

Gestão e Simulação de Dispositivos Inteligentes, Casas Inteligentes e Fornecedores de Energia

Relatório do Trabalho Prático de POO

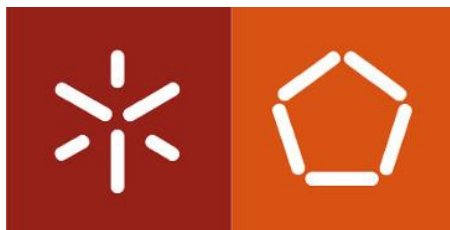
Programação Orientada a Objetos

2º Ano | 2º Semestre

Grupo 28

José Pedro Batista Fonte a91775

Dinis Gonçalves Estrada a97503



Mestrado Integrado em Engenharia Informática

Departamento de Engenharia Informática

Universidade do Minho, Braga, Portugal

Maio 2022

ÍNDICE

- Introdução -----	3
- SmartDevices e as suas subclasses -----	4
- Casa Inteligente: Descrição e Métodos -----	5
- Fornecedores: Descrição e Abordagem-----	6
- Fatura: Descrição e Métodos-----	7
- Estado: Descrição e Estatísticas-----	7
- Parser: Descrição -----	8
- Interfaces: Serializable e Comparable-----	8
- Error handling-----	9
- Menus e Submenus: Estrutura e abordagem M-V-C -----	10
- Conclusão -----	12
- Anexos -----	13
- Figura Diagrama de Classes -----	13

Introdução

No âmbito da cadeira de Programação Orientada a Objetos, o grupo 28 desenvolveu um programa em Java de Gestão e Simulação de Dispositivos Inteligentes, Casas Inteligentes e Fornecedores de Energia.

Segundo o enunciado apresentado o objetivo do trabalho é criar um sistema que monitorize e registre a informação sobre o consumo energético das habitações de uma comunidade. Em cada casa existem um conjunto muito alargado de dispositivos que são todos controlados a partir do programa. Cada casa tem a si associada um fornecedor que compete num mercado de energia com outros fornecedores.

De forma a ter uma interação intuitiva com o programa o grupo construiu uma interface via terminal onde é possível, criar, editar, guardar e carregar estados, assim como executar uma simulação sobre o mesmo. Na simulação é possível avançar o tempo, emitir faturas, executar estatísticas e editar o estado – por exemplo, ligar e desligar dispositivos, mudar fornecedores, etc.

O relatório está estruturado com a descrição das classes e técnicas utilizadas para resolver os problemas propostos, seguido do modo como construímos os Menus e finalizado pelo tratamento de erros.

Nos anexos, encontra-se o diagrama de classes, com todas as classes, as suas variáveis de instância e os seus métodos, assim como todas as relações entre as mesmas.

SmartDevices e as suas subclasses

Segundo o enunciado existem 3 *Smartdevices* diferentes – o *SmartSpeaker*, *SmartBulb* e *SmartCamera*. O grupo decidiu criar a superclasse ***Smartdevice*** com os métodos e atributos comuns a todos. Depois os smartdevices são declarados em subclasses próprias onde se define as suas v.i. e métodos específicos.

SmartDevice

```
private String id;  
  
private boolean on;  
  
private float custo_inst;  
  
private float consumo_diario;
```

Estes são os atributos comuns a todos os Smartdevices. Um **identificador**, o seu **estado** -Ligado ou Desligado – o **custo de instalação**, que existe mesmo que não esteja ligado, e o consumo diário que é definido pelas subclasses pois segundo o enunciado o seu consumo diário depende de v.i. de cada SmartDevice.

<u>SmartSpeaker</u>	<u>SmartCamera</u>	<u>SmartBulb</u>
private int volume; private String channel; private String brand; private float brand_comsupption;	private Resolution resolution; private float file_size;	private int tone; private float dimensão; private float valor_fixo;

Casa Inteligente: Descrição e métodos

CasaInteligente

```
private String id;  
  
private String owner;  
  
private int nif;  
  
private Fornecedor fornecedor;  
  
private Map<String,Smartdevice> devices;  
  
private Map<String,List<String> > locations;
```

Segundo o enunciado uma casa inteligente, tem as v.i. apresentadas em cima para caracterizar uma casa inteligente, mais em específico os dois Maps, que são do tipo HashMap são onde se guarda a lista de Smartdevices de uma casa sendo a key o seu id e a lista de Devices por divisão, sendo a key a divisão e o valor respetivo uma lista de ids.

Para além dos métodos normais (construtores + setters + getters) é de destacar que esta classe tem métodos como o custoCasa(), que permite aplicar a formula do fornecedor e calcular o custo total da casa, assim como todos os métodos de desligar e ligar dispositivos.

Fornecedores: Descrição e abordagem

Fornecedor

```
private String name;  
  
private float valor_base;  
  
private float imposto;  
  
private float desconto;
```

Segundo o enunciado existe um mercado de fornecedores onde a única diferença entre eles é o modo como calculam o preço. Deste modo, o grupo decidiu criar a classe abstrata Fornecedor, onde o método *formulaPreço()* é abstrato e só é definido nas subclasses. Deste modo criamos um “construtor” de Subclasses, que cria 3 subclasses: FornecedorA, FornecedorB e FornecedorC, com o método *formulaPreço()* definido do seguinte modo:

- **FornecedorA:** $\text{formulaPreco} = (\text{ValorBase} * \text{Consumo_Diario} * ((1 + \text{Imposto}) / 100)) * ((1 - \text{Desconto}) / 100)$
- **FornecedorB:** $\text{formulaPreco} =$
 $\text{numeroDispositivos} < 10 ? (\text{ValorBase} * \text{Consumo_Diario} * ((1 + \text{Imposto}) / 100))$
 $* ((1 - \text{Desconto}) / 100) :$
 $(\text{ValorBase} * \text{Consumo_Diario} * ((1 + \text{Imposto}) / 100)) * ((1 - \text{Desconto} * 2) / 100)$
- **FornecedorC:** $\text{formulaPreco} =$
 $\text{numeroDispositivos} < 10 ? (\text{ValorBase} * \text{Consumo_Diario} * ((1 + \text{Imposto}) / 100))$
 $* ((1 - \text{Desconto}) / 100) :$
 $\text{numeroDispositivos} < 20 ? (\text{ValorBase} * \text{Consumo_Diario} * ((1 + \text{Imposto}) / 100))$
 $* ((1 - \text{Desconto} * 2) / 100) :$
 $(\text{ValorBase} * \text{Consumo_Diario} * ((1 + \text{Imposto}) / 100)) * ((1 - \text{Desconto} * 3) / 100)$

Fatura: Descrição e métodos

Fatura

```
private CasaInteligente casa;  
  
private Localdate inicio_faturacao;  
  
private LocalDate fim_faturacao;  
  
private float consumo;  
  
private float custo;
```

A necessidade de criar a classe faturas derivou de no enunciado pedir para haver períodos de faturação onde se emite faturas. A fatura tem o seu início e fim, está associada a uma casa e calcula o seu consumo e o seu custo.

Estado: Descrição e Estatísticas

Estado

```
private Localdate date;  
  
private Map<String,CasaInteligente> casas;  
  
private Map<String,Fornecedor > fornecedores;
```

A classe Estado surge como solução para a parte da simulação do trabalho. O Estado tem a sua data e guarda em HashMaps todas as casas e de todos os fornecedores. Um Estado guarda toda a informação sobre a configuração atual do programa e é sobre ele que são executados os métodos das estatísticas e das faturas:

1. qual é a casa que mais gastou naquele período
2. qual o comercializador com maior volume de facturação
3. dar uma ordenação dos maiores consumidores de energia
4. listar as facturas emitidas por um comercializador
5. fatura de uma casa

Parser: Descrição

A classe Parser é responsável por interpretar um documento .csv e gerar a configuração de Fornecedores e de Casas. Os métodos da classe percorrem as linhas do ficheiro e de acordo com a formatação apresentada cria casas, divisões, dispositivos e fornecedores.

De notar o seguinte, como o ficheiro não tem todas as informações que as classes necessitam, muitas coisas foram automatizadas, por exemplo o ficheiro não indica:

- o tipo de fornecedor, nem os seus valores. Para isso o fornecedor é escolhido aleatoriamente entre as 3 opções possíveis e os valores são predefinidos.
- o id do smartdevice e se está ligado/desligado. Para isso o id do *smartdevice* tem 3 formatos *smtblb/smtspk/smtcmr* concatenado com 6 dígitos aleatórios. Estar ligado/desligado também é aleatório com as probabilidades 33% ligado e 77% desligado.
- o identificador da casa. Sendo que o grupo considerou que o mesmo proprietário pode ter mais do que uma casa, resolvemos o problema concatenando ao nome do proprietário 6 dígitos aleatórios.

Interfaces: Serializable e Comparable

As interfaces implementadas por quase todas classes são a *Serializable* e a *Comparable*.

A Interface *Serializable* é utilizada para guardar e carregar a classe num ficheiro Objeto, a Interface *Comparable* é utilizada para a classe poder ser comparável.

Em maior detalhe, as classes CasaInteligente e Fatura implementam *Comparable* sendo que os critérios de comparação são:

- CasaInteligente – o critério primário é o custo da casa, o critério secundário é a ordem alfabética do nome da casa.
- Fatura – o critério primário é o custo da fatura, o critério secundário é a ordem alfabética do nome da casa.

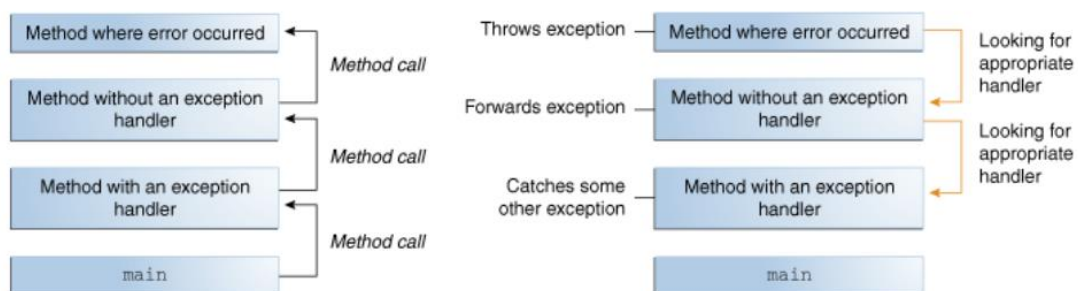
Em algumas situações o *compareTo()* da CasaInteligente é substituído por outro critério, nesse caso o consumo da casa.

ErrorHandling

Todos os objetos têm v.i.s que tem de ser introduzidas e só estão corretas se cumprirem certos critérios. Um exemplo é ao inicializar um fornecedor este não pode ter valores de preço de energia, descontos ou impostos negativos.

A abordagem do grupo foi criar várias subclasses que estendem a classe *Exception* e tem uma mensagem customizada para ser mais fácil e intuitivo detetar os erros. As subclasses criadas estão no package *ErrorHandling*.

A implementação foi segundo a metodologia apresentada nas aulas, que indica que ao nível mais granular as classes devem fazer *throw Exception*, depois no controller temos vários *try...catch* onde se “apanha” os erros e imprimem a mensagem. A vantagem desta abordagem é que depois podemos correr o menu novamente sempre que os valores estão incorretos.

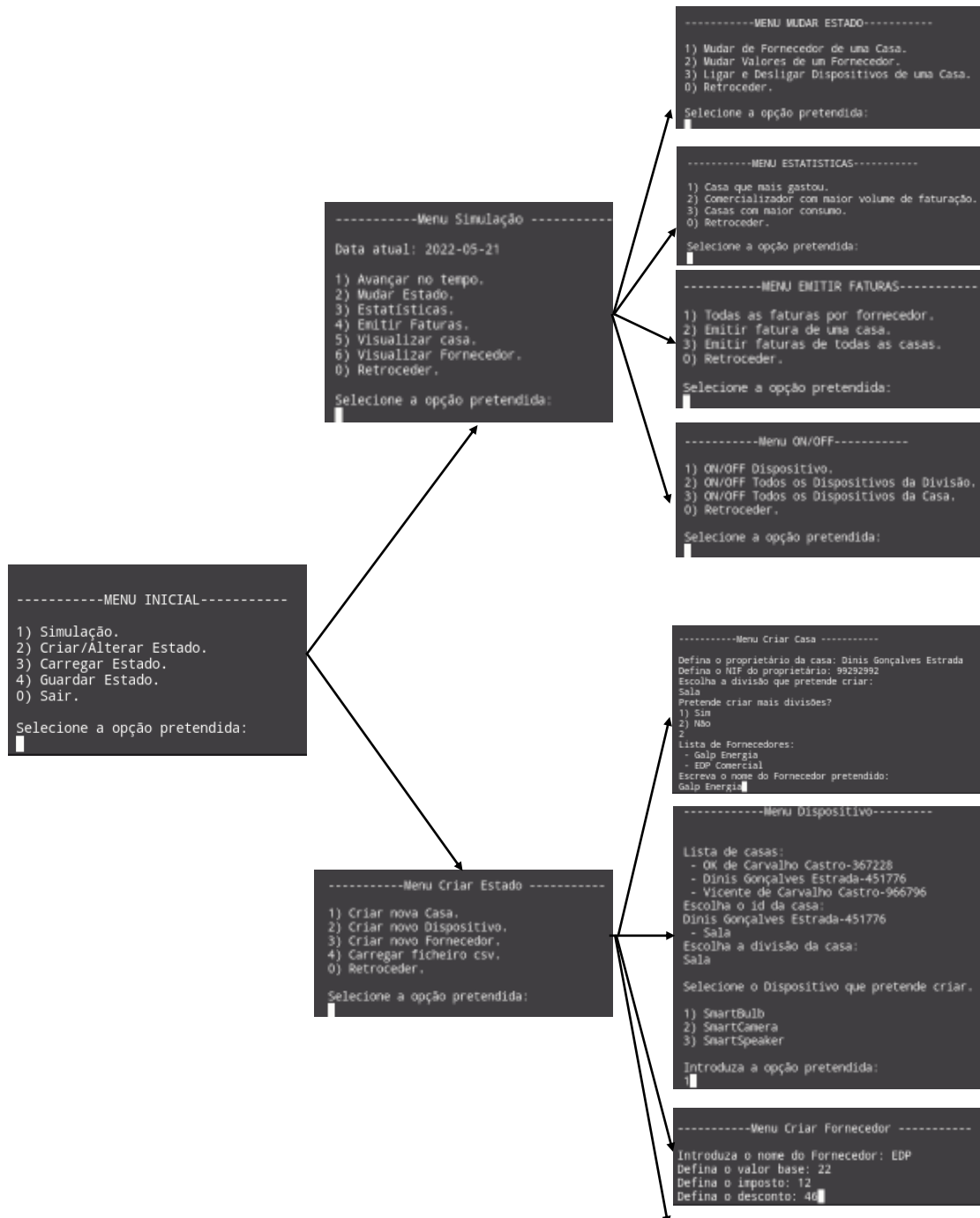


Menus e Submenus: Estrutura e abordagem M-V-C

De modo a criar uma interface intuitiva para o utilizador interagir com a aplicação o grupo utilizou a abordagem **Model-View-Controller**. A abordagem M-V-C passa por criar uma separação entre BackEnd(Model) e Front-End(View) conectadas pelo Controller.

O *Model* são todas as classes criadas e apresentadas anteriormente, o View é unicamente a classe Menu e o Controller são todas as classes Controller*.java.

A estrutura do menu é a seguinte.



Introduza o nome do ficheiro csv que pretende carregar:
logs1.csv

O utilizador consegue ver o conteúdo do estado selecionando os submenus de “i) Visualizar ...”. Nestes submenus imprime-se os Fornecedores, Casas e Faturas na formatação seguinte:

```
### FornecedorB ###  
Fornecedor: Galp Energia | ValorBase: 2.0 | Imposto: 15.0 | Desconto: 10.0  
Pressione enter para continuar...
```

```
### House ###  
ID: Vicente de Carvalho Castro-966796 | Owner: Vicente de Carvalho Castro | NIF: 365597405 | Fornecedor:  
  
### FornecedorA ###  
Fornecedor: Galp Energia | ValorBase: 0.1 | Imposto: 6.0 | Desconto: 5.0  
  
Devices:  
  
### SmartDevice ###  
ID: smtblb-517397 | Ligado: false | Custo de Instalação: 0.05 | Consumo Diário: 0.017499998  
Type: SmartBulb | Tone: 2 | Dimensão: 7.0 | Valor Fixo: 0.005  
  
Rooms:  
Sala de Jantar 1: smtblb-517397,  
Pressione enter para continuar...
```

```
##### Fatura #####  
Casa: Dinis Gonçalves Estrada-451776  
Fornecedor: Galp Energia | FornecedorB  
Inicio: 2022-05-21 | Fim: 2022-05-21 | Período: 0 dia(s)  
Consumo: 0.0 kWh  
Valor: 0.0 €
```

Conclusão

O grupo considera que na globalidade conseguiu atingir os objetivos ao qual se propôs, devido à falta de tempo e à falta de participação do 3º elemento do grupo - **Diogo Miguel Serra Silva a96277**, o grupo não conseguiu executar a 4 tarefa do projeto – automatização de processos.

Numa análise global o grupo assinala como aspetos positivos a facilidade da linguagem para construção de estruturas de dados, a intuitividade do paradigma de programação orientada a objetos e a forma como é possível gerir grandes projetos assegurando a segurança dos dados.

Quanto aos aspetos negativos, consideramos que a linguagem usada, Java, tem muita syntax inútil e o paradigma de programação tem muita repetição de código devido aos construtores, setters e getters.

O projeto pareceu-nos indicado quanto ao nível de dificuldade e de carga trabalho.

Anexos

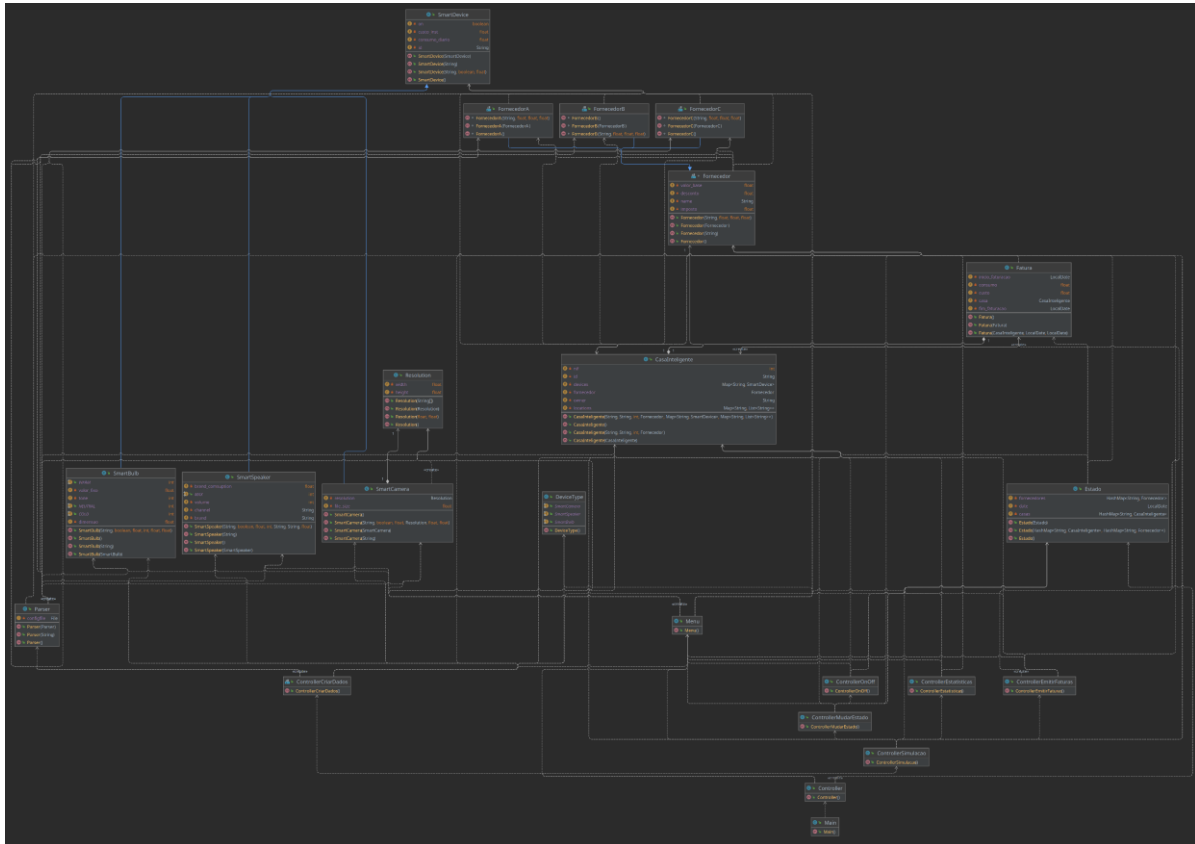


Figura 1: Diagrama de Classes