



Universidade do Minho
Mestrado Integrado em Engenharia Informática

Laboratórios de Informática III

Projeto em Java

Relatório

Sistema de Gestão de Vendas

2019/2020

Grupo 29



João Nuno Costa Neves
A81366



Gonçalo Pinto Nogueira
A86617



Carlos Manuel Marques Afonso
A82529

Índice

Conteúdo

1. Introdução	3
2. Arquitetura da Aplicação	4
2.1 Model.....	5
2.2 Controller	12
2.3 View	13
3. Testes de desempenho	13
4. Interpretação dos resultados e dificuldades encontradas.....	17
5. Conclusão.....	18

1. Introdução

Este projeto foi nos solicitado pelos docentes da unidade curricular de Laboratórios de Informática III, e tem como principal objetivo a realização de um programa que faça a gestão de vendas de uma cadeia de distribuição com três filiais. No entanto, desta vez, divergindo do primeiro projeto, o paradigma de programação é diferente, uma vez que este projeto foi implementado em Java.

Trata-se então de um problema de contexto real, que reflete a gestão de grandes volumes de dados, pelo que se deve procurar implementar soluções otimizadas.

Por fim, de forma a analisar a eficiência deste projeto, foram efetuadas análises à performance.

2. Arquitetura da Aplicação

Neste projeto são 3 as componentes mais relevantes da aplicação, os módulos de dados os quais designados por “Model”, o módulo de controlo designado por “Controller” e finalmente o módulo de apresentação designado por “View”.

Diagrama de Classes

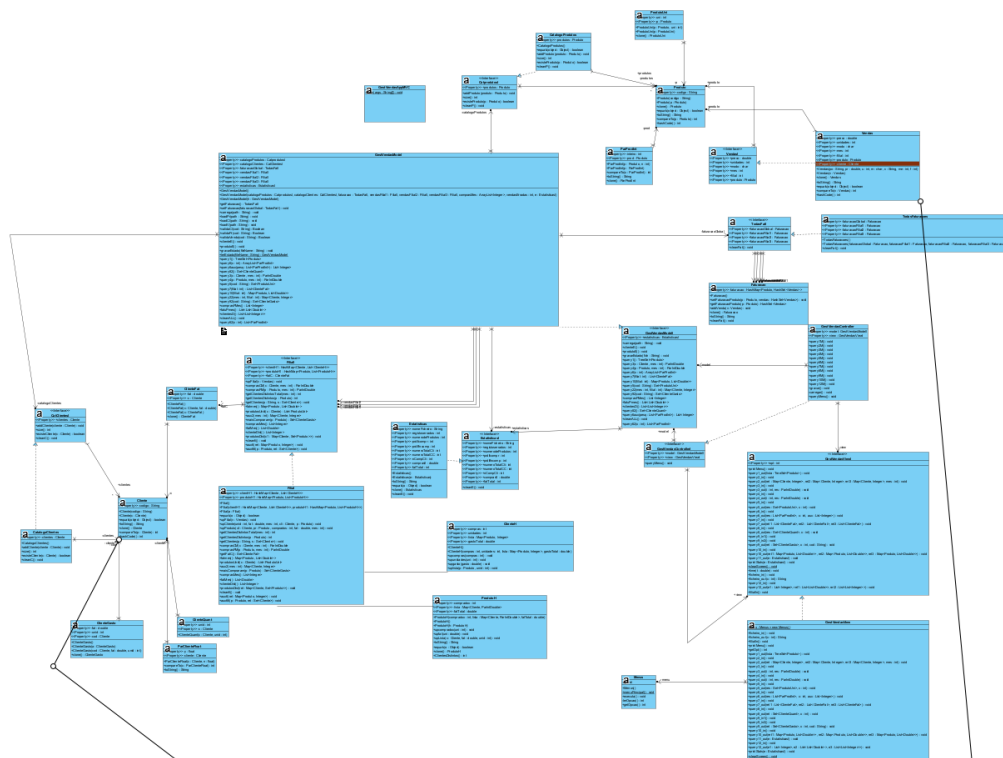


Figura 1 Diagrama de Classes

2.1 Model

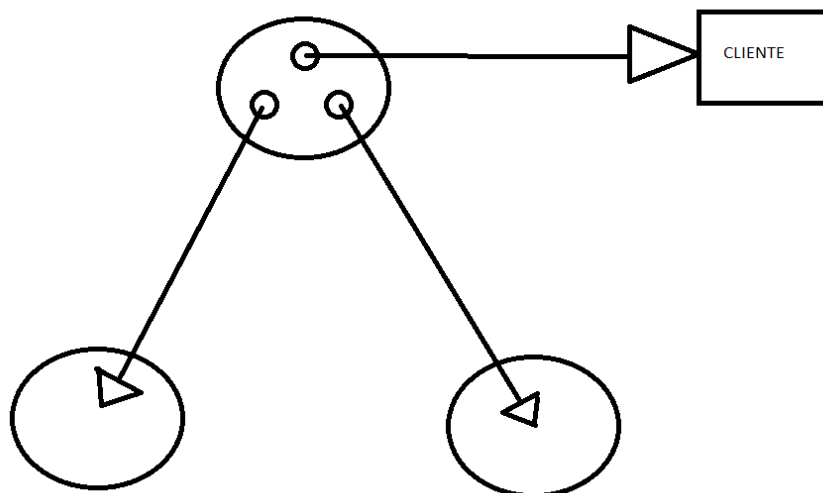
Esta parte da aplicação é a mais importante pois aborda as estruturas utilizadas para guardar as informações lidas bem com os métodos que devolvem as respostas relativas às queries interativas, os métodos para guardar e carregar estados de ficheiros .dat e por fim métodos que devolvem estatísticas sobre os ficheiros lidos.

2.1.1 Catálogo de Clientes

Esta é a classe responsável pelo armazenamento dos clientes.

A classe é definida assim pelo atributo:

```
TreeSet<Cliente> clientes;
```



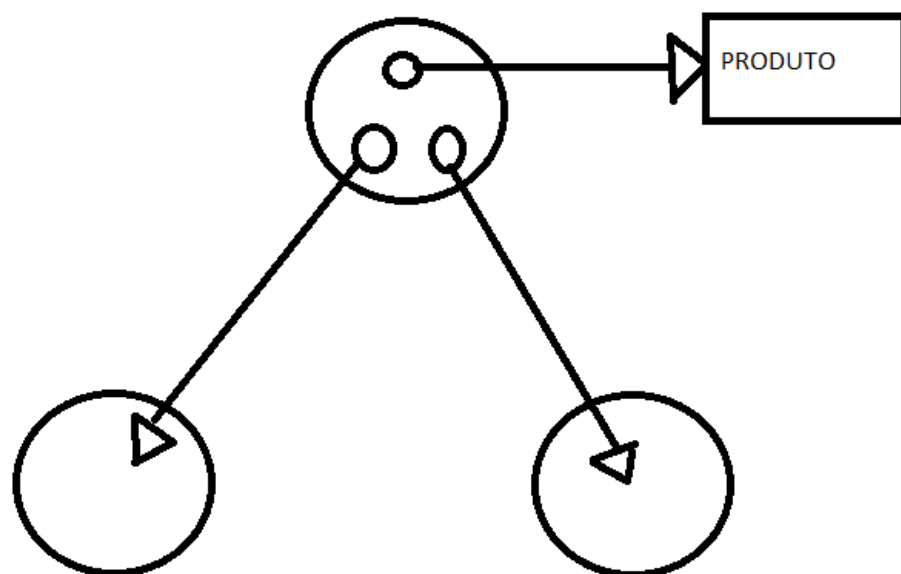
Escolhemos guardar desta forma os clientes para os poder ordenar.

2.1.2 Catálogo de Produtos

Esta é a classe responsável pelo armazenamento dos produtos.

A classe é definida pelo seguinte atributo:

```
TreeSet<Produto> produtos;
```



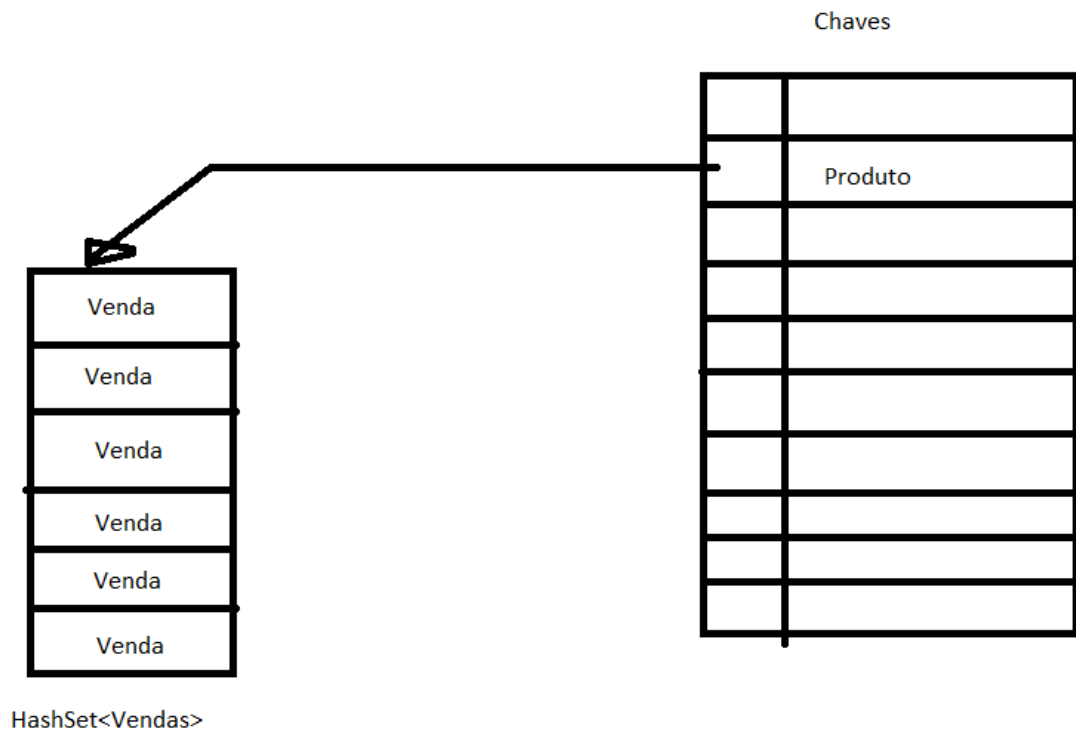
Tal como nos clientes guardamos os produtos usando um TreeSet para se conseguir ordenar.

2.1.3 Faturação

Esta é a classe responsável pelo armazenamento dos dados das vendas. A cada produto é associado o conjunto de vendas desse artigo.

A classe Faturacao é definida pelo seguinte atributo:

```
HashMap<Produto,HashSet<Venda>> faturacao;
```

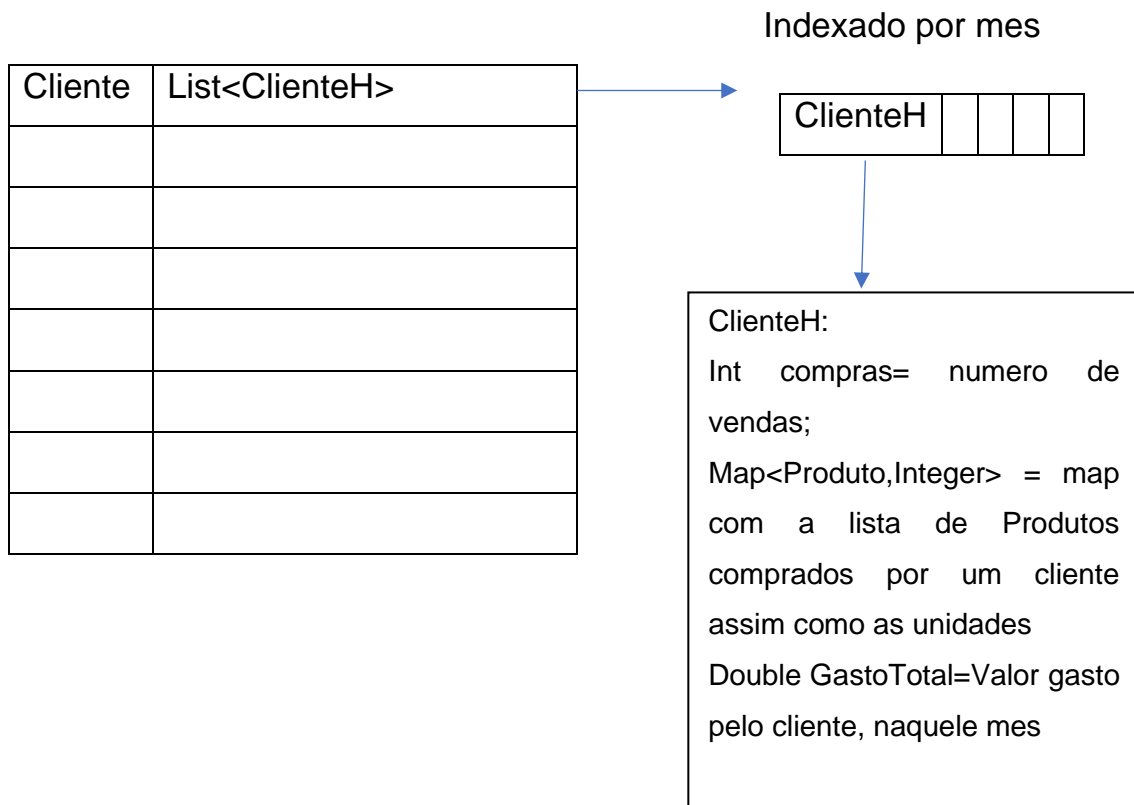


2.1.4 Filial

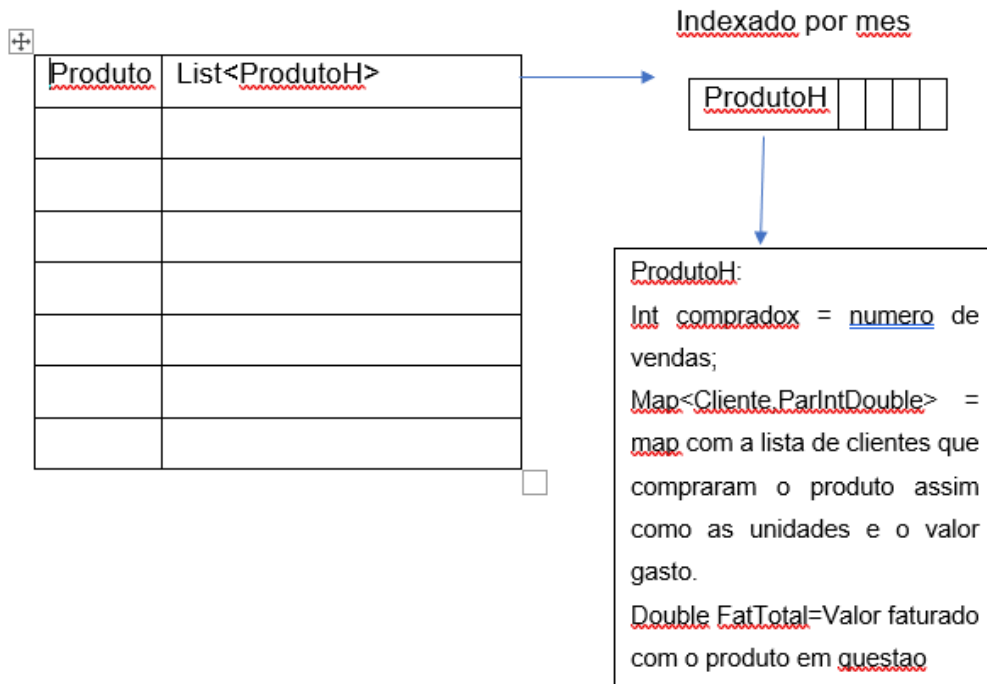
Esta classe é responsável pelo armazenamento dos dados de cada Filial. Cada produto é associado a um conjunto de dados divididos por mês, e cada Cliente é também associado a um conjunto de dados divididos por mês, de forma a facilitar a resposta eficiente às queries.

Por isso a Classe Filial é definida pelos seguintes atributos :

```
HashMap<Cliente<List<ClienteH>>> clienteh1;
```



HashMap<Produto<List<ProdutoH>>produtoh1;



2.1.5 Estatísticas

As estatísticas são responsáveis por apresentar ao utilizador os dados referentes ao último ficheiro de vendas lido, sendo que esta foram divididas em dois grupos estatísticos. O primeiro diz respeito aos dados adquiridos assim que é efetuada a leitura do ficheiro de vendas, posteriormente, o segundo é adquirido através da análise das estruturas explicitadas anteriormente.

1 Grupo

```
//nome do ficheiro
    private String nomeFicheiro;
//numero total de registos de venda errados
    private int registoserrados;
//numero total de produtos
    private int numerodeProdutos;
//numero total de diferentes produtos comprados
    private int prdifcompr;
//numero total de produtos diferentes nao comprados
    private int prdiffncomp;
//numero total de clientes
    private int numeroTotalCli;
//numero total de clientes que realizaram compras
    private int numeroTotalCC;
//numero total de clientes que nao compraram nada
    private int nCompCli;
//numero total de compras de valor igual a 0.0
    private double compras0;
//faturacao total
    private int fatTotal;
```

2 Grupo

O 2 grupo tem por base 3 metodos, sendo que cada um da resposta a uma alínea presente no enunciado deste trabalho, ou seja, o método :

```
public List<Integer> comprasPMes()
```

devolve uma lista com o total de compras efetuadas por mês.

O método :

```
public List<List<Double>> fatuPmes()
```

devolve o total faturado por mês, dividido nas três filiais.. sendo que posteriormente para apresentação ao utilizador, é também apresentado o total global, que é a soma das 3 filiais.

Por fim , o método:

```
public List<List<Integer>> clientesD()
```

Devolve o numero de clientes distintos que fizeram compras, mês a mês, filial a filial.

2.1.6 GestVendasModel

A classe é definida pelos seguintes atributos:

CatprodutosI catalogoProdutos;

CatClientesI catalogoClientes;

TodasFatI faturacao;

FilialI vendasFilial1;

FilialI vendasFilial2;

FilialI vendasFilial3;

EstatisticasI estatisticas;

De salientar que o tipo de cada variável de instância é um interface o que permite apenas utilizar as funções lá definidas sobre essas variáveis.

Esta é a classe responsável pela leitura dos dados do ficheiro Vendas, implementação dos métodos que calculam os resultados pedidos pelas queries bem como carregar e gravar estado.

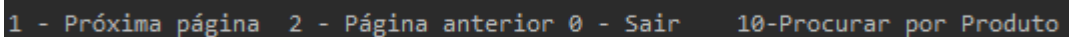
2.2 Controller

Esta componente da aplicação serve como motor de todo o projeto e coordena tudo o que o mesmo faz. É, por isso, o responsável por toda a interligação de todas as componentes do trabalho conseguindo comunicar com as outras duas grandes partes do trabalho (Model e View), realizando pedidos efetuados pelo utilizador e enviando a informação do resultado desse pedido á view .

2.3 View

Esta componente é responsável pela apresentação de informações ao utilizador, sendo comandada pelo Controller que lhe transmite o que é para mostrar ao utilizador.

Entre as várias informações que este modulo pode apresentar estão os diferentes tipos de menus existentes na aplicação bem como tabelas para a apresentação de resultados de forma eficiente e objetiva e por fim estão também implementados navegadores de listas de resultados de forma a ser apenas apresentados alguns valores de cada vez para uma melhor observação de resultados por parte do utilizador, sendo que dada a vasta quantidade de resultados na query¹⁰, esta também apresenta uma pesquisa específica de um determinado produto, de forma a facilitar a análise de resultados.



1 - Próxima página 2 - Página anterior 0 - Sair 10-Procurar por Produto

3. Testes de desempenho

“Specs” da máquina onde foram realizados os testes à performance:

Processador: Intel® Core™ i7-7700HQ CPU @ 2.80GHz

Gráfica: Nvidia GeForce GTX 1060 GDDR5 @ 6,0 GB (192-bit)

RAM: 16GB RAM

Disco: 612GB HD

Todos os valores apresentados são médias obtidas após cinco tentativas, e os resultados obtidos relativos às queries foram aplicados usando o ficheiro de 1 milhão.

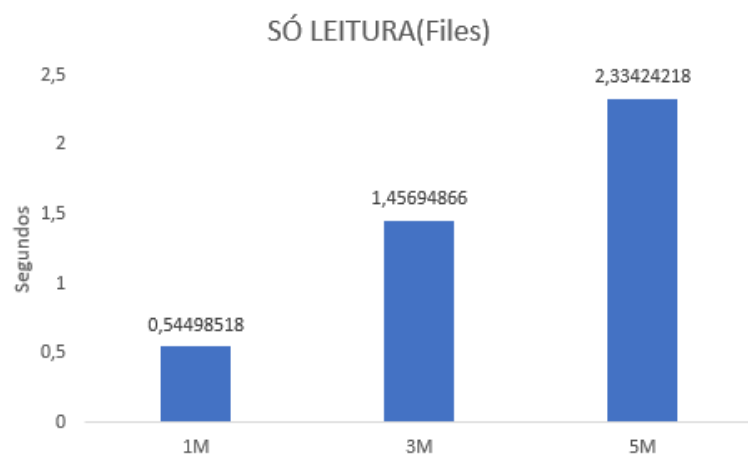
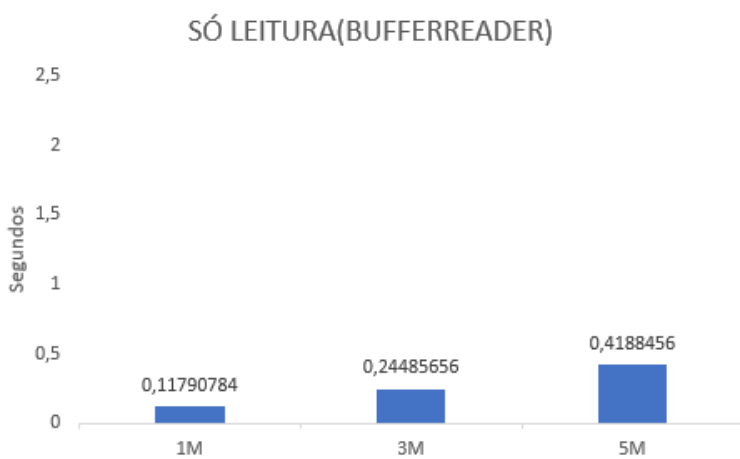


Figura 2-Comparação de Leitura

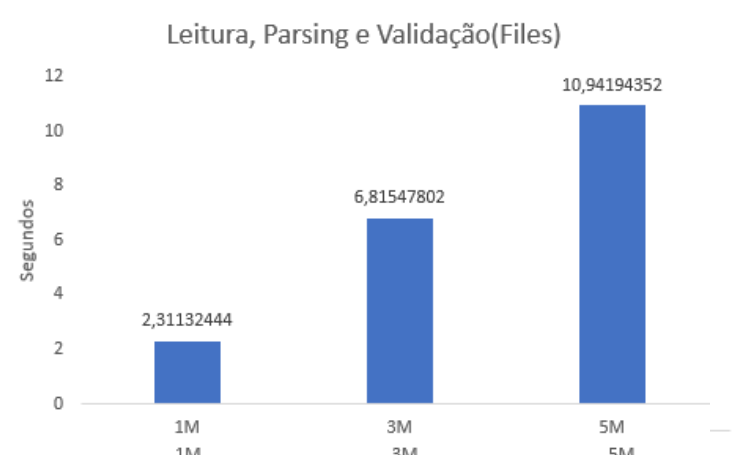
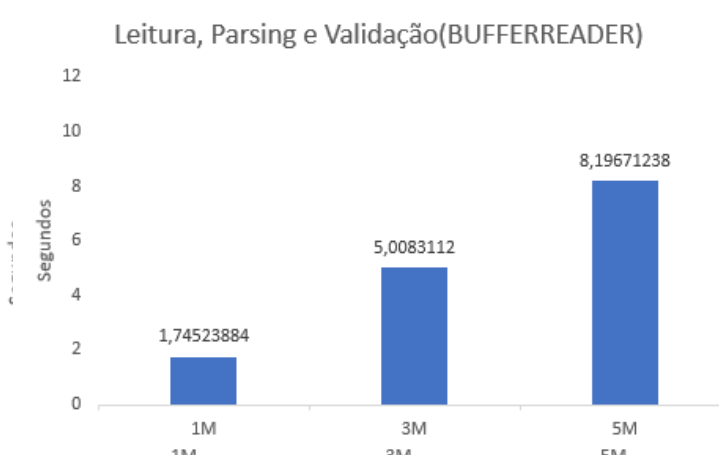


Figura 3-Comparação de tempos Leitura, Parsing e Validação

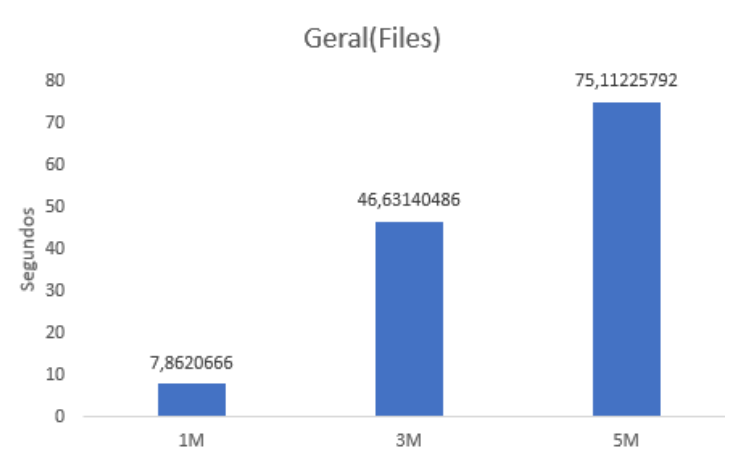
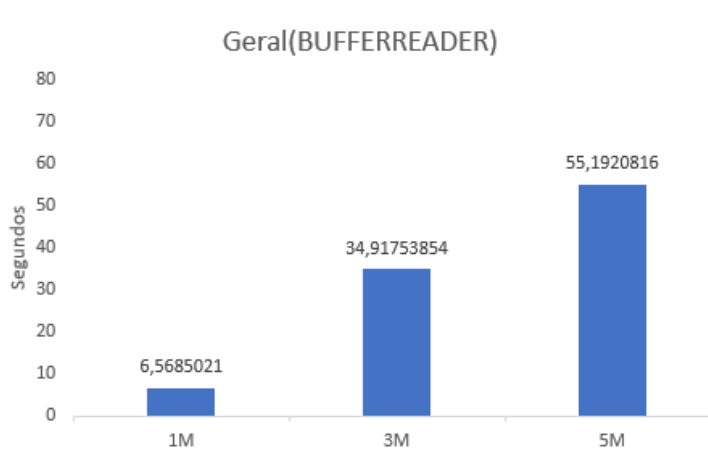


Figura 4 - Comparação de carregamento total de um ficheiro

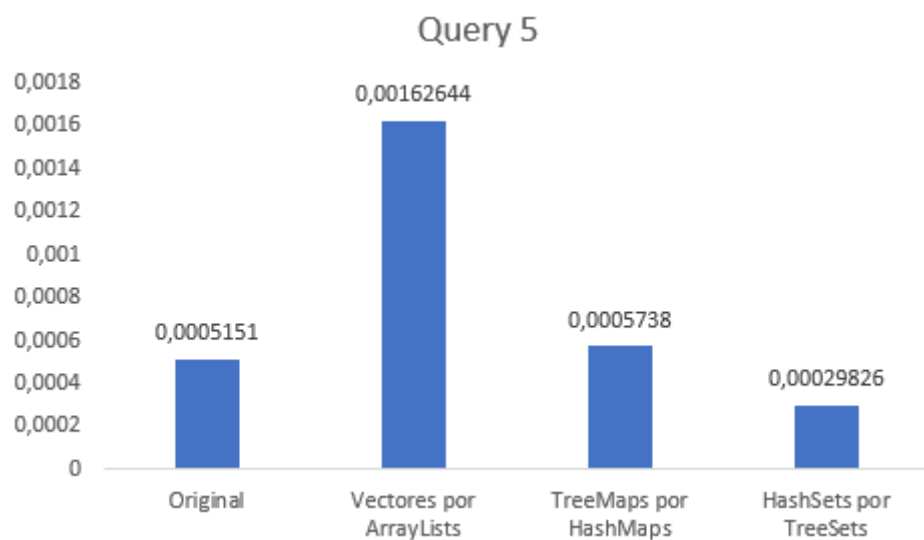


Figura 5-Comparações com diferentes Estruturas Query 5

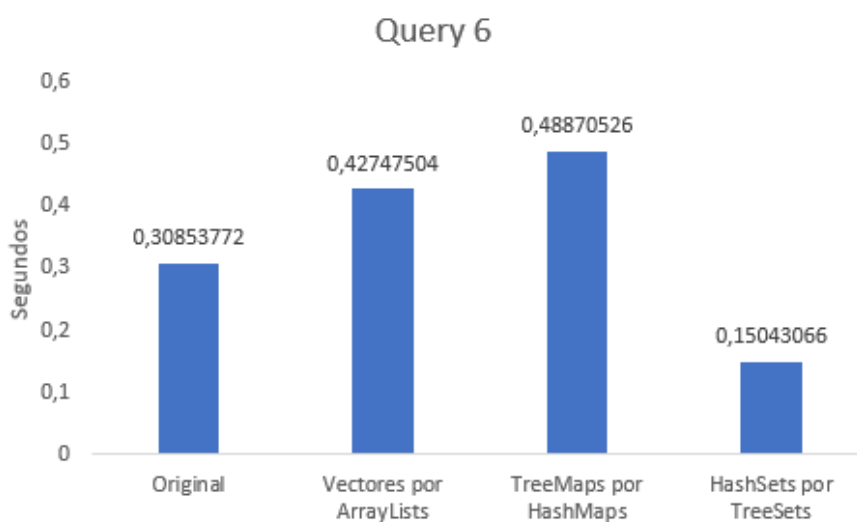


Figura 6- Comparações com diferentes Estruturas Query 6

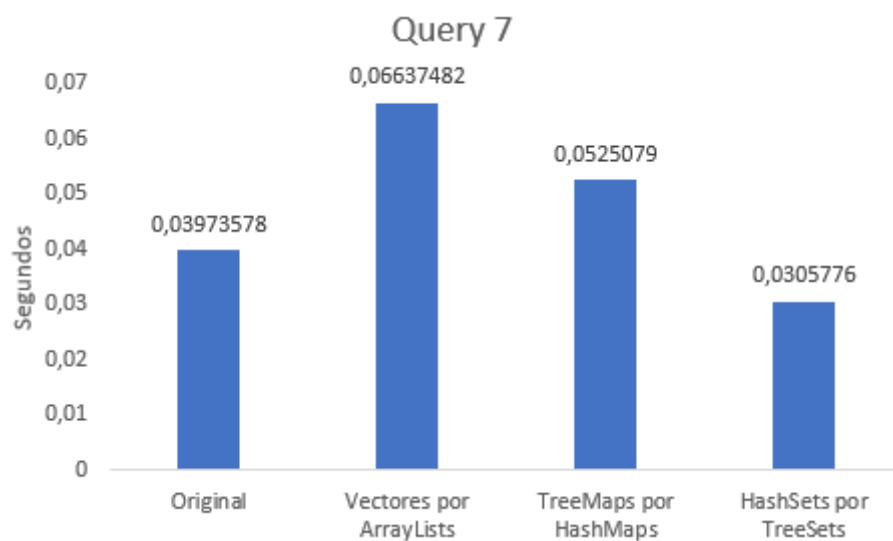


Figura 7-Comparações com diferentes Estruturas Query 7

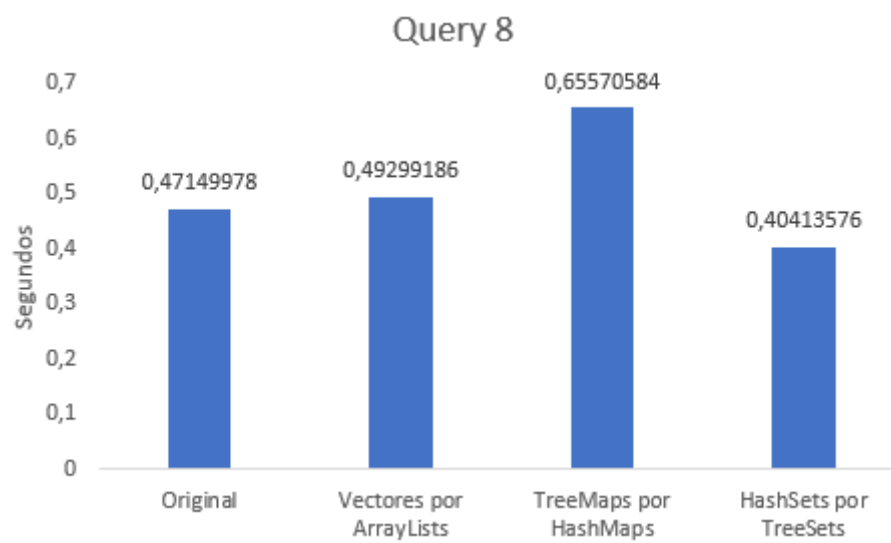


Figura 8-Comparações com diferentes Estruturas Query 8

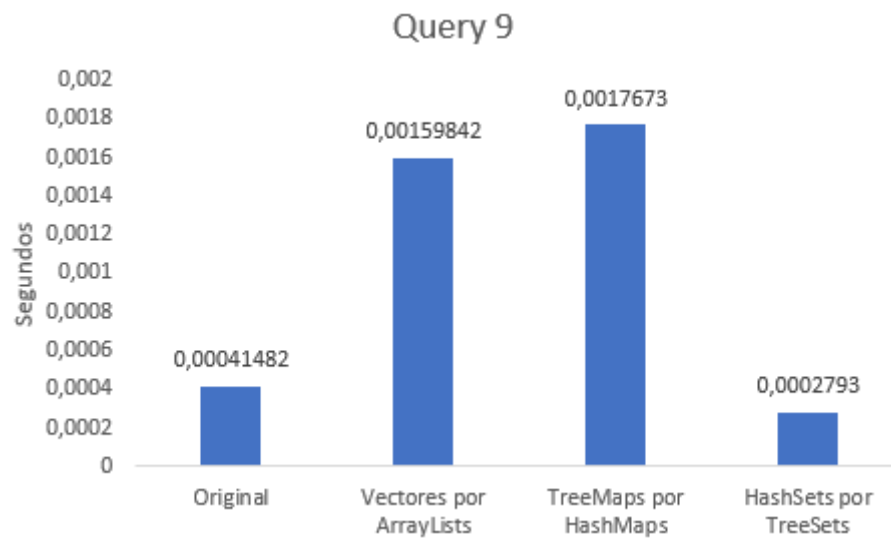


Figura 9-Comparações com diferentes Estruturas Query 9

4. Interpretação dos resultados e dificuldades encontradas

Ao longo deste projeto, tentamos sempre atingir o melhor resultado possível, a nível de tempos de execução das queries.

Após a análise dos resultados obtidos, e dos tempos de execução com estruturas diferentes, podemos destacar que HashSets apresentam uma melhor performance, mas isto pode não significar que seja a melhor opção. É de reparar que esta estrutura devolve as respostas às queries, mas de forma não ordenada, uma vez que os HashSets não implementam comparadores “no próprio corpo”. Para obter respostas ordenadas, seríamos obrigados à implementação de novos métodos de ordenação, o que, por fim, poderia ditar uma perda de performance.

Posto isto, que pensamos que tomámos, acima de tudo, as melhores decisões a nível de quais estruturas usar, pois assim comprovam os resultados das análises.

5. Conclusão

De acordo com o objetivo pretendido do projeto, isto é, gerir um rápido acesso a um número bastante elevado de informações, concluímos que esse objetivo foi cumprido.

Ao longo do trabalho, fomos conseguindo diminuir gradualmente o tempo de execução das funcionalidades do programa até a um ponto onde, de facto, não conseguimos aprimorar mais os nossos tempos de execução.

Também podemos assegurar que a modularidade e o encapsulamento de dados foram bem gerido.

Após uma análise na globalidade da eficiência do projeto em questão, conseguimos afirmar que o resultado é satisfatório, pois conseguimos responder a todas as queries propostas, e de uma forma “engenhosa”, conforme foi nos foi pedido.

Concluindo, com a elaboração deste projeto foram muitas as competências adquiridas e aprimoradas relativamente á linguagem de programação orientada a objetos. Para além disto, a grande melhoria foi a forma diferente que tivemos de pensar com o intuito de resolver problemas em larga escala, tornando todos os elementos do projeto mais racionais e obedecendo a regras de forma a manter a integridade e a abstração do maior número de dados, permitindo nos uma maior preparação para ambientes futuros, ou seja, mais profissionais e complexos.