Escola de Engenharia

Universidade do Minho – Departamento de Informática

Mestrado Integrado em Engenharia Informática (MIEI)
Licenciatura em Engenharia Informática (LEI)

2021/2022

RELATÓRIO – Projeto Prático

Programação Orientada aos Objetos (POO)

Simulação de uma Casa Inteligente

Grupo 61

A92847, Guilherme Sousa Silva Martins
A94942, Miguel Velho Raposo
A97777, Millena de Freitas Santos



Conteúdo

1 Introdução

Este relatório tem como objetivo descrever os componentes deste trabalho e rever as decisões tomadas para sua implementação. O software foi implementado para um sistema de gestão de faturas, no qual há vários fornecedores de energia, fórmulas que o fornecedor pode escolher para o cálculo do custo da fatura e casas com seus SmartDevices. O utilizador pode escolher carregar um ficheiro de texto com os dados a inserir no sistema ou pode escolher carregar o último objeto salvo.

Após selecionar a forma que obtém os dados, pode escolher entre a versão automatizada e a versão básica. Com esta automatizada, o utilizador pode carregar um ficheiro com os comandos que deseja executar. Enquanto que na versão básica, tudo é feito por escolhas e inserção de dados no terminal.

O programa permite inserir Casa, Fornecedor e SmartDevice, assim como também é possível modificar a fórmula do fornecedor, o fornecedor da casa e ligar/desligar dispositivos. Portanto, decisões tiveram que ser tomadas para responder aos desafios de implementar estas funcionalidades acima descritas e que serão mais elaboradas ao decorrer deste documento.

Como encontra-se no âmbito de POO, foi necessário prestar atenção e medir soluções para que respeitassem a abstração e hierarquia de classes. Isto para que fosse possível obter a Modularidade e o Encapsulamento. Diretrizes muito importantes para que o código seja reutilizável, para que facilite mudanças ou adição de novas funcionalidades.

Nas próximas seções, tudo será esclarecido mais detalhadamente.

2 Arquitetura

Foi utilizado mo delo MVC. No qual há o Model, neste projeto chama-se Entities e agrega todas as classes que são utilizadas no armazenamento das estruturas de dados e tratamento dos dados. A View que é responsável por todas as operações correspondentes à comunicação do usuário com o programa e, por fim, o Controller que serve como um intermediário para a comunicação entre o Model e a View. Portanto, assim, foi possível obter uma melhor gestão do conteúdo.

3 Modos de execução do programa

Há dois modos de execução do programa: básico e avançado. Antes de abordar os pormenores da estrutura da aplicação, é necessário conhecer estas duas funcionalidades possíveis.

3.1 Funcionalidade Básica

Esta é a funcionalidade principal do programa. O utilizador pode inserir Casa, Fornecedor e SmartDevice e também pode alterar o fornecedor da Casa, o estado do dispositivo e a fórmula que o fornecedor utiliza para o cálculo da fatura. Porém, as modificações não são refletidas de imediato no estado do programa.

A solução que obtivemos para que isto pudesse ser implementado foi que, primeiramente, estas alterações são passadas como pedidos pendentes ao programa e guardadas. Estes pedidos são realizados apenas após o utilizador pedir que as faturas sejam geradas. Portanto, todas estas alterações serão apenas refletidas no próximo ciclo de faturas.

Em termos de código, quando o utilizador pede para avançar no tempo, o método responsável é chamado. Este método gera as faturas e, por fim, percorre os comandos pendentes para executá-los.

```
public void moveForward(LocalDate toDate) throws DateAlreadyExistsException, InvalidDateException,
    LocationDoesntExist {
       if(this.fromDate.compareTo(toDate) >= 0){
           //se data de inicio for igual ou maior que data de fim
           throw new InvalidDateException("Data invalida " + toDate + ". Deve ser a partir de " +
               this.fromDate);
       }
       generateInvoices(toDate);
       runCommands();
   }
```

3.2 Funcionalidade Avançada

Nesta funcionalidade avançada e opcional ao utilizador, um ficheiro de texto pode ser carregado com os comandos que o utilizador deseja realizar. Só há 4 comandos possíveis:

3.2.1 setMode

Este comando é para ligar ou desligar um dispositivo de uma casa. Deve ser escrito da seguinte forma:

data setMode idcasa idsmartdevice modo

No qual o modo deve corresponder a ON ou OFF. Por exemplo:

2022-05-20 setMode 1234 lampada1 ON

Foi decidido que, ao contrário da funcionalidade básica, nesta avançada este comando de ligar/desligar um dispositivo é feito imediatamente quando é chamado. Portanto, as faturas passam a ter dispositivos com seus próprios estados diferentes no decorrer da faturação. Por exemplo, numa fatura o mesmo dispositivo pode ter permanecido ligado durante 4 dias e desligado durante os outros dias. Isto não acontece na funcionalidade básica, visto que as alterações ficam pendentes para serem executadas após a geração das faturas.

Para que isto pudesse ser feito, foi necessário a implementação de registos de mudanças do estado do dispositivo. Cada vez que é feita uma mudança: ligar ou desligar, esta é registada. Assim, há a possibilidade de efetuar o cálculo do consumo daquele dispositivo no intervalo de tempo da fatura.

3.2.2 changeSupplier

Este comando é para mudar o fornecedor de uma casa. Assim como na funcionalidade básica, decidimos que ainda faz sentido ter esta mudança implementada após a geração das faturas, para que a casa cumpra um certo prazo de aviso ao fornecedor com o qual possui um contrato de energia.

Deve ser escrito da seguinte forma:

data changeSupplier idcasa idfornecedor

2022-05-20 changeSupplier 1234 EDP

Portanto, no próximo ciclo de faturação, esta casa terá como fornecedor a EDP.

3.2.3 changeFormula

Este comando é para mudar a fórmula que o fornecedor calcula o custo para a fatura. Assim como na funcionalidade básica, decidimos que ainda faz sentido ter esta mudança implementada após a geração das faturas, de tal forma que o fornecedor possa avisar todos seus clientes que tal mudança ocorrerá. Há 6 fórmulas previamente guardadas no programa. O fornecedor pode ter qualquer uma destas. Caso seja necessário uma nova fórmula, é preciso implementar o código. O que também torna-se fácil devido à modularidade. Isto será

melhor explicado no decorrer do relatório.

Deve ser escrito da seguinte forma:

data changeFormula idfornecedor nomeFormula

2022-05-20 changeFormula EDP Formula1

3.2.4 generateInvoices

Este comando faz com que todas as faturas sejam geradas e guardadas. E, em seguida, realiza todos os comandos pendentes como mudança de fornecedor ou mudança de fórmula do fornecedor. A data passada é a utilizada para data final da fatura.

data generateInvoices

2022-05-30 generateInvoices

4 Classes

4.1 SmartDevice

```
private String id;
private double consumptionBase;
private NavigableMap<LocalDate, Integer> logs;
```

Esta classe é uma classe abstrata que será superclasse de todas a classes de dispositivos da aplicação: SmartCamera, SmartBulb e SmartSpeaker. Possui as variáveis de instância em comum para todos os dispositivos. O id, o valor base de consumo do dispositivo e os registos.

Como mencionado anteriormente, para que o dispositivo pudesse ter diversos estados durante um período de faturamento para a funcionalidade avançada, é preciso ter os registos das mudanças de estado guardados. Para isto, foi implementado o NavigableMap com chave para a data a efetuar a mudança e o valor é um inteiro que representa o modo. 1 representa ON e 0 representa OFF. Para a funcionalidade básica, a data será a data inicial da próxima fatura, já para a funcionalidade avançada, a data será a passada pelo comando de texto.

A escolha do NavigableMap foi para a utilização de métodos auxiliares(lowerEntry, higherEntry) para o cálculo do consumo energético do disposito, visto que há registos de mudanças do estado. Era preciso ter uma forma que, os registos utilizados para o cálculo estivessem dentro do intervalo da data inicial e da data final de faturamento. Caso não tenha um registo para a data inicial, deve procurar o exato anterior, pois foi o último estado deste dispositivo. Caso o estado seja ON, utiliza-se esse, caso contrário segue a procura.

Após encontrar um com estado ON, procura sempre o próximo registo, que de acordo com a lógica terá o estado contrário ao anterior OFF. Faz-se então o cálculo dos dias que o dispositivo esteve ligado e guarda. E continua sempre até que atinja a data final da fatura. E, por fim, com a quantidade de dias guardada pode-se calcular o consumo energético total daquele dispositivo.

Quando um dispositivo é criado, se não for passado o modo ligado ou desligado, assume como ligado.

```
public double totalConsumo(LocalDate fromDate, LocalDate toDate){

      LocalDate inicio = fromDate;
      LocalDate intermedio =fromDate;
      int mode;
      int dias = 0;
      double total = consumoEnergetico();

      while(intermedio.compareTo(toDate) < 0) {
          if (this.logs.containsKey(inicio)) {
              mode = this.logs.get(inicio);
          }
          else {
              Map.Entry<LocalDate, Integer> value = this.logs.lowerEntry(inicio);
              mode = value.getValue();

          }

          Map.Entry<LocalDate, Integer> value = this.logs.higherEntry(inicio);

          if(value == null){
              intermedio = toDate;

          }
          else {
              intermedio = value.getKey();

          }

          if (mode==1) { // ON
              dias += ChronoUnit.DAYS.between(inicio,intermedio);
          }

          inicio = intermedio;


      }

      total *= dias;
      return total;

  }
```

4.2 SmartBulb

```
private double dimension;
public static final int WARM = 2;
public static final int NEUTRAL = 1;
public static final int COLD = 0;
private int tone;
```

Esta classe extende a classe SmartDevice, herdando portanto suas variáveis e seus métodos. Acresce portanto a especialidade da lâmpada que dita a tonalidade desta. Podendo ser quente, neutra ou fria.

Caso não seja passada a tonalidade, é assumida portanto a neutra.

```
public double consumoEnergetico(){
        double valor;
        switch(this.tone){
            case COLD:
                valor = super.getConsumptionBase()*0.15;
                break;
            case WARM:
                valor = super.getConsumptionBase()*0.35;
                break;
            default:
                valor = super.getConsumptionBase()*0.22;
                break;
        }
        return super.getConsumptionBase() + valor;
    }
```

A fórmula para consumo basicamente dita que tonalidade fria é mais económica, em seguida neutra e, por fim, a quente.

4.3 SmartSpeaker

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

Esta classe extende a classe SmartDevice, herdando portanto suas variáveis e seus métodos. Acresce portanto o volume, o canal que está a tocar e a marca do dispositivo. O volume máximo estabelecido é 100.

```
public double consumoEnergetico(){
        double consumo;
        if(this.volume <= 0.1*MAX){
            consumo = super.getConsumptionBase();
        }
        else if(this.volume > 0.1*MAX && this.volume <= 0.5*MAX){
                consumo = super.getConsumptionBase()*1.25;
        }
        else{
                consumo = super.getConsumptionBase()*1.5;
        }
        return consumo;
    }
```

O consumo energético é em função do volume. Quanto mais alto, maior o consumo.

4.4 SmartCamera

```
private int resolutionX;
private int resolutionY;
private double fileSize;
```

Esta classe extende a classe SmartDevice, herdando portanto suas variáveis e seus métodos. Acresce a resolução(largura e altura) e o tamanho do ficheiro.

```
public double consumoEnergetico(){
        double resolution = (double) (this.resolutionX/this.resolutionY)*0.01;
        return this.fileSize * super.getConsumptionBase() * resolution;
    }
```

O consumo energético é em função da resolução e do tamanho do ficheiro.

4.5 Casa

```
private String owner;
private String NIF;
private Map<String, SmartDevice> devices; // identificador -> SmartDevice
private Map<String, List<String>> locations; // Espaço -> Lista codigo dos devices
private String supplier;
private double totalInstallationCost;
```

Esta classe representa a Casa, tendo portanto o nome do dono responsável pela fatura, o NIF que será usado como chave de procura pois consegue-se garanter sua unicidade.

Há um mapa de SmartDevices, no qual a chave é o id do dispositivo e o valor é o dispositivo em si. A escolha da implementação de um Map para a estrutura de dados é devido a eficiência e facilidade de encontrar um dispositivo em específico e efetuar mudanças nele como mudar tonalidade, volume, ligar e desliga, entre outras. Assim garantimos este tempo de procura constante.

A estrutura de dados escolhida para organizar os dispositivos por divisão da Casa foi também um Map no qual a chave é o nome da divisão e o valor é uma Lista com os id's dos SmartDevices pertencentes a esta divisão.

Novamente, com o Map tem-se tempo constante na procura de quais dispositivos uma divisão específica possui. Como os dispositivos já são guardados no Map anteriormente mencionado, não houve a necessidade de guardar suas cópias neste, sendo apenas necessário o id para buscar o dispositivo em si no Map devices, e assim, há uma melhor gestão de memória.

Guarda-se o nome do fornecedor sempre que uma Casa é criada. Além do construtor vazio, não há construtor que permita criar casa sem passar a informação do fornecedor. Desta forma, a casa terá fornecedor e também facilita a mudança do fornecedor da casa.

E, por fim, a variável totalInstallationCost serve para facilitar no cálculo da fatura. A cada SmartDevice inserido, há a adição de uma taxa de instalação na primeira fatura após ter sido installado. Portanto, sempre que há a instalação de um dispositivo, este soma o custo de instalação que o fornecedor da casa cobra a esta variável installationCost. Quando a fatura é calculada, soma este valor ao custo total da fatura e esta variável é atualizada novamente para 0.

4.6 Invoice

```
private String id;
private String owner;
private String nif;
private String supplier;
private LocalDate fromDate;
private LocalDate toDate;
private double totalCost;
private double totalConsumption;
```

Esta classe representa a Fatura. Há variáveis que representam o comum de se encontrar numa fatura. Há o NIF, o nome do dono da casa, o nome do fornecedor, o intervalo de datas, o custo total e o consumo energético total.

4.7 Interface - FormulaConsumo

```
public double calculaTotal(double taxes, double consumption, double dailyCost, double nDevices);
public FormulaConsumo clone();
public String toString();
```

Um dos desafios foi a possibilidade de ter fórmulas diferentes entre fornecedores e o fornecedor poder escolher outra fórmula para utilizar. A solução implementa para isto foi esta interface FormulaConsumo e a criação de 6 classes que implementam esta interface. Portanto, cada uma destas 6 classes possui sua fórmula diferente para o método calculaTotal, possui seu clone e toString.

4.8 Fórmulas 1 a 6

Para diversificar os tipos de fórmulas de consumo energético que os vários fornecedores/comercializadores estas classes foram criadas como explicado anteriormente.

As diversas fórmulas variam entre beneficiar quem tem mais dispositivos ligados, beneficiar quem consumiu menos para atingir um objetivo mais ecológico, beneficiar quem consumiu mais para um objetivo mais capitalista, entre outros.

4.9 Fornecedor

```
private String supplier;
private final double dailyCost = 2.5;
private final double taxes = 0.23;
private FormulaConsumo formulaConsumo;
private List<String> invoices;
private double installationCost;
```

Esta classe é para implementar o Fornecedor de energia. Há o nome, o custo fixo por dia para todos os fornecedores, o imposto fixo para todos que equivale ao IVA de 23%, a fórmula que vai utilizar para o cálculo, uma lista com o id das faturas já geradas e o quanto cobra para instalar um dispositivo.

A lista com os id's das faturas foi implementada para facilitar a estatística que pede todas as faturas geradas por um determinado fornecedor.

4.10 Command

```
private LocalDate date;
private String name; //setMode, changeSupplier, changeFormula, setModeLocation
private String command1;
private String command2;
private String command3;
```

Os comandos foram construídos de forma genérica para que pudessem funcionar também para a funcionalidade avançada.

Esta classe foi criada para solucionar um desafio proposto no enunciado deste trabalho. Ligar/Desligar um dispositivo, alterar o fornecedor da casa e alterar a fórmula que um fornecedor utiliza deve ser registado apenas para a próxima fatura. Ou seja, apenas é executado após pedir para gerar um ciclo de faturas.

Para isto, há esta classe Command, na qual guarda o nome do da modifação que o utilizador deseja fazer e os argumentos necessários para tal ser feito. Por exemplo, para ligar/desligar é preciso dizer a casa, o dispositivo e se quer ligar ou desligar. Para mudar o fornecedor é preciso dizer a casa e qual o novo fornecedor. E, por fim, para mudar a fórmula é preciso dizer qual o fornecedor e qual a nova fórmula.

Estes comandos ficam então guardados como pendentes e após invocar o ciclo de faturas, são devidamente realizados.

4.11 Model

```
private Map<String, Casa> casas;
private Map<String, Fornecedor> fornecedores;
private Map<String, Invoice> invoices;
private Map<String, FormulaConsumo> formulas;
private List<Command> commands;
private LocalDate fromDate;
```

Esta classe representa o modelo da aplicação. É onde todas as coleções estão guardadas, na qual o estado é guardado e onde tudo é gerido.

Há uma coleção de casas, fornecedores, faturas e fórmulas. Estas coleções foram implementadas por Maps justamente visando a eficiência no tempo de procura constante, algo que é muito feito durante o funcionamento deste trabalho.

Há a lista de comandos pendentes como mencionado anteriormente. Como esta lista será percorrida para efetuar cada comando, não houve a necessidade de ter um Map para isto.

E, por fim, há a data inicial de cada simulação do ciclo de faturas. Sempre que as faturas são geradas, a data inicial é atualizada para a data final anterior.

4.12 Window, Cores

Estas classes possuem variáveis estáticas e funções *auxiliares* que tornam a experiência entre o utilizador e a interface visualmente mais bonita.

4.13 Menu

A classe Menu é responsável por apresentar as opções disponíveis para o utilizador manipular os dados do Model. Para facilitar a execução dos menus, qualquer valor inserido que não esteja dentro das opções, o programa volta a um menu anterior ou o programa termina.

4.14 Object

Para guardar e carregar o estado em objeto.

```
public static void writeObject(Model model) throws IOException {
    FileOutputStream f = new FileOutputStream(new File("modelObject.txt"));
    ObjectOutputStream o = new ObjectOutputStream(f);
    o.writeObject(model);
    o.close();
}
```

A função `writeObject`

```
public static Model loadObject(String fileName)
throws FileNotFoundException, IOException, ClassNotFoundException{
    FileInputStream fis = new FileInputStream(fileName);
    ObjectInputStream ois = new ObjectInputStream(fis);
    Model m = (Model) ois.readObject();
    ois.close();
    return m;
}
```

A função `loadObject`

5 Exceções

5.0.1 DateAlreadyExistsException

Caso o utilizador queira registar um modo(ligar/desligar) de um dispositivo quando já existe um registo para aquele dia. Pois neste programa considera-se apenas possível trocar estados por dia e não dentro do mesmo dia.

5.0.2 DeviceAlreadyExists

Caso o utilizador queira adicionar um disposito com um id que já existe e está associado a outro dispositivo.

5.0.3 DeviceDoesntExists

Caso o utilizador queira alterar o estado de um dispositivo que não existe.

5.0.4 FormulaDoesntExist

Caso a fórmula escolhida para o fornecedor não exista.

5.0.5 HouseAlreadyExists

Caso o utilizador queira criar uma casa com um nif que já existe e está associado a outra casa.

5.0.6 HouseDoesntExists

Caso a casa pedida não exista.

5.0.7 InvalidDateException

Caso a data seja inválida. Por exemplo, data final anterior a data inicial da fatura.

5.0.8 LocationAlreadyExists

Caso o utilizador queira criar uma divisão já existente.

5.0.9 LocationDoesntExist

Caso não exista a divisão solicitada.

5.0.10 NoInvoicesAvailable

Caso queira efeturar estatísticas, porém não há faturas geradas.

5.0.11 SupplierAlreadyExists

Caso queira criar um fornecedor e já existir um com o nome escolhido.

5.0.12 SupplierDoesntExists

Caso não exista o fornecedor solicitado.

6 Diagrama de Classes

Como há muitas variáveis de instância, muitos métodos e muitas classes, decidimos por mostrar por partes o diagrama primeiro para que pudesse ser bem detalhado e, por fim, o diagrama completo como um todo. De qualquer forma, é aconselhável aplicar o zoom para melhor percepção dos componentes.

6.1 Casa

Figura 1: Casa

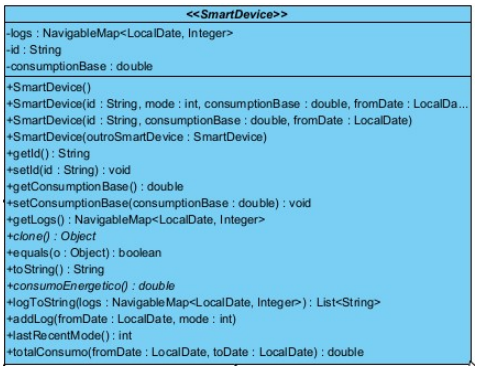

Figura 2: SmartDevice

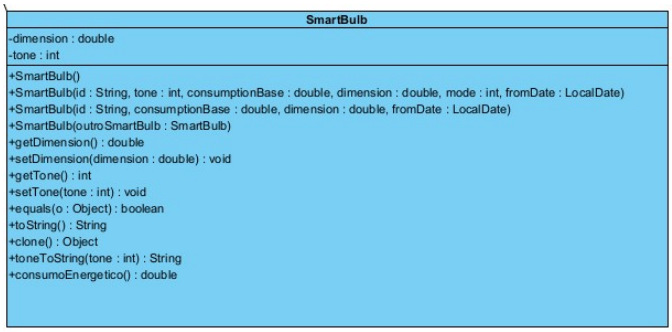

Figura 3: SmartBulb

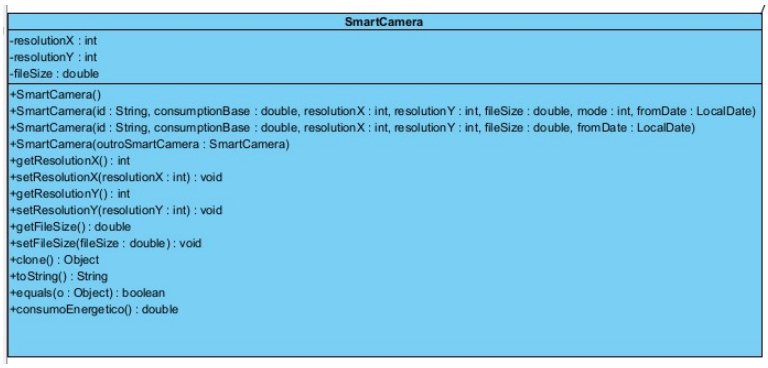

Figura 4: SmartCamera

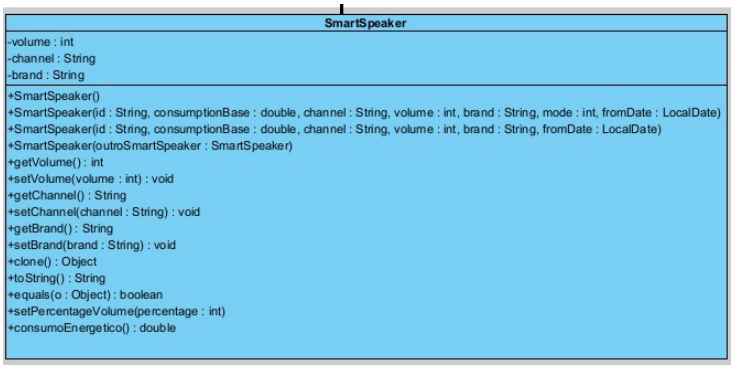

Figura 5: SmartSpeaker

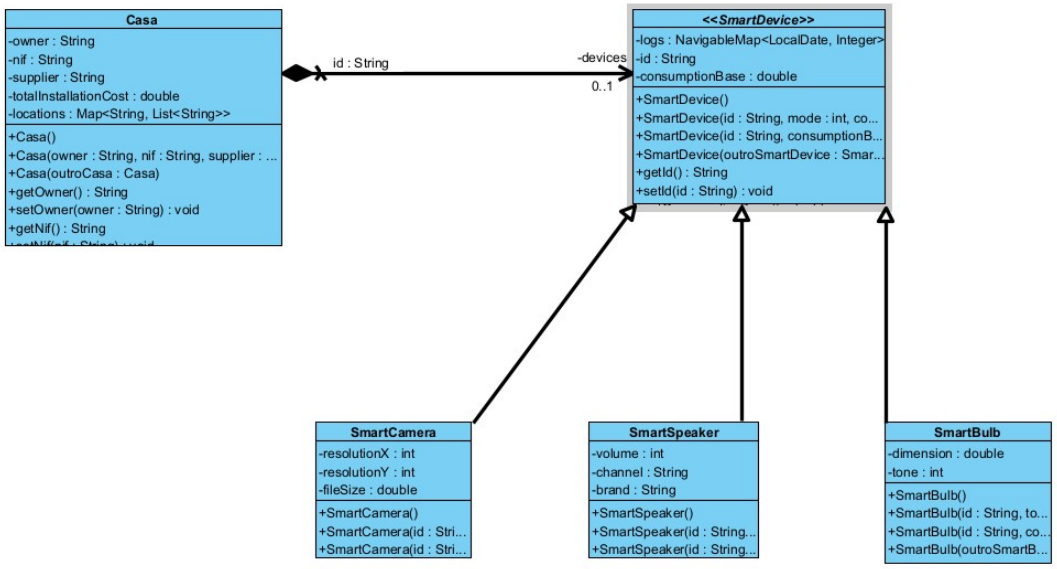

Figura 6: Relação Casa x SmartDevices x (SmartBulb, SmartSpeaker, SmartCamera)

6.2 Fornecedor

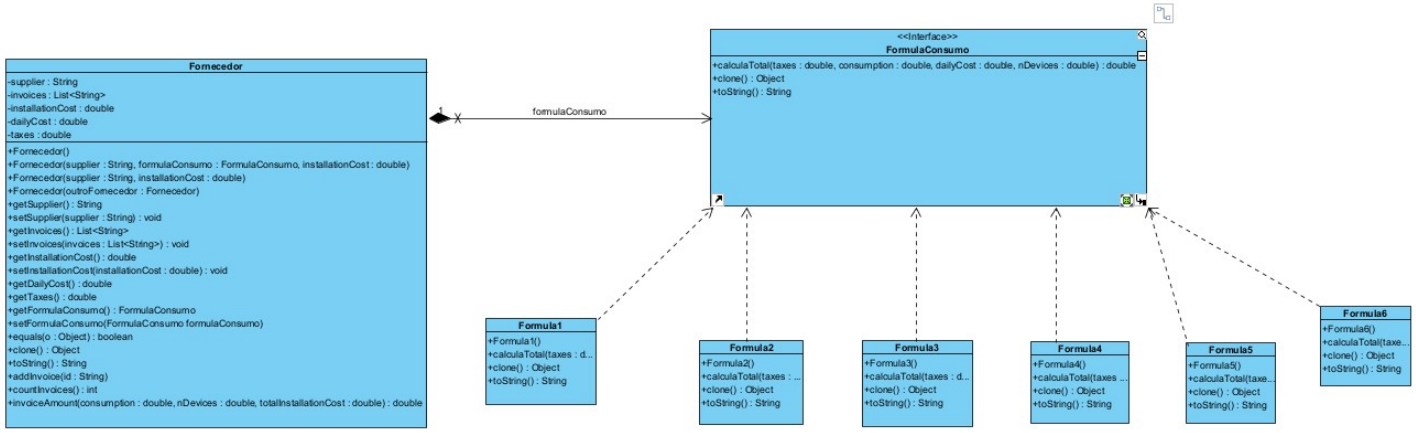

Figura 7: Relação Fornecedor x Formulaconsumo e FormulaConsumo x Formulas

6.3 Invoice

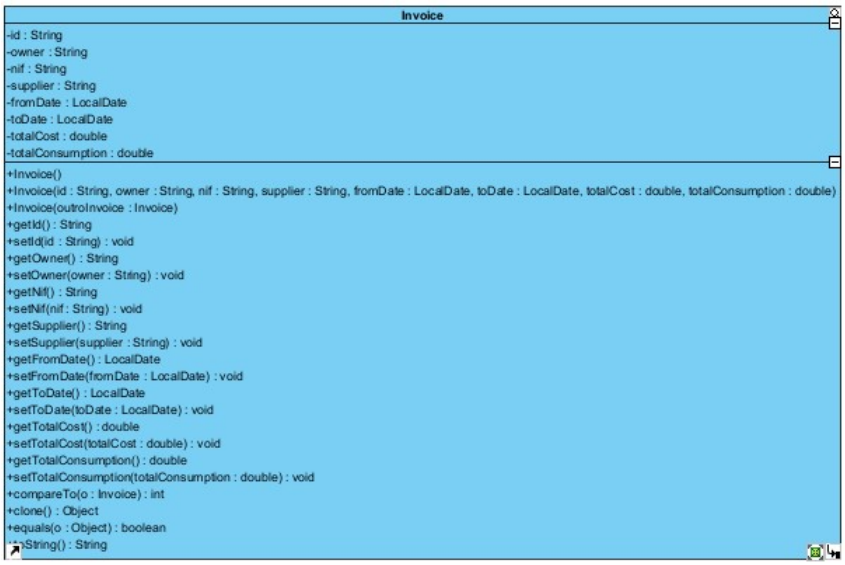

Figura 8: Fatura

6.4 Command

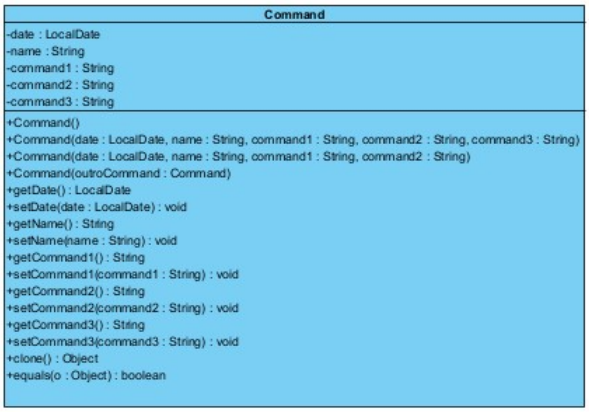

Figura 9: Command

6.5 Model

Figura 10: Model sem Composição

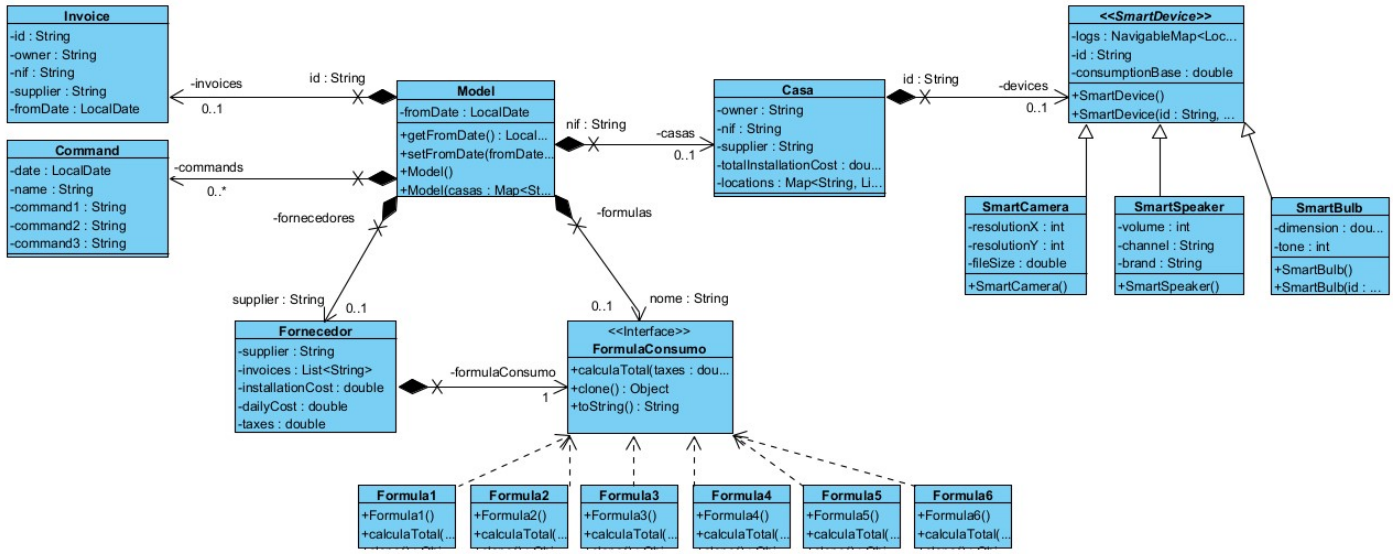

Figura 11: Model Completo

7 Utilização

7.1 Menu inicial

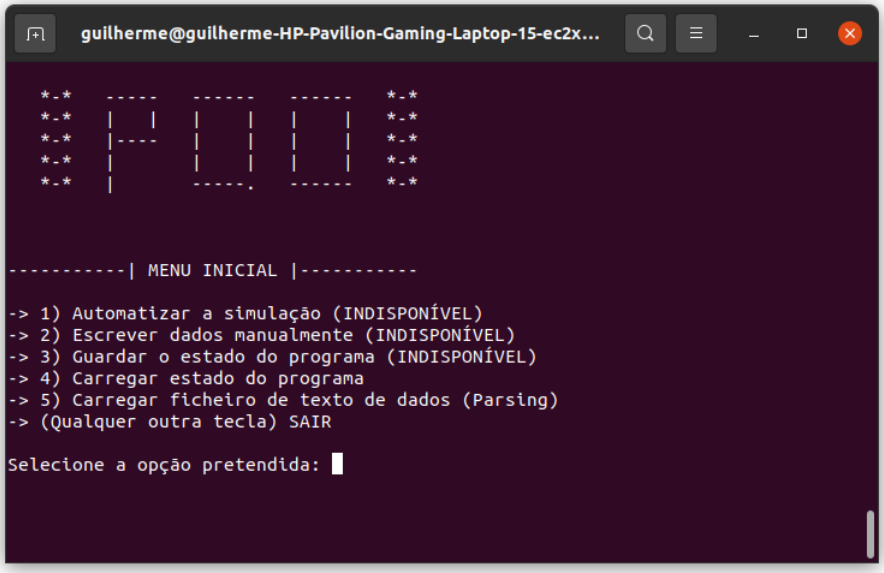

Figura 12: Menu inicial 1

Ao executar o programa, o utilizador deve escolher entre carregar os dados de um ficheiro de texto ou carregar o último estado salvo do programa. Estas opções são obrigatórias para o funcionamento, visto que as demais opções encontram-se indisponíveis. Após carregar os dados de alguma forma, as demais opções tornam-se disponíveis.

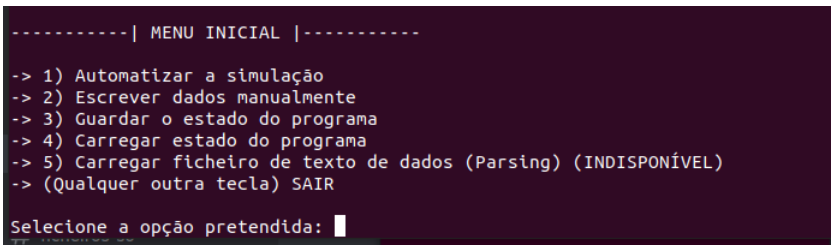

Figura 13: Menu inicial opções disponíveis

O utilizador pode escolher automatizar a simulação ao fornecer o caminho para o ficheiro de texto que contém os comandos a realizar. Também pode escolher a execução do programa com a funcionalidade básica através de menus ou então guardar o estado do programa.

O utilizador eventualmente sempre retorna para este menu até que decida sair do programa.

7.2 Menu - Escrever manualmente

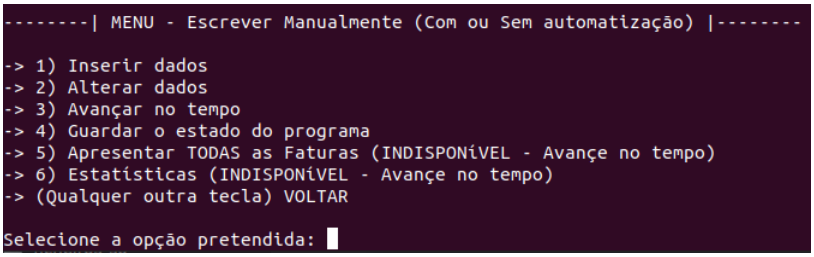

Figura 14: Menu Escrever manualmente

Caso o utilizador escolha executar o programa e efetuar pedidos através dos menus, este menu é exibido.

Pode escolher inserir ou alterar dados, avançar no tempo para gerar um ciclo de faturas, guardar o estado do programa, apresentar todas as faturas ou gerar estatísticas.

Caso o utilizador ainda não tenha avançado no tempo ou ainda não tenha faturas guardadas no estado do programa atual, as opções de apresentar as faturas e efetuar estatísticas apresentam-se indisponíveis.

E, por fim, tem a opção de sair daquele menu.

7.3 Avançar no tempo

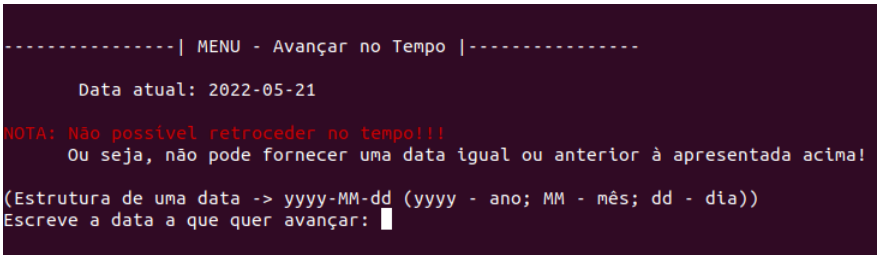

Figura 15: Menu Escrever manualmente

Quando o utilizador escolhe avançar no tempo, é pedido que coloque a data para a qual quer avançar. Caso esta seja menor que a data atual do programa, é devidamente explicado que isto não é possível. Também é demonstrada a estrutura da data que o programa espera receber.

7.4 Inserir

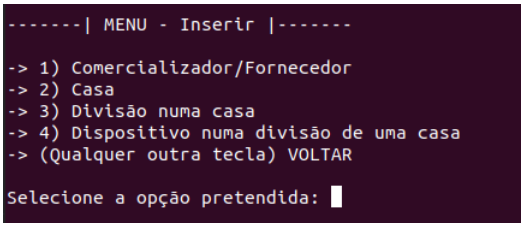

Figura 16: Menu Inserir

Nesta opção de inserir dados, o utilizador pode inserir casa, fornecedor, divisão numa casa ou dispositivo numa divisão de uma casa. E para cada opção é devidamente conduzido para inserir as informações necessárias.

Por exemplo, para Casa há a opção de ver todos os nifs existentes.

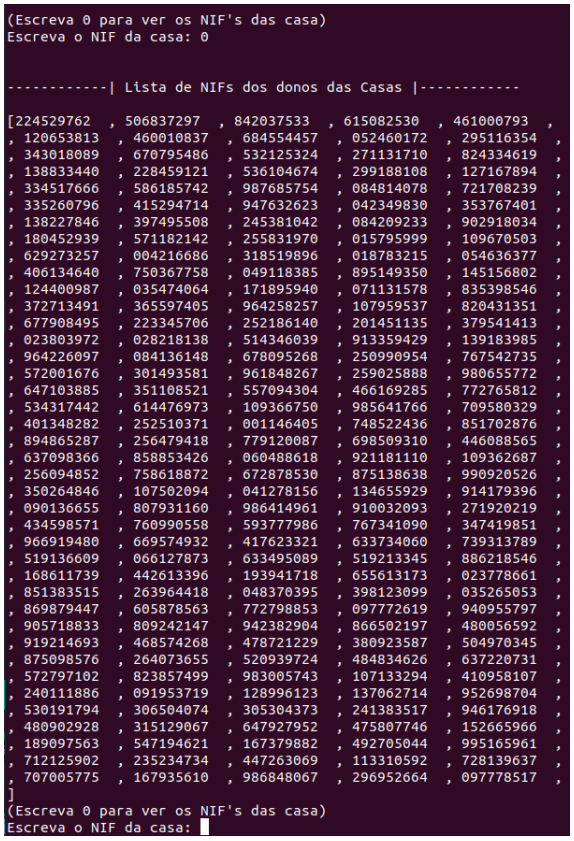

Figura 17: Ver NIFs

7.5 Alterar

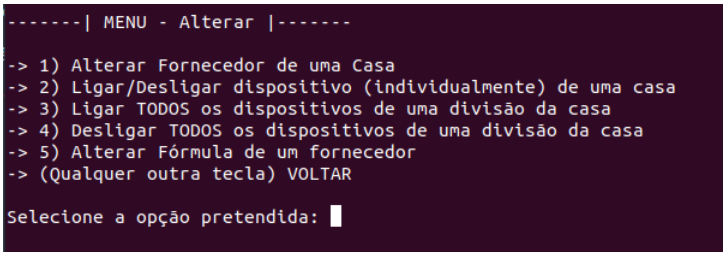

Figura 18: Menu Alterar

Nesta opção de alterar componentes do programa, o utilizador pode alterar o fornecedor de uma casa, ligar/desligar um dispositivo, ligar ou desligar todos os dispositivos de uma divisão de uma casa ou alterar a fórmula de um fornecedor.

Após escolher, é devidamente conduzido a inserir as informações necessárias para concluir a alteração.

Por exemplo, para ligar/desligar um dispositivo, há opção de ver os NIFs e os ids dos dispositivos daquela casa.

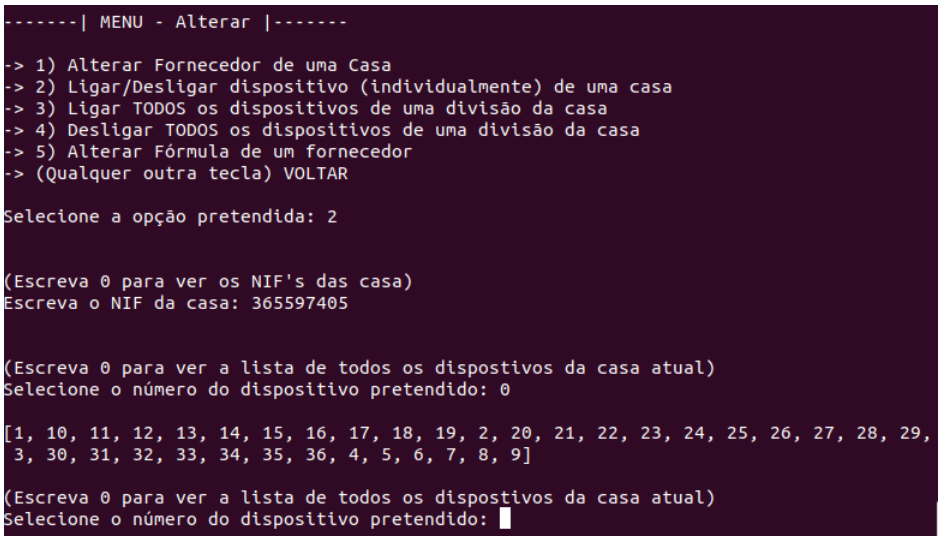

Figura 19: Menu Alterar

De seguida mostra os modos disponíveis:

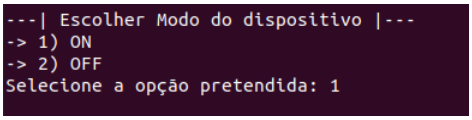

Figura 20: Menu Alterar 2

7.6 Estatísticas

Caso o utilizador queira gerar estatísticas, é exibido este menu com todas as possívies e, após a escolha, o resultado é mostrado.

```
-------| Obter dados para as Estatísticas |-------

-> 1) (NIF da) Casa que mais gastou até à data atual da simulação
-> 2) (Nome do) Comercializador com maior volume de faturação
-> 3) Listar TODAS as faturas emitidas por um comercializador
-> 4) Listar os maiores consumidores de energia durante um período
-> (Qualquer outra tecla) VOLTAR

Selecione a opção pretendida: 1

---| 1. Casa que mais gastou até à data atual da simulação |---

Owner: Carlos Emanuel Leite Machado
NIF: 398123099
Fornecedor: Muon
Custo total: 127 614,20 €
```

Figura 21: Menu Estatísticas

7.7 Automatização

```
(Para teste: db/auto.txt)
Escreva o 'path' do ficheiro com os comandos de automatização: db/auto.txt


************************************************
* Inicializado o parsing automático dos comandos *
************************************************

   Por favor, espere!

*********************
* Parsing CONCLUIDO *
*********************


-----------| MENU INICIAL |-----------

-> 1) Automatizar a simulação
-> 2) Escrever dados manualmente
-> 3) Guardar o estado do programa
-> 4) Carregar estado do programa
-> 5) Carregar ficheiro de texto de dados (Parsing) (INDISPONÍVEL)
-> (Qualquer outra tecla) SAIR

Selecione a opção pretendida:
```

Figura 22: Automatização

Caso o utilizador tenha escolhido efetuar a funcionalidade automatizada, pode gerar as estatísticas e voltar ao menu inicial.

8 Conclusão

Com o projeto realizado, foi possível consolidar todos os tópicos lecionados nas aulas teóricas, quer práticas, da unidade curricular de Programação Orientada aos Objetos. A resolução foi implementada respeitando a Modularidade e o Encapsulamento, o que nos fez perceber melhor como estes conceitos beneficiam o programador, pois desta forma podemos introduzir funcionalidades ou classes ao sistema com mais facilidade. E também torna a aplicação mais fácil e intuitva para utilizadores e programadores.

Com a hierarquia das classes implementadas para os SmartDevices, é trivial poder adicionar novos dispositivos no futuro da aplicação. Da mesma forma que também torna-se simples adicionar novas opções de fórmulas de cálculo do custo da fatura.

Conseguimos implementar o avançar do tempo, as estatísticas e os pedidos pendentes. Também fizemos a nossa versão automatizada segundo o que nos fez sentido.

O modelo MVC foi escolhido pois permite melhorar a organização entre o que é passado do/para o utilizador e o tratamento de dados do programa.

Apesar de admitirmos que o programa criado não possui a organização mais adequada para um modelo MVC e que este poderia ser aperfeiçoado, isso não nos impediu de criarmos um programa completo e coerente.

Portanto, concluímos que este projeto foi realizado com sucesso, apesar das dificuldades em implementar o MVC e resoluções para certos desafios passados no enunciado.

9 *Webgrafia*

https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller