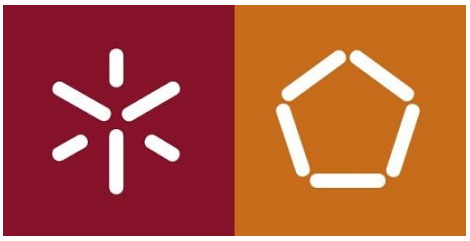


Escola de Engenharia
Departamento Informática
Mestrado Integrado em Engenharia Informática
Laboratórios de Informática III

Sistema de Gestão e Vendas

Projeto Java

Grupo 50

Fernando Henrique Carvalho Lopes (A89472)

Luis Guilherme Guimarães de Araújo (A86772)

Pedro Almeida Fernandes (A89574)

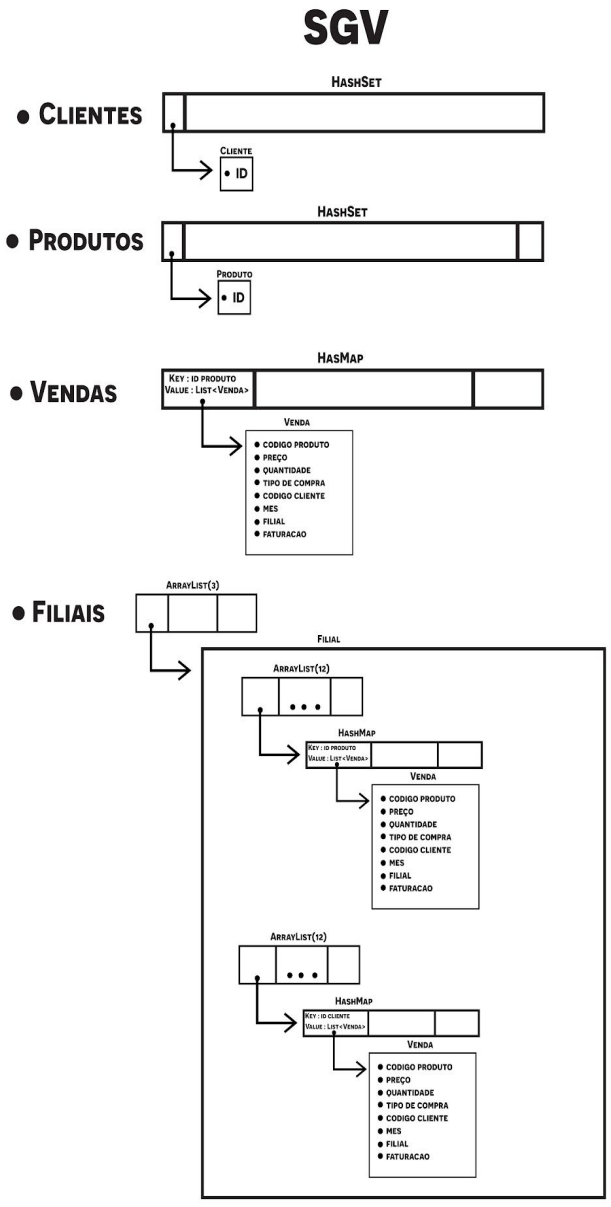
Braga, abril 2020

Índice

Introdução

Neste projeto tínhamos como objetivo realizar um sistema de gestão de vendas. Para o qual tivemos de usar interfaces e coleções de JCF (“Java Collections Framework”) como estruturas de dados para nos darem resposta quer a consultas interativas de informação relativas à gestão básica de uma cadeia de distribuição, quer a dados estatísticos das mesmas. Embora com uma performance inferior ao que seria de esperar num projeto realizado em C, visto que Java é bastante mais lento, tentámos dar respostas eficazes a cada um dos problemas que nos foram propostos. Usámos durante todo o trabalho um pouco de tudo, List, Set e Map, conforme nos foi parecendo mais eficaz ou útil.

Estrutura da Base de Dados



- Para armazenar quer clientes quer produtos optamos por usar HashSets contendo a String de identificação de cada um dos clientes. Fizemos esta escolha pela maior rapidez desta estrutura.

- Para as Vendas usamos um HashMap, onde a key é um ID de produto e o value é a lista de compras em que esse produto foi comprado. Mais uma vez optamos por HashMap em vez de TreeMap devido à maior rapidez de acesso aos elementos desta estrutura.

- As filiais é a estrutura mais complexa do nosso trabalho. Na sua base tem um ArrayList com tamanho igual ao número de filiais. Dentro de cada um destes existem mais dois ArrayLists cada um com tamanho fixo de 12, correspondendo cada índice a um mês do ano. Estes ArrayLists diferem na key, sendo um deles um ID de produto e no outro um ID de cliente, sendo o value de ambos as vendas em que cada key está incluída nesse mês nessa filial.

Diagrama de Classes Model

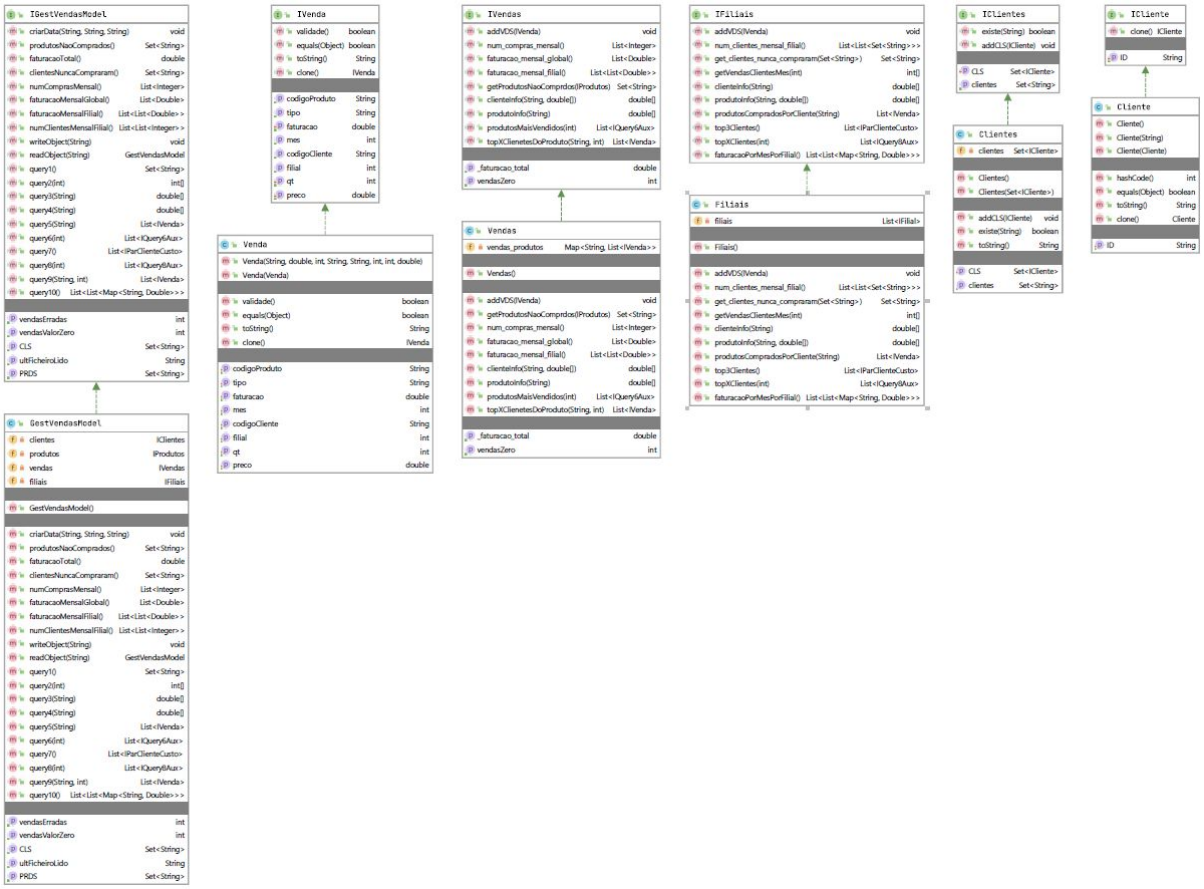


Diagrama de Classes Controller

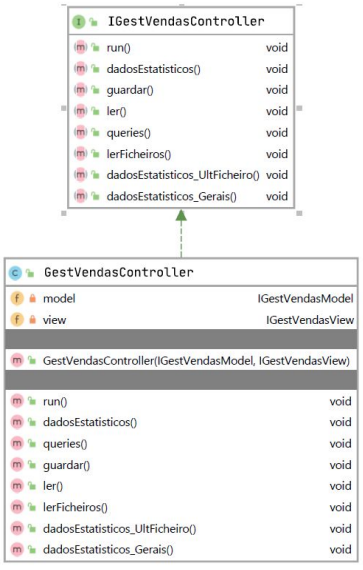


Diagrama de Classes View

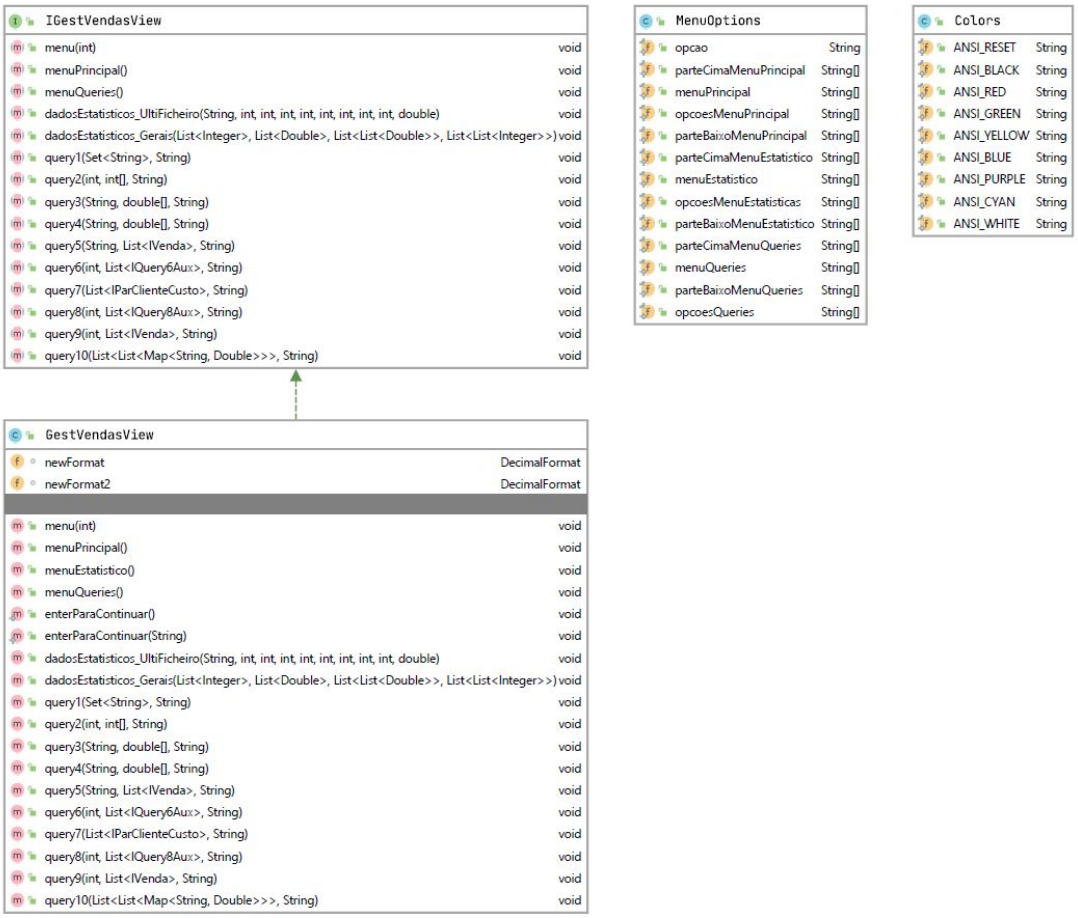
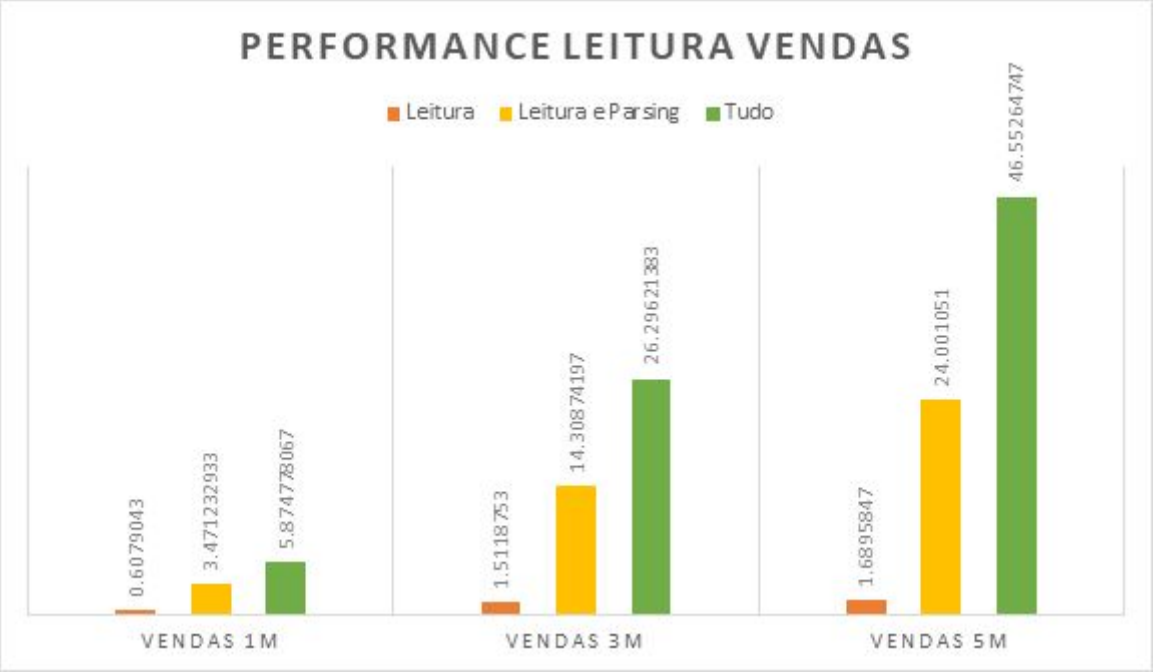


Diagrama de Classes Adicionais

Performance das diversas Leituras das Vendas



	Leitura	Leitura e Parsing	Tudo
Vendas 1M	0.6079043	3.471232933	5.874778067
Vendas 3M	1.5118753	14.30874197	26.29621383
Vendas 5M	1.6895847	24.001051	46.55264747

Perante estes dados, conseguimos perceber que a leitura dos diversos ficheiros de dados aumenta consoante o tamanho destes, o que seria de esperar. Conseguimos ainda perceber que a leitura é mais demorada na parte do parsing, pois este terá de validar todas as componentes que envolvem as vendas. Seria ainda mais lento, se as estruturas dos clientes e produtos fossem guardadas em estruturas de pouca acessibilidade, pois esta leitura é crucial para a validação das vendas. Com isto, ficou nos óbvia a escolha das estruturas dos clientes e produtos. Para que conseguíssemos corresponder aos objetivos impostos pelos docentes, decidimos armazenar as vendas globais relacionando-as com o seu código de produto. Ainda para responder às queries, optamos por dividir a parte das vendas por filiais em filiais, meses e a relaçao produto-venda, cliente-venda.

Performance Query 5

1 - Usando HashMap e dado Cliente : V1817

Foram realizadas 3 tentativas e entre cada uma delas era fechado completamente o programa. Foram obtidos os seguintes resultados :

0.0011434 segundos
0.0011455 segundos
0.0012316 segundos

A média obtida foi de 0.0011735 segundos.

2 - Usando TreeMap e dado Cliente : V1817

Foram realizadas 3 tentativas e entre cada uma delas era fechado completamente o programa. Foram obtidos os seguintes resultados :

1. 0.0011561 segundos
2. 0.0017266 segundos
3. 0.0013267 segundos

A média obtida foi de 0.0014031333 segundos.

Conclusão Performance Query 5

	Teste 1	**Teste 2**	**Teste 3**	**Média**
Opção 1	0.0011434	0.0011455	0.0012316	0.0011735
Opção 2	0.0011561	0.0017266	0.0013267	0.0014031

Como podemos observar pelos resultados a opção 1 é mais rápida que a opção 2 uma vez que esta segunda leva mais aproximadamente 1,20 mais a terminar a realização da função. No trabalho final optamos pela opção 1 devido à sua maior rapidez na execução.

Performance Query 6

1 - Usando HashMap + ArrayList dado número : 50

Foram realizadas 3 tentativas e entre cada uma delas era fechado completamente o programa. Foram obtidos os seguintes resultados :

1. 0.3582697 segundos
2. 0.3915097 segundos
3. 0.3765906 segundos

A média obtida foi de 0.3754567 segundos.

2 - Usando HashMap + Vector dado número : 50

Foram realizadas 3 tentativas e entre cada uma delas era fechado completamente o programa. Foram obtidos os seguintes resultados :

1. 0.3785324 segundos
2. 0.3922858 segundos
3. 0.3663184 segundos

A média obtida foi de 0.3790455 segundos.

3 - Usando TreeMap + ArrayList dado número : 50

Foram realizadas 3 tentativas e entre cada uma delas era fechado completamente o programa. Foram obtidos os seguintes resultados :

1. 0.3485466 segundos
2. 0.3856750 segundos
3. 0.3976803 segundos

A média obtida foi de 0.3863874 segundos.

4 - Usando TreeMap + Vector dado número : 50

Foram realizadas 3 tentativas e entre cada uma delas era fechado completamente o programa. Foram obtidos os seguintes resultados :

1. 0.4029049 segundos
2. 0.3848316 segundos
3. 0.3714259 segundos

A média obtida foi de 0.3863875 segundos.

Conclusão Performance Query 6

	Teste 1	Teste 2	Teste 3	Média
Opção 1	0.3582697	0.3915097	0.3765906	0.3754567
Opção 2	0.3785324	0.3922858	0.3663184	0.3790455
Opção 3	0.3485466	0.3856750	0.3976803	0.3863874
Opção 4	0.4029049	0.3848316	0.3714259	0.3863875

Como podemos observar os resultados para esta query são muito semelhantes entre as diferentes escolhas de estruturas, no entanto podemos ver em geral que embora ligeiramente, HashMaps são mais rápidos que TreeMaps e ArrayLists que Vectors, daí termos escolhido a opção 1.

Performance Query 7

1 - Usando HashMap + ArrayList

Foram realizadas 3 tentativas e entre cada uma delas era fechado completamente o programa. Foram obtidos os seguintes resultados :

1. 0.3068524 segundos
2. 0.3155548 segundos
3. 0.3112300 segundos

A média obtida foi de 0.3112124 segundos.

2 - Usando HashMap + Vector

Foram realizadas 3 tentativas e entre cada uma delas era fechado completamente o programa. Foram obtidos os seguintes resultados :

1. 0.3231382 segundos
2. 0.3186103 segundos
3. 0.2922593 segundos

A média obtida foi de 0.3113359 segundos.

3 - Usando TreeMap + ArrayList

Foram realizadas 3 tentativas e entre cada uma delas era fechado completamente o programa. Foram obtidos os seguintes resultados :

1. 0.6218828 segundos
2. 0.6333417 segundos
3. 0.6057227 segundos

A média obtida foi de 0.6203157 segundos.

4 - Usando TreeMap + Vector

Foram realizadas 3 tentativas e entre cada uma delas era fechado completamente o programa. Foram obtidos os seguintes resultados :

1. 0.6134824 segundos
2. 0.6210597 segundos
3. 0.6203023 segundos

A média obtida foi de 0.6182815 segundos.

Conclusão Performance Query 7

	Teste 1	Teste 2	Teste 3	Média
Opção 1	0.3068524	0.3155548	0.3112300	0.3112124
Opção 2	0.3231382	0.3186103	0.2922593	0.3113359
Opção 3	0.6218828	0.6333417	0.6057227	0.6203157
Opção 4	0.6134824	0.6210597	0.6203023	0.6182815

Tal como na query anterior, a diferença entre ArrayLists e Vectors é mínima, no entanto a diferença de TreeMap para HashMap é enorme, sendo as HashMaps cerca de 2 vezes mais rápida, daí termos selecionado a opção 1.

Performance Query 8

1 - Usando HashMap + ArrayList dado número 10

Foram realizadas 3 tentativas e entre cada uma delas era fechado completamente o programa. Foram obtidos os seguintes resultados :

1. 0.1942595 segundos
2. 0.1897359 segundos
3. 0.1962842 segundos

A média obtida foi de 0.1934265 segundos.

2 - Usando HashMap + Vector dado número 10

Foram realizadas 3 tentativas e entre cada uma delas era fechado completamente o programa. Foram obtidos os seguintes resultados :

1.
2. 0.1968147 segundos
2. 0.1852795 segundos
3. 0.2066170 segundos

A média obtida foi de 0.1962371 segundos.

3 - Usando TreeMap + ArrayList dado número 10

Foram realizadas 3 tentativas e entre cada uma delas era fechado completamente o programa. Foram obtidos os seguintes resultados :

1. 0.4155390 segundos
2. 0.3928306 segundos
3. 0.3934014 segundos

A média obtida foi de 0.4005821 segundos.

4 - Usando TreeMap + Vector dado número 10

Foram realizadas 3 tentativas e entre cada uma delas era fechado completamente o programa. Foram obtidos os seguintes resultados :

1. 0.4048842 segundos
2. 0.3892372 segundos
3. 0.4090221 segundos

A média obtida foi de 0.4010492 segundos.

Conclusão Performance Query 8

	Teste 1	Teste 2	Teste 3	Média
Opção 1	0.1942595	0.1897359	0.1962842	0.1934265
Opção 2	0.1968147	0.1852795	0.2066170	0.1962371
Opção 3	0.4155390	0.3928306	0.3934014	0.4005821
Opção 4	0.4048842	0.3892372	0.4090221	0.4010492

A diferença entre ArrayLists e Vectors volta a ser mínima e a diferença de TreeMap para HashMap volta a ser enorme, sendo as HashMaps cerca de 2 vezes mais rápida, daí termos selecionado a opção 1.

Performance Query 9

1 - Usando HashMap + ArrayList dado nº 5 e prod RX1771

Foram realizadas 3 tentativas e entre cada uma delas era fechado completamente o programa. Foram obtidos os seguintes resultados :

1. 0.0029043 segundos
2. 0.0024547 segundos
3. 0.0029652 segundos

A média obtida foi de 0.0027747 segundos.

2 - Usando HashMap + Vector dado nº 5 e prod RX1771

Foram realizadas 3 tentativas e entre cada uma delas era fechado completamente o programa. Foram obtidos os seguintes resultados :

1. 0.0027175 segundos
2. 0.0032711 segundos
3. 0.0029454 segundos

A média obtida foi de 0.0029780 segundos.

3 - Usando TreeMap + ArrayList nº 5 e prod RX1771

Foram realizadas 3 tentativas e entre cada uma delas era fechado completamente o programa. Foram obtidos os seguintes resultados :

1. 0.0032135 segundos
2. 0.0028137 segundos
3. 0.0032330 segundos

A média obtida foi de 0.0030867 segundos.

4 - Usando TreeMap + Vector nº 5 e prod RX1771

Foram realizadas 3 tentativas e entre cada uma delas era fechado completamente o programa. Foram obtidos os seguintes resultados :

1. 0.0028886 segundos
2. 0.0027767 segundos
3. 0.0028145 segundos

A média obtida foi de 0.0028266 segundos.

Conclusão Performance Query 9

	Teste 1	Teste 2	Teste 3	Média
Opção 1	0.0029043	0.0024547	0.0029652	0.0027747
Opção 2	0.0027175	0.0032711	0.0029454	0.0029780
Opção 3	0.0032135	0.0028137	0.0032330	0.0030867
Opção 4	0.0028886	0.0027767	0.0028145	0.0028266

Embora não seja a melhor em termos de performance, nesta query escolhemos a opção 3, uma vez que as propriedades das TreeMap eram úteis para nós e os tempos desta query são muito baixos.

Conclusão

A partir dos testes de performance realizados no programa, e segundo as queries propostas pelos docentes, conseguimos concluir que as estruturas que melhor se adaptam ao cenário presente são as estruturas de rápido acesso, como por exemplo Maps e Sets. Isto deve-se ao facto de podermos aceder rapidamente à informação necessária para a realização de um método, como por exemplo, validação de uma venda.

A ordenação de informação também seria indispensável para algumas questões, porém concluímos que seria mais eficiente se esta tivesse rápido acesso e quando fosse necessária a sua ordenação, o processo seria realizado através de uma Tree ou mesmo através da ordenação de uma lista.