

# Assignmnet 2

André Pedrosa [85098], João Abílio [84732]

## Recuperação de informação

Departamento de Eletrónica, Telecomunicações e Informática

Universidade de Aveiro

13 de outubro de 2019

## 1 Introdução

Este relatório apresenta uma explicação do trabalho desenvolvido para o segundo assignment da disciplina "Recuperação de Informação", explicando as decisões tomadas e o funcionamento da solução.

Esta segunda entrega tem como objetivo fazer incrementos à entrega anterior de maneira a aplicar o método SPIMI durante o método de indexação, aplicar pesos *td-idf* aos termos usando o esquema de indexação *Inc.ltc* e adicionar ao indexar as posições dos termos nos documentos nos quais ele aparece.

No fim serão apresentados resultados às questões colocadas no enunciado do assignment com o index resultante do programa.

Devido ao elevado número de classes criadas, o diagrama de classes vai ser dividido em vários que vão sendo apresentados ao longo do relatório. Estes diagramas foram gerados através do IDEA IntelliJ, consequentemente em anexo é disponibilizada a legenda da convenção usada.

## 2 Decisões de implementação

Durante a implementação deste assignment não foi dada atenção a questões de memória, no entanto seguimos uma aproximação iterativa em que o pedido de informação (conteúdo de ficheiros a ler, documentos, ...) pode ser condicionado segundo as limitações de memória em implementações futuras facilmente.

Partes da nossa solução foram moduladas já a pensar nos futuros assignments, possibilitando a indexação ser feita em diferentes formatos de documentos e a informação presente no index poder variar.

### 3 Data Flow

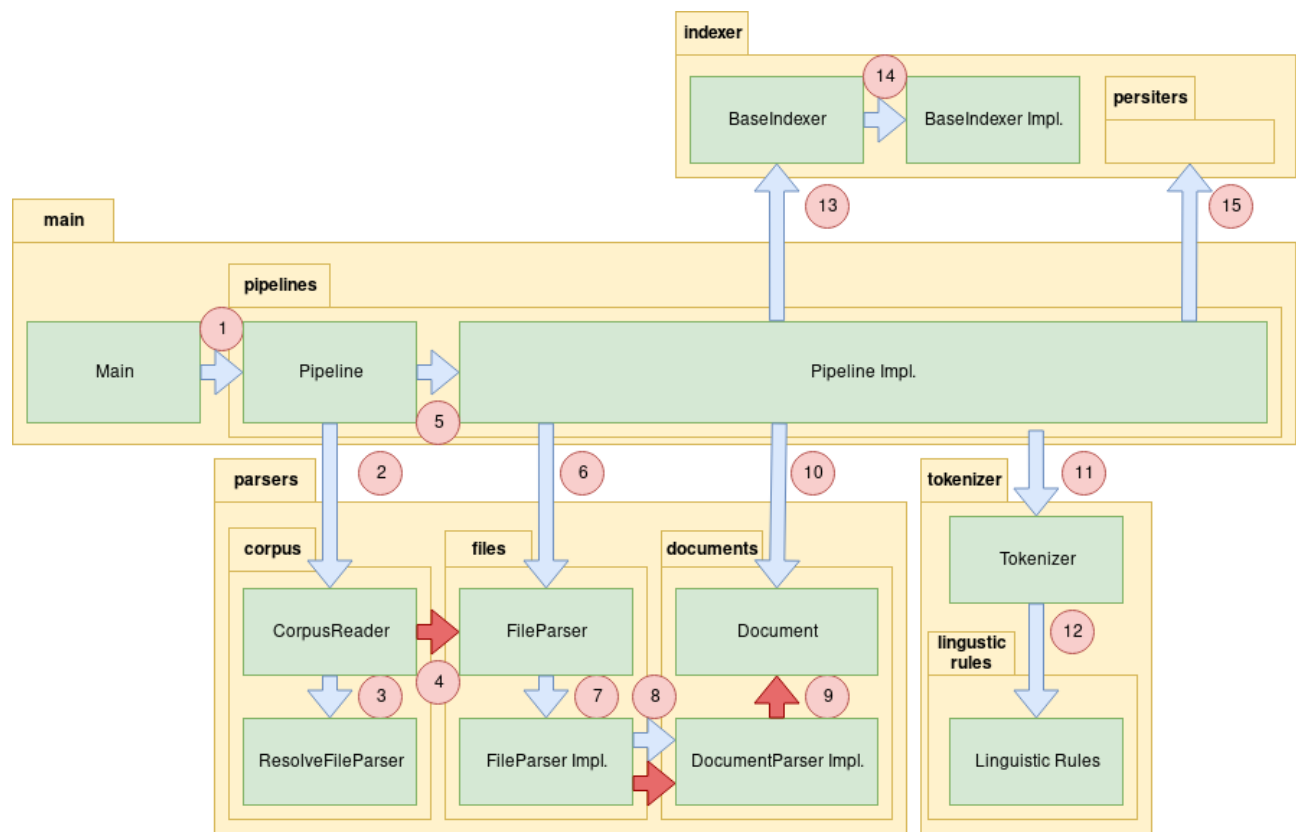


Figure 1: Diagrama de sequência da solução

O pipeline da nossa solução está implementado no método main da classe Main. Primeiramente, aqui é instanciado o respectivo Tokenizer e o Indexer.

O Main instância um classe do tipo CorpusReader, sobre a qual fará um for each, no qual em que cada iteração receberá um FileParser. O Main itera também com um for each sobre o FileParser. Para cada ciclo do último for each mencionado, a classe FileParser está continuamente a ler linhas do ficheiro até ter um documento válido. Neste momento, passa o conteúdo lido a um DocumentParser que extrai do documento a informação importante (identifier e outros campos) criando uma classe Document, a qual será devolvida para o for each no método main.

Este documento é registado no index (associar um document id ao identifier do documento) e posteriormente os seu termos são indexados.

Por último, as estruturas internas do index são escritas para disco.

## 4 Packages

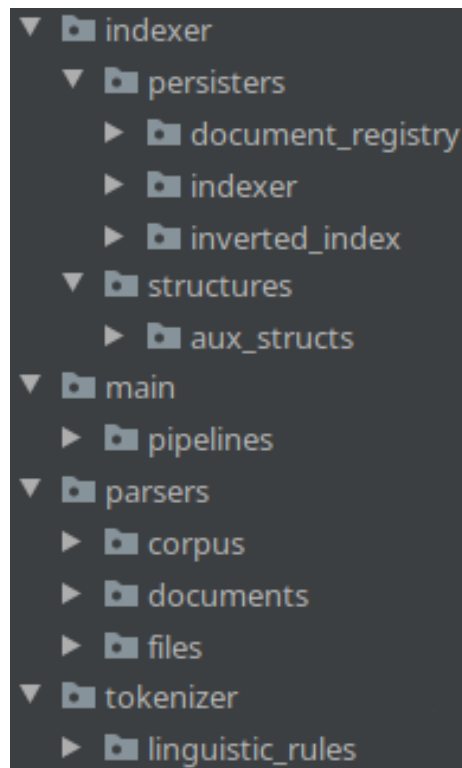


Figure 2: Árvore de packages da solução

Nesta secção vai ser apresentada uma descrição para cada package presente na nossa solução apresentando as principais classes e os seus principais métodos.

### 4.1 main

Neste package encontra-se a classe com o método main onde é feito o processamento dos argumentos e opções do programa e onde é definido o pipeline de processamento.

Tem ainda a classe responsável por consultar o index de maneira a obter os dados para responder às questões propostas no enunciado.

## 4.2 parsers

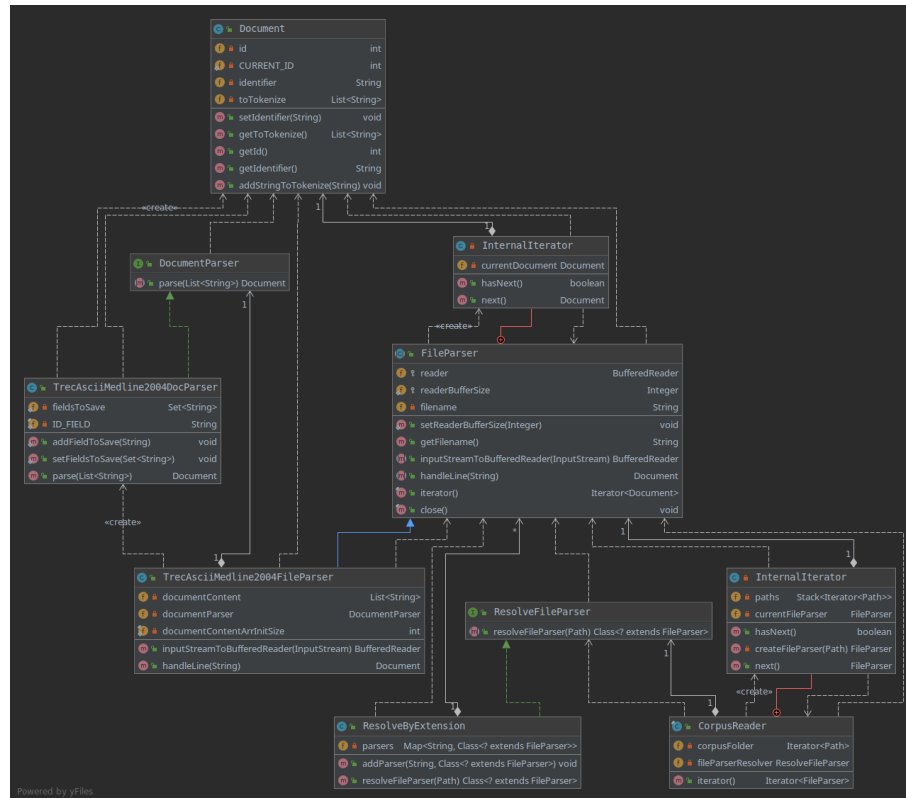


Figure 3: Diagrama de classes do package *parsers*

Este package não sofreu grandes alterações relativamente à entrega anterior, apenas foi alterado algumas mensagens de erro, os id dos documentos agora começa em 0 e algumas operações sobre strings obtidas dos ficheiros do corpus /forem removidas, como `.trim()`, para tentar reduzir o número de instanciações da class `String`.

### 4.3 tokenizer

Este package também não sofreu grandes alterações relativamente à entrega anterior, apenas foi alterado a maneira como o conteúdos dos campos são transformado em tokens. Essa operação foi agrupada numa única string variável do tipo `String` para novamente para tentar evitar o número de instanciações da classe `String`

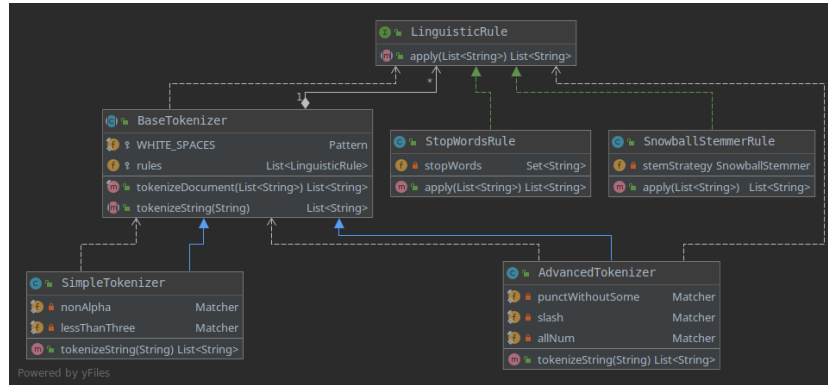


Figure 4: Diagrama de classes do package *tokenizer*

## 4.4 indexer

Package com as classes que armazenam em memória o index invertido e a associação entre o id de um documento e o seu identifier.

Aqui está presente a class *BaseIndexer* que serve como classe base para diferentes implementações de indexers. O index invertido é guardado numa estrutura do tipo mapa, permitindo ao programador definir a implementação desta interface, sendo por defeito usado um *HashMap*. A classe base referida é genérica o que possibilita que sejam criados diferentes indexers com a mesma estrutura, o que leva às classes descendentes a implementar o método *indexTerms* que guarda os termos de um documento no index invertido, com as estruturas específicas desse indexer.

### 4.4.1 indexer.structures

Neste package estão os blocos que constroem o index invertido e que permitem a extensibilidade do mesmo. Tanto a chave do index invertido como o valor presente na lista associada descende do tipo *Block* que possui uma key (no caso do termo é o próprio term e nos documentos o seu id) pela qual é comparável entre si. Em casos em que seja necessário ter mais informação associada (contagens por exemplo) a classe *BlockWithInfo*, descendente de *Block*, permite isso mesmo.

Para distinguir termos de documentos foram criadas as interfaces *BaseTerm* e *BaseDocument*, logo classes que guardam informação sobre termos devem descender do tipo *Block* e implementar a interface *BaseTerm* e classes que guardam informação sobre documentos devem descender do tipo *Block* e implementar a interface *BaseDocument*.

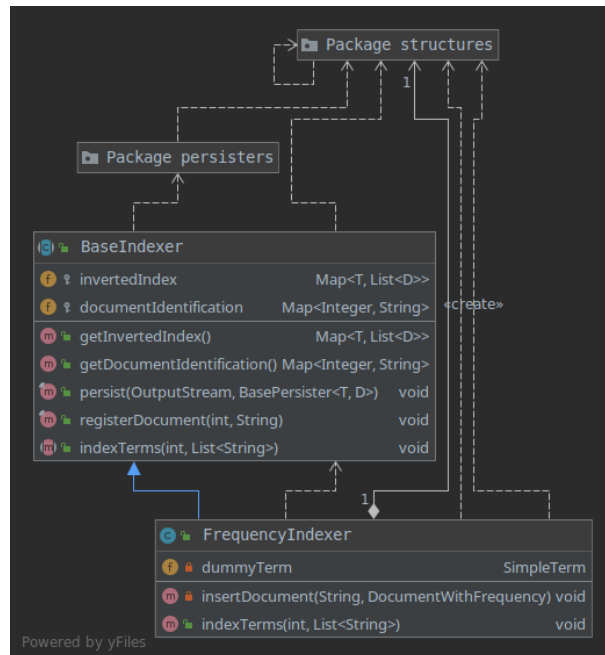


Figure 5: Diagrama de classes do package *indexer*

#### 4.4.2 indexer.persisters

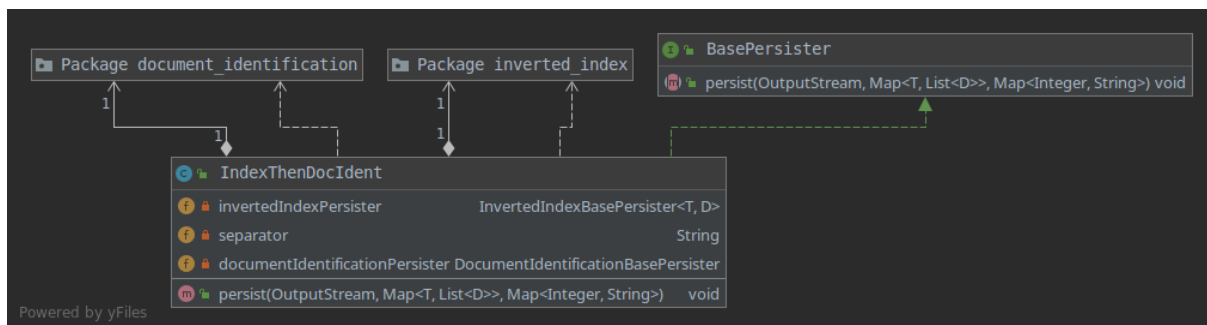


Figure 7: Diagrama de classes do package *indexer.persisters*

Aqui encontram-se as classes responsáveis por implementar as diversas estratégias de guardar as estruturas internas da class `BaseIndexer` para disco. Como este indexer apresenta duas estruturas internas, damos a possibilidade de criar diferentes estratégias para cada estrutura, assumindo sempre que guardamos para o mesmo ficheiro as duas estruturas. A classe `BaseIndexer`, no método *`persist`*, recebe um `BasePersister` que irá aplicar a estratégia para guardar ambas as estruturas.

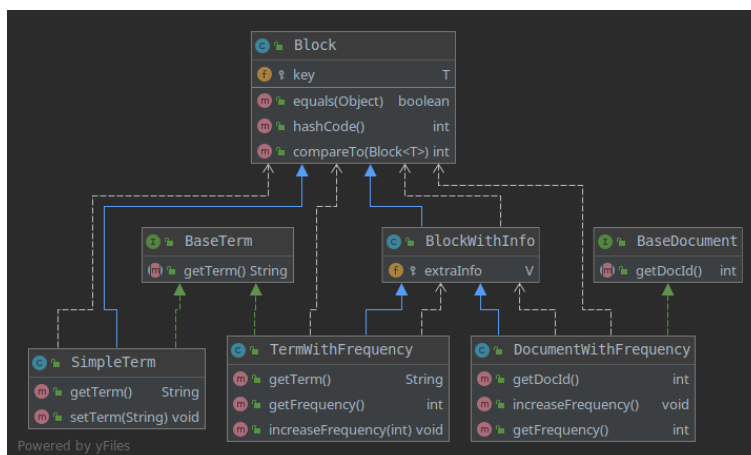


Figure 6: Diagrama de classes do package *indexer.structures*

## 5 Resultados

Resultados usando apenas o ficheiro 2004\_TREC\_ASCII\_MEDLINE\_1.gz

	<b>SimpleTokenizer</b>	<b>AdvancedTokenizer</b>
Tempo de indexação (mm:ss)	3:13	2:09
Tamanho do index em disco (MB)	238	209
Tamanho do vocabulário	254914	427035
Primeiros 10 termos (em ordem alfabética) que aparecem em apenas um documento	aaaa aaaai aaaasf aaaat aaab aaact aaaaction aaaga aaah aaahc	000case 000diseasegen 000for 000g 000gener 000iu 000kb 000mer 000meter 000molecularweight
Dez termos com a maior frequência nos documentos	and : 1014861 the : 1011732 with : 311814 for : 304357 from : 117323 patients : 112027 human : 106054 cell : 90208 cells : 85435 study : 84058	cell : 144666 patient : 137526 effect : 134752 human : 109488 studi : 106189 use : 87725 activ : 87489 rat : 81501 diseas : 79692 treatment : 78885

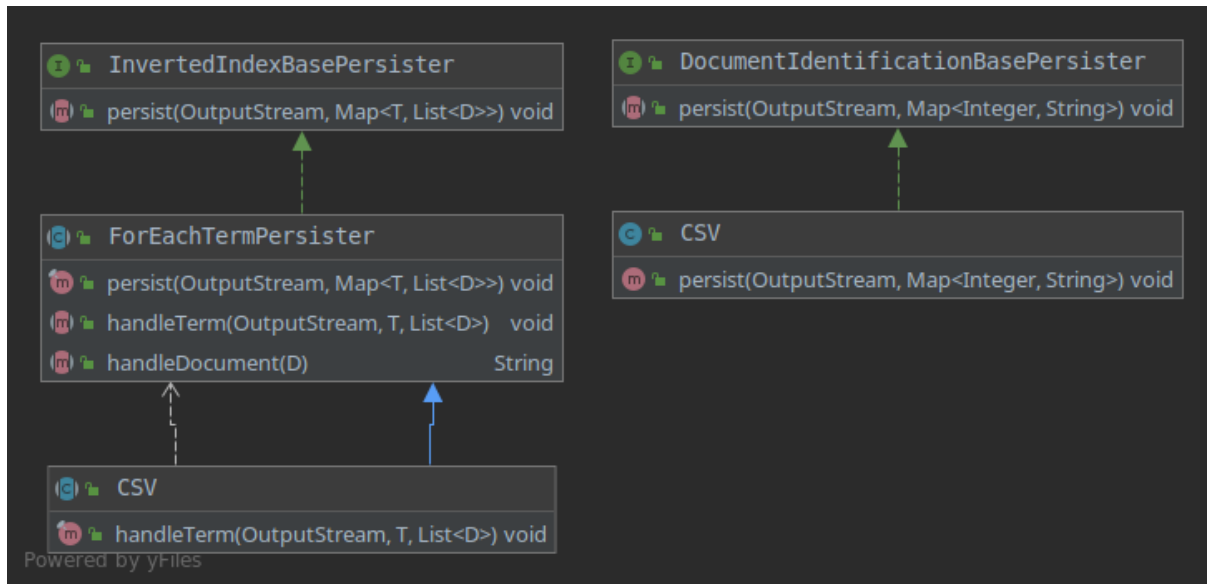




















Figure 8: Diagrama de classes do package *indexer.persisters.inverted\_index* à esquerda e do package *indexer.persisters.document\_identification* à direita

## 6 Anexos

Item	Description
	The green arrow corresponds to the <code>implements</code> clause in a class declaration.
	The gray arrow corresponds to a call from the origin class to a method of the destination class.
	The blue arrow corresponds to the <code>extends</code> clause in a class declaration.
	This sign appears for the inner classes.



Icon	Description
	Class
	Abstract class
	Interface
	Method/function
	Interface method
	Static method
	Constant
	Field
	Property
	Final annotation
Visibility modifiers	
	Private
	Protected
	Public
	Static