Universidade do Minho

# LICENCIATURA EM CIÊNCIAS DA COMPUTAÇÃO

## Programação Orientada aos Objetos

Trabalho Prático - Grupo nº15


**Bruno Miguel Ferreira Fernandes**
(a95972)

**Bruno Daniel Rolo Neiva**
(a95311)

**José Pedro Silva Ferreira**
(a96798)

# Conteúdo

# 1. Introdução

Este projeto consiste na construção de um sistema que monitorize e registe a informação sobre o consumo energético das habitações de uma comunidade, com o intuito de colocar em prova os conhecimentos obtidos ao longo do semestre na Unidade Curricular de Programação Orientada aos Objetos. Este sistema foi desenvolvido na linguagem de programação Java.

# 2. Classes

## 2.1 SmartHome

```
public class SmartHome implements Serializable {
    13 usages
    private String name; //dono da casa
    12 usages
    private int NIF; //NIF dono da casa (nome e NIF utilizar nas faturas) logo nao deve poder ser privado
    17 usages
    private List<String> divisoes;      //Divisoes da casa
    14 usages
    private Map<String,SmartDevice> devices; //  Devices todos de uma casa
    17 usages
    private Map<String, List<String>> devicesPorDivisao;   //colecao de dispositivos por divisao de uma casa
    13 usages
    private Comercializador comercializador; // ter em conta a Composicao entre as classes
    //ex:  Sala = [devices]  devices queremos que sejam as referencias
         //key    values
```

Esta classe é a mais geral do programa, a partir dela temos acesso a todas as informações de uma casa, desde o nome do seu proprietário aos dispositivos que ela possui.

## 2.2 SmartDevice

```
public abstract class SmartDevice implements Serializable {
    /**
     * Instancias
     * */
    10 usages
    private String id; // codigo fabricante
    11 usages
    private boolean on;
```

É uma classe onde a partir dela podemos identificar os *smartdevices* e atribuir o seu estado, se está ligado ou desligado.

## 2.3 SmartBulb

```
public class SmartBulb extends SmartDevice {
    11 usages
    private Tonalidade tone;
    9 usages
    private int tam; //dimensao (cm)
    9 usages
    private double consumod; //consumo diario
```

Esta classe é uma classe mais especifica da classe "SmartDevice" no qual identificamos as lâmpadas das casas e onde atribuímos os pesos específicos da classe como vemos acima.

## 2.4 SmartCamera

```
public class SmartCamera extends SmartDevice {

    10 usages
    private String resolucao;
    10 usages
    private int tamficheiro;
    9 usages
    private double consumod;
```

Esta classe é uma classe mais especifica da classe "SmartDevice" na qual identificamos as camaras das casas e onde atribuímos os pesos específicos da classe como vemos acima.

## 2.5 SmartSpeaker

```
public class SmartSpeaker extends SmartDevice {

    1 usage
    public static final int MAX = 100; //volume máximo

    14 usages
    private int volume;
    10 usages
    private String radio;
    10 usages
    private String marca;
    10 usages
    private double consumod;
```

Esta classe é uma classe mais especifica da classe "SmartDevice" no qual identificamos as colunas das casas e onde atribuímos os pesos específicos da classe como vemos acima

## 2.6 Comercializador

```
public class Comercializador implements Serializable {

    2 usages
    public static final int PrecoBase = 10;
    3 usages
    public static final double Imposto = 0.23;
```

Esta classe é a responsável pela identificação dos comercializadores de energia e é responsável pela fórmula que determina o valor do custo diário de energia.

## 2.7 Parser

O parser é uma classe construída pela equipa docente, que sofreu alterações de acordo com aquilo que era pretendido. Esta classe é responsável por ler um ficheiro de texto e organizar os Comercializadores, Casas e Divisões, já criadas, no mesmo.

## 2.8 Simulacao

Esta classe é responsável por fazer a simulação do valor do consumo de energia de várias casas. Esta classe também é capaz de avançar no tempo para emitir as faturas, guardar essa nova data para avançar novamente, alterar o estado dos *devices* das casas e imprimir as faturas respetivas a um Comercializador.

## 2.9 SaveSimulacao

Esta classe salvaguarda os dados da simulação que podem ser carregados para serem novamente simulados.

## 2.10 Controller

```
public class Controller implements Serializable{
                    //ID,Casa
    13 usages
    private Map<Integer,SmartHome> casas;
    9 usages
    private Map<String,Comercializador> comercializadores;
    8 usages
    private Map<String,SmartDevice> devices;
    1 usage
    private Simulacao simulacao;
```

Esta classe é responsável pela gestão do menu inicial, comunica com outras classes executando as suas operações.

## 2.11 View

```
public class View {

    16 usages
    private Controller controller;
    33 usages
    private Scanner input;
```

A classe View, é a interface do programa. Esta classe é o que permite a interação utilizador-programa. Comunica única e exclusivamente com o Controller e possui vários métodos, métodos esses que tem a função de mostrar algum tipo de informação ou apresentar mensagens de erro.

## 2.12 Main

Esta classe é a classe principal que inicializa as componentes necessárias para o funcionamento do programa.

# 3. Exceptions

## 3.1 ComercializadorjaExisteException

Esta exceção foi criada para alertar o utilizador que o comercializador que inseriu já está guardado.

## 3.2 ComercializadorNaoExisteException

Esta exceção foi adicionada no âmbito de avisar o utilizador que o comercializador não existe.

## 3.3 DataImpossivelException

Esta exceção foi criada para alertar que existe um erro relacionado com as datas inseridas.

## 3.4 DivisaoJaExisteException

Esta exceção foi adicionada no caso de o utilizador ter definido uma divisão já existente na casa.

## 3.5 DivisaoNaoExisteException

Esta exceção foi inserida para informar o utilizador que a divisão definida não existe ou não foi definida de forma correta.

## 3.6 LinhaIncorretaException

Esta exceção foi criada com o intuito de comunicar ao utilizador que uma das linhas está de forma incorreta.

## 3.7 NumeroFormatoImpossivelException

Esta exceção foi criada com o intuito de alertar que o número do NIF está num formato errado.

## 3.8 RespostaImpossivelException

Esta exceção tem o intuito de informar o utilizador que a resposta que colocou não é aceite.

## 3.9 SimulacaoImpossivelException

Esta exceção serve para notificar o utilizador que a simulação que tentou efetuar não foi possível de se executar.

## 3.10 SmartDeviceJaExisteException

Esta exceção foi criada para avisar o utilizador que o *SmartDevice* criado já existe na casa.

## 3.11 SmartDeviceNaoExisteException

Esta exceção foi criada para avisar o utilizador que o *SmartDevice* que está a tentar alterar não existe.

### 3.12 SmartHomeJaExisteException

Esta exceção foi adicionada para alertar o utilizador que a SmartHome definida já existe.

### 3.13 SmartHomeNaoExisteException

Esta exceção foi acrescentada para notificar ao utilizador que a SmartHome definida não existe ou foi definida de forma incorreta.

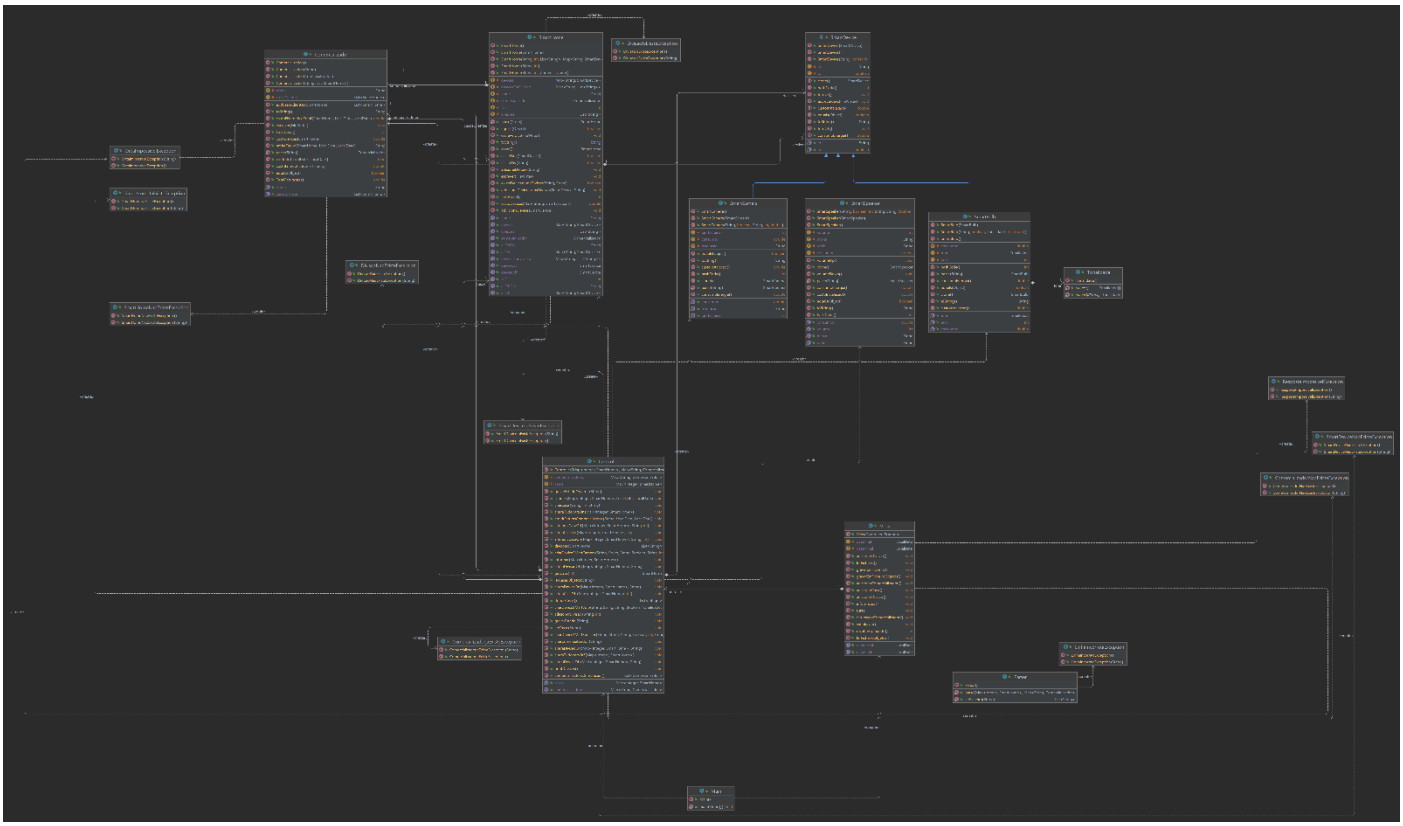
# 4. Estrutura do Projeto

O nosso projeto segue a estrutura Model View Controller (MVC), estando por isso organizado em três camadas:

- A camada de dados (o modelo) é composta pelas Classes SmartHome, SmartDevice, SmartBulb, SmartSpeaker, SmartCamera, Comercializador e Simulacao.
- A camada de interação com o utilizador (a vista, ou apresentação) é composta unicamente pela classe View.
- A camada de controlo do fluxo do programa (o controlador) é composta pela classe Controller.

O projeto inteiro foi inspirado na ideia de encapsulamento.

# 5. Diagrama de Classes



# 6. Conclusão

De um modo geral, pensamos ter um trabalho bem conseguido. Conseguimos cumprir todos os requisitos mínimos e ainda mais alguns. Uma das dificuldades que mais tivemos foi na ordenação dos maiores consumidores de energia e também na automatização da Simulação.

Em suma, pensamos ter cumprido o objetivo colocado.