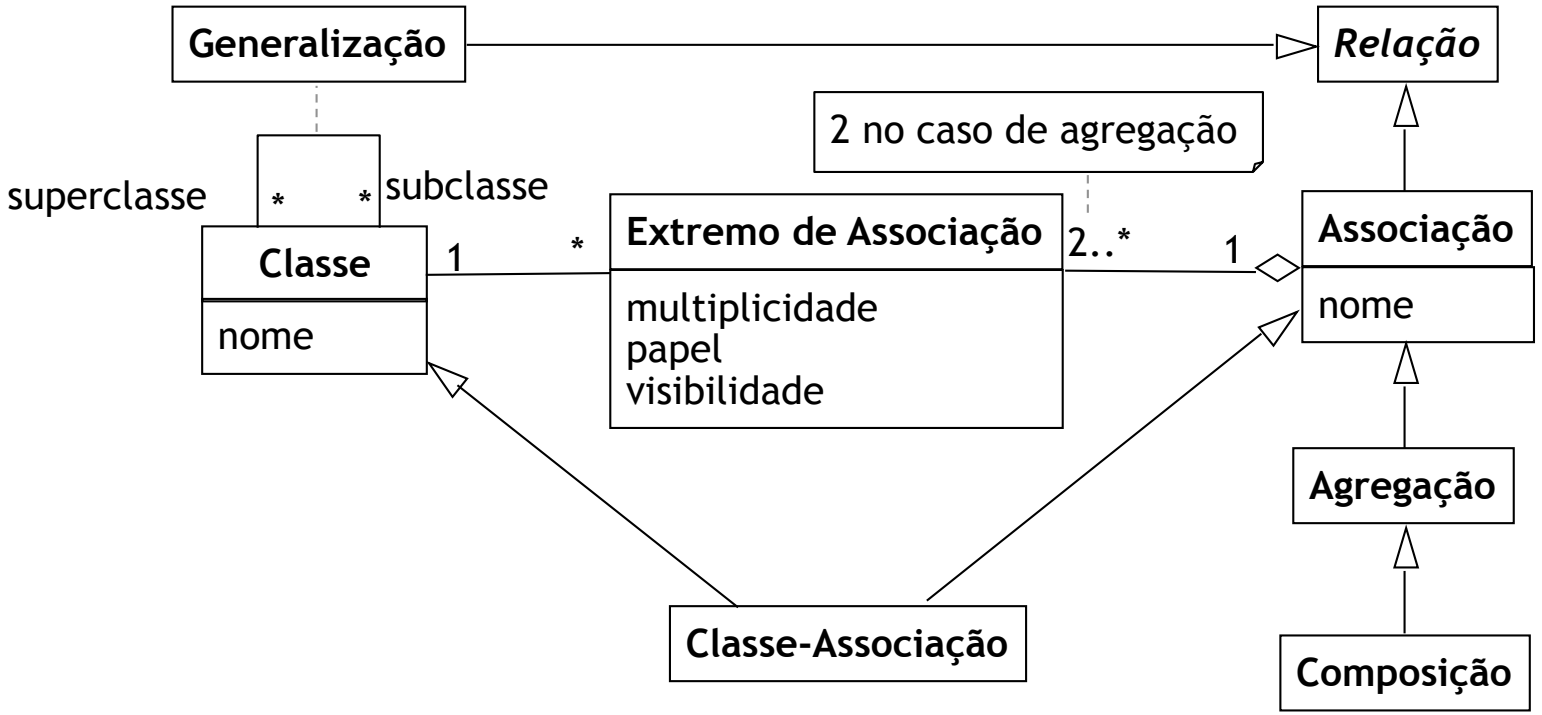
Desenvolvimento de Sistemas Software

Aula Teórica 17: Modelação Estrutural / Diagramas de Classe III

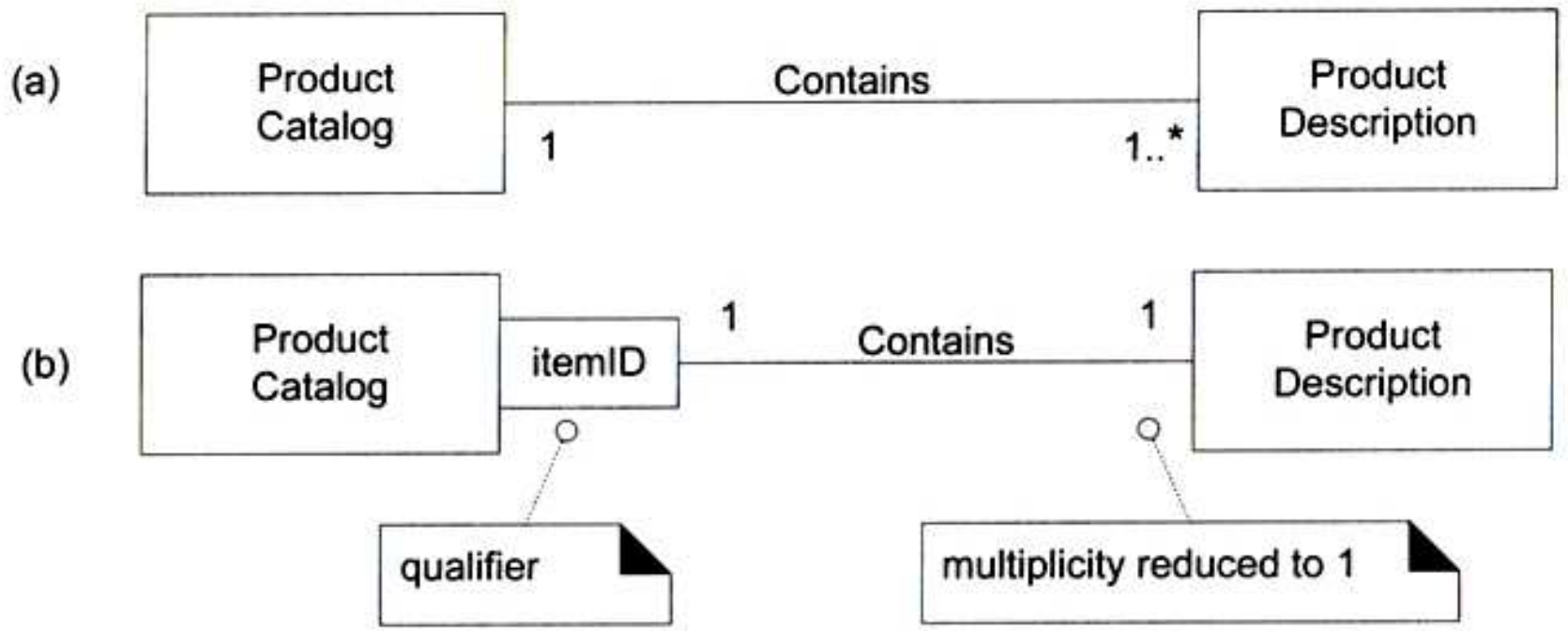
Desenvolvimento de Sistemas Software
António Nestor Ribeiro, José Creissac Campos, F. Mário Martins

Meta-modelo do UML (relações entre classes)

- É possível descrever em UML a semântica dos diagramas



DC: Associação Qualificada



Em b) é explicitamente especificado agora que, o qualificador itemID é uma "chave única" de relacionamento entre "Product Catalog" e "Product Description". Sendo única, o relacionamento passa a ser agora lido como "Um catálogo de produtos contém 1 e só uma descrição de produto para cada valor de itemID".

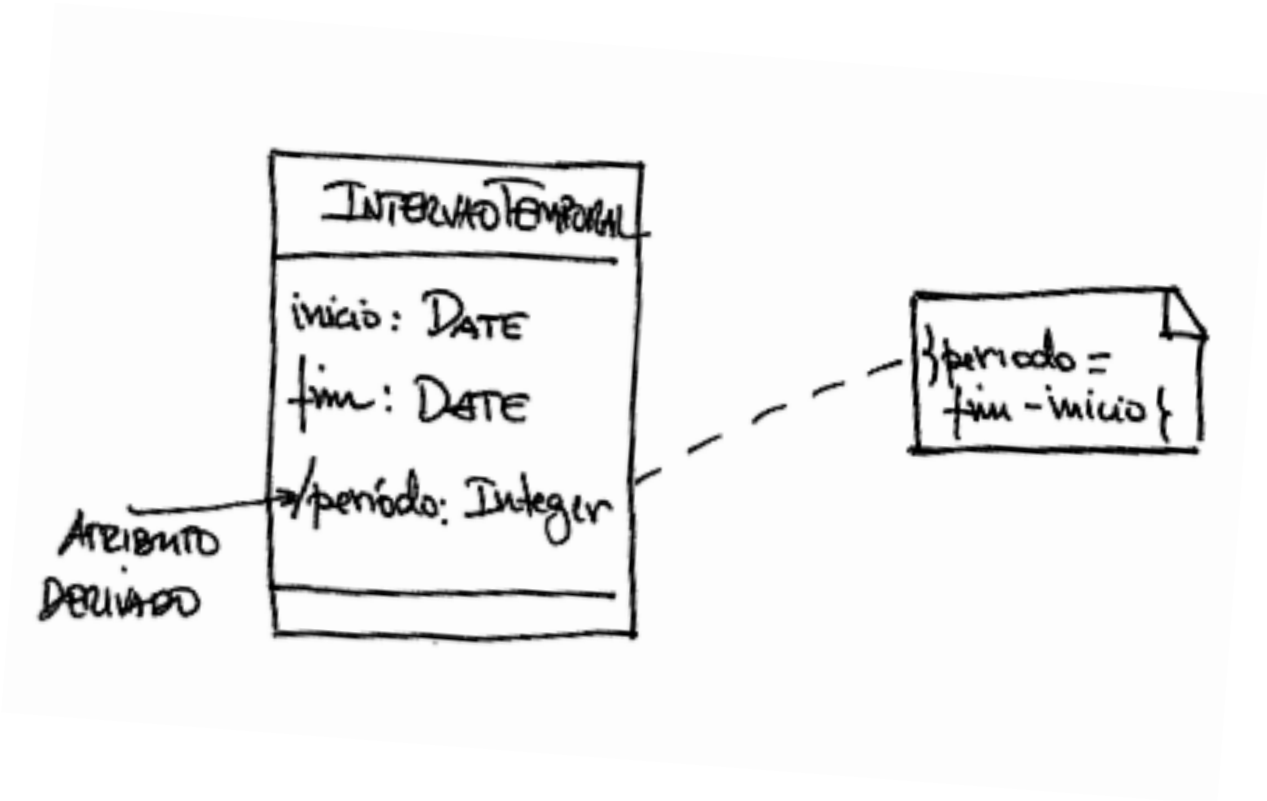
Associações qualificadas

- "Produto" é chave na relação

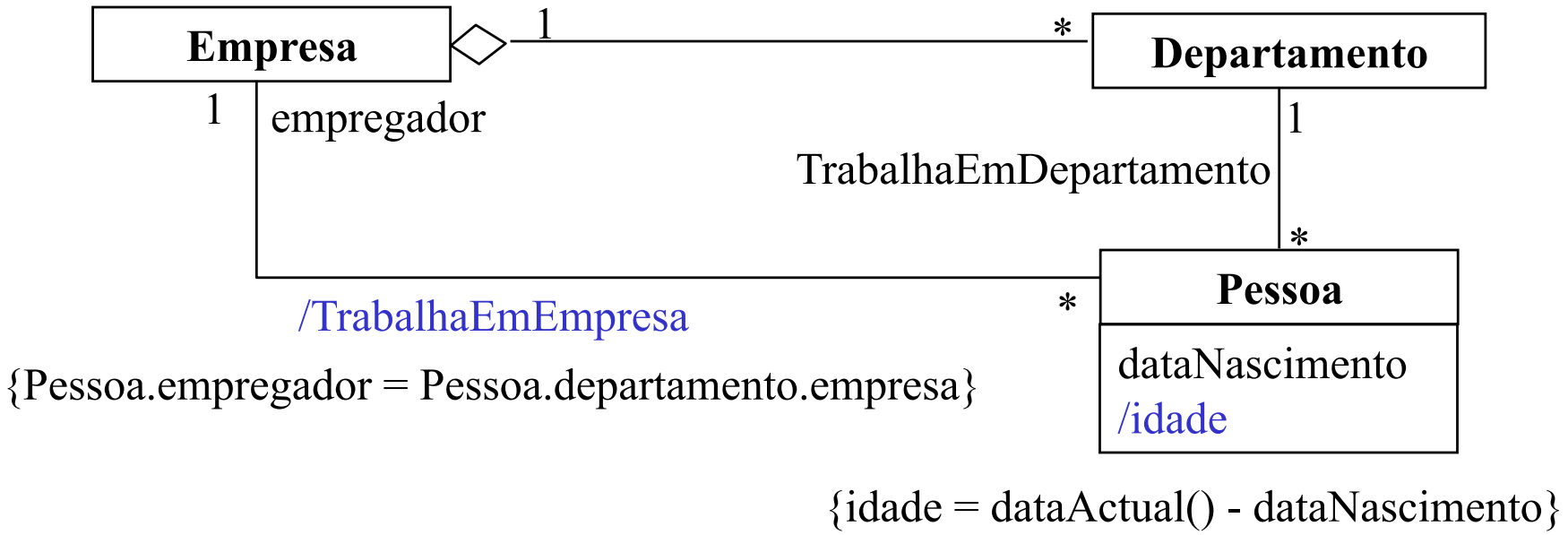

```
public class Encomenda {

  private Map<Produto, LinhaEncomenda> linhas;
  ...
  public LinhaEncomenda getLinhaEnc(Produto umProd);
  public void addLinhaEncomenda(Integer qt,
                                Produto umProd);
  ...
}
```

Diagramas de Classe

- Atributos derivados

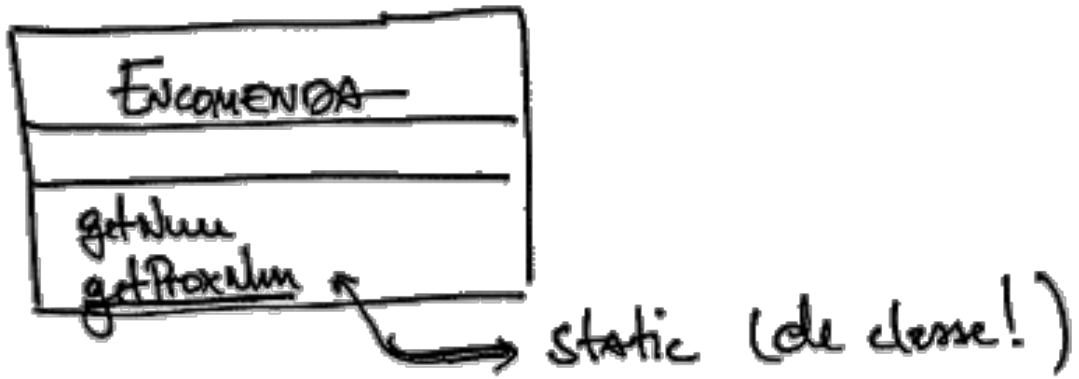


Exemplos de atributos derivados



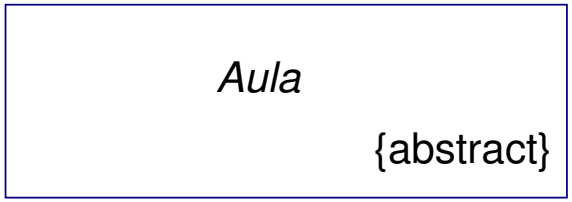
Operações e variáveis de classe

- Variáveis de classe são variáveis globais a todas as instâncias de uma classe.
- Métodos de classe são métodos executados directamente pela classe e não por uma das suas instâncias (logo, não têm acesso directo a variáveis/métodos de instância).
- São representados tal como variáveis/métodos de instância, mas sublinhados.
- Deve evitar-se abusar de operações e variaveis de classe.

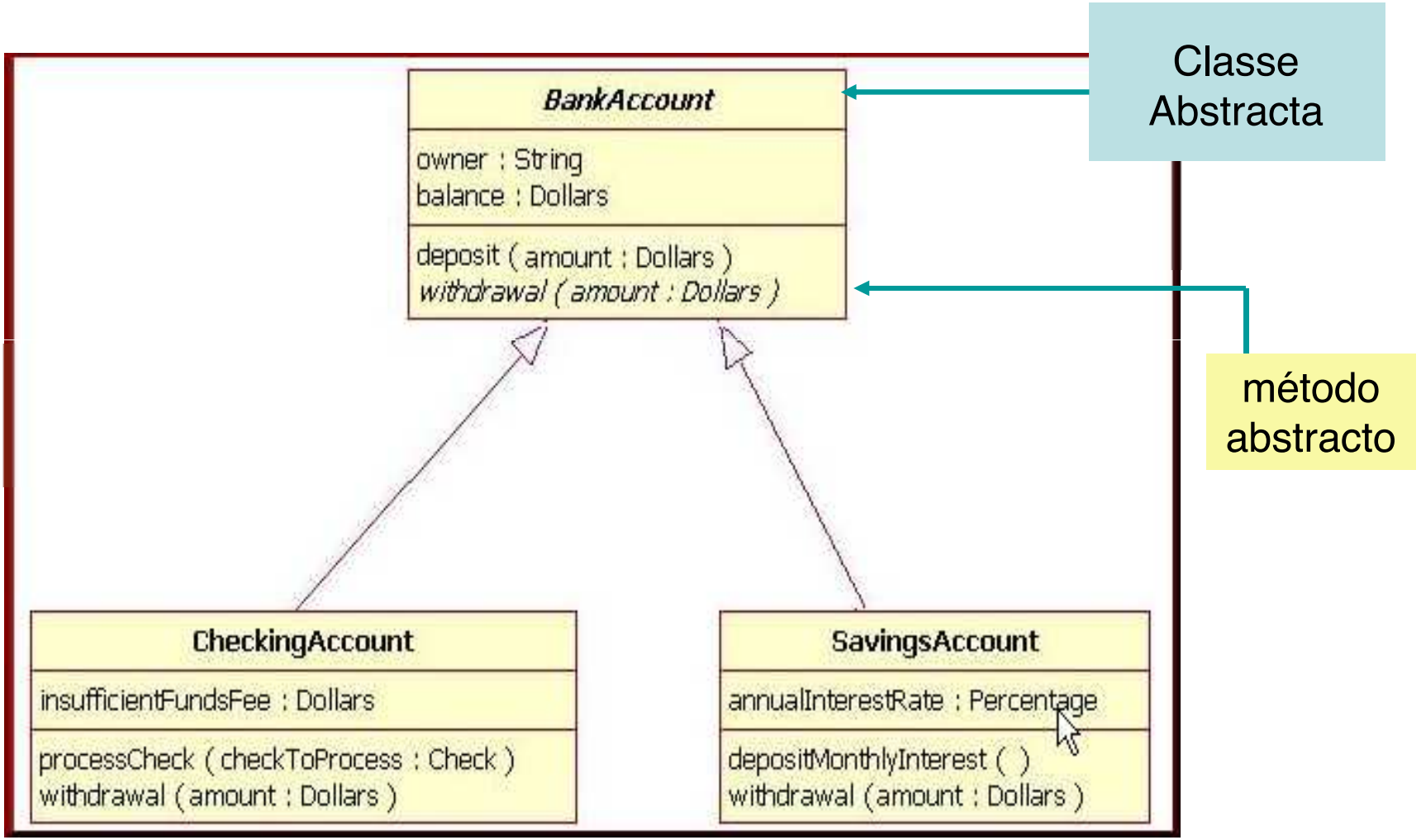


Classes abstratas

- Nem sempre ao nível da super-classe é possível saber qual deverá ser o método associado a uma operação.
- Quando se está a utilizar uma hierarquia de classes para representar sub-tipos, pode não fazer sentido permitir instâncias da super classe.
- Uma operação abstracta é uma operação que não tem método associado na classe em que está declarada.
- Uma classe abstracta é uma classe da qual não se podem criar instâncias e que pode conter operações abstractas.
- Classes concretas (não abstractas) não podem conter métodos abstractos!
- Notação: em *itálico* ou através da propriedade {abstract}.



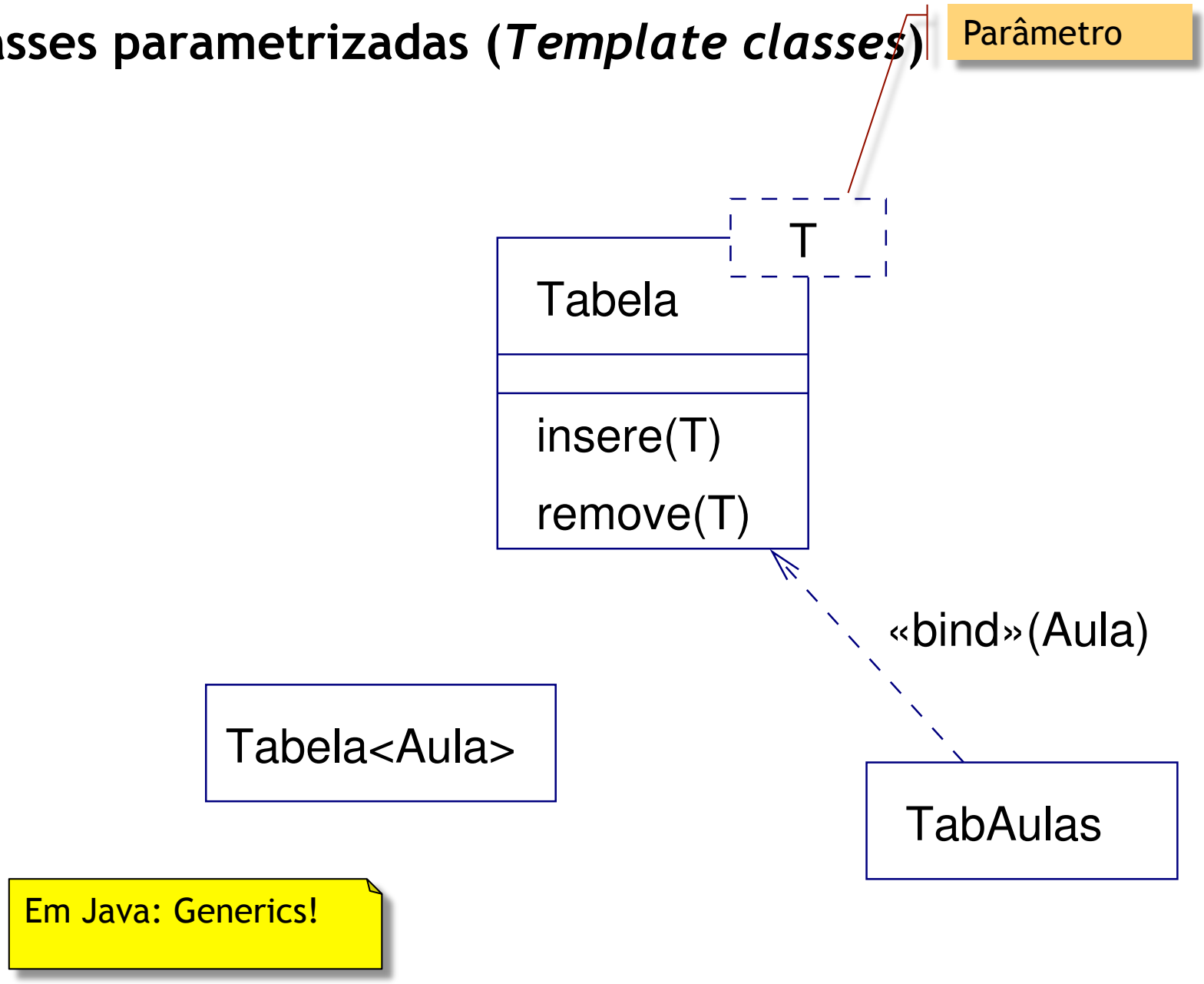
Exemplo


BankAccount
owner : String
balance : Dollars
deposit (amount : Dollars)
withdrawal (amount : Dollars)
CheckingAccount
insufficientFundsFee : Dollars
processCheck (checkToProcess : Check)
withdrawal (amount : Dollars)
SavingsAccount
annualInterestRate : Percentage
depositMonthlyInterest ( )
withdrawal (amount : Dollars)
Classe Abstracta
método abstracto

Classes root, leaf e active

- Classes etiquetadas com a propriedade {root} não podem ser generalizadas.
  - Por exemplo, se o modelo apresenta classes pertencentes ao ambiente de desenvolvimento que irá ser utilizado, não será viável generalizar tais classes.
- Classes etiquetadas com a propriedade {leaf} não podem ser especializadas (classes final no Java).
  - Por exemplo, se o sistema contém uma classe que fornece serviços de encriptação, por motivos de segurança não é desejável que os métodos associados às operações dessa classe possam ser redefinidos (isto também pode ser controlado ao nível das operações).
- Classes etiquetadas com a propriedade {active} são consideradas ativas
  - Por exemplo, uma *thread*.

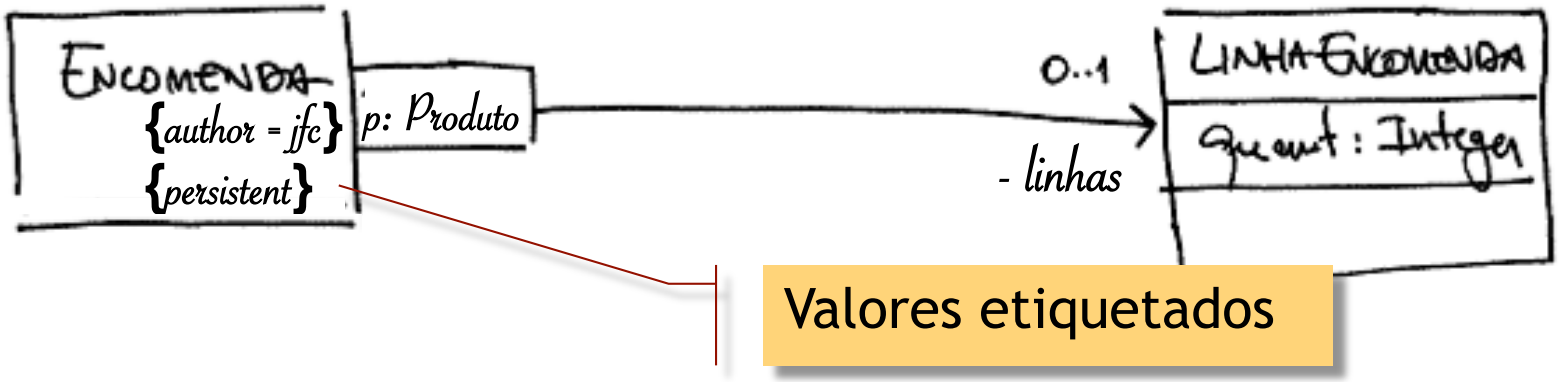
Classes parametrizadas (*Template classes*)



Mecanismos de extensibilidade

- Estereótipos
- “Tagged Values” (valores etiquetados)
- Restrições (“constraints”)

- **Valores Etiquetados**
  - Definem novas propriedades das “coisas”
  - Trabalham ao nível dos meta-dados

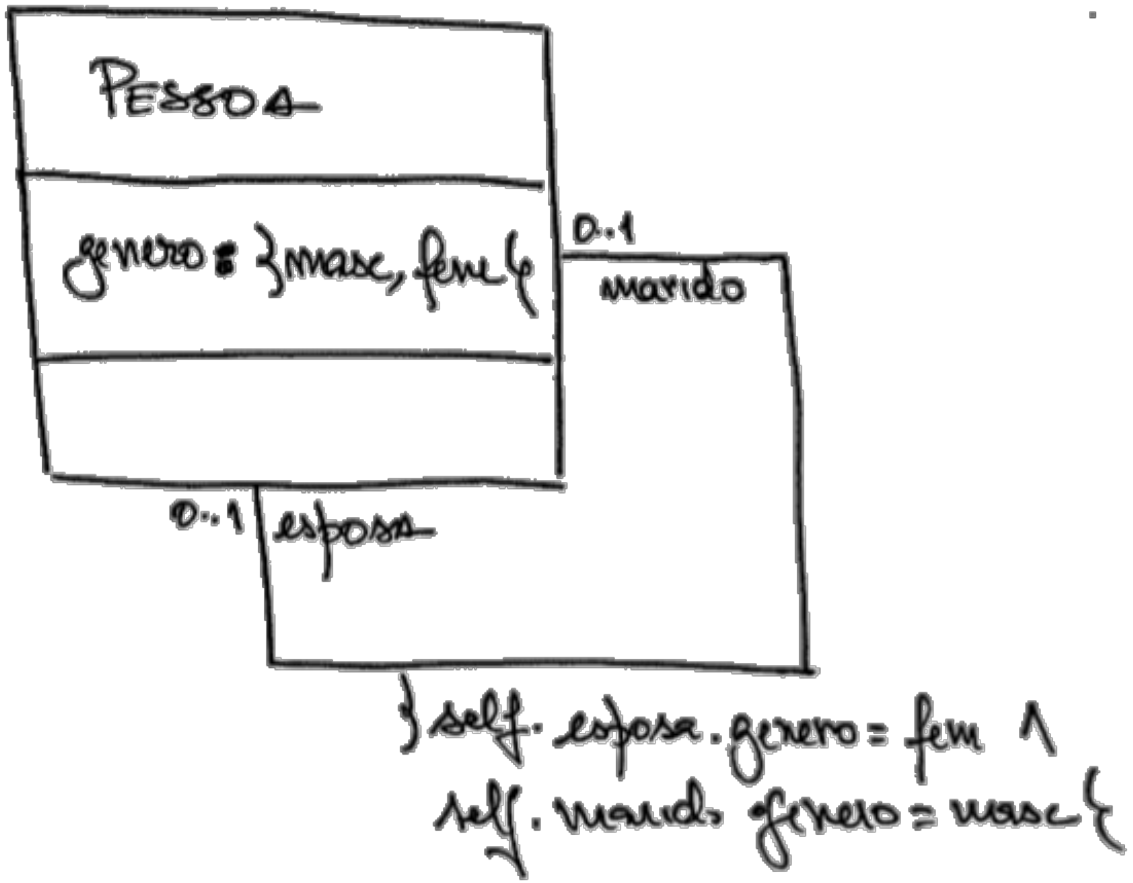


Mecanismos de extensibilidade

- **Estereótipos**
  - Permitem a definição de variações dos elementos de modelação existentes (ex: «include», «extend» são estereótipos de dependência)
  - Possibilitam a extensão da linguagem de forma controlada
    - Cada estereótipo pode ter a si associado um conjunto de valores etiquetados
  - Trabalham ao nível dos meta-dados
  - Meta-tipo de dados ≠ Generalização

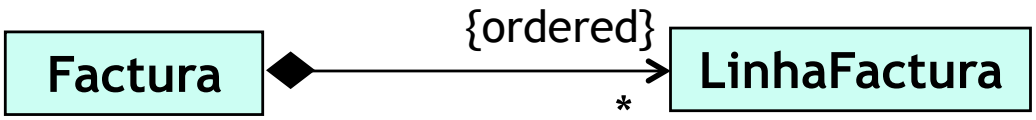
Mecanismos de extensibilidade

- **Restrições**
  - Utiliza-se quando a semântica das construções diagramáticas do UML não é suficiente



Restrições às associações

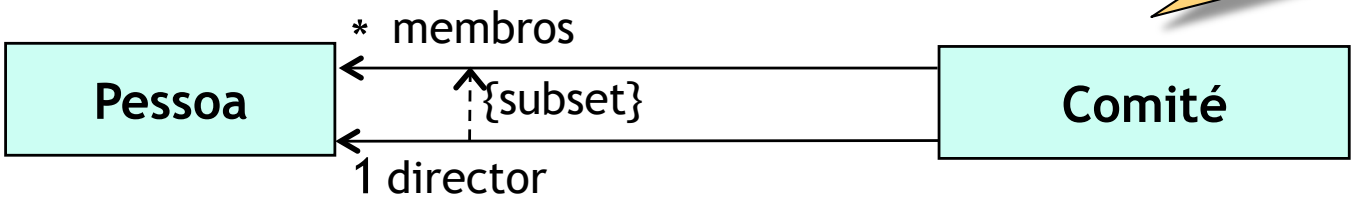
uma factura é constituída por um conjunto ordenado de 0 ou mais linhas



Restrição aplicada a um dos papeis (*roles*).

E.g. ordered, sorted

o director de um comité tem que ser um dos seus membros

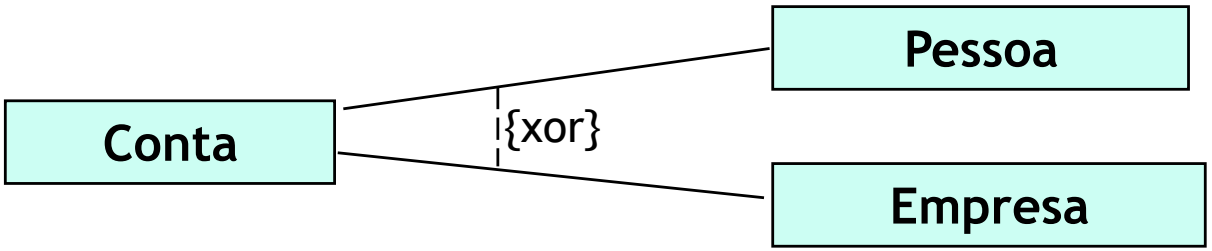


Restrição aplicada a duas associações (com direcção).

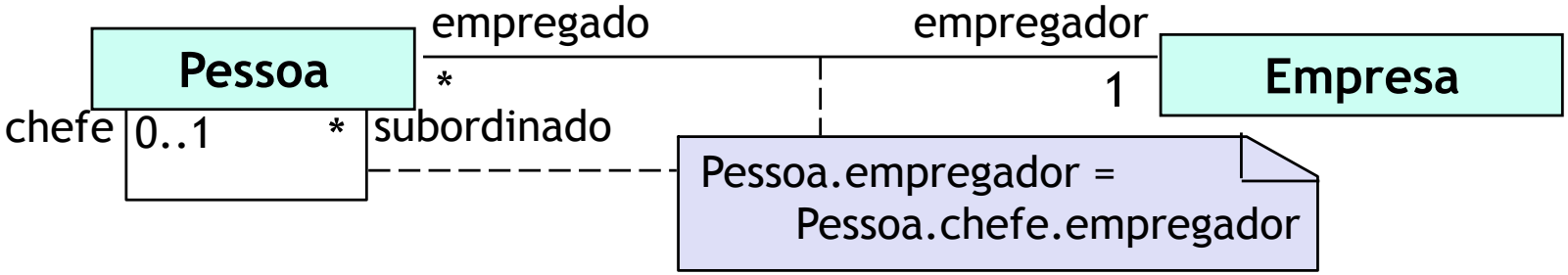
Restrições às associações

Restrições aplicadas a duas associações (sem direção). E.g. associações mutuamente exclusivas.

uma pode ser de uma pessoa ou de uma empresa (mas não de ambos)



o empregador do chefe, é o empregaddor do subordinado



Declaração de atributos / operações

- **Atributos**

Obrigatório!

«esteréotipo» visibilidade / nome : *tipo* [*multiplicidade*] = valorInic {propriedades}

- Exemplos

  nome

  - nome = “JCC” {addedBy=“jfc”, data=“18/11/2011”}
  - nome : String [1..2] {leaf, addOnly, addedBy=“jfc”}

Propriedades comuns:
changeability:
changeable – pode ser alterado (o *default*)
frozen – não pode ser alterado (final em Java)
addOnly – para multiplicidades > 1 (só adicionar)
leaf – não pode ser redefenido
ordered – para multiplicadedes > 1

Declaração de atributos / operações

- **Operações**

Obrigatório!

in | out | inout | return

«*esteréotipo*» *visibilidade* nome (direção nomeParam : tipo = valorOmiss) : *tipo* {propriedades}

- Exemplos

por omissão é "in"

setNome

+ setNome(nome = "SCX") {abstract}

+ getNome() : String {isQuery, risco = baixo}

+ getNome(out nome)

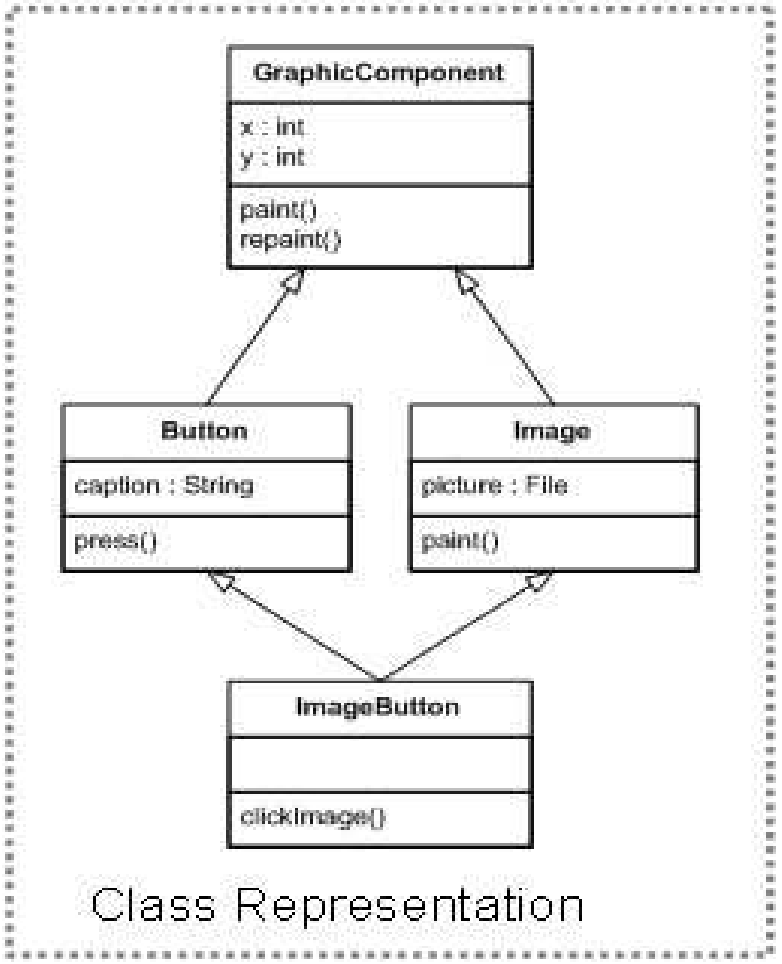
+ «create» Pessoa()

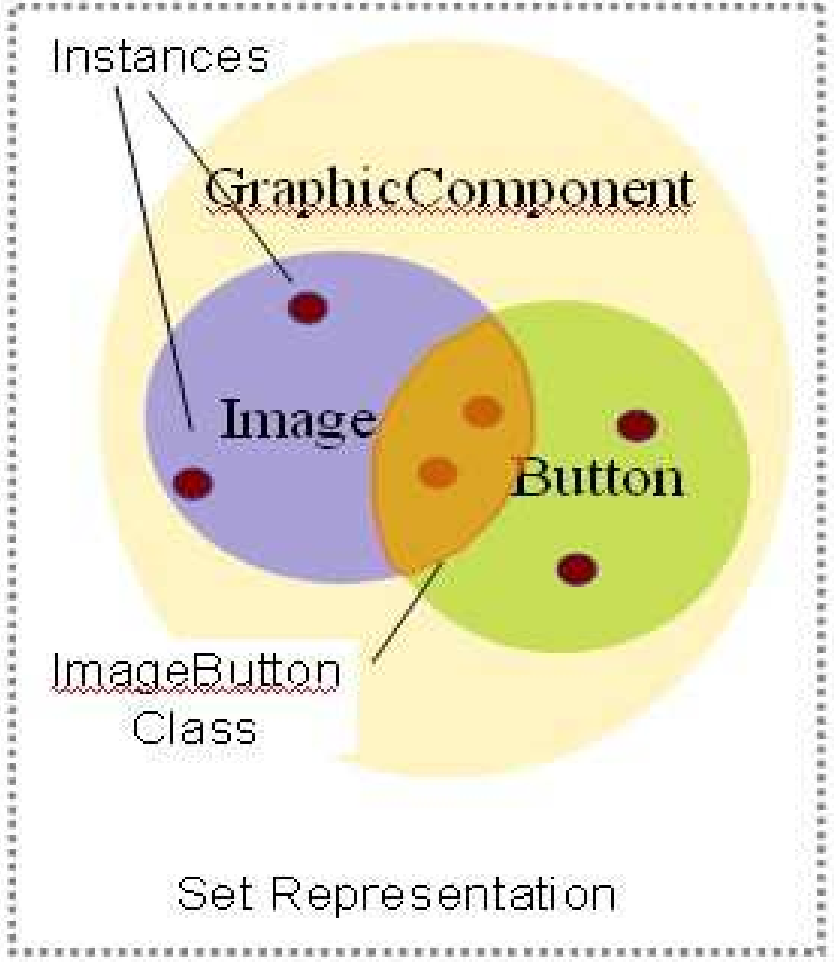
in – parâmetro de entrada
out – parâmetro de saída
inout – parâmetro de entrada/saída
return – operação retorna o parâmetro como um dos seus valores de retorno

Propriedades comuns:
abstract – operação abstrata
leaf – não pode ser redefenido
isQuery – não altera o estado do objecto

Herança Múltipla

Generalização e Conjuntos de Instâncias





Interfaces

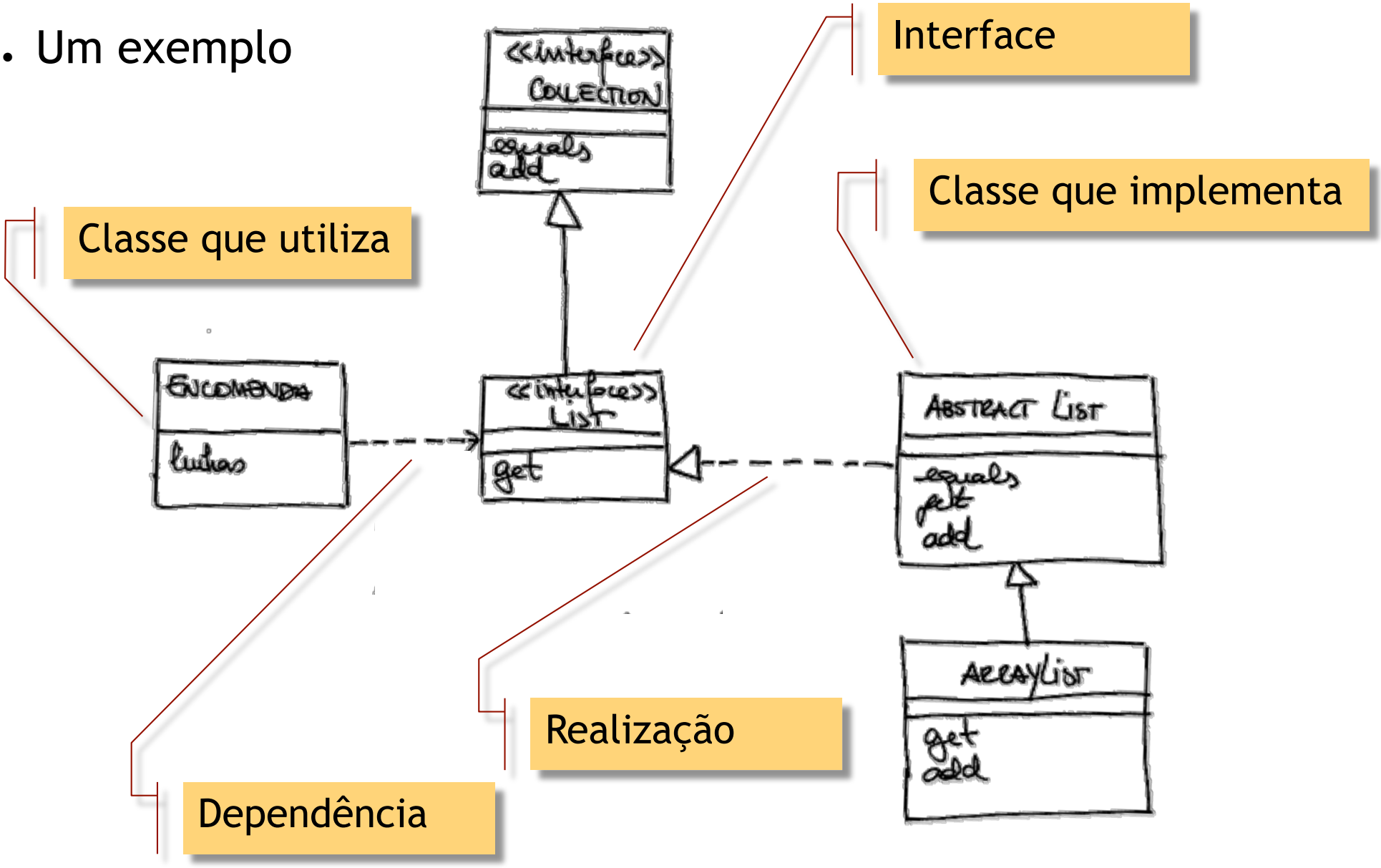
- Uma interface especifica um tipo abstracto – um conjunto de operações externamente visíveis que uma classe (ou componente, subsistema, etc.) deve implementar
- semelhante a classe abstracta só com operações abstractas e sem atributos nem associações
- separação mais explícita entre interface e (classes de) implementação
- interfaces são mais importantes em linguagens que têm herança simples de implementação e herança múltipla de interface (como em Java)

Interfaces

- Relação de concretização de muitos para muitos entre interfaces e classes de implementação
- Vantagem em separar interface de implementação: os clientes de uma classe podem ficar a depender apenas da interface em vez da classe de implementação
- Notação UML:
  - classe com estereótipo «interface» (ligada por relação de realização à classe de implementação), ou
  - notação "lollipop" – círculo (ligado por linha simples à classe de implementação).

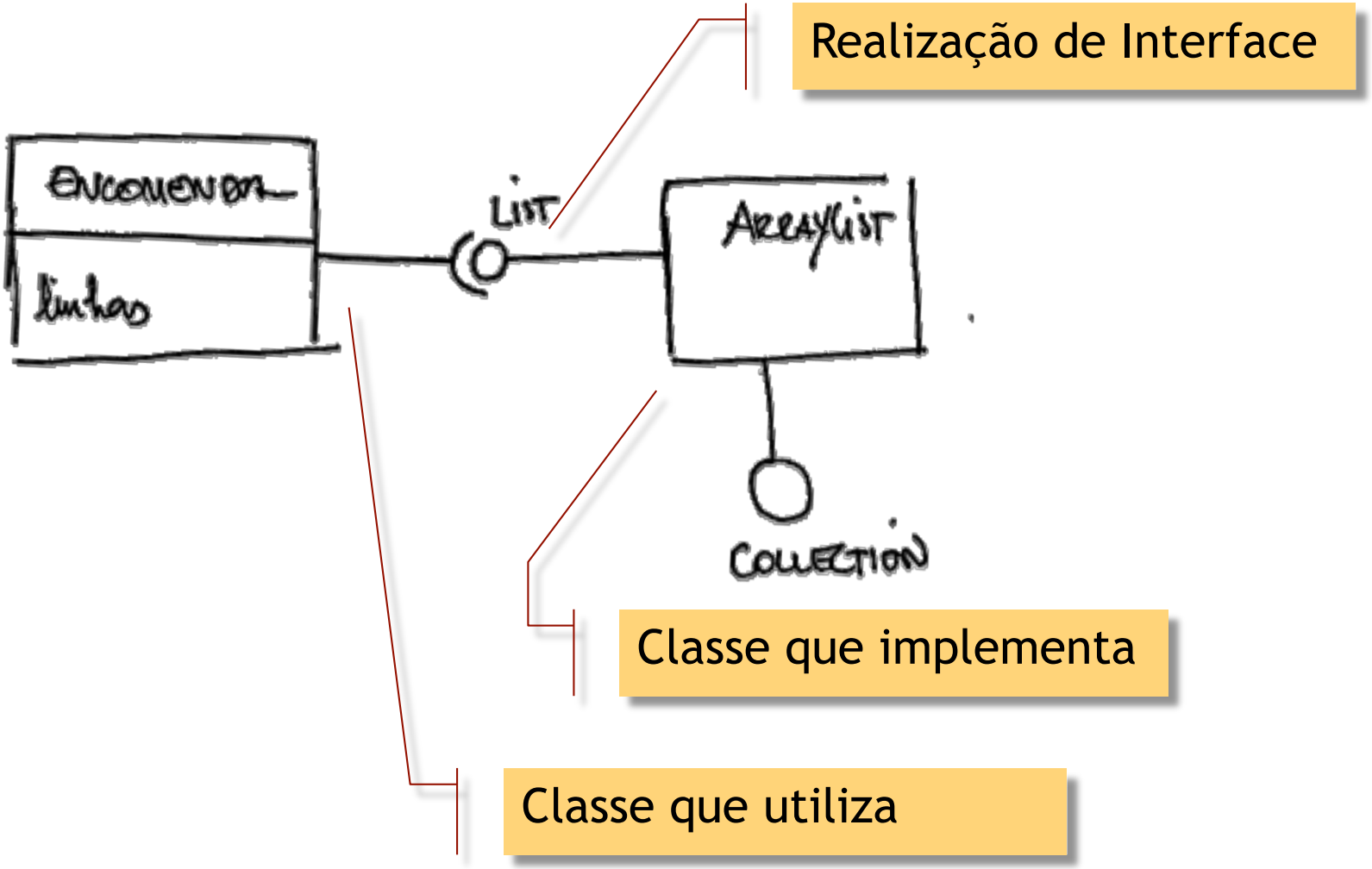
Interfaces

- Um exemplo

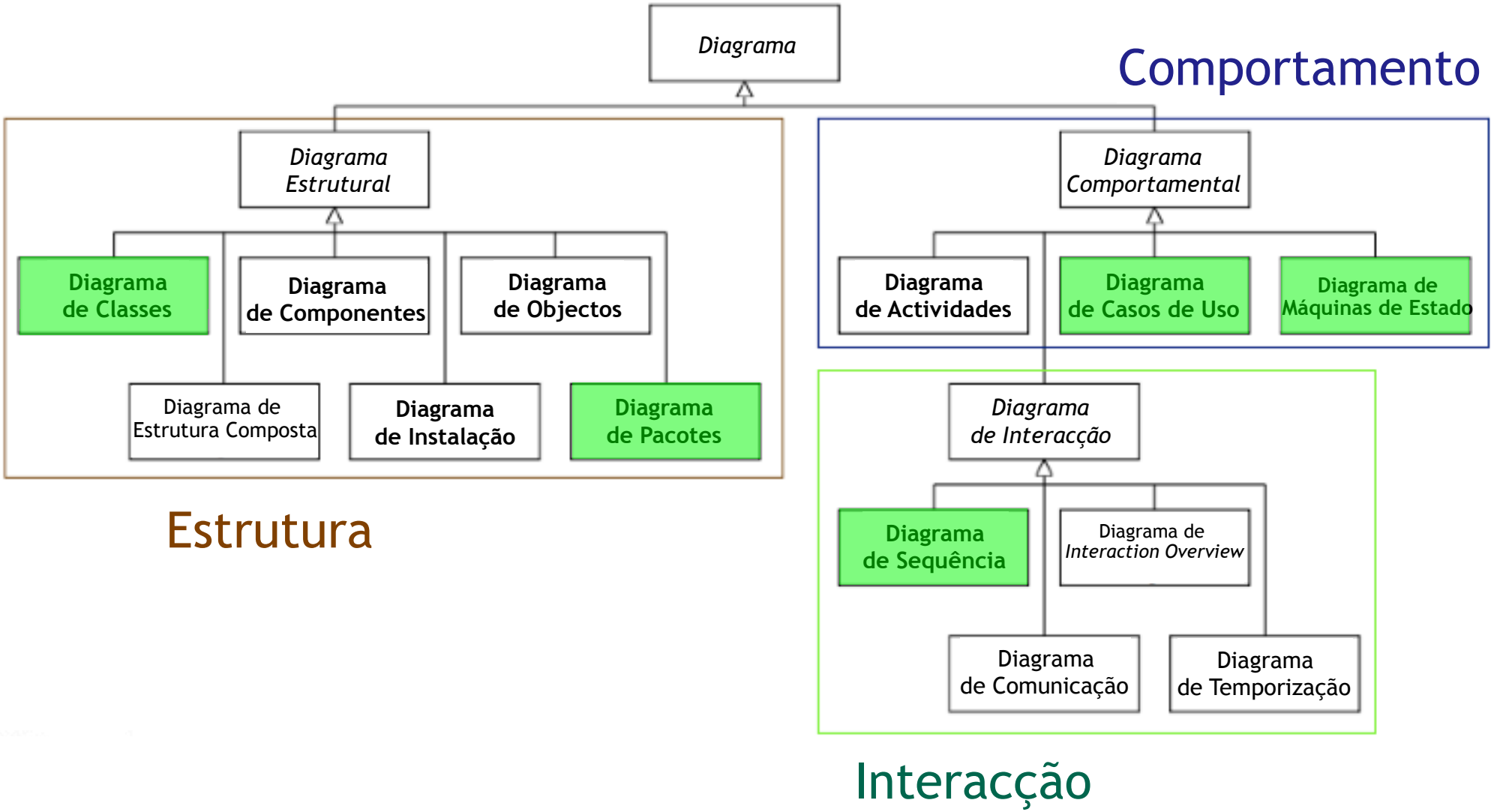


Interfaces

- Interfaces - utilização da notação “lollipop” do UML 2.x



Diagramas da UML 2.x



Modelação Estrutural

Sumário

- Diagramas de Classe III
  - Qualificação de associações
  - Atributos derivados
  - Operações e variáveis de classe
  - Classes abstratas
  - Classes root, leaf e active
  - Classes parametrizadas
  - Mecanismos de extensibilidade: valores etiquetados, estereótipos e restrições
  - Restrições às associações
  - Sintaxe da declaração de atributos / operações
  - Interfaces