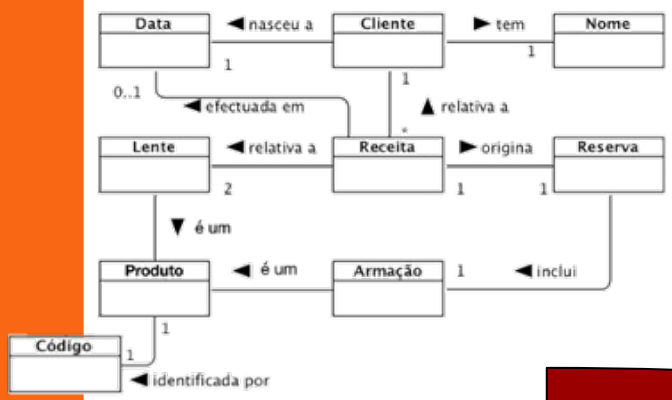
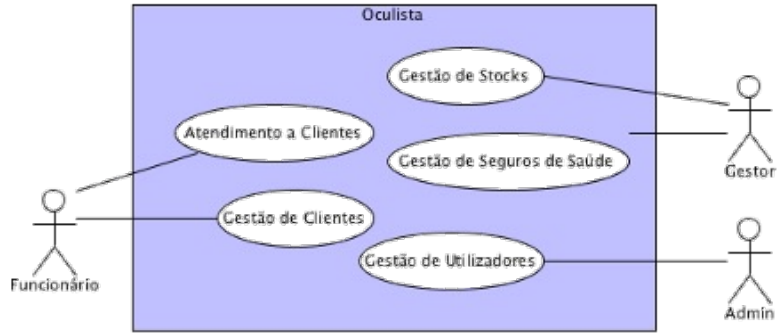
FASE 7 DO PROJETO
Diagramas de Package

Modelação Estrutural II
(Diagramas de Package)

Resumindo o exemplo...



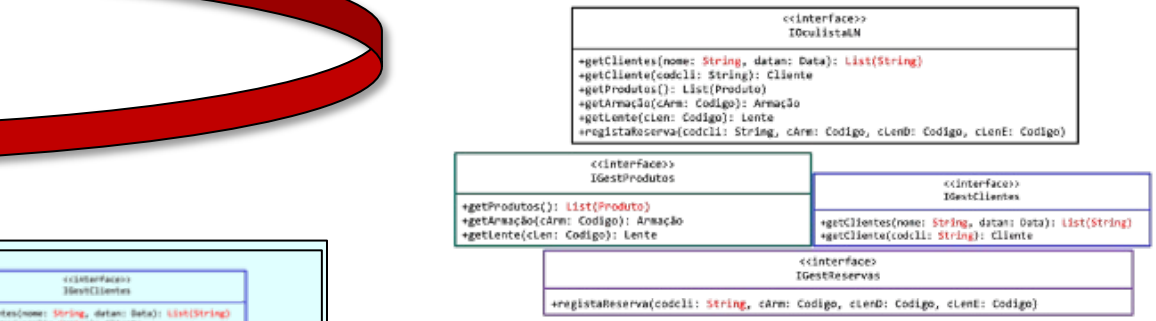


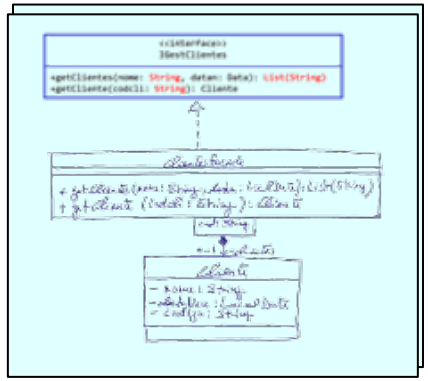
Use Case: Reservar armação e lentes

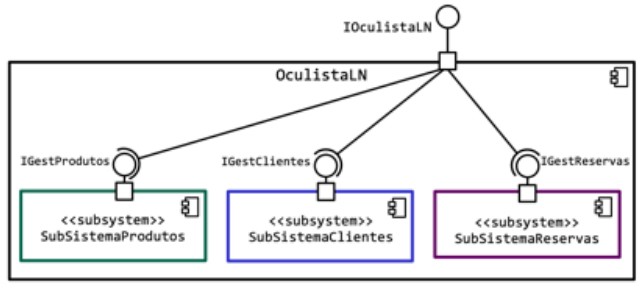
Descrição: Funcionário regista uma reserva de armação e lentes
Pós-condição: Reserva fica registada
Fluxo normal:
1. Funcionário indica nome e/ou data de nascimento do cliente
2. Sistema procura clientes
3. Sistema apresenta lista de clientes
4. Funcionário selecciona cliente
5. Sistema procura cliente
6. Sistema apresenta detalhes do cliente
7. Funcionário confirma cliente
8. Sistema procura produtos e apresenta lista
9. Funcionário indica Código de armação e lentes
10. Sistema procura detalhes dos produtos
11. Sistema apresenta detalhes dos produtos
12. Funcionário confirma produtos
13. Sistema regista reserva dos produtos
14. <<include>> imprimir talão

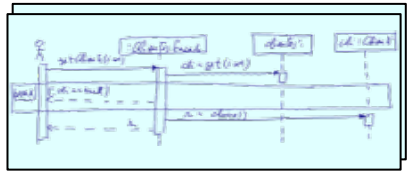
Fluxo alternativo: [lista de clients tem tamanho 1] (passo 3)
3.1. Sistema apresenta detalhes do único cliente da lista
3.2. regressa a 7

Fluxo de excepção: [cliente não quer produto] (passo 12)
11.1. Funcionário rejeita produtos
11.2. Sistema termina processo

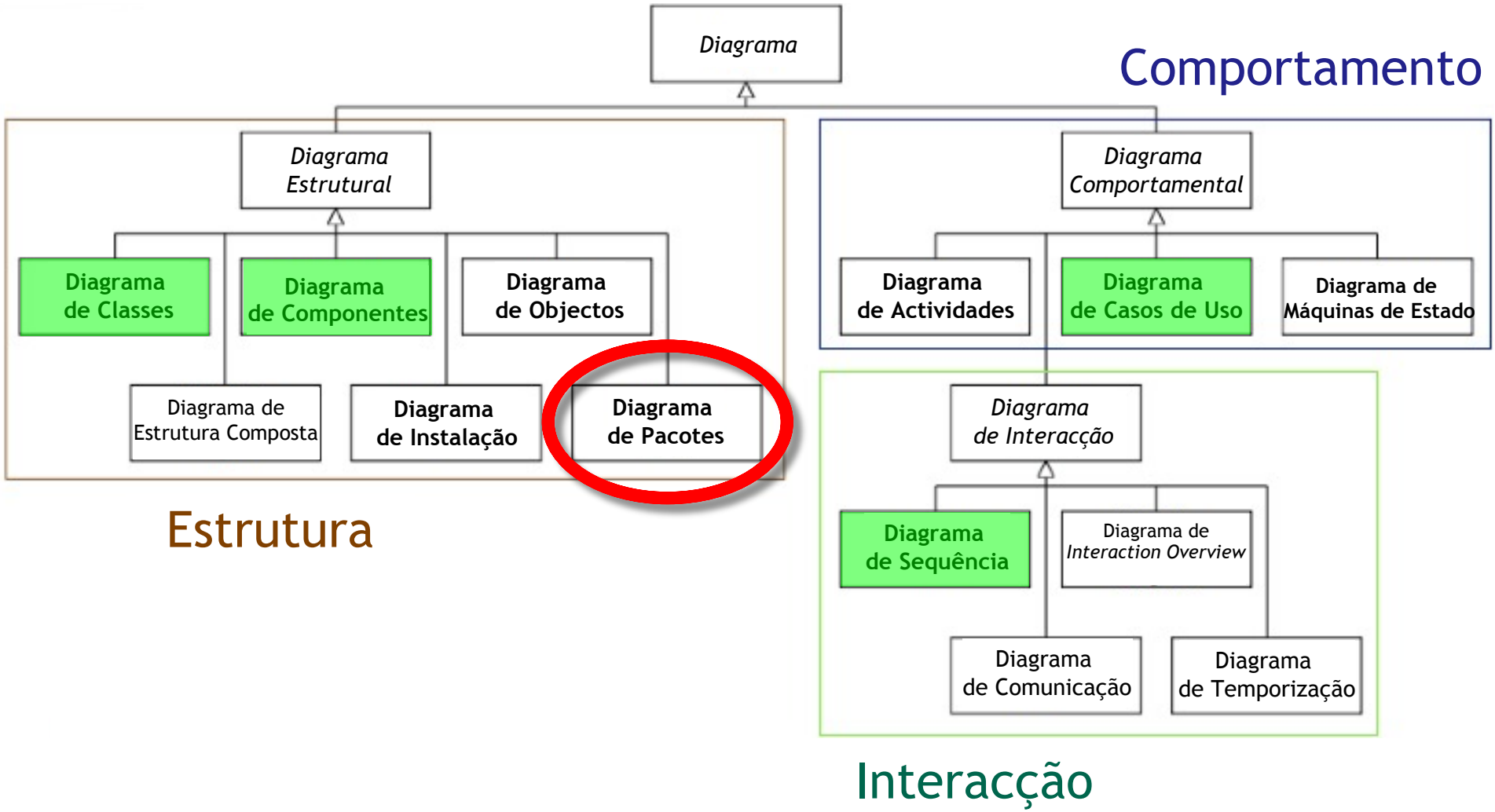




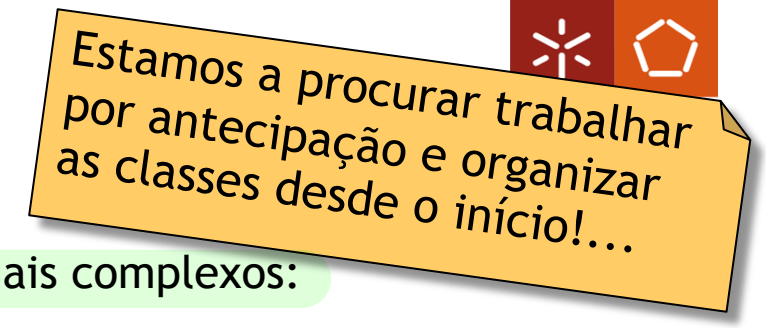




Diagramas da UML 2.x



Diagramas de *Package*



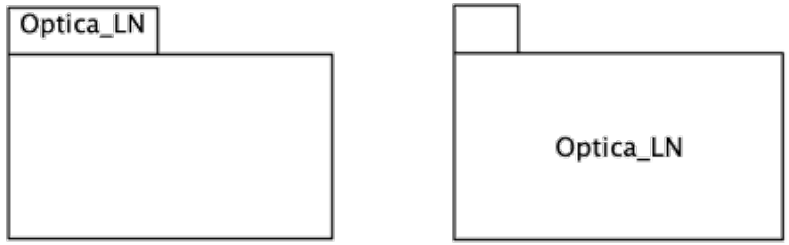
- À medida que os sistemas software se tornam mais complexos:
  - Torna-se difícil efectuar a gestão de um número crescente de classes
  - A identificação de uma classe e o seu papel no sistema dependem do contexto em que se encontram
  - É determinante conseguir identificar as dependências entre as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

- Em UML os agrupamentos de classe designam-se por *packages* (pacotes) e 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.)

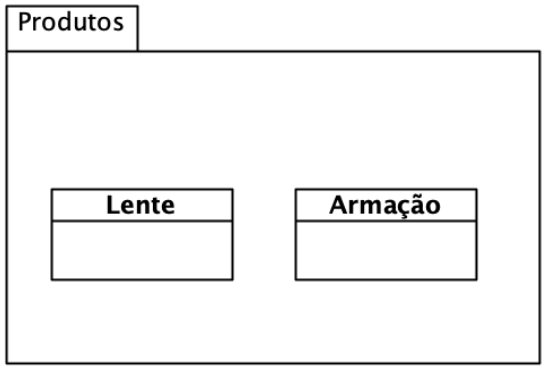
- Representam os *packages* e as relações entre *packages*

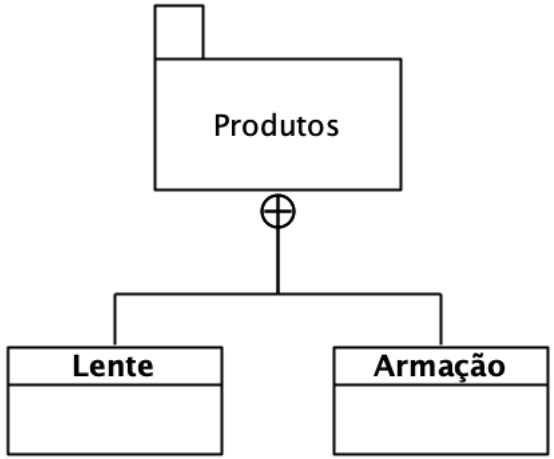


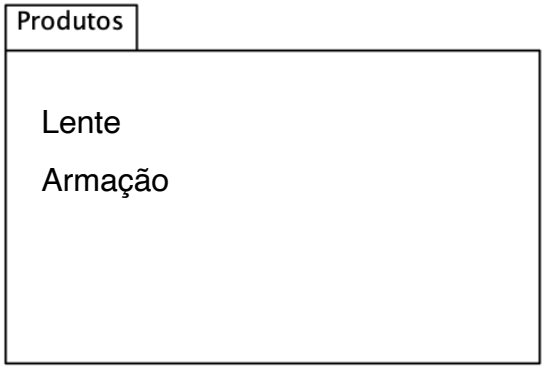
- Representam mais do que agrupamentos de classes:
  - *Packages* de classes (*packages* lógicos) - em diagramas de classes
  - *Packages* de componentes – em diagramas de componentes
  - *Packages* de nós – em diagramas de distribuição
  - *Packages* de casos de uso – em diagramas de *use cases*
- Um *package* é assim o dono de um conjunto de entidades, que vão desde outros *packages*, a classes, interfaces, componentes, *use cases*, etc.

Diagramas de *Package* (cont.)

- O conteúdo de um *package* pode ser representado de diversas formas:





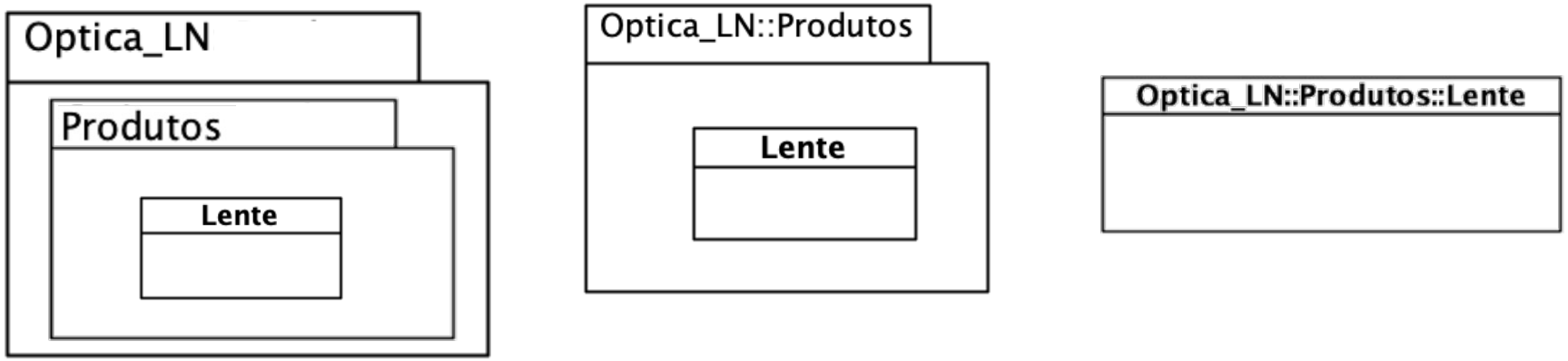



```
package Produtos;
class Lente {
      ...
}
```

```
package Produtos;
class Armação {
      ...
}
```

Diagramas de *Package* (cont.)

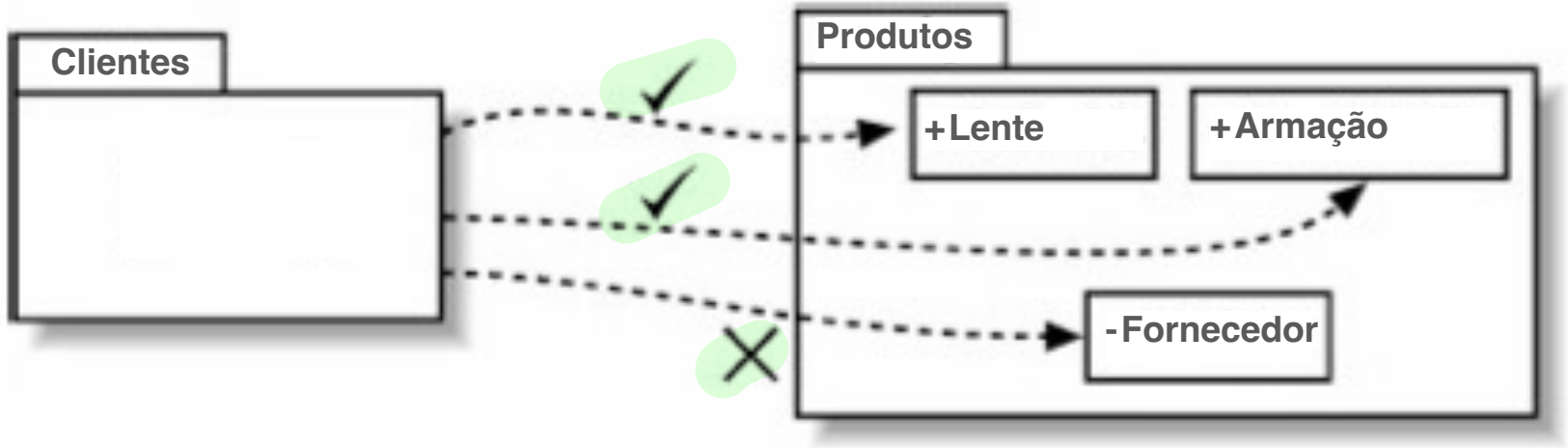
- Qualificação de classes e *packages*:



- 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 definição da visibilidade dos elementos de um *package* utiliza a mesma notação e semântica dos diagramas de classe. Vamos usar:
  - "+" - público
  - "-" - privado



Diagramas de *Package* (cont.)

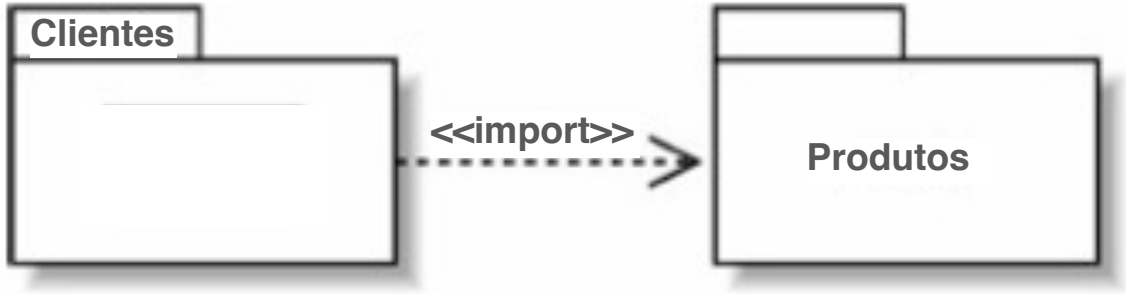
- Existem várias formas de especificar dependências entre pacotes:
  - Dependência (simples) - quando uma alteração no package de destino afecta o package de origem
  - <<import>> - o *package* origem importa o contéudo público do *package* destino – evita necessidade de qualificar os elementos importados (na forma packageA::classeB).
    - Mecanismo similar ao que permite fazer *import* de um package em Java

      ```
      import java.util.*;
      ```
  - <<access>> - o package origem acede a elementos públicos do package destino, mas necessita de qualificar completamente os nomes desses elementos.
    - Utilizar uma classe de um *package* sem o importar:

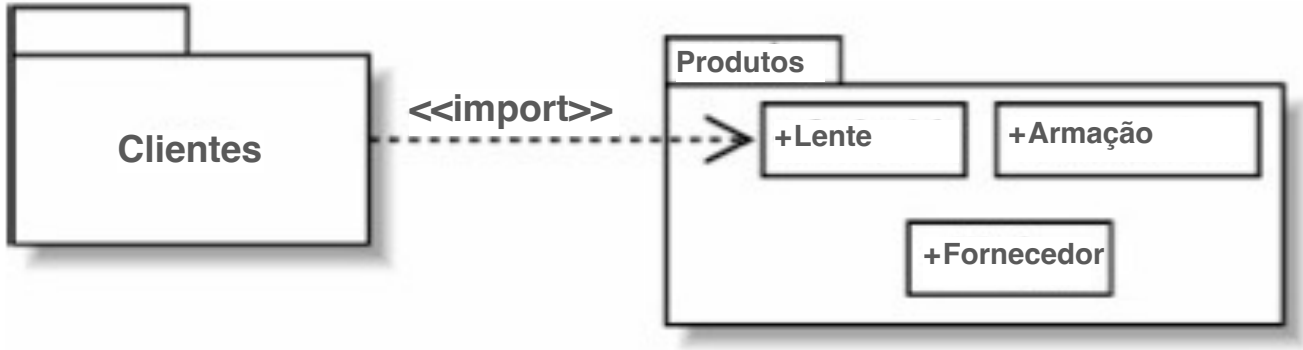
      ```
      java.util.HashMap<Produto> produtos;
      ```
  - <<merge>> - o *package* origem é fundido com o *package* destino para gerar um novo.

Diagramas de *Package* - «import»

- O package Clientes importa todas as definições públicas de Produtos:

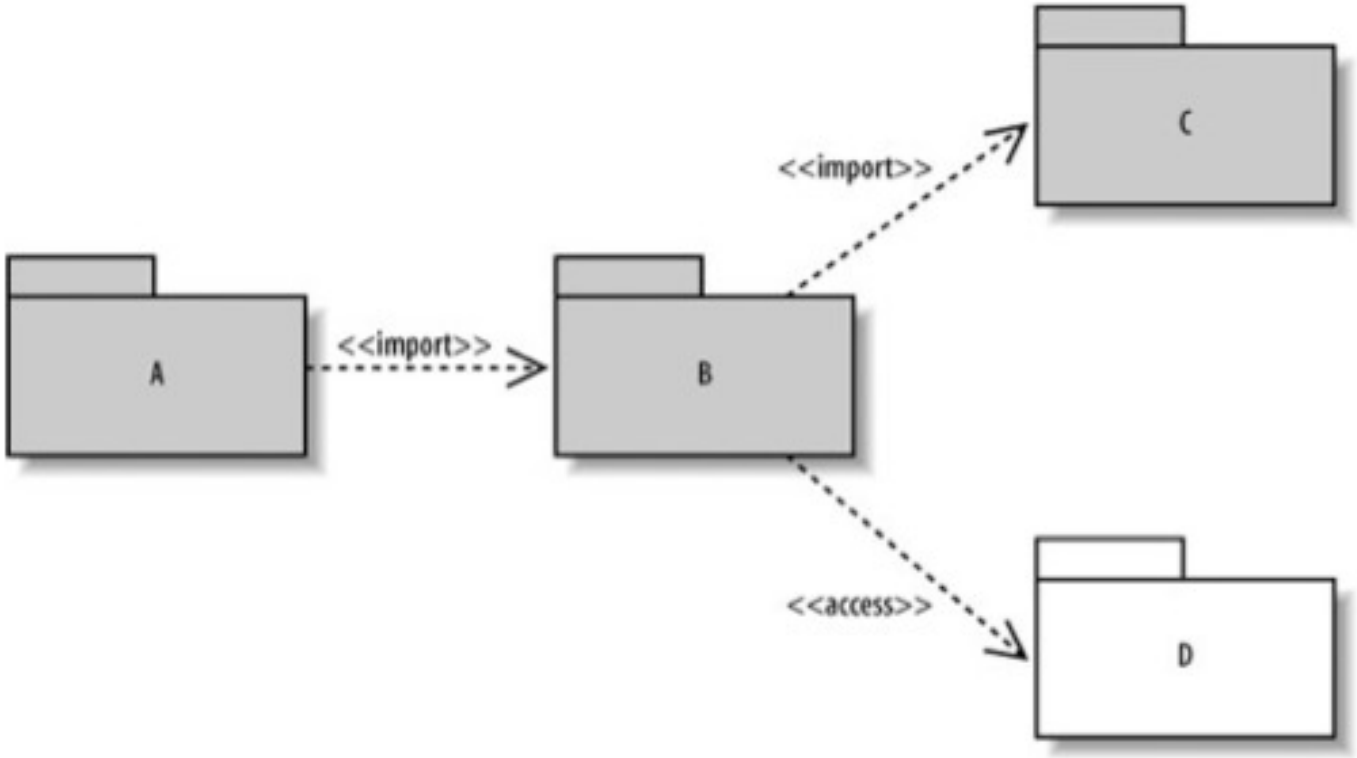


  - Definições privadas de packages importados não são acessíveis por quem importa.
- O package Clientes apenas importa a classe Lente do package Produtos:



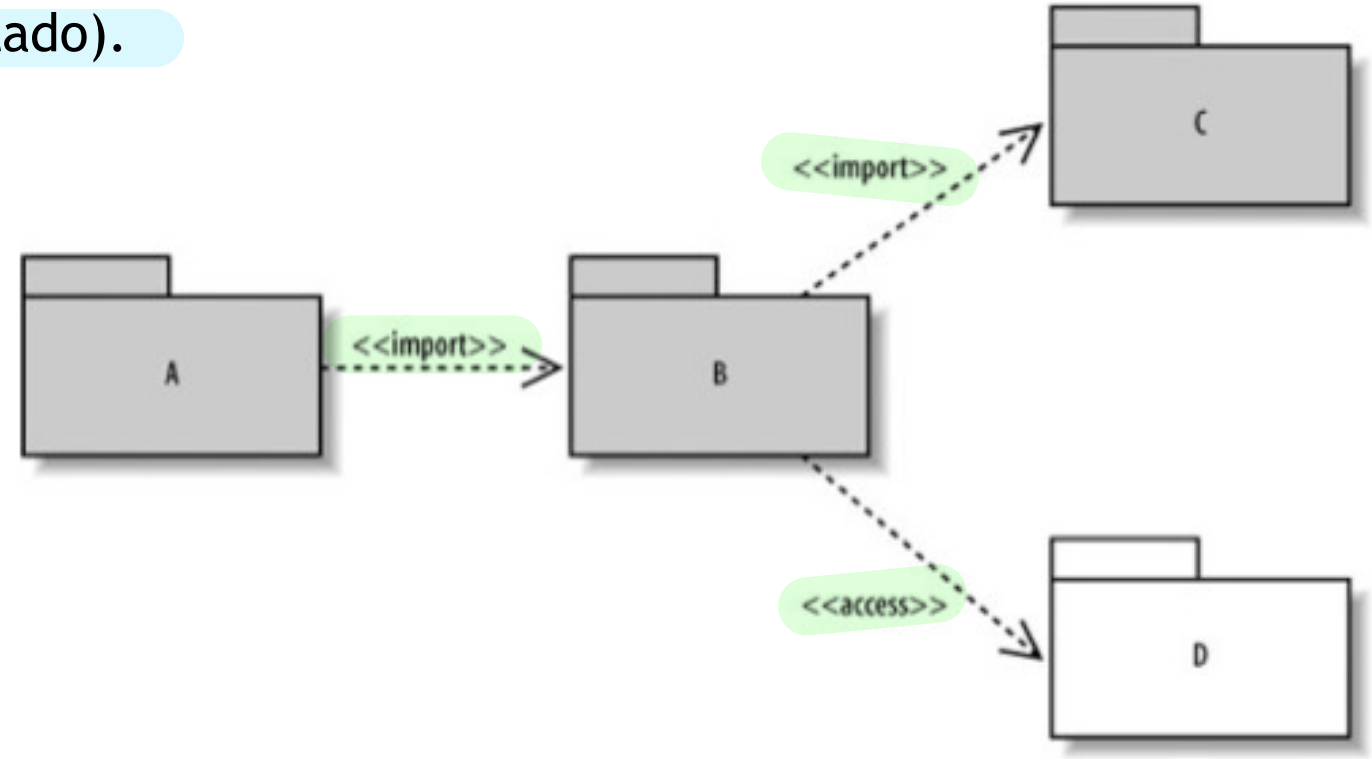
Diagramas de *Package* (cont.)

- 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).

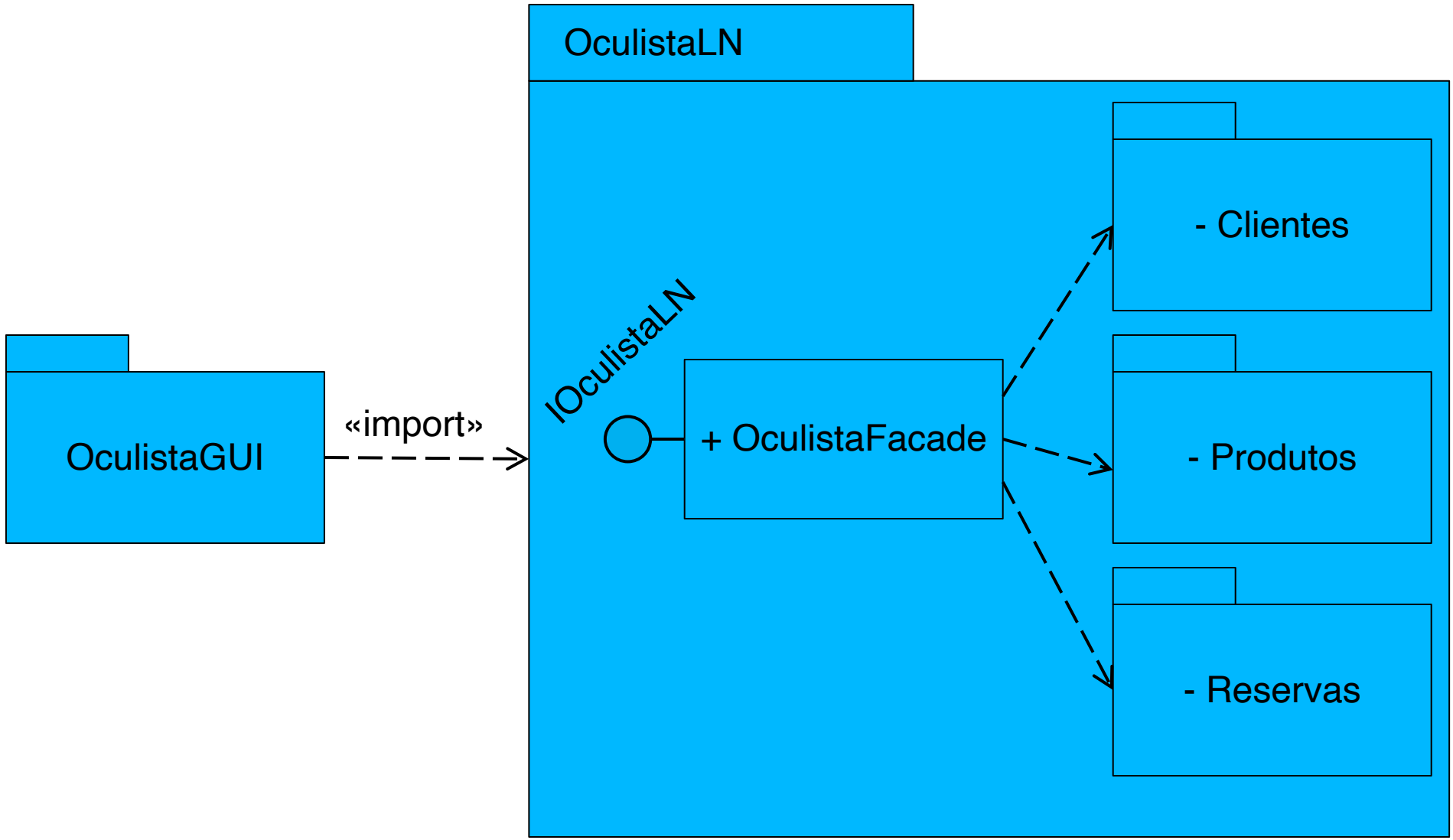


Diagramas de *Package* – «import» vs. «access»

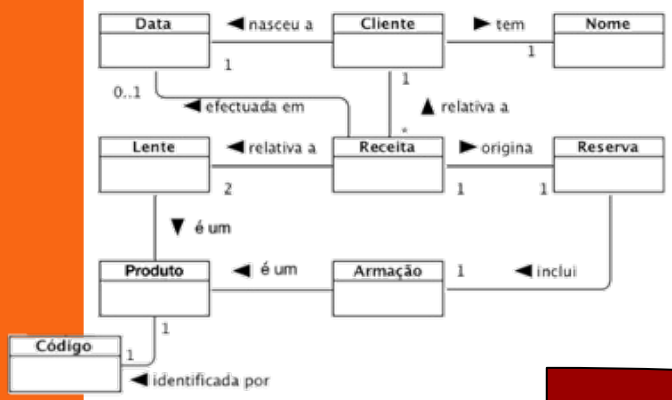
- O package B vê os elementos públicos em C e D.
- A importa B, pelo que vê os elementos públicos em B e em C (porque este é importado por B)
- A não tem acesso a D porque D é apenas acedido por B (não é importado).

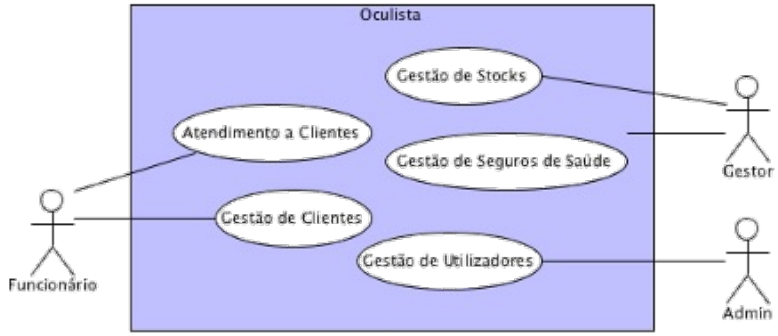


Primeira versão da arquitectura da implementação...



Resumindo o exemplo...



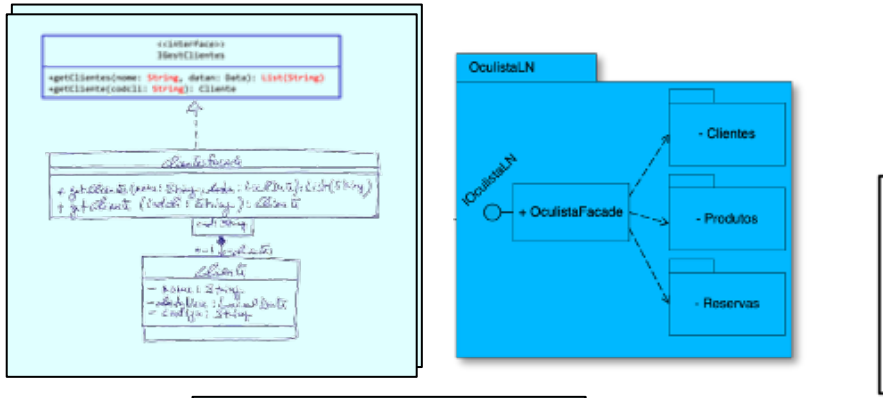


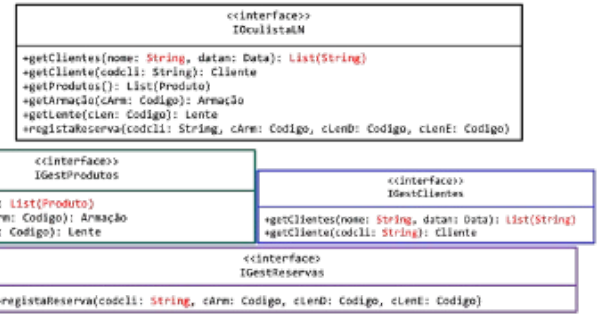
Use Case: Reservar armação e lentes

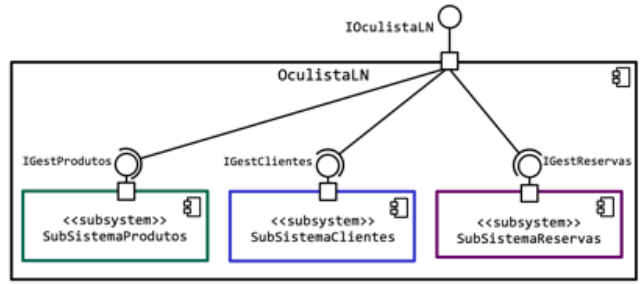
Descrição: Funcionário regista uma reserva de armação e lentes
Pós-condição: Reserva fica registada
Fluxo normal:
1. Funcionário indica nome e/ou data de nascimento do cliente
2. Sistema procura clientes
3. Sistema apresenta lista de clientes
4. Funcionário selecciona cliente
5. Sistema procura cliente
6. Sistema apresenta detalhes do cliente
7. Funcionário confirma cliente
8. Sistema procura produtos e apresenta lista
9. Funcionário indica Código de armação e lentes
10. Sistema procura detalhes dos produtos
11. Sistema apresenta detalhes dos produtos
12. Funcionário confirma produtos
13. Sistema regista reserva dos produtos
14. <<include>> imprimir talão

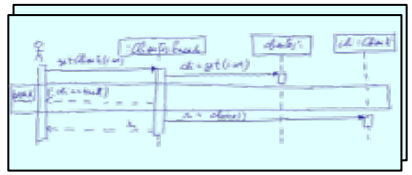
Fluxo alternativo: [lista de clients tem tamanho 1] (passo 3)
3.1. Sistema apresenta detalhes do único cliente da lista
3.2. regressa a 7

Fluxo de excepção: [cliente não quer produto] (passo 12)
11.1. Funcionário rejeita produtos
11.2. Sistema termina processo

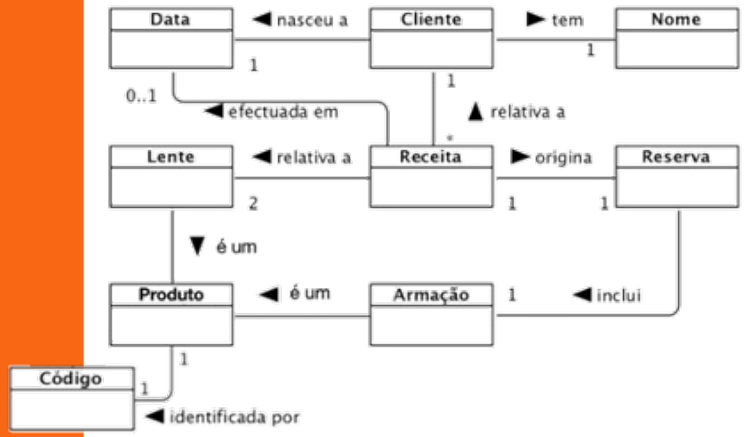


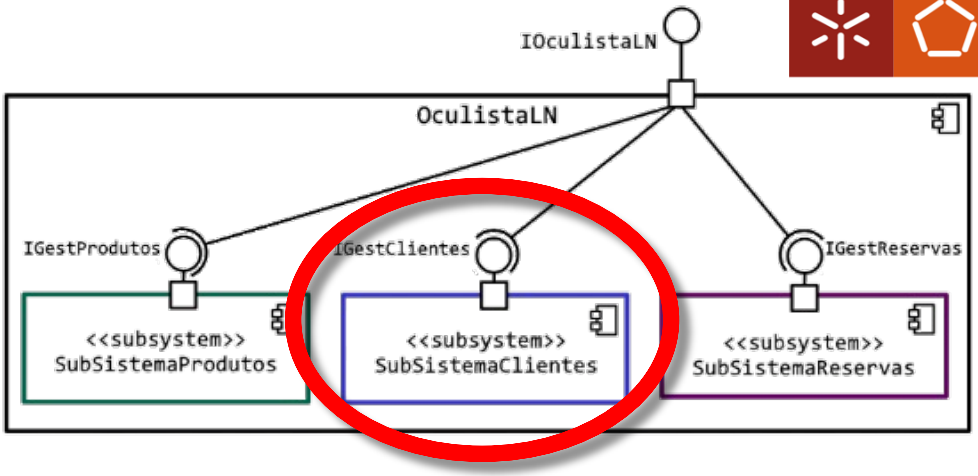


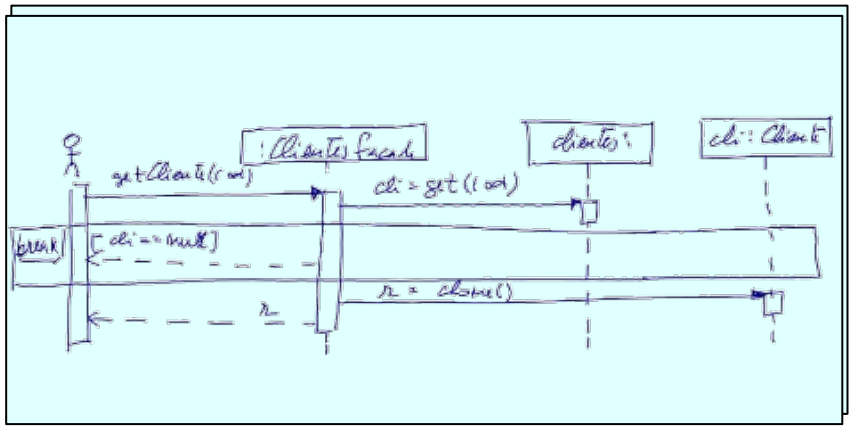


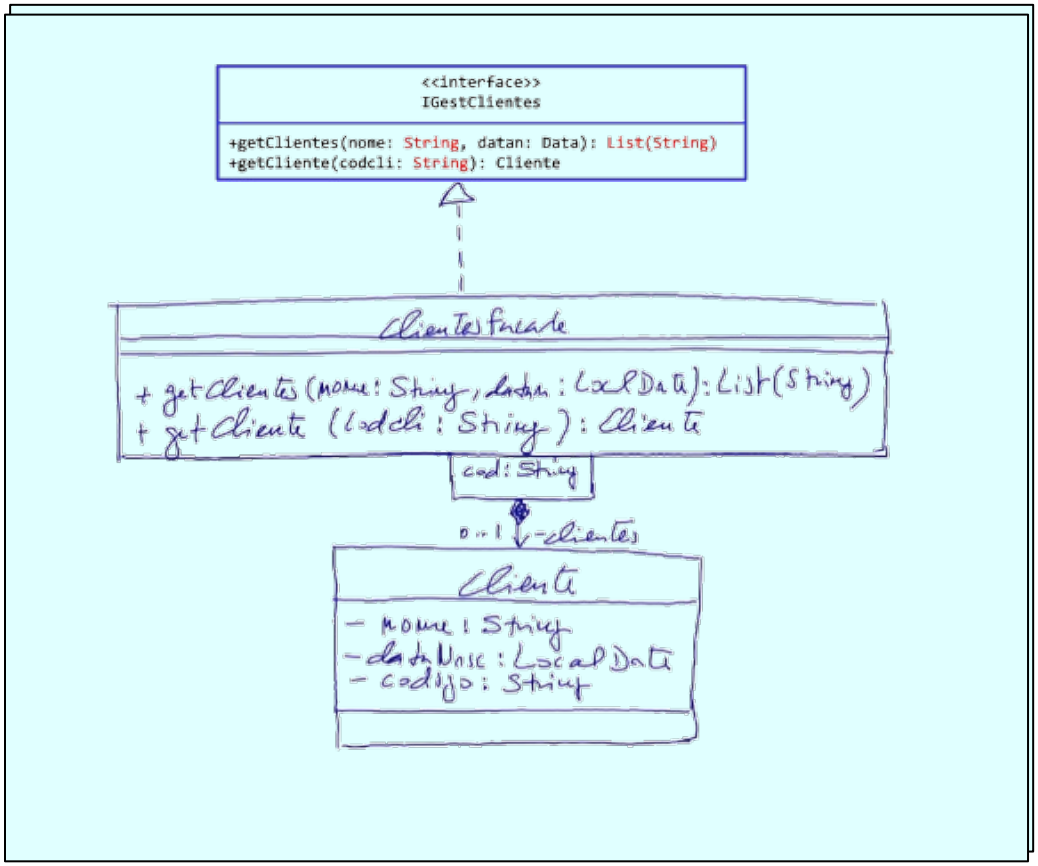


Ponto da situação...









Ponto da situação…

Cada package terá o seu Diagrama de Classes.