

**Universidade do Minho**

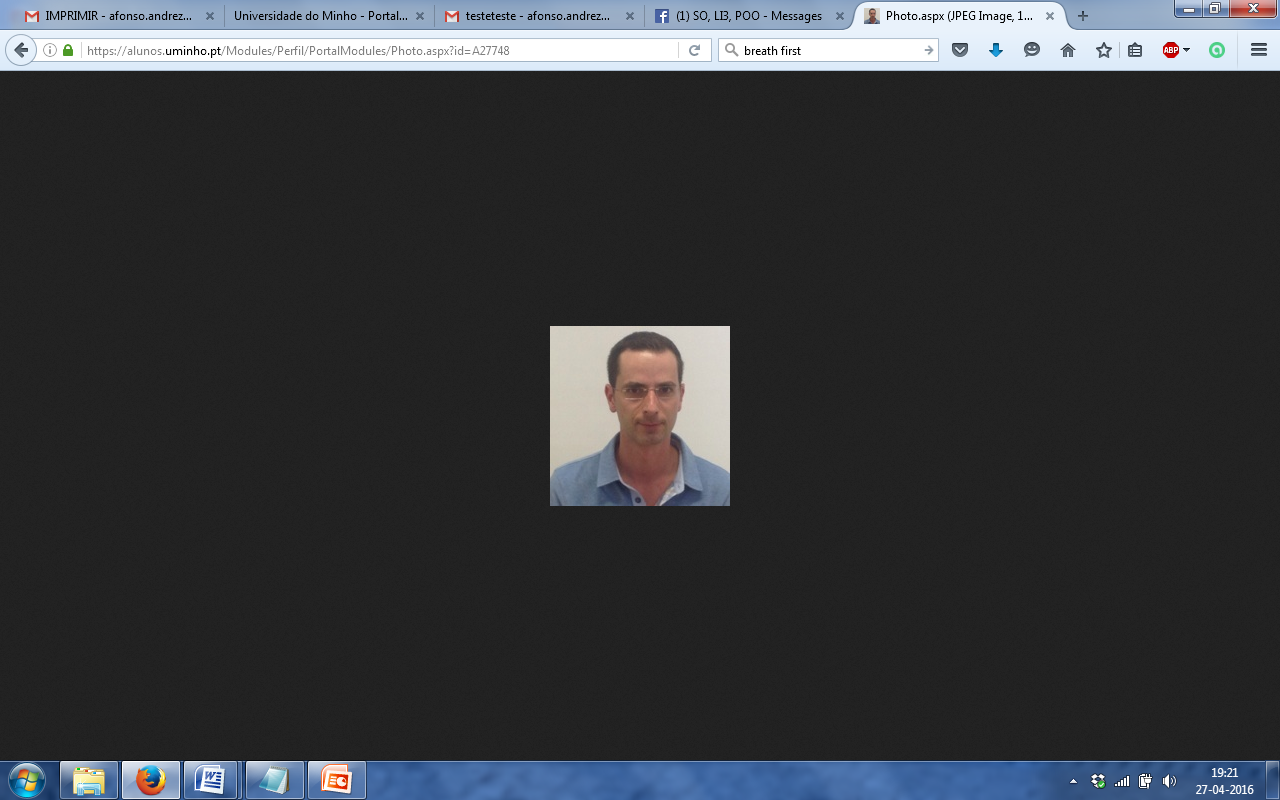
Escola de Engenharia

Mestrado Integrado em Engenharia Informática

**Laboratórios de Informática III**

Ano Letivo de 2015/2016

**Hipermercado**

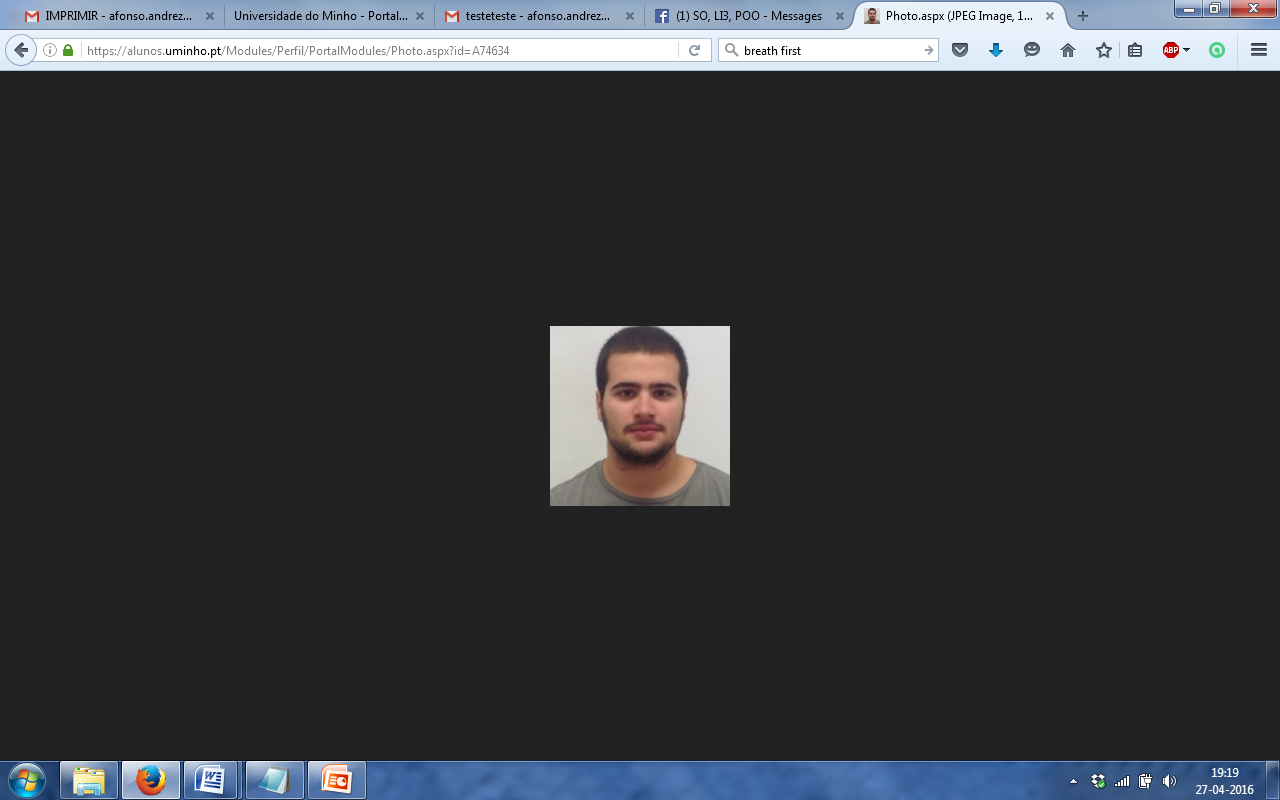


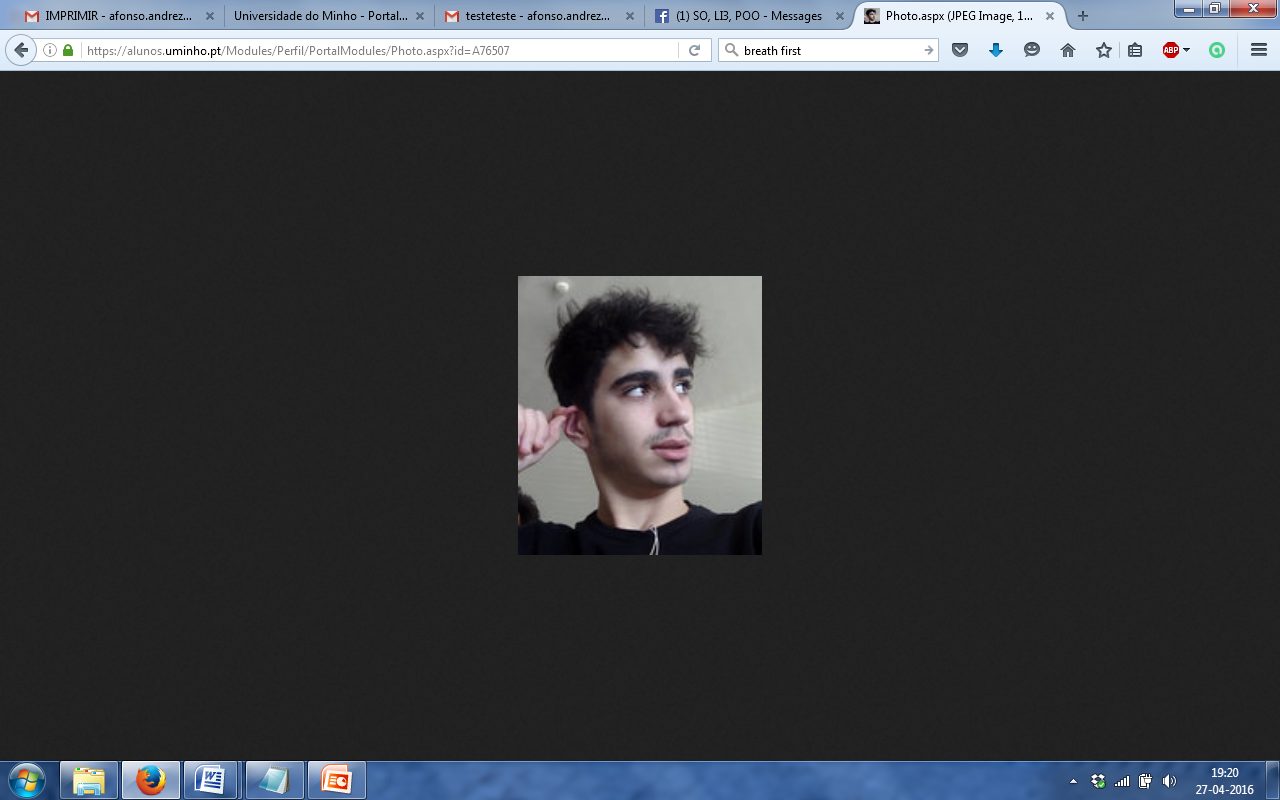
**A27748 - Gustavo José Afonso Andrez**

**A74634 - Rogério Gomes Lopes Moreira**

**A67664 - Samuel Gonçalves Ferreira**

Abril, 2016





**Docentes:**

Doutor Fernando Mário Junqueira Martins

Doutor Francisco Mendes Freitas Gomes Fonte

17 Junho 2016

# Índice

Índice ii

1. Relatório 1

1.1. Introdução 1

1.2. Descrição das classes implementadas 1

1.2.1 Catálogo de Clientes 2

1.2.2 Catálogo Produtos 3

1.2.3 Faturação 4

1.2.4 Filial 6

1.2.5 Hipermercado 7

1.2.6 Classes auxilares 9

1.3. Diagrama de classes 10

1.4. Testes de performance 11

1.4.1 Tempos de leitura do ficheiro base 11

1.4.1 Tempos de resposta das queries 12

1.5. Apresentação de Resultados 16

1.6. Conclusão 17

1. Relatório
   1. Introdução

O objetivo deste trabalho é, recorrendo à linguagem Java, conceber uma aplicação que permita ao utilizador obter informação organizada de um grande volume de dados de um modo eficiente. Nesta programação em larga escala as estruturas a implementar para guardar a informação deverão ser adequadas de modo a que os tempos de solicitação sejam reduzidos.

A classe agregadora deste projeto é a classe Hipermercado. Deverão ser desenvolvidas funcionalidades de carregamento de conjunto de códigos de clientes e produtos para um catálogo de clientes e produtos. A aplicação deverá também permitir carregamento de dados de vendas para a Faturação e Filiais.

Assim, o programa deverá permitir ao utilizador consulta dos dados através da seleção de queries e dados estatísticos definidos.

Para desenvolvimento da aplicação recorreu-se ao IDE NetBeans.

* 1. Descrição das classes implementadas

Neste ponto do trabalho pretendemos apresentar de um modo sintético mas claro, os atributos nas classes principais do projeto bem como os métodos presentes nas mesmas. Não são apresentados os métodos padrão (construtores, gets, sets, equals…).

No último ponto deste capítulo são compiladas as restantes classes, fazendo-se um breve descrição de cada uma delas.

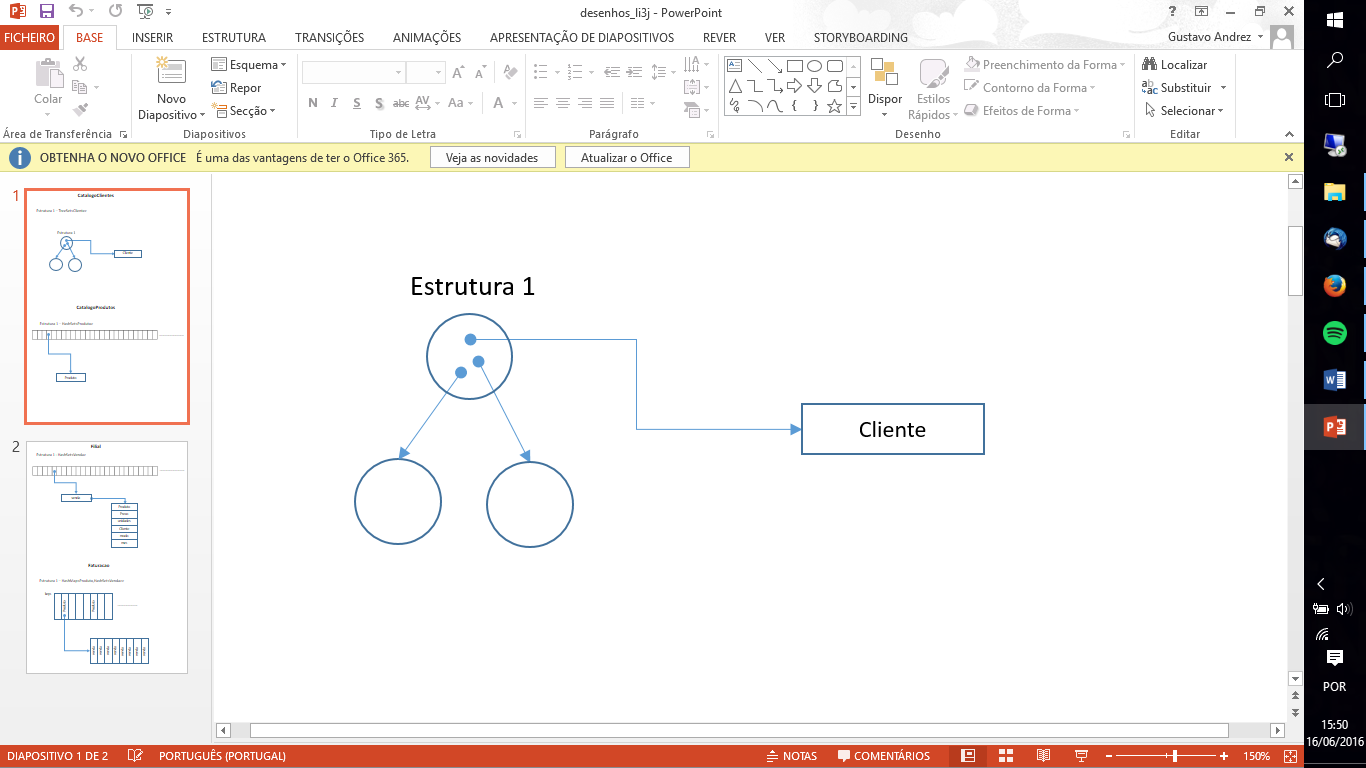
* + 1. Catálogo de Clientes

Esta é a classe responsável pelo armazenamento dos clientes.

A classe CatalogoCliente*s* é definida assim pelo atributo:

TreeSet<Cliente> clientes;

A figura seguinte apresenta esquematicamente esta estrutura:



**Figura 1-** Implementação da estrutura do catálogo de Clientes

Nesta classe foram definidos os seguintes métodos:

|  |  |
| --- | --- |
| public void addCliente(Cliente cliente) | Adiciona um cliente à árvore |
| public int size() | Devolve o nº de elementos da árvore |
| public boolean existeCliente(Cliente c) | Verifica se um cliente existe na árvore |

**Tabela 1-** Métodos da classe CatalogoCliente*s*

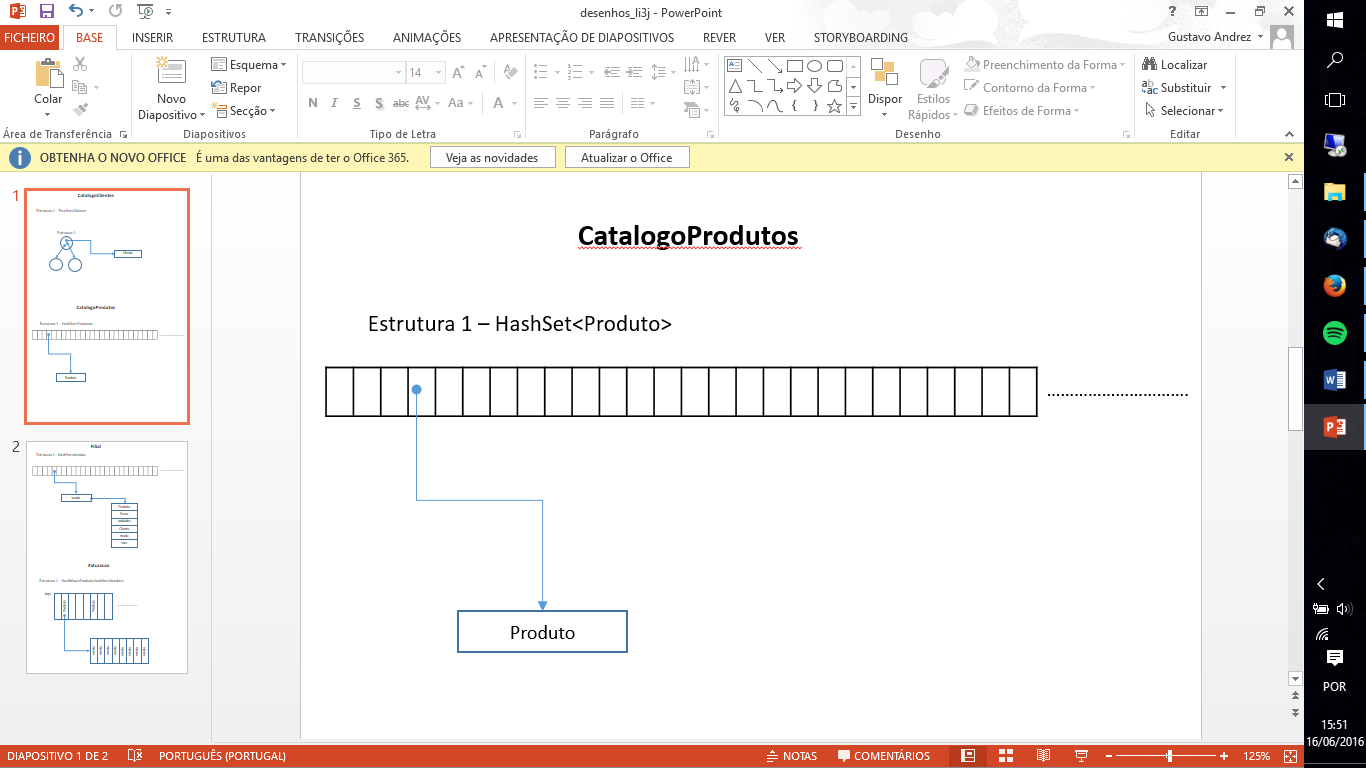
* + 1. Catálogo Produtos

Esta é a classe responsável pelo armazenamento dos produtos.

A classe CatalogoProdutos é definida pelo seguinte atributo:

HashSet<Produto> produtos;

A figura seguinte apresenta esquematicamente esta estrutura:



**Figura 2-** Implementação da estrutura do catálogo de Produtos

Nesta classe foram definidos os seguintes métodos:

|  |  |
| --- | --- |
| public void addProduto(Produto produto) | Adiciona um cliente ao Set |
| public int size() | Devolve o nº de elementos do Set |
| public boolean existeProduto(Produto p) | Verifica se um produto no Set |

**Tabela 2-** Métodos da classe CatalogoProdutos

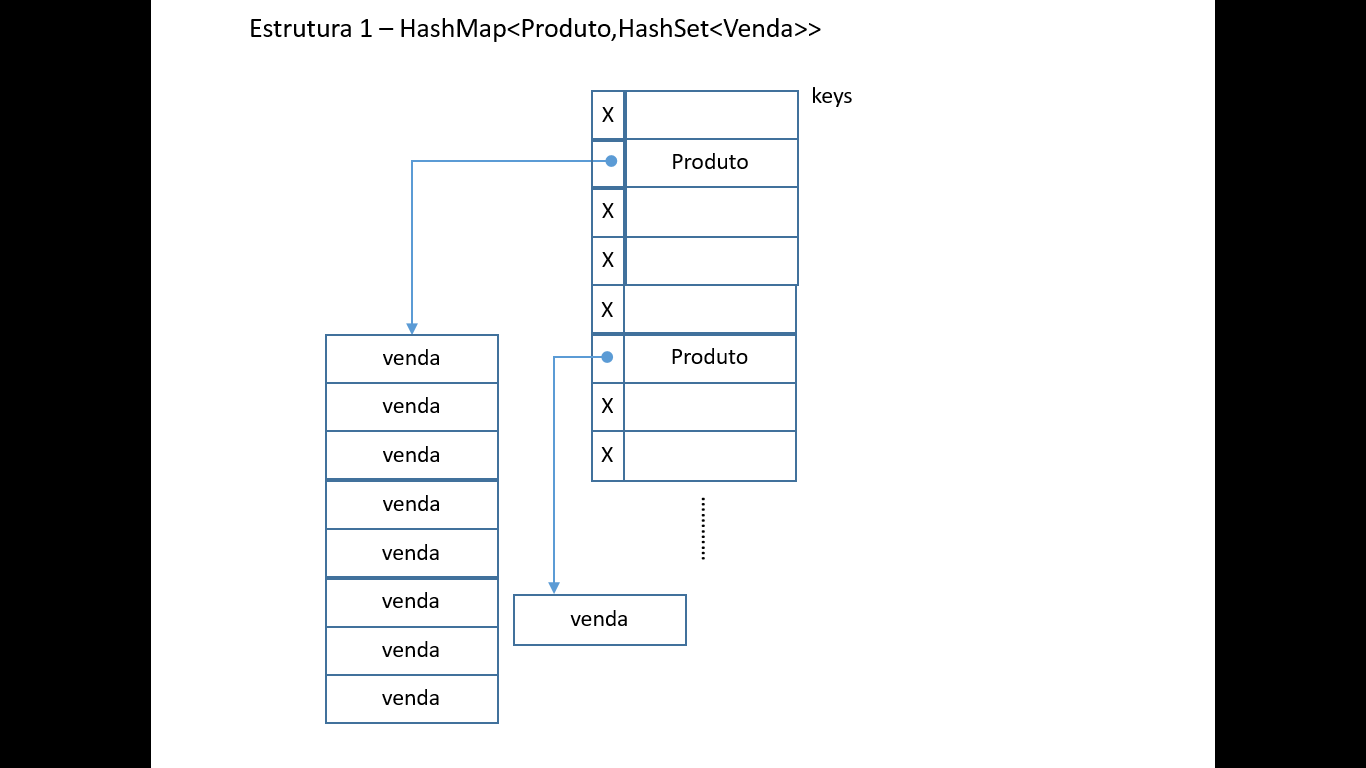
* + 1. Faturação

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;

A figura seguinte apresenta esquematicamente esta estrutura:



**Figura 3-** Implementação da estrutura da Faturação

Nesta classe foram definidos os seguintes métodos:

|  |  |
| --- | --- |
| HashSet<Venda> getFaturacaoProduto(Produto p) | Devolve o conjunto de vendas de um produto |
| void setFaturacaoProduto(Produto p,HashSet<Venda> vendas) | Adiciona um produto e respectivas vendas ao HashMap |
| void addVenda(Venda v) | Adiciona uma venda ao HashMap |

**Tabela 3-** Métodos da classe Faturacao

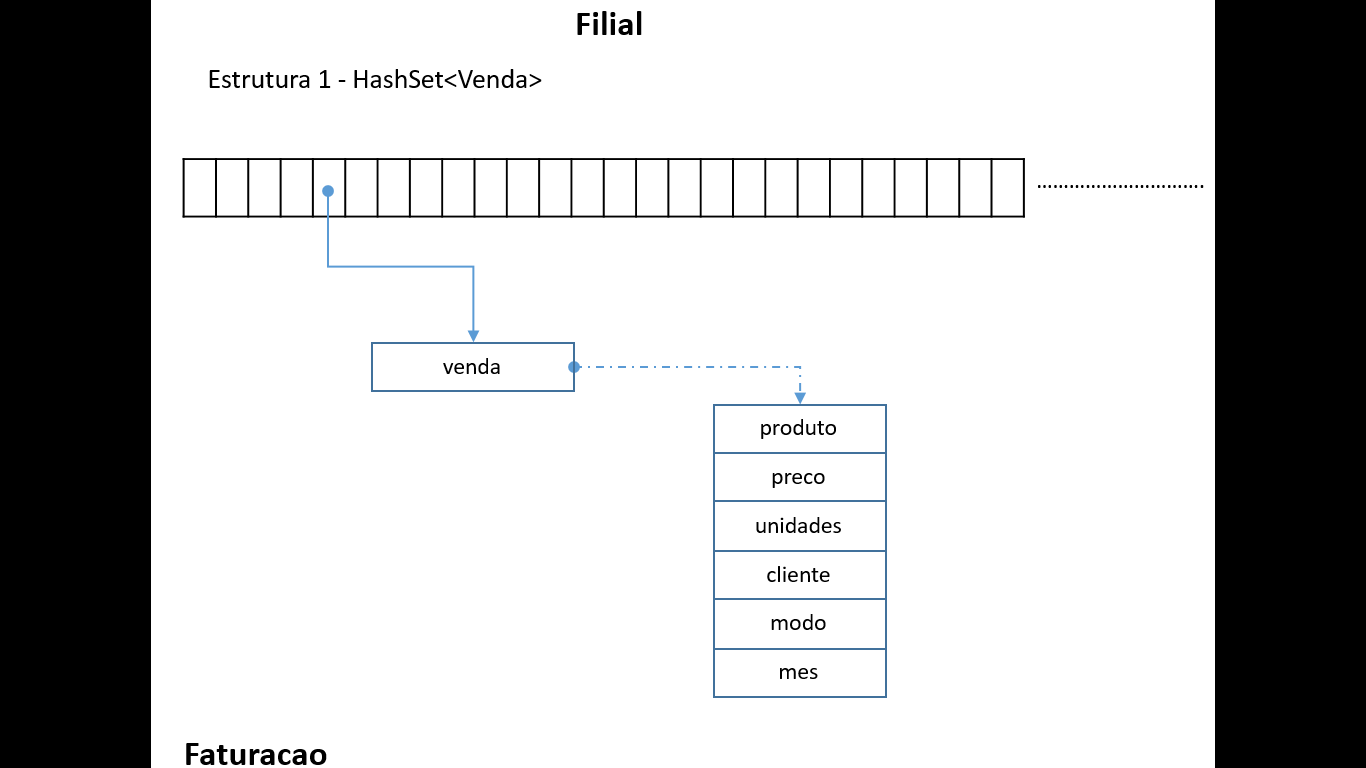
* + 1. Filial

Esta é a classe responsável pelo armazenamento dos dados das vendas distinguidos por filial.

A classe Filial é definida pelo seguinte atributo:

HashSet<Venda> vendas

A figura seguinte apresenta esquematicamente esta estrutura:



**Figura 4-** Implementação da estrutura da Filial

* + 1. Hipermercado

A classe Hipermercado é definida pelos seguintes atributos:

CatalogoProdutos catalogoProdutos;

CatalogoClientes catalogoClientes;

Faturacoes faturacao;

Filial vendasFilial1;

Filial vendasFilial2;

Filial vendasFilial3;

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

Nesta classe foram definidos os seguintes métodos (não são apresentados métodos respetivos às queries):

|  |  |
| --- | --- |
| boolean verificaVenda(Venda venda) | Verifica se todos os parâmetros da venda são válidos:  - se cliente e produto existem nos respectivos catálogos;  - se o preço é maior ou igual a zero;  - se o numero de unidades é maior que zero; |
| void gravarEstado(String fich) throws IOException | Serializa o estado do Hipermercado para um ficheiro; |
| static Hipermercado lerEstado(String fich) throws IOException, ClassNotFoundException | Restaura o estado do Hipermercado a partir de um ficheiro serializado; |
| TripleFloat leituraVendas(String fich) | Preenche os atributos do Hipermercado a partir da leitura de um ficheiro de vendas. Retorna um triplo com a informação do número de vendas lidas, válidas e a preço zero; |
| int nClientesCompraram() | Devolve o número de clientes que nunca realizaram compras; |
| float totalFaturado() | Devolve o somatório dos valores faturados em todas as vendas; |
| ArrayList<Integer> totalCompras() | Devolve uma lista com 12 posições, cada uma delas com o número de compras efetuadas no mês correspondente; |
| ArrayList<ArrayList<Float>> faturacaoMes() | Devolve uma matriz 4x12 posições, em que 3x12 posições contêm o somatório dos valores faturados por mês e filial e 12 posições apresentam o somatório dos valores faturados por mês (faturação global); |
| ArrayList<Integer> clientesPorMes() | Devolve uma lista com 12 posições, cada uma delas com o número de clientes distintos que realizaram compras nesse mês; |

**Tabela 4-** Métodos da classe Hipermercado

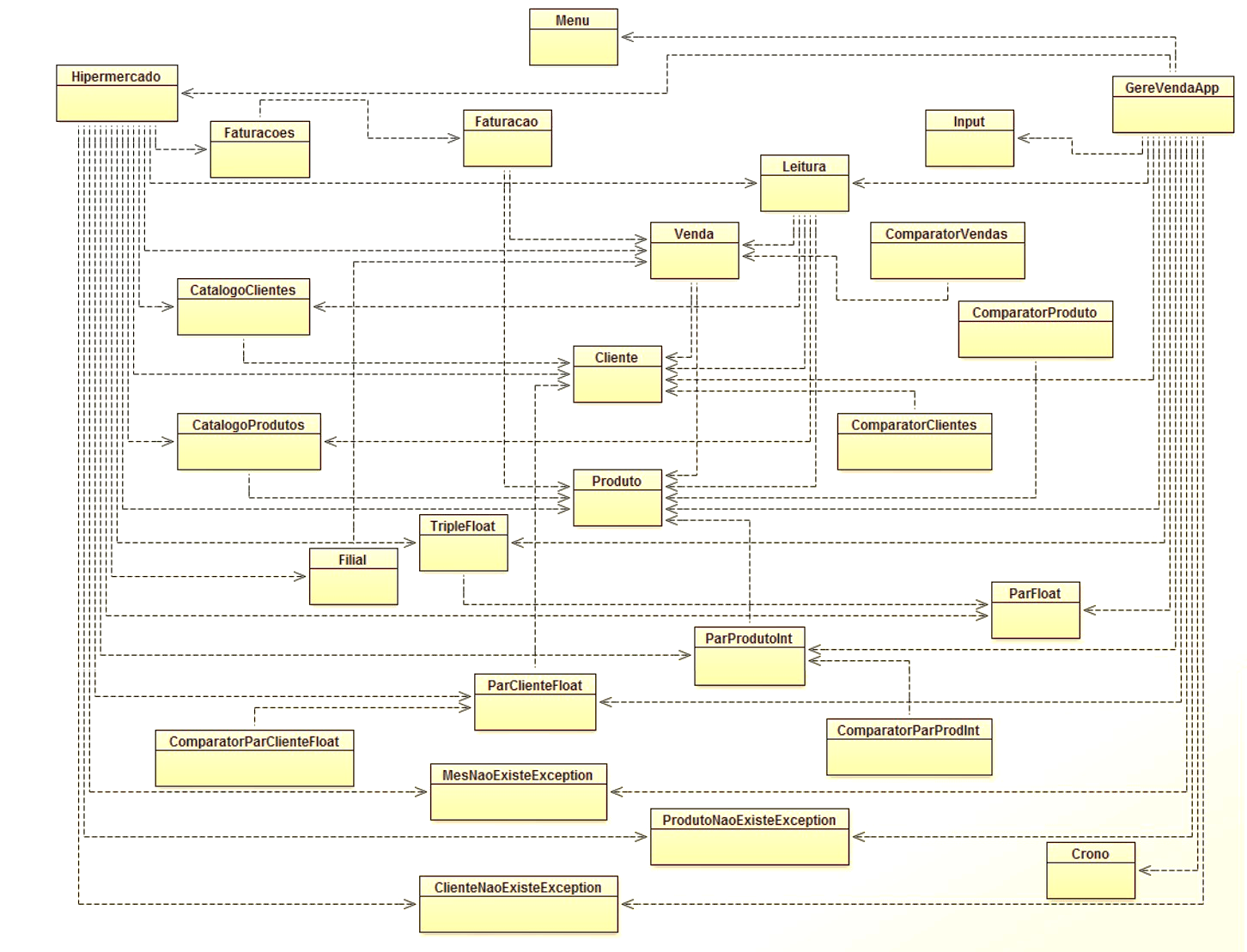
* + 1. Classes auxilares

|  |  |
| --- | --- |
| **Classe** | **Descrição** |
| GereVendaApp | Métodos que permitem a interação do utilizador com os dados do programa |
| Faturacoes | Possui 4 instâncias de Faturacao correspondendo à faturação global e às 3 filiais |
| Produto | uma string com o código do produto |
| Cliente | uma string com o código do cliente |
| Venda | Possui os sete campos que caracterizam uma venda |
| Leitura | Possui os métodos que permitem fazer a leitura dos ficheiros para catálogo e métodos de *parsing* das vendas |
| Input | Código fornecido para leitura do ecrã |
| Menu | Faz a gestão dos menus |
| Crono | Código fornecido para registo dos tempos de execução |
| TripleFloat | Tuplo com um float e um par de floats |
| ParClienteFloat | Tuplo com cliente e float |
| ParProdutoInt | Tuplo com produto e float |
| ParFloat | Tuplo de floats |
| ComparatorClientes | Ordena os clientes por ordem alfabética |
| ComparatorProduto | Ordena os produtos por ordem alfabética |
| ComparatorVendas | Ordena as vendas pela ordem dos parâmetros que os constituem (ordem original do ficheiro Vendas) |
| ComparatorParClienteFloat | Ordena o tuplo ParClienteFloat primeiro descendentemente pelo float e seguidamente os clientes por ordem alfabética |
| MesNaoExisteException | Definição da excepção de mês inválido |
| ProdutoNaoExisteException | Definição da excepção de produto inexistente |
| ClienteNaoExisteException | Definição da excepção de cliente inexistente |

**Tabela 5-** Métodos das classes auxiliares

* 1. Diagrama de classes

Após concluída a implementação do projeto, foi elaborado o respectivo diagrama de classes apresentado abaixo. Neste diagrama podem ser observadas as relações entre as diversas classes que constituem o programa. Analisando o diagrama vemos que na aplicação criada as classes Hipermercado e GereVendaApp são aquelas se relacionam com maior número de classes.



**Figura 5-** Diagrama de classes

* 1. Testes de performance
     1. Tempos de leitura do ficheiro base

|  |  |  |  |
| --- | --- | --- | --- |
| Tipo | Ficheiro | Modo | Tempo (seg.) |
| Sem *parsing* | 1M | *Scanner* | 3,36 |
| *BufferedReader* | 0,27 |
| 3M | *Scanner* | 9,07 |
| *BufferedReader* | 2,77 |
| 5M | *Scanner* | 14,00 |
| *BufferedReader* | 3,41 |
| Com *parsing* | 1M | *Scanner* | 4,03 |
| *BufferedReader* | 3,04 |
| 3M | *Scanner* | 15,35 |
| *BufferedReader* | 10,14 |
| 5M | *Scanner* | 27,64 |
| *BufferedReader* | 19,67 |

**Tabela 6-** Tempos de execução de leitura com e sem *parsing*

**Gráfico 1-** Tempos de execução de leitura sem *parsing*

**Gráfico 2-** Tempos de execução de leitura com *parsing*

* + 1. Tempos de resposta das queries

Nas queries em que é exigido um critério de ordenação, foram implementados algoritmos que usam TreeSets como forma de ordenação. Esta poderá não corresponder à solução mais eficiente mas é uma forma prática e flexível de proceder à ordenação de diferentes estruturas de dados.

O uso de TreeSets como forma de ordenação de dados faz com que a sua alteração para um HashSet resulte num resultado incorreto.

Existem também queries em que são usados arrayLists como forma de devolver os resultados sob a forma de lista. Dada a dimensão destes arrays, optámos por não fazer variar, nos testes, estas estruturas.

Os testes realizados e apresentados abaixo tiveram como *input* o ficheiro de 5 milhões de vendas.

A variante usada no projeto encontra-se assinalada a cor verde.

**Gráfico 3-** Tempos de execução da query 5 com diferentes estruturas com argumento igual a cliente Z5000

**Gráfico 3-** Tempos de execução da query 6 com diferentes estruturas

**Gráfico 4-** Tempos de execução da query 7 com diferentes estruturas

**Gráfico 5-** Tempos de execução da query 8 com diferentes estruturas com argumento igual a 1000 produtos

**Gráfico 6-** Tempos de execução da query 9 com diferentes estruturas com argumento igual a 1000 clientes e Produto AF1084

* 1. Apresentação de Resultados

Foi feito o registo do tempo de resposta às queries, medido em tempo de execução com recurso à classe Crono.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Query \ ficheiro | 1M (ms) | 3M (ms) | 5M (ms) | Argumento |
| Estatística 1 | 442,5 | 1563,07 | 2460,45 | ---- |
| Estatística 2 | 808,55 | 3072,04 | 4448,72 | ---- |
| 1 | 307,37 | 313,77 | 306,46 | Mês = 3 |
| 2 | 387,4 | 1992,51 | 897,74 | Mês = 3 |
| 3 | 109,28 | 706,42 | 809,13 | Cliente = Z5000 |
| 4 | 0,35 | 1,23 | 0,74 | Produto = AF1084 |
| 5 | 146,69 | 913,99 | 946,73 | Cliente = Z5000 |
| 6 | 383,39 | 601,09 | 812,46 | X = 20 |
| 7 | 594,97 | 2009,69 | 3467,83 | ----- |
| 8 | 554,47 | 1898,57 | 3369,79 | X = 20 |
| 9 | 3,38 | 4,69 | 4,46 | X = 20  Produto = AF1084 |

**Tabela 7 -** Tempos de execução\* das estatísticas e queries conforme o ficheiro de vendas carregado

\* *valores adquiridos em pc com processador 2,5 GHz Intel Core i5.*

* 1. Conclusão

A partir da análise dos gráficos dos testes de performance (tempo de execução das queries), verifica-se que em algumas das queries a utilização de TreeSets não é a opção mais eficiente. Por isto, consideramos que seria vantajoso repensar os algoritmos utilizados por forma que a ordenação não dependesse destas estruturas.

Como conclusão salientamos os seguintes aspetos:

- todas as queries estão a apresentar todos os dados solicitados;

- os tempos de carregamento e execução das queries são bastante satisfatórios;

- consideramos que o código elaborado está bem organizado;

- a resposta às queries é apresentado de um modo limpo e organizado;

Pelo referido, consideramos que no geral os objetivos do trabalho foram cumpridos.