EXA 863 - MI-PROGRAMAÇÃO 2018.2

Relatório – Importação e transporte de mercadorias de alto valor

Ian Zaque Pereira de Jesus dos Santos1

1Curso de Bacharelado em Eng. de Computação – Universidade Estadual de

Feira de Santana

(UEFS) – Campus Feira de Santana

CEP 44.036-900 – Feira de Santana – BA – Brasil

ianzaque.uefs@gmail.com

**1. Introdução**

Desde antes do surgimento do comércio havia a necessidade de transportar mercadorias, pessoas, animais, alimentos e etc. Hoje em dia a situação não é diferente, porém com grande disponibilidade tecnológica surgiram novas preocupações e situações que precisam ser manuseadas e administradas corretamente. No mercado atual, é necessário que existam empresas que possam transportar materiais e objetos para outros lugares e assim movimentar a economia. Para isso é interessante que haja um meio de lidar com envio e recebimento de cargas importadas e exportadas.

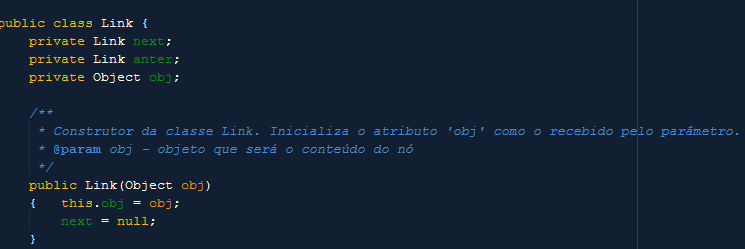
Pensando nisso, a empresa ImportaFácil que trabalha com mercadorias de grande valor monetário, encomendou um software que seja capaz de atender suas necessidades de administração com as cargas. A empresa já havia contratado uma outra empresa de construção de softwares mas esta foi proibida de atuar devido à problemas com a Justiça Federal. Mesmo assim, a ImportaFácil permaneceu com alguns dados já iniciados pela empresa barrada e contratou outra empresa para o serviço.

É interessante que a empresa contratada construa um software que contemple os requisitos pedidos pela empresa disponibilizados nos casos de uso que relata o contrato. Além disto, foi pedido um diagrama de classes que represente este software criado e um relatório contando e explicando o funcionamento do programa. Este relatório, portanto, no tópico Fundamentação teórica explicitará os assuntos suscintamente para dar base à Metodologia que trata dos procedimentos utilizados para tentar solucionar o problema e a criação de uma possível solução. Não somente, mostra também no tópico Resultados e Discussões o que foi debatido e quais dificuldades se mostraram presentes, assim como objetivos cumpridos. No tópico conclusão, esclarecem-se quais os temas aprendidos pelos estudantes e expõe comentários sobre propostas futuras.

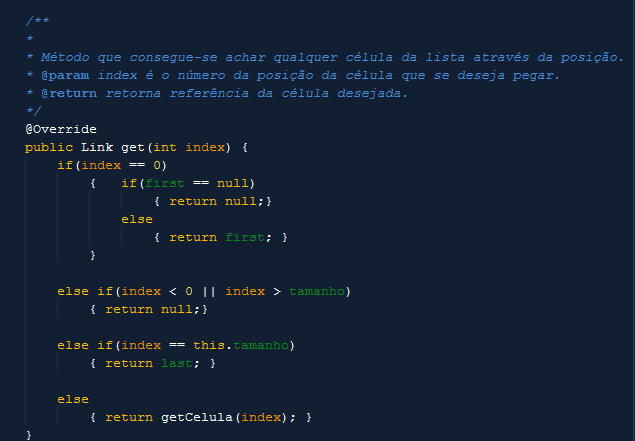
**2. Fundamentação teórica e Metodologia**

Para a construção deste software deve ser feita usando a linguagem de programação Java, que tem por princípio orientação à objetos. Este tipo de programação envolve e visa dar funcionalidades a certos objetos que são instâncias de uma classe onde é desenvolvido o código para este objeto. Pode-se então ter várias classes diferentes e de vários tipos e também vários objetos de uma mesma classe. Estas são as classes desenvolvidas para este projeto:

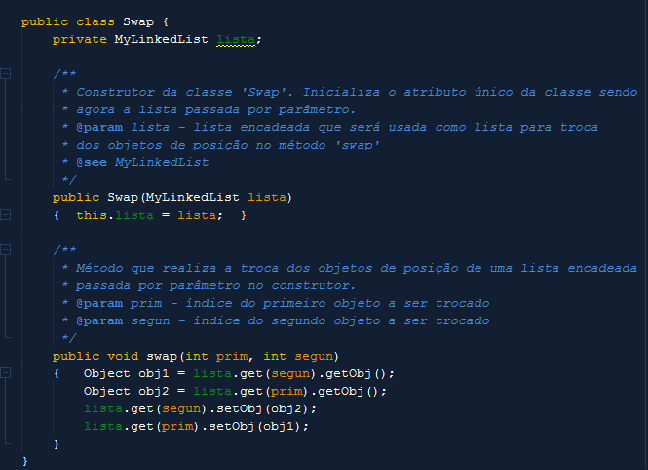
- Classe *Link*

Essa classe é uma classe recursiva e tem função de guardar um objeto fazendo referência a ele através de dois atributo que é da mesma classe *Link*. Um destes atributos fará referência a algum objeto desta mesma classe anterior e outro para um posterior. Porém, não usou-se esta primeira capacidade se não na classe que será detalhada a seguir. No construtor desta classe os atributos recebem *null*.

- Classe MyLinkedList

Esta classe é uma classe que é usada para armazenar dados e objetos em seus nós. Cada nó é um objeto da classe *link* que se liga ao próximo e assim por diante até chegar ao final da lista encadeada. No construtor desta classe os atributos do tipo Link são inicializados com *null* e o tamanho com 0. Esta classe possui métodos de adicionar, remover e retornar elementos por posição, no começo e no final da lista. Quando quer selecionar um item da lista através do método ‘get(int)’ retorna-se uma referência do tipo *Link* para o objeto que está na posição selecionada pelo índice. As demais retornam um objeto que pode ser comparado do tipo *Comparable*. A classe implementa a interface *IList* que dá os métodos que foram implementados na classe.

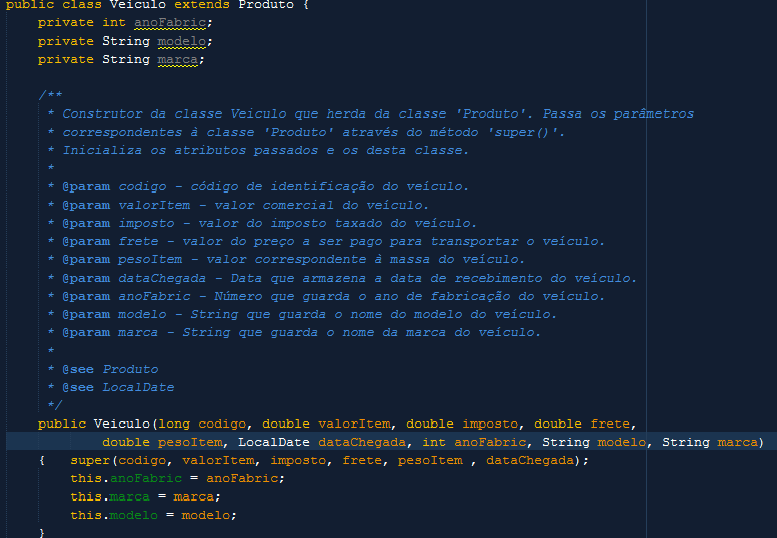
- Classe Swap

A classe Swap é responsável por trocar dois objetos de posição. Na realidade, ela troca o conteúdo de um nó do tipo *Link*. Possui um atributo do tipo MyLinkedList que recebe no construtor a lista que será usada para fazer as trocas. Esta classe possui o método de mesmo nome da classe que faz as trocas baseada no índice da posição da lista. Usa-se esta classe nas classes de ordenação por critérios.

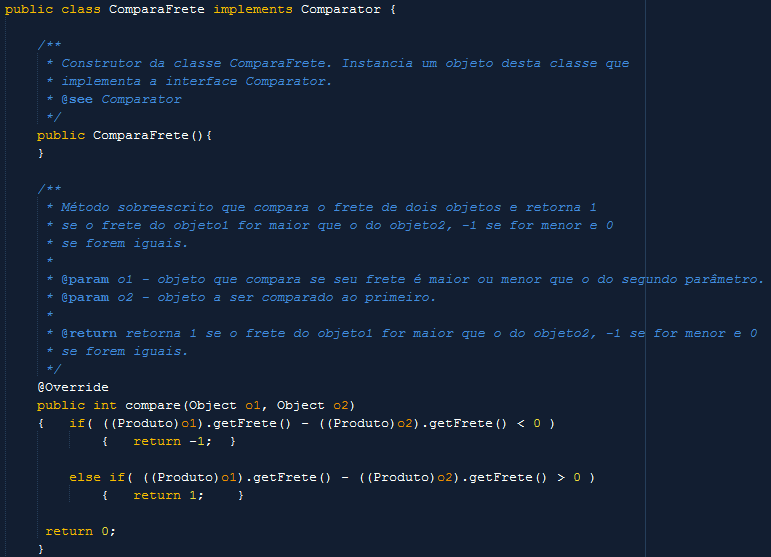
- Classe Produto

A classe Produto é interessante de se analisar. Possui os atributos que foram pedidos no contrato e implementa as interfaces *Comparable* e *Comparator*, e é usada para gerar os itens que são transportados pela empresa ImportaFácil. Estes atributos mencionados são inicializados no construtor através da passagem de parâmetro, e além destes atributos, foi interessante usar uma variável do tipo *boolean* para saber se o item foi vendido ou não. O último atributo mencionado é inicializado com *false* pois quando cadastra-se um produto ele ainda está indo para o armazém da empresa. Possuis os métodos para retornar os atributos e para modifica-los e também os métodos sobreescritos *compare(Object,Object)* e *compareTo(Object)* oriundos das interfaces implementadas. Um método sobrescrito é aquele que tem a mesma assinatura de outro porém sua implementação é diferente. Esta é a definição de polimorfismo, um dos princípios da linguagem Java. Esta classe possui também o método sobrescrito *equals(Object)* que retorna verdadeiro caso dois produtos forem iguais e falso caso contrário.

- Classes filhas de Produto

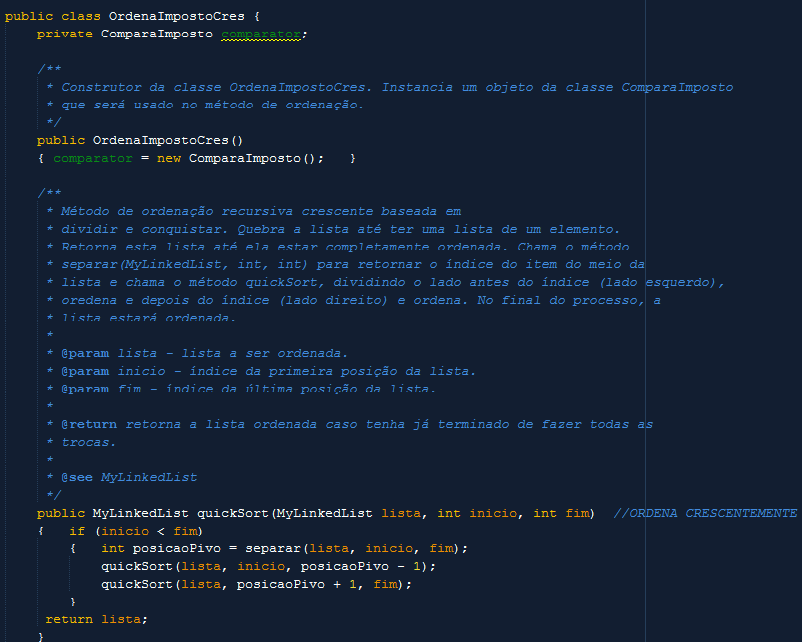
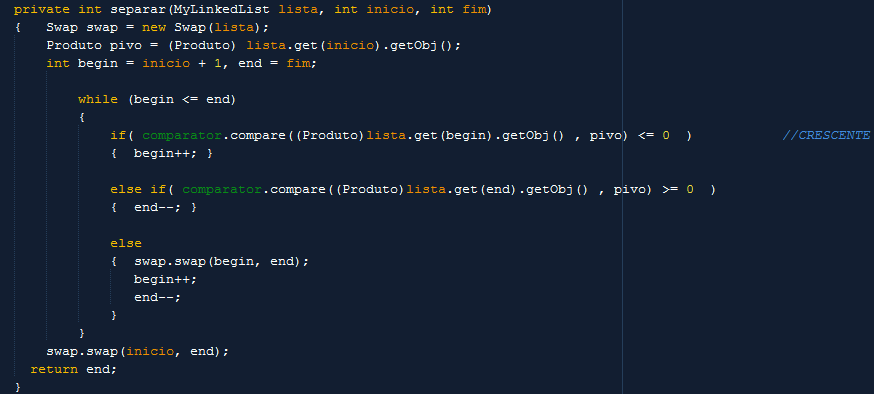
Existem 2 classes que herdam de Produto, ou seja, possuem todos os métodos e atributos da classe mãe e outros atributos e métodos (Conceito de herança). Uma delas é Veiculo que também possui classes filhas (Carro, Moto – que herda de Carro – e Barco) e a outra é ItemFragil que igualmente possui classes filhas (Quadro e Joia). Exemplo abaixo:

- Classes Comparadoras

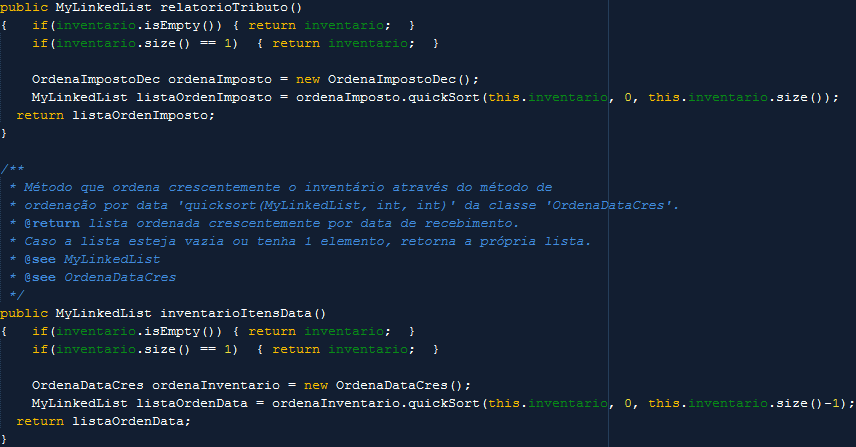
No contrato da empresa, foi pedido alguns casos de uso onde é preciso carregar um meio de transporte com objetos da classe Produto porém é preciso que estes estejam ordenados. Portanto, foram implementadas classes que comparam objetos de um mesmo tipo. As classes implementam também a interface *Comparator* que dá o método *compare(Object)* e algumas a interface *Comparable* mas muitas vezes esta não foi utilizada. As classes que implementam *Comparable* possuem um atributo do tipo Produto para ser usado no seu método implementado e o construtor. Já as que implementam apens *Comparator* têm apenas o construtor e o método de sua interface. Exemplo de classe que compara valores dos produtos e implementa apenas *Comparator* a seguir:

- Classes de ordenação

Como foi dito anteriormente, foi preciso ordenar listas encadeadas dinamicamente. Para isso, foi escolhido o método q*uickSort,* baseado em dividir e conquistar, onde ele quebra a lista toda até ter 1 elemento, ordena os retornos e a lista retornará ordenada para a chamada do método. Cada classe de ordenação possui um atributo específico para aquele tipo de ordenação. Por exemplo: a classe OrdenaValorCres possui um atributo ComparaValor que é usado para ver qual produto tem o maior ou o menor valor e assim, trocar as posições caso seja necessário. As classes de ordenação possuem classes “irmãs” que não tem relação direta, mas possuem uma semelhança. A diferença entre elas é que se uma ordena crescentemente, a outra ordena decrescentemente. A única que não possui esta característica é OrdenaFreteDec que não existe uma classe OrdenaFreteCres. Exemplo desse tipo de classe e seus métodos:

 - Classe Armazem

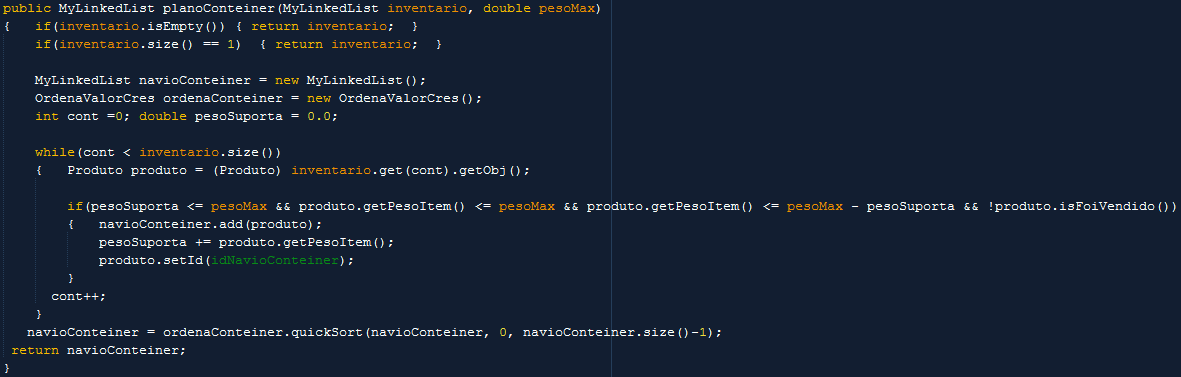
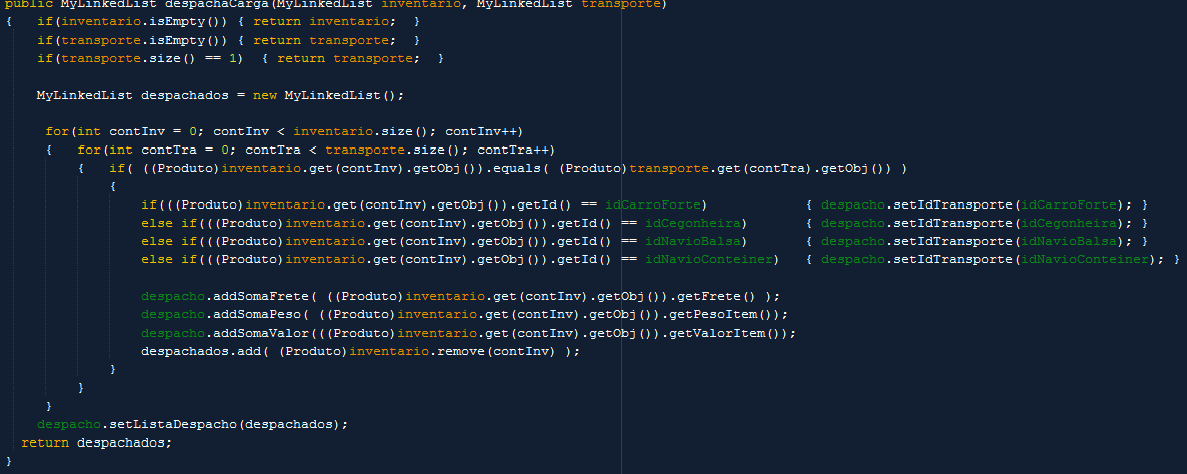
A classe Armazem é única. Possui um atributo apenas do tipo MyLinkedList, inicializado no construtor, e é a partir dele onde as operações são programadas. A classe possui os métodos de retornar o atributo, mas também possui o método de cadastrar, remover e retornar um Produto. Esses métodos são implementados usando o método de MyLinkedList correspondente à opção desejada. Não obstante, possui o método que gera o relatório de tributação (retorna um objeto MyLinkedList) que visa ordenar o inventário a partir do valor do imposto a ser pago decrescentemente. Além deste, possui também o método de inventário de itens no depósito ordenando crescentemente a partir da data de recebimento. Exemplo a seguir:

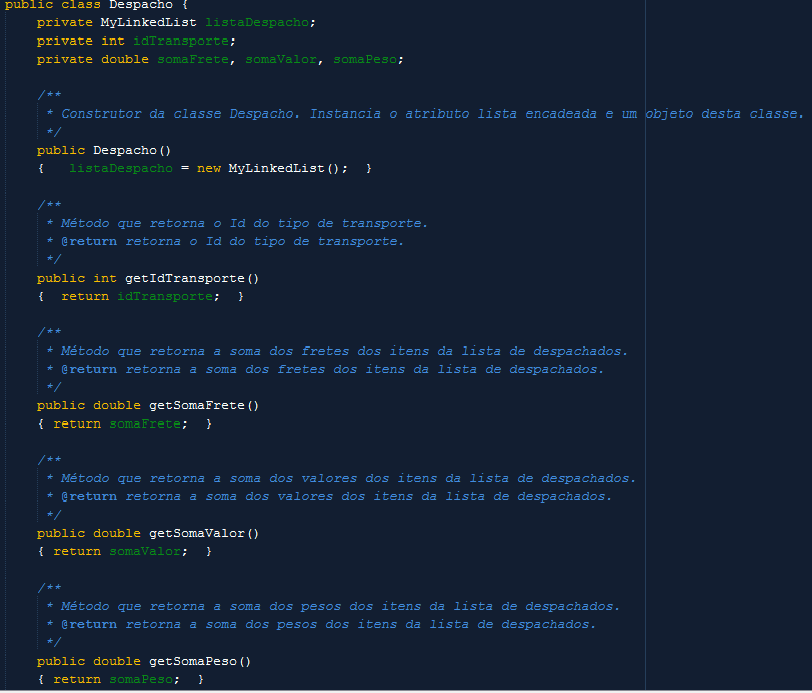
- Classe PlanoDeCarga

Nesta classe acontece boa parte das operações requisitadas no contrato. Possui atributos do tipo inteiro que são usados para identificar o transporte onde as cargas são levadas e um atributo do tipo Despacho que guarda as informações de uma operação de transporte de mercadoria. Esta classe possui um construtor e os métodos de plano de carga pedidos. Cada método possui sua especificação mas, generalizando, cada método recebe uma lista com os itens cadastrados e um valor de peso máximo “cabível” no transporte.

O método checa cada item do inventário e analisa qual tem as características que o permitem ser carregados para ser levado e o id de transporte do Produto é marcado com o do tipo do meio de transporte em que ele está inserido. Caso carregue-se um Produto e não houver mais espaço o transporte parte assim desse jeito, mesmo que sobre algum espaço vazio.

Possui também o método de despachar cargas. Ele é bem semelhante ao plano de carga, porém checa quais itens são iguais aos do inventário e apaga de lá. Cada atributo (peso, valor, frete) de cada Produto despachado é somado a um atributo da classe Despacho – que será explanado futuramente – e, após isso, adiciona-se o item removido à lista de itens excluídos do inventário. Exemplo abaixo de um dos métodos de plano de carga e exemplo do método de despachar carga.

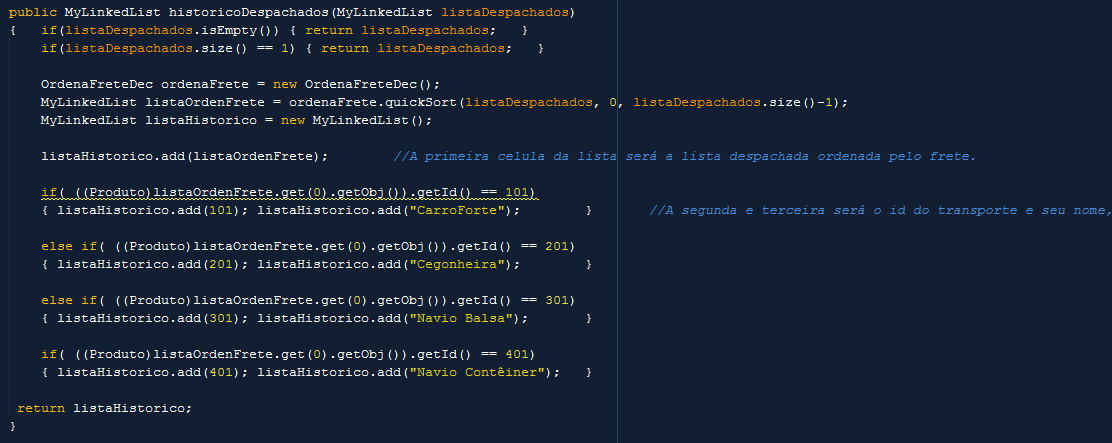
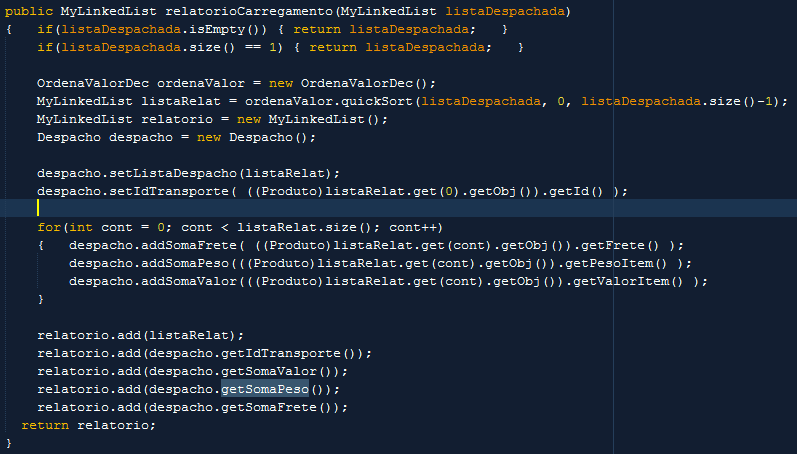
- Classe Despacho

Despacho é uma classe que armazena as informações de um plano de carga e despacho da carga. Possui atributos que guardam a soma dos pesos, valores e fretes dos itens despachados, e um atributo MyLinkedList que se torna depois a lista dos que foram despachados do inventário. Possui o construtor e os métodos de retornar e modificar os atributos que são privados. Interessante notar que todos os atributos das classes são privados de acesso, podendo ser acessados apenas pelos métodos de acesso. Desse jeito as propriedades ficam protegidas de serem modificadas diretamente. Tal característica é definida como Encapsulamento de dados.

- Classe Sistema

A classe Sistema é simples arbitrariamente falando. Não possui atributos apenas dois métodos que retorna objetos MyLinkedList. Um dos métodos é o de gerar o histórico de itens despachados que tem como meta ordenar decrescentemente pelo frete uma lista que é adicionada no primeiro nó de outra lista. No segundo nó, é adicionado o id do tipo de transporte que os itens foram levados e no terceiro nó armazena-se uma String com o nome do tipo de transporte que levou as cargas.

O segundo método é o que gera o relatório de carregamento. Nele, usa-se um objeto da classe Despacho que vai adicionando em seus atributos as somas do peso, frete e valor dos itens despachados. Os itens são ordenados pelo valor da carga e, assim como no método anterior, a lista onde estes itens estão é adicionada no primeiro nó de outra lista encadeada. No segundo, terceiro e quarto nós são adicionados o id de transporte da lista, a soma dos valores dos itens, soma dos pesos dos itens e soma dos fretes dos itens, respectivamente. Os métodos podem ser vistos abaixo:



**3. Resultados e discussões**

Os resultados após a conclusão do código e do diagrama de classes foi proveitoso em sua essência. Foi possível revisar os conhecimentos adquiridos no semestre e no problema anterior sobre listas encadeadas, polimorfismo, interfaces, encapsulamento, estrutura de dados, Javadoc e etc. Além disso, novos estudos renderam novos saberes sobre herança, ordenação de listas encadeadas, modelagem UML, modelagem de testes de unidade dentre muitos outros tópicos sobre programação em Java.

Este relatório explicita as etapas que foram utilizadas para o funcionamento do programa desenvolvido. E, partindo desses procedimentos, pode-se visualizar que o programa cumpre com a solução dos requisitos apresentados. Por ter-se usado o método de ordenação quicksort – não tão eficaz para listas encadeadas – o desempenho possa não ser muito eficiente, assim como a criação de muitos objetos do tipo MyLinkedList que pode carregar muito a memória e perder capacidade de processamento em um exemplo com uma quantidade de itens bem maior que o que foi testado para a confecção do software.

As sessões tutoriais foram indispensáveis para a implementação do código pois dúvidas que somente surgem na prática foram sanadas. Várias ideias, sugestões, metas e questionamentos surgiram, fato que contribuiu para a melhoria de todos os estudantes. Além disso, a tutoria do professor Claudio Goes promovia sempre a discussão sobre boas ideias visando sempre pensar numa situação mais real do problema apresentado.

**4. Conclusão**

Ao finalizar este relatório, conclui-se que o foi possível compreender e entender como se utiliza listas encadeadas, métodos de ordenação, polimorfismo, herança, encapsulamento, Javadoc, estrutura de dados e criação de testes de unidade que executam com 100%. Não apenas isto houve, também, a utilização de conhecimentos já existentes, aprendidos nos problemas passados. É interessante ressaltar que, mesmo com dificuldades iniciais para a compreensão do problema, o desenvolvimento e evolução do programa foram ocorrendo simultaneamente ao ganho de entendimento nas sessões e nas aulas teóricas.

O acesso ao código fonte do programa, ao diagrama de classes e este relatório são possíveis através do link do Drive a seguir: https://drive.google.com/drive/folders/1TesTUxHgh8RQNbqDUYq8jPBzRMPSbMFa?usp=sharing

**5. Bibliografia Consultada**

https://www.devmedia.com.br/algoritmos-de-ordenacao-em-java/32693

http://sites.ecomp.uefs.br/joao/home/courses/exa806/aulas

http://sites.ecomp.uefs.br/joao/home/courses/exa805/aulas