|  |  |
| --- | --- |
| EENG | **Escola de Engenharia**  Departamento de Engenharia Informática  Mestrado Integrado em Engenharia Informática  Laboratórios de Informática III |

Relatório de Laboratórios de Informática III

**GereVendas**

**(Trabalho Prático de Java)**

Grupo 1

João Martins (A68646)



João Pereira (A75273)



Manel Castro (A71646)



Índice

[1. Arquitetura de classes 2](#_Toc453495970)

[1.1 CatalogoProdutos 0](#_Toc453495971)

[1.1.2 Atributos 0](#_Toc453495972)

[1.1.3 Esquema da estrutura de dados utilizada 0](#_Toc453495973)

# Arquitetura de classes

A arquitetura de classes utilizada neste projeto de Java é até certo ponto uma “tradução” da organização dos módulos do projeto de C, ainda que tenhamos efetuado algumas alterações e adaptações dos vários tipos de dados para reduzirmos os tempos de leitura dos ficheiros de texto. Além das alterações que visam reduzir o tempo de leitura (e que irão ser discutidas mais adiante), também acrescentámos:

* pares e triplos para agrupar os resultados das várias *queries*;
* estatísticas relativas ao último ficheiro lido e estatísticas gerais, relativas aos dados do hipermercado;
* uma classe de medição de tempos (a classe **Crono**, disponibilizada pelo professor Mário Martins);

A arquitetura de classes utilizada pelo grupo tem então a seguinte forma:

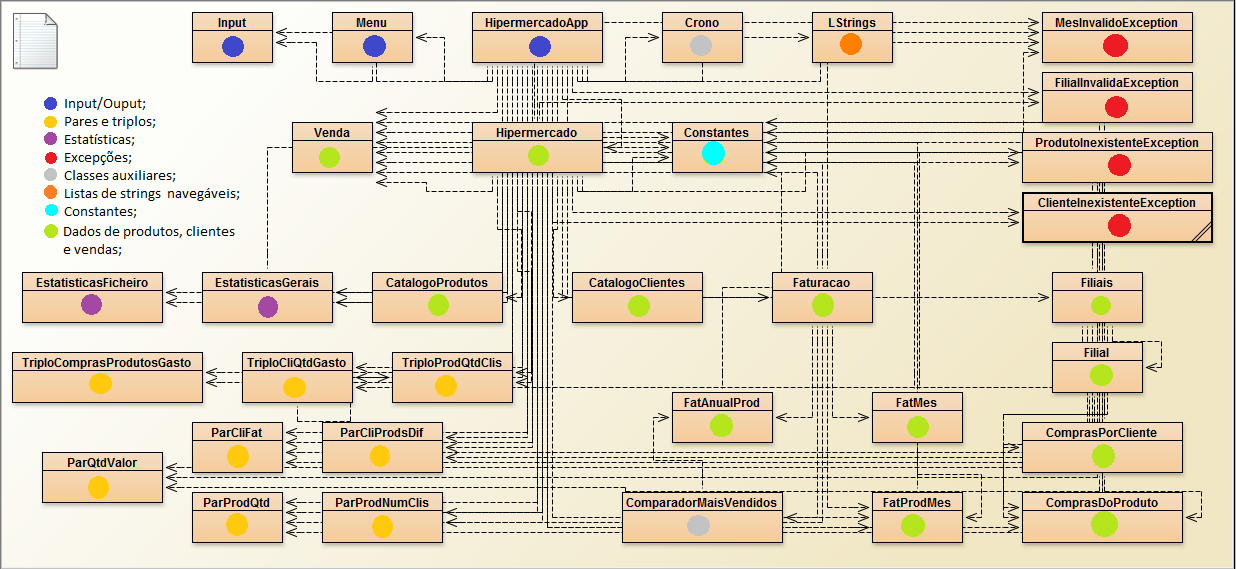


Figura 1-Arquitetura de classes

## CatalogoProdutos

Cada instância da classe **CatalogoProdutos** permite o registo de códigos de produto (valores da classe String), bem como a verificação da existência de um determinado código de produto e a consulta do número total de produtos registados nessa instância.

No contexto da aplicação desenvolvida, utilizamos uma instância de CatalogoProdutos nas v.i. de Hipermercado para registar os códigos de todos os produtos disponíveis, obter o total de produtos existentes no hipermercado e verificar a existência dos códigos de produto que surgem nas linhas do ficheiro de vendas e dos códigos de produto que o utilizador introduz em algumas das *queries* interativas.

### 1.1.1 Atributos

De forma a conseguirmos inserções e consultas em tempo tendencialmente constante, optámos por usar um HashSet<String> para armazenar os códigos dos produtos de cada instância de CatalogoProdutos. Contudo, para podermos mudar facilmente a implementação para outro tipo de Set, declaramos a única v.i. de CatalogoProdutos da seguinte forma:



Assim, se pretendermos alterar a implementação do conjunto de códigos de produtos basta modificarmos a expressão codigosProdutos = new HashSet<>(), nos construtores de CatalogoProdutos, conforme o tipo de Set que pretendamos utilizar.

**Nota:** de referir que utilizamos um **Set** (e não uma **List**) porque cada código de produto é único e, por conseguinte, é necessário garantir a inexistência de elementos repetidos em cada instância de CatalogoProdutos e um **Set** permite-nos precisamente alcançar esse fim.

### 1.1.2 Esquema da estrutura de dados utilizada

…

NR1091

AF1184

DV1252

\*

\*

\*

AF1184

\*

\*

…

DV1252

\*

NR1091

Figura 2-HashSet (esquema da estrutura de dados utilizada no catálogo de produtos)

## CatalogoClientes

As instâncias e API de **CatalogoClientes** são semelhantes às de CatalogoProdutos, diferindo apenas no conteúdo dos catálogos, já que num caso temos clientes e no outro temos produtos. Assim, cada instância de CatalogoClientes permite-nos: inserir códigos de cliente, obter o número total de clientes registados na mesma e testar se um dado código de cliente existe nesse catálogo.

No caso da aplicação desenvolvida, utilizamos uma instância de CatalogoClientes nas v.i. de Hipermercado, de forma a armazenar os códigos dos vários clientes registados no hipermercado, obter o total de clientes registados, comprovar a existência dos códigos de cliente que surgem nas linhas do ficheiro de vendas e validar códigos de cliente introduzidos pelo utilizador em algumas das *queries* interativas da aplicação.

### Atributos

Cada instância de CatalogoClientes tem uma única v.i. que, tal como a v.i. de CatalogoProdutos, é um Set<String> implementado com um HashSet<String>, permitindo-nos a realização inserções e consultas em tempo tendencialmente constante.

Assim, na declaração de v.i. de CatalogoClientes temos:



**Nota:** tal como os códigos de produtos, os códigos de cliente também são únicos. A utilização de um **Set** (em vez de uma **List**) garante a inexistência de códigos de cliente repetidos em cada instância de CatalogoClientes.

### Esquema da estrutura de dados utilizada

…

Z5000

B3304

S4262

\*

\*

\*

Z5000

\*

\*

…

S4262

\*

B3304

Figura -HashSet (esquema da estrutura de dados do catálogo de clientes)

## Faturacao

Cada instância de **Faturacao** guarda informação relativa às vendas mensais e do ano todo, filial a filial e globalmente. Na parte das vendas de um dado mês temos apenas dados relativos aos produtos vendidos nesse mês. Na informação do ano todo temos dados sobre todos os produtos, mesmo os que nunca foram vendidos, para que a faturação possa indicar quais produtos nunca foram comprados. É também importante notar que não é feita qualquer referência a clientes, já que toda a informação sobre quem realizou cada uma das compras é mantida pelas respetivas filiais.

No contexto da aplicação desenvolvida, temos uma instância de Faturacao nas v.i. da classe de Hipermercado, que nos permite responder a questões como:

* Quais produtos nunca foram comprados e o seu respetivo total (*query* 1);
* Qual foi o número global de vendas realizadas num dado mês (*query* 2);
* Quantas vezes um produto foi comprado e qual foi o total faturado com esse produto, para cada um dos meses (*query* 4);
* Quais foram os X produtos mais vendidos em todo o ano (*query* 6);

**Nota:** na *query* 6, X é um inteiro dado pelo utilizador.

### Atributos

Nas v.i. de instância de Faturacao temos:



, onde:

* **nfiliais** é o número de filiais para as quais a instância guarda informação de faturação (guardamos este valor porque embora no caso da nossa aplicação conheçamos o número de filiais, se mudássemos para outro contexto o número de filiais poderia ser diferente);
* **fatMensal** é um *array* que na posição de índice i tem uma instância de FatMes, com informação relativa à faturação do mês i (a classe FatMes será descrita a seguir, como parte integrante das estruturas de dados usadas na faturação);
* **todosProdutos** mapeia códigos de produtos em instâncias de FatAnualProd com informação relativa à faturação anual do produto utilizado como chave e pesquisa (a classe FatAnualProd também será descrita abaixo);

### Esquema das estruturas de dados utilizadas

private Map<String, FatAnualProd> todosProdutos; (implementado com TreeMap)

private int nfiliais;

private FatMes[] fatMensal;

(*padding*)

FatMes (Jan)

…

FatMes (Fev)

FatMes (Dez)

[0]

[1]

…

[2]

[12]

…

<”NR1076”, FatAnualProd>

<”AF1182”, FatAnualProd>

<”ZQ1206”, FatAnualProd>

…

…

…

Figura -Esquema de estrutura de dados da faturação

## FatMes

Cada instância de FatMes guarda dados relativos à faturação de um dado mês, nomeadamente:

* Total de vendas do mês;
* Total faturado no mês;
* Faturação de cada um dos produtos vendidos nesse mês (em instâncias FatProdMes);

**Nota:** além dos dados referidos, cada instância de FatMes guarda ainda o mês a que diz respeito e o número de filiais. O número de filiais é guardado para que, aquando da construção da faturação de um produto no mês (FatProdMes) saibamos para quantas filiais teremos que guardar informação. Poderíamos ter usado o valor 3, mas nesse caso as classes FatMes e FatProdMes não seriam independentes do contexto da nossa aplicação. Relativamente a guardarmos o mês nas v.i. de FatMes, fizemo-lo porque embora no contexto das instâncias de Faturacao saibamos a que mês diz respeito cada instância de FatMes (através da posição que essa instância ocupa no array FatMensal[] ), noutro contexto poderíamos não saber. Assim, se guardarmos o mês numa v.i. de FatMes, garantimos também a independência de contexto.

