Programação Orientada aos Objetos – Trabalho Prático

LEI - Universidade do Minho

Ano letivo 2021/2022, 2ºsemestre

Bernardo Costa A95052, José Braz A96168, Simão Costa A95176



Índice

1 Introdução

Este projeto consistiu em desenvolver um sistema que armazene informação sobre os ***Smartdevices*** presentes numa ***Smarthouse,*** com recurso ao paradigma de Programação Orientada aos Objetos em *Java.* Durante a execução deste trabalho, a nossa principal dificuldade, que acabou por ser aquela que nos consumiu mais tempo, foi armazenar informação passada previamente no menu principal. No entanto, após a resolução deste problema (que acabou por se revelar de relativamente baixa complexidade), o resto da execução fluiu com normalidade.

Ao longo do relatório entraremos em detalhe em relação ao desenvolvimento das várias componentes da solução implementada para responder ao enunciado que nos foi proposto.

2 Classes

2.1 Diagrama de classes

De seguida apresentamos um diagrama de classes. Isto pode ser interpretado como um breve esquema daquilo que iremos abordar mais aprofundado durante o relatório: a arquitetura e estruturação do projeto.

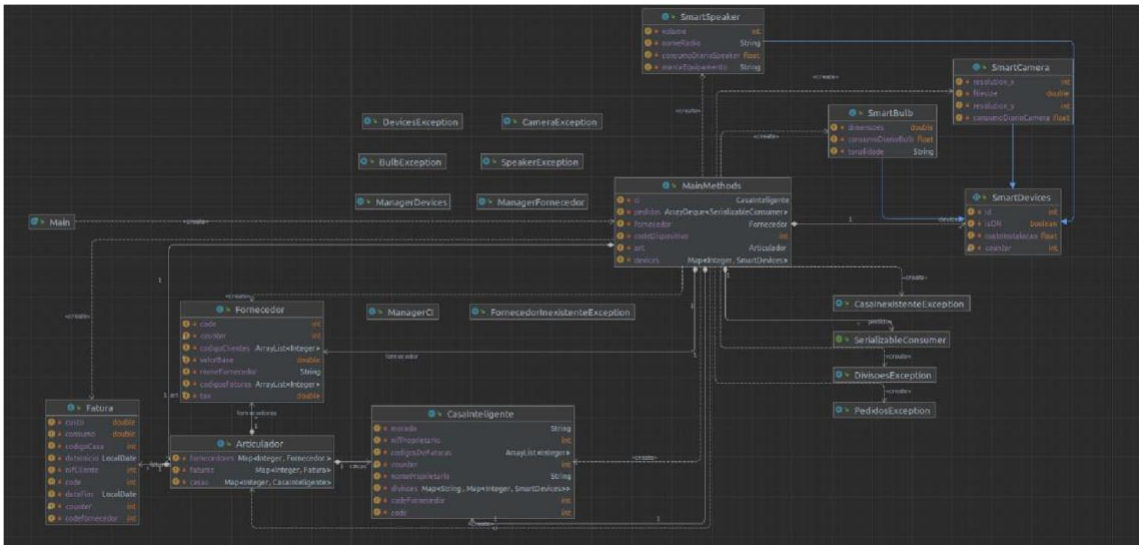

Figura 1-Diagrama de classes

2.2 Main

```
Bem Vindo ao Smart Home System!


Selecione uma das seguintes opções:
1 - Criar um novo Fornecedor,
2 - Criar uma nova Casa Inteligente,
3 - Adicionar um novo dispositivo a Casa Inteligente,
4 - Remover um dispositivo de uma Casa Inteligente,
5 - Remover uma Casa Inteligente,
6 - Alterar configuração de dispositivos,
7 - Alterar o Fornecedor de uma dada Casa Inteligente,
8 - Avançar para determinada data & Gerar Fatura.
9 - Consultar estatísticas de consumo,
10 - Salvar Estado do Programa
11 - Carregar um Estado de Programa
```

Esta é a classe responsável pela apresentação do menu interativo. Este é apresentado na *printscreen* acima e é importante para fases mais avançadas como criação de estatísticas ou avanço do tempo, mas também para as funcionalidades mais elementares como, a título de exemplo, criação de uma casa. A importância desta classe reside na interação com o utilizador e efeitos de debug.

2.3 Articulador

```
private Map<Integer, CasaInteligente> casas;
private Map<Integer, Fornecedor> fornecedores;
private Map<Integer, Fatura> faturas;
```

Figura 2 – Declaração de variáveis de instância na classe Articulador

A classe Articulador é responsável pela sincronia entre métodos e classes. Deste modo, o nome espelha o funcionamento desta mesma classe. O propósito é relacionar partes distintas do programa (adição e remoção de casas, a título de exemplo) de modo que a apresentação do mesmo seja mais eficiente e coesa.

2.4 MainMethods

```
private CasaInteligente ci;
private Fornecedor fornecedor;
private Articulador art;
private Map<Integer, SmartDevices> devices;
private ArrayDeque<SerializableConsumer> pedidos;
private int codeDispositivo;
```

Figura 3 – Declaração de variáveis de instância na classe MainMethods

Esta classe (MainMethods) é responsável por implementar e criar os métodos que iremos usar na Main. É, resumidamente, uma forma de fazer com que a Main seja mais compacta. É importante ressalvar também que todos os métodos presentes nesta classe serão, evidentemente, fulcrais para todas as funcionalidades que o menu interativo apresenta. Como tal, os métodos que criam dispositivos através da passagem do parâmetro código, ou que fazem um ranking das casas que mais consomem, entre outros, terão que estar presentes nesta classe devido à sua importância no Menu interativo.

2.5 CasaInteligente

```
private static int counter = 0;
private String nomeProprietario;
private int nifProprietario;
private String morada;
private int code;
private int codeFornecedor;
private ArrayList<Integer> codigosDeFaturas;
private Map<String, Map<Integer, SmartDevices>> divisoes;
```

Figura 4 – Declaração de variáveis de instância na classe CasaInteligente

A classe CasaInteligente é a representação de uma *Smarthouse* e dos atributos que a mesma deve ter. Para além destas variáveis presentes no construtor, definimos métodos importantes como adicionar dispositivos a uma dada divisão e removê-los, adicionar uma divisão e verificar a existência das mesmas ou dos dispositivos que a constituem bem como associar uma fatura a uma dada casa.

2.6 Fatura

```
private static int counter = 0;
private LocalDate dataInicio;
private LocalDate dataFim;
private int codefornecedor;
private int nifCliente;
private double consumo;
private double custo;
private int codigoCasa;
private int code;
```

Figura 5 – Declaração de variáveis de instância na classe Fatura

Nesta classe definimos o que é uma fatura e o que esta contem. Precisaremos desta classe para consultar consumos energéticos ou para listar as faturas por consumidor (uma das Estatísticas sobre o estado do programa).

2.7 SmartDevices

```
private static int counter = 0;
private boolean isON;
private int id;
private float custoInstalacao;
```

Figura 6 – Declaração de variáveis de instância na classe SmartDevices

A classe *SmartDevices* é bastante geral no que toca aos métodos implementados. Aqui realizamos métodos para saber se um determinado dispositivo está ligado ou desligado, que tipo de dispositivo é e o seu custo de instalação. Estes atributos que apresentamos na forma de variáveis (Figura 6) são comuns a todas os *devices*, daí que estejam presentes nesta classe.

2.8 SmartBulb

```
private String tonalidade;
private double dimensoes;
private float consumoDiarioBulb;
```

Figura 7 – Declaração de variáveis de instância na classe SmartBulb

Na classe SmartBulb implementamos métodos relacionados com as lâmpadas, onde acabamos por definir a sua estrutura de acordo, listando cada um dos componentes necessários à criação da mesma. Ou seja, definir a tonalidade das mesmas, as dimensões, o seu ID, o seu consumo diário, etc.

2.9 SmartSpeaker

```
private int volume;
private String nomeRadio;
private String marcaEquipamento;
private float consumoDiarioSpeaker;
```

Figura 8 – Declaração de variáveis de instância na classe SmartSpeaker

Na classe SmartSpeaker implementamos métodos relativos a dispositivos de áudio espalhados pela Casa Inteligente, indicando algumas das caraterísticas para a sua criação e manipulação. Algumas são próprias do mesmo, como o nome da rádio que está a tocar, sendo outras mais gerais e contribuindo para as estatísticas (consumo diário, por exemplo).

2.10 SmartCamera

```
private int resolution_x;
private int resolution_y;
private double filesize;
private float consumoDiarioCamera;
```

Figura 9 – Declaração de variáveis de instância na classe SmartCamera

Nesta classe aplica-se a estratégia das duas anteriores. É dedicada a SmartCameras e os métodos são maioritariamente *getters* e *setters* das variáveis presentes no construtor de uma SmartCamera.

2.11 Fornecedor

```
private static int counter = 0;
private String nomeFornecedor;
private int code;
private final double valorBase = 0.15;
private final double tax = 1.06;
private ArrayList<Integer> codigosFaturas;
private ArrayList<Integer> codigoClientes;
```

Figura 10 – Declaração de variáveis de instância na classe Fornecedor

Nesta classe definimos métodos e variáveis que estarão associadas a fornecedores de energia. Esta classe será essencial para as faturas e cálculos de energia gasta. Nesta classe também calculamos um dado estatístico

importante: o preço total de energia diária consumida por cliente. Este dado é obtido através da seguinte fórmula

(valorBase * casa.energiaTotalDiariaCasa() * tax) * 0.9

3 Estrutura do Projeto

Durante o projeto, fomo-nos baseando na metodologia da Programação Orientada aos Objetos. Baseamo-nos parcialmente na estratégia *Model View Controller*, porém, quando esta foi abordada em contexto de sala de aula já o nosso trabalho estava bastante adiantado, pelo que seria inviável mudar totalmente a abordagem nessa fase. Porém, classes como o Articulador seguem estratégias abordadas na Aula Prática nº17, nomeadamente o Padrão Facade. No entanto, todo o projeto está de acordo com as aprendizagens em contexto de aula (nomeadamente no que diz respeito ao paradigma). Talvez a única exceção seja a definição e método de escolha das resoluções de uma câmara. No entanto, isso deve-se ao facto de querermos aumentar o realismo do nosso programa, dado que existem resoluções pré-definidas no mercado e foram essas que usamos. No entanto é possível observar camadas diferentes como por exemplo a Main que é direcionada à interação com o utilizador ou o Articulador que é direcionado controlo de fluxo e articulação do programa.

4 Conclusão

Globalmente, o grupo considera que o trabalho foi bem conseguido, visto que atingimos praticamente todos os objetivos que nos foram propostos.

Com o desenvolvimento deste projeto, ao longo do semestre, sentimos que nos foi permitido aprofundar todos os aspetos e os conhecimentos adquiridos face à linguagem de programação Java. Seguimos as regras de encapsulamento e do paradigma de Programação Orientada aos Objetos, desenvolvendo um código que permite que o projeto cresça de forma controlada. O tema foi do nosso agrado por se aproximar de uma realidade que nos é bastante próxima. Em suma, julgamos que este projeto serviu como um bom complemento para consolidar os conhecimentos já adquiridos em aula.