Universidade do Minho

Mestrado Integrado em Engenharia Informática

Sistema de Gestão de Vendas

Laboratórios de Informática 3

2º Ano, 2º Semestre, 2019/2020

Grupo 74

Francisco Peixoto A84668

Renato Gomes A84696

Conteúdo

Capítulo 1

Introdução

Este relatório aborda a resolução do projeto prático implementado na primeira parte desta disciplina, sendo agora desenvolvido em Java. A organização do nosso projeto baseia-se no modelo *MVC (model, view, controller)*.

O projeto consistia na implementação de um sistema de gestão de vendas, tendo sido fornecido ficheiros cujos dados eram necessários para responder a uma lista de queries dada pelos professores.

Para isto, tínhamos de criar estruturas de dados para armazenar toda a informação relevante, e depois de carregada, estaríamos então em condições para executar as queries.

Dito isto, neste relatório são explicadas as nossas decisões tomadas para a elaboração deste projeto.

Capítulo 2

Resolução do Problema

Para esta segunda parte do trabalho, e para responder corretamente às queries pretendidas, optámos por uma implementação que apesar de ter tempos de load iniciais relativamente mais lentos face ao trabalho em C, todas as queries apesar de muito mais complexas são significativamente mais rápidas, chegando mesmo a ser em caso geral instantâneas.

Dividimos o trabalho por módulos, ou seja, separamos o código por pequenos pedaços reutilizáveis de modo a serem reaproveitados, e garantir que a sua manutenção não tenha impacto direto nos outros. Assim, a programação torna-se mais inteligente e menos suscetível a erros, isto também permitiu que ao programar o código nos conseguíssemos focar apenas na implementação de pequenos aspetos da interface de cada modulo.

O projeto foi inicializado com a validação dos ficheiros de texto fornecidos de modo a garantir que não existia informação errada. A validação dos produtos e dos clientes foi feita de modo semelhantes uma vez que só diferiam de um carácter.

O programa preenche também diversas estruturas como é o caso da faturação e da filial, que o grupo considera que são sucintas e compactas.

Capítulo 3

Modelo MVC e Classes

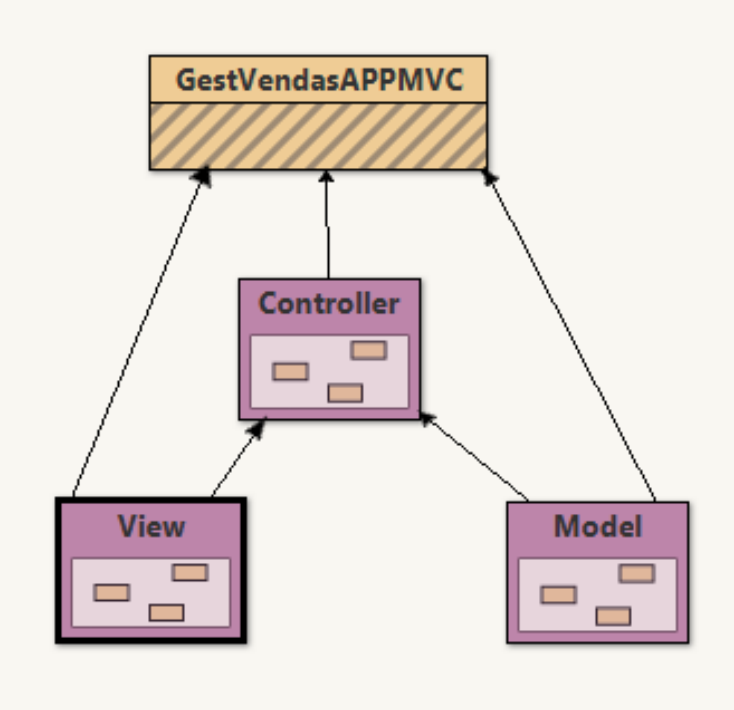

Diagrama da estrutura dos packages

3.1 Model

O model é a classe agregadora de tudo o que representa leitura e tratamento de dados, nesta classe estão presentes funcionalidades como leitura de ficheiros, procuras que envolvam percorrer mapas e manipulação de dados.

As interfaces principais presentes no package Model são as seguintes:

- **GestVendasModel**: Inicia estruturas
- **ICatalogo_Clientes e ICatalogo_Produtos**: Trabalham as estruturas catálogos
- **IFaturacao**: Estrutura principal referente à faturação
- **IFilial**: Opera sobre registos das interfaces IProduto/ICliente

3.1.1 Catálogos

A classe *Catalogo_Clientes* representa o catálogo que contém todos os clientes, lidos e validados do ficheiro clientes.txt, só que desta vez foi instanciado num *Set*.

A classe *Catalogo_Produtos* representa o catálogo que contém todos os produtos, lidos e validados do ficheiro produtos.txt, só que desta vez foi instanciado num *Set*.

3.1.2 Faturação

A classe *Faturacao* contém as estruturas de dados responsáveis pela resposta eficiente a questões quantitativas que relacionam as filiais com a sua faturação mensal.

```
1 public class Faturacao implements Serializable, IFaturacao {
2     private double[][] tabela_faturacao;
3     private int[][] tabela_vendas;
4     private double total_faturado;
5 }
```

3.1.3 Filial

A classe *Filial* contém as estruturas de dados responsáveis pela resposta eficiente a questões quantitativas que relacionam os registos de produtos com os registos de clientes.

```
public class Produto implements IProduto{
    private final String id;
    private boolean vendido;

    private Map[] cliente_unidades;
    private TreeMap<String, Double> sorted_cliente_faturacao;
}
```

No exemplo apresentado destacamos apenas alguns pontos da estrutura Produto que nos ajudam a resolver da forma que o grupo achou mais correta as queries.

Como por exemplo o mapa sorted_cliente_faturacao é usado na query 9, que apesar de acrescentar poucos segundos na leitura do ficheiro faz com que correr a query com 1 ou 5 milhões de linhas seja praticamente a mesma coisa.

```
public class Cliente implements ICliente {
    private String id;
    private boolean comprou;

    private Map[] produto_unidades;
    private Map[] produto_faturacao;
}
```

No caso do tratamento de dados referente aos registos de cliente são adotadas soluções análogas às referidas anteriormente, neste caso realçamos o uso de arrays de mapas que nos permitem guardar produtos comprados por cliente em unidades e faturação.

3.1.4 Estatística

Na classe *Model* foi criado um *toString()* para responder de uma forma rápida e eficaz aos pedidos de consultas estatísticas, contendo vários dados, tais como, o nome do último ficheiro de vendas lido, o número de vendas válidas, entre outros.

```
(...)
sb.append("Estatisticas do ficheiro: ").append("Vendas").append("\n
    ")
    .append("N mero de clientes: ").append(total_clientes).append(
    "\n")
    .append("N mero de clientes que compraram: ").append(
    clientes_compraram).append("\n")
    .append("N mero de clientes que n o compraram:")
    .append(total_clientes-clientes_compraram).append("\n")
(...)
    .append("N mero de registos de vendas errados: ").append(
    registos_errados).append("\n")
    .append("N mero de compras de pre o 0: ").append(compras_0).
    append("\n")
    .append("Fatura  o total: ").append(faturacao_total).append("
    \n");
(...)
```

O diagrama seguinte explica sucintamente as relações entre cada uma das classes.

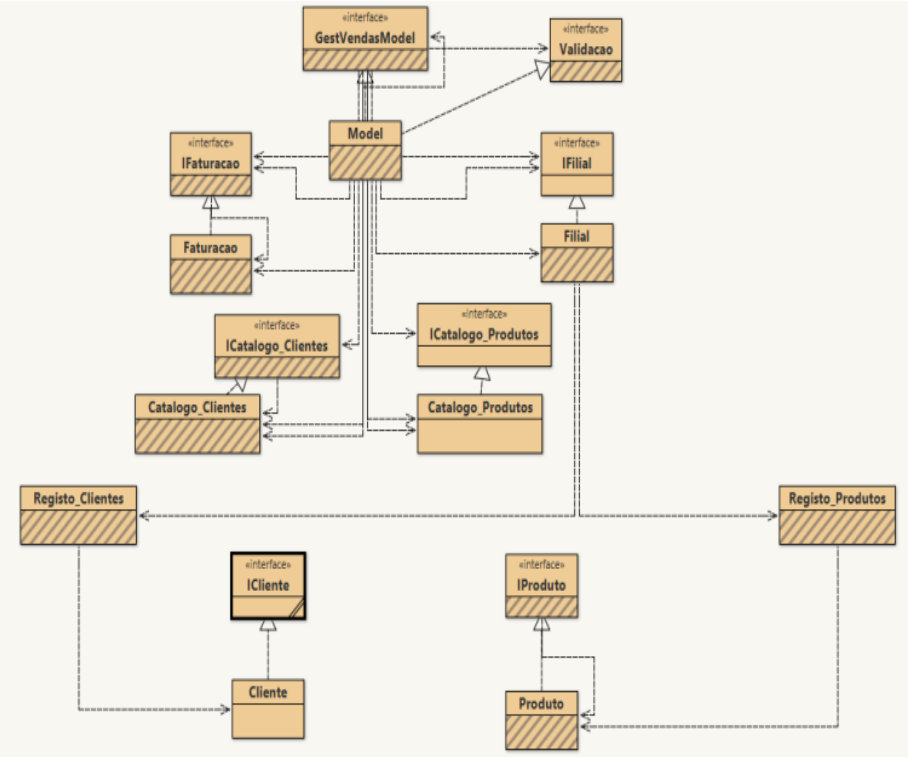

Diagrama da estrutura do package Model

3.2 Controler

A classe *GestVendasControler*, de acordo com o modelo MVC, é a classe que faz a conexão entre o *utilizador* a *view* e o *model*, isto é, recebe input do utilizador, através da classe *Input*, respostas do *model*, e efetua pedidos à *view*, para as apresentar. Tem as seguintes variáveis de instância:

```
1 public class Controler implements GestVendasControler {
2     private boolean estado;
3     private GestVendasModel model;
4     private GestVendasView view;
5     private Input input;
6 }
```

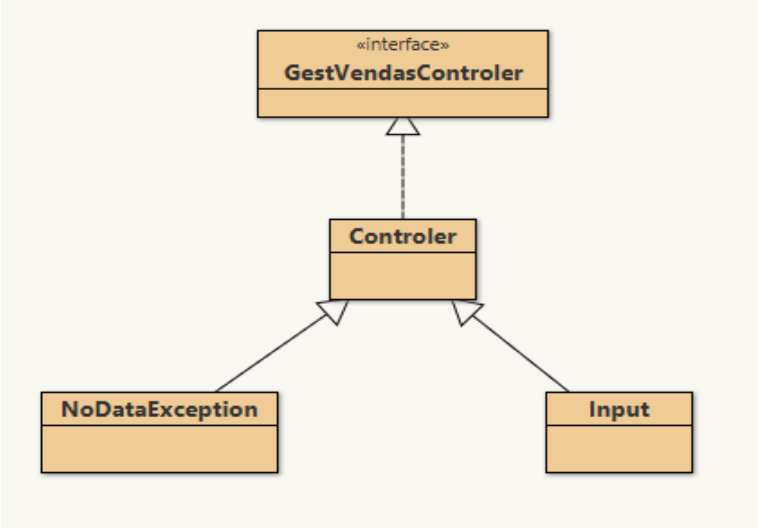

Diagrama da estrutura do package Controler

3.3 View

A classe *GestVendasView*, de acordo com o modelo MVC, é a classe responsável por apresentar output ao utilizador.

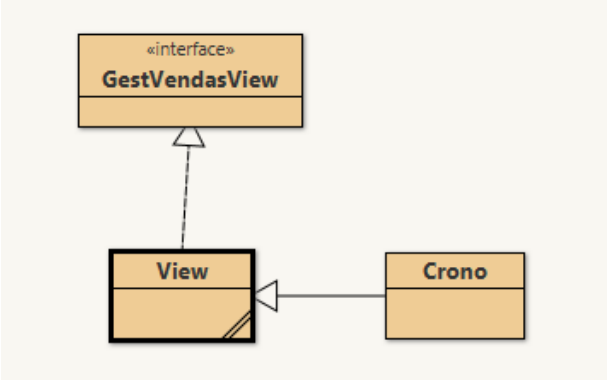

Diagrama da estrutura do package View

3.4 Interfaces

As interfaces, como abordado nas aulas, possuem bastantes vantagens, tais como permitir que várias pessoas consigam trabalhar em simultâneo no projeto abstraindo-se da implementação interna dos métodos e permite posteriormente que seja mais simples escalar o programa para dimensões superiores. Desta forma, foi-nos sugerido e até incentivado que as integrássemos no projeto. Assim, em baixo estão mencionadas as interfaces principais de cada package, é de notar que as outras interfaces criadas já foram referidas anteriormente.

- GestVendasModel
- GestVendasView
- GestVendasControler

3.5 Comparadores

Uma vez que a solução de muitas das queries requer ordenações foram implementados os seguintes comparadores, de modo a obter eficientemente de modo a obter eficientemente os dados pedidos.

```
1     IProduto extends Comparable<Produto>
2
3     public int compareTo(Produto p) {
4         return ( p.quantas_vendeu()-this.unidades_compradas);
5     }
6
7     ICliente extends Comparable<Cliente>
8
9     public int compareTo(Cliente c) {
10        return (int)(c.produtos_distintos() - this.
      produtos_distintos());
11    }
```

3.6 Exceptions

As *exceptions* são fundamentais na execução de um programa para o seu bom funcionamento quando acontece algo errado, tal como input invalido do utilizador ou dados não iniciados. Foi implementada a seguinte *exception*:

```
1     public class NoDataException extends Exception {
2
3     private static final String data_warning = "O estado n o foi
      iniciado.";
4
5     public NoDataException(){}
6
7     public String toString() {
8         return data_warning;
9     }
10}
```

Capítulo 4

Testes de Performance

4.1 Tempos de Leitura

Buffered Reader	Simples	Com Parsing	Com Parsing e Validação
Vendas 1M	0.1592 segundos	0.5467 segundos	1.3324 segundos
Vendas 3M	0.2369 segundos	0.9942 segundos	3.2344 segundos
Vendas 5M	0.4171 segundos	1.6230 segundos	5.4241 segundos

Tabela 4.1: Tabela correspondente à leitura de ficheiro segundo a Classe Buffered Reader.

Files	Simples	Com Parsing	Com Parsing e Validação
Vendas 1M	0.6894 segundos	0.8170 segundos	1.7442 segundos
Vendas 3M	1.7454 segundos	2.4276 segundos	4.9914 segundos
Vendas 5M	2.9005 segundos	40631 segundos	8.3791 segundos

Tabela 4.2: Tabela correspondente à leitura de ficheiro segundo a Classe Files.

4.2 Troca de Collections

Nestes testes o grupo percebeu que certas estruturas não podiam ser trocadas, isto é, a sua troca implicaria alterações no algoritmo. Por exemplo, estruturas de *TreeSet/TreeMap* que eram criadas com uma ordem específica, ao trocar para *HashSet/HashMap* perdiam essa ordem e o algoritmo tornava-se inválido. Assim, mudaram-se todas as outras *Collections*, chegando a conclusão que as estruturas utilizadas no trabalho são as mais adequadas, apesar da diferença quase imperceptível resultante das mudanças pedidas pelos professores.

4.3 Queries

4.3.1 Load Ficheiros

Tal como no projeto em C, o grupo considerou que sacrificar um pouco os tempos de inicialização do programa em prole de melhores tempos de execução das queries seria a melhor opção, com isto e dando o caso especifico do ficheiro Vendas_5M apesar do seu tempo de leitura ser de (em media) 51.21 segundos a query mais lenta no nosso caso é a Query 10 com uma media de 0.52 segundos.

O grupo considerou todos os resultados obtidos como sendo uma demonstração da eficiência do programa e das estrutura usadas, estando portanto satisfeito com os resultados obtidos nos testes de performance.

Tempo de Load

Ficheiro	Teste1	Teste2	Teste3	Media
Vendas_1M	14.24s	13.45s	13.25s	13.65s
Vendas_3M	32.53s	32.07s	29.54s	31.38s
Vendas_5M	54.51s	51.66s	47.47s	51.21s

4.3.2 Query 1

Os tempos de execução para a Query1 são sempre muito semelhantes, já que esta query, na nossa estrutura, depende do número de produtos e não do numero de linhas do ficheiro de vendas.

Conclusão a query é escalável no que toca ao numero de vendas e depende do numero de produtos existentes.

A estratégia implementada foi a seguinte: cada produto têm um campo boleano, que refere se este foi ou não vendido, a query percorre os produtos e agrega os não comprados.

Tempos da Query1

Ficheiro	Teste1	Teste2	Teste3	Media
Vendas_1M	0.05s	0.03s	0.03s	0.037s
Vendas_3M	0.05s	0.03s	0.03s	0.037s
Vendas_5M	0.05s	0.04s	0.03s	0.04s

Tempos da Query1 por Letra

Ficheiro\Letra	A	B	R	Z
Vendas_1M	0.036s	0.035s	0.035s	0.035s
Vendas_3M	0.036s	0.035s	0.035s	0.036s
Vendas_5M	0.035s	0.037s	0.039s	0.036s

4.3.3 Query 2

Os tempos de execução para a Query 2 são sempre muito semelhantes, já que esta query, na nossa estrutura,apenas vai realizar uma leitura de dados totalmente carregados na altura de inicialização.

Conclusão a query é escalável no que toca ao numero de vendas já que depende unicamente de uma leitura de dados.

A estratégia implementada foi a seguinte: foram criados 2 arrays, um presente na faturação responsável pela contagem de vendas realizadas e outro array presente na estrutura filial.

Tempos da Query2

Ficheiro	Teste1	Teste2	Teste3
Vendas_1M	0.0078s	0.0023s	0.0016s
Vendas_3M	0.0075s	0.0014s	0.0014s
Vendas_5M	0.0079s	0.0026s	0.0015s

Tempos da Query2 por Filial

Ficheiro	Teste1	Teste2	Teste3
Vendas_1M	0.0053s	0.0017s	0.0016s
Vendas_3M	0.0054s	0.0020s	0.0014s
Vendas_5M	0.0049s	0.0017s	0.0014s

4.3.4 Query 3

O resultado dos testes de performance da query 3 vieram comprovar ao grupo que mais uma vez não há muita diferença entre executar as diferente queries nos diferentes ficheiros de vendas, independentemente do ficheiro escolhido, a media de tempo da query 3 mantinha-se à volta de 0.00004 segundos.

4.3.5 Query 4

Como se verifica pela tabela abaixo a Query 4 encontra-se mais uma vez dentro do espectável, os tempos de execução são todos muito próximos. Isto é porque a Query 4 utiliza recursos parecidos aos usados na Query 3.

Tempos da Query4 total e por Filial

Ficheiro	Filial	Total
Vendas_1M	0.066s	0.083s
Vendas_3M	0.07s	0.09s
Vendas_5M	0.06s	0.07s

4.3.6 Query 5

Na Query 5 o programa realiza uma leitura a um dos mapas presentes na estrutura e procede a colocar os seus elementos relevantes numa lista, que tem uma certa formatação, por esta razão os tempos de execução da Query 5 são bastante baixos: rondam os 0.002 0.003 segundos independentemente do ficheiro de vendas.

4.3.7 Query 6

Como conseguimos verificar na tabela abaixo, os tempos de execução da Query 6 não depende do N numero pedido na tarefa, de facto o tempo até diminui, isto pode ser explicado pela gestão de memoria da maquina virtual em Java.

Tempos da Query6

Ficheiro	10	100	1000
Vendas_1M	0.066s	0.068s	0.071s
Vendas_3M	0.13s	0.10s	0.12s
Vendas_5M	0.34s	0.16s	0.15s

4.3.8 Query 7

Na Query 7 o programa realiza uma leitura a um dos mapas presentes na estrutura e procede a colocar os seus elementos relevantes numa lista, mais uma vez, que tem uma certa formatação, por esta razão os tempos de execução da Query 7 são bastante baixos: rondam os 0.0002 segundos independentemente do ficheiro de vendas.

4.3.9 Query 8

O comportamento da Query 8 é análogo ao comportamento da Query 7, sendo a diferença principal que a 7 tem um numero fixo de elementos a procurar e a 8 não, ou seja a Query 8 demonstra ser mais lenta dependendo do numero pedido e do ficheiro de vendas, mas sempre com resultados satisfatórios como se pode verificar na tabela abaixo.

Tempos da Query8

Ficheiro	10	100	1000
Vendas_1M	0.0004s	0.0018s	0.0144s
Vendas_3M	0.0009s	0.0052s	0.0474s
Vendas_5M	0.0012s	0.0097s	0.1028s

4.3.10 Query 9

Na Query 9 os tempos não variam muito com o N (numero pedido na tarefa), já que muitas vezes não há muitos clientes a comprara um certo produto, por exemplo pedir a Query 9 com o produto NR1093 com N=10 ou N=1000 no ficheiro de 3M resulta numa diferença de tempo de 0.01 segundos.

As diferenças entre ficheiros de vendas são umas das mais acentuadas em todo o projeto, mas mais uma vez como se pode verificar abaixo os resultados são bastante satisfatórios, na opinião do grupo.

Tempos da Query9

Ficheiro	Tempo
Vendas_1M	0.049s
Vendas_3M	0.075s
Vendas_5M	0.081s

Capítulo 5

Conclusão

Em suma, o trabalho foi realizado com sucesso. Tivemos de nos adaptar a um novo estilo de paradigma, ser eficazes e implementar soluções otimizadas para conseguir uma performance que satisfizesse os objetivos do grupo.

Achamos que este trabalho nos ajudou bastante a entender como a otimização e o processamento de grandes volumes de dados não é igual em todos os paradigmas da programação, tal como os seus cuidados.

Após a conclusão deste trabalho, pensamos que os elementos deste grupo saem com uma perceção mais clara do que é ser eficiente na hora de aceder a estruturas de dados e de processar grandes quantidades de informação.

Ficamos, então, contentes com o nosso desempenho e com o resultado obtido, acreditando que todos os objetivos foram cumpridos.