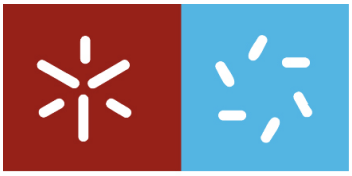


Universidade do Minho

Licenciatura em Ciências da Computação

POO - Trabalho Prático

Grupo nº5

Simão Pedro Batista Caridade Quintela
(A97444)

Pedro Alexandre Silva Gomes
(A91647)

Gonçalo da Silva
(A95696)

21 de maio de 2022



Conteúdo

Capítulo 1

Introdução

Este relatório descreve o desenvolvimento do projeto prático da Unidade Curricular de Programação Orientada aos Objetos, inserida no 2ºano da Licenciatura em Ciências da Computação da Universidade do Minho.

Este trabalho consistia no desenvolver de um sistema que monitorize e registe a informação sobre o consumo energético das habitações de uma comunidade. Em cada casa existe um conjunto muito alargado de dispositivos que são todos controlados a partir deste programa, os *SmartDevice*. Para que a realização do referido fosse possível, utilizamos os conhecimentos da linguagem *Java*, adquiridos ao longo do semestre.

Capítulo 2

Estrututa do projeto

O nosso projeto segue o modelo de estrutura ***Model View Controller* (MVC)**, que consite na organização deste em três camadas:

- A camada de dados **(o modelo)** é composta pelas classes *CasaInteligente, SmartDevice, SmartBulb, SmartSpeaker, SmartCamera, Comercializador, Fatura, Comunidade, Tuple, SaveProgramText, Parser, SimulParser*.
- 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*.

Em todo o projeto respeitamos a ideia de encapsulamento, como pedido, utilizando a estratégia de **agregação**.

Capítulo 3

Classes

Estas são as classes que constituem o nosso programa:

3.1 App

```
static Scanner scan = new Scanner(System.in);
static Controller controller = new Controller(comunidade);
```

Esta classe contém a *main* do programa na qual este é inicializado. Nesta mesma classe são inicializados o *Controller*, a *View* e o *Scanner*, sendo a *View* inicializada com os parâmetros *Controller* e *Scanner*.

3.2 Comunidade

```
private String nomeDaComunidade;
private Map<String, CasaInteligente> casas; // proprietario -> Casa
private Map<String, Comercializador> mercado; // fornecedor ->
    Comercializador
```

Esta classe tem como propósito guardar todas as casas criadas, bem como comercializadores e o nome dado à própria comunidade. Para isso, utilizamos dois **Maps** sendo que o primeiro associa o proprietário ao objeto casa e o segundo associa o fornecedor ao objeto fornecedor.

3.3 CasaInteligente

O **package CasaInteligente** contém as classes *CasaInteligente*, todos os tipos de *SmartDevices* incluindo a classe abstrata *SmartDevice* e todas as classes de Test associadas a estas.

3.3.1 CasaInteligente

```
private String proprietario;
private int NIF;
private String fornecedor;
private Map<String, SmartDevice> devices; // identificador ->
    SmartDevice
private Map<String, List<String>> locations; // Espa o -> Lista
    codigo dos devices
private List<Fatura> faturas; // lista de faturas que foram geradas e
    associadas    casa
```

Esta classe tem várias variáveis de intância com informação acerca da casa e do seu proprietário. Em termos de estruturas de dados, utilizamos um **Map** para guardar os *SmartDevices* da casa em que a **key** é o **ID** do device do tipo **String** e o correspondente **value** é o objeto *SmartDevice* associado. Utilizamos também outro **Map** que, a cada divisão, associa uma lista de **IDs** de *SmartDevices*, que nela estão contidos, sendo que as **keys** são **Strings** a representar o nome da divisão e os **values** são do tipo **List<String>** no qual guardamos os **IDs** dos *SmartDevices* presentes na divisão. Por fim, temos uma estrutura **List<Fatura>** para guardar na casa todas as faturas emitidas cujo o destinatário é a própria casa.

3.3.2 CasaInteligenteTest

Esta classe contém alguns testes que fomos utilizando durante o desenvolver do projeto.

3.3.3 SmartDevices

Este **package** contém todas as classes correspondentes aos SmartDevices e respetivos testes.

- SmartDevice

  ```
  private String id;
  private boolean on;
  private LocalDate time;
  private float consumption;
  private float consumptionPerDay;
  private int custoInstalacao;
  ```

  Esta é uma classe que tem por objetivo a repretentação dos 3 tipos de *SmartDevices(SmartBulb, SmartCamera e SmartSpeaker)*. Decidimos que esta seria uma classe abstrata, devido ao facto de, os diferentes tipos de *SmartDevices* terem muita informação em

comum **(ID, Estado, ConsumoEnergetico, ...)**, ou seja, a nossa intenção foi agrupar todas as características numa classe genérica, em que os diferente tipos de *SmartDevices* a extendem e a variação de informação entre *SmartDevices* é feita na própria classe correspondente do *SmartDevice*.

- SmartDeviceTest
- SmartBulb

```
private static final int WARM = 80;
private static final int NEUTRAL = 60;
private static final int COLD = 40;
private int tone;
private int dimensions;
```

Esta classe contém todas as informações relevantes quanto a *SmartBulb* (tonalidade em que está ligada, dimensões) e possui métodos que contam o consumo energético num dado período de tempo, bem como métodos de **turnOn** e **turnOff**.

- SmartBulbTest
- SmartSpeaker

```
private static final int MAX = 100; //volume maximo
private int volume;
private String channel;
private String brand;
```

Esta classe contém todas as informações relevantes quanto a *SmartSpeaker* (volume, marca e canal em que está ligado) e possui métodos que contam o consumo energético num dado período de tempo, bem como métodos de **turnOn**, **turnOff**, **VolumeUp** e **VolumeDown**.

- SmartCamera

```
private int xRes;
private int yRes;
private int fileSize;
private float custoInstalacao;
```

Esta classe contém todas as informações relevantes quanto a *SmartCamera* (resolução e tamanho de ficheiro) e possui métodos que contam o consumo energético num dado período de tempo, bem como métodos de **turnOn**, **turnOff**.

3.4 ComercializadoresEnergia

Neste **package** estão todas as classes relacionadas com fornecedores de energia.

3.4.1 Comercializador

```
private String nomeEmpresa;
private int numeroDispositivos;
private int valorBase;
private int imposto;
private Map<String, List<Fatura>> faturas;  // Propriet rio -> Lista
    de Faturas
```

Esta classe representa um Comercializador de Energia e contém informação sobre o mesmo como, por exemplo, **nome da empresa, valor base, imposto, etc...** . Quanto à fórmula escolhida para cada fornecedor, tentamos dar uma certa liberdade ao fornecedor de poder escolher um número dispositivos que, quando ultrapassado, a cobrança seja diferente. A fórmula escolhida foi:

```
if(numeroDispositivosCasa > numeroDispositivos(var. de instancia))
    consumo = consumoDoDispositivo / 1500 * (imposto / 100) * 0.9
else
    consumo = consumoDoDispositivo / 1500 * (imposto / 100) * 0.75
```

Quanto a estruturas de dados, utilizamos um **Map** para guardar todas as faturas emitidas pelo comercializador em questão, sendo que as **keys** são o nome do proprietário da casa e os values são a lista de faturas emitidas para a mesma do tipo **List<Fatura>**.

3.4.2 Fatura

```
private int codigo;
private int nif;
LocalDate dataEmissao;
private float total;
private String empresa;
private String cliente;
```

A classe *Fatura* representa um objeto fatura que é utilizado para registar o consumo de uma casa num dado período de tempo. Para isso, utilizamos variáveis como, **cliente, empresa, dataEmissao, etc...** .

3.5 ConsumeComparator

Esta classe é usada para comparar consumos.

3.6 View

```
private Controller controller;
private Scanner scan;
```

A *View* é a classe como a qual se faz a interação com o utilizador. Para isso, solicita operações ao Controller que são respondidas com sucesso.

3.7 Controller

```
private Comunidade comunidade;
private int idFatura;
private LocalDate timeNow;
```

A classe *Controller* satisfaz os pedidos da *View* e, para isso, utiliza 3 variáveis de instância **(comunidade, idFatura e timeNow)**. A variável **comunidade** guarda a comunidade inicializada à criação do *Controller*, a **idFatura** guarda o número de faturas criadas e a variável **timeNow** guarda o tempo atual em que está a correr o programa.

3.8 Parser

A classe *Parser* lê um ficheiro de texto com informações acerca da comunidade e gera todos os objetos resultantes disso.
Fornecedor:<NomeFornecedor>, <NumeroDispositivos>
Casa:<NomeProprietario>,<NifProprietario>,<NomeFornecedor>
DivisaoEDevices:<Divisao>,<Devices>
SmartCamera:<Resolucao>,<Tamanho>,<Consumo>
SmartSpeaker:<Volume>,<CanalRadio>,<Marca>,<Consumo>

3.9 SaveProgramText

Esta classe tem como objetivo gravar o programa num ficheiro de texto.

3.10 SimulParser

A classe *SimulParser* é utilizada para ler o ficheiro de configuração de uma simulação.
Demos a possibilidade do utilizador realizar 6 ações:

1. Turn On (turnOn)
2. Turn Off (turnOff)
3. Mudar de fornecedor (mudar)
4. Mudar número de dispositivos num fornecedor (mudarNumDisp)
5. Mudar localização de um dispositivo (novaLoc)
6. Remover um dispositivo (remover)

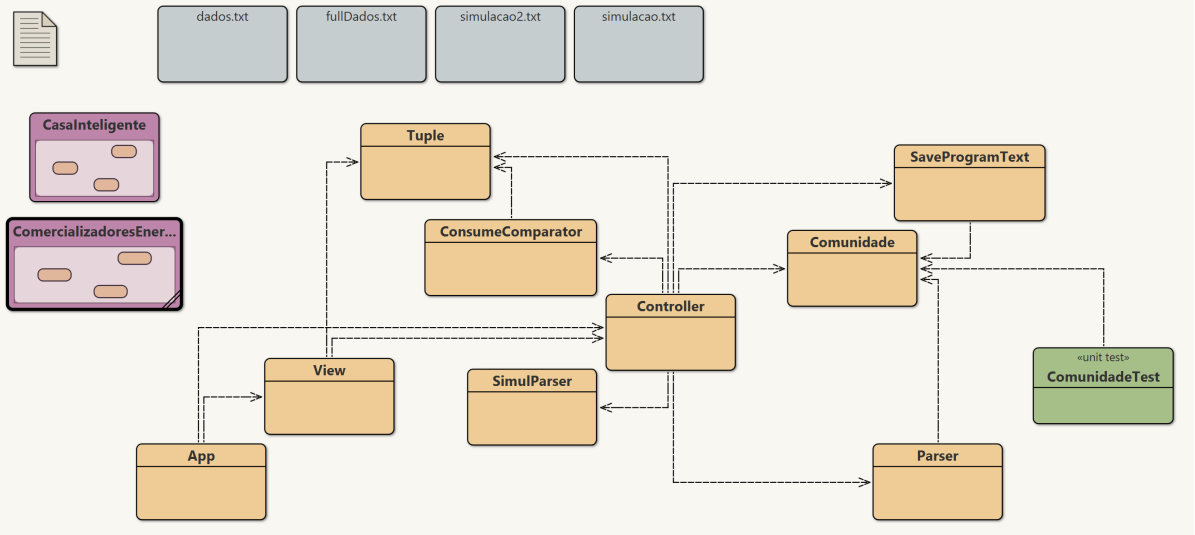
Foram usadas as seguintes convenções:
data,proprietario,idDispositivo,turnOn
data,proprietario,idDispositivo,turnOff
data,proprietario,fornecedor,mudar
data,fornecedor,numDispositivos,mudarNumDisp
data,proprietario,idDispositivo,novaLoc,localizacao
data,proprietario,idDispositivo,remover

3.11 Tuple

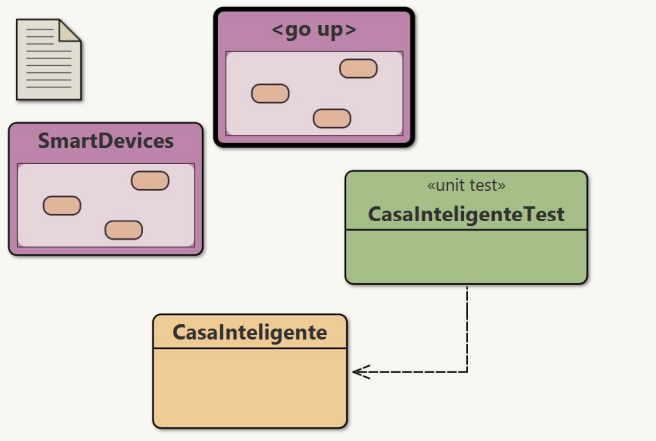
```
private String p1;
private float p2;
```

A classe *Tuple* foi criada para responder à necessidade de retornar mais do que um parâmetro ao desenvolver as estatísticas do programa, sendo as variáveis de instância **p1** e **p2** o primeiro e segundo elemento do tuplo, respetivamente.

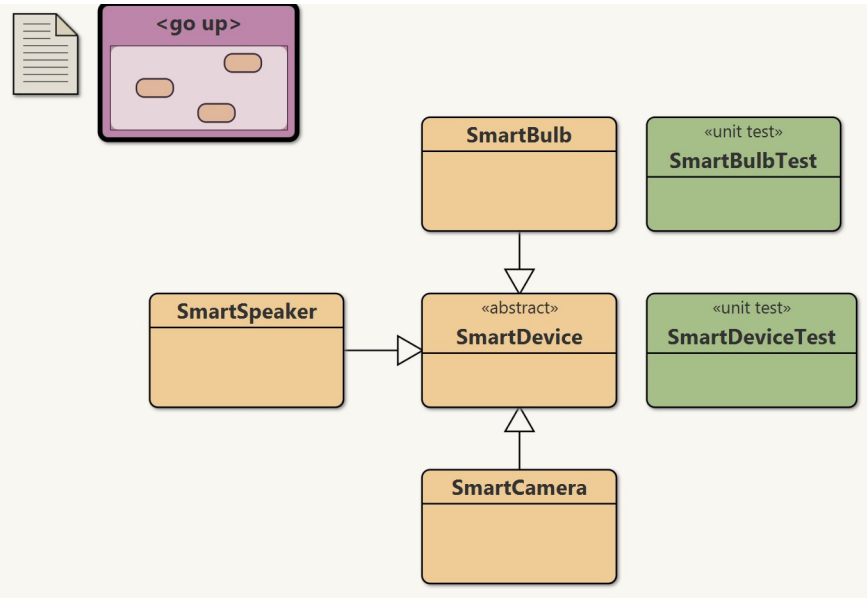
3.12 Diagrama de classes do *blueJ*



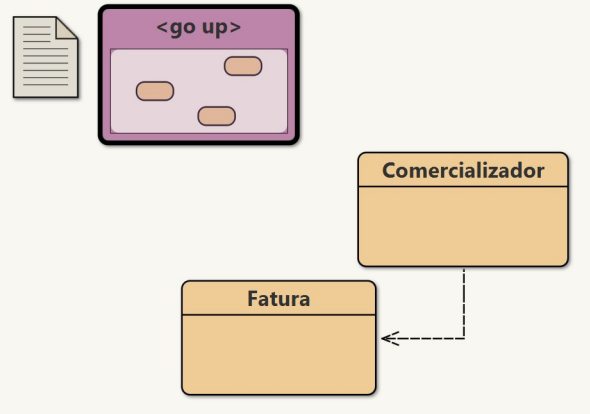
1. Diagrama principal com as super classes, interfaces de simulação e ficheiros .txt



2. Diagrama da classe CasaInteligente



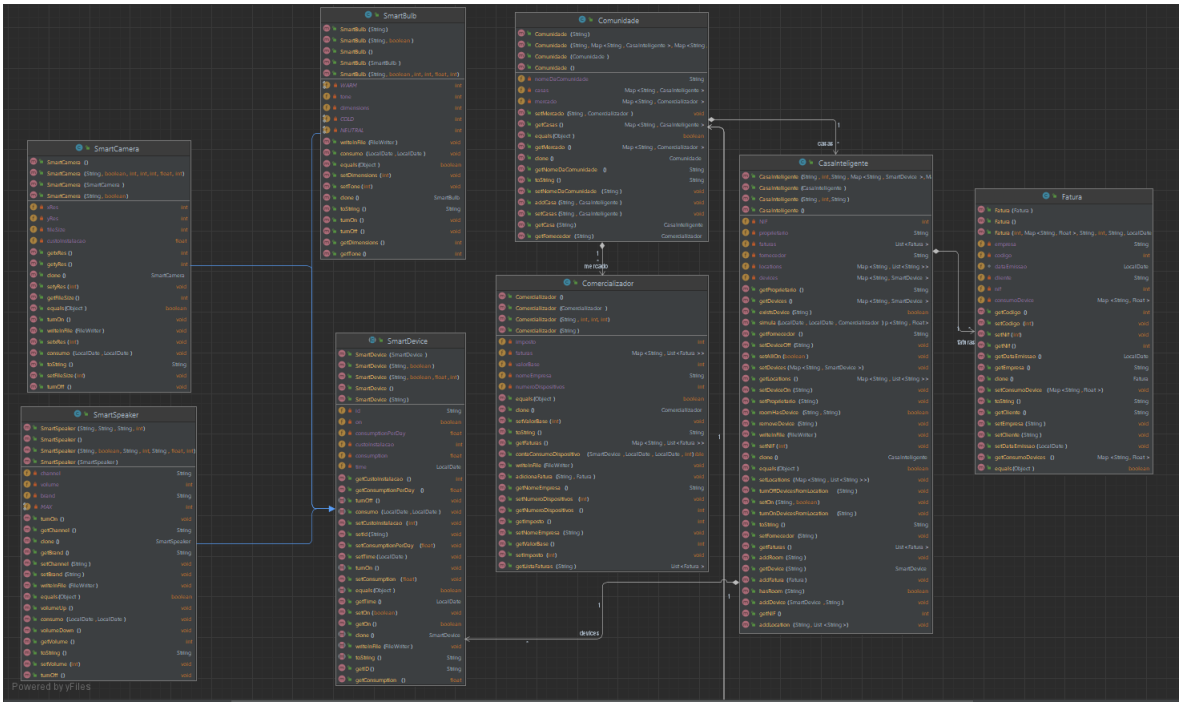
3. Diagrama da classe SmartDevices

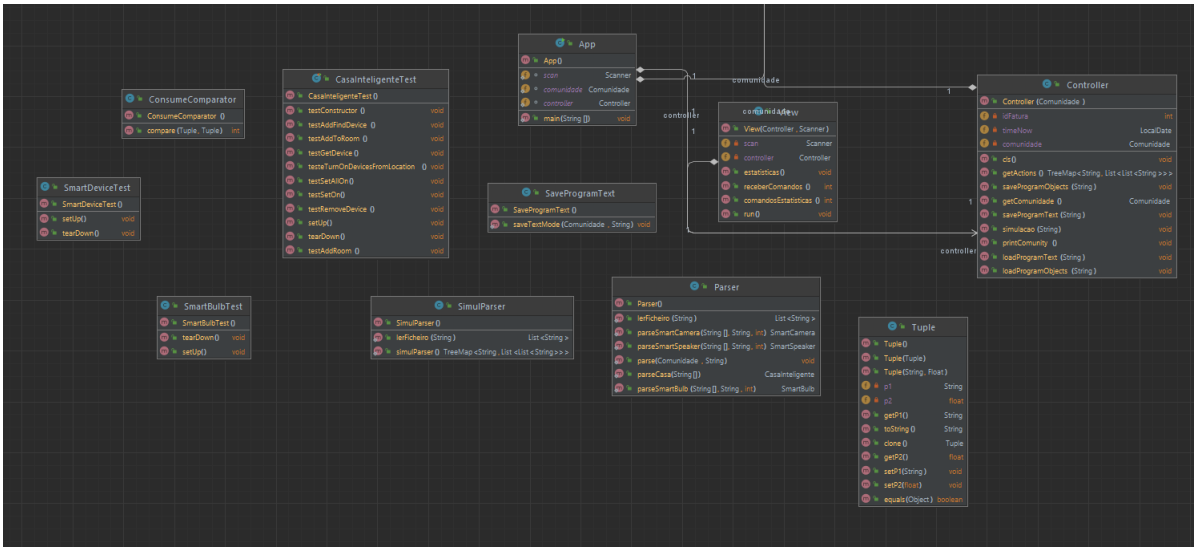


4. Diagrama da classe ComercializadoresEnergia

Capítulo 4

Diagrama do *Intellij*

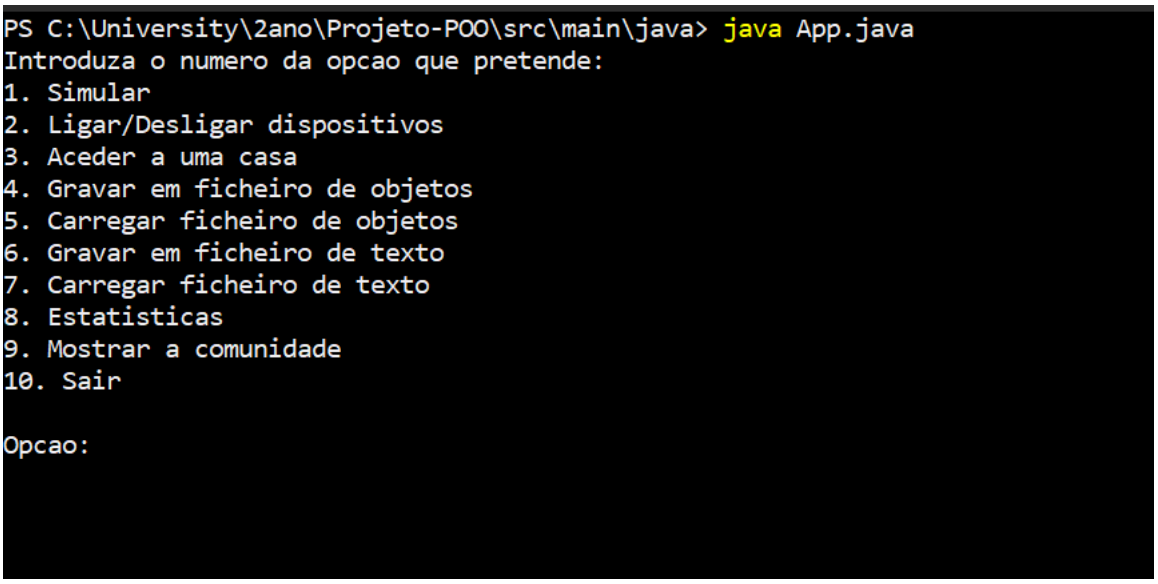



App
App()
scan Scanner
comunidade Comunidade
controller Controller
main(String[]) void
CasaInteligenteTest
CasaInteligenteTest()
testConstructor() void
testAddFindDevice() void
testAddToRoom() void
testGetDevice() void
testeTurnOnDevicesFromLocation() void
testSetAllOn() void
testSetOn() void
testRemoveDevice() void
setUp() void
tearDown() void
testAddRoom() void
ConsumeComparator
ConsumeComparator()
compare(Tuple, Tuple) int
SmartDeviceTest
SmartDeviceTest()
setUp() void
tearDown() void
SaveProgramText
SaveProgramText()
saveTextMode(Comunidade, String) void
View
View(Controller, Scanner)
scan Scanner
controller Controller
estatisticas() void
receberComandos() int
comandosEstatisticas() int
run() void
Controller
Controller(Comunidade)
idFatura int
timeNow LocalDate
comunidade Comunidade
cls() void
getActions() TreeMap<String, List<List<String>>>
saveProgramObjects(String) void
getComunidade() Comunidade
saveProgramText(String) void
simulacao(String) void
printComunity() void
loadProgramText(String) void
loadProgramObjects(String) void
SmartBulbTest
SmartBulbTest()
tearDown() void
setUp() void
SimulParser
SimulParser()
lerFicheiro(String) List<String>
simulParser() TreeMap<String, List<List<String>>>
Parser
Parser()
lerFicheiro(String) List<String>
parseSmartCamera(String[], String, int) SmartCamera
parseSmartSpeaker(String[], String, int) SmartSpeaker
parse(Comunidade, String) void
parseCasa(String[]) CasaInteligente
parseSmartBulb(String[], String, int) SmartBulb
Tuple
Tuple()
Tuple(Tuple)
Tuple(String, Float)
p1 String
p2 float
getP1() String
toString() String
clone() Tuple
getP2() float
setP1(String) void
setP2(float) void
equals(Object) boolean
comunidade
controller
1

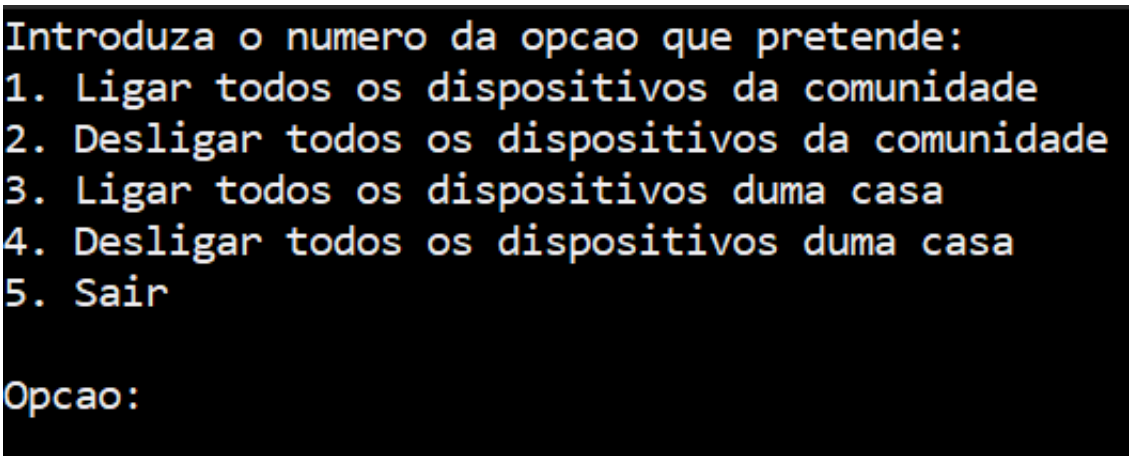
Capítulo 5

Funcionalidades do Programa

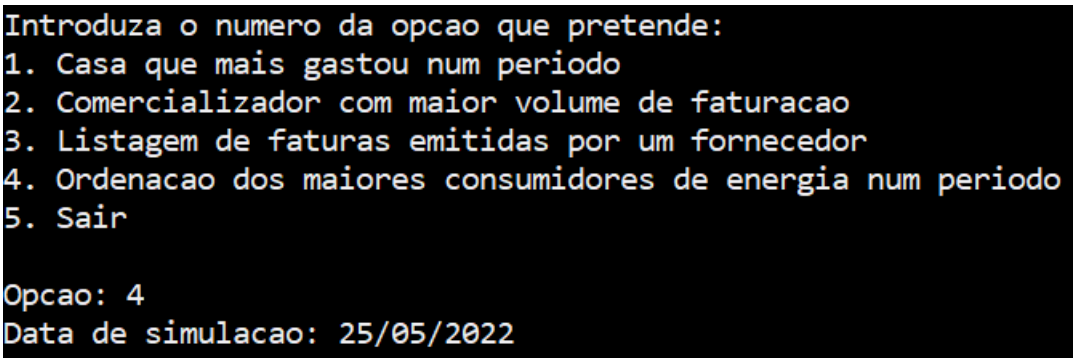
5.1 Menu Principal



5.2 Menu Dispositivos



5.3 Menu Estatísticas



Capítulo 6

Conclusão

Em conclusão, a nível geral, e tendo em conta o panorama apresentado nos capítulos anteriores e os objetivos pedidos para esta realização, como grupo, achamos que todos os objetivos foram cumpridos, conseguindo superar com sucesso, todas as dificuldades que fomos encontrando, sempre com um olhar crítico e empenho. Acreditamos que aprendemos os conteúdos e objetivos desta Unidade Curricular e ,em particular, deste projeto.