Instituto Superior de Engenharia do Porto

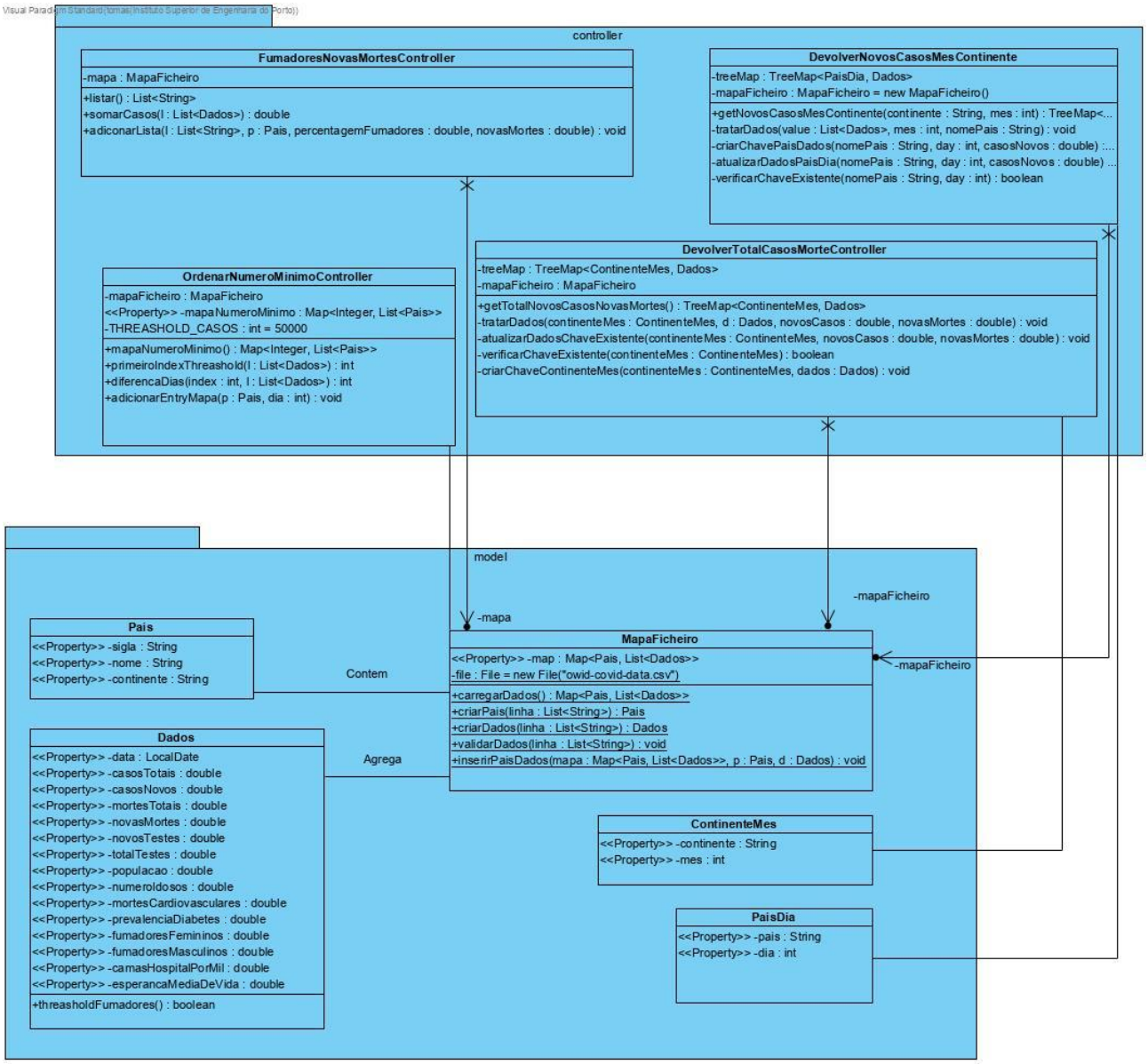
ESINF - 1º TRABALHO PRÁTICO

Gabriel Pelosi - 1180017

Rui Mendes – 1170963

TUMA: 2DN

Diagrama de classes



Funcionalidades

Problema:

Carregar e guardar a informação relativa aos países e respetivos dados da pandemia COVID-19 a partir do ficheiro de texto fornecido.

Interpretação do problema:

Ao analisar o ficheiro podemos reparar que a informação nele contida se encontra separada em dois tipos. O primeiro é a localização, nomeadamente o nome do país em causa, com o seu respetivo código e continente. O segundo tipo são os dados propriamente ditos, cada linha, para além da localização, tem uma data e as variáveis relevantes ao problema.
Podemos reparar então que qualquer estrutura que usemos, para efeitos de um organizado agrupamento de dados, deveria guardar apenas uma instância de cada país, adicionalmente, as várias variáveis contidas no ficheiro deveriam ser guardadas, tendo em conta a data a que pertencem e o Pais em que foram registados.

Nossa solução:

Na decisão de qual estrutura de dados usar para guardar os dados, decidimos num HashMap do seguinte tipo:

```
private Map<Pais, List<Dados>> map;
```

Optamos por esta estrutura visto cumprir os pontos definidos na nossa interpretação, nomeadamente: Permite-nos evitar a repetição de países; permite-nos guardar todos os dados de um pais e associa-los ao mesmo; Através da implementação de List para os valores do mapa, podemos garantir que todas as linhas do ficheiro estarão isoladas umas das outras; Por último, a interface map é extremamente poderosa, fornecendo formas ágeis de aceder e manipular a estrutura, mantendo uma complexidade de código baixa.
O resto da implementação desta funcionalidade segue uma direção simples. Utilizamos um scanner para ler todas as linhas do ficheiro. Em seguida, e através dos seguintes métodos, criamos o Pais e os Dados com o texto da linha.

```
public static Pais criarPais(List<String> linha) {

    return new Pais (linha.get(0).replaceAll( regex: "\"",  replacement: ""),
            linha.get(1).replaceAll( regex: "\"",  replacement: ""),
            linha.get(2).replaceAll( regex: "\"",  replacement: ""));

}

/** Cria os dados com os últimos 15 valores da linha
 * @param linha lista de strings com a linha do ficheiro
 * @return retorna os dados
 */
public static Dados criarDados(List<String> linha) {
    validarDados(linha);
    return new Dados(linha.get(3), Double.parseDouble(linha.get(4)), Double.par
            Double.parseDouble(linha.get(7)), Double.parseDouble(linha.get(8)),
            Double.parseDouble(linha.get(10)), Double.parseDouble(linha.get(11)
            Double.parseDouble(linha.get(13)), Double.parseDouble(linha.get(14)
            Double.parseDouble(linha.get(16)), Double.parseDouble(linha.get(17)
}
```

Verificamos que qualquer “NA” lido no ficheiro é substituído por “0”, para fins de contas futuras.

```java
/** Verifica que qualquer campo com "NA" seja substituido com "0"
 * @param linha lista de strings com a linha do ficheiro
 */
public static void validarDados(List<String> linha) {
    ListIterator<String> it  = linha.listIterator();
    String s;
    while(it.hasNext()){
        s = (String) it.next();
        if(s.equalsIgnoreCase( anotherString: "NA")){
            it.set("0");
        }
    }
}
```

E no final inserimos os dados no Mapa.

```java
/** Insere novos calores criados no mapa
 * @param mapa mapa onde se vão inserir os dados
 * @param p Um pais, chave do mapa
 * @param d, Lista de dados, value do mapa
 */
public static void inserirPaisDados(Map<Pais, List<Dados>> mapa, Pais p, Dados d) {
    List<Dados> dados;
    if(!mapa.containsKey(p)) {
        dados = new LinkedList<>();
        dados.add(d);
        mapa.put(p, dados);
    }else{
        dados = mapa.get(p);
        dados.add(d);
    }

}
```

Todos os métodos e variáveis até agora descritos estão contidos na classe MapaFicheiro, uma classe pertencente ao pacote Model, cuja única responsabilidade é a criação do mapa e a importação das linhas do ficheiro para este.

Melhoramentos possíveis:

De acordo com a nossa interpretação do problema, acreditamos a nossa implementação da interface Maps como a estrutura de dados mais eficiente para guardar e recolher informação do ficheiro, de todas aquelas contidas da Java Collections Framework.

Na obstante, reconhecemos a possibilidade de haver outros métodos de leitura mais eficiente do que aqueles que nós usamos, e uma melhoria deste trabalho incluiria outras abordagens relativas a esta questão.

Problema:

Apresentar uma lista de países ordenados por ordem crescente do número mínimo de dias que foi necessário para atingir os 50.000 casos positivos.

Interpretação do problema:

Na nossa interpretação o problema proposto implica que vai haver um número de 0 ou mais países associados a cada dia. Daqui podemos deduzir que não haveria dias repetidos. Qualquer estrutura de dados que usemos para guardar esta informação vai ter de ter isso em consideração.

Nossa solução

Durante o nosso estudo da Java Collections Framework, uma das coisas que podemos reparar é que o tipo de Objeto TreeMap organiza as suas chaves de acordo com a implementação do método compareTo() da class Object. Decidimos então guardar a informação na seguinte estrutura:

```
private Map<Integer, List<Pais>> mapaNumeroMinimo;
```

Achamos esta a melhor solução, pois a classe Integer fará o TreeMap ordenar as chaves por ordem decrescente, isto significa que não teremos de ordenar os dias manualmente, simplificando bastante o código. Isto é possível pois, tal como referido acima, os dias são não repetíveis.
Para inserir os dados no mapa acima é primeiro necessário manipular os dados do ficheiro, ou seja, o mapa utilizado para solucionar a funcionalidade 1. Solucionamos esse obstáculo do seguinte modo:

Primeiro iteramos sobre o entrySet do mapa.

```
for(Map.Entry<Pais, List<Dados>> entry: mapaPaises.entrySet()){
    index = primeiroIndexThreashold(entry.getValue());
    dia = diferencaDias(index, entry.getValue());
    if(dia >= 0){
        adicionarEntryMapa(entry.getKey(), dia);
    }
}
```

Depois, dentro desse ciclo, iteramos sobre os dados de um determinado Pais e obtemos o primeiro index com casos acima de 50.000.

```
/** retorna o primeiro index com totalCasos acima de 50.000
 * @param l, lista com os dados de um determinado pais
 * @return inteiro com o index
 */
public int primeiroIndexThreashold(List<Dados> l) {
    for(Dados d : l) if (d.getCasosTotais() > THREASHOLD_CASOS){
        return l.indexOf(d);
    }
    return -1;

}
```

Em seguida, fazemos a diferença entre a data desse index e a primeira data disponível para esse país.

```
public int diferencaDias(int index, List<Dados> l){
    if(index < 0) return -1;
    return (int) ChronoUnit.DAYS.between(l.get(0).getData(), l.get(index).getData());
}
```

Em seguida inserimos o dia e o pais no mapa, se o primeiro destes for válido.

```
public void adicionarEntryMapa(Pais p, int dia){
    List<Pais> l;
    if(mapaNumeroMinimo.containsKey(dia)){
        l = mapaNumeroMinimo.get(dia);
        l.add(p);
    }else{
        l = new LinkedList<>();
        l.add(p);
        mapaNumeroMinimo.put(dia, l);
    }
}
```

Melhoramentos possíveis:

Acreditamos a nossa implementação da interface Map como a estrutura mais eficiente para guardar a informação, principalmente por, tal como referido acima, guardar as chaves do tipo Integer por ordem crescente, independentemente de ordem de inserção. Num possivelmente melhoramento futuro o nosso foco principal provavelmente iria incidir sobre uma iteração mais eficiente dos dados, de modos a baixar a complexidade do código adotando outros métodos que não o ciclo foreach.

Problema:

Devolver o total de novos_casos/novas_mortes por continente/mês, ordenado por continente/mês.

Interpretação do problema:

A interpretação desse tópico foi obtida diante ao fato de que seria necessário manipular todos os dados do ficheiro. Diante isso, seria necessário criar uma estrutura para agrupar esses dados de forma correta e para cada dado analisado, ou o programa salvaria ele com sua chave ou faria a atualização do mesmo através de sua chave.

Nossa solução:

Para agrupar os dados de acordo com o enunciado, foi criada uma estrutura TreeMap para guardar de acordo com cada Continente e cada Mês (Atributos da classe ContinenteMes) um objeto da classe Dado que representaria somente os novos casos e as novas mortes totais.

```
private TreeMap<ContinenteMes, Dados> treeMap;
```

Tendo o Continente e o mês como atributos da chave, eliminaríamos verificações desnecessárias dos dados para somá-los ou criá-los na estrutura, dessa forma, será apenas necessário executar o containsKey(), e dependendo do valores de retorna, é criada uma nova chave com seus dados iniciais, ou é feita uma busca no mapa para atualizar os valores respetivos daquela chave.
No primeiro método é feito a busca inicial, para cada país, é feita a analise dos seus dados.

```
public TreeMap<ContinenteMes, Dados> getTotalNovosCasosNovasMortes(){

    Map<Pais, List<Dados>> map = mapaFicheiro.getMap();
    treeMap = new TreeMap<>();
    String continente;
    int month;
    double novosCasos;
    double novasMortes;
    ContinenteMes continenteMes;

    for(Map.Entry<Pais, List<Dados>> e : map.entrySet()){
        continente = e.getKey().getContinente();
        for(Dados d : e.getValue()){
            month = d.getData().getMonthValue();
            novosCasos = d.getCasosNovos();
            novasMortes = d.getNovasMortes();
            continenteMes = new ContinenteMes(continente, month);
            tratarDados(continenteMes, d, novosCasos, novasMortes);

        }
    }

    return treeMap;
```

Depois, é verificado se o continente e o mês do país analisado existe no mapa.

```
private void tratarDados(ContinenteMes continenteMes,
                         Dados d,
                         double novosCasos,
                         double novasMortes) {

    if(verificarChaveExistente(continenteMes))
        atualizarDadosChaveExistente(continenteMes,
                novosCasos,
                novasMortes);
    else
        criarChaveContinenteMes(continenteMes,
                new Dados(d.getCasosNovos(),
                        d.getNovasMortes()));
}
```

Caso exista, ou seja, um objeto ContinenteMes com os valores analisado esteja presente no mapa, é feito um get do Dado que aquela chave representa e assim, seus valores são atualizados.

```
private void atualizarDadosChaveExistente(ContinenteMes continenteMes,
                                          double novosCasos,
                                          double novasMortes) {

    Dados dados = treeMap.get(continenteMes);
    dados.setCasosNovos(dados.getCasosNovos() + novosCasos);
    dados.setNovasMortes(dados.getNovasMortes() + novasMortes);
}
```

Caso a chave não exista, é criada uma nova junto com seus respetivos dados.

```
/**Verifica se a chave existe no mapa.
 * @param continenteMes Chave a ser verificada.
 * @return true se a chave recebida existir no mapa, falso se a chave não existir.
 */
private boolean verificarChaveExistente(ContinenteMes continenteMes){
    return treeMap.containsKey(continenteMes);
}

/**Cria uma chave e seus respectivos dados no mapa.
 * @param continenteMes chave a ser criada no mapa.
 * @param dados dados relativos a chave a serem criados no mapa
 */
private void criarChaveContinenteMes(ContinenteMes continenteMes, Dados dados){
    treeMap.put(continenteMes,dados);
}
```

Melhoramentos possíveis:

Um possível melhoramento seria diminuir o número de operações do método, encontrar uma forma de utilizar somente um ciclo for, dessa forma o algoritmo estaria mais otimizado e teria uma menor complexidade.

Problema:

Devolver para cada dia de um determinado mês e para um dado continente o total de novos casos de cada país.

Interpretação do problema:

Para solucionar o terceiro tópico do enunciado foi implementada a classe DevolverNovosCasosMesContinente para tratar os dados da forma pedido e agrupá-los numa estrutura adequada. Essa estrutura é um TreeMap, tendo como chave um objeto do tipo PaisDia, o qual guarda o continente e o dia dos dados analisados. Esse tipo de chave foi pensado pois os países que não pertençam ao continente pedido e os dados que não sejam do mês pedido, não devem ser analisados.

Nossa Solução:

Esse tipo de solução foi apresentado para poupar operações desnecessárias, dessa forma é feita a verificação da chave com um único containsKey() ao invés de operações If.

E o objeto chave do mapa possui apenas o nome do país e o dia, pois o continente e o mês passados por parâmetro não precisam ser verificados no mapa, apenas uma verificação inicial dos dados é necessária, isso por que, caso o país não seja do continente desejado ou o dado não seja do mês apontado, esses valores não precisam ser analisados nunca.

Inicialmente é feita a validação do continente.

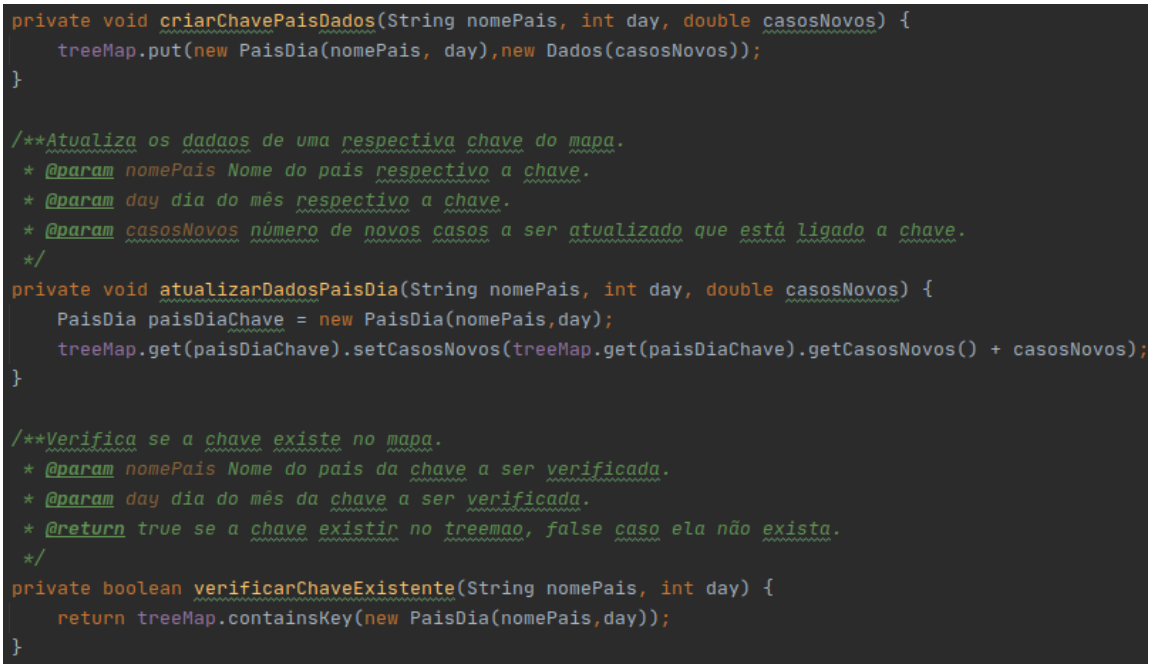
```
public TreeMap<PaisDia, Dados> getNovosCasosMesContinente(String continente, int mes){
    Map<Pais, List<Dados>> paisDadosMap = mapaFicheiro.getMap();

    for (Map.Entry<Pais, List<Dados>> entry: paisDadosMap.entrySet()){
        if (entry.getKey().getContinente().equalsIgnoreCase(continente)){
            tratarDados(entry.getValue(), mes, entry.getKey().getNome());
        }
    }
    return treeMap;
```

Após isso, é feita a validação do mês. Caso o mês e o continente sejam validos, é feita a verificação se a chave existe ou não no mapa, ou seja, se aquele país e aquele dia já foram inicializados.

```
private void tratarDados(List<Dados> value, int mes,String nomePais) {
    for (Dados d : value){
        if (d.getData().getMonthValue() == mes){
            if (verificarChaveExistente(nomePais,d.getData().getDayOfMonth()))
                atualizarDadosPaisDia(nomePais,d.getData().getDayOfMonth(), d.getCasosNovos());
            else
                criarChavePaisDados(nomePais,d.getData().getDayOfMonth(), d.getCasosNovos());
```

Caso a chave não exista ela é criada com seus dados iniciais, caos ela exista, seus dados são apenas atualizados através do get para buscar o objeto daquela chave.



```java
private void criarChavePaisDados(String nomePais, int day, double casosNovos) {
    treeMap.put(new PaisDia(nomePais, day),new Dados(casosNovos));
}

/**Atualiza os dadaos de uma respectiva chave do mapa.
 * @param nomePais Nome do pais respectivo a chave.
 * @param day dia do mês respectivo a chave.
 * @param casosNovos número de novos casos a ser atualizado que está ligado a chave.
 */
private void atualizarDadosPaisDia(String nomePais, int day, double casosNovos) {
    PaisDia paisDiaChave = new PaisDia(nomePais,day);
    treeMap.get(paisDiaChave).setCasosNovos(treeMap.get(paisDiaChave).getCasosNovos() + casosNovos);
}

/**Verifica se a chave existe no mapa.
 * @param nomePais Nome do pais da chave a ser verificada.
 * @param day dia do mês da chave a ser verificada.
 * @return true se a chave existir no treemao, false caso ela não exista.
 */
private boolean verificarChaveExistente(String nomePais, int day) {
    return treeMap.containsKey(new PaisDia(nomePais,day));
}
```

Melhoramentos possíveis:

Um possível melhoramento seria diminuir o número de operações do método, encontrar uma forma de utilizar somente um ciclo for, dessa forma o algoritmo estaria mais otimizado e teria uma menor complexidade. Mas para isso ser possível seria necessário mudar a estrutura inicial dos dados analisados, pois com essa estrutura é sempre necessário dos ciclos para analisar dados.

Problema:
Devolver numa estrutura adequada, todos os países com mais de 70% de fumadores, ordenados por ordem decrescente do número de novas mortes.

Interpretação do problema:
Achamos este bastante direto, o problema consistiria principalmente numa forma eficiente de ordenar os Países com atenção ao número de novasMortes.

Nossa solução:
A partir do enunciado podemos concluir que todas as variáveis envolvidas vão ter uma relação de Um Para Um entre elas. Um pais, com uma percentagem, e uma soma de novasMortes. Logo, achamos que uma estrutura do tipo mapa não seria apropriada. Restava-nos a interface Set e List. Optamos pela última, pois esta proporciona acesso à interface ListIterator que facilita a ordenação.

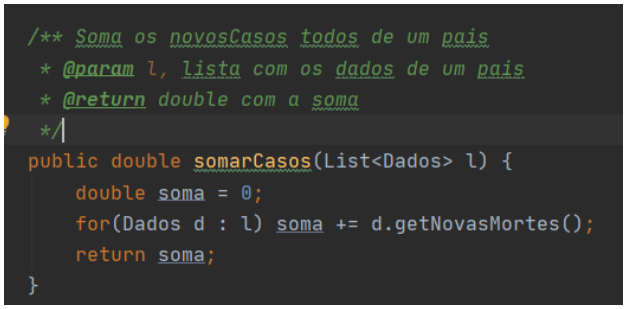
Começamos por iterar sobre o entry set do mapa com os dados do ficheiro. Em seguida, para cada entrySet, verifica se o país ultrapassou o limite de fumadores estabelecido no enunciado (70%).

```java
/** Retorna a lista com nomes de Paises, percentagens de fumadores e soma de todos os novos casos
 * @return lista do tipo String, cada String é um pais e os seus dados
 */
public List<String> listar(){
    Map<Pais, List<Dados>> m = mapa.getMap();
    List<String> l = new LinkedList<>();
    double novasMortes;
    double percentagemFumadores;
    for(Map.Entry<Pais, List<Dados>> e : m.entrySet()){
        if(e.getValue().get(0).threasholdFumadores()){
            percentagemFumadores = e.getValue().get(0).getFumadoresFemininos() + e.getValue().get(0).getFumadoresMasculinos();
            novasMortes = somarCasos(e.getValue());
            adiconarLista(l, e.getKey(), percentagemFumadores, novasMortes);
        }
    }
    return l;
}
```

No caso de o Pais ter o número mínimo de fumadores necessário, irá obter esse número, em seguida vai somar todas as novasMortes desse país e adicionar os valores relevantes à List<String>, através dos métodos abaixo.

```java
/**Adiciona uma nova String à lista, index de inserção depende do número de novosCasos
 * @param l, lista com os paises ordenados
 * @param p, Pais a inserir na lista
 * @param percentagemFumadores, percentagem a inserir na lista
 * @param novasMortes, novos casos a inserir na lista
 */
public void adiconarLista(List<String> l, Pais p, double percentagemFumadores, double novasMortes) {
    ListIterator<String> i = l.listIterator();
    String[] linha = new String[3];
    boolean colocado = false;
    double d;
    while (i.hasNext() && !colocado){
        linha = i.next().split( regex: ",");
        d = Double.parseDouble(linha[2]);
        if(novasMortes > d){
            i.previous();
            i.add(p.getNome() + "," + percentagemFumadores + "," + novasMortes);
            colocado = true;
        }
    }
    if(!colocado) i.add(p.getNome() + "," + percentagemFumadores + "," + novasMortes);

}
```



```java
/** Soma os novosCasos todos de um pais
 * @param l, lista com os dados de um pais
 * @return double com a soma
 */
public double somarCasos(List<Dados> l) {
    double soma = 0;
    for(Dados d : l) soma += d.getNovasMortes();
    return soma;
}
```

Possíveis melhoramentos:

Achamos que certos métodos usaram formas ineficientes e voláteis para manipularem o mapa com os dados do ficheiro a modos de obter a funcionalidade. Melhorar este código possivelmente consistiria em reescrever toda a manipulação do mapa ou até, muito atentamente, mudar a estrutura em que guardamos os dados do ficheiro.