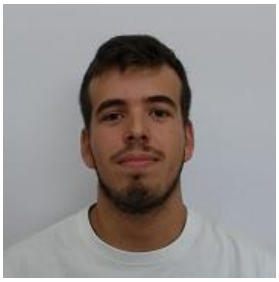
PROJETO LI3
2019

Universidade do Minho

Mestrado Integrado em Engenharia Informática

Filipa Santos, Hugo Cardoso, João Cunha





ÍNDICE

1. INTRODUÇÃO

O projeto de Laboratórios de Informática III encontra-se dividido em duas etapas, uma em C e outra em Java. Neste relatório, vamos abordar a segunda. O tema deste ano consiste em fazer uma aplicação que simula a gestão de vendas de uma empresa, isto é, determinar numa certa venda o código do cliente que efetuou essa tal venda, o código do produto que comprou, a quantidade de unidades, o preço total, etc.

A aplicação que criamos começa por ler os ficheiros de produtos, clientes e vendas de uma certa empresa. Esses dados são guardados em classe escolhidas por nós, de modo a tornar esta leitura mais rápida. De seguida, é possível consultar estes dados e, através das queries, adquirir resultados específicos, como, por exemplo, os códigos de produtos que nunca foram comprados.

Neste relatório será abordado o motivo de optarmos por certas estruturas para as classes criadas e alterações feitas para que a consulta das queries seja mais eficiente, bem como a perfomance do nosso programa.

2. CLASSES

2.1 CATÁLOGO DE PRODUTOS (CATPRODS)

Inicialmente, a estrutura optada para guardar os produtos era um Set de Produtos. Mas, para evitar ter uma classe "Produto" e, principalmente, para facilitar o acesso a cada cliente e mudar a flag, optamos por um Map<String,Integer>.

A string corresponde ao código do produto e a flag serve para verificar se o produto foi comprado (flag = 1) ou não (flag = 0). Para além disso, também temos uma variável que vai contando o número de produtos válidos lidos. Todas as variáveis são obviamente privadas (private) sendo que só devem ser acedidas nesta classe.

```
private int validos;
private Map<String,Integer> prods;
```

	Flag
AA1111	0
BB2222	0
CC3333	1
…	…

2.2 CATÁLOGO DE CLIENTES (CATCLIS)

Para os clientes, tivemos uma abordagem igual. Optamos por um Map<String,Integer> em que a string corresponde ao código do cliente e a flag serve para verificar se o cliente efetuou compras (flag = 1) ou não (flag = 0).

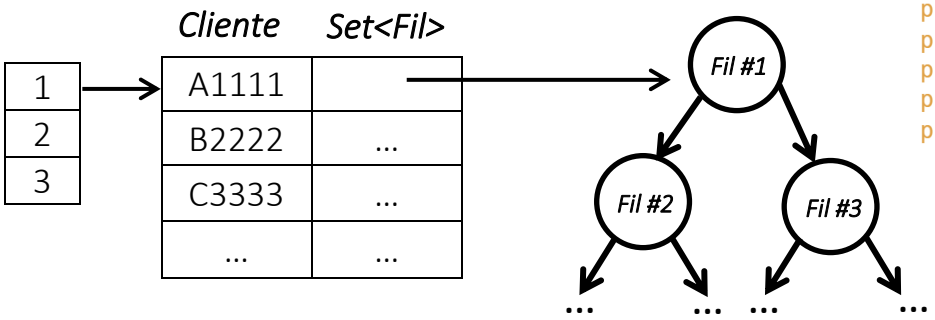
```
private int validos;
private Map<String,Integer> clis;
```

Cliente	*Flag*
A1111	1
B2222	0
C3333	1
…	…

2.3 FILIAL e FIL

Relativamente às vendas, precisavamos de dividir a sua informação por dois módulos distintos, um relativamente à facturação e outro às filiais. Para a classe filial, decidimos guardar as nossas vendas na seguinte variável do tipo List<Map<String,Set<Fil>>>. Cada Map da List vai corresponder a cada uma das filiais. Em cada Map encontram-se todas as vendas dessa filial, organizadas da seguinte maneira: a String vai ser cada o cliente da venda e todas as suas vendas (dessa filial) são adicionadas a um Set<Fil> (organizada pelo código do cliente), em que cada Fil contém os vários dados da venda que consideramos necessários.

```
private List<Map<String,Set<Fil>>> fils;
```

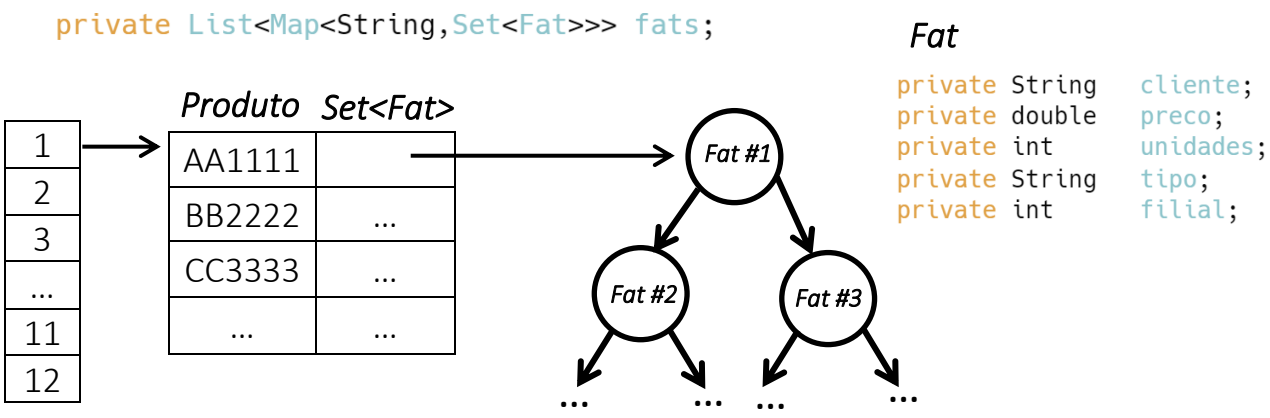


Fil

```
private String   produto;
private double   preco;
private int      unidades;
private String   tipo;
private int      mes;
```

2.4 FACTURAÇÃO e FAT

Com a classe Faturação, implementamos um raciocínio semelhante. Guardamos as nossas vendas na seguinte variável do tipo List<Map<String,Set<Fil>>>. Cada Map da List vai corresponder a cada um dos meses. Em cada Map encontram-se todas as vendas desse mês, organizadas da seguinte maneira: a String vai ser cada o produto da venda e todas as suas vendas (desse mês) são adicionadas a um Set<Fat> (organizada pelo código do produto), em que cada Fat contém os vários dados da venda que consideramos necessários.



2.5 CLASSES PRINCIPAIS

Para executar todo este programa, decidiu-se criar diversas classes, cada uma com uma função específica: ler ficheiros, criar estruturas, gerir as funcionalidades do programa dependendo do input, apresentar menus e resultados ao utilizador, etc.

2.5.1 APP (GereVendasAppMVC)

Nesta classe é criado o método main, em que são chamados todas as outras classes, de uma maneira sintetizada. Todo o verdadeiro raciocínio vai ser executado no Model, View e Controller.

2.5.2 MODEL (GereVendasModel)

A classe Model vai ser responsável por gerar toda a informação desde ler os ficheiros, inserir os dados nas respetivas estruturas até gerar os resultados das queries / consultas. Para tal, vamos ter uma variável para cada estrutura de dados diferente, que vão ser atualizadas nas leituras dos ficheiros.

Para além disso, vai ter outras variáveis que são apenas auxiliares na realização das queries / consultas, de modo a melhorar o tempo destas. A variável "nrFil" permite mudar o número de filiais desejadas facilmente.

```
private ICatProds catProds;
private ICatClis catClis;
private IFaturacao faturacao;
private IFilial filial;
```

```
private double tempo;
private String fich;
private int nrVI;
private double fat;
private int vZero;
private final int nrFil = 3;
```

2.5.3 V I E W (GereVendasView)

A classe View vai tratar de tudo que seja sobre o output do programa. Tudo o que é apresentado ao utilizador é printado nesta classe. Existem métodos para imprimir erros, menus, resultados de cada query, etc. Como variáveis de instância, definimos três menus possíveis que podem ser apresentados ao utilizador dependendo da sua escolha (menu principal, consultas estastísticas e queries).

```
private List<String> menuQ;
private List<String> menuP;
private List<String> menuC;
```

2.5.4 C O N T R O L L E R (GereVendasController)

Como não é suposto haver contacto direto entre a classe View e Model, esta classe é que vai gerir que ações tomar dependendo das escolhas do utilizador. Ou seja, se o utilizador decidir realizar uma certa query, chama-se o Model para determinar o resultado e o View para apresenta-lo ao utilizador. Para tal, esta classe têm uma instância de cada uma destas classes.

```
private IGereVendasModel model;
private IGereVendasView view;
```

2.6 O U T R A S

Para auxiliar o funcionamento da aplicação, houve a necessidade de criar outras classes meramente auxiliares. A Venda tem o propósito de guardar momentaneamente todos os parâmetros da venda e aferir a sua validade, antes de serem inseridos na Filial e Faturação.

As classes CompFils e CompFats são usadas nas queries, de modo a obtermos resultados organizados. Já a classe Input, tem como função ler os dados que o utilizador insere, tendo vários métodos diferentes consoante o tipo de variável que queremos pedir. Por fim, temos a classe crono, que é utlizada para medir tempos de execução de leitura, queries, etc.

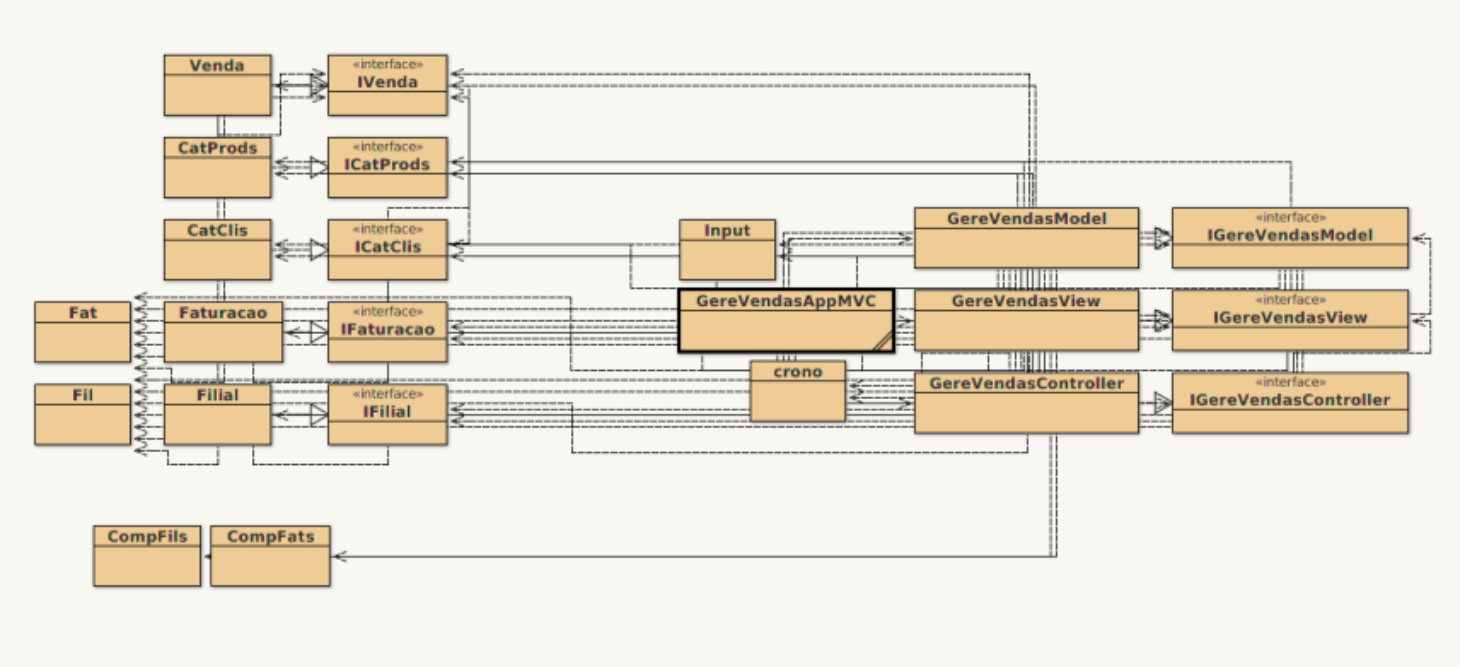
3. INTERFACES

Neste projeto, foi-nos incentivado o uso de interfaces nas principais classes. Na prática, não chegamos a ter várias classes que implementam uma interface, apenas uma para cada. De facto, este programa não exigiu múltiplas classes que implementam uma interface, por exemplo, só existe um tipo de Catálogo de Produtos, só existe um GeraVendasView, etc. Porém, implementar as respetivas interfaces permite modificar este programa muito mais facilmente, se desejado. A criação de, dando o mesmo exemplo que acima, um Catálogo de Produtos alternativo, que implementa a Interface de Catálogo de Produtos (ICatProds), é feita sem muitas mudanças sendo que o programa trabalha sempre com interfaces, generalizando o código.

Além disto, também trabalhamos com interfaces indiretamente ao escolher que tipo de estruturas usamos para diferentes ocasiões. Isto é, ao usarmos, p.e., um Set, às vezes é mais beneficial defini-lo como TreeSet e outras vezes como HashSet – depende das operações que efetuamos neste.

4. ARQUITETURA FINAL DA APLICAÇÃO

Felizmente, através do BlueJ, adquirimos um esquema muito perceptível da nossa aplicação:



5. TESTES DE PERFORMANCE

Foi pedido como parte do trabalho fazer vários testes de tempos para o nosso programa. Realizamos estes testes num Computador Fixo, devido à capacidade limitada dos nossos Portáteis.

5.1 LEITURA DO TXT

	#1	#2	#3	#4	#5	Média
1M	0.406	0.438	0.454	0.414	0.421	0.427
3M	0.628	0.619	0.615	0.637	0.627	0.626
5M	0.785	0.757	0.745	0.804	0.798	0.778

5.2 LEITURA DO TXT + PARSING

	#1	#2	#3	#4	#5	Média
1M	0.906	0.922	0.895	0.898	0.926	0.909
3M	1.718	1.778	1.688	1.727	1.693	1.721
5M	2.597	2.533	2.502	2.548	2.557	2.547

5.3 LEITURA DO TXT + PARSING + VALIDAÇÃO

	#1	#2	#3	#4	#5	Média
1M	1.153	1.156	1.104	1.112	1.104	1.126
3M	2.253	2.202	2.251	2.274	2.213	2.238
5M	3.531	3.519	3.371	3.413	3.517	3.470

5.4 QUERIES

Para testarmos os tempos de execução das queries, usando diferentes estruturas, decidimos usar o ficheiro de vendas "Vendas_1M.txt". Como várias queries usam vários tipos de estruturas, experimentamos várias combinações para cada uma.

QUERY 5:

TreeMap	X	X		
HashMap			X	X
ArrayList	X		X	
Vector		X		X
TEMPO	0.330	0.385	0.381	0.383

QUERY 6: (200 produtos)

TreeMap	X	X		
HashMap			X	X
ArrayList	X		X	
Vector		X		X
TEMPO	2.147	2.162	1.292	1.308

QUERY 7:

TreeMap	X	X		
HashMap			X	X
ArrayList	X		X	
Vector		X		X
TEMPO	0.467	0.428	0.419	0.478

QUERY 8:

TreeMap					X	X	X	X
HashMap	X	X	X	X				
ArrayList	X	X			X	X		
Vector			X	X			X	X
TreeSet	X		X		X		X	
HashSet		X		X		X		X
TEMPO	0.738	0.635	0.746	0.589	0.869	0.766	0.870	0.728

QUERY 9: (produto mais vendido, WX1593, 15 clientes distintos)

TreeMap	X	X		
HashMap			X	X
ArrayList	X		X	
Vector		X		X
TEMPO	0.533	0.531	0.526	0.541

Estudando estes vários resultados, podemos chegar a certas conclusões. É verdade que todos estes tempos foram bastante semelhantes e, se efetuássemos vários testes, poderíamos chegar a resultados diferentes. Mas, mesmo assim, podemos notar certas tendências: ArrayList, HashMap e HashSet. Em grande parte dos nossos métodos, foram estas as estruturas usadas. Porém, gostaríamos de ter obtido estes resultados mais cedo para que pudessemos alterar certas queries e melhor alguns tempos.

5. CONCLUSÃO

Em conclusão, consideramos a nossa prestação nesta trabalho boa. Estamos satisfeitos com os tempos obtidos e com a organização alcançada neste programa. Este projeto ajudou-nos a compreender o conceito de encapsulamento e o modo como o Java traz aspetos beneficiais para tal.

Por outro lado, sentimos que dado mais tempo, teriamos alcançado uma apresentação do nosso código mais limpa, em termos de documentação, dado que não a conseguimos terminar em todos os módulos, por motivos de gestão de tempo. Era necessário uma maior atenção ao detalhe para atingirmos o resultado que queriamos.

Do grupo 23, do turno PL5, constituído por:

João da Cunha e Costa, nº 84775

Hugo André Coelho Cardoso, nº 85006

Filipa Alves dos Santos, nº 83631