A89583 - Bruno Filipe de Sousa Dias
A89597 - Luís Enes Sousa
A85088 - Pedro Alferedo Fontes Machado

Laboratórios de Informática III

Projeto em JAVA - Grupo 7

Introdução

Este trabalho foi realizado no âmbito da Unidade Curricular de Laboratórios de Informática III. O trabalho passa pela implementação de um sistema capaz de dar resposta a um leque de queries pedidas, sobre a leitura de um conjunto de dados fornecidos em formato de texto pelos docentes. No fundo, consiste na implementação de um Sistema de Gerenciamento de Vendas para uma cadeia de distribuição com 3 filiais.

A primeira fase do trabalho passaria pelo desafio de implementar este sistema na linguagem JAVA. Há semelhança do que foi feito no projeto escrito em C, este trabalho teria como função aumentar o conhecimentos do alunos em JAVA, numa fase ainda um pouco crua da aprendizagem da linguagem, mas também de forçar os alunos a respeitarem os paradigmas da POO (Programação Orientada a Objetos).

Era permitido aos alunos ter como recurso os diferentes tipos de bibliotecas, sem haver inicialmente qualquer restrição por parte dos docentes, ao contrário do que aconteceu com o projeto em C.

Além dos aspetos referidos, o trabalho tinha também como focos prioritários a modularidade, o encapsulamento de dados, o código reutilizável (essenciais para uma boa prática da Programação Orientada a Objetos) e também o MVC (Modelo Vista Controlador). No entanto, algo que é mais importante no projeto de JAVA do que no projeto de C, eram os testes de performance, que passavam por testar dos mais diferentes parâmetros, desde a maneira como a leitura de um ficheiro de texto era realizada, até à forma como as estruturas de dados e as diferentes queries estavam implementadas, com o grande objetivo de se conseguir alcançar o melhor equilíbrio possível entre a qualidade do código e ainda a performance em termos de tempo de load (carregamento dos dados para as estruturas) vs. tempo de resposta da aplicação às diferentes queries.

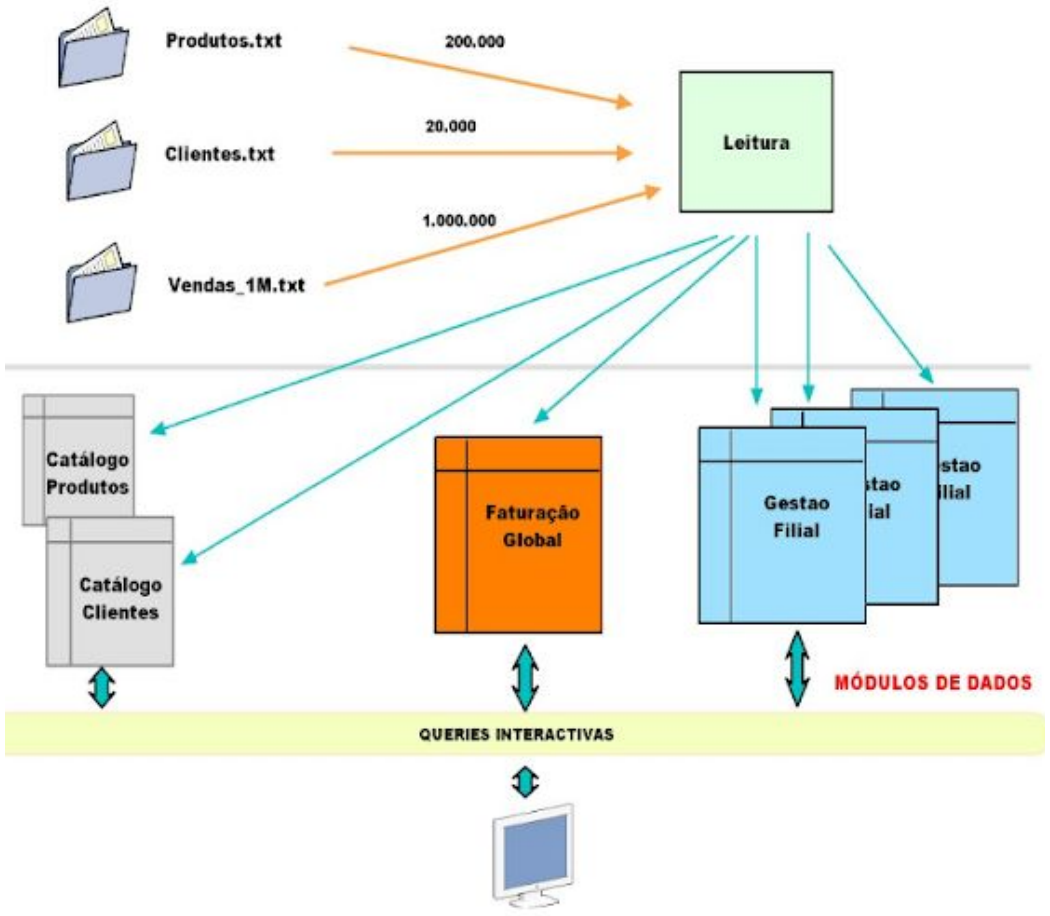
Uma parte muito importante do trabalho foi a análise prévia e cuidada de como poderiam ser implementadas as diferentes estruturas, e se estas poderiam ser “replicadas” do projeto C, embora com algumas alterações, de modo a obter a melhor performance possível. Como havia acontecido no Projeto em C ao longo da realização do trabalho e depois da realização dos Testes de Performance foram surgindo pormenores que poderiam ser ajustados e que otimizaram em grande quantidade a performance final e assim foi feito.

Ferramentas Utilizadas

Na realização deste trabalho, como referido, era possível a utilização das diferentes bibliotecas e API's que o JAVA apresenta. Como sabemos o JAVA possui uma quantidade de classes e API’s tão vasta que são poucas as que usamos. No entanto, e escapando um pouco à normalidade do mesmo leque de API’s todas as vezes utilizadas, operamos também uma API denominada *Swing* como forma de recurso para a construção da Vista do MVC (Modelo Vista Controlador).

Descrição da Estrutura do Projeto

A arquitetura do software é definida por 4 módulos principais: Catálogos de Produtos e Clientes, Faturação Global e Vendas por filial cujos dados são retirados dos ficheiros de texto Produtos.txt, Clientes.txt e Vendas_1M.txt, uma interface que permite a comunicação máquina-utilizador (Modelo Vista Controlador).



Arquitetura de referência para o SGV

O ficheiro Produtos.txt contém cerca de 200.000 códigos de produtos. Cada linha é um código de Produto que é formado por duas letras maiúsculas seguidas de quatro dígitos.

O ficheiro Clientes.txt contém cerca de 20.000 códigos de clientes. Cada linha é um código de Cliente que é formado por uma letras maiúsculas seguidas de quatro dígitos.

O ficheiro Vendas.txt contém cerca de 1.000.000 de registo de vendas, onde cada código é formado por: código do Produto, código do Cliente, preço unitário do produto, número de unidades vendidas, o modo de compra (N - normal ou P - desconto) e o mês da venda.

Catálogo Clientes

Os dados no Catálogo de Clientes são guardados num Set de Interfaces Cliente. Assim, como podemos entender temos uma classe Cliente, onde é definido o cliente, sendo neste caso constituída apenas por uma String. Além disso temos ainda uma interface que é implementada pelo Cliente denominada ICliente. Desta forma, em cada posição do Set temos um ICliente representa um Cliente. Como estamos a utilizar um Set, não temos então códigos de Cliente Repetidos e estes têm como vantagem ficarem ordenados por ordem alfabética (ordem natural), se for utilizada uma TreeSet.

Ao contrário do Projeto em C, e depois de um breve conselho dos professores na apresentação do mesmo, decidimos como é possível ver, utilizar um tipo de Catálogo um pouco mais simples, evitando também um uso dispendioso e evitável de memória, utilizando apenas Strings.

Catálogo Produtos

Os dados no Catálogo de Produtos são guardados num Set de Interfaces Produtos. Assim, como podemos entender temos uma classe Produto, onde é definido o Produto, sendo neste caso constituída apenas por uma String, onde é guardado o código de identificação do mesmo. Além disso temos ainda uma interface que é implementada pela classe Produto denominada IProduto. Desta forma, em cada posição do Set temos um IProduto que representa um Produto. Como estamos a utilizar um Set, não temos então códigos de Cliente Repetidos e estes têm como vantagem ficarem ordenados por ordem alfabética (ordem natural), se for utilizada uma TreeSet.

Ao contrário do Projeto em C, e depois de um breve conselho dos professores na apresentação do mesmo, e tal como feito no catálogo apresentado anteriormente, decidimos optar por um tipo de Catálogo um pouco mais simples, evitando o uso dispendioso e evitável de memória, utilizando apenas Strings.

Faturação Global

A Faturação contém todas as estruturas que serão responsáveis por responder de forma eficiente a questões quantitativas que relacionam os produtos às suas vendas mensais, em modo Normal(N) ou em Promoção(P).

O catálogo das faturas é constituído por um Map, predefinido do Java, onde em cada posição temos uma key, que possui um código de Produto, aos quais os valores do seu value estão relacionados. Neste value, que denominamos NodoFatura possuímos mais

uma vez o código do Produto, igual ao que está na key(decidimos colocar o código de Produto também no value, porque achamos que pode ser útil por diversas razões, uma vez que não temos nenhuma informação da key, que não faça parte do value, o que achamos um ponto super positivo). Além do código de Produto possuímos ainda 2 Lists, onde cada lista possui, neste caso, 3 posições, uma para cada filial. Uma das lists remete para o tipo de compras Normal(N) e a outra remete para o tipo de compras em Desconto(P). Dentro de cada posição de cada uma das listas possuímos 3 arrays de 12 posições. Cada posição remete para cada mês. O primeiro array possui o total de Unidades compradas desse produto para cada mês. O segundo array possui o total faturado por aquele produto para cada mês. E por fim, o terceiro array possui o número total de Compras efetuadas deste produto para cada mês (entendemos uma venda, por exemplo com 140 unidades, como uma única venda aqui).

É importante relembrar que o "Catálogo" da Faturação possui todos os produtos que foram introduzidos a partir dos ficheiros, mesmo aquele que nunca foram vendidos, nem uma única vez, sendo estes guardados no mapa, mas com tudo a 0, em todos os parâmetros.

Filial

A Filial contém todas as estruturas que, para uma dada filial, serão responsáveis por responder de forma eficiente a questões de relacionamento entre Clientes e Produtos.

O catálogo das filiais é constituído por um Map, predefinido do Java, onde em cada posição temos uma key, que possui um código de Cliente, aos quais os valores do seu value estão relacionados. Neste value, que denominamos NodoFilial possuímos mais uma vez o código do Cliente, igual ao que está na key(decidimos colocar o código de Cliente também no value, porque achamos que pode ser útil por diversas razões, uma vez que não temos nenhuma informação da key, que não faça parte do value, o que achamos um ponto super positivo). Além do código de Cliente possuímos ainda 2 Maps, onde cada mapa representa o tipo de compras feita. Assim, um dos Maps representa as compras normais(N) e o outro Map representa as compras em Desconto(P).

É importante agora perceber a maneira como estes Maps são implementados (são de forma análoga, apenas muda o tipo de compra). A única diferença é que se estivermos a inserir uma Venda feita no modo Normal(N) inserimos os dados da mesmo num Map, e se a venda for feito em desconto(P) inserimos os dados no outro Map. Assim, o Map é inicialmente vazio para todos os clientes. Cada posição do Map possui uma key, que vai representar um código de um Produto (que fez pelo menos uma compra naquele Cliente, com naquele modo, naquela filial). Seguidamente no value dessa posição vamos ter mais uma vez o código do Produto, pelas várias razões já apontadas acima, e ainda três arrays com 12 posições cada. Estes arrays são iguais aos referidos no catálogo da Faturação e representam o total de Unidades que esse cliente comprou daquele produto, o total

Faturado e ainda o total de Vendas (lembrando dividido por meses, e que relacionam sempre o cliente em estudo com o Produto em estudo).

É importante realçar que o “Catálogo” da Filial possui todos os clientes, mesmo os que nunca compraram nada. Cada posição, no seu value é constituído inicialmente por maps vazios (size é 0), estando no entanto implementados. Inserimos uma posição no Map caso apareça um produto novo para um dado Cliente, caso contrário, já tenha aquele produto, para aquele modo, apenas incrementamos os valores que já tínhamos antes com os novos. Relembramos ainda que existe um catálogo destes para cada filial.

Gest Vendas

O Gest Vendas vai ser o centro do Model de Dados do modelo MVC. Este GestVendas vai possuir, um catálogo de Clientes, um catálogo de Produtos, um catálogo de Faturação, e ainda um catálogo de Filial, para cada uma das diferentes filiais que o utilizador pode escolher na sua config. Além disso possui ainda várias outras variáveis que contêm dados sobre as vendas e dados lidos. Temos ainda aqui definidas todas as implementações das queries pedidas pelos docentes, tanto as estatísticas como as interativas. Importante perceber que a classe GestVendas implementa tambem ela ainterface IGstVendas.

Dificuldades e Problemas Encontrados

Ao longo do trabalho foram surgindo vários desafios, aos quais tentamos responder da forma mais eficaz possível. Uma das dificuldades iniciais foi perceber a API da swing, a qual ainda demorou um pouco, mas que depois achamos que valeu a pena, e que ficou muito mais fácil de implementar o resto da Vista depois.

A outra dificuldade foi encontrar o equilíbrio da maneira como as coisas ficavam implementadas, uma davam mais jeito para fazer um get, outras era melhores para varrimento. Por vezes queríamos usar um SET, mas nesta API não conseguimos utilizar o método set. Outra que na verdade nem faz muito sentido, era querer organizar uma TreeMap por uma ordem que ordenava pelos values não keys, se por exemplo cada key tivesse um Cliente e cada value uma quantidade que até podíamos ir mexendo, mas no fim quiséssemos tudo ordenado pela quantidade (value) e não pelo código do Cliente (key).

Testes de Performance

STREAMREADER TESTS	MAPS TESTS	SETS TESTS	LIST TESTS
LEITURA BUFFERED READER: 92,910 mseg FILES: 134,227 mseg **LEITURA + PARSING** BUFFERED READER: 549,577 mseg FILES: 462,293 mseg **LEITURA + PARSING + VALIDAÇÃO** BUFFERED READER: 789,392 mseg FILES: 745,399 mseg	**LOAD PRODUTOS** HASH_MAP 89,279 mseg TREE_MAP 70,116 mseg **LOAD CLIENTES** HASH_MAP 52,940 mseg TREE_MAP 14,920 mseg **LOAD VENDAS** HASH_MAP 2015,296 mseg TREE_MAP 2895,665 mseg **QUERY 1** HASH_MAP 150,13349 mseg TREE_MAP 144,75882 mseg **QUERY 2** HASH_MAP 137,23584 mseg TREE_MAP 132,56476 mseg **QUERY 3** HASH_MAP 0,96334 mseg TREE_MAP 0,43320 mseg **QUERY 4** HASH_MAP 24,24059 mseg TREE_MAP 35,59358 mseg **QUERY 5** HASH_MAP 2,47393 mseg TREE_MAP 0,70370 mseg **QUERY 6** HASH_MAP 76,23329 mseg TREE_MAP 87,48775 mseg **QUERY 7** HASH_MAP 154,62139 mseg TREE_MAP 144,96666 mseg **QUERY 8** HASH_MAP 310,04615 mseg TREE_MAP 294,23409 mseg **QUERY 9** HASH_MAP 19,57909 mseg TREE_MAP 41,31767 mseg **QUERY 10** HASH_MAP 239,63688 mseg TREE_MAP 225,82559 mseg **QUERY E1** HASH_MAP 125,70561 mseg TREE_MAP 142,23737 mseg **QUERY E2** HASH_MAP 689,59025 mseg TREE_MAP 747,16797 mseg	**LOAD PRODUTOS** HASH_SET 104,423 mseg LINK_HASH_SET 98,141 ms TREE_SET 183,841 mseg **LOAD CLIENTES** HASH_SET 38,386 mseg LINK_HASH_SET 27,228 ms TREE_SET 45,121 mseg **LOAD VENDAS** HASH_SET 1964,791 mseg LINK_HASH_SET 1735,088 ms TREE_SET 2474,994 mseg **QUERY 1** HASH_SET 143,71810 mseg LINK_HASH_SET 120,0489 ms TREE_SET 119,80436 mseg **QUERY 2** HASH_SET 140,26026 mseg LINK_HASH_SET 125,24712ms TREE_SET 128,49759 mseg **QUERY 3** HASH_SET 0,98497 mseg LINK_HASH_SET 0,28911 ms TREE_SET 0,25434 mseg **QUERY 4** HASH_SET 20,28579 mseg LINK_HASH_SET 17,37393 ms TREE_SET 18,54049 mseg **QUERY 5** HASH_SET 2,50134 mseg LINK_HASH_SET 0,56137 ms TREE_SET 0,57305 mseg **QUERY 6** HASH_SET 411,24747 mseg LINK_HASH_SET 366,1549 ms TREE_SET 383,31593 mseg **QUERY 7** HASH_SET 144,13802 mseg LINK_HASH_SET 137,47082ms TREE_SET 135,10421 mseg **QUERY 8** HASH_SET 326,12256 mseg LINK_HASH_SET 302,93736ms TREE_SET 310,62565 mseg **QUERY 9** HASH_SET 22,75311 mseg LINK_HASH_SET 16,20842 ms TREE_SET 15,04378 mseg **QUERY 10** HASH_SET 249,91672 mseg LINK_HASH_SET 243,6186ms TREE_SET 240,66618 mseg **QUERY E1** HASH_SET 192,80909 mseg LINK_HASH_SET 193,2474ms TREE_SET 198,67851 mseg **QUERY E2** HASH_SET 848,67459 mseg LINK_HASH_SET 805,34775ms TREE_SET 836,88410 mseg	**LOAD PRODUTOS** VECTOR 112,138 mseg ARRAY_LIST 107,891 mseg **LOAD CLIENTES** VECTOR 31,517 mseg ARRAY_LIST 7,688 mseg **LOAD VENDAS** VECTOR 1991,447 mseg ARRAY_LIST 1834,357 mseg **QUERY 1** VECTOR 142,94384 mseg ARRAY_LIST 129,53276 mseg **QUERY 2** VECTOR 139,81937 mseg ARRAY_LIST 126,27354 mseg **QUERY 3** VECTOR 0,98069 mseg ARRAY_LIST 0,32227 mseg QUERY 4 VECTOR 20,91937 mseg ARRAY_LIST 17,72928 mseg **QUERY 5** VECTOR 2,50051 mseg ARRAY_LIST 0,65092 mseg **QUERY 6** VECTOR 78,00427 mseg ARRAY_LIST 73,42573 mseg **QUERY 7** VECTOR 152,22058 mseg ARRAY_LIST 133,52555 mseg **QUERY 8** VECTOR 319,21149 mseg ARRAY_LIST 290,79458 mseg **QUERY 9** VECTOR 15,38930 mseg ARRAY_LIST 14,89084 mseg **QUERY 10** VECTOR 239,01795 mseg ARRAY_LIST 237,55809 mseg **QUERY E1** VECTOR 162,67318 mseg ARRAY_LIST 148,97954 mseg **QUERY E2** VECTOR 804,51136 mseg ARRAY_LIST 788,18209 mseg

***Para caber tudo na mesma página, LINKED_HASH_SET ficará escrito como LINK_HASH_SET

Foram realizados testes de performance, os quais estão apresentados acima, que passam por obter tempos do load de ficheiros e das queries, usando os ficheiros Vendas_1M.txt (1 Milhão de vendas) (neste caso em específico apresentado acima), Vendas_3M.txt (3 Milhões de vendas) e Vendas_5M.txt (5 Milhões de vendas), utilizando a classe Crono. Com estes testes chegamos a alguns resultados interessantes e alguns deles um pouco menos esperados.

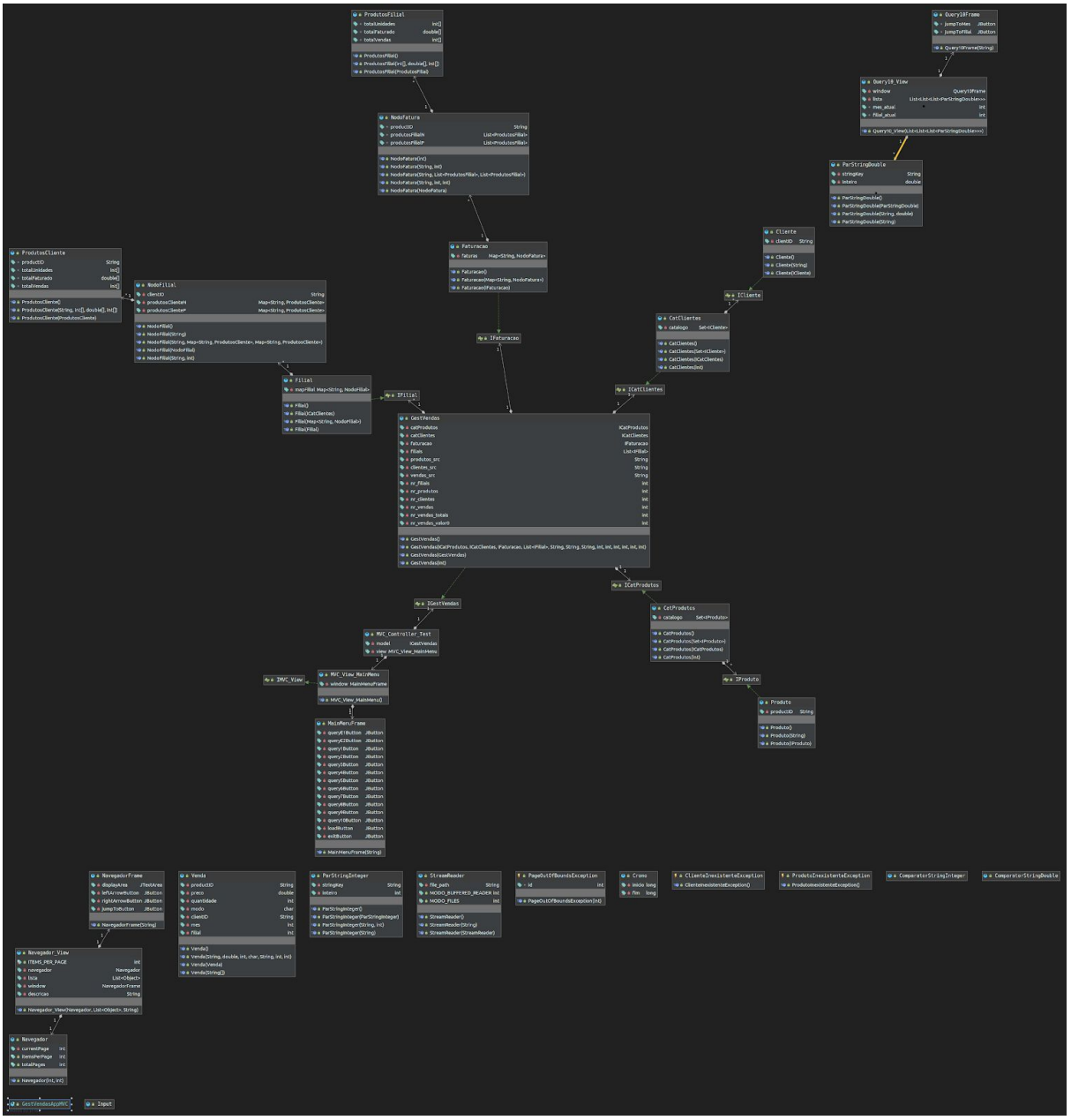
Estes tempos de teste foram registados numa máquina com um processador de 2.2GHz e com uma memória RAM de 16GB.

Estes testes são feitos com repetições de 5 vezes de cada parâmetro a estudar, havendo no entanto sleeps constantes entre as funções, de maneira a evitar principalmente sobreaquecimento do CPU. Assim, um teste pode decorrer até os 4 minutos. Uma vez que se testa tudo 5 vezes, e existem sleeps no meio de cada teste em particular dentro destas 5 repetições.

Apesar de alguns resultados parecem discrepantes, é de relembrar que estamos a observar os números na ordem dos milissegundos, e dessa maneira, qualquer mudança de tempos pareça bastante abrupta, embora não seja sequer perceptível para um ser humano. Além disso, basta um resultado de um teste em articular das 5 repetições ser anomalamente maior, que vai aumentar de imediato a média dos tempos do parâmetro em estudo. Assim, ao decorrer várias vezes o teste, podem também os resultados ser diferentes, dependendo inclusive de todos os processos que estão a decorrer no background, entre outros aspetos.

De qualquer maneira, qualquer pessoa tem a opção de poder realizar os testes na sua máquina, conforme o explicado no README.md no projeto do GitHub!

Diagrama de Classes



O Diagrama de Classes visto em cima foi gerado pelo IntelIJ automaticamente.

Conclusão

Concluído o trabalho, achamos que foram atingidos os objetivos do mesmo e ficamos a reter alguns pontos essenciais. Conseguimos perceber as dificuldades de programar em grande escala, desta vez em Java, e as complicações que isso pode trazer. Conseguimos também perceber que por vezes é difícil conseguir alcançar um balanço que satisfaça e agrade tanto a segurança dos dados (encapsulamento) por parte de terceiros, ou por vezes, até do próprio utilizador, bem como a rapidez de resposta a algumas questões. Não só isso, mas também a relação entre tempo de load/ tempo de respostas às diferentes queries.

Outra parte importante de reter foi a maneira como todas as estruturas de dados são organizadas, e a forma de como esta organização pode por vezes melhorar ou piorar um algoritmo, sendo necessário refazer a estrutura de modo a otimizar cada vez mais a performance, até se chegar a um *sweet spot*. Este ponto leva também à importância da reutilização de código e da generalização do mesmo, uma vez que com um bom código que respeitasse essas condições conseguiria-se reformular o trabalho depois de alterar as estruturas, apenas mudando alguns pontos chave em algumas funções de forma bastante simples. Se isto foi importante no projeto de C, no projeto de JAVA teve o dobro da importância, acrescida também pelos paradigmas da POO. Além disso para testes tornava tudo mais fácil, podíamos alterar uma estrutura com TreeMaps para HashMaps em apenas alguns segundos. No entanto achamos estes pormenores muito mais fáceis de realizar em Java do que em C, também pela quantidade de API já predefinidas neste tipo de linguagem.

Por fim, é importante perceber que o tempo de resposta neste projeto foi geralmente mais alto neste projeto em JAVA do que em C, se utilizássemos o tipo de estruturas mais próximo e idêntico possível entre as duas linguagens. No entanto, em termos de queries, e além destas serem diferentes do projeto anterior, a diferença passava muitas vezes pelos milissegundos, não sendo notável a olho nú. No entanto, a diferença de tempo revela-se mais pertinente no load e carregamento das estruturas de dados a partir dos ficheiros.