

Estruturas de informação

2022/2023

4 dezembro

### Instituto Superior de Engenharia do Porto

**Pedro Teixeira, 1211184 JoãoFernandes,1211681** **David Mendonça,1211572** **Afonso Machado, 1190326**

**1**

# Índice

Contents

[Índice 2](#_Toc123589170)

[Introdução 3](#_Toc123589171)

[US 307 4](#_Toc123589172)

[Enunciado 4](#_Toc123589173)

[Implementação e análise de complexidade 4](#_Toc123589174)

[Diagrama de Classes 7](#_Toc123589175)

[US 308 8](#_Toc123589176)

[Enunciado 8](#_Toc123589177)

[Implementação e análise de complexidade 8](#_Toc123589178)

[Diagrama de classes 11](#_Toc123589179)

# Introdução

Este relatório tem o propósito de demostrar os conteúdos que foram abordados e praticados durante a realização do segundo Sprint do Projeto Integrador do 1ºsemestre do 2ºano de Licenciatura em Engenharia Informática.

Relativamente à cadeira de ESINF este Sprint continha User Stories que envolviam o desenvolvimento de funcionalidades que usassem a linguagem JAVA e que permitiriam gerir a informação sobre vários clientes e produtores numa rede de produção biológica e as respetivas encomendas e stock de cada produtor ou Hub. A informação encontrava-se em arquivos de texto com formato CSV.

As funcionalidades incluem o carregamento de um ficheiro contendo todas as encomendas e stock existente, por parte dos clientes e produtores, respetivamente.

Ao longo deste projeto, foram utilizados Graphs e MapGraphs que foram desenvolvidos durante as aulas pratico-laboratoriais tendo utilizado também os algoritmos disponibilizados nas mesmas.

Para cada uma dessas funcionalidades vamos demonstrar o respetivo diagrama de classes, algoritmo utilizados e análise de complexidade.

# US 307

## Enunciado

* Importar a lista de cabazes

## Implementação e análise de complexidade

No Sprint 1 houve a importação de informação acerca de clientes e produtores a partir de um ficheiro CSV. Neste 2ºSprint vamos proceder para a importação tanto das encomendas por parte dos clientes, como o stock disponibilizado pelos produtores, em dados dias.

Text

Description automatically generated

Figura 1 – método importBasketList

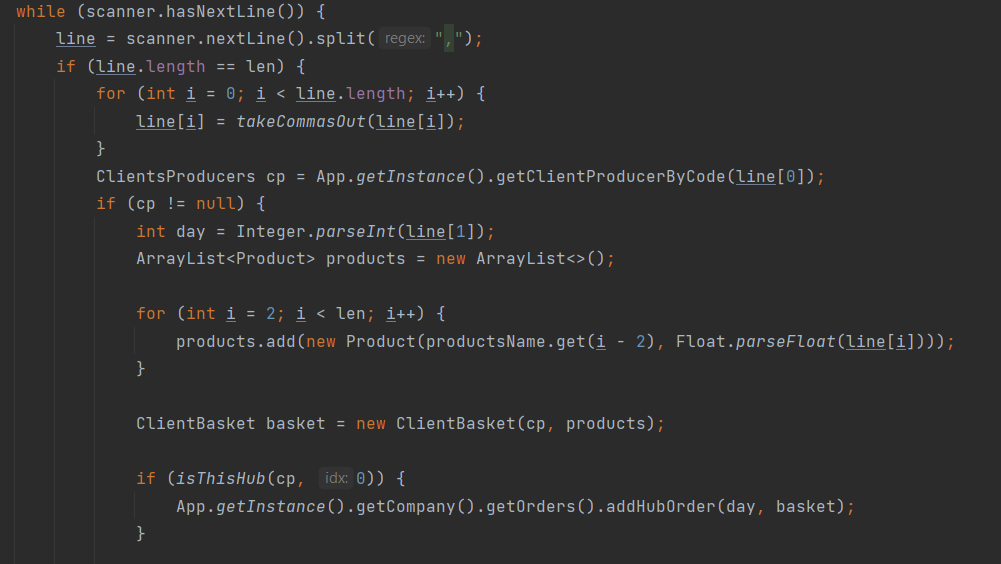


Figura 2 – continuação figura 1

Text

Description automatically generated

Figura 3 – fim do método importBasketList

A importação do conteúdo do ficheiro é feita a partir de um método estático (importBasketList) da classe FileReaderApp. O ficheiro é digitado previamente a partir da interface que é apresentada ao utilizador.

Cada linha do ficheiro para ser corretamente validade deve seguir a seguinte estrutura:



Eventualmente pode existir mais produtos.

Em primeiro lugar o scanner lê uma linha do ficheiro.

De seguida, divide o seu conteúdo a partir de um identificador, que neste caso é a “,”. Se existir aspas num nome, estas são retiradas e adicionadas de volta ao array.

A partir do conteúdo dividido sabemos que a posição 0 do array está identificado o cliente, produtor ou hub.

É retirada da linha o dia que a encomenda ou colocação de stock vai estar presente.

Tendo em conta se é hub/produtor ou cliente, colocamos em diferentes stores que podem ser acedidas na classe App.

Tendo em conta o pior caso possível, a análise de complexidade desta US é a seguinte:

Na figura 1, evidenciamos a presença de um pequeno ciclo for, que tem grau de complexidade O(i-2), sendo i o número de elementos do array. Na figura 2, temos O(i) e O(i-2) na primeira e segunda estrutura “for” respetivamente. Por fim, na figura 3, não existe nada mais no algoritmo que comprometa a análise da complexidade. Concluindo, a complexidade total tendo em conta o pior caso possível é O(3i-4).

## Diagrama de Classes

A picture containing graphical user interface

Description automatically generated

Figura 4 – diagrama de classes US 307

# US 308

## Enunciado

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

## Implementação e análise de complexidade

Partindo do ponto que previamente foi importado, tanto informação quanto aos clientes e produto, tanta informação sobre a encomendas e stock existente, temos as condições necessárias para criar uma lista de expedição de um certo dia.

Na interface desta funcionalidade é pedido ao utilizador o dia em que pretende saber o que foi expedido na totalidade, ou seja, que clientes foram satisfeitos, e que produtores forneceram os produtos das suas encomendas.

Antes de criar a lista de expedição do dia selecionado, temos antes de tudo de calcular os excedentes de dias, se o dia selecionado não for o primeiro que é apresentado no ficheiro, ou seja, fazer uma espécie de simulação das listas de expedição dos dias anteriores, e assim desta forma também conseguimos saber se os hub (se estes forem definidos previamente) terão produtos expostos para venda, o que simboliza que alguma das encomendas de um dado hub foi correspondida, mesmo que não seja na sua totalidade.

Text

Description automatically generated

Figura 5 – método CalculateSurplus

Neste excerto, é onde iniciamos o cálculo/simulação dos excedentes de stock. Retiramos da chave os dias que são iguais e maiores que o dia inserido, e enviamos para um método que através de recursividade, dar-nos-á acesso aos excedentes dos dois dias anteriores ao dia que utilizador digitou.

Text

Description automatically generated

Figura 6 – início do método CalculateSurplusDay

Esta figura corresponde ao início do método recursivo que tem como objetivo criar os excedentes de stock. Começa por identificar o stock do primeiro dia que estava no keyset que restringimos no método da figura 1, e também se existirem os excedentes do dia anterior e 2 dias anteriores, respetivamente.

Text

Description automatically generated

Figura 7 – continuação figura 6

Text

Description automatically generated

Figura 8 – método recursivo isThisHub

De seguida, começamos por verificar se quem faz encomenda é um hub, e se for o caso, o booleano isHub fica a true.

Depois vamos ter duas estruturas “for” encadeadas que servem para percorrer em cada encomenda cada um dos produtos encomendados. Para cada produto encomendado verificamos no stock existente e excedentes se existe algum produtor que o tenha, independentemente se consegue corresponder à totalidade do pedido, daí o termo “sem restrições” (que também se dirige a não haver limitações de distância, ou seja qualquer produtor pode servir qualquer cliente).

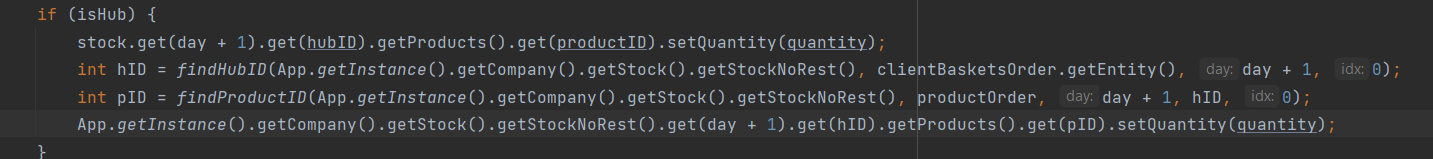


Figura 9 – atualização de stock do dia seguinte de um hub

Depois de atualizado o stock sofrido pela montagem de um cabaz para uma dada encomenda é verificado se o recetor é um hub a partir do booleano que foi alterado ou não no início de cada encomenda, se for, no dia seguinte esse hub vai ter o stock atualizado com aquilo que ele recebeu da encomenda feita.

Tendo em conta o pior caso possível, a análise de complexidade desta US é:

Na figura 8, conseguimos observar que existem duas estruturas “for”, logo podemos concluir que inicialmente temos O(), considerando n, o número de encomendas.

Dentro dessas estruturas são chamados os métodos isThisHub (figura 8) e FindHubID que têm a mesma complexidade já que são recursivos e têm praticamente o mesmo percurso, só que retornam objetos diferentes. A complexidade destes dois algoritmos é O(), sendo n o número de hubs na HubStore.

Text

Description automatically generated

Figura 10 – método recursivo findProductID

Como acontece anteriormente dentro da segunda estrutura “for” são chamados métodos recursivos com a mesma praticidade dos dois anteriores, só que retornam objetos diferentes, que neste caso são os índices de produtor, e índices de produtos com stock para corresponder a encomendas feitas. A complexidade destes algoritmos também são O , tendo em conta que n simboliza o número de produtores e o número de produtos, respetivamente.

## Diagrama de classes

A picture containing graphical user interface

Description automatically generated

Figura 11 – diagrama de classes US 308