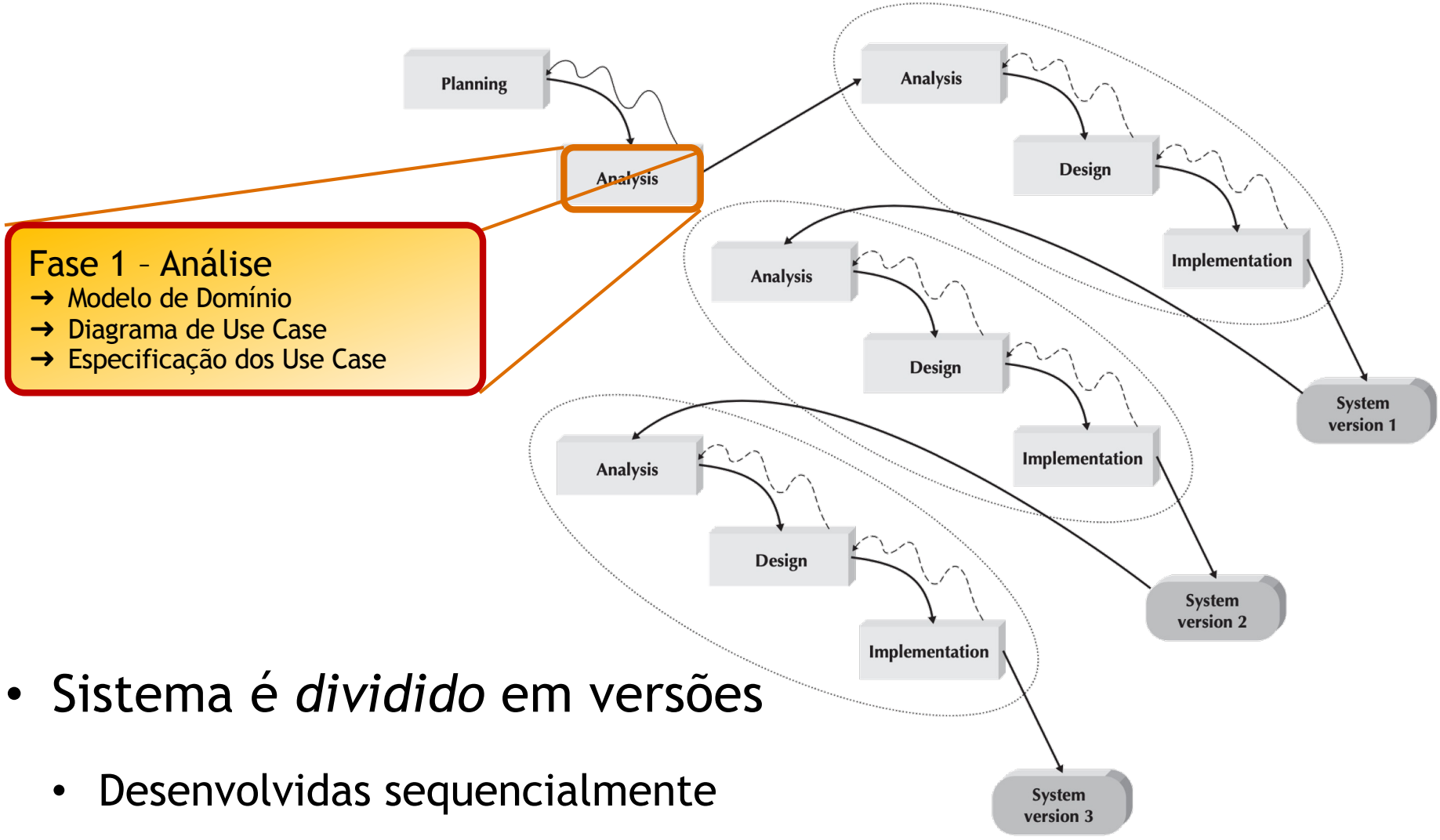
Identificação de APIs e subsistemas
(Diagramas de Componentes/Interfaces)

Desenvolvimento de Sistemas Software
José Creissac Campos / António Nestor Ribeiro

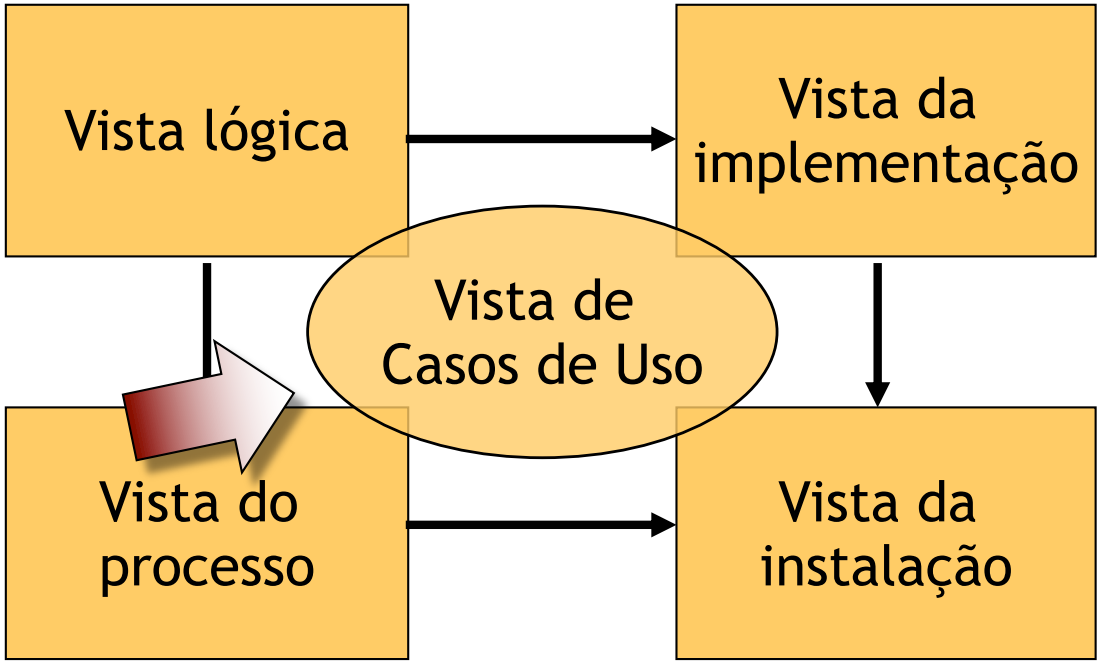
Desenvolvimento Iterativo e Incremental

- Desenvolvimento faseado (*Phased development*)



- Sistema é *dividido* em versões
  - Desenvolvidas sequencialmente

Onde estamos…

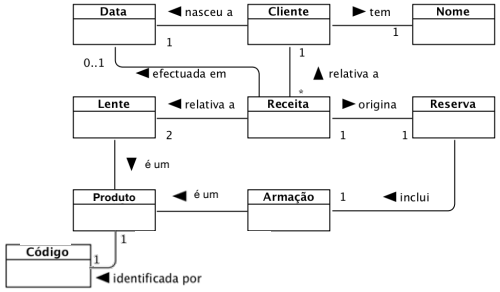


Conceptual Físico

(Kruchten, 1995)

Dos requisitos à implementação

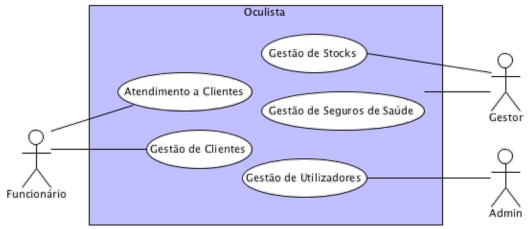
- Como fazer?

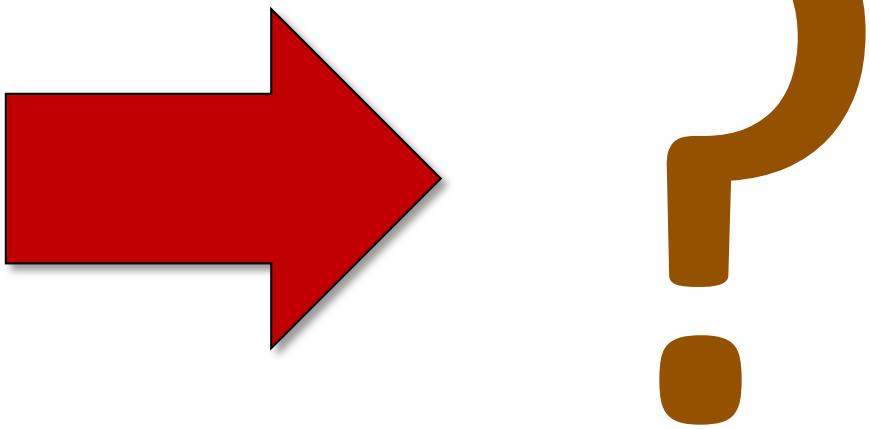


Use Case: Reservar armação e lentes

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 cliente
3. Sistema apresenta detalhes do cliente
4. Funcionário confirma cliente
5. 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

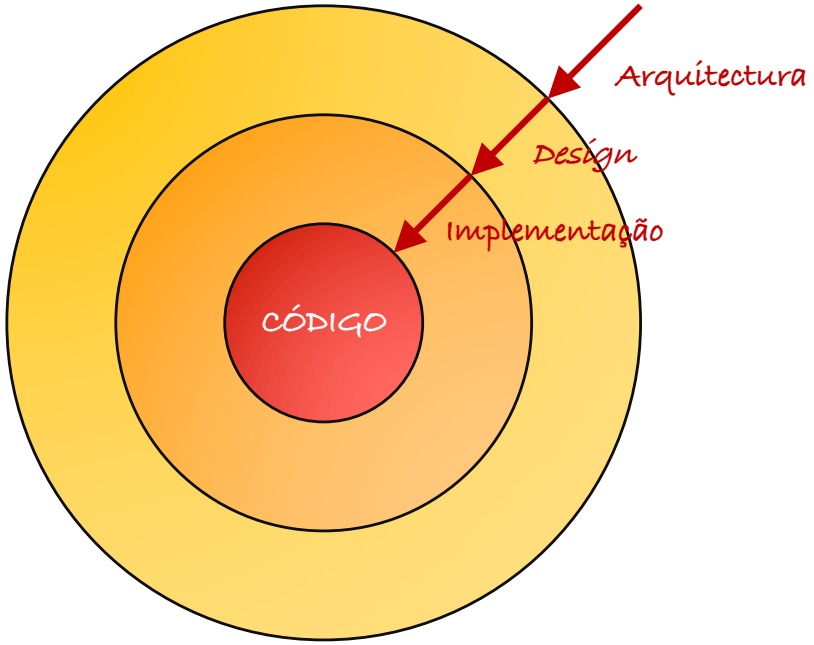
Fluxo de excepção: [cliente não quer produto] (passo 12)
11.1. Funcionário rejeita produtos
11.2. Sistema termina processo





Arquitectura do Sistema…

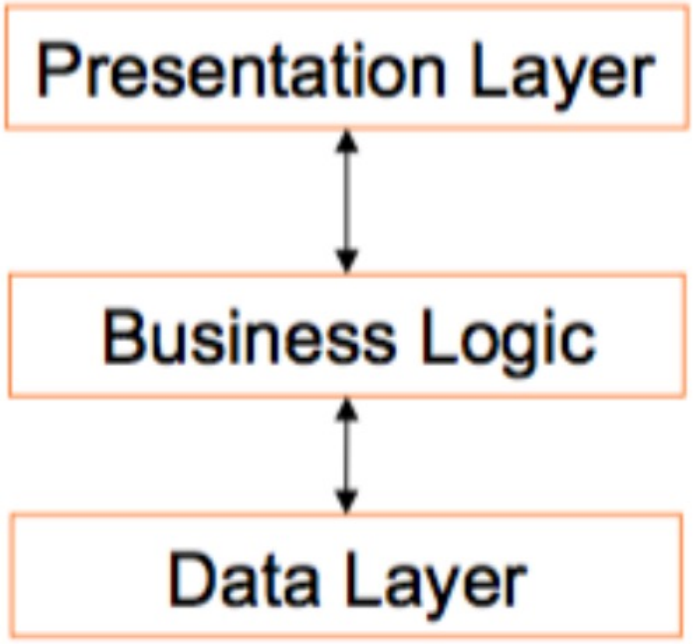
- Define o contexto para a concepção (*design*) e implementação do sistema



- Decisões arquitecturais são as mais fundamentais
  - Alterá-las terá repercussões (em cadeia) significativas

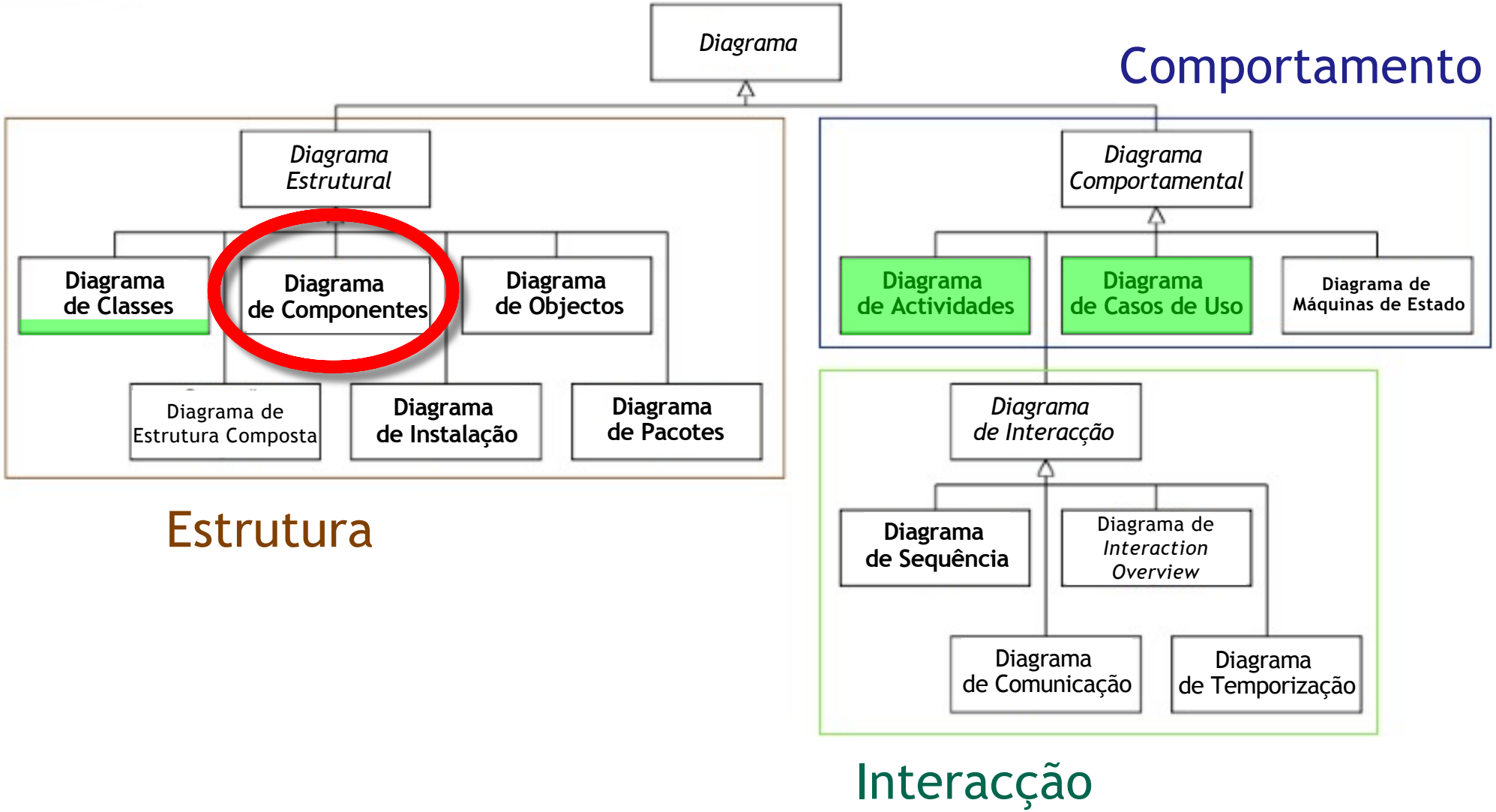
Padrão arquitetural base…

- Arquitetura em três camadas



- Como representar (em UML)?

Diagramas da UML 2.x



Componentes

- O que é um componente?
  - **Um pedaço de software reutilizável, bem encapsulado e "facilmente" substituível.**
  - São blocos (peças) que combinados constroem o sistema pretendido.
  - A dimensão dos componentes não é homogénea, existindo num mesmo sistema, componentes de diferentes dimensões.

- Quais são os bons candidatos a serem componentes do sistema?
  - Os grandes *blocos* do Sistema (cf. arquitectura em camadas)
  - Itens que desempenham uma funcionalidade que é utilizada recorrentemente em diferentes sistemas
    - Exemplos: componentes de *logging*, *parsers*, componentes de gestão de carrinhos de compra (*shopping carts*), etc.

Diagramas de Componentes

- Um Diagrama de Componentes descreve
  - Os componentes do sistema
  - As dependências entre eles
- Pode ser desenhado a diferentes níveis
  - código fonte
  - componentes binários (e.g. bibliotecas)
  - componentes executáveis
- Permite identificar, em cada nível, o que é necessário para construir o sistema

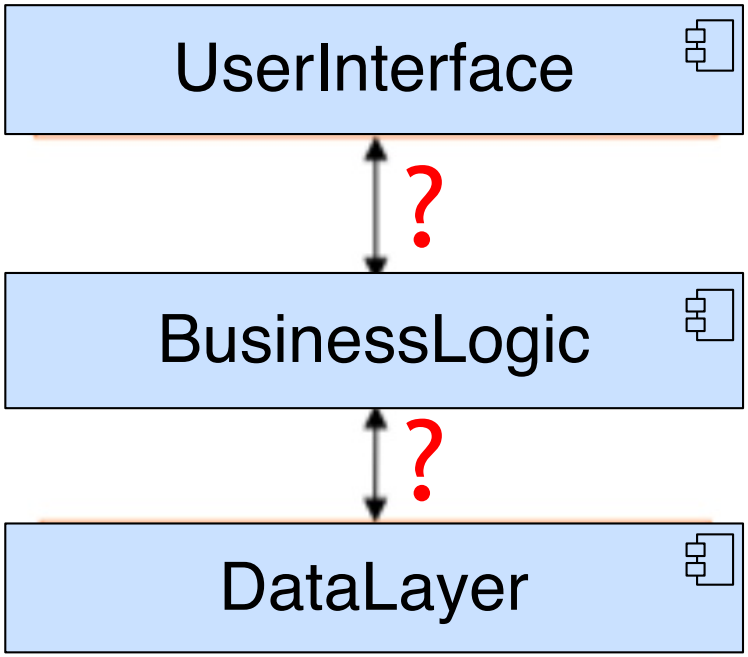
Diagramas de Componentes

- Notação:

- Alguns estereótipos de Componente:
  - «component» - componente genérico
  - «subsystem» - decomposição hierárquica do sistema global
  - «process» - componente transacional
  - «service» - componente funcional sem estado

Padrão arquitetural base…

- Arquitetura em três camadas



- Como é que os componentes *falam* entre si?
  - O que sabem uns sobre os outros?

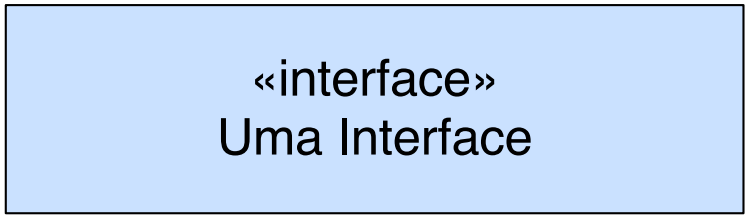
reutilizável, bem encapsulado e facilmente substituível

Interfaces

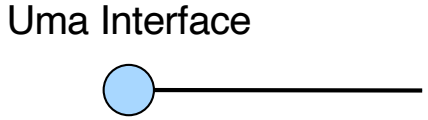
- Uma interface especifica um tipo abstracto
  - um conjunto de operações externamente visíveis que uma classe (ou componente, subsistema, etc.) deve implementar
  - semelhante a classe abstracta só com operações abstractas e sem atributos nem associações (em Java com atributos)
- Separação entre interface e (as classes de) implementação
- 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 ficam a depender apenas da interface em vez da classe de implementação
- Notação UML:
  - classe com estereótipo «interface»

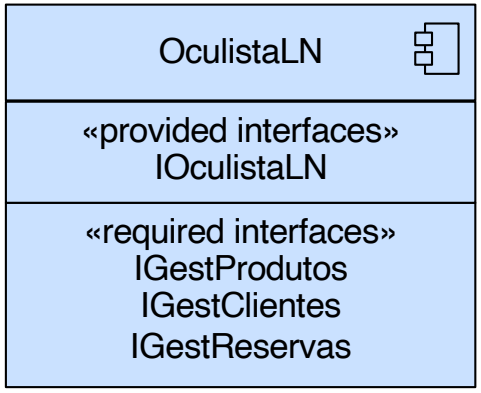


  - notação “lollipop”

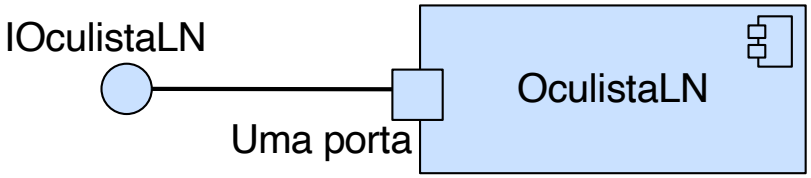


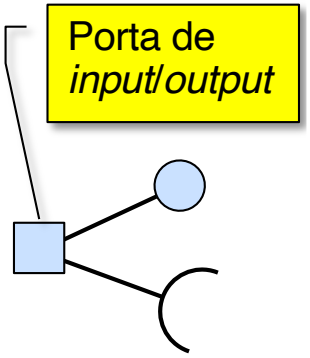
Diagramas de Componentes

- Interfaces – Indicam os serviços requeridos / fornecidos pelo componente

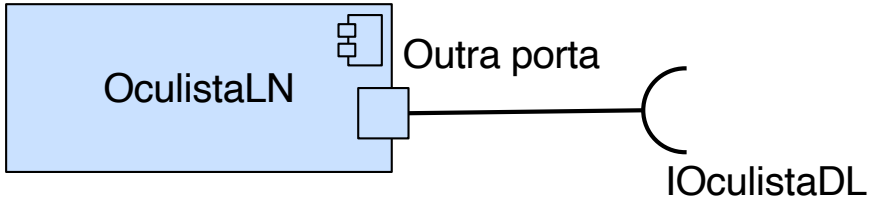


- Portas (*ports*)
  - Identificam pontos de interacção com o componente
  - porta de *output* – Componente fornece (implementa/concretiza) interface



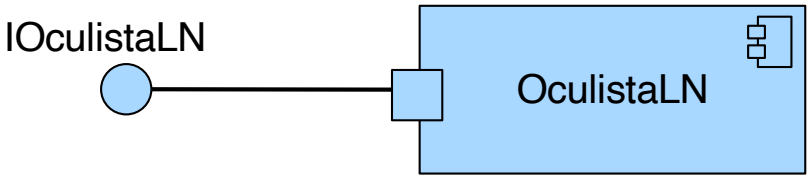


  - porta de *input* – Componente requer (utiliza/depende de) interface

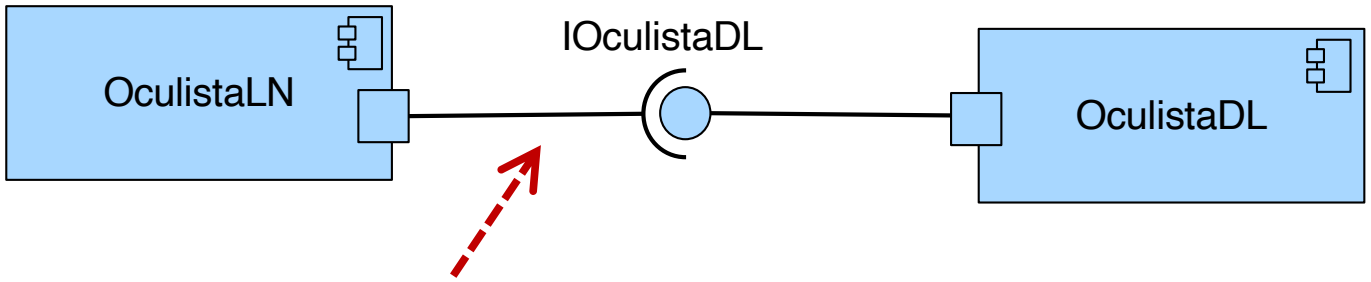


Diagramas de Componentes

- **Relação de concretização** (*realization*): um componente pode concretizar (implementar os serviços de) uma ou mais interfaces
  - Normalmente quer dizer que tem classes que implementam esses interfaces
  - Diz-se que as interfaces são fornecidas ou exportadas
  - Um componente poderá ser substituído por outro componente que implementa as mesmas interfaces

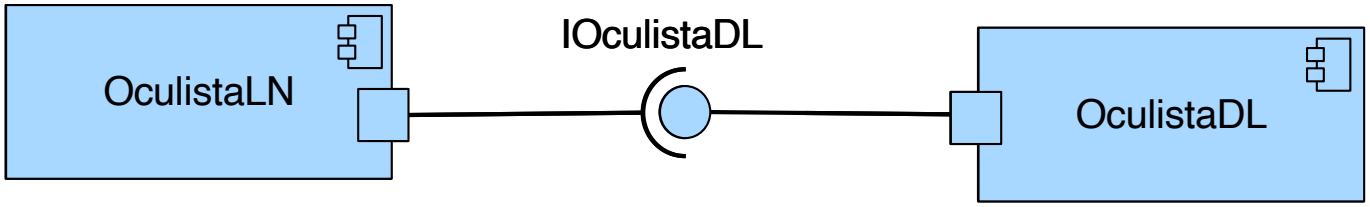


- **Relação de dependência**: um componente pode usar uma ou mais interfaces
  - Diz-se que essas interfaces são requeridas ou importadas
  - Um componente que usa outro componente, através de uma interface, não deve depender da implementação (do outro componente), mas apenas da interface



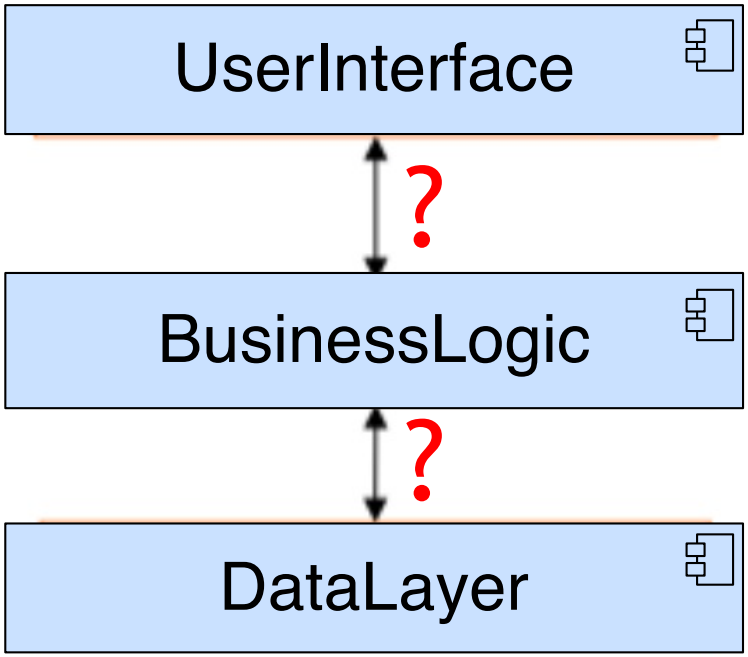
Diagramas de Componentes

- **Um componente pode ser substituído por outro componente que implementa as mesmas interfaces**



Padrão arquitetural base…

- Arquitetura em três camadas

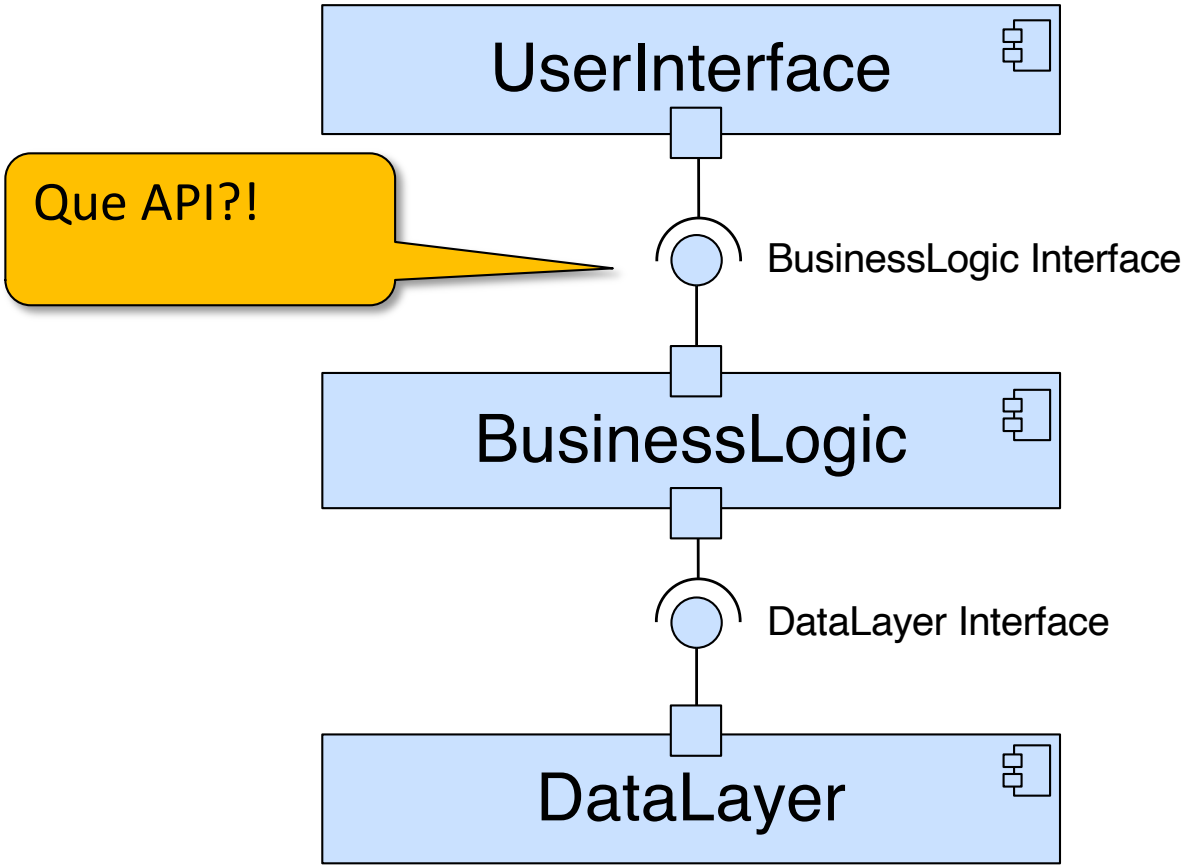


- Como é que os componentes *falam* entre si?
  - O que sabem uns sobre os outros?

reutilizável, bem encapsulado e facilmente substituível

Padrão arquitetural base...

- Arquitetura em três camadas



Ponto Prévio...

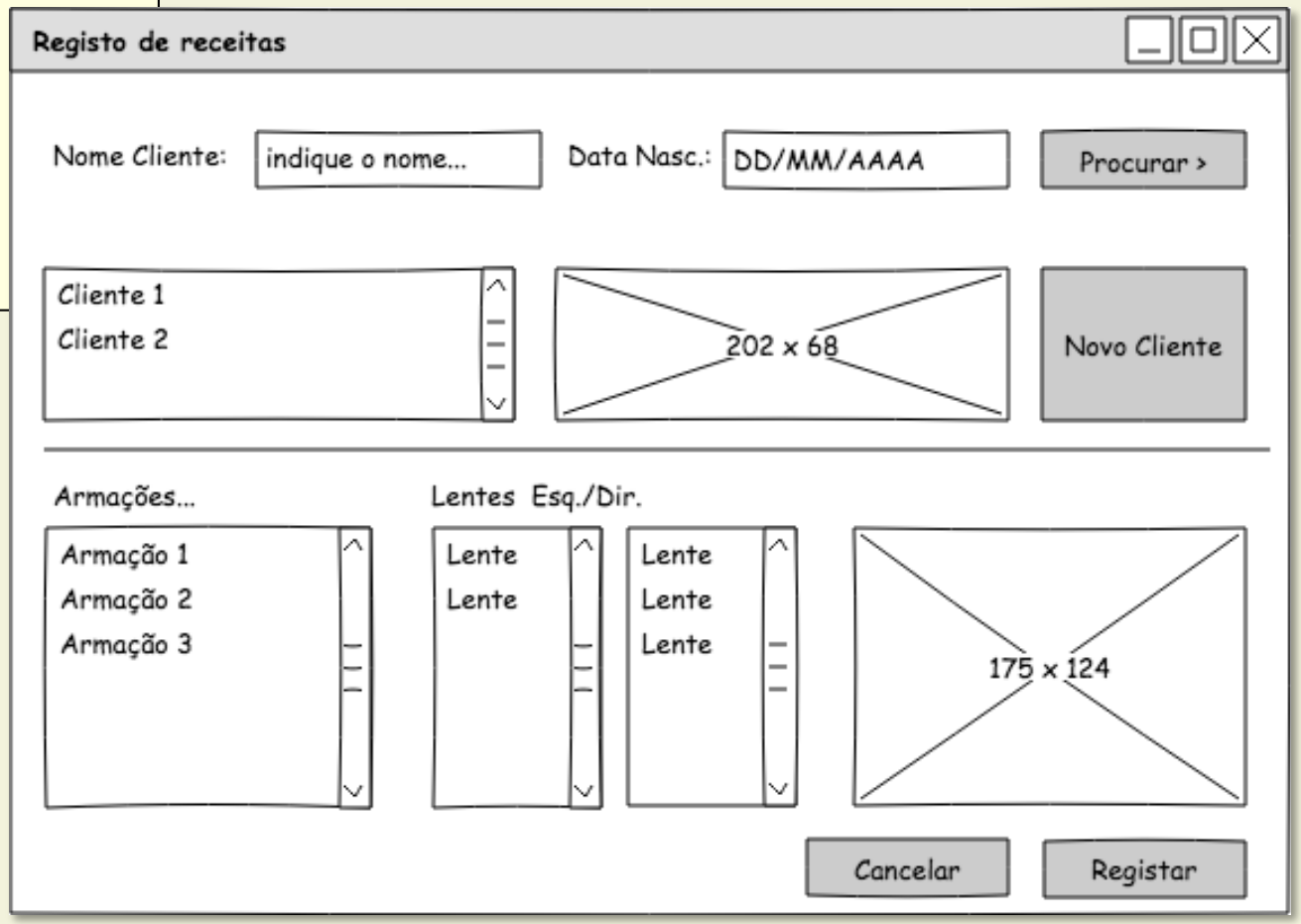
Use Case: Reservar armação e lentes

Descrição: Funcionário regista uma reserva de armação e lentes
Pós-condição: Reserva fica registada
Fluxo normal:

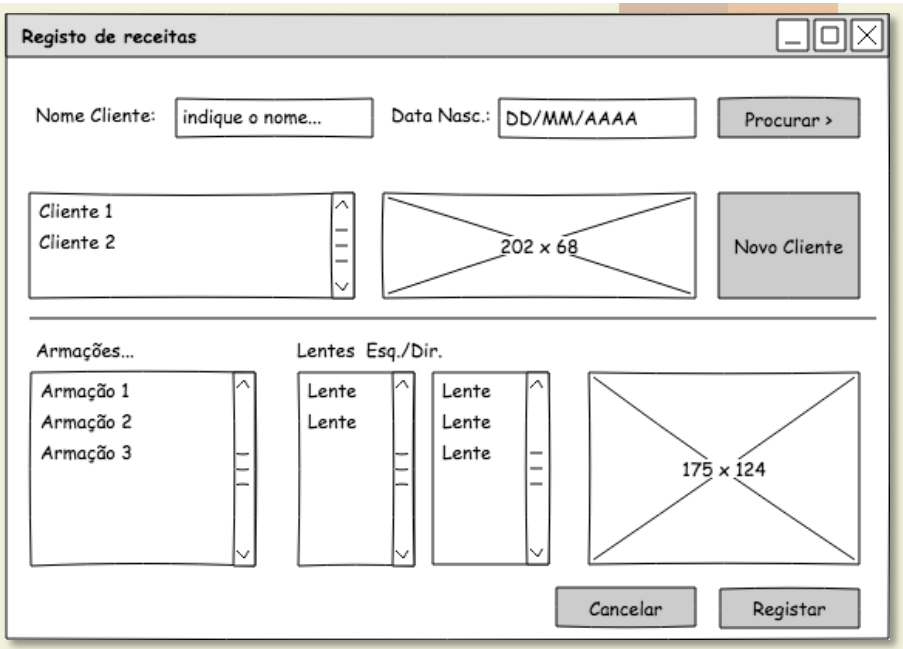
1. Funcionário indica nome e/ou data de nascimento do cliente
2. Sistema procura cliente
3. Sistema apresenta detalhes do cliente
4. Funcionário confirma cliente
5. Funcionário indica Código de armação e lentes
6. Sistema procura detalhes dos produtos
7. Sistema apresenta detalhes dos produtos
8. Funcionário confirma produtos
9. Sistema regista reserva dos produtos
10. <<include>> imprimir talão

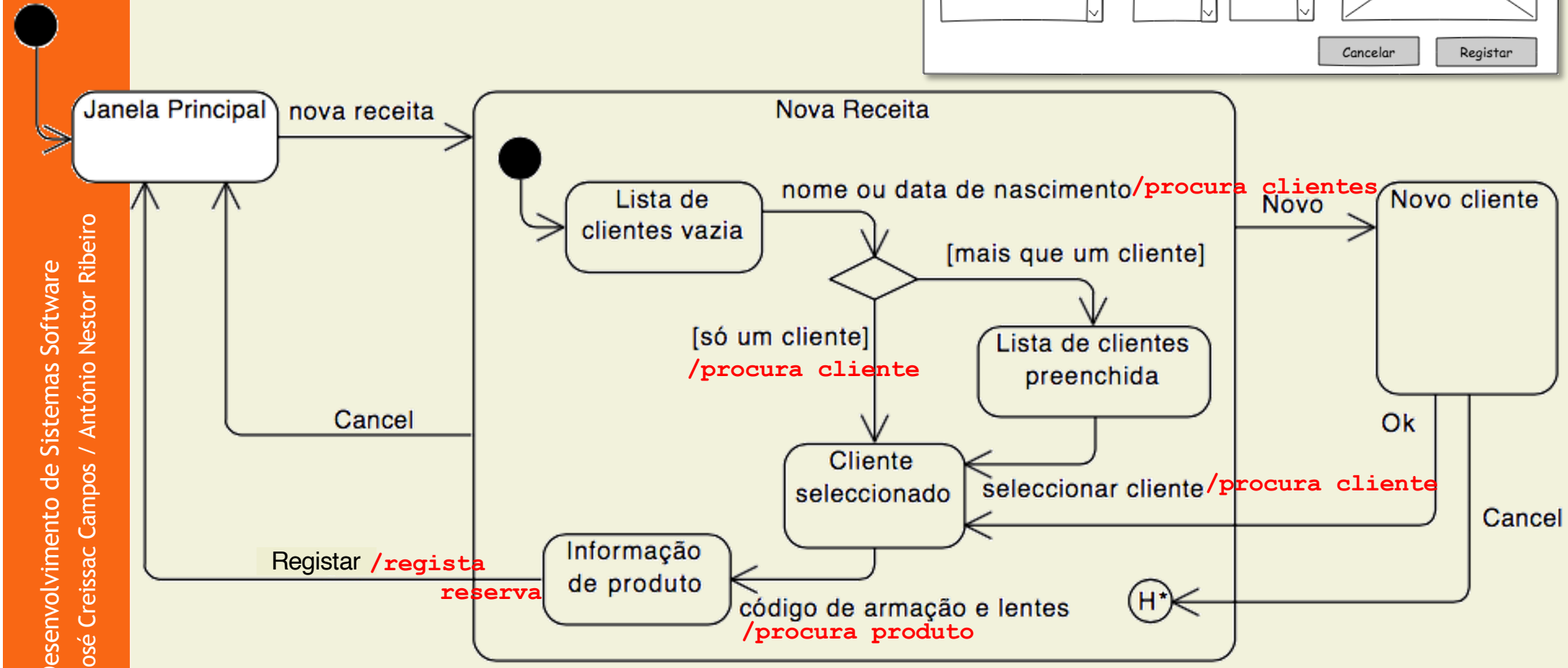
Fluxo de excepção: [cliente não quer produto] (passo 8)

8.1. Funcionário rejeita produtos
8.2. Sistema termina processo



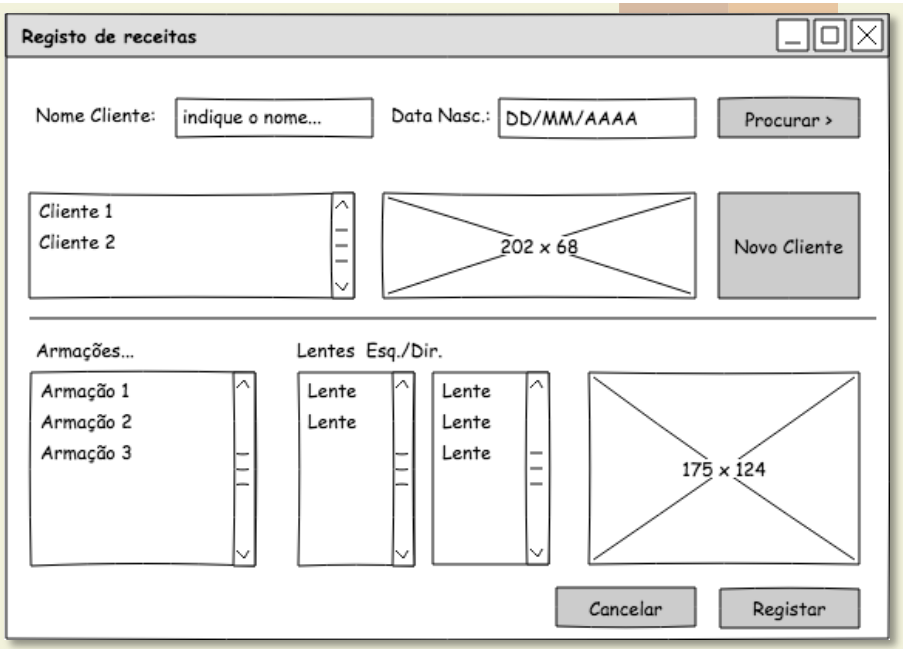
Ponto Prévio...

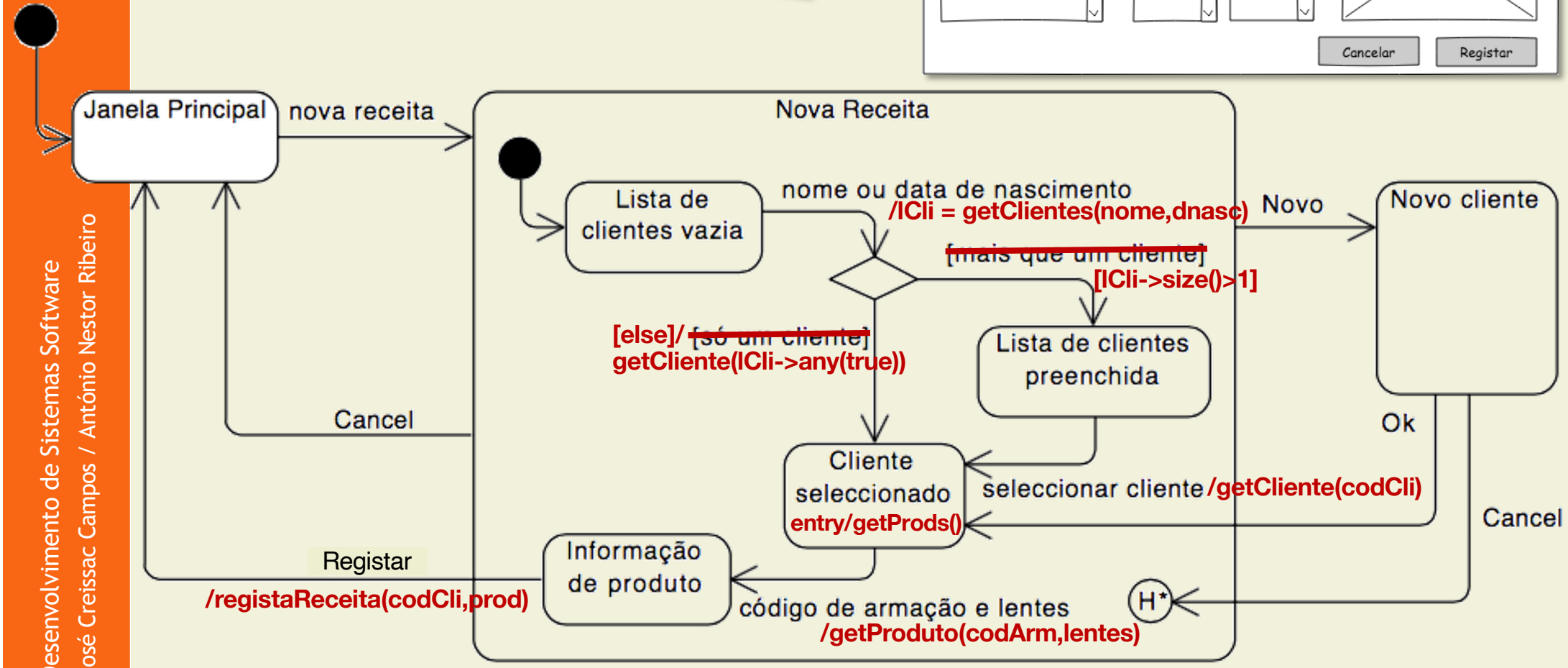




Ponto Prévio...

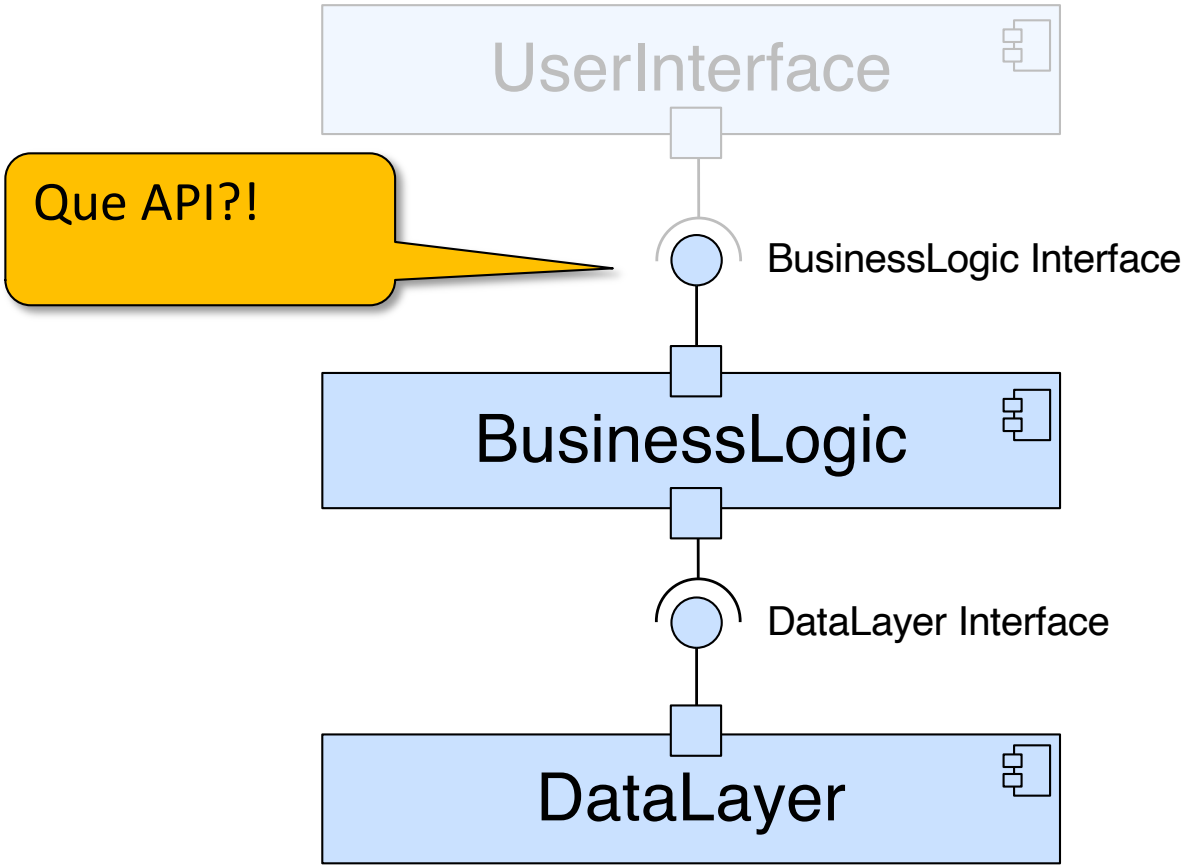
Vamos adotar uma abordagem simplificada...



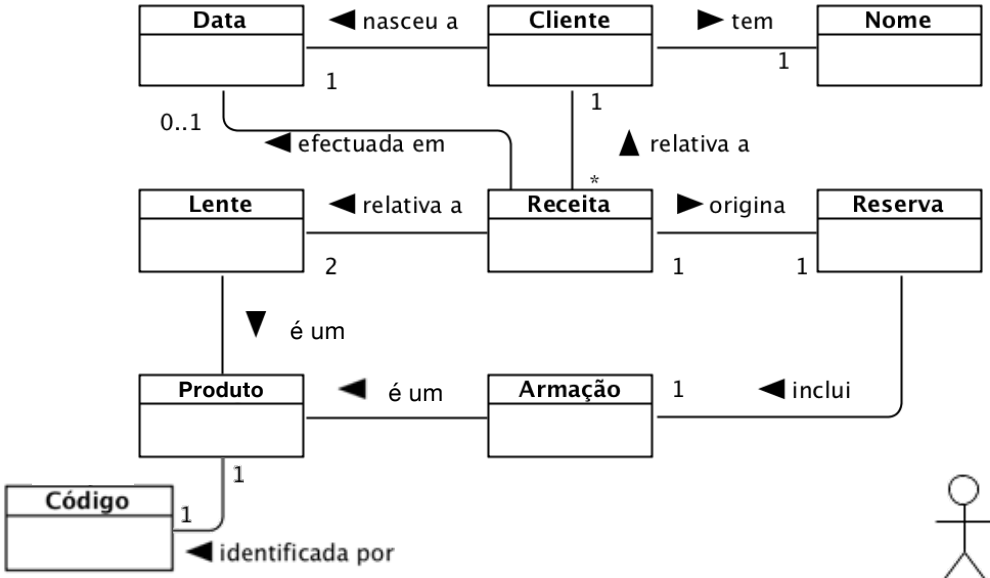


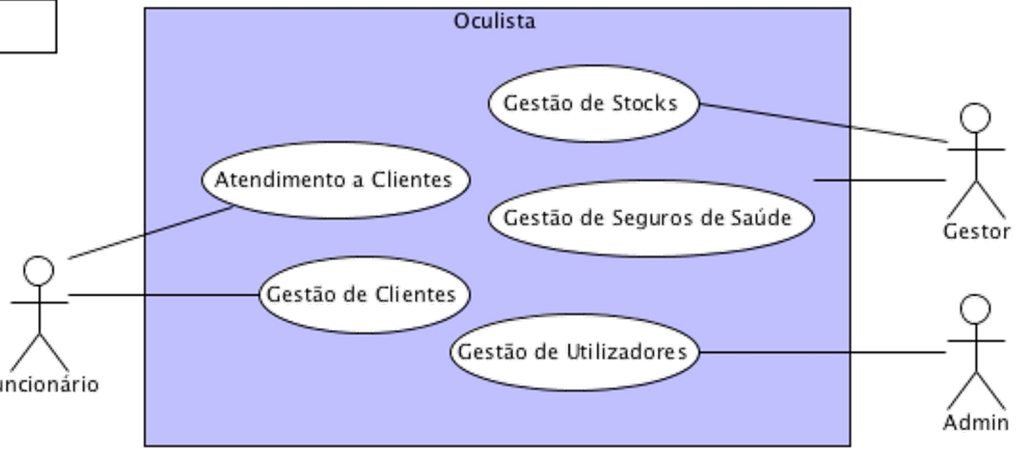
Padrão arquitetural base...

- Arquitetura em três camadas



Um 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 cliente
3. Sistema apresenta detalhes do cliente
4. Funcionário confirma cliente
5. Funcionário indica Código de armação e lentes
6. Sistema procura detalhes dos produtos
7. Sistema apresenta detalhes dos produtos
8. Funcionário confirma produtos
9. Sistema regista reserva dos produtos
10. <<include>> imprimir talão

Fluxo de excepção: [cliente não quer produto] (passo 8)
8.1. Funcionário rejeita produtos
8.2. Sistema termina processo

Casos de uso como conjuntos de transações

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 cliente
3. Sistema apresenta detalhes do cliente
4. Funcionário confirma cliente
5. Funcionário indica Código de armação e lentes
6. Sistema procura detalhes dos produtos
7. Sistema apresenta detalhes dos produtos
8. Funcionário confirma produtos
9. Sistema regista reserva dos produtos
10. <<include>> imprimir talão

Fluxo de excepção: [cliente não quer produto] (passo 8)

8.1. Funcionário rejeita produtos

8.2. Sistema termina processo

Casos de uso como conjuntos de transações

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 cliente**
3. Sistema apresenta detalhes do cliente
4. Funcionário confirma cliente
5. Funcionário indica Código de armação e lentes
6. **Sistema procura detalhes dos produtos**
7. Sistema apresenta detalhes dos produtos
8. Funcionário confirma produtos
9. **Sistema regista reserva dos produtos**
10. <<include>> imprimir talão

Fluxo de excepção: [cliente não quer produto] (passo 8)

8.1. Funcionário rejeita produtos

8.2. Sistema termina processo

Identificação de responsabilidades

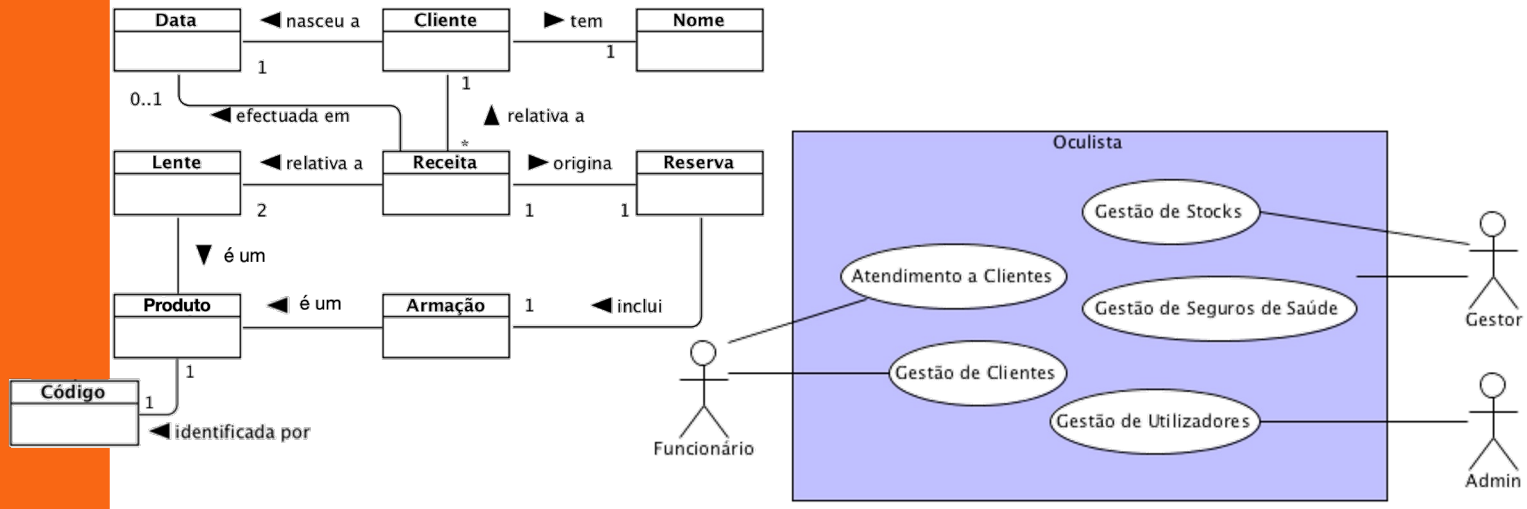
Responsabilidades que a lógica de negócio tem que cumprir para satisfazer o use case! – cf. guiado pelos Use Cases

OculistaLN
procura cliente por nome e data de nascimento procura detalhes dos produtos regista reserva dos produtos

API da lógica de negócios para suportar o Use Case.

«interface» IOculistaLN
+getCliente(nome: String, datan: Data): Cliente +getArmação(cArm: Codigo): Armação +getLente(cLen: Codigo): Lente +registaReserva(codcli: ?, cArm: Codigo, cLenD: Codigo, cLenE: Codigo)

Identificação de responsabilidades



Use Case: Reservar armação e lentes

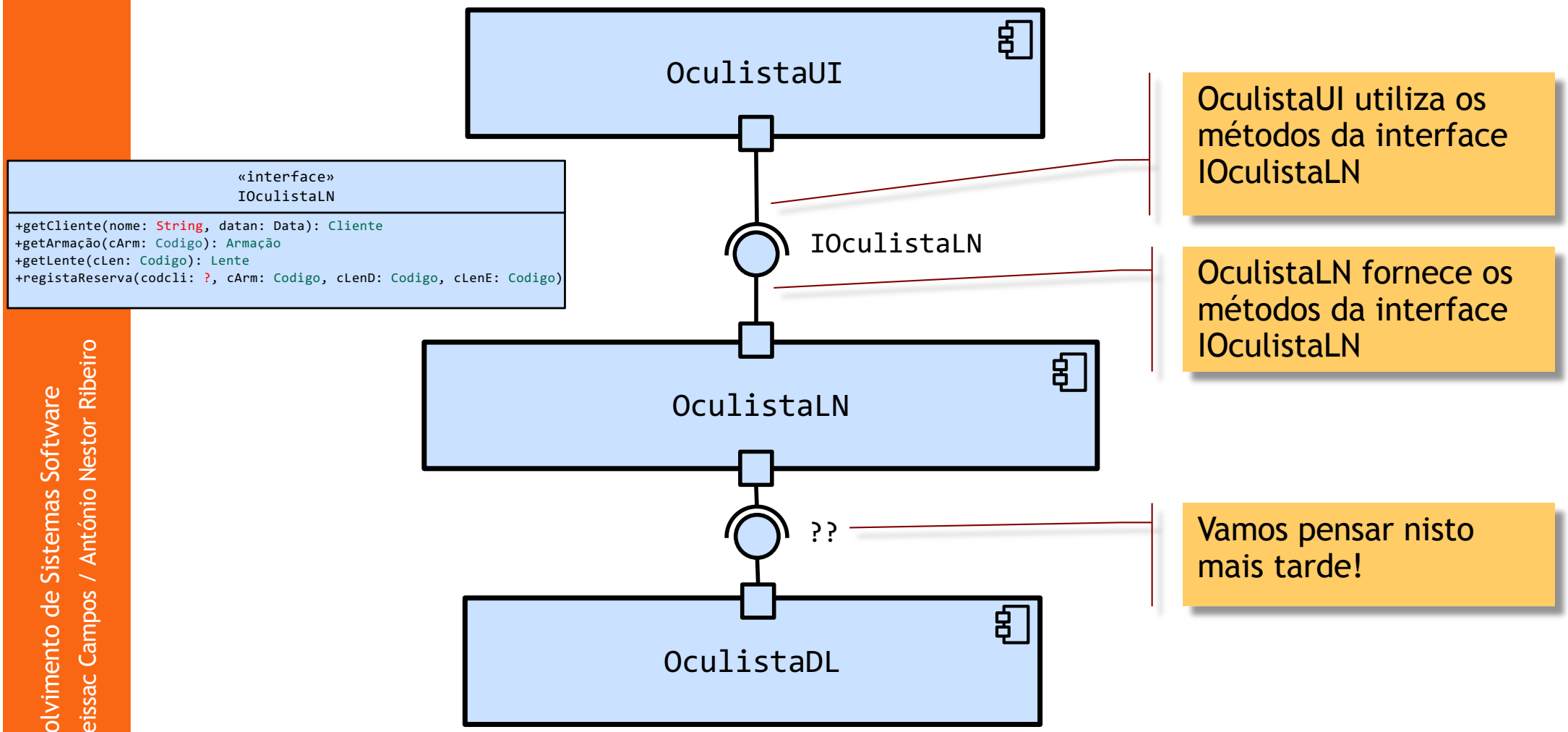
Descrição: Funcionário regista uma reserva de armação e lentes
Pós-condição: Reserva fica registada
Fluxo normal:
1. Funcionário indica nome e/ou data de nascimento do cliente
2. Sistema procura cliente
3. Sistema apresenta detalhes do cliente
4. Funcionário confirma cliente
5. Funcionário indica Código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

Fluxo de excepção: [cliente não quer produto] (passo 8)
8.1. Funcionário rejeita produtos
8.2. Sistema termina processo

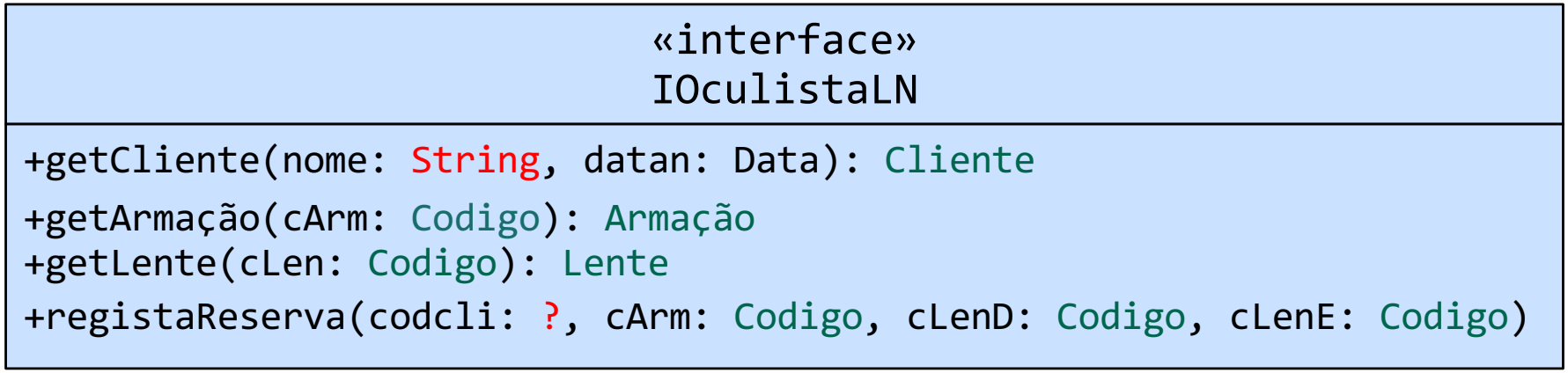
API da lógica de negócios para suportar o Use Case.

```
«interface»
IOculistaLN
+getCliente(nome: String, datan: Data): Cliente
+getArmação(cArm: Codigo): Armação
+getLente(cLen: Codigo): Lente
+registaReserva(codcli: ?, cArm: Codigo, cLenD: Codigo, cLenE: Codigo)
```

Arquitectura de 3 camadas para o exemplo



Identificação de subsistemas



de subsistemas

API global da lógica de negócios para suportar o Use Case.

```
«interface»
IOculistaLN
+getCliente(nome: String, datan: Data): Cliente

+getArmação(cArm: Codigo): Armação
+getLente(cLen: Codigo): Lente

+registaReserva(codcli: ?, cArm: Codigo, cLenD: Codigo, cLenE: Codigo)
```

APIs parciais dos subsistemas…

Operações sobre Clientes

```
+getCliente(nome: String, datan: Data): Cliente
```

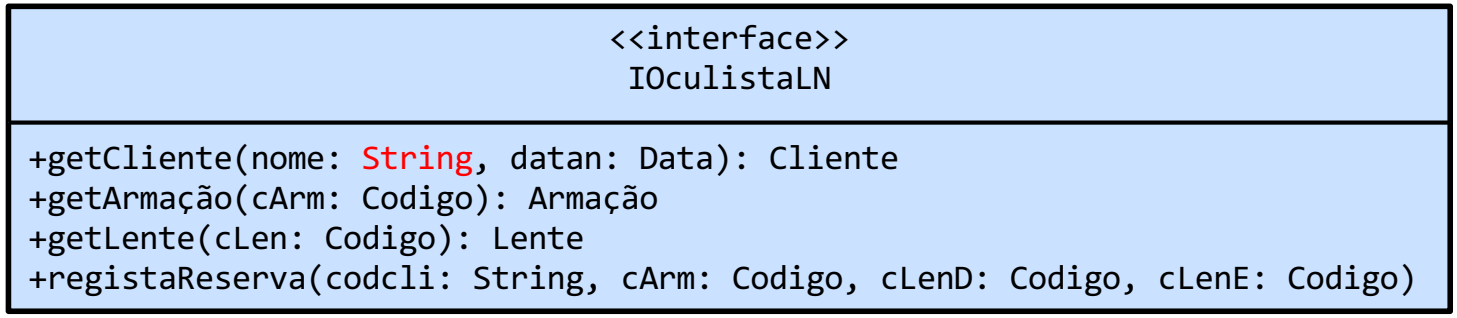
Operações sobre Produtos

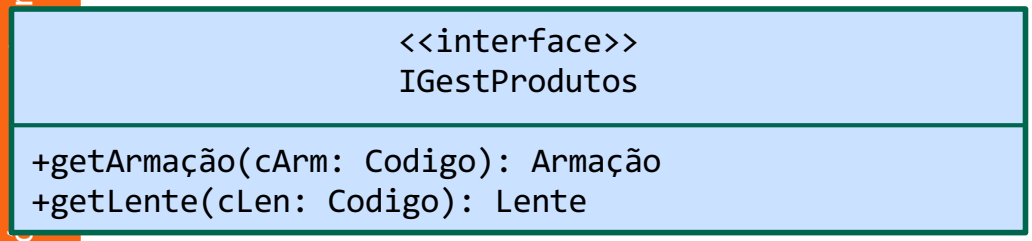
```
+getArmação(cArm: Codigo): Armação
+getLente(cLen: Codigo): Lente
```

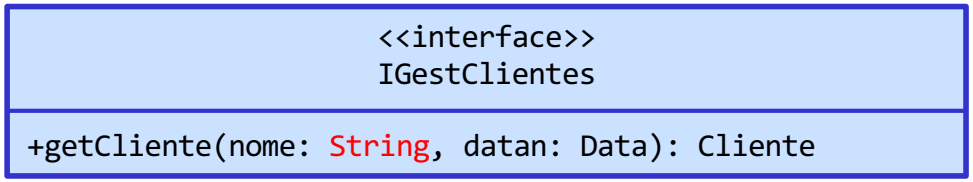
Operações sobre Reservas

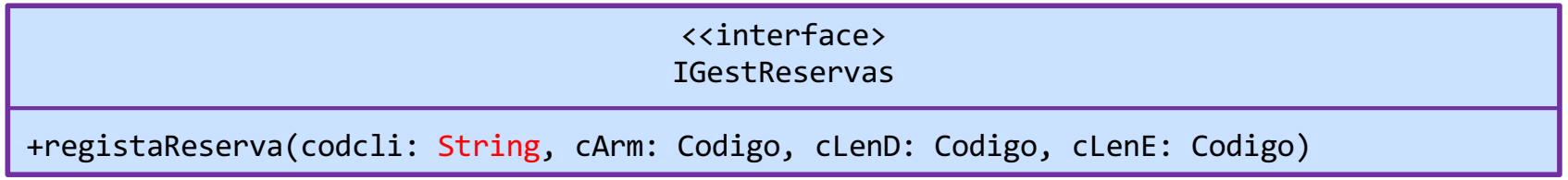
```
+registaReserva(codcli: ?, cArm: Codigo, cLenD: Codigo, cLenE: Codigo)
```

Interfaces para o exemplo

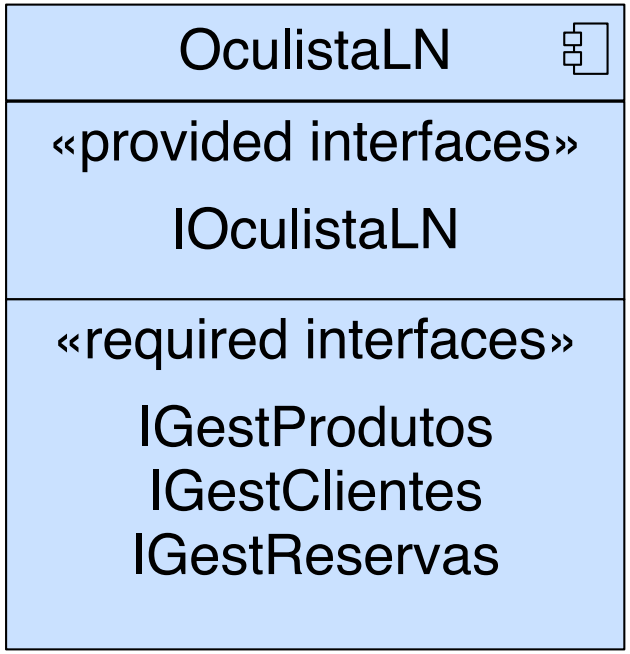




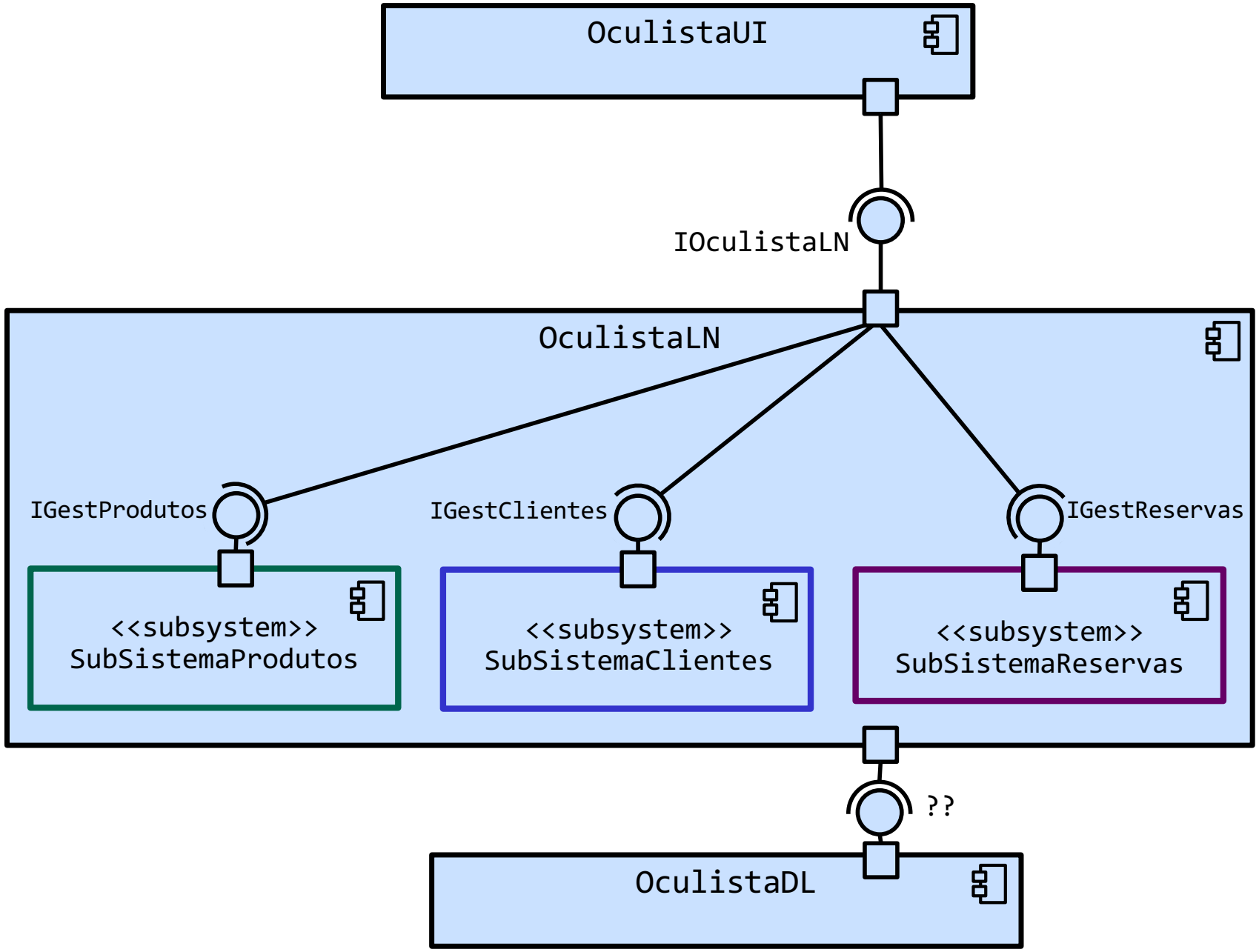




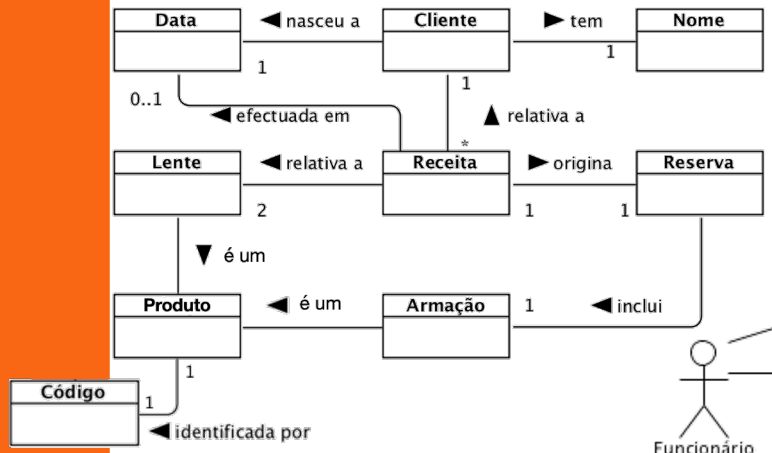
Componente da lógica de negócio

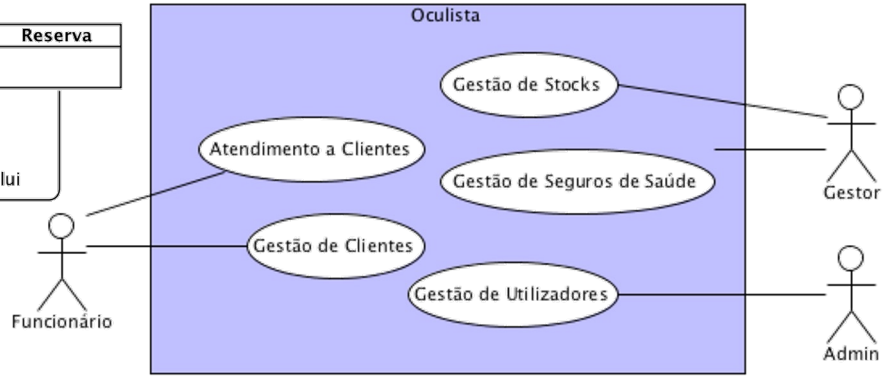


Primeira versão da arquitectura



Em resumo...



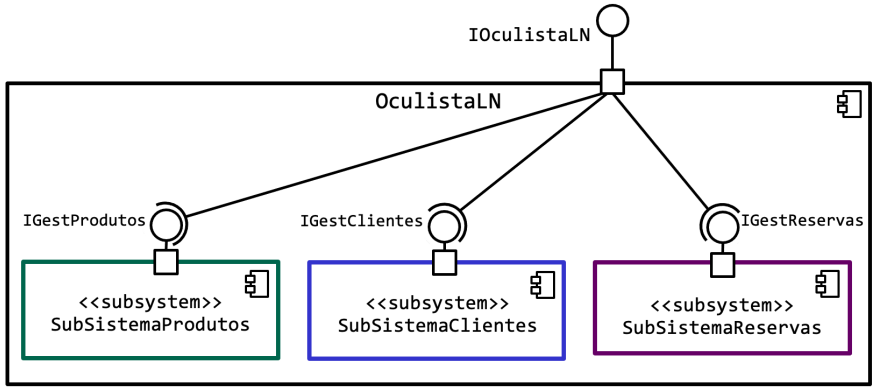


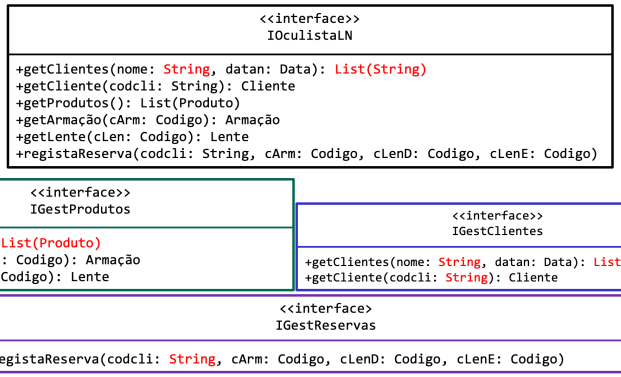
Use Case: Reservar armação e lentes

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 clientes 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



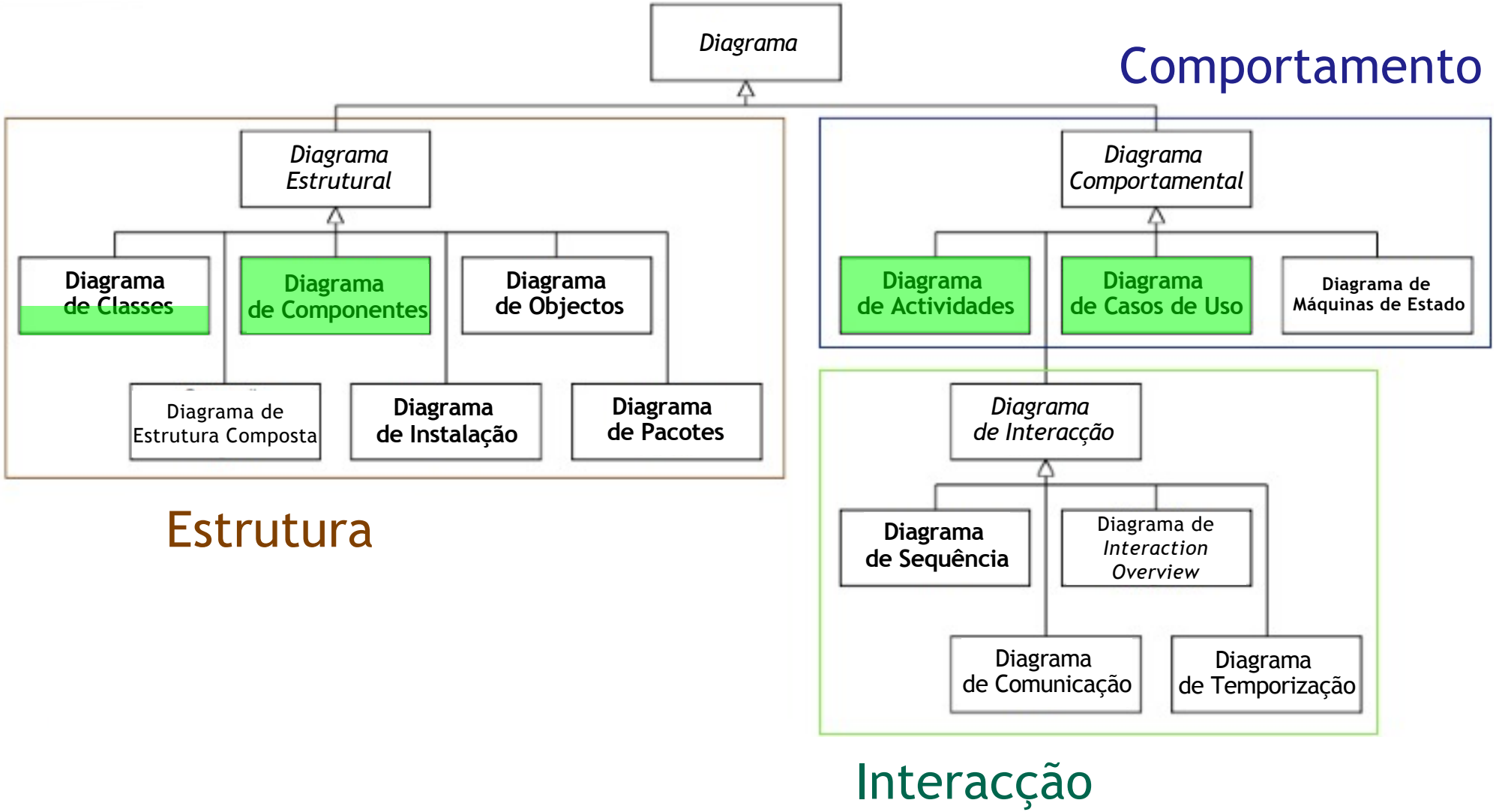


Em resumo…

Em DSS adoptamos o seguinte método para a passagem sistemática de UCs para DSS:

- Para dada Caso de Uso:
  - Identificamos responsabilidades da lógica de negócio
  - Expressamos essas responsabilidades como métodos da interface da lógica de negócio
- Após processar alguns Casos de Uso:
  - Agrupamos os métodos em sub-interfaces
  - Dividimos a lógica de negócio em sub-sistemas (um para cada sub-interface)
- Desenhamos o diagrama de componentes respetivo

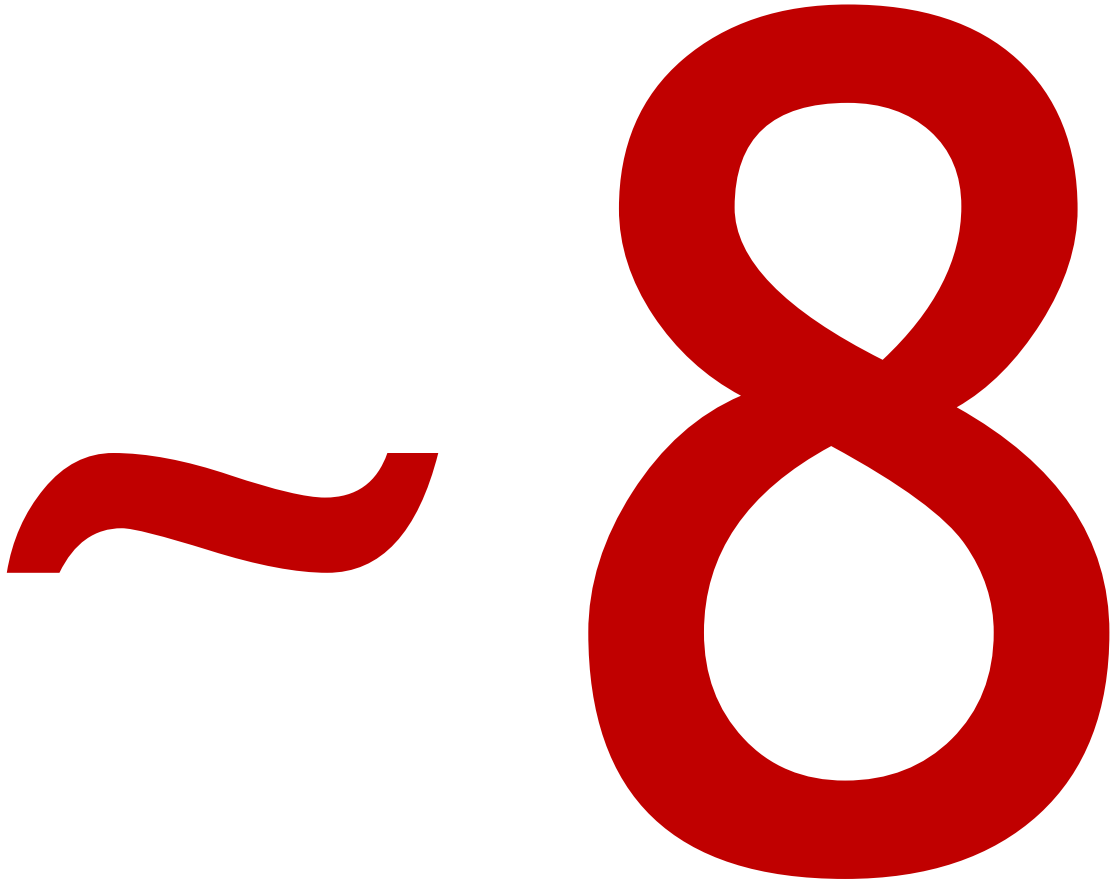
Diagramas da UML 2.x



Convites por aceitar no GitHub

78

Grupos inexistentes no GitHub



Avaliação da Fase 1

- Avaliação qualitativa
  - Também formativa

- Experiência mostra que é melhor começar *na frente*…

Pontos prévios

- Modelo de domínio não é esquema de BD (nem diagrama de classes!)
  - Normalização da BD?!
- Use cases dizem o que o programador fica obrigado a fazer
  - Não o que o utilizador pode fazer
- Não compliquem!!!

- Atenção aos pedidos via Bb
  - Vejam o e-mail
- Funcionamento dos grupos
  - Juntem-se para discutir a integração dos contributos

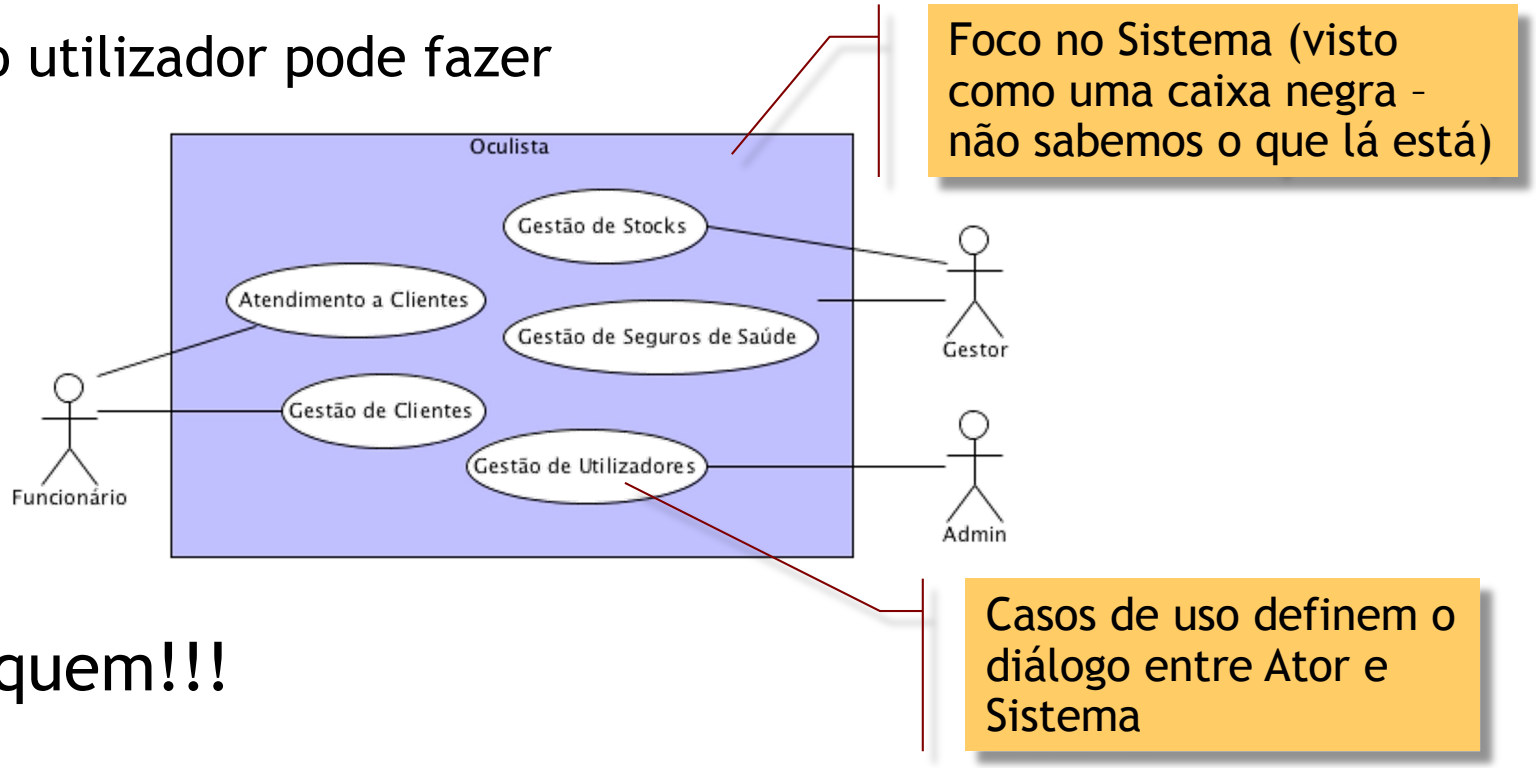
Pontos prévios

- Modelos de domínio não é esquema de BD (nem diagramas de classe!)
  - Normalização da BD?!

- Use cases dizem o que o programador fica obrigado a fazer
  - Não o que o utilizador pode fazer
- Não compliquem!!!

Pontos prévios

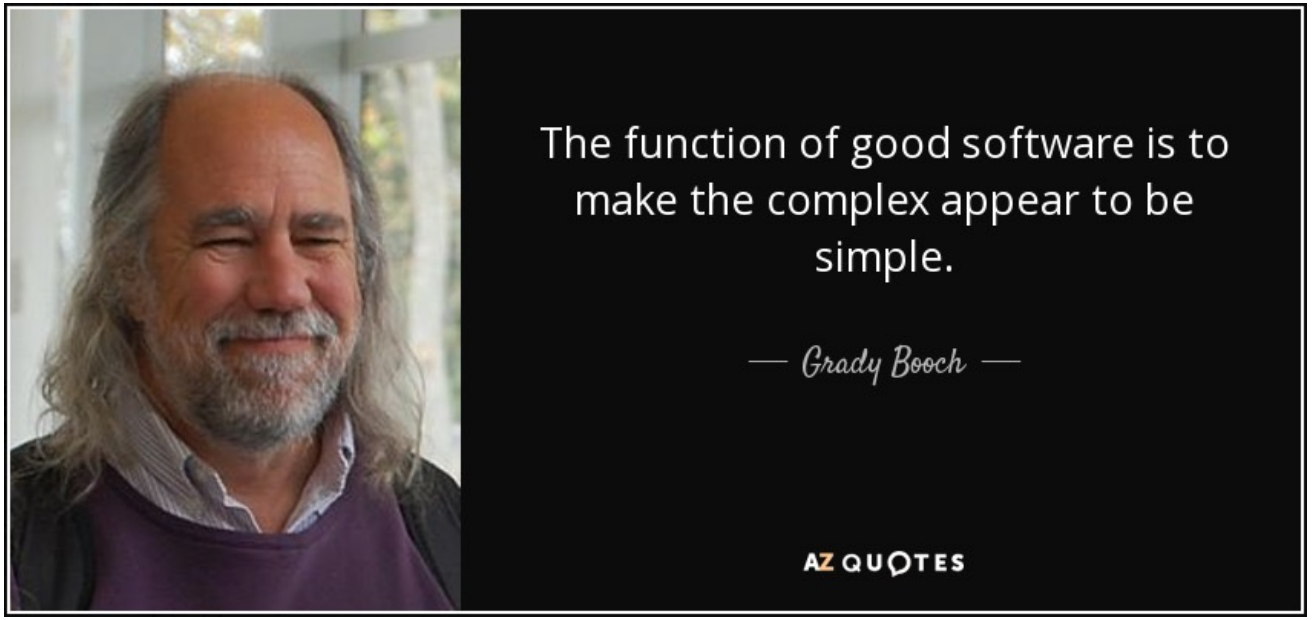
- Modelos de domínio não é esquema de BD (nem diagramas de classe!)
  - Normalização da BD?!
- Use cases dizem o que o programador fica obrigado a fazer
  - Não o que o utilizador pode fazer



- Não compliquem!!!

Pontos prévios

- Modelos de domínio não é esquema de BD (nem diagramas de classe!)
  - Normalização da BD?!
- Use cases dizem o que o programador fica obrigado a fazer
  - Não o que o utilizador pode fazer
- Não compliquem!!!



Ponto prévio – quanto mais simples melhor

- Simples… mas **não** simplista!
- Complexidade essencial vs complexidade acidental
  - Complexidade intrínseca do problema

    vs.
  - Complexidade da solução que criamos