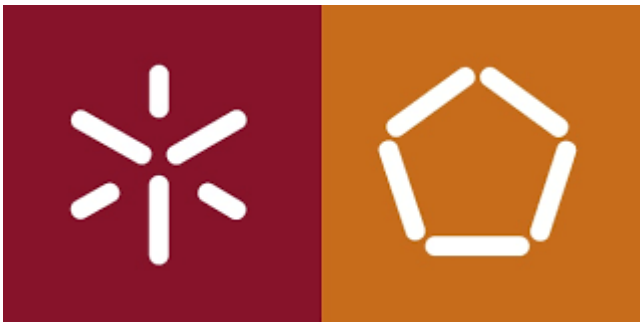


UNIVERSIDADE DO MINHO

LICENCIATURA EM ENGENHARIA INFORMÁTICA

PROJETO DE
PROGRAMAÇÃO ORIENTADA AOS OBJETOS

GRUPO

José Carvalho (a94913)

Miguel Silva (a87031)

Ana Rita Poças (a97284)

ÍNDICE

1. Introdução

O projeto que nos foi proposto tem como objetivo a implementação de um programa que consiga monitorizar e registar a informação sobre o consumo energético das habitações de uma comunidade. É suposto neste trabalho prático, termos um produto resultante respeitador das as normas da programação orientada aos objetos (encapsulamento, modularidade e as regras abordadas ao longo da unidade curricular) assim como um conjunto de requisitos indicados pela equipa docente. Neste documento iremos explicar o nosso raciocínio e a forma como resolvemos os problemas propostos e encontrados durante as nossas implementações.

2. Classes

De forma a enfrentarmos os desafios propostos no enunciado do projeto, decidimos estruturar o nosso programa e classes, que abordaremos individualmente em detalhe, de acordo com as entidades com que trabalhamos.

Criamos portanto uma classe **Main** onde é inicializado o programa, uma classe **App**, que é a nossa Facade ou Modelo (conceito explicado com maior detalhe posteriormente) e também criamos classes que definem individualmente um **Comerciante**, uma **Fatura**, uma casa (**House**), uma **Pessoa** que é proprietária da casa e os dispositivos inteligentes (**SmartDevices**) que estão localizados nas casas.

De uma forma geral o nosso programa funciona à base de agregação. Optamos assim de forma a termos situações em que por exemplo, ao mudar a fórmula de um comerciante, evitamos o trabalho de ir a todas as casas uma a uma a ver se o comerciante é o mesmo e mudar a fórmula, pois, assim n casas que têm o mesmo comerciante, todas elas irão partilhar o apontador do fornecedor evitando o custo computacional já referido (*lookups* mais rápidos e eficientes). Isto também se aplica a quando queremos mudar o estado de um certo dispositivo.

Em contrapartida, achamos por bem usar a composição nas faturas dos comerciantes, de forma a garantir a 100% que nenhuma fatura que já tenha sido emitida seja alterada. Usamos também a composição nas relações da App com partes relativas a Input Output.

Ou seja, embora utilizemos por norma agregação, não nos prendemos a um método e analisamos sempre em que situações é benéfico agregação ou composição consoante as nuances das partes do projeto que íamos desenvolvendo.

2.1 Main

Esta classe é o ponto de partida no programa. Resumidamente a classe Main só invoca determinados métodos dos controladores de forma a alterar / consultar a informação respetiva que o controlador tem acesso.

É importante referir que esta classe é a que faz a ligação de todos os controladores e é a que inicializa os mesmos.

2.2 App

Esta é a classe principal do nosso programa. A app, segundo a metologia MVC, pode ser vista como o nosso Model, e quando algo exterior quer consultar alguma coisa da app, a app trabalha sempre com cópias. Ou seja, a app no fundo trabalha sob a forma de composição de forma a garantirmos encapsulamento na relação do interior para o exterior.

Nesta classe usamos como recurso os Maps para guardar a informação. No mapeamento dos fornecedores é feita a associação <String,Comerciante>, em que a string é o nome do respetivo comerciante. Para os devices fazemos a associação <String, SmartDevices> em que aqui a String corresponde ao id do device. Nas casas associamos <String,House> em que a String corresponde à localidade da respetiva casa. Já as pessoas são da forma <Integer,Pessoa> em que o Integer corresponde ao Nif da pessoa.

É nesta classe que são criadas todas as instâncias de Smart Devices, Comerciantes, Houses e etc. e também é nesta classe que encontramos os principais métodos das queries, bem como os métodos que possibilitam as mudanças de informação dos dados das diversas partes do projeto, o que faz sentido pois como mencionado é a partir da classe App que fazemos a ligação entre as múltiplas fatias do bolo que compõem o nosso programa.

2.3 Comerciante

Esta classe pretende definir um Comercializador/Fornecedor de energia, e para tal tem como variáveis de instância o seu nome, fórmula de cálculo, assim como um Map em que a chave é uma data e o valor é uma lista de faturas, para representar as faturas emitidas por si ao longo do tempo.

Nesta classe estão codificados alguns dos métodos auxiliares à resolução das queries, como por exemplo *getLucro* e *getFaturasDoComerciante*.

Aproveitamos para salientar que nas seguintes classes devido a estas representarem tipos de entidades no nosso programa, cada módulo conta com métodos construtores, getters e setters, toString, clone e equals.

2.4 Pessoa

Esta classe contém as variáveis de instância nome e NIF e tem como objetivo definir uma entidade Pessoa, que posteriormente irá definir o proprietário de uma casa e também o cliente associado a uma fatura.

2.5 Fatura

Esta classe contém a informação relativa a uma fatura, nomeadamente o cliente a que se destina (sob a forma de agregação), o local da casa, o consumo, o preço, a data de emissão e o imposto associado.

Esta classe só é usada pela classe Comerciantes e como já foi referido, é usada sob a forma de composição para garantir que a informação não seja alterada.

Usamos agregação na relação **Fatura** -> **Pessoa**, para um caso invulgar de a pessoa querer alterar o próprio nome.

2.6 House

A classe House representa a estrutura de uma casa. Uma casa tem associado a si um proprietário, um local que é representado como uma string, um fornecedor e 2 maps, um que representa as divisões e outro que representa os Smart Devices. Usamos agregação nas relações **House** -> **Pessoa**, **House** -> Comerciante e para **House** -> **SmartDevices**, de forma a que se alguma informação associada a estes casos seja alterada, a informação também é alterada diretamente nos devidos apontadores das casas.

2.7 SmartDevices

De modo a estruturar os três tipos de dispositivos com que podemos trabalhar, decidimos criar uma super classe abstrata chamada **SmartDevices** e três subclasses que herdam métodos da superclasse chamadas **SmartBulb**, **SmartCamera** e **SmartSpeaker**. Cada uma destas subclasses contém os métodos específicos que definem um aparelho do tipo escolhido. Estas classes contam com os métodos genéricos, construtores, getters e setters presentes em todas as classes que definimos, sendo que devido a serem subclasses, têm de utilizar o método super() para aceder a partes importantes da superclasse ao serem feitos os construtores, cada subclasse tem construtores diferentes com campos diferentes, de forma a distinguir os tipos de dispositivos. É importante notar também que, cada um dos três tipos de dispositivo tem uma forma diferente de calcular o seu consumo diário e como tal, os métodos de cálculo de consumo diário são portanto distintos em cada uma das subclasses.

3. Classes de Teste

Na pasta destinada a testes, colocamos todos os testes relativos às diversas classes principais. Tivemos recurso ao JUNIT e tentamo-nos focar em certos aspetos mais relevantes, nomeadamente testes relativos a agregação ou composição, dependendo das classes, e, também, outros métodos importantes, tal como o *avancadias* do **App**, etc. Para os testes tanto desta última classe como de outras assentamos muito no uso dos getters e setters para averiguar se a informação estava a ser atualizada tal como deveria.

4. Model View e controller

De modo a simplificar o programa, nós criamos 5 partes distintas de MVC. Uma para os SmartDevices, uma para as casas, uma para os Fornecedores, uma para as pessoas e uma para as queries. Desta forma, não temos métodos de views misturados, embora haja métodos muitos similares entre eles. Por exemplo, o controlador dos fornecedores trabalha só com a view dos fornecedores e assim sucessivamente.

Apesar disso, no controlador das casas, achamos por bem, de modo a cumprir com requisitos e com metas pessoais, meter a opção de acrescentar diretamente múltiplos devices em um certo número de casas, então o controlador das casas têm uso auxiliar do controlador dos SmartDevices de modo a conseguir então de facto realizar esta operação.

Agora passaremos a explicar alguns aspetos importantes relativos à parte de input/output. Primeiramente só é possível alterar dados, isto é por exemplo mudar a fórmula de um fornecedor ou mudar o estado de vários dispositivos, após avançar no tempo. No entanto, é possível consultar a informação em qualquer instante. Antes do programa carregar os dados, é perguntado ao utilizador se quer carregar um ficheiro em binário com dados, ou se quer utilizar o ficheiro texto ‘log.txt’ disponibilizado pelos professores. No final do programa, o utilizador tem que inserir um nome para o ficheiro em que o programa irá guardar os dados sob o formato binário e futuramente este ficheiro pode servir para carregamento de dados.

Relativamente a interface, é possível consultar um menu, mas primeiramente o nosso programa quer saber se o utilizador quer adicionar um device / casa / pessoa / fornecedor ao programa, ou consultar / alterar algum fator dos mesmos. Criamos 9 opções, 4 para adicionar e outras 4 para a consulta ou alteração de dados e 1 para as queries. Nas 8 primeiras, o programa a partir delas torna-se intuitivo, mas na opção das queries ou ‘Q’ é a opção em que se pode avançar no tempo. Esta opção também contém outras operações de consulta de informação e possibilita mudar o imposto se o utilizador assim pretender.

Apresentaremos agora algumas imagens que demonstram o nosso programa em funcionamento:

- Parte inicial do programa em que utiliza o ficheiro log.txt:

```
Deseja utilizar os dados iniciais?
Y - Sim
Outros inputs - Não
N
Qual o imposto inicial?
23
```

- Avançar no tempo:

```
Diga qual a próxima operação. Introduza a letra 'M' ou 'm' ou 'H' ou 'h' para aceder a um menu de comandos
Q
Pretende consultar estatísticas sobre o estado do programa ou mudar o fornecedor?
1 - Realizar queries
2 - Mudar imposto
1
O que pretende consultar?
1 - Qual o comercializador com maior volume de faturação
2 - A casa que gastou mais naquele período
3 - Dar uma ordenação dos maiores consumidores de energia num período a determinar
4 - Desejo avançar dias
5 - Nada
4
Insira um número
31
Diga qual a próxima operação. Introduza a letra 'M' ou 'm' ou 'H' ou 'h' para aceder a um menu de comandos
```

- Consultar as Faturas emitidas pelo fornecedor “EDP Comercial”:

```
Diga qual a próxima operação. Introduza a letra 'M' ou 'm' ou 'H' ou 'h' para aceder a um menu de comandos
F
Consultar informação ou altera-la nos fornecedores?
1 - Consultar
2 - Alterar
3 - Nada
1
Algum critério de filtragem?
1 - Não
2 - Têm mais que um certo número inteiro de faturas emitidas
3 - Têm volume de faturação maior ou igual a um certo número real
4 - Segue uma determinada Formula
5 - Têm um certo nome
5
Qual o nome do Comerciante?
EDP Comercial
O que quer consultar?
1 - Ocorrências das Formulas
2 - Faturas do fornecedor
3 - Número de faturas
4 - Lucro do fornecedor
5 - Número de Fornecedores que respeitam um certo predicado
2
```

- Mudar o fornecedor para “EDP Comercial” a todas a casas cujo fornecedor é a “Endesa”:

```
Diga qual a próxima operação. Introduza a letra 'M' ou 'm' ou 'H' ou 'h' para aceder a um menu de comandos
C
Consultar ou mudar dados?
1 - Consultar
2 - Mudar
3 - Nada
2
O que deseja consultar/alterar?
1 - Acrescentar n devices
2 - Acrescentar n divisões
3 - Mover n devices para uma divisão (Só funciona para 1 casa)
4 - Ligar ou desligar devices seguindo um certo predicado
5 - Mudar fornecedor
6 - Mudar Proprietário
5
Algum o critério de filtragem para a casa?
```

```
Algum o critério de filtragem para a casa?
1 - Não
2 - Localidade
3 - Têm um certa divisão
4 - Têm um certo número de divisões
5 - Têm um certo device
6 - Têm um certo número de devices
7 - Têm um certo proprietario
8 - Têm um certo fornecedor
8
Insira uma string para nome do fornecedor
Endesa
Insira uma string para nome do fornecedor
EDP Comercial
```

- Número de casas neste momento que têm como fornecedor a “Endesa”:

```
Diga qual a próxima operação. Introduza a letra 'M' ou 'm' ou 'H' ou 'h' para aceder a um menu de comandos
C
Consultar ou mudar dados?
1 - Consultar
2 - Mudar
3 - Nada
1
Algum o critério de filtragem para a casa?
1 - Não
2 - Localidade
3 - Têm um certa divisão
4 - Têm um certo número de divisões
5 - Têm um certo device
6 - Têm um certo número de devices
7 - Têm um certo proprietario
8 - Têm um certo fornecedor
8
Insira uma string para nome do fornecedor
Endesa
O que deseja consultar?
```

```
Insira uma string para nome do fornecedor
Endesa
O que deseja consultar?
1 - Comerciante
2 - Proprietário
3 - Lista de devices seguindo um certo predicado
4 - Lista de divisões
5 - Lista de localidades
6 - Número de casas
6
0
```

- Comerciante com maior volume de faturação:

```
Diga qual a próxima operação. Introduza a letra 'M' ou 'm' ou 'H' ou 'h' para aceder a um menu de comandos
Q
Pretende consultar estatísticas sobre o estado do programa ou mudar o fornecedor?
1 - Realizar queries
2 - Mudar imposto
1
O que pretende consultar?
1 - Qual o comercializador com maior volume de faturação
2 - A casa que gastou mais naquele período
3 - Dar uma ordenação dos maiores consumidores de energia num período a determinar
4 - Desejo avançar dias
5 - Nada
1
Comerciante:
Nome: Iberdrola
```

- Se avançarmos outra vez no tempo em 60 dias o novo comerciante com maior volume de faturação é:

```
Diga qual a próxima operação. Introduza a letra 'M' ou 'm' ou 'H' ou 'h' para aceder a um menu de comandos
Q
Pretende consultar estatísticas sobre o estado do programa ou mudar o fornecedor?
1 - Realizar queries
2 - Mudar imposto
1
O que pretende consultar?
1 - Qual o comercializador com maior volume de faturação
2 - A casa que gastou mais naquele período
3 - Dar uma ordenação dos maiores consumidores de energia num período a determinar
4 - Desejo avançar dias
5 - Nada
1
Comerciante:
Nome: EDP Comercial
```

- Número total de devices existentes na aplicação:

```
0
1 - Consultar Dados
2 - Alterar Dados
Outros Inputs - Nada
1
Deseja realizar uma consulta mais espefica ou deseja apenas ver o total de SmartDevices?
1: Consulta Específica
2: Total de SmartDevices que obedecem a um dado predicado
Outros Inputs - Nada
2
Algum critério de filtragem?
1 - Não
2 - Consumo Superior a um dado valor
3 - Consumo Inferiores a um dado valor
4 - Ligados
5 - Desligados
6 - SmartCameras
7 - SmartBulbs
8 - SmartSpeakers
Outros Inputs - Nada
1
6555
Diga qual a próxima operação. Introduza a letra 'M' ou 'm' ou 'H' ou 'h' para aceder a um menu de comandos
```

- Nifs de todas as pessoas existentes na aplicação:

```
Diga qual a próxima operação. Introduza a letra 'M' ou 'm' ou 'H' ou 'h' para aceder a um menu de comandos
P
Pretende consultar  informação da Pessoa?
1 - Sim
2 - Não
1
Pretende aplicar algum critério de filtragem?
1 - Não pretendo
2 - A pessoa tem um NIF superior a um certo número inteiro
3 - A pessoa tem um certo nome
1
O que pretende consultar??
1 - O nome da pessoa
2 - O NIF da pessoa
3 - Número de Pesosas que respeitam um dado predicado
4 - Nada
2
```

- Guardar em ficheiro binário:

```
Diga qual a próxima operação. Introduza a letra 'M' ou 'm' ou 'H' ou 'h' para aceder a um menu de comandos
Sair
Insira o nome do ficheiro onde guardará os resultados os binário
save.bin
```

- Ir buscar informação a um ficheiro binário:

```
Deseja utilizar os dados iniciais?
Y - Sim
Outros inputs - Não
Y
Insira o nome do ficheiro onde armazenou os resultados
save.bin
```

As restantes opções são muito semelhantes às que apresentamos aqui.

5. Exceptions

As nossas exceções, embora importantes para o nosso projeto, não foram algo que planeamos previamente antes de codificar as classes. Resumidamente, à medida que o projeto foi avançando nós tivemos em conta se faria sentido criar uma excepção, ou aplicar uma já existente. Assim sendo, as nossas exceções focam-se em aspetos como fazer sets de objetos nulls, ou números que excedem um determinado intervalo e por último querer ir buscar uma certa informação que não está guardada na ap, ou seja, cenários de erro muito específico que embora possam surgir noutros cenários abordam problemas específicos que tivemos de enfrentar no nosso programa.

6. Interface Formulas

Esta interface é usada para definirmos diversas fórmulas de cálculo. Criamos somente 2 fórmulas de cálculo, e devido ao tempo limitado, optamos por não acrescentar mais fórmulas. Mas se quiséssemos criar mais *n* fórmulas, apenas teríamos de efetuar poucas alterações ao nosso programa, devido ao facto de termos usado este processo de interfaces, tornando o código genérico.

7. Resolução de queries

As resoluções das queries concentram-se todas na classe app.

Algumas queries são intuitivas de se perceber, nomeadamente ir buscar as faturas de um certo comerciante, basicamente, na app consultamos o devido comerciante, e a partir daí é invocado um método da classe comerciante que devolve uma lista de faturas, sob a forma de composição, tendo em consideração que a data de faturação tem de estar entre os dois valores dados como input.

As queries de ir buscar o maior consumidor e os maiores consumidores são muito parecidas, ao ponto em que a query de maior consumidor invoca o método respetivo à dos maiores consumidores e depois vai só buscar o primeiro elemento.

Por último, a query talvez mais importante do programa, é a query que faz avançar os dias. Esta query está associada ao método ***avancadias*** que recebe como input o número de dias a avançar. Este método avança então os dias, e depois vai a cada uma das casas e calcula o consumo diário de cada uma, de seguida vai buscar o proprietário e o comerciante da respetiva casa e cria uma nova fatura com um preço inicial a 0. Por último é invocado o método **addFatura** do devido comerciante em depois é feito o cálculo do preço. A parte do preço é feita internamente do comerciante, pois o preço depende da fórmula de cálculo associado.

8. Conclusão

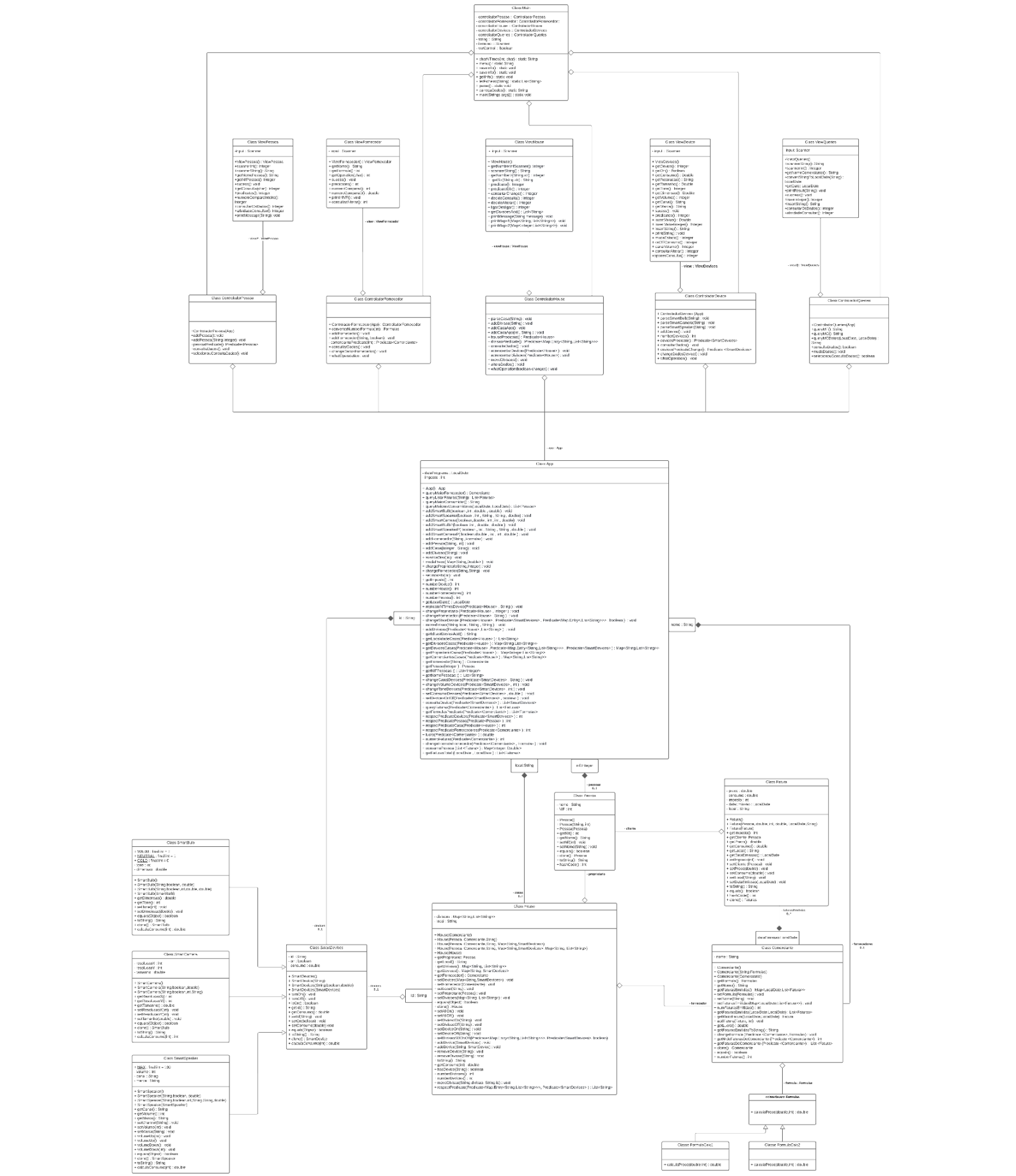
Por último, acreditamos que o nosso programa está bom, visto que a maioria dos requisitos foram estabelecidos de forma sucedida. No entanto, julgamos que, perante um maior prazo de entrega, haveria certas partes que queríamos ter implementado, como por exemplo usarmos as Functions e os Consumers, fazendo assim o nosso código mais genérico, bem como fazer a auto simulação.

No geral, tentamos sempre trabalhar com aspetos aos quais estamos confortáveis de forma a garantirmos a total funcionalidade do programa, e obviamente também demos a devida atenção ao encapsulamento, eficiência e código genérico.

Nós usamos muitos Maps, pois consideramos ser uma forma extremamente eficiente e simples de guardar dados. Como já foi referido, também usamos bastante a agregação para evitar mudar a informação em muitos sítios, mas obviamente tivemos que ter um cuidado extra para não termos dados corrompidos.

9. Diagrama UML

Link para visualizar o nosso diagrama:
https://lucid.app/lucidchart/44e9d80d-2602-4c45-8ce1-4a7c837796d1/view?page=0_0&invitationId=inv_f607578f-0965-413f-bb33-89b8fdf6322c#
(Imagem do diagrama está na próxima página)


Class Main
Class ViewPersones
Class ViewFornecedor
Class ViewHouse
Class ViewDevice
Class ViewQueries
Class ControladorPessoa
Class ControladorFornecedor
Class ControladorHouse
Class ControladorDevice
Class ControladorQueries
Class App
Class SmartBulb
Class SmartCamera
Class SmartSpeaker
Class SmartDevices
Class Pessoa
Class Fornecedor
Class House
Class Device
Class Queries
Class Comercante
Class FormulaCusto1
Class FormulaCusto2