# Programação Orientada a Objetos

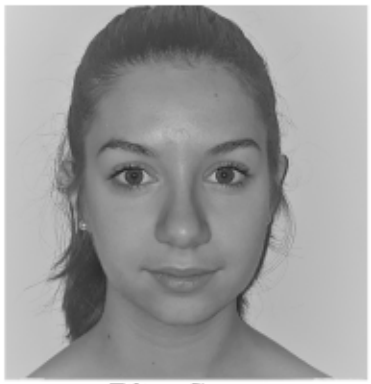
LEI — 2.º ano — 2.º semestre
Universidade do Minho

## House Sync: Uma Aplicação Smart House

Grupo 44



Gonçalo Vilar Vale
(a96923)


Filipa Gomes
(a96556)


Miguel Gomes
(a93294)


Braga, 21 de maio de 2022

**Resumo**

Este relatório, desenvolvido no âmbito da unidade curricular de Programação Orientada a Objetos, foca-se no desenvolvimento de uma aplicação de simulação de consumos de uma *smart city*. A simulação não só permitirá criar uma simulação, como também importar um ficheiro que permitirá criar uma simulação com os parâmetros definidos.

Neste documento descrevemos sucintamente a aplicação desenvolvida, as principais funções utilizadas, assim como as decisões tomadas pelo grupo e os obstáculos encontrados e superados durante o projeto.

# Conteúdo

# 1. Introdução

Para este trabalho foi proposto o desenvolvimento de uma aplicação capaz de criar uma simulação de uma *Smart City* e calcular as devidas despesas dos *Smart Devices* utilizados. A aplicação a respeita o model view controler, de modo que se respeite os requisitos do paradigma de programação orientada a objetos. Este projeto foi concebido na língua inglesa devido a uma maior facilidade em termos de programação e legibilidade de código e classes.

## Estrutura do Relatório

Este relatório encontra-se dividido em 5 capítulos, sendo este o primeiro. Cujo propósito é o de introduzir o projeto, contextualizando-o.

Já no capítulo **2**, pretende-se descrever de forma mais elaborada o problema, com a especificação dos objetivos e requisitos.

No capítulo seguinte, **3**, fazemos referência às estruturas de dados utilizadas para a implementação do código da aplicação.

Em seguida, no capítulo **4**, fazemos uma breve demonstração e explicação do comboio de pensamento para a implementação do nosso código.

No capítulo **5**, a acompanhar uma síntese do trabalho desenvolvido e da solução implementada, termina-se o relatório com uma análise dos problemas que ocorreram durante todo o processo.

# 2. Análise e Especificação

## 2.1 Descrição do problema

O problema proposto consiste na criação de um simulador de uma *Smart City*, contendo uma lista de *Suppliers* de rede, diferentes *Houses* isto tudo numa determinada data ou intervalo de datas. Cada *House* tem diferentes *Divisions* tendo estas diferentes *Smart Devices*.

## 2.2 Especificação de Requisitos

De forma a ser possível a construção correta do nosso sistema tivemos de seguir alguns requisitos base de gestão de entidades, sendo eles:

- capacidade de criação de *SmartDevices*, *Houses* e *Suppliers*;
- capacidade de ligar e desligar, ou seja, mudar o mode dos *SmartDevices*;
- que seja permitido a mudança de localização temporal;
- habilidade de cálculo de consumo e gestão de *Invoices*.

Existindo também requisitos estatísticos, como:

- permitir saber a *House* que mais gastou naquele período;
- permitir saber qual *Supplier* tem maior volume de faturação;
- listar as faturas emitidas por um *Supplier*;
- visualização de uma lista ordenada dos maiores consumidores de energia durante um período a determinar.

# 3. Conceção da Resolução

Foi-nos pedido o desenvolvimento de uma aplicação de uma *SmartCity* onde fosse possivel cumprir o paradigma da programação orientada a objetos e que os requisitos mencionados no capítulo **2**. Para tal, definimos as seguintes classes:

## 3.1 Package MVC_House_Sync

As classes neste package são as classes principais da aplicação. É através delas que gerimos a interação com o utilizador e com os dados, seguindo o paradigma de programação de Model, View, Control.

### 3.1.1 Model

Esta classe define-se como o estado do projeto. Os seus métodos permitem interagir com toda a informação através da classe *Controler*.

### 3.1.2 View

Esta classe implementa métodos de IO com o utilizador. Os métodos permitem imprimir infor- mação e possivelmente devolver uma resposta dada pelo utilizador. O uso dos métodos *ask_input_s* e *ask_input_i* permitem rapidamente criar uma "pergunta"ao utilizador, e são frequentemente usados na classe de controlo, poupando uma quantidade considerável de código para menus diferentes.

### 3.1.3 Controler

Esta classe une o aspeto de interação com o utilizador da View com o modelo de dados Model. De uma forma geral, esta classe funciona à base de uma sequência de métodos de menus que levam uns aos outros através de switch cases (que representam as várias escolhas do utilizador). Os atributos desta classe são usados para gerir *Houses*, *Divisions* e *SmartDevices* dependendo das escolhas do utilizador. O construtor desta classe tenta primeiro carregar o modelo de dados a partir de um ficheiro de dados chamado logs.txt. Se ocorrer um erro a abrir o ficheiro, o modelo de dados é inicializado vazio.

## 3.2 Package Simulator

A pasta *Simulator* contem as classes *Events*, *Invoices* e *Simulator*. A classe *Events* é constituída por um evento e a sua respetiva data. A classe *Invoice* é constituída pelo consumo mensal, id da fatura, *Address*, data de início e data final, consumo final, preço a pagar, preço por watt e aproximação de valores.

## 3.3 Package House

A pasta *House* contem as classes *Address*, *Divisions* e *House*. A classe *Address* é constituída por a rua, número da rua, cidade e código-postal. A classe *Divisions* é constituída pelo nome da divisão, um *set* de *SmartDevices* e um *Atomic Integer* que é um contador para o identificador da divisão em causa.

## 3.4 Package SmartDevices

A pasta *SmartDevices* contem as classes *SmartBulb*, *SmartSpeaker*, *SmartCamera* e *SmartDevice*. Cada uma destas classes contem as características de cada um dos *SmartDevices* respetivos. A classe *SmartDevices* é constituída pelo id do *device*, estado, data de instalação, nome do *device*, *brand*, consumo energético, custo base, histórico e contador de tempo ligado.

## 3.5 Package Auxiliar

A pasta *Auxiliar* contém as classes *Auxiliar_Methods*, *Consumption*, *FilaHandler*, *MyRandom*, *Pair*, *State* e *Wait*. *Auxiliar_Methods* é a classe que contem funções necessárias para o projeto porém não se encaixam em nenhuma classe em específico. Na classe *Consumption* o objetivo é ir buscar o uso energético. A classe *FileHandler* implementa métodos estáticos de leitura e escrita para ficheiros de objetos, de forma a que seja facilmente guardado e carregado o estado da simulação. A classe *Pair* é utilizada, para representar intuitivamente um par de um dado tipo de dados. É utilizada no projeto, por exemplo, para definir o código-postal de um *Address*. *MyRandom* é uma classe que serve para gerar valores *random*. A classe *State* tem como objetivo guardar os *logs* dos respetivos *SmartDevices*. Por fim a classe *Wait* serve para que haja *delay* na apresentação, neste caso, de menus de maneira a que o utilizador possa ler.

# 4. Codificação

## 4.1 Gestão de dados

Através dos menus definidos em Controler e os métodos de IO em View, facilitamos a criação e edição de novas *Houses* e *Suppliers* ao utilizador. Funcionalidade de ordenação e filtragem de dados também permitem ao utilizador encontrar dados relevantes rapidamente.

## 4.2 Implementação

Para a possível interação do utilizador com o simulador foram criados menus com o objetivo de facilitar o uso do simulador. Como exemplo, temos o primeiro menu que aparece após entrar no simulador:

```
    //base menu
    public String baseMenu(){
        clear();
        loadingMenu();
        System.out.println('''
            Select an option:

            1 - Manage simulation
            2 - Manage houses
            3 - Present billing statistics
            4 - View Menu

            0 - Quit
            ''');
        return (input.nextLine());
    }
```

O processo de escolha de opções foi feito através de um switch com o input do utilizador. Neste caso se no primeiro menu o utilizador escolher a opção 1, ou seja premir a tela 1, esta função será ativada com o case 1, ou seja, abriria outro menu de edição da simulação.

```
public void firstMenu(){
    boolean flag = true;
    while(flag) {
        View.clear();
        String choice = view.baseMenu();
        switch (choice) {
            case '1':
                crSimMenu_2();
                break;
            case '2':
                manage_houses();
                break;
            case '3':
                billing_menu();
                break;
            case '4':
```

```
                    view_menu();
                    break;
                case '0':
                    if (this.unsavedChanges) {
                        loadingMenu_controler();
                    }
                    else System.exit(0);
                    break;
                default:
                    View.unrecognizedCommandError();
                    break;
            }
        }
}
```

## 4.3 Alternativas, Decisões e Problemas de Implementação

Durante a nossa implementação, vimo-nos deparados com alguns problemas, que eventualmente não conseguimos corrigir. O maior contratempo com que nos deparamos foi o cálculo dos preços nas faturas que em alguns casos dava valores absurdos como preço final como, por exemplo, 1.73215322452E42. Isto resultou em alguma perda de tempo atentar decifrar o problema, porém conseguimos baixar os preços por watt (muito) dando um preço final mais razoável. Também ocorreu o problema de remoção de qualquer categoria de objeto, desde *House* a *Suppliers* e *SmartDevices*, resolveu-se ocultar esta funcionalidade.

# 5. Conclusão

Através da realização deste trabalho foi possível consolidar e colocar em prática os conhecimentos obtidos ao longo da unidade curricular de Programação Orientada a Objetos. Além disso, adquirimos um maior domínio da linguagem de programação Java e do Paradigma de Programação Orientada a Objetos através do desenvolvimento do projeto.

Relativamente às dificuldades enfrentadas durante a execução deste trabalho, o grupo reconheceu que a maior dificuldade foi

Apesar desta limitação, concluímos ter feito um bom trabalho face àquilo que nos foi proposto e consideramos que os conhecimentos adquiridos serão úteis em futuros projetos.

# 6. ANEXOS

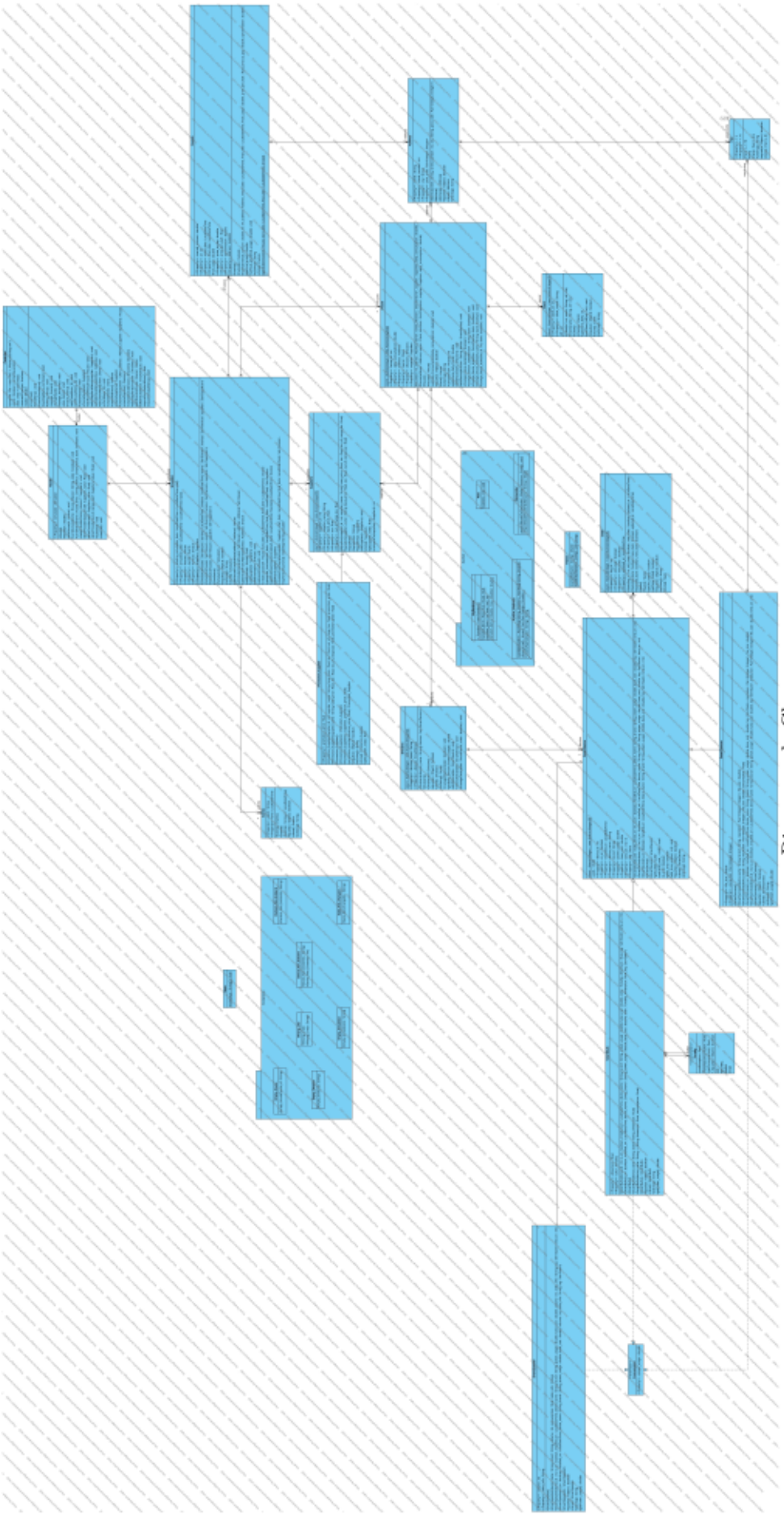


Diagrama de Classes

**foi utilizado um software com free licence para gerar o diagrama*