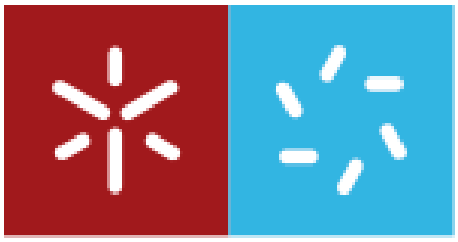

Universidade do Minho

Licenciatura em Ciências da Computação

Trabalho Prático POO

Tiago Adriano Moreira (92046)

José Miguel Barbosa (95088)

Leonardo Lordello Fontes (96308)

Índice

4. Conclusão

Capítulo 1

Introdução ao Trabalho

No âmbito da cadeira de programação orientada a objetos desenvolvemos um projeto, proposto pela equipa docente. O projeto reflete um sistema de monitorização e registo de informação sobre o consumo energético das habitações de uma comunidade. Durante o seu desenvolvimento foram utilizados os conhecimentos adquiridos nas aulas.

Na sua conceção surgiram diversas dificuldades e questões, como por exemplo: a estrutura do projeto, quer em termos de organização de pastas, quer em termos de código concreto, quer em termos de fluxo do programa.

Capítulo 2

Estrutura do trabalho

O código do nosso projeto foi subdividido em pastas que por sua vez têm classes que estão intimamente ligadas. No seu interior é notório o padrão de desenvolvimento MVC. Este padrão é claramente evidenciado em cada uma das pastas, onde o modelo, a visualização e os controladores são espelhados pelos sufixos no nome das classes Model, View e Controller, respetivamente.

Model: Todas as classes com sufixo Model, estão encarregues da gestão dos dados requisitados e atualizados pelo Controller específico da parte

View: Todas as classes cujo sufixo View são constituídas por menus, onde ocorrem os inputs do cliente, e validadores do input, isto é, métodos privados cuja função é garantir a validade dos inputs do cliente.

Controller: As classes que têm como sufixo Controller, adquirem a função de controlar o fluxo de dados pelo cliente dado, isto é, redireciona o cliente para submenus ou para menus anteriores e redireciona a informação por ele fornecida aos métodos do Model específico dessa pasta.

Para o auxílio da compilação do nosso código, bem como a gestão de dependências foi usado o software Maven.

Capítulo 3

Classes

I. Main

A Classe Main é a responsável pelo arranque do programa

II. MainController

A Classe MainController está encarregue de redirecionar o cliente para os submenus principais (aqueles visíveis no Menu Principal).

III. MainModel

A Classe MainModel é classe que armazena os Models das Classes EnergySuppliers, Factory e SmartHouses.

IV. MainView

A Classe MainView contém o design do Menu Principal do projeto

V. Owner

A Classe Owner está diretamente relacionada com a Classe SmartHouse. Cada SmartHouse tem de ter um proprietário, sendo que ele pode ser dono de mais do que uma propriedade. É uma classe muito simples unicamente definida por duas String (nif e name).

VI. SmartHouse

```
private Owner owner;
private String address;
private String energySupplier;
```

```
private final Map<String, SmartDeviceModel> smartDevices;
```

A Classe SmartHouse é uma das classes mais importantes do projeto. Contém a informação relevante às SmartHouses, sendo esta, o proprietário, o endereço e o fornecedor de energia correspondentes. Cada casa tem uma coleção que mapeia uma String (que corresponde ao nome de uma divisão) à um grupo de SmartDevices uma vez que varias das suas variáveis são bastante importantes no cálculo do custo que um owner terá de pagar. É a base da organização uma vez que é através destes mesmos SmartDevices e do EnergySupplier que os custos mais importantes são calculados.

VII. SmartHouseController

A Classe SmartHouseController corresponde ao Controller da classe SmartHouse

VIII. SmartHouseModel

A Classe SmartHouseModel corresponde ao Model da classe SmartHouse

IX. SmartHouseView

A Classe SmartHouseView corresponde ao View da classe SmartHouse

X. SmartDevice

```
private enum state {ON, OFF}
private State state;
private double InstalationCost;
private String factoryCode;
private double energyConsumption;
private LocalDateTime lastStateChange;
```

A Classe SmartDevice é uma superclasse, de onde obtemos três outras subclasses SmartDeviceBulb, SmartDeviceCamara e SmartDeviceSpeaker. A Superclasse é abstrata, uma vez que nela se encontram métodos também eles abstratos, isto

acontece, pois, houve necessidade de termos apenas assinaturas em SmartDevice que serão implementadas pelas subclasses referidas acima que, por si só, não são abstratas. A importância desta classe é bastante elevada, uma vez que varias das suas variáveis são bastante importantes no cálculo do custo que um Owner terá de pagar.

XI. SmartDeviceCamara

A Classe SmartDeviceCamara é uma das três subclasses da Superclasse SmartDevice, que refere ao dispositivo "Camara Inteligente". Este tem como características especificas, o Tamanho do Ficheiro e a Resolução da Camara.

XII. SmartDeviceSpeaker

A Classe SmartDeviceSpeaker é uma das três subclasses da Superclasse SmartDevice, que refere ao dispositivo "Coluna Inteligente". Este tem como características especificas, a Marca, o Volume atual e máximo e a Estação de Radio da coluna

XIII. SmartDeviceBulb

A Classe SmartDeviceBulb é uma das três subclasses da Superclasse SmartDevice, que refere ao dispositivo "Lâmpada Inteligente". Este tem como características especificas, a Tonalidade (que pode ser Neutral, Warm e Cold) e a sua Dimensão.

XIV. SmartDeviceModel

A Classe SmartDeviceModel é o Model correspondente à superclasse SmartDevice e às suas 3 subclasses.

XV. EnergySupplier

```
private String name;
```

```
private final static double BASE_COST = 5;
private final static double TAX = 0.05;
private String formula;
```

A Classe EnergySupplier é a classe referente aos Fornecedores de Energia. Para além do nome, esta possui também duas constantes double, comuns a todos os EnergySupplier, correspondentes ao custo base e ao imposto. Estes dois valores são depois usados na fórmula, que é especifica a cada Fornecedor, e que nos dará o custo que cada casa terá de pagar consoante o seu consumo energético. Este consumo é calculado com a ajuda da variável energyConsumption da Superclasse SmartDevices e do número de dispositivos existentes.

XVI. EnergySupplierModel

A Classe EnergySupplierModel é o Model correspondente à classe EnergySupplier

XVII. EnergySupplierController

A Classe EnergySupplierController é o Controller correspondente à classe EnergySupplier

XVIII. EnergySupplierView

A Classe EnergySupplierView é o View correspondente à classe EnergySupplier

XIX. FactoryModel

A Classe FactoryModel é uma componente essencial para o bom funcionamento do projeto. Este Model é responsável por guardar todos os dispositivos que foram criados, ou seja, mesmo que um dispositivo já esteja numa casa, esta classe tem-no armazenado numa coleção. Isto permite que o recuperemos caso seja removido de uma casa. Os dispositivos são todos depositados no SmartDeviceModel. Também permite uma certa flexibilidade caso queiramos transitar este dispositivo para alguma outra classe que faça sentido (Não é do interesse neste projeto em específico, mas poderia ter alguma loja ou algo do

género). Este Model é o único local no projeto onde se cria, atualiza e destrói dispositivos, qualquer dispositivo removido é destruído e deixa de estar disponível. Todas as operações de remoção total e atualização estão resignadas aos dispositivos somente na fábrica, ou seja, não é possível que a partir desta classe, atualizar e remover dispositivos que já estejam numa casa, garantindo assim o encapsulamento.

XX. FactoryController

A Classe FactoryController é o Controller da Factory e ajuda ao bom funcionamento do FactoryModel.

XXI. FactoryView

A Classe FactoryView é o View da Factory, ou seja, é onde estão armazenados vários menus relevantes à classe respetiva.

XXII. StateModel

A Classe StateModel é responsável por carregar e guardar o estado do MainModel

XXIII. StateController

A Classe StateController representa o Controller do State que redireciona a informação de carregar e guardar o estado num caminho fornecido pelo StateView.

XXIV. StateView

A Classe StateView representa o View do State.

XXV. DataStatusModel

A Classe DataStatusModel trata de calcular toda a estatística no trabalho pedida, vai receber do Controller, uma lista de faturas (uma lista de objetos da classe Invoicer) e a partir dai, fará várias coisas propostas, como por exemplo:

A casa que gasta mais dinheiro em energia e a companhia de eletricidade com maiores ganhos.

XXVI. DataStatusController

A Classe DataStatusController é importante no "apoio" à classe DataStatusModel, uma vez que é deste Controller, que saem vários dados importantes utilizados pela mesma.

XXVII. DataStatusView

A Classe DataStatusView é o View do DataStatus e contém o menu com os vários cálculos e requisitos pretendidos.

XXVIII. ParseLogs

A Classe ParseLogs está responsável por transformar o ficheiro de logs fornecido pelo professor que contem as entidades necessárias ao funcionamento da simulação.

XXIX. Invoicer

A Classe Invoicer armazena as informações de uma fatura

XXX. ManualSimulation

A Classe ManualSimulation é a responsável por gerar faturas com a informação obtida através das casas (SmartHouse) e dos respetivos fornecedores de energia (EnergySupplier). A simulação acontece entre duas datas e durante a mesma não é possível alterar o estado das entidades.

XXXI. AutomaticSimulation

A Classe AutomaticSimulation semelhante à Classe ManualSimulation é responsável por gerar faturas entre duas datas. A diferença é que nesta operação é permitida a alteração do estado das entidades (por exemplo é possível mudar a fórmula de um fornecedor de energia ou ligar/desligar dispositivos)

XXXII. SimulationController

A Classe SimulationController tem como função iniciar uma simulação manual ou automática independentemente

XXXIII. SimulationView

A Classe SimulationView contém menus das simulações. No caso da ManualSimulation, é pedido ao usuário os dias de início e fim da simulação. Já na AutomaticSimulation é pedido o caminho do arquivo que o cliente deseja carregar (onde supostamente se encontrará o arquivo de texto que vai ser utilizado na simulação)

Diagramas de Classes

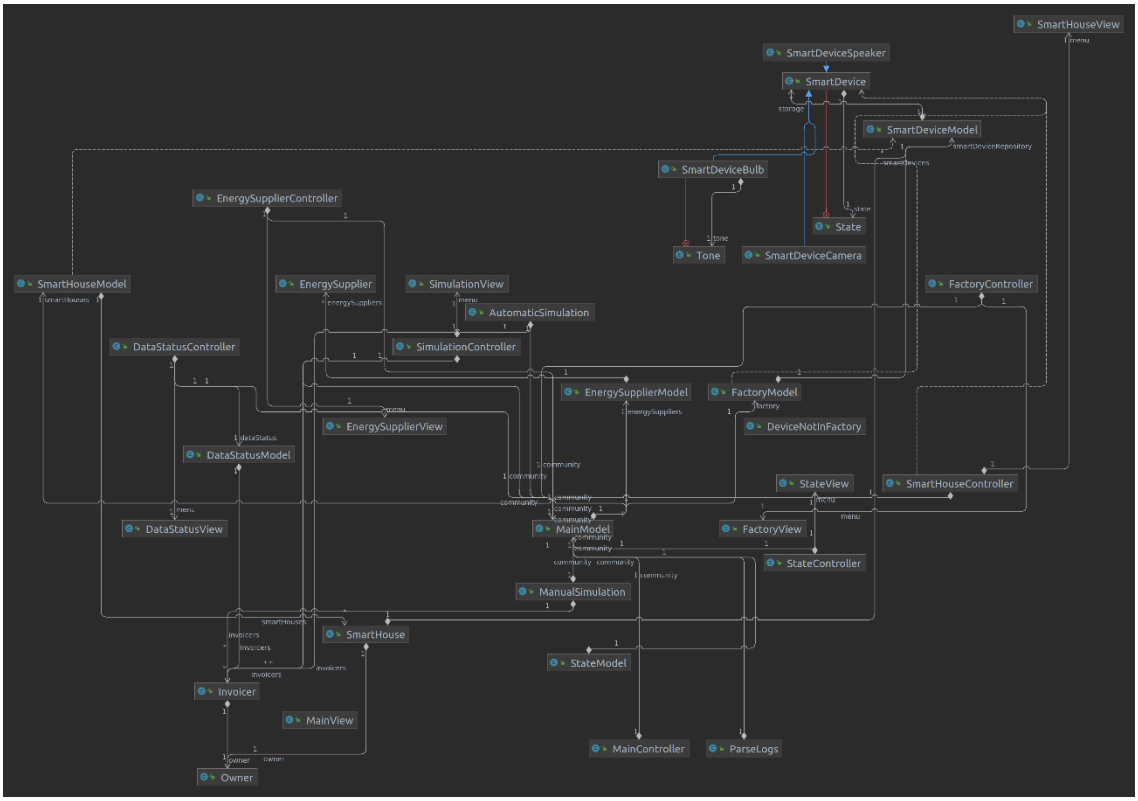

Figura 1:

Diagrama de classes do projeto (apenas com as classes), gerado pelo *IntelliJ*

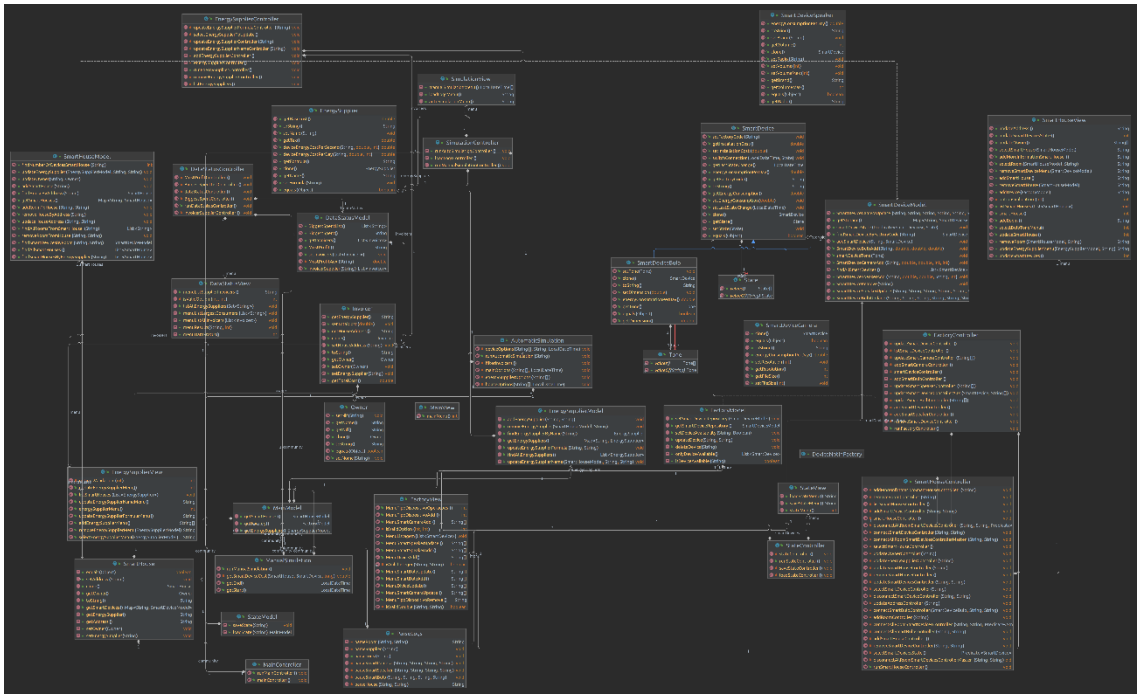

Figura 2:

Diagrama de classes do projeto (com os métodos), gerado pelo *IntelliJ*

Capítulo 4

Conclusão

Num panorama geral, encontramos várias dificuldades ao longo da realização do projeto. Contudo, graças à nossa organização de grupo, que esteve em constante contacto, os problemas foram todos ultrapassados. Também graças à nossa dinâmica de grupo conseguimos criar um código, muito suscetível a mudanças para corrigir possíveis erros ou criar funcionalidades.