

Assignmnet 1

André Pedrosa [85098], João Abílio [8]

Recuperação de informação

Departamento de Eletrónica, Telecomunicações e Informática

Universidade de Aveiro

16 de outubro de 2019

1 Introdução

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

A linguagem de programação usada foi o Java e o programa desenvolvido tem como objetivo indexar uma coleção de documentos, criando um index invertido que faz a associação entre termos e os documentos nos quais e quantas vezes aparece-se.

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

Ao longo do relatório serão apresentados diagramas de classe que foram gerados através do IDEA IntelliJ, consequentemente em anexo é disponibilizada a legenda dos vários icons.

JAVA DOC

2 Decisões de implementação

Durante a implementação deste assignment não foi dada atenção a questões de memória, no entanto seguimos uma aproximação de streams em que o pedido de informação a esta informação pode ser condicionado segundo as limitações de memória.

Partes da nossa solução foram moduladas já a pensar nos futuros assignments, possibilitando a indexação ser feita a diferentes formatos de documentos e a informação presente do index poder variar.

3 Packages

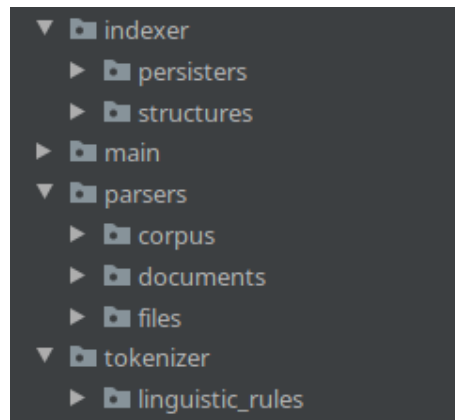


Figure 1: Árvore de packages da solução

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

3.1 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.

3.2 parsers

Neste package encontram-se as classes com a responsabilidade de fazer o processamento do corpus. Este processamento engloba percorrer a pasta do corpus, abrir os vários ficheiros, retirar os documentos dos ficheiros e recolher as partes a indexar dos documentos.

3.2.1 parsers.corpus

Package onde é feita a iteração sobre os ficheiros a serem indexados, criando as classes necessárias para as classes seguintes poderem ler destes ficheiros.

A classe `CorpusReader` implementa a interface `Iterable` o que permite receber os ficheiros a processar numa aproximação do tipo stream, como foi mencionado anteriormente. Quando é chamado o método `hasNext` do iterador da classe `CorpusReader`, a pasta do corpus é percorrida recursivamente (usando uma stack para continuar a recursividade nas chamadas seguintes) até encontrar um ficheiro, o qual será retornado na próxima chamada do método `next`.

Nesta altura é necessário escolher o `FileParser` (mencionado mais à frente) adequado para o tipo de ficheiro para isso a classe `CorpusReader` tem uma interface `ResolveFileParser` que é

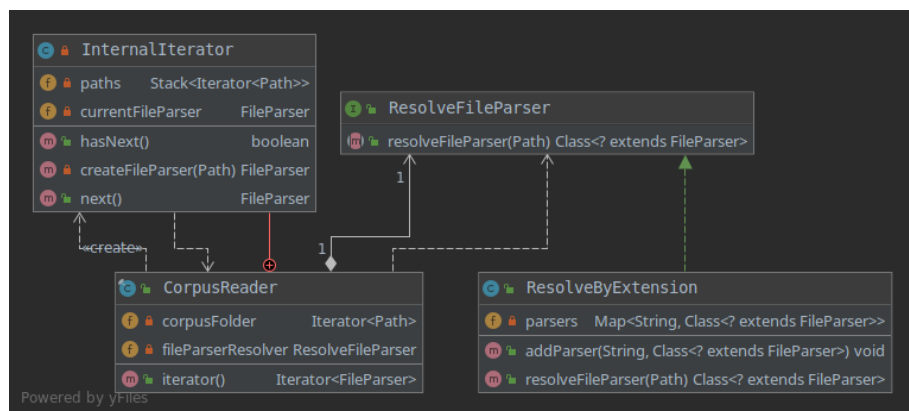


Figure 2: Diagrama de classes do package *parsers.corpus*

responsável por fazer a associação entre o ficheiro e FileParser adequado. Para este assignment foi desenvolvido uma classe que escolhe o FileParser segundo a extensão do ficheiro.

3.2.2 parsers.files

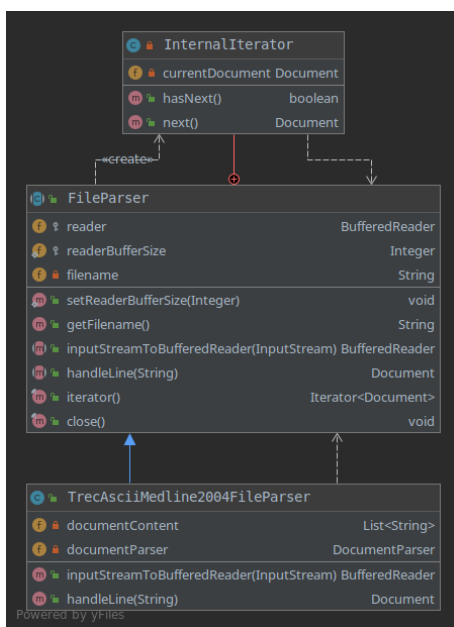


Figure 3: Diagrama de classes do package *parsers.files*

Mais uma vez, para obtermos os documentos de cada ficheiro seguimos uma aproximação de streams, em que a classe FileParser implementa a interface Iterable. O método *hasNext* do iterador da classe FileParser lê linha a linha de um BufferedReader até que o método *handleLine* retorne uma referência para um objeto do tipo Document (do package parsers.documents) não nula, a qual será retornada na próxima chamada do método *next*.

Para cada formato de ficheiro diferente deverá ser criada uma classe descendente da classe `FileParser`, implementando o método *handleLine* que retorna objetos `Document` quando as linhas lidas até ao momento completam um documento, e o método *inputStreamToBufferedReader*, que transforma uma `InputStream` num `BufferedReader` permitindo inserir os necessários wrappers. Este último método permite abrir todos os ficheiros da mesma maneira e deixando para os `FileParsers` a responsabilidade de criar os necessários wrappers *exemplo* `InputStream > GZIPInputStream > InputStreamReader > BufferedReader`.

3.2.3 parsers.documents

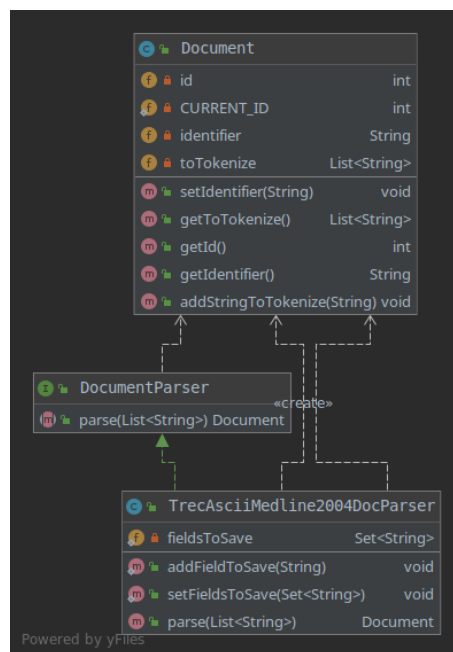


Figure 4: Diagrama de classes do package *parsers.documents*

O documento devolvido no método *handleLine* resulta do parsing do conteúdo do documento por um `DocumentParser`. O objetivo desta classe é retirar do conteúdo do documento a informação necessária para a indexação e criar um objeto `Document` associado. Este objeto tem um `id` que é incrementado a cada `Document` criado, não havendo `ids` repetidos, um `identifier`, que é utilizado para fazer a associação do `id` para o documento, e apresenta uma lista do conteúdo a ser tokenizado e posteriormente indexado.

Para cada formato diferente de documentos deverá ser criada uma classe descendente da classe `DocumentParser`, implementando o método *parse* que percorre o conteúdo do documento lido e retira a informação a indexar. Isto é útil para casos por exemplo em que tenhamos um ficheiro comprimido (`.gz`) e outro em plain text, em que em ambos os documentos estão no mesmo formato, onde podemos usar o mesmo document parser para os dois casos.

3.3 tokenizer

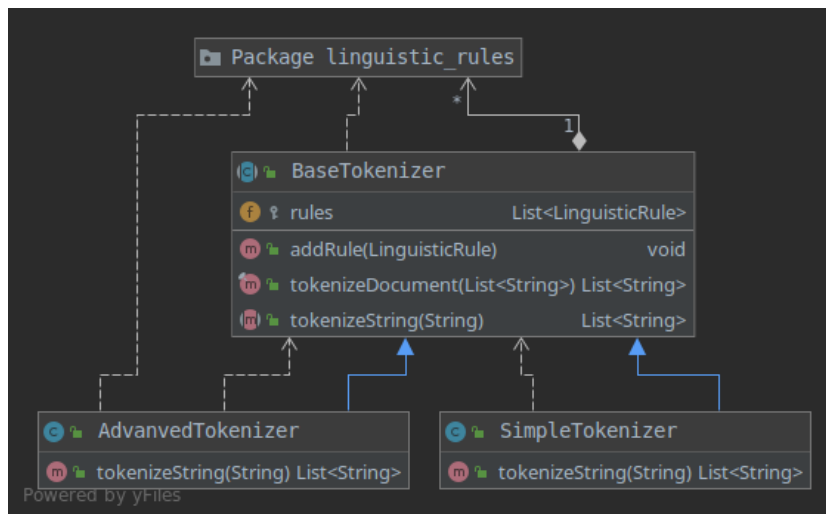


Figure 5: Diagrama de classes do package *tokenizer*

Aqui encontram-se as classes que transformam partes dos documentos em termos (Tokenizers). A classe principal, *BaseTokenizer*, é a classe base das diferentes implementações de tokenizers. As classes descendentes desta devem implementar o método *tokenizeString* ou aplicar regras ao conteúdo recebido um conjunto de regras, devolvendo uma lista de termos. Para os tokenizers não existe a noção de documento, simplesmente aplicam regras a conteúdo recebido.

3.3.1 tokenizer.linguistic_rules

TODO

3.4 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.

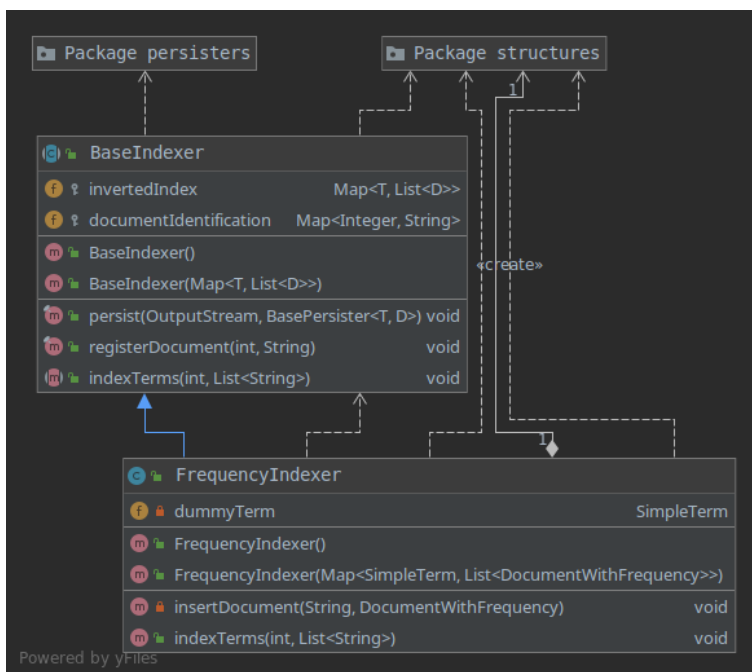


Figure 6: Diagrama de classes do package *indexer*

3.4.1 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 isto 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.

3.4.2 indexer.persisters

4 Data Flow

5 Resultados

6 Anexos

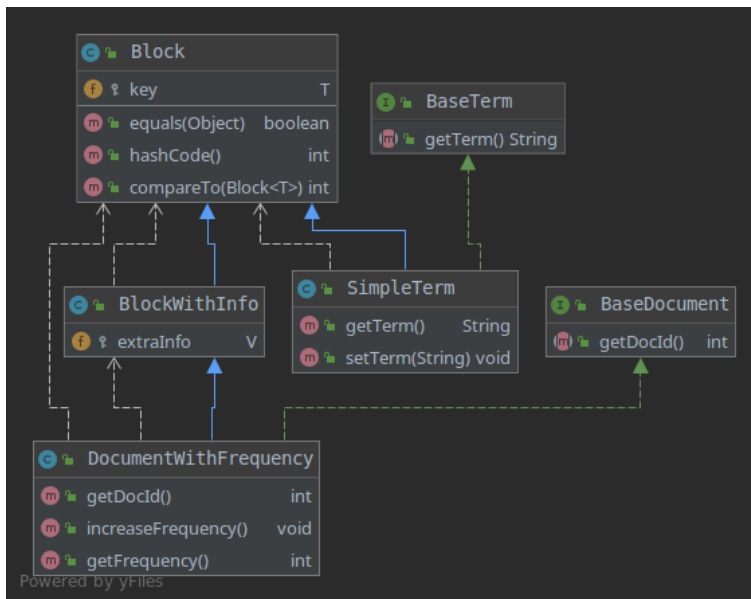


Figure 7: Diagrama de classes do package *indexer.structures*

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 of 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.

Figure 8: Diagrama de classes do package *indexer.structures*