

Assignmnet 1

André Pedrosa [85098], João Abílio [84732]

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 aparece-se.

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 de 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

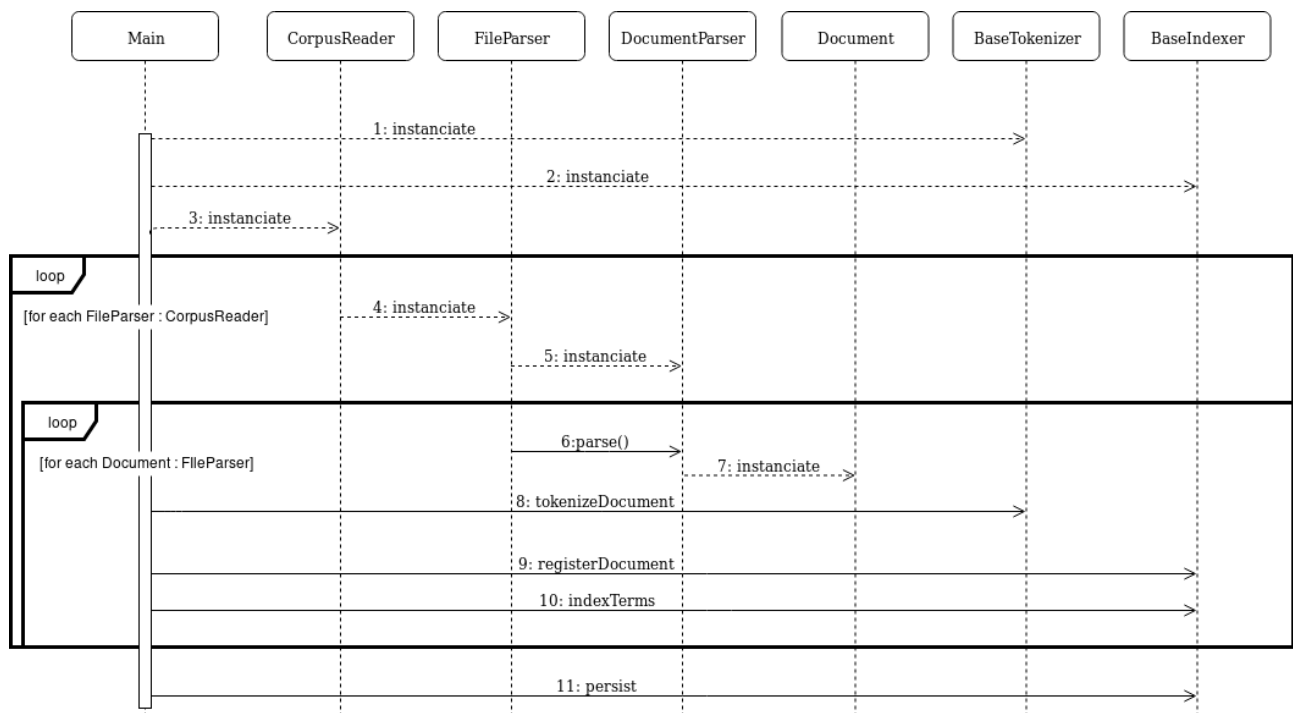


Figure 1: Diagrama de sequência da solução

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

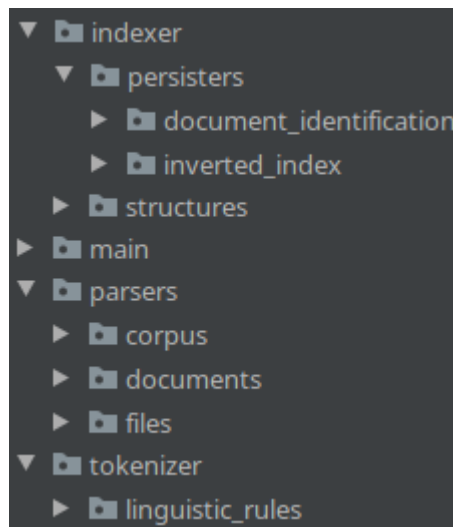


Figure 2: Árvore de packages da solução

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

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.

4.2.1 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 iterativa, como foi mencionado anteriormente. Quando é

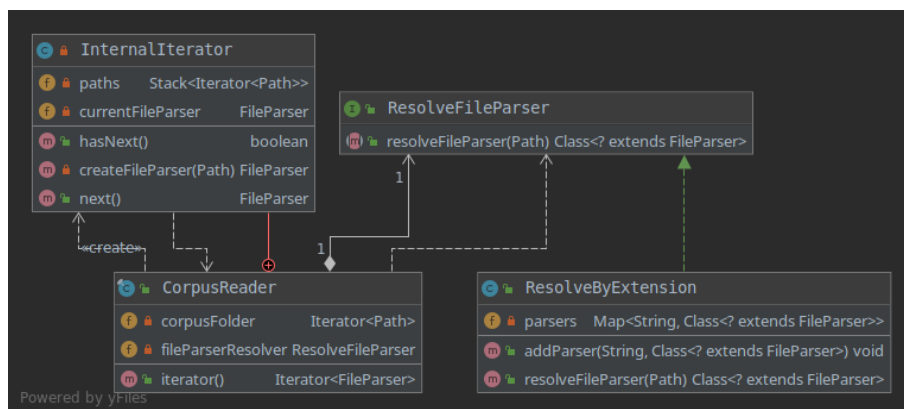


Figure 3: Diagrama de classes do package *parsers.corpus*

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 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.

4.2.2 parsers.files

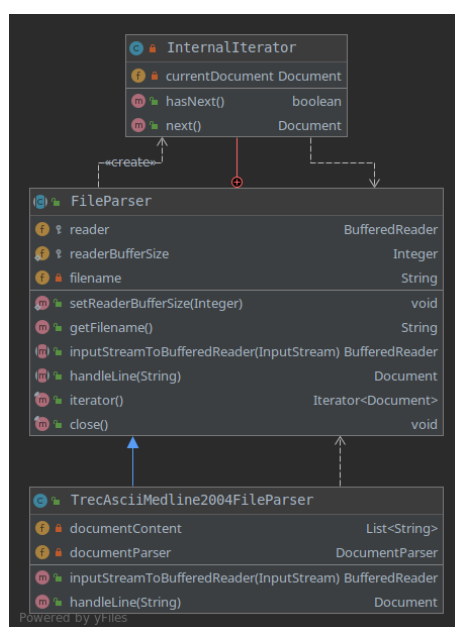


Figure 4: Diagrama de classes do package *parsers.files*

Mais uma vez, para obtermos os documentos de cada ficheiro seguimos uma aproximação iterativa, 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`.

4.2.3 parsers.documents

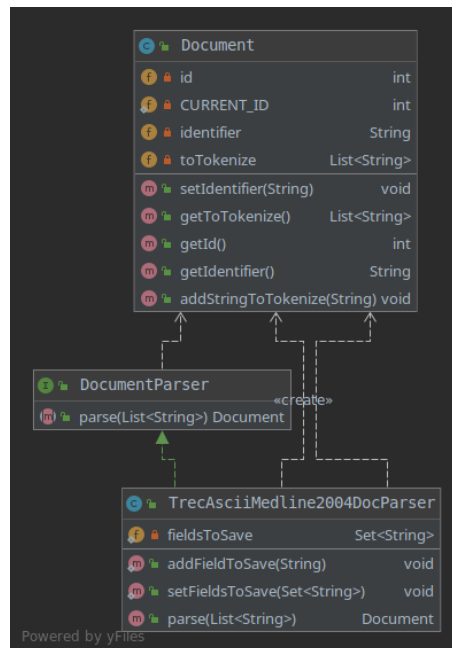


Figure 5: Diagrama de classes do package `parsers.documents`

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 ao 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 tokenizar. 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, permitindo-nos usar o mesmo document parser para os dois casos.

4.3 tokenizer

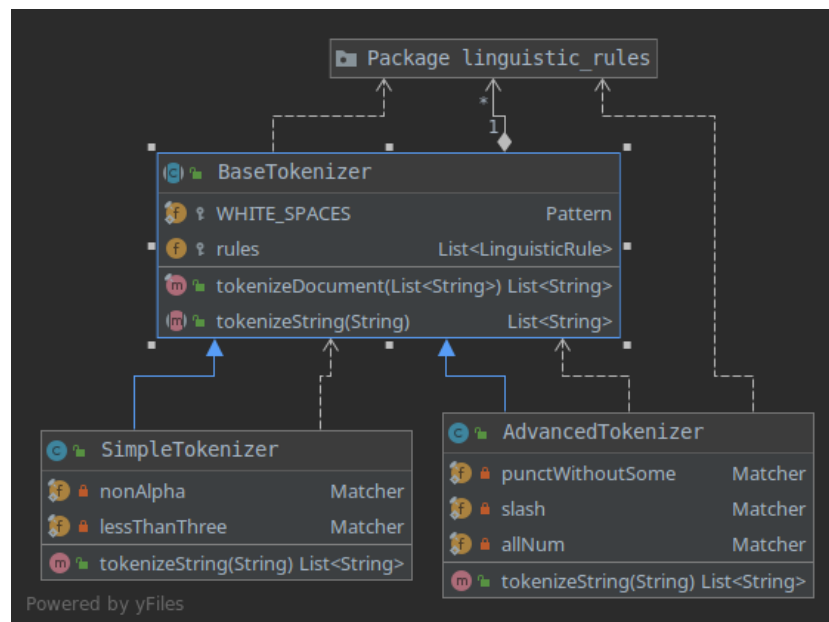


Figure 6: Diagrama de classes do package *tokenizer*

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* on aplicam regras ao conteúdo recebido, devolvendo uma lista de termos. Para os tokenizers não existe a noção de documento, simplesmente aplicam regras a conteúdo recebido.

No nosso tokenizer avançado mantemos as palavras com hífen, eliminamos termos com apenas dígitos e com menos de 3 caracteres e aplicamos ainda stemming e uma filtragem de stop words.

4.3.1 tokenizer.linguistic_rules

De maneira a poder aplicar as mesmas regras em diferentes tokenizers foi criado uma interface comum que aplica regras linguísticas a termos. Estas regras linguísticas podem ser, por exem-

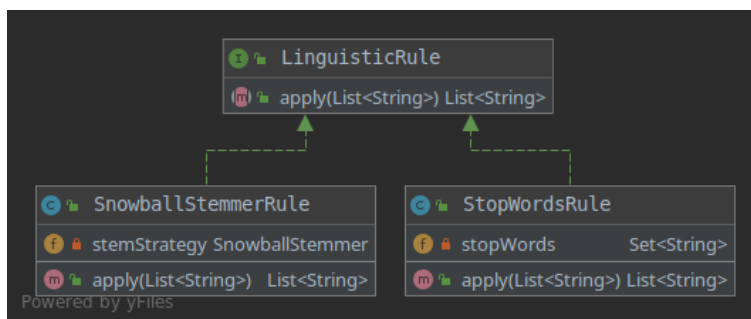


Figure 7: Diagrama de classes do package *tokenizer.linguistic_rules*

plo, fazer a exclusão se certos termos que cumprem um conjunto de regras (Stop Words) ou a transformação dos termos (Stemmer).

4.4 indexer

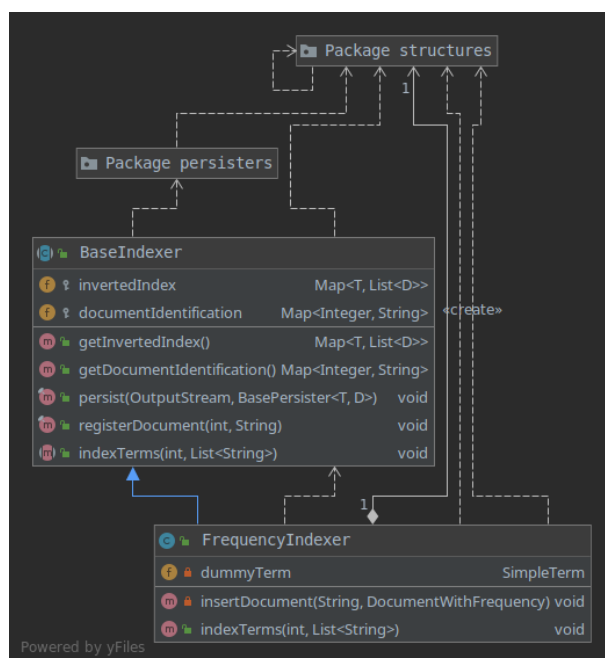


Figure 8: Diagrama de classes do package *indexer*

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

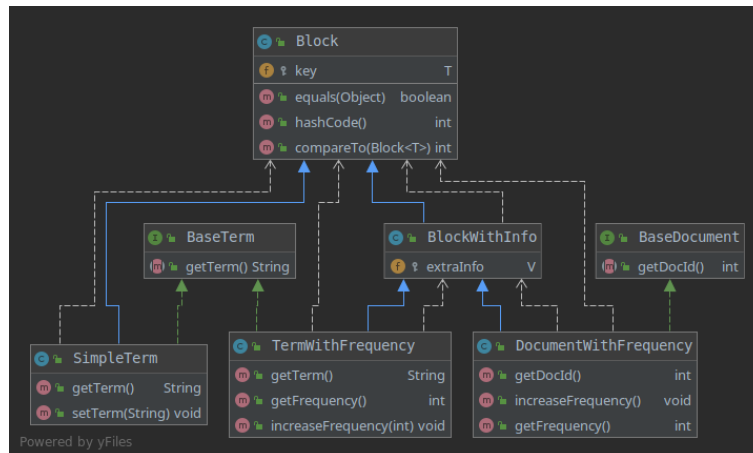


Figure 9: Diagrama de classes do package *indexer.structures*

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.

4.4.2 indexer.persisters

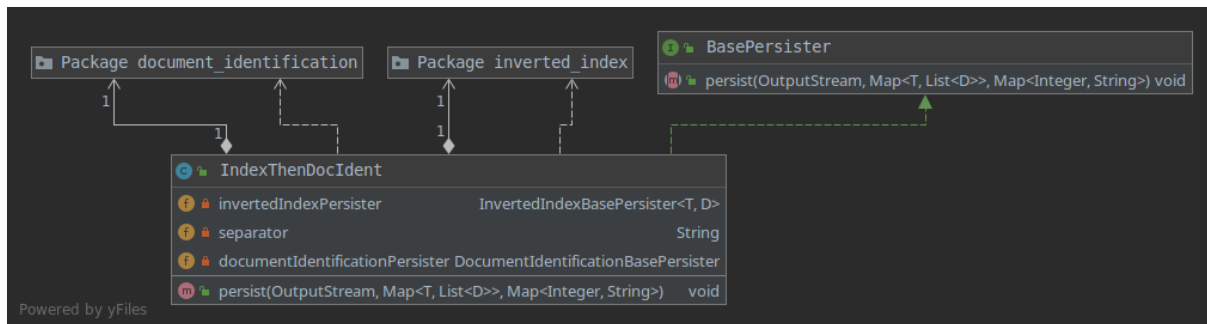


Figure 10: Diagrama de classes do package *indexer.persisters*

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.

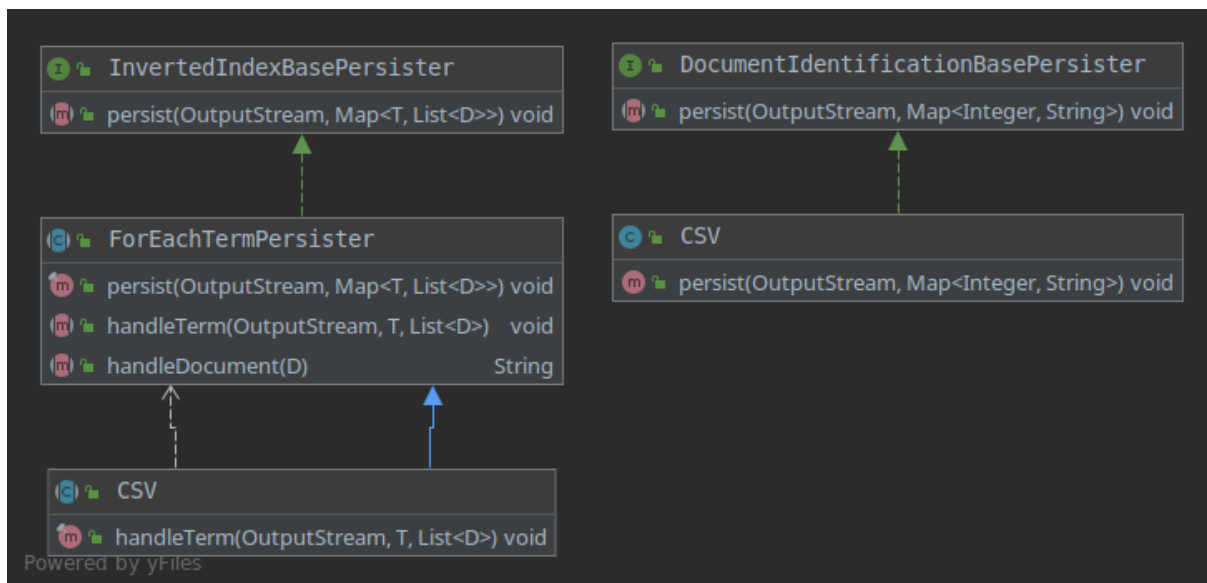






Figure 11: Diagrama de classes do package *indexer.persisters.inverted_index* à esquerda e do package *indexer.persisters.document_identification* à direita















5 Resultados

Resultados usando apenas o ficheiro 2004_TREC_ASCII_MEDLINE_1.gz

	SimpleTokenizer	AdvancedTokenizer
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 aaaction 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

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 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.

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