

## 4 Lab: Modelação de interações e visualização de código

### Enquadramento

#### Objetivos de aprendizagem



- Explicar a colaboração entre objetos necessária para implementar uma interação de alto nível ou uma funcionalidade de código, recorrendo a diagramas de sequência.
- Usar vistas estruturais (classes) e comportamentais (interações) para descrever um problema.

#### Preparação

- Informação tutorial: [“What is Sequence Diagram”](#)

#### Entrega

Cada grupo deve designar um “pivot” para o lab, que se encarrega de recolher os contributos dos colegas e fazer a entrega. Este papel é rotativo.

O exercício tem atividades para serem realizadas numa **aula** prática. O grupo deve submeter uma entrega, com as respostas aos exercícios assinalados com  

A entrega é um breve relatório, identificando o lab e os autores, e destacando (sublinhado) o aluno que foi o pivot. Anote os seus diagramas, colocando uma nota (*UML note element*) com a informação dos autores e a data de preparação.

### Exercício

#### Parte A: representar interações com diagramas de sequência

##### E4.1 Interpretar um exemplo de interação (nível de sistema)

Explique, por palavras suas, a interação entre as várias “peças” que deve acontecer quando um utilizador realiza uma transação no Multibanco (ATM). Consulte a informação incluída no primeiro resultado da pesquisa, [nesta página](#).

##### E4.2 Controlo de um robot

O módulo “NXT-brick” permite atuar sobre robots LEGO Mindstorms programáveis. Considere que um programador pretende documentar a forma de interagir com o robot, utilizando uma app Android que está a desenvolver (“StormApp”) e que interage com o NXT por Bluetooth. Construa os diagramas de sequência relevantes.

*Estabelecer a conexão inicial:*

“O utilizador arranca a StormApp e escolhe o botão de pesquisa de robots na vizinhança. Para isso, a app deve solicitar a inicialização do subsistema Bluetooth (SB) do dispositivo. Caso necessário, o SB deve informar a app que é preciso pedir permissão (de acesso ao Bluetooth) ao utilizador, como é normal em Android. Nesse caso a app solicita a autorização para usar o Bluetooth e o utilizador confirma. A permissão é comunicada ao SB. O SB indica à app que está

disponível. A app solicita ao SB uma pesquisa de dispositivos alcançáveis, que lança pedidos de descoberta. O módulo NXT recebe um pedido e responde com a indicação do seu endereço MAC. O SB responde à app com a lista de dispositivos NXT encontrados. A app informa o utilizador dos dispositivos alcançáveis, numa lista. O utilizador escolhe o NXT pretendido e a app estabelece a ligação.”

*Avançar para a direita:*

“O utilizador escolhe a ação de navegação na StormApp. A app envia um comando de navegação para o “NXT-brick”; o NXT avalia a exequibilidade do comando; se necessário, o NXT envia o comando de avançar ao motor relevante.”

### E4.3 Visualização de código por objetos

Considere a implementação existente (Lab44codigo.zip) para registar pedidos num restaurante. Para facilitar, o programa gera automaticamente uma ementa, com alguns pratos adicionados e, depois, criar um pedido, escolhendo dois pratos dessa ementa (DemoClass.java → main()). O *output* está exemplificado a seguir.

Para explorar esta implementação, considere usar uma ferramenta<sup>1</sup> com destaque de sintaxe para Java (e.g.: [Visual Studio Code](#), Eclipse).

#### **A preparar os dados...**

```
A gerar .. Prato [nome=Dieta n.1,0 ingredientes, preco 200.0]
    Ingrediente 1 adicionado: Cereal [nome=Milho; Alimento [proteinas=19.3,
calorias=32.4, peso=110.0]]
    Ingrediente 2 adicionado: Peixe [tipo=CONGELADO; Alimento [proteinas=31.3,
calorias=25.3, peso=200.0]]
A gerar .. Prato [nome=Combinado n.2,0 ingredientes, preco 100.0]
    Ingrediente 1 adicionado: Peixe [tipo=CONGELADO; Alimento [proteinas=31.3,
calorias=25.3, peso=200.0]]
    Ingrediente 2 adicionado: Legume [nome=Couve Flor; Alimento [proteinas=21.3,
calorias=22.4, peso=150.0]]
A gerar .. Prato [nome=Vegetariano n.3,0 ingredientes, preco 120.0]
    Ingrediente 1 adicionado: Cereal [nome=Milho; Alimento [proteinas=19.3,
calorias=32.4, peso=110.0]]
    Ingrediente 2 adicionado: Cereal [nome=Milho; Alimento [proteinas=19.3,
calorias=32.4, peso=110.0]]
A gerar .. Prato [nome=Combinado n.4,0 ingredientes, preco 100.0]
    Ingrediente 1 adicionado: Cereal [nome=Milho; Alimento [proteinas=19.3,
calorias=32.4, peso=110.0]]
    Ingrediente 2 adicionado: Cereal [nome=Milho; Alimento [proteinas=19.3,
calorias=32.4, peso=110.0]]
```

**Ementa para hoje:** Ementa [nome=Menu Primavera, local=Loja 1, dia 2020-11-22T21:08:45.624777300]

Dieta n.1	200.0
Combinado n.2	100.0
Vegetariano n.3	120.0
Combinado n.4	100.0

]

#### **Pedido gerado:**

Pedido: Cliente = Joao Pinto

prato: Prato [nome=Combinado n.2,2 ingredientes, preco 100.0]

<sup>1</sup>Se tem experiência de desenvolver com outro IDE, também pode usá-lo, e.g.: Eclipse.

```
prato: Prato [nome=Combinado n.2,2 ingredientes, preco 100.0]
datahora=2020-11-22T21:08:45.813778700]
Custo do Pedido: 200.0
Calorias do Pedido: 95.4
```

### A) Visualização da estrutura do código

A tabela da secção Suporte (adiante) mostra algumas situações-tipo de código (em Java) e a construção correspondente no modelo.

- Identifique, na solução dada, a ocorrência de classes. Represente-as num diagrama.
- Verifique os atributos associados a cada classe. Represente-os.
- Quando uma classe usa atributos cujo tipo de dados é outra classe do modelo, significa que se estabelece uma associação direcionada. Se o atributo for multivalor (um *array*, uma lista, uma coleção), a associação pode ser representada como uma agregação. Represente-as.
- Procure identificar situações de especialização (uma classe estende a semântica de uma classe mais geral, marcado com a palavra *extends*).
- Procure identificar nas classes operações que oferecem. Represente-as.
- Inclua a representação de Interfaces e Enumerados.

Nota: pode ignorar certas operações, designadamente:

<b>get</b> Atributo() <b>set</b> Atributo( parâmetro)	As operações <i>get/set</i> de um atributo que pertence à classe não devem de ser representadas ( <i>getters</i> e <i>setters</i> ).
<i>toString()</i> <i>equals()</i> <i>compareTo()</i>	Estas operações, quando existam, <b>não</b> precisam de ser representadas neste exercício. Têm um propósito predefinido e não vai ser importante para perceber o desenho.

### B) Visualização da interação entre objetos de código

Analisando o Código disponível, procure ilustrar as interações entre objetos que ocorrem quando as seguintes operações são solicitadas:

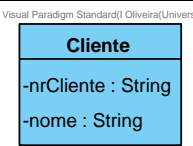
Pedido → calcularTotal();

Pedido → calcularCalorias();

Para isso, recorra a um diagrama de sequência. Para criar cada *lifeline*, pode arrastar a classe correspondente (Pedido,...) da árvore do modelo para o diagrama, caso já as tenha criado.

## Suporte

Sumário da correspondência de conceitos entre código Java e construções da UML.

<pre>public class Cliente {     private String nrCliente;     private String nome; }</pre>	 <p>Visual Paradigm Standard() Oliveira(Universi</p> <p><b>Cliente</b></p> <p>-nrCliente : String</p> <p>-nome : String</p>
--	---

```

public class ClientsPortfolio {
    private String portfolioTitle;
    private List<Client> myClientsList;

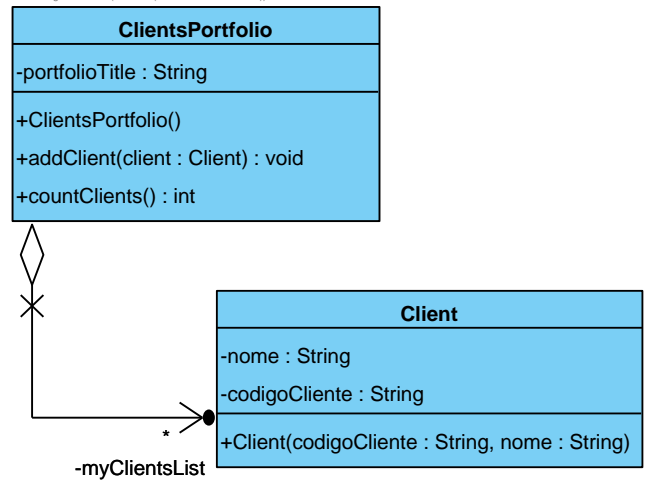
    public ClientsPortfolio( String initialTitle)
    {
        portfolioTitle = initialTitle;
        myClientsList = new ArrayList<>();
    }

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

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

```

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



- ➔ ClientsPortfolio prevê dois atributos; um deles, representa uma coleção de objetos Cliente e pode ser modelado como uma associação.

```

public class Peixe extends Alimento {
    private TipoPeixe tipo;

    public Peixe(TipoPeixe tipo, double proteínas,
double calorias, double peso) {
        super(proteínas, calorias, peso);
        this.tipo = tipo;
    }

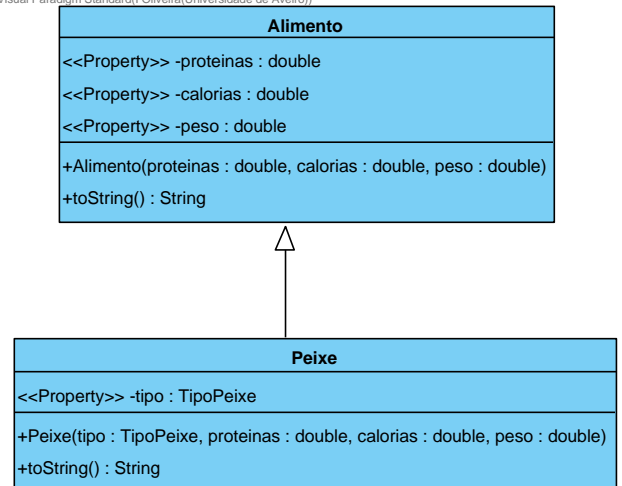
    public TipoPeixe getTipo() {
        return tipo;
    }

    public void setTipo(TipoPeixe tipo) {
        this.tipo = tipo;
    }

    public String toString() {
        // todo
    }
}

```

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



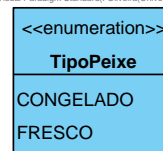
- ➔ Extends declara uma relação de herança
- ➔ Um atributo que tem set tem get pode ser marcado com o esteriótipo *property*.

```

public enum TipoPeixe {
    CONGELADO,
    FRESCO
}

```

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

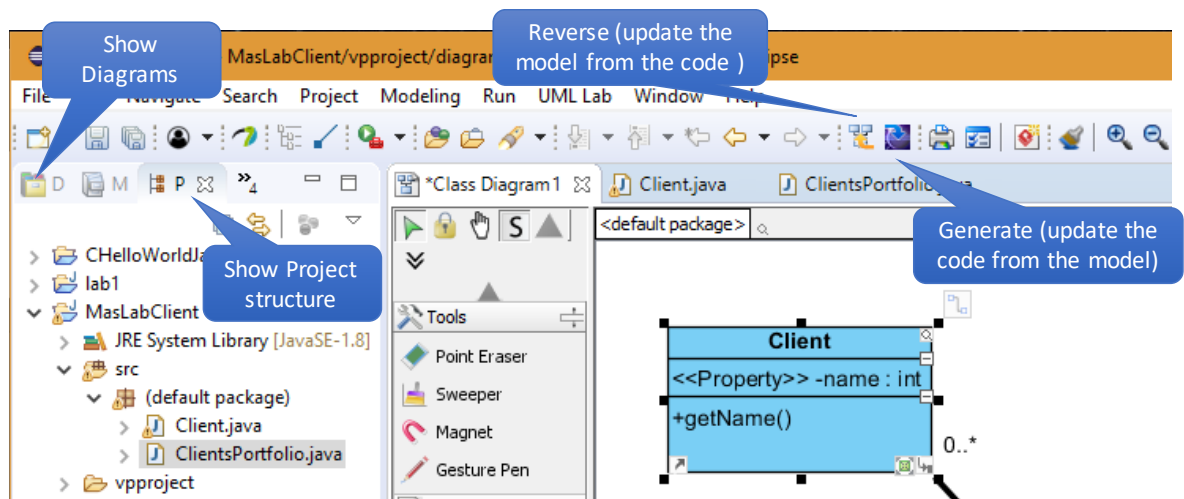


## Material suplementar

A versão Ultimate do IntelliJ IDEA, a que os alunos podem ter acesso no âmbito da [licença académica](#), inclui também um editor UML, que permite fazer o *round-trip engineering* de forma muito prática (diagrama de classes).

O Visual Paradigm pode ser instalado como um plug-in do Eclipse, permitindo o *round-trip engineering* de código Java. A documentação do Visual Paradigm explica como:

- Configurar a integração Eclipse / VP ([Tutorial 1: Getting Started](#)). Execute os passos sequencialmente até à secção “UML Modeling in Eclipse” e pare aqui (exclusive).  
 Note que depois de ter feito a integração (i.e., ficheiros copiados), deve fechar o VisualParadigm e trabalhar a partir do Eclipse.  
 Nota adicional para os utilizadores de Linux<sup>2</sup>
- Sincronizar o modelo com o código ([Tutorial 2 ...with Round-trip Engineering](#)). Execute todos os passos sequencialmente (no Eclipse). Note a referência no último ponto (“3. This is the end of the tutorial...” em que se sugere que introduza código nas classes e verifique que as alterações são refletidas no modelo).



<sup>2</sup> Dependendo da forma como o Eclipse foi instalado, pode obter erros de permissões na escrita de ficheiros relacionados com a integração das ferramentas. Para contornar o problema, considere utilizar uma versão do Eclipse dentro da área do utilizador. Para isso, transfira o zip com a instalação e expanda para uma subpasta da *home* do utilizador e utilize esta instância.