

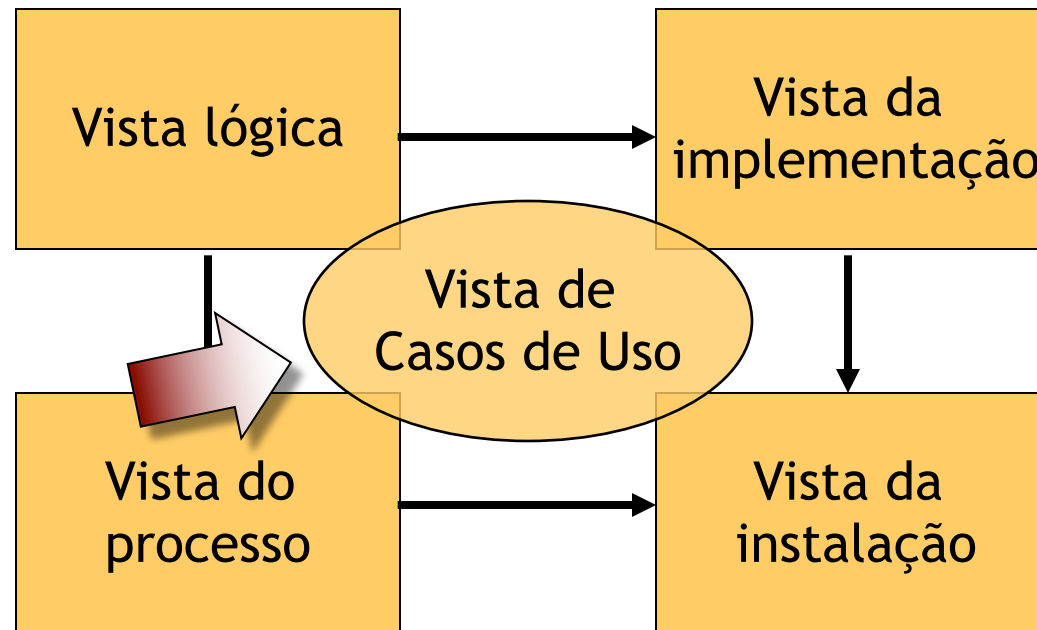


# Desenvolvimento de Sistemas Software

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



# Onde estamos...



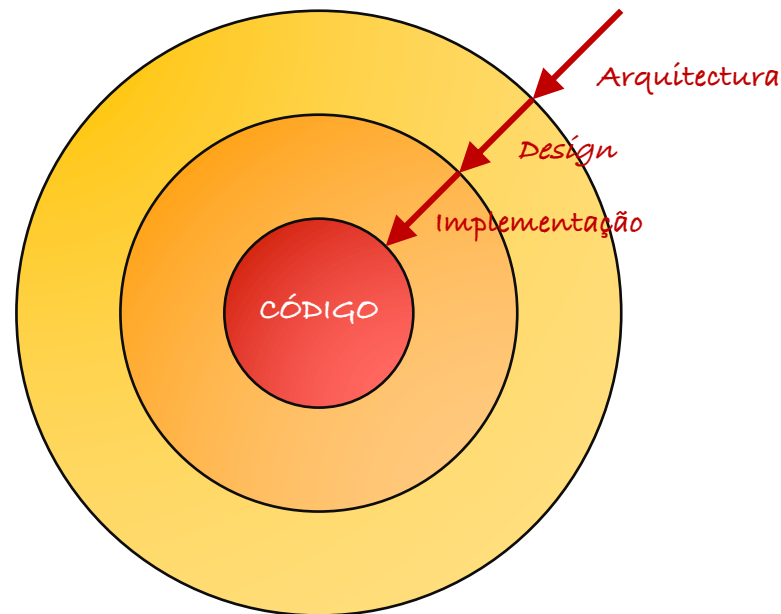
Conceptual

Físico

(Kruchten, 1995)

# 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



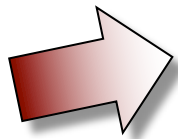
# Fases do ciclo de vida do desenvolvimento de sistemas

## Planeamento

- Decisão de avançar com o projecto
- Gestão do projecto

## Análise

- Análise do domínio do problema
- Análise de requisitos



## Concepção

- Concepção da Arquitectura
- Concepção do Comportamento

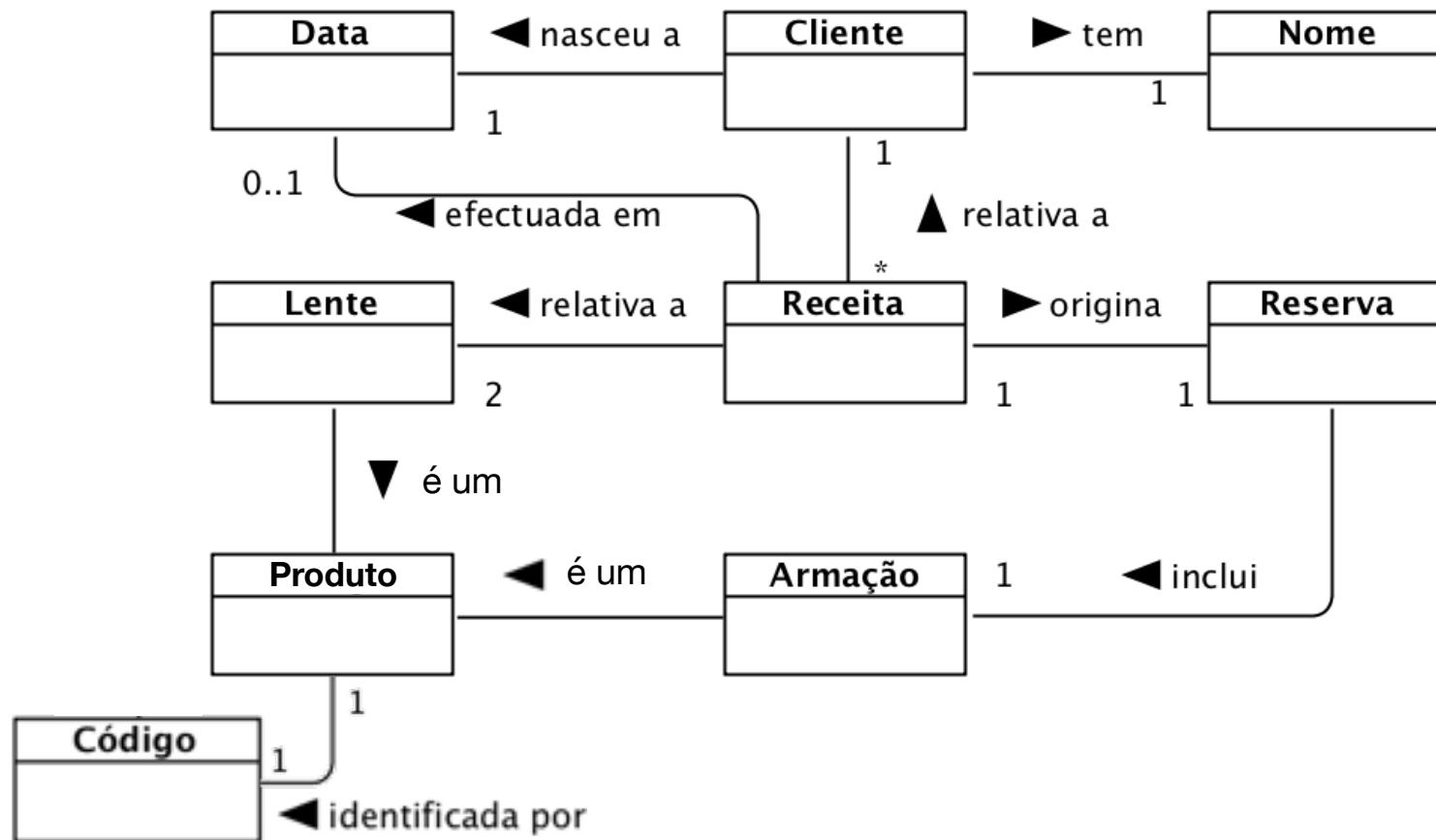
## Implementação

- Construção
- Teste
- Instalação
- Manutenção



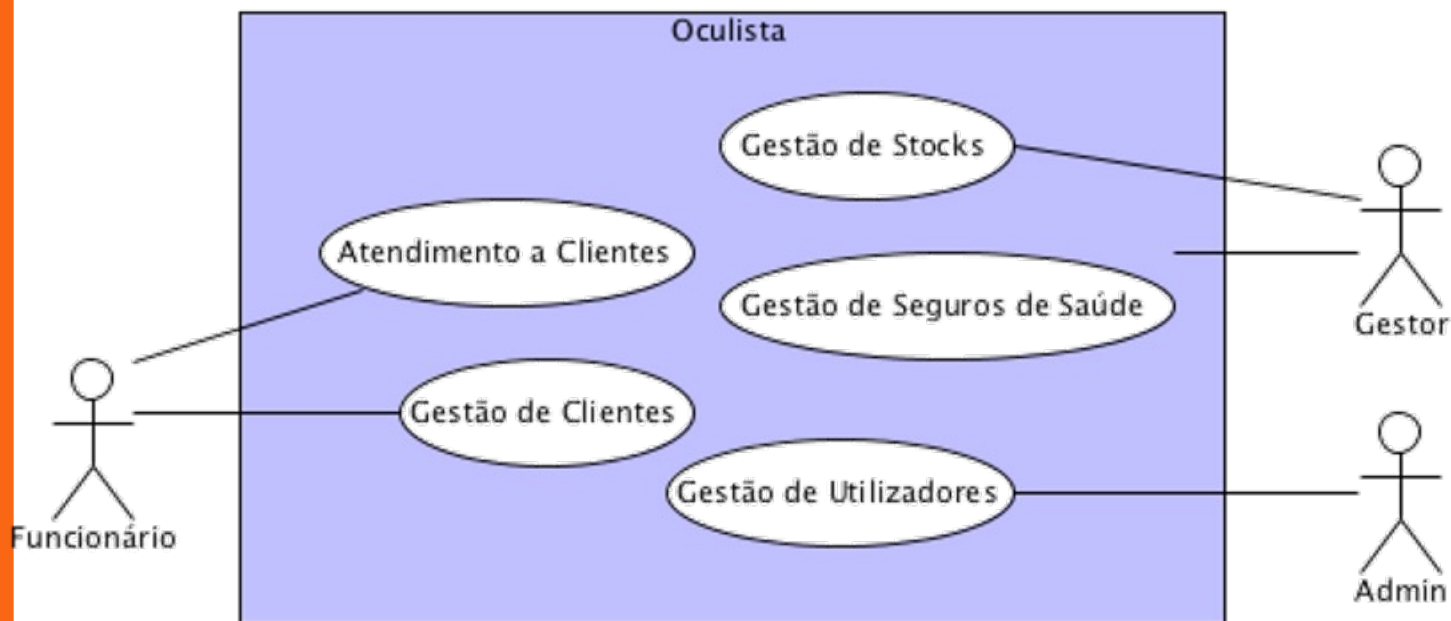
# Um exemplo...

- Modelo de Domínio



# Um exemplo...

## • Diagrama de Use Case



### 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 de exceção:** [cliente não quer produto] (passo 12)

- 11.1. Funcionário rejeita produtos
- 11.2. Sistema termina processo

11.1. Sistema termina processo

11.2. Sistema termina processo



# 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 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 de excepção: [cliente não quer produto] (passo 12)

- 11.1. Funcionário rejeita produtos
- 11.2. Sistema termina processo



# Identificação 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 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 de excepção:** [cliente não quer produto] (passo 12)

- 11.1. Funcionário rejeita produtos
- 11.2. Sistema termina processo





# Identificação de responsabilidades

OculistaLN

procura clientes  
procura cliente  
procura produtos  
procura detalhes dos produtos  
regista reserva dos produtos

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

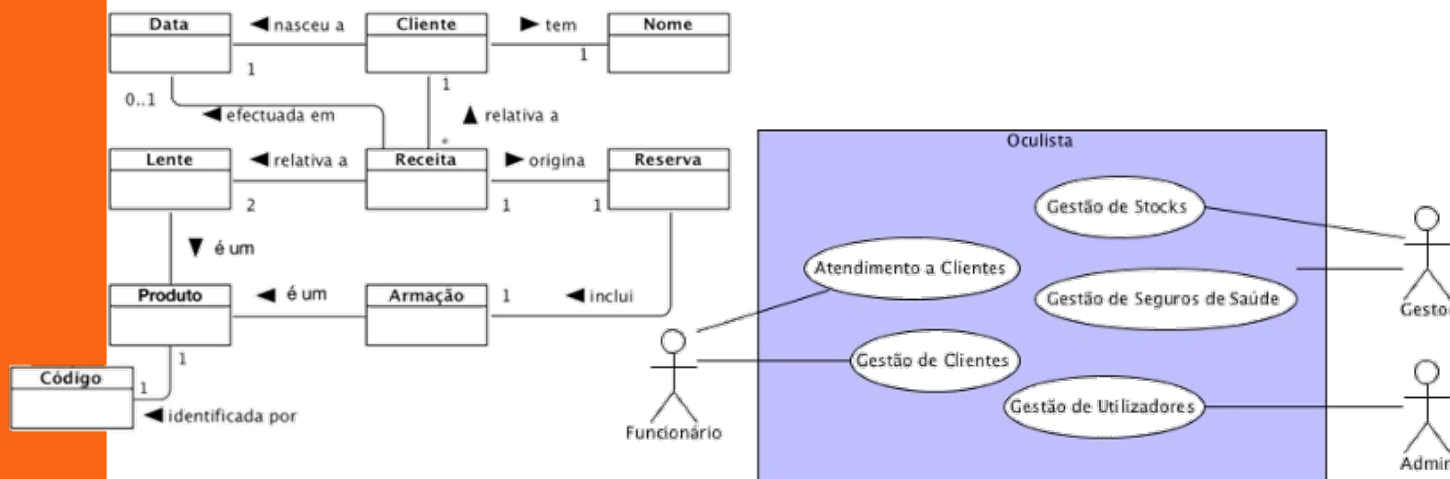
API da lógica de negócios para suportar o Use Case.

OculistaLNFacade

```
+getClientes(nome: ?, datan: Data): ?
+getCliente(codcli: ?): Cliente
+getProdutos(): ?
+getArmação(cArm:Codigo): Armação
+getLente(cLen:Codigo): Lente
+registaReserva(codcli: ?, cArm:Codigo, cLenD:Codigo, cLenE:Codigo)
```



# Identificação de responsabilidades



API da lógica de negócios para suportar o Use Case.

## 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 11)  
11.1. Funcionário rejeita produtos  
11.2. Sistema termina processo

## OculistaLNFacade

```
+getClientes(nome: ?, datan: Data): ?
```

```
+getClient(codcli: ?): Cliente
```

Operações sobre Clientes

```
+getProdutos(): ?
```

```
+getArmação(cArm:Codigo): Armação
```

```
+getLente(cLen:Codigo): Lente
```

Operações sobre Produtos

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

Operações sobre Reservas



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

## de responsabilidades

### OculistaLNFacade

```
+getClientes(nome: ?, datan: Data): ?  
+getCliente(codcli: ?): Cliente  
+getProdutos(): ?  
+getArmação(cArm:Codigo): Armação  
+getLente(cLen:Codigo): Lente  
+registraReserva(codcli: ?, cArm:Codigo, cLenD:Codigo, cLenE:Codigo)
```

APIs parciais dos subsistemas...

```
+getClientes(nome: ?, datan: Data): ?  
+getCliente(codcli: ?): Cliente
```

Operações sobre Clientes

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

Operações sobre Produtos

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

Operações sobre Reservas



# Diagramas de Componentes

- Como definir quais os componentes software do sistema?
  - Modelo em camadas?
  - Sub-sistemas?
  - Utilização de bibliotecas e serviços externos?
- 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



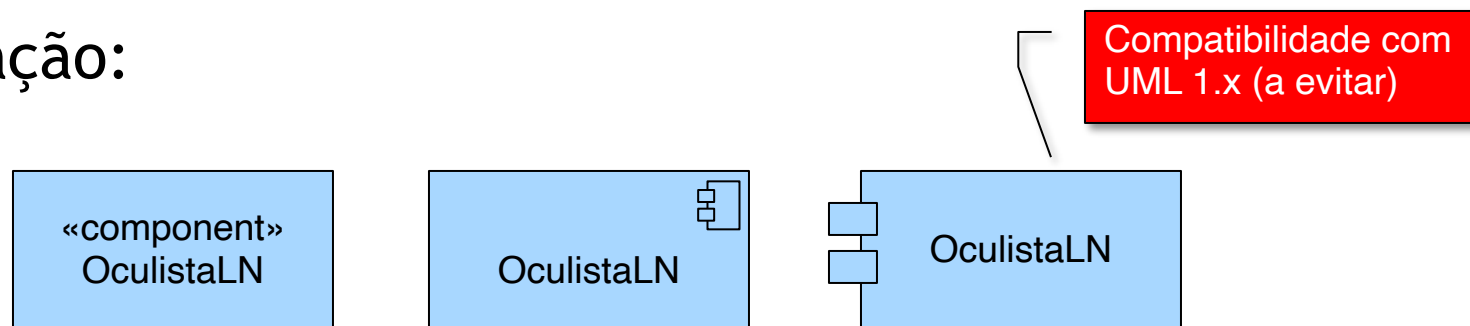
# Diagramas de Componentes

- 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* de XML, componentes de gestão de carrinhos de compra (*shopping carts*), etc.

# Diagramas de Componentes

- Componente
  - Uma parte modular do sistema
  - Comportamento definido pelas interfaces fornecidas/requeridas

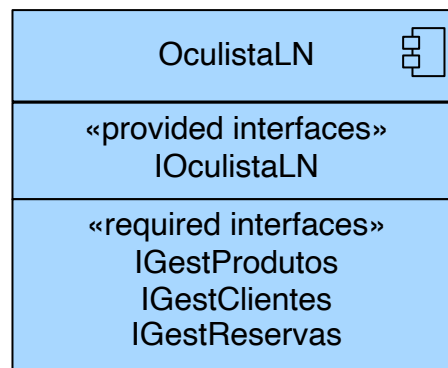
- Notação:



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

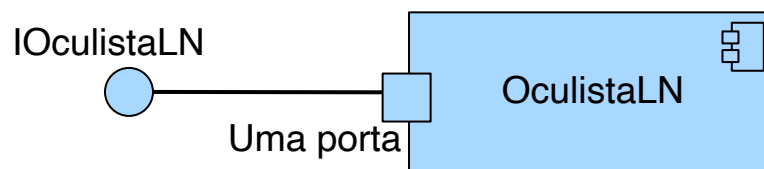
# Diagramas de Componentes

- Interfaces - Indicam os serviços requeridos / fornecidos pelo componente

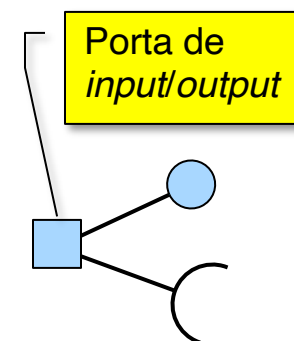
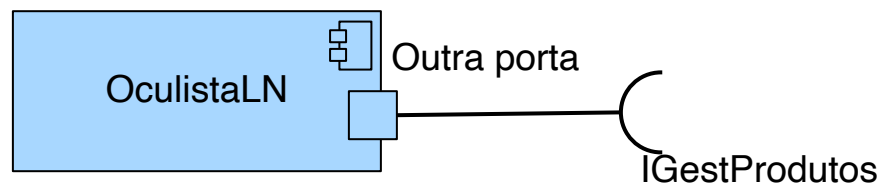


- Portas (*ports*)

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



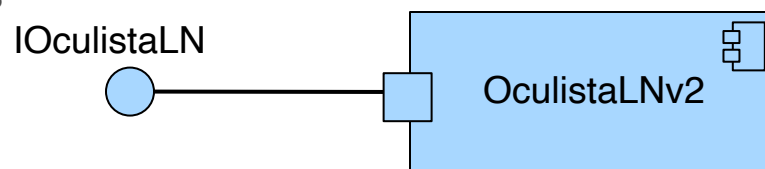
- porta de *input* - Componente requer (utiliza) interface



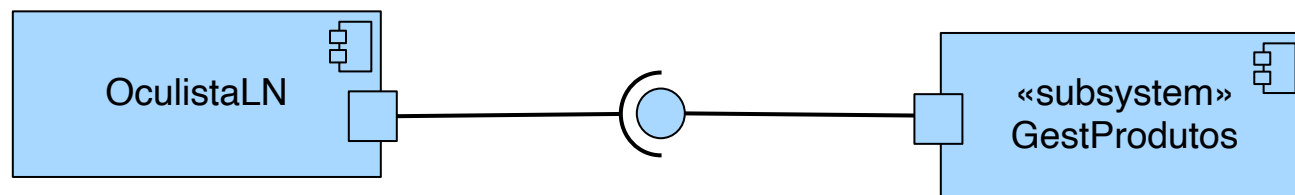


# Diagramas de Componentes

- **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



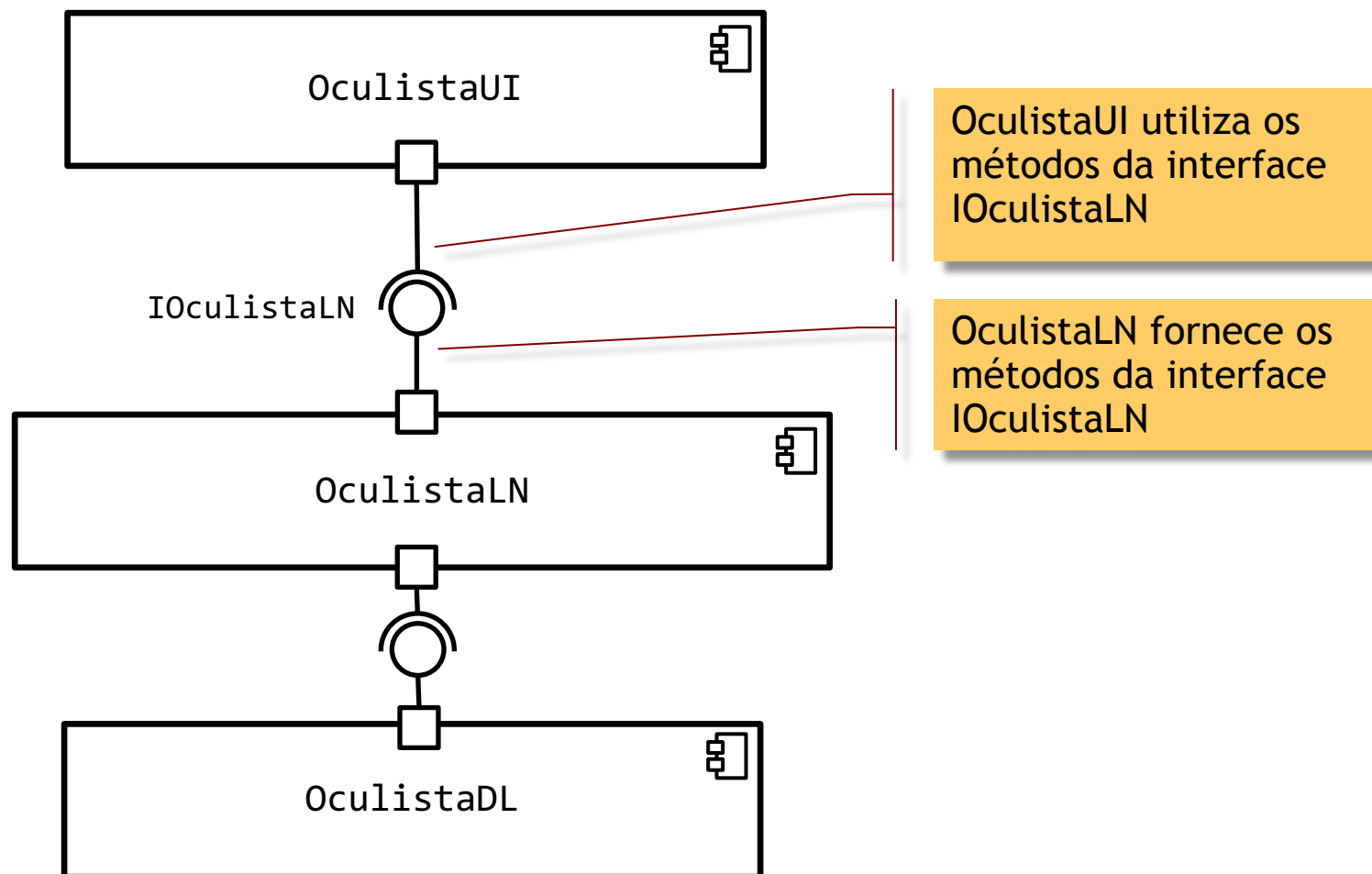
- **Relação de dependência:** um componente pode usar uma ou mais interfaces
  - Diz-se que essas interfaces são requeridas ou importadas
  - Um componente que usa outro componente, através de uma interface bem definida, não deve depender da implementação (do outro componente), mas apenas da interface





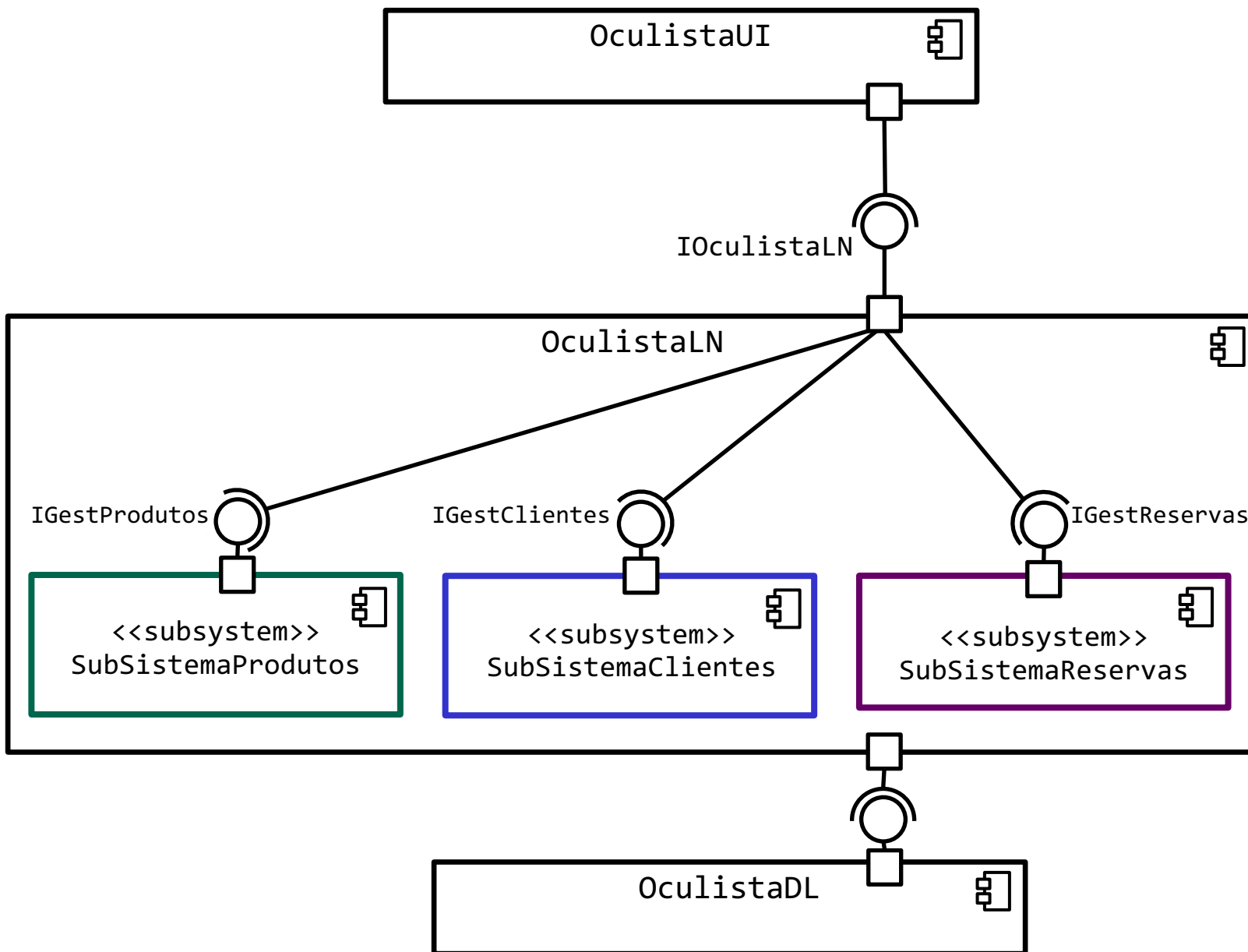


# Arquitetura de 3 camadas para o exemplo





# Primeira versão da arquitectura





# 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 ficam 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).



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

## de responsabilidades

### OculistaLNFacade

```
+getClientes(nome: ?, datan: Data): ?  
+getCliente(codcli: ?): Cliente  
+getProdutos(): ?  
+getArmação(cArm:Codigo): Armação  
+getLente(cLen:Codigo): Lente  
+registraReserva(codcli: ?, cArm:Codigo, cLenD:Codigo, cLenE:Codigo)
```

APIs parciais dos subsistemas...

```
+getClientes(nome: ?, datan: Data): ?  
+getCliente(codcli: ?): Cliente
```

Operações sobre Clientes

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

Operações sobre Produtos

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

Operações sobre Reservas



# Primeira versão da arquitectura

<<interface>> IOculistaLN
<pre> +getClientes(nome: String, datan: Data): List(String) +getCliente(codcli: String): Cliente +getProdutos(): List(Produto) +getArmação(cArm:Codigo): Armação +getLente(cLen:Codigo): Lente +registraReserva(codcli: String, cArm:Codigo, cLenD:Codigo, cLenE:Codigo) </pre>

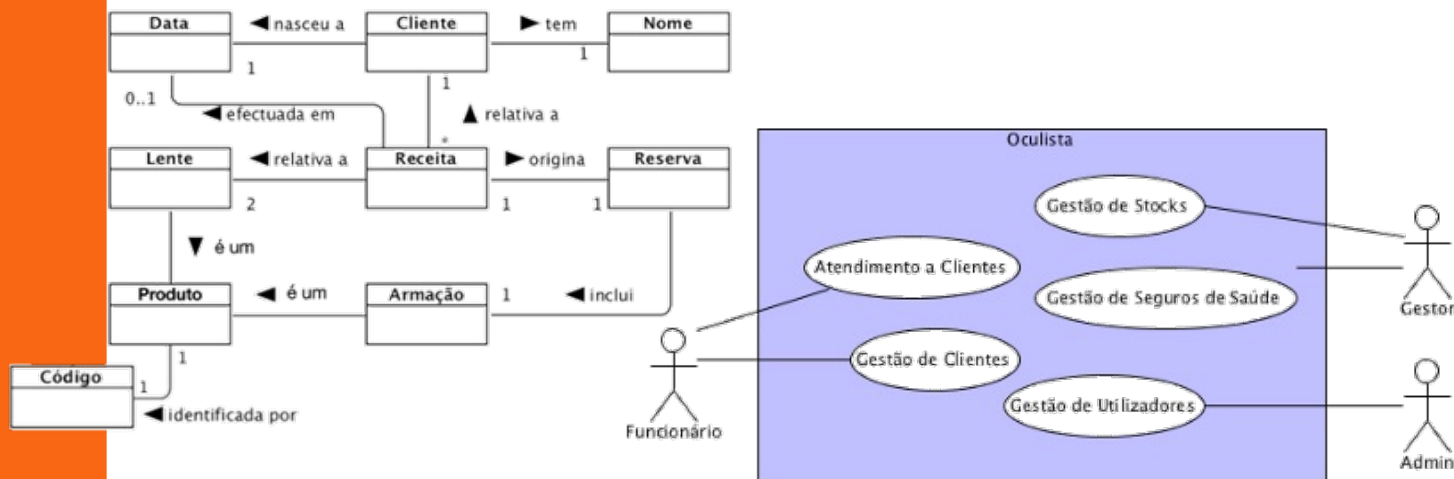
<<interface>> IGestProdutos
<pre> +getProdutos(): List(Produto) +getArmação(cArm:Codigo): Armação +getLente(cLen:Codigo): Lente </pre>

<<interface>> IGestClientes
<pre> +getClientes(nome: String, datan: Data): List(String) +getCliente(codcli: String): Cliente </pre>

<<interface>> IGestReservas
<pre> +registraReserva(codcli: String, cArm:Codigo, cLenD:Codigo, cLenE:Codigo) </pre>



# Em resumo...



## Use Case: Reservar armação e lentes

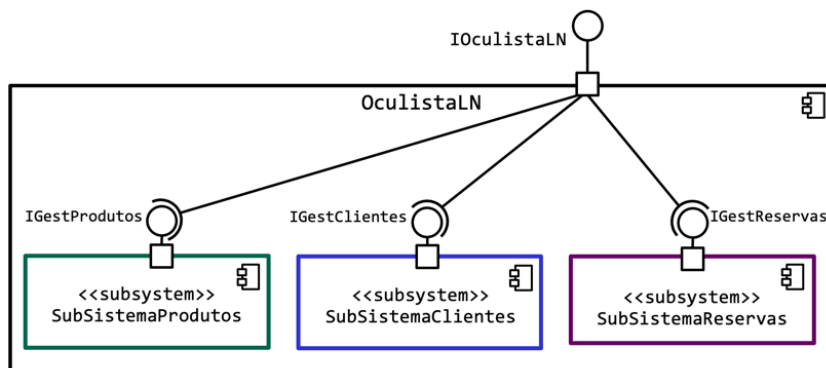
Descrição: Funcionário regista uma reserva de armação e lentes  
 Pós-condição: reserva fica registada

### 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)  
 12.1. Funcionário rejeita produtos  
 12.2. Sistema termina processo



```

<<interface>>
IOculistaLN

+getClientes(nome: String, datan: Data): List(String)
+getClient(codcli: String): Cliente
+getProdutos(): List(Produto)
+getArmação(cArm: Codigo): Armação
+getLente(cLen: Codigo): Lente
+registraReserva(codcli: String, cArm: Codigo, cLenD: Codigo, cLenE: Codigo)
  
```

```

<<interface>>
IGestProdutos

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

```

<<interface>>
IGestClientes

+getClientes(nome: String, datan: Data): List(String)
+getClient(codcli: String): Cliente
  
```

```

<<interface>>
IGestReservas

+registraReserva(codcli: String, cArm: Codigo, cLenD: Codigo, cLenE: Codigo)
  
```



## Em resumo...

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

- Dividimos os fluxos em sequências de *transacções*
- *Identificamos responsabilidades da lógica de negócio*
- *Identificamos métodos*
- *Agrupamos os métodos em sub-sistemas*





# Diagramas da UML 2.x

