Problema 2: ImportaFácil– Importação e Transporte de Mercadorias de Alto Valor

Daniel Cavalcante Dourado

Engenharia de computação – Universidade Estadual de Feira de Santana (UEFS)

Av. Transnordestina, s/n - Novo Horizonte, Feira de Santana - BA, 44036-900

Feira de Santana – BA – Brasil

daniel10dourado@gmail.com

1.Introdução

Com a chegada do final de ano, as empresas mercantilistas estão se planejando para disponibilizar grandes descontos nos preços dos produtos, consequentemente as comprar irão aumentar exponencialmente.

Dadas às circunstancias previstas e, tendo em mente que a Black Friday, dia que inaugura a temporada de compras natalícias com significativas promoções em muitas lojas retalhistas e grandes armazéns, está muito próxima, a empresa, ImportaFácil, entrou em contato com os professores do MI- Algoritmos 2-2018.2, para solicitar aos estudantes um software capaz de contemplar todas as suas necessidades.

Devido a problemas judiciais, a antiga empresa responsável por realizar o software teve de encerrar criação do mesmo, os novos responsáveis pelo produto tiveram acesso ás informações/requisitos que o problema irá conter.

Por trabalhar com o recebimento e distribuição de mercadorias importantes, vindas da China, Coréia do Sul e EUA, a empresa tem como prioridade para o software os seguintes requisitos: recebimento de itens, cadastrando informações específicas de cada produto, manipulação dos meios de transporte, a empresa conta com 4 meios de transporte, marítimos e terrestres, todos com limitações, e a entrega, os itens serão separados e organizados de acordo com os transportes que os levarão, cada item possui um, ou mais, meio(s) de transporte

capaz de transporta-lo.

Dadas todas ás informações sobre o que seria desenvolvido, foi dado, como prazo de entrega do software, 34 dias, juntamente com o código, foram enviados para empresa um diagrama de classe, afim de tornar o entendimento dos usuários a respeito do funcionamento do software maior, testes unitários, para que todas as especificações fossem avaliadas, e um relatório do problema, para que possa ser avaliado o entendimento e os caminhos tomados pelo programador para a conclusão do código.

2. Fundamentação Teórica

Para a criação do software, em linguagem de programação Java, foi utilizado o Netbeans IDE, por ter sido considerado, entre as opções, o melhor para uso. Foi preciso aprimorar o conhecimento em áreas da programação orientada a objetos(POO), tais como estrutura de dados, métodos “dividir e conquistar” de ordenação nessas estrutura, herança e mais algumas funcionalidades da linguagem.

2.1 Lista Encadeada

Devido as necessidades da empresa em adicionar itens conforme fosse necessário, percorrer os que foram adicionados e ordena-los, foi escolhido a estrutura de lista encadeada para a manipulação dos produtos (Figura 1), uma sequência de células; cada célula contém um objeto (todos os objetos são do mesmo tipo) e o endereço da célula seguinte[Ime-USP,2018].

A lista tem seus benefícios, como poder inserir quantos objetos forem necessários e no momento que for conveniente, e alguns malefícios, como a necessidade de ter que acessar "nó" por "nó" até encontrar o objeto requisitado, demandando processamento, contudo, ela foi avaliada como a melhor para o problema, com base na implementação e funcionalidades da mesma.

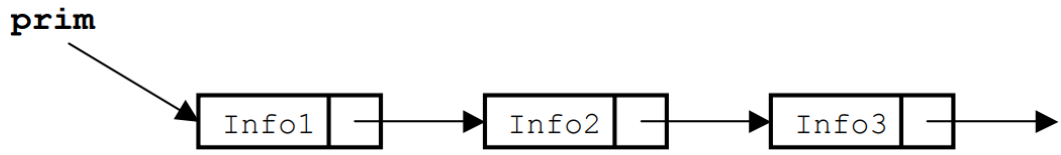

Figura 1. Funcionamento de uma Lista Encadeada

Cada "info" representa um Objeto da classe "Produto", que foi adicionado na lista, essa classe Produto contém os atributos que um produto cadastro deve conter, peso, preço, etc. Por ser uma lista simplesmente encadeada, cada "Nó" contem apenas seu conteúdo (Objeto inserido) e uma referência para o próximo "Nó"(Objeto inserido posteriormente).

2.2 Método de Ordenação MergeSort

A etapa de transporte requisitada pela Empresa consiste em, após receber e armazenar os itens em um estoque, lista encadeada, ordena-los com base em diferentes parâmetros: peso, valor, data de chegada, custo de frete, etc, dadas as circunstâncias e, por estar utilizando a lista encadeada como estrutura de dados, foi escolhido o método de ordenação chamado MergeSort, um algoritmo de "dividir e conquistar", no qual primeiro dividimos o problema em subproblemas. Quando as soluções para os subproblemas estiverem prontas, as combinamos para obter a solução final para o problema [Baeldung, 2018], (Figura 2).

Essa classe consiste em 3 métodos, "List mergeSort(List cabeça)","List getMiddle(List cabeça)" e "List sortedMerge(List a, List b)". Para ordenar uma lista é necessário que a referência do começo da lista(head ou cabeça) seja alterada para um nova, essa é criada pelo método "mergeSort" que, recursivamente, "chama" o método "getMidle" até que a lista, ou vetor, seja um conjunto de listas/vetores de tamanho(s) um ou dois, após isso, é aplicado o método "sortedMerge", recursivamente, que utiliza um objeto "Comparator" para analisar se

será feito, ou não, a troca entre os objetos. Após o fim das chamadas recursivas esses subVetores/subListas são concatenados e o produto final é a estrutura de dados ordenada por um parâmetro escolhido.

O MergeSort é um algoritmo recursivo, a complexidade do tempo pode ser expressa como a seguinte relação recursiva: O (nLogn).

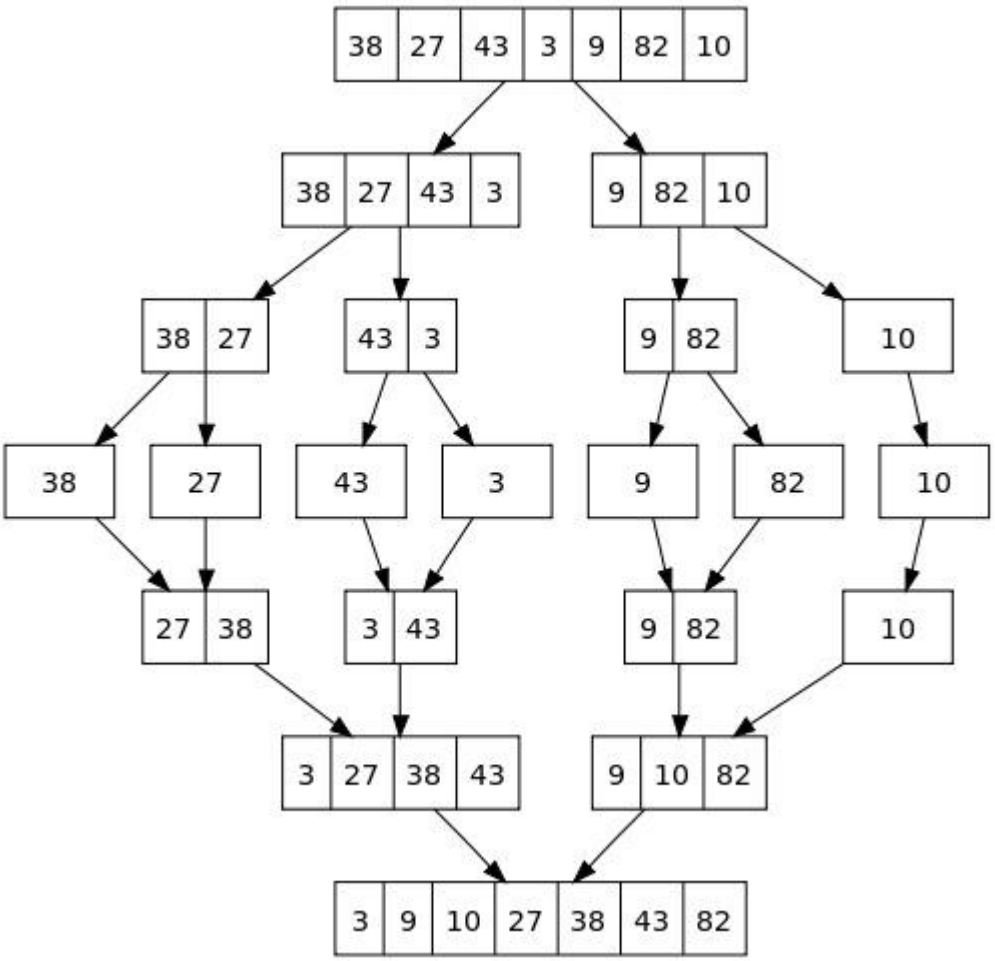

Figura 2. Funcionamento MergeSort

Nessa imagem, o Merge Sort é realizado em um vetor de numero inteiros, após a divisão do vetor em subVetores, esses tem seus números comparados e, se necessário, tem os mesmos trocados. Após realizar todas as trocas necessárias, os subVetores são concatenados e o vetor está do tamanho original e ordenado.

2.3 Classes Comparator

Originalmente, o método “sortedMerge” do MergeSort recebe, como parâmetro, dois objetos do tipo “Object” ou do tipo ”List”, classe que contém os dados do “nó” e a referência para o próximo elemento da lista, e compara os valores dos dados de cada objeto, porém, por ter que ordenar a lista por diferentes parâmetros e por diferentes ordens, crescente e decrescente, foi necessário criar 7 classes de comparação, (Figura 3), que implementam a interface do Java “Comparator” , presente no pacote “java.util.Comparator”.

Essa interface tem apenas um método, “compare”, que recebe dois objetos, A e B, por exemplo, se o dado usado na comparação de “A” for menor que o de B, será retornado -1, o contrário, será retornado 1 e, se ambos dados forem iguais, será retornado 0(Figura 4).

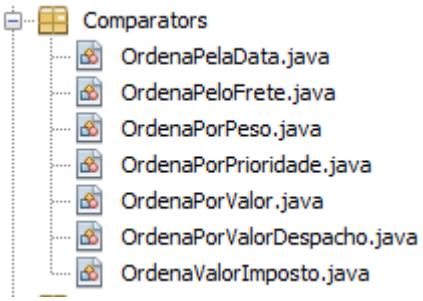

Figura 3. Classes de Comparação

Foi necessário criar, no total, 7 classes para realizar comparações, uma pra cada tipo de parâmetro avaliado, cada classe dessa foi instanciada e passada no construtor de um objeto do tipo Sort, classe de ordenação que utiliza o MergeSort, ao passar um objeto de uma dessas classes, ele será utilizado como parâmetro de avaliação do método “sortedMerge” para avaliar se será feita ou não a troca entre os objetos.

```java
package Comparators;

import br.uefs.ecomp.organizer.model.Produto;
import java.util.Comparator;

public class OrdenaPeloFrete implements Comparator{
    @Override
    public int compare(Object t, Object t1) {
        if(((Produto)t).getCustoFrete()<((Produto)t1).getCustoFrete())
            return -1;
        else if(((Produto)t).getCustoFrete()>((Produto)t1).getCustoFrete())
            return 1;
        return 0;
    }
}
```

Figura 4. Exemplo de Classe de Comparação

Nessa classe, está sendo feita a comparação entre dois "Objects", t e t1, esses objetos são convertidos em "Produtos", para que se possa ter acesso ao "custoFrete" dos dois , comparados e, por fim, é retornado um valor positivo, negativo, ou 0.

3. Metodologia

Afim de compreender melhor o funcionamento, as classes utilizadas e como elas se relacionam no software, foi desenvolvido um Diagrama de Classes de Projeto (DCP), a partir dele foi pensado, em parceria com colegas de sessão, como deveria ser a estrutura do código.

Cada produto adicionado, independente de seu tipo, possui alguns atributos comuns a todos os outros, peso, custo do frete, valor do item, etc, e alguns outros produtos possuem características semelhantes apenas à alguns Produtos, com base nesse Fato, foi decidido utilizar um dos artifícios característicos da POO, a Herança entre classes, no qual, uma ou mais classe(s)(SubClasses), herdam as características de uma outra classe(SuperClasse) assim, tendo acesso aos mesmos atributos e métodos oriundos da SuperClasse mas, podendo acrescentar novos dados.

Ao analisar as "User Stories" do problema, foi entendido que os planos de Carga, assim como os relatórios de despacho e de estoque, seriam listas encadeadas, ordenadas de várias formas possíveis e contendo itens previamente selecionados. Após essa conclusão foi feita a escolha do MergeSort como método de ordenação, por ter sido considerado o método de ordenação mais eficaz pra listas encadeadas, e a criação das classes de comparação.

Todas as etapas do Software foram sendo testadas logo após serem implementadas, foi criada uma classe "main" para que fosse possível realizar testes, informais, e obter informações a respeito de como o código estava se comportando em situações normais e adversas.

.

4. Resultados e Discussões

Após a criação e realizações de testes informais na classe “main”, foi criada as classes de testes, pedida pelos criadores do problema, dentro delas, foram testadas as user stories, avaliando se o comportamento do Software obedecia ao que era esperado pela Empresa contratante.

Para facilitar a visualização geral dos resultados dos testes, assim como no problema requisitado passado, está presente um teste “allTests” que “chama” todos os testes do sistema e gera uma porcentagem a respeito da aprovação deles(Figura 5).

```
package br.uefs.ecomp.organizer;

import br.uefs.ecomp.organizer.model.DespachoTeste;
import br.uefs.ecomp.organizer.model.EstoqueTeste;
import br.uefs.ecomp.organizer.model.PlanoDeCargaTeste;
import org.junit.runner.RunWith;
import org.junit.runners.Suite;

import br.uefs.ecomp.organizer.util.ListTest;


@RunWith(Suite.class)
@Suite.SuiteClasses({
        ListTest.class,
        DespachoTeste.class,
        EstoqueTeste.class,
        PlanoDeCargaTeste.class,
})
public class AllTests { }
```

Figura 5. Classe AllTests

Nesse teste é “chamado” os demais testes, ListTest, testando lista encadeada, DespachoTest, contemplando user stories 8,9,10, EstoqueTest, contemplando user stories 1,2 e 3, e o DespachoTest, contemplando user stories de 4 a 7.

4. Conclusão

O software criado cumpriu todos os requisitos propostos pela empresa ImportaFacil, garantindo assim a eficiência prevista para o programa.

Os comentários, padrão e JavaDoc, tornaram o entendimento simples e acessível ao futuro usuário, sem que seja necessário um entendimento considerável em linguagem de programação Java,

Afim de economizar gastos de transportes à empresa, durante os planos de carga, caso um objeto não pudesse ser adicionado, devido ao seu peso, poderia ser avaliado um outro produto com peso suficientemente pequeno para preencher a capacidade restante de carga, dessa forma, mais itens poderiam ser transportados e assim menos transportes seriam utilizando, minimizando gastos à empresa.

5. Bibliografia consultada

Baeldung, (2018), “Merge Sort in Java”, disponível em: https://www.baeldung.com/java-merge-sort. Acessado em 12/11/2018.

Geek For Geeks(2018), “Merge Sort”, disponível em: https://www.geeksforgeeks.org/merge-sort/ . Acessado em 12/11/2018.

Ime-Usp(2018),” Lista Encadeada”, disponível em https://www.ime.usp.br/~pf/algoritmos/aulas/lista.html .Acessado em 12/11/2018