T09 - APIs e subsistemas

2 de dezembro de 2023 17:43

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

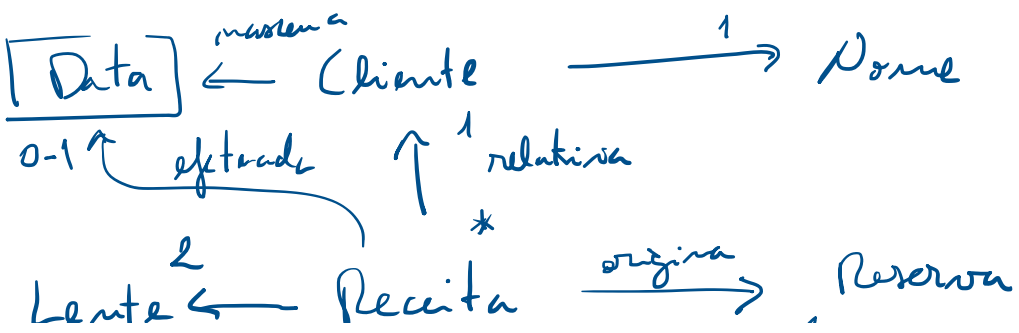
Mais => Melhor,
Simples, mas não simplista

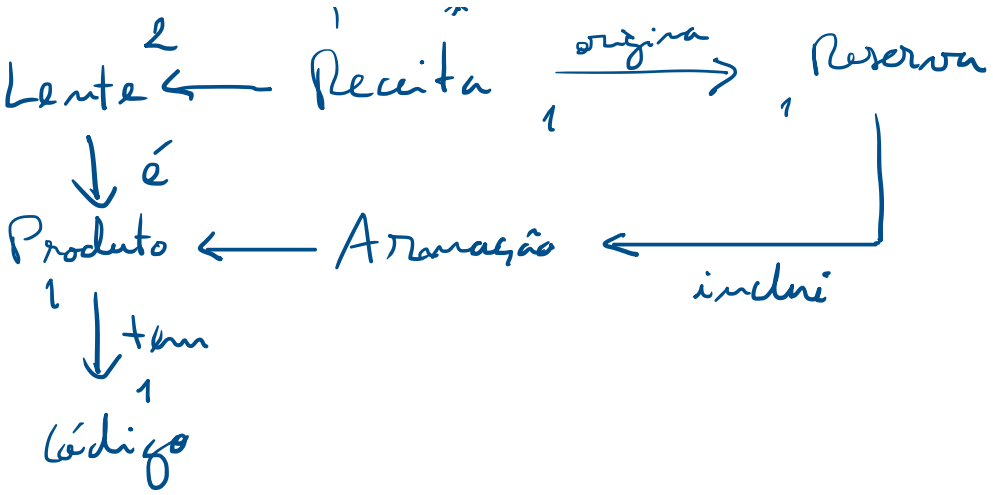
COMPLEXIDADE
- Essencial (intrínseca)
- Acidental (da solução criada)

4 Regras do DESIGN SIMPLES

1. Passar todos os testes
2. Expressar claramente a sua intenção
3. Não conter duplicados
4. Minimizar o número de classes e métodos

Decisões arquiteturais são as mais fundamentais
- alterá-las terá repercurssões (em cadeia) significativas





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)

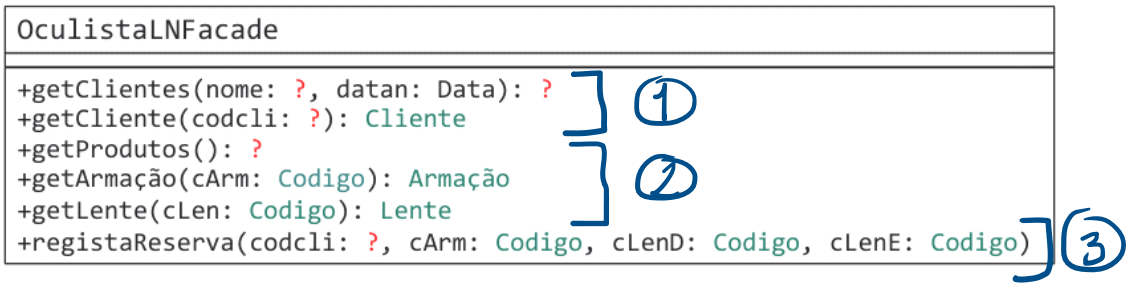
11.1. Funcionário rejeita produtos
11.2. Sistema termina processo

Identificação de responsabilidades

↳ Oculista LN

- Procura clientes
- procura cliente
- procutra produtos
- procura detalhes dos produtos
- regista reserva dos produtos

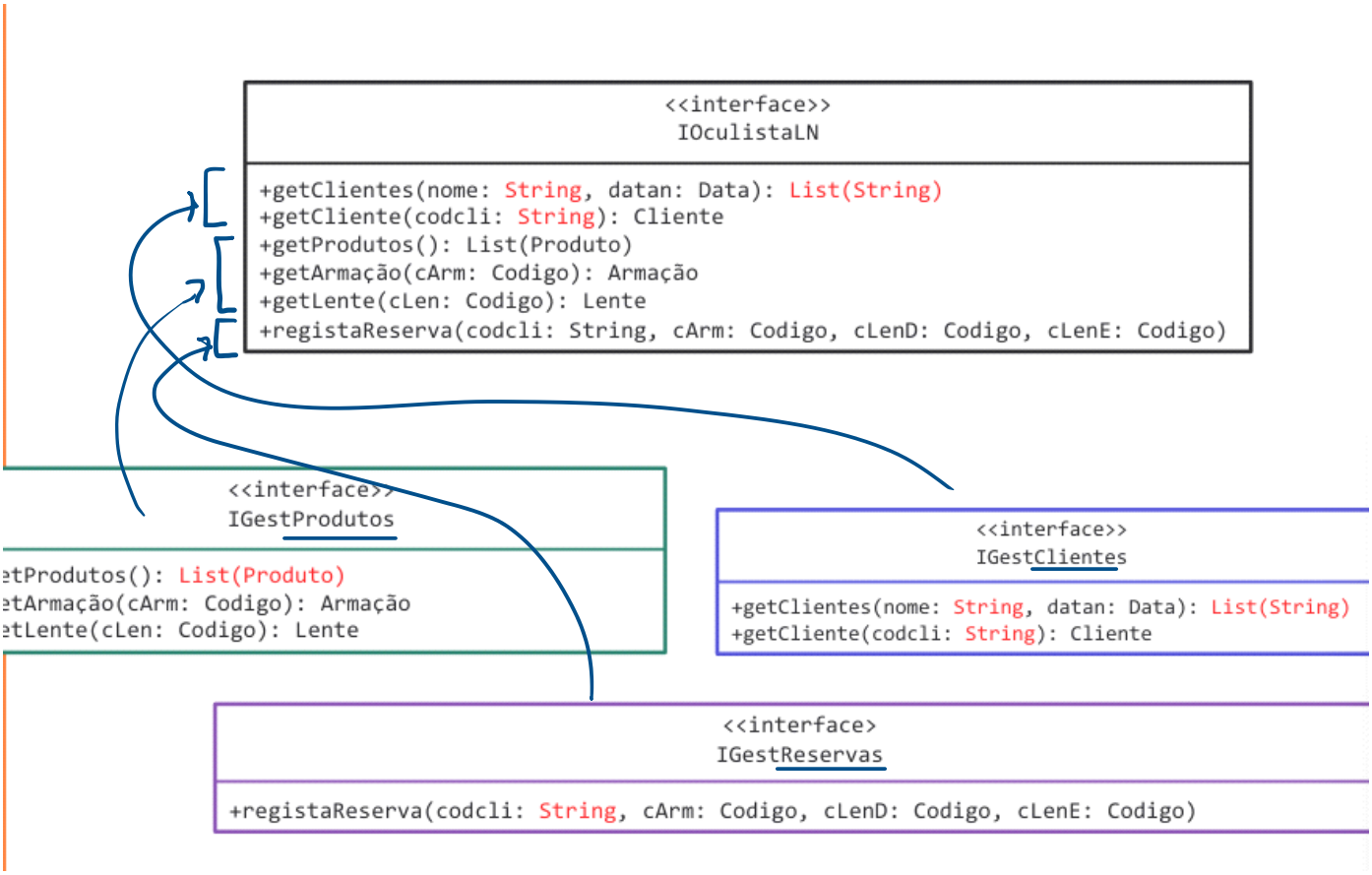
- procura produtos
- procura detalhes dos produtos
- regista reserva dos produtos



1 → Operações sobre Clientes
2 → Operações sobre Produtos
3 → Operação sobre Reservas

INTERFACES

- Uma interface especifica um tipo abstrato: um conjunto de operações externamente visíveis que uma classe (ou componente, subsistema, etc.) deve implementar.
- Semelhante a classe abstrata só com operações abstratas 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)
- Relação 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>>



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 gonte
- componentes binários (e.g. bibiotecas)
- componentes executáveis

Permite identificar, em cada nível, o que é necessário para construir o sistema.

O que é um componente?

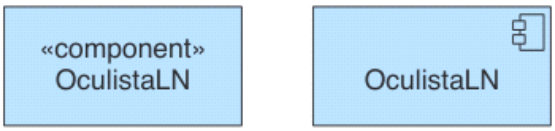
- Um pedaço de software reutilizável, bem encapsulado e "facilmente" substituível.

- São blocos (peças) que combinados constroem o sistema pretendido.
- A dimensão dos componentes não é homogénea, existindo num mesmo sistema, componentes de diferentes dimensões.

Qauis são os bons candidatos a serem componentes do sistema?

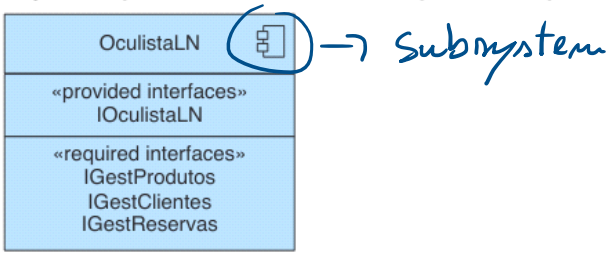
- Os grandes blocos do sistema (cf. arquitetura em camadas)
- Itens que desempenham um funcionalidade que é utilizada recorrentemente em diferentes sistemas
  - Exemplos: componentes de logging, parsers de XML, componentes de gestão de carrinhos de compra, etc.

- Notação:



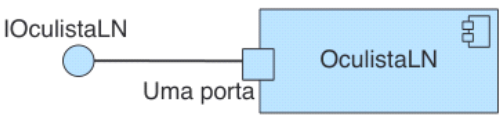
- Alguns estereótipos de Componente:
  - «component» - (!) componente genérico
  - «subsystem» - decomposição hierárquica do sistema global
  - «process» - componente transacional
  - «service» - componente funcional sem estado

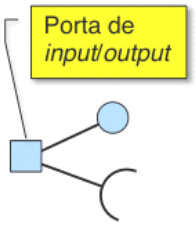
- Interfaces - Indicam os serviços requeridos / fornecidos pelo componente



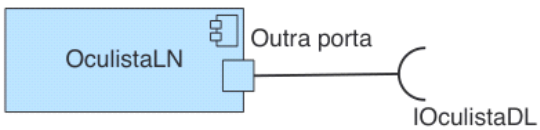
→ subsystem

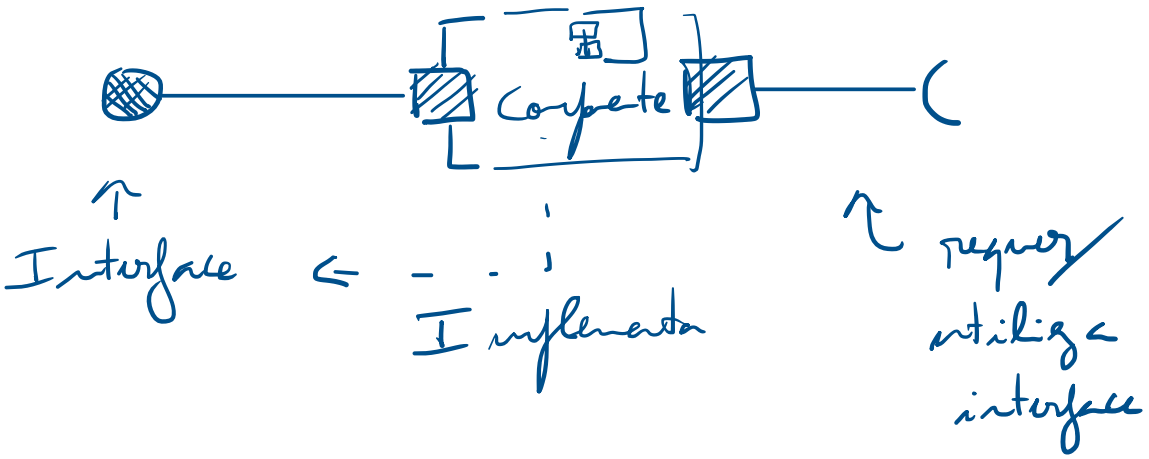
- Portas (*ports*)
  - Identificam pontos de interacção com o componente
  - porta de *output* - Componente fornece (implementa/concretiza) interface





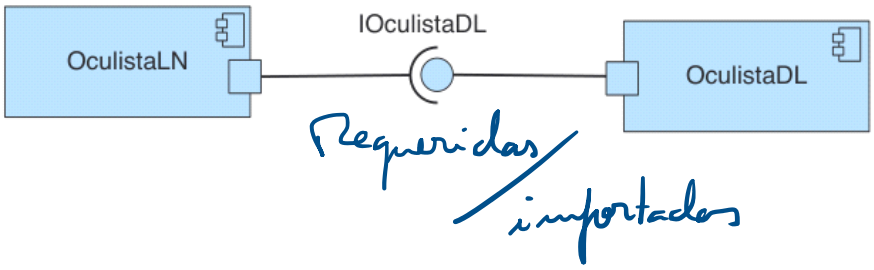
  - porta de *input* - Componente requer (utiliza) interface

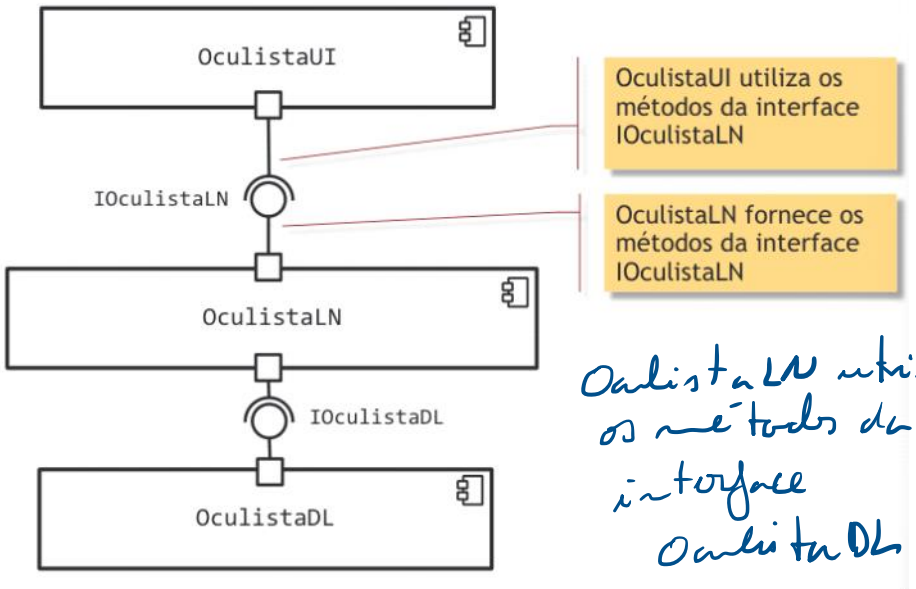


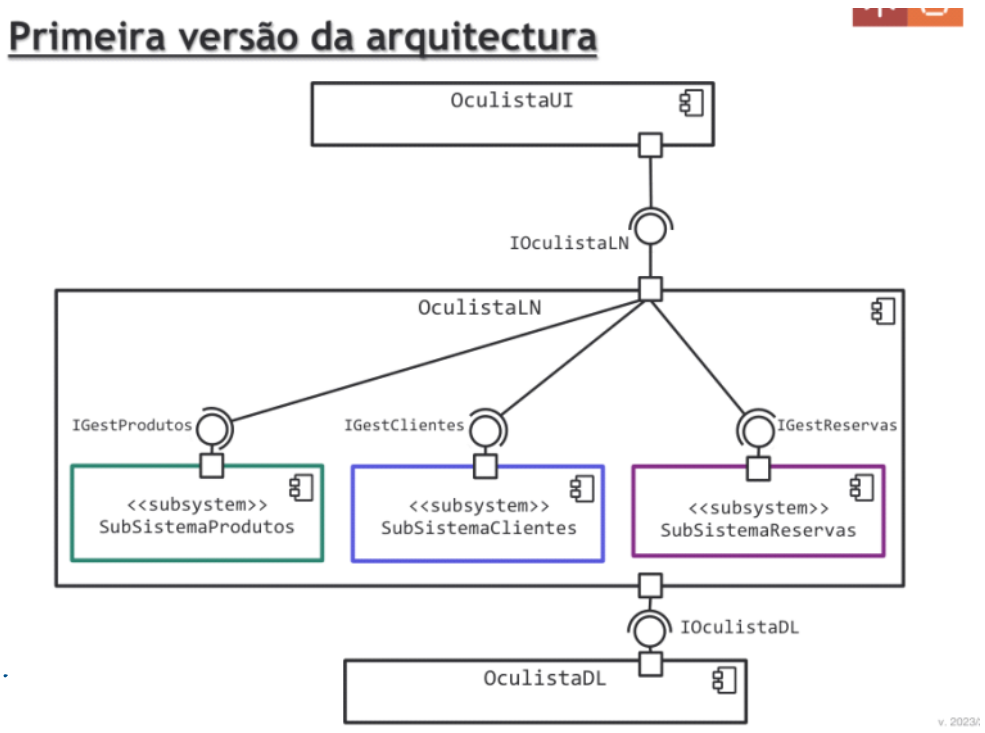


- **Relação de concretização** (*realization*): um componente pode concretizar (implementar os serviços de) uma ou mais interfaces
  - Normalmente quer dizer que tem classes que implementam esses interfaces
  - Diz-se que as interfaces são fornecidas ou exportadas
  - Um componente poderá ser substituído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







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