4 Lab: Modelos de comportamento (interações)

4.1 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"

4.2 Representar interações de alto-nível com diagramas de sequência

Neste Lab, não há um *template* específico para o relatório com as respostas aos exercícios. Cada grupo organizará o seu relatório, com referências às secções do guião.

4.2.1 Interação a nível de sistema: sistemas de pagamentos online

Considere que está a desenvolver uma aplicação (de vendas) e pretende integrar com um sistema de pagamento automático.

Para isso, a sua aplicação terá de "dialogar" com um ponto de acesso programático (*API – Application Programming Interface*) de um fornecedor de pagamentos eletrónicos.

Como programador, veja a documentação técnica disponível no fornecedor do serviço, (por exemplo, a EasyPay)

Considere o caso concreto da criação de um pedido de pagamaneto frequente. Explique, por palavras suas, como se deve processar a integração de sistemas.

4.3 Interações no código (por objetos)

Considere a implementação existente (ver: 🗁Práticas/🗁Lab 04 support/DemoEmentas.zip) de um projeto em Java que gere pedidos de um restaurante.

Para facilitar, o programa gera uma ementa aleatória quando executado, com alguns pratos adicionados e, depois, simula um pedido, escolhendo dois pratos dessa ementa (DemoClass.java → main()). O *output* está exemplificado a seguir.

Para explorar esta implementação, considere usar uma ferramenta[1] com destaque de sintaxe, como o Visual Studio Code com o plug-in "Extension Pack for Java" instalado.

Nota: para resolver o exercício, não é preciso dominar a linguagem Java, nem ter um ambiente de desenvolvimento configurado[2], ou sequer executar o programa (embora possa fazê-lo).

[1] Se tem experiência de desenvolver com outro IDE, também pode usá-lo, e.g.: Eclipse, IntelliJ,…
[2] Querendo instalar o ambiente de desenvolvimento para Java, é necessário instalar um JDK (Java Development Kit), e.g. a versão Temurin 17 → https://adoptium.net/installation

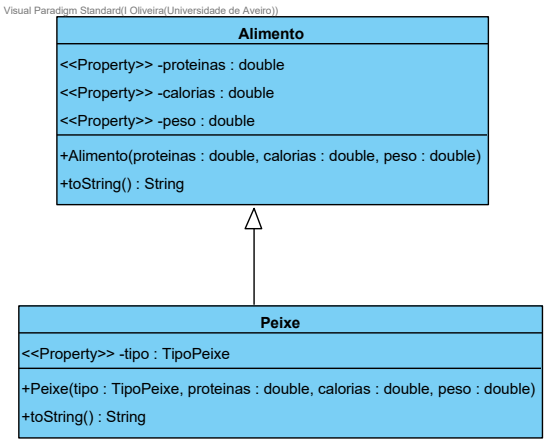
Tabela 1: *output* do programa principal, simulando um pedido de comida.

```
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]
         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
```

4.3.1 Visualização da estrutura do código



O documento de apoio mostra algumas situações-tipo de código (em Java) e a construção correspondente no modelo (ver: 🗁Práticas/🗁Lab 04 support/🗎 Java to UML).

- Identifique, na solução dada (pasta **src/ementas/*** , após expandir o Zip), 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 (i.e., um *array*, uma lista, uma coleção), a associação pode ser representada como uma agregação. Represente as associações que se podem inferir.
- Procure identificar situações de especialização (uma classe estende a semântica de uma classe mais geral, relação "*is a*").
- Procure identificar as operações oferecidas pelos objetos de cada classe. Represente-as.

Nota: neste exercício, para simplificar, pode ignorar certas operações, designadamente:

getAtributo() ***set***Atributo(parâmetro)	As operações *get/set* seguidas do nome de um atributo que pertence à classe são **triviais** (chamam-se *getters* e *setters*)

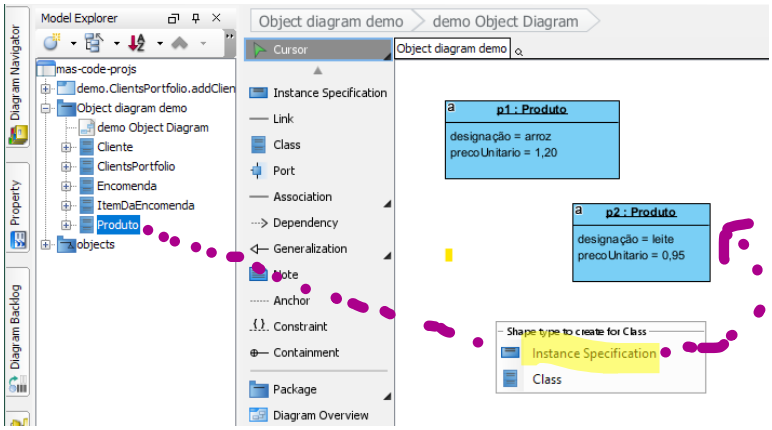
	e geralmente não são representadas no modelo (para maior simplicidade).
public ***NomeDaClasse***(parâmetros)	As operações de uma classe cujo nome da operação é igual ao nome da classe chamam-se *construtores*. Veja que no exemplo junto os construtores foram incluídos, mas pode omiti-los neste exercício.
toString() *equals()* *compareTo()*	Estas operações, podem ou não existir em várias classes e significam sempre o mesmo (têm um propósito predefinido), Por isso mesmo, **não são decisivas para entender um modelo** e podem ser omitidos neste exercício. Veja que no exemplo junto os *toString*() foram incluídos.

4.3.2 Visualização das instâncias (objetos)

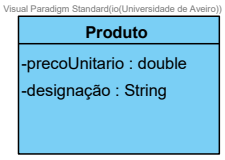
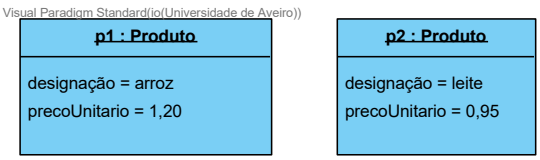
A visualização anterior foca a estrutura das entidades necessárias e responde à pergunta: que tipo de objetos (categorias) estão envolvidos e como se relacionam? Podemos, no entanto, pensar também em termos de objetos (quantas instâncias de cada classe estão envolvidas?).

Considerando a informação que se pode inferir do *output* representado na Tabela 1, podemos ter uma boa ideia de quantos objetos são instanciados de cada tipo e do seu estado (valores dos atributos). Com esta informação[3], prepare um diagrama de objetos. O diagrama pode ser preparado na ferramenta habitual, ou em papel[4].

Nota 1: no diagrama de objetos, representamos instâncias (*instance specification*) que podemos criar facilmente ao "**arrastar**" a classe pretendida para o diagrama.



Nota 2: o diagrama de classes e o diagrama de objetos são distintos. Por exemplo, sendo p1 e p2 instâncias da classe Produto, é importante não confundir:

	
Classe Produto. Representada no diagrama de classes.	Algumas **instâncias** concretas da classe Produto, designadas p1 e p2. Os atributos (=*slots*) recebem valores concretos. Representa-se no diagrama de objetos.

[3] Podemos ainda obter infomação sobre as instâncias criados analisando o código em si, com especial atenção para o operador ***new***, do Java.
[4] É bastante "penoso" criar um diagrama de objetos e definir a informação dos *slots* no Visual Paradigm… É mais fácil fazer "à mão" e digitalizar…

4.3.3 Visualização da interação entre objetos de código

Analisando o código disponível, procure ilustrar as interações entre objetos que ocorrem quando a seguinte operação é solicitada:

- 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.

4.3.4 Diagrama de sequência como instrumento de "descoberta"/planeamento

Assuma que o modelo (parcial) de um domínio (Figura 1) pode ser adaptada para criar as classes essenciais de programação (numa linguagem por objetos).

Pretende-se implementar uma operação "reportarEstado()" que deve permitir aferir, para um projeto, o número total de tarefas associadas, a contagem de tarefas realizadas no prazo, realizadas com atraso, e em progresso.

"Explique" visualmente como implementar esta interação (, i.e., mostre num diagrama de sequências as ativações necessárias). Note que, para isso, precisa de considerar novas operações a adicionar aos tipos (=classes) existentes. Pode explorar alguma analogia com a alínea 4.3.3.

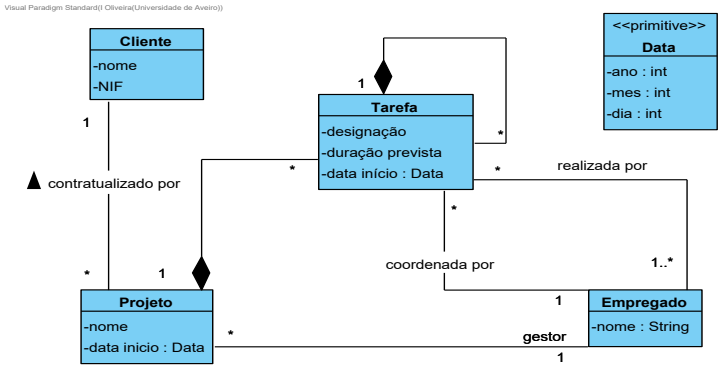


Figura 1

4.4 Gestão de parcerias

Considere a seguinte informação recolhida pelo analista sobre a gestão da relação com os estafetas parceiros (de um serviço de entrega de comida):

- A "frota" de estafetas é bastante dinâmica, estabelecida através da adesão dos interessados à plataforma. Os membros candidatam-se na Plataforma de forma espontânea; após a análise dos elementos solicitados, deverá haver uma aprovação da candidatura pela direção de logística; a adesão pode ser suspensa pela plataforma, ou até cancelada, em função de denúncias recebidas. O estafeta pode pedir para terminar a sua adesão.

A partir deste trecho (hipotético), explique como é que a informação aqui presente podia ser usada para criar os seguintes modelos (e a pertinências de o fazer). Opcional: pode concretizar, se aplicável, com um "mini-diagrama" (para os casos considerados adequados.)

a) Modelo de atividades
b) Modelo de casos de utilização
c) Modelo do domínio (com o diagrama de classes)
d) Modelo de interação (com o diagrama de sequência)
e) Modelo de (máquina de) estados (com o diagrama de estados).