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.

4.2.1 Interação a nível de sistema

Explique, por palavras suas, a interação entre as várias "peças" que deve acontecer quando um utilizador realização um levantamento no Multibanco (ATM), como na Figura 1.

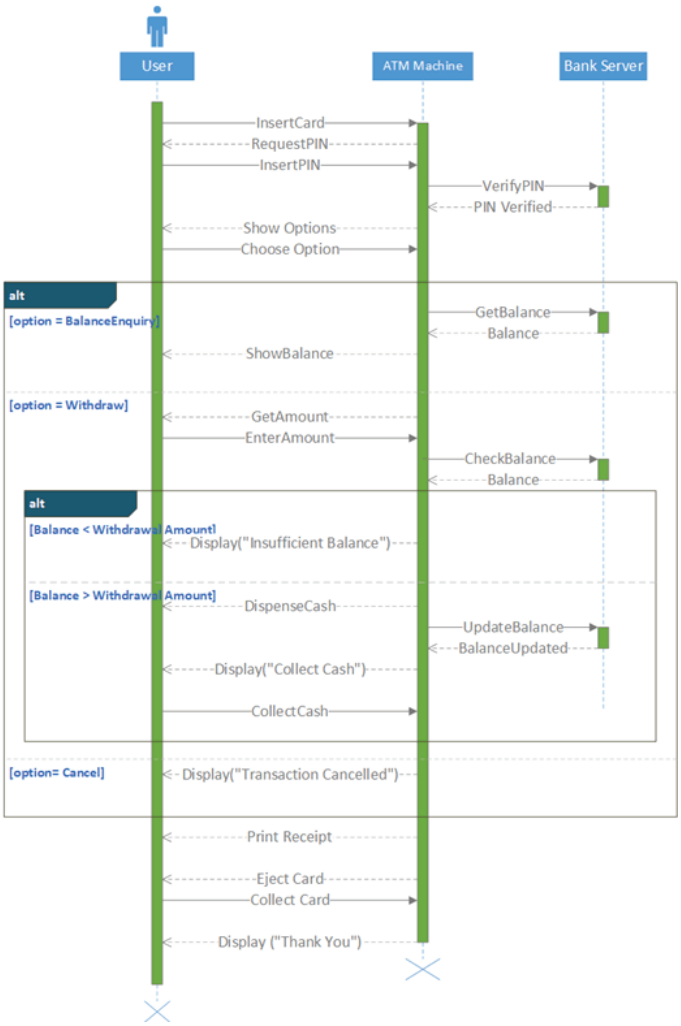


Figura 1: Interação associada à utilização de um ATM (primeiro resultado, nesta página).

4.2.2 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 dipositivos 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."

4.3 Diagramas de interação no desenho e implementação

4.3.1 Dos casos de utilização para os DSS

Considere o contexto da encomenda de comida online que explorou no lab anterior (exercício 3.4).

Escolha um caso de utilização mais importante (ver secção 2.2 do relatório do Lab 3) e desenvolva a narrativa completa, se ainda não o fez (e.g. Criar pedido).

Construa, para este caso de utilização, um diagrama de sequência de sistema (DSS).

O DSS mostra, para um cenário particular de um caso de utilização, os eventos que os atores geram, a sua ordem e integrações inter-sistemas. Todos os sistemas são tratados como uma "caixa preta"; o diagrama enfatiza os eventos que chegam a/solicitam a "fronteira" do sistema.

Esta abordagem é discutida na secção 10 do Livro "Applying UML and Patterns", de C. Larman.

4.3.2 Visualização de código por objetos (classes)

Considere a implementação existente (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

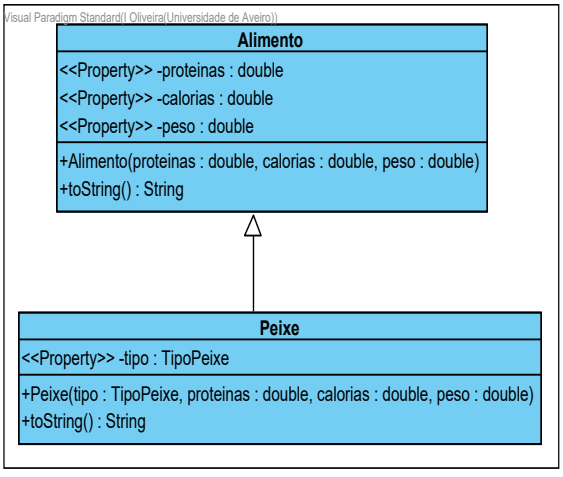
[1] Se tem experiência de desenvolver com outro IDE, também pode usá-lo, e.g.: Eclipse, IntelliJ IDEA.

ambiente de desenvolvimento configurado, ou sequer executar o código (embora possa fazê-lo e ajude a compreender o exemplo).

```
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
```

Visualização da estrutura do código

A tabela da secção 4.4 mostra algumas situações-tipo de código (em Java) e a construção correspondente no modelo.



- Identifique, na solução dada (pasta **src/ementas/***), 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).
- Procure identificar as operações oferecidas por 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 não devem de ser representadas (chamam-se *getters* e *setters*).
public ***NomeDaClasse***(parâmetros)	As operações de uma classe cujo nome da operação é igual ao nome da classe **não** precisam ser representadas (chamam-se *construtores*). Veja que no exemplo junto os construtores foram incluídos.
toString() *equals()* *compareTo()*	Estas operações, quando existam, **não** precisam de ser representadas neste exercício. Têm um propósito predefinido e não vai ser importante para perceber o desenho. Veja que no exemplo junto os *toString*() foram incluídos.

4.3.3 Visualização da interação entre objetos de código

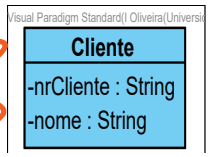
Analisando o Código disponível, procure ilustrar as interações entre objetos que ocorrem quando as seguinte operação é solicitada:

- 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.

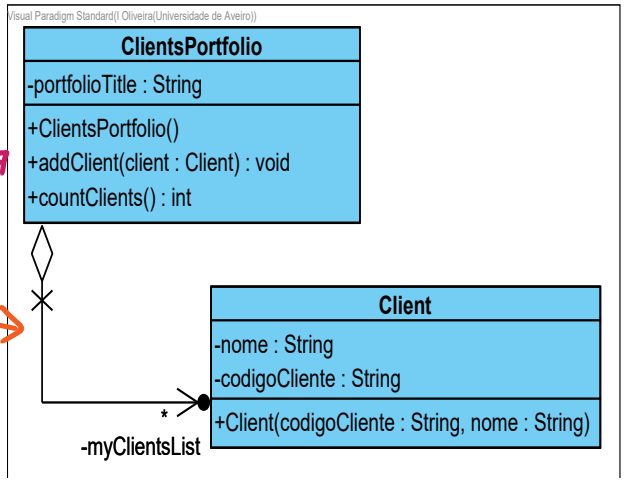
4.4 Suporte: correspondência de conceitos entre código Java e construções da UML

```
public class Cliente {
	private String nrCliente;
	private String nome;
}
```



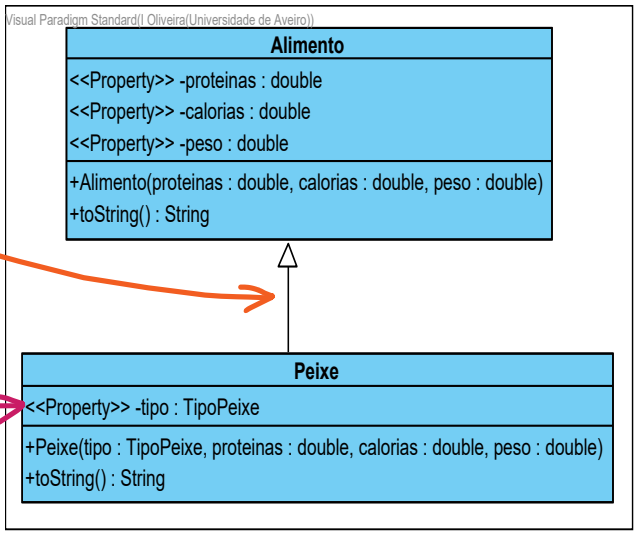
➔ As classes em código são definidas num bloco "**class**"

```
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();
	}
}
```



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

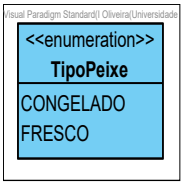
```java
public class Peixe extends Alimento {
    private TipoPeixe tipo;

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

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

    public String toString() {
        // todo
    }
}
```



➔ *Extends* declara uma relação de herança

➔ Um atributo que tem *set* e tem *get* pode ser marcado com o esteriótipo *property*.

```java
public enum TipoPeixe {
    CONGELADO,
    FRESCO
}
```



➔ Tipo especial de estrutura que enumera uma lista de valores admissíveis.