

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 apresentadas as medidas de eficiência pedidas no enunciado do assignment para cada novo método de indexação.

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 Data Flow

O data flow da nossa solução, de modo geral não sofreu grandes alterações, apenas migramos o código de execução da pipeline de indexação para uma classe separada, em vez de ser definida na classe Main.

Na figura 1 está representado a sequência de execução da nossa solução, onde as setas azuis significam que a classe origem executa um método da classe destino e as setas vermelhas significam que a classe origem cria a classe destino. Deixando de parte as classes que têm uma seta vermelha a apontar para si, todas as classes são instanciadas na classe Main. Na figura existem

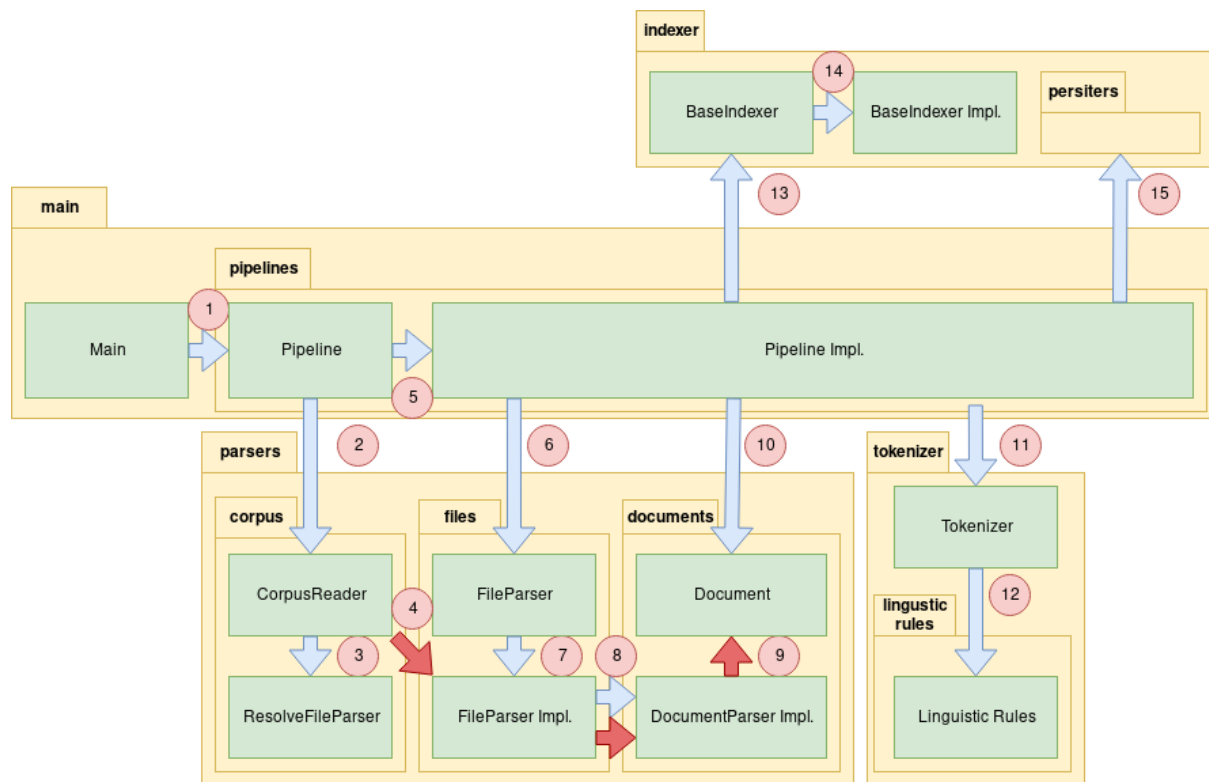


Figure 1: Diagrama de sequência da solução

várias classes em que é apresentado a classe base e a sua implementação, isto pois algumas classes base apresentam métodos *final* que chamam depois métodos abstratos que devem estar definidos em classes de descendentes (padrão Template Method).

3 Packages

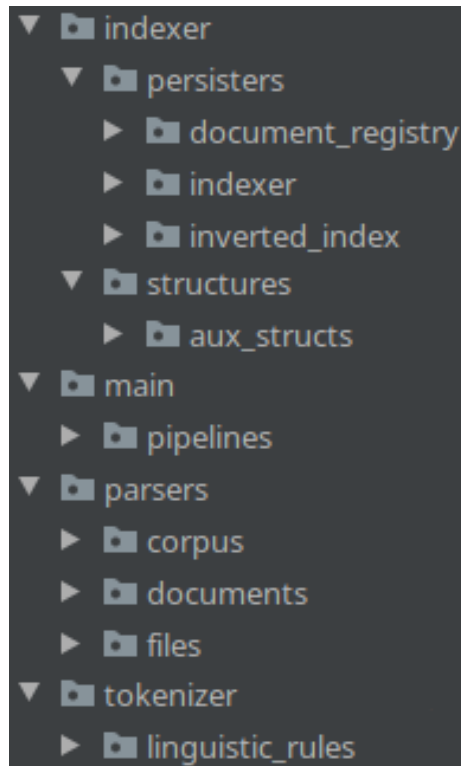


Figure 2: Árvore de packages da solução

Nesta secção vão ser apresentadas as alterações feitas a cada package relativamente à entrega anterior.

3.1 main

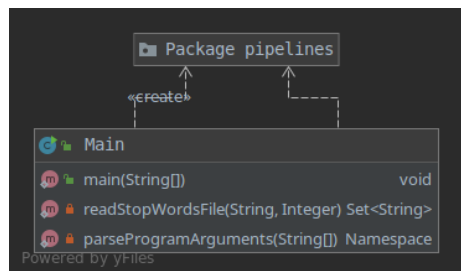


Figure 3: Diagrama de classes do package *main.pipelines*

Como foi dito na secção anterior, o código de execução da pipeline de indexação que estava presente na classe *Main*, foi migrado para para uma classe do tipo *Pipeline*. Ou seja, a classe

Main apenas tem a responsabilidade de instanciar as classes necessárias para a execução da pipeline.

3.1.1 pipelines

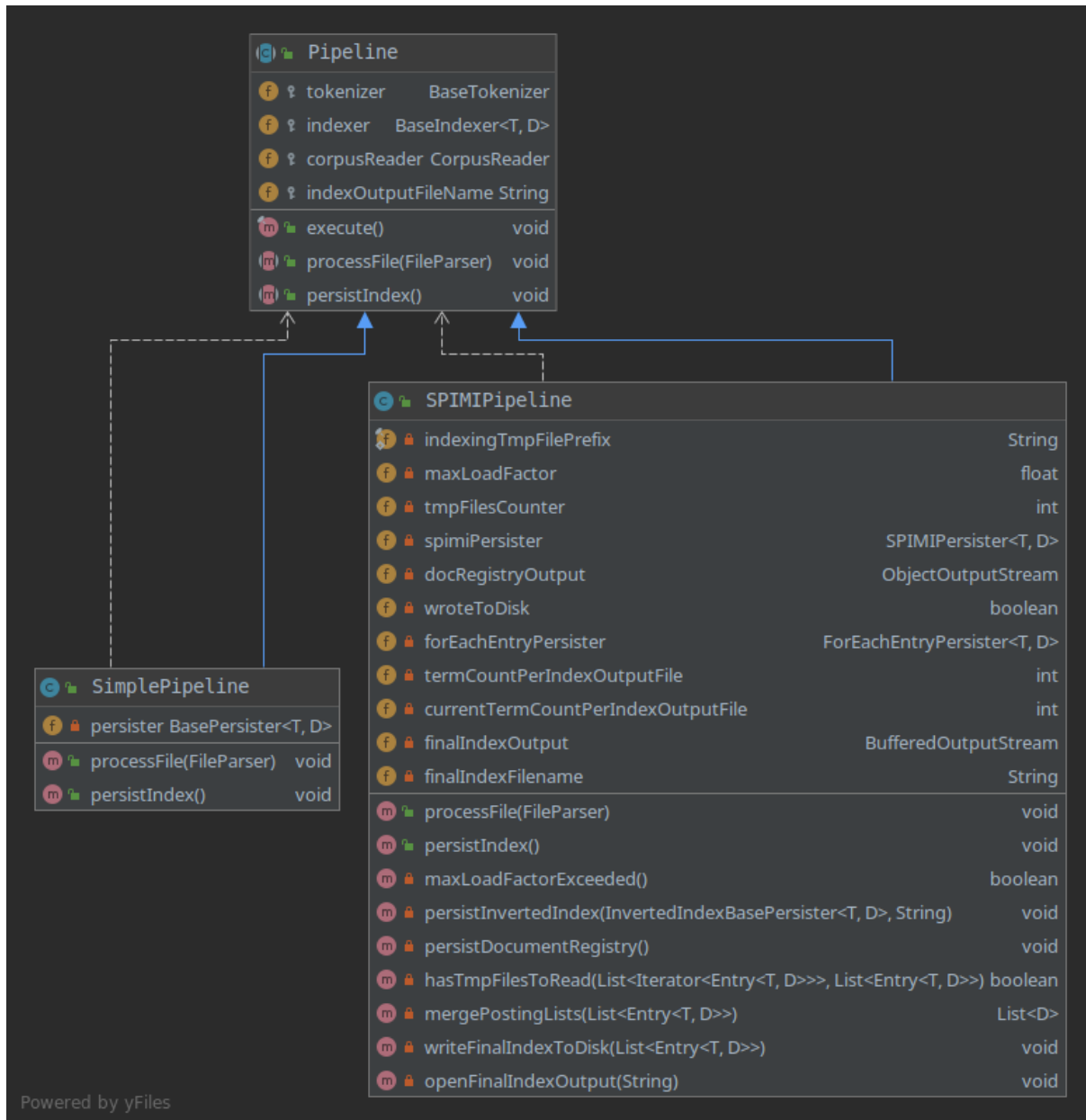


Figure 4: Diagrama de classes do package *main.pipelines*

Este package tem as classes que definem a sequência de execução do método de indexação. A classe principal (Pipeline) está encarregue de iterar sobre o corpus e passar cada FileParser à implementação de Pipeline escolhida (método processFile) e depois disso executar o método

para persistir o index.

A classe SimplePipeline representa a pipeline que foi definida no assignment anterior, em que não tem considerações relativamente à memória no processo de indexação. A SPIMIPipeline apresenta uma implementação em que o método de indexação é feito segundo o algoritmo SPIMI, em que durante a indexação, sempre que a memória ocupada chegar a um limite, o index atual é escrito para disco, ordenado pelos termos. Na altura de persistir o index, esta pipeline faz um merge dos vários ficheiros criados na fase anterior. Este merge é feito lendo blocos (implementado com a classe BufferedInputStream) de todos os ficheiros temporários criados. Para fazer merge destes blocos, é mantida uma lista à qual é adicionada iterativamente a entry (termo e posting list) em que o termo é menor (alfabeticamente). Caso haja o mesmo termo em diferentes blocos é feito o merge das posting lists segundo os document ids.

Os ficheiros temporários foram escritos em binário (usando a class ObjectOutputStream), já que se revelou ser muito mais rápido do que aplicar a forma de persistência que o utilizador escolhe no Main, no caso deste assignment CSV. Para guardar o index final foram criados vários ficheiros, tendo cada um um número máximo de entries e para saber quais os termos presentes em cada ficheiros deste foi inserido como sufixo ao nome dos ficheiros o primeiro termo que guardam.

3.2 parsers

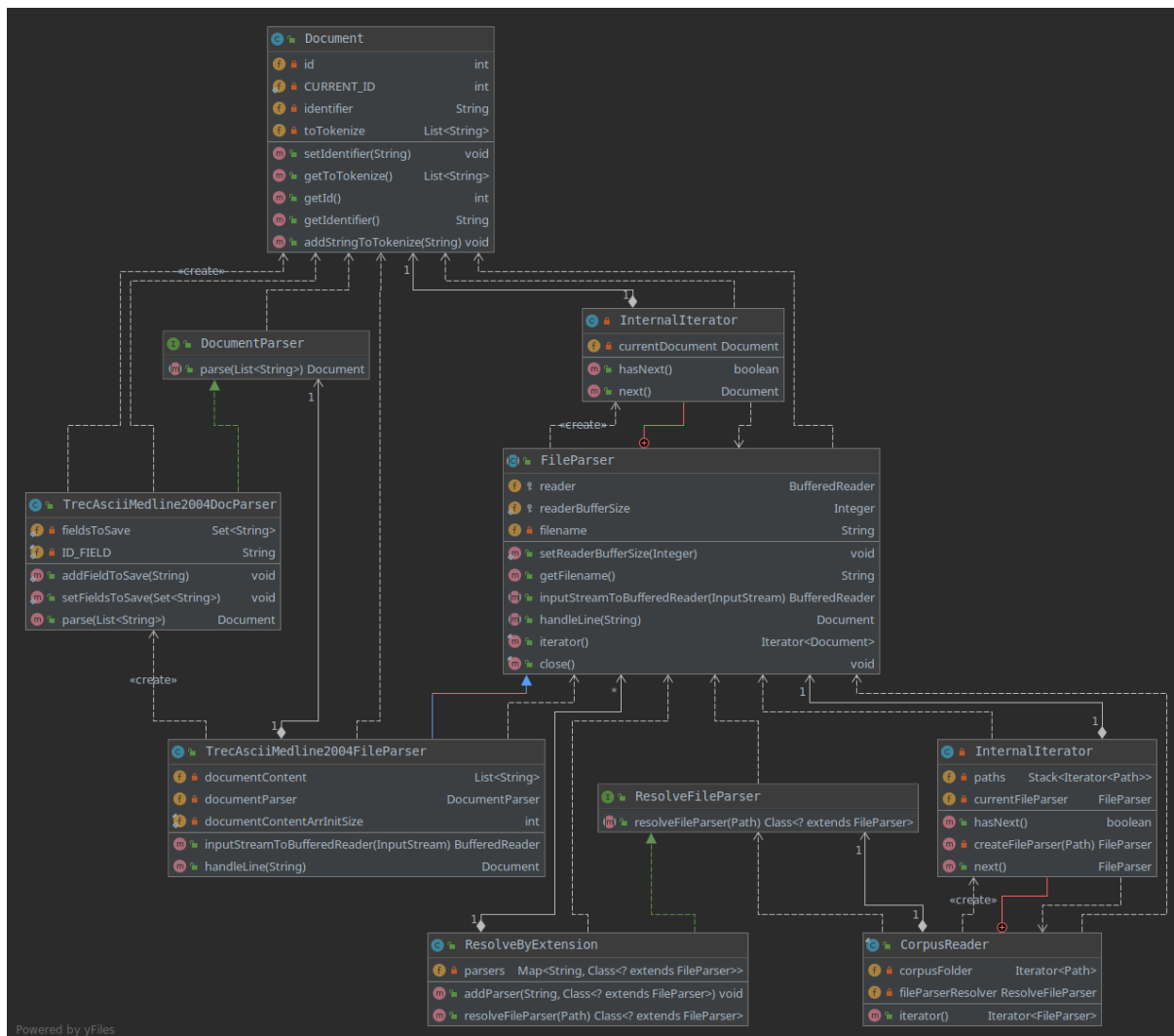


Figure 5: Diagrama de classes do package *parsers*

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, que não alteram o resultado final, obtidas dos ficheiros do corpus foram removidas, como `.trim()`, para tentar reduzir o número de instanciações da classe `String`.

3.3 tokenizer

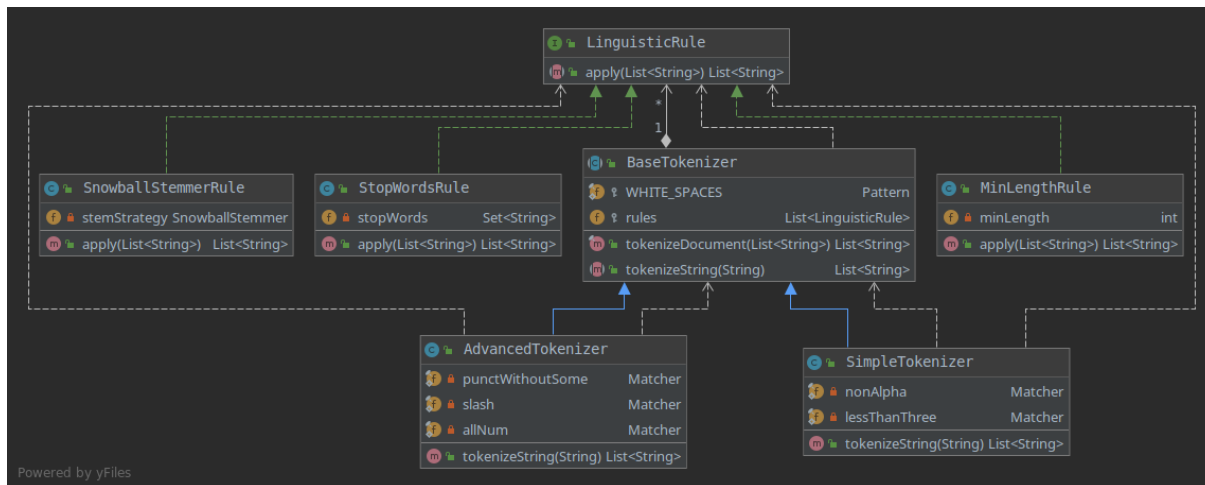


Figure 6: Diagrama de classes do package *tokenizer*

Este package também não sofreu grandes alterações relativamente à entrega anterior, apenas foi criada uma nova Linguística Rule que remove os termos que têm menos de um certo número de caracteres.

3.4 indexer

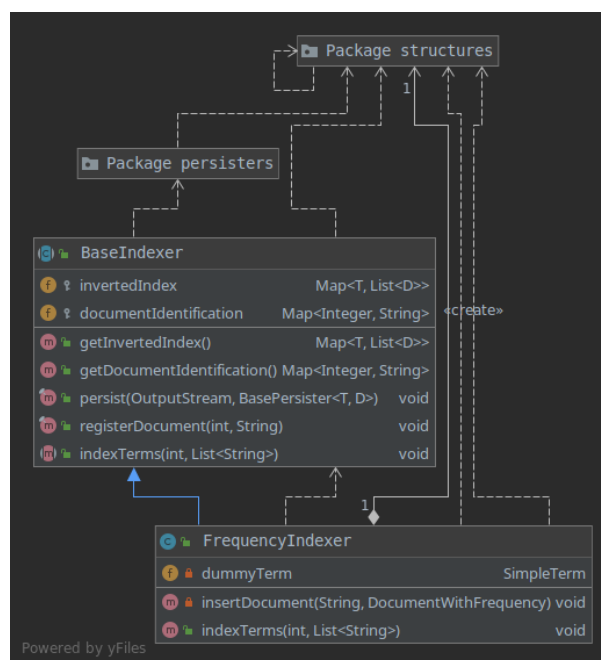


Figure 7: Diagrama de classes do package *indexer*

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.

3.4.1 indexer.structures

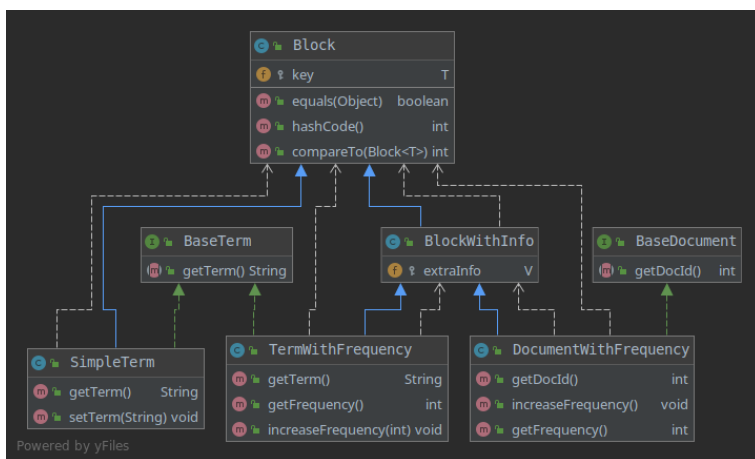


Figure 8: Diagrama de classes do package *indexer.structures*

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`.

3.4.2 indexer.persisters

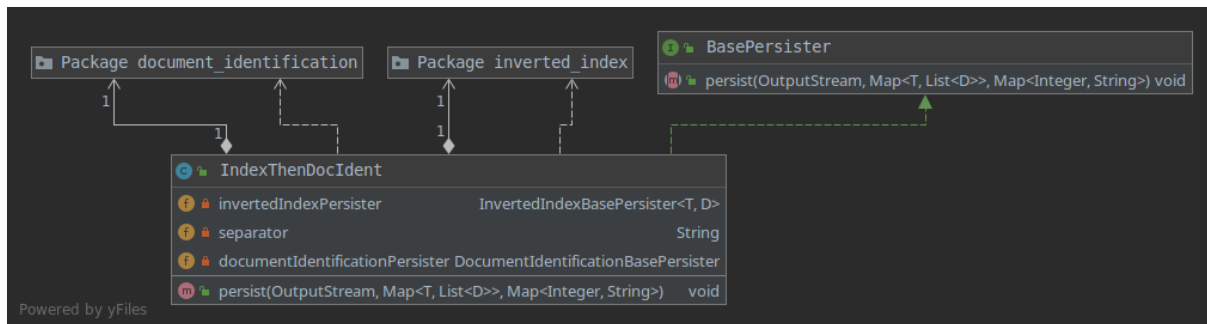


Figure 9: Diagrama de classes do package *indexer.persisters*

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.

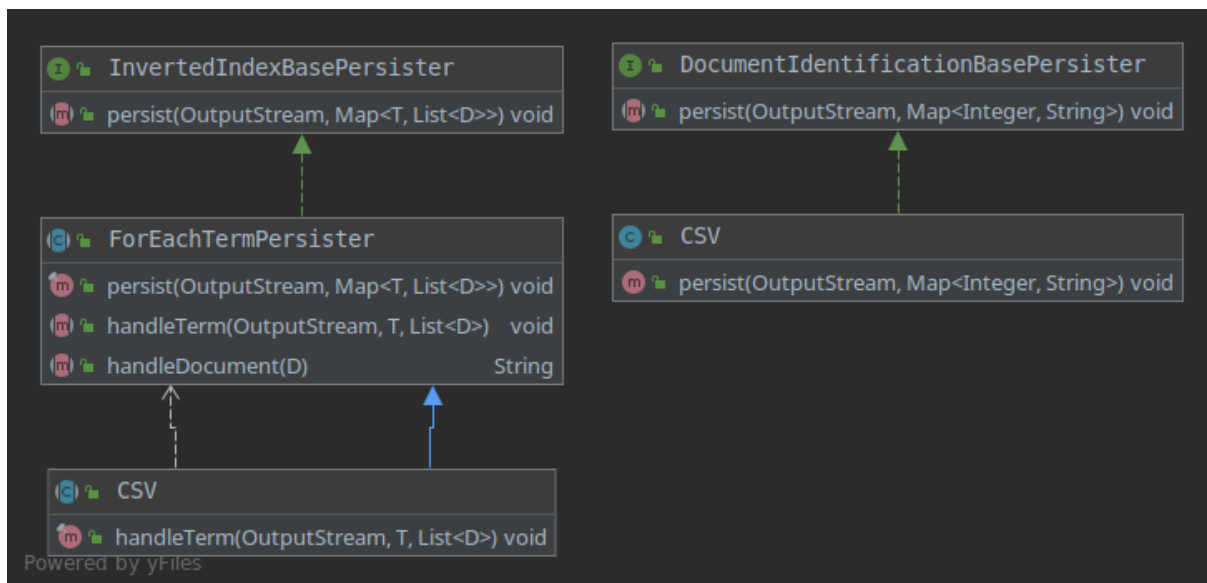






Figure 10: Diagrama de classes do package *indexer.persisters.inverted_index* à esquerda e do package *indexer.persisters.document_identification* à direita















4 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

5 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