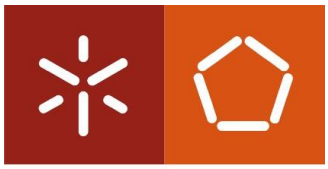

Universidade do Minho

Projeto de Java - Gestão das Vendas de uma Cadeia de Distribuição

Grupo nº92

Leonardo de Freitas Marreiros (a89537)

Conteúdo

1. Introdução

O objetivo fundamental deste projeto consistiu em elaborar um sistema de gestão básica de uma cadeia de distribuição capaz de ler e realizar consultas interativas baseadas em ficheiros. Para isto foi necessário aplicar conhecimentos de interfaces e coleções de forma crítica tendo em vista a melhor eficiência e resultados; e também a compreensão e entendimento de conceitos de encapsulamento e abstração dos dados. Além disso, foi posto em prática novamente a noção de arquitetura MVC (Model View Controller) de forma a implementar a modularidade.

2. Arquitetura

Foram criadas 3 Packages principais que dividem o projeto de forma coerente com a arquitetura MVC: Model, View, Controller (e Exceptions).

Model: Parte lógica da aplicação. Responsável por tudo que a aplicação vai fazer a partir dos comandos da camada de controle. Estruturas de dados.

View: Onde os dados solicitados do Model são exibidos. Não se dedica em saber como a informação foi retirada ou de onde ela foi obtida. Responsável pelo output das queries e pelo Navegador.

Controller: Faz a mediação da entrada e saída. Envia essas ações para o View onde serão realizadas as operações necessárias.

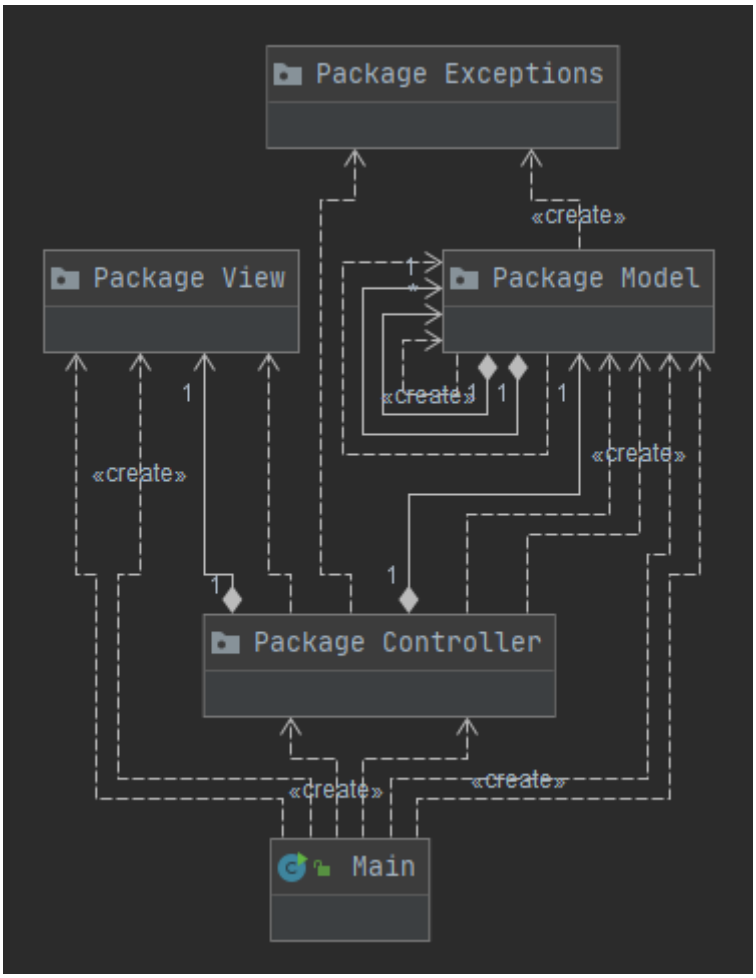

Figura 1 - Diagrama de Classes Geral

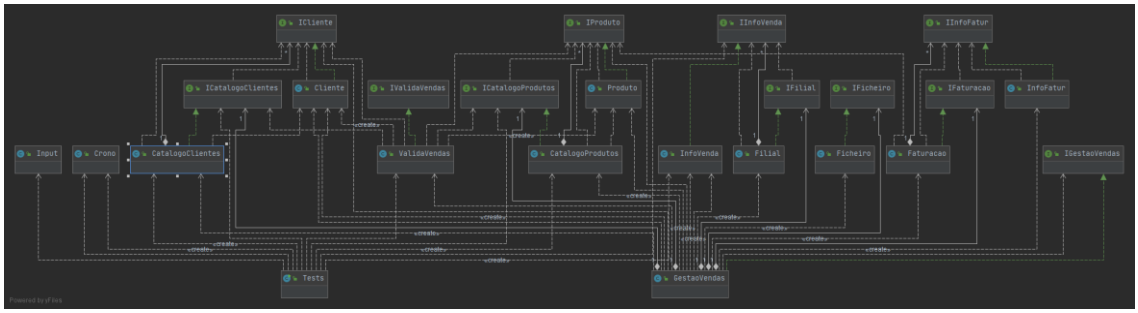

Figura 2- Diagrama de Classes do Model

2.1 Model

2.1.1 Cliente e Produto

Classes que lidam com cada cliente e produto individualmente. Apenas têm como variável de instância uma String que corresponde ao seu código.

2.1.2 CatalogoClientes e CatalogoProdutos

Classes que lidam com o conjunto de Clientes e Produtos. Foram utilizados HashSets para prevenir valores repetidos indesejados. Nestas classes são lidos os ficheiros correspondentes aos Clientes e Produtos.

2.1.3 InfoVenda

Classe que faz parsing de uma linha do ficheiro de vendas e guarda cada campo relativo a uma venda.

2.1.4 InfoFatur

Classe que faz parsing de uma linha do ficheiro de vendas e guarda cada campo relativo a informações de um produto.

2.1.5 ValidaVendas

Classe responsável por validar uma linha do ficheiro, isto é, valida uma venda. Verifica se o produto e cliente dessa linha existem no Catálogo assim como valida se cada um dos outros campos está dentro dos limites esperados.

2.1.6 Filial

Classe que, para cada Cliente válido, lhe associa a lista de vendas que efetuou: produtos comprados, unidades, mês de compra, etc. Foi utilizado um HashMap em que a Key é um código de Cliente e o Value é a lista de vendas que esse Cliente efetuou (List<InfoVenda>).

2.1.7 Faturação

Classe que, para cada Produto do Catálogo, lhe associa uma lista de informações acerca da sua faturação, isto é, número de unidades/ faturação num certo mês/filial. Produtos com listas vazias equivalem a Produtos que não foram comprados. Foi utilizado um HashMap em que a Key é o código de um Produto e o Value é uma lista de faturas (List<InfoFatur>).

2.1.8 Ficheiro

Classe que guarda as informações gerais dos ficheiros lidos: número de produtos, clientes, linhas lidas, lidas válidas, clientes por mês, etc.

2.1.9 GestaoVendas

Classe que engloba todos os outros módulos. É constituído pelas seguintes estruturas: um Catálogo de Clientes, um Catálogo de Produtos, uma Faturação, três Filiais e um Ficheiro. É neste módulo que é feita a leitura e preenchimento destas estruturas, e também a realização das queries pretendidas.

2.2 View

2.2.1 View

Este módulo é responsável pelo output dos dados: textos, tabelas, listas, etc. Inclui funções de apresentação das queries, menus, e o Navegador de listas.

2.3 Controller

2.3.1 Controller

Módulo que faz a ligação entre o Model e o View. Pede inputs e realiza ações baseadas nesses inputs.

2.4 Exceptions

2.4.1 InvalidProductException

Módulo desenvolvido para marcar a presença de um Produto inexistente.

2.4.2 InvalidClientException

Módulo desenvolvido para marcar a presença de um Produto inexistente.

2.5 Pontos Negativos

Tenho consciência que o projeto desenvolvido tem algumas limitações ou pontos que podiam ter sido melhorados:

- Procura de Produto Específico:

Seja esta a operação talvez mais demorosa do projeto, é fácil perceber o porquê de isto acontecer: imaginemos que apenas o último cliente da estrutura Filial comprou um produto específico, ora, para o encontrar teríamos de passar por todos os clientes e respetivas listas de vendas, apenas para o encontrar no fim da estrutura. Como é óbvio, a complexidade desta operação é bastante elevada. Sendo assim, podia ter sido feita uma estrutura diferente, TreeSets em vez de Lists por exemplo, ou um segundo HashMap com o código do produto na Key em vez do código do cliente.

- Apresentação Query 10:

Apesar de a Query10 executar em tempo aceitável, o problema nesta query foi a apresentação da informação. Uma vez que a quantidade de informação é demasiada para caber numa lista de Strings que seria depois enviada para o Navegador, foi necessário encontrar uma solução diferente. Decidi dividir os Produtos em Grupos e dar ao utilizador a opção de escolher em que grupo estava situado o Produto que pretendia procurar, depois, com esta informação era apresentada a subList da Lista de resultados da query10 que englobava esses produtos. No entanto, a utilização do método subList(x,y) para valores elevados de x e y revelou-se ser bastante dispendiosa em termos de tempo.

- Returns:

Alguns Returns do tipo List<Map.Entry<….>> poderiam ser trocados para estruturas do tipo Hash/TreeMaps.

- Navegador:

O Navegador poderia ter sido desenvolvido como um módulo independente.

2.6 Pontos Positivos

Da mesma forma, creio que o projeto também tem aspetos que são de valorizar:

- Tempo de leitura dos ficheiros:

Depois de ter realizado os diferentes testes de leitura dos ficheiros, creio que o resultado final com a utilização de BufferedReader e parallelStream seja bastante eficiente.

- Exceptions e validação de inputs:

A criação de exceptions (cliente válido, produto válido) e constante verificação de inputs válidos (mês válido, filial válida, para as queries que pedem um limite, verifica se o limite introduzido é maior que 0, etc) cria um projeto fluído, de fácil utilização e sem casos particulares indesejados.

- Navegador:

O Navegador apresenta um conjunto de funcionalidades extremamente úteis como a escolha de página ou a procura por elemento que tornam a navegação de listas imensamente mais fácil. Além disso, o navegador também dá a opção ao utilizador de escolher o número de elementos por página.

- Apresentação dos dados de forma limpa e organizada:

A cuidada apresentação estética, tornam o trabalho organizado e fácil de utilizar.

3. Testes de Performance

Foram feitos diversos testes para medir o desempenho do programa. Começando pela leitura dos ficheiros, foi criado um módulo Tests que dado um ficheiro, mede os tempos de leitura; leitura e parse; leitura, parse e validação, utilizando Files e BuffuredReader. Obtemos assim as tabelas 1 e 2 onde é possível comprovar que o BufferedReader é de forma geral mais rápido que o Files. Em todas estas leituras, foi utilizado o parallelStream que se revelou ser extremamente mais rápido do que o anteriormente implementado Stream, pelo que foi aproveitado o paralelismo intrínseco da JVM8.

De seguida, utilizando a classe Crono, foram registados os tempos de execução das queries para os diferentes ficheiros, obtendo a Tabela 3. Após ter feito isto, foram testadas as implementações de diferentes coleções: TreeMaps ao invés de HashMaps (Tabela 4) e TreeSets ao invés de HashSets (Tabela 5). Com isto, foi possível verificar que em qualquer destas implementações o tempo de leitura revelou-se mais rápido com a utilização de HashMaps, principalmente comparando com a implementação de TreeMaps. Em relação aos tempos de execução das queries, as diferentes implementações revelaram demorar, de forma geral, um tempo análogo aos HashMaps.

	1 Milhão	3 Milhões	5 Milhões
Leitura	285	824	1339
Leitura + parsing	1428	3758	6437
Leitura + parsing + validação	1607	5067	11799

Tabela 1: Tempo em ms com Files

	1 Milhão	3 Milhões	5 Milhões
Leitura	233	735	1639
Leitura + parsing	1125	3355	5901
Leitura + parsing + validação	1579	4392	12587

Tabela 2: Tempo em ms com BufferedReader

	1 Milhão	3 Milhões	5 Milhões
Leitura	2849	9604	17153
Query 1	9	10	10
Query 2	52	111	154
Query 3	0.248	0.543	0.166
Query 4	396	1144	1529
Query 5	0.377	0.755	0.807
Query 6	443	1142	1669
Query 7	37	58	73
Query 8	231	524	1081
Query 9	92	154	164
Query 10	104	193	210
Query E1	49	93	126
Query E2	128	258	376

Tabela 3: Tempo em ms das Queries

	1 Milhão	3 Milhões	5 Milhões
Leitura	3657	11576	20418
Query 1	11	12	9
Query 2	53	117	186
Query 3	0.428	0.616	0.707
Query 4	356	1166	2021
Query 5	0.408	0.903	0.848
Query 6	384	1111	1785
Query 7	46	65	88
Query 8	239	480	1133
Query 9	113	169	238
Query 10	210	165	258
Query E1	39	93	142
Query E2	179	374	623

Tabela 4: Tempo em ms das Queries com TreeMaps em vez de HashMaps

	1 Milhão	3 Milhões	5 Milhões
Leitura	3728	9012	17500
Query 1	6	10	11
Query 2	67	133	188
Query 3	0.642	0.520	0.685
Query 4	328	928	1511
Query 5	0.490	0.412	0.426
Query 6	402	1095	1680
Query 7	40	56	76
Query 8	260	970	1770
Query 9	135	140	173
Query 10	108	153	213
Query E1	41	87	123
Query E2	137	252	358

Tabela 5: Tempo em ms das Queries com TreeSets em vez de HashSets

4. Conclusão

Para concluir, gostaria de referir que, apesar dos Pontos Negativos referidos, o programa cumpre todos os requisitos propostos com um tempo de execução bastante satisfatório. Além disso, foram realizados todos os testes de desempenho apresentados o que promoveu a análise de diferentes soluções e escolha daquelas que acredito que sejam as melhores.

Finalmente, num trabalho futuro, consideraria a exploração de diferentes estruturas e organização de dados de forma a melhorar alguns dos aspetos menos positivos do projeto.