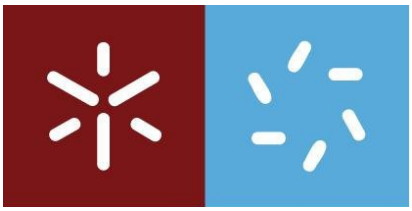


Universidade do Minho

Licenciatura em Ciências da Computação

POO - Trabalho Prático

Grupo 3

Breno Fernando G. Marrão (A97768)
Tales Andre M. Rovaris (A96314)
Tiago Passos Rodrigues (A96414)

2021/2022

Conteúdo

Capítulo 1

Introdução

Este projeto consiste em realizar uma aplicação de gerenciamento de casas inteligentes, os smart devices da mesma e comercializadores de energia baseada no conceito de programação orientada a objetos da linguagem Java.

O grupo encontrou algumas dificuldades em realizar o parsing das linhas de um ficheiro de texto e fazer as devidas alterações, tomamos a decisão de todas as classes implementarem a interface serializable para podermos gravar em ficheiro no modo binário.

Capítulo 2

Classes e subclasses

2.1 Main

A classe main inicia o programa e a classe menu que faz parte do paradigma Model-View que demos nas aulas Teóricas.

2.2 Menu

Nessa classe foi implementado o View onde faz-se o contacto entre utilizador e programa. Nessa classe estão disponíveis métodos de criação e alteração de vários estados dos comercializadores e casas inteligentes como também dos seus Smart Devices através do terminal ou através de ficheiros. O grupo decidiu implementar a classe menu como uma ponte entre programa e utilizadores para o melhor funcionamento do programa e melhor divisão do trabalho entre as diferentes classes.

```
private Simulacao simular;
private Scanner menu;
```

2.3 Simulacao

Nesta classe é armazenada toda a informação a respeito das casas inteligentes, comercializadores de energia e smart devices , usando conceitos de Facade e Controller. Nesta classe também é armazenada a informação que recebemos do usuário através da classe menu , usufruindo dos conceitos de composição e encapsulamento , pois consideramos a melhor alternativa devido ao tema do projeto e não haver problemas em compartilhamento de dados dentro do programa. Aqui estão definidas as seguintes variaveis de instancia :

```
private Map<Integer, CasaInteligente> casas;
private Map<String,ComercializadorEnergia> comercializadores;
private LocalDateTime dia;
private Map<String,Marca> marcas;
```

Além dessas funcionalidades a classe também implementa método de Simulação manual por dias e um método que automatiza a simulação através de informação em um ficheiro, salvar e ler

em ficheiros, estatísticas relativas aos dados. Decidimos guardar toda a informação nesta classe para ser mais simples de gravar e ler em ficheiros , com a escrita em binário .

2.4 ComercializadorEnergia

Esta Classe guarda informação sobre um comercializador de energia, emite faturas e calcula o gasto de uma casa pela a energia gasta que está associada a este. Optamos por usar uma lista para guardar as faturas pois não precisamos de as acessar. As variáveis de instância desta classe são:

```
private String nome;
private double custoDiarioEner;
private static double IMPOSTO = 2.0;
private double volumeFatura;
private List<String> faturas;
```

O grupo decidiu implementar o comercializador de energia como uma classe sem relação direta com as casas inteligentes para poder tratar as casas inteligentes com mais independência sem necessidade de acessar sempre o comercializador de maneira a facilitar a procura e aplicação de métodos tanto da casa inteligente como dos smart devices.

2.5 CasaInteligente

Nesta classe é onde podemos encontrar toda informação referente a uma casa inteligente como todos seus devices, todos os quartos e quais devices estão nesses quartos. Decidimos guardar em uma estrutura map para facilitar a procura , também é possível saber o gasto de energia e o gasto financeiro da casa , os métodos referentes a esta classe são os de ligar e desligar smart devices e os adicionar.

```
private String proprietario;
private String morada;
private int NIF;
private Map<String, SmartDevice> devices;
private Map<String, List<String>> locations;
private String comercializadorEn;
private double gastoCasa;
private double gastoEnergia;
private double gastoSimulacao;
```

2.6 SmartDevices

Esta classe é a uma classe que está contida em uma hierarquia na qual é a superclasse. Definimos como abstracta para podermos obrigar as subclasses a ter o método que calcula o custo de energia de um smart device, tomamos esta decisão para podermos agrupar vários tipos de subclasses como SmartDevices e conseguirmos calcular os seus custos. A classe contém o id do smart device , se está ligado ou desligado e o seu custo de instalação.

```
private String id;
private boolean on;
private double custoInstalation;
```

2.7 SmartBulb

A classe SmartBulb é uma subclasse do SmartDevice que retrata uma lâmpada , logo conterá informações como o tom da luz , a dimensão e o seu custo de energia diário em relação à dimensão e o tom da luz.

```
public static final int WARM = 2;
public static final int NEUTRAL = 1;
public static final int COLD = 0;
private int tone;
private int dimensao;
private int custoDiario;
```

2.8 SmartCamera

A classe SmartCamera é uma subclasse do SmartDevice que retrata uma câmera, logo conterá informações como qualidade de resolução da câmera e o tamanho do arquivo que conterá o que foi gravado. O seu custo diário irá ser relativo a esses parâmetros.

```
public static final double QUATROK = 1;
public static final double ULTRAHD = 0.6;
public static final double HD= 0.3;

//variaveis de instancia
private double resolucao;
private double tamanho_ficheiro;
```

2.9 SmartSpeaker

A classe SmartSpeaker é uma subclasse do SmartDevice que retrata um altifalante , logo conterá informações como o volume , canal em que se encontra e a marca que é uma classe que estará associada por composição ao SmartSpeaker. O seu custo diário irá ser relativo a esses parâmetros.

```
public static final int MAX = 20; //volume maximo
private int volume;
private String channel;
private Marca marca;
```

2.10 Marca

A classe marca é uma classe que está associada com o SmartSpeaker , sendo a marca de cada SmartSpeaker. Decidimos criar esta classe como uma entidade separada para estar mais associada ao paradigma de programação orientada a os objetos, esta classe contém o nome da marca e o custo associado a essa marca.

```
private String nome;
private int custo;
```

Capítulo 3

Estrutura

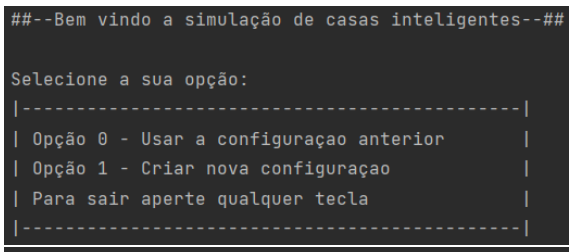
O nosso projeto procura implementar uma fragmentação do programa semelhante ao MVC e utilizando também o conceito de Facade para realizar a gestão dos dados e suas alterações.

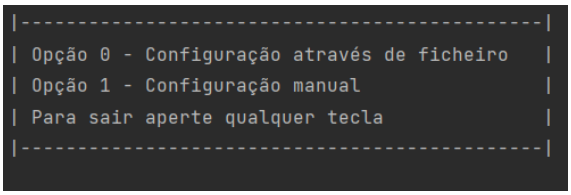
1. A classe Menu interage com o utilizador, recebendo os dados de input e trata deles enviando-os para a classe Simulacao.
2. A classe Simulacao recebe os dados diretamente do Menu e trata deles, criando as Casas, bem como os SmartDevices e Comercializadores de energia pertencentes. Também implementa métodos de cálculos de estatística, também como gestão dos mesmos , visto que o acesso aos dados é feito por esta classe.

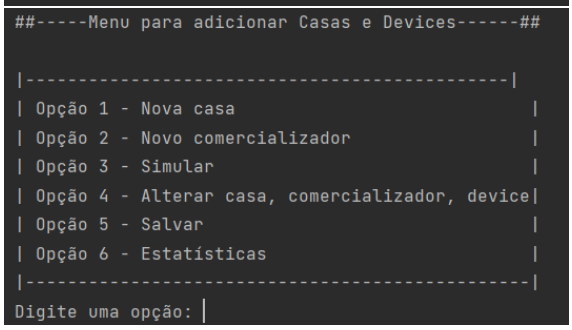
O projeto foi feito baseado no conceito de composição, visto que buscamos encapsular a informação com o objectivo de cada entidade não partilhar nenhum tipo de informação, mesmo que tenham os mesmos atributos, tratam-se de objetos diferentes.

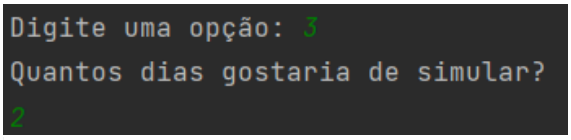
Capítulo 4

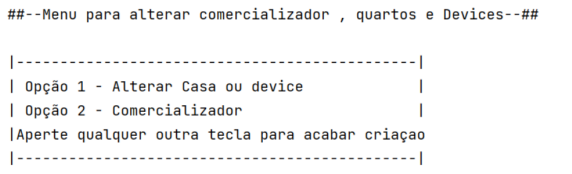
Testes

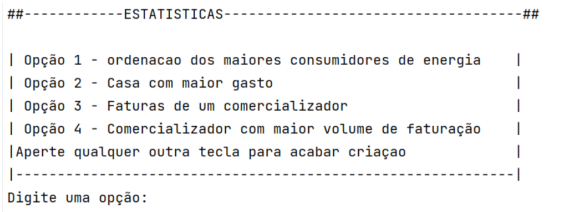












Capítulo 5

Conclusão

Este projeto foi do mais essencial para compreender e consolidar os conhecimentos aprendidos nas aulas teóricas e práticas sobre programação orientada a objetos.

Capítulo 6

Diagrama de Classes

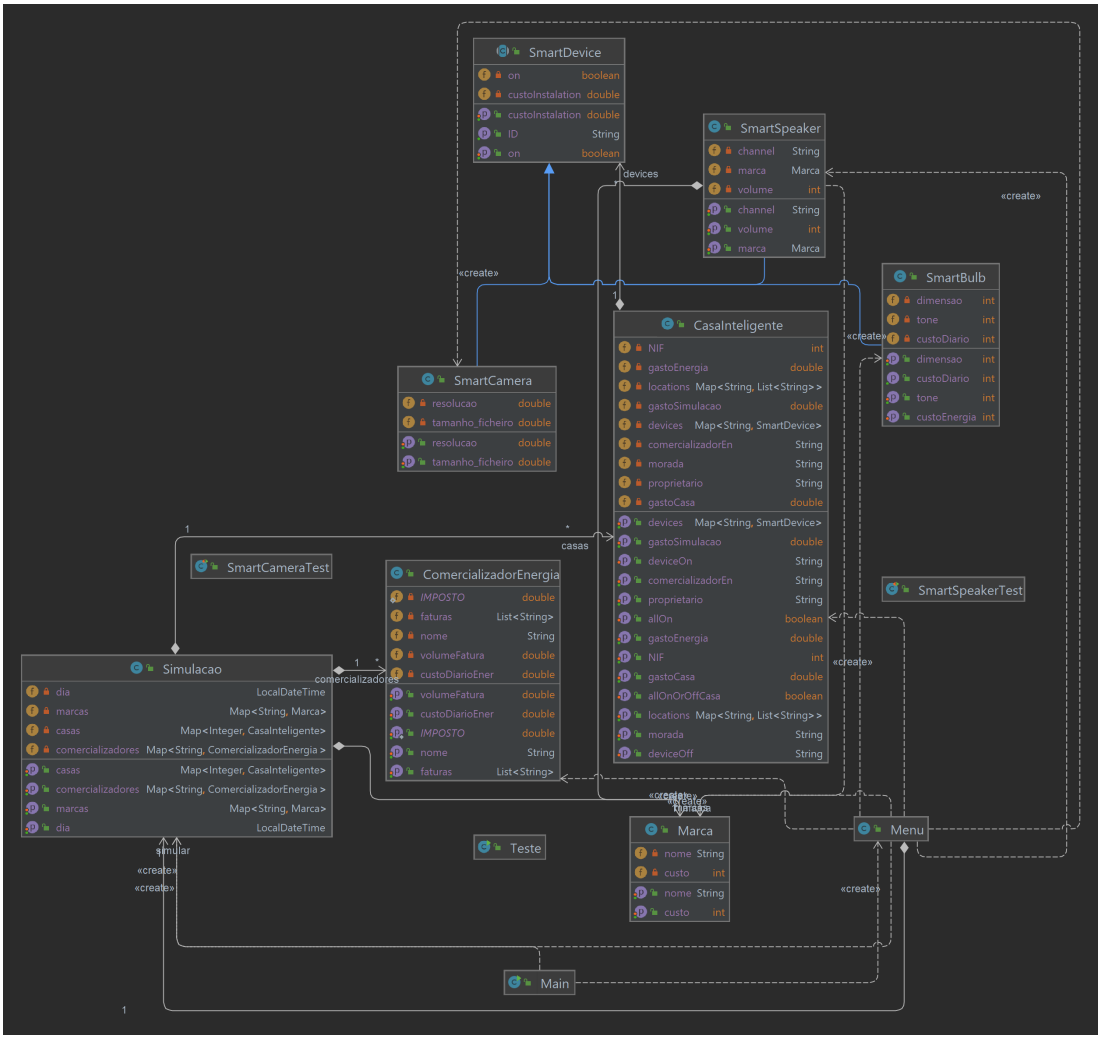


Figura 6.1: Diagrama de classes do programa, gerado pelo *IntelliJ*