Universidade do Minho

2019/2020

Laboratórios de Informática III

Mestrado Integrado em Engenharia Informática

2ª Fase

Grupo 16

José Pedro Castro Ferreira A89572

Luís Carlos Sousa Magalhães A89528

Jaime Abreu Fernandes de Oliveira A89598

A89572

A89528

A89589

Índice

Introdução

Este projeto foi apresentado no presente ano letivo no âmbito da Unidade Curricular de *Laboratórios de Informática III* do 2º Ano do curso de *Mestrado Integrado em Engenharia Informática*.

Para esta segunda fase do projeto, a linguagem de programação usada na sua execução foi a linguagem Java (na sua versão mais recente, à data da realização do projeto). Este projeto proporciona uma oportunidade para desenvolver um projeto de larga escala com o objetivo de criar uma aplicação que simule um Sistema de Gestão de Vendas.

Foi utilizado o IDE *IntelliJ* para a realização deste projeto.

Este documento visa apresentar as soluções adotadas para o desenvolvimento do projeto, bem como explicitar as estruturas implementadas.

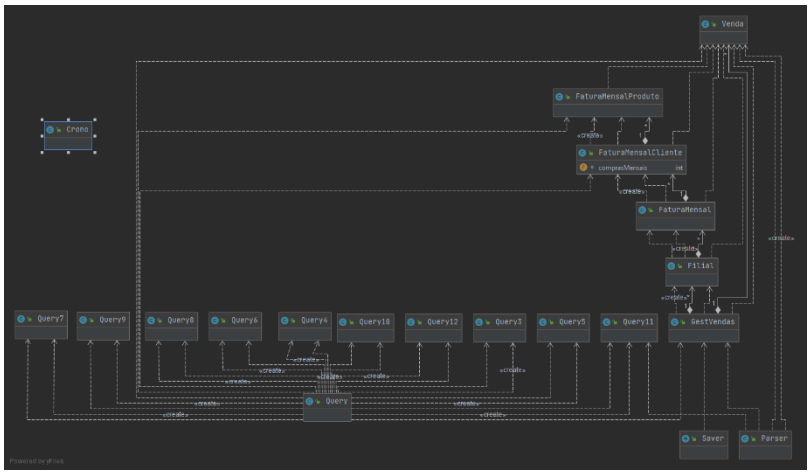
Estrutura Do Projeto

Sendo uma aplicação de um projeto de larga escala, esta aceita a entrada de três ficheiros de texto: um ficheiro com a listagem dos códigos de produtos, um ficheiro com a listagem dos códigos de clientes e um terceiro com os dados da faturação das vendas.

Para o tratamento de dados dos ficheiros dos clientes e produtos implementou-se a mesma estrutura, já o ficheiro de dados de faturação teve uma estruturação diferente.

O projeto segue o modelo MVC, garantido encapsulamento e tornando a aplicação mais genérica e flexível.

O model é composto por várias classes que se relacionam como é apresentado no diagrama abaixo, gerado no IDE utilizado.



1. Estruturas de Dados

Foi criada de início uma estrutura GestVendas com 4 componentes, explicitadas nos parágrafos seguintes.

Como já foi dito anteriormente, a estrutura de dados utilizada para armazenar os dados dos ficheiros de clientes e dos produtos: um Set<String> em que cada elemento será um código de cliente ou de produto. O objetivo desta estrutura foi o de armazenar os valores ordenados na estrutura.

Sabendo que a faturação das vendas continha informação sobre existirem diferentes Filiais associadas ao ficheiro de dados das vendas e analisando previamente as Tarefas que teríamos de desenvolver ao longo do Projeto, percebe-se facilmente que uma divisão dos dados das vendas em duas estruturas (uma global e outra que faça a divisão filial a filial) seria benéfico para a realização do trabalho, a nível de simplicidade, performance e flexibilidade.

Para a estrutura que armazena os dados da Faturação como uma Faturação Global, utilizou-se um *Map<K,V>* onde a *key* é o código do produto e o *value* uma *List<E>* em que E é do tipo *Venda*. Este tipo de dados guarda a informação lida por cada linha do ficheiro de faturação de vendas, depois de efetuado o *parsing* da *String*.

Para a estrutura que armazena os dados separados pelas respetivas Filiais utilizou-se um *Map<K,V>* onde a *key* é o *ID* da filial e o *value* é um objeto da classe *Filial* e é composto por um identificador e um *Map<K,V>* onde a *key* é o identificador de um mês e cada *value* é uma *FaturaMensal*. A FaturaMensal contém a informação sobre quantas vendas foram realizadas na respetiva filial num certo mês e um Map<K,V> onde a *key* é o código de um cliente e o *value* uma *FaturaMensalCliente*. A *FaturaMensalCliente* contém a informação de quantas compras o respetivo cliente fez e quanto gastou nesse mês na filial em análise, assim como um Map<K,V> que tem como key um código de Produto e como value uma FaturaMensalProduto que armazena a informação relativa à faturação (como quantidade vendida, faturação) de cada produto na filial e no mês em questão para um cliente que queríamos analisar.

2. Módulos

Visto ser uma aplicação em grande escala, foram utilizados diversos módulos para a execução do projeto. O resultado final de toda a estrutura do projeto baseou-se no modelo *MVC (model-view-contorller)*.

Encapsulamento

Em Java, é fácil encapsular os dados, uma vez que a própria linguagem tem mecanismos que facilitam esta prática, como a simples declaração de varáveis de instância como private.

A utilização de APIs do Java tornou o código bastante abstrato e flexível sendo que a qualquer momento podemos mudar uma estrutura e pouco ou nada teremos de alterar no código.

3. Tarefas/"Queries"

A aplicação foi desenvolvida de forma a conseguir fornecer ao utilizador um conjunto de informações sobre o *Sistema de Gestão de Vendas* de forma simples e legível.

Para isso, o projeto foi baseado nas *queries* expostas no enunciado do mesmo. Estas *queries* dividem-se em dois tipo: **Estatísticas** (2) e **Interativas** (10). As 12 *queries* foram implementadas com sucesso no projeto. Os testes e comentários das *queries* foram feitos em base do ficheiro com um milhão de vendas.

3.0 Queries Estatísticas

Aquando da inicialização do controlador, é criada uma variável de instância ao controlador da classe *Query11*. Aquando da leitura e validação de cada linha do ficheiro de vendas, essa variável de instância é atualizada com os dados sobre o ficheiro lido.

Feita a leitura, validação e registo das vendas, serve-se das estruturas criadas para guardar a informação do *SGV* para tirar os dados pedidos pelo enunciado, sendo que maior parte desses dados são variáveis de classes, facilmente acessíveis.

A informação resultante das duas *queries* estatísticas são apresentadas ao utilizador de forma eficaz.

3.1 Queries Interativas

3.1.1

Tal como nas *queries estatísticas*, a informação obtida desta *query* é devolvida de forma eficaz através um sistema de output paginado, que permite ao utilizador navegar livremente pelas páginas e consultar a informação pretendida.

3.1.2

Como a estrutura de dados das Filiais já é organizada em 1ª ordem à Filial e em 2ª ordem aos meses, quando o programa recebe um mês válido introduzido pelo utilizador, vai percorrer as *n* Filiais que estão registadas no *SGV* e somar o número de vendas mensais em cada Filial para obter o número global. De relembrar que para cada filial, em cada mês temos uma *FaturaMensal*, que tem no seu conteúdo a quantidade de vendas mensais na respetiva filial no dado mês.

3.1.3

Para apresentar quantas compras um cliente fez, quantos produtos distintos comprou e quanto gastou no total para cada mês, existe uma estrutura *Query3* que

guarda exatamente estes valores. Para obter esta informação é consultada a *FaturaMensalCliente* em cada mês para as 3 filiais sendo que esta tem em si toda a informação necessária para responder à *query 3*.

3.1.4

Percorrendo a *Faturação* do *SGV*, conseguimos ir buscar a informação pedida para um código de produto dado pelo utilizador e guardar em variáveis de instância do método que resolve a *query 4*. A informação é depois guardada num *Map* em que a *key* é o mês e o *value* uma estrutura *Query4* que vai guardar a informação pedida no enunciado.

3.1.5

Existe uma classe *Query5* responsável por guardar o código dos produtos e a quantidade de unidades desse produto comprados por um cliente dado pelo utilizador. Percorrendo as diferentes filiais para os diferentes meses, conseguimos analisar a *FaturaMensalCliente* do cliente dado, e percorrendo a informação desta classe que a cada Produto comprado pelo cliente faz corresponder uma *FaturaMensalProduto*, conseguimos facilmente receber a informação pedida no enunciado.

3.1.6

Lido um número introduzido pelo utilizador, vamos percorrer as Filiais em cada mês, para todos os Clientes ver as unidades compradas para cada produto e somar tudo para chegarmos ao número anual de unidades vendidas dos produtos. Tiramos também a informação de quantos clientes compraram cada produto. Todas estas informações são armazenadas numa *Query6*. Os x produtos mais vendidos são apresentados ao utilizador por páginas caso sejam mais de 30 ou individualmente e devidamente separados caso contrário.

3.1.7

Sendo que a *FaturaMensalCliente* guarda a informação referente ao gasto de um cliente num certo mês e numa filial, conseguimos somar estes valores para os 12 meses obtemos os valores totais de gastos para todos os clientes. Esta relação cliente/gasto é guardada na estrutura *Query7*. Daqui apenas temos de selecionar os 3 com maiores gastos para cada Filial e apresentar devidamente a informação ao utilizador.

3.1.8

Basta armazenar a quantos produtos diferentes um cliente compra numa filial num certo mês, para todas as filiais e todos os meses e depois ver nessa

informação quantos produtos distintos comprou cada cliente. Essa informação é guardada na estrutura *Query8* e basta escolher os x com maior quantidade, sendo x um valor dado pelo utilizador. A informação pode ser apresentada por páginas caso o utilizador deseje ver mais de 30 elementos ou sequencial caso contrário.

3.1.9

Acedendo à *FaturaMensal* para todos os meses e filiais do *SGV*, facilmente calculamos quantos dos clientes têm nas suas *FaturaMensalCliente* registo do produto dado pelo utilizador e quantas vezes compraram o produto. Na estrutura *Query9* armazena-se essa informação e basta encontrar os x clientes (número também dado pelo utilizador) com mais compras desse produto e imprimir de modo semelhante à *Query8*, vista anteriormente.

3.1.10

Dado um produto pelo utilizador, o programa percorre a estrutura das Filiais do *SGV* e de cada Filia, para cada mês, tira a informação referente ao lucro da *FaturaMensalProduto*. A informação é apresentada sequencialmente sendo que para cada mês, é apresentada a informação para as *n* Filiais do *SGV*.

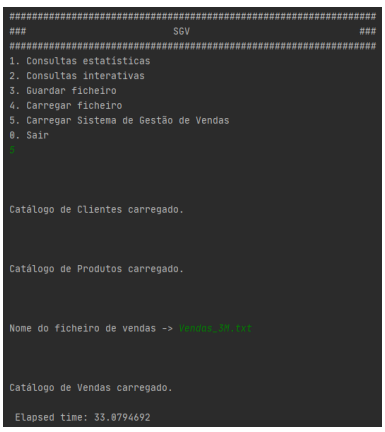
4. Testes/Desempenho

Dado que a execução das tarefas expõe para o utilizador o tempo que cada uma demora a executar, nesta parte do relatório vamos apenas apresentar os valores do tempo de leitura dos 3 ficheiros de vendas disponibilizados.

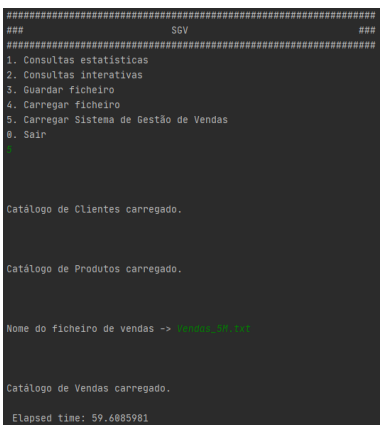
De facto, os tempos não são ótimos para a leitura dos ficheiros e poderiam ser melhorados mas devemos também ter em consideração que a *Query11* é realizada ao mesmo tempo que a leitura do Ficheiro. Isto é, quando a vamos executar, na verdade só estamos a imprimir o seu resultado porque já foi executada aquando da leitura dos ficheiros. Por este motivo, o desempenho da leitura é significativamente reduzido.

Seguem-se então os resultados da leitura.

Vendas_1M.txt



Vendas_3M.txt



Vendas_5M.txt

Conclusão

De acordo com os dados obtidos relativos à performance da aplicação é verificado que o maior problema nesta implementação é o tempo que as queries requerem uma pesquisa sobre todos os elementos.

Foram testados alguns processos sobre como resolver este problema até chegar ao concretizado, percebendo que ainda não é a implementação ideal. Acreditamos que este problema seria facilmente resolvido com um pouco mais de aprofundamento dos nossos conhecimentos da linguagem utilizada e com uma melhor planificação das estruturas (que apesar não ser ideal, foi mais bem conseguida do que na 1ª Fase do projeto).

Foi-nos também pedido que o projeto tivesse a funcionalidade de guardar e carregar ficheiros binários com a informação do programa. No entanto, essa parte do programa não ficou como seria desejável. Nenhum problema ocorre ao guardar o ficheiro, mas não conseguimos resolver o facto de não conseguir carregar o mesmo.

Acreditamos que o objetivo do trabalho, de desenvolver uma aplicação que seguisse o modelo MVC, capaz de realizar as tarefas apresentadas no enunciado com os resultados devidamente apresentados, mantendo o código simples e flexível, foi concretizado com sucesso.