Mestrado Integrado em Engenharia Informática
Relatório da Unidade Curricular
Laboratórios de Informática 3
Projecto em Java
2019/2020

Luís Manuel Pereira 77667
Pedro Miguel Duarte Araújo 70699
Simão Freitas Monteiro 85489

Maio, 2020

Índice

1 Introdução

No seguimento do trabalho desenvolvido até à data na unidade curricular de LI3 e recorrendo a um paradigma de programação diferente, a Programação Orientada a Objectos, a segunda fase desta unidade curricular retoma o desenvolvimento, em parâmetros similares, de um sistema de gestão de vendas de hipermercado (SGV). No entanto, usando uma linguagem de programação que obdece ao novo paradigma, nomeadamente **JAVA**.

Irá-se assim desenvolver uma aplicação *desktop* que continua a usar conceitos relativos a modularidade e encapsulamento de dados, com código robusto e reutilizável. Trata-se de um problema de contexto real que reflecte a gestão de grandes volumes de dados, pelo que se procura implementar soluções optimizadas que num contexto prático possam ser escaláveis.

Este *use case* requer um sistema capaz de ler e processar as linhas de texto (após *load* de ficheiros de texto - .txt) que contêm códigos de produtos e clientes, bem como o registo de todas as compras e vendas efectuadas, por produtos, cliente e filial (incluindo detalhes de produto como preço e o seu stock).

Para avaliar a eficiência na resolução de uma listagem de tarefas foram efectuados testes de performance ao sistema desenvolvido.

2 Arquitectura da Aplicação

No seguimento do que foi efectuado anteriormente, foi implementada a já conhecida e usada arquitectura MVC (Model-View-Controller), cuja Figura 1 visa demonstrar os diferentes *packages* e *classes* presentes na aplicação, estes serão explicados detalhadamente nas secções seguintes.

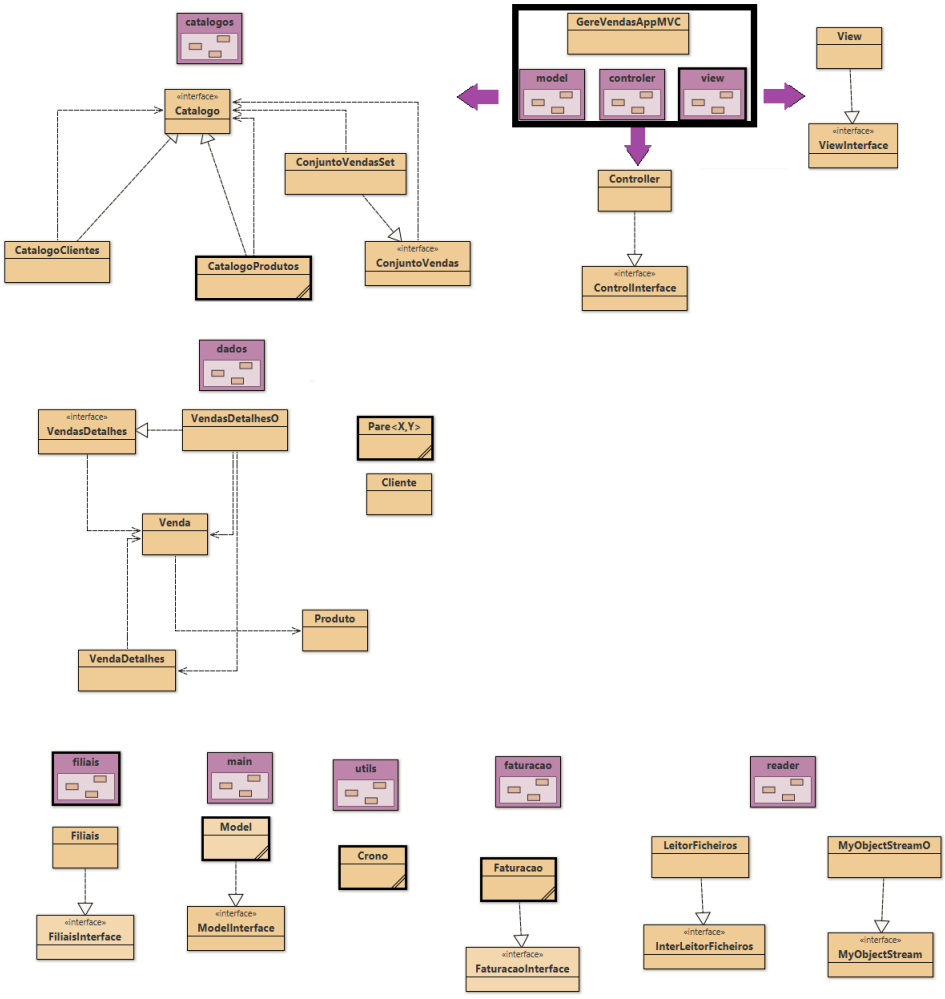

Figure 1: Diagrama de classes, representando o modelo MVC

2.1 *Model*

O *Model* que se encontra no presente trabalho consiste num *package* chamado de *"model"*. Este, como podemos observar pela Figura 1, contém diversos *packages* incluídos no mesmo.

- Um deste *packages*, o ***main***, contém a principal *classe* da aplicação, de nome *"IModel"* (Figura 2), que é responsável por responder às diversas *queries* assim como memorizar diversos tipos de dados;

```
package model.main;

import model.data.Par;

import java.util.List;
import java.util.Map;

public interface IModel {
    String[] loadData();

    int[] estatisticas();
    double faturacaoTotal();
    int[] purchasesPerMonth();
    double[][] faturacaoPorMesPorFilial();
    int[][] nClientsPurchaseMonthFilial();

    List<String> query1();
    List<Par<Integer,Integer>> query2(int mes);
    List<Par<Integer, Par<Integer,Double>>> query3(String cliente);
    List<Par<Integer, Par<Integer,Double>>> query4(String produto);
    Map<String, Integer> query5(String cliente);
    Map<String, Par<Integer,Integer>>query6(int x);
    List<Map<String, Double>> query7();
    Map<String, Integer> query8(int x);
    Map<String, Par<Integer, Double>> query9(String produto, int x);
    List<double[]> query10(String produto);
}
```

Figure 2: *Classe IModel*

- No *package* ***Cat***, temos uma interface definida pela *classe ICat* tem a função de armazenar os dados (Figura 3). A esta por sua vez estão associadas duas *subclasses* chamadas de *CatClientes e CatProdutos* que são responsáveis por implementar a interface e assim como memorizar os catálogos e os produtos;

```java
package model.Cat;

import java.util.Collection;
import java.util.Map;

public interface ICat {
    void add(String key);
    void add(String key, Object value);
    void remove(String key);
    void remove(Collection<String> keys);
    boolean isInCat(String key);
    int catSize();
    Object getValue(String key);
    Collection<Object> getAllValues();
    Collection<String> getAllKeys();
    Map<String, Object> getAll();
    ICat clone();
}
```

Figure 3: *Classe ICat*

- O *package* ***faturacao*** (Figura 4, é responsável pelas informações relativas:

  - Aos produtos e respectivas vendas (mensais), tendo em consideração o custo associado;

  - Á filial onde a venda foi efectuada.

```java
package model.faturacao;

import model.data.Par;
import model.data.Venda;

import java.util.Collection;
import java.util.List;
import java.util.Map;

public interface IFacturacao {

    void adicionaVendas(Collection<Venda> vendas);

    int PurchasesValueZero();
    int numeroProdutos();
    double faturacao();
    double faturacao(int mes, int filial);
    double faturacao(String produto, int mes);
    int purchases();
    int purchases(int mes);
    int purchases(int mes, int filial);
    int purchases(String produto, int mes);
    int numberClientsForProduct(String produto, int mes);

    Map<String, Par<Integer,Integer>> mostBoughtProducts(int x);
    List<double[]> faturacaoPorProduto(String produto);

    List<String> getProducts();
}
```

Figure 4: *Classe IFaturcao*

Definiu-se facturação como sendo um catálogo especial de produtos, onde cada produto tem um objecto *IVendasData*, Figura 6, este objecto tem como objectivo memorizar todas as vendas desse produto.

- O comportamento do *package* ***filiais*** (Figura 5) é similar ao do anteriormente descrito, *package* ***faturacao***, a diferença prende-se pelo o facto de que este é relativo à informação dos clientes e suas respectivas compras. Tem assim também em consideração o custo associado e a filial onde a compra foi efectuada.

```java
package model.filiais;

import model.Cat.ICat;
import model.data.Par;
import model.data.Venda;

import java.util.Collection;
import java.util.Map;

public interface IFiliais {

    void addSales(Collection<Venda> vendas);

    int nFiliais();
    int clientsWithPurchases();
    int clientsWithPurchases(int mes, int filial);
    int compras(String cliente, int mes);
    int difProductsPerClient(String cliente, int mes);
    double clientSpent(String cliente, int mes);

    ICat produtosComprados(String cliente);
    Map<String, Double> topClients(int filial);
    Map<String, Integer> mapClientsDifProducts(int x);
    Map<String, Par<Integer,Double>> topClientsForeachProduct(String produto, int x);
}
```

Figure 5: *Classe IFiliais*

Definiu-se filial como sendo um catálogo especial de clientes, onde cada cliente tem um objecto *IVendasData*, Figura 6, este objecto tem como objectivo memorizar todas as compras efectuadas por um cliente.

- Como referido anteriormente, o objecto *IVendasData*,Figura 6, serve para memorizar um conjunto de vendas e para fornecer estatísticas sobre a mesmas. Este objecto encontra-se inserido no *package* ***data***.

```java
package model.data;

import java.util.List;

public interface IVendasData {
    void addVenda(Venda v);
    double faturacaoTotal(int mes, int filial);
    double faturacaoTotal(int mes);
    double faturacaoTotalF(int filial);
    double faturacaoTotal();
    int salesTotal(int mes, int filial);
    int salesTotal(int mes);
    int purchaseTotal(int mes, int filial);
    int purchaseTotal(int mes);
    int purchaseTotal();
    Par<Integer, Double> purchaseTotal(String produto);
    int boughtProducts();
    List<String> boughtProducts(int mes);
    int ClientsWithPurchases(int mes);
    int ClientsWithPurchases();
    List<String> ClientsWithPurchases(int mes, int filial);
    int purchaseEqualZero();
    int salesSize();
}
```

Figure 6: *Objecto IVendasData*

- O *package* ***reader***, Figura 7, contém classes que tem como funções ler os ficheiros de dados, fazer *parsing* e validação dos códigos clientes e produtos.

```java
package model.reader;

import model.Cat.ICatVendas;
import model.Cat.ICat;

import java.io.IOException;

public interface IReader {
    void readVendas(ICatVendas vendas, String pathFile) throws IOException;
    void readCat(ICat cat, String pathFile) throws IOException;
    String[] readPath() throws IOException;
}
```

```java
package model.reader;

import Controller.GestVendas;
import view.IView;

public interface IObjectStream {
    void loadGestVendas(GestVendas c, IView view);
    void saveGestVendas(GestVendas c, IView view);
}
```

Figure 7: *Classes IReader* e *IObjectStream*

2.2 *View*

Inserido no *package* ***view***, Figura 8, temos a *classe IView* sendo que esta tem como funções principais, realizar os *prints* dos resultados das *queries* e dos menus assim como funções que permitem a interacção do utilizar com a aplicação.

```java
import model.data.Par;

import java.util.List;
import java.util.Map;

public interface IView {

    void printNewLines();
    void printSpaces();
    void printInputError();
    void proceed();
    void proceed(String message);

    void printEstatisticas(String[] config, int[] estaticas, double faturacao);
    void printMonthlyPurchases(int[] compras);
    void printFaturacaoPerMonthPerFilial(double[][] faturacao);
    void printDifClientsPerMonthPerFilial(int[][] clientes);

    void printQuery1(List<String> a);
    void printQuery2(List<Par<Integer,Integer>> a);
    void printQuery3(List<Par<Integer, Par<Integer,Double>>> a);
    void printQuery4(List<Par<Integer, Par<Integer,Double>>> a);
    void printQuery5(Map<String, Integer> a);
    void printQuery6(Map<String, Par<Integer,Integer>> a);
    void printQuery7(List<Map<String, Double>> a);
    void printQuery8(Map<String, Integer> a);
    void printQuery9(Map<String, Par<Integer,Double>> a);
    void printQuery10(List<double[]> a);
    void printQueryTime(String time);

    int getCommand(int min, int max);
    int getInt(int min, int max);
    int getMes();
    String getCodCli();
    String getCodProd();
    void printMenu();
    void printClose();
    String printCommand(String say);
}
```

Figure 8: *Classe IView*

A Figura 9, mostra o resulta obtido pelo utilizador quando escolhe a *Query 7*.

```
QUERY 7 :

Filial 1
T4720 : 2353597.9 €
F4737 : 2107203.6700000004 €
O1488 : 2077077.1799999997 €

Filial 2
Z1848 : 2173068.69 €
Y1687 : 2122078.94 €
R2459 : 2089734.6199999996 €

Filial 3
R2722 : 2184829.85 €
K3556 : 2132755.4999999995 €
U3268 : 2129578.62 €

Elapsed time : 0.1776001 sec
Continue by pressing Enter..
```

Figure 9: Resultado da *Query 7*

2.3 *Controller*

O *packge* ***Controller*** que por sua vez contém a *classe IGestVendas*, Figura 10, é responsável pela correta ordem de execução dos pedidos do utilizador à aplicação.

```java
package Controller;

import model.main.IModel;
import view.IView;

import java.io.IOException;

public interface IGestVendas {
    void setModel(IModel model);
    void setView(IView view);
    void start() throws IOException;
}
```

Figure 10: *Classe IGestVendas*

3 Testes de Leitura e Performance

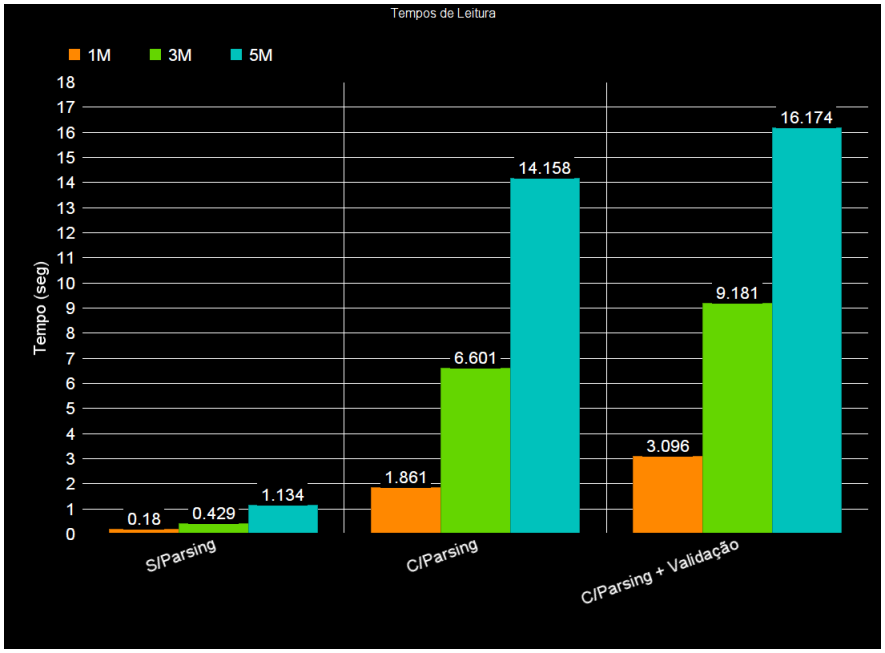

Figure 11: Resultados dos tempos de leitura

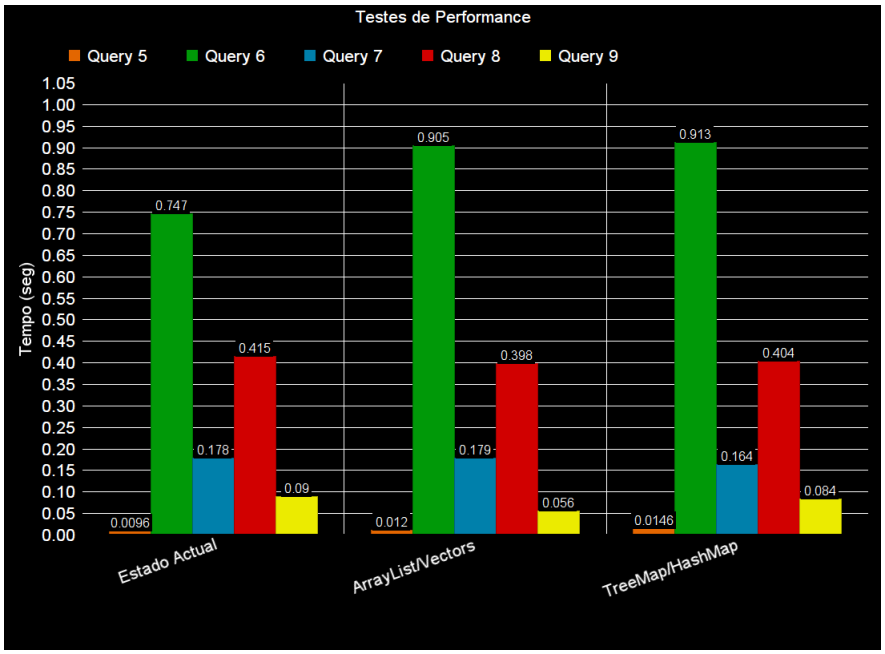

Figure 12: Resultados dos testes de performance

4 Conclusões

O objectivo deste trabalho passou pela implementação de um sistema de gestão de vendas de hipermercado, cujos dados recolhidos encontram-se em ficheiros texto relativos aos clientes, produtos e vendas dos referidos hipermercados. Neste trabalho foi desenvolvido um sistema que incorpora modularidade e encapsulamento de dados, sendo depois possível obter resultados perante determinadas *queries*. Os desafios do problema passam pela gestão de grandes volumes de dados, com vista à implementação de uma solução optimizada e escalável, atendendo à ainda maior quantidade de dados obtidos num contexto real de hipermercados.

Os desafios encontrados ao longo do trabalho incluíram armazenar, aceder, gerir e retirar eficientemente uma grande quantidade de dados.

Adicionalmente, o trabalho desenvolvido consolidou conhecimentos adquiridos ao longo do curso, nomeadamente nas unidades curriculares de Programação Orientada a Objectos. Nesta segunda fase do projecto e ao contrário da primeira, foi obtido sucesso no desenvolvimento de uma resposta a praticamente todas as *queries* o desenvolvimento deste trabalho fomentou o interesse relativo a estrutura e organização de dados, dando uma noção de contexto prático a engenharia informática.

Bibliografia

[1] F. Mário Martins, *PROJETOS DE POO EM JAVA*. 2014.

[2] F. Mário Martins, *JAVA 8 - POO + CONSTRUÇÕES FUNCIONAIS*. 2017.