Universidade do Minho

Licenciatura em Ciências da Computação

Trabalho Prático de POO
(Programação Orientada aos Objetos)
Grupo nº17
21/05/2022

Bruno Miguel Fernandes Araújo a97509
Maria Helena Alves Machado Marques Salazar a75635
Filipe José Silva Castro a96156



Índice:

Introdução:

Com este trabalho, foi pretendido que construíssemos um programa, utilizando o paradigma de programação com base em objetos, capaz de monitorizar e registar dados sobre o consumo energético de uma dada comunidade. O uso deste paradigma permite uma fácil reutilização de código através da utilização de classes, o que faz a manutenção e construção de código mais simples.

Neste relatório descrevemos as diferentes Classes criadas durante a construção do sistema, e as variáveis de instância e métodos pertencentes a cada.
Escolhemos também incluir uma seção com Exceptions, onde explicamos as diferentes exceções que usamos para notificar os problemas que ocorreram .
Finalmente, acrescentamos alguns comentários sobre certos aspetos das funcionalidades numa secção denominada Observações.

Pretendemos que este relatório seja utilizado como um guia, de modo a facilitar a leitura do código por parte dos docentes,

Classes:

CasaInteligente:

Variáveis de Instância:

```
public class CasaInteligente implements Serializable{
    private String proprietario; //Nome do proprietario
    private int nif; //nif do proprietario
    private Map<String, List<SmartDevice>> divisoes;//isto representa as Divisões com os seus devices que pertencem á casa
    private Map<String, SmartDevice> devices; //Código/ID do device acompanhado com ele mesmo
                                              //isto representa os Devices que se encontram na casa
    private String fornecedor; //Nome do fornecedor associado a esta Casa
```

Implementação do serializable pois precisamos desta componente para escrever ,Objetos que envolvem a classe CasaInteligente ou até uma CasaInteligente, para um ficheiro.

Métodos:

Além dos típicos gets, sets e construtores temos:

```
public void trocarFornecedor(String f){
```

Como o próprio nome indica este troca o Fornecedor de uma casa para o que é recebido(“String f”) no método.

```
public void fixCasa(){
```

Um método que vai ser utilizado na aplicação que irá corrigir o hashmap “devices” , iremos abordar mais à frente o motivo deste método existir.

```
public void setDivisionOn(String s, boolean b) {
```

Como o próprio nome indica, este método liga/desliga todos os Devices que se encontram numa divisão, ele recebe o nome desta e um booleano b que corresponde a ligar (true) e desligar (false).

```
public void setDeviceOn(String s,boolean b) {
```

Como o próprio nome indica, este método liga/desliga um Device específico, ele recebe o código/ID deste (“String s”), e um booleano b que corresponde a ligar (true) e desligar (false).

```
public double FaturadaCasa(){
```

Este método calcula e retorna o total do consumo dos Dispositivos da casa.

```
public double CustototalInstalacao(){
```

Este método calcula e retorna o total do custo da instalação dos Dispositivos da casa.

```
public void addDevice(String x,SmartDevice s) {
```

Este método adiciona um device a uma divisão da Casa, ele recebe a divisão e o Device que quer adicionar (É de notar que ambos os hashmaps devices e divisões são atualizados, no sentido em que adicionamos em ambos uma nova entry).

```
public boolean roomExists(String s){
```

Este método verifica a existência de uma divisão na casa devolvendo true caso pertença à casa e false no caso oposto,ele recebe o nome da divisão.

```
public boolean deviceExists(String s){
```

Este método verifica a existência de um device na Casa devolvendo true caso pertença à casa e false no caso oposto, ele recebe o Código/ID do device.

```
public void addRoom(String s) {
```

Este método adiciona uma divisão à casa, ele recebe o nome da divisão.

```
public String toString(){
```

Este método é o toString() da casa, este representa a Casa em String.

Fornecedor:

Variáveis de Instância:

```
public class Fornecedor implements Serializable{
    private String nome; // Nome do Fornecedor
    private double valorBase;//Valor base do Fornecedor(este valor é comum para todos os Fornecedores)
    private double imposto;//Imposto (este valor é comum para todos os Fornecedores)
    private int rand; //Valor random 0 ou 1 que vai decidir qual formula usar
    private int rand2; //Valor random de 0 a 20 que vai ser usado na formula
```

Implementação do serializable pois precisamos desta componente para escrever ,Objetos que envolvem a classe Fornecedor ou até um Fornecedor, para um ficheiro.

Métodos:

Além dos típicos gets, sets e construtores temos:

```
public double precodia(){
    double resultado=0;
    switch(this.getrand()){
        case(0):
        resultado=(this.getValorBase()  * 1 + this.getImposto()) + this.getrand2();
        break;
        case(1):
        resultado=(this.getValorBase())  + (1 + this.getImposto() * this.getrand2());
        break;
    }
    System.out.print("Custo imposto pelo Fornecedor("+getNome()+") (Por dia): " +resultado+"\n");
    return resultado;
}
```

Este é o custo imposto diário devido ao fornecedor.

SmartDevice

É de notar que esta é uma super Class de SmartBulb,SmartSpeaker,SmartCamera.
Variáveis de Instância:

```
public abstract class SmartDevice{
    private boolean on; //Booleano corresponde a ligado(true) e desligado(false)
    private String codigo;//Codigo do Device gerado aleatóriamente ,o valor varia entre 0 e 10000000
    private double custoInstalacao;//Custo da Instalação do Device(este valor é comum para todos Devices,independentemente do tipo que sejam)
```

Métodos:
Além dos típicos gets, sets e construtores temos:

```
public boolean equals(Object o){
```

Este é o método equals para SmartDevices, acabamos por não necessitar do seu uso.

```
public void turnOn(){
```

Este método liga um device.

```
public void turnOff(){
```

Este método desliga um device.

```
public abstract SmartDevice clone();
```

Este método é o método clone , definido como abstrato para depois ser usado livremente sem termos de ter preocupação com o tipo de Objeto estes são (SmartBulb,SmartSpeaker,SmartCamera) acrescento que acabamos por não necessitar deste método pois não recorremos á Composição em nenhuma situação.

```
public abstract double consumoDispositivo();
```

Este método está definido como abstrato para facilitar o cálculo do consumo do Dispositivo sem termos de nos preocupar com o seu tipo de Objeto (SmartBulb,SmartSpeaker,SmartCamera) , obviamente as suas subclasses terão a sua própria definição deste método.

SmartSpeaker

Variáveis de Instância:

```
public class SmartSpeaker extends SmartDevice implements Serializable{
    private double volume; //Volume do Dispositivo
    private String radio;  //Nome da Radio que se encontra ligado
    private String marca;  //Nome da marca do Dispositivo
    private double consumoBase; //Consumo base do dispositivo
```

Como podemos ver o SmartSpeaker pertence à hierarquia do SmartDevice, sendo uma subclasse.
Também implementa Serializable pelos mesmo motivos que os das classes ,que o também implementam, previamente abordadas.

Métodos:

```
public double consumoDispositivo(){
```

Assim como já foi explorado na descrição do SmartDevice, este método é a versão do SmartSpeaker no cálculo e retorno do consumo do Dispositivo,é de notar que se o SmartSpeaker estiver desligado, o valor retornado é 0.

```
public SmartDevice clone(){
```

Outro método que também já foi previamente mencionado na descrição do SmartDevice, é o método que clona ,neste caso, SmartSpeakers's.

```
public String toString(){
```

Este método é o toString() do SmartSpeaker, ele representa o SmartSpeaker em String.

SmartBulb

Variáveis de Instância:

```
public class SmartBulb extends SmartDevice implements Serializable{
    private String tonalidade; //Tonalidade do Dispositivo varia entre(Cold,Neutral,Warm)
    private double dimensao; //Dimensão do Dispositivo
    private double consumoBase; //Consumo Base do Dispositivo
```

O SmartBulb pertence à hierarquia do SmartDevice assim como o SmartSpeaker, é uma subclasse. Também implementa Serializable pelos mesmo motivos que os das classes ,que o também implementam, previamente abordadas.

Métodos:

Além dos típicos gets, sets e construtores temos:

```
public double consumoDispositivo(){
```

Este método é a versão do SmartBulb no cálculo e retorno do consumo do Dispositivo,é de notar que se o SmartBulb estiver desligado o valor retornado é 0 e que o peso da tonalidade está escrito da seguinte forma:
Cold corresponde a 1.
Neutral corresponde a 2.
Warm corresponde a 3.

```
public SmartDevice clone(){
```

É o método que clona ,neste caso, SmartBulb's.

```
public String toString(){
```

Este método é o toString() do SmartBulb, ele representa o SmartBulb em String.

SmartCamera

Variáveis de Instância:

```
public class SmartCamera extends SmartDevice implements Serializable{
    private double altura; //altura do Dispositivo
    private double largura; //largura do Dispositivo
    private double tamanho; // tamanho do Dispositivo
    private double consumoBase; //Consumo base do Dispositivo
```

A SmartCamera, assim como o SmartBulb e o SmartSpeaker, é uma subclasse de SmartDevice.
Também implementa Serializable pelos mesmo motivos que os das classes ,que o também implementam,previamente abordadas.

Métodos:
Além dos típicos gets, sets e construtores temos:

```
public double consumoDispositivo(){
```

Este método é a versão da SmartCamera no cálculo e retorno do consumo do Dispositivo,é de notar que se a SmartCamera estiver desligada, o valor retornado é 0.

```
public SmartDevice clone(){
```

É o método que clona SmartCamera's.

```
public String toString(){
```

Este método é o toString() da SmartCamera, ele representa a SmartCamera em String.

Parser

Esta é a classe que vai processar os dados dos logs.

Variáveis de Instância:

```
public class Parser{
    private Map<String,CasaInteligente> Casas=new HashMap(); //Registo das Casas no parser(Nome do proprietario acompanhado pela sua casa)
    private Map<String,Fornecedor> Fornecedores=new HashMap();//Registo dos Fornecedores (Nome do fornecedor acompanhado por ele mesmo)
```

Métodos:

```
public void parse() throws FileNotFoundException,IOException{
```

Este método é onde ocorre o parse do ficheiro "logs.txt" e o registo do hashmap das Casas no ficheiro Casas.txt e do hashmap dos Fornecedores no ficheiro Fornecedores.txt.
A indicação "throws FileNotFoundException,IOException" é necessária para as ações de leitura e registo previamente mencionadas.

```
public List<String> lerFicheiro(String nomeFich) {
```

Este método como o próprio nome indica lê o ficheiro que neste caso vai ser o logs.txt e retorna uma Lista com linhas do ficheiro.

```
public void addCasa(CasaInteligente b){
```

Este método adiciona a Casa que recebe ao hashmap das Casas.

```
public void addFornecedor(Fornecedor b){
```

Este método adiciona o Fornecedor que recebe ao hashmap dos Fornecedores.

```
public Fornecedor parseFornecedor(String input){
```

Este método como o próprio nome indica faz parse de uma linha que aparece a seguir de um "Fornecedor:"e devolve um Fornecedor criado com a componente que leu/processou.

```
public CasaInteligente parseCasa(String input){
```

Este método como o próprio nome indica faz parse de uma linha que aparece a seguir a um "Casa:"e devolve uma Casa criada com as componentes que leu/processou.

```
public SmartBulb parseSmartBulb(String input){
```

Este método como o próprio nome indica faz parse de uma linha que aparece a seguir a um "SmartBulb:" e devolve um SmartBulb criado com as componentes que leu/processou.

```
public SmartSpeaker parseSmartSpeaker(String input){
```

Este método como o próprio nome indica faz parse de uma linha queque aparece a seguir a um "SmartSpeaker:" e devolve um SmartSpeaker criado com as componentes que leu/processou.

```
public SmartCamera parseSmartCamera(String input){
```

Este método como o próprio nome indica faz parse de uma linha queque aparece a seguir a um "SmartCamera:"e devolve uma SmartCamera criado com as componentes que leu/processou.

Fatura

Variáveis de Instância:

```
public class Fatura implements Serializable{
    private String proprietario; //Nome do proprietario da casa que foi faturada
    private String fornecedor;//Nome do fornecedor associado á casa faturada
    private LocalDate dataInicial;//Data inicial da fatura
    private LocalDate dataFinal;//Data final da fatura
    private double valor;//Valor da fatura
```

A implementação do serializable tem o mesmo significado que naqueles que já foram previamente mencionados.

Métodos:

Além dos típicos gets, sets e construtores temos:

```
public String toString(){
```

Este método é o toString() da Fatura, ele representa o Fatura em String.

APP

Métodos:

```
public static Map<String,CasaInteligente> fixCasas(Map<String,CasaInteligente> Casas){
```

Tivemos de criar este método pois tivemos um problema na leitura do ficheiro "Casas.txt", como o Código/ID dos devices é random, ao ler o ficheiro e a guardar no hashmap das Casasnonfix, este gerava novamente o Código dos devices fazendo com que o hasmap "devices" que existe em cada casa tenha
um código nas keys diferente do código do próprio device (o device associado aquela key).

```
public static void main(String[] args)throws FileNotFoundException,IOException{
```

É neste método que se encontra o código da aplicação que vai interagir com o cliente.

Opções disponíveis na Aplicação:

9.11 Opção (11) Estatística

9.11.3 Opção(4) Casa que gastou mais no intervalo de tempo.
Após pedir um intervalo, mostra a Casa que gastou mais nesse intervalo.

9.11.3 Opção(3) Comercializador/Fornecedor com maior volume de faturação
Mostra o Comercializador/Fornecedor faturou mais.

9.11.2 Opção(2) Listar as faturas emitidas por um comercializador
Após pedir o nome do Comercializador/Fornecedor, este indica todas as faturas emitidas por esse Comercializador/Fornecedor.

9.11.1 Opção(1) Criar Listar os consumidores por ordem de consumo num intervalo
Após pedir um intervalo, lista todos os consumidores com faturas nesse intervalo por ordem de consumo(do maior consumista para o menor)

9.10 Opção(10)Verificar estado de uma casa
Após pedir o nome do proprietário, este devolve a casa(print da casa).

9.9 Opção(9)Criar Fornecedor
Após pedir o nome que pretende dar ao Fornecedor, este cria um Fornecedor com esse nome.

9.8 Opção(8)Criar Casa
Para criar a casa pede os dados neste formato Nome,Nif,Fornecedor.

9.7 Opção(7)Trocar Fornecedor
Após pedir o nome do proprietário e o nome de um Fornecedor, este troca o fornecedor daquele proprietário por aquele que inseriu.

9.6 Opção(6)Criar divisão de uma Casa
Após pedir o nome do proprietário e o nome que pretende dar á divisão, este cria uma divisão.

9.5 Opção(5)Criar dispositivo de numa Divisão
Após pedir o nome do proprietário da casa e da divisão onde quer criar um device, este pergunta que tipo de Device quer.

9.5.3 Opção(3) Criar SmartCamera
Para criar uma SmartCamera pede os dados neste formato LarguraxAltura,Tamanho,ConsumoBase.

9.5.2 Opção(2) Criar SmartBulb
Para criar uma SmartBulb pede os dados neste formato
Tonalidade,Dimensão,ConsumoBase.

9.5.1 Opção(1) Criar SmartSpeaker
Para criar uma SmartSpeaker pede os dados neste formato
Volume,Radio,Marca,ConsumoBase.

9.4 Opção(4)Ligar ou Desligar Dispositivos/Divisões.
Após pedir o nome do proprietário da Casa onde quer ligar
Dispositivos/Divisões
9.4.1 Opção(1) Ligar
9.4.1.1 Opção(1) Dispositivo
Pede o código/ID do dispositivo que quer ligar e liga-o.
9.4.1.0 Opção(0) Divisão
Pede o nome da divisão que quer ligar todos os dispositivos e liga-os.
9.4.0 Opção(0) Ligar
9.4.1.1 Opção(1) Dispositivo
Pede o código/ID do dispositivo que quer desligar e desliga-o.
9.4.1.0 Opção(0) Divisão
Pede o nome da divisão que quer desligar todos os dispositivo
e desliga-os.

9.3 Opção(3)Criar uma fatura de uma casa.
Pede o nome do proprietário da casa e mostra a sua fatura.

9.2 Opção(2)Atualizar intervalo de tempo
Pede uma data inicial e uma data final para definir o intervalo e atualiza.

9.1 Opção(1) Save/Load do estado
9.1.1 Opção(1) Save
Salva o Estado das casas, fornecedores e das faturas num ficheiro
chamado Estado
9.1.0 Opção(0)Load
Carrega o Estado das casas,fornecedores e das faturas.

9.0 Opção(0) Sair
9.0.1 Opção(1) Sim
O continuar passa a false terminando então a execução da APP.
9.0.0 Opção(0) Quero Voltar
Volta para o Menu Principal.

Exceções:

DeviceNaoExisteException
Esta exception manifesta-se quando um dispositivo não existe na casa.

FaturaNaoExisteException
Esta exception ocorre quando nenhuma fatura encontra-se registada (Não foram efetuadas faturas ao longo do correr do programa ou ,se foi feito um load do estado, no ficheiro não tem nenhuma fatura).

FornecedorNaoExisteException
Esta exception manifesta-se quando o comercializador/fornecedor em questão não existe.

FornecedorJaExisteException
Esta exception ocorre quando o fornecedor que tentamos criar já existe.

CasaNaoExisteException
Esta exception manifesta-se quando a casa em questão não existe.

DivisaoNaoExisteException
Esta exception ocorre quando a divisão em questão não existe

DivisãoJaExisteException
Esta exception manifesta-se quando a divisão que tentamos criar já existe naquela casa.

NúmeroInvalidoException
Esta exception ocorre quando o número que foi inserido não tem uma opção associada.

FaturaNaoExisteNoIntervaloException
Esta exception manifesta-se quando não existe um fatura registada naquele intervalo de tempo.

IntervaloInvalidoException
Esta exception ocorre quando o Intervalo de tempo inserido é inválido, a primeira data inserida (a data do começo do intervalo) é mais recente que a segunda data inserida(a data do final do intervalo).

Observações:

Na aplicação é possível voltar de qualquer opção para o menu principal, inserindo um formato errado do que é pedido, isto só não é possível no Cria Fornecedor pois ele cria um Fornecedor com qualquer string e no Atualizar a Data mas esse é intencional pois queríamos que obrigasse a meter uma data Válida antes de voltar ao menu principal.

Para nós cada proprietário tem só uma casa, tendo isto em conta, é possível criar uma Casa com um nome de um proprietário que já tem uma Casa, interpretamos que o proprietário mudou de casa em vez de o permitir que tenha uma nova.

Nas estatísticas Opção(4)"Casa que gastou mais num intervalo" e na
Opção(3) "Comercializador/Fornecedor com maior volume de faturação", caso haja mais do que um em qualquer destas opções, ele escolhe o que aparecer primeiro na travessia do hashmap das Faturas.

Na criação de um SmartBulb a tonalidade tem de ser obrigatoriamente "Cold", "Neutral", "Warm" , nós colocamos uma condição no parseSmartBulb de forma a que caso outra coisa seja escrita neste campo, ele crasha, mas o nosso código está estruturado de uma forma que caso hajam estes "crashs" nos tipos etc de input ele volta para o menu principal e faz print de uma mensagem a avisar sobre o formato estar incorreto.

Na Criação de uma Casa e de um Fornecedor , é possível estes terem nomes que consistem em números, não interpretamos como um problema, por mais improvável que seja na vida real , há de tudo neste mundo.

Diagrama de Classes:

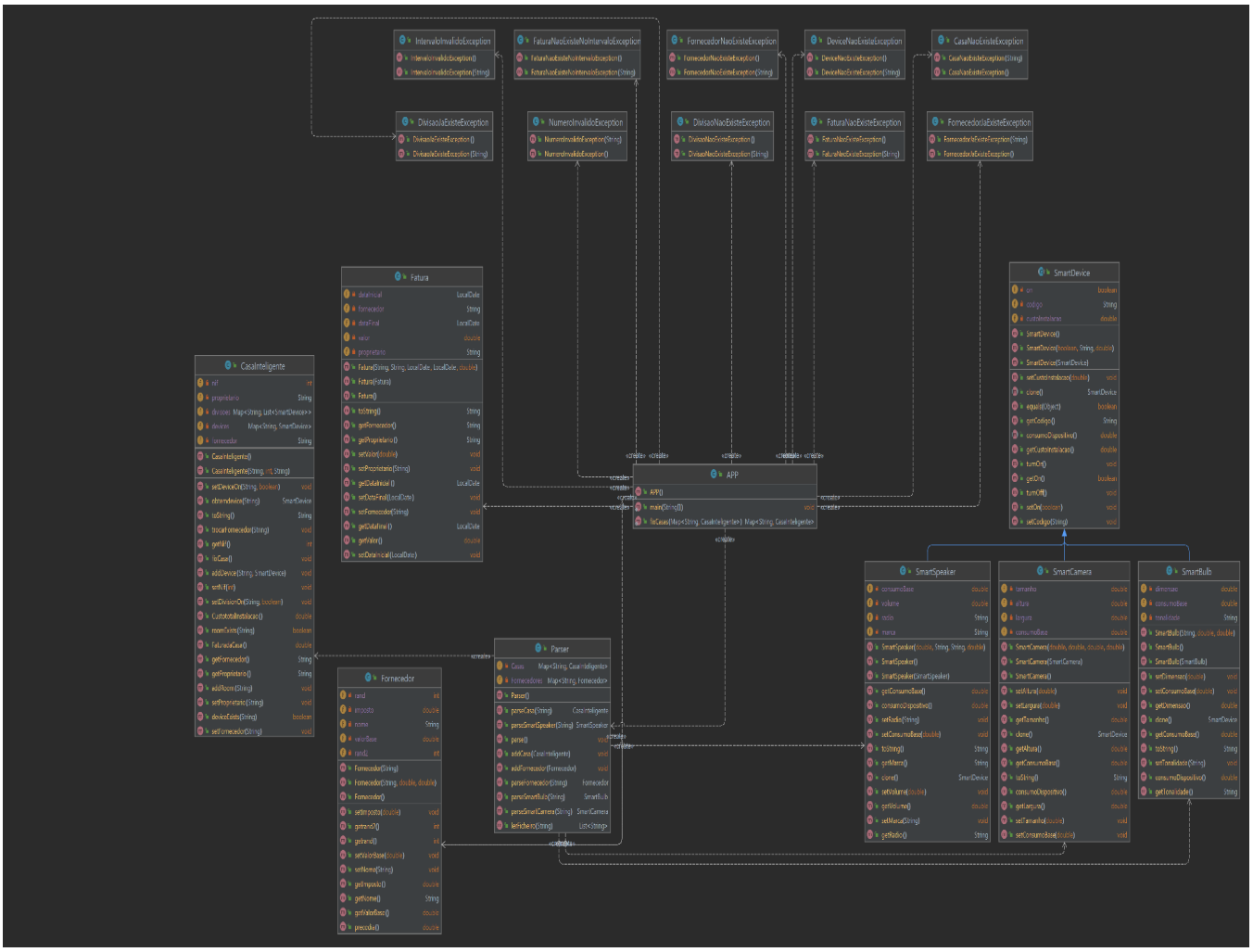

Diagrama de Classes gerado pelo IntelliJ.

(Tendo em conta que os dados não são bem visíveis, colocamos na Pasta “Relatório” a imagem do diagrama)

Conclusão:

Com a finalização deste trabalho, fomos capazes de consumar a maioria dos pontos que nos foram propostos, culminando num sistema capaz de executar comandos como criação de dispositivos, casas e fornecedores, ligar e desligar dispositivos/divisões, ser capaz de calcular toda a estatística sobre o estado do programa requerida, durante um período de tempo introduzido pelo utilizador, entre outros.
Apesar disto, tivemos dificuldades em alguns tópicos, mas aquele que se salientou foi sem dúvida a APP, precisamos de investir muito tempo para debbuging , houve vários casos frustrantes, o “ == ” não funcionava e tínhamos de usar equals e o pior de todos foi o problema que tivemos com o processo de ligar Dispositivos, ficamos muito tempo a tentar descobrir o problema mas eventualmente conseguimos e criamos o fixCasa() e fixCasas().

(Ps: Não nos podemos esquecer também o facto de o código da APP que tínhamos desenvolvido na altura ter desaparecido ao colocá-lo numa package, tivemos de investir um tempo extra para fazer o código novamente, foi um fenómeno muito estranho).