Gesthiper

Relatório do Trabalho Prático de C da Unidade Curricular de LI III

Grupo 11:

Gustavo da Costa Gomes (a72223)

José Carlos da Silva Brandão Gonçalves (a71223)

Tiago João Lopes Carvalhais (a70443)

Indice

Arquitetura de Classes

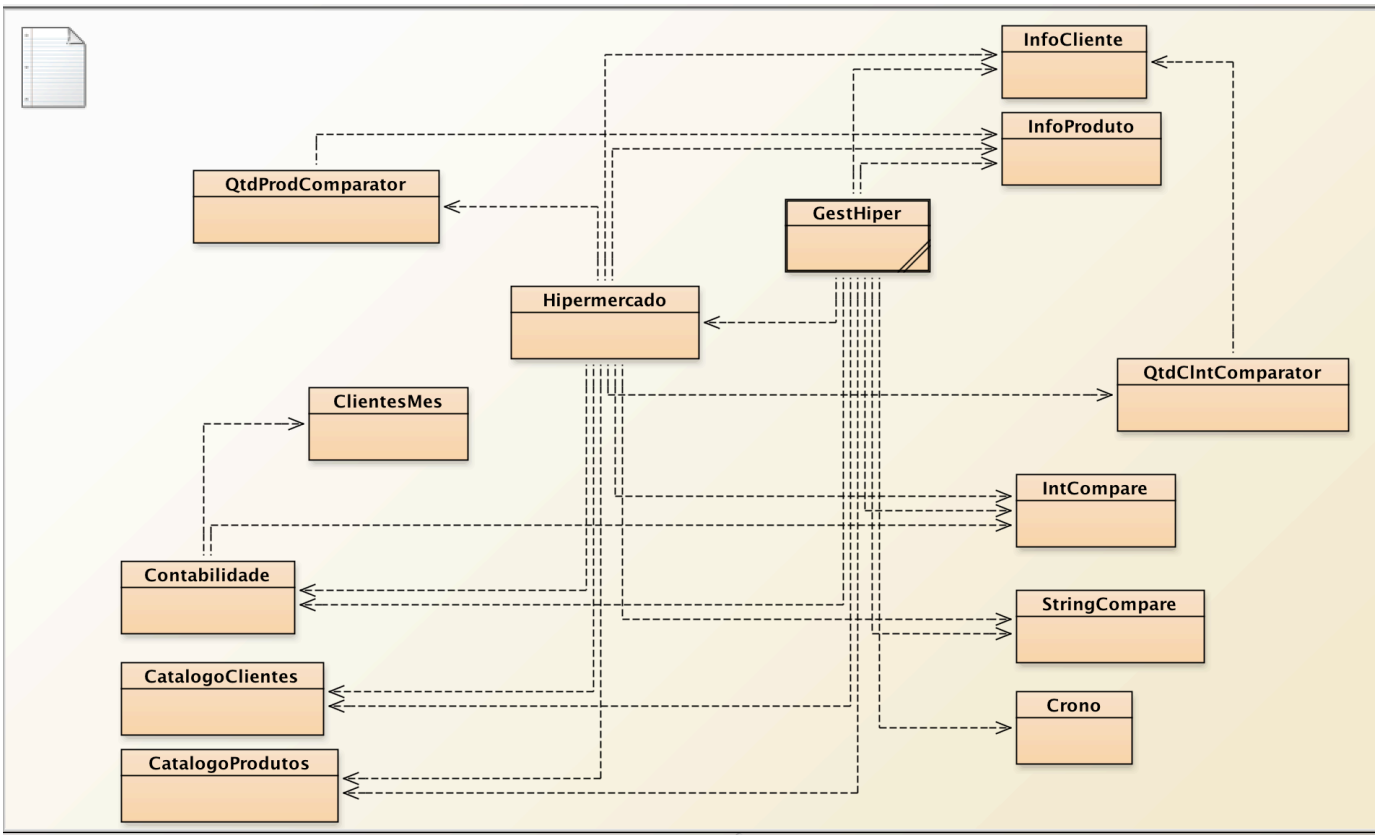


Figure 1

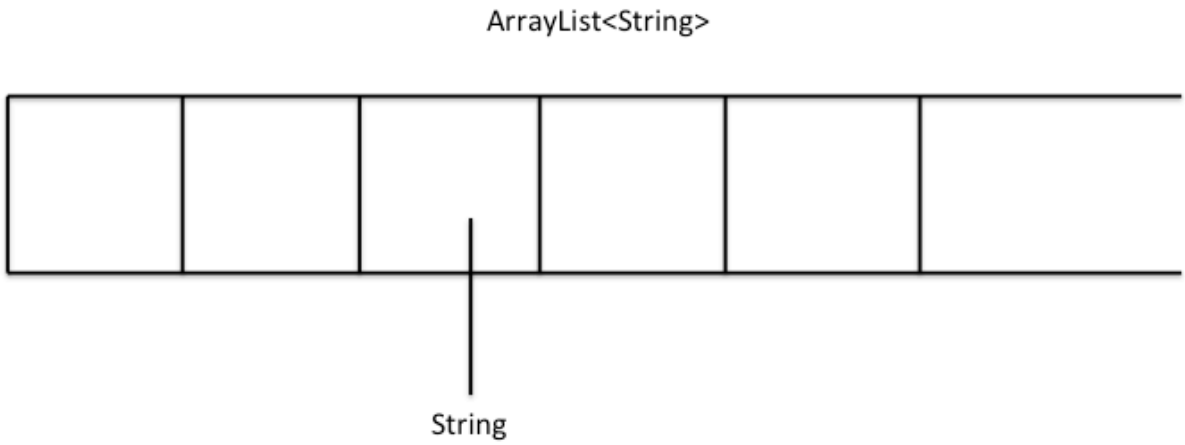
Na figura 1 pode ser observado a arquitetura de classes utilizadas completa no qual inclui classes de comparator de modo a certas estruturas estarem ordenadas consoante esses comparator.

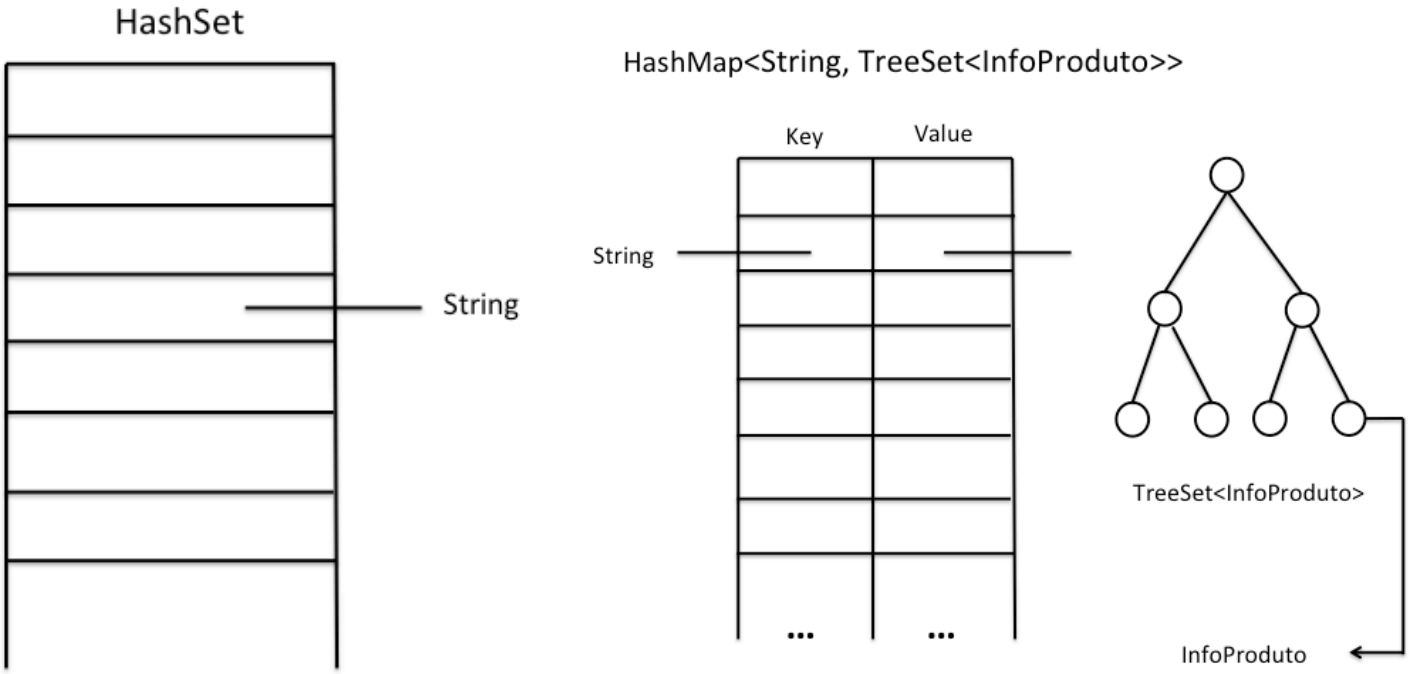
As classes principais são a Contabilidade, os catálogos (CatálogoClientes e CatálogoProdutos), Hipermercado(onde se faz a leitura dos .txt para as estruturas), GestHiper(contém o menu e a main bem como todas as queries) e as classes InfoCliente e InfoProduto que são classes que identificam o que é ser um produto e o que é ser um cliente.

Estruturas Utilizadas

As Estruturas que foram utilizadas para a realização do Projeto foram: HashMap<String, TreeSet<InfoProduto>>, ArrayList<String> e HashSet<String> .

As imagens seguintes demonstram um "desenho" do que estas estruturas são de um modo visual,





Agora vamos analisar ao pormenor cada classe tendo em conta as estruturas por ela usada e explicar o que faz cada classe.

1. Contabilidade

Esta classe é responsável por todas as queries relacionadas com a contabilização dos produtos vendidos, clientes que compraram por cada mês, entre outras.

Para tal são necessárias algumas variáveis:

```
private ArrayList<String> invalidComp;
private ArrayList<Integer> totalCompMes;
private ArrayList<Double> factMes;
private ArrayList<ClientesMes> clientesMes;
private double faturacao_total;
private int compras_gratis;
```

A variável clientesMes com a ajuda da classe analisada em 1.1 tem como objetivo guardar todos os clientes que fizeram compras com base no mês, ou seja, cada posição do ArrayList clienteMes refere-se a um mês e em cada uma dessas posições tem um ClientesMes que é a classe em 1.1 que guarda os códigos de cliente.

As variáveis faturação_total e compras_gratis tal como os nome sugerem são variáveis que contabilizam, com base no ficheiro de compras.txt lido, a faturação ganha no ano inteiro e o número de compras em modo grátis, respectivamente.

A variável factMes é responsável por guardar o valor da facturação ganha por cada mês com base no ficheiro de compras.

A variável totalCompMes guarda o número total de compras efectuados pelos clientes mensalmente.

E a variável invalidComp guarda todas as linhas do ficheiro compras lido que são inválidas, onde estas podem ser inválidas por várias razões, por exemplo, modos de compra sem ser os usuais(Normal ou Promoção).

1.1. ClientesMes

Esta classe o que faz é guardar na estrutura clientes todos os códigos de clientes que compraram algum produto por cada mês consoante o ficheiro de compras.txt utilizado. Esta é usada como classe auxiliar para que na classe Contabilidade se consiga então obter esses clientes por cada mês, como já foi mencionado em 1.

```
private HashSet<String> clientes;
```

2. CatalogoClientes

Esta classe é uma das mais importantes porque é nesta que se guardam os códigos de clientes em diferentes estrutura, ou seja, na variável catalogo_clientes guarda-se a informação de todos os códigos de cliente lidos do ficheiro clientes.txt facultado pelos docentes.

A estrutura clientes_compradores guardará todos os clientes que com base no ficheiro compras.txt compraram algum produto em algum mês do ano.

E a estrutura clientes_invalidos faz o mesmo que o clientes_compradores mas em vez de adicionar os que compraram, o que faz é comparar os clientes existentes com os clientes que compraram e guarda os clientes que não compraram nenhum produto.

```
private HashSet<String> catalogo_clientes;
private HashSet<String> clientes_compradores;
private HashSet<String> clientes_invalidos;
```

2.1. InfoCliente

As variáveis de instância desta classe são:

```
private String codigo;
private int quantidade;
private int mes;
private double preco_produto;
private String modo;
```

Com estas é possível descrever o comportamento que um cliente tem, por exemplo, um cliente é algo que contem um código(código de cliente) que é uma String, tem um integer que é a quantidade do produto que comprou, o integer mes que diz qual o mês em que comprou essa

quantidade desse produto, o preco_produto que é o preço total pago por essa quantidade e uma String modo que indica se a compra efectuada foi em modo Promoção (“P”) ou em modo Normal(“N”).

Esta classe contem 4 construtores, clone e os respectivos set e get definidos que servirão de auxilio noutras classes.

3. CatalogoProdutos

Esta classe é importante porque é nesta que se guardam os códigos de produtos em diferentes estruturas, ou seja, na variável catalogo_produtos guarda-se a informação de todos os códigos de produto lidos do ficheiro produtos.txt facultado pelos docentes.

A estrutura produtos_comprados guardará todos os produtos que com base no ficheiro compras.txt foram comprados em algum mês do ano e pelo menos por um cliente.

A estrutura produtos_invalidos guardará todos os produtos que não foram comprados em nenhum mês do ano e nem por nenhum cliente, com base no mesmo ficheiro compras.txt.

```
private HashSet<String> catalogo_produtos;
private HashSet<String> produtos_comprados;
private HashSet<String> produtos_invalidos;
```

3.1. InfoProduto

As variáveis de instância desta classe são:

```
private String codigo;
private int quantidade;
private int mes;
private double preco_produto;
private String modo;
```

Estas permitem caracterizar um produto e tal como referido em 2.1, servem como auxilio noutras classes.

A String codigo é o campo onde se guarda o código de produto, com uma certa quantidade comprada num determinado mês ao preço

(preco_produto) e no modo(Normal ou Promoção).

4. Hipermercado

É nesta classe que se faz a leitura dos ficheiros .txt e os insere nas suas estruturas. Também possui várias funções de auxilio para a resolução das queries usando a classe Contabilidade(1.) e possui funções de validação das linhas do ficheiro de compras.
Para tal acontecer são necessárias as seguintes variáveis:

```
private String compras_nome;
private CatalogoClientes catalogo_clientes;
private CatalogoProdutos catalogo_produtos;
private Contabilidade contabilidade;
private HashMap<String, TreeSet<InfoProduto>> produtosPorCliente;
private HashMap<String, TreeSet<InfoCliente>> clientesPorProduto;
```

Usando a função readCatalogos permite adicionar a partir dos ficheiros clientes.txt e produtos.txt os seus conteúdos nas respetivas estruturas usando para tal efeito classes BufferedReader e FileReader.

A outra função muito importante é a que permite ler o ficheiro de compras, readCompras, e que vai validando os valores. Caso não sejam válidos, adiciona a um ArrayList invalidComp(mencionado em 1.)

No caso de ser válido realiza as operações e insere nas estruturas produtosPorCliente e clientesPorProduto para posterior consulta na maior parte das queries realizadas.

5. GestHiper

Esta classe é a classe que contem a main que utiliza todas as outras classes e é a única com interação que permite executar os pedidos do utilizador e obter a resposta para aquele pedido.

Em certas *queries* houve a necessidade de implementar uma forma de navegação mais amigável ao utilizador, nomeadamente nas queries 1,2 e 7 e recorreu-se a uma função auxiliar chamada navigation.

A função navigation funciona da seguinte forma:

1. O programa pergunta ao utilizador quantos resultados este deseja ver em cada página;
2. De seguida, é calculado o intervalo, que consiste numa divisão inteira entre o número total de códigos a serem apresentados e o valor anteriormente escolhido pelo utilizador. Irá ser acrescentada uma página a esse intervalo, pois a última sabe-se que terá sempre N (valor dado pelo usuário) ou menos valores. Note-se que, no caso de o resto da divisão inteira do total de códigos por N ser zero, a última página não irá mostrar qualquer resultado;
3. Por fim, aparecem duas opções no ecrã: uma para sair e outra para escolher a página, onde aparece o número de páginas (de 1 até intervalo+1). No caso de o utilizador escolher uma página, irão aparecer, de imediato, no monitor os resultados referentes à mesma. Caso o usuário opte por sair, o programa sai da *querie*.

Testes de Performance

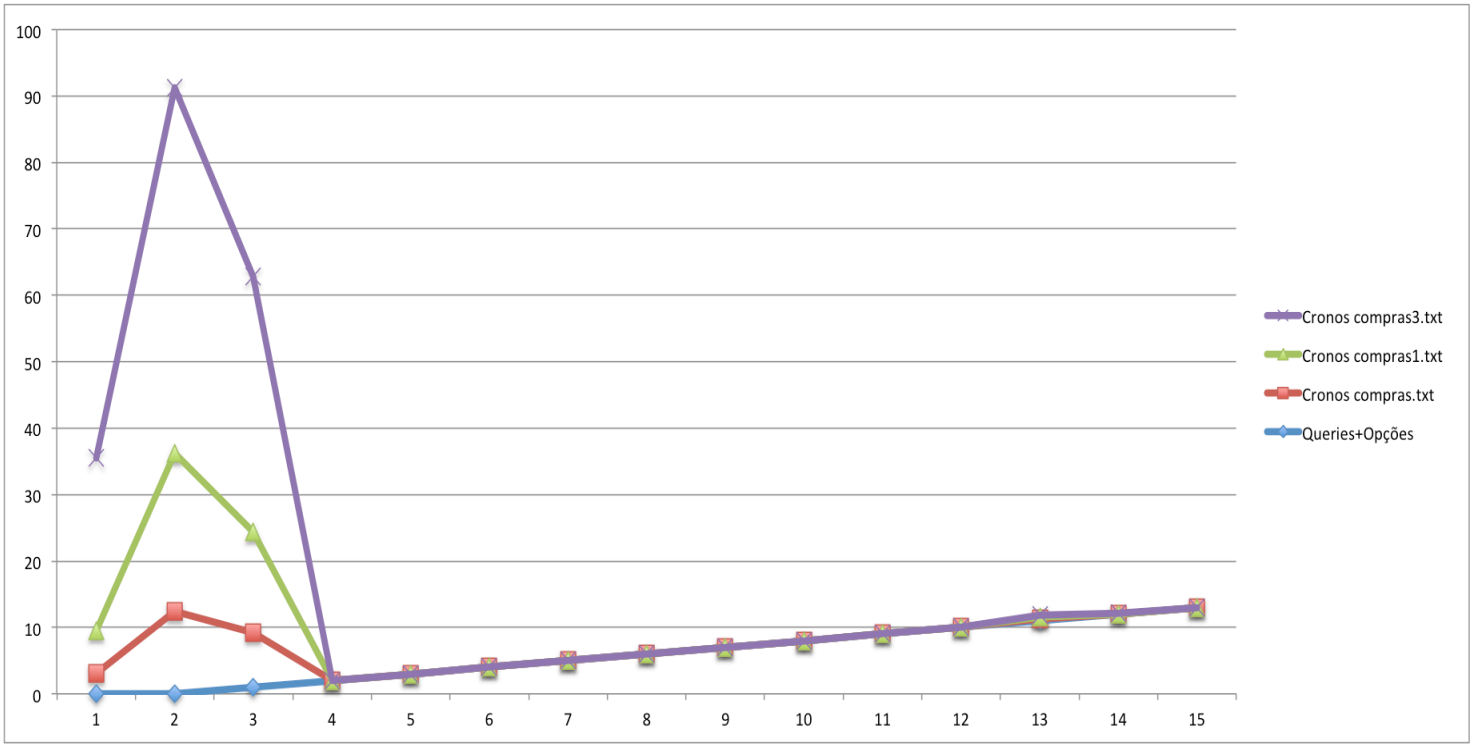


Figure 2

Neste capitulo analisar-se-á os resultados obtidos nas medições de tempos efectuados com os três ficheiros .txt que nos foram facultados para a realização do projeto.

Na figura 2, pode-se observar graficamente os resultados da nossa versão final, com as estruturas definidas anteriormente.

Na figura 3 observa-se os resultados obtidos após as modificações das estruturas, HashMap->TreeMap e HashSet->TreeSet.

Também irá ser abordado a questão da influência do parsing na leitura dos ficheiros .txt para as nossas estruturas apresentando os tempos obtidos com e sem parsing.

Pode ser observado na figura 2 que a opção mais custosa é de facto a opção 2 que diz respeito á leitura binária, isto é, quando estamos a carregar para a JVM os dados em binário que foram guardados anteriormente e para voltar a reestabelecer a JVM com esses. Ou seja, permite que as estruturas contenham as informações que outrora tiveram.

Essa operação é de facto a mais custosa de todas e verifica-se para todos os ficheiros .txt utilizados, mas também a operação inversa, guardar dados num ficheiro binário, representado pela opção 3 (na imagem), que diz respeito á query 1, que é a segunda operação mais custosa e a que também usa mais CPU tal como a carrega binário.

Todas as restantes queries são conseguidas em tempo quase instantâneo, isto é, todas abaixo de meio segundo e em alguns casos tão curtas que é necessário expoentes de base 10 na ordem de -4 e -5, por outras palavras, se não tivessem muitas casas decimais daria sempre 0,0 segundos.

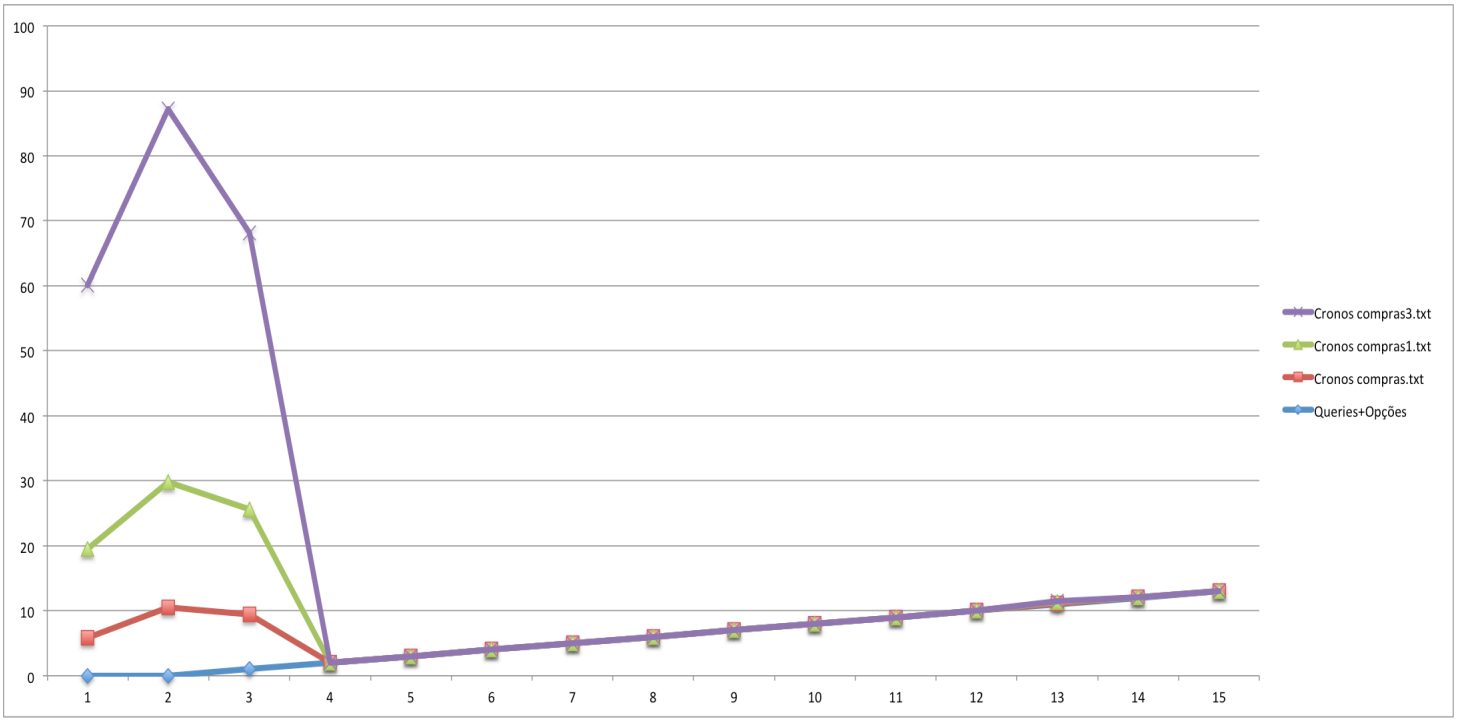


Figure 3

O gráfico da figura 3 mostra os tempos obtidos após efetuar as mudanças de estruturas tal como nos foi pedido na parte dos testes.

Podemos ver que apresenta diferenças mas nada de muito significativo na maior parte dos casos, mas como é óbvio existe algumas diferenças, por exemplo nas opções referentes á leitura normal dos .txt e á leitura dos dados em binário(1 e 2 na figura 3).

As mudanças efectuadas foram as seguintes:

Contabilidade:
Mudaram-se os quatro ArrayList para Vectores.

Catalogos:
Mudaram-se os três HashSet<String> para TreeSet<String>

ClientesMes:
Mudou –se HashSet<String> para TreeSet<String>

Hipermercado:
Mudou –se HashMap<String,TreeSet<InfoProduto>> para
TreeMap<String,TreeSet<InfoProduto>>
e

HashMap<String,TreeSet<InfoCliente>> para TreeMap<String,TreeSet<InfoCliente>>

Em relação aos tempos obtidos na leitura dos ficheiros .txt com e sem parsing, mais concretamente para o ficheiro compras, foram:

Compras.txt:

- Com parsing → 3,1584 segundos
- Sem parsing → 0,1913 segundos

Compras1.txt:

- Com parsing → 8,71379 segundos
- Sem parsing → 0,79149 segundos

Compras3.txt:

- Com parsing → 39,5824 segundos
- Sem parsing → 1,66324 segundos

Como se pode verificar pelos tempos anteriores a leitura sem parsing ocorre muito mais rápida e a melhor situação que o demonstra é para o ficheiro de 3 milhões onde a leitura é cerca de 23,8 vezes mais rápida que na versão com parsing.

Conclusão

Para finalizar, este projeto permitiu a consolidação da linguagem JAVA na construção de programas de larga escala e permitiu por comparação com o projeto anterior, muito semelhante mas noutra linguagem, ver as diferenças que existem na sua elaboração.

Concluiu-se que em JAVA pelo facto de já existirem as estruturas definidas o trabalho mais demorado que é o de implementar as estruturas não é necessário. Apenas a construção de classes com essas estruturas já definidas é que requer o tempo na elaboração e reduzido por comparação com o projeto anterior onde se teve de construir todas as estruturas.

Pelas razões anteriores observa-se que apesar de JAVA ser de implementação mais rápida também tem algumas contrapartidas e a principal é o facto de os tempos obtidos em algumas queries em comparação com o projeto em C observa-se que são ligeiramente mais lentas em alguns casos.

Também é de realçar que as operações de guardar e de leitura de binário são muito benéficas mas acarretam custos elevados para ficheiros muito grandes tal como referido anteriormente. No caso do ficheiro compras3.txt, com 3 milhões de linhas, observou-se que são necessários quase 4 minutos e ás vezes mais uns segundos para carregar o ficheiro binário de volta para a JVM e isso ainda é muito elevado e verificando-se também uma maior percentagem de CPU utilizado.