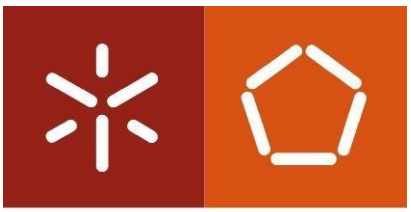

UNIVERSIDADE DO MINHO

Licenciatura Em Engenharia Informática

POO - Trabalho Prático

Grupo 34

José Neiva (A92945) José Freitas (A96140)
Maurício Pereira (A95338)

Ano Letivo 2021/2022



Conteúdo

Capítulo 1

Introdução e principais desafios

Este projeto consistiu no desenvolvimento de uma aplicação de gestão do consumo energético das habitações de uma comunidade na linguagem de programação Java, de forma a pôr em prática os conhecimentos adquiridos ao longo do semestre.

Consideramos que o maior desafio tenha sido fazer a simulação dos consumos gerais de cada casa, pois todas as casas têm dispositivos diferentes em sítios diferentes que podem, ou não, estar ligados.

Tivemos também bastantes incertezas sobre as formulas a utilizar para o calculo dos consumos de cada SmartDevice, pois são todos diferentes.

Capítulo 2

Classes

1. Main

Classe responsável por arrancar com o programa. Para isto o método *main* inicia a execução do Controller;

2. View

View é a classe responsável por todos os prints da Interface permitindo ao menu de ter uma coerência. Por exemplo, no main menu encontramos “Select a House”, “Devices”, “LoadDatabase”, “SaveDatabase”, “View Database”, “Simulate” e “Quit”. Isto permite ao utilizador de se guiar e aceder ao local do programa ao qual ele deseja aceder.

3. SmartDevice

```
private int id;
private State state;
```

SmartDevice é a classe base de todos os Devices (Bulb,Camera,Speaker). Permite guardar o id e o estado de cada device

4. Controller

Classe, que juntamente com outras, gere o fluxo da aplica¸c˜ao. Esta é a classe respons´avel por gerir o menu inicial, disponível para todas as entidades, possibilitando a interação do utilizador.
Contém todos os Controllers para cada uma das entidades, sendo eles:

- houseController que permite adicionar e retirar devices de um quarto e também modificar as informações escritas na database de uma casa específica.

- deviceController que permite inicializar qualquer smart device entre o speaker, a lâmpada e a câmara.

- simulationController que permite simular o consumo de energia de uma ou todas as casas durante um espaço de tempo específico. Também é possível verificar a casa com o maior consumo e alterar o fornecedor de energia.

5. SmartBulb

```
private double dimension;
private State consumption;
private Tonality ton;
```

SmartBulb é a classe responsável por toda a informação das lâmpadas. Permite a manipulação de aspetos como a dimensão da lâmpada, o seu consumo e a sua tonalidade.

6. SmartCamera

```
private CameraResolution resolution;
private int size;
```

SmartCamera é a classe responsável por toda a informação das câmaras. Permite a manipulação de aspetos como a resolução da câmara, o seu consumo e o seu tamanho.

7. SmartSpeaker

```
private int volume;
private String channel;
private String SpeakerBrand;
private Double consumption;
```

SmartCamera é a classe responsável por toda a informação relativa aos speakers. Permite a manipulação de aspetos como o seu volume, o seu canal, a sua marca e o seu consumo.

8. Simulation

```
private DataBase database;
private List<Bill> bills;
private String LocalDate currentDate;
```

Simulation é a classe responsável por fazer a simulação do consumo energético das habitações da comunidade. Esta classe possui métodos capazes de devolver uma simulação do gasto de uma casa em um determinado tempo (em dias) definido pelo utilizador.

9. Bill

```
private  double totalCost;
private double consumo;
private LocalDate emissionDate;
private LocalDate startDate;
private long period;
```

Esta classe é responsável por fazer a "fatura" da simulação, ou seja, tem o custo total associado, o consumo, a data de emissão a data de Inicio e ainda o período de tempo.

10. House

```
private String ownerName;
private String nif;
private EnergySeller seller;
private Map<Integer, SmartDevice> devices;
private Map<String, List<Integer>> locations;
```

A classe House é a classe responsável pela informação relativa às casas. Isto inclui a manipulação de dados tais como o nome do vendedor e proprietário, o seu nif e a localização.

11. Exceptions

Criamos as classes de Exceptions (**DeviceDoesnotExistException, HouseDoesNotExistException, IncorrectLineException, RoomDoesNotExistException, TonalityDoesNotExistException**) para os casos em que algo pesquisado pelo user não exista na Data Base do programa. Neste caso, com a ajuda da classe, o programa envia uma String a informar de que tal pesquisa não existe.

12. EnergySeller

```
private String energySeller;
private Double priceKw = 0.20;
private Double tax = 0.10;
```

A classe EnergySeller é a classe responsável pela informação relativa aos vendedores de energia. Isto inclui a manipulação de dados tais como o nome, o preço e a taxa dos vendedores.

Capítulo 3

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 da Entidades e pelas classes Main, EnergySeller, House,Simulation, Bill,ReaderWriter, Exceptions.
- 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 unicamente pela classe Controller.

Como foi referido anteriormente, todo o projeto baseia-se na ideia de encapsulamento, cuja relação é de agregação.

Capítulo 4

Conclusão

Pensamos ter respondido de forma adequada ao que nos foi pedido no enunciado. Conseguimos respeitar o encapsulamento no nosso código, que é um dos principais fatores da UC e conseguimos ultrapassar os desafios que nos foram colocados à medida que fomos aprendendo novos conceitos e nos habituamos a trabalhar com a linguagem de programação Java. Sabemos também que há muito onde melhorar, e havia ainda bastantes coisas para fazer e otimizar, contudo, esforçamo-nos bastante.

Capítulo 5

Diagrama de Classes

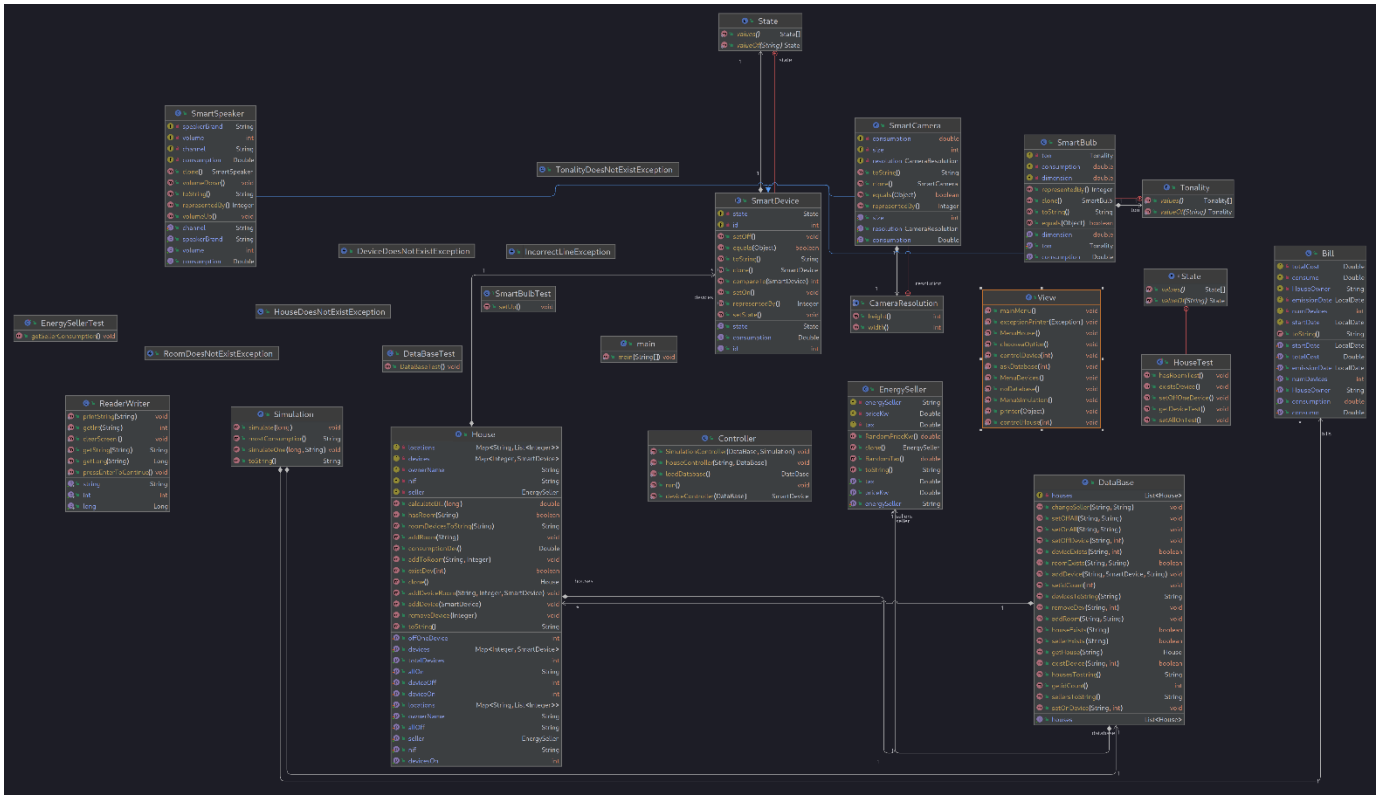

Figura A.1: Diagrama de classes do programa, gerado pelo *IntelliJ*