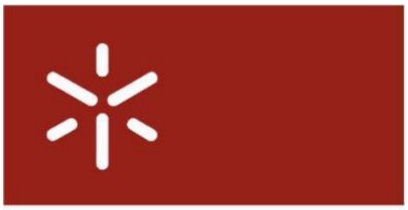


Universidade do Minho

Relatório LI3

Sistema de Gestão de Vendas

Laboratórios de Informática III

MIEI – 2º ano – 2º semestre
2019/2020

Projeto em C

Realizado pelo Grupo 10:

Rúben Correia Cerqueira A89593
Júlio Miguel de Sá Lima Magalhães Alves A89468
Alexandra de Barros Reigada A84584

ÍNDICE

1. Introdução

Após a realização do projeto de gestão de vendas proposto na primeira parte do semestre, foi-nos solicitado realizar um projeto semelhante ao anterior, diferindo na linguagem de programação usada, passando de C para Java.

Fruto das diferenças entre C e Java, achamos mais fácil a implementação das estruturas de dados e da modularidade uma vez que esta caraterística está implícita na linguagem, através da JCF ("Java Collections Framework") e do sistema por classes.

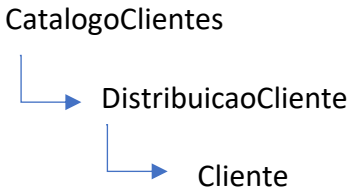
Além disso, este projeto foi concebido com base na criação de interfaces para cada módulo de modo a garantir uma maior possibilidade de expansão e flexibilidade do código.

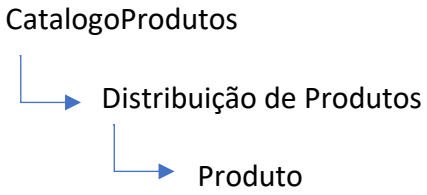
2. Descrição da Estrutura de Dados

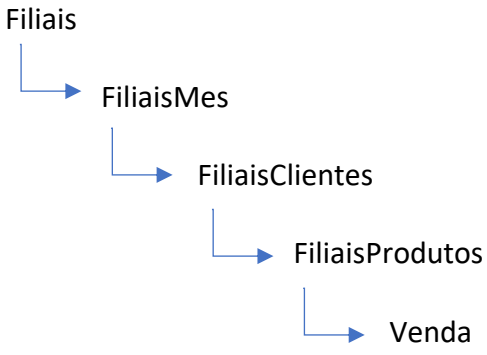
Sobre a organização da estrutura de dados, decidimos adotar uma muito semelhante ao projeto de C, sendo a principal diferença a substituição de árvores binárias por HashMaps.

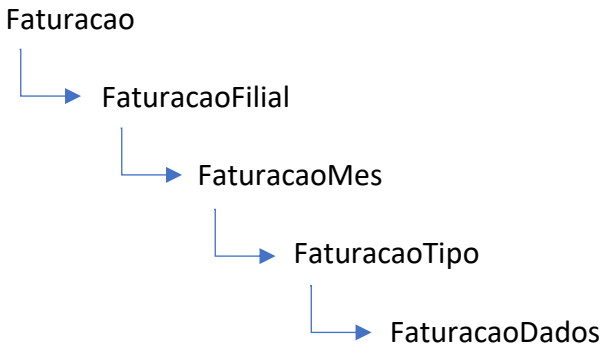
Também foi proposta a estruturação do projeto baseando no modelo MVC (Model-View-Controller) para se tornar mais simples a comunicação entre as diferentes partes do projeto.

O modelo é constituído por 4 módulos principais, Faturação, Filiais, Catálogo de Produtos e Catálogo de Clientes, em que as suas funções consistem em registar dados de faturação do sistema, salvaguardar os registos de venda, armazenar os dados acerca dos clientes e armazenar os dados acerca dos produtos, respetivamente.









A vista é constituída a partir de uma única classe chamada UI.

O controlador é constituído pelo módulo controller, que contém o menu e comunica com a view e o modelo.

2.1 Catálogo de Clientes

A estrutura de dados que constitui o catálogo de clientes é composta por um array de 26 posições que representam cada letra do alfabeto, por exemplo, a posição 0 corresponde à letra 'A' e por aí em diante. Cada posição do array redireciona para uma distribuição de cliente, que é um map<String,ClienteI> , sendo a key o código do cliente e o value a interface ClienteI que guarda as informações do cliente.

2.2 Catálogo de Produtos

A estrutura de dados do catálogo de produtos é semelhante à dos clientes, apenas diferindo na string guardada, uma vez que o código dos produtos é diferente do código dos clientes.

2.3 Gestão de Filiais

A estrutura de dados que compõe as filiais é composta por um array de 3 posições, que correspondem às filiais em causa e nos redireciona para FiliaisMes que em cada posição nos indica outro array de 12 posições, correspondentes aos meses do ano e nos redireciona para FiliaisClientes. As FiliaisClientes são um hashmap com o código do cliente como key e FiliaisProdutosI como value que contem os produtos que o cliente comprou nesse mês e nessa filial. Já as FiliaisProdutos também são um hashmap que tem como key o código do produto e como value uma List<VendaI>, sendo que a VendaI possui as informações sobre essa compra (o preço, a quantidade comprada e o tipo de promoção em vigor).

Além disso também temos, em cada módulo e submódulo, um contador de vendas efetuadas.

No módulo principal, Filiais, também temos informação da quantidade de vendas a custo 0 registadas.

2.4 Faturação Global

A Faturação é um Map<String,FaturacaoFilialI>, sendo a string o código de um produto e o seu value é a interface FaturacaoFilialI. Esta está estruturada num array com 3 posições (uma para cada filial) e em cada posição temos uma interface de FaturacaoMes. A FaturacaoMes é idêntica à FaturacaoFilial, no entanto esta está organizada num array com 12 posições (uma para cada mês) e em cada posição temos a interface FaturaçãoTipoI. A FaturacaoTipo é um array com 2 posições, sendo a primeira posição para as vendas no modo sem promoção e a segunda posição para as vendas com promoção. Cada posição envia-nos para a FaturacaoDados que armazena a quantidade de produtos e a faturação dessa venda.

Também temos informação, em cada módulo e submódulo da faturação obtida em cada estrutura.

Além disso, no módulo principal, também temos dois conjuntos de dados que nos indicam os códigos de cliente que realizaram compras e os códigos de produtos que foram comprados,

2.5 Auxiliar

Para a realização de certas queries sentimos a necessidade de criar uma classe auxiliar chamada MyPair, que consiste na possibilidade de ter tuplos de dados.

Foi-nos facilitada duas classes auxiliares, de nome Crono e Input cujas funções são analisar tempos de execução e ler dados do stdin, respetivamente.

Também tinha sido criada uma classe auxiliar denominada InfoProd (continha informações acerca da faturação, vendas e conjunto de clientes, resultantes da query 4) que posteriormente foi retirada pois foi encontrada uma solução alternativa mais eficiente.

3. Decisão

Para este trabalho prático decidimos optar por usar o mesma organização da estrutura de dados usada em C uma vez que observamos uma boa eficiência com esta organização e já estávamos familiarizados com ela.

Também decidimos alterar as árvores por mapeamentos de hash após analisarmos testes de performance realizados, que iremos abordar mais pormenorizadamente no próximo tópico.

Pensámos também em usar listas, mas estas não são eficientes no que toca à procura de um elemento, uma vez que a procura deste é linear, ou seja, desde o primeiro elemento da lista até ao elemento encontrado, ou, no pior dos casos, até ao fim caso não encontre.

4. Testes de Performance

	HashMap	TreeMap	LinkedHashMap	HashMap	TreeMap	Vector	Array	TreeSet	ArrayList	LinkedList	InfoProd[]
	Limite 5/1000	Limite 5/1000	Limite 5/1000								
1								0.109885			
2							Valor global: 0.00919905		Valor por filial: 3.2235E-5		
3									1.43961E-4		
4									0.12746		0.20705
5				0.001646	5.7357E-4						
6	1.02425 / 0.841039	2.96733 / 2.876913						Limite 5/1000: 0.39366/ 35.02555			
7					0.340937	0.60258	0.624826		0.325322	0.266758	
8		0.49875/ 0.50864									
9		0.065014/ 0.0029959	0.358139 / 0.2155								
10				0.557193							

Tempos de execução para o ficheiro de 1 Milhão em que as estruturas de dados são HashMap.

	HashMap	TreeMap	LinkedHashMap	HashMap	TreeMap	Vector	Array	TreeSet	ArrayList	LinkedList	InfoProd[]
	Limite 5/1000	Limite 5/1000	Limite 5/1000								
1								0.238676			
2							Valor global: 0.01196509		Valor por filial: 9.1974E-5		
3									3.95243E-4		
4									0.131706		0.26248
5				0.001197							
6	2.72377 / 3.43642							Limite 5/1000: 0.57106/ 55.01769			
7					0.7575366						
8		2.084192/ 1.76035									
9		0.081074 / 0.04639									
10				0.566486							

Tempos de execução para o ficheiro de 3 Milhões em que as estruturas de dados são HashMap.

	HashMap	TreeMap	LinkedHashMap	HashMap	TreeMap	Vector	Array	TreeSet	ArrayList	LinkedList	InfoProd[]
	Limite 5/1000	Limite 5/1000	Limite 5/1000								
1								0.07015			
2							Valor global: 0.01166669		Valor por filial: 3.5527E-5		
3									3.808325E-4		
4									0.250988		0.252034
5				0.00271							
6	7.92847 / 8.252264							Limite 5/1000: 0.681676 / 98.4391			
7					3.98795						
8		5.144611/ 4.61646									
9		0.0993151 / 0.07197									
10				0.504901							

Tempos de execução para o ficheiro de 5 Milhões em que as estruturas de dados são HashMap.

	HashMap	TreeMap	LinkedHashMap	HashMap	TreeMap	Vector	Array	TreeSet	ArrayList	LinkedList	InfoProd[]
	Limite 5/1000	Limite 5/1000	Limite 5/1000								
1								0.072229			
2							Valor global: 0.014299245		Valor por filial: 5.9812E-5		
3									3.88581E-4		
4									0.14093106		0.2971532
5				0.001467	6.09185E-4						
6	1.09551 / 0.90524	2.769365 / 2.775488						Limite 5: 0.671291			
7					0.3756396	0.493353	0.6000818				
8		0.554966/ 0.517174									
9		0.092797 / 0.05596									
10				0.557193							

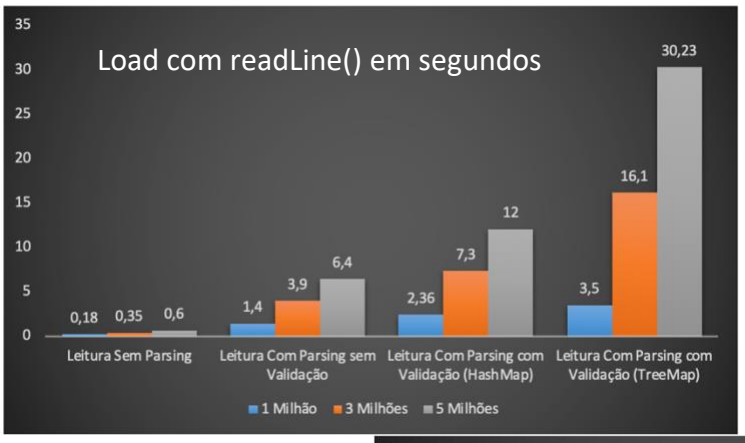
Tempos de execução para o ficheiro de 1 Milhão em que as estruturas de dados são TreeMap.

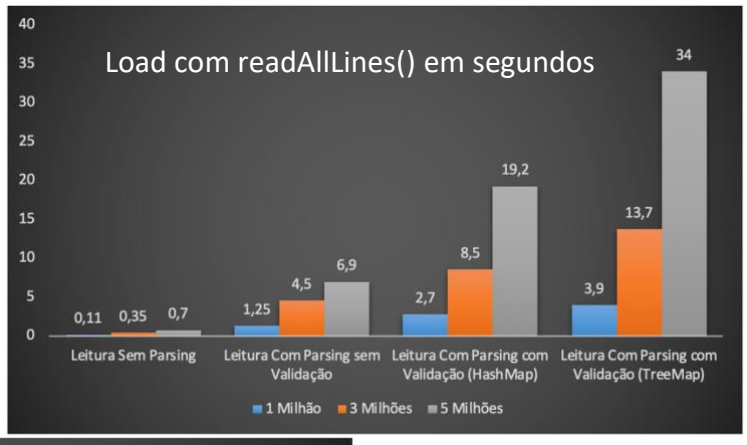
	HashMap	TreeMap	HashMap	TreeMap	Vector	Array	TreeSet	ArrayList	InfoProd[]
	Limite 5/1000	Limite 5/1000							
1							0.639249		
2						Valor global: 0.02058796		Valor por filial: 4.0796E-5	
3								0.0010519	
4								0.501705	0.352202
5			0.001947	0.001486					
6	3.306329 / 2.907025	9.499757 / 9.585726					Limite 5: 1.017132		
7					1.50389	1.613079		1.0541222	
8		2.2708679/ 1.904079							
9		0.170049 / 0.136435							
10			0.581122						

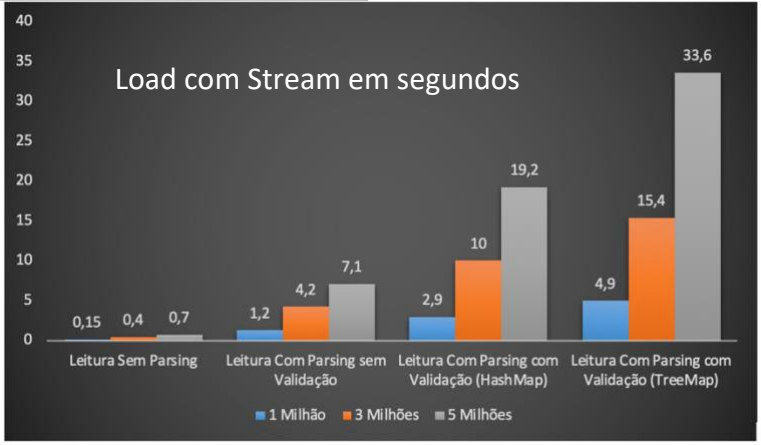
Tempos de execução para o ficheiro de 3 Milhões em que as estruturas de dados são TreeMap.

	HashMap	TreeMap	HashMap	TreeMap	Vector	Array	TreeSet	ArrayList	InfoProd[]
	Limite 5/1000	Limite 5/1000							
1							0.087468		
2						Valor global: 0.0182001		Valor por filial: 6.8165E-5	
3								0.00133997	
4								0.639618	0.579832
5			0.003081	0.001296					
6	7.0470307 / 6.691141	20.85644/ 18.59367					Limite 5: 2.931671		
7					2.360096	2.3864323		1.097119	
8		3.108198/ 4.4382126							
9		0.212945 / 0.178955							
10			0.562101						

Tempos de execução para o ficheiro de 5 Milhões em que as estruturas de dados são TreeMap.







Observando os tempos de execução acima apresentados, podemos constatar que, comparando as estruturas de dados em HashMap e em TreeMap, as estruturas de dados em TreeMap demoraram sempre mais tempo a executar que as estruturas de dados em HashMap. Isto deve-se essencialmente à diferença entre os custos de procura de cada estrutura, visto que a complexidade do tempo de procura da TreeMap é log(n) e o da HashMap é log(1), além disso, observa-se uma grande discrepância nos tempos de carregamento entre estas duas estruturas de dados, uma vez que complexidade do tempo de inserção da TreeMap é log(n) e o da HashMap é log(1).

Para a query 4, como já foi dito anteriormente, tentamos fazer com recurso a uma classe auxiliar (InfoProd), mas observando os tempos de execução em comparação com ArrayList, esta era mais lenta.

Na query 6 , apesar de para limites pequenos o treeSet ser mais rápido, quando usamos limites maiores, começa a demorar tempos demasiado grandes e, por causa disso, decidimos utilizar a aproximação alternativa (HashMap).

Também se pode concluir que, olhando para os gráficos, que o método de carregamento de dados readLine tem um tempo de execução inferior em comparação com os restantes métodos, sendo esse o nosso critério de decisão.

Quanto às restantes queries, testamos para as várias estruturas de dados disponíveis (Map testamos para TreeMap e HashMap, Set testamos para TreeSet e HashSet e List testamos para ArrayList, Vector e LinkedList) e escolhemos as estruturas com menor tempo de execução.

5. Conclusão

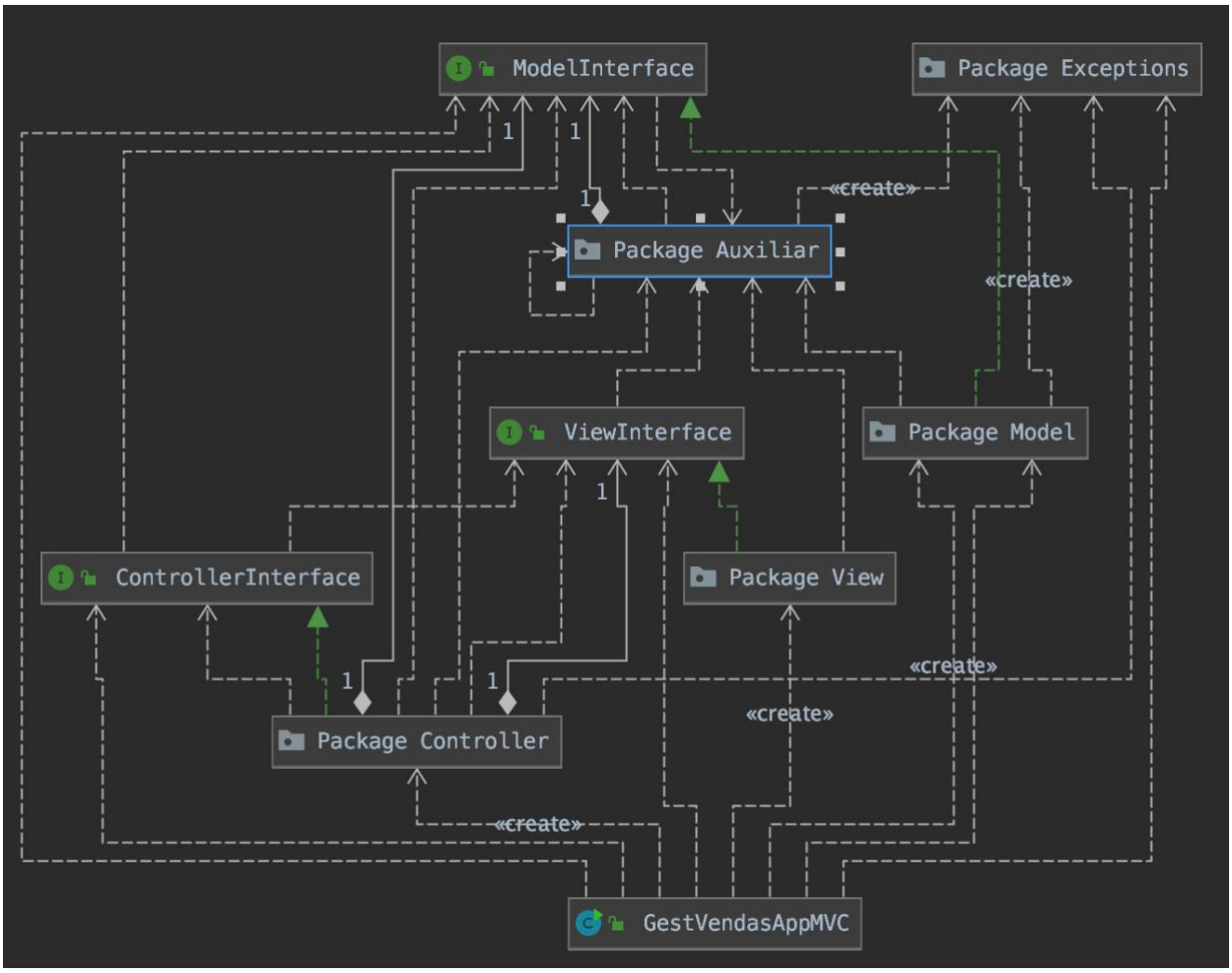
Em suma, sentimos que este trabalho foi útil para consolidar conhecimentos adquiridos no paradigma de programação orientada a objetos, mais concretamente, na linguagem Java.

O nosso trabalho prático, apesar de bons tempos de execução na globalidade, tem as suas fraquezas, nomeadamente nas queries 6 e 8, pois acreditamos que os seus tempos de execução, com um pouco mais de trabalho, podiam ser melhorados. Além disso o programa poderia ter funcionalidades interessantes adicionais como navegar pelas estruturas que armazenam os dados.

Também a nossa vista poderia ser mais trabalhada e mais agradável esteticamente, porém este não era o nosso foco principal, visto que demos mais importância ao funcionamento correto das queries e aos testes de performance.

Apesar disto, é importante realçar a organização do projeto, da simplicidade das estruturas implementadas e o seu bom funcionamento geral que nos permitiu criar esta aplicação.

6. Diagrama de Classes



7. Desenho da Estrutura de Dados