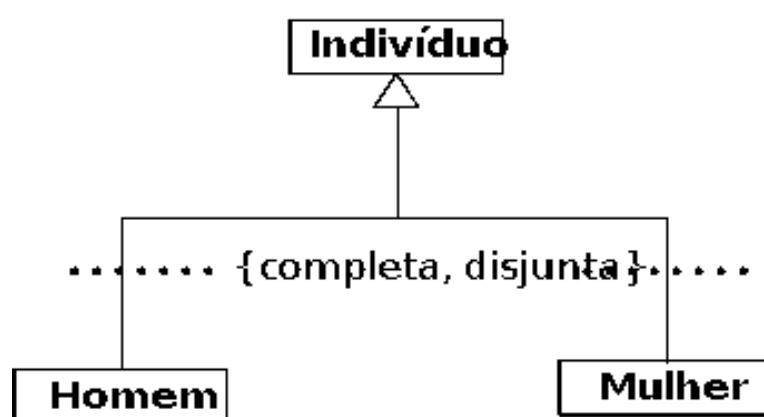
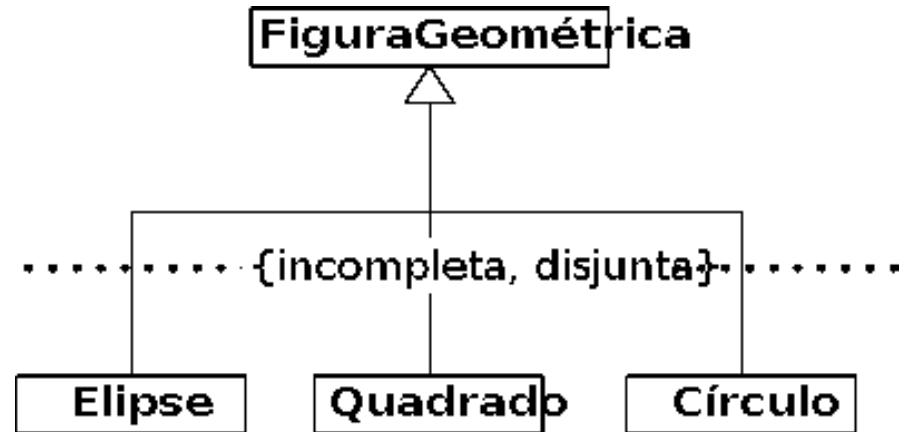
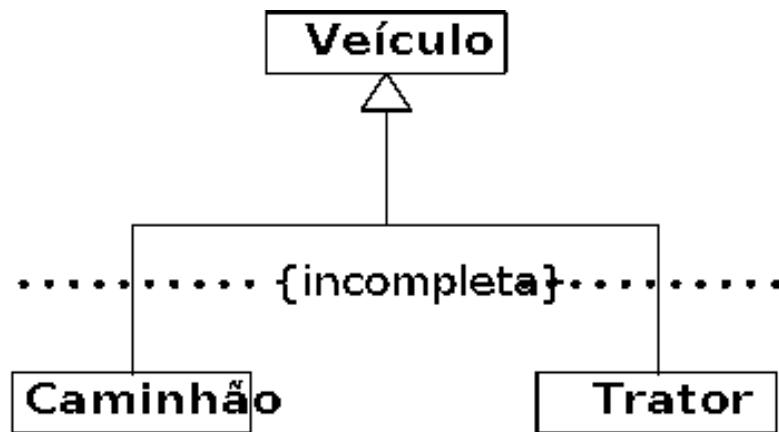


Diagrama de Classes – Exemplo Carona Solidária

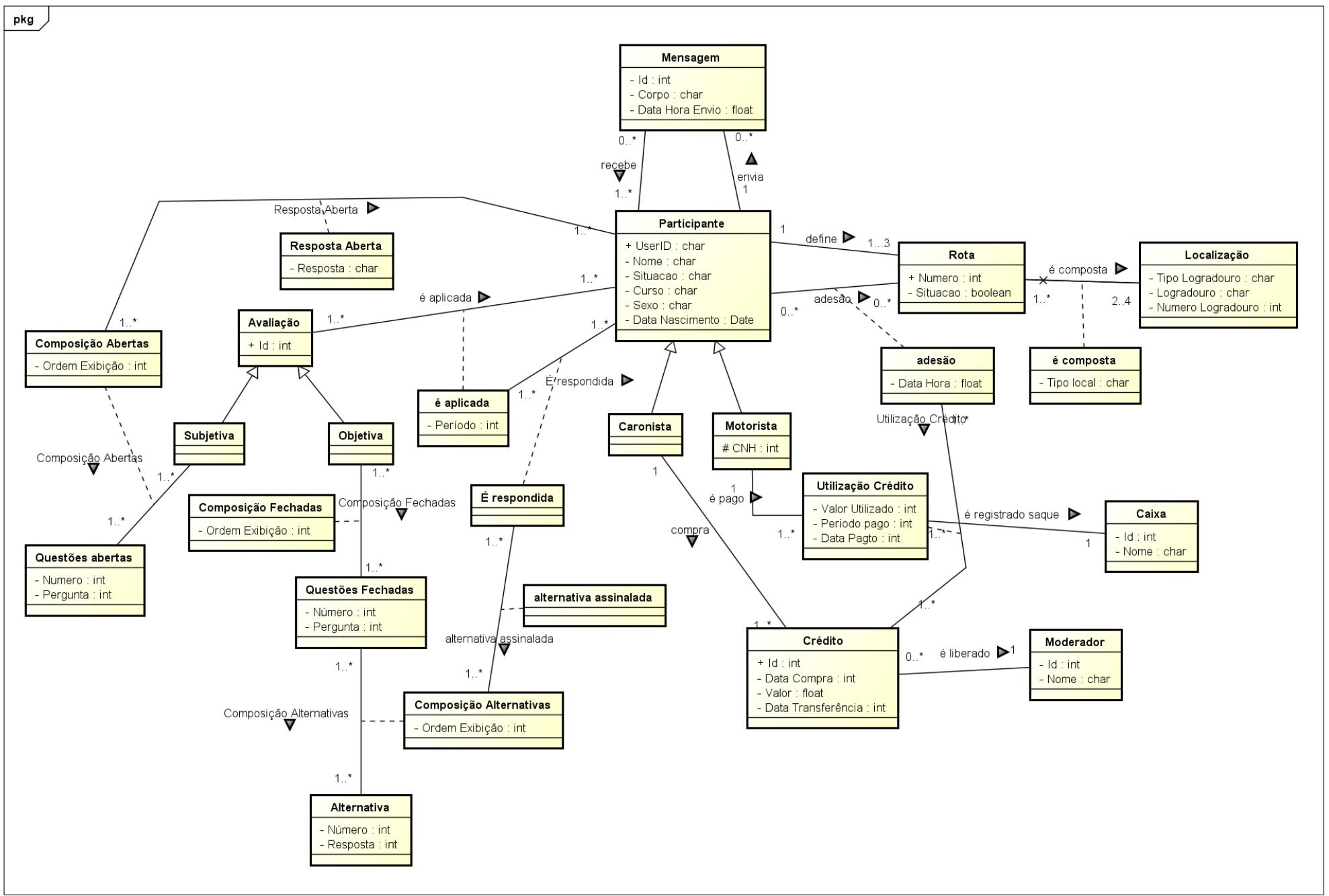
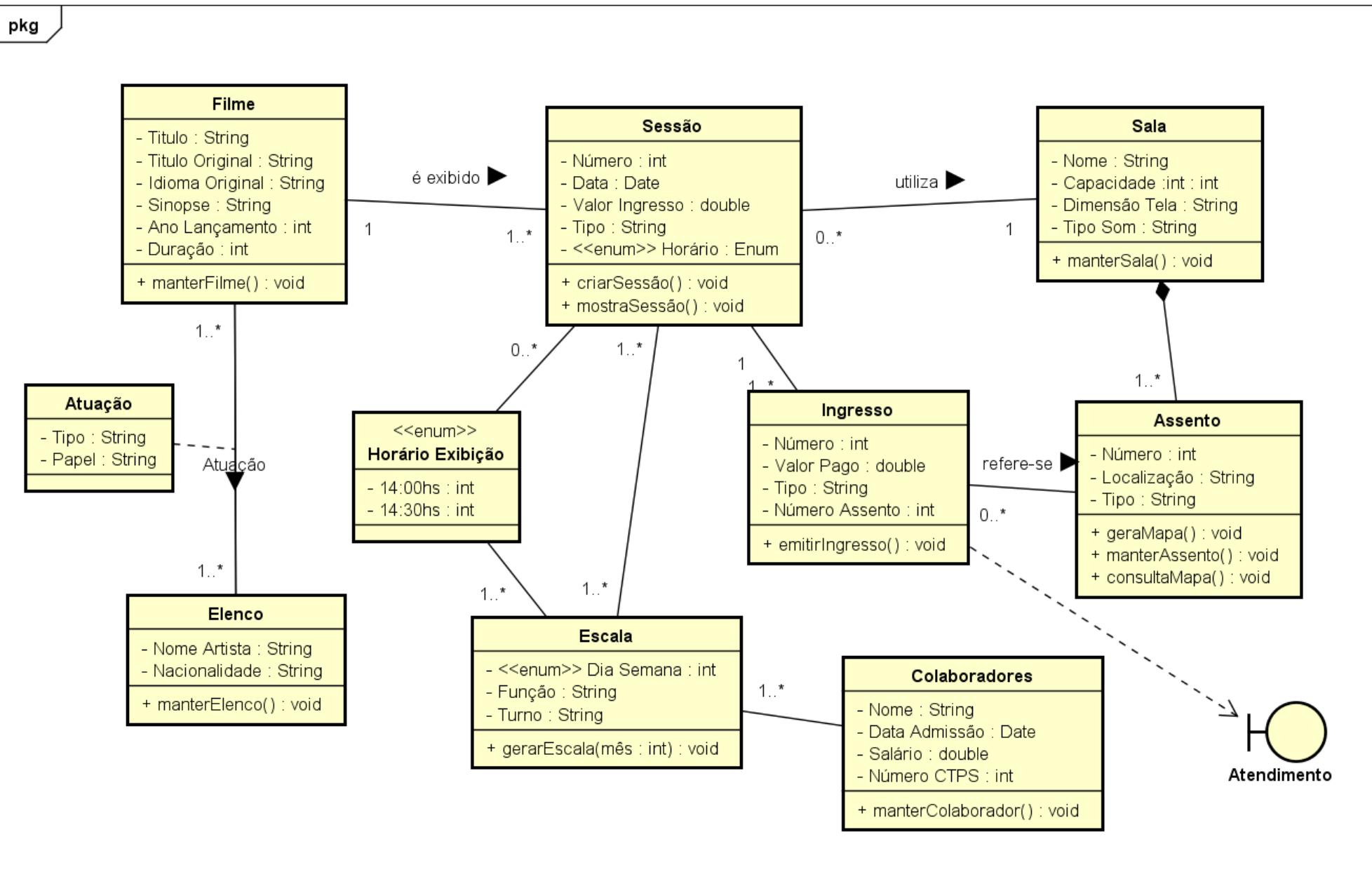
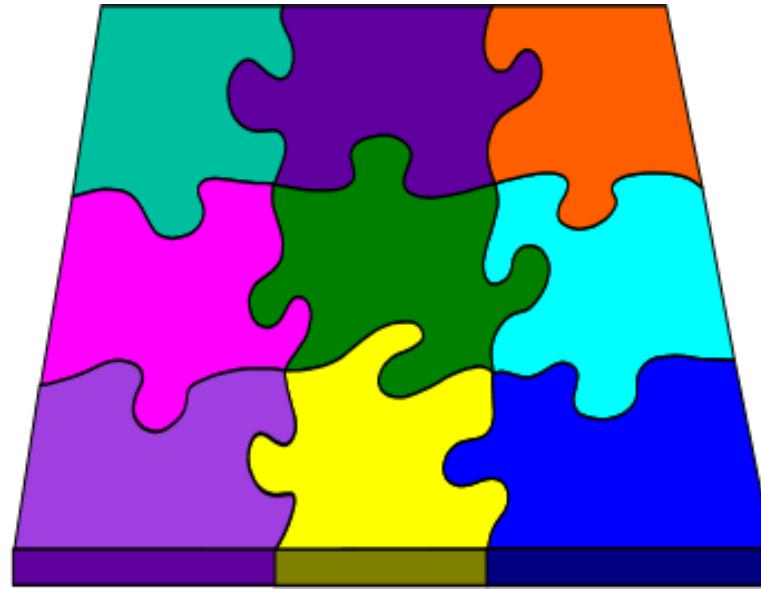


Diagrama de Classes – Exemplo Cinema





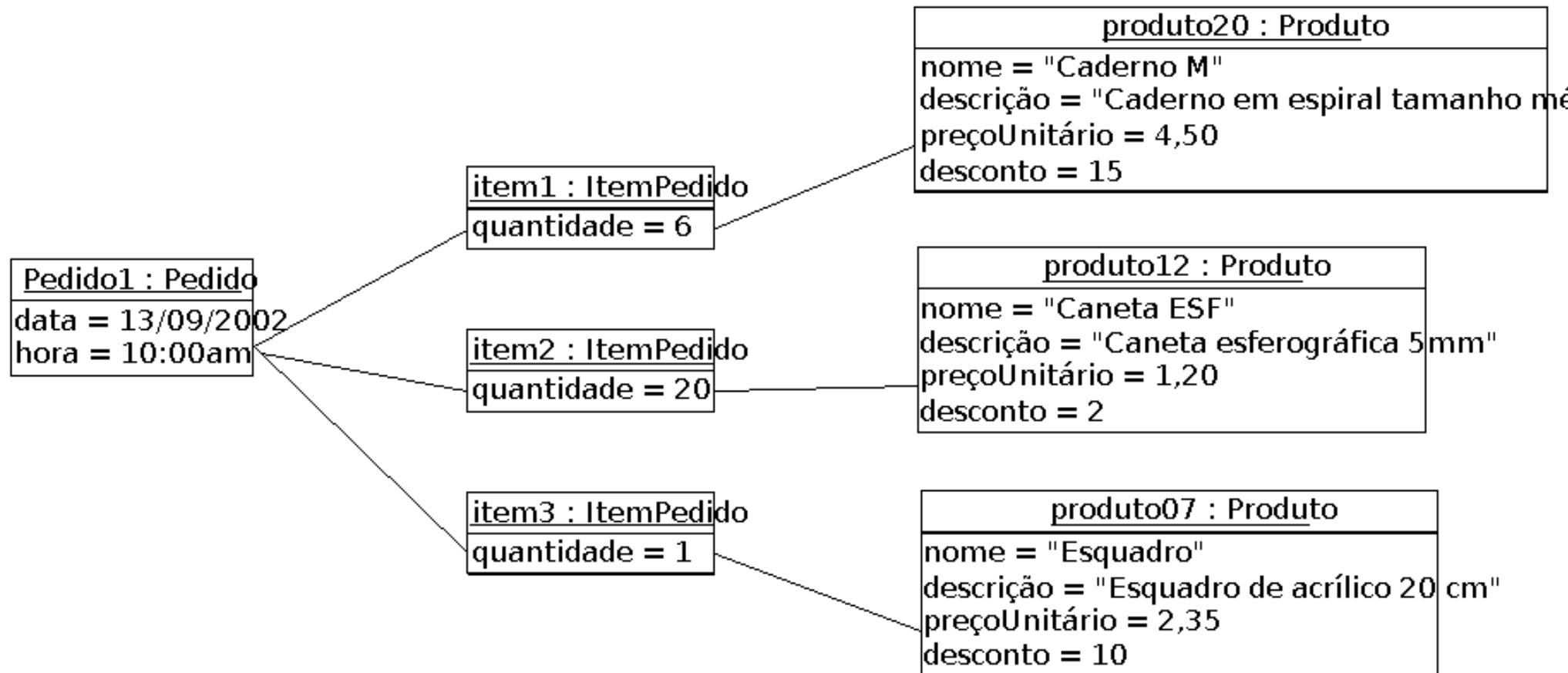
5.3 DIAGRAMA DE OBJETOS

DIAGRAMA DE OBJETOS

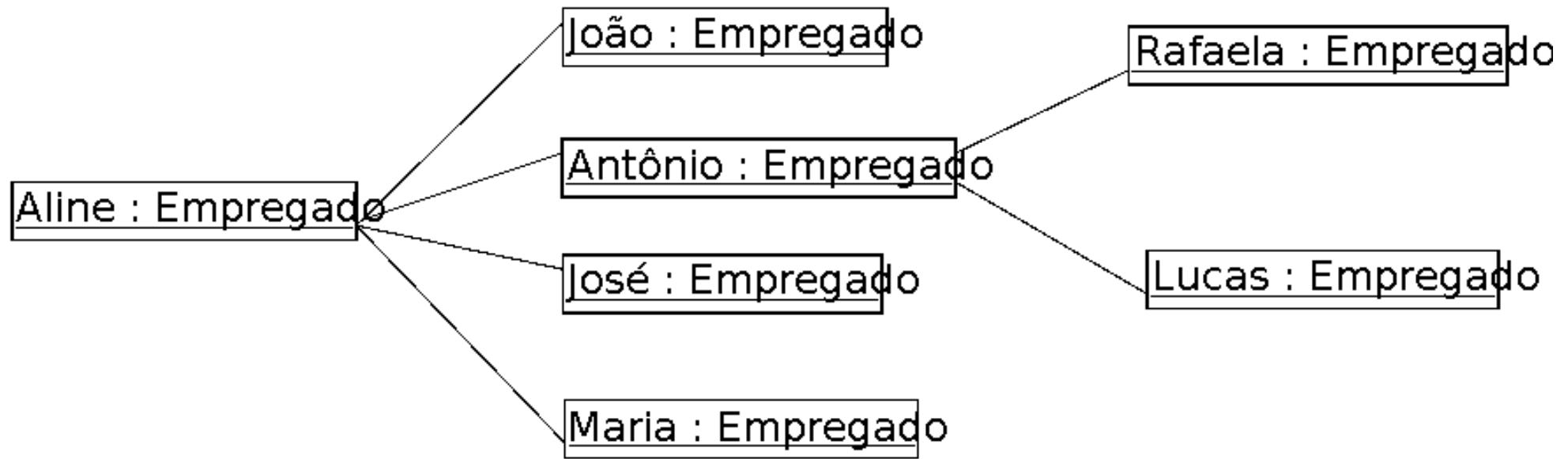
- Além do diagrama de classes, A UML define um segundo tipo de diagrama estrutural, o diagrama de objetos.
- Pode ser visto com uma instância de diagramas de classes
- Representa uma “fotografia” do sistema em um certo momento.
 - exibe as ligações formadas entre objetos conforme estes interagem e os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

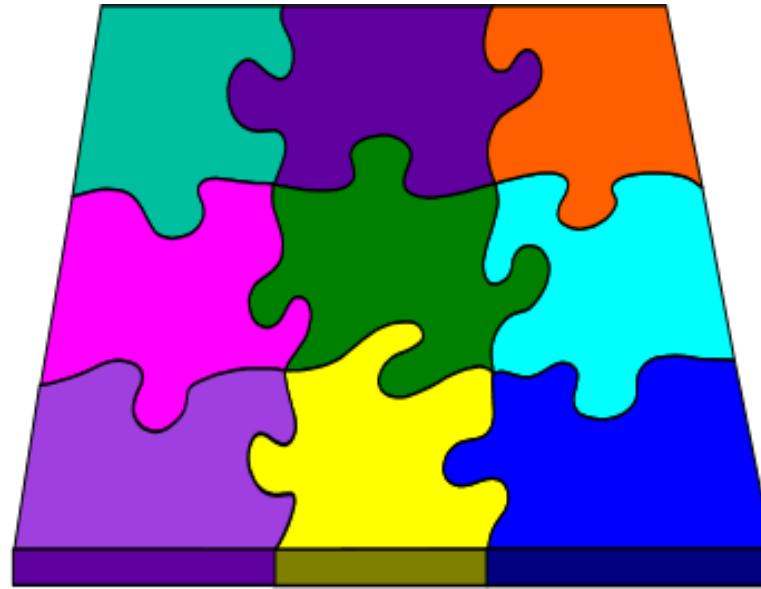
Formato	Exemplo
<u>nomeClasse</u>	<u>Pedido</u>
<u>nomeObjeto: NomeClasse</u>	<u>umPedido: Pedido</u>

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 (Analisis 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.
- Nesta 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 o 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 dos 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**...

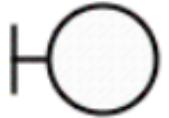
CATEGORIZAÇÃO BCE

- Na categorização BCE (Boundary,Control,Entity), 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

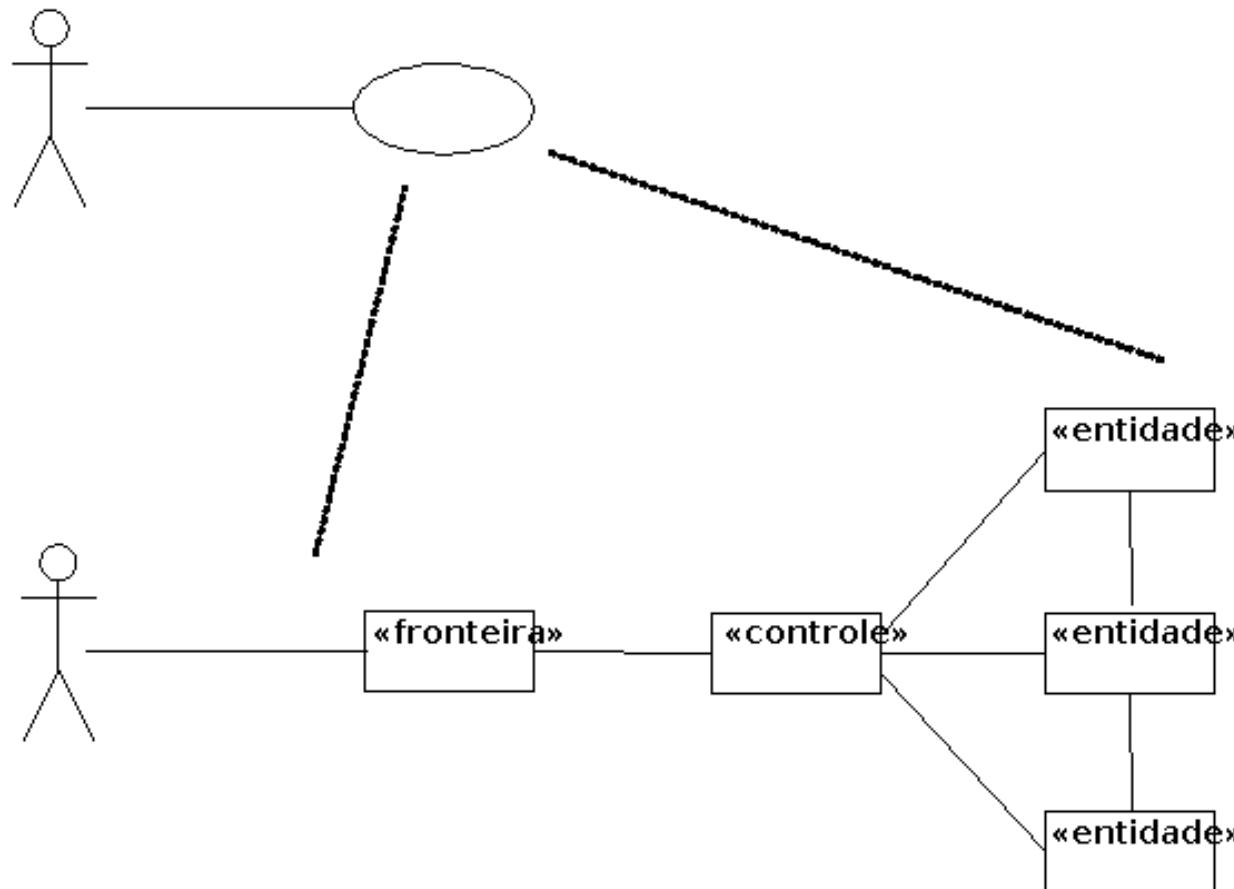
- A categorização BCE parte do princípio de que cada objeto em um SSOO é especialista em realizar um de três tipos de tarefa, a saber:
 - se comunicar com atores (**fronteira**),
 - manter as informações (**entidade**) ou
 - coordenar a realização de um caso de uso (**controle**).
- A categorização BCE 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 CLASSESS 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.



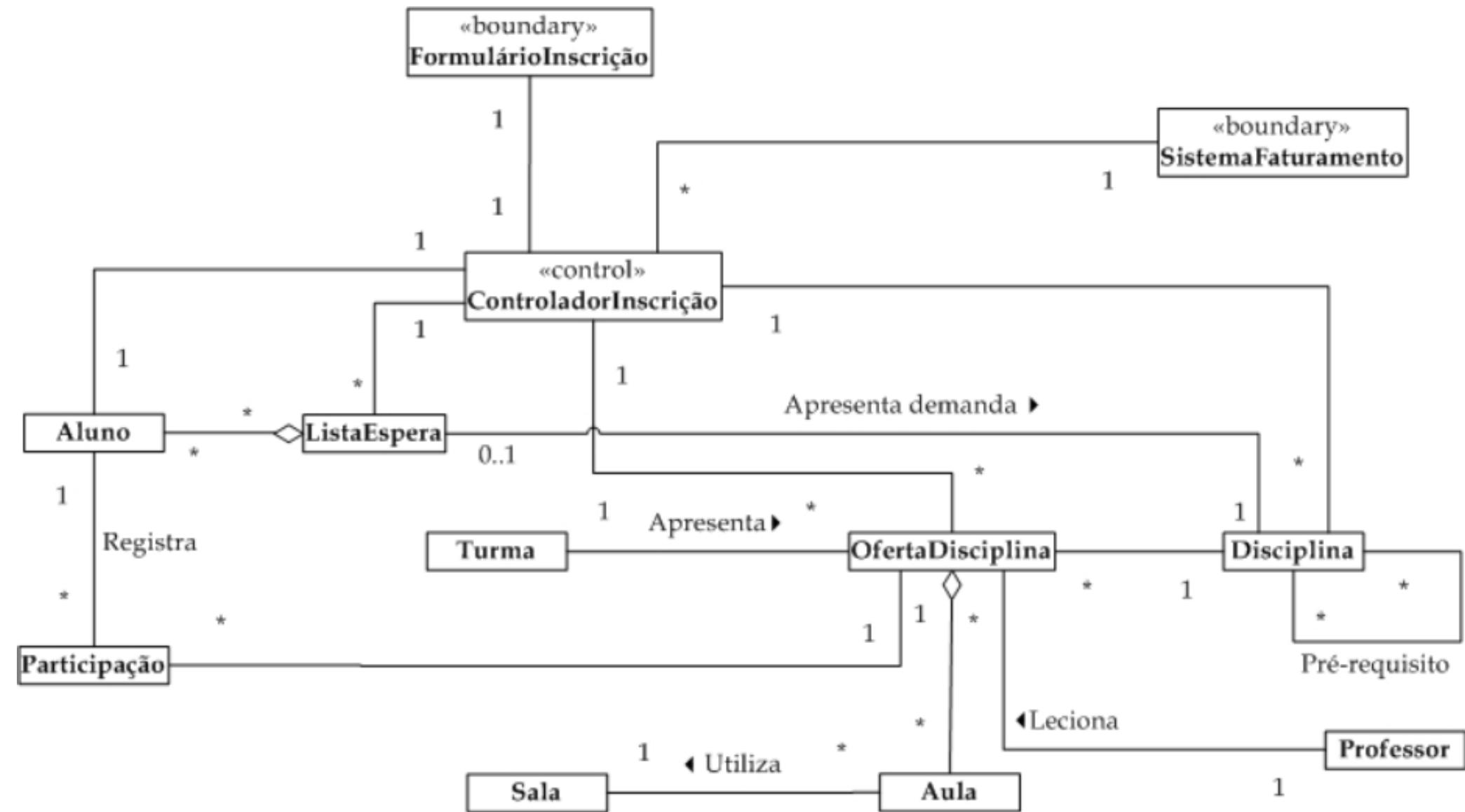
CONSTRUÇÃO DE UMA VCP

- 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.

RESPONSABILIDADES DE UMA CLASSE

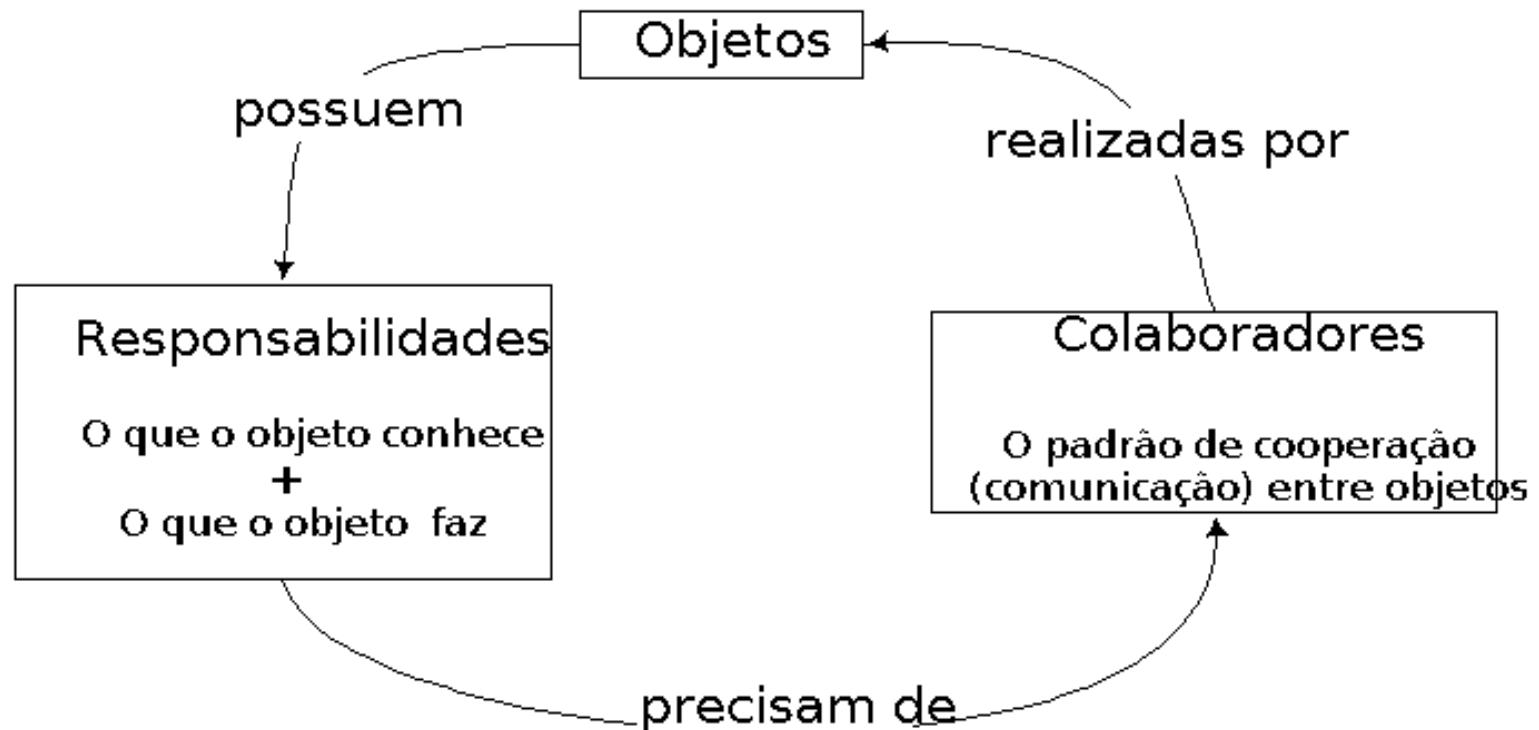
- Em um SSOO, objetos encapsulam comportamento.
 - O comportamento de um objeto é definido de tal forma que ele possa cumprir com suas **responsabilidades**.
- Uma responsabilidade é uma obrigação que um objeto tem para com o sistema no qual ele está inserido.
 - Através delas, um objeto colabora (ajuda) 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

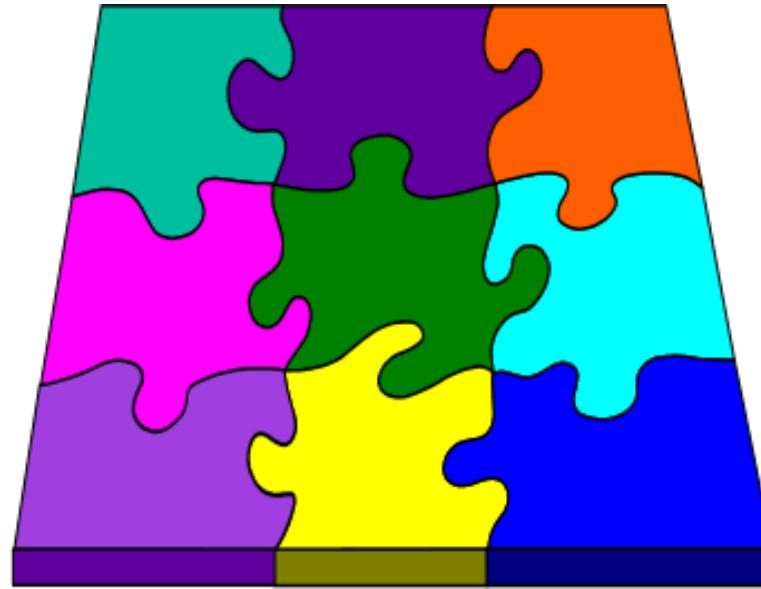
- Exemplo: considere clientes e seus pedidos:
 - Um objeto Cliente *conhece* seu nome, seu endereço, seu telefone, etc.
 - Um objeto Pedido *conhece* sua data de realização, *conhece* o seu cliente, *conhece* os seus itens componentes e *sabe fazer* o cálculo do seu total.
- Exemplo: quando a impressão 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.



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*)



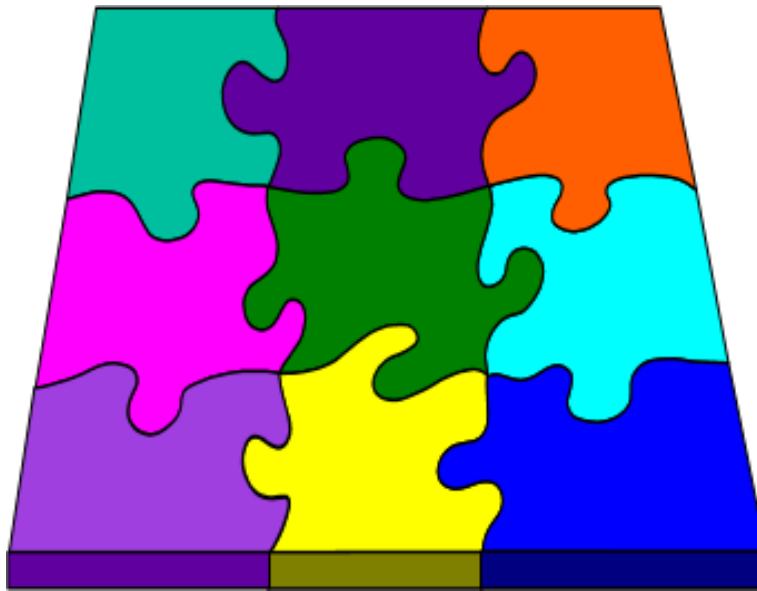


ORGANIZAÇÃO DA DOCUMENTAÇÃO

- 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. Uma 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

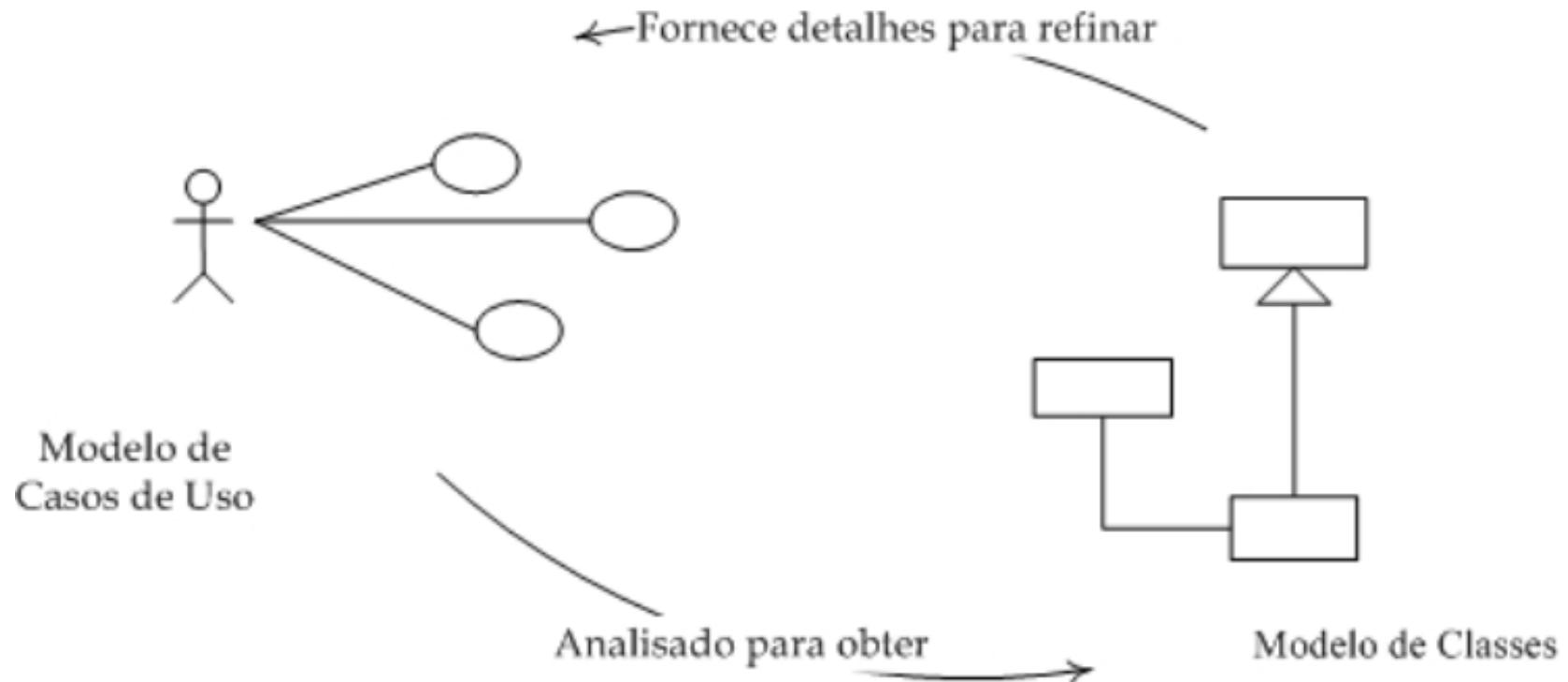
- Em um desenvolvimento dirigido a casos de uso, após a descrição dos casos de uso, é possível iniciar a identificação de classes.
- As classes identificadas são refinadas para retirar inconsistências e redundâncias.
- As classes são documentadas e o diagrama de classes inicial é construído, resultando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

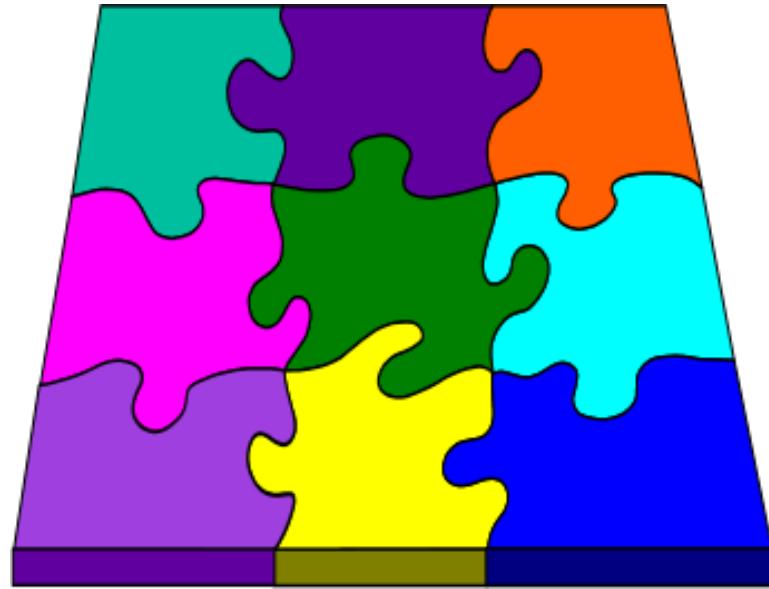
MODELO DE CLASSES NO PROCESSO DE DESENVOLVIMENTO

- 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.





5.7 HERANÇA

RELACIONAMENTO DE HERANÇA

- Na modelagem de classes de projeto, há diversos aspectos relacionados ao de ***relacionamento de herança***.
 - Tipos de herança
 - Classes abstratas
 - Operações abstratas
 - Operações polimórficas
 - Interfaces
 - Acoplamentos concreto e abstrato
 - Reuso através de delegação e através de generalização
 - Classificação dinâmica

TIPOS DE HERANÇA

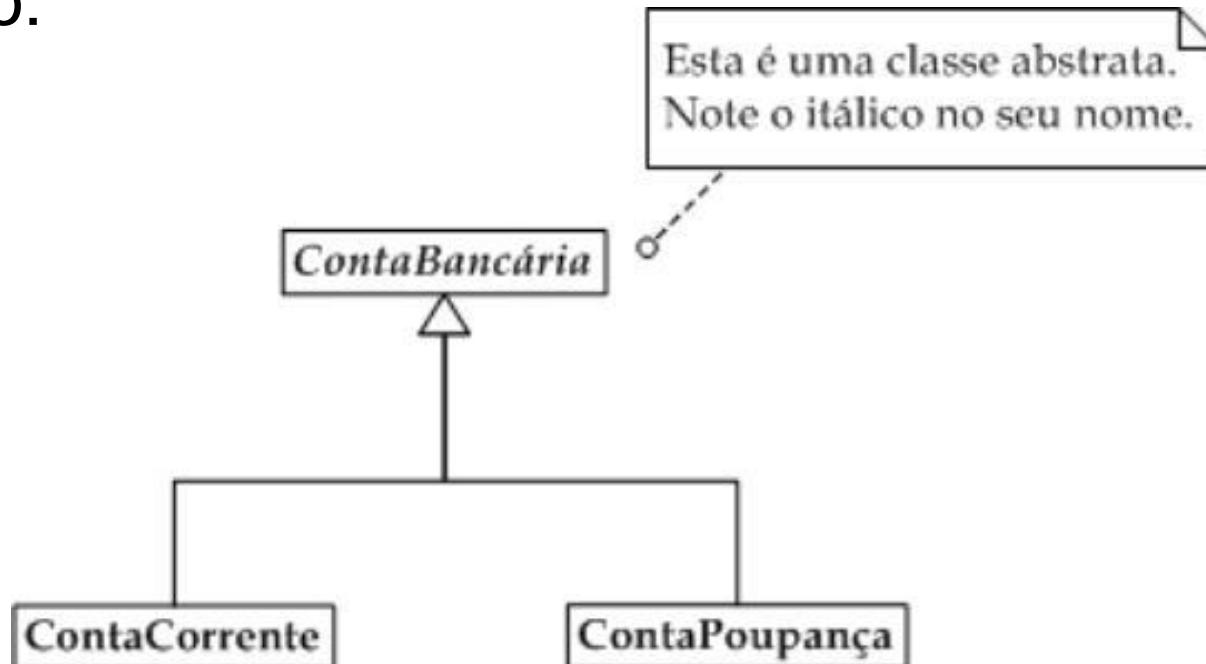
- Com relação à quantidade de superclasses que certa classe pode ter.
 - **herança múltipla**
 - **herança simples**
- Com relação à forma de reutilização envolvida.
 - Na **herança de implementação**, uma classe reusa alguma implementação de um “ancestral”.
 - Na **herança de interface**, uma classe reusa a interface (conjunto das assinaturas de operações) de um “ancestral” e se compromete a implementar essa interface.

CLASSES ABSTRATAS

- Usualmente, a existência de uma classe se justifica pelo fato de haver a possibilidade de gerar instâncias a partir da mesma.
 - Essas classes são chamadas de **classes concretas**.
- No entanto, podem existir classes que não geram instâncias “diretamente”.
 - Essas classes são chamadas de **classes abstratas**.
- Classes abstratas são usadas para organizar hierarquias gen/spec.
 - Propriedades comuns a diversas classes podem ser organizadas e definidas em uma classe abstrata a partir da qual as primeiras herdam.
- Também propiciam a implementação do **princípio do polimorfismo**.

CLASSES ABSTRATAS (CONT)

- Na UML, uma classe abstrata pode ser representada de duas maneiras alternativas:
 - Com o seu nome em *itálico*.
 - Qualificando-a com a propriedade `{abstract}`
- Exemplo:



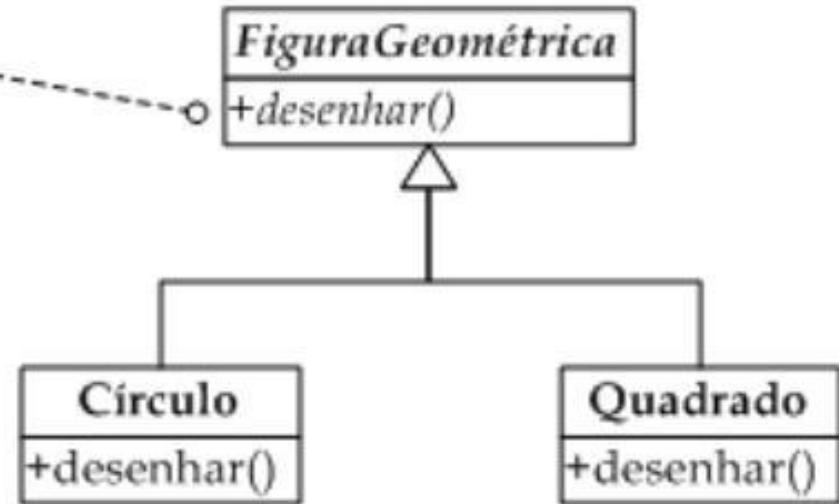
OPERAÇÕES ABSTRATAS

- Uma classe abstrata possui ao menos uma ***operação abstrata***, que corresponde à especificação de um serviço que a classe deve fornecer (sem método).
 - Uma classe qualquer pode possuir tanto operações abstratas, quanto operações concretas (ou seja, operações que possuem implementação).
 - Entretanto, uma classe que possui pelo menos uma operação abstrata é, por definição abstrata, abstrata.
- Uma operação abstrata definida com visibilidade pública em uma classe também é herdada por suas subclasses.
- Quando uma subclasse herda uma operação abstrata e não fornece uma implementação para a mesma, esta classe também é abstrata.

OPERAÇÕES ABSTRATAS (CONT)

- Na UML, a assinatura de uma operação abstrata é definida em itálico.

Operação abstrata. Note o itálico em sua assinatura. Subclasses devem implementar o comportamento desta operação para serem concretas.



OPERAÇÕES POLIMÓRFICAS

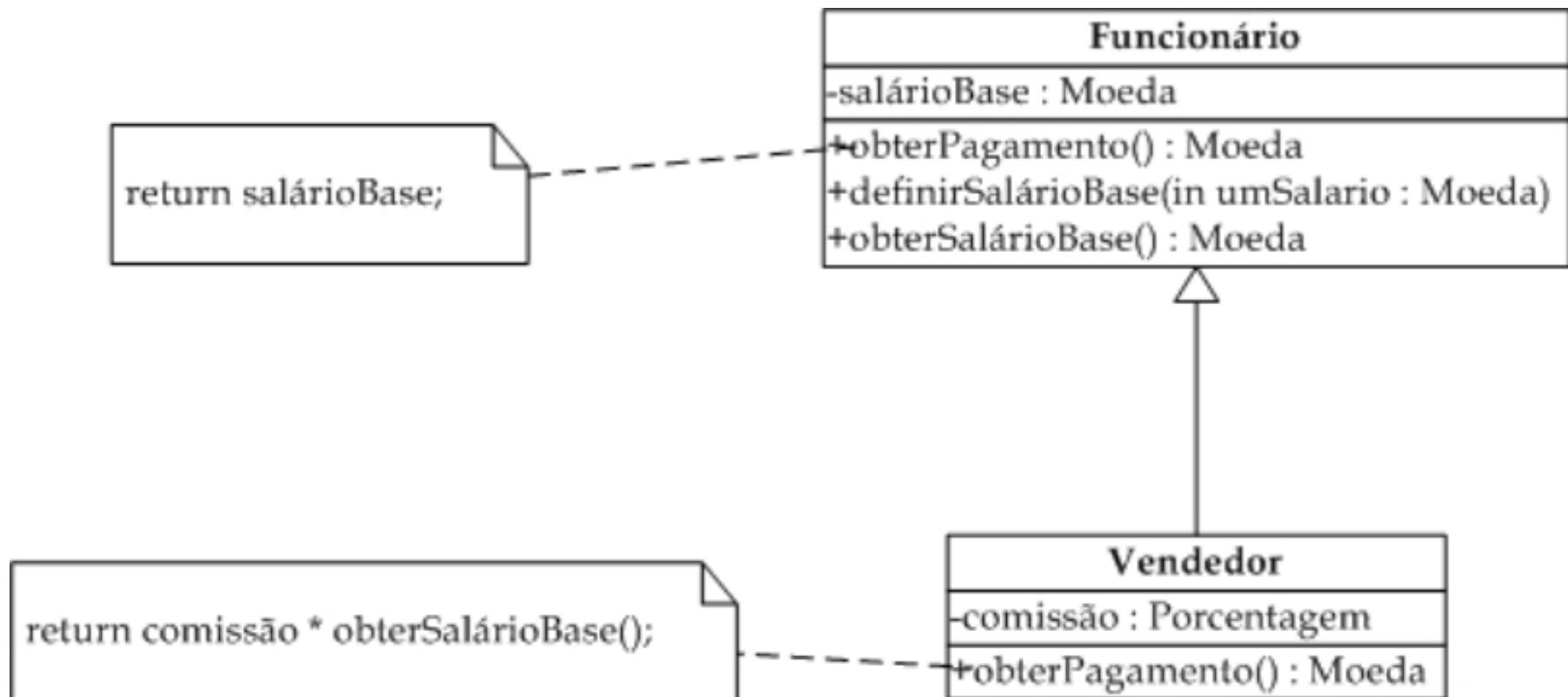
- Uma subclasse herda todas as propriedades de sua superclasse que tenham visibilidade pública ou protegida.
- Entretanto, pode ser que o comportamento de alguma operação herdada seja diferente para a subclasse.
- Nesse caso, a subclasse deve redefinir o comportamento da operação.
 - A assinatura da operação é reutilizada.
 - Mas, a implementação da operação (ou seja, seu **método**) é diferente.
- Operações polimórficas são aquelas que possuem mais de uma implementação.

OPERAÇÕES POLIMÓRFICAS (CONT)

- Operações polimórficas possuem sua assinatura definida em diversos níveis de uma hierarquia gen/spec.
 - A assinatura é repetida na(s) subclasse(s) para enfatizar a redefinição de implementação.
 - O objetivo de manter a assinatura é garantir que as subclasses tenham uma interface em comum.
- Operações polimórficas facilitam a implementação.
 - Se duas ou mais subclasses implementam uma operação polimórfica, a mensagem para ativar essa operação é a mesma para todas essas classes.
 - No envio da mensagem, o remetente não precisa saber qual a verdadeira classe de cada objeto, pois eles aceitam a mesma mensagem.
 - A diferença é que os métodos da operação são diferentes em cada subclasse.

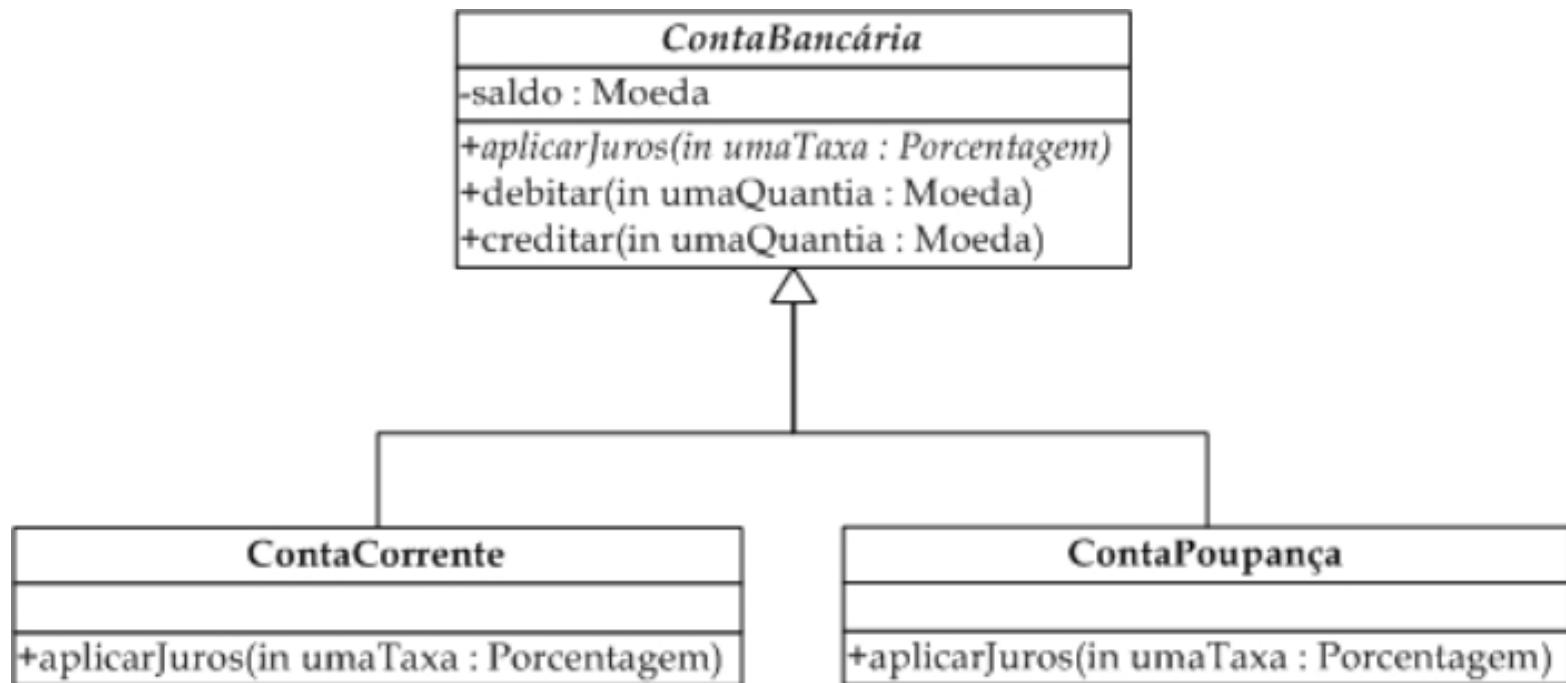
OPERAÇÕES POLIMÓRFICAS (CONT)

- A operação obterPagamento é polimórfica.



OPERAÇÕES POLIMÓRFICAS (CONT)

- Operações polimórficas também podem existir em classes abstratas.



OPERAÇÕES POLIMÓRFICAS (CONT)

- Operações polimórficas implementam o **princípio do polimorfismo**, no qual dois ou mais objetos respondem a mesma mensagem de formas diferentes.

```
ContaCorrente cc;  
ContaPoupanca cp;  
...  
List<ContaBancaria> contasBancarias;  
...  
contasBancarias.add(cc);  
contasBancarias.add(cp);  
...  
for(ContaBancaria conta : contasBancarias) {  
    conta.aplicarJuros();  
}
```

INTERFACES

- Uma **interface** entre dois objetos compreende um conjunto de **assinaturas de operações** correspondentes aos serviços dos quais a classe do objeto cliente faz uso.
- Uma interface pode ser interpretada como um **contrato de comportamento** entre um objeto cliente e eventuais objetos fornecedores de um determinado serviço.
 - Contanto que um objeto fornecedor forneça implementação para a interface que o objeto cliente espera, este último não precisa conhecer a verdadeira classe do primeiro.

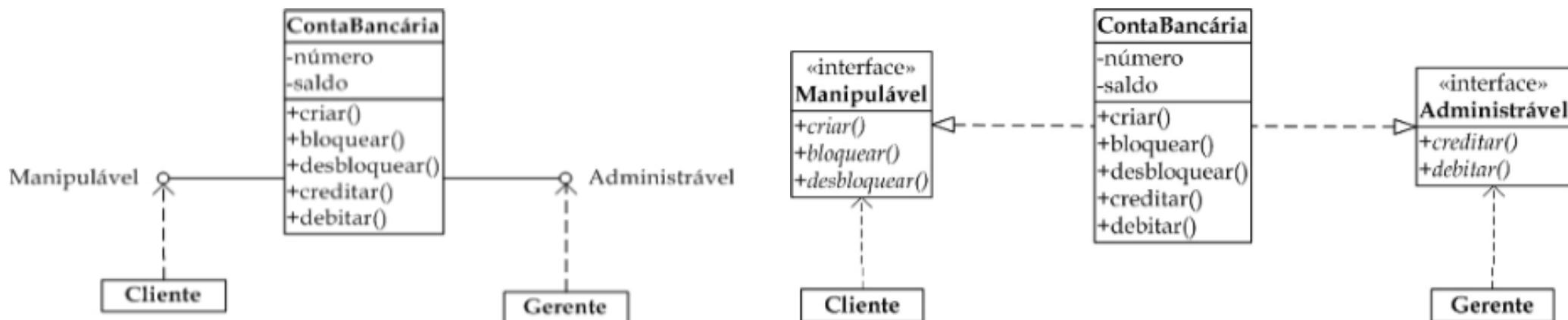
INTERFACES (CONT.)

- Interfaces são utilizadas com os seguintes objetivos:
 - 1. Capturar semelhanças entre classes não relacionadas sem forçar relacionamentos entre elas.
 - 2. Declarar operações que uma ou mais classes devem implementar.
 - 3. Revelar as operações de um objeto, sem revelar a sua classe.
 - 4. Facilitar o desacoplamento entre elementos de um sistema.
- Nas LPOO modernas (Java, C#, etc.), interfaces são definidas de forma semelhante a classes.
 - Uma diferença é que todas as declarações em uma interface têm visibilidade pública.
 - Adicionalmente, uma interface não possui atributos, somente declarações de assinaturas de operações e (raramente) constantes.

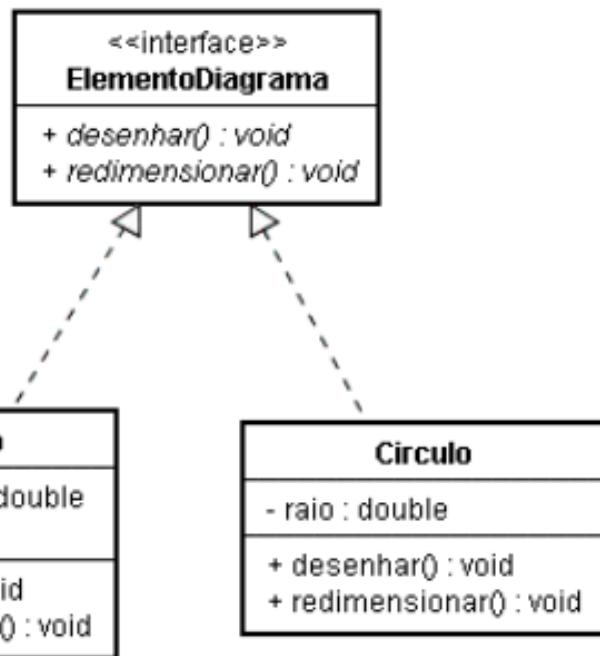
INTERFACES (CONT)

- Notações para representar interfaces na UML:

- A primeira notação é a mesma para classes. São exibidas as operações que a interface especifica. Deve ser usado o estereótipo <<interface>>.
- A segunda notação usa um segmento de reta com um pequeno círculo em um dos extremos e ligado ao classificador.
 - Classes clientes são conectadas à interface através de um relacionamento de notação similar à do relacionamento de dependência.



INTERFACE (CONT)



```
public interface ElementoDiagramma {  
    double PI = 3.1425926; //static and final constant.  
    void desenhar();  
    void redimensionar();  
}
```

```
public class Circulo implements ElementoDiagramma {  
    ...  
    public void desenhar() { /* draw a circle */ }  
    public void redimensionar() { /* draw a circle */ }  
}
```

```
public class Retangulo implements  
ElementoDiagramma {  
    ...  
    public void desenhar() { /* draw a rectangle */ }  
    public void redimensionar() { /* draw a rectangle */ }  
}
```