RELATÓRIO DE ESTRUTURAS DE INFORMAÇÃO

Projeto 1 – Países/Covid-19

Síntese

Ao longo do documento iremos demonstrar o modo como gerimos a informação que nos é fornecida através de um ficheiro de Excel, bem como a explicação dos nossos métodos para responder aos diferentes requerimentos que a aplicação necessitaria ter, tendo em conta a eficiência.

Realizado por:
Fábio Fernandes 1191430
Bárbara Pinto 1191507

Introdução

Na disciplina de Estruturas de Informação foi-nos proposto, para primeiro projeto, desenvolver uma aplicação que permita gerir a informação de vários países relacionada com a pandemia COVID-19, assim como aplicar os conhecimentos adquiridos na Java Collections Framework para responder eficientemente aos requisitos que nos foram colocados ao longo deste projeto.

1. Requisito nº1 – Carregar e guardar informação do ficheiro Excel.

Para este primeiro procedimento decidimos armazenar a informação recolhida do ficheiro Excel que nos foi disponibilizado na plataforma moodle em 5 classes, sendo elas: *World*, *Continent*, *Country*, *Death*, *Case* e *Smoker*.

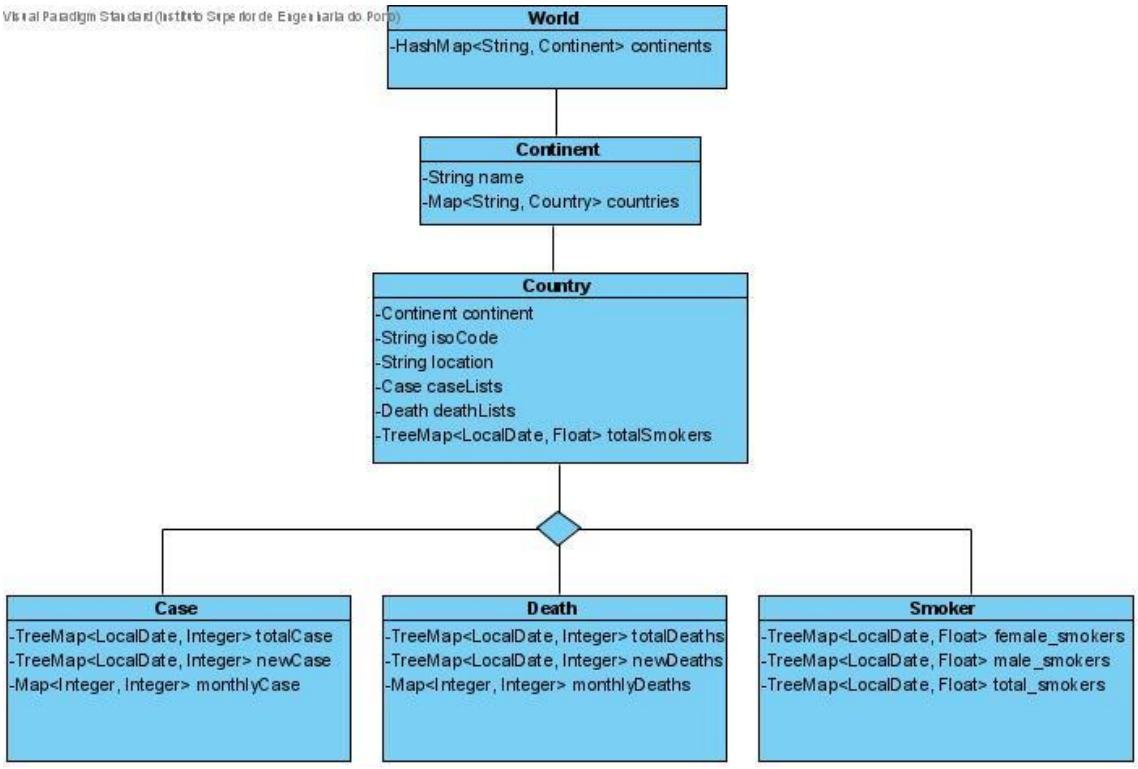

*Figura 1: Estrutura de classes e atributos **funcionais** do projeto.*

De notar que só estão representadas na figura 1 as classes e os atributos que são utilizados para a implementação dos requisitos do trabalho, porém para toda a informação que não está a ser utilizada, foram criadas 2 classes *RiskFactors* (que inclui cardiovasc_death_rate e diabetes_prevalence) e *DemographicIndicators* (que inclui population, aged_65_older, life_expectancy e hospital_beds_per_thousand), de forma a tratar e armazenar a informação. Estas classes são ainda atributos adicionais da classe *Country*.

A estrutura deste programa foi concebida com o objetivo de encapsular informação em grupos facilmente acessíveis. O isolamento da informação dos casos, mortes e fumadores de cada país permite que o cálculo dos valores necessários e a geração de estatísticas seja da responsabilidade de "cada país", o que torna a quantidade de dados a manusear de uma só vez mais reduzida, bem como reduz a necessidade de algoritmos de busca por dados de uma entidade específica.

Como um pequeno exemplo, de maneira a verificar o número de casos positivos de um determinado país, teríamos de seguir o caminho: *World* → *Continent* → *Country* e por fim aceder à classe *Case* para recolher a informação pretendida.

Para atingir uma organização eficiente no contexto desta organização de classes, recorremos a *Maps* para obter uma pesquisa mais eficiente, através da indexação de valores a chaves (*Keys*). Neste caso, a *Key* das classes que manipulam os casos, as mortes e os fumadores são a data, por representarem valores únicos e muito relevantes para a elaboração das listagens. Dessa maneira, dado um país e uma data específica (ou até um mês específico), a obtenção dos valores relevantes é feita de forma instantânea e eficiente.

De notar que as classes *Continent* e *World* são formadas por *Maps* de continentes e países, respetivamente, indexados por uma *String* (o nome do continente ou o código do país, neste caso) para tornar mais eficiente a verificação de objetos já lidos do ficheiro. Por exemplo, um dado país tem tantas linhas no ficheiro quanto os seus registos diários, mas não existe necessidade de criar repetidamente um objeto *Country* por cada uma dessas linhas, logo faz-se uma busca pelo seu *"iso code"* como *Key* e rapidamente se verifica se já foi gerado.

2. *Requisito nº2 – Apresentar uma lista de países ordenados por ordem crescente do número de dias para atingir os 50 000 casos positivos de COVID-19.*

Acedendo à lista de continentes contida na classe *World*, podemos entrar na classe *Continent*, habilitando-nos a obter a localização de todos os países nele contidos. Assim, apenas necessitamos de aceder à classe *Case*, que contém um *TreeMap* com a informação organizada por data (*Key*) com o número total de casos (*Values*).

A busca pela data em que o total de casos ultrapassa pela primeira vez os 50.000 é iterativa – complexidade *O(n)* – no entanto, é sempre feita à priori uma verificação do total de casos mais recente. Caso este total não atinja os 50.000, o algoritmo não efetua a busca iterativa, poupando tempo de execução.

Tendo a data onde a condição se verifica (atingir 50.000 casos), só necessitamos de fazer uma simples subtração entre essa data e a data do primeiro registo.

3. ***Requisito nº3 – Apresentar o total de novos casos assim como novas mortes por continente/mês***

Neste caso em específico necessitamos de uma pequena particularidade: encontrar a data de início bem como a data final de cada mês – com especial cuidado para registos com meses incompletos –, para possibilitar uma contagem eficiente de todos nos novos casos por mês.

Em adição decidimos criar atributos especializados nas classes *Case* e *Death,* que reúnem as somas dos novos casos e agregam por mês num *Map<K,V>* com K = Nº do mês e V = Novos casos/mortes do mês.

Descobrimos que esta solução se mostrou bastante útil, uma vez que os valores mensais, sendo necessários nesta e noutras listagens (no requisito 4, por exemplo) passavam a ser calculados e armazenados uma única vez ao longo da execução do programa.

Sendo assim, este ponto tornou-se simples, percorrendo os todos países dos diversos continentes, bastava o número do mês pretendido para retirar eficientemente o respetivo valor, por ordem alfabética e cronológica, das suas mortes e novos casos.

4. ***Requisito nº4 – Devolver para cada dia de um determinado mês e para um dado continente, os países por ordem decrescente de casos positivos.***

Toda a organização previamente concebida facilitou a conquista deste requisito, isto porque todos os métodos bem como tratamento de informação já estavam delineados, no entanto é necessário estruturá-la da maneira pretendida e eficientemente.

Acedemos à classe *World* e procuramos pelo continente desejado, procura essa feita através de *Keys* no formato de *String* como já referido anteriormente, obtendo assim a lista de países contida nele. Posteriormente, o mês introduzido permite-nos através dos atributos especializados obter os valores dos casos positivos para cada dia do mês. Sendo assim organizamos todos os países num *ArrayList* ordenados de forma decrescente pelo número de casos positivos, utilizando para o efeito um *Comparator* e fazendo uso da classe *Collections* e seu *sorting algorithm* nativo, que se constitui eficiente por natureza.

5. ***Requisito nº5 – Devolver numa estrutura adequada, todos os países com mais de 70% de população de fumadores, ordenados por ordem decrescente do número de novas mortes.***

Para responder a esta etapa, criamos também um atributo especializado com o total de fumadores, reunindo a soma dos fumadores femininos e masculinos e, consequentemente, poupando a repetição de cálculos desnecessários ao longo da execução do programa. Posto isto, estabelecemos para o país a restrição de 70.0 (valor

em percentagem que nunca pode passar os 100 valores) através de um método booleano.

Para a ordenação da informação foi utlizado o mesmo método que descrito no requisito anterior.

Conclusão

Com este trabalho ficamos a conhecer de forma mais aprofundada as ferramentas da *Java Collections Framework*, aplicando esses conhecimentos através do uso de *Maps* e indexação *Key -> Value* para tornar pesquisas mais eficientes, assim como o uso de *TreeMaps* e o método *Collections.sort* para garantir a rápida ordenação de dados.

Podemos concluir que as ferramentas da *Java Collections Framework* realmente são boas para pôr em prática porque não só conseguem proporcionar uma melhor organização de toda a informação, como podem melhorar significativamente a eficiência do programa que estamos a construir.