41951- ANÁLISE DE SISTEMAS

Visualização e desenho de Código (por objetos)

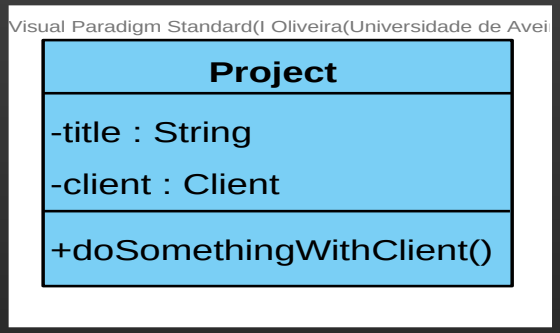
Ilídio Oliveira

v2024-03-05

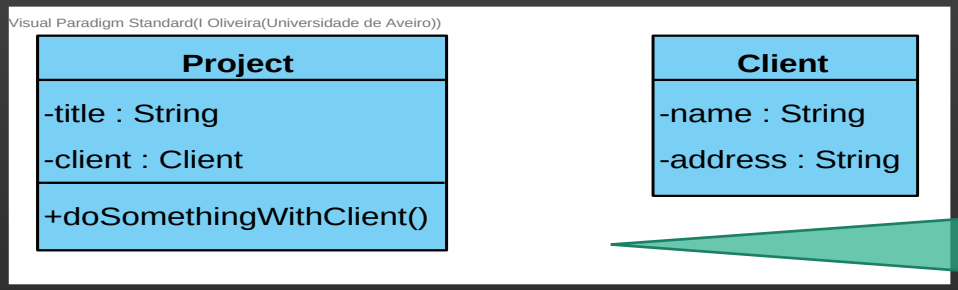
universidade de aveiro
departamento de eletrónica, telecomunicações e informática
deti

Visualização de código Java com classes

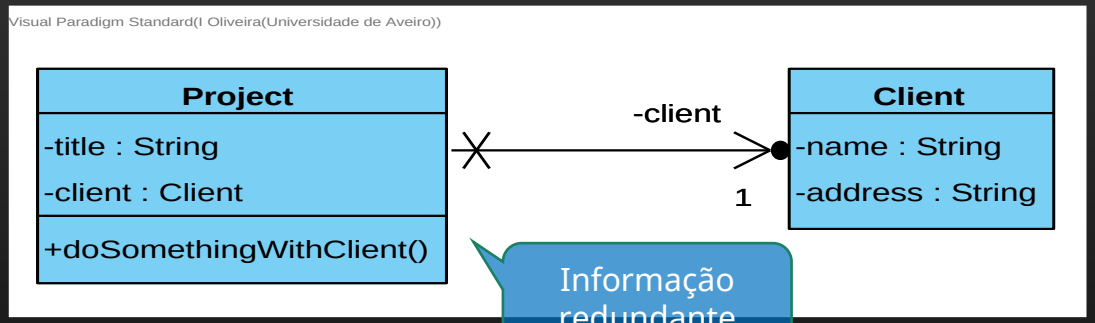
```java
public class Project {

    private String title;
    private Client client;

    public void doSomethingWithClient() {
        // todo
    }

}
```



Visualização do código com classes

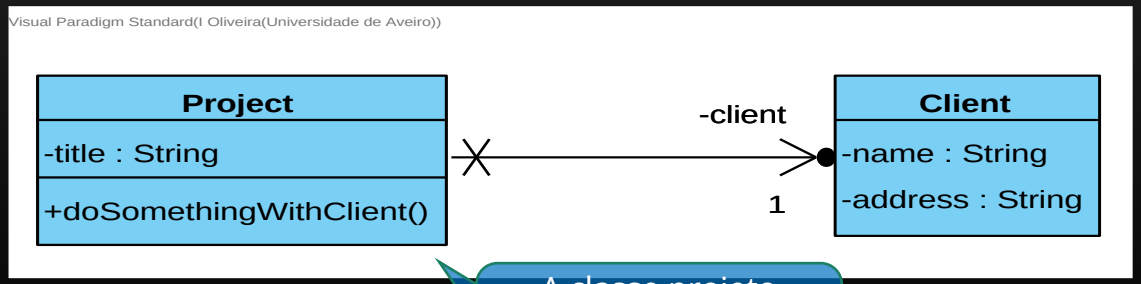


Cada objeto da classe Projeto guarda informação sobre o respetivo Cliente, ou seja, referencia outro objeto.



Informação redundante

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



A classe projeto define dois atributos: *title* e *client*.

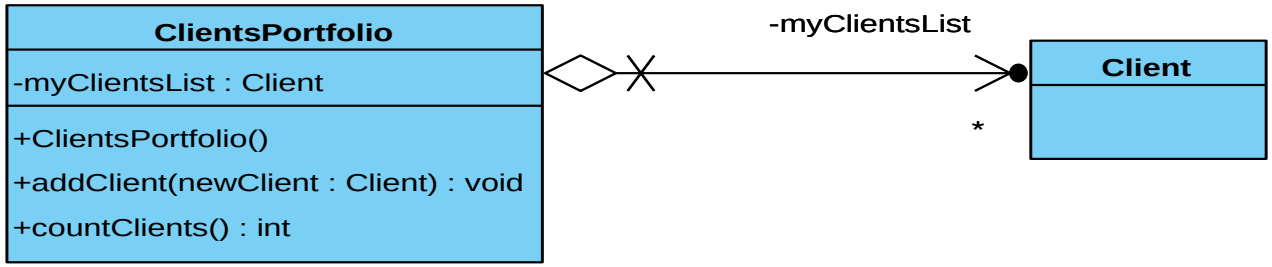
```java
public class ClientsPortfolio {

    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)

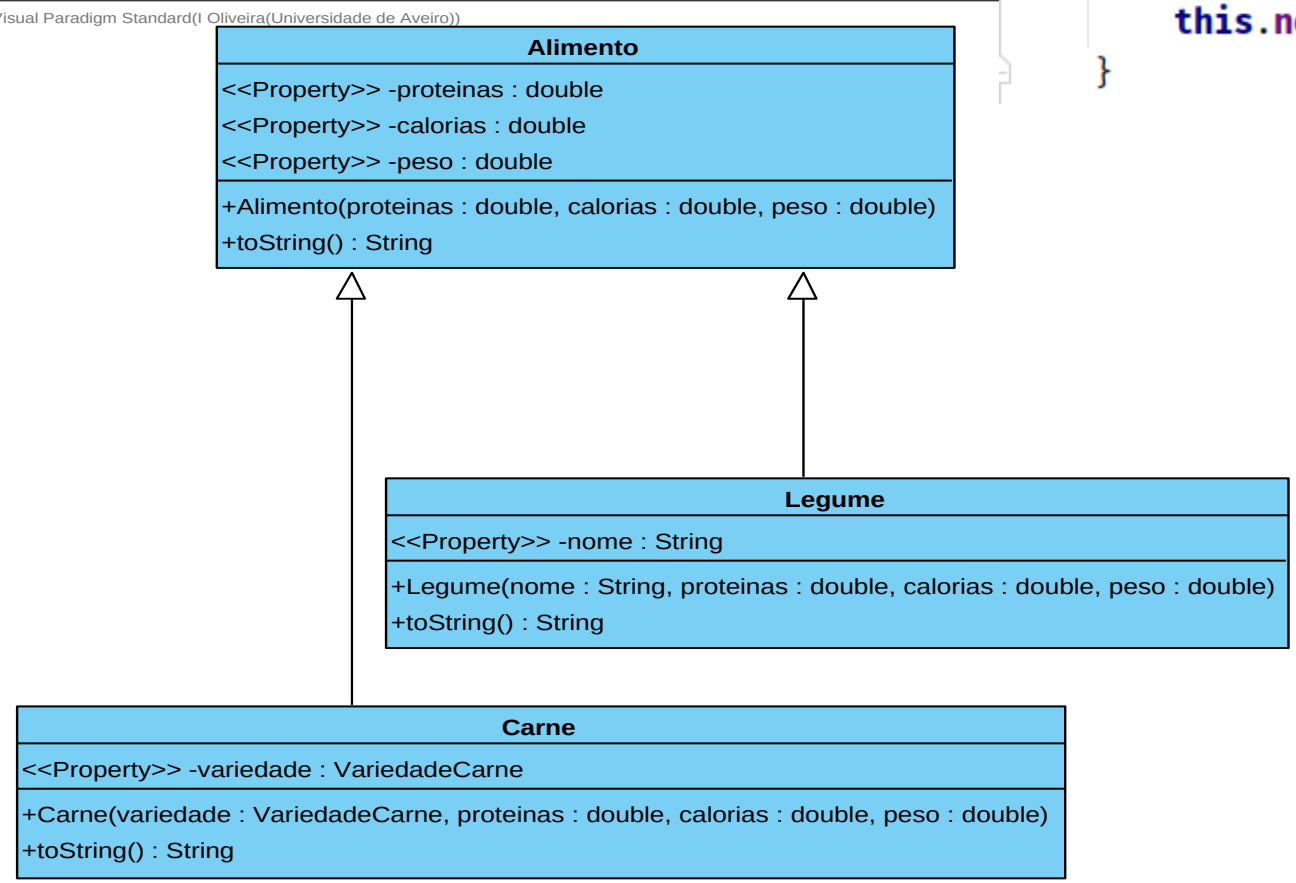
Operação especial: usado na inicialização de cada instância da classe (método Construtor)

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

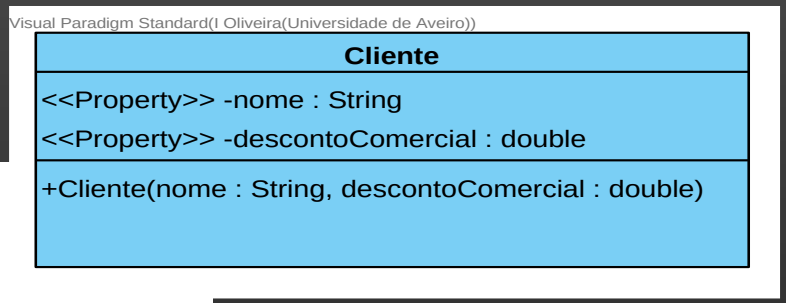


Generalização

```java
public class Legume extends Alimento {
    private String nome;

    public Legume(String nome, double protein
        super(proteinas, calorias, peso);
        this.nome = nome;
    }
```



O esteriótipo “property”




```java
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”:

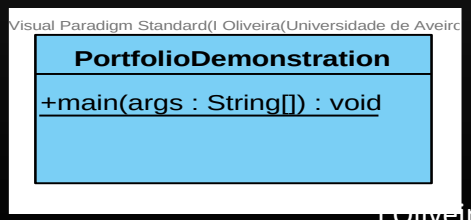
- Atributo *abc*
- *getAbc()*
- *setAbc()*

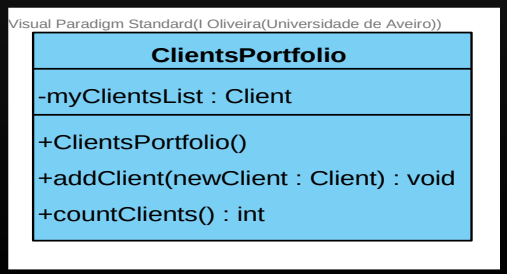
Podemos associar o esterótipo “property” e omitir os getters e setters

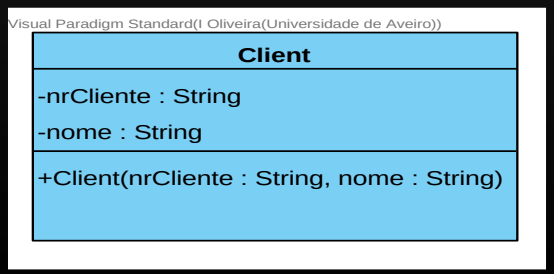
Objetos enviam mensagens

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

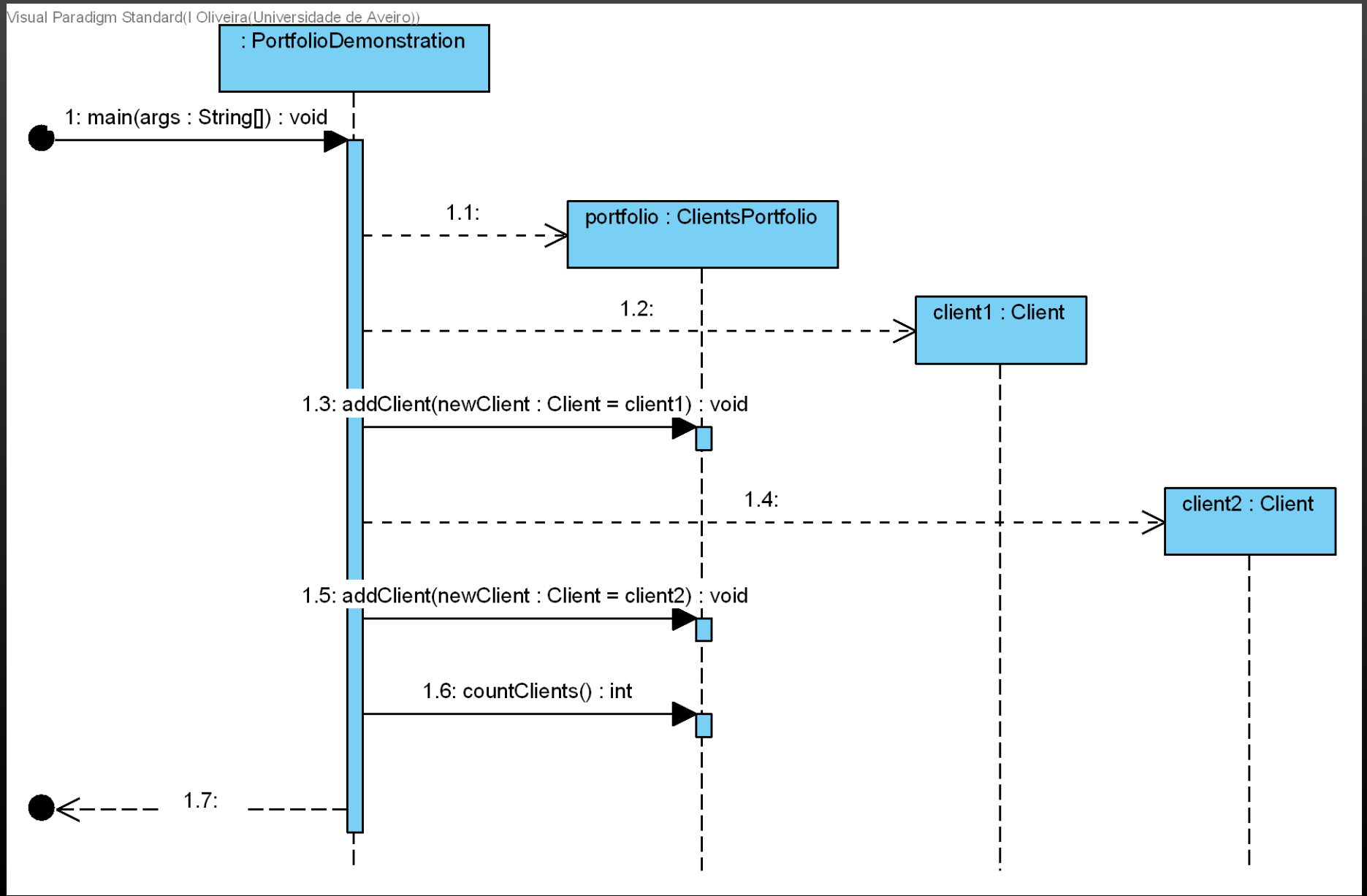
```
public class PortfolioDemonstration {
    public static void main(String[] args) {
        // obter um novo objeto da classe ClientsPortfolio
        ClientsPortfolio portfolio = new ClientsPortfolio();

        // obter um novo objeto da classe Cliente e adicioná-lo ao porfolio
        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() );
    }
}
```



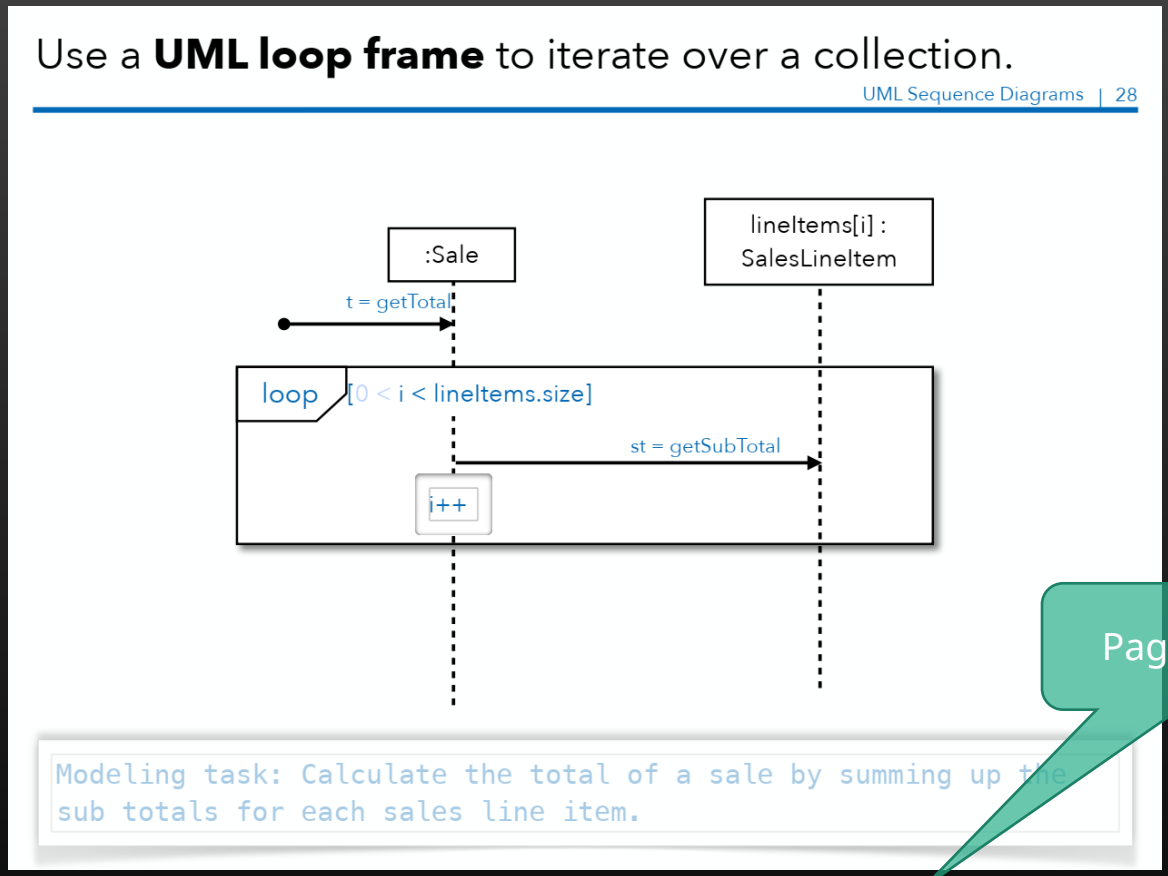




…que podem ser vistas num modelo dinâmico



Alguns exemplos adicionais



https://stg-tud.github.io/eise/WS18-SE-08-Modeling-dynamic_Part.pdf

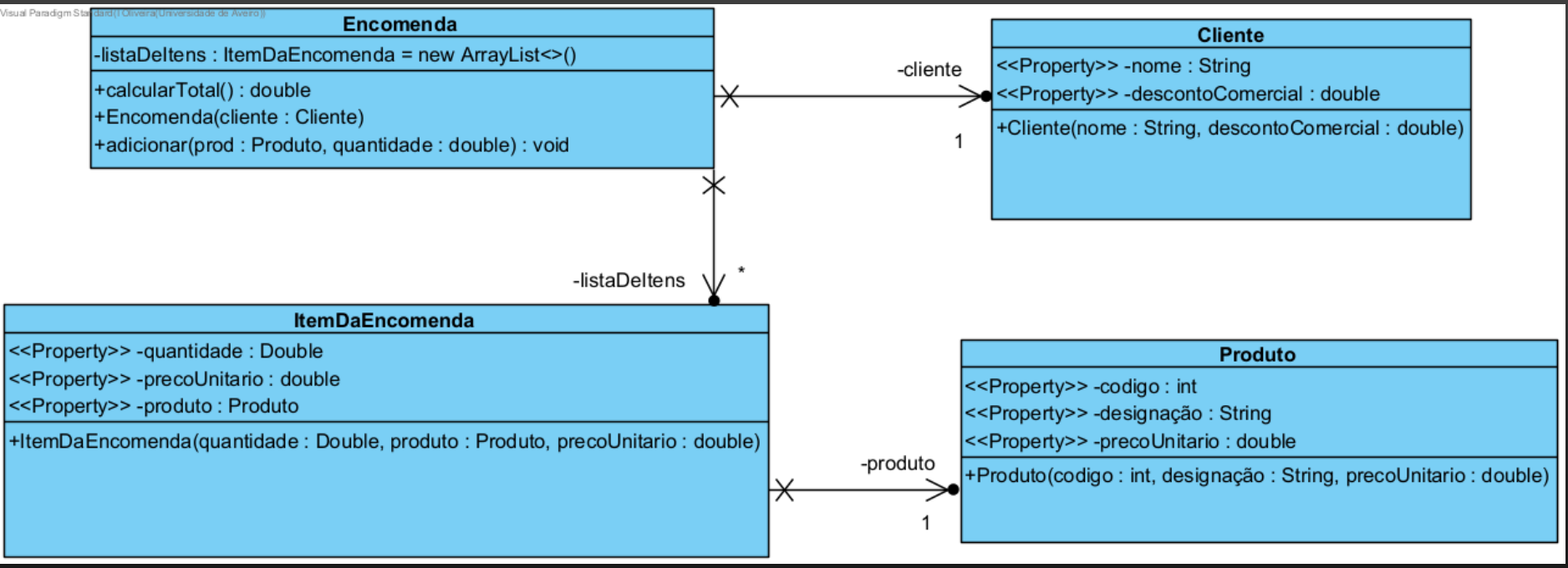
UML para “visualizar” o código: estrutura e interação

O 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<ItemDaEncom
    }
```

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

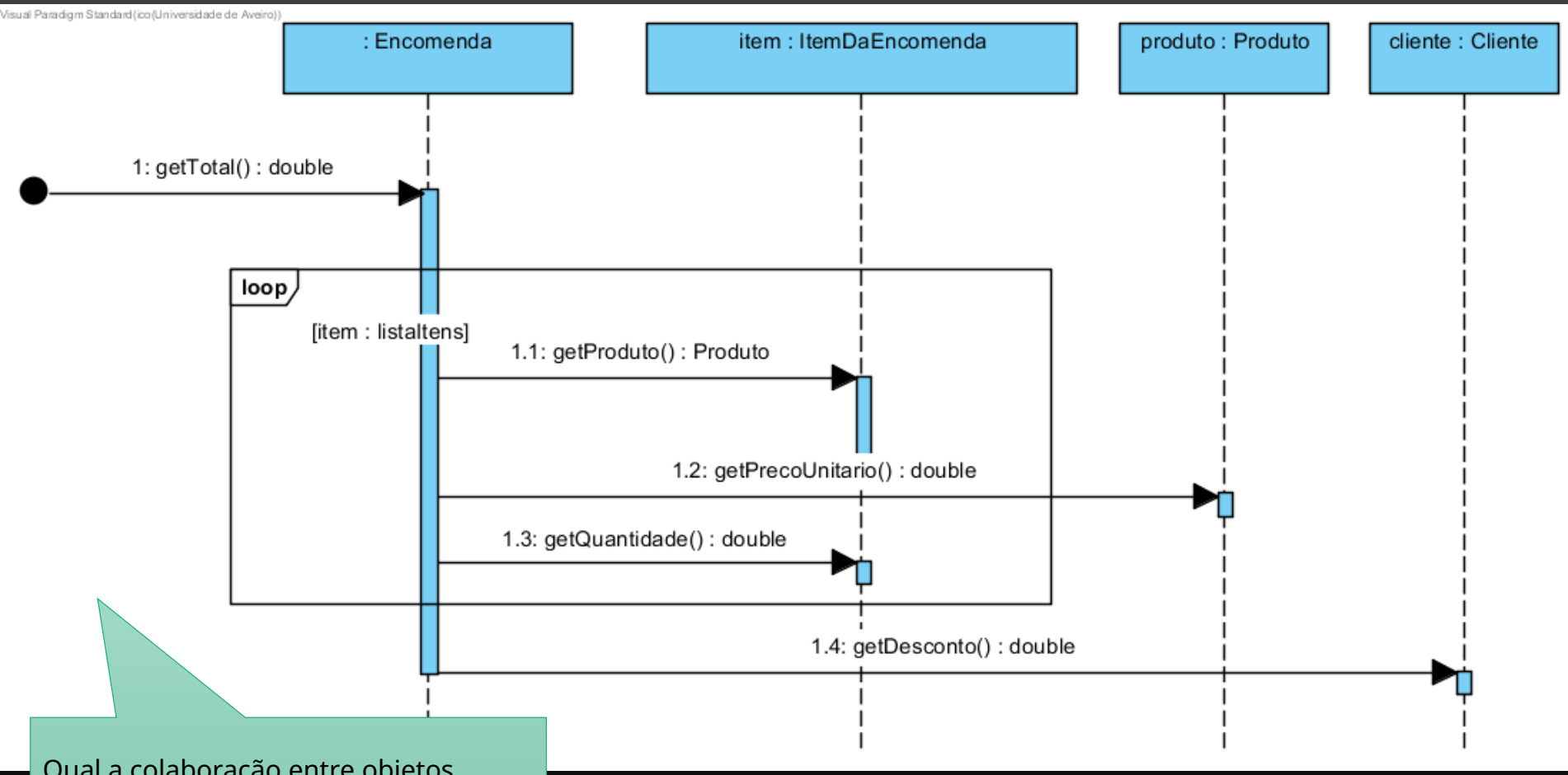
Vista estrutural (definição das classes)



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

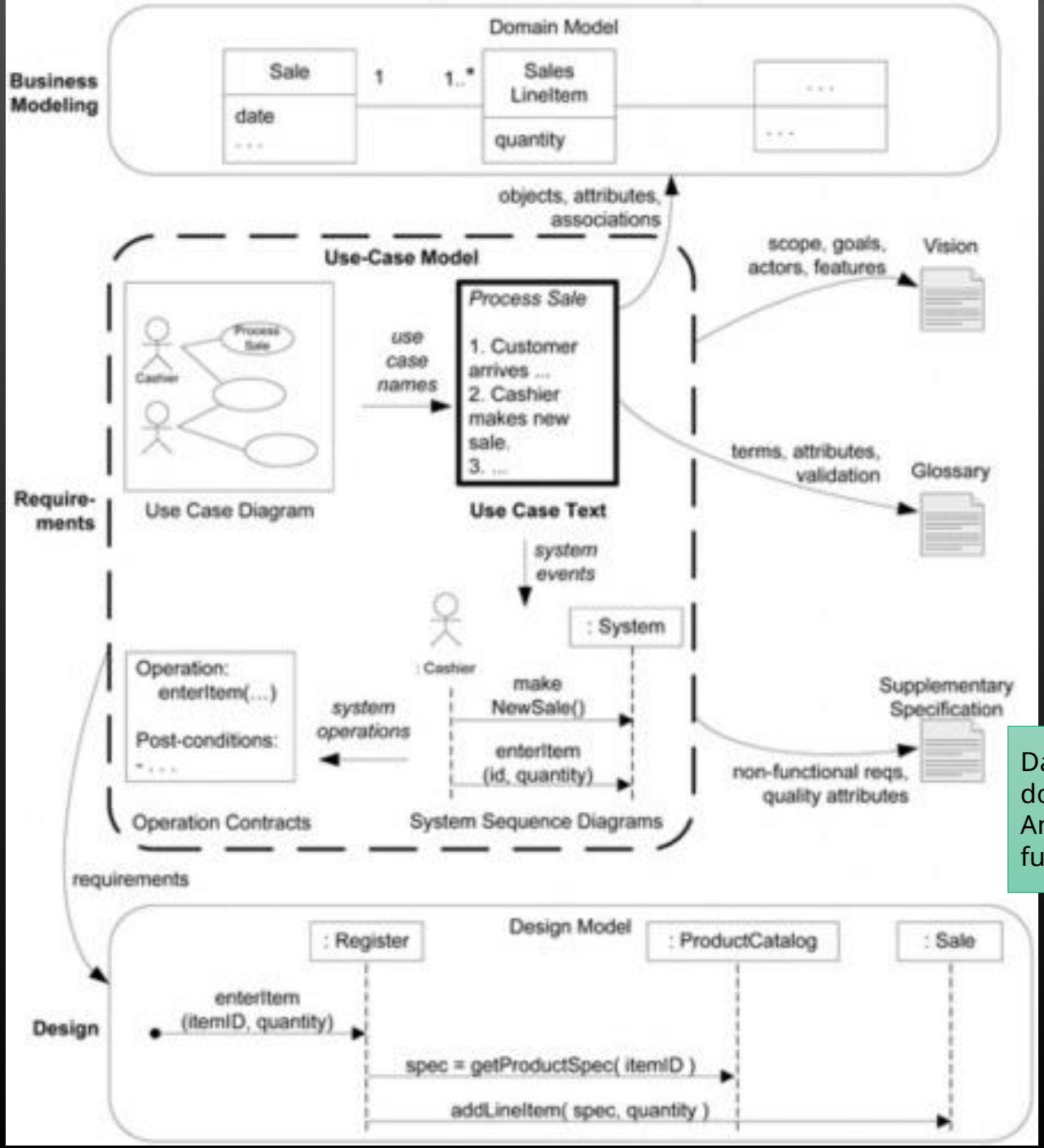
O esteriótipo <<Property>> marca atributos que têm *getter* e *setter*

Vista dinâmica (interações entre objetos)



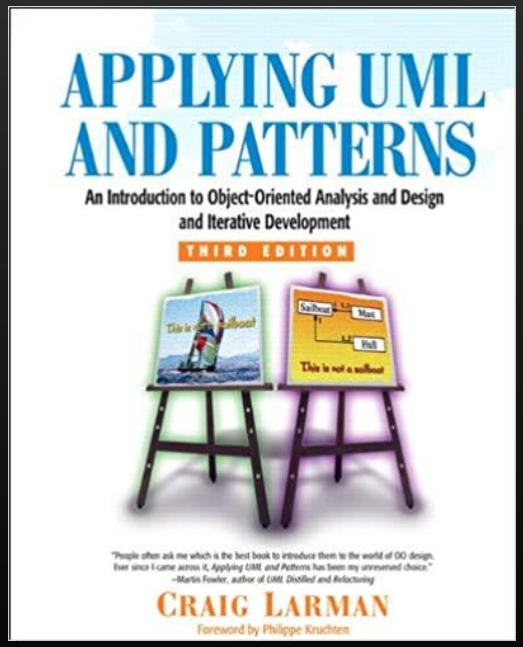
Qual a colaboração entre objetos necessária para implementar Encomenda#getTotal()?

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 functional)

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



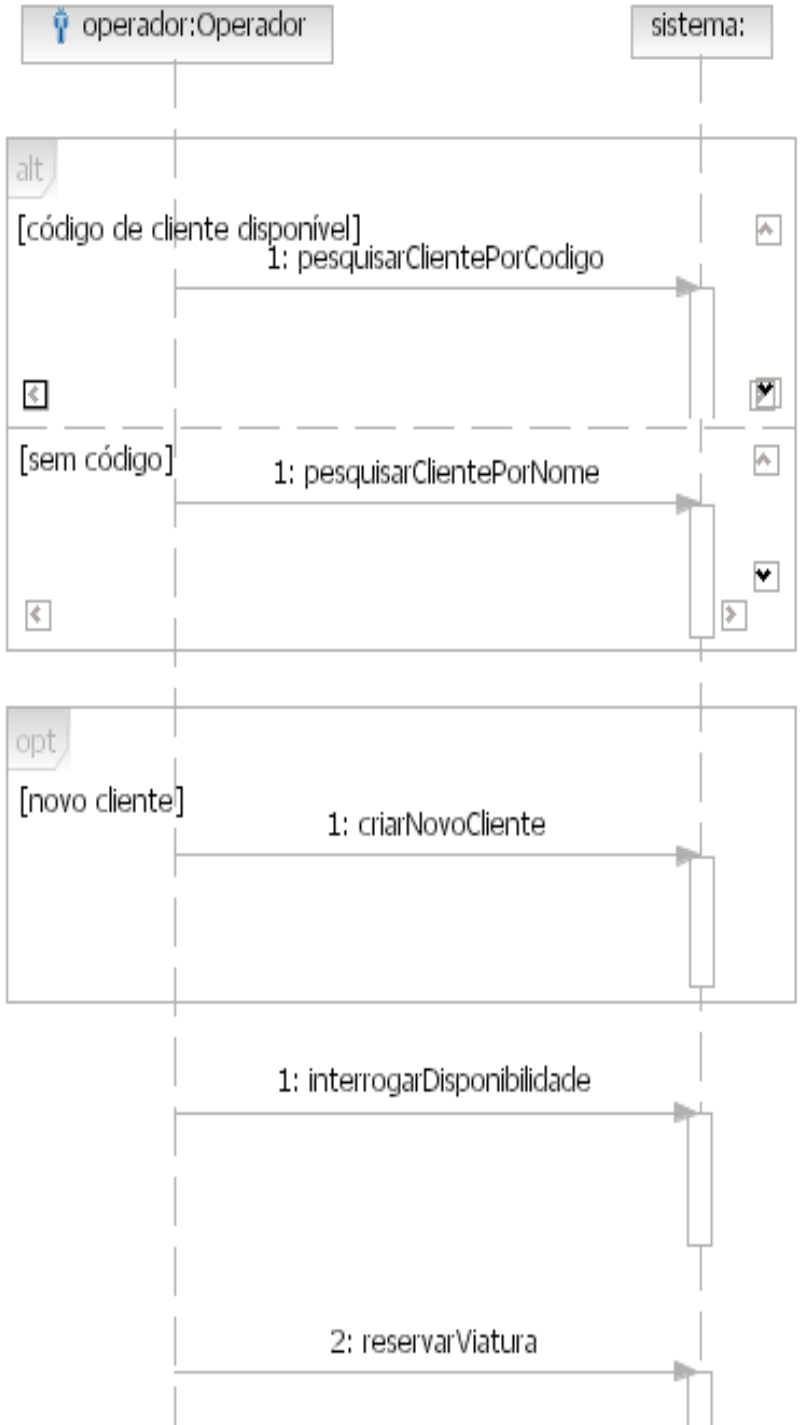
Iniciado quando um cliente telefona para o callCenter para solicitar uma reserva.

O operador pesquisa o cliente por código ou nome.

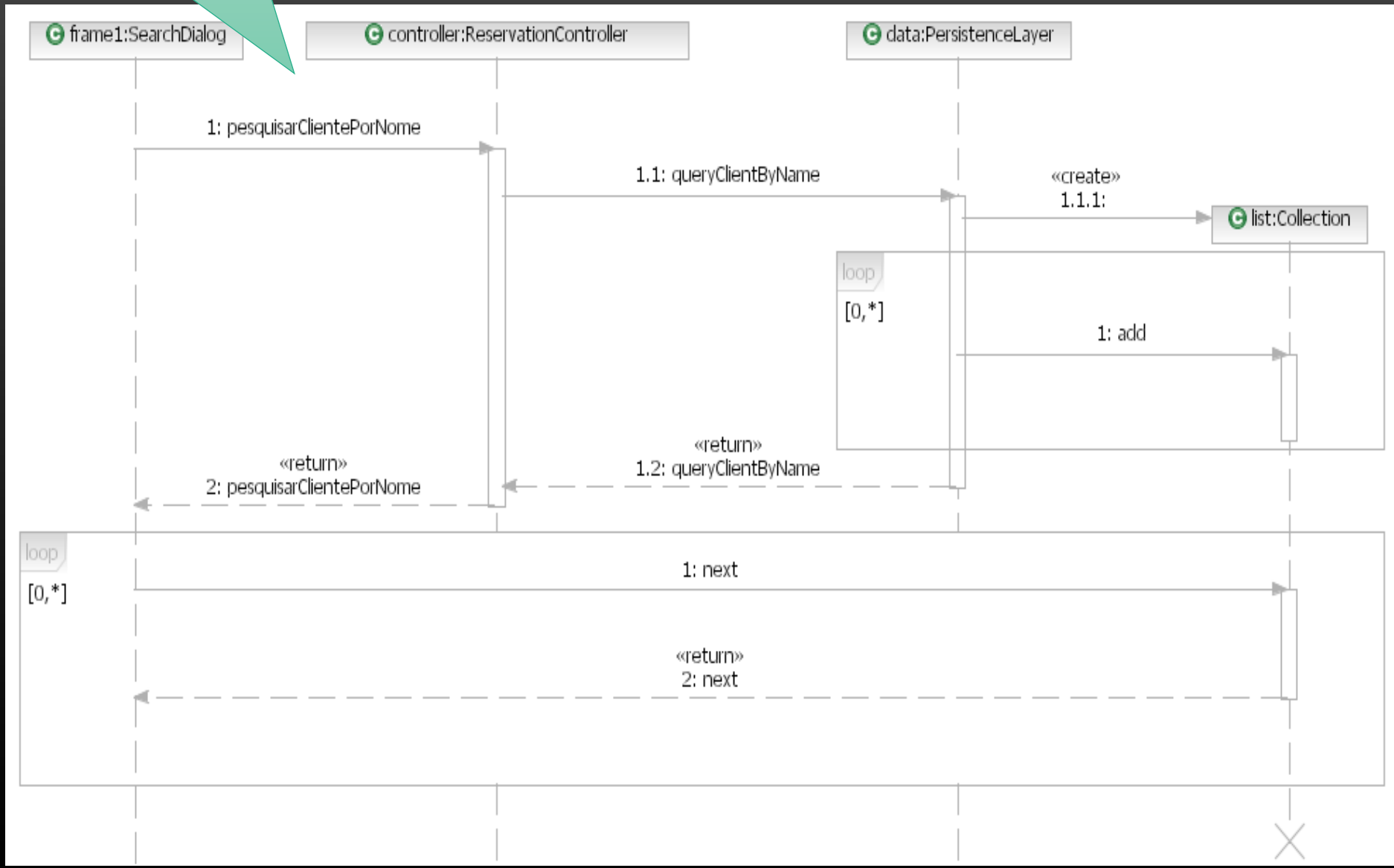
Se o cliente ainda não existe no sistema, os dados desse novo cliente são recolhidos e o cliente registado.

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

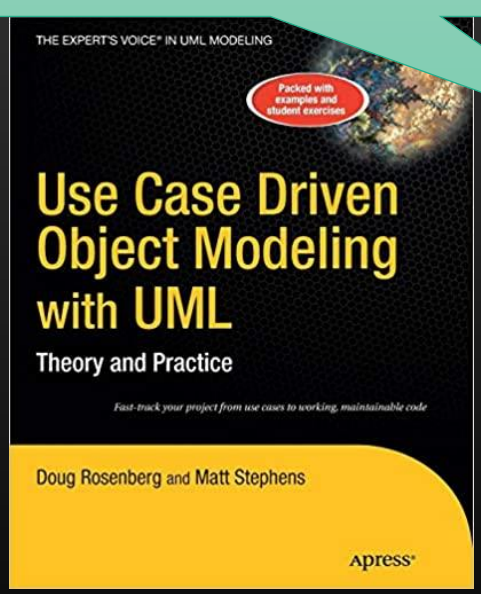
O cliente é informado do código de reserva (gerado pelo sistema).

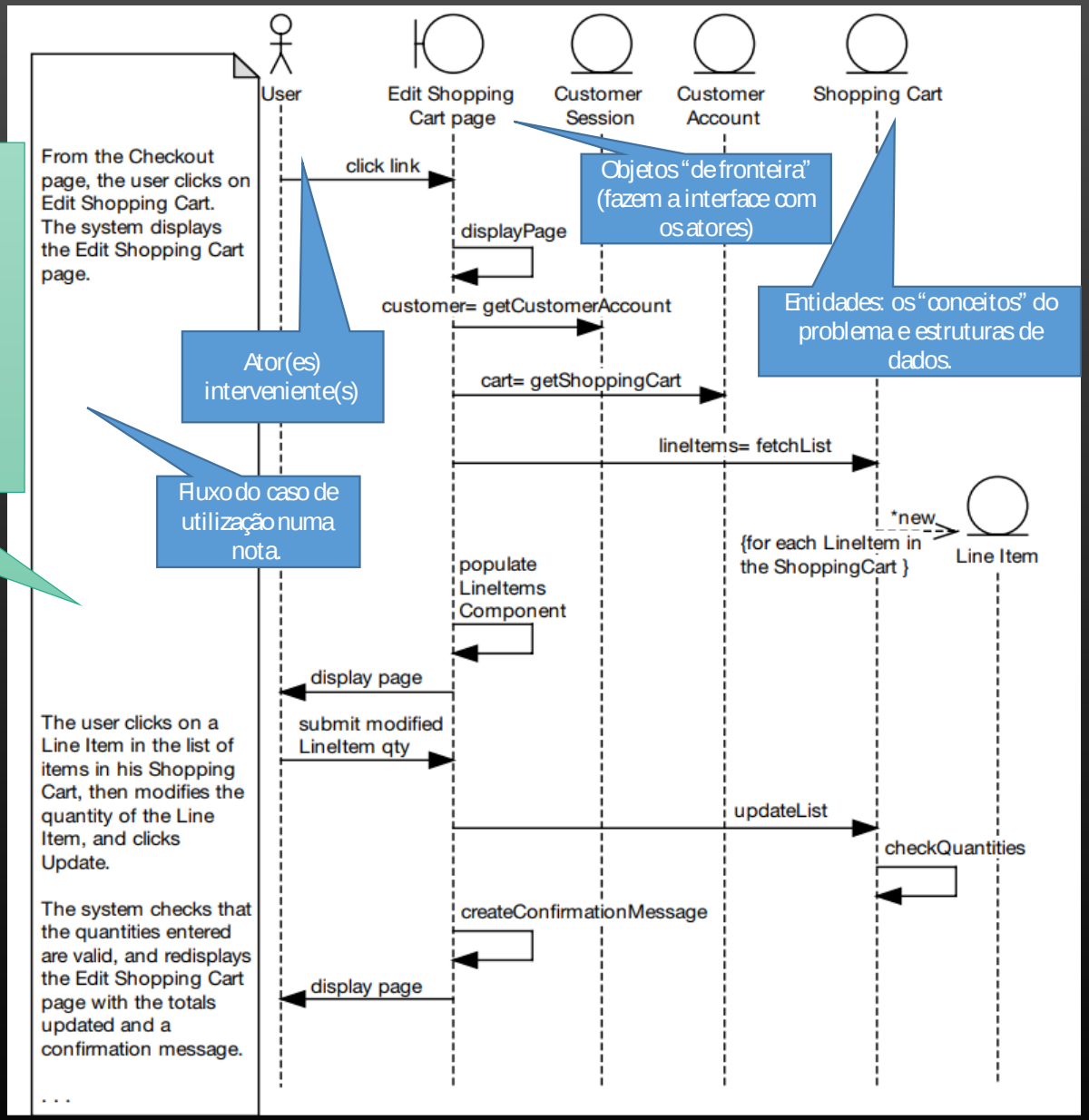


Expansão de cada operação de sistema: qual a colaboração concreta de objetos que a realiza? Processo de descoberta.



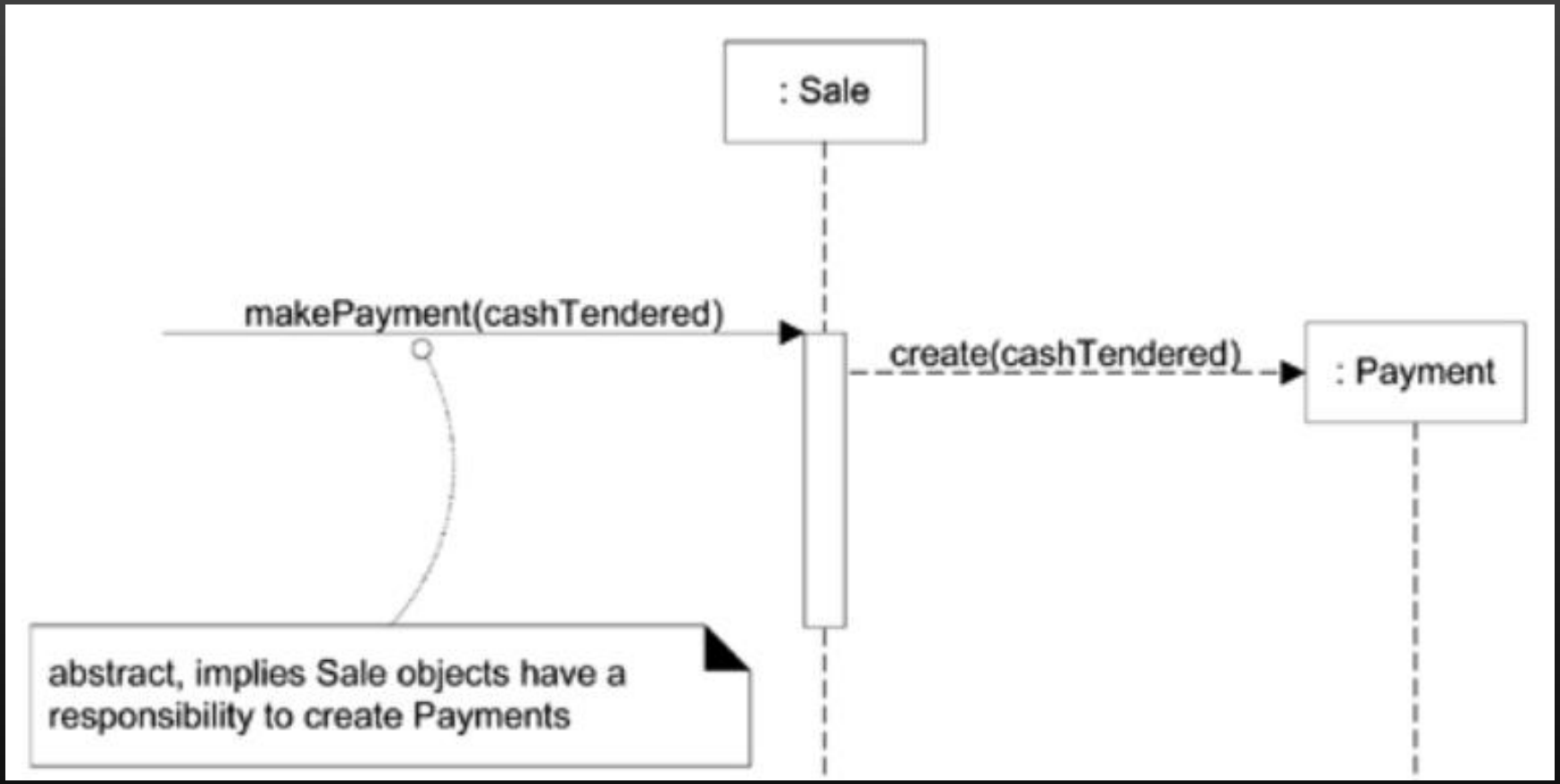
In Rosenbeg:
Da análise para o desenho: utilização dos resultados preparados pelo Analista para desenvolver o “modelo de robustez”
Três categorias de classes:
- Fronteiras
- Controladores
- Entidades





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

Ao desenhar um diagrama de interação, estamos a atribuir responsabilidades



Referências

Core readings	Suggested readings
• [Dennis15] – Chap. 8	• [Larman04] – Chap. 17 and 18 • Slides by M. Eichberg : SSD and OO-Design