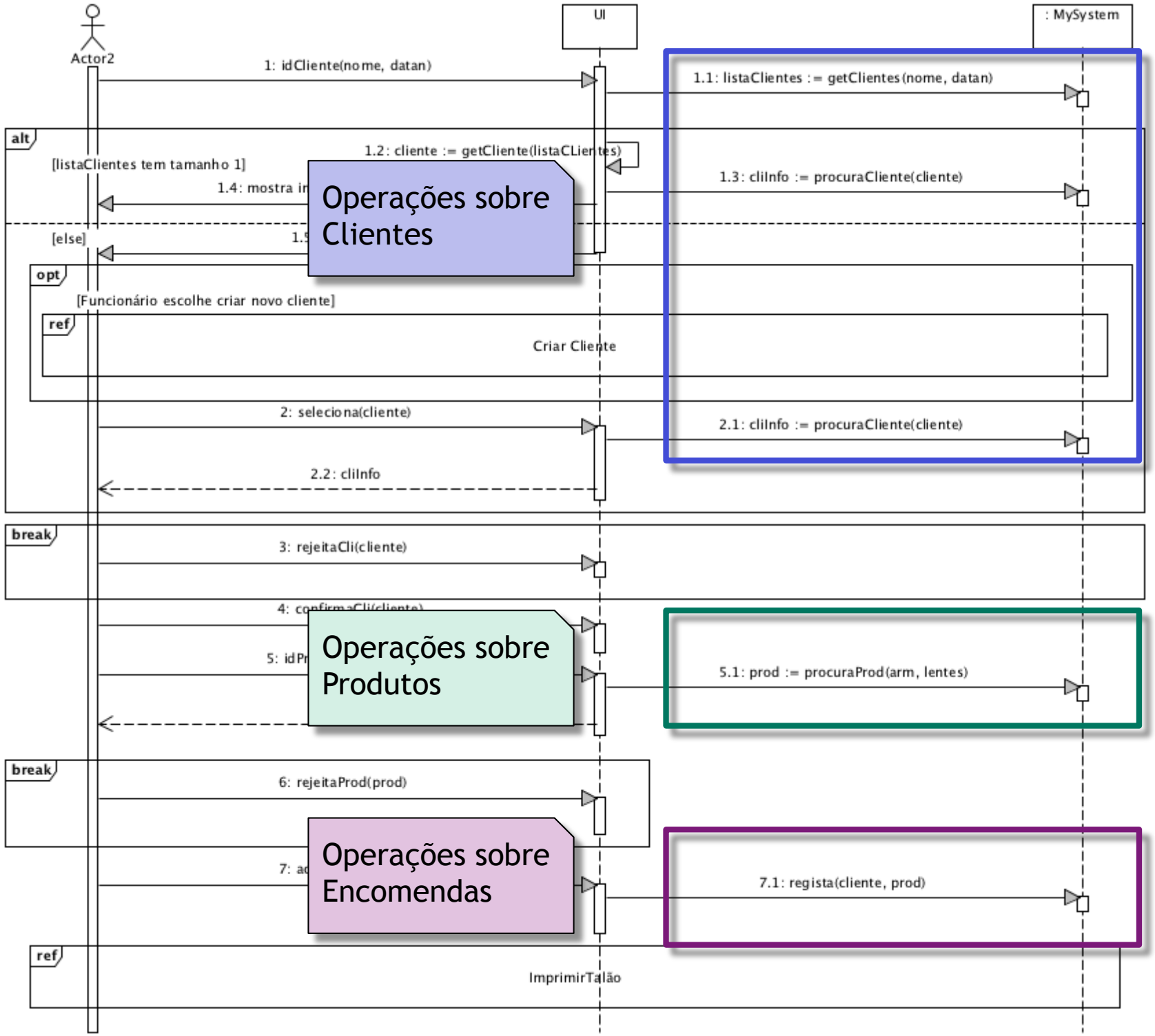
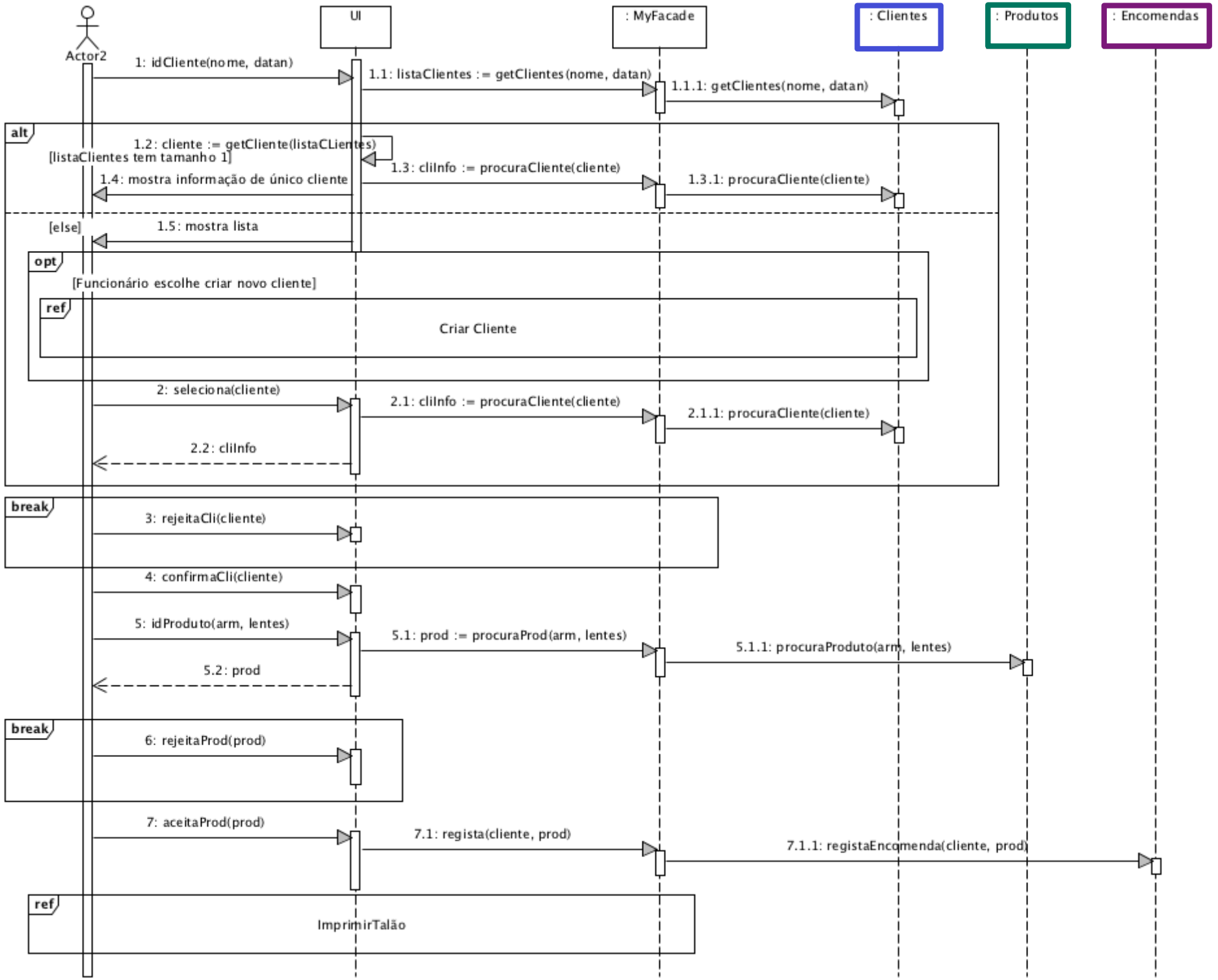
Desenvolvimento de Sistemas Software

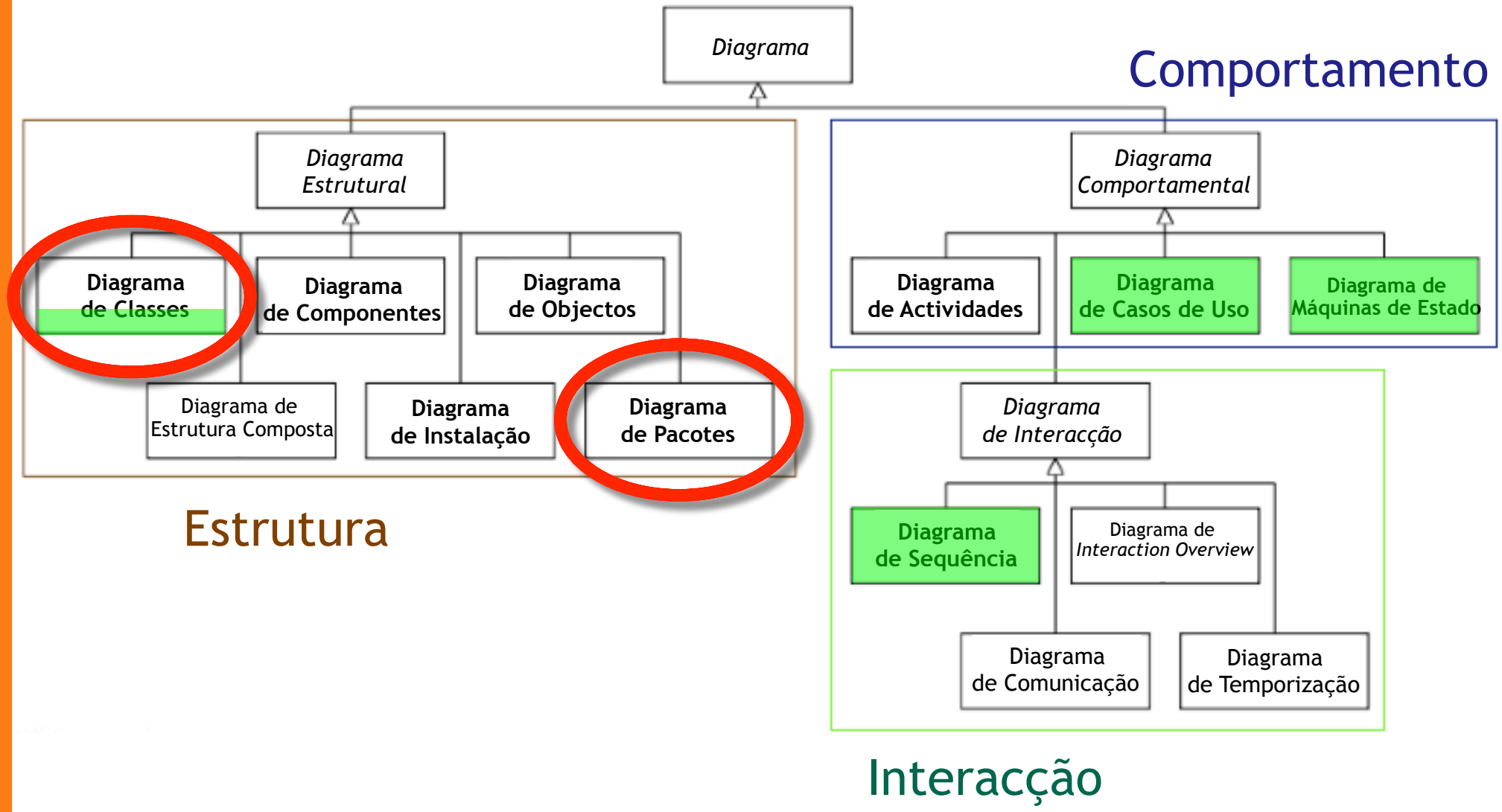
Aula Teórica 16: Modelação Estrutural

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

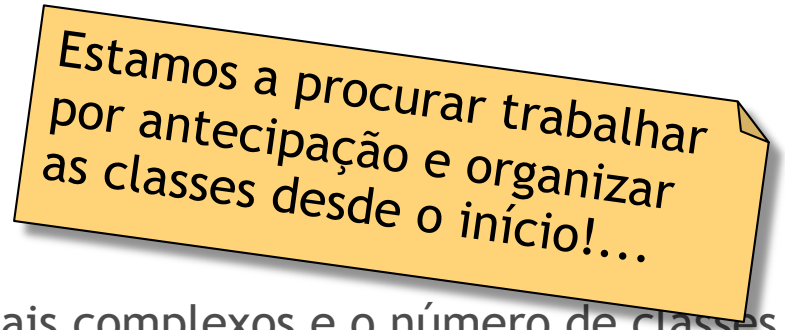

Actor2
UI
: MySystem
1: idCliente(nome, datan)
1.1: listaClientes := getClientes(nome, datan)
alt
[listaClientes tem tamanho 1]
1.2: cliente := getCliente(listaClientes)
1.3: cliInfo := procuraCliente(cliente)
1.4: mostra in
Operações sobre Clientes
[else]
opt
[Funcionário escolhe criar novo cliente]
ref
Criar Cliente
2: seleciona(cliente)
2.1: cliInfo := procuraCliente(cliente)
2.2: cliInfo
break
3: rejeitaCli(cliente)
4: confirmaCli(cliente)
5: idPr
Operações sobre Produtos
5.1: prod := procuraProd(arm, lentes)
break
6: rejeitaProd(prod)
7: ad
Operações sobre Encomendas
7.1: regista(cliente, prod)
ref
ImprimirTalão


Actor2
UI
: MyFacade
: Clientes
: Produtos
: Encomendas
1: idCliente(nome, datan)
1.1: listaClientes := getClientes(nome, datan)
1.1.1: getClientes(nome, datan)
alt
1.2: cliente := getCliente(listaClientes)
[listaClientes tem tamanho 1]
1.3: cliInfo := procuraCliente(cliente)
1.4: mostra informação de único cliente
1.3.1: procuraCliente(cliente)
[else]
1.5: mostra lista
opt
[Funcionário escolhe criar novo cliente]
ref
Criar Cliente
2: seleciona(cliente)
2.1: cliInfo := procuraCliente(cliente)
2.1.1: procuraCliente(cliente)
2.2: cliInfo
break
3: rejeitaCli(cliente)
4: confirmaCli(cliente)
5: idProduto(arm, lentes)
5.1: prod := procuraProd(arm, lentes)
5.1.1: procuraProduto(arm, lentes)
5.2: prod
break
6: rejeitaProd(prod)
7: aceitaProd(prod)
7.1: regista(cliente, prod)
7.1.1: registaEncomenda(cliente, prod)
ref
ImprimirTalão

Diagramas da UML 2.x



Diagramas de Package



- À medida que os sistemas software se tornam mais complexos e o número de classes aumenta:
  - Torna-se difícil efectuar a gestão das diversas classes
  - A identificação de uma classe e o seu papel no sistema dependem do contexto em que se encontram
  - É determinante conseguir identificar as dependências entre as diversas classes de um sistema.

- Em UML os agrupamentos de classe designam-se por *packages* (pacotes), que correspondem à abstracção de conceitos existentes nas linguagens de programação:
  - Em Java esses agrupamentos são os *packages*
  - Em C++ designam-se por *namespaces*

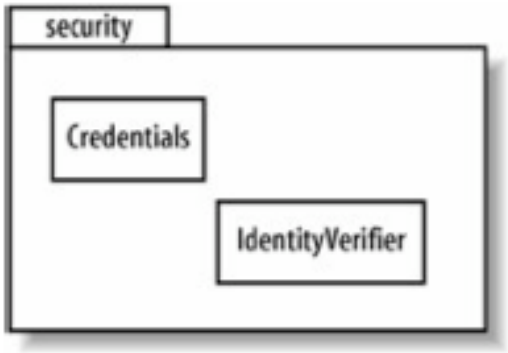
- A identificação das dependências entre os vários packages permite que a equipa de projecto possa descrever informação importante para a evolução do sistema

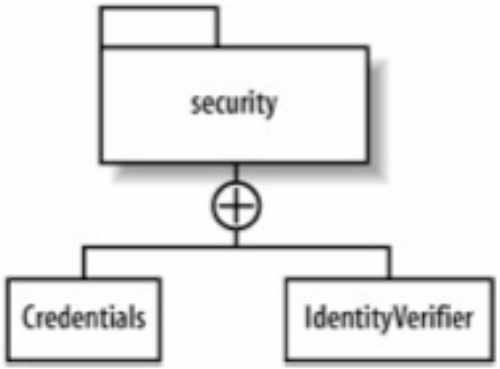
Diagramas de Package (cont.)

- Um diagrama de package representa os packages e as relações entre packages
- Os diagramas de packages em UML representam mais do que relações entre classes:
  - Packages de classes (packages lógicos) - em diagramas de classes
  - Packages de componentes – em diagramas de componentes
  - Packages de nós – em diagramas de distribuição
  - Packages de casos de uso – em diagramas de *use cases*

Diagramas de Package (cont.)

- Um package é desta forma o dono de um conjunto de entidades, que vão desde outros packages, a classes, interfaces, componentes, use cases, etc.
- Essa forma de agregação pode ser representada de diversas formas:



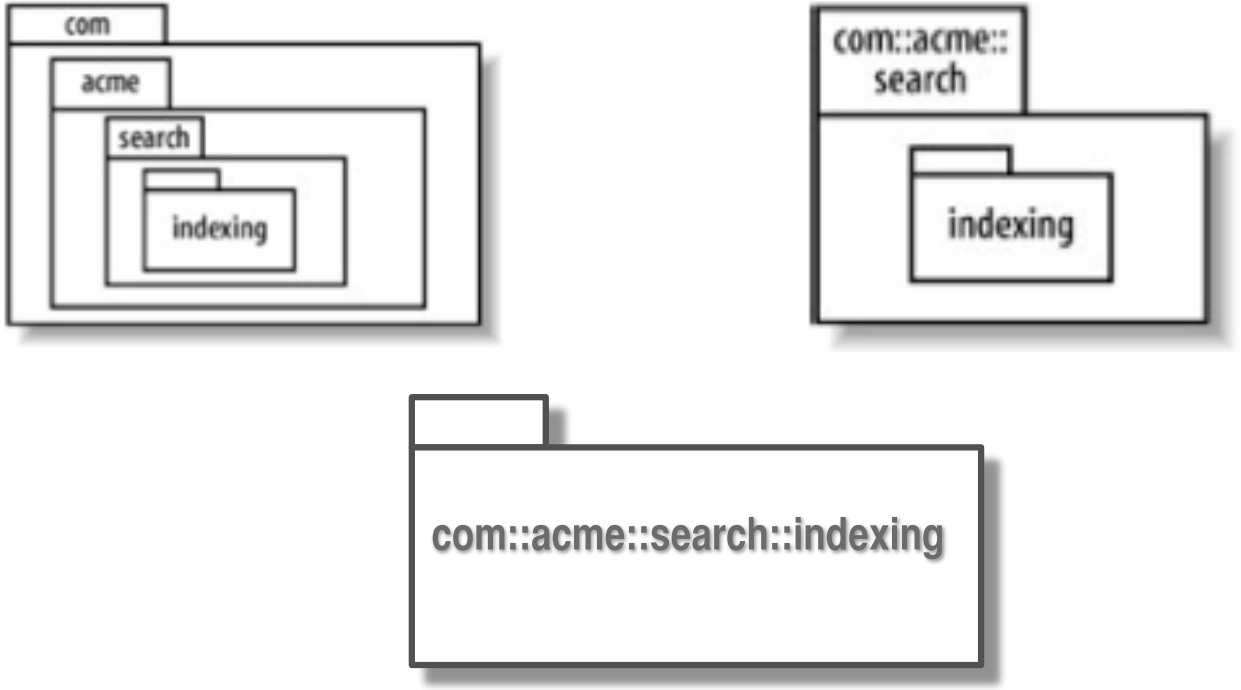



```
package security;

public(?) class Credentials {....}
```

Diagramas de Package (cont.)

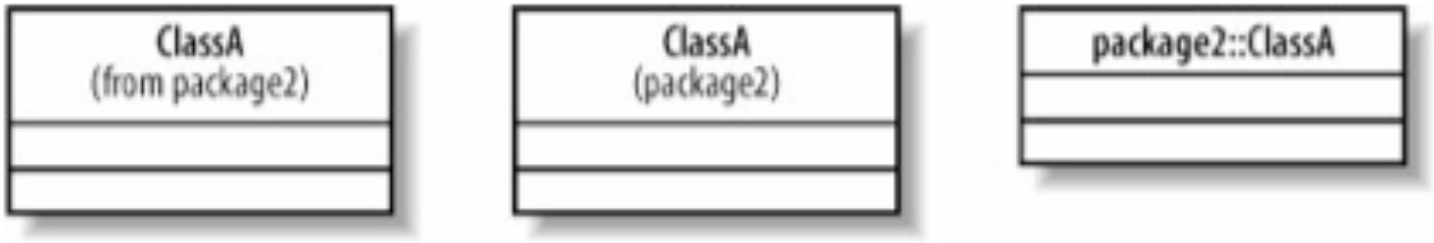
- Existem várias formas de representar a agregação de packages
- Essas regras definem também a qualificação dos nomes das classes



Os três diagramas representam a mesma informação

Diagramas de Package (cont.)

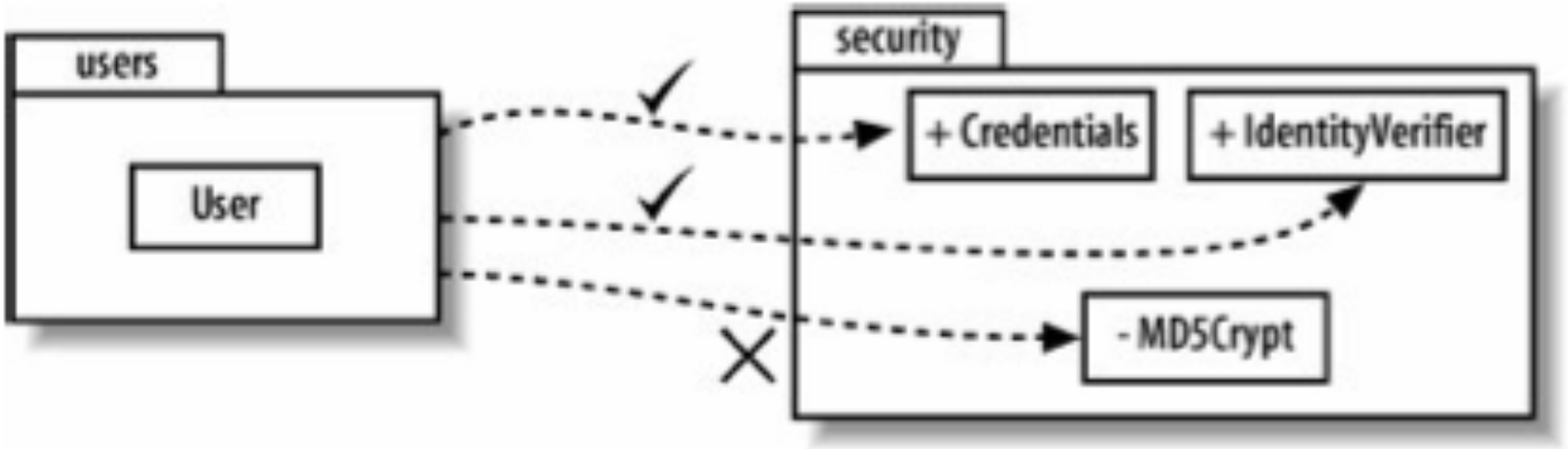
- Por uma questão de identificação do contexto (o *namespace*) de uma classe é usual que as ferramentas identifiquem no diagrama de classes, qual é o agrupamento lógico a que uma classe pertence.



- A notação **nomePackage::nomeClasse**, identifica (qualifica) inequivocamente uma classe.
  - Tal como em Java com a utilização do nome completo (ex: java.lang.String)
  - Permite que existam classes com nome idêntico nas diversas camadas que constituem uma aplicação

Diagramas de Package (cont.)

- A visibilidade dos elementos de um package utiliza uma notação e semântica similar à vista nos diagramas de classe.
  - "+" - público
  - "-" - privado
  - "#" - protected (só acessível/visível por filhos do package em causa)



Diagramas de Package (cont.)

- Existem várias formas de especificar *dependências* entre pacotes de uma arquitectura lógica
  - *Dependência (simples)* - quando uma alteração no package de destino afecta o package de origem

  - <<*import*>> - o package origem importa o contéudo do package destino que é por este exportado. Logo, não necessita de qualificar completamente os elementos importados (na forma packageA::classeB).
    - Este mecanismo é similar ao mecanismo Java que permite fazer import de um package

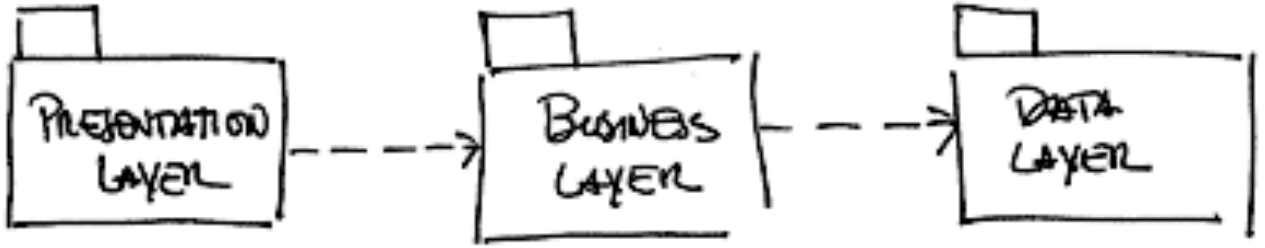
```
import java.util.*;
```

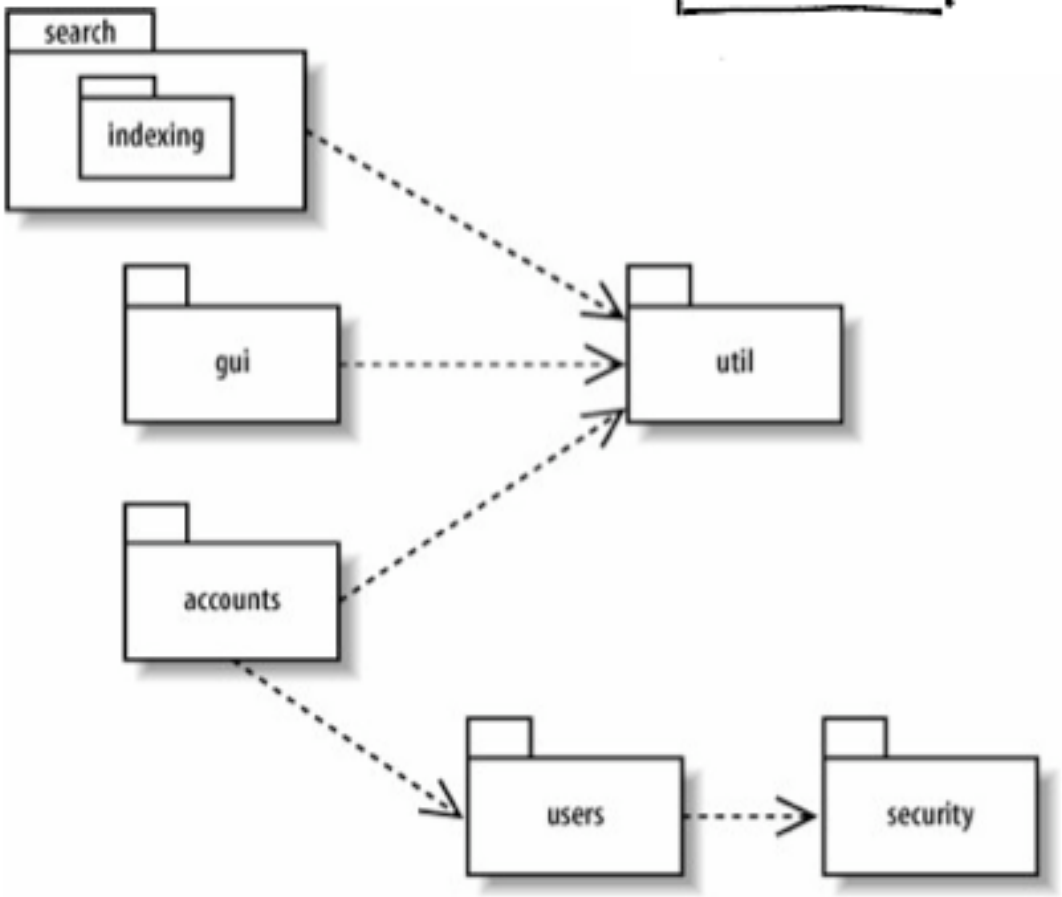
  - <<*access*>> - o package origem acede a elementos exportados pelo package destino, mas necessita de qualificar completamente os nomes desses elementos.

  - <<*merge*>> - o package origem é *fundido* com o package destino para gerar um novo.

Diagramas de Package (cont.)

- Exemplos de utilização de dependência:





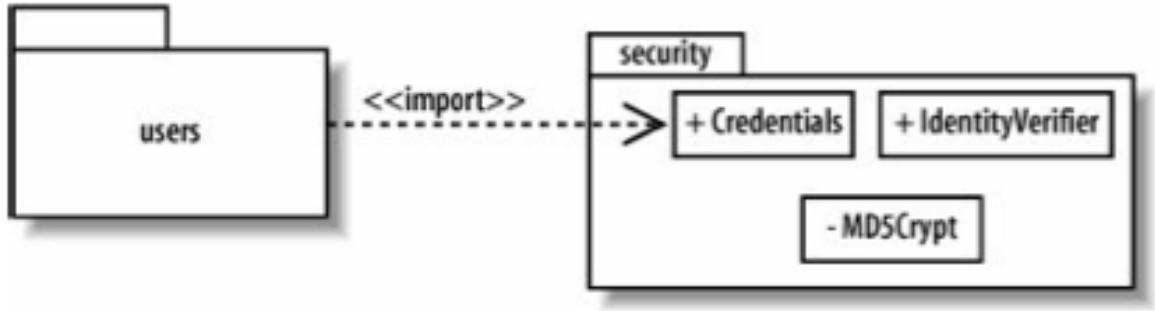
Diagramas de Package (cont.)

- Utilização de <<*import*>>

  - O package *users* importa todas as definições públicas de *security*, apenas por nome

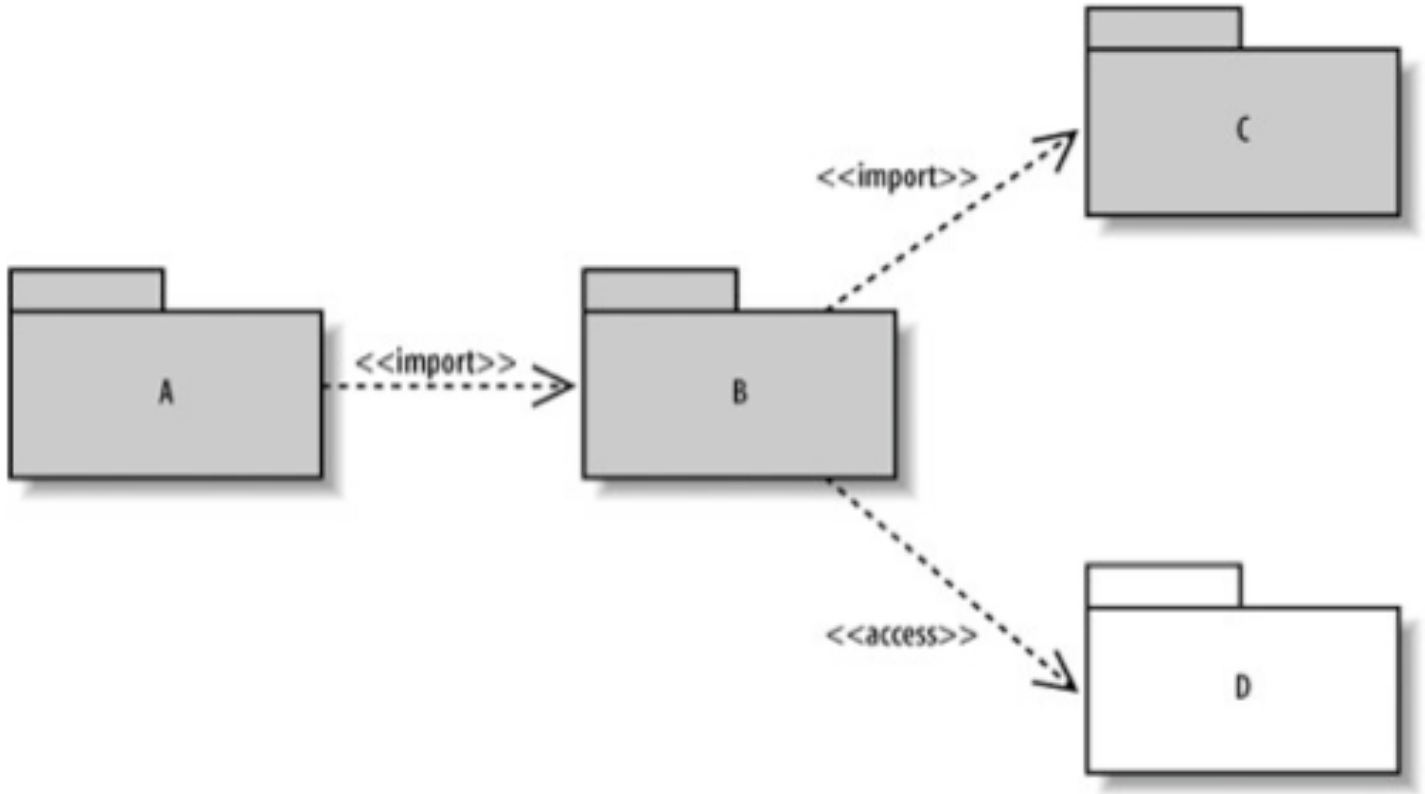
  - Definições privadas de packages importados não são acessíveis por quem importa.

  - O package *users* apenas importa a classe *Credentials* do package *security*

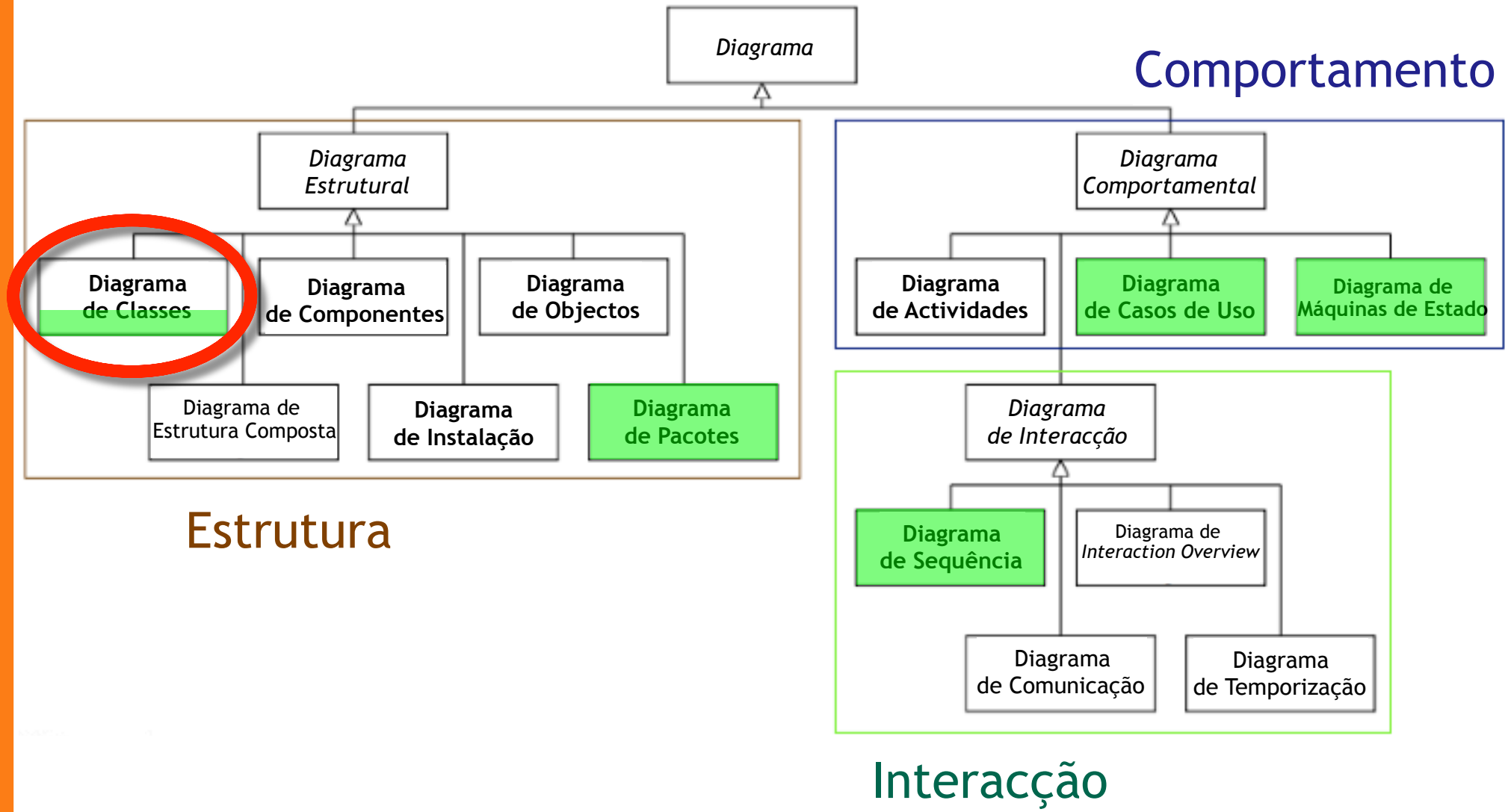


Diagramas de Package (cont.)

- Utilização de *<<import>> e <<access>>*
  - O package *B* vê os elementos públicos em *C* e *D*.
  - *A* importa *B*, pelo que vê os elementos públicos em *B* e em *C* (porque este é importado por *B*)
  - *A* não tem acesso a *D* porque *D* só é acedido por *B* (e não é importado).



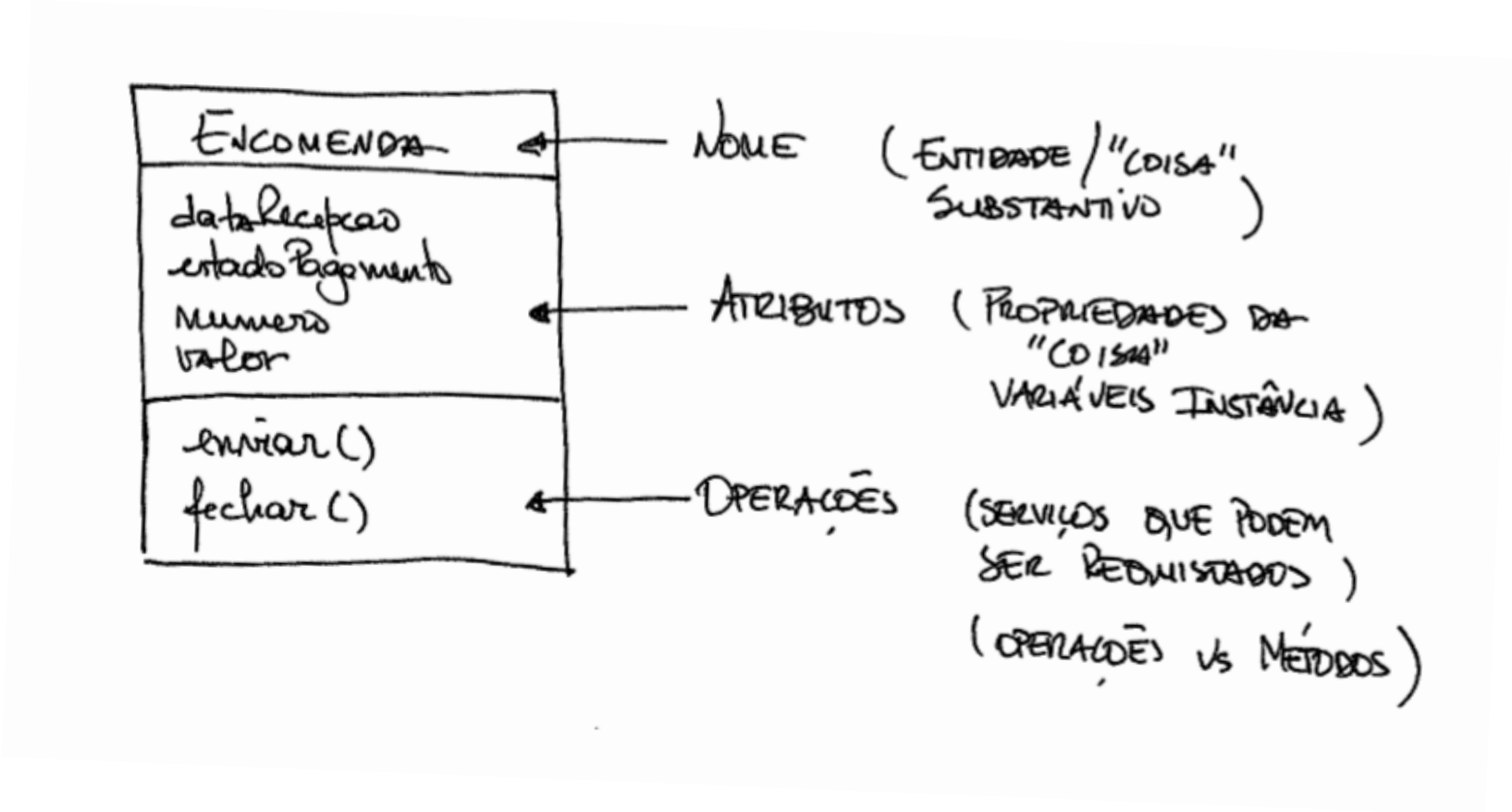
Diagramas da UML 2.x



Revisão do conceito de classe

- Base de um qualquer sistema OO
- Cada classe descreve um conjunto de objectos com a mesma estrutura de dados e comportamento

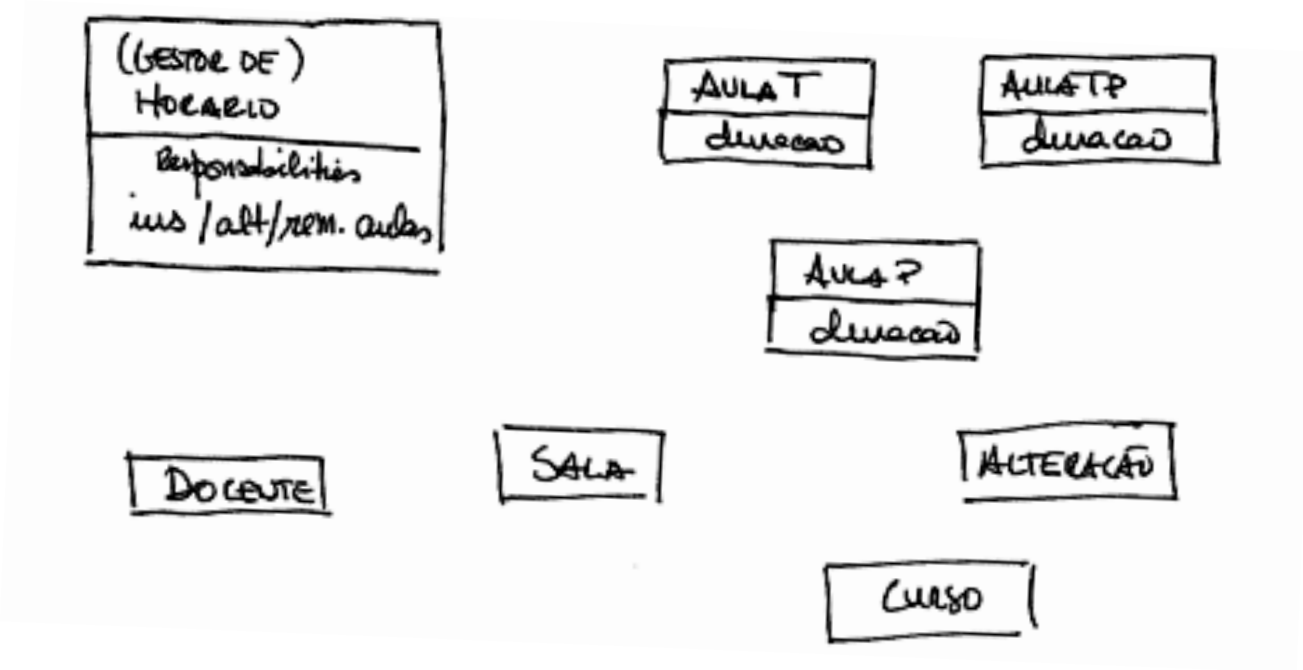
- Exemplo:



Níveis de modelação

- Podemos considerar 3 níveis de modelação:
  - Conceptual
  - Especificação
  - Implementação

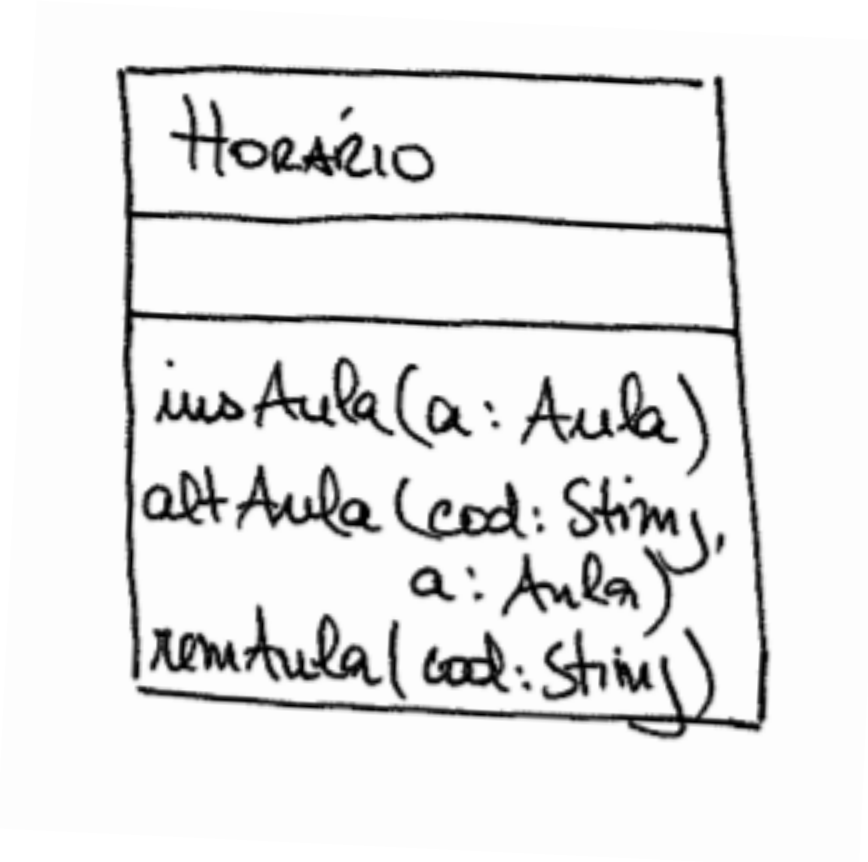
- Nível Conceptual
  - Representação dos conceitos no domínio de análise
  - Não corresponde necessariamente a um mapeamento directo para a implementação
  - Pensar nas responsabilidades de cada abstracção



Níveis de modelação

- Nível de especificação
  - Definição das interfaces (API's)
  - Substituir as responsabilidades por operações/atributos que as satisfaçam

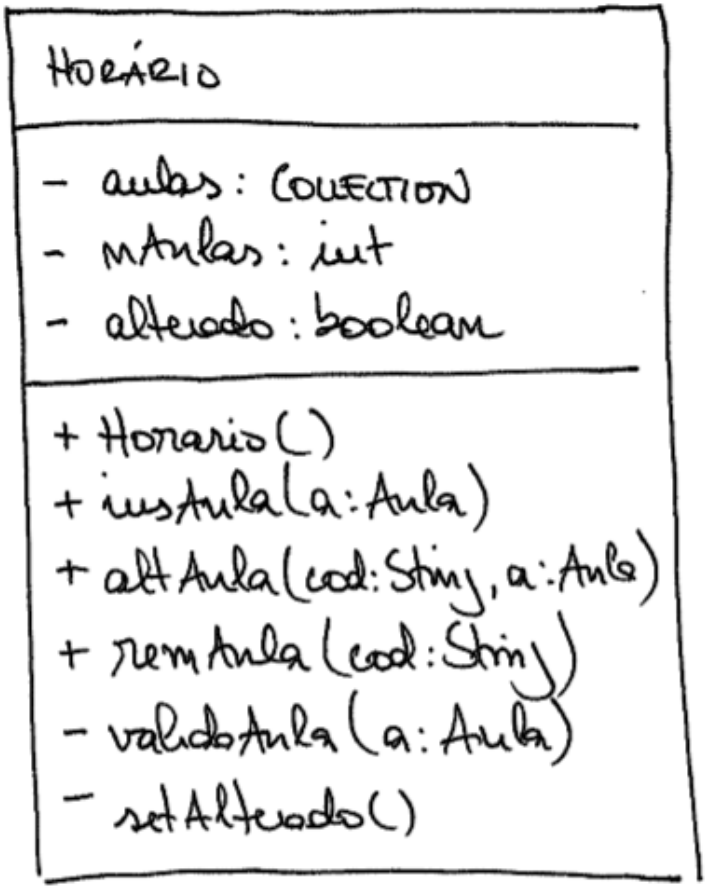
  - Exemplo:



Níveis de modelação

- Nível de implementação
  - Definição concreta das classes a implementar - *geração de código*
  - Definição dos relacionamentos estruturais entre as entidades

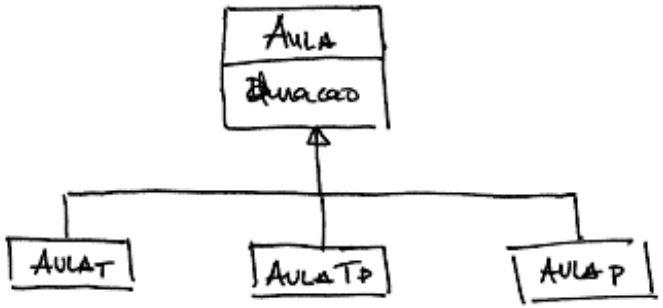
  - Exemplo:



Níveis de visibilidade:
- \- - privado
- \+ - publico
- # - protected
- " " - package

Relações entre as classes

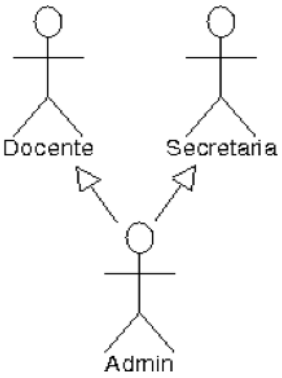
- Podemos considerar:
  - Generalização
  - Dependência
  - Associação



- **Generalização**
  - Relação entre uma coisa mais geral (pai/super-classe) e uma coisa mais específica (filho/sub-classe)
  - Uma relação de “is a” do ponto de vista dos tipos de dados

  - Vantagens:
    - Substitubilidade
    - Polimorfismo (sub-classe redefine o método)
    - Herança simples vs Herança multipla

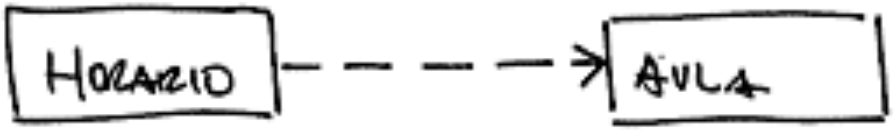
  - Também pode ser utilizada para use cases, actores, etc.

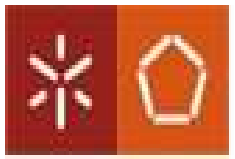


Relações entre as classes

- **Dependência**
  - Utiliza-se para mostrar que a origem da relação *usa* o destino
  - Uma alteração no destino (o *usado*) pode alterar a origem (quem *usa*)

  - Exemplos: parâmetro num método, variável global





Quando se modela um **sistema**, alguns objectos estarão certamente relacionados com outros, através de relacionamentos diferentes, mas que semanticamente devem ser bem definidos.

Há 5 tipos de **associações** expressáveis em Diagramas de Classes de UML:

1) Associação bi-direccional;

2) Associação uni-direccional;

3) Agregação de Classes (Agregação básica e Composição);

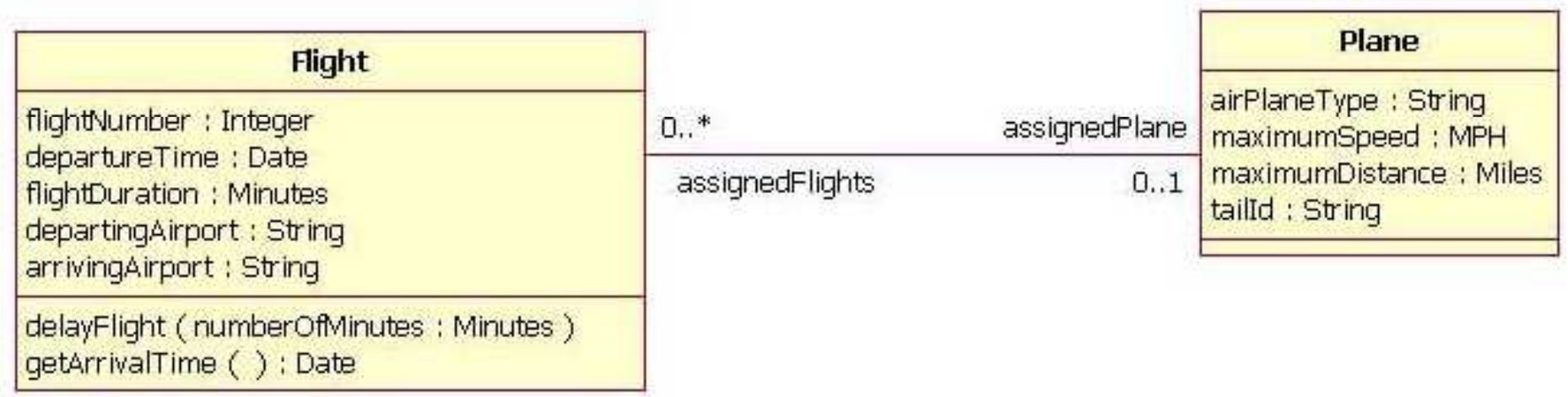
4) Associações Reflexivas;

5) Associação com Classes de Associação;

DC: Relacionamentos

Associação bi-direccional

Uma associação bidireccional é especificada por uma linha sólida que une duas classes, tendo no seu início e fim atributos de multiplicidade. Os papéis de cada classe na associação são descritos textualmente.

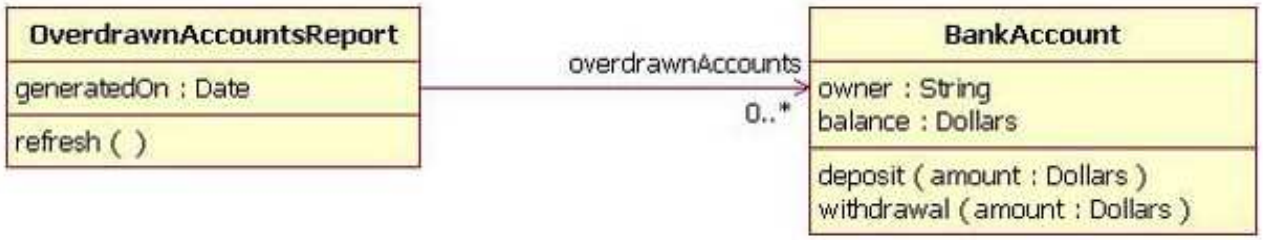


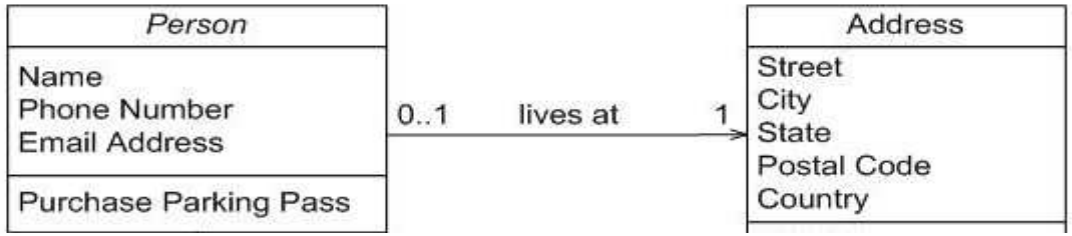
Ler: Cada instância de Voo tem a si "atribuído" (*assigned*), num dado momento, ou 0 ou 1 instância de Avião. Cada instância de Avião tem a si atribuídos 0 ou mais voos.

DC: Relacionamentos

Associação uni-direccional

Uma associação unidireccional é especificada por uma linha sólida com seta, indicando que apenas uma das classes "conhece" o seu relacionamento com a outra (e o seu significado).



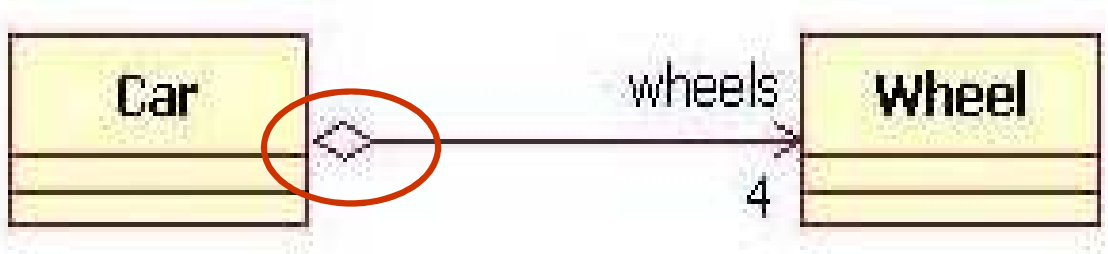


Em geral, ao nível conceptual é preferível definir relações bi-direccionais (é sempre posssível responder à "questão inversa", ou seja, de uma entidade sabemos da outra e vice-versa). Quando passamos para o nível de especificação, começamos a ter que definir direcções.

DC: Agregação Básica

Agregação básica

Numa agregação básica, especifica-se que há uma associação entre duas classes tão forte que uma (ou mais delas) fazem parte da definição e estrutura interna da outra, e até em certo número (recordar que uma classe define objectos com dada estrutura representada por variáveis de instância que são objectos de outras classes).

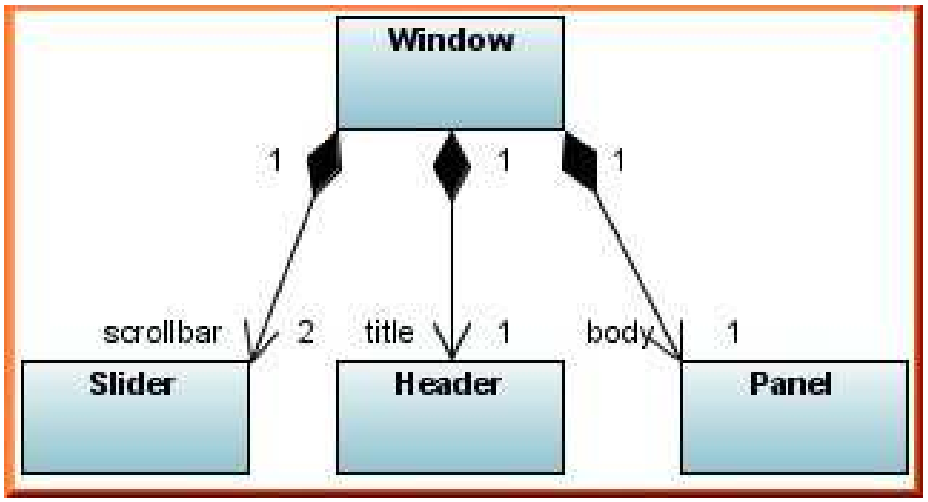


Aspecto importante da semântica: Embora estando cada instância de Carro associada por agregação básica (ou agregação) a 4 instâncias de Roda, se tal instância de Carro for destruída as instâncias de Roda são independentes e, portanto, permanecem "vivas", não sendo destruídas.

DC: Composição

Agregação por Composição

Numa composição, especifica-se que há uma associação entre duas classes tão forte que uma (ou mais delas) fazem parte da vida da outra e, portanto, quando a instância da classe superior é destruída, todas as instâncias das classe compostas devem também ser destruídas. As classes que compõem não têm sentido sem a sua class superior.

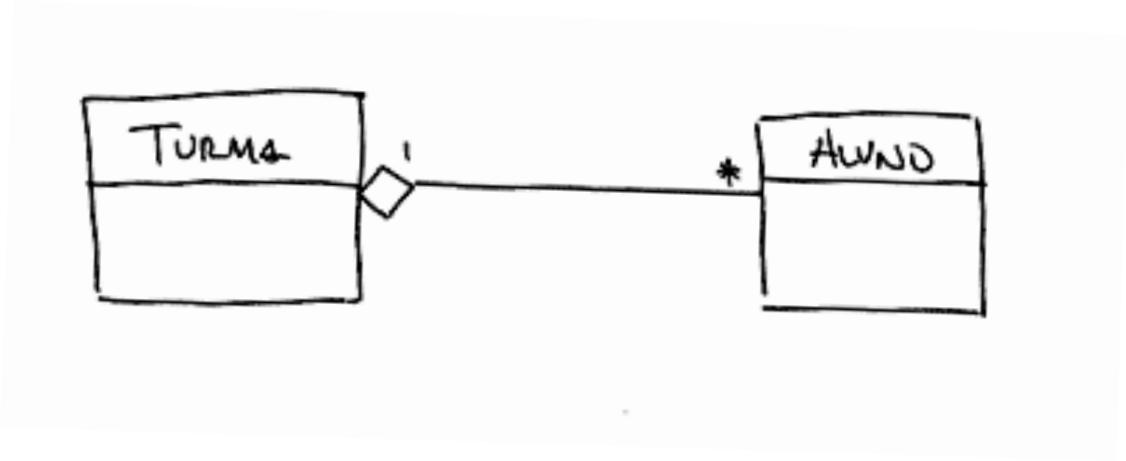


Decisões do tipo:

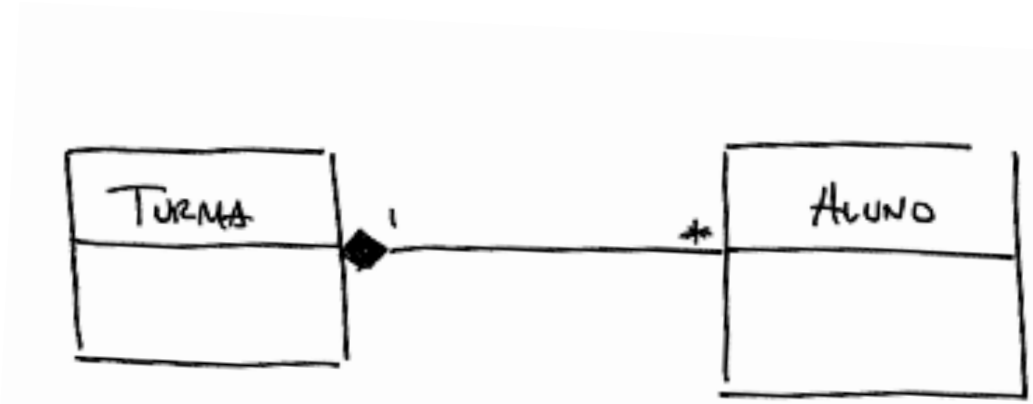
Se uma Biblioteca for fechada os livros que lhe estavam associados são destruídos ou não ?

Relações entre as classes

- Agregação



- Composição

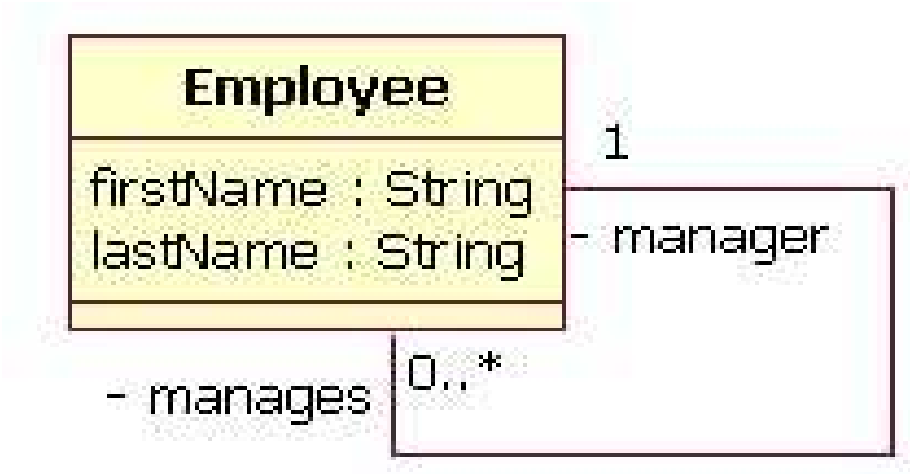


Quais as diferenças entre uma e outra abordagem?

DC: Outras Associações

Associação Reflexiva

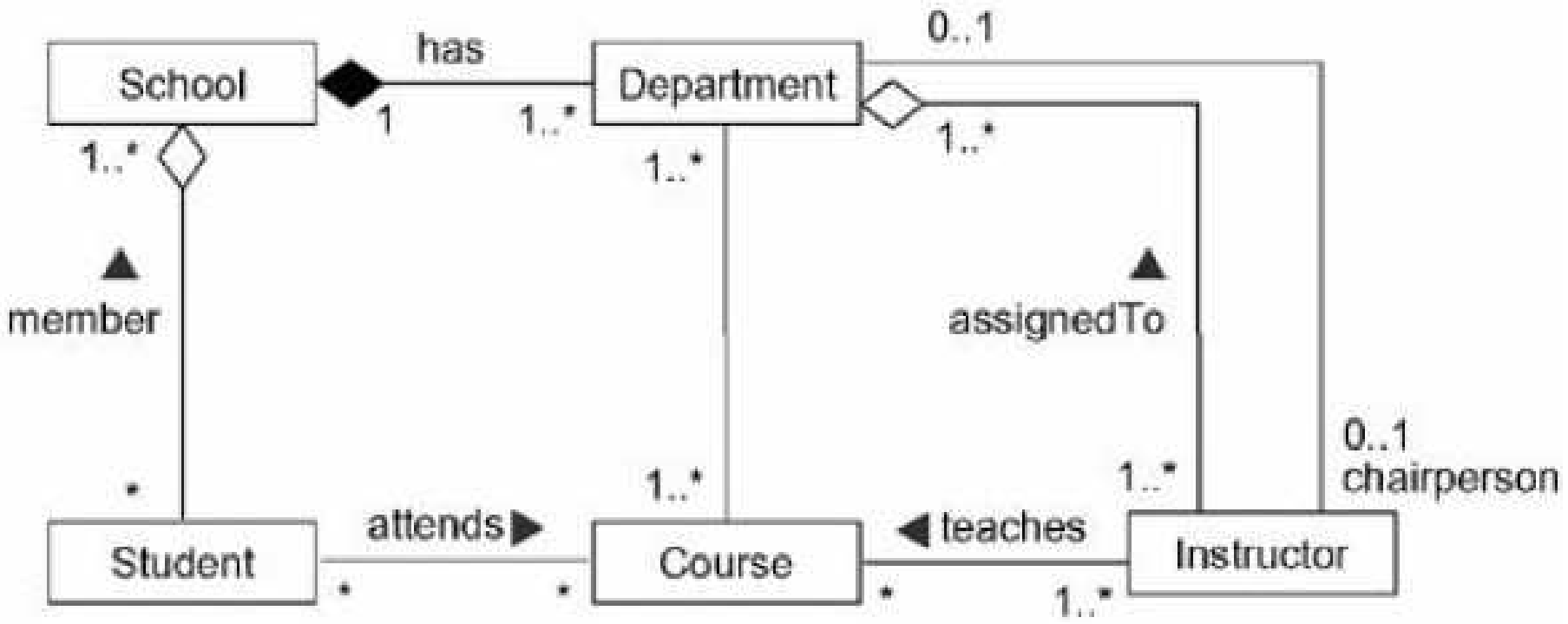
Numa associação reflexiva, especifica-se que uma instância de uma classe pode referenciar uma outra da mesma classe, ainda que entre elas haja uma dada semântica de relacionamento que pode derivar de papéis semânticos diferentes (cf. Actores e generalização).



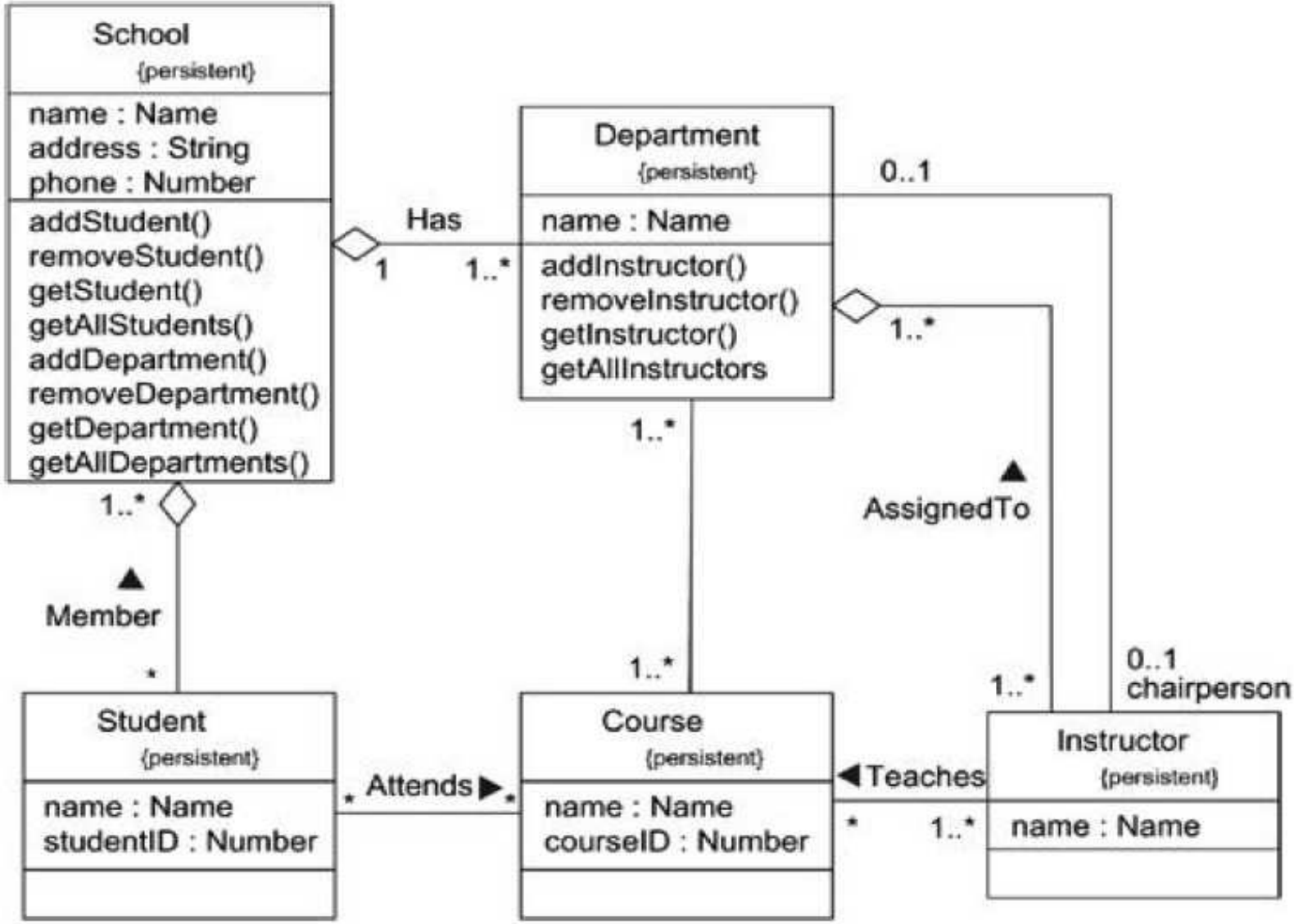
Uma instância de Empregado pode ser "manager" de 0 ou mais "empregados". Quer tenha ou não sentido, é o que está especificado.

Universidade do Minho
Departamento de Informática

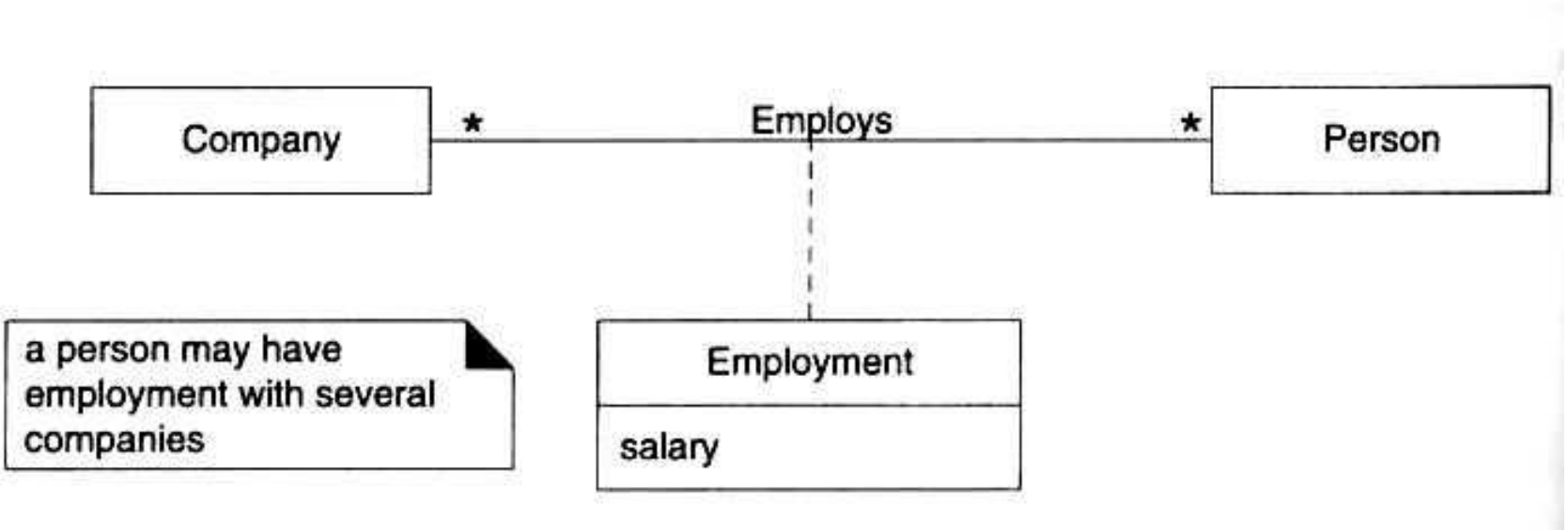
DC: Exemplo - Interpretação



DC: Exemplo - Interpretação

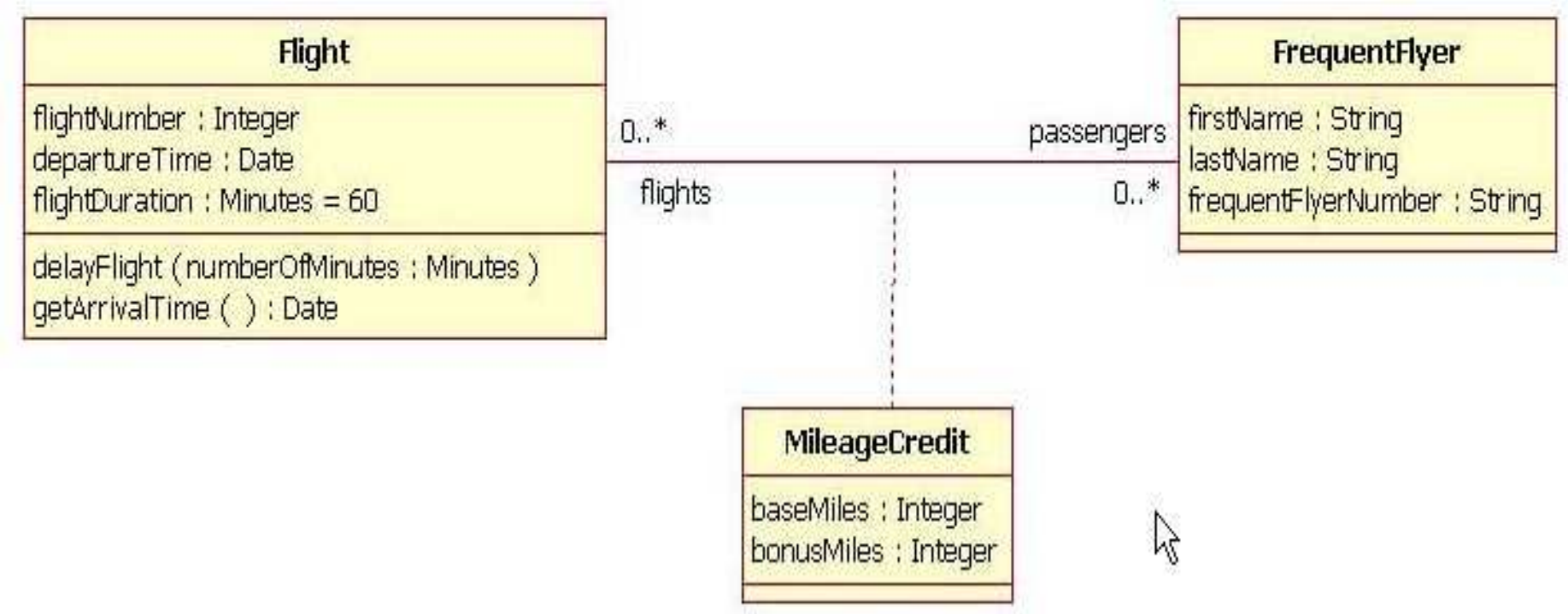


DC: Classes de Associação



Quando se relaciona uma instância de "Empresa" com uma instância de "Pessoa" deve "juntar-se" uma instância de "Emprego", que transporta consigo o valor do salário. Assim, cada "Pessoa" poderá ter um salário distinto por emprego, ou seja, por cada uma das suas associações a uma empresa.

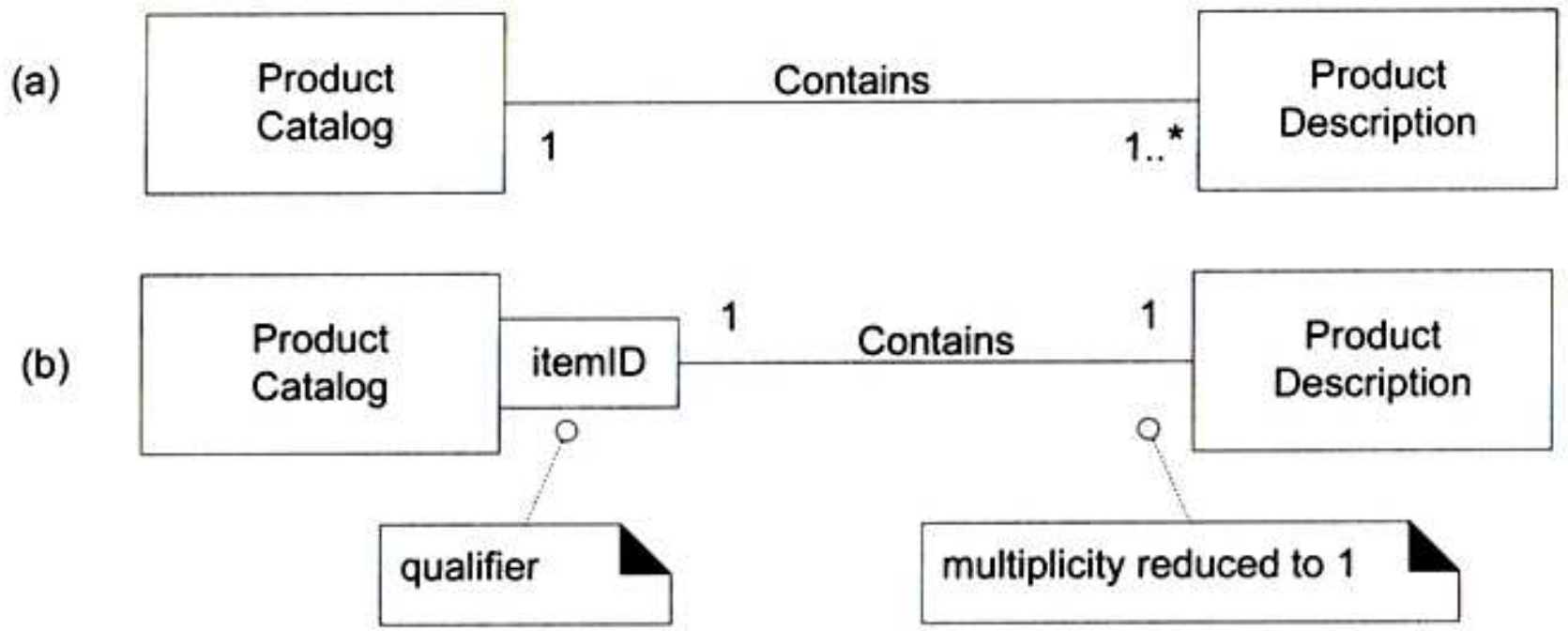
DC: Classes de Associação



Quando se relaciona uma instância de "Voo" com uma instância de "Passageiro Frequente" deve "juntar-se" uma instância de "Crédito em Milhas" (isto porque a relação é O..*).

Universidade do Minho
Departamento de Informática

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".

Modelação Estrutural

Sumário

- Diagramas de *Package*
  - Representação de *Packages*
  - Relações entre packages
    - Composição: diferentes representações gráficas, qualificação, visibilidade
    - Dependências: simples, «import», «access», «merge»

- Diagramas de Classe II
  - Níveis de modelação
  - Relações entre as classes
    - Herança/especialização
    - Dependências
    - Associação bidireccional vs. unidireccional
    - Agregação vs. Composição vs. Associação simples
    - Classes de associação
    - Associações qualificadas