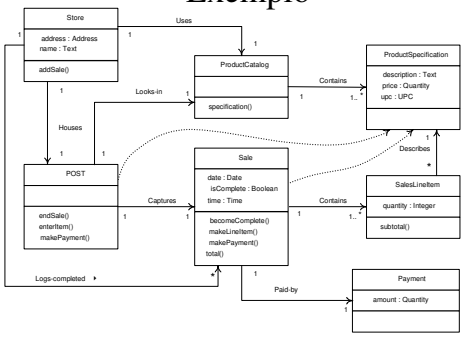
Diagrama de Classes

Diagrama de Classes

- O caso de uso fornece uma perspectiva do sistema de um ponto de vista externo (do ator)
- Internamente os objetos colaboram para atender às funcionalidades do sistema
- Demonstra a estrutura estática dessa colaboração, mostra as classes de um sistema, seus atributos e operações, e como as classes se relacionam.
  - O diagrama de objetos (que pode ser visto como uma instanciação do diagrama de classes) também representa a estrutrua estática

Diagrama de Classes – Um Exemplo



Perspectivas de um Diagrama de Classes

- O diagrama de classes evolui com o sistema e pode ter diferentes perspectivas

Na análise – identificamos objetos (classes) no domínio do problema

No projeto – pensamos em objetos (classes) para a solução

Perspectivas de um Diagrama de Classes

- **O *modelo conceitual (análise)*** representa as classes no domínio do negócio em questão. Não leva em consideração restrições inerentes à tecnologia a ser utilizada na solução de um problema.
- **O *modelo de classes de especificação (projeto)*** é obtido através da adição de detalhes ao modelo anterior conforme a solução de software escolhida.
- **O *modelo de classes de implementação*** corresponde à implementação das classes em alguma linguagem de programação.

Definição de Objetos

- Conceitual: representa uma entidade, "coisa" , processo ou conceito do mundo real e que possui:
  - Identidade – valor de uma característica que o identifica para reconhecimento
  - Atributos – qualidades, características
  - Comportamento – habilidades de processamento

Definição de Objetos

- De implementação: representa um módulo de sw que recebe e produz dados
  - Identidade – identificador em lg de implementação
  - Atributos – variáveis e seus tipos, que recebem diferentes valores e definem o estado do objeto
  - Comportamento – funções ou procedimentos, os resultados dessas funções determinam o comportamento do objeto

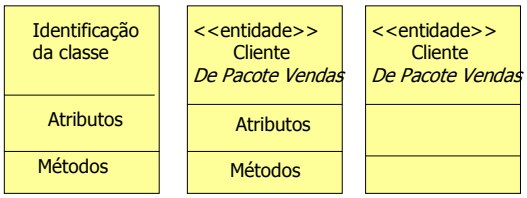
Definição de Classes

- Conceitual: são agrupamentos de objetos, são abstrações de um coletivo de entidades do mundo real
- O modelo genérico desse coletivo contém atributos e comportamentos comuns.

Definição de Classes

- De implementação: corresponde a um tipo de uma lg de programação
- Um modelo genérico para criar variáveis que armazenarão os objetos correspondentes.

Notação UML para Classes



Atributo

- Característica, qualidade de um objeto ou classe.
- Seus valores servem para diferenciar objetos (Instâncias)

Overriding (ou Sobreposição)

Mecanismo para redefinir ou tornar um atributo não aplicável

Notação UML para Atributos

- A maioria é opcional, seu uso vai depender do tipo de visão no qual estamos trabalhando e podem ser abstratos ou utilizar a notação de uma lg de programação

[Visibili/d]Nome[Multiplici/d]:[Tipo]=[Valor][{Proprie/ds}]

Notação UML para Atributos - Visibilidade

- **+** : visibilidade pública: o atributo ü visível no exterior da classe.
- **-** : visibilidade privada : o atributo é visível somente por membros da classe.
- **#** : visibilidade protegida: o atributo é visível também por membros de classes derivadas

Notação UML para Atributos - Multiplicidade

- Usada para especificar atributos que são arranjos
- Indica dimensão de vetores e matrizes

Ex: notas[10]

matrizDeValores[5,10]

Notação UML para Atributos - Tipos

- Indicam o formato do valores que o atributo pode assumir
- Na visão conceitual o tipo é abstrato

Ex: dataDaVenda: tipoData

- Na visão de implementação utilizam-se os tipos da lg de programação

Ex: salario: float

Notação UML para Atributos – Valor Inicial e Propriedades

- Pode-se indicar o valor ou conteúdo do atributo imediatamente após a sua criação, ou o seu valor default

Ex: resultado: int=0

- As propriedades descrevem comentários ou indicações sobre o atributo, podem mostrar se ele é ou não opcional

Ex: dataDaVenda {valor constante}

Métodos ou Serviços

Processamento realizado por um objeto que indica o seu comportamento, em função do recebimento de uma mensagem

Mensagens

Ativação de um método. Um objeto se utiliza de um serviço ou se comunica com outro objeto enviando uma mensagem

Métodos ou Serviços

Ligação Dinâmica (late binding)

Mecanismo pelo qual se decide o destino de uma mensagem em tempo de execução

Ex: int (*f)(); a função só definida

i =f(); → durante a execução

Métodos ou Serviços

Sobrecarga de operadores (Overloading)

Operações de mesmo nome, mas implementadas de maneira diferente

Ex: operações de adição, subtração implementadas diferentemente para real e inteiro

Métodos ou Serviços

Polimorfismo

Possibilidade de uma função poder manipular valores com tipos (formas) diversas

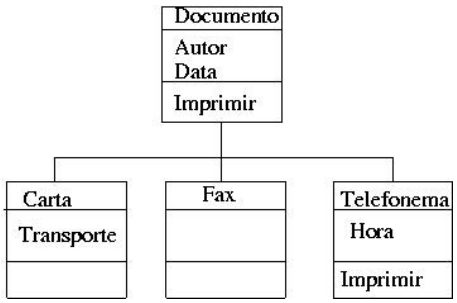
Ex: function comp(L:lista):integer

↑

qualquer tipo de lista (genérico)

mesma implementação

Métodos ou Serviços



Métodos ou Serviços

Construtor/Destrutor

Função para criação/remoção de instâncias de objetos/classes

Notação UML para Métodos

- A notação e uso vai depender do tipo de visão no qual estamos trabalhando e podem ser abstratos ou utilizar a notação de uma lg de programação

[Visibili/d]Nome(Parâmetros)]:[Retorno][{Proprie/ds}]

Notação UML para Métodos - Parâmetros

- Os parâmetros – dados de entrada e/ou saída para o método são representados por

Nome-do-Parâmetro:Tipo=Valor-Padrão

Ex:

- Visão conceitual

ImprimirData(data:TipoData)

Visão de implementação

- ArmazenarDados(nome:char[30],salario:float=0.0)

Notação UML para Métodos – Tipo de Retorno

- O Valor-de-Retorno indica se o método retorna algum valor ao término de sua execução e qual o tipo de dado do valor retornado.

Ex:

- Visão Conceitual

  CalcularValor(): TipoDinheiro

- Visão de implementação

  ArmazenarDados(nome:char[30]): bool

Notação UML para Métodos – Visibilidade e Propriedades

- Visibilidade – similar ao de atributo
- Comentários ou restrições para os métodos
- Ex:

  Area() {Área <=600}

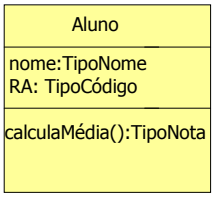
  Restringe a 600 unidades o valor máximo das áreas a calcular

Notação UML para Métodos – Propriedades

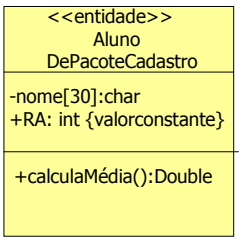
- É útil distingüir operações de métodos. Operações é algo que se evoca sobre um objeto (a chamada do procedimento). Para realizar uma operação a classe implementa um método (o corpo do procedimento)
- É útil distingüir operações que alteram ou não o estado (atributos) de uma classe
- Não alteram: query, métodos de obtenção, getting methods
- Alteram: modificadores, métodos de atribuição ou fixação, setting methods

Exemplo de Uma Classe

Visão Conceitual



Visão de Implementação



Construção do Diagrama de Classes – Refinamentos Sucessivos

- VisãoAbstrata → detalhadamento (impl.)
- Na visão conceitual: cada classe pode ser vista como um conceito ou um tipo, e os métodos são identificados numa fase posterior.
- Na visão de implementação: os métodos aparecem obrigatoriamente e consideramos aspectos de controle, estereótipos, pacotes, etc.
- Podem existir outras visões intermediárias (por exemplo: de domínio e de aplicação, (análise) de especificação (projeto)

Construção do Diagrama de Classes

- Na fase de análise constrói-se primeiramente um diagrama de classes sem se preocupar em definir os métodos, ao qual chamaremos de Modelo Conceitual
- Na fase de projeto os métodos são adicionados e o Modelo Conceitual é refinado gerando o Diagrama de Classes

Diagrama de Classes

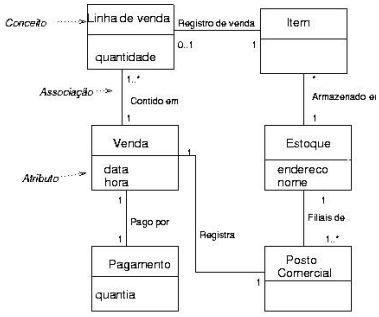
Perspectiva Conceitual
Modelo Conceitual

Modelo Conceitual

- Representação de conceitos no domínio do problema
- Deve mostrar: conceitos, associações entre conceitos e atributos de conceitos

(na fase de análise, não se preocupa ainda em representar métodos os serviços)

Modelo Conceitual - Exemplo



Modelo Conceitual – Exemplo

Criando um Diagrama de Classes- Perspectiva Conceitual

- 1. Liste os conceitos candidatos para os casos de usos em questão usando a lista de categorias comuns e identificação textual de nomes.
- 2. Desenhe-os em um modelo conceitual.
- 3. Adicione as associações necessárias para registrar os relacionamentos para os quais é preciso preservar alguma memória
- 4. Adicione os atributos necessários para cumprir os requisitos de informação.

Identificando Conceitos

- É uma entidade (idéia, coisa ou objeto) do mundo real.
- Um bom modelo conceitual deve superestimar o número de conceitos.
- Os conceitos são associados ao estereótipo de classe << entidade >>
  - um estereótipo para uma classe UML é um classificador que mostra o tipo ao qual a classe pertence.

Identificando Conceitos (Entidades) –Regras Úteis

- É melhor especificar demais do que de menos
- Não exclua entidades simplesmente porque os requisitos não indicam a necessidade de guardar informações sobre eles (comum em projeto de BD)
- Comece fazendo uma lista de entidades candidatos a partir de um *checklist.*
- Considere os substantivos e frases nominais nas descrições textuais do domínio do problema como possíveis candidatos a entidades ou atributos

Checklist: Classes Entidades Típicas

Categoria	Exemplos
Objeto físico ou tangível	Terminal de ponto-de-venda Avião
Especificação, projeto, ou descrição de coisas	Especificação de produto Descrição de vôo
Lugares	Loja Aeroporto
Transações	Venda, Pagamento Reserva
Itens de transação	Itens de venda Parcelas de pagamento
Papéis de pessoas	Operador Piloto
Container de coisas	Loja Avião

Classes Entidades Típicas

Categoria	Exemplos
Coisas em um *container*	Item Passageiro
Sistemas externos	Serviço de crédito Controle de tráfego aéreo
Nomes abstratos	Fome Aracnofobia
Organizações	Departamento de vendas Companhia aérea
Eventos	Venda, Assalto, Reunião Vôo, Decolagem
Regras e políticas	Política de devolução Política de cancelamento

Classes Entidades Típicas

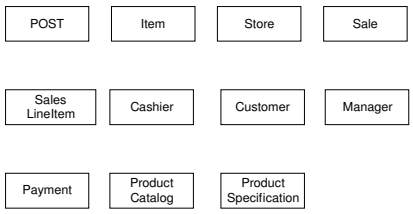
Categoria	Exemplos
Catálogos	Catálogo de produtos Catálogo de peças
Registros de finança, trabalho, contrato, questões legais	Recibo, Contrato de trabalho Registro de manutenção
Instrumentos e serviços financeiros	Linha de crédito Ações
Manuais, livros	Manual do empregado Manual de reparos

Identificando Entidades a partir dos Casos de Uso

Ação do Ator	Resposta do Sistema
1. Este caso de uso começa quando um **Cliente** chega no **caixa** com **itens** para comprar.	
2. O **Operador** registra o **identificador** de cada **item**. Se há mais de um do mesmo **item**, o **Operador** também pode informar a **quantidade**.	**3.** Determina o **preço do item** e adiciona informação sobre o **item** à **transação de venda** em andamento. Mostra a **descrição** e o **preço** do **item** corrente.

- Usar com cuidado!
  - Linguagens naturais: imprecisão e ambigüidade

Entidades Candidatas para o Sistema Posto Comercial



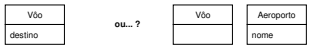
- Conceitos restritos ao caso de uso *Comprar Itens - Versão 1*

Entidades de Relatório

- Não incluir no modelo conceitual quando:
  - Toda informação contida no relatório é derivada de outras fontes
- Incluir no modelo conceitual quando:
  - Relatório tem um papel especial em termos das regras de negócio
    - Ex.: Recibo de venda dá direito à devolução dos itens comprados

Criando o Diagrama de Classes- Perspectiva Conceitual

- Estratégia do "fazedor de mapas":
  - Usar nomes existentes no vocabulário do domínio
  - Incluir apenas conceitos pertinentes para os requisitos (casos de uso) em questão
  - Excluir tudo que não há no domínio do problema
- Erro comum: atributo em vez de entidade

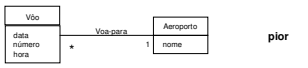


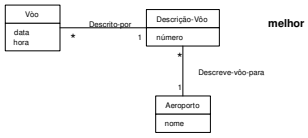
  - Atributos normalmente correspondem a um texto ou número no mundo real

Entidades de Especificação ou Descrição

- A especificação ou descrição de um objeto deve ser representada como uma entidade em separado
  - evita perda de informação quando o objeto é deletado
  - reduz informações redundantes ou duplicadas
- Muito comum no domínio de produtos e vendas
  - Ex.:

Entidades de Especificação ou Descrição – Outro exemplo

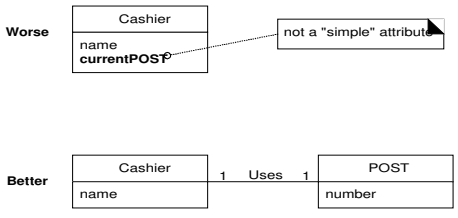




Identificando Atributos

- Atributos devem preferencialmente representar tipos primitivos de dados ou de valores simples
  - Ex.: *Data*, *Número*, *Texto*, *Hora*, *Endereço*, etc.
- Atributos <u>não</u> devem ser usados para:
  - Representar um conceito complexo
  - Relacionar conceitos (atributo "chave-estrangeira")

Identificando Atributos



Atributos de Tipo Não-Primitivo

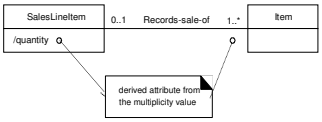
- Um atributo deve ser de tipo não-primitivo quando:
  - É composto de seções separadas
    - Ex.: endereço, data
  - Precisa ser analisado ou validado
    - Ex.: CPF, número de matrícula
  - Possui outros atributos
    - Ex.: Um preço promocional com prazo de validade
  - É uma quantidade com uma unidade
    - Ex.: valores monetários, medidas

Adicionando Atributos ao Sistema POST

Conceito	Atributos e justificativa
Pagamento	*quantia*—Para determinar se pagamento é suficiente e calcular troco.
Especificação-Produto	*descrição*—Para mostrar na tela e imprimir no recibo. *UCP*—Para localizar especificação do item. *preço*—Para calcular o total da venda.
Venda	*data*, *hora*—Para imprimir no recibo e registrar no log de vendas.
Item de Venda	*quantidade*—Para registrar a quantidade digitada quando há mais de um do mesmo item.
Loja	*nome*, *endereço*—Para imprimir no recibo.

Atributo Derivado

- Um atributo *"derivado"* é um atributo cujo valor pode ser deduzido a partir de outras informações
  - Ex.: *quantidade* em *Item de Venda*—pode ser deduzido a partir da multiplicidade da associação entre *Item de Venda* e *Item*



Identificando Relacionamentos

- Os relacionamentos entre as classes representam a interação entre seus objetos: em geral, representam a utilização de serviços e/ou a organização entre as mesmas.
  - Devem refletir o domínio do problema
- Tipos:
  - Associação (associação simples)
  - Agregação/composição
  - Generalização
  - Dependência
  - Refinamentos

1. Associações

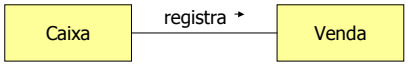
- Conexão entre classes/objetos Relacionamento que descreve uma série de ligações entre duplas de classes/ objetos
- Uma ligação significa por exemplo que:
  - elas "conhecem uma a outra"
  - "estão conectadas com"
  - para cada X existe um Y

Associações

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

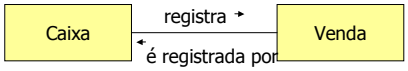
Associações

- O mais comum, com apenas uma conexão, representada por uma linha sólida e um nome (geralmente verbo)



Associações

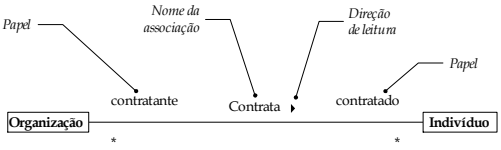
- Podem possuir dois nomes, significando um nome para cada sentido da associação e os papéis de cada classe



Nome de associação, direção de leitura e papéis

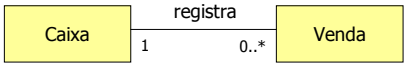
- Para melhor esclarecer o significado de uma associação no diagrama de classes, a UML define três recursos de notação:
  - ***Nome da associação***: fornece algum significado semântico à mesma.
  - ***Direção de leitura***: indica como a associação deve ser lida
  - ***Papel***: para representar um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

Exemplo (Nome de associação, direção de leitura e papéis)

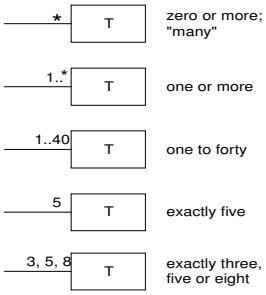


Associações – Cardinalidade (Multiplicidade)

- Especifica o número de objetos de cada classe envolvidos com a associação
- A leitura da cardinalidade é diferente para os diferentes sentidos da associação



Associações - Multiplicidade

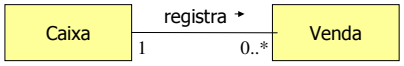


Multiplicidades

- Cada associação em um diagrama de classes possui duas multiplicidades, uma em cada extremo da linha de associação.

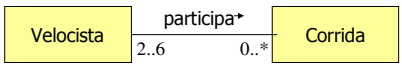
Exemplo (multiplicidade)

- A caixa pode registrar várias vendas
- Uma venda é registrada em somente uma caixa
- Pode haver uma caixa que não registra nenhuma venda



Exemplo (multiplicidade)

- 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 (a zero ou mais)



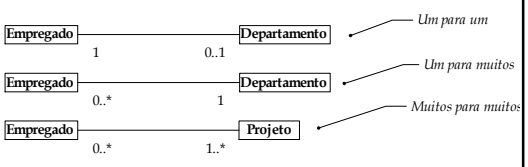
Conectividade

- 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 X Multiplicidade

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

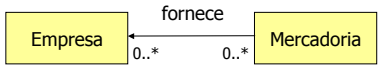
- Uma característica de uma associação que indica a necessidade (ou não) da existência desta associação entre objetos.
- A participação pode ser *obrigatória* ou *opcional*.
  - Se o valor mínimo da multiplicidade de uma associação é igual a 1 (um), significa que a participação é obrigatória
  - Caso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

Associações - Navegabilidade

- Mostra a direção de navegação, ou seja do canal de comunicação entre os objetos e classes.
- Uma associação é navegável da classe A para a classe B, se, dado um objeto de A, consegue-se obter de forma direta os objetos relacionados da classe B.
- É importante na visão de especificação e implementação -> visibilidade entre objetos, capacidade de um objeto de uma classe mandar mensagem a um objeto de outra classe

Associações - Navegabilidade

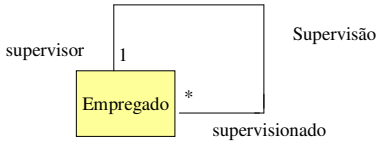
Dada uma mercadoria pode-se identificar diretamente a empresa fornecedora, mas a volta não é verdadeira



Associações reflexivas

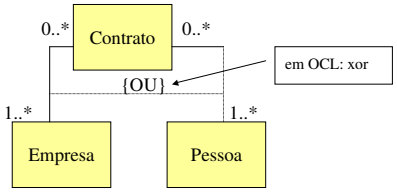
- Associa objetos da mesma classe.
  - Cada objeto tem um papel distinto na associação.
- A utilização de papéis é bastante importante para evitar ambigüidades na leitura da associação.
- Uma associação reflexiva *não* indica que um objeto se associa com ele próprio.
  - Ao contrário, indica que objetos de uma mesma classe se associam

Exemplo (associação reflexiva)



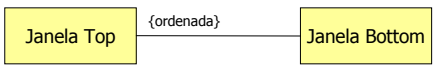
Associação Exclusiva

- Podem ser especificadas restrições em duas ou mais associações
- Na associação exclusiva objetos de uma classe podem participar de no máximo uma das associações ao mesmo tempo



Associação Ordenada

- Especifica uma ordem entre os objetos da associação. Ex: janelas de um sistema têm que ser ordenadas na tela (uma está no topo, uma está no fundo e assim por diante).

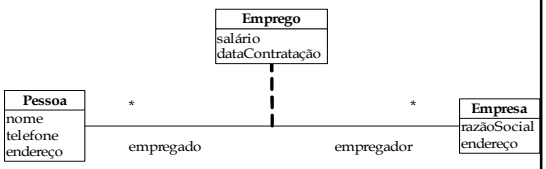


Classes de Associações (ou Associativas

- É uma classe que está ligada a uma associação, ao invés de estar ligada a outras classes.
- É criada quando duas ou mais classes estão associadas, e é necessário manter informações sobre esta associação que possui operações e métodos (portanto ela é uma classe)
- Uma classe associativa pode estar ligada a associações de qualquer tipo de conectividade.

Notação para uma classe associativa

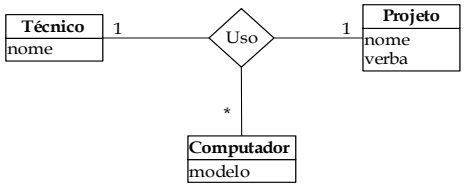
- *Representada pela notação utilizada para uma classe. A diferença é que esta classe é ligada a uma associação.*



Associações n-árias

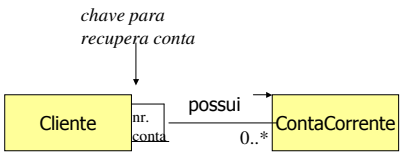
- São utilizadas para representar a associação existente entre objetos de n classes.
- Uma ***associação ternária*** é um caso mais comum (menos raro) de associação n-ária (n = 3).
- Na notação da UML, as linhas de associação se interceptam em um losango.

Exemplo (associação ternária)



Associação Qualificada

- Usadas com multiplicidades 1..* ou *. O qualificador (chave) identifica um objeto



2. Agregação/ Composição

- Casos particulares de associação
  - conseqüentemente, multiplicidades, participações, papéis, etc. podem ser usados igualmente
  - onde se puder utilizar uma agregação/composição, uma associação também poderá ser utilizada.
- Representam uma relação *todo-parte*
- Uma das classes é uma parte ou está contida em outra.

Agregação/Composição

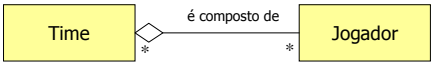
- Características particulares:
  - São assimétricas: se um objeto A é parte de um objeto B, B não pode ser parte de A.
  - Propagam comportamento, no sentido de que um comportamento que se aplica a um todo automaticamente se aplica as suas partes.
  - As partes são criadas e destruídas pelo todo, na classe do objeto todo, existem operações para remover e adicionar as partes

Como identificar

- Sejam duas classes associadas, X e Y. 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?*
- Palavras chaves: consiste em, contém, é parte de, tem, possui, é composta de, faz parte de, etc.

Agregação, características

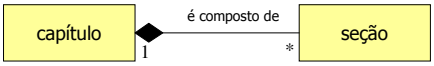
- A destruição de um objeto todo não implica necessariamente a destruição de suas partes
- Um objeto pode pertencer a mais de um composto, ou estar contido nele várias vezes
  - conhecida como agregação de compartilhamento (ou compartilhada)



Notação losango sem preenchimento

Composição, características

- A destruição de um objeto todo implica necessariamente a destruição de suas partes
- Uma classe pertence a um único composto (vive nele).
  - conhecida como agregação não compartilhada

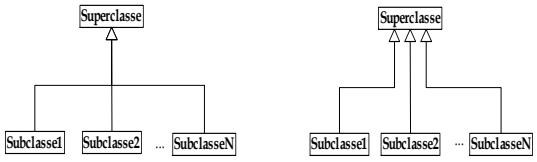


Notação losango com preenchimento

3. Generalização/Especialização

- Relacionamento entre uma classe geral e outra mais específica (de herança). A classe mais específica pode ser usada no lugar da mais geral e herda suas características
  - É como se as características da superclasse estivessem definidas também nas suas subclasses
- Representa o relacionamento é-um (is-a).

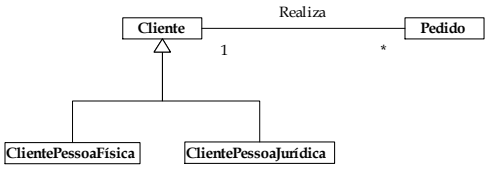
Notações para generalização



Outros nomes: classe base e derivada, classe mãe (pai) e filha; tipo e sub-tipo; pai e herdeira; geral e específica; ancestral e descentente

Herança de associações

- Atributos e operações e associações são herdados pelas subclasses.

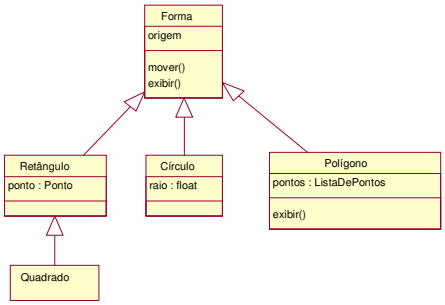


Generalização/Especialização

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.
  - Herança: "Gerentes são tipos especiais de funcionários".
  - Associação: "Gerentes chefiam departamentos".

Hierarquias de generalização



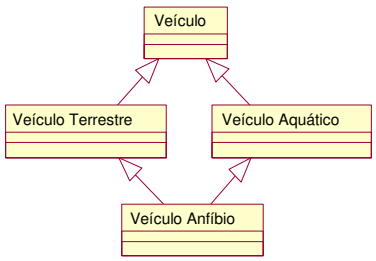
Generalização/Especialização

- 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.

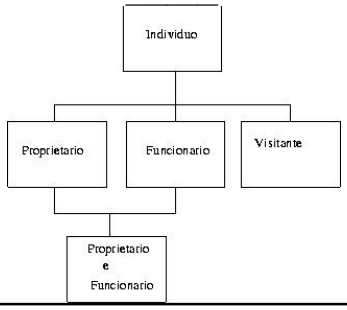
Herança múltipla

- ***Herança múltipla***: Uma classe pode ter mais de uma superclasse.
  - Tal classe herda de todas a suas superclasses.
- O uso de herança múltipla deve ser evitado.
  - Esse tipo de herança é difícil de entender.
  - Requer políticas para resolver conflitos de herança
  - Algumas LPs não dão suporte à implementação desse tipo de herança (Java e Smalltalk).

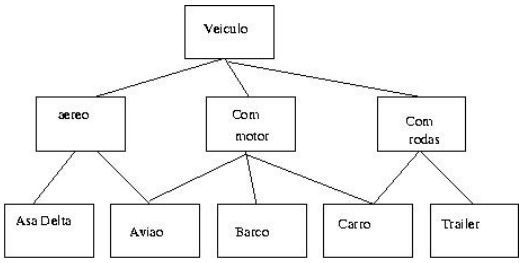
Exemplo (Herança múltipla)



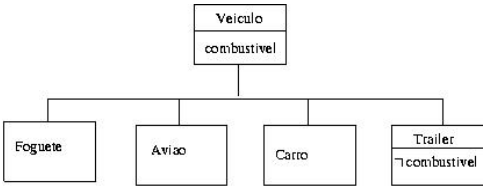
Herança Múltipla



Herança Múltipla (Entrelaçamento)



Herança Múltipla (Overriding)



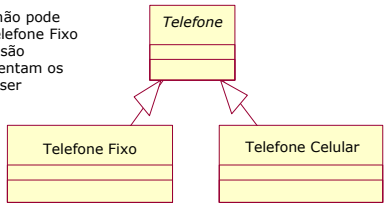
Classes abstratas e concretas

- Usualmente, a existência de uma classe se justifica pelo fato de haver a possibilidade de gerar instâncias (***classes concretas***).
- No entanto, podem existir classes que não geram instâncias diretas: ***classes abstratas***.
- 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.

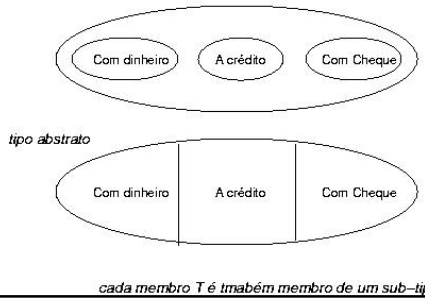
Notação para classes abstratas

- Na UML, uma classe abstrata é representada com o seu nome em *itálico*.

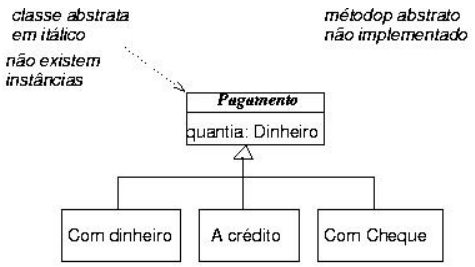
A classe Telefone não pode ser instanciada. Telefone Fixo e Telefone Celular são concretas, implementam os métodos e podem ser instanciadas.



Classes abstratas – Sistema Post



Classes abstratas – Sistema Post



Restrições sobre generalizações

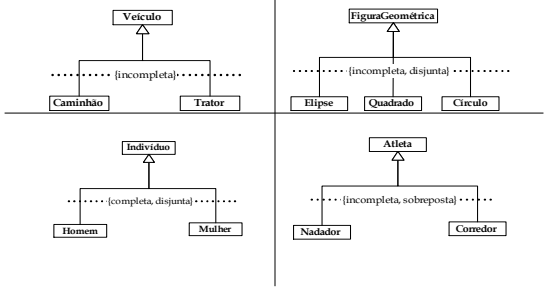
- Restrições sobre generalizações são representadas (entre chaves) no diagrama de classes próximas à linha do relacionamento.
- Restrições predefinidas pela UML:

– Sobreposta – Disjunta	– Completa – Incompleta

Restrições sobre generalização e na herança

- Sobreposição (overlapping): caso em que um objeto da superclasse pode pertencer simultaneamente a mais do que uma subclasse
- Disjuntiva (disjoint): superclasses podem se especializar em apenas uma subclasse
- Uma generalização está completa se já foram especificadas todas as sub-classes até aquele ponto e está incompleta se existir a possibilidade de uma outra especialização, caso em que um objeto da superclasse pode não pertencer a nenhuma das subclasses

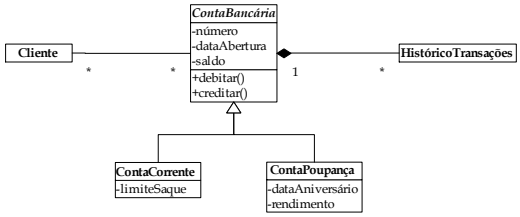
Exemplos (Restrições sobre generalizações)



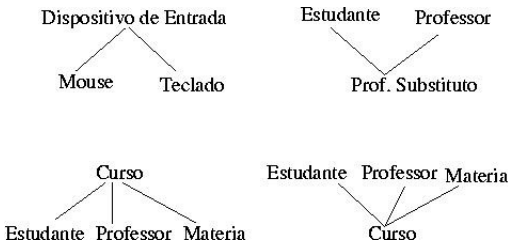
Como identificar

- O seguinte teste pode ser realizado para identificar se duas classes X e Y se relacionam por generalização: ***X é um tipo de Y?***
- ***Regra da substituição***: seja a classe A uma generalização de outra B. Não pode haver diferenças entre utilizar instâncias de B ou de A, do ponto de vista dos usuários de A.
  - Ou seja, é inadequado o uso de generalização onde nem todas as propriedades da superclasse fazem sentido para a subclasse.

Conformidade das subclasses à superclasse



Quais hierarquias são adequadas?

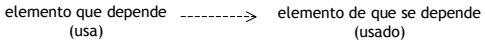


Dicas

- Deve-se evitar a construção de hierarquias de generalização muito profundas (com mais de três níveis)
  - dificultam a leitura do diagrama.
- Papéis e subclasses não devem ser confundidos.
  - Um papel corresponde ao uso de uma certa classe em uma associação.Uma classe pode assumir vários papéis.
  - Deve-se evitar a criação de subclasses em situações que podem ser resolvidas através da utilização de papéis.

4. Dependências

- Conexão entre dois elementos, representando que uma mudança no elemento independente afeta o dependente.



5. Refinamentos

- Relacionamentos entre duas descrições de uma mesma coisa, mas em níveis de abstração diferentes
- São usados em modelos de coordenação, ou seja, modelos que mostram como os modelos de diferentes fases se relacionam

Identificação dos Relacionamentos

- Na perspectiva conceitual representam-se relacionamentos conceituais
- As associações são estabelecidas analisando-se os papéis.
- A generalização representa a hierarquia entre tipos e seus sub-tipos
- A agregação entre um todo e suas partes.

Perspectiva Conceitual Associações

- Indicam conhecimento de um relacionamento que precisa ser preservado durante algum tempo
  - Duração de milisegundos ou anos, dependendo do contexto

Associações Típicas

Categoria	Exemplos
A é uma parte física de B (*)	Gaveta - POST Asa - Avião
A é uma parte lógica de B (*)	Item de Venda - Venda Escala - Vôo
A está fisicamente contido em B (*)	POST - Loja Passageiro - Avião
A está logicamente contido em B (*)	Descrição-Item - Catálogo Vôo - Roteiro de Viagem
A é uma descrição de B	Descrição-Item - Item Descrição-Vôo - Vôo
A é um item de uma transação ou relatório B	Item de Venda - Venda Opção de Reserva - Reserva
A é conhecido/registrado/reportado/capturado em B (*)	Venda - POST Reserva - Terminal de Reserva

(*) Alta prioridade

Associações Típicas

Categoria	Exemplos
A é um membro de B	Operador - Loja Piloto - Companhia Aérea
A é uma sub-unidade organizacional de B	Departamento - Loja Manutenção - Companhia Aérea
A usa ou gerencia B	Operador - POST Piloto - Avião
A se comunica com B	Cliente - Operador Agente de Reserva - Passageiro
A está relacionado com uma transação B	Cliente - Pagamento Passageiro - Bilhete
A é uma transação relacionada com outra transação B	Pagamento - Venda Reserva - Cancelamento
A é possuído por B	POST - Loja Avião - Companhia Aérea

Identificando Associações

- Regras úteis:
  - Focar nas associações cujo conhecimento deve ser preservado
  - Usar nomes baseados em expressões verbais que façam sentido quando lidas no contexto do modelo
  - Evitar mostrar associações deriváveis ou redundantes
  - É mais importante identificar conceitos do que associações
  - Associações demais tendem a confundir um modelo ao invés de iluminá-lo

Adicionando Associações ao Modelo Conceitual do Sistema POST

- Relacionamentos fundamentais
  - Venda *Capturada-em* POST
    - Para conhecer a venda corrente, calcular total e imprimir recibo
  - Venda *Paga-por* Cliente
    - Para saber se a venda foi paga, calcular troco, e imprimir recibo
  - Catálogo-Produto *Contém* Especificação-Item
    - Para obter a especificação de um item, dado um UPC

Aplicando o Checklist

Categoria	Exemplos
A é uma parte física de B	N.A.
A é uma parte lógica de B	Item de Venda - Venda
A está fisicamente contido em B	POST - Loja Item - Loja
A está logicamente contido em B	Especificação-Produto - Catálogo Catálogo - Loja
A é uma descrição de B	Especificação-Produto - Item
A é um item de uma transação ou relatório B	Item de Venda - Venda
A é conhecido/registrado/reportado/capturado em B	Venda (corrente) - POST Venda (completada) - Loja

Aplicando o Checklist

Categoria	Exemplos
A é um membro de B	Operador - Loja
A é uma sub-unidade organizacional de B	N.A.
A usa ou gerencia B	Operador - POST Gerente - POST
A se comunica com B	Cliente - Operador
A está relacionado com uma transação B	Cliente - Pagamento Operador - Pagamento
A é uma transação relacionada com outra transação B	Pagamento - Venda
A é possuído por B	POST - Loja

Eliminando Associações Redundantes

Associação	Consideração
Venda *Iniciada-por* Operador	Conhecimento não exigido nos requisitos; derivável da associação Operador *Registra-vendas-em* POST.
Operador *Registra-vendas-em* POST	Conhecimento não exigido nos requisitos.
POST *Inicializado-por* Gerente	Conhecimento não exigido nos requisitos.
Venda *Iniciada-por* Cliente	Conhecimento não exigido nos requisitos.
Loja *Armazena* Item	Conhecimento não exigido nos requisitos.
Item de Venda *Registra-venda-de* Item	Conhecimento não exigido nos requisitos.

Preservando Associações de Compreensão

- Preservar apenas associações de conhecimento pode resultar num modelo que não transmite um completo entendimento do domínio
  - Ex.: Venda *Iniciada-por* Cliente
    - Remoção deixa de fora um aspecto importante do domínio— o fato de que um cliente gera uma venda
- Regra geral:
  - Enfatizar associações de conhecimento, mas preservar associações que enriquecem o entendimento do domínio

Identificação dos Relacionamentos

- Na perspectiva de implementação representa um canal de comunicação entre duas classes
- A necessidade desse canal é dada pelos diagramas de interação. Sempre que existir uma mensagem trocada entre dois objetos nesses diagramas existirá uma associação entre eles, que representa as responsabilidades das classes.

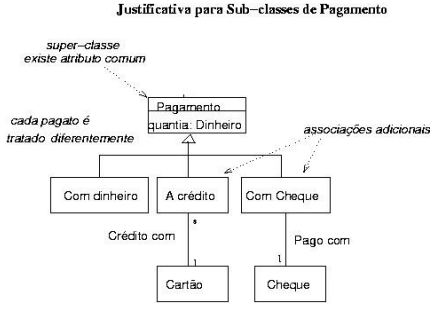
Identificando Generalizações

- Quando particionar em Sub-Classes
- a sub-classe tem atributos adicionais de interesse
- a sub-classe tem associações adicionais de interesse
- a sub-classe será manipulada ou usada de maneira diferente da super-classe ou das outras sub-classes
- a sub-classe se comporta diferente da super-classe ou das outras sub-classes

Identificando Generalizações

- Quando criar uma Super-Classe
- as potenciais sub-classes representam variações de um conceito similar
- as sub-classes satisfazem 100% a regra
- ``is-a"
- todas as sub-classes possuem um atributo comum que poderá ser expresso na super-classe
- todas as sub-classes possuem uma associação comum que poderá ser relacionada à super-classe

Identificando Generalizações



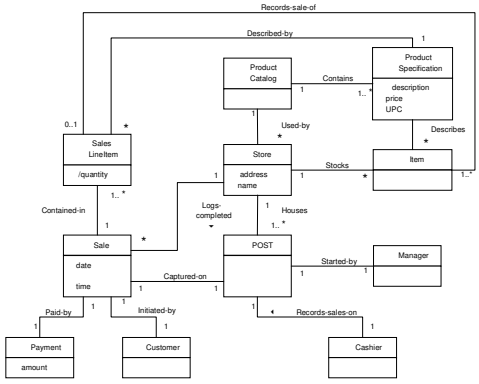
Identificando Agregações

- Verificar se algumas classes podem ser agrupadas em algum composto:
  - elas são geralmente criadas/destruídas no mesmo instante.
  - possuem relacionamentos comuns

Identificando Agregações

- Verificar se alguma classe pode ser sub-dividida em partes.
  - as partes possuem tempo de vida limitado ao tempo de vida do composto
  - possuem relacionamentos particulares e de interesse

Exemplo de um modelo conceitual



Referências

- Boock, G. and Rumbaugh, J. *The Unified Modeling Language User Guide* . Addison-Wesley, 1999
- Arlow, J. and Neustadt, I. UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design, 2nd Edition, The Addison-Wesley Object Technology Series, 2005.
- Rumbaugh, J.; Jacobson, I. and Booch , G. The Unified Modeling Language Reference Manual, 2nd Edition, The Addison-Wesley Object Technology Series, 2004.
- Boock, G.; Rumbaugh, J. and Jacobson, I; Unified Modeling Language User Guide, 2nd Edition, The Addison-Wesley Object Technology Series, 2005.
- Jacobson, I; Boock, G. and Rumbaugh, J., Unified Software Development Process, Addison-Wesley, Janeiro 1999.
- Larman, C. *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design* Prentice-Hall, New Jersey - USA, 1997
- Bezerra, E. *Princípios de Análise e Projeto com a UML*, ed. Campus-Elsevier. 2003.