

40431: Modelação e Análise de Sistemas

# Desenho por objetos: UML na visualização do código

Ilídio Oliveira

v2021-11-26 | TP6

# Objetivos de aprendizagem

Interpretar diagramas de classes (de Código)

Representar construções de código (em Java) nos modelos da UML

Interpretar diagramas de sequência que modelam colaboração entre objetos )

Explicar como os casos de utilização podem ser usados para orientar as atividades de desenho

Explicar os princípios do baixo acoplamento e alta coesão em OO

# Cenário: modelar a colaboração que ocorre num restaurante

**Cliente entra no restaurante e chama o Empregado (de mesa).**

**Cliente pede informações sobre as opções do dia.**

**Empregado anota o novo pedido, com os pratos pedidos.**

**Empregado avisa Cozinha (informa a nova comanda)**

**Cozinheiro confecciona o pedido, usando os ingredientes necessários.**

**Cozinheiro disponibiliza os pratos confeccionados, quando pronto.**

**Empregado entrega pedido ao cliente.**

**Vista estrutural: que “tipos de coisas” (i.e.: classes)?**

Papéis de pessoas?

Lugares e pontos de serviço?

Transações de bens/serviços?

Itens numa transação?

...

**Vista dinâmica: como é que os objetos (instâncias) colaboram?**

Quais os objetos que participam?

O que é que cada objeto solicita de outro?



# “Tipos de coisas”: alguns candidatos

## Papéis de pessoas?

Cliente, Empregado, Cozinheiro

## Lugares e pontos de serviço?

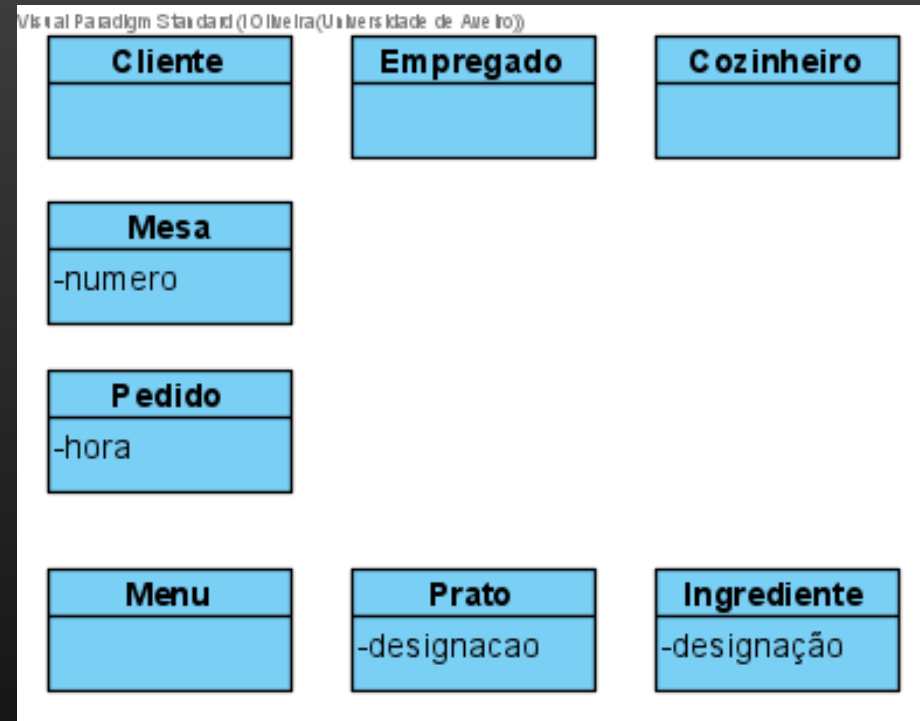
Mesa? Restaurante? Sala?

## Transações de bens/serviços?

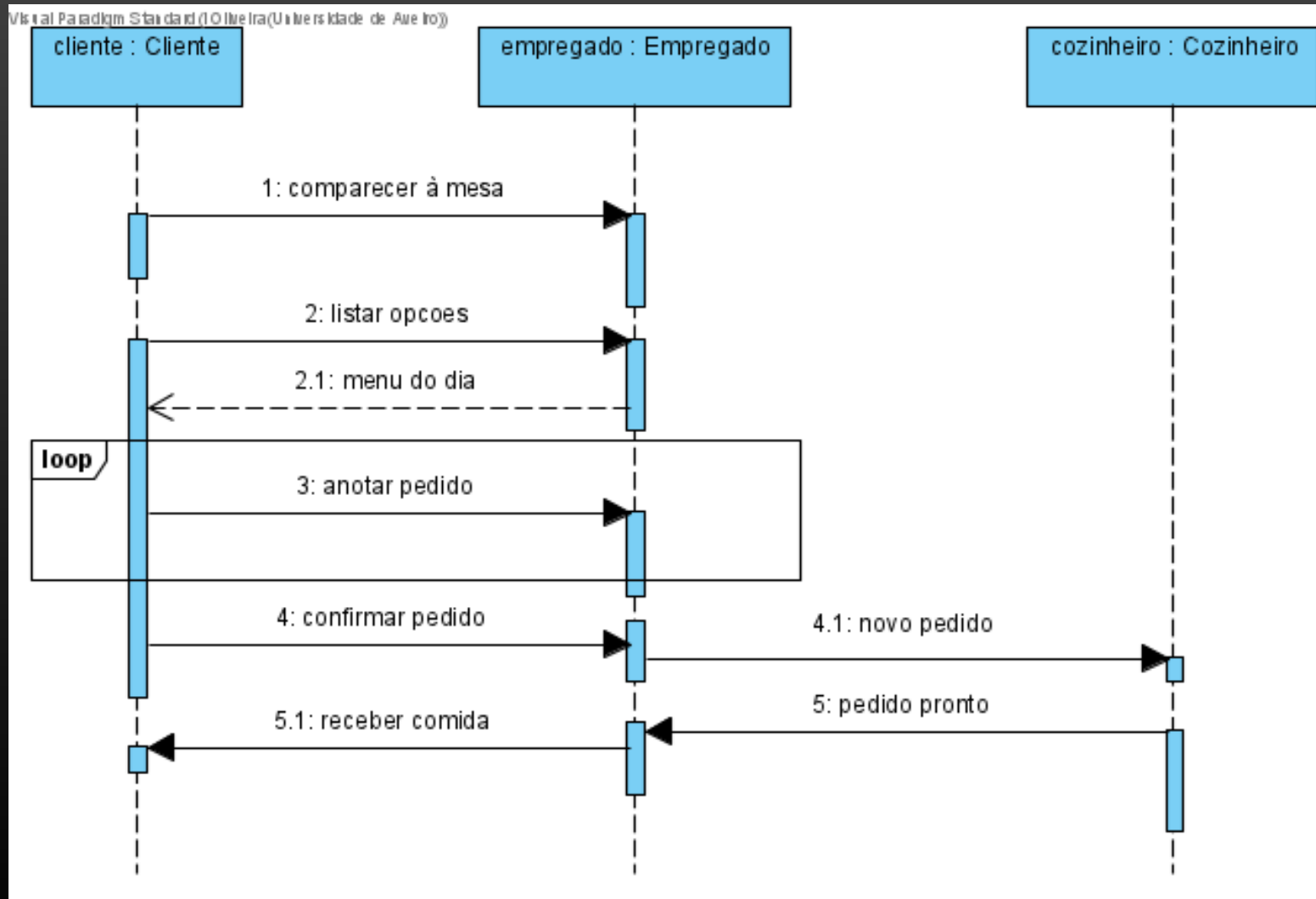
Pedido; comanda/talão?

## Itens numa transação?

Prato/Opção; Menu; Ingredientes?



# Interação entre “participantes”



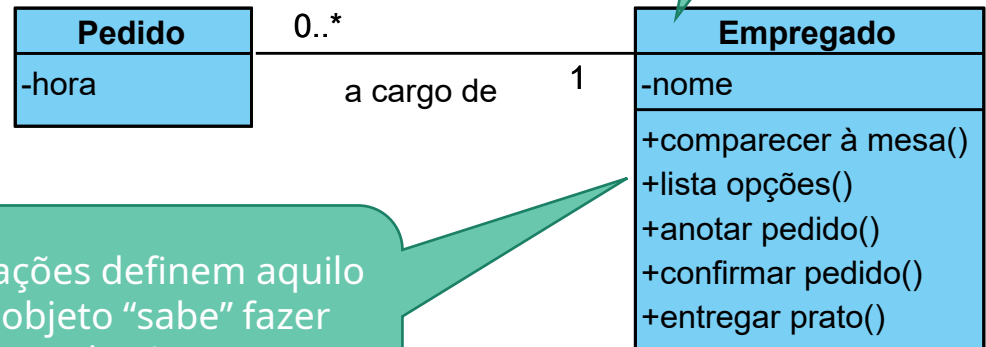
# Os diagramas de classes e os de interação distribuem responsabilidades

## Análise por classes define dois grupos de responsabilidades:

- O que é que cada tipo é responsável por conhecer/guardar
- O que é que cada tipo é responsável por fazer

Atributos e objetos associados definem o âmbito do que o objeto guarda.

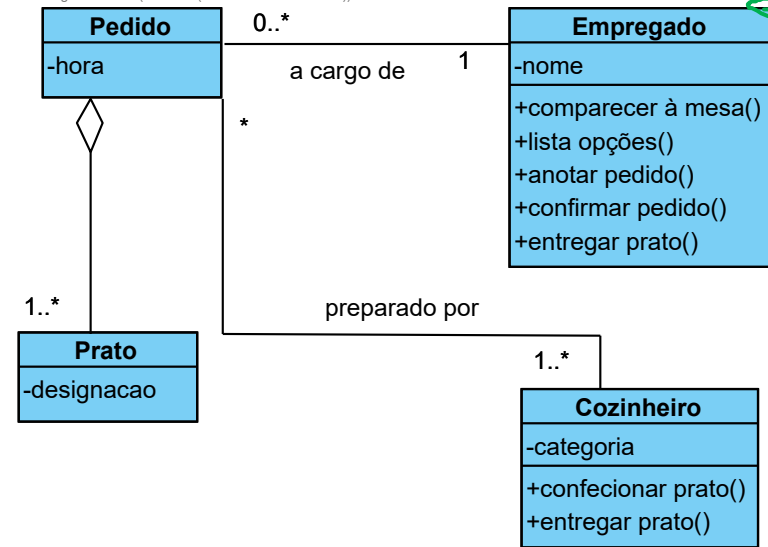
Visual Paradigm Standard (I Oliveira (Universidade de Aveiro))



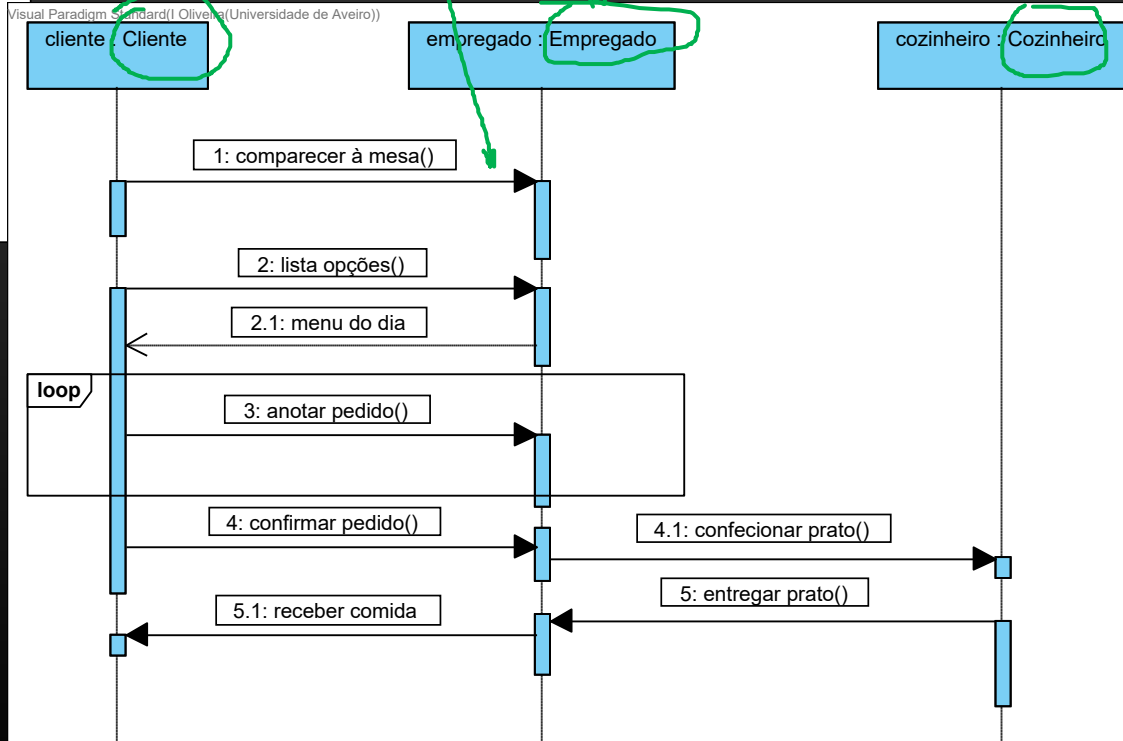
As operações definem aquilo que o objeto "sabe" fazer (sobre o conhecimento que guarda)

# Vista complementares

Visual Paradigm Standard (I. Oliveira (Universidade de Aveiro))



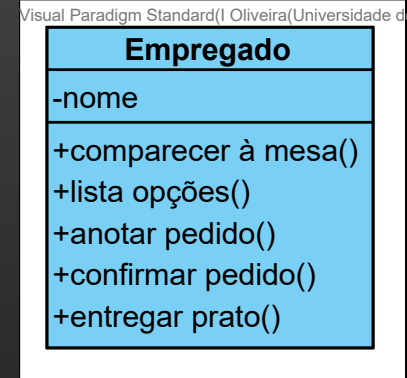
Visual Paradigm Standard (I. Oliveira (Universidade de Aveiro))



# Este raciocínio, no domínio do problema, pode ser aplicado para o para o código?

## Aproveitar o modelo do domínio para “inspirar” a implementação do código!

- Modelo do domínio explora o vocabulário do problema
- Modelo do domínio explica os relacionamentos relevantes e algumas regras (formas de associar objetos)
- A implementação não usa diretamente as representações do domínio do problema...



```
package emp;

public class Empregado {

    private String nome;

    public Empregado() {
    }

    public void anotarPedido(Pedido pedido) {
        // todo
    }

}
```



## Em código....

Não é um conceito do domínio, mas uma entidade que faz sentido no "universe" do software.

Atributos e objetos associados definem aquilo que o objeto "conhece" (o que o objeto guarda)

### A classe passa a representar uma entidade do software

- Pode ser o "mesmo conceito" do domínio
- Mas pode ser outro tipo de entidade, com significado apenas para o software

### Mesmo mecanismo mental

- classificar em tipos (=classes)
- a classe funciona como uma unidade modular, especializada, com conhecimento e operações limitadas
- os objetos são instâncias de classes
- os objetos colaboram "em rede"!

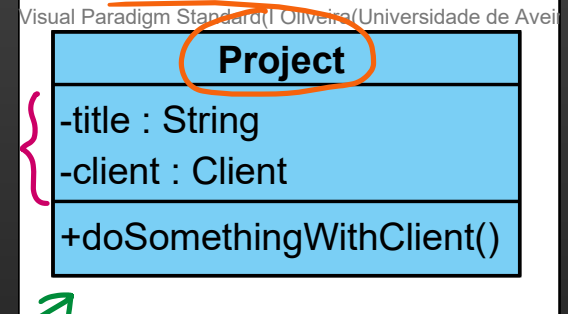
•	EmployeeController	
repository		EmployeeRepository
•	all()	CollectionModel<EntityModel<Employee>>
•	newEmployee(Employee)	Employee
•	one(Long)	EntityModel<Employee>
•	replaceEmployee(Employee, Long)	Employee
•	deleteEmployee(Long)	void

As operações definem aquilo que o objeto "sabe" fazer (sobre o conhecimento que guarda/tem acesso)

# Visualização do código com a UML

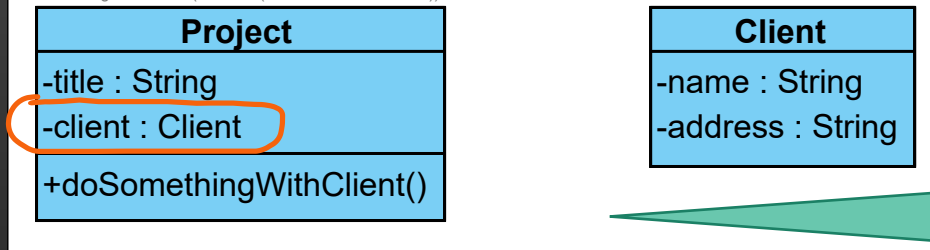
# Visualização de código Java com classes

```
public class Project {  
    private String title;  
    private Client client;  
  
    public void doSomethingWithClient() {  
        // todo  
    }  
}
```



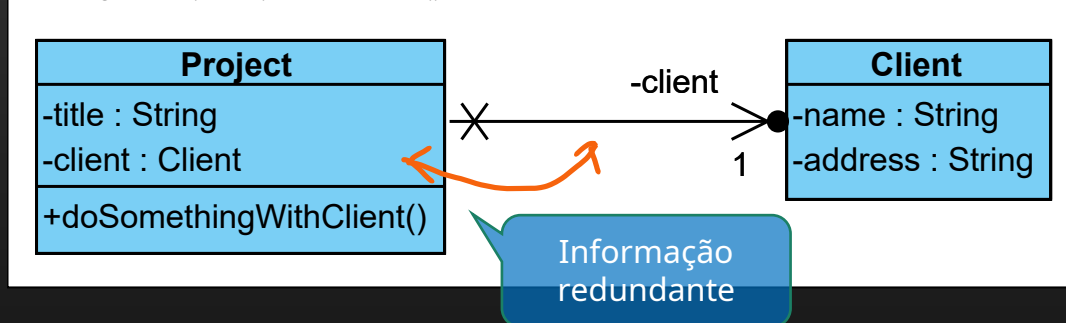
# Visualização do código com classes

Visual Paradigm Standard(I Oliveira(Universidade de Aveiro))



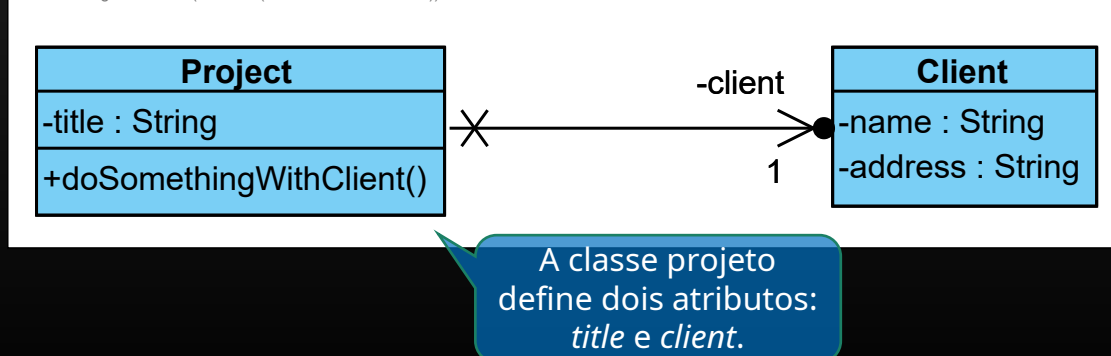
Cada objeto da classe Projeto guarda informação sobre o respetivo Cliente, ou seja, referencia outro objeto.

Visual Paradigm Standard(I Oliveira(Universidade de Aveiro))



Modelos semanticamente equivalentes. Mostrar os atributos como associações evidencia os relacionamentos.

Visual Paradigm Standard(I Oliveira(Universidade de Aveiro))



```

public class ClientsPortfolio {

    private ArrayList<Client> myClientsList;

    public ClientsPortfolio() {
        myClientsList = new ArrayList<>();
    }

    public void addClient(Client newClient) {
        this.myClientsList.add(newClient);
    }

    public int countClients() {
        return this.myClientsList.size();
    }

}

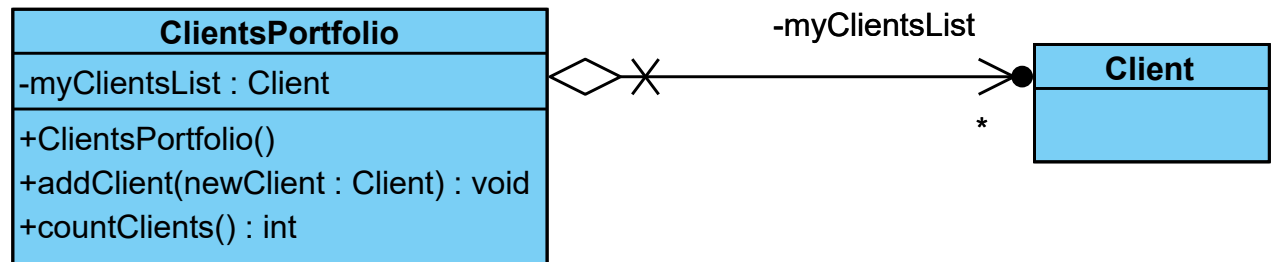
```

Classe

Atributo (neste caso, é uma lista de objetos do tipo Client)

Operação especial: usado na inicialização de cada instância da classe (método Construtor)

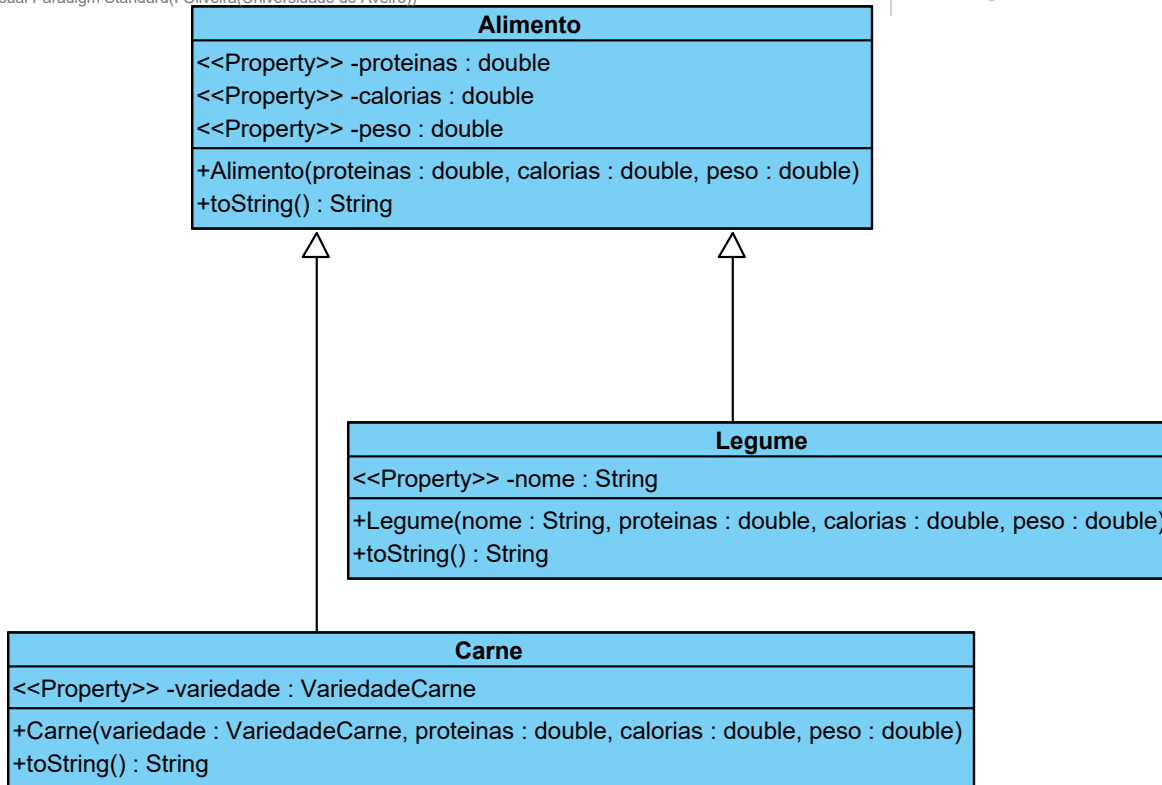
Operações (que podem requerer parâmetros e produzir um valor de retorno ou *void*)



# Generalização

```
public class Legume extends Alimento {  
    private String nome;  
  
    public Legume(String nome, double proteínas, double calorias, double peso) {  
        super(proteínas, calorias, peso);  
        this.nome = nome;  
    }  
}
```

Visual Paradigm Standard (I Oliveira (Universidade de Aveiro))



# O esteriótipo "property"

Visual Paradigm Standard (I Oliveira(Universidade de Aveiro))

Cliente
<<Property>> -nome : String
<<Property>> -descontoComercial : double
+Cliente(nome : String, descontoComercial : double)

```
public class Cliente {  
    private String nome;  
    private double descontoComercial;  
  
    public Cliente(String nome, double descontoComercial) {  
        this.setNome(nome);  
        this.setDescontoComercial(descontoComercial);  
    }  
  
    public String getNome() {  
        return nome;  
    }  
  
    public void setNome(String nome) {  
        this.nome = nome;  
    }  
  
    public double getDescontoComercial() {  
        return descontoComercial;  
    }  
  
    public void setDescontoComercial(double descontoComercial) {  
        this.descontoComercial = descontoComercial;  
    }  
}
```

As operações que têm o nome igual ao da classe chamam-se construtores, e são usados para obter instâncias, passando dados de inicialização do objeto.

Uma vez que os atributos são geralmente de acesso privados do objeto, em Java, é comum o "trio":

- **Atributo *abc***
- ***get*Abc()**
- ***set*Abc()**

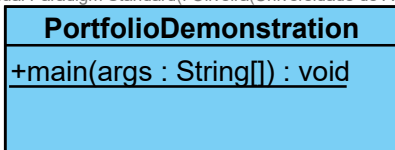
Podemos associar o esteriótipo "property" e omitir os getters e setters

# Objetos enviam mensagens

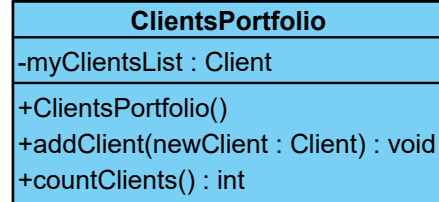
Operação especial: esta classe pode ser usada para arrancar um programa.

```
public class PortfolioDemonstration {  
    public static void main(String[] args) {  
        // obter um novo objeto da classe ClientsPortfolio  
        ClientsPortfolio portfolio = new ClientsPortfolio();  
  
        // obter um novo objeto da classe Cliente e adicioná-lo ao portfolio  
        Client client1= new Client( "C103", "Logistica Tartaruga");  
        portfolio.addClient( client1 );  
  
        Client client2 = new Client( "C104", "Jose, Maria & Jesus Lda");  
        portfolio.addClient( client2 );  
  
        System.out.println( "Clients count: " + portfolio.countClients() );  
    }  
}
```

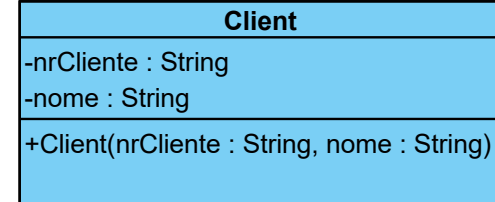
Visual Paradigm Standard (I Oliveira (Universidade de Aveiro))



Visual Paradigm Standard (I Oliveira (Universidade de Aveiro))



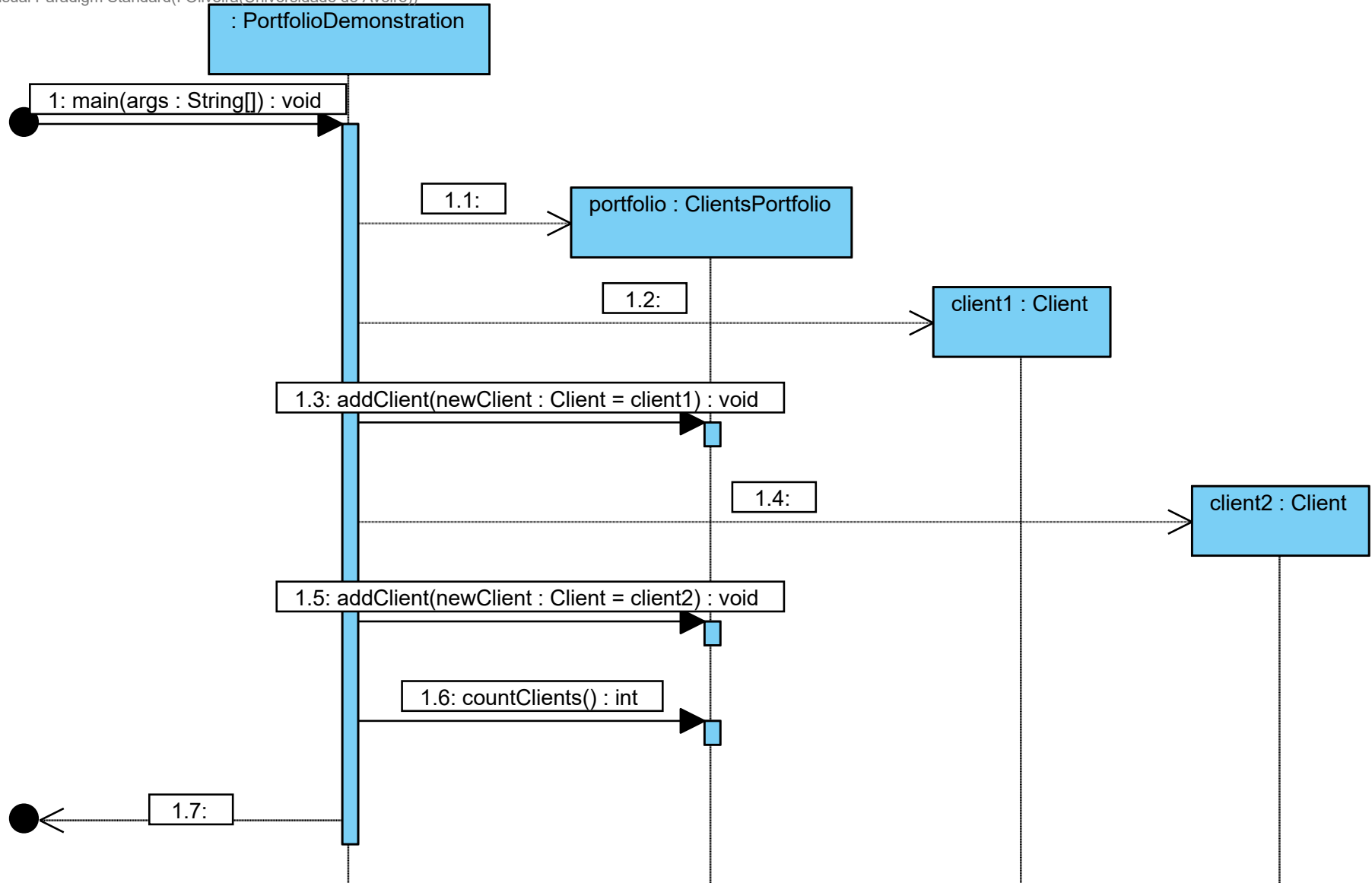
Visual Paradigm Standard (I Oliveira (Universidade de Aveiro))





## ...que podem ser vistas num modelo dinâmico

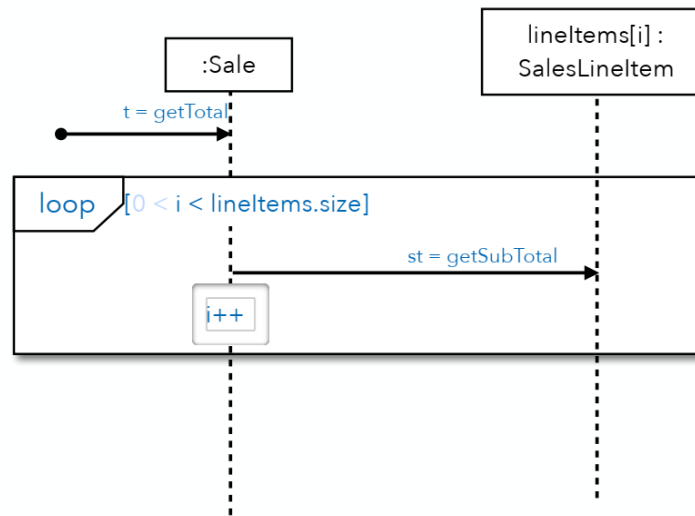
Visual Paradigm Standard (I Oliveira(Universidade de Aveiro))



# Alguns exemplos adicionais

Use a **UML loop frame** to iterate over a collection.

UML Sequence Diagrams | 28



Modeling task: Calculate the total of a sale by summing up the sub totals for each sales line item.

Pag. 27 a 34

[http://stg-tud.github.io/eise/WS18-SE-08-Modeling-dynamic\\_Part.pdf](http://stg-tud.github.io/eise/WS18-SE-08-Modeling-dynamic_Part.pdf)

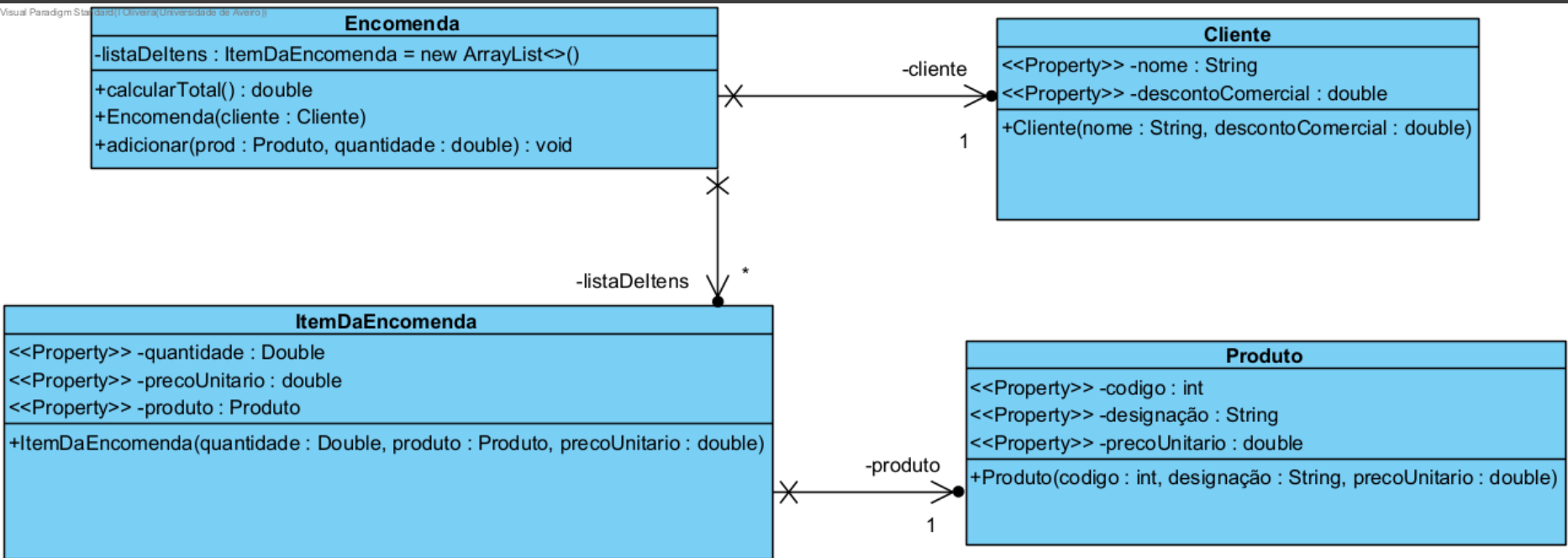
# UML para “visualizar” o código: estrutura e interação

# 0 objetos Java colaboram para realizar objetivos

```
public class Encomenda {  
  
    private Cliente cliente;  
    private ArrayList<ItemDaEncomenda> listaItens;  
  
    public double getTotal() {  
        double total = 0.0;  
  
        Produto produto;  
        for (ItemDaEncomenda item : this.listaItens) {  
            produto = item.getProduto();  
            total += produto.getPrecoUnitario() * item.getQuantidade();  
        }  
  
        total = total * (1 - this.cliente.getDesconto());  
        return total;  
    }  
  
    public Encomenda(Cliente theClient) {  
        super();  
        this.cliente = theClient;  
        listaItens = new ArrayList<ItemDaEncomenda>();  
    }  
}
```

Quais são as classes envolvidas?  
O que podemos descobrir sobre o seu "esqueleto" (operações e assinaturas, atributos)?

# Vista estrutural (definição das classes)

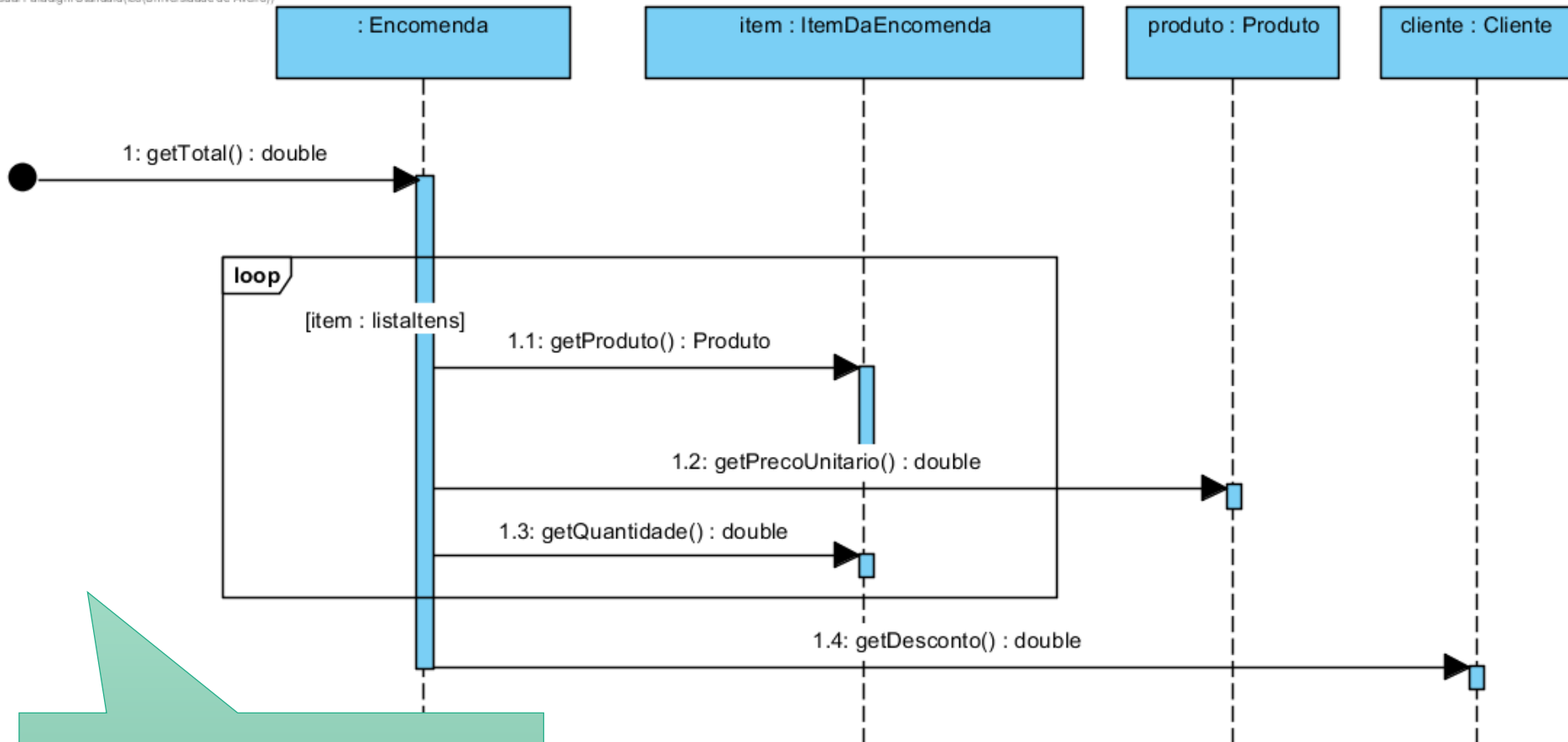


Os atributos que implicam um relacionamento entre classes estão representados como associações.

O esteriótipo <<Property>> marca atributos que têm *getter* e *setter*

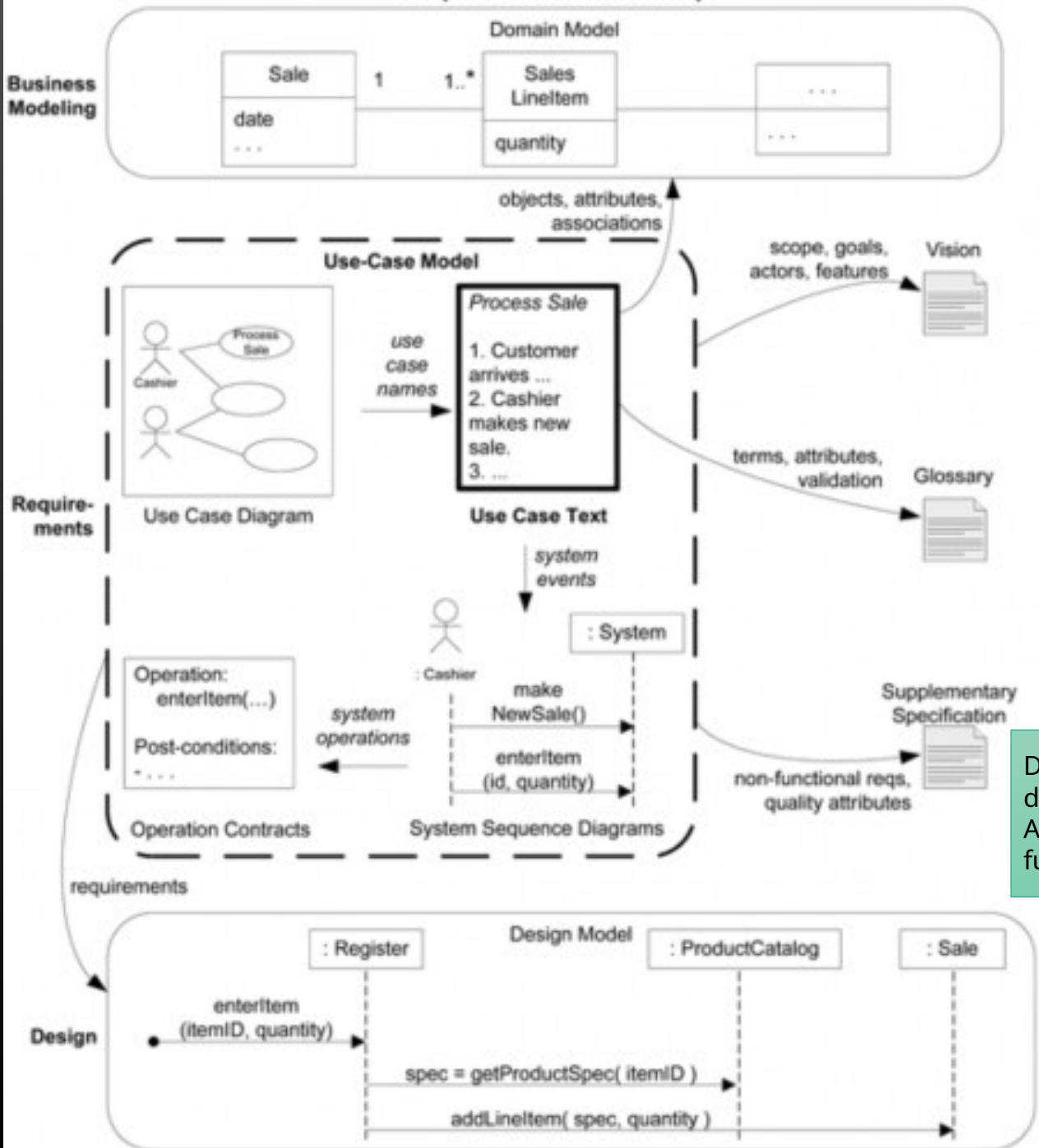
# Vista dinâmica (interações entre objetos)

Visual Paradigm Standard (ico(Universidade de Aveiro))



Qual a colaboração entre objetos necessária para implementar `Encomenda#getTotal()`?

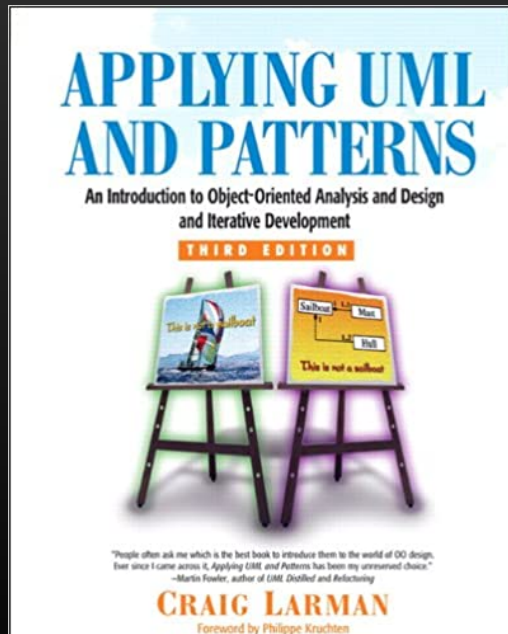
**Do código podemos ir para o modelo.  
E se começarmos a “pensar” a solução  
pelo modelo?**



Da análise para o desenho: utilização dos resultados preparados pelo Analista (modelo do domínio, descrição funcional)



In Larman:  
 Passo de transição intermédio:  
 Diagrama de Sequência de Sistema  
 (levantamento das funções “externas”  
 de entrada no Sistema, a partir do CaU)



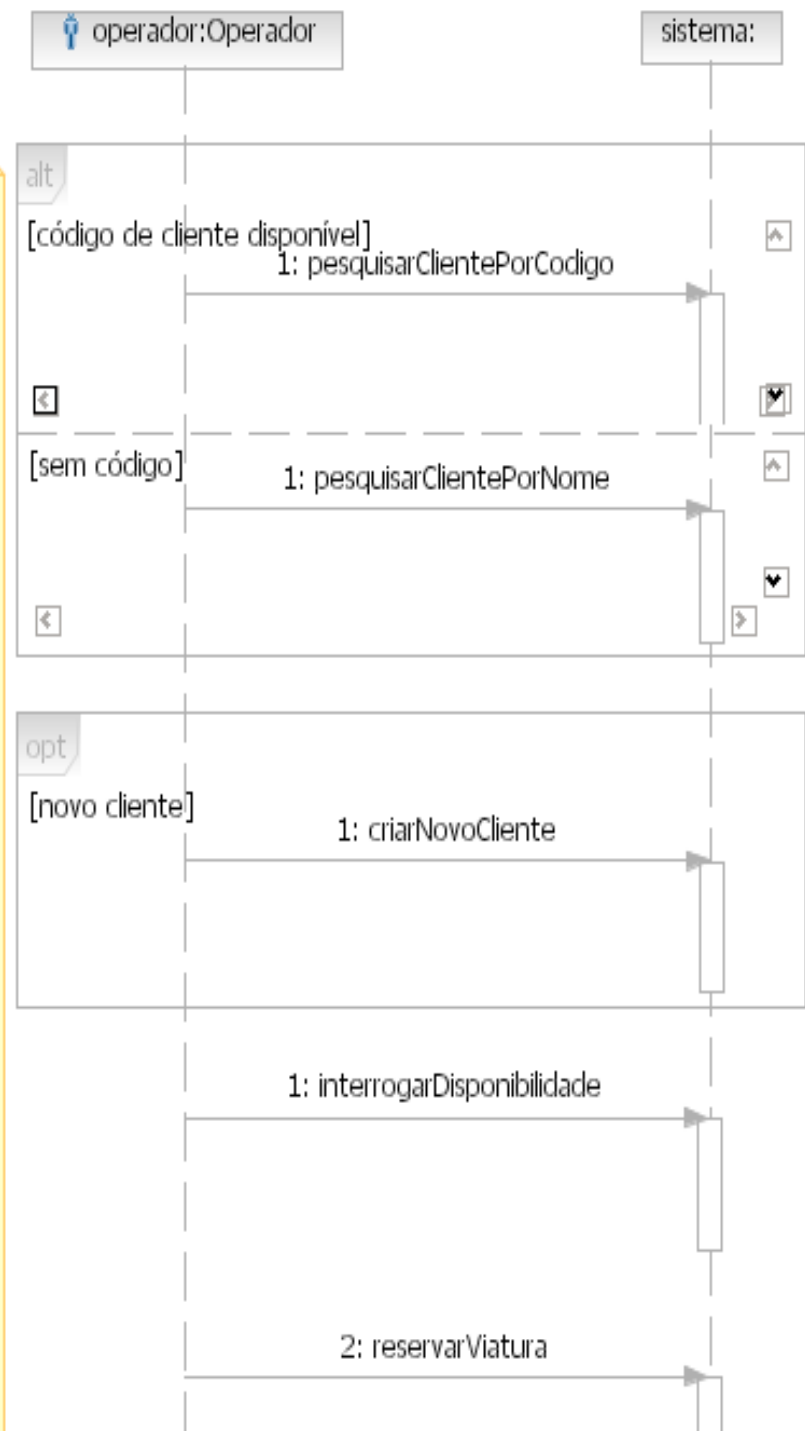
Iniciado quando um cliente telefona para o callCenter para solicitar uma reserva.

O operador pesquisa o cliente por código ou nome.

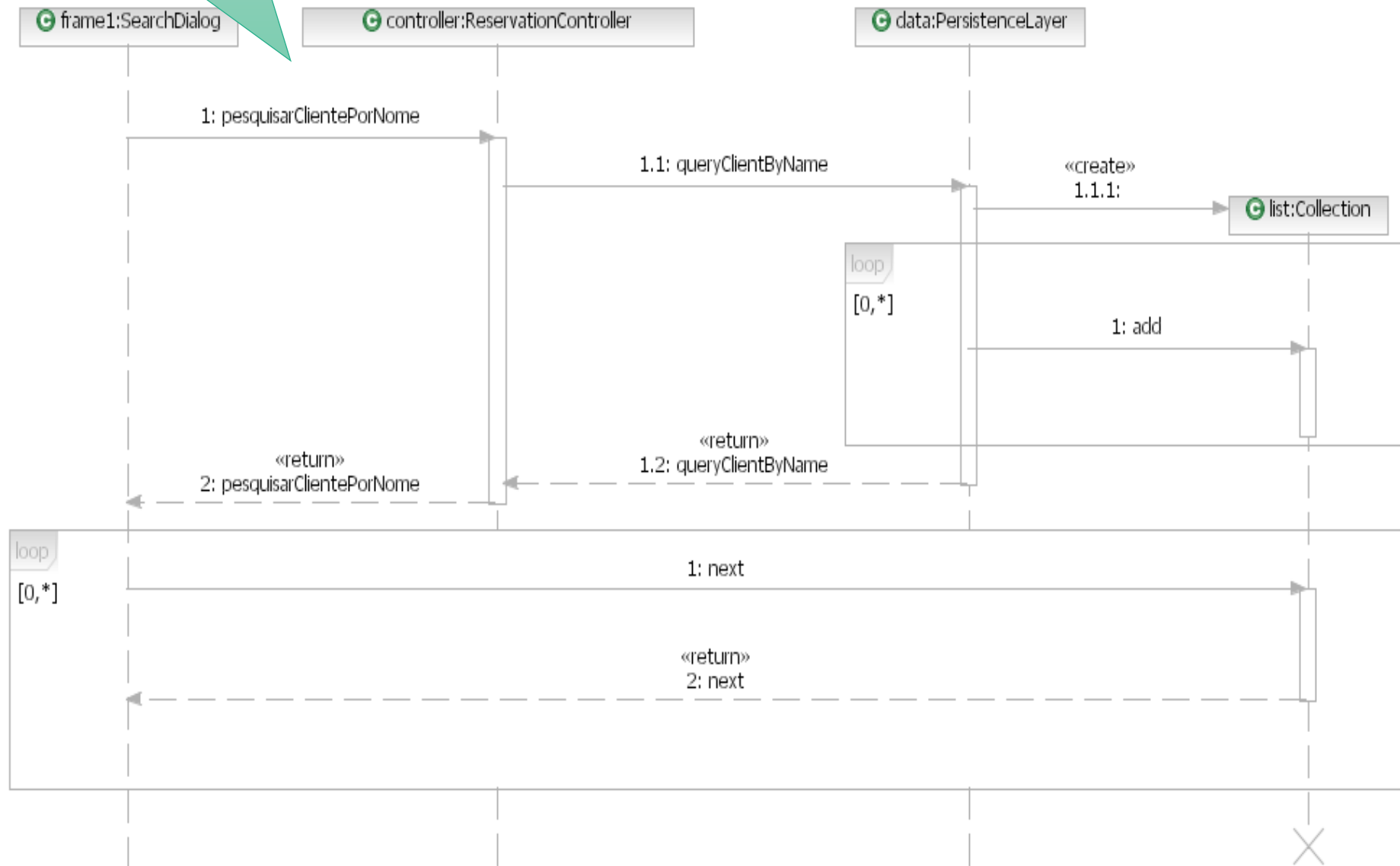
Se o cliente ainda não existe no sistema, os dados desse novo cliente são recolhidos e o cliente registado.

Os elementos da reserva são recolhidos pelo operador, que verifica se existe disponibilidade para o período pretendido. Nesse caso, a reserva é confirmada.

O cliente é informado do código de reserva (gerado pelo sistema).



Expansão de cada operação de sistema: qual a colaboração concreta de objetos que a realiza? Processo de descoberta.

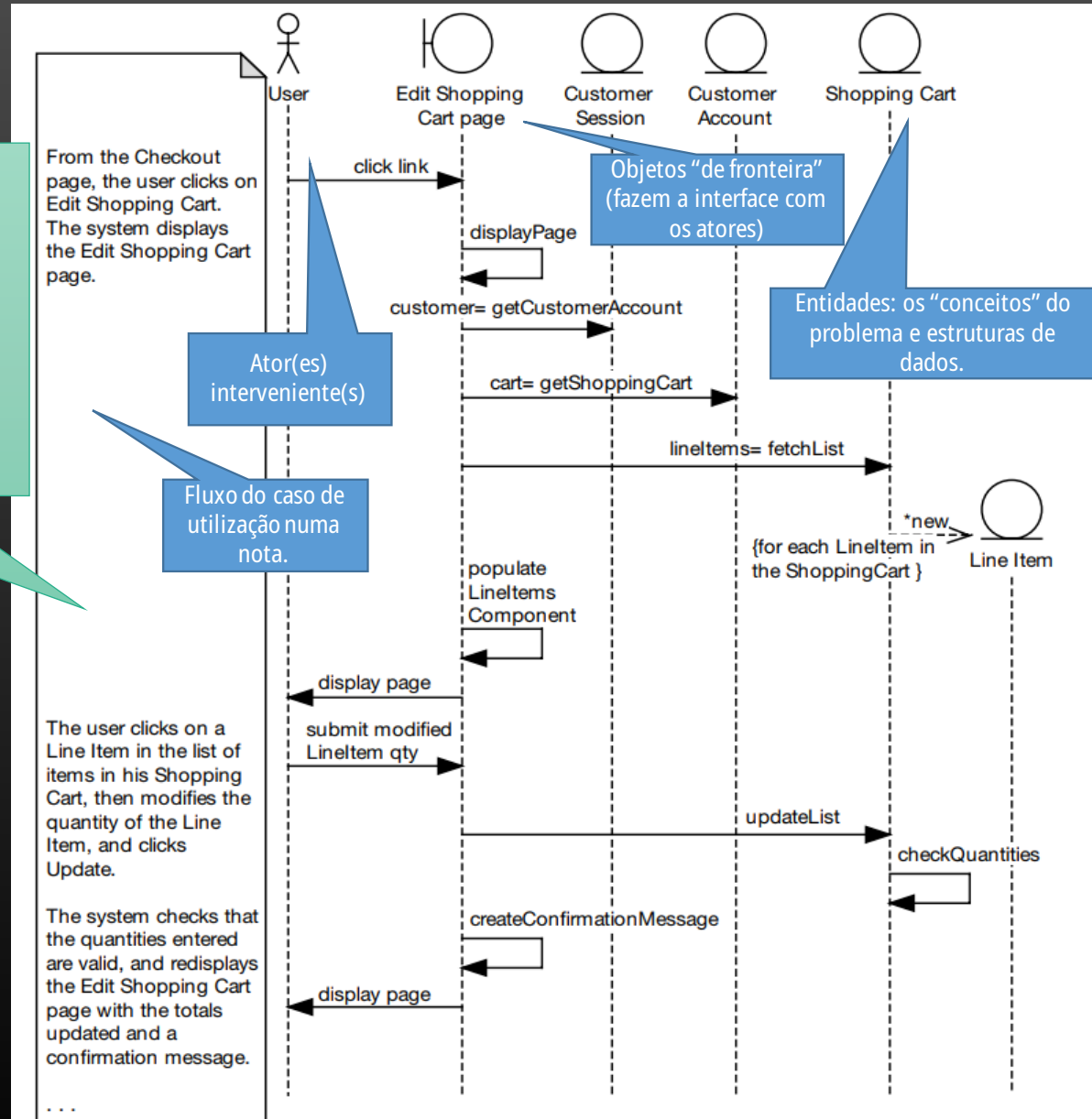
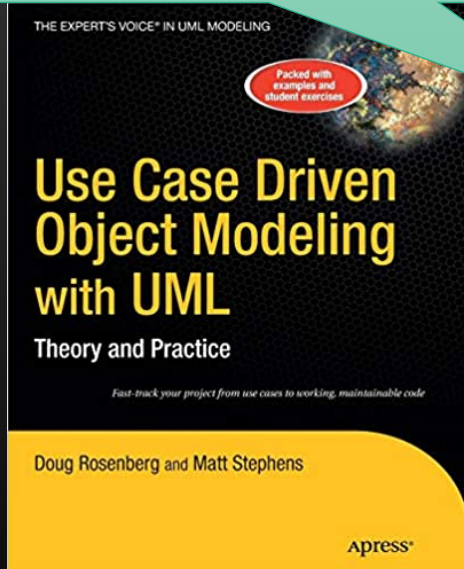


In Rosenbeg:

Da análise para o desenho: utilização dos resultados preparados pelo Analista para desenvolver o “modelo de robustez”

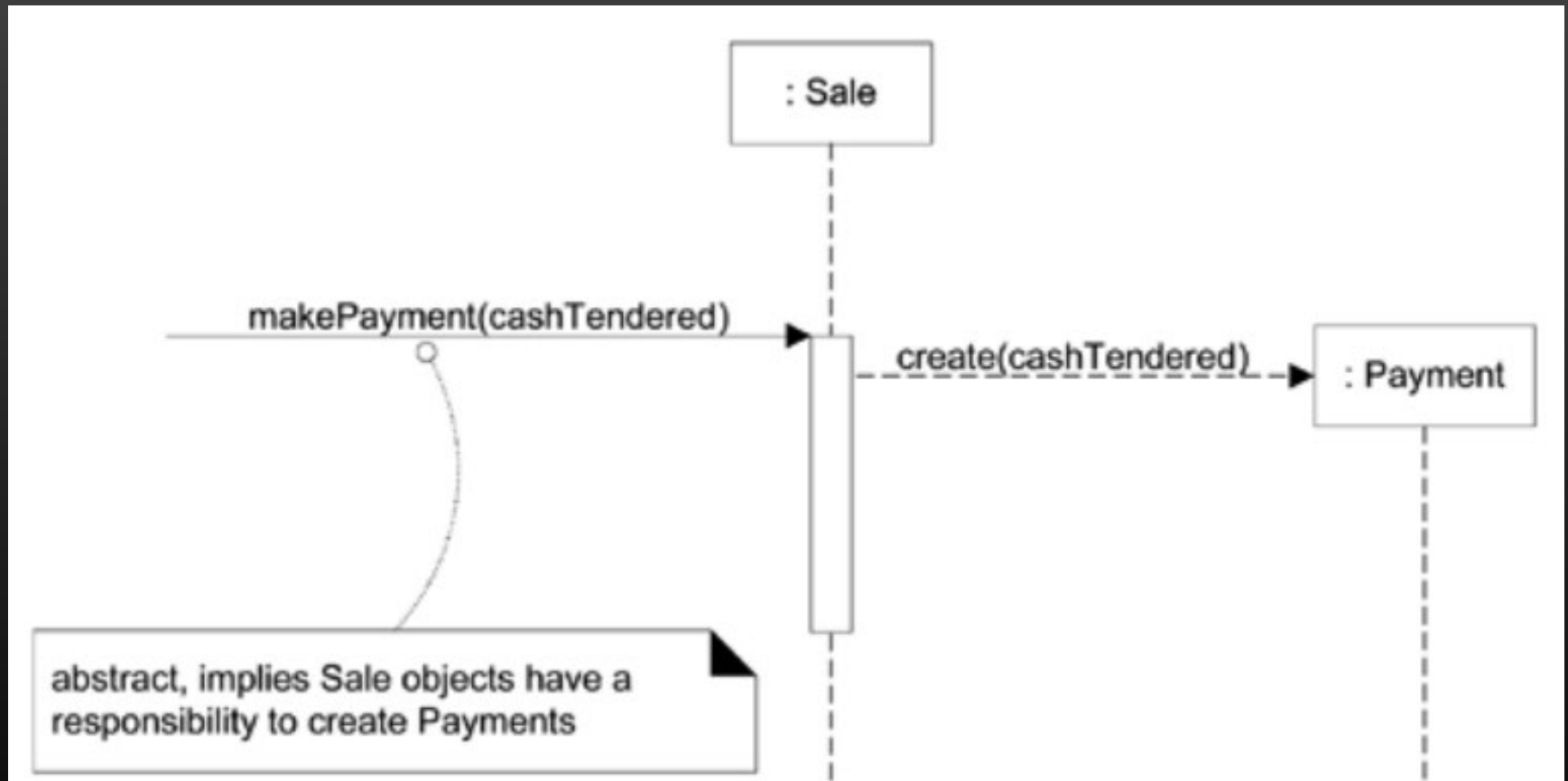
Três categorias de classes:

- Fronteiras
- Controladores
- Entidades



**“Pensar por objetos” é aplicar princípios para “distribuir” as responsabilidades pelas classes**

## Ao desenhar um diagrama de interação, estamos a atribuir responsabilidades



# Como atribuir responsabilidades aos objetos?

Não é uma ciência exata


Por isso temos...

Bom e mau desenho

Desenho eficiente e ineficiente

Desenho elegante e tenebroso...

**Implicações na facilidade de  
manter e evoluir uma solução**



**“Desenho”**, no ciclo de engenharia do software, significa o processo de planejar/idealizar o código. A pessoa que lidera o desenho é o “arquiteto de software”.

Sempre que, mesmo num problema simples, começamos por nos interrogar: quais as classes? Como é que elas vão estar interdependentes?, estamos a “desenhar” o o código (fazendo escolhas).

# Responsabilidades de um objeto

## Fazer

Fazer alguma coisa sobre o seu estado, como calcular alguma coisa, criar objetos,...

Iniciar uma ação em outros objetos

Coordenar/controlar as ações em outros objetos

## Saber

Conhecer o seu estado interno (“escondido”)

Conhecer os objetos relacionados

# CrITÉRIOS para o desenho

- Um conjunto de métricas para avaliar o desenho
- Acoplamento (*coupling*): refere-se ao grau de proximidade/interdependência da relação entre classes
- Coesão (*coesion*): refere-se ao grau com que os atributos e métodos de uma classe estão relacionados internamente.

Uma classe que tem muitos atributos que são objetos de outro tipo, tem um *coupling* elevado: **depende de** outras classes.

Uma classe que mantém, internamente, detalhes das Vendas e dos Produtos vendidos, não é coesa: em vez de ter um **foco único**, está a assumir várias responsabilidades.



## Coupling

Mede a força/intensidade da dependência de uma classe de outras

A classe C1 está emparelhada com C2 se precisa de C2, direta ou indiretamente.

Uma classe que depende de outras 2 tem um “coupling” mais baixo que uma que dependa de 8.

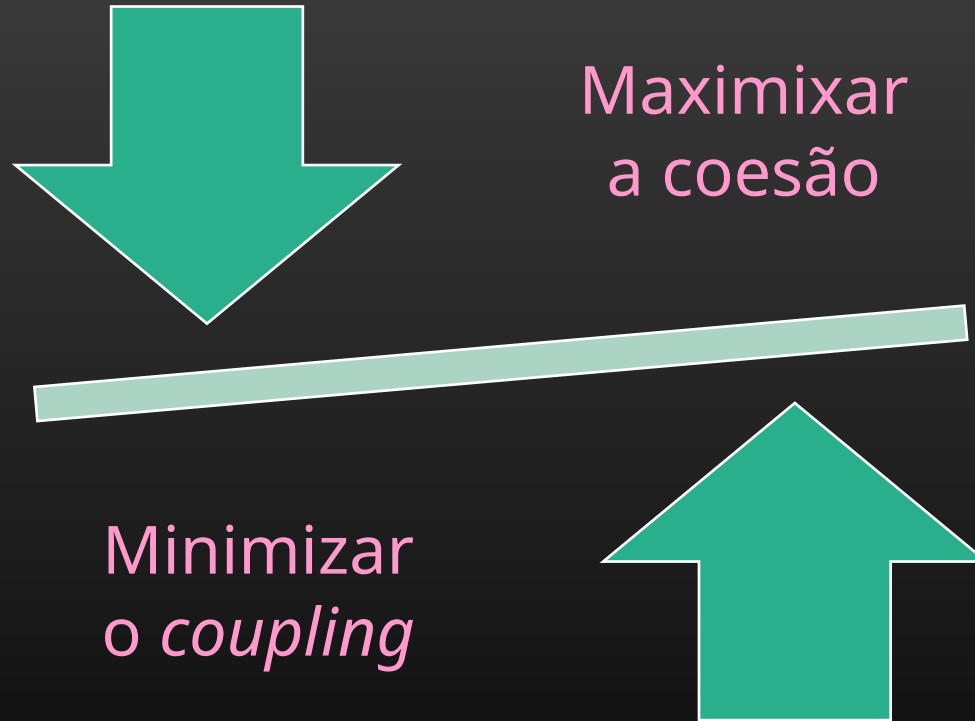
## Coesão

Mede a força/intensidade do relacionamento dos elementos de uma classe entre si.

Todas as operações e dados de uma classe devem estar natural e diretamente relacionados com o conceito que a classe modela

Uma classe deve ter um foco único (vs. responsabilidades desgarradas)

# Critérios gerais para um melhor desenho



# Common Forms of Coupling in Java

- Type X has an attribute that refers to a type Y instance or type Y itself

```
class X{ private Y y = ...}  
class X{ private Object o = new Y(); }
```

- A type X object calls methods of a type Y object

```
class Y{f(){;}}  
class X{ X(){new Y.f();}}
```

- Type X has a method that references an instance of type Y (E.g. by means of a parameter, local variable, return type,...)

```
class Y{}  
class X{ X(Y Y){...}}  
class X{ Y f(){...}}  
class X{ void f(){Object y = new Y();}}
```

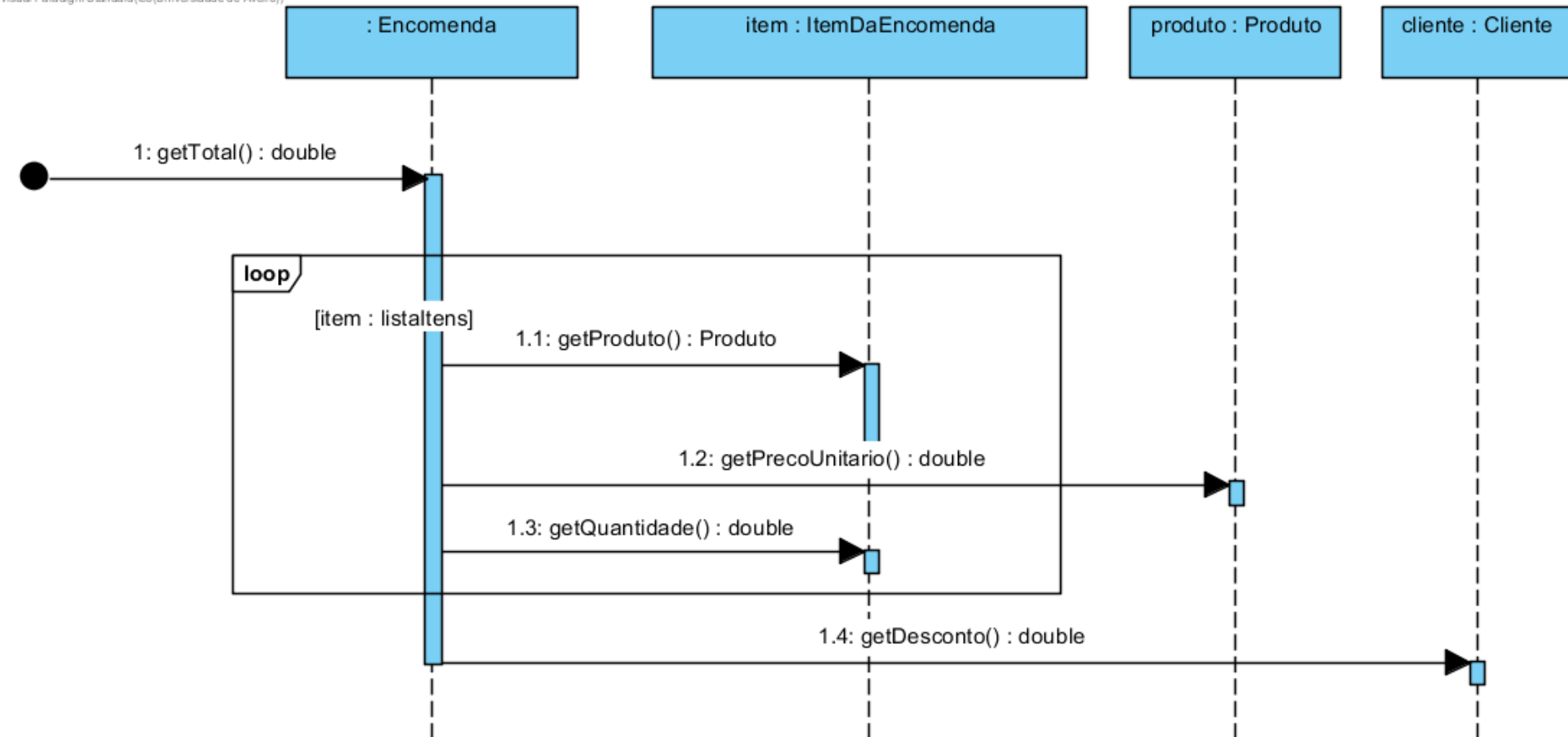
- Type X is a subtype of type Y

```
class Y{}  
class X extends Y{}
```

- ...

# Coupling de interação

Visual Paradigm Standard (co(Universidade de Aveiro))



# Coesão

Qual é a hipótese que oferece maior coesão?

Qual é o que é mais fácil de avariar/dar problemas?

De que é que precisamos 80% das vezes?....



# Coesão

Uma classe, objeto ou método coesos têm um único “foco”

## Coesão a nível dos métodos

O método executa mais do que um propósito/operação?

Realizar mais do que uma operação é mais difícil de entender e implementar

## Coesão a nível da classe

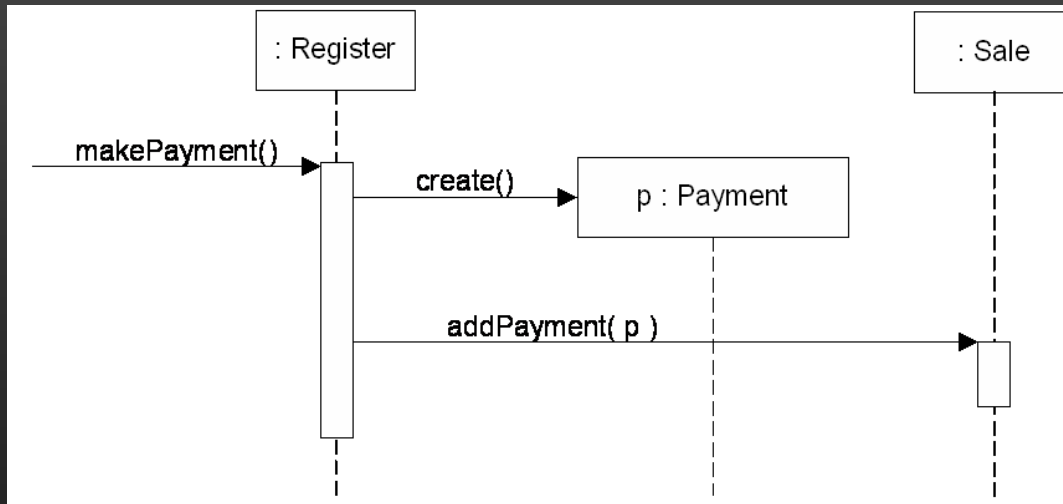
Os atributos e métodos representam um único objeto?

As classes não devem misturar papéis, domínios ou objetos

## Coesão na especialização/generalização

As classes numa hierarquia devem mostrar uma relação “tipo-de”

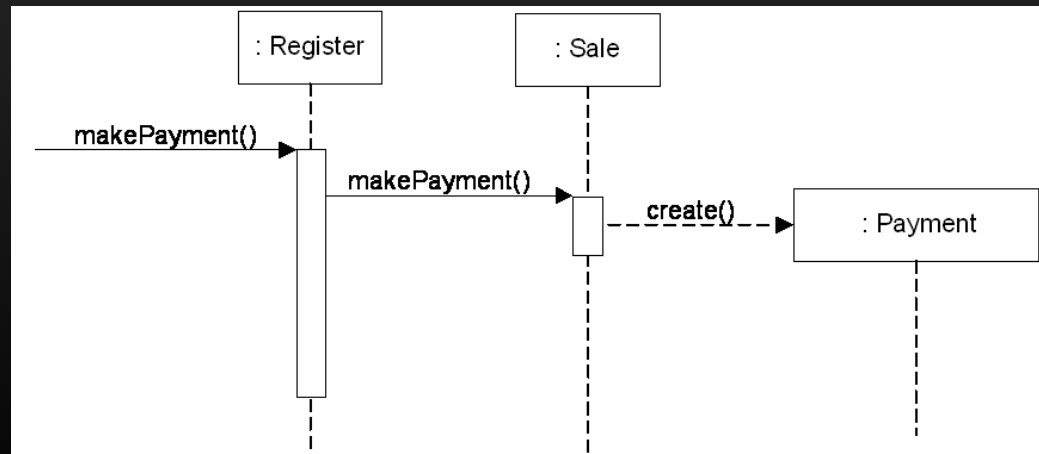
# Exemplos



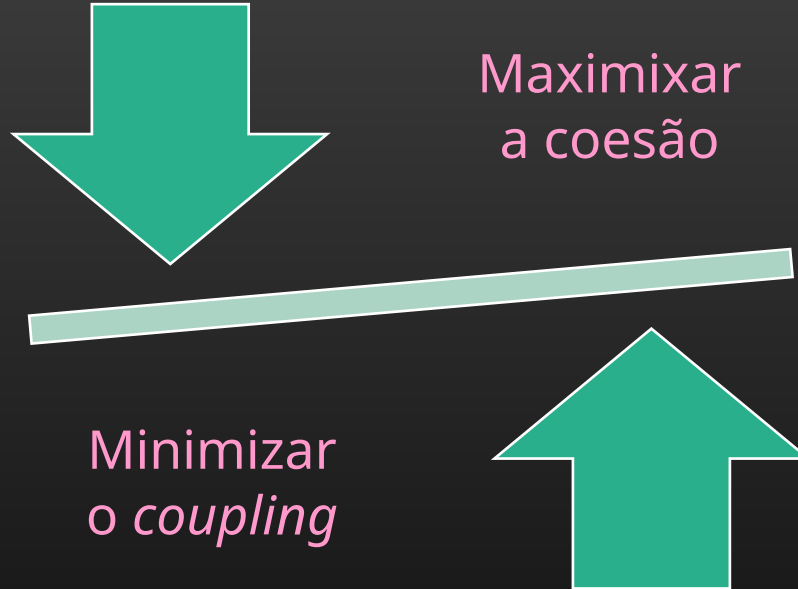
Qual é a hipótese que oferece maior coesão?

No primeiro caso, Register conhece informação de pagamentos e de vendas.

No segundo caso, Register apenas se relaciona com Venda (e não precisa de representar a lógica dos pagamentos)



# É preciso balancear



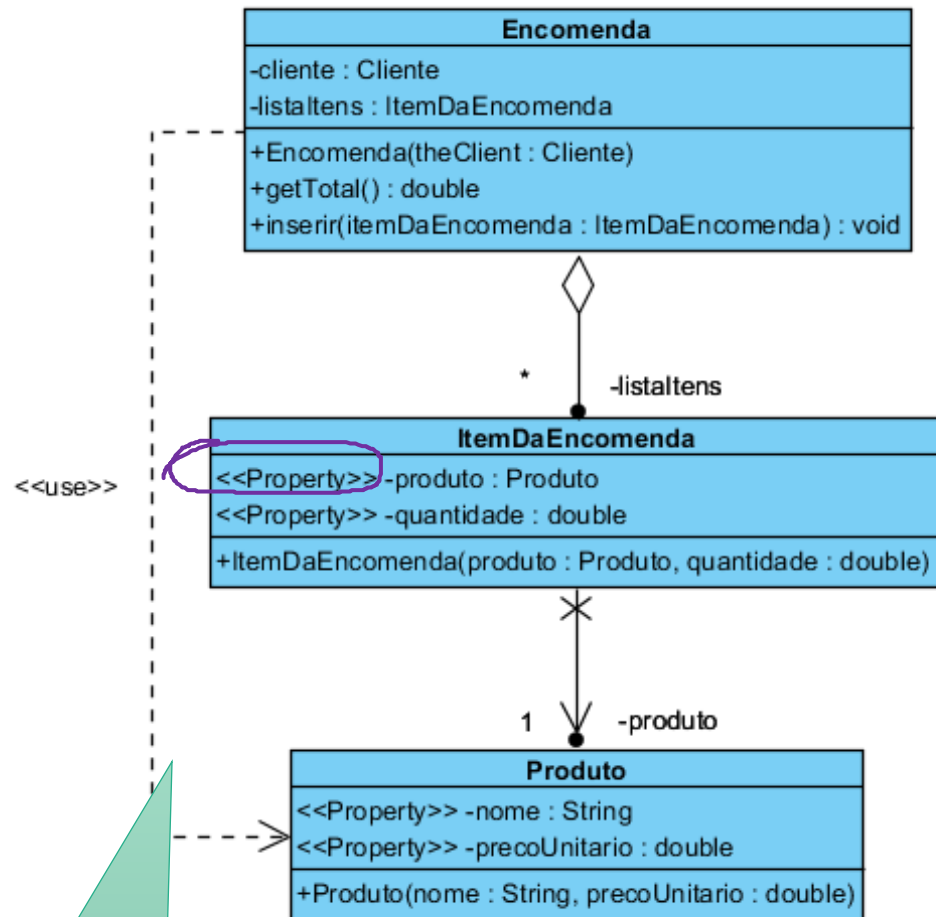
Por hipótese, a situação com melhor *coupling* (mais baixo possível) seria ter uma única classe na solução.

Mas essa seria a pior escolha do ponto de vista da coesão.



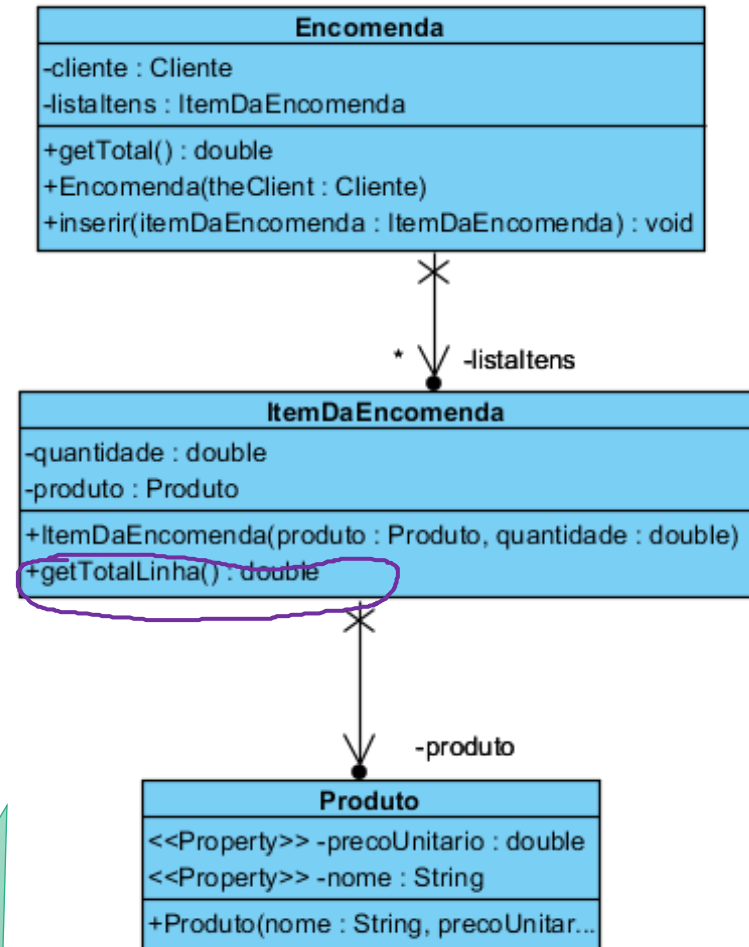
# Avaliação de coupling/coesão: exemplo da encomenda

Visual Paradigm Standard (© Universidade de Aveiro)



getTotal() consulta o preço unitário definido em Produto

Oliveira



getTotal() pede ao "item da encomenda" para lhe dar o total da linha.

# GRASP (Larman)

## *Generic Responsibility Assignment Principles*

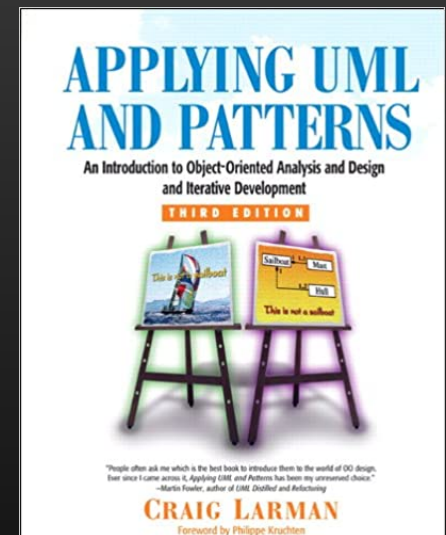
↓ *Coupling*

↑ *Cohesion*

*Information Expert*

*Creator*

*Controller*



Existem recomendações/princípio para orientar a distribuição de responsabilidades pelos objetos. E.g.: GRASP

# Referências

Core readings	Suggested readings
<ul style="list-style-type: none"><li>• [Dennis15] – Chap. 8</li></ul>	<ul style="list-style-type: none"><li>• [Larman04] – Chap. 17 and 18</li><li>• Slides by M. Eichberg : <a href="#">SSD</a> and <a href="#">OO-Design</a></li></ul>