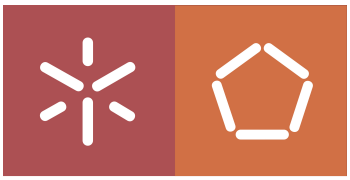

Universidade do Minho
Escola de Engenharia

GestVendas

Relatório do projeto de *Java* - LI3

Grupo 66

José Magalhães,A85852

Luís Ramos,A83930

Luís Vila,A84439

Conteúdo

1 Introdução

No âmbito da unidade curricular Laboratórios de Informática III, foi proposto desenvolver um projeto em *JAVA*, mais concretamente um Sistema de Gestão de Vendas (SGV). Este passa por armazenar e tratar informação presente em ficheiros TXT. Esses ficheiros incluem informação sobre os Produtos, os Clientes e sobre as Vendas. Toda esta informação está organizada por códigos.

Utilizou-se no nosso trabalho estruturas de dados criadas por nós, apresentadas à frente, pelo que esta decisão permitiu moldar as estruturas conforme as nossas necessidades.

Assim, criaram-se estruturas que fossem de fácil acesso conforme as nossas necessidades de coletar informação, tentando assim potenciar a eficiência do nosso programa.

2 Diagrama de classes

2.1 *GestVendasAppMVC*

Classe que contém a *main*. Esta, tendo apenas 8 linhas de código, é de extrema importância pois dá inicio às 3 principais classes deste programa, o *GestVendasController*, o *GestVendasModel* e o *GestVendasView*, dando assim início à aplicação através do *GestVendasController*.

```
public class GestVendasAppMVC implements Serializable {


    public static void main(String[] args){
       IGestVendasModel model = new GestVendasModel();
       model.createData();
       if (model==null) { out.println("Erro"); System.exit(-1);}


       IGestVendasView view = new GestVendasView();


       IGestVendasController control = new GestVendasController();
       control.setModel(model);control.setView(view);
       control.start();
       System.exit(0);
    }
}
```

GestVendasModel

Respeitando o modelo *MVC*, a classe *GestVendasModel* contém todas as variáveis e métodos necessários para o funcionamento do programa, como podemos analisar pelo seguinte código.

```
private CatalogoClientes catClientes;
private CatalogoProdutos catProdutos;
```

```
private Filial filial1;
private Filial filial2;
private Filial filial3;
private FaturacaoGlobal faturacaoGlobal;
private UltimoFicheiro uf;
```

GestVendasView

A classe *GestVendasView*, por sua vez,é responsável pela apresentação não só de menus, mas também de algumas respostas,consoante o pretendido, ao utilizador.

```
private Menu menu;
```

O *Menu* acima referido contém na sua definição as seguintes variáveis de instância:

```
private int numEscolhido;
private List<String> escolhas;
```

GestVendasController

Por fim, a classe a *GestVendasController* efetua a comunicação entre estes as duas classes acima referidas, sendo, assim, o elo de ligação de toda a aplicação.

```
private IGestVendasView view;
private IGestVendasModel model;
private boolean ficheiroLido;
private boolean dadosLidos;
```

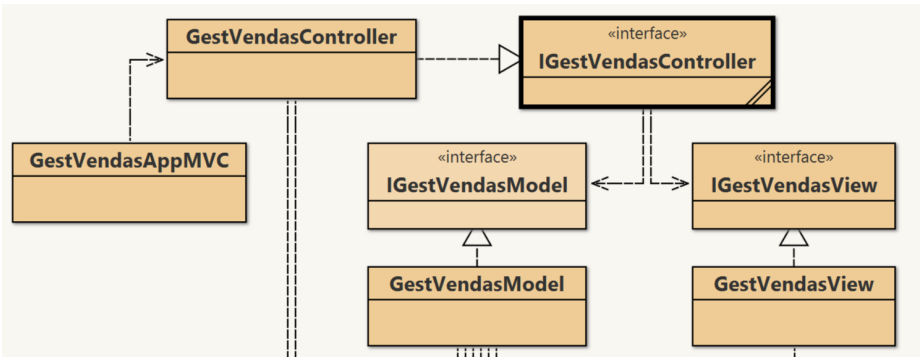

Figura 1: Modelo MVC.

3 Análise das *Querys*

Query	CatClientes	CatProdutos	FaturacaoGlobal	Filial
1		VARR	VARR	
2			Mês	Mês
3				Cliente
4			Produto	Produto
5				Cliente
6			VARR	VARR
7				VARR
8				VARR
9				Produto
10			VARR	

Figura 2: Tabela das *querys*.

Depois de concluída esta tabela, chegou-se à conclusão de como seriam organizadas as estruturas de dados, sempre definidas consoante o mais rentável para esta *App*.

A faturação seria organizada preferencialmente por mês, e de seguido por código de Produto.

A organização da filial por sua vez passava, em primeiro lugar por código de Cliente sendo depois por código de Produto, satisfazendo, assim, todas as *querys*, sempre com o intuito de maximizar a eficiência da aplicação.

4 Estruturas de dados

4.1 Classe *CatalogoProdutos*

```
private HashSet<String> catProdutos;
```

Variável criada com o objetivo de guardar todos os códigos dos produtos dados no ficheiro *Produtos.txt*.

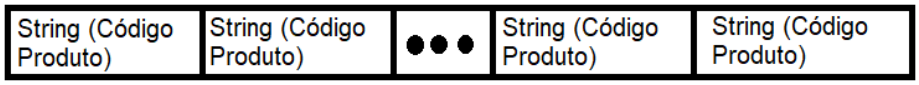

Figura 3: Estrutura do catálogo de produtos.

4.2 Classe *CatalogoClientes*

```
private HashSet<String> catClientes;
```

Variável criada com o objetivo de guardar todos os códigos dos clientes dados no ficheiro *Clientes.txt*.

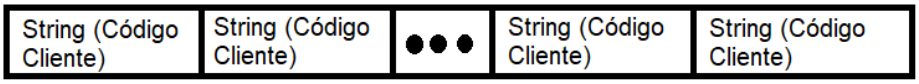

Figura 4: Estrutura do catálogo de clientes.

4.3 Classe *FaturacaoGlobal*

```
private HashMap<Integer, Faturacao> produtosPorMes;
```

Esta classe contém uma variável, um *HashMap* como acima demonstrado. Este é constituído por uma *Key* que, neste caso, corresponderá ao número equivalente ao mês, sendo que o seu respetivo *Value* será uma classe auxiliar chamada *Faturacao*.

Classe Faturacao

```
private HashMap<String, Fatura> produtosVendidos;
```

Esta contém também um *HashMap*, como variável, em que a *Key* é o código de cada produto vendido num determinado mês. Cada código de produto, terá como correspondência um *Value*,sendo este uma outra classe auxiliar denominada *Fatura*, demonstrada de seguida.

Classe Fatura

```
private int[] quantidade, nVendas;
private double[] receita;
```

Esta última classe contém três *arrays*, cada um com tamanho fixo referente ao número de filial (3) correspondente. Dos ditos anteriormente, dois serão de *Integer*, contendo as quantidades vendidas e o número de vezes que um determinado produto foi vendido.

O terceiro e último corresponde ao tipo *Double*,que contém a receita de cada produto.

De notar que esta estrutura será fundamental para a concretização de várias *querys*.

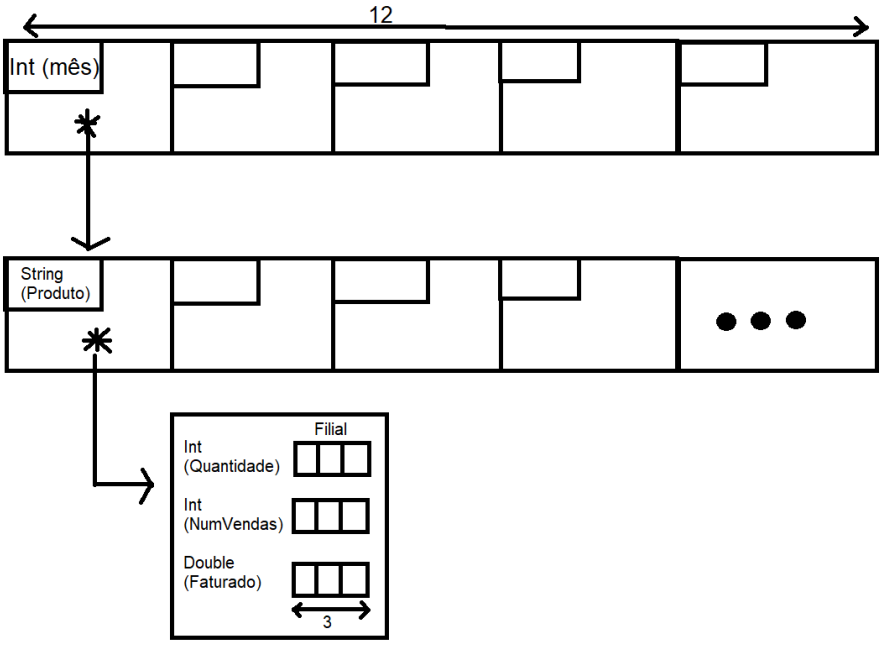

Figura 5: Estrutura da faturação global.

4.4 Classe *Filial*

```
private HashMap<String,InfVendaCli> filial;
```

Esta classe contém como variável um *HashMap*. Este é constituído por uma *Key* que, neste caso, corresponderá ao código de cada cliente que efetuou compras, sendo que o respetivo *Value* será uma classe auxiliar chamada *InfVendaCli*.

Classe InfVendaCli

```
private HashMap<String,InfVendaProd> produtosComprados;
```

Esta contém também um *HashMap*, como variável, em que a *Key* é o código de cada produto comprado pelo cliente. Cada código de produto, terá como *Value* uma outra classe auxiliar denominada *InfVendaProd*.

Classe InfVendaProd

```
private int quantidade;
private double preco;
private int mes;
```

Esta última classe contem três variáveis, duas delas do tipo *Integer* e uma do tipo *Double*. As duas primeiras correspondem à quantidade comprada pela cliente e o mês correspondente à compra em questão. A terceira variável corresponde ao valor do preço na altura da compra do produto.

De notar que esta estrutura será fundamental para a concretização da maioria das *querys* pedidas.

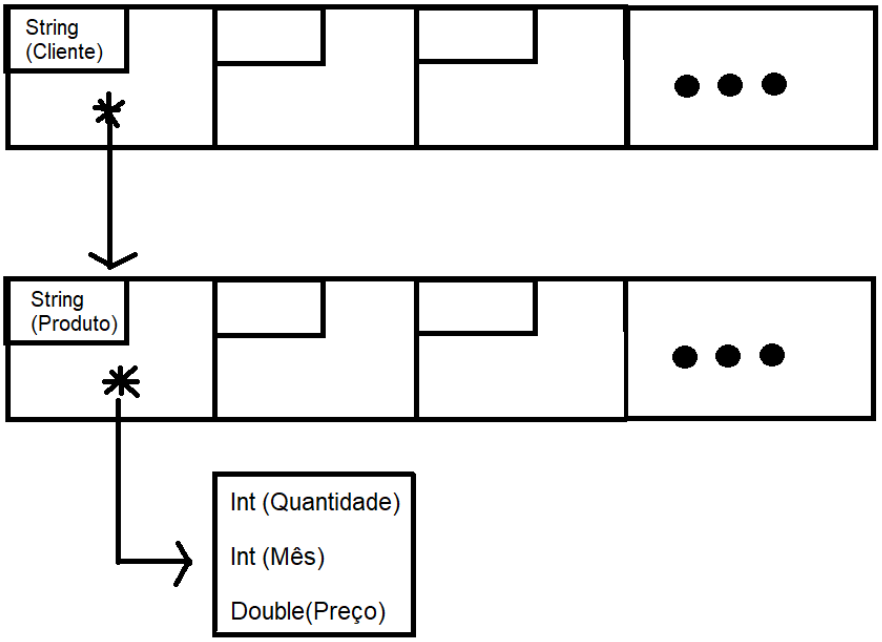

Figura 6: Estrutura da filial.

5 Testes de performance

5.1 *Teste 1*

A primeira tabela é referente ao teste de *performance* que consiste apenas em ler os ficheiros base *Vendas 1M*, *Vendas 3M* e *Vendas 5M*, usando as duas classes dadas nas aulas TP, *BufferedReader()* e *Files*.

Estes dados permitiram chegar à conclusão que, nestes modos, o método de leitura *BufferedReader()* seria o mais vantajoso para o projeto.

	WithFiles_1M	WithBR_1M	WithFiles_3M	WithBR_3M	WithFiles_5M	WithBR_5M
1	0,629	0,697	1,502	0,667	2,217	1,118
2	0,273	0,308	0,661	0,564	0,663	0,841
3	0,195	0,146	0,553	0,6018	0,731	0,709
4	0,406	0,163	0,666	0,427	0,893	0,783
5	0,144	0,264	0,367	0,565	0,802	0,812
6	0,373	0,133	0,537	0,37	0,883	0,745
7	0,204	0,133	0,504	0,336	1,036	0,783
8	0,218	0,135	0,553	0,397	1,005	0,823
9	0,192	0,124	0,529	0,377	0,853	0,997
10	0,211	0,128	0,495	0,543	0,861	0,901
média	0,2845	0,2231	0,6367	0,48478	0,9944	0,8512

Figura 7: Teste 1.

5.2 *Teste 2*

A seguinte tabela é semelhante à tabela anterior, sendo que esta já tem em consideração o tempo de *parsing*, ou seja, da separação dos vários campos da linha, criando assim uma *Venda*.

Estes novos dados contrariam o que em cima foi verificado, vencendo o *Files*.

	WithFiles_1M	WithBR_1M	WithFiles_3M	WithBR_3M	WithFiles_5M	WithBR_5M
1	1,022	0,847	2,247	2,1	4,772	4,045
2	0,838	0,914	2,025	2,396	4,088	3,567
3	0,729	0,777	2,36	2,23	4,878	5,589
4	0,743	0,943	2,066	2,06	4,534	3,666
5	0,711	0,684	2,089	3,439	3,374	3,345
6	0,813	0,735	2,439	2,649	3,922	5,143
7	0,839	0,74	2,252	2,35	4,161	3,522
8	0,619	0,744	2,064	2,006	3,505	4,026
9	0,679	0,671	2,035	2,05	3,807	4,857
10	0,686	0,675	2,423	4,093	3,887	3,512
média	0,7679	0,773	2,2	2,5373	4,0928	4,1272

Figura 8: Teste 2.

5.3 *Teste 3*

Como 3º teste, são apresentados os tempos da leitura do ficheiro, *parsing* das linhas e ainda a posterior validação das vendas.

Foi possível concluir que a variação dos tempos são bastante pequenas, resultando assim em ligeiras diferenças.

	WithFiles_1M	WithBR_1M	WithFiles_3M	WithBR_3M	WithFiles_5M	WithBR_5M
1	0,973	1,079	2,602	2,461	4,051	4,024
2	1,005	1,009	2,888	2,505	3,825	3,986
3	1,009	1,033	2,614	2,682	3,786	3,817
4	1,018	1,039	2,624	2,542	3,78	3,832
5	1,002	1,031	2,588	2,454	4,036	3,806
6	1,022	1,022	2,604	2,647	4,012	3,866
7	0,996	1,007	2,575	2,63	3,984	3,999
8	1,047	1,041	2,591	2,608	3,935	4,103
9	0,989	0,973	2,543	2,594	3,952	3,997
10	0,972	0,978	2,602	2,619	3,894	4,032
média	1,0033	1,0212	2,6231	2,5742	3,9255	3,9462

Figura 9: Teste 3.

5.4 *Teste 4*

O último teste consiste em fazer testes de *performance* das queries mais complexas (de 5 a 9). Para tal, tiveram de ser efetuadas algumas mudanças no código de forma a resolver as mesmas, alterando as estruturas de modo a coincidir com o pedido no enunciado.

	Antes (1M)	Depois (1M)	Antes (3M)	Depois (3M)	Antes (5M)	Depois (5M)
5	0,006	0,006	0,007	0,007	0,007	0,008
6	0,55	0,32	1,049	0,65	1,212	0,976
7	0,144	0,142	0,189	0,177	0,221	0,221
8	0,553	0,797	1,334	2,155	3,156	5,508
9	0,156	0,179	0,708	0,606	1,122	3,663

Figura 10: Teste 4.

Analisando a tabela conclui-se que com as mudanças efetuadas, a eficiência de algumas das queries melhorou, porém, outras acabaram por se tornar mais lentas.

6 Conclusão

Quanto à eficiência do projeto, no geral, consideramos que foi positiva, pois permite carregar os ficheiros numa média de 4.1 segundos pelo que ainda resolve as queries em tempo aceitável. O método usado para este carregamento foi o *BufferedReader()* pois, embora ambos os métodos possuam tempos bastante equilibrados, considerámos, após testes iniciais, que este traria mais vantagens à *App*.

Quanto à parte pedagógica, este projeto foi muito enriquecedor para o nosso coletivo, pois melhorou a nossa capacidade de trabalhar com estruturas de dados, assim como fez com que adquirissemos agilidade com a linguagem de programação *JAVA*. Permitiu também que descobríssemos propriedades sobre esta que desconhecíamos até ao momento.

Para além disso, ensinou-nos também a fazer o tratamento de grandes quantidades de informação, tendo sido bastante desafiante encontrar um equilíbrio entre a qualidade e eficiência.

Em suma, o esforço coletivo foi grande, de modo a garantir boas soluções para o proposto, mas que, por fim, consideramos que os objetivos principais foram cumpridos.