

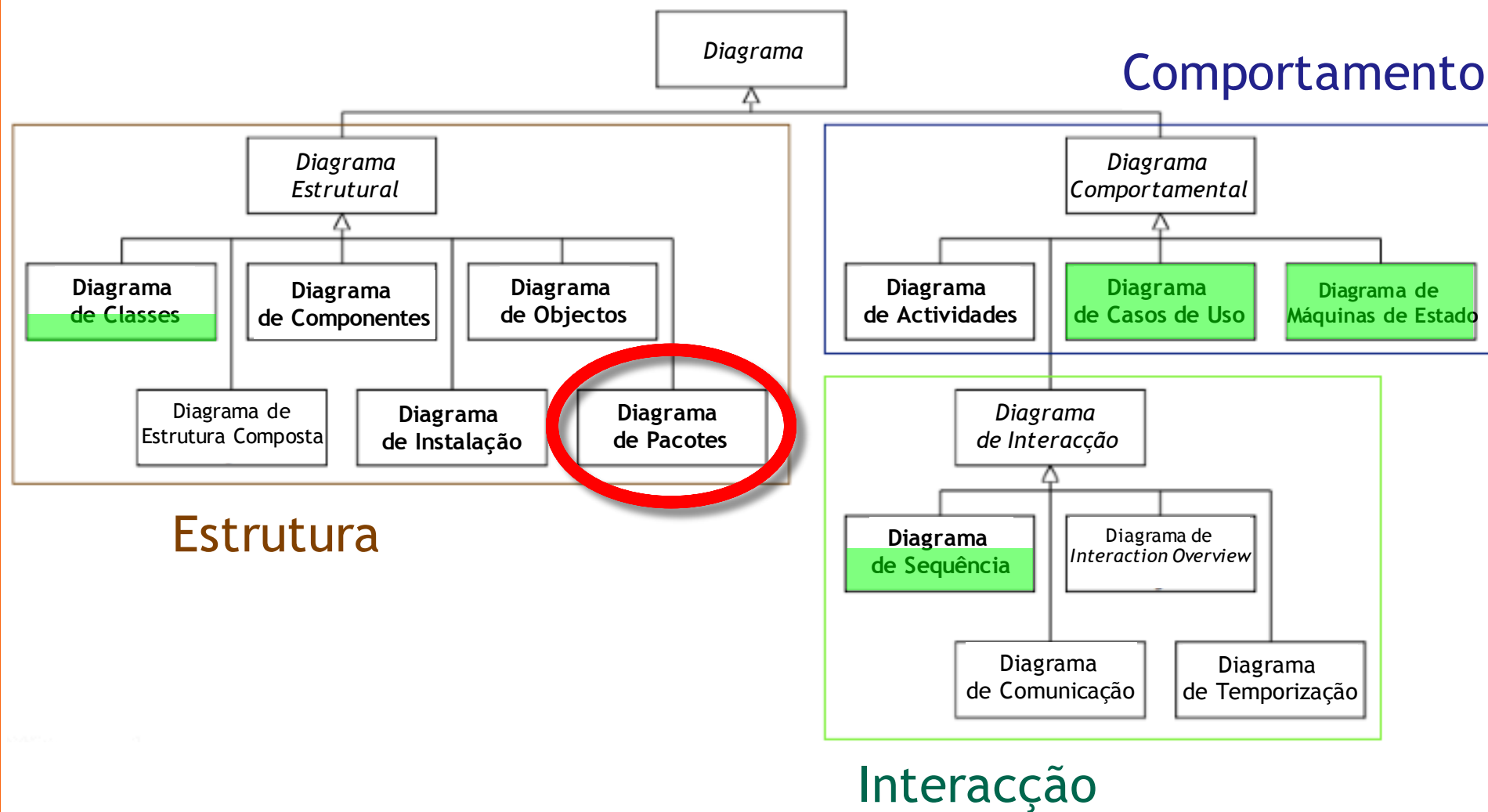


Desenvolvimento de Sistemas Software

Aula Teórica 14: Modelação Estrutural - Diagramas de Package



Diagramas da UML 2.x



Diagramas de Package

Estamos a procurar trabalhar por antecipação e organizar as classes desde o início!...

- 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



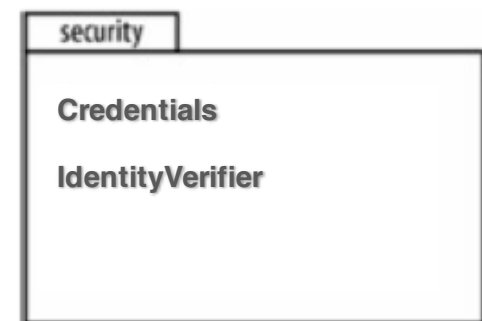
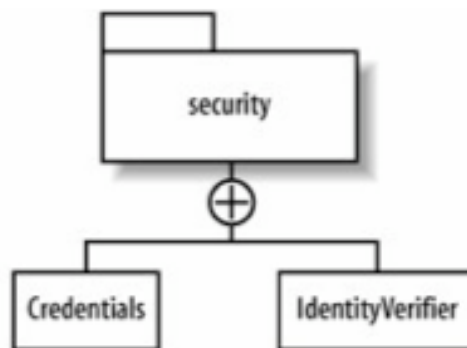
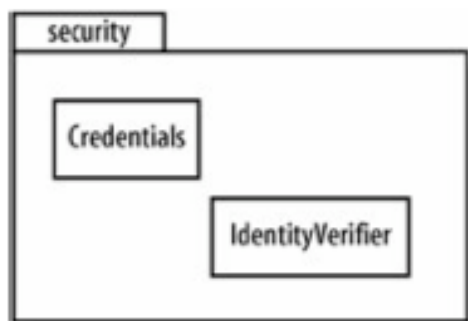
Diagramas de Package (cont.)

- 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:

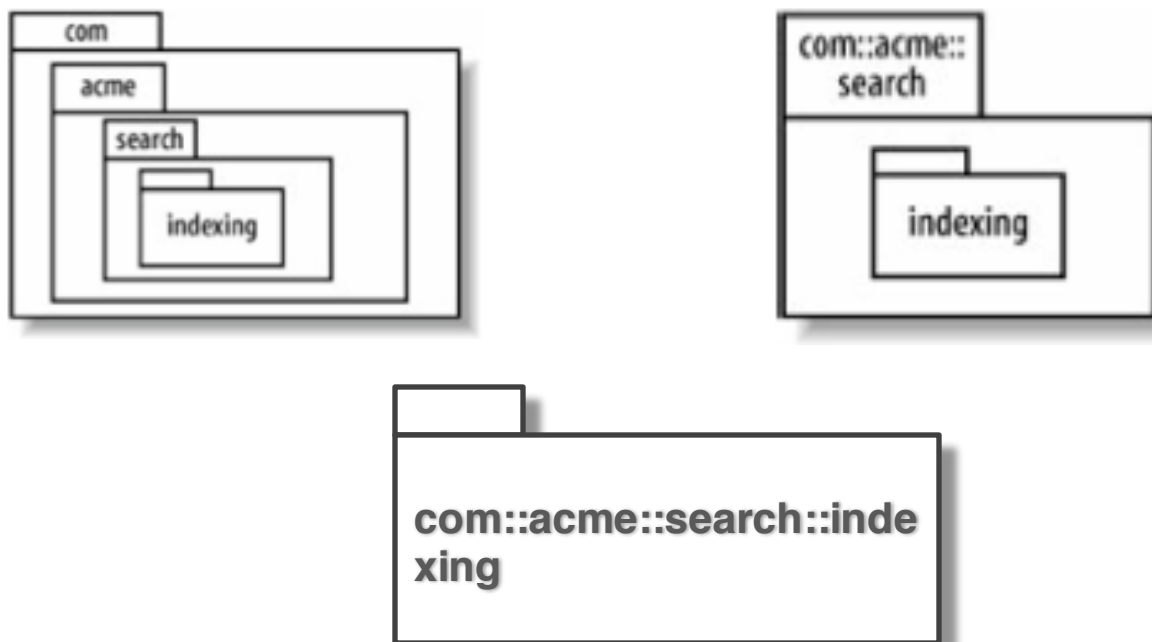


```
package security;  
public/private/protected(?) class Credentials {  
    ...  
}
```



Diagramas de Package (cont.)

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

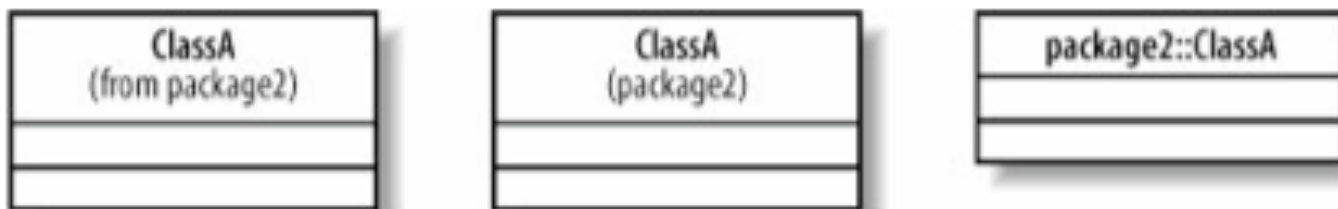


Os três diagramas representam a mesma informação



Diagramas de Package (cont.)

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

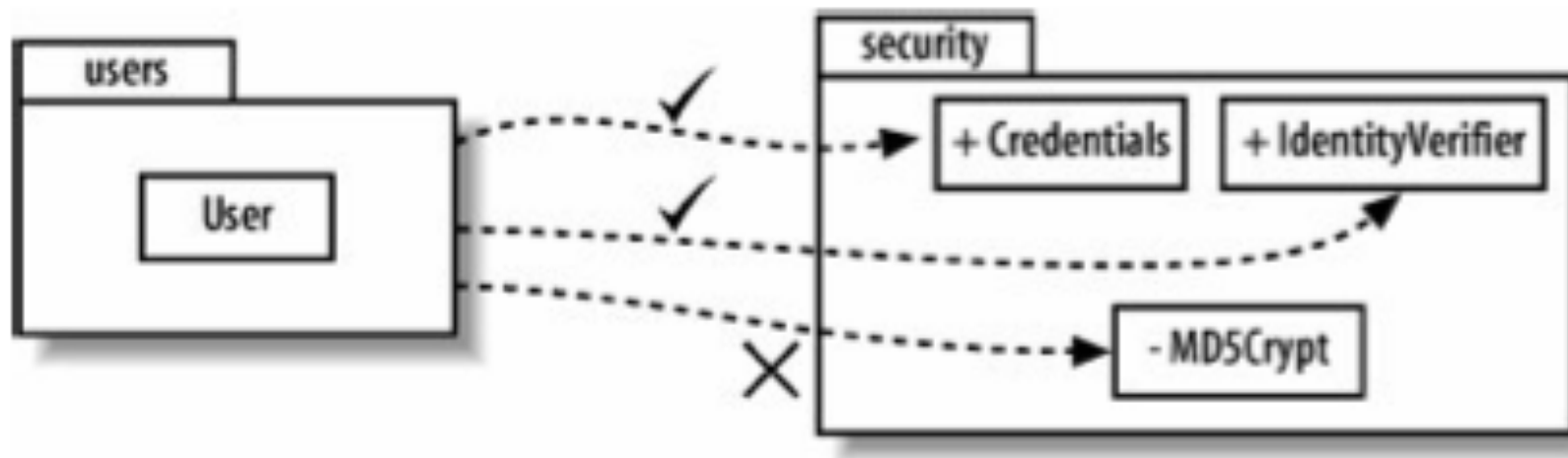


- A notação **nomePackage::nomeClasse**, identifica (qualifica) inequivocamente uma classe.
 - Tal como em Java com a utilização do nome completo (ex: `java.lang.String`)
 - Permite que existam classes com nome idêntico nas diversas camadas que constituem uma aplicação



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



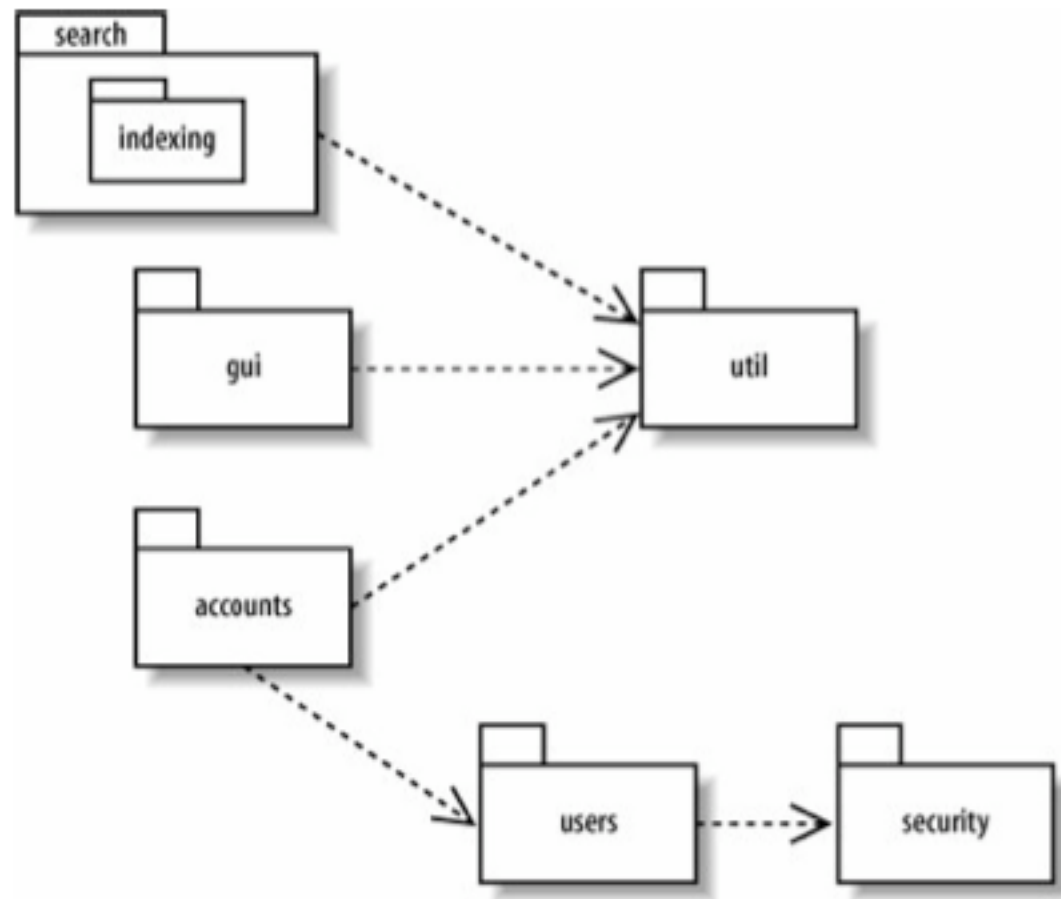
- `<<import>>` - o package origem importa o conteúdo do package destino que é por este exportado. Logo, não necessita de qualificar completamente os elementos importados (na forma `packageA::classeB`).
 - Este mecanismo é similar ao mecanismo Java que permite fazer import de um package

```
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 dependências simples:



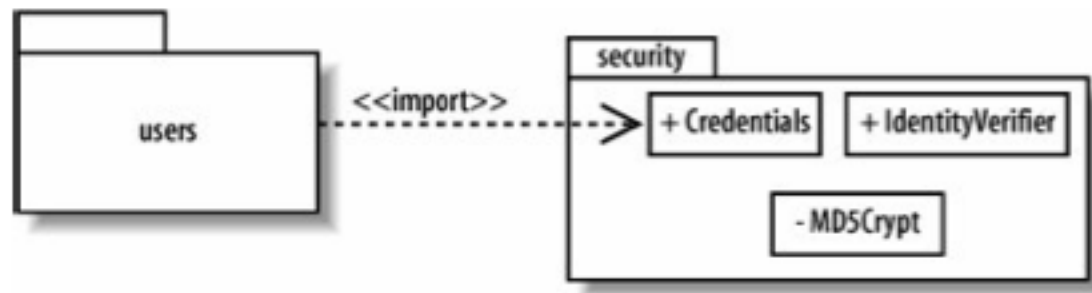


Diagramas de Package (cont.)

- Utilização de `<<import>>`
- O package *users* importa todas as definições públicas de *security*, apenas por nome



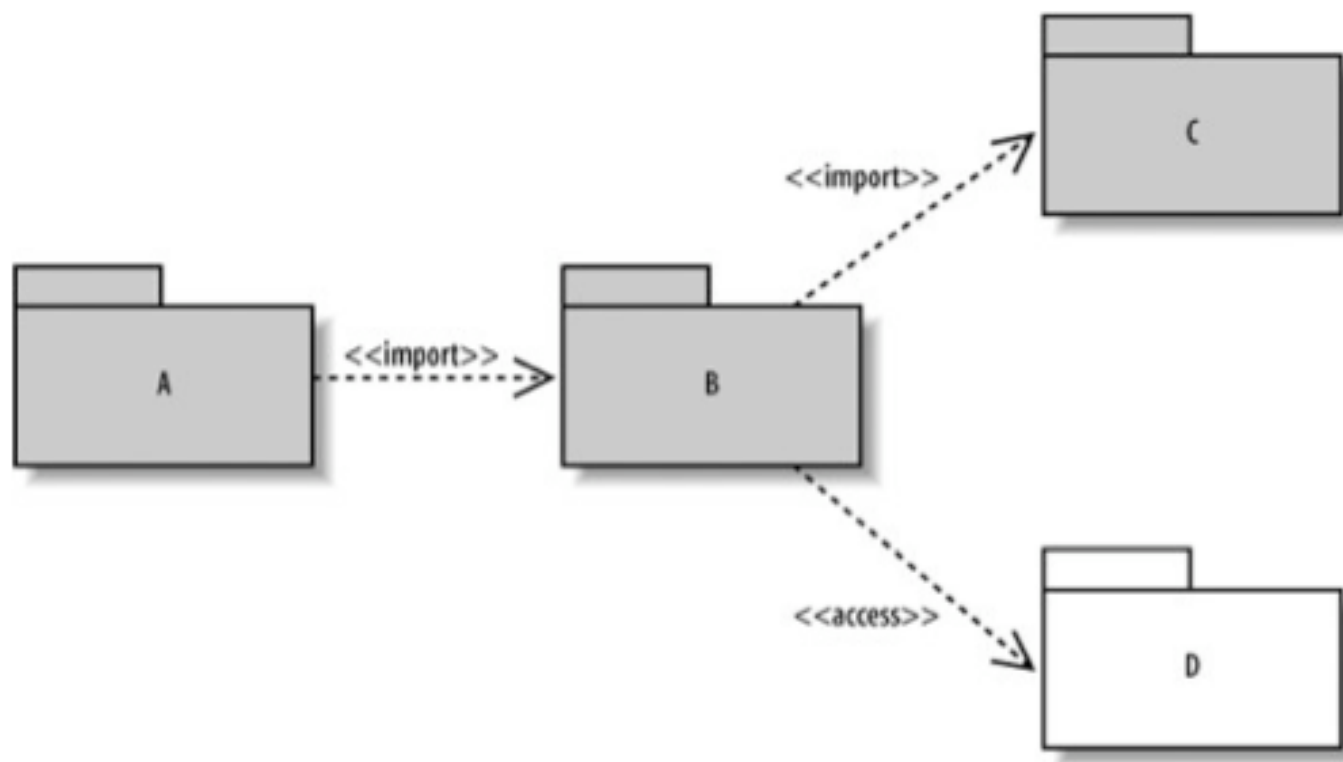
- Definições privadas de packages importados não são acessíveis por quem importa.
- O package *users* apenas importa a classe *Credentials* do package *security*



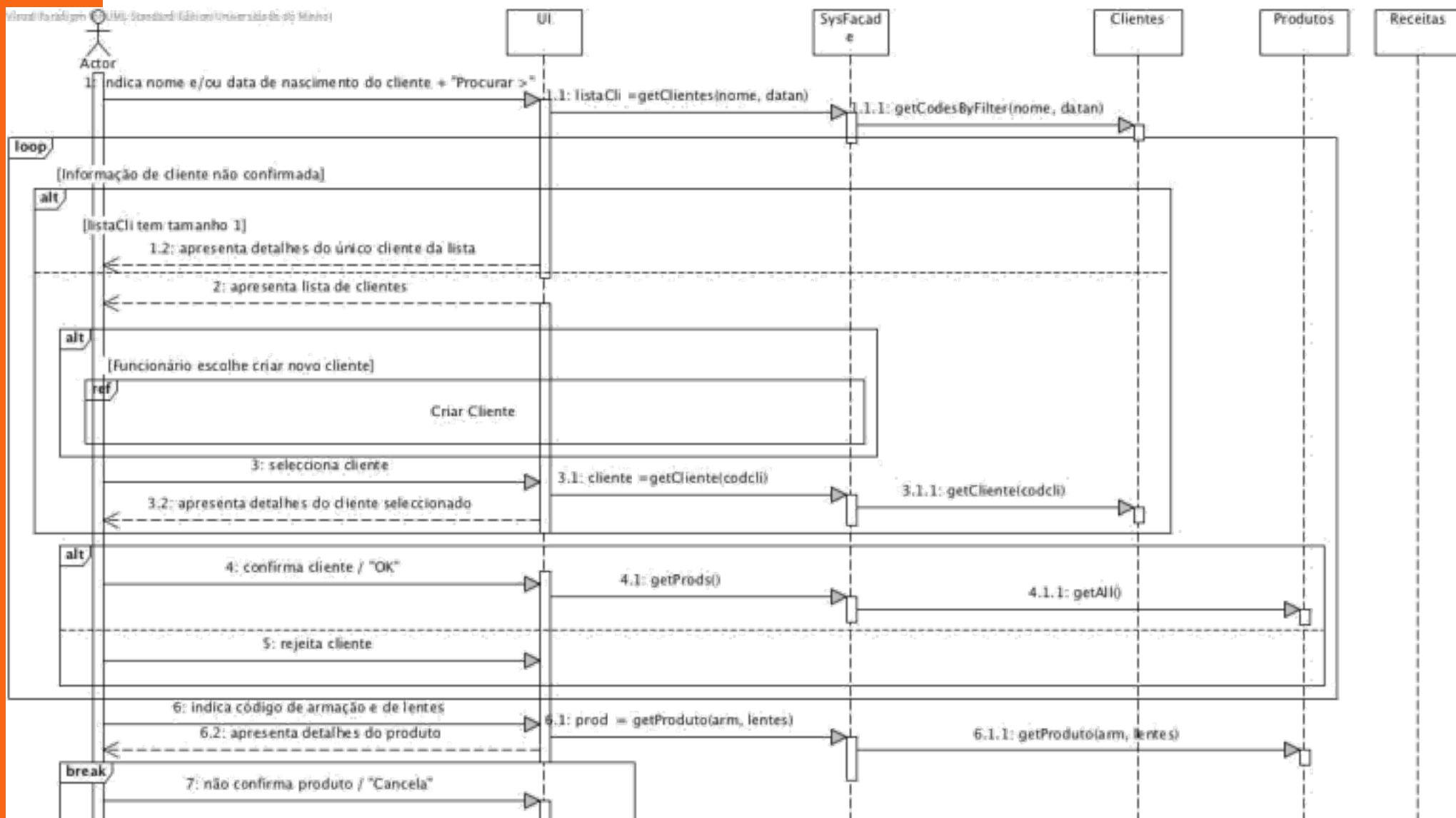


Diagramas de Package (cont.)

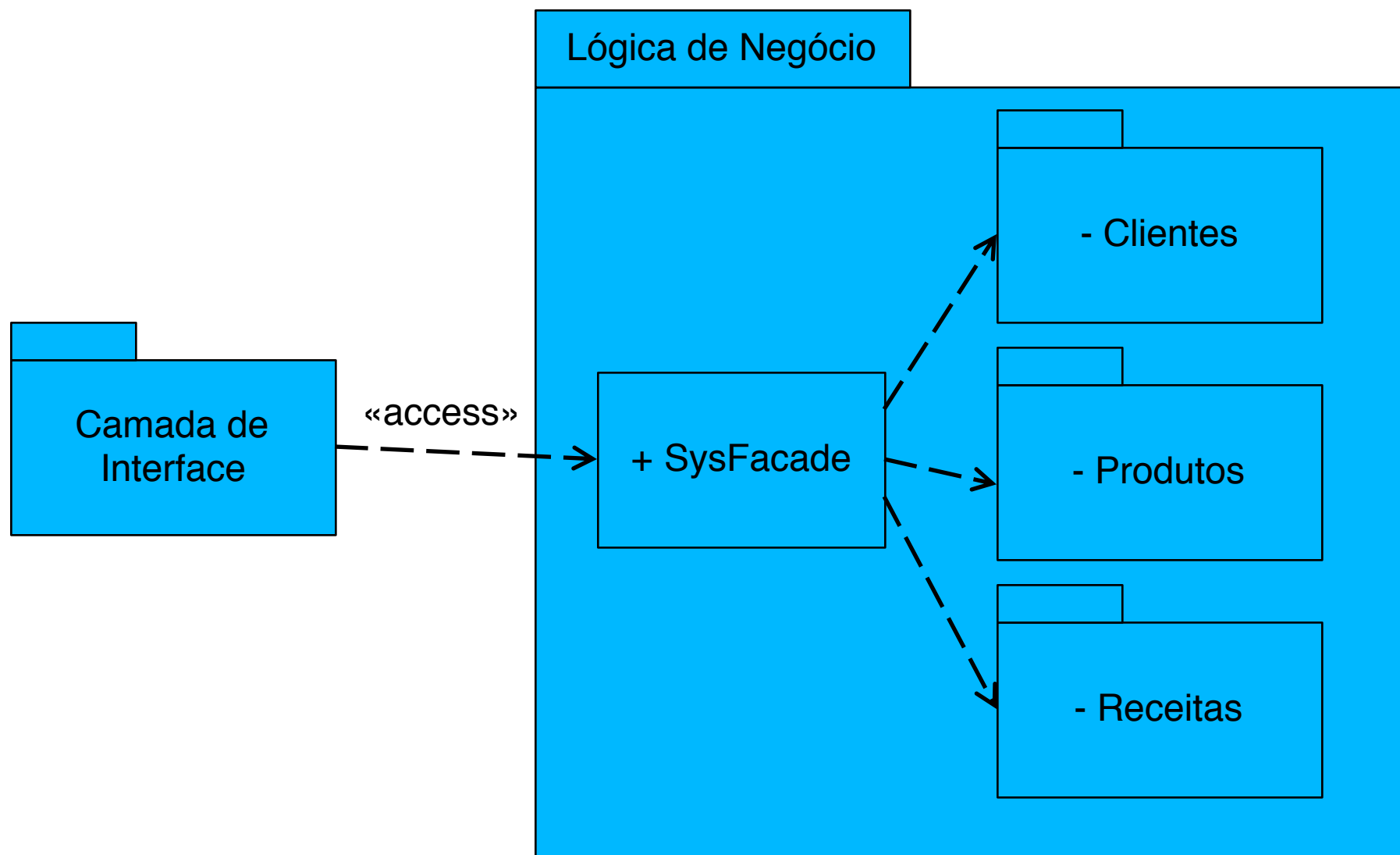
- Utilização de `<<import>>` e `<<access>>`
 - O package *B* vê os elementos públicos em *C* e *D*.
 - *A* importa *B*, pelo que vê os elementos públicos em *B* e em *C* (porque este é importado por *B*)
 - *A* não tem acesso a *D* porque *D* é apenas acedido por *B* (não é importado).



Relembrando o exemplo...



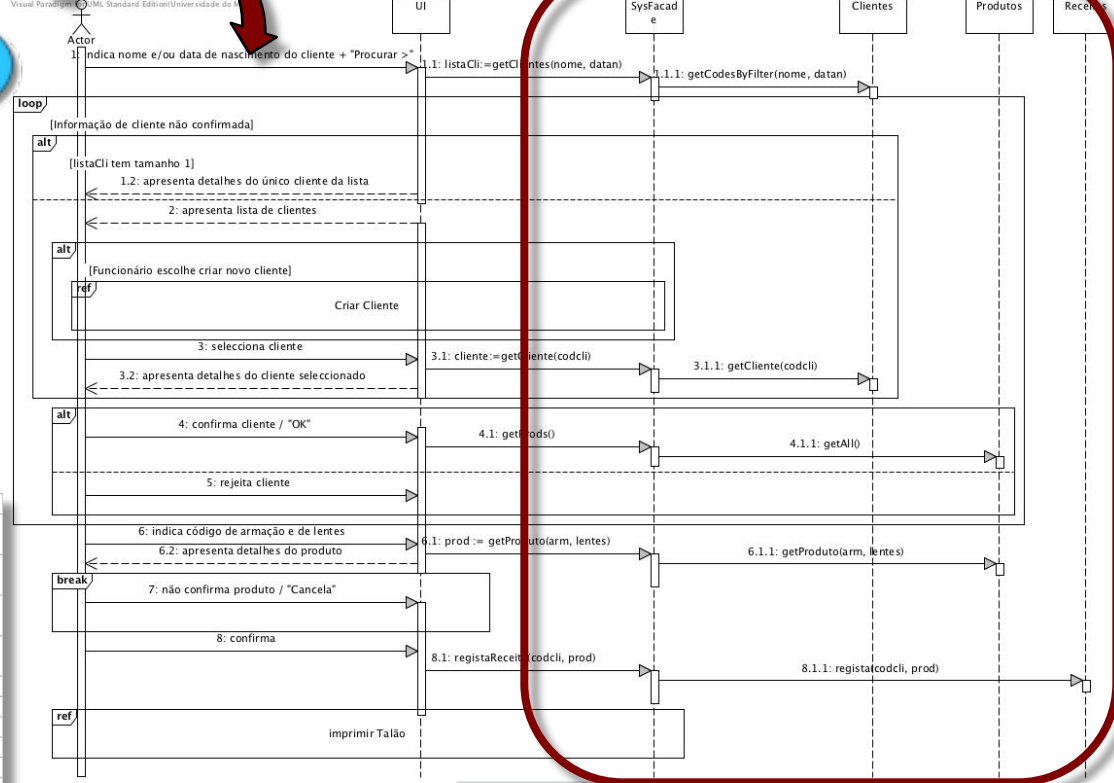
Primeira versão da *arquitettura* da solução...



Paradigma UML Standard Edition(Universidade do V)

Actor

Indica nome e/ou data de nascimento



```

graph LR
    Input(( )) --> SysFacade[+ SysFacade]
    SysFacade -.-> Clientes[- Clientes]
    SysFacade -.-> Produtos[- Produtos]
    SysFacade -.-> Receitas[- Receitas]
  
```



Modelação Estrutural

Sumário

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