**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](mailto: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 transporte, 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 programados para a conclusão do código.

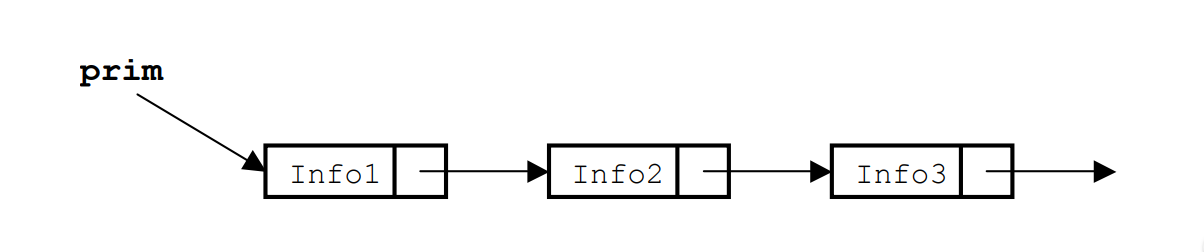
**2. Fundamentação Teórica**

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 de ordenação em estrutura de dados, herança e mais algumas funcionalidades da linguagem.

**2.1 Lista Encadeada**

Devido as necessidades do produto 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 implementação no problema, com base na implementação e funcionalidades.



**Figura 1. Funcionamento de uma Lista Encadeada**

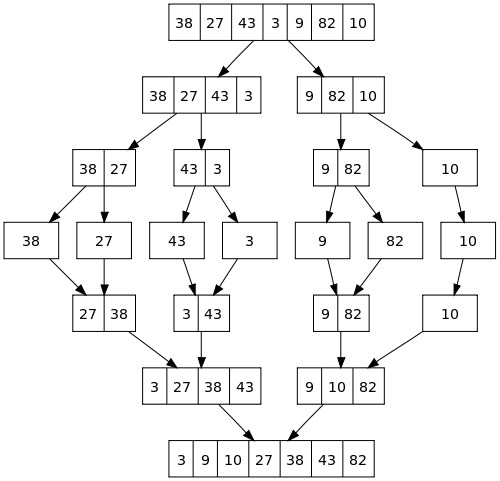
Cada “info” representa um Objeto da classe “Produto” que foi adicionado na lista, essa classe Produto contém os atributos que um produto cadastro deve conter, peso, preço, etc. Por ser uma lista simplesmente encadeada, cada “Nó” contem apenas seu conteúdo (Objeto inserido) e uma referência para o próximo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



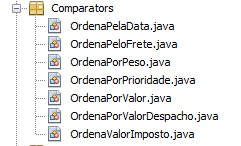
**Figura 2. Funcionamento MergeSort**

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

**2.3 Classes Comparator**

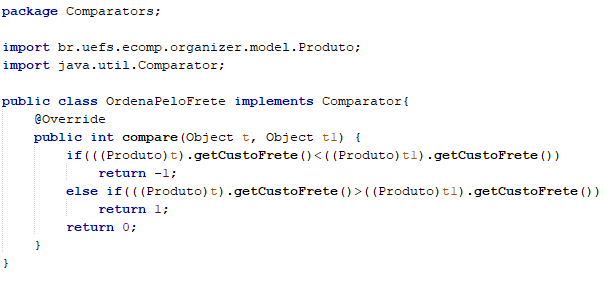
Originalmente, o método “sortedMerge” do MergeSort recebe, como parâmetro, dois objetos do tipo “Object” ou do tipo ”List”, classe que contém os dados do “nó” e a referência para o próximo elemento da lista, e compara os valores dos dados de cada objeto, porém, por ter que ordenar a lista por diferentes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

Essa interface tem apenas um método, o “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.



**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**

.

**3. Resultados e Discussões**

Devido ao redirecionando da entrada/saída de dados, para poder realizar testes e averiguar o funcionamento do código, um arquivo de entrada e outro de saída foram criados, no arquivo de entrada contém todos os dados necessários para explorar as funções do código, cadastro, alterar camisa, remover jogador, sortear, exibir todos jogadores, exibir somente os sorteados e filtrar por posição.

Devido ao fato de que o código foi sendo feito já modularizado, os erros e problemas que ocorriam podiam ser facilmente localizados, o que ajudou na solução desses.

**4. Conclusão**

O software criado cumpriu todos os requisitos propostos porém, há uma possibilidade de que, durante a exibição de jogadores sorteados, a exibição contenha menos jogadores do que foi pedido. Isso acontece porque mesmo não gerando números iguais no sorteio, durante uma segunda chamada da função de sortear, um número sorteado possa já ter sido escolhido. Na folha do problema, há escrito que um jogador não pode ser sorteado mais de uma vez, o código não sorteia um jogador mais de uma vez porém, pode imprimir uma quantidade inferior de jogadores em relação ao que foi pedido. Tal questão pode, ou não, ser avaliada como um erro.

O código foi feito de modo que tanto as saídas como o código fonte estivessem simples e explicativos. Deve-se ter cuidado em relação a formatação do arquivo que serviu como entrada para o código, a formatação deve estar completamente correta, um erro simples no texto pode afetar o desempenho do código, que irá criar arquivo de saída sem sentido ou talvez sequer execute.

O papel do software é importante e será efetivo quando for utilizado, mas a remoção de jogadores poderia ser feito com base em algum parâmetro, poderia haver mais um dado no arquivo que pudesse constatar se um jogador usou ou não substâncias proibidas e , caso este jogador fosse sorteado, a função de remover seria aplicada a ele e não somente a um nome escolhido, sem um parâmetro preciso, presente no arquivo.

**5. Bibliografia consultada**

FORBELLONE, A. V. L., EBERSPACHER, H. F. Lógica de Programação: A Construção de Algoritmos e Estrutura de Dados. 2. ed. Makron Books, 2000.