Diagramas de Classes

Metodologias e Desenvolvimento de Software

Pedro Salgueiro
pds@uevora.pt
CLV-256

Diagramas de classes

- Diagramas de classes
  - Tipo de diagrama muito usado
    - O mais usado
  - Muitos conceitos de modelação

- Conceitos base
  - O essencial
  - Usado em todos os tipo de diagrama de classes

- Conceitos avançados
  - Nem sempre são necessários/usados

Diagramas de classe

- Processo de desenvolvimento de software
  - Análise/levantamento de requisitos;
  - **Análise conceptual;**
  - Desenho;
  - Implementação;
  - Testes;

Diagramas de classe

- Análise conceptual
  - use cases (ponto de vista utilizador)
  - **classes e relações (estrutura dos “dados”)**
  - interações (modelação dos processos)
  - *constraints* (restrições, regras de negócio)
  - **Não** envolve
    - programação
    - implementação física
    - modelo relacional

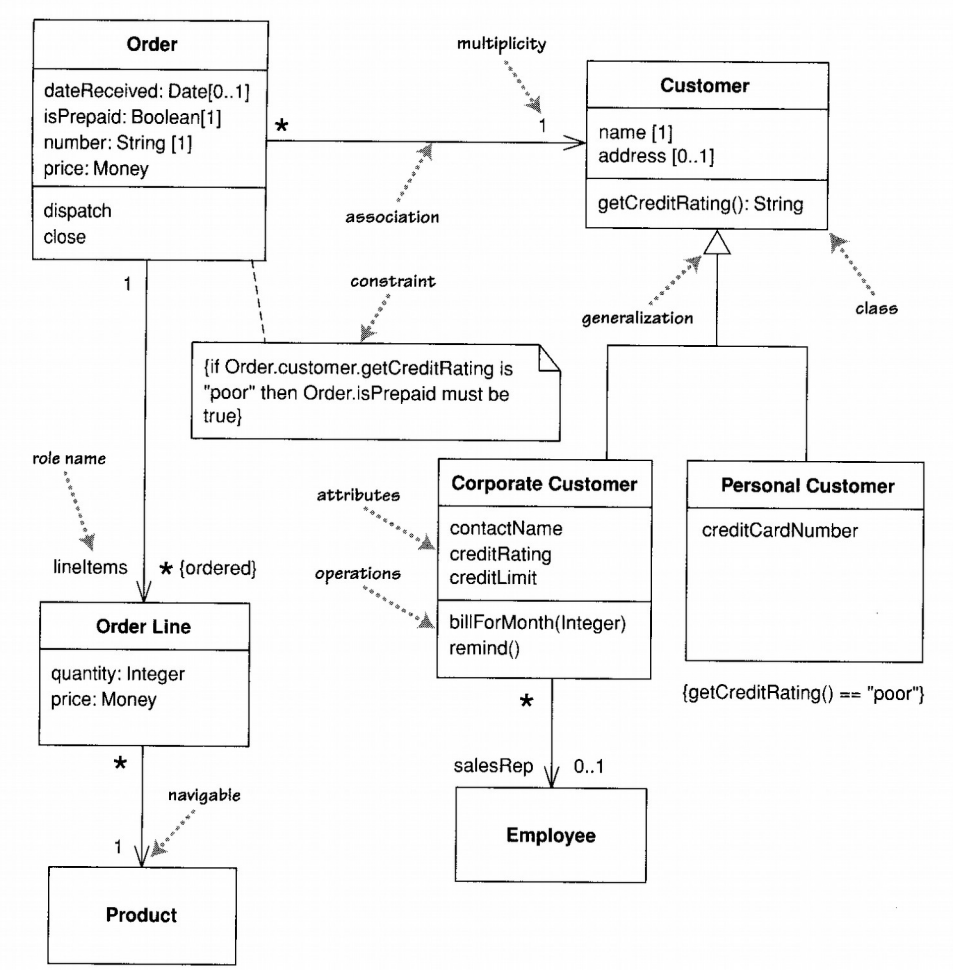
Diagramas de classe

- Diagramas de classe
  - São os mais “fáceis” de apreender
  - Descrevem as “coisas” que o problema envolve
  - ”Correspondem” à análise entidade-relação clássica

Diagramas de classes

- Diagramas de classes
  - Classes
    - “Coisas” do problema
  - Descrevem
    - Tipos de objetos num sistema
    - Relações que existem entre objetos
  - Indica
    - Propriedades das classes
    - Operações dos objetos

Diagramas de classes

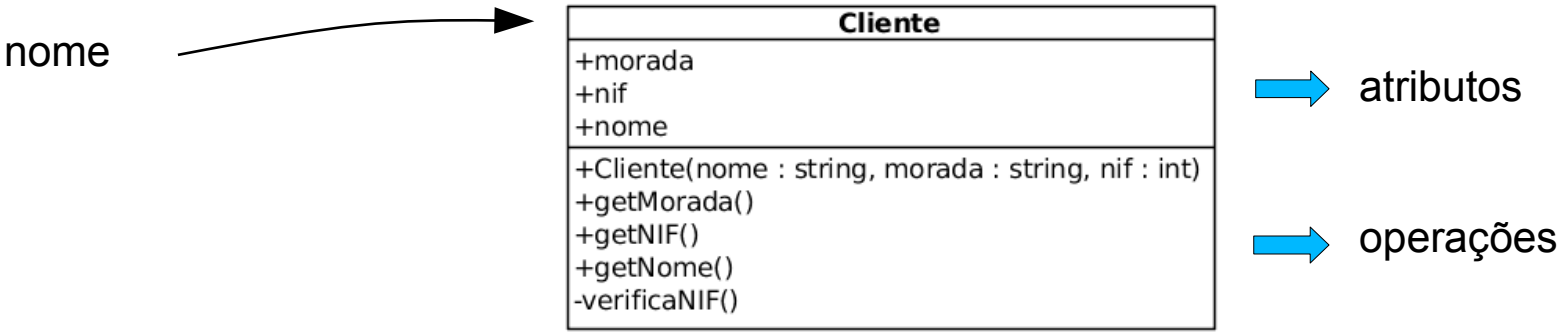


Diagramas de classes

- Diagramas de classes
  - Classes
  - Propriedades
  - Atributos
  - Associações
  - Multiplicidade
  - Operações
  - Generalizações
  - Notas e comentários
  - Dependências
  - Restrições

Classes

- Classes
  - “Coisas” do problema
  - Representação
    - “caixa” com 1, 2 ou 3 secções
      - identidade / nome
      - atributos
      - operações / métodos



Classes

- Como “descobrir” as classes para um problema
  - podemos procurar:
    - os substantivos do “discurso”
    - refletem **entidades** do mundo real
      - livros, revistas, leitores, títulos, cópias, bibliotecário, …
    - capturam o vocabulário do sistema
    - definem as “fronteiras” do problema a resolver
  - “coisas” com “persistência”
    - mais que os processos
    - são os blocos base do sistema

Classes

- Problema exemplo: Biblioteca
  - “pretende-se um sistema que suporte o funcionamento de uma biblioteca”;
  - “a biblioteca empresta livros e revistas, constantes da “base de dados” da biblioteca a leitores, que se registam no sistema”;
  - “novos livros são adquiridos pela biblioteca; títulos muito pedidos são comprados em várias cópias; livros e revistas antigos são “abatidos” quando estão obsoletos ou em más condições”;
  - “o bibliotecário atende os leitores e recorre ao sistema para o seu trabalho: inserção de novos livros e revistas, abate, empréstimos, etc”;
  - “um leitor pode reservar um livro ou revista que não esteja disponível num dado momento; logo que esteja disponível ou tenha sido adquirido o leitor é notificado”;
  - “esta reserva é cancelada quando o documento é emprestado ao leitor ou se ele desiste explicitamente”;
  - “o sistema deverá poder ter evoluções futuras”;
  - “o sistema, na versão inicial, não trata das relações com as livrarias e editoras”;

Classes

- Classes
  - “coisas” do problema:
    - “a biblioteca empresta livros e revistas, constantes da “base de dados” da biblioteca a leitores, que se registam no sistema”;
    - “novos livros são adquiridos pela biblioteca; títulos muito pedidos são comprados em várias cópias”;

Classes

- Classes
  - “coisas” do problema:
    - “a biblioteca empresta **livros** e **revistas**, constantes da “base de dados” da biblioteca a **leitores**, que se registam no sistema”;
    - “novos **livros** são adquiridos pela biblioteca; **títulos** muito pedidos são comprados em várias cópias”;

Classes

- bibliotecário?
  - interessa para o meu sistema?
  - sim, é um ator importante:
    - é ele quem usa o sistema…
  - vou guardar informação sobre ele?
    - depende
    - preciso?
      - "dará" origem a uma classe
    - não preciso?
      - não "dará" origem a uma classe

Propriedades

- Elementos estruturais de uma classe
- “Tipicamente” são “campos/atributos” de uma classe
- Conceito único
  - Duas notações distintas
- Possíveis notações:
  - Atributos
  - Associações
  - Embora pareçam diferentes, são o mesmo

Atributos

- Notação para descrever uma propriedade
  - Representados de forma textual
  - Na caixa que representa a classe
- Descrever propriedades da classe
  - Exemplo: nome, morada, telefone
- Sintaxe

```
visivility name: type multiplicity = default {property-string}
```

- Exemplo

```
- name: String [1] = "Untitled" {readOnly}
```

Atributos

- **Exemplo:**

```
name: String [1] = "Untitled" {readOnly}
```

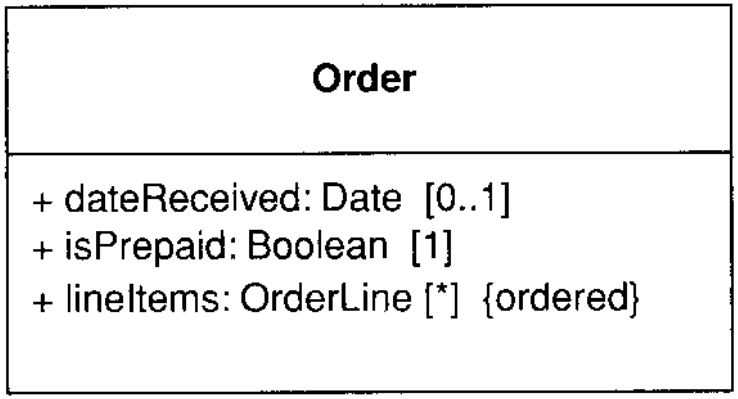
- **Sintaxe:**

```
visivility name: type multiplicity = default {property-string}
```

  - **`visibility:`** Indica se o atributo é publico (+) ou privado (-)
  - **`name:`** Nome do atributo. Normalmente corresponde a um campo/atributo da classe numa liguagem de programação
  - **`type:`** Tipo de objeto que pode ser "guardado" no atributo. Normalmente corresponde ao tipo de dados de um atributo numa linguagem de programação
  - **`multiplicity:`** quantos objectos
  - **`default:`** valor default para quando o objeto é criado sem especificar qual o valor do atributo
  - **`{property-string}:`** propriedades adicionais. Se omitido, assume-se que o atributo pode ser alterado

Atributos

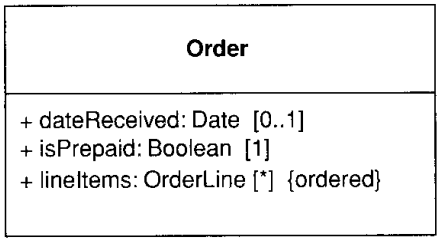
Representação

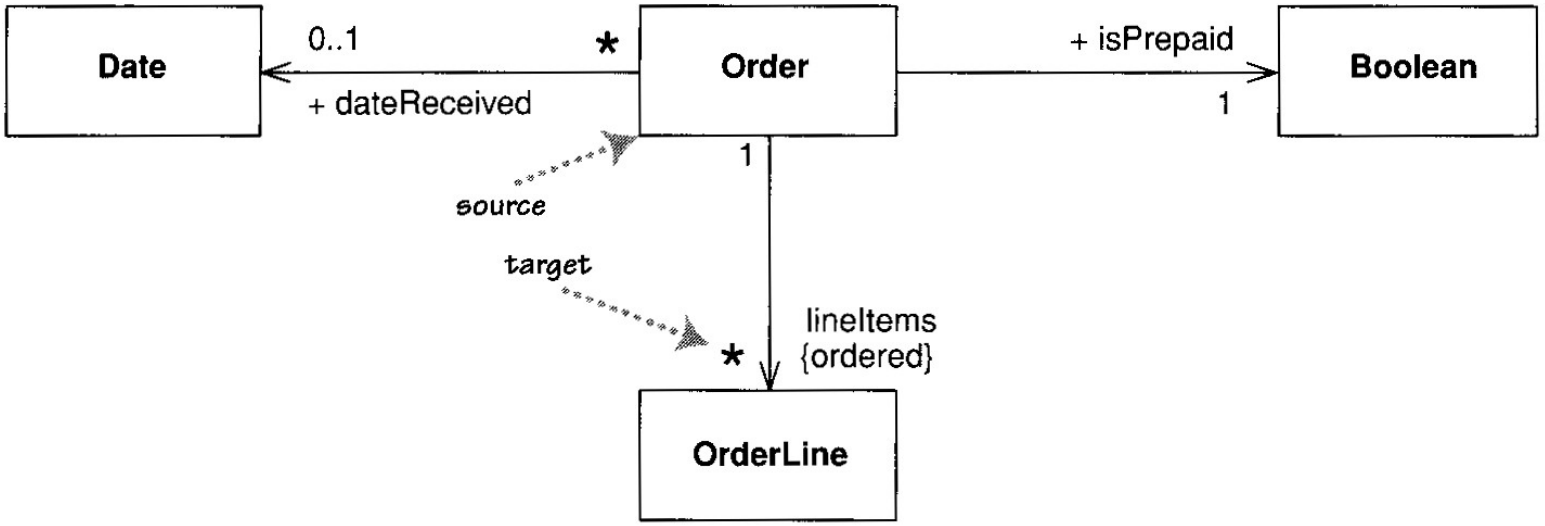


Associações/Relações

- Notação para representar propriedades
  - “Alternativa” aos atributos
  - Informação dos atributos pode ser incluída nas associações
- Representam relações/ligações entre classes
- Linha/Arco dirigido
  - Liga duas classes
    - source (origem)
    - target (destino)
    - propriedade
    - multiplicidade

Exemplo


Order
+ dateReceived: Date [0..1]
+ isPrepaid: Boolean [1]
+ lineItems: OrderLine [*] {ordered}


Date
0..1
*
+ dateReceived
Order
+ isPrepaid
1
Boolean
1
source
target
*
lineItems
{ordered}
OrderLine

Associações/Relações

Como encontrar relações

- "empresta livros ... a leitores";
- "um leitor pode reservar uma revista";

- as classes relacionam-se:
  - leitor - livro
  - leitor – revista

- substantivos explicados por duas classes:
  - um empréstimo envolve "sempre" um leitor e um livro

- ou corresponde a verbos do discurso:
  - Reservar - leitor reserva revista

Associações/Relações

Outra forma de encontrar relações:

- É necessário “navegar” de uma classe para outra?
- Uma classe interage com outra?
- Equivalente?
  - sim

Associações

- Podem ser entre classes
  - professor lecciona cadeira
  - fábrica produz produtos
  - pessoa trabalha num departamento
- Pode ser recursivas (sobre a própria classe)
  - mapa de uma cidade: “rua inicia-se e termina numa rua”
  - disciplina: “precedências”

Atributos e Associações

- Quando usar Atributos ou Associações?
  - Na teoria: são “equivalentes”
- Regra geral
  - Usar atributos:
    - Tipos de dados simples
    - Datas, booleanos, inteiros, etc…
  - Usar associações:
    - Tipos de dados complexos
    - Classes importantes do problema
    - Exemplo: cliente, encomenda, etc...

Multiplicidade

- Diz respeito a uma propriedade
  - Multiplicidade de uma propriedade
- Indica quantos objetos podem pertencer uma propriedade
- Multiplicidades mais comuns
  - **1 :** (uma encomenda deve ter exatamente um cliente)
  - **0..1 :** (um cliente pode ou não ter um representante de vendas)
  - *** :** (um cliente não precisa de fazer uma encomenda e não existe um limite para o nº de encomendas que o cliente pode fazer – zero ou mais encomendas)

Multiplicidade

- Possibilidades
  - Limite inferior
    - >= 0
  - Limite superior
    - >0 **ou** *(ilimitado)
  - Limite inferior == limite superior
    - Pode usar-se apenas um nº: 1 .. 1 → 1
  - Caso especial
    - * → 0 .. *
  - Exemplo:
    - **2..4** (Nº de jogadores num jogo de canasta)

Multiplicidade

- Nos atributos
  - Opcional
    - Limite inferior: 0
  - Obrigatório
    - Limite inferior: >= 1
  - *Singlevalued*
    - Limite superior: 1
  - *Multivalued*
    - Limite superior: >1, normalmente *

Multiplicidade

- Atributos *multivalued*
  - Representam um conjunto (set)
  - Ordenação
    - Default: não ordenado
      - Ex: “Listar encomendas” → encomendas não ordendas
      - {unordered}
      - {bag}
    - Ordem relevante
      - `{ordered}`
  - Duplicados
    - Default: não permitido
      - {unique}
    - permitido
      - {nonunique}
      - {bag}
- Multiplicidade *default*:
  - 1
  - Cuidado: o diagrama pode omitir a multiplicidade

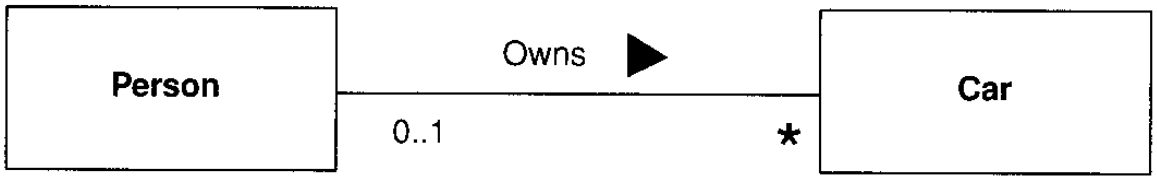
Associações bidirecionais

- Duas propriedades relacionadas
  - "seguir" as propriedades em qualquer sentido
- Usar duas **setas** ou **nenhuma**

- Notação dos atributos
  - Classe `Car`
    - Tem atributo `owner:Person[1]`
  - Classe `Person`
    - Tem atributo `cars:Car[*]`

Associações bidirecionais

- Representação alternativa
- Aplicar um *label*
  - Verbo ou frase
  - Permite usar a associação em frases/textos



Operações

- Ações feitas por uma classe
- Métodos de uma classe
- Ficam de parte (normalmente)
  - Métodos que “apenas” manipulam propriedades
    - Podem ser inferidos
- Sintaxe
  - `visibility name (parameter-list) : return type {property-string}`

Operações

- Sintaxe
  - `visibility name (parameter-list) : return type {property-string}`
  - `visibility`: Indica se a operação é publica (+) ou privada (-)
  - `name`: Nome do operação.
  - `parameter-list`: lista de parametros da operação
  - `return type`: tipo de retorno da operação
  - `{property-string}`: propriedades adicionais da operação

Operações

- Sintaxe
  - `visibility name (parameter-list) : return type {property-string}`
- Parâmetros: `(parameter-list)`
  - Notação semelhante à dos atributos:
    - `direction name: type = default value`
    - `name, type, default value:` igual aos atributos
    - `direction:` input(in), output(out) ou ambos (inout). Se omitido assume-se input.
  - Exemplo

```
+ balanceOn (date:Date) : Money
```

Operações

- Sintaxe
  - `visibility name (parameter-list) : return type {property-string}`
- `property-string`
  - propriedades adicionais
  - “tipo” da operação
- tipos
  - *query* – operações que obtêm um valor da classe, sem fazer alterações da classe
  - *modifiers* – operações que mudam o “estado” da classe
  - *setting/setters* – operações que apenas atribuem um valor a um atributo de uma classe
  - *getting/getters* – operações que apenas lêem um valor a um atributo de uma classe

Generalizações

- “Coisa mais genérica que outra”
- Relação entre “coisas” com
  - **algumas diferenças e**
  - **muitas semelhanças**

- Exemplo
  - Cliente (classe geral)
    - Cliente pessoal (classe específica)
    - Cliente empresarial (classe específica)
  - Uma classe geral
    - superclasse
    - classe “pai”
  - Várias classes específicas
    - Classe “filho”
    - subtipos

Generalizações

- Herança
  - Classe “filho” herda todas as propriedades da classe “pai”
  - *override(sobrescrever)* de qualquer propriedade
    - Atributos
    - Operações
- Princípio de utilização
  - Substituibilidade
    - Classe “filho” poder substituir a classe “pai”
    - Um “Cliente Pessoal” é um “Cliente”
    - Posso substituir um “Cliente” por um “Cliente Pessoal”

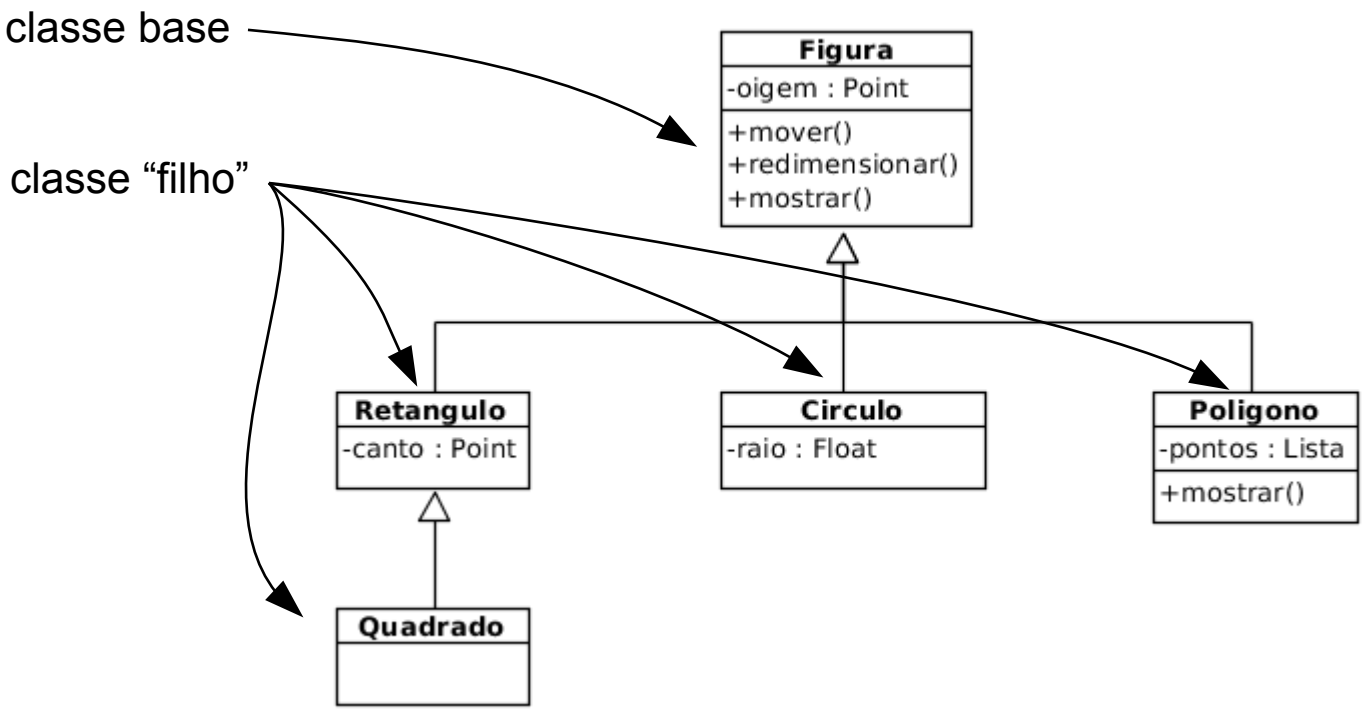
Generalizações

- Forma de “sub-typing”
  - Não única
  - Outra forma → *interfaces*
- A é subtipo de B
  - Se A pode substituir B
  - A subclasse de B

Generalizações

- Alguns exemplos
  - baleia é mamífero que vive no mar;
  - pinheiro é um tipo de árvore;
  - circulo e retângulo são figuras geométricas;

Generalizações



Generalizações

- Generalizações (herança)
  - simples
    - pode originar polimorfismo
    - a classe geral pode ser abstrata
      - i.e.: não pode ser instanciada
      - pessoa → pessoa administrativa; docente; aluno
  - múltipla
    - um veículo anfíbio :
      - é um veículo motor: herda aspectos da classe veículo;
      - é um “barco”: herda aspectos da classe barco;

Generalizações

- Generalizações (herança)
  - completa / incompleta
    - completa: as sub-classes constituem o universo
      - Pessoa → Homem, Mulher
    - incompleta: podem surgir mais sub-classes posteriormente
      - O mais comum

Generalizações

Como descobrir generalizações/herança?

- Classes com atributos ou operações comuns;
- Com responsabilidades comuns;
- “Substantivos diferentes” na descrição do problema usados indiferentemente em certos processos
  - “… um aluno identifica-se com o BI”;
  - “… um professor identifica-se com o BI”;

Generalizações

Como descobrir generalizações/herança?

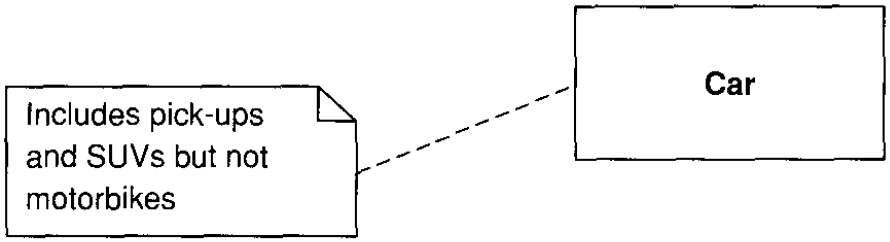
- Empregado Administrativo e Empregado Técnico
  - têm atributos comuns:
    - Número de Empregado
    - BI
    - Nome
    - Data de nascimento
- "Mecanicamente" pode fazer-se uma generalização para Empregado; é natural...;
  - mas não será uma associação com "tipo_de_tarefa"?;
  - se uma pessoa puder ser simultaneamente duas particularizações deve pensar-se numa associação!!!

Como descobrir generalizações/herança?

- Se existe uma conjunto de atributos comuns a várias “classes”
  - **não usar herança** se a distinção é feita por **valores** de outros atributos
- Classe geral: paredes de um edifício
  - *sub-classe 1: paredes_de_alvenaria*
  - *sub-classe 2: de aglomerado_de_madeira*
- Talvez fosse melhor **só** a classe **paredes** com atributo **material**
  - o domínio deste atributo seria:
    - “paredes_de_alvenaria”, “aglomerado_de_madeira”

Notas e comentários

- Comentar diagramas diagramas
  - Qualquer tipo diagrama
- Podem ser
  - Isolados
    - Dizem respeito ao diagrama todo
  - Relacionados com um outros elementos
    - “ligação” com uma linha tracejada



Dependências

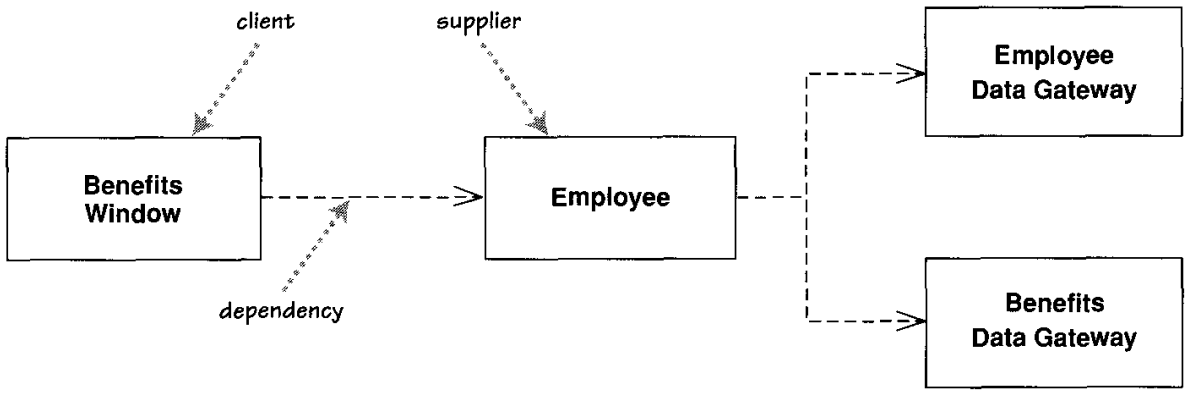
- Dependências entre dois elementos
  - Um “cliente” e um “fornecedor”
  - Se a alteração da definição de um elemento (“fornecedor”)
    - implicar alterações de outro elemento (“cliente”)
- Quando pode existir dependência
  - Uma classe envia uma “mensagem” a outra;
  - Uma classe “tem” outra classe como parte dos seus dados;
  - Uma classe “menciona” outra através de parâmetros ou atributos;
  - Se uma classe muda o seu interface, qualquer classe que “comunica” com ela, pode deixar de conseguir comunicar;

Dependências

- Importância
  - Crescimento do sistema
    - Maior complexidade
    - Maior rede de dependências
    - Gestão/controlo de dependências
      - Previne “alguns” problemas inerentes à alteração dos sistemas
- Em UML
  - Dependências entre qualquer tipo de elementos
  - Usadas para mostrar quando alterações num elemento, podem provocar alterações noutro elmento

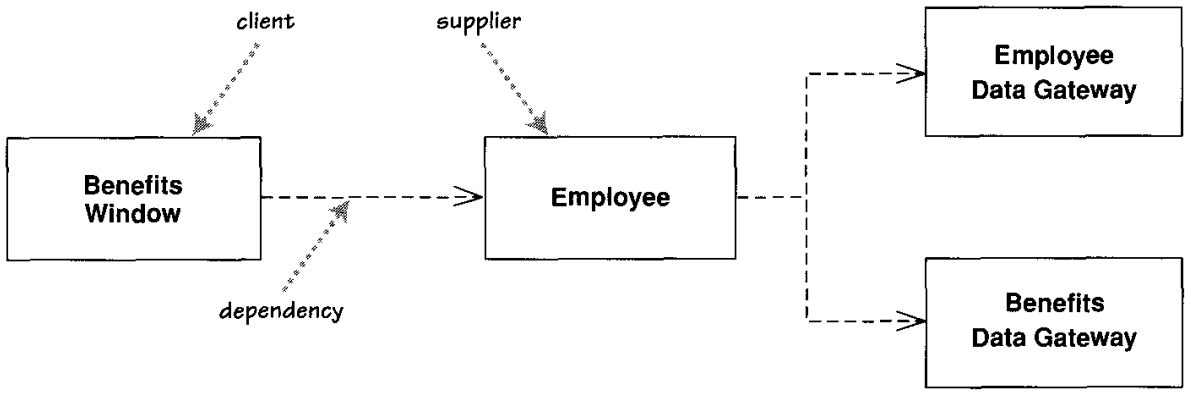
Dependências

Exemplo



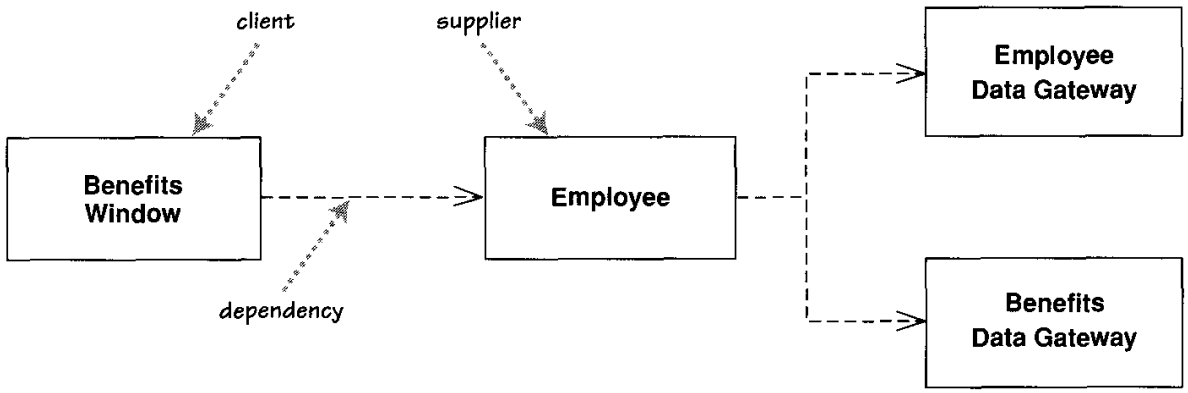
- "Benefits Window" → Interface do sistema, ou classe de "apresentação"
- "Employee" → classe que representa os empregados, representa o comportamento essencial do sistema

Exemplo



- "Benefits Window" depende de "Employee"
  - Se "Employee" for alterada → "Benefits Window" tem que ser alterada
  - Dependência unilateral
    - Se "Benefits Window" for alterada → "Employee" não necessita de ser alterada

Exemplo



- "Employee" depende de "Employee Data Gateway" e "Benefits Data Gateway"
- O que acontece se "Employee Data Gateway" ou "Benefits Data Gateway" forem alterados?
  - Apenas o "Employee" necessita de ser alterado

Dependências

- Implícitas em
  - Associações “navegáveis”
  - Generalizações

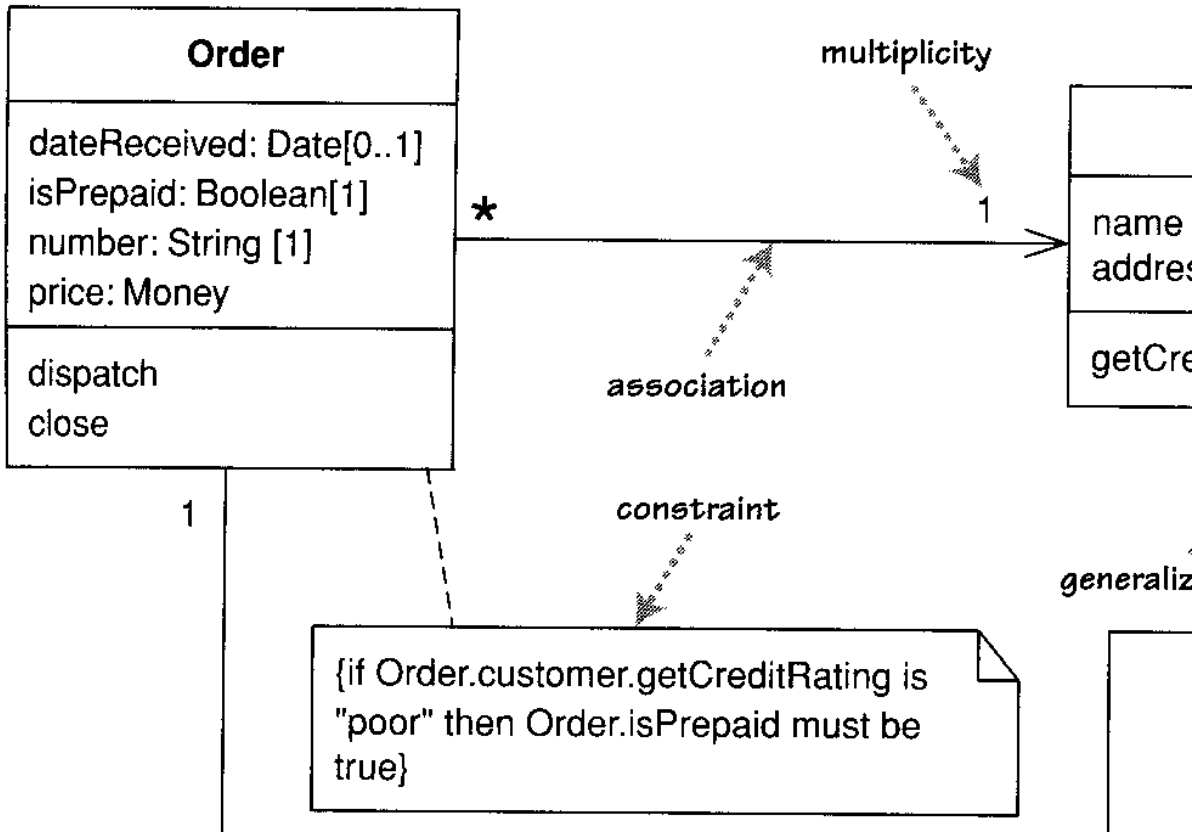
- Regras
  - Minimizar dependências
  - Cuidado com dependências cíclicas
  - Incluir/“mostrar” apenas dependências importantes nos diagramas

Restrições

- Diagramas de Classes
  - Estrutura do problema
  - Especificam restrições globais do sistema
    - Que elementos, como são compostos, quais as suas relações
    - Não conseguem especificar todas as restrições

- Restrições (Constraint Rules)
  - Qualquer tipo de restrição sobre o sistema
    - **Linguagem natural**
    - Linguagem de programação
    - UML Object Constraint Language (OCL)
  - Representado como uma “nota”
    - Restrição coloca-se entre { }

Restrições


Order
dateReceived: Date[0..1]
isPrepaid: Boolean[1]
number: String [1]
price: Money
dispatch
close
multiplicity
*
1
association
name
addres
getCre
1
constraint
generaliz
{if Order.customer.getCreditRating is "poor" then Order.isPrepaid must be true}

Diagramas de classes

- Conceitos base
  - 90% da utilização
- Restantes 10%
  - Inúmeros conceitos que podem ser necessários

Diagramas de classes

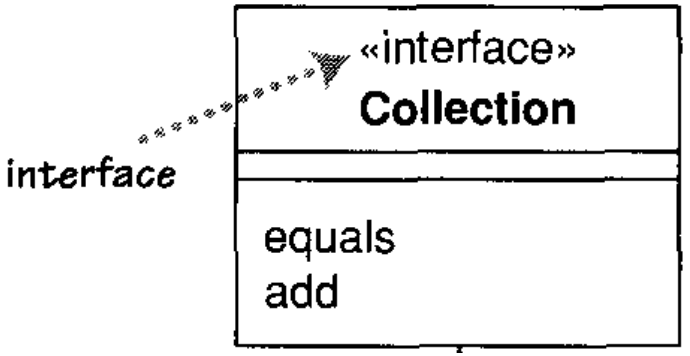
Keywords

- Linguagens gráficas
  - Símbolos gráficos
  - Conhecê-los e saber o seu significado
  - Problema
    - Mesmo símbolo com significados diferentes

- *Keywords*
  - Distinguir diferentes tipos de utilização para a mesma entidade gráfica

Keywords

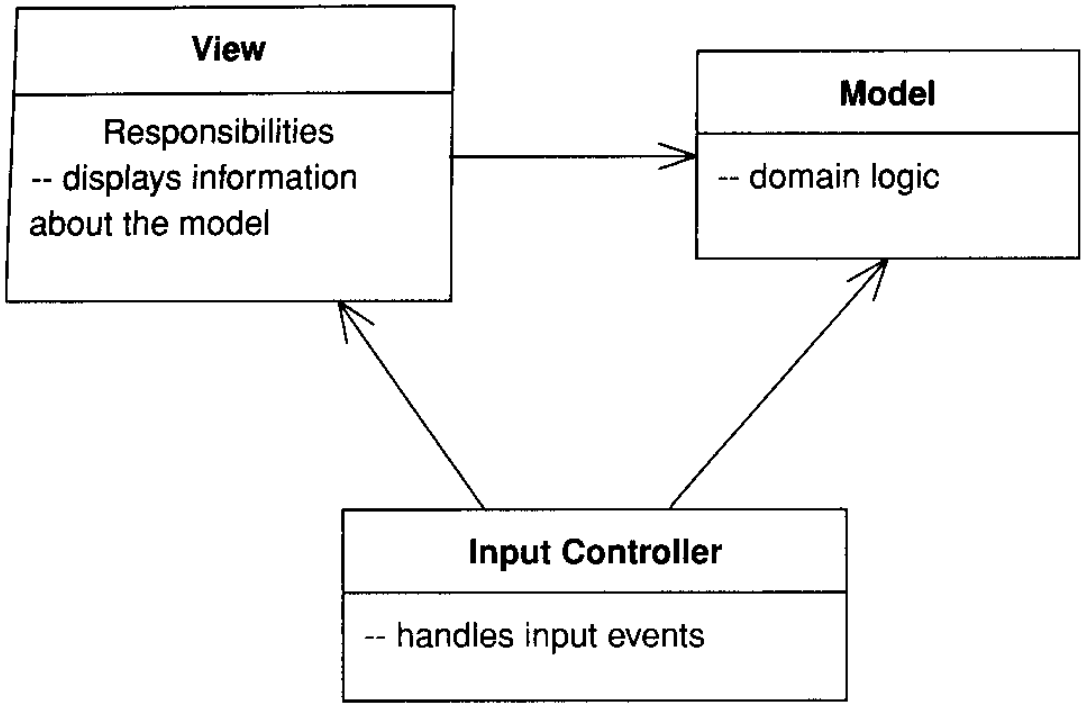
- Colocadas entre
  - « » (aspas angulares)
  - { }
- Qual usar
  - Regras pouco definidas
- Exemplo
  - «Interface»
  - {Abstract}



Responsabilidades

- Responsabilidade
  - Algo que um objeto/entidade deve fazer
  - É responsável por fazer determinada operação/tarefa
- Representação
  - “Compartimento” especial nas classes
    - Com o nome “Responsabilidades” ou “Responsabilities” (opcional)
  - Descrição textual da responsabilidade

Responsabilidades

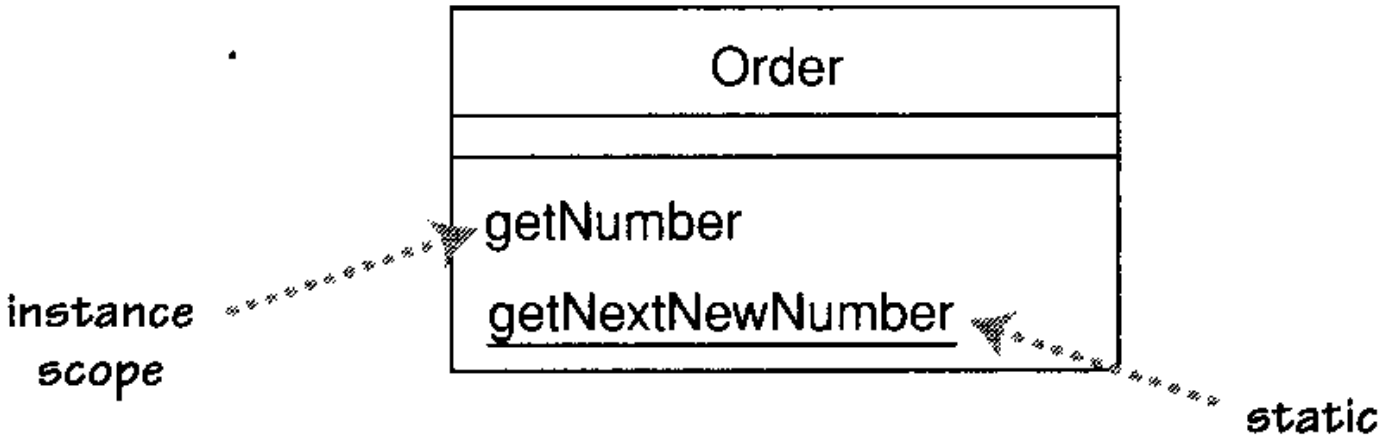


Operações e atributos estáticos

- Operações e atributos de classe “normais”
  - Aplicam-se a uma instância/objeto da classe
  - Dependem do objeto
  - Não fazem sentido sem um objeto

- Operações e atributos de classe estáticos
  - Referem-se à classe e não aos objetos
  - São independentes dos objetos
  - Diagrama de classes
    - Representados com um sublinhado

Operações e atributos estáticos

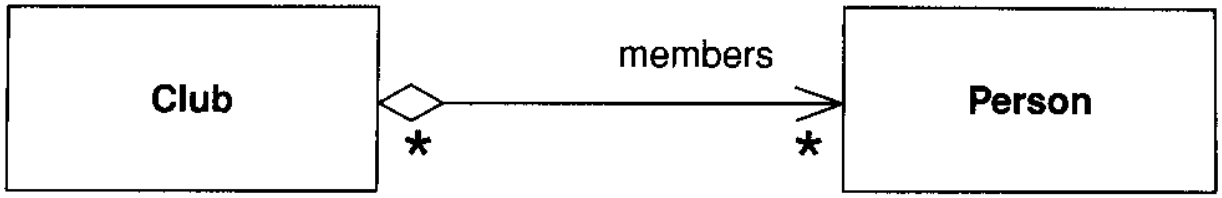


Agregações e Composições

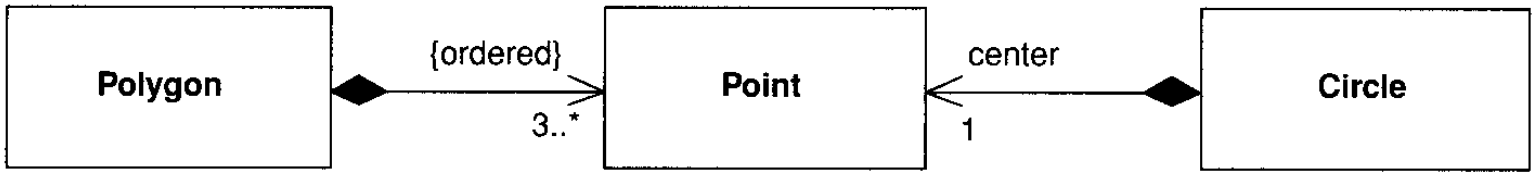
- Tipo de associações entre classes
  - Relação do tipo “parte de”
  - Conceito que dá origem a confusões
- Agregações
  - Exemplo: Rodas e motor são partes de um carro
- Composições
  - Entidade que é “parte de”
  - Tem vários potenciais “donos”
  - Só pode ter um “dono” ao mesmo tempo
  - Apenas pode fazer “parte de” uma classe ao mesmo tempo
  - Não existe sem fazer “parte de” outra classe
  - Se a classe “dona” for apagada, a classe “parte de” deixa de existir

Agregações e Composições

- Agregação



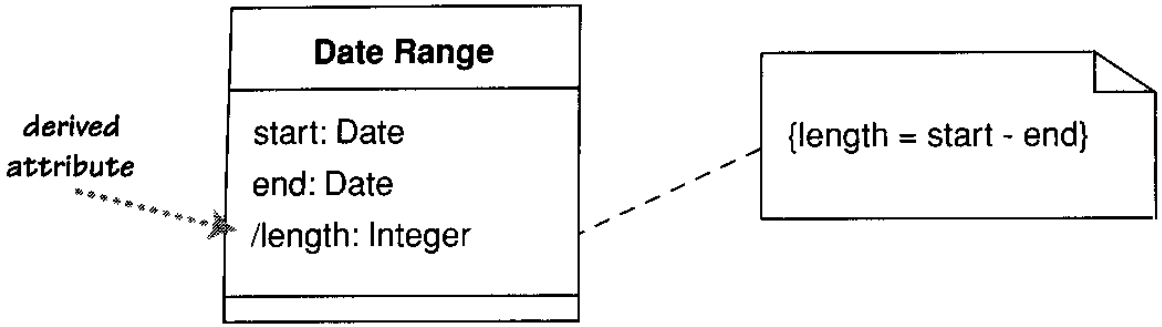
- Composição



Propriedades derivadas

- Propriedades
  - Calculadas com base noutras propriedades
- Exemplo
  - Data de inicio
  - Data de fim
  - Duração
- Quando usar
  - Uma propriedade que deriva de outras
  - Forma de especificar uma “restrição” entre valores

Propriedades derivadas



Interfaces e Classes Abstratas

- Classe abstrata
  - Classe que não pode ser instanciada
  - Instancia-se uma sub classe
  - Tipicamente: operações abstratas
    - Operações sem implementação
    - Declaração de “como” os clientes podem usar a classe
  - Representação
    - Colocar o nome em itálico
    - *Keyword*
      - `{abstract}`

Interfaces e Classes Abstratas

- Interface abstrata
  - Classe sem implementação
  - Todas os elementos são abstratos
  - Mapeamento direto
    - `Interfaces` das linguagens de programação
      - C#
      - Java
  - Representação
    - Keyword: `«interface»`

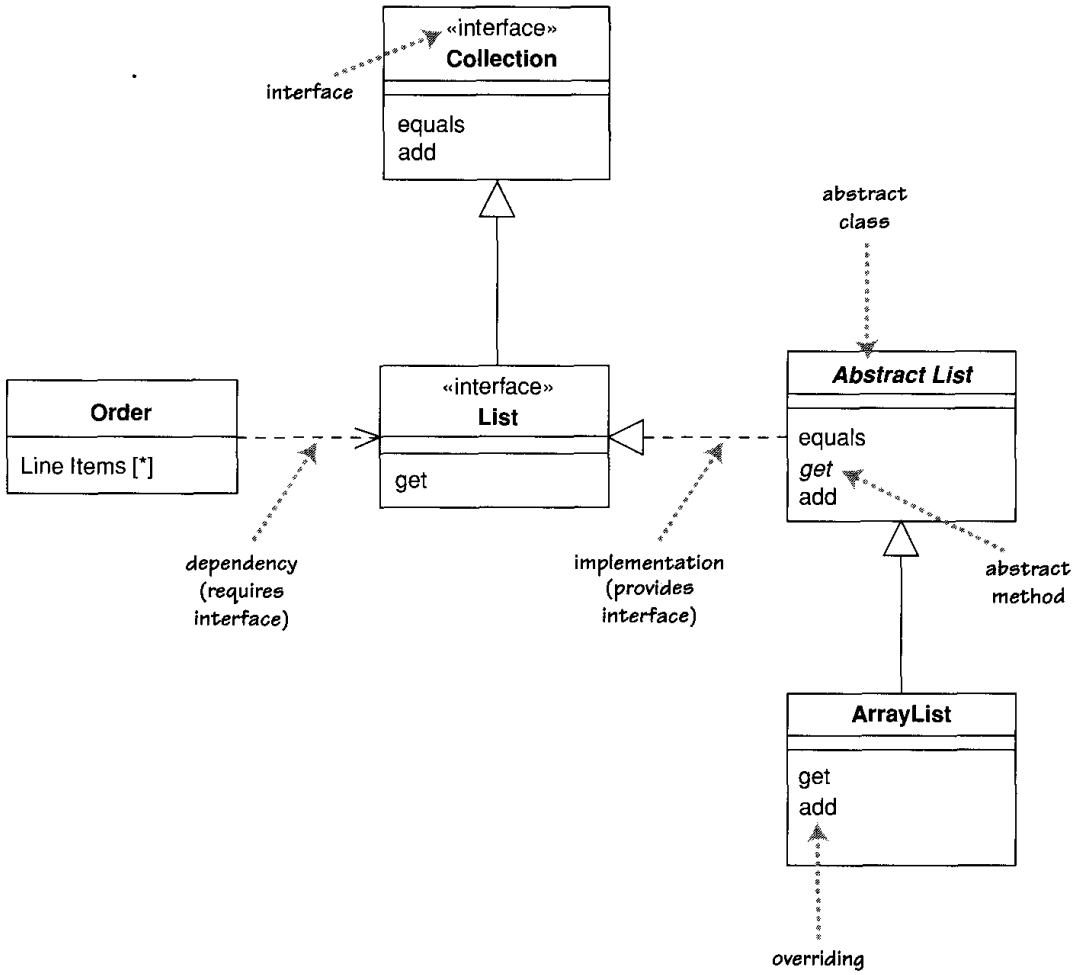
Interfaces e Classes Abstratas

- Relações entre classes e interfaces
  - Classe pode precisar de um interface
  - Classe pode fornecer o interface
- Classe fornece o interface
  - Pode substituir o interface
  - Java ou C++
    - Quando uma classe implementa um interface
    - Quando uma classe implementa um sub tipo do interface
- Classe requer o interface
  - Se necessita de uma instancia do interface para funcionar

Diagramas de classes

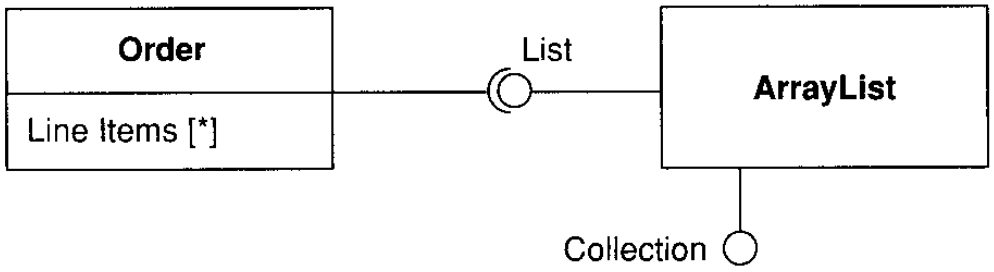
Interfaces e Classes Abstratas

- Collection
  - Interface mais geral
- List
  - Interface
  - Sub tipo de Collection
  - Implementado por AbstractList
- Order
  - Classe que depende do interface List
- AbstractList
  - Classe abstracta
  - Implementa List
  - Alguns elementos abstractos
- ArrayList
  - SubClasse de AbstractList
  - Reimplementa o get e o add



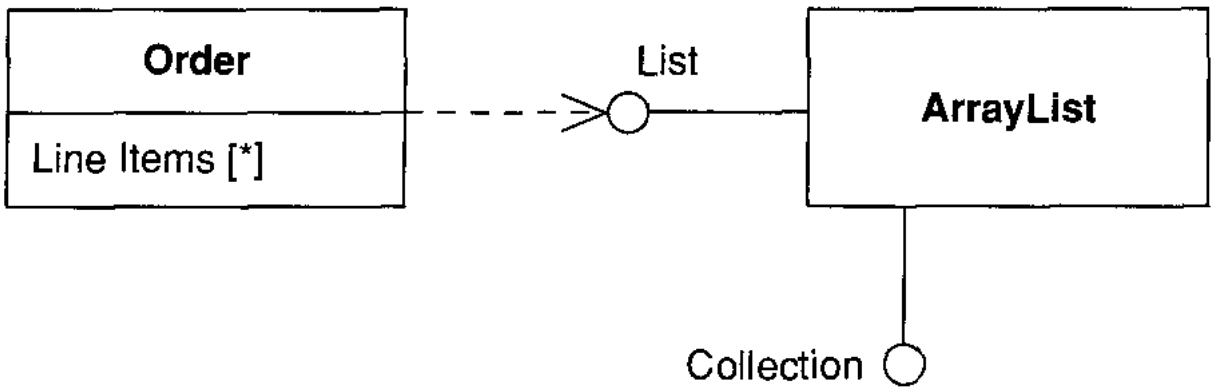
Interfaces e Classes Abstratas

- Interfaces
  - Notação alternativa
  - Classe implementa Interface
    - Representado como um circulo ou uma bola
    - **ArrayList** implementa os Interfaces **List** e **Collection**
  - Classe requer um Interface
    - Representado por uma “ficha”/”tomada” ligada ao Interface
    - Classe **Order** requer o Interface **List**



Interfaces e Classes Abstratas

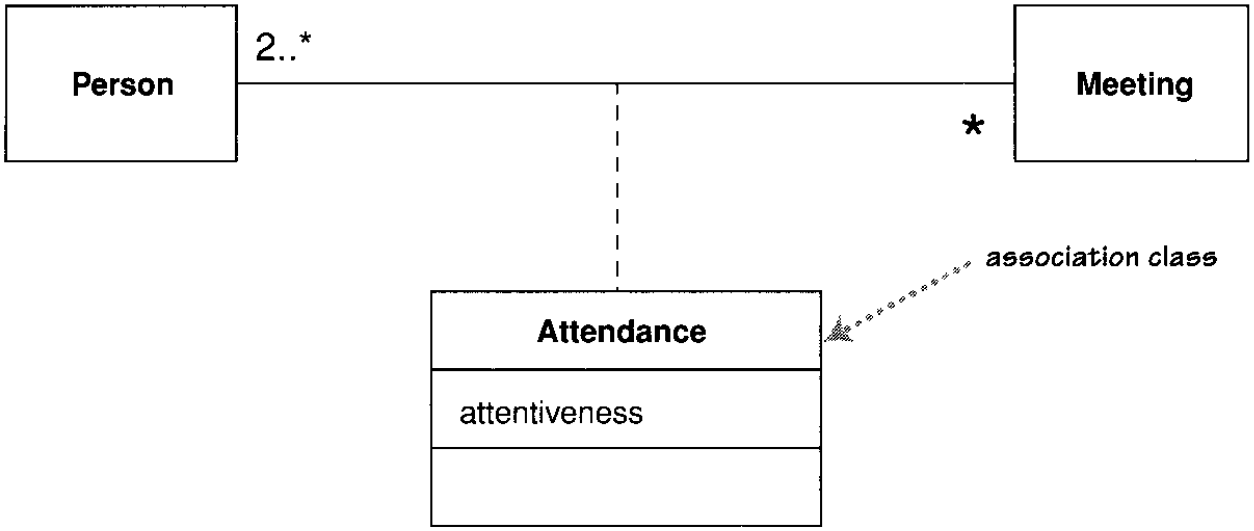
- Interfaces
  - Outra representação
    - Requer um Interface
    - Notação mais velha
    - Representada por uma dependência



Classes Associativas

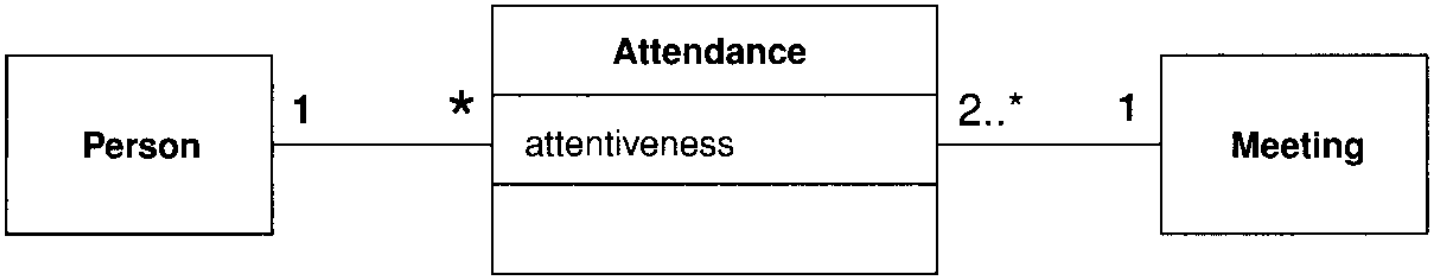
- *Association Class*
  - Permitem adicionar atributos, operações e outras características a relações ou associações
  - Exemplo
    - Numa **Reunião** participam 2 ou mais **Pessoas**
    - Cada **Pessoa** pode participar em várias **Reuniões**
    - Como representamos a *participação* de uma **Pessoa** em cada **Reunião**?

Classes Associativas



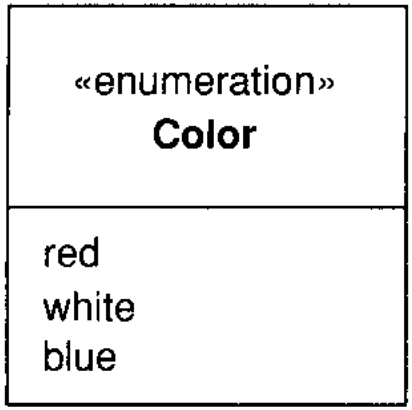
Classes Associativas

- Alternativa
  - Promover a Classe Associativa a uma Classe
  - 1 **Pessoa** tem várias **Participações**
  - 1 **Reunião** tem 2 ou mais **Participações**
  - 1 **Participação** apenas pertence a uma **Pessoa** e a uma **Reunião**



Enumerações

- Conjunto de valores sem propriedades
  - Apenas interessa o seu valor simbólico
  - Especificar os valores permitidos para um atributo
  - Representados numa “classe”
    - Keyword «enumeration>



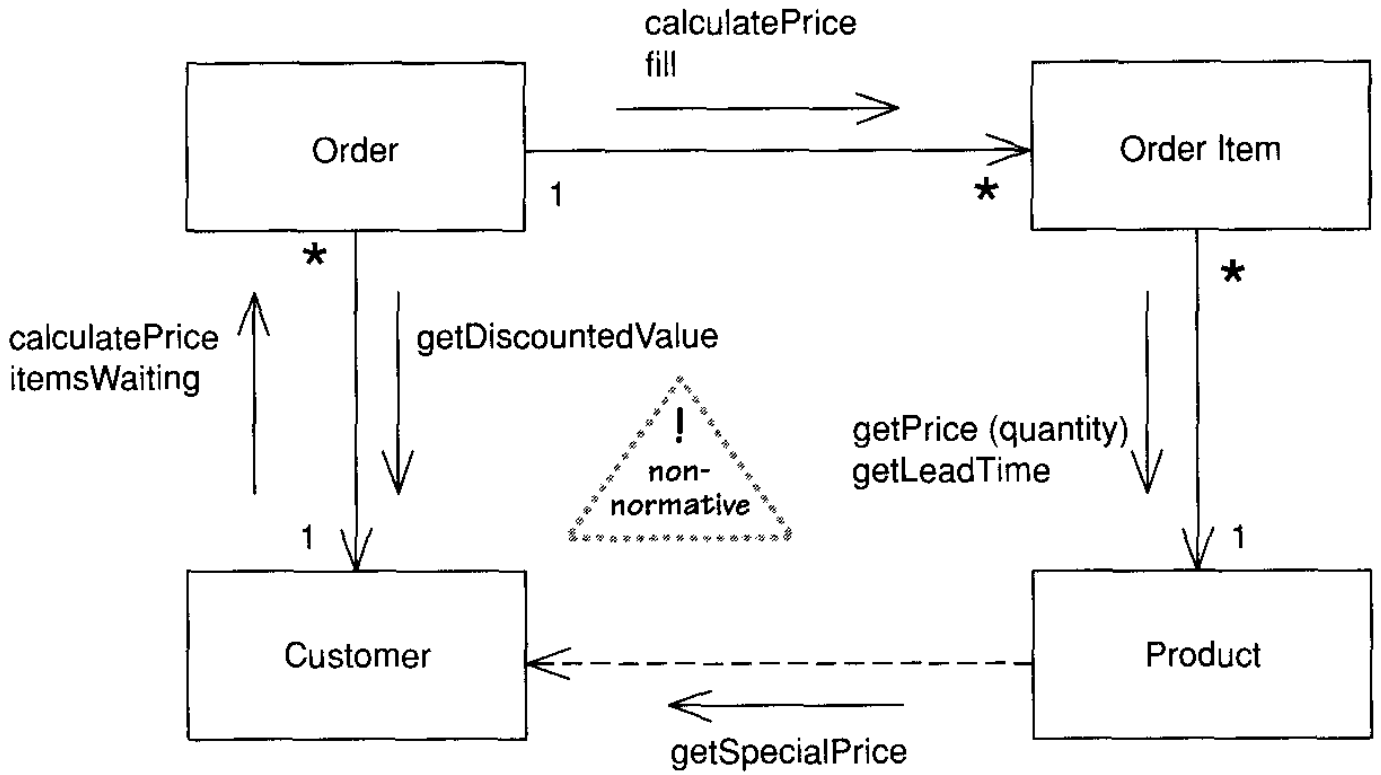
Visibilidade

- Elementos privados e públicos
  - Públicos (**+**)
    - Visíveis por qualquer entidade
  - Privados (**-**)
    - Visíveis apenas pela classe
  - Protegidos (**#**)
    - Visíveis apenas pela classe, ou suas subclasses
  - Package (**~**)
    - Package → “arrumar” elementos
    - Visíveis para qualquer elemento dentro da mesma classe

Mensagens

- Indicar invocação de operações
  - Não pertence ao UML standard
  - Pode ser *conveniente*
  - Representação
    - Seta que indica o sentido da invocação
    - Mensagens *junto* aos arcos das relações ou das dependências

Mensagens


calculatePrice
fill
Order
Order Item
1
*
*
*
calculatePrice
itemsWaiting
getDiscountedValue
!
non-
normative
getPrice (quantity)
getLeadTime
1
1
Customer
Product
getSpecialPrice

Bibliografia

- UML Distilled A Brief Guide to the Standard Object Modeling Language. Martin Fowler. 3rd edition. Addison-Wesley Professional. 2003. Capítulos 3 e 5.
- Software Engineering. Ian Sommerville. 10th Edition. Addison-Wesley. 2016. Capítulo 5.