Universidade do Minho
Mestrado Integrado em Engenharia Informática

Projeto Laboratórios de Informática 3

GRUPO 65
Ana Teresa Gião Gomes - A89536
Maria Quintas Barros - A89325
Maria Beatriz Araújo Lacerda - A89535

31 de Maio 2020

Figure 1: A89536

Figure 2: A89525

Figure 3: A89535

1 Introdução

No âmbito da unidade corricular de Laboratórios de Informática III, foi nos proposto, para segunda fase, o desenvolvimento do projeto em linguagem Java, que consiste num Sistema de Gestão de Vendas (SGV) de uma cadeia de distribuição. Já tinhamos anteriormente abordado este projeto na linguagem C e agora o desafio é convertê-lo em linguagem Java para que, deste modo, tenhamos oportunidade de comparar os resultados obtidos relativamente à performance, diferenças no nivel de dificuldade das linguagens e entre muitos outros aspetos.

2 Estruturação

2.1 GestVendas

```
public class Model implements Serializable{
    private String path;
    private Produtos p;
    private Clientes c;
    private VT v;
    private Faturacao f;
    private Filial[] filial;
    public static int N = 3;
```

Figure 4: Variáveis da classe GestVendas

A estrutura que implemeta o nosso Model é composta por: - Uma string com o nome do ficheiro;
- A classe Produtos;
- A classe Clientes;
- A classe VT que contem uma lista de Vendas com a informação das vendas válidas lidas no ficheiro;
- A classe Faturaçao;
- A um array de classes Filial;
- Uma variavel estática que define o número de filiais com que trabalhamos no nosso sistema;

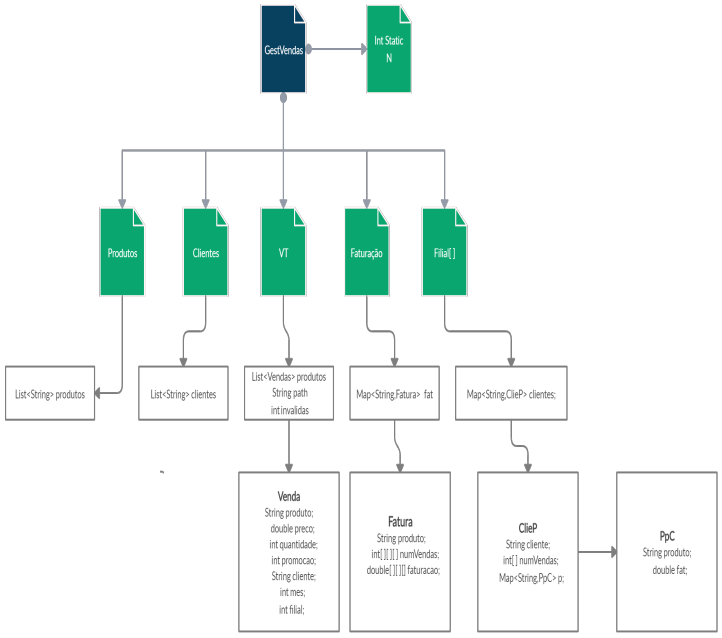

Figure 5: Estruturação do projeto

2.2 Filial

A nossa classe Filial contem um apenas um Map. Nesse Map temos como key uma String que corresponde ao código dos clientes. Como value temos uma outra classe: CliP.
Nesta classe temos uma string que indica o código do cliente, um array de inteiros que armazena o número de vendas daquele cliente por mês e ainda um outro Map. Este Map vai, por sua vez, ter como key o código de um produto e como value uma terceira classe: Ppc.
Por último, esta classe vai conter uma string com o código de um produtos e um double que contem a faturação desse mesmo produto.

2.3 Faturação

Na classe Faturação criamos um Map que tem como key um código de um produto comprado e como value uma segunda classe: Fatura. Em fatura temos uma string que contem um código de um produto, um array que contem o número de Vendas por filial, mes e promoçaõ (numVendas) e ainda um segundo array que contem a faturação, também ela por filial, mes e promoção.

2.4 Estruturas Elementares

Temos como classes base do nosso trabalho a classe:
- VT: A nossa classe VT é responsável por ler e validar as vendas do ficheiro dado. Temos, por esta razão, uma lista de Vendas nesta classe e ainda um inteiro (inavlidas). Deste modo, através do método de validação das vendas, se a venda for válida é acrescentada à nossa lista, caso contrario o inteiro é incrementado. Acrescentamos, ainda, uma String que nos indica o ficheiro que se encontra a ser lido no momento.
- Produtos: Esta classe contem uma lista de strings com os códigos dos produtos válidos lidos no ficheiro criando assim um catálogo de produtos.
- Clientes: Por último criamos a classe Clientes que contem uma lista de strings com os códigos dos clientes válidos lidos no ficheiro formando o catálogo de clientes.

3 Modularização e Abstração de Dados

Para que o nosso código seja, para além de eficiente, também seguro, tivemos o cuidado de declarar todas as nossas variaveis de instancia como private . Deste modo, para o acesso de variaveis entre classes é necessário criar getters e setters para estas, o que faz com que as haja comunicação limitad

4 Queries Iterativas

- **Querie 1:**
  Com esta querie pretendemos ordenar alfabeticamente uma lista com os códigos dos produtos nunca comprados e o seu respetivo

total.
Para isto criamos uma lista de strings, q1, que irá armazenar os códigos dos produtos. Em seguida percorremos a lista p, que contem os códigos de todos os produtos válidos. Se o código do produto consta na lista dos produtos válidos, mas não está no map da faturação (o que é verificado através do método contemProduto) então adicionamos esse codigo a q1.
Por último ordenamos a nossa lista através do sorted alfabeticamente(ordem natural).

- **Querie 2:**
  Criamos esta querie com o objetvo de, dado um mês e a opcao de escolher entre resultados globais ou para cada filial, obtermos o número de vendas e o número de clientes distintos que participaram nessas vendas.

  Para isto criamos um Entry.Map, cujo quer a key, quer o value representam um inteiro. No primeiro inteiro armazenamos o número de clientes e no segundo o número de vendas.

  Recorremos ao uso de uma flag para distinguir se queremos resultados globais ou filial a filial.
  Depois, para cada caso, determinamos o numero de clientes (numClientes) e o numero de vendas (numVendas) para o mês inserido pelo utilizador.

- **Querie 3:**
  Esta querie tem o objetivo de, dado um cliente introduzido pelo utilizador, retornar as suas compras, os produtos distintos que este comprou e quanto gastou, para cada mês.

  Assim, desde já, definimos que o cliente introduzido teria de ser válido para devolver os dados que pretendiamos.
  Depois, criamos um ciclo que percorria todos os meses do ano e, deste modo, obtemos para cada um dos meses uma lista com as suas compras (List¡Venda¿ cli), o numero de produtos distintos (numP) e faturação correspondente (fat).

- **Querie 4:**
  Com esta querie, pretendemos determinar, dado um produto introduzido pelo utilizador, o numero de vezes que este foi comprado, por

quantos clientes diferentes e quanto gastou nessas compras, para todos os meses, Deste modo, primeiramente, definimos que o produto introduzido teria de ser válido para devolver os dados que pretendiamos. Posteriormente, criamos um ciclo que percorria todos os meses do ano, de forma a dar-nos para cada um, uma lista com as compras desse produto (List¡Venda¿ prod), o numero de clientes que o compraram (numC) e a faturação correspondente (fat).

- **Querie 5:**
  Para implementarmos esta querie começamos por definir que o cliente introduzido teria de ser válido para devolver os dados que pretendiamos. Depois, criamos um comparador que ordenasse por ordem decrescente de quantidade a lista de códigos de produtos que o cliente mais comprou.

- **Querie 6:**
  Para começar, determinamos o conjunto dos produtos mais vendidos no ano, limitado por um numero introduzido pelo utilizador. Depois, com a ajuda de um ciclo que percorre esse conjunto de produtos, determinamos o numero total de clientes distintos que os compraram, e por fim definimos o numero de clientes para cada um deles.

- **Querie 7:**
  Nesta querie, começamos por criar um ciclo que percorre todas as filiais e um comparador que ordena os clientes tendo em conta o dinheiro faturado. Depois definimos o limite a 3 e, assim, para cada filial, determinamos os 3 maiores compradoes.

- **Querie 8:**
  Com esta querie, determinamos o conjunto dos clientes que compraram mais produtos diferentes, limitado por um numero introduzido pelo utilizador. Depois, com a ajuda de um ciclo que percorre esse conjunto de clientes, determinamos o numero total de produtos distintos comprados, e por fim, definimos o numero de produtos para cada um deles, ordenados por ordem decrescente

- **Querie 9:**
  Nesta querie, começamos por utilizar um comparador por quantidade definido na classe Venda e um limite introduzido pelo utilizador para chegarmos a uma lista de vendas. Posteriormente, determinamos uma lista de clientes e uma lista de faturações, ordenadas segundo esse mesmo critério anterior. Assim, ate atingirmos

o limite introduzido ou o fim da lista de clientes, iremos retornar cada elemento dessa lista associado à faturação correspondente.

- **Querie 10:**
  Para dar resposta a esta querie criamos um Map onde armazenamos os dados da nossa classe Fatura. Em seguida, percorremos a nossa lista de produtos válidos e se o produto não estiver contido na fatura (ou seja não for comprado) criamos uma nova Fatura e acrescentamos ao Map garantindo, assim, que obtemos a faturação de todos os produtos do ficheiro.

5 Queries Estatísticas

- **Consulta 1:** Esta consulta tem como objetivos apresentar ao utilizador os dados referentes ao último ficheiro de vendas lido tais como:

  - nome do ficheiro
  Obtido a partir da string "path" dentro do nosso Model.

  - número total de registos de venda errados
  Tal como anteriormente, criamos uma variável private na classe VT, desta vez um inteiro. De seguida, na nossa função que valida as vendas, incrementamos esta variável conforme o aparecimento de vendas invalidas.

  - número total de produtos
  Para apresentar o número total de produtos simplesmente imprimimos o tamanho da lista de produtos validados nesse ficheiro.

  - total de diferentes produtos comprados
  Esta variável esta armazenada no primeiro inteiro do nosso Map.Entry no método estatistica1. Para isto, através da classe VT e do método????? distinct() aplicado na stream, conseguimos criar uma lista com o número de produtos comprados e distintos. Após isso aplenas temos de ver o tamanho desta lista através do size().

  - total de não comprados
  Para obter este número subtraímos ao numero de produtos válidos o número que obtemos anteriormente.

- número total de clientes
Para apresentar o número total de produtos simplesmente imprimimos o tamanho da lista de clientes validados nesse ficheiro.

- Número de clientes que compraram
Armazenanos este valor na key do Entry.Map interior no método estatistica1. Para o obter fizemos um processo semelhante ao que utilizamos para descobrir o número total de produtos distintos, mas desta vez fizemos o get dos clientes.

- total de clientes que nada compraram
Para obter este valor subtraimos o numero do clientes que compraram ao número de clientes total.

- total de compras de valor total igual a 0.0
Para obter esta variavel percorremos a lista de vendas da classe VT e filtramos as vendas que tinham um preço de 0, e obtemos o código do cliente. Após isto passamos para dentro do value do nosso Map.Entry interior o tamanho da lista criada.

- faturação total
Com dois ciclos, um para percorrer as filiais e outro para percorrer os meses calculamos a faturaçao através do método faturacaoTotal da classe Faturaçao e somamos os resultados numa variavel.

- **Consulta 2:**
  Para criar esta consulta, armazenamos num map os seguintes números gerais respeitantes aos dados atuais:

  - Número total de compras por mês
  Número obtido através da classe VT e do método numVendas inserido nessa mesma classe. Este método filtra das vendas existentes, aquelas que nos é de interesse (de cada mês) e conta o número de produtos. Esta informação é armazenada na variável "numVendas"

  - Faturação total por mês para cada filial e o valor total global
  Obtemos esta variável através do método faturacaoVendas, também ele na classe VT. Neste método o preço dos produtos que nos é de interesse (de um mês e de uma filial específica) é multiplicado pela

quantidade dos mesmos produtos. Em seguida, esta informação é armazenada na variavel "FatT"

- Número de distintos clientes que compraram em cada mês filial a filial
Por último utilizamos o método clientesDistintos, também na classe VT. Este método filtra a partir todas as vendas aquelas com a filial e o mês pretendido, obtem os códigos de clientes das vendas filtradas e coleta para uma lista os códigos distintos. Esta informação é depois armazenada na variável "cli".

Deste modo, obtemos os resultados por mês e filial das informações pedidas. Os resultados globais são obtidos através de ciclos na função printconsulta2 na classe View.

6 Testes de Tempo

Ficheiro	**Tempo Global** *(s)*
1 Milhão	≈ 4.45
3 Milhões	≈ 11.6
5 Milhões	≈ 17.26

Querie	**1 Milhão** *(s)*	**3 Milhões** *(s)*	**5 Milhões** *(s)*
Querie 1	0.12	0.20	0.26
Querie 2	0.25	1.03	2.90
Querie 3	0.93	4.14	12.63
Querie 4	1.51	4.55	12.39
Querie 5	0.04	0.21	1.49
Querie 6	0.91	3.10	7.95
Querie 7	0.79	3.73	6.67
Querie 8	0.97	4.14	29.21
Querie 9	0.05	0.33	1.27
Querie 10	0.23	1.51	0.36
Consulta 1	0.38	18.67	3.56
Consulta 2	1.81	6.46	10.82

Os restante resultados dos nossos testes encontram-se em anexo.

7 Projeto C vs Projeto Java

Analisando o nosso trabalho feito em C alguns aspetos que tentamos melhorar foram:
- A definição de uma variável estática N no GestVendas, tornando, assim, mais simples o acrescentar/remover filiais das nossas estruturas e manter o sistema em funcionamento.
- Utilização de mais HashMaps ao invés de, por exemplo, TreeSets, para melhorar a eficiencia do sistema, visto que o uso de árvores binárias de procura foi um fator de peso na ineficiência do nosso projeto em C. Mais tarde, a partir dos testes efetuados, verificamos isto novamento nos gráficos criados.
- Melhoramento do nosso MVC, definindo e separando mais claramente o nosso view e o nosso controller.

8 Grafo de Dependências

O grafo de dependências encontra-se em anexo.

9 Conclusão

Tal como na primeira parte do trabalho, notamos que uma boa organização das nossas estruturas é fundamental para a eficácia das queries. Posto isto, devido ao facto das nossas estruturas para a querie 8 serem muito pesadas e complexas, foi a que nos causou mais problemas e dificuldades, resultando também em tempos de execução maiores. Nesta segunda fase, sentimos uma maior à vontade à linguagem Java comparativamente a C pois a primeira correr numa máquina virtual, discartando assim a preocupação com leaks de memória. Por outro lado, também concluimos que o vasto leque de ferramentas nativas da linguagem Java, foi fundamental para nos ajudar na resolução dos problemas deste projeto. Deste modo, se nos propusessem novamente este projeto dificilmente escolheriamos a linguagem C para implementarmos, principalmente porque, para respondermos a tudo o que este projeto requer, não necessitamos necessariamente das vantagens que apenas o C pode oferecer, dando assim prioridade a linguagens com um nível de abstração superior.