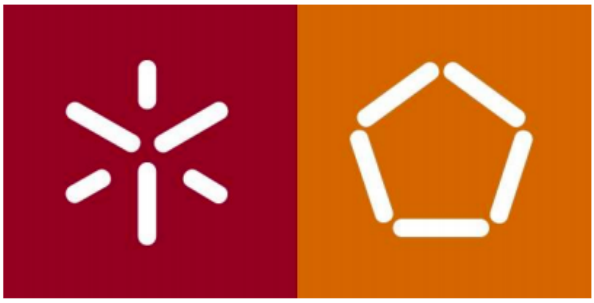

Universidade do Minho

Mestrado Integrado em Engenharia Informática

31/05/2020

Laboratórios de Informática III
Projeto em Java

António Gonçalves (A85516)
Ricardo Costa (A85851)
Eduardo Conceição (A83870)

Contents

1 Introdução

Neste relatório iremos explicar as opções tomadas ao longo da realização da parte Java do projeto da unidade curricular de Laboratórios de Informática III.

Tal como o projeto em C, o objetivo deste trabalho é criar um Sistema de Gestão de Vendas (SGV), capaz de ler informação de ficheiros que contêm clientes, produtos e vendas. Após ler estes ficheiros, o utilizador poderá pedir várias *queries* para visualizar a informação agora presente no SGV. Nesta fase, as *queries* estão divididas em consultas estatísticas e consultas interativas.

Além disso, é também possível armazenar a informação presente nas nossas estruturas de dados em ficheiro em formato binário e ler também essa mesma informação.

Do mesmo modo da parte C, nesta segunda fase vai ser necessário criar as estruturas necessárias para armazenar a informação contida nos ficheiros, para depois se poder realizar consultas e obter várias informações relativamente aos dados.

2 Resolução

2.1 Diagrama de Classes

As classes do nosso projeto estão organizadas de acordo com o seguinte diagrama de classes.

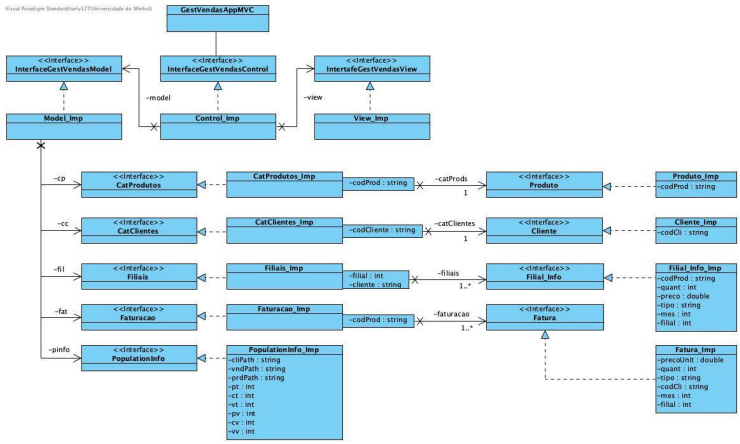

Figure 1: Diagrama de Classes do nosso projeto

Assim, a classe *Model_Imp* implementa a interface *InterfaceGestVendasModel* e contém as várias estruturas de dados principais onde está armazenada toda a informação extraída dos ficheiros (*CatProdutos*, *CatClientes*, *Filiais*, *Faturacao*) e também *PopulationInfo*, onde está armazenada infomação relativa aos ficheiros lidos, como o seu *path*, o número total de linhas lidas para e número total de linhas válidas para cada ficheiro.

As classes *CatProdutos_Imp*, *CatClientes_Imp* e *Faturacao_Imp* possuem um *Map* cuja chave é o código de produto, código de cliente e código de produto novamente. O valor correspondente à chave é a interface que tem a ligação na imagem acima, que contém as informações necessárias para essa estrutura. A classe *Filial_Imp* contem um *Map* de *Maps*, correspondente a cada filial. Daí, a primeira chave ser um inteiro correspondente a filial e a segunda chave uma *string* correspondente ao código de cliente. O valor correspondente a esta segunda chave é a interface *Filial_Info* que é implementada por uma classe que contem as informações necessárias para esta estrutura.

De seguida temos a classe *View_Imp* que implementa a interface *InterfaceGestVendasView* que serve o propósito de estabelecer toda a comunicação com o utilizador, a nível de apresentar resultados ou pedir informações.

Por fim, temos a classe *Control_Imp* que implementa a interface *InterfaceGestVendasControl* que trata de estabelecer o fluxo de execução do projeto e comunicar com a *View* e o *Model*.

A classe final que contém a *main*, *GestVendasAppMVC*, inicializa instâncias do *Control*, *Model* e *View* e pede ao *Control* para começar a execução do programa.

2.2 Queries Estatísticas

2.2.1 Query 1

Para esta *query*, em que devemos mostrar informações sobre os ficheiros lidos, como nome, número de linhas válidas e linhas totais, fizemos uma classe chamada *Population_Info*. Esta classe guarda informação persistente no *Model_Imp*, e é preenchida uma vez que os ficheiros são lidos. Depois, uma vez que a *query* é chamada, apenas imprimimo-la no *stdout*.

2.2.2 Query 2

Nesta *query*, devemos mostrar o total faturado, total de produtos comprados e de clientes diferentes que compraram em cada mês. Para este efeito, vamos ter de percorrer a Faturação para obter as informações relativas aos produtos e as Filiais para as informações dos Clientes. Na Faturação, percorremos o *Map* e vamos somando a faturação de cada produto em cada mês e o número de vezes que o mesmo foi comprado e guardando num objeto auxiliar *StatsQ2*. Depois, nas Filiais, fazemos um processo semelhante para cada cliente, verificando se ele fez compras naquele mês e, caso tenha feito, se já tinha sido tido em conta. Se não, então contabilizamo-lo e adicionamo-lo ao objeto *StatsQ2*. Por fim, quando este processo estvier completo, imprimimos o resultado do m étodo *toString()* da classe *StatsQ2*.

2.3 Queries Interativas

2.3.1 Query 1

Para a *query* 1 é necessário obter a lista dos produtos que nunca foram comprados. Para isto, percorremos o *Map* onde os Produtos estão guardados, guardando os seus códigos numa lista, que devolvemos para o *Model_Imp*. Depois, na Faturação, obtemos a lista dos vários códigos de produto que servem como *key* para o *Map* que constitui a Faturação, e devolvêmo-la para o *Model_Imp*. Neste, as duas listas vão ser comparadas, e cada produto que exista na primeira lista mas não na segunda é adicionado a uma lista adicional, que, no final, será ordenada pela ordem natural das *Strings*, e será impressa, mostrando todos os produtos nunca comprados, vinte a vinte.

2.3.2 Query 2

Na *query* 2, é-nos pedido que calculemos o o número de clientes e de vendas num dado mês, filial a filial ou um valor global. Para isso, dividimos a *query* em duas, dando ao utilizador a opção de qual pesquisa quer efetuar.

Assim, para obter o valor filial a filial, temos uma estrutura que é preenchida nas Filiais, *Query2*, que guarda dois inteiros, um para o número de clientes e outro para o de produtos. Assim, percorremos cada filial e vamos adicionando os valores ao objeto auxiliar para depois poder ser devolvido. Repetimos este processo para todas as filiais.

Para obter valores globais, fazemos o mesmo, mas uma vez só, e com ajuda dum *Map* para não contarmos clientes repetidos.

2.3.3 Query 3

Na *query* 3 é-nos pedido que calculemos o número de compras, de produtos diferentes e o total gasto por um cliente em cada mês. Para isto, percorremos o *Map* das filiais e, para cada objeto *Filial_Info*, verifica se o mês dele já foi visto. Se sim, verificamos se o produto já foi visto. Se não, adiciona os valores a uma estrutura auxiliar *Query3* e adiciona o produto a um *Map* auxiliar. Se sim, apenas alteramos as informações que já estão guardadas. Caso o mês em questão não tiver sido visto ainda, adicionamos mais uma entrada ao *Map* auxiliar.

No final, imprimimos o *Map* que resulta deste processo.

2.3.4 Query 4

Na *query* 4 é-nos pedido que determinemos para cada mês, quantas vezes é que um produto foi comprado, por quantos clientes diferentes e o total faturado.

Sendo assim, perguntámos ao utilizador qual é o código de produto que pretende analisar e criámos uma estrutura auxiliar para podermos armazenar os valores relativos ao que é pedido para um dado mês. De seguida, procuramos o código de produto na estrutura da faturação e percorremos a lista contendo todas as diferentes vendas para esse mesmo produto. Criámos também uma lista de uma classe auxiliar *StringInt* para guardar os clientes e respetivo mes que j á foram verificados, para não contabilizarmos mais que uma vez o mesmo cliente. Assim, em cada iteração verificamos se já contabilizamos ou não esse cliente num dado mês e so contabilizamos se ainda não tiver sido. Relativamente ao número de vezes que foi comprado e total faturado, vão sendo atualizados a cada iteração. Os resultados são guardados numa lista de instâncias da classe auxiliar *Query4*.

2.3.5 Query 5

Para a *query* 5, devemos devolver uma lista com os produtos que um dado cliente comprou e quantas unidades comprou.

Para este efeito, vamos percorrer as Filiais e, quando encontramos o cliente, percorremos as suas compras e, para código que encontramos, verificamos se já o vimos, comparando com os códigos guardados num *Map* auxiliar. Caso já esteja lá, atualizamos o número de vezes que foi comprado. Caso contrário, adicionamos uma nova entrada a esse *Map* com o novo produto e sua quantidade.

Quando este processo acabar, colocamos os resultados numa *List* de objetos *StringInt*, que guardam um inteiro e uma *string*, e ordenamos a lista utilizando o *Comparator-Query5*, que ordena a lista por ordem decrescente de quantidades e, caso sejam iguais, por ordem alfabética. Depois, devolvemos a lista e imprimimo-la.

2.3.6 Query 6

Na *query* 6 é pedido para determinar o conjunto dos **X** produtos mais vendidos, indicando também o número total de clientes distintos que os compraram.

O primeiro passo é pedir ao utilizador quantos produtos pretender ver. De seguida, percorremos a estrutura da faturação e para cada código de produto percorremos a lista correspondente às várias vendas com esse código de produto. Em cada iteração, atualizámos a quantidade, ou seja, o número de unidades vendidas e antes de somar o número de clientes, verifcamos se esse cliente já foi contabilizado, verificando se já existe numa lista criada para o efeito, onde se vai adicionando os clientes que já foram contabilizados.

Estes valores são armazenados numa lista de *StringInt2*, onde é armazenada a informação do código do cliente, número de clientes distintos e quantidade. Esta lista é depois ordenada segundo um *Comparator* criado para o efeito, que a ordena por ordem decrescente do valor da quantidade. Por fim, a esta lista é aplicado o método *sublist* entre os valores 0 e **X** fornecido pelo utilizador. Assim, só é apresentado os **X** produtos mais vendidos.

2.3.7 Query 7

Nesta *query*, é necessário calcular os três clientes que mais dinheiro gastaram em cada filial. Para isto, percorremos as filiais e guardamos para cada cliente a quantidade de dinheiro que ela gastou. Guardamos isto numa estrutura chamada *StringDouble*, que guarda, como o nome sugere, uma *string* e um número real. Depois de percorrermos uma filial, guardamos todos os valores dos clientes dessa filial numa lista, ordenamos a lista com o *ComparatorQuery9*, e guardamos apenas os primeiros tr ês elementos com o método *subList*. Fazendo isto para todas as filiais, devolvemos um *Map*, em que as *keys* são as filiais e o *values* são as listas com os três clientes que mais gastaram.

2.3.8 Query 8

Nesta *query* devemos mostrar os N clientes que mais produtos distintos compraram. Para este efeito, vamos guardar cada cliente das Filiais num *Map*, cujos *values* vão ser a lista dos produtos que comprou. Sempre que encontramos um cliente, adicionámo-lo ao *Map* se ele não existia lá e, sempre que, para esse cliente encontramos um produto, inserimos esse produto se ainda não estava lá. No final, convertemos o *Map* obtido num *Map<String,Integer>*, em que cada *key* é código do produto e o *value* é o tamanho da lista de produtos que estava associado a ele. No final, convertemos este *Map* numa *List* de *StringInt*, objeto que já tínhamos descrito anteriormente, e devolvemos para ser impressa.

2.3.9 Query 9

Nesta *query* é pedido para determinar o conjunto dos **X** clientes que mais compraram um dado produto dado pelo utilizador e para cada um desses clientes, o valor gasto por eles.

Assim, é necessário primeiro pedir ao utilizador qual o código de produto que pretende analisar e quantos clientes deseja ver apresentados.

A seguir encontrámos o código de produto dado na estrutura faturação e percorremos a lista correspondente às varias vendas em que esse código de produto está presente. Para esta *query* foi criado inicialmente um *Map*, cuja chave é um código de cliente e o valor um *StringDouble* (que contém uma *string* e um *double*). Como já foi feito em outras *queries*, verificamos se o código de cliente que estivermos a considerar já foi ou não anteriormente verificado. Se o cliente ainda nao tiver sido visto nenhuma vez e adicionado ao *Map* como chave e o valor e uma instancia de *StringDouble*, que contem novamente o cliente e o valor gasto. Se ja tiver sido verificado antes, o *Map* e atualizado na posição correspondente e o cliente e adicionado a lista de clientes.

Depois deste processo, as instâncias de *StringDouble* do *Map* são armazenadas numa lista e essa lista é ordenada segundo um comparator por ordem decrescente do valor gasto. Por fim, a lista é depois truncada usando o método *substring* e os resultados são apresentados.

2.3.10 Query 10

Para a última *query* é pedido que se determine mês a mês, e para cada mês, filial a filial, a faturação total com cada um dos produtos.

Para isto é preciso percorrer a estrutura da faturação e para código de produto a lista correspondente às várias vendas com esse mesmo produto. Para cada iteração no *Map* da faturação é inicializado uma classe auxiliar *Query10* que contém 3 listas de *doubles* correspondentes às 3 filiais e um código de produto. Assim, em cada iteração da lista de faturas esta instância da classe auxiliar vai sendo atualizada, dependendo da filial em que estivermos (ou seja a filial diz-nos qual lista e que vamos atualizar). O mês dá-nos a posição na lista em que temos de atualizar o valor e então é calculado o novo valor tendo em conta o valor que já estava na posição dessa lista com a soma do cálculo do novo valor (que é a quantidade vezes o preço por unidade).

No fim, esta instância da classe auxiliar *Query10* é adicionada a uma lista e o mesmo processo é efetuado para todos os códigos de produto. Como cada elemento desta lista contém muitas informações (mês a mês e filial a filial) são apresentados os resultados para um produto de cada vez através de um método que permite iterar pela lista (avançar e recuar).

3 Testes de Performance

Foram realizados testes de performance para medir o tempo de leitura, *parsing* e validação das linhas do ficheiros de vendas.

Estes testes foram realizados segundo dois métodos diferentes de leitura dos ficheiros. Um com a classe *BufferedReader* e outro com a classe *File*. Os resultados apresentados são calculados através da média entre 10 medições de tempo para cada um dos casos. A tabela seguinte apresenta os resultados obtidos para o ficheiro de 1 milhão de vendas:

	BufferedReader	Files
Leitura apenas	0,128257042	0,238822126
Leitura e *parsing*	0,581102726	0,735096058
Leitura,*parsing*, validação, armazenamento	3,47857182	-

Relativamente ao resultados obtidos para apenas a leitura do ficheiro sem *parsing* foram usados os seguintes programas:

- **BufferedReader**

```
public static void main(String[] args) {
    String line = "";

    try{
```

```
Chrono.start();
        BufferedReader bf = new BufferedReader(new
                            FileReader("Vendas_1M.txt"));
        while((line = bf.readLine()) != null){

        }
        System.out.println(Chrono.getTImeString());
    }
    catch (IOException e){
        e.printStackTrace();
    }
}
```

- **Files**

```
public static void main(String[] args) {
    Path p = Paths.get("Vendas_1M.txt");
    try {
        Chrono.start();
        List<String> s = Files.readAllLines(p);
        System.out.println(Chrono.getTImeString());
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Em relação ao segundo caso, leitura e *parsing*, ou seja, separação dos campos das linhas de vendas e criação de instâncias de uma classe **Venda** contendo todas as informações de uma linha:

- **BufferedReader**

```
public static void main(String[] args) {
    String line = "";

    try{
        Chrono.start();
        BufferedReader bf = new BufferedReader(new
                            FileReader("Vendas_1M.txt"));
        while((line = bf.readLine()) != null){
            String[] fields = line.split(" ");
            Venda v = new Venda(fields[0],
            Double.parseDouble(fields[1]),
            Integer.parseInt(fields[2]),
            fields[3], fields[4],Integer.parseInt(fields[5]),
            Integer.parseInt(fields[6]));
```

```
        }
        System.out.println(Chrono.getTImeString());
    }
    catch (IOException e){
        e.printStackTrace();
    }
}
```

- **Files**

```
public static void main(String[] args) {
    Path p = Paths.get("Vendas_1M.txt");
    try {
        Chrono.start();
        List<String> s = Files.readAllLines(p);
        for(String s1 : s) {
            String[] fields = s1.split(" ");
            Venda v = new Venda(fields[0],
            Double.parseDouble(fields[1]),
            Integer.parseInt(fields[2]),
            fields[3], fields[4],
            Integer.parseInt(fields[5]),
            Integer.parseInt(fields[6]));
        }
        System.out.println(Chrono.getTImeString());
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Para o último caso, foi utilizado mesmo o projeto realizado para a medição dos tempos, ou seja, relativamente ao caso anterior, foi realizada também a validação das linhas do ficheiro de vendas (verificar se existem os códigos de produto e cliente e validação da linha) e armazenamento nas estruturas presentes (basicamente, a *query* 0).

4 Documentação

A documentação gerada em *javadoc* encontra-se no repositótio *GitHub* em *proj-Java/docs/index.html*

5 Conclusão

Em geral, acreditamos que o trabalho foi bem sucedido. Cumprimos todos os objetivos impostos, sendo que, ao contrário da primeira fase, conseguimos implementar todas as *queries* corretamente sem ter as mesmas falharem.

Em termos de trabalho futuro, poderíamos talvez melhorar as performances de algumas *queries*, cujos algoritmos de resolução não serão necessariamente os melhores, nomeadamente a 8.

Em suma, acreditamos que fizemos um bom trabalho.