

Relatório do Trabalho Prático de POO

Trabalho realizado por :

João Bernardo Teixeira Escudeiro (a97075)-MIEI

Hugo dos Santos Martins(a95125) MIEI

Diogo Rafael Rodrigues Aires (a91685) LCC

***ÍNDICE:***

* Introdução
* Classes:

•SmartDevice;

•SmartBulb;

•SmartSpeaker;

•SmartCamera;

•House;

•Fornecedor;

•Fatura;

•Principal;

•Parser;

•Menus;

•Main;

* Diagrama de classes
* Conclusão

***INTRODUÇÃO***

No âmbito da cadeira Programação Orientada aos Objetos, construímos um sistema que permite monitorizar e registar toda a informação energética, de uma determinada casa, na sua respetiva comunidade. É ainda necessário, referir que este registo do consumo energético é somente direcionado para alguns *SmartDevices* em questão.

Para a construção deste projeto, foram desenvolvidos variados tipos de classes, abordados posteriormente, tentando ao máximo, abordar e respeitar os conceitos lecionados ao longo do semestre.

Uma imagem com exterior, árvore, pessoa, relva

Descrição gerada automaticamenteO grupo:

João Bernardo Teixeira Escudeiro (a97075)-MIEI

Uma imagem com pessoa, exterior, relva, árvore

Descrição gerada automaticamente

Hugo dos Santos Martins(a95125) MIEI

Uma imagem com pessoa, parede, interior, homem

Descrição gerada automaticamente

Diogo Rafael Rodrigues Aires (a91685) LCC

***CLASSES:***

***SmartDevice***

Tal como o nome indica, esta é a classe (abstrata) que trata todo tipo informações e características comuns, relativas a um SmartDevice. Estas características são : o estado do dispositivo (*ON* ou *OFF*) e o seu respetivo ID. Nesta mesma classe encontramos alguns construtores, que definem todo o estado dum determinado dispositivo.

Os métodos foram criados para que haja um interação direta com os objetos das subclasses, como por exemplo, *turnOn* e *turnOff* são ambos métodos que permitem ao programa ligar ou desligar o respetivo dispositivo.

As seguintes classes (ou subclasses) estendem a classe SmartDevice, herdando assim, as variáveis de instância e métodos da classe pai.

->***SmartBulb***:

Herdando as instâncias e métodos da classe Pai, foram acrescentadas variáveis de instâncias. Uma delas é o *dailyConsumpt* que tal como nome indica, indica o consumo diário do aparelho, a outra é a *tone* que define a tonalidade da lâmpada(WARM,NEUTRAL ou COLD).

->***SmartSpeaker***:

Herdando as instâncias e métodos da classe Pai, foram acrescentadas variáveis de instâncias. As variáveis desta classe são : *dailyConsumpt* que indica o consumo diário do aparelho , *Brand* que indica a marca da coluna ,*Volume* e *Channel* que indica o canal rádio.

->***SmartCamera***:

Herdando as instâncias e métodos da classe Pai, foram acrescentadas variáveis de instâncias. As variáveis que esta classe possui são : *dailyConsumpt* que indica o consumo diário do aparelho , *res* que indica a resolução da camera e também o *filesize* que indica o tamanho do ficheiro gerado pela camera.

É ainda necessário referir que existem métodos que identificam o estado do aparelho, e como este está a ser utilizado, ou , também, metodos que trabalham diretamente com as instâncias privadas de cada subclasse, como por exemplo, na *SmartSpeaker*, temos um método *volumeUp* que, evidentemente, permite ao utilizador trabalhar com a instância “volume”.

Em cada uma destas classes tem os respetivos construtores que invocam o *super*, os *getters* e *setters*, e também todos os métodos que ajudam na alteração e interpretação das respetivas variáveis .

***House***

Este é a classe que trata do objeto casa, ou, neste caso, “House”.

Nesta classe, foram atribuídas variáveis de instância como: *nif*, *owner* e *forn\_name* que permitem assim identificar o dono e fornecedor energético da mesma. Foram também implementadas outras instâncias (coleções ) como: *room*, *devices* e *faturas* (dum tipo geral Map) que permitem então procurar e detetar dispositivos duma determinada sala ou então procurar uma respetiva faturas numa lista delas.

Evidentemente foram criados métodos que permitem ligar e desligar todos os aparelhos da casa(*setAllOn* e *setAllOff*), que permitem ligar ou desligar todos os aparelhos dentro de uma divisão , adicionar uma fatura a lista de faturas da casa (*addFatura*), adicionar um dispositivo a uma respetiva sala (*addDevToRoom*) ou até adicionar uma divisão a uma casa (*addRoom*) bem como remover , e também métodos auxiliares que permitem perceber se um dado dispositivo está na casa e até mesmo se está numa divisão.

***Fornecedor***

Tal como o nome indica, esta classe, é referente ao fornecedor energético.

Como tal, as instâncias criadas foram ao encontro do retrato da classe, como: energyprice, faturas(Map<Integer, Fatura>), que retém todas as faturas emitidas pelo mesmo, lista(Map<String, House>), que contém a lista de todas as casas que alimenta, e obviamente, a instância name, que identifica o próprio.

Dentro desta classe, temos métodos relacionados com os objetos tratados por esta, maioritarimente faturas. Como tal, esta classe possui *setFatura*, *removeFatura*, *existsFatura* entre outros.

O fornecedor, também trata de informações relacionadas com os objetos do tipo *House*. Assim foram desenvolvidos os métodos: *existsHouse*, *removeHouse*, entre outros.

***Fatura***

Está é a classe referente aos objetos Fatura. Tal como acontece, trata todo o tipo de informação necessária para a construção de uma, parcialmente idêntica a uma fatura que podemos encontrar no nosso dia a dia. Nesta encontramos algumas variáveis como *id*, *prop*, *costbefore*, *fornecedor*, *tax*, *fullprice*, *before* e *after* (before e after são variáveis do tipo LocalDate que nos ajudam a perceber o intervalo de tempo a que aquela fatura diz respeito) . Instâncias estas necessárias para a identificação e criação duma determinada fatura.

Nesta classe são também implementados métodos que ajudam na emissão de faturas , bem como os *setters* e *getters*.

***Principal***

Como o próprio nome indica , esta classe é a classe mais “geral do programa”. Aqui estão contempladas a coleção de casas e a coleção de Fornecedores . Ambas estas coleções são *Maps* que fazem corresponder : numa delas o *nif* do proprietário a uma instância do tipo casa (Map <String , House> houses ) , e na outra faz corresponder o nome do fornecedor à respetiva instância do tipo fornecedor (Map <String , Fornecedor> forn ) .

São também criados os métodos de adicionar e remover uma casa /fornecedor a cada um dos *Maps* , bem como implementados todos os métodos que ajudam nas estatísticas e em todas as funcionalidades que o menu possui.

***Parser***

Esta classe apenas se encarrega de fazer o parse do ficheiro dos Logs , fornecido pelo professor e povoar uma instância da classe Principal com todos os aparelhos, divisões, casas, fornecedores existentes nesse ficheiro .

***Menus***

Podemos identificar esta classe como a interação entre o cliente e o programa, isto é, tudo o que o cliente pedir, consoante as opções do sistema, será efetuado através dos métodos desta mesma.

Como tal, existe apenas uma instância, ler, cujo objetivo será ler o comando e, consoante este, efetuar as operações pedidas, através de alguns métodos disponíveis, tais como: createHouse, removeHouse (criar/remover uma casa), createForn, removeForn e changeForn (crir/remover/alterar fornecedor duma determinada casa), addToHouse, cujo adiciona um respetivo Device numa casa, entre outros métodos.

Aquando da inicialização do menu é tentado abrir um estado de um programa previamente gravado num ficheiro. Caso o programa não consiga abrir o ficheiro , então são utilizados os users que estão no ficheiro dos Logs .

Após realizar as operações pretendidas e quando se fecha o Menu é gravados num ficheiro de objetos o estado do programa atual , bem como dois inteiros respetivos ao id do *SmartDevice* e ao id da fatura. Estes ids têm de se guardar visto que é importante não haver conflitos entre ids aquando do preenchimento , quer do ficheiro de objetos quer do ficheiro do professor.

***Main***

Tal como o nome, esta classe possui o método main, cujo relaciona todo o programa, interação com o utilizador e o respetivo input/output.

***Exceptions***

Existem algumas exceções no código que forma criadas para serem lançadas , quando um aparelho não existe , ou quando uma casa não existe e é procurada . O grupo considera que devia ter utilizado mais exceções , mas após o código já estar concluído era difícil ter de rescrevê-lo com pouco tempo .

***Consumo Energético:***

Em relação ao consumo de cada Device :

SmartBulb Warm: Consumo Base + 0,002\*SIZE

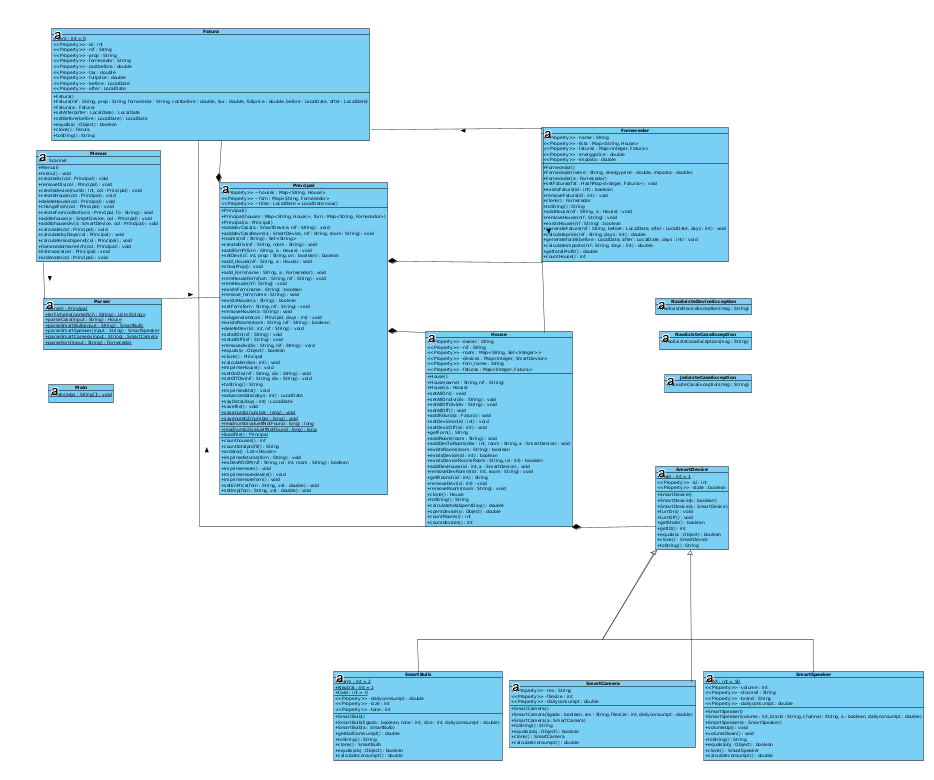
SmartBulb Neutral: Consumo Base + 0,001\*SIZE

SmartBulb Cold: Consumo Base + (1/2000)\*SIZE

SmartSpeaker : 0,01\*Volume + Consumo Base

SmartCamera : FileSize\*0,01+0,001\*res + Consumo Base

***DIAGRAMA DE CLASSES***



***Nota:***No zip também contém um ficheiro.vpp que contém o diagrama.

***CONCLUSÃO***

Tendo em conta o desenvolvimento do trabalho ao longo do semestre , bem como os objetivos finais do mesmo , o grupo auto-avalia-se de forma positiva , visto que o programa respeita o encapsulamento (são sempre retornadas cópias dos objetos) e os outros princípios básicos da programação orientada aos objetos .

Assim , e tendo em conta que o programa permite adicionar e remover dispositivos, criar e remover salas , criar e remover casas , bem como fornecedores, entre outras funcionalidades adicionais , bem como realizamos as estatísticas pedidas pelo professor , consideramos que o objetivo geral deste trabalho prático foi atingido com sucesso.