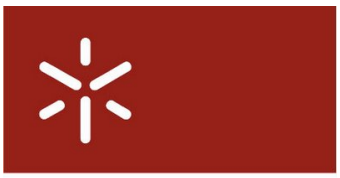


Universidade do Minho

Relatório - Laboratórios de Informática III

MIEI - 2º ano - 2º semestre
Universidade do Minho

Gestão das Vendas de uma Cadeia de Distribuição - Java

Grupo 33

Dinis Peixoto
A75353

Ricardo Pereira
A74185

Marcelo Lima
A75210

16 de Novembro de 2017

Conteúdo

1. *Introdução*

Este é o segundo projecto solicitado pelos docentes da unidade curricular *Labotatórios de Informática III*, o objectivo principal do projecto é o desenvolvimento de um programa com a capaciddade de gerir as vendas de uma cadeira de distribuição com vários filiais, tal como no projecto anterior. Porém, ao contrário do primeiro que foi desenvolvido com os conhecimentos da unidade curricular *Programação Imperativa*, neste foram aplicados os conhecidos da UC *Programação Orientada a Objectos* onde é lecionada a linguagem de programação *Java* e as suas utilidades. O principal desafio do projecto seria a programação em larga escala, uma vez que se trata de uma simulação de uma cadeira por onde passam milhões de dados.

2. *Descrição das Classes*

A arquitetura da aplicação é desenvolvida conforme quatro classes principais: *Catálogo de Clientes*, *Catálogo de Produtos*, *Facturação global* e Vendas por *Filial*. Sendo que estas são utilizadas nas classes *Hipermercado* e *HipermercadoApp*, responsáveis pela estrutura de armazenamento de todo o conjunto de dados e pela estrutura da aplicação apresentada ao utilizador, respectivamente.
Adjacente a estas classes existe também uma responsável pela leitura dos ficheiros como input para aplicação: *Clientes.txt*, *Produtos.txt* e *Vendas_1M*, *Vendas_3M* ou *Vendas_5M*, conforme desejado, entre outras classes úteis à realização do projeto, tais como *Comparators*, *Exceptions* e classes essenciais à realização das diferentes queries (exemplo: *ParStringDouble*).
Todas estas classes estão diretamente relacionados com uma interface onde o utilizador final pode desfrutar de todas as funcionalidades que a aplicação implementa.

Ficheiros

Clientes.txt:

Ficheiro com o código de 20.000 clientes. O código de cada cliente é representado por uma letra maiúscula seguida de quatros digitos que representam um

número entre 1000 e 5000.

Produtos.txt:

Ficheiro com o código de 200.000 produtos. O código de cada produto é representado por duas letras maiúsculas seguidas de quatros digitos que representam um número entre 1000 e 1999.

Vendas_1M.txt:

Ficheiro com o código de 1.000.000 de vendas. O código de cada venda inclui: um código de produto, um preço (entre 0 e 999.99), uma quantidade (entre 1 e 200), uma letra indentificando o tipo de compra (Normal ou Promoção), um código de cliente, o mês da compra e por fim o filial onde ocorreu a venda.

Vendas_3M.txt:

Ficheiro com o código de 3.000.000 de vendas. Estrutura idêntica ao ficheiro anterior.

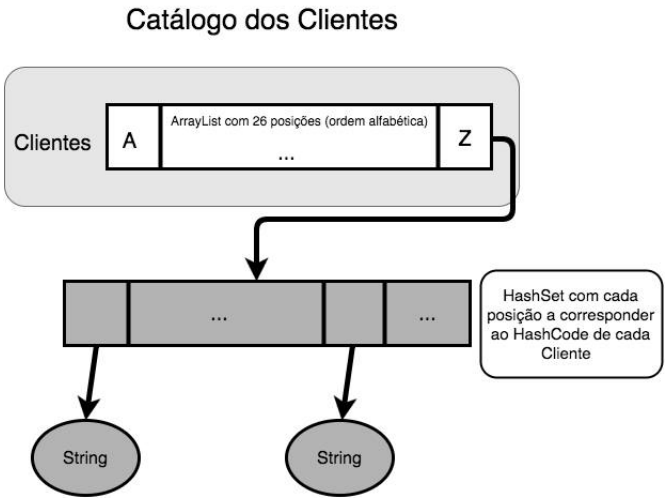
Vendas_5M.txt:

Ficheiro com o código de 5.000.000 de vendas. Estrutura idêntica aos ficheiros anteriores.

Todos estes ficheiros foram utilizados como input no projecto anterior da UC, neste projecto foi adicionado um novo tipo de input: *ficheiros binários*. Estes podem ser carregados para a aplicação ao invés de usar os ficheiros de texto anteriormente mencionados, fornecem, no entanto, um carregamento muito mais demorado que o anterior. Podem também, se necessário, ser utilizados para guardar o estado da aplicação, para ser mais tarde carregado novamente.

2.1 Catálogo de Clientes

2.1.1 Estrutura



O Catálogo de Clientes é onde são armazenados todos os clientes que senencontram no ficheiro Clientes.txt. Como se trata de uma larga quantia de clientes,

necessitámos de os guardar numa estrutura adequada:

List<Set<String>> catalogo

Correspondendo a um ArrayList de 26 posições, onde cada posição corresponde a um HashSet(no qual se encontra todos os clientes cujo código começa por uma determinada letra).

2.1.2 Atributos

- private List<Set<String>> catalogo;

2.1.3 Funções

- **public CatClientes()**

  Construtor vazio do Católogo de Clientes.

- **public CatClientes(Collection <String>collection)**

  Construtor por parâmetros do Catálogo de Clientes.

- **public CatClientes(CatClientes catalogoClientes)**

  Construtor por cópia do Catálogo de Clientes.

- **public List<Set<String>> getCatalogo()**

  Função que retorna o Catálogo de Clientes.

- **public List<String> getClientes()**

  Função que retorna a lista de todos os Clientes.

- **public List<String> getClientesLetra(char letra)**

  Função que retorna uma lista de Clientes começados por uma determinada letra.

- **private static int calculaIndice(char letra)**

  Função que calcula o índice dado a primeira letra de um cliente.

- **public int totalClientes()**

  Função que retorna o total de clientes presentes no Catálogo.

- **public int totalClientesLetra(char letra)**

  Função que retorna o total de clientes de uma determinada letra presentes no Catálogo.

- **public void insereCliente(String cliente)**

  Função que insere um cliente ao catálogo.

- **public void removeCliente(String cliente)**

  Função que remove um cliente do catálogo.

- **public boolean existeCliente(String cliente)**

  Função que determina a existência ou não de um cliente no catálogo.

- **public boolean equals(Object obj)**

  Função que testa a igualdade.

- **public CatClientes clone()**

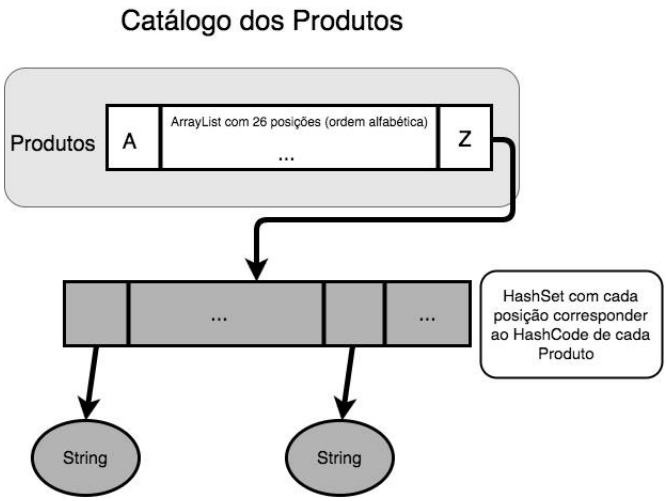
  Função que faz clone do Catálogo de Clientes.

- **public void limpar()**

  Função que limpa o catálogo.

2.2 Catálogo de Produtos

2.2.1 Estrutura



O Catálogo de Produtos é onde são armazenados todos os produtos que se encontram no ficheiro Produtos.txt. Tal como no caso anterior, como se trata de uma larga quantia de produtos, necessitámos de os guardar numa estrutura adequada:

List<Set<String>> catalogo

Correspondendo a um ArrayList de 26 posições, onde cada posição corresponde a um HashSet(no qual se encontra todos os produtos cujo código começa por uma determinada letra).

2.2.2 Atributos

- private List<Set<String>> catalogo;

2.2.3 Funções

- **public CatProdutos()**

  Construtor vazio do Catálogo de Produtos.

- **public CatProdutos(Collection <String>collection)**

  Construtor por parâmetros do Catálogo de Produtos.

- **public CatProdutos(CatProdutos catalogoProdutos)**

  Construtor por cópia do Catálogo de Produtos.

- **public List<Set<String>> getCatalogo()**

  Função que retorna o Catálogo de Produtos.

- **public List<String> getProdutos()**

  Função que retorna a lista de todos os Produtos.

- **public List<String> getProdutosLetra(char letra)**

  Função que retorna uma lista de Produtos começados por uma determinada letra.

- **private static int calculaIndice(char letra)**

  Função que calcula o índice dado a primeira letra de um produto.

- **public int totalProdutos()**

  Função que retorna o total de produtos presentes no Catálogo.

- **public int totalProdutosLetra(char letra)**

  Função que retorna o total de produtos de uma determinada letra presentes no Catálogo.

- **public void insereProduto(String produto)**

  Função que insere um produto ao catálogo.

- **public void removeProduto(String produto)**

  Função que remove um produto do catálogo.

- **public boolean existeProduto(String produto)**

  Função que determina a existência ou não de um produto no catálogo.

- **public boolean equals(Object obj)**

  Função que testa a igualdade.

- **public CatProdutos clone()**

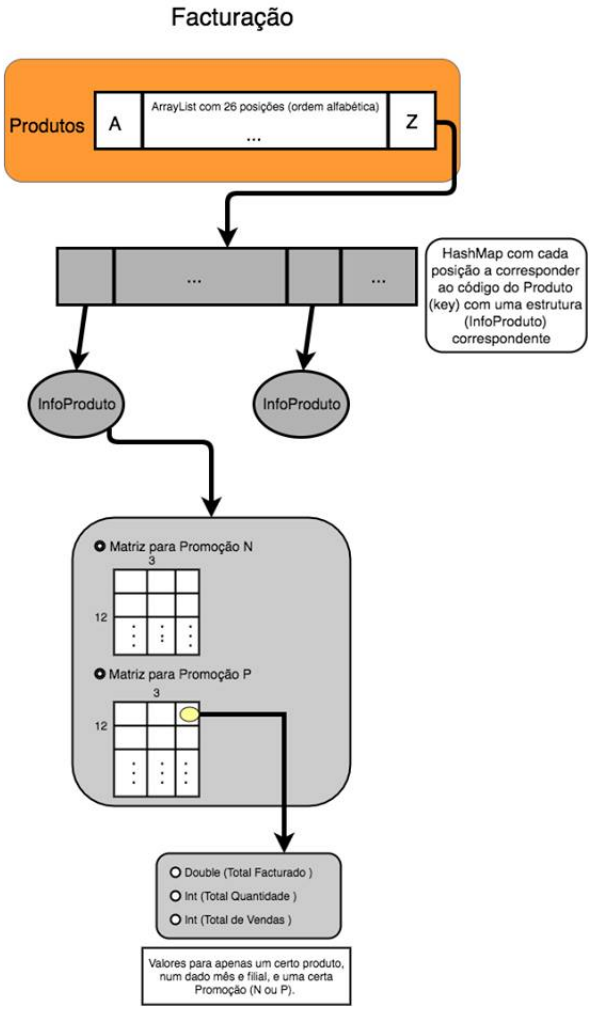
  Função que faz clone do Catálogo de Produtos.

- **public void limpar()**

  Função que limpa o catálogo.

2.3 Facturação

2.3.1 Estrutura



A Facturação apresenta uma estrutura mais complexa comparada aos catálogos apresentados anteriormente. A sua estrutura inicial, possui um ArrayList.

List<Map<String,InfoProduto>> produtos

Este ArrayList, possui a mesma estrutura que o Catálogo de Produtos, ou seja, estruturado por 26 posições, indexando desta forma pelas letras iniciais dos produtos, para uma melhor repartição dos mesmos. Para além disso, em cada posição existe um HashMap, em vez do HashSet como nos catálogos, para podermos associar a um produto (key) uma outra estrutura (value). Essa estrutura possui toda a informação do Produto.

PrecoQuantidade N[][]

PrecoQuantidade P[][]

Estruturado com duas matrizes, de dimensões 12 x 3 (No meses x No Filiais), sendo uma matriz para promoção (P) e a outra matriz para sem promoção (N).

double totalprice

int totalVendas
int totalQuant

Cada posição da matriz, possui uma outra estrutura com o total facturado, a quantidade total, e total de vendas. Estes valores são referentes a um dado produto, mês, filial e promoção.

2.3.2 Atributos

Facturação

- private List<Map<String,InfoProduto>> produtos;

InfoProduto

- private PrecoQuantidade N[][];
- private PrecoQuantidade P[][];

PrecoQuantidade

- private double totalprice;
- private int totalVendas;
- private int totalQuant;

2.3.3 Funções

- **public Facturacao()**
  Construtor vazio da Facturação.
- **public Facturacao(Facturacao f)**
  Construtor por cópia da Facturação.
- **public void insereVenda(Venda v)**
  Insere uma venda na Facturação.
- **public boolean existeProd(String prod)**
  Verifica se um Produto existe na Facturação.
- **public void limpar()**
  Apaga a lista dos produtos da Facturação.
- **public List<Map<String,InfoProduto>> getFactProdutos()**
  Retorna um clone da Facturação.
- **private static int calculaIndice(char letra)**
  Calcula o índice da lista correspondente a uma determinada letra.

- **public int nProdDif()**

  Calcula o número de produtos diferentes comprados.

- **public double total()**

  Calcula o total Facturado.

- **public int zeros()**

  Calcula o total de compras de valor igual a 0.

- **public List<ParIntDouble> getNumVendasFactTotal (List<ParIntDouble>lista,String prod)**

  Calcula para cada mês, o número de vendas e o total facturado com um dado produto.

- **public void getProdMaisVendidos(Set<ParStringInt> prods)**

  Calcula para cada Produto, o número de unidades compradas.

- **public int totalProdutos()**

  Calcula o número de produtos diferentes.

- **public int totalVendas(int mes)**

  Calcula o número de vendas num determinado mês.

- **public Facturacao clone()**

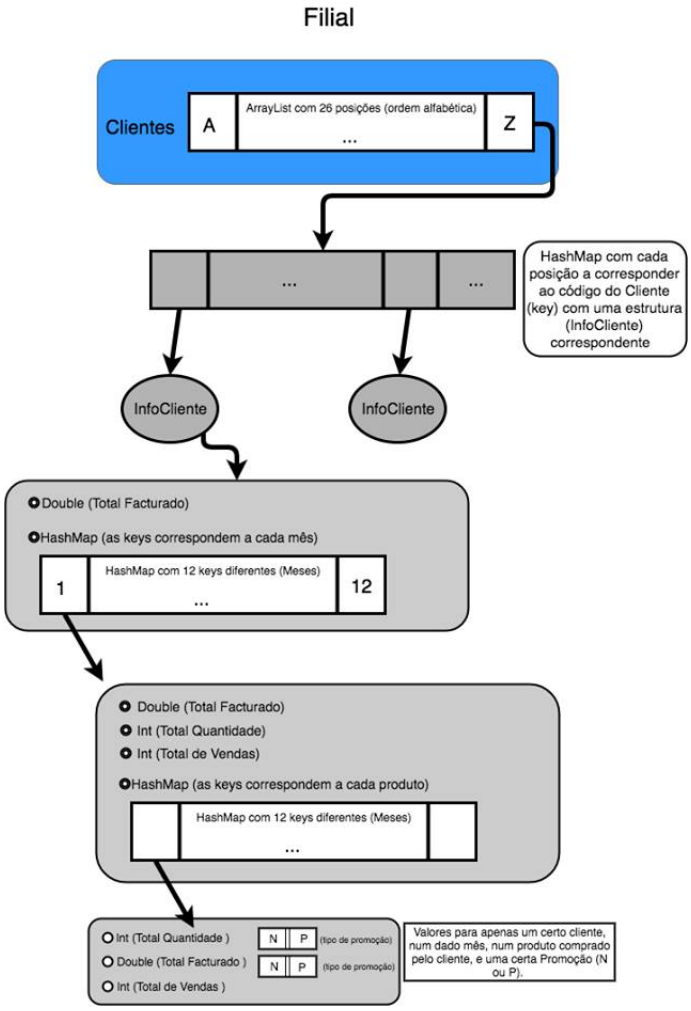
  Função que faz clone da Facturação.

- **public boolean equals(Object obj)**

  Função que testa a igualdade.

2.4 Filial

2.4.1 Estrutura



A Filial apresenta uma estrutura apenas para uma das várias Filiais. Esta apresenta a estrutura mais complexa comparado às apresentadas anteriormente.

List<Map<String,InfoCliente>> clientes

A estrutura inicial apresenta-se com um ArrayList, com 26 posições, e indexam para a letra inicial, representando os Clientes.

No array os clientes estão separados pela letra inicial, em cada posição do array. Com isto, em cada posição podemos encontrar um HashMap para podermos associar a um cliente (key) uma outra estrutura (value).

Map<Integer,InfoMes> meses

double totFact

A estrutura possui, um double que representa o total facturado por um cliente, e um HashMap indexado por meses, ou seja, para um mes (key) temos uma outra estrutura associada (value), podendo ter assim a informação separada por meses, para um cliente.

Map<String,InfoClienteProduto> produtos
int quantidade
int numVendas
double totGasto

A estrutura que representa um mes de um cliente, possui um inteiro para a quantidade comprada, outro inteiro para o número de vendas, e um double para o total gasto pelo cliente. Para além disso, possui uma estrutura HashMap que possui todos os produtos que o cliente comprou. No HashMao a key (código do produto) temos ainda mais uma esturtura associada (value).

int quantity[]
double totGasto[]
int numVendas

Por fim, a última esturutra, possui informações muito restritas. Apenas tem a informação correspondente a um certo cliente e produto (comprado pelo cliente). Esta estrutura contém dois array's de duas posições e um inteiro. Um array possui as quantidades e outro possui os preços, correspondentes ao total facturado. As posições representam se os valores provém de uma promoção (posição 1) ou sem promoção (posição 0). O inteiro representa o número de vendas, ou seja, o número de vezes que o cliente comprou aquele produto.

2.4.2 Atributos

Filial

- private List<Map<String,InfoCliente>> clientes;

InfoCliente

- private Map<Integer,InfoMes> meses;
- private double totFact;

InfoMes

- private Map<String,InfoClienteProduto> produtos;
- private int quantidade;
- private int numVendas;
- private double totGasto;

InfoClienteProduto

- private int quantity[];
- private double totGasto[];
- private int numVendas;

2.4.3 Funções

- **public Filial()**

  Construtor vazio de uma Filial.

- **public Filial(Filial f)**

  Construtor por cópia de uma Filial.

- **public int getNumVendMes(Set<String> clientes,int mes)**

  Calcula o número de vendas num determinado mês, e guarda os clientes que as fizeram.

- **public ParIntDouble getNumCompTotMes(Set<String> prods,int mes,String cli)**

  Calcula o número de compras e o total gasto por um determinado cliente, num determinado mês e guarda os respectivos produtos comprados.

- **public void getNumCompTotMesProd(List<Set<String>> cli, String prod,int mes)**

  Guarda numa estrutura, para cada mês, os clientes que compraram determinado produto.

- **public void getProdsMaisComp(String cli,Map<String,Object> prods)**

  Guarda numa estrutura os produtos e as repectivas quantidades que um determinado cliente comprou.

- **public void getProdsMaisVend(Map<String,ParIntSet> prodscli)**

  Guarda numa estrutura os produtos e as respectivas quantidades compradas, assim como os clientes que os compraram.

- **public List<ParStringDouble>getCompTot()**

  Guarda num estrutura cada cliente com a respectiva facturação total.

- **public void getCliMaisCompDif(Map<String,Set<String>> cliprods)**

  Guarda numa estrutura cada cliente com os respectivos produtos comprados.

- **public void getCliMaisCompProd(Map<String,ParIntDouble> cliquant, String prod)**

  Guarda numa estrutura cada cliente com a respectiva quantidade e facturação de um determinado produto.

- **public void insereVenda(Venda v)**

  Insere uma determinada venda na estrutura.

- **public void limpar()**

  Apaga a estrutura da Filial.

- **public List<Map<String,InfoCliente>> getFilialClientes()**
  Retorna a estrutura da Filial.
- **public void total(Set<String> lista)**
  Guarda numa estrutura todos os clientes que fizeram compras na Filial.
- **public void totalMes(double lista[])**
  Guarda numa estrutura a factoração de cada mês.
- **public void getCliMes(Set<String>clie, int mes)**
  Guarda numa estrutura os clientes que compraram num determinado mês.
- **private static int calculaIndice(char letra)**
  Calcula o índice correspondente a uma determinada letra.
- **public Filial clone()**
  Função para fazer clone.
- **public boolean equals(Object obj)**
  Função que testa a igualdade.

2.5 Hipermercado

Esta é a estrutura onde todas as anteriores são aglomeradas, podendo mesmo dizer-se a estrutura principal de todo o projecto.

2.5.1 Atributos

- private static final int num_filiais = 3;
- private CatClientes catalogoClientes;
- private CatProdutos catalogoProdutos;
- private Facturacao facturacao;
- private Filial filiais[];
- private DadosEstatisticos dados;

2.5.2 Funções

- **public Hipermercado()**
  Construtor vazio.
- **public Hipermercado(Hipermercado h)**
  Construtor por cópia.

- **public CatClientes getCatClientes()**

  Função que retorna o Catálogo de Clientes.

- **public CatProdutos getCatProdutos()**

  Função que retorna o Catálogo de Produtos.

- **public Facturacao getFacturacao()**

  Função que retorna a facturação.

- **public List<Filial> getFilial()**

  Função que retorna as filiais.

- **public boolean isEmpty()**

  Função que testa se o Hipermercado se encontra vazio.

- **public DadosEstatisticos carregaDados(String ficheiro_clientes, String ficheiro_produtos, String ficheiro_vendas)**

  Função que faz a leitura dos ficheiros, carregando a informação para o Hipermercado.

- **public void limpar()**

  Função para limpar o Hipermercado.

- **public void gravaObj(String fich) throws IOException**

  Gravar o estado da classe Hipermercado num determinado ficheiro.

- **public static Hipermercado leObj(String fich) throws IOException, ClassNotFoundException**

  Iniciar a classe Hipermercado com o estado guardado num determinado ficheiro.

- **public Set<String> getProdsNaoComp()**

  Função que retorna uma lista ordenada alfabeticamente com os códigos dos produtos não comprados.

- **public ParIntInt getNumVendNumCliMes(int mes) throws MesInvalidoException**

  Função que dado um mês válido, determina o número total de vendas realizadas e o número total de clientes distintos que as fizeram.

- **public List <TriploIntIntDouble> getNumCompNumProdTot(String cli) throws ClienteInexistenteException**

  Função que dado um código de cliente determina, para cada mês, quantas compras este fez, quantos produtos distintos comprou e quando gastou no total.

- **public List <TriploIntIntDouble> getNumCompNumCliTot(String prod) throws ProdutoInexistenteException**

  Função que dado o código de um produto existente determina, mês a mês quantas vezes foi comprado, por quantos clientes diferentes e o total facturado.

- **public TreeSet <ParStringInt> getProdsMaisComprados(String cli) throws ClienteInexistenteException**

  Função que dado o código de um cliente determina a lista de códigos de produtos que mais comprou, ordenada por ordem decrescente de quantidade e, para quantidades iguais por ordem alfabética.

- **public TreeSet <TriploStringIntInt> getProdsMaisVend(int x)**

  Determina o conjunto dos N produtos mais vendido em todo o ano, indicando o número total de clientes distintos que o compraram.

- **public List<List<ParStringDouble>> getMaioresComp()**

  Determina para cada filial, a lista dos três maiores compradores em termos de dinheiro facturado.

- **public TreeSet<ParStringInt> getCliMaisCompDif(int x)**

  Determina os códigos dos N clientes que compraram mais produtos diferentes.

- **public TreeSet<TriploStringIntDouble> getCliMaisCompProd(String prod, int x) throws ProdutoInexistenteException**

  Dado o código de um produto, determinar o conjunto dos N clientes que mais o compraram, e para cada um indicar o valor gasto.

- **public Hipermercado clone()**

  Função que faz um clone.

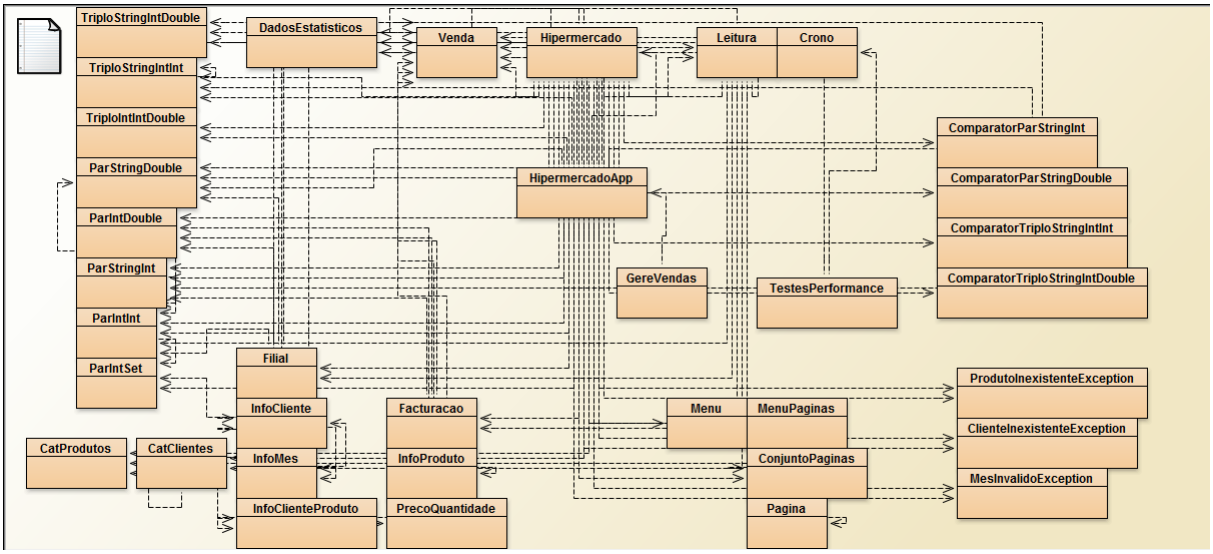
- **public void getFactTotalMes(double lista[],int filial)**

  Guarda numa estrutura a factoração de cada mês numa determinada filial.

- **public void getCliMes(Set<String> cli,int mes,int filial)**

  Guarda numa estrutura os clientes que compraram num determinado mês, numa determinada filial.

2.6 Diagrama de Classes



3. *Interface para o utilizador*

3.1 Menus disponíveis

3.1.1 Menu Principal

Quando o utilizador executa o nosso programa, a primeira coisa que lhe é apresentada é o *Menu Principal*, onde estão presentes quatro opções que o utilizador pode escolher.
O utilizador não pode, no entanto, aceder a qualquer uma destas sem antes utilizar o comando *1* responsável pelo carregamento de dados, conseguindo com isto armazenar dados nas estruturas para posteriormente utilizar as diferentes funcionalidades do nosso programa.

```
****************************** Menu Principal ******************************
   1 - Carregar dados
   2 - Consultas Estatísticas
   3 - Menu
   4 - Gravar dados
                                                          0 - Sair
********************************************************************************
Opção:
```

- **1 - Carregar Dados**

  Fazer o carregamento dos dados.

- **2 - Consultas Estatísticas**

  Apresentar, se possível, as estatísticas dos dados lidos.

- **3 - Menu**

  Apresentar, se possível, o Menu de todas as funcionalidades (queries) do programa.

- **4 - Gravar Dados**

  Gravar os dados.

3.1.2 Carregar Dados

Para efetuar o carregamento de dados para o programa, são apresentadas ao utilizador três opções, sendo que duas destas efetuam a leitura através de ficheiros de texto com informação e uma outra efetua a leitura através de um ficheiro binário.

```
****************************** Menu Leitura ******************************
   1 - Ler ficheiros default
   2 - Escolher ficheiros a ler
   3 - Carregar dados já existentes
                                                        0 - Anterior
**************************************************************************
Opção:
```

- **1 - Ler ficheiros default**

  Faz o carregamento de dados através da leitura dos ficheiros de texto default (*Clientes.txt*, *Produtos.txt* e *Vendas_1M.txt*).

- **2 - Escolher ficheiros a ler**

  O utilizador pode escolher os ficheiros de texto por onde deseja fazer o carregamento de dados.

- **3 - Carregar dados já existentes**

  Fazer o carregamento de dados através de um ficheiro binário com informação relativa à classe Hipermercado no mesmo.

3.1.3 Consultas Estatísticas

Após qualquer uma destas leituras ser realizada com sucesso o utilizado pode então desfrutar de todas as outras funcionalidades que o programa lhe proporciona. Sendo que a primeira são as consultas estatísticas da informação que acabou de carregar para a aplicação.

```
****************************** Menu Consultas ******************************
   1 - Número total de compras por mês
   2 - Facturação por mês
   3 - Número de clientes que compraram em cada mês
                                                        0 - Anterior
**************************************************************************
Opção:
```

- **1 - Número total de compras por mês**

  Apresenta o número total de compras efetuadas em cada mês.

- **2 - Facturação por mês**

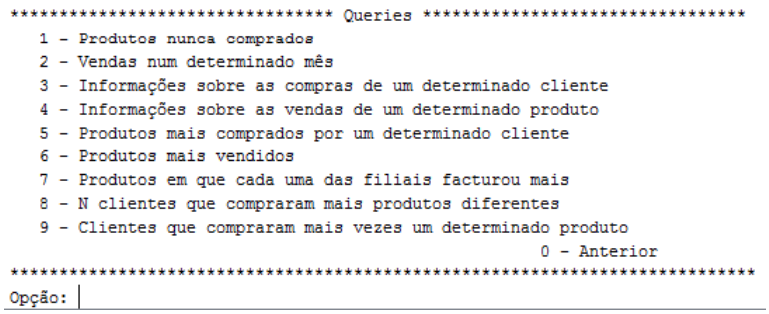
  Apresenta o total facturado em cada um dos filiais, assim como o total, por mês.

- **3 - Número de clientes que compraram em cada mês**

  Apresenta o número de clientes diferentes que compraram em cada mês.

3.1.4 Menu

Pode também aceder ao *Menu* onde se encontram grande parte das funcionalidades da aplicação.



- **1 - Produtos nunca comprados**

  Apresenta uma lista de produtos que nunca foram comprados.

- **2 - Vendas num determinado mês**

  Apresenta o número de vendas realizadas assim como o número de clientes diferentes que as fizeram de um determinado mês.

- **3 - Informações sobre as compras de um determinado cliente**

  Apresenta a tabela com informações sobre as compras de um determinado cliente, dentro destas incluem-se o número de compras, o número de produtos e o total gasto.

- **4 - Informações sobre as vendas de um determinado produto**

  Apresenta a tabela com informações sobre as vendas de um determinado produto por mês, dentro das quais, o número de vendas, o número de clientes e o total facturado.

- **5 - Produtos mais comprados por um determinado cliente**

  Apresenta uma lista dos produtos mais comprados e a sua quantidade, de um determinado cliente.

- **6 - Produtos mais vendidos**

  Apresenta uma lista com os N produtos mais vendidos, a quantidade vendida e o número de clientes que comprou cada um destes.

- **7 - Produtos em que cada uma das filiais facturou mais**

  Apresenta uma tabela com os clientes que gastaram mais dinheiro em cada uma das filiais, assim como a quantidade que gastaram.

- **8 - N clientes que compraram mais produtos diferentes**

  Apresenta a lista dos N clientes que compraram mais produtos diferentes, assim como a quantidade de produtos diferentes que compraram.

- **9 - Clientes que compraram mais vezes um determinado produto**

  Apresenta uma lista dos clientes que mais vezes compraram um dado produto, assim como a quantidade e o valor gasto.

3.1.5 Gravar Dados

Após o utilizador ter desfrutado da nossa aplicação pode, por fim, gravar os dados num ficheiro binário para os poder ler novamente mais tarde.
Exemplo da execução deste comando:

```
***************************** Menu Principal ******************************
   1 - Carregar dados
   2 - Consultas Estatísticas
   3 - Menu
   4 - Gravar dados
                                                        0 - Sair
*****************************************************************************
Opção: 4
Nome do ficheiro: hipermercado.dat
A gravar... Este processo pode ser demorado!
Concluído!
Pressione ENTER para continuar!
```

3.2 Mais sobre a interface

A nossa interface recebe grande parte das vezes números como comandos, é exemplo disso o comando *0* que funciona sempre como um comando de Anterior/Sair. Existem no entanto exeções a esta regra em algumas das queries presentes na interface, quando pedem um Cliente/Produto em específico, por exemplo.
Tendo em conta o facto de que os comandos presentes na interface são, salvo raras excepções, números, decidimos que seria confortável para o utilizador ter uma apresentação por páginas, com a opção do utilizador escolher especificamente uma página do catálogo que lhe é apresentado, em vez da utilização de outro tipo de comandos que fizessem avançar/recuar apenas uma página.

```
****************************** Página 1 de 47 ******************************
                                    AA1022
                                    AA1143
                                    AA1236
                                    AB1432
                                    AB1643
                                    AC1285
                                    AC1365
                                    AC1874
                                    AD1283
                                    AD1865
                                    AE1156
                                    AE1280
                                    AF1094
                                    AF1620
                                    AG1028
                                    AG1065
                                    AI1080
                                    AI1123
                                    AI1782
                                    AJ1051
                                                          0 - Sair
********************************************************************************
Página:
```

Houve ainda outras queries em que a informação era demasiada, e para conforto do utilizador esta informação teve de ser bem organizada com a utilização de tabelas.

```
************************* Maiores compradores em cada filial ***************************

|  Filial  |                                Compradores                                    |
|__________|_______________________________________________________________________|
|    1     | T4720 -> 2353597,900000 | F4737 -> 2107203,670000 | O1488 -> 2077077,180000 |
|    2     | Z1848 -> 2173068,690000 | Y1687 -> 2122078,940000 | R2459 -> 2089734,620000 |
|    3     | R2722 -> 2184829,850000 | K3556 -> 2132755,500000 | U3268 -> 2129578,620000 |

********************************************************************************************
Tempo demorado: 0.632342736 segundos.
Pressione ENTER para continuar!
```

Toda esta interface para o utilizador, apesar de simples, foi feita por nós de maneira a que o utilizador podesse tirar o maior proveito da mesma, com comandos simples e fáceis de memorizar. Foi também nossa preocupação fazer com que a interface fosse fácil para o utilizador se acostumar, com pouco tempo de utilização.

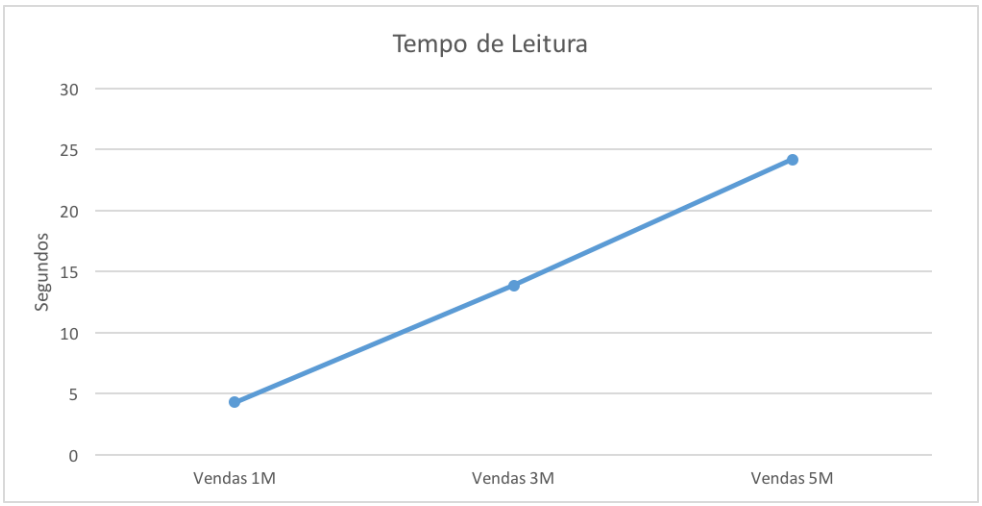
4. *Testes de Performance*

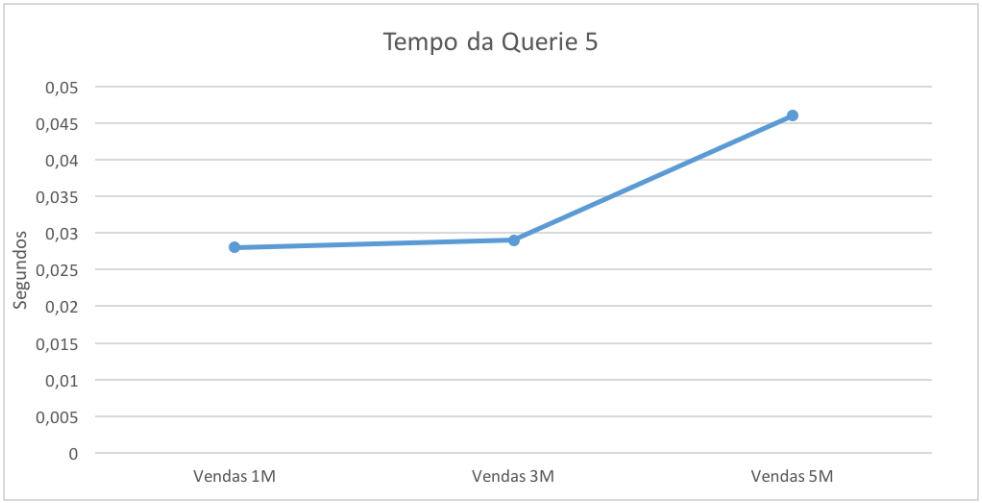
4.1 Leitura e Queries

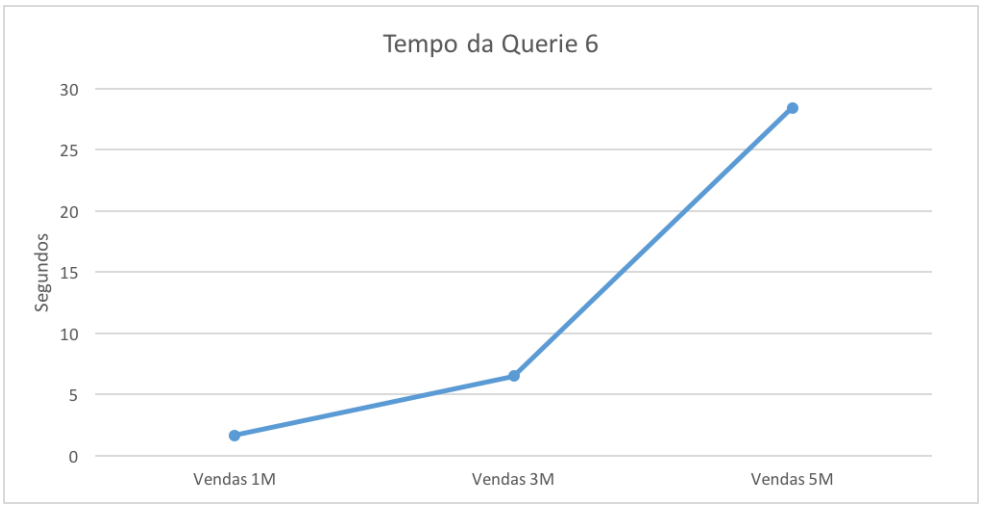
Os gráficos apresentados a seguir, são resultados de testes de performance com os ficheiros Vendas_1M.txt, Vendas_3M.txt e Vendas_5M.txt, num computador com as seguintes características:

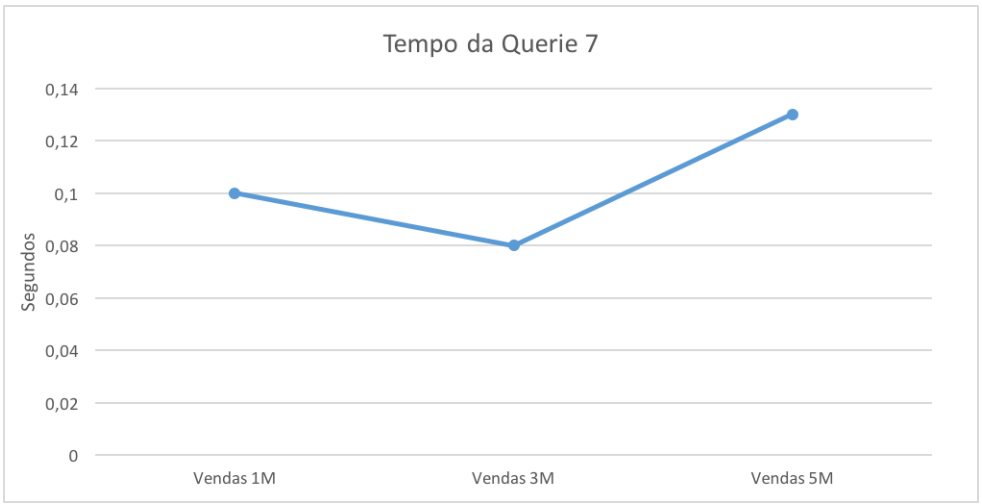
MacBook Air (13-inch, Early 2014)	
Processador	1,4 GHz Intel Core i5
Memória	4 GB 1600 MHz DDR3
Placa gráfica	Intel HD Graphics 5000 1536 MB

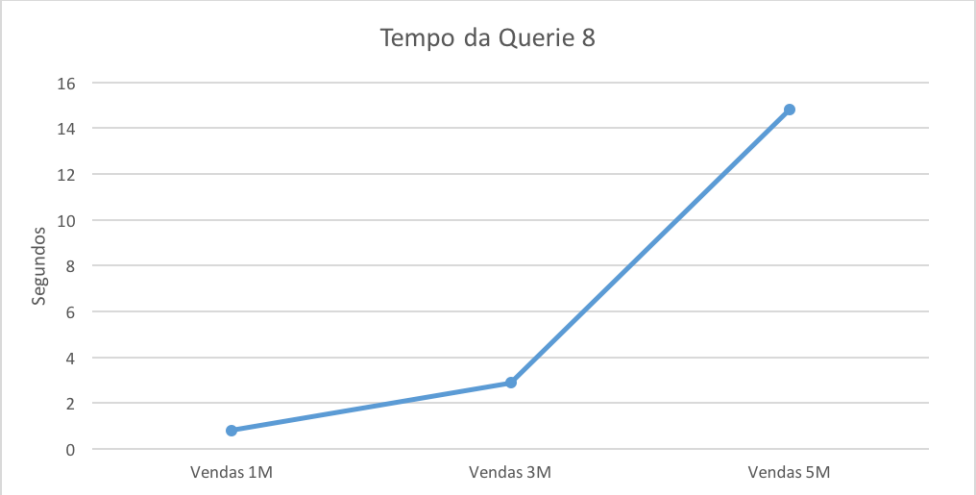
Os testes realizados são dirigidos à leitura dos ficheiros, e também à execução das queries 5,6,7,8 e 9, para os diferentes ficheiros.

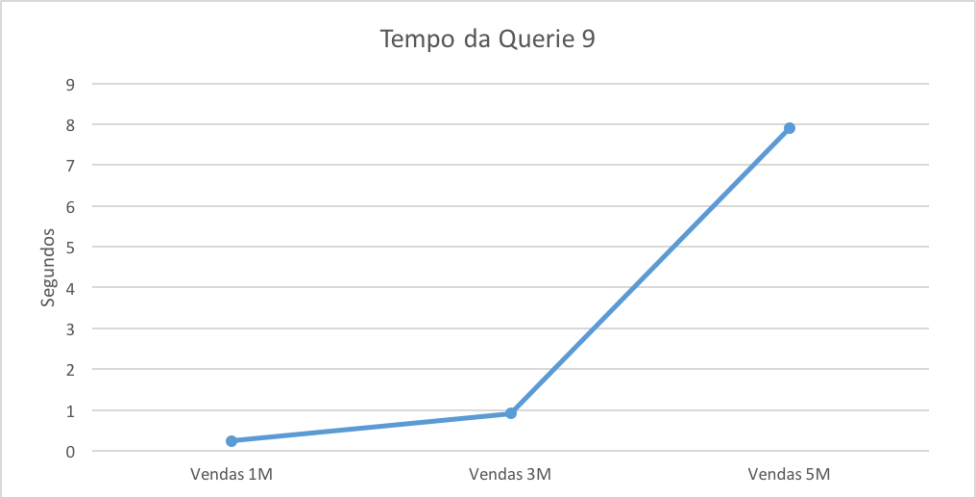



Tempo da Querie 5
Segundos
0,05
0,045
0,04
0,035
0,03
0,025
0,02
0,015
0,01
0,005
0
Vendas 1M
Vendas 3M
Vendas 5M


Tempo da Querie 6
Segundos
30
25
20
15
10
5
0
Vendas 1M
Vendas 3M
Vendas 5M

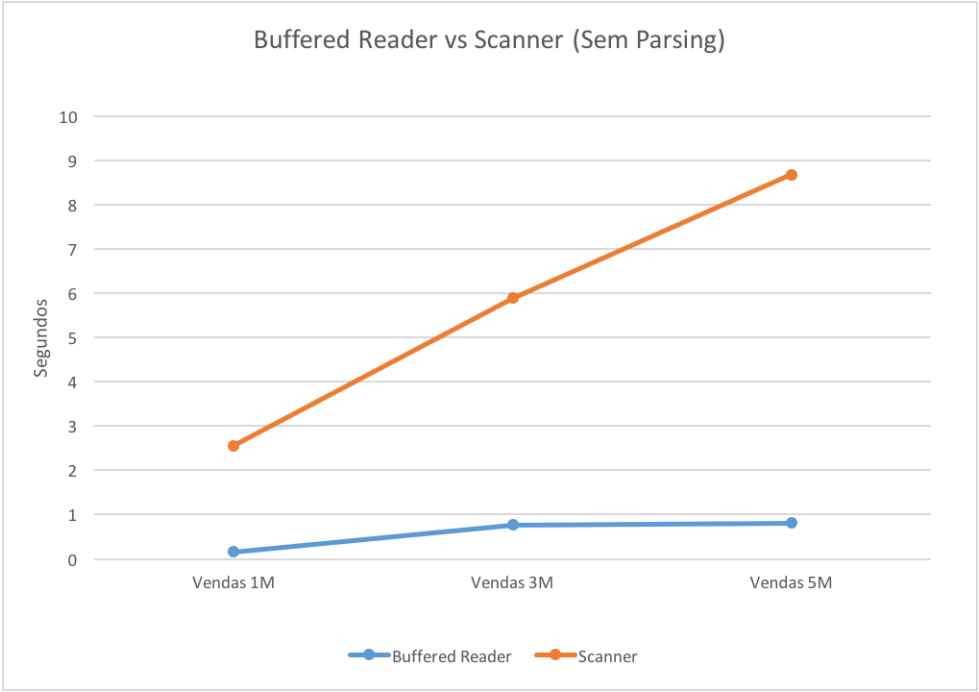

Tempo da Querie 7
Segundos
0,14
0,12
0,1
0,08
0,06
0,04
0,02
0
Vendas 1M
Vendas 3M
Vendas 5M



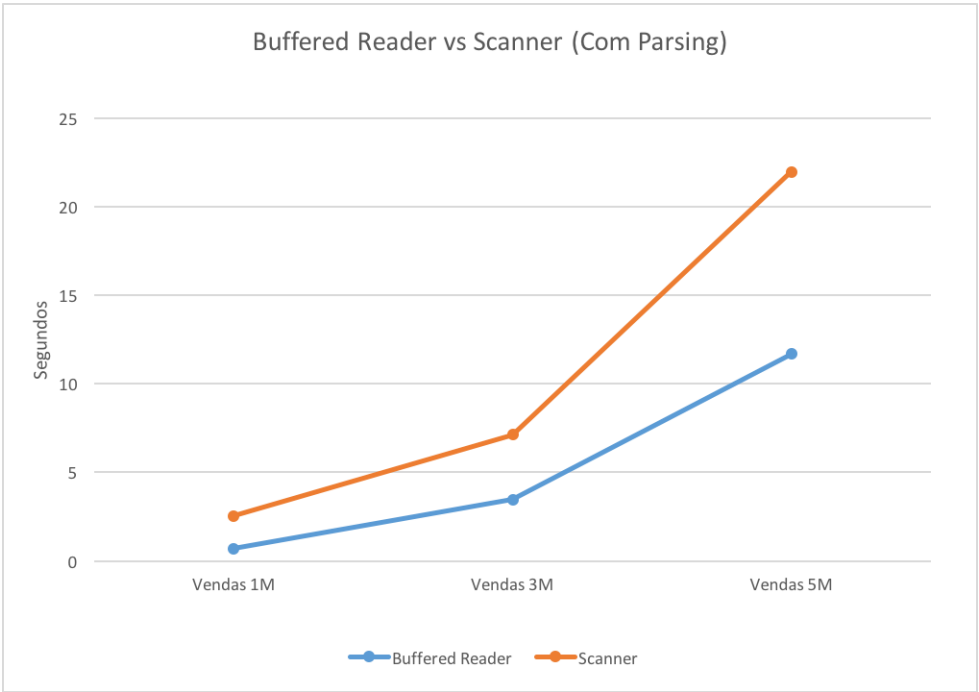


4.2 Comparações

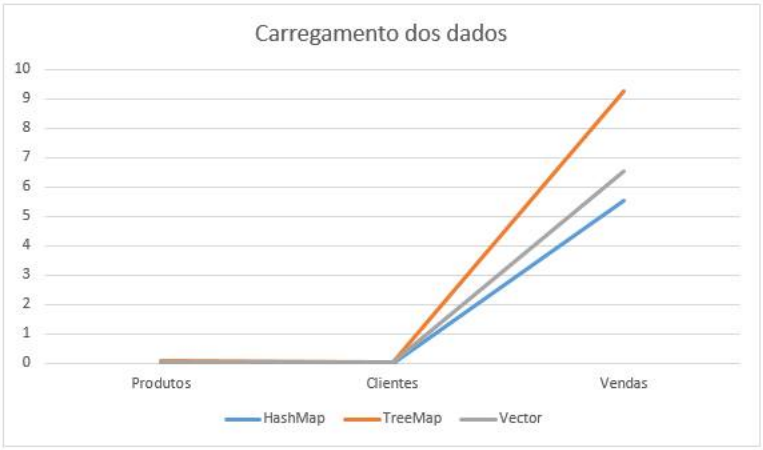
	Vendas 1M	Vendas 3M	Vendas 5M
Buffered Reader	0,16	0,77	0,81
Scanner	2,55	5,89	8,67



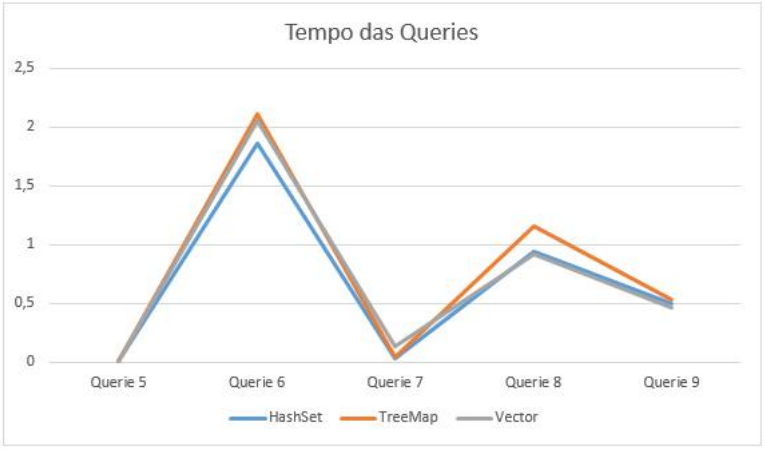
	Vendas 1M	Vendas 3M	Vendas 5M
Buffered Reader	0,68	3,46	11,68
Scanner	2,54	7,11	21,95



	Produtos	Clientes	Vendas
HashMap	0,043907674	0,00992576	5,540801615
TreeMap	0,090439446	0,016109552	9,247916567
Vector	0,039769135	0,015041291	6,557837319



Queries	5	6	7	8	9
HashMap	0,005157471	1,866999004	0,027679069	0,941234065	0,502714826
TreeMap	0,006084444	2,109992101	0,039466576	1,155119456	0,537842524
Vector	0,007770896	2,050869828	0,13862376	0,920504424	0,466871575



.

5. *Conclusão*

Após a realização do projeto em C, foi-nos dito pelo próprio professor docente que o projecto em Java seria muito mais acessível.
Realmente, verificamos, durante a sua realização, que este projeto, tanto em termos de tempo necessário como em termos de dificuldade, se mostrou bastante inferior relativamente ao anterior.
Posto isto, uma das grandes diferenças que observamos foi na elaboração das estruturas que armazenam os dados. Apesar de, no geral, estas estruturas sejam semelhantes em ambos os projetos, a sua construção foi bastante divergente. Ao contrário do projeto anterior, onde foi necessário construir de raiz as próprias estruturas, como por exemplo *AVLs*, neste projeto tínhamos ao nosso dispor inúmeras *APIs* de estruturas tais como *Maps* e *Sets* que nos pouparam bastante tempo e tornaram este processo de implementação bastante mais acessível.
Em relação aos Maps, estas foram sem dúvida as estruturas que mais vantagem nos trouxe neste projeto pois permitiram-nos, por exemplo, associar a cada cliente os respetivos produtos comprados de forma bastante eficiente. A utilização de árvores juntamente com *Comparators* fez com que pudéssemos implementar vários métodos de comparação sem que tivéssemos de alterar muito código, o que não era possível no projeto anterior.
Por fim, com a realização deste projeto, pudemos aplicar os conhecimentos obtidos na UC Programação Orientada a Objectos bem como ganhar mais experiência no que toca a este tipo de programação.
Através do trabalho em equipa e de um bom espírito crítico pensamos ter desenvolvido um bom projeto e obtido o resultado esperado.