

|  |
| --- |
| RELATÓRIO ESINF  Projeto Integrador  2022/2023 |
|  |
| 8 de janeiro  JOSÉ GOUVEIA - 1211089  PEDRO PEREIRA – 1211131  TIAGO OLIVEIRA – 1211128  ALEXANDRE GERAÇÃO – 1211151  RICARDO VENÂNCIO – 1210828  DIOGO CARVALHO - 1200611 |

# Índice

[Índice 2](#_Toc124111754)

[US307 3](#_Toc124111755)

[Introdução à *User Story* 3](#_Toc124111756)

[Código 4](#_Toc124111757)

[*Load Orders* 4](#_Toc124111758)

[*Load Products* 5](#_Toc124111759)

[*Compute If Absent* 5](#_Toc124111760)

[Complexidade 6](#_Toc124111761)

[US308 7](#_Toc124111762)

[dispatchListWhitoutRestriction 7](#_Toc124111763)

[makeExpeditionListFullProducts e makeExpeditionFull 9](#_Toc124111764)

[makeExpedetionListRemainProducts e makeExpeditionRemain 12](#_Toc124111765)

[US309 15](#_Toc124111766)

[Introdução à User Story 15](#_Toc124111767)

[Código 15](#_Toc124111768)

[*Restricted Dispatch List* 15](#_Toc124111769)

[*Get N Closest Producers to Hub* 17](#_Toc124111770)

[Complexidade 18](#_Toc124111771)

[US310 19](#_Toc124111772)

[Código 19](#_Toc124111773)

[*minimumDistance* 20](#_Toc124111774)

[*closestPointOfInterest* 21](#_Toc124111775)

[Complexidade 22](#_Toc124111776)

[US311 23](#_Toc124111777)

[Complexidade 27](#_Toc124111778)

# US307

Importar a lista de cabazes.

## Introdução à *User Story*

É pedido nesta *user story* que a lista de cabazes (armazenadas em ficheiros .csv) seja informatizada na aplicação Java, como tal, a forma mais **eficiente** de fazê-lo foi utilizando **duas árvores *AVL***, ora para **encomendas** ora para **produções**.

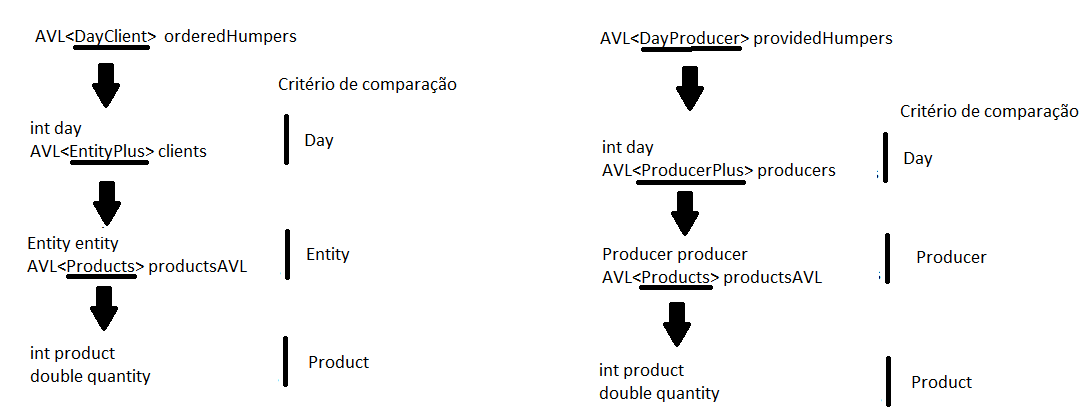


Figura 1 - Esquema da estruturação

Nota: “*Entity”* é um objeto que é estendido pelas classes: *Company*, *Producer*, *Client*. Como tal, a AVL de *EntityPlus* vai conter clientes (o código inibe Producers).

## Código

### *Load Orders*Uma imagem com texto Descrição gerada automaticamente

Uma imagem com texto

Descrição gerada automaticamente

Figura 2 - Código método Load Orders

Antes de iniciar a abertura do ficheiro, é sempre feita uma **validação do caminho** e também uma **verificação da existência do grafo** (relativo ao *sprint* passado).

Em cada linha lida do ficheiro, a primeira palavra vai definir o tipo de entidade, caso seja um cliente/empresa/produtor vai fazer uma **pesquisa numa *AVL* que contém todas as entidades** (com as respetivas identificações de localizações e as próprias), é instanciada corretamente a entidade (como Cliente caso seja cliente, etc.…), caso contrário (não corresponde a nenhum tipo de entidade esperada) a linha vai ser passada à frente.

A partir de agora há uma divisão entre produtores e clientes. Os objetivos são semelhantes, ambos têm de **organizar a informação**. Caso não exista o dia em causa na (respetiva) árvore, caso contrário entra-se na árvore e pesquisa-se pelo próximo critério na próxima árvore.

### *Load Products*

Uma imagem com texto

Descrição gerada automaticamente

Figura 3 - Código método Load Products

Este método vai fazer o trabalho comum a ambos os **lados** (produtores e clientes). Vai ler o **identificador do produto e a respetiva quantidade** para de seguida armazenar na ***AVL***.

### *Compute If Absent*

Uma imagem com texto

Descrição gerada automaticamente

Figura 4 - Código método Compute If Absent

Tal e qual como o método de mapas, tem o mesmo objetivo. Vai pesquisar na estrutura pela chave, caso esteja **ausente** a chave vai ser **inicializada** com o *lambda* enviado através de *function*.

## Complexidade

Dado o problema necessitar de uma leitura de um ficheiro, é incontornável a complexidade . Dentro deste *loop*, o método, no caso, que mais complexidade tem é *Load Products com*  para **o pior/médio** **caso** e para o **melhor cenário.** Conclui-se então que a complexidade final do método é:

* Pior caso:
* Caso médio:
* Melhor caso:

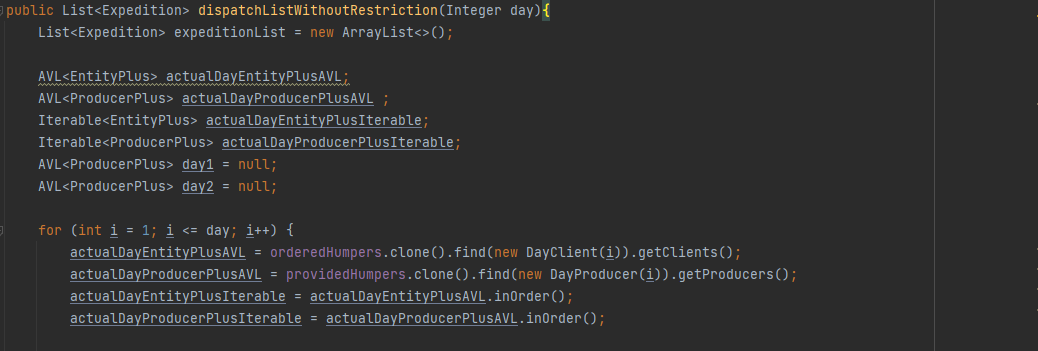
Foram escolhidas árvores *AVL* como estrutura de armazenamento para estes dados, para assegurar, independentemente do caso, uma **complexidade** **logarítmica** em todas as suas operações, mas sobretudo **nas pesquisas** que serão necessárias para outras funcionalidades.

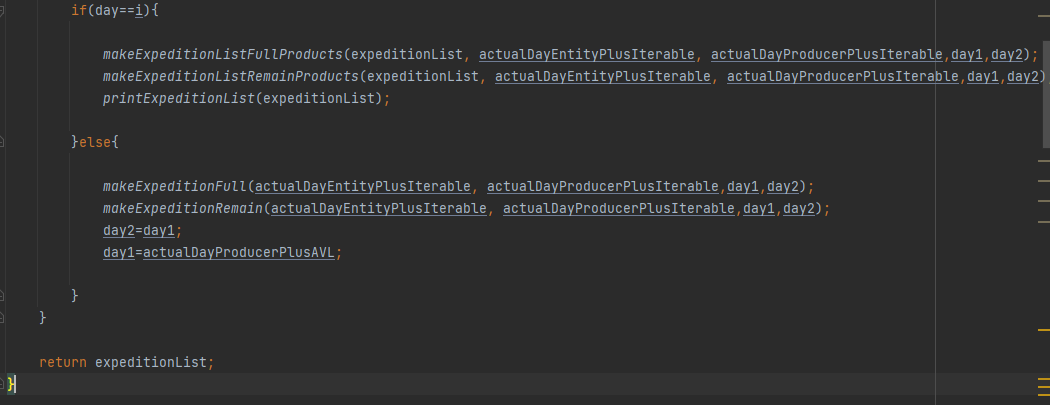
# US308

Gerar uma lista de expedição para um determinado dia que forneça os cabazes sem qualquer restrição quanto aos produtores.

Para a realização desta User Story vamos precisar de vários métodos sendo o principal deles o seguinte.

### dispatchListWhitoutRestriction



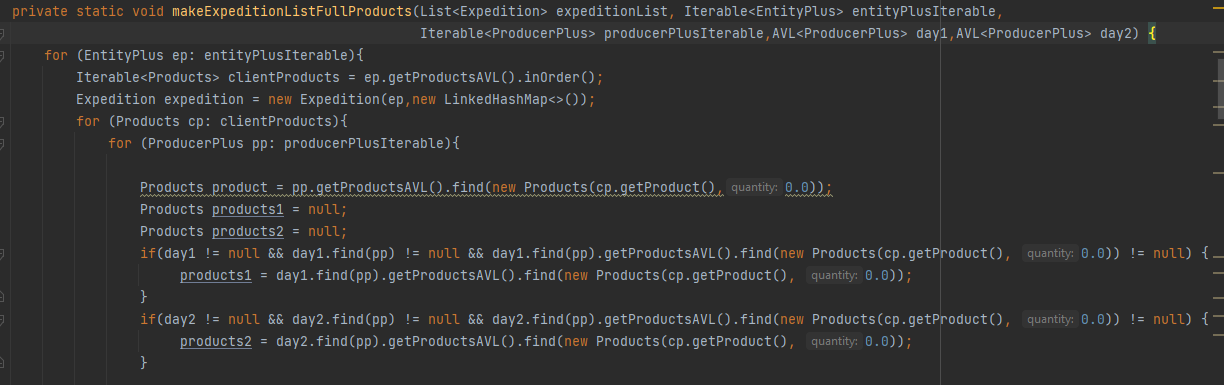


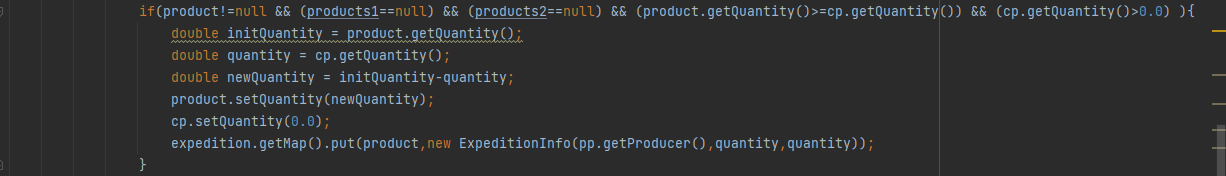
Este método começa por inicializar uma AVL de “EntityPlus”, uma árvore que teria um Cliente e uma AVL de produtos encomendados por eles ordenada por dias, uma AVL de “ProducerPlus” semelhante á arvóre anterior mas para Productor e neste caso uma AVL de produtos fornecidos ordenada por dia, dois Iterable referentes ás arvores anteriores e outras duas AVL de “ProducerPlus” chamadas de day1 e day2 que contem os restos ou seja produtos que ainda não foram comercializados onde na sua nomecltura contêm o numero referente aos dias desde que foram introduzidas no sistema.

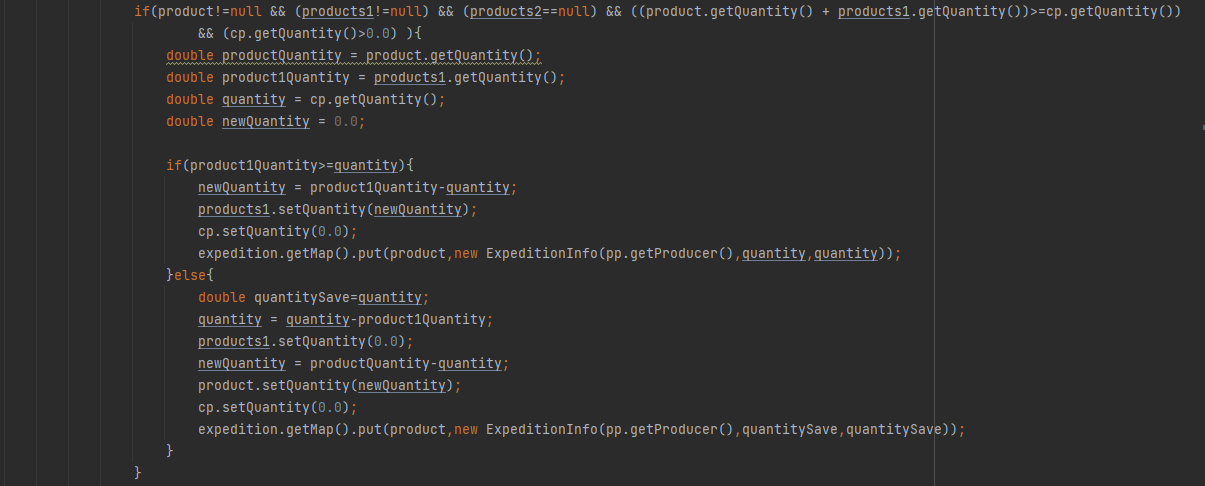
De seguida ocorre um ciclo for i, que inicia no dia 1 e termina no dia recebido por parâmetro, de modo a calcular a lista de expedição desse dia. Se o i for igual ao dia ele vai chamar os métodos “makeExpeditionListFullProducts”, “makeExpeditionListRemainProducts” e “printExpeditionList”, caso i seja diferente de dia ele chama os métodos “makeExpeditionFull” e “makeExpeditionRemain”, e atualiza as Avl day1 e day2, tornando a Avl day2 igual a day1 e a Avl day1 passa a ser igual à “actualDayProducerPlus” que após passar que métodos vai contenter apenas as sobras do dia i.

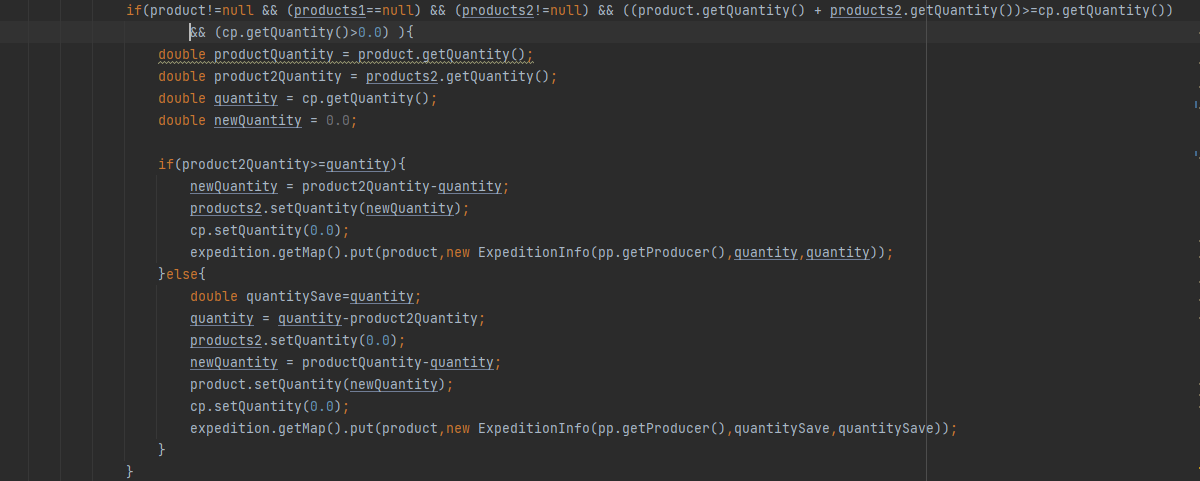
No fim é retornada a “expeditionList” uma Lista de “Expedition”, um objeto que contém um “EntityPlus” refernte a um cliente e um Mapa onde cada produto pedido é uma key e para cada producto existe uma entry com “ExpeditionInfo” que contêm informação acerca do Produtor que forneceu aquele produto, a quantidade pedida por aquele cliente e a quantidade fornecida que em alguns casos por não satisfazer totalmente o pedido do cliente e em outros casos ainda pior pode não ser sequer minimanente satisfeita.

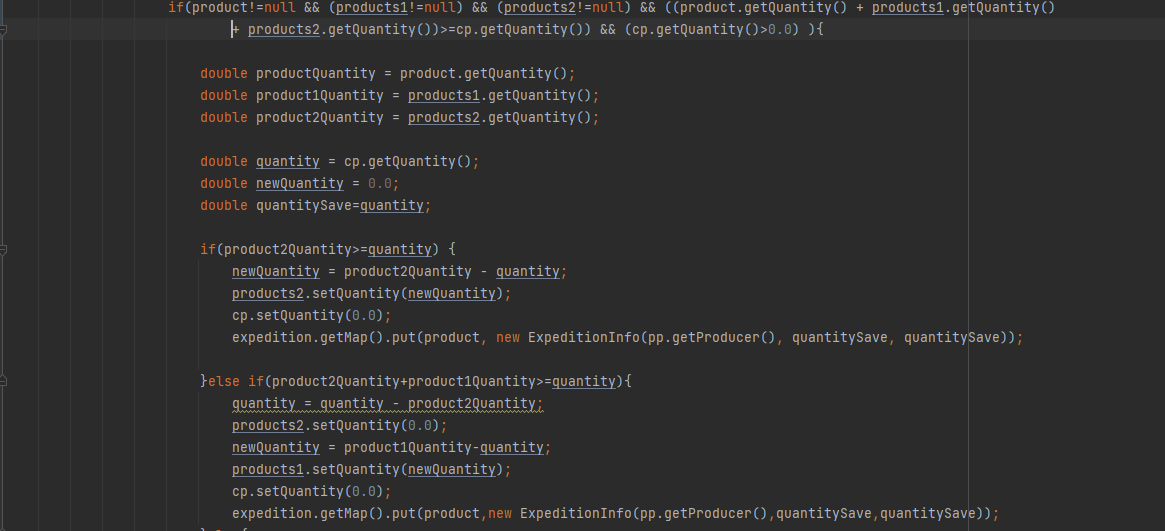
### makeExpeditionListFullProducts e makeExpeditionFull

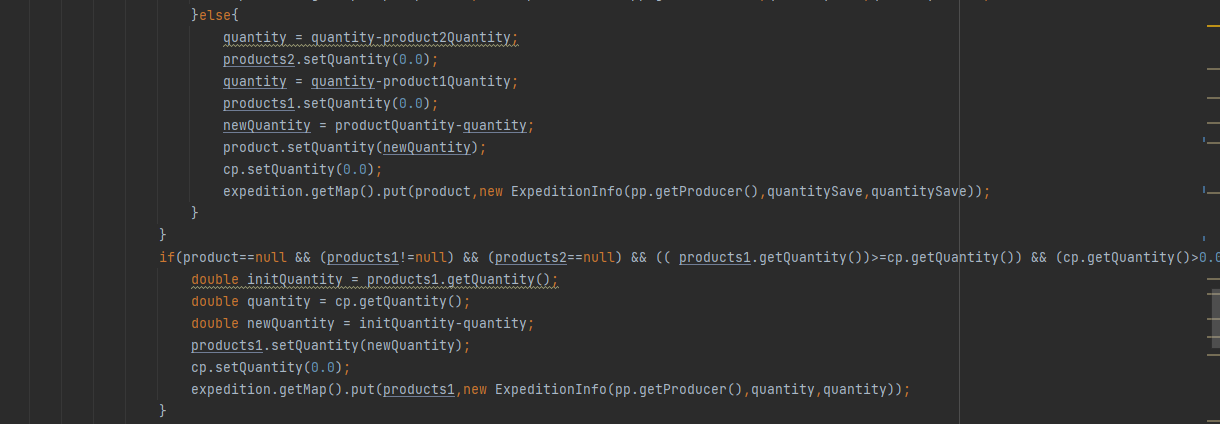


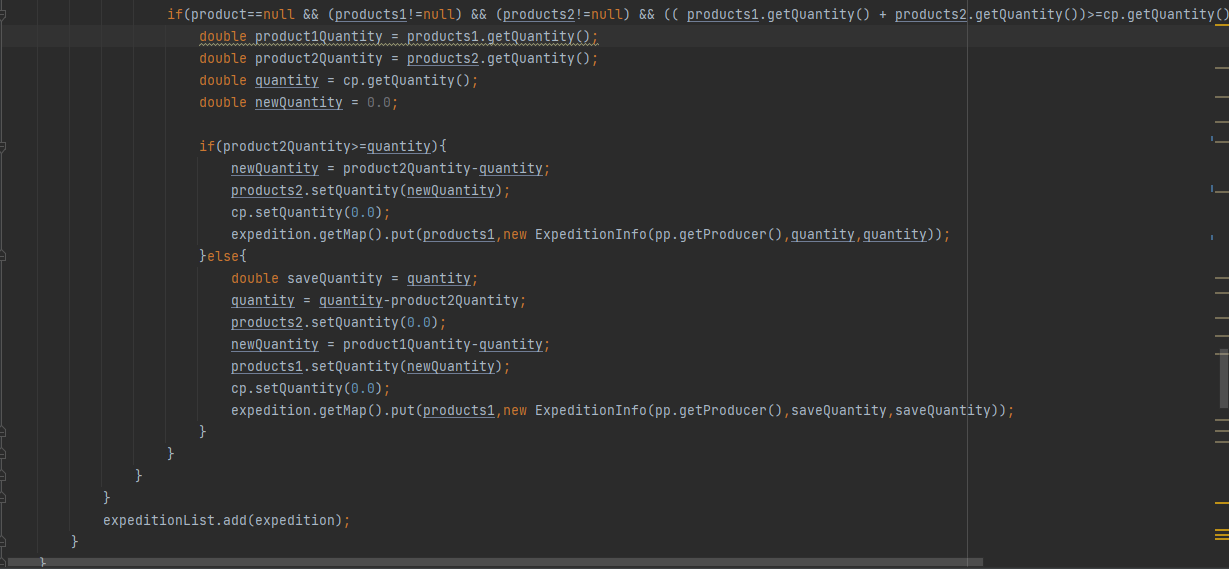












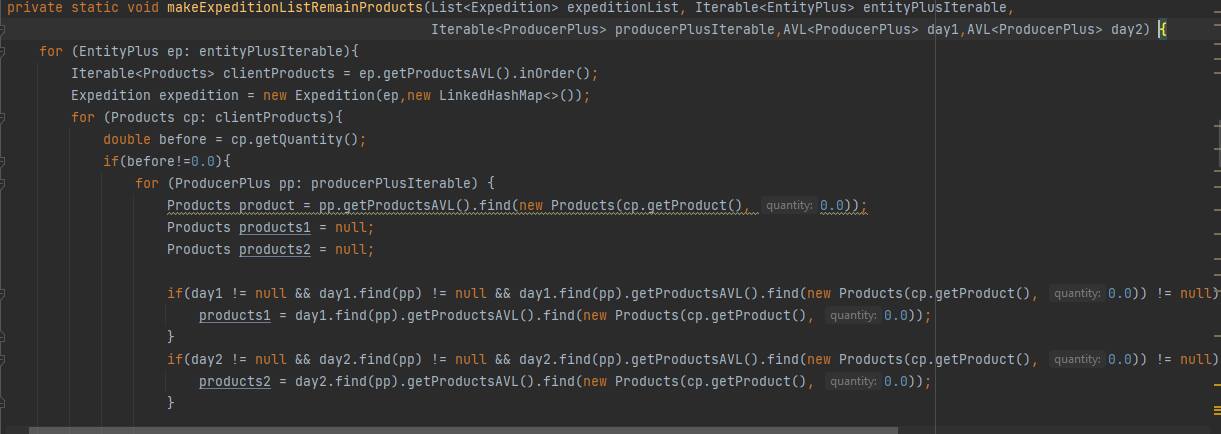
Estes dois métodos são bastantes semelhantes ambos criar Expedition para satisfer os pedidos dos clientes, apenas produtos totalmente satisfeitos, mas diferem apenas no facto de que no primeiro os dados são adicionados a uma lista preenchendo assim uma lista de expedição.

Neste método iremos percorrer o Iterable de “EntityPlus” , para cada elemento vamos fazer um “get” à sua arvore de Productos, obter o seu Iterable e percorre-la mas antes vamos criar uma “Expedition”. Ao percorrer o Iterable de Productos vamos percorrer também o Iterable de “ProducerPlus” para cada ProducerPlus verificamos se ele tem um producto para satisfazer o pedidos nos cliente pesquisando em três árvores, na do dia de hoje, nas sobras do dia de antes e nas sobras de dois dias atrás, de seguida entra numa zona de 6 ‘if´s’ verificando as combinações possiveis de haver ou não produtos disponiveis em cada Avl verificando se a quantidade que existe é suficiente para satisfazer completamente o cliente.

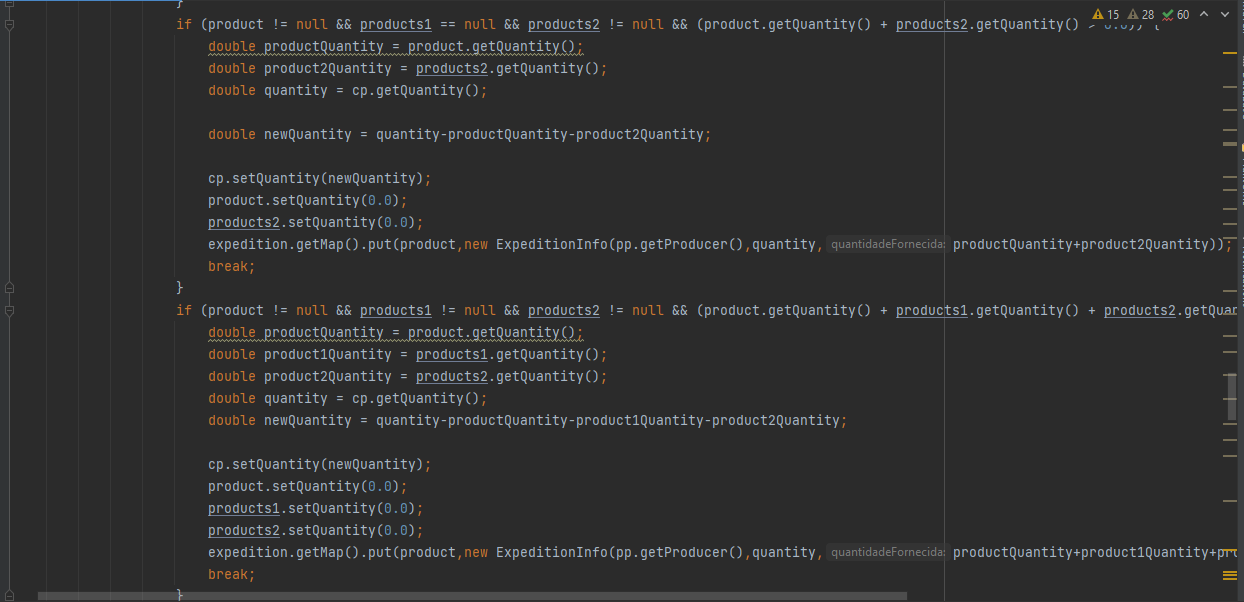
Os 6 ‘if´s’ cobrem todas as conbinações de haver ou não produtos disponiveis em cada Avl de seguida ao entrar num if vai pegar na avl mais antiga e verificar se o quantidade existente satisfaz ou não cliente caso não avança e verifica se as duas mais antigas juntamte fornencem , assim de modo a que os produtos disponiveis nas AVL mais antigas tenham prioridade uma vez que eles são sobras, atulizando sempre as quantidades quer nas AVL de Produtores quer nas de Consumidores.

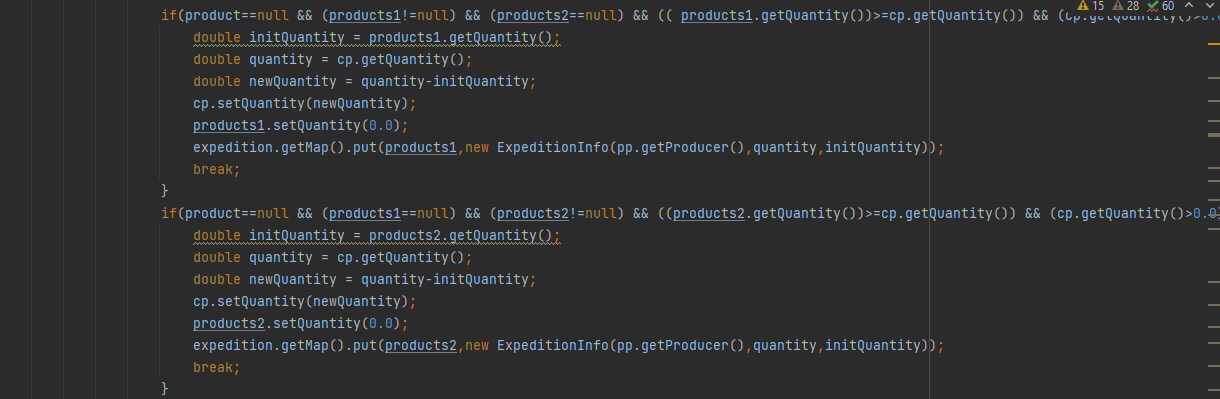
Este método é não deterministico uma vez que no pior caso ele vai ser O(n3log(n)) pois vai percorrer a lista de Cliente de seguida a lista de Productos e depois a lista de Produtores. No melhor Caso vai ser O(n2log(n)) uma vez que vai percorrer apenas a lista de Clientes e de Produtos sendo a de Productores completa na primeira iteração.

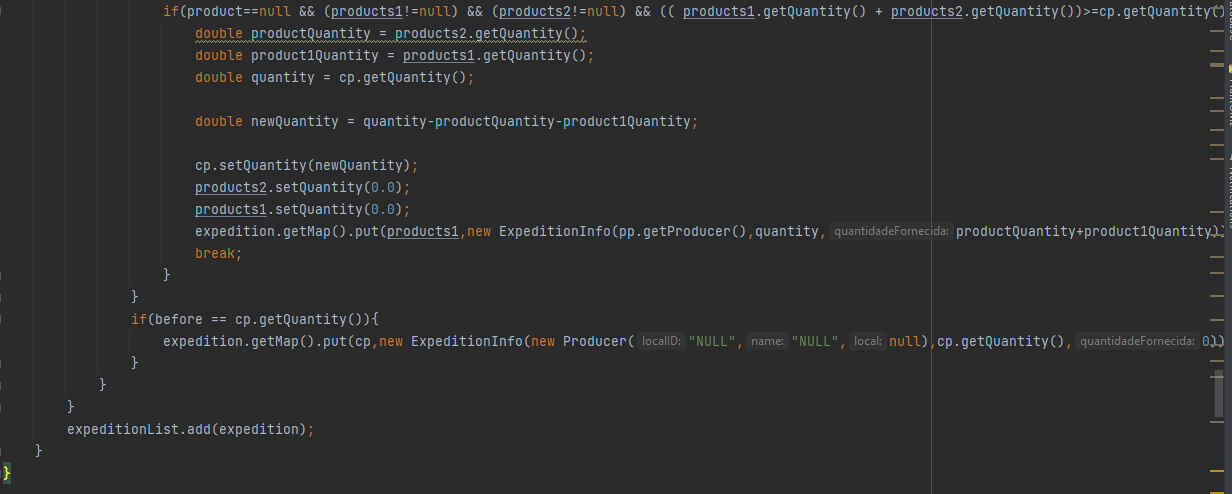
### makeExpedetionListRemainProducts e makeExpeditionRemain











Estes dois métodos também são bastantes semelhantes ambos criar também Expedition mas neste caso apenas para satisfazer parcialmente os clientes, no entanto e à semelhança do anterior também diferem no facto de o método “makeExpedetionListRemainProducts” irá adicionar elementos à Expedition List.

Neste método iremos percorrer o Iterable de “EntityPlus” , para cada elemento vamos fazer um “get” à sua arvore de Productos, obter o seu Iterable e percorre-la mas antes vamos criar uma “Expedition”. Ao percorrer o Iterable de Productos vamos percorrer também o Iterable de “ProducerPlus” para cada ProducerPlus verificamos se ele tem um producto para satisfazer o pedidos nos cliente pesquisando em três árvores, na do dia de hoje, nas sobras do dia de antes e nas sobras de dois dias atrás, de seguida entra numa zona de 6 ‘if´s’ verificando as combinações possiveis de haver ou não produtos disponiveis em cada Avl verificando mas neste caso não verifica se a quantidade que existe é suficiente para satisfazer completamente o cliente.

Os 6 ‘if´s’ cobrem todas as conbinações de haver ou não produtos disponiveis em cada Avl de seguida ao entrar num if vai suprir os produtos disponivéis caso existam e adicionar á expedition caso não exista produtos disponiveis vai ser adicionada uma expedição com Producer “Null” indicando que nenhum Produtor consegue fornecer o produto pedido.

Este método é não deterministico uma vez que no pior caso ele vai ser O(n3log(n)) pois vai percorrer a lista de Cliente de seguida a lista de Productos e depois a lista de Produtores. No melhor Caso vai ser O(n2log(n)) uma vez que vai percorrer apenas a lista de Clientes e de Produtos sendo a de Productores completa na primeira iteração.

# US309

Gerar uma lista de expedição para um determinado dia que forneça apenas com os N produtores agrícolas mais próximos do *hub* de entrega do cliente.

## Introdução à User Story

Esta *user story* pede para gerar uma lista de expedição com **restrições**, nomeadamente, com os **N produtores mais próximos da entrega do cliente.**

## Código

### *Restricted Dispatch List*

Uma imagem com texto

Descrição gerada automaticamente

Este método vai receber um certo dia para o qual é desejado realizar a lista de expedição, bem como o número de produtores mais próximos do *hub* de entrega do cliente. Primeiramente é declarada a lista que obterá a informação de cada cliente/cabaz e os seus produtos e produtores. De seguida é feita uma instância da classe “DispatchListWithoutResttriction” para poder utilizar os métodos que estão nesta classe.

Depois é criada uma AVL de “EntityPlus”, que é uma árvore que teria os Clientes e para cadu um destes, uma AVL de produtos encomendados por este, ordenada por dias. É criada outra AVL de “ProducerPlus” que teria os Produtores e neste caso uma AVL de produtos fornecidos ordenada por dia para cada Produtor. São criados dois Iterable referentes ás arvores anteriores e outras duas AVL de “ProducerPlus” chamadas de day1 e day2 que irão conter os restos, ou seja, os produtos que ainda não foram comercializados e que, de acordo com a quantidade de dias desde os quais foram inseridos no programa, serão colocados na AVL correta.

É feita ainda a instância da classe ClosestDistributionHub para de seguida determinar os hubs mais próximos de todos os pontos da rede.

De seguida ocorre um ciclo for, que inicia no dia 1 e termina no dia recebido por parâmetro, de modo a calcular a lista de expedição desse dia. Aqui iremos também obter a lista de clientes/cabazes, bem como os produtores e os seus produtos no dia pretendido. O programa entra então noutro ciclo for onde, para cada cliente na lista previamente obtida, vai obter o seu hub mais próximo e inserir o cliente na AVL. De seguida vamos obter os N produtores mais próximos e guardá-los na sua AVL.

Se o i for igual ao dia, vão ser chamados os métodos “makeExpeditionListFullProducts”, “makeExpeditionListRemainProducts” e caso i seja diferente de dia vão ser chamados os métodos “makeExpeditionFull” e “makeExpeditionRemain”, e atualiza as AVL day1 e day2, tornando a AVL day2 igual a day1 e a Avl day1 passa a ser igual à “actualDayProducerPlus” que, após executar os métodos, vai contenter apenas as sobras do dia i.

No fim é retornada a “dispatchList”, uma Lista de “Expedition” (um objeto que contém um “EntityPlus” referente a um cliente e um Mapa. Neste Mapa, cada produto pedido é uma key e para cada produto existe uma entry com “ExpeditionInfo”, que contêm informação acerca do Produtor que forneceu aquele produto, a quantidade pedida por aquele cliente e a quantidade fornecida que em alguns casos por não satisfazer totalmente o pedido do cliente e em outros casos ainda pior pode não ser sequer minimanente satisfeita.

### *Get N Closest Producers to Hub*

Uma imagem com texto

Descrição gerada automaticamente

Este método recebe o hub mais próximo de um certo cliente (hub de entrega), para o qual vai determinar os N (producersNumber) produtores mais próximos do hub. Recebe ainda um objeto da classe DayProducer que contém todos os produtores e as informações dos seus produtos num certo dia.

Primeiramente vai ser criada uma AVL que irá conter os produtores mais próximos e que será retornada. De seguida vai ser determinado o caminho mais curto desde o hub até todos os pontos da rede. No primeiro ciclo for irá ser feita a seleção dos produtores bem como a sua organização descendente por distância ao hub, através da verificação: se o vértice de destino naquele caminho da lista “paths” é ou não uma instância da classe “Producer”, se for vai ser colocado no Mapa criado previamente. O segundo ciclo for serve para selecionar o número desejado de produtores mais próximos e adiconá-los à AVL que vai ser retornada.

## Complexidade

Ao analisar os métodos anteriormente representados, verificam-se as seguintes complexidades:

* Restricted Dispatch List:
* *Get N Closest Producers to Hub*:

# US310

Para uma lista de expedição diária gerar o percurso de entrega que minimiza a distância total percorrida.

## Código

O construtor desta clase recebe o grafo onde estão todos os clientes, produtores e empresas, a lista de expedição, um mapa onde ficará os cabazes entregues em cada hub e uma mapa onde esta o hub mais proximo de cada cliente, produtor e empresa.

Text

Description automatically generated

Este metodo ira prencher os atributos producerPerHub, hubMap, producerMap e clientHub, estes atributos irão auxiliar a resolução desta user storie. Os atributos hubMap e producerMap são mapas que contem hubs e produtores respeitivamente e um boolean a representar se já foi visitado. O atributo producersPerHub é um mapa que contem um set de produtores, estes produtores são todos ao quais temos de passar antes de ir ao respeitivo hub.

Text

Description automatically generated

### *minimumDistance*

O metodo minimumDistance recebe uma lista onde será colocado o caminho mais curto que permita realizar a entrega dos cabazes fornecidos por cada produtor na lista de expedição aos seus respeitivos hubs. Primeiro ira verificar que nenhum dos atributos necessarios sejam null, vamos a acrescentar o ultimo hub a lista “route” (sendo esse o nosso ponto de partida), de seguida serão inicializados as listas que o algoritmo shortestPaths irá usar (este metodo irá utilizar o algoritmo de dijkstra para determinar todos os caminhos mais curtos desde o ponto atual, neste caso o ponto de partida), ainda é inicializado uma lista onde vamos colocar todos os produtores por onde já passamos. De seguida vai entrar no ciclo while, que será executado tantas vezes como o numero de produtores mais o numeor de hubs, depois será executado o metodo shortestPaths e o metodo closestPointOfInterest, este retorna a Entity da ultima posição do caminho selecionado, acrescenta o caminho selecionado a lista route e finalmente limpa as lista utilizadas pelo metodo shortestPaths e diminui a variavel numberOfProducersAndHubs.

Text

Description automatically generated

### *closestPointOfInterest*

O metodo closestPointOfInterest recebe uma lista com lista que contem todos os caminhos calculados com o metodo shortestPaths, a lista minDistance com a distancia percorrida em cada percurso, a lista dos produtores visitados e uma lista que irá ser prenchida com o percurso selecionado. O metodo primeiro descarta o caminho de comprimento 1 é de seguida entra num ciclo while que será executado pelomenos ate a lista de caminhos acabar. Primeiro vai ver qual é o caminho mais curto e o seu destino, caso seja um Hub vamos verificar se ele é um hub ao qual temos de ir, se já estivemos no hub e se já fomos a todos os produtores que fornecem aquele hub, Caso cumpra estos requisitos vamos trocar de false para true o valor desse hub no hubMap, vai acrescentar o caminho ao resultPath e vai retornar o ultimo elemento desse caminho. Caso seja um produtor vai verificar se é um produtor ao qual temos de ir e se ainda não passamos por ele, se estas condições se cumprirem vamos tocar para true o valor desse produtor no producerMap vai acrescentar o caminho ao resultPath, irá acrescentar esse produtor a lista de produtores visitados e vai retornar o ultimo elemento desse caminho. Caso o destino desse caminho não seja um produtor ou um hub vamos retirar esse caminho da lista dos caminhos.

Text

Description automatically generated

## Complexidade

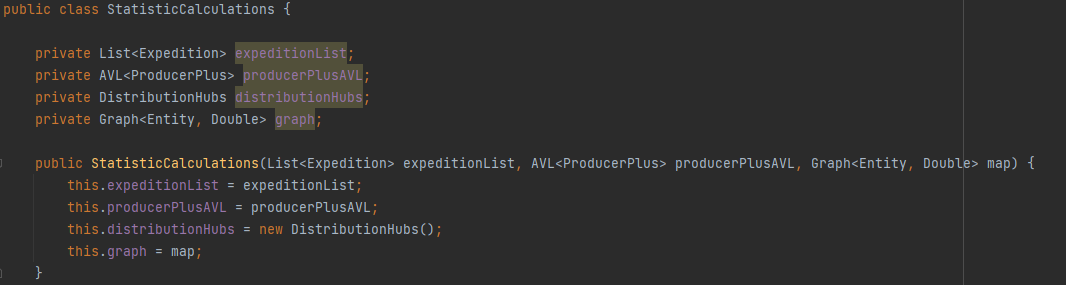
A complexidade destes metodos são as seguintes:

* closestPointOfInterest :
* shortestPaths:
* minimumDistance:

# US311

- Para uma lista de expedição calcular estatísticas: • por cabaz: nº de produtos totalmente satisfeitos, nº de produtos parcialmente satisfeitos, nº de produtos não satisfeitos, percentagem total do cabaz satisfeito, nº de produtores que forneceram o cabaz. • por cliente: nº de cabazes totalmente satisfeitos, nº de cabazes parcialmente satisfeitos, nº de fornecedores distintos que forneceram todos os seus cabazes • por produtor: nº de cabazes fornecidos totalmente, nº de cabazes fornecidos parcialmente, nº de clientes distintos fornecidos, nº de produtos totalmente esgotados, nº de hubs fornecidos. • por hub: nº de clientes distintos que recolhem cabazes em cada hub, nº de produtores distintos que fornecem cabazes para o hub.

A classe “StatisticCalculation” recebe como parâmetro “List<Expedition> expeditionList” a lista de expedição na qual o usuário pretende efetuar os cálculos referentes a esta US, recebe um parâmetro “AVL<ProducerPlus> producerPlusAVL” cujo intuito é calcular os produtos que se encontram esgotados e um parâmetro “Graph<Entity,Double> map” que é utilizado para calcular a distancia do cliente aos Hubs existentes. O atributo “distributionHubs” é uma instância da classe “DistributionHubs”.



A classe contém quatro métodos, para calcular as estatísticas pedidas

* + calculateHumperStatistics
  + calculateClientStatistics
  + calculateProducerStatistics
  + calculateProducerStatistics

Método calculateHumperStatistics:

O método começa por inicializar um Map<Entity,StatisticsPerHumper>, statisticsPerHumper, que vai retornar os valores calculados.

Para cada Expedition da ExpeditionList passada como parâmetro, o método vai verificar se a mesma é uma entidade que existe em statisticsPerHumper, se existir, isto significa que vai aceder à segunda parte da entidade, onde se encontram armazenados os produtos parcialmente satisfeitos e os produtos não satisfeitos.

Dentro de um ciclo “for” vão ser efetuadas verificações com a finalidade de incrementar os valores anteriormente referidos.

Após o ciclo são guardados os valores em statisticsPerHumper.

No caso da entidade ainda não existir em statisticsPerHumper, os valores anteriores vão ser inicializados a zero e depois incrementados por um ciclo “for” à semelhança da parte anterior.

Método calculateClientStatistics:

O método começa por inicializar um Map<Entity,StatisticPerClient>, statistics, que vai retornar os valores calculados.

Para cada Expedition em ExpeditionList cira uma variável statistic do tipo StatisticPerClient. De seguida vai ser verificado se entidade de Expedition existe em statistic, se não existir é criada, inicializada e inserida na mesma.

O próximo passo é realizar o cálculo entre as quantidades pedidas e as quantidades fornecidas e incrementar os devidos valores de acordo com os resultados.

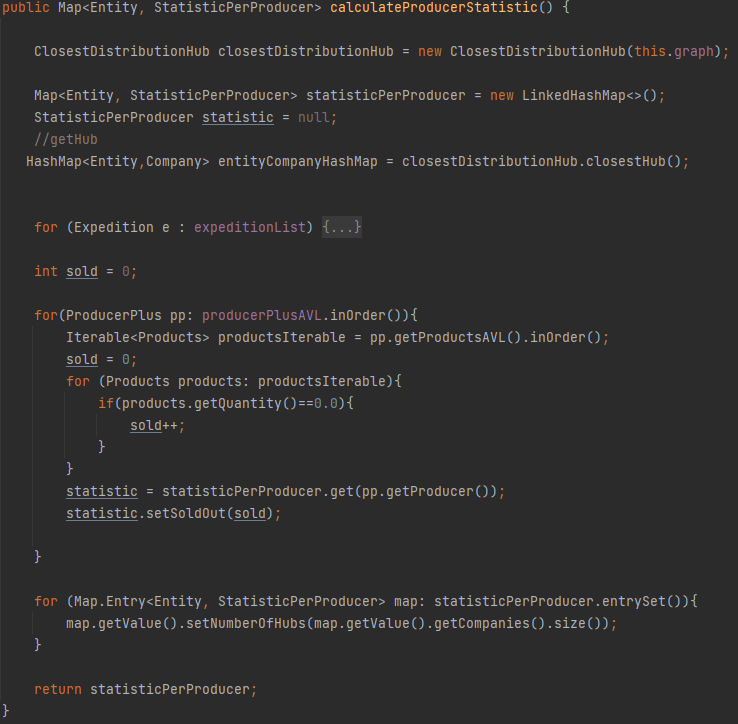


Método calculateProducerStatistics:

O método começa por declarar e inicializar uma variável ClosesestDistribuicioHub closesestDistribuicioHub a partir da variável graph pertencente à classe.

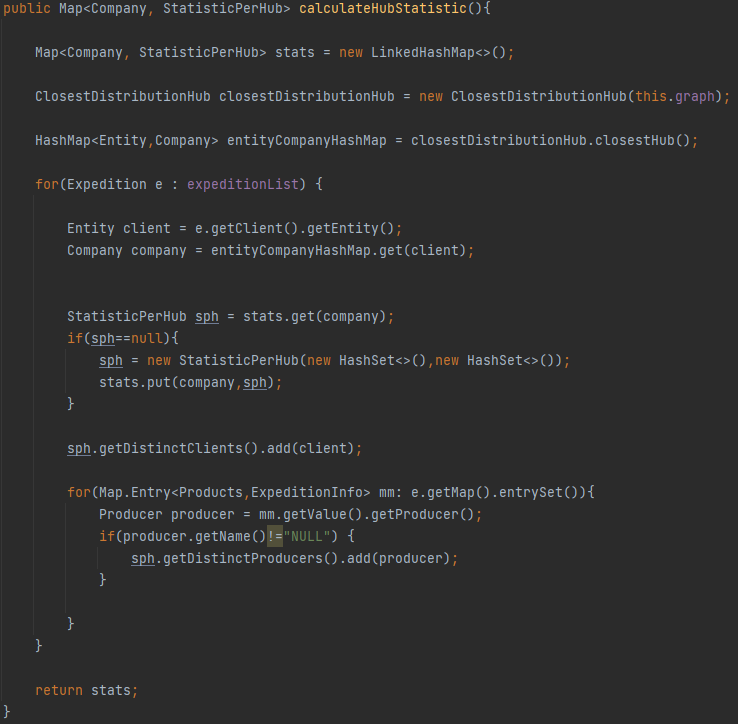
De seguida, para cada Expedition é feito o cálculo dos valores pretendidos, à semelhança dos métodos anteriores.

O último passo do método é calcular a partir do atributo de classe producerPlusAVL os artigos esgotados.



Método calculateHubStatistic:

O método, à semelhança do método anterior calcula o “closestDistribuitionHub”. De seguida, é calculado para cada Expedition o número de produtores distintos.



## Complexidade

* Calculate Humper Statistics:
* Calculate Client Statistics:
* Calculate Producer Statistics:
* Calculate Producer Statistics: