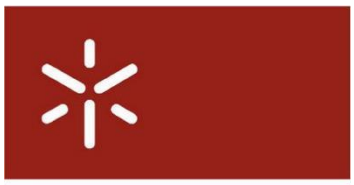


Universidade do Minho

Universidade do Minho

Relatório do Trabalho Prático LI3

Mestrado Integrado em Engenharia Informática

Grupo 31

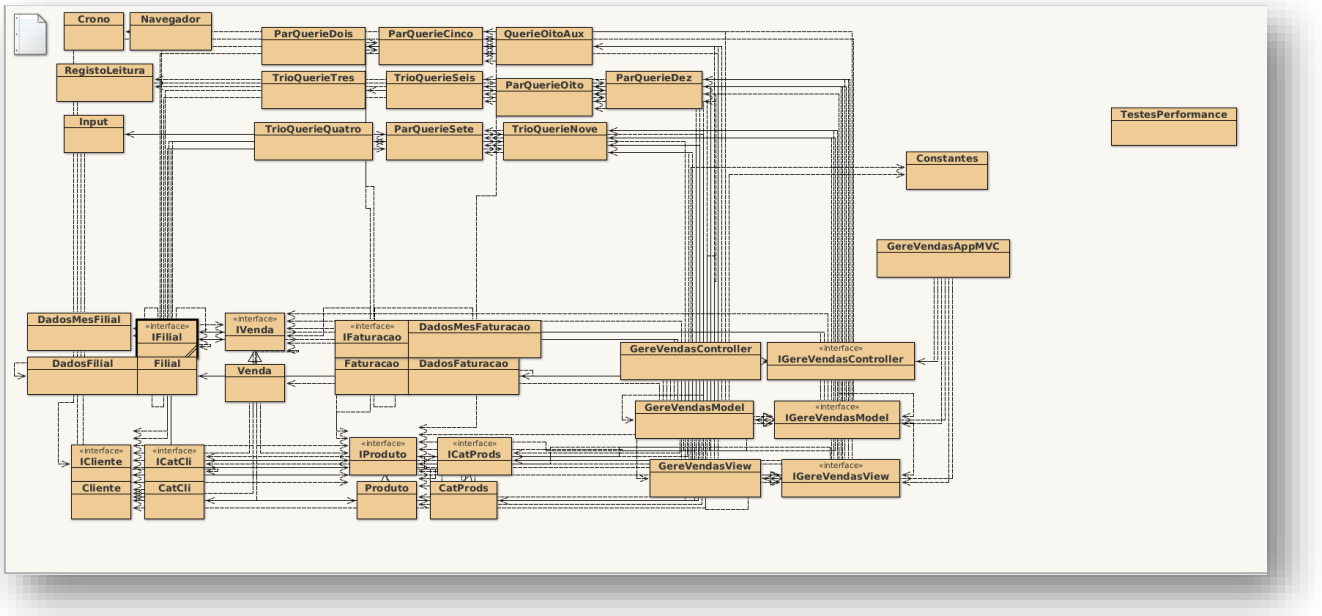
André Morais	A83899
Luís Ribeiro	A85954
Pedro Rodrigues	A84783

Introdução

O trabalho realizado foi desenvolvido em linguagem Java e teve como principal objetivo a gestão das vendas de uma distribuidora com 3 filiais. Para a concretização do objetivo, utilizamos ficheiros ".txt" com informação útil sobre clientes, vendas e produtos.

Para facilitar a extração de informação dos ficheiros criamos algumas estruturas de dados.

O objetivo deste relatório é justificar as escolhas dos tipos de dados, estruturas de armazenamento e também explicar as estratégias usadas para a resposta das questões propostas.



Esquema das nossas classes

Classes

Foram criadas várias classes de forma a garantir o encapsulamento. Segue-se uma breve descrição de cada uma dessas classes e estruturas de dados:

Produto & Clientes & Vendas

As classes correspondentes aos Produtos, Clientes e Vendas, respetivamente, que permitem fazer algumas operações sobre estes.

- A classe Produto implementa a seguinte interface:

```
/**
 * Interface correspondente ao Produto que permite fazer
 * algumas operaçoes sobre os produtos
 *
 * @author Grupo31
 * @version 2019
 */

public interface IProduto
{
    public String getCodigo ();
    public void setCodigo (String cod);
    public boolean equals (Object o);
    public int hashCode ();
    public int compareTo (IProduto p);
    public String toString ();
    public IProduto clone ();
}
```

Interface IProduto

- A classe Cliente implementa a seguinte interface:

```
/**
 * Interface correspondente ao Cliente que permite fazer
 * algumas operaçoes sobre os clientes
 *
 * @author Grupo31
 * @version 2019
 */
public interface ICliente extends Serializable
{
    public String getCodigo ();
    public void setCodigo (String cod);
    public boolean equals (Object o);
    public String toString ();
    public ICliente clone ();
    public int hashCode ();
    public int compareTo (ICliente p);
}
```

Interface ICliente

- A classe Venda implementa a seguinte interface:

```java
/**
 * Interface correspondente ao Venda que permite fazer
 * algumas operaçoes sobre as vendas
 *
 * @author Grupo31
 * @version 2019
 */
public interface IVenda
{
    public IProduto getProduto ();
    public double getPreco ();
    public int getQuantidade ();
    public char getTipo ();
    public ICliente getCliente ();
    public int getMes ();
    public int getFilial ();
    public void setProduto (IProduto p);
    public void setPreco (double p);
    public void setQuantidade (int quant);
    public void setTipo (char tipo);
    public void setCliente (ICliente c);
    public void setMes (int mes);
    public String toString();
    public boolean equals(Object o);
    public IVenda clone ();
}
```

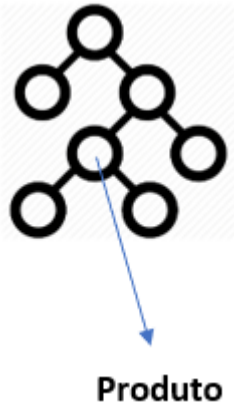
Interface IVenda

CatProds

Classe correspondente ao catálogo de produtos que permite fazer algumas operações sobre o catálogo.
Optamos pela utilização de um set, mais concretamente um *TreeSet*, pois na *Query* 1 será necessário colocar todos os produtos por ordem alfabética, e assim, para poupar tempo na *Query* 1, é necessário que estes produtos sejam guardados por uma ordem, assim, guarda-se os produtos numa *TreeSet* para que seja garantida uma ordem entre eles~

- A classe CatProds implementa a seguinte interface:

```java
/**
 * Interface correspondente ao catalogo de produtos que permite
 * fazer algumas operaçoes sobre o catalago
 *
 * @author Grupo31
 * @version 03/06/2019
 */
public interface ICatProds
{
    public Set <IProduto> getProdutos ();
    public void setProdutos (Set <IProduto> catp);
    public ICatProds clone ();
    public boolean existeProduto (IProduto p);
    public void insereProduto (IProduto p);
    public int numeroProdutos ();
}
```



Interface ICatProds

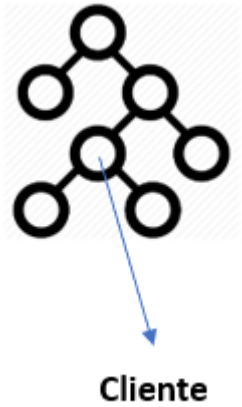
CatCli

Classe correspondente ao catálogo de Clientes que permite fazer algumas operações sobre o catálogo.
Aqui optamos por uma *TreeSet*, apesar de não ser necessário manter nenhuma ordem, por isso, provavelmente seria apenas necessário a utilização de um *HashSet*, mas a implementação desta classe permite alterar esta estrutura com muita facilidade, por isso, podemos fazer testes de performance muito facilmente com estas duas implementações de Set.

- A classe CatCli implementa a seguinte interface:

```
/**
 * Interface correspondente ao catalogo de clientes, que permite
 * fazer algumas operaçoes sobre o catalago
 *
 * @author Grupo31
 * @version 2019
 */
public interface ICatCli
{
    public Set <ICliente> getClientes();
    public void setClientes (Set <ICliente> clientes);
    public ICatCli clone ();
    public boolean existeCliente (ICliente c);
    public void insereCliente (ICliente c);
    public int numeroClientes ();
}
```

Interface ICatCli



Faturação

Classe que representa a faturação e as operações que se podem realizar sobre ela.
A organização da faturação foi idêntica ao trabalho em C, devido ao pedido das Queries, as vendas foram organizadas por meses, tendo sido criadas as classes auxiliares *DadosMesFaturacao* e *DadosFaturacao* que correspondem à organização das vendas num mês e os dados guardados de uma venda, respetivamente. Assim, na *DadosMesFaturacao*, está guardado um *Map* onde contém todas as vendas (*Value*- lista com os *DadosFaturacao*) para um determinado produto (*Key*) naquele exato mês.

```
public class Faturacao implements IFaturacao, Serializable
{
    private List<DadosMesFaturacao> faturacao;
```

```
public class DadosMesFaturacao implements Serializable
{
    private Map <IProduto,List<DadosFaturacao>> fat;
```

```
public class DadosFaturacao implements Serializable
{
    private int unidades;
    private double preco;
    private int filial;
```

- A classe Faturação implementa a seguinte interface:

```java
/**
 * Interface que representa a faturacao e as operaçoes que
 * se podem realizar sobre ela
 *
 * @author Grupo31
 * @version 2019
 */
public interface IFaturacao
{
    public void insereVenda (IVenda v);
    public IFaturacao clone ();
    public List<DadosMesFaturacao> getFaturacao ();
    public boolean produtoComprou(IProduto p);
    public int vendasMesProduto (IProduto p, int mes);
    public double faturadoTotalProdutoMes (IProduto p, int mes);
    public List<ParQuerieCinco> maisCompradosAno (int x);
    public int produtosCompradosDist ();
    public double faturacaoTotal ();
    public int comprasGratis();
    public List<String> getTotalVendasMes ();
}
```

Interface IFaturacao

Filial

Classe que representa uma Filial e as operações que se podem realizar sobre ela.
Mais uma vez, a organização da Filial foi idêntica ao trabalho em C, foram organizadas por meses, tal como a faturação, e foram também criadas as classes auxiliares *DadosMesFilial* e *DadosFilial* que correspondem ao mesmo que correspondiam na Faturação mas adaptada à Filial. Assim, na *DadosMesFilial*, está guardado o mesmo *Map* que tem a *DadosMesFaturacao*, mas neste caso, a *Key* corresponde ao cliente, para que as vendas sejam organizadas pelos clientes, e o *Value* será uma lista com os *DadosFilial*.

```java
public class Filial implements IFilial, Serializable
{
    private List <DadosMesFilial> filial;
```

```java
public class DadosMesFilial implements Serializable
{
    private Map<ICliente,List<DadosFilial>> fil;
```

```java
public class DadosFilial implements Serializable
{
    private IProduto produto;
    private char tipo;
    private int unidades;
    private double preco;
```

- A classe Filial implementa a seguinte interface:

```java
/**
 * Interface que representa uma filial e as operaçoes que
 * se podem realizar sobre ela
 *
 * @author Grupo31
 * @version 2019
 */
public interface IFilial
{
    public void setFilial (List <DadosMesFilial> fil);
    public List <DadosMesFilial> getFilial ();
    public void insereVenda (IVenda v);
    public IFilial clone ();
    public List <ICliente> clientesDiferentesMes (int mes);
    public int totalVendasMes (int mes);
    public int comprasMesCliente (ICliente c, int mes);
    public List<IProduto> produtosClienteComprouMes (ICliente c, int mes);
    public double gastoTotalClienteMes (ICliente c, int mes);
    public List<ICliente> clientesCompraramProdMes (IProduto p, int mes);
    public List<DadosFilial> prodsMaisCompradosCli (ICliente c);
    public List<ICliente> clientesDist (IProduto p);
    public List<ParQuerieSete> maioresCompradores ();
    public List<QuerieOitoAux> maisCompraramDif();
    public List<TrioQuerieNove> clientesMaisCompraram(IProduto p);
    public List<List<ParQuerieDez>> faturacaoTotalProduto ();
    public List<String> todosClientes ();
    public List<String> totalFaturadoMeses();
    public List<String> clientesDistMeses ();
}
```

Interface IFilial

MVC

- **MODEL**

Classe que representa o *Model* no modelo MVC e contem toda a "base de dados" da aplicação e as operações sobre ela.

```java
/**
 * Classe que representa o Model no model MVC e
 * contem toda a "base de dados" da aplicacao e as operaçoes sobre ela
 *
 * @author Grupo31
 * @version 2019
 */
public class GereVendasModel implements IGereVendasModel, Serializable
{
    private ICatCli catcli;
    private ICatProds catprods;
    private IFaturacao fatp;
    private List<IFilial> filiais;
    private RegistoLeitura data;
```

```java
/**
 * Classe que reprenseta um registo de leitura de ficheiros
 *
 * @author Grupo31
 * @version 2019
 */
public class RegistoLeitura implements Serializable
{
    private String nomeFich;
    private int vendasErradas;
```

Esta classe contem todos os métodos necessários para a realização de todas as Queries.

- A classe Model implementa a seguinte interface:

```
/**
 * Interface que representa o Model no model MVC e
 * contem toda a "base de dados" da aplicacao e as operaçoes sobre ela
 *
 * @author Grupo31
 * @version 2019
 */
public interface IGereVendasModel
{
    public ICatCli getCatCli ();
    public ICatProds getCatProds ();
    public IFaturacao getFaturacao ();
    public List<IFilial> getFiliais ();
    public RegistoLeitura getRegistoLeitura ();
    public void setCatCli (ICatCli catc);
    public void setCatProds (ICatProds catp);
    public void setFaturacao (IFaturacao fatp);
    public void setFiliais (List<IFilial> fil);
    public void setRegistoLeitura (RegistoLeitura d);
    public IGereVendasModel clone();
    public void createData();
    public TreeSet<IProduto> querie1 ();
    public ParQuerieDois [] querie2 (int mes);
    public TrioQuerieTres [] querie3 (ICliente c);
    public void lerProdutos (String nomeFich);
    public void lerClientes (String nomeFich);
    public void lerVendas (String nomeFich);
    public IVenda linhaToVenda (String linha);
    public TrioQuerieQuatro [] querie4 (IProduto p);
    public List<ParQuerieCinco> querie5 (ICliente c);
    public List<TrioQuerieSeis> querie6 (int x);
    public List<List<ParQuerieSete>> querie7 ();
    public List<ParQuerieOito> querie8(int n);
    public List<TrioQuerieNove> querie9(IProduto p, int x);
    public List<List<List<ParQuerieDez>>> querie10(String p);
    public List<String> querie11 ();
    public List<String> querie121 ();
    public List<String> querie122 (int filial);
    public List<String> querie123(int filial);
    public void guardaEstado ();
    public IGereVendasModel carregarEstado ();
}
```

Interface IGereVendasModel

- **VIEW**

Classe correspondente ao *View* do modelo MVC, e responsável por todo o IO do sistema.

- A classe View implementa a seguinte interface

```
 * Interface correspondente ao View do modelo MVC, e responsavel
 * por todo o IO do sistema
 *
 * @author Grupo31
 * @version 2019
 */

public interface IGereVendasView
{
    public IGereVendasView clone ();
    public Navegador getNav();
    public void setNav (Navegador n);
    public int menuInicial ();
    public void querie1 (TreeSet <IProduto> prods);
    public int menu2 ();
    public void querie2 (ParQuerieDois [] lista);
    public void querie3 (TrioQuerieTres [] lista);
    public String menu3 ();
    public void clienteNaoExiste();
    public void produtoNaoExiste();
    public String menu4 ();
    public void querie4 (TrioQuerieQuatro [] lista);
    public String menu5 ();
    public void querie5 (List<ParQuerieCinco> lista);
    public int menu6 ();
    public void querie6 (List<TrioQuerieSeis> lista);
    public void querie7 (List<List<ParQuerieSete>> lista);
    public int menu8 ();
    public void querie8 (List<ParQuerieOito> lista);
    public String [] menu9 ();
    public void querie9 (List<TrioQuerieNove> lista);
    public String menu10 ();
    public void querie10 (List<List<List<ParQuerieDez>>> lista);
    public void querie11 (List<String> l);
    public int menu12 ();
    public void querie121 (List<String> l);
    public int menu122 ();
    public void querie122 (List<String> l);
    public void querie123 (List<String> l);
    public int lerVendasMenu();
    public int lerFicheirosMenu ();
}
```

Interface IGereVendasView

Esta classe contém ainda o navegador, fundamental para a apresentação dos resultados.

```java
/**
 * Classe que torna possivel a navegaçao com sistema de paginas no ecra.
 *
 * @author Grupo31
 * @version 2019
 */
public class Navegador
{
    private int numeroLinhas;
    private int numeroColunas;
```

- **CONTROLLER**

Classe correspondente ao Controlador no modelo MVC que gere as operações da aplicação.

```java
public interface IGereVendasController
{
    /**
     * Funcao que altera o model da app.
     *
     * @param model Novo model da app
     */
    public void setModel(IGereVendasModel model);

    /**
     * Funcao que altera a view da app.
     *
     * @param view Nova view da app
     */
    public void setView(IGereVendasView view);

    /**
     * Funcao que inicia o controlador.
     *
     */
    public void start ();
}
```

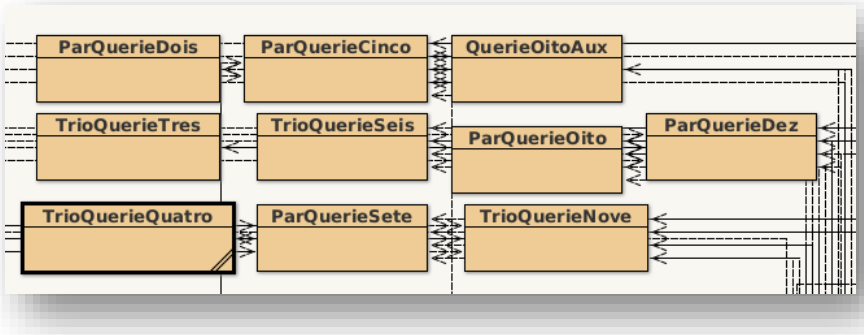
GereVendasAppMVC

GereVendasAppMVC é a classe principal que contém apenas a função *main*, onde inicia a base de dados e o modelo MVC:

```
public class GereVendasAppMVC implements Serializable
{
    public static void main(String [] args)
    {
        IGereVendasModel model = new GereVendasModel();
        model.createData();
        if (model==null){
            out.println("ERRO...");
            System.exit(-1);
        }
        IGereVendasView view = new GereVendasView(); //menu principal, alguns dialogos e navegador
        IGereVendasController control = new GereVendasController();
        control.setModel(model);
        control.setView(view);
        control.start();
        System.exit(0);
    }
}
```

Classes Auxiliares

Foram criadas também algumas classes auxiliares para facilitar a realização das *queries* e a comunicação entre *Model-View-Controller*.



Exemplo de uma das Classes:

```
/**
 * Classe auxiliar para a realizacao da querie 4
 *
 * @author Grupo31
 * @version 2019
 */
public class TrioQuerieQuatro implements Serializable
{
    private int vendas;
    private int clientes;
    private double faturado;
```

Constantes

Todas as contantes do sistema são guardadas numa classe à parte:

```java
/**
 * Classe que contem as constantes necessarias para o funcionamento da app.
 *
 * @author Grupo31
 * @version 2019
 */
public class Constantes
{
    private Constantes (){};

    public static final int MESES = 12;
    public static int FILIAIS = 3;
    public static final String OBJECT_STREAM_FILE_NAME = "gestVendas.dat";
    public static final String CONFIGS_FILE_NAME = "configs.txt";
    public static final int CONFIG_FILE_LINES = 4;
    public static int NUMERO_LINHAS_NAVEGADOR = 10;
    public static int NUMERO_COLUNAS_NAVEGADOR = 10;
}
```

Aspetos Relevantes

- **Leitura de Ficheiros**
  - **Vendas 1M:**

```
[Leitura do ficheiro de produtos]: 0.077415828 s

[Leitura do ficheiro de clientes]: 0.011409519 s

[Leitura do ficheiro de vendas]: 3.29598837 s

[Tempo total de leitura de ficheiros]: 3.29604656 s
```

Leitura de ficheiro em Java

```
Tempo total:5.504957 s
```

Leitura de ficheiro em C

  - **Vendas 3M:**

```
[Leitura do ficheiro de produtos]: 0.119949079 s

[Leitura do ficheiro de clientes]: 0.015314029 s

[Leitura do ficheiro de vendas]: 17.526932698 s

[Tempo total de leitura de ficheiros]: 17.527025728 s
```

Leitura de ficheiro em Java

```
Tempo total:30.406437 s
```

Leitura de ficheiro em C

- **Vendas 5M:**

```
[Leitura do ficheiro de produtos]: 0.104994944 s

[Leitura do ficheiro de clientes]: 0.015896633 s

[Leitura do ficheiro de vendas]: 30.833644322 s

[Tempo total de leitura de ficheiros]: 30.833719573 s
```

Leitura de ficheiro em Java

```
Tempo total:44.157094 s
```

Leitura de ficheiro em C

- **Exemplo do nosso navegador:**

```
[Query 1]: 0.183075123 s

Numero elementos:928

Page: 1 of 10

AA1022 AA1143 AA1236 AB1432 AB1643 AC1285 AC1365 AC1874 AD1283 AD1865
AE1156 AE1280 AF1094 AF1620 AG1028 AG1065 AI1080 AI1123 AI1782 AJ1051
AJ1262 AK1433 AL1144 AL1344 AL1932 AN1192 AN1693 AN1791 AO1576 AO1629
AP1182 AR1875 AS1139 AT1875 AV1529 AV1603 AV1883 AV1927 AW1504 AW1880
AX1657 AY1352 AZ1243 BA1206 BA1939 BA1974 BB1352 BC1307 BC1416 BF1787
BG1375 BH1212 BI1159 BI1893 BL1746 BL1936 BM1938 BN1180 BP1868 BQ1191
BR1216 BS1760 BT1618 BU1669 BU1766 BX1583 BX1879 BY1837 CA1352 CB1532
CB1681 CC1817 CC1923 CE1238 CE1505 CE1884 CE1926 CF1508 CF1527 CF1546
CF1903 CG1616 CI1158 CK1436 CK1779 CL1149 CL1454 CL1761 CM1435 CN1464
CO1543 CP1026 CP1445 CQ1735 CR1134 CR1273 CR1500 CS1579 CS1697 CT1260

[P] Proxima Pagina
[A] Pagina Anterior
[B] Atras
```

Teste de Performance

- **Testes gerais** (procura de cliente e produto em *TreeSet*)

```
Teste de performance sem parsing de linhas:

                        BufferReader                ReadAllLines
[Vendas_1M.txt]:        0.59546516 s               0.391713694 s
[Vendas_3M.txt]:        0.80320111 s                0.601457846 s
[Vendas_5M.txt]:        1.921790031 s                2.499469948 s

Teste de performance com parsing de linhas:

                        BufferReader                ReadAllLines
[Vendas_1M.txt]:        1.420712568 s                1.25671012 s
[Vendas_3M.txt]:        4.504840164 s                5.588433701 s
[Vendas_5M.txt]:        7.288114262 s                15.862395132 s

Teste de performance com parsing e validacao de linhas:

                        BufferReader                ReadAllLines
[Vendas_1M.txt]:        1.327946333 s                1.701474519 s
[Vendas_3M.txt]:        5.123749296 s                4.563197758 s
[Vendas_5M.txt]:        7.822468549 s                7.725536602 s
```

- **Testes com diferentes tipos de estruturas**

Conclusão

Concluído o trabalho, entendemos que a solução apresentada é eficiente e bem estruturada. Foram efetuadas várias medições de tempo, que comprovam a boa performance das abordagens seguidas nas várias *queries*. A nível de gestão de memória, houve o cuidado de dar *clone* em algumas estruturas, para garantir o encapsulamento.

Houve certamente uma aprendizagem notória relativamente a testes de performance e a diferentes métodos de leitura de ficheiros juntamente com o modelo MVC.

No geral faz-se uma avaliação muito positiva das diversas métricas consideradas e entende-se que a abordagem ao problema foi concisa e direta.