Práctica 2: Preprocesado de documentos, parte 2: análisis del texto

Juan Manuel Consigliere Picco y Óscar Pérez Tobarra

Ejercicio 1 (Juan Manuel)

Este ejercicio nos pedía hacer un estudio estadístico de los tokens que nos dan cada una de las salidas de los analizadores que escojamos. Empiezo explicando el código:

```
public static void Analizador(Analyzer analyzer, String nombre_analizador, String texto, String nombre_archiv
    TokenStream stream = analyzer.tokenStream(null, texto);
    Map<String, Integer> ocurrencias = new HashMap<String, Integer>();

    stream.reset(); //Se le llama antes de usar incrementToken()
    while(stream.incrementToken()){ //Itera de token en token
        String palabra = stream.getAttribute(CharTermAttribute.class).toString(); //Pasamos el apartado de te
        if(!ocurrencias.containsKey(palabra))
                ocurrencias.put(palabra, 1);
        else{
            int valor = ocurrencias.get(palabra);
            ocurrencias.replace(palabra, valor, valor+1);
        }
    }

    stream.end(); //Se le llama cuando se termina de iterar
    stream.close(); //Liberas los recursos asociados al stream

    List<Map.Entry<String, Integer>> words = ocurrencias.entrySet().stream().collect(Collectors.toList());
    Collections.sort(words, new Comparator<Map.Entry<String, Integer>>(){
        public int compare(Map.Entry<String, Integer> o1, Map.Entry<String, Integer> o2){
            return o2.getValue().compareTo(o1.getValue());
        }
    });

    PrintWriter writer = new PrintWriter(new File("./csv/" + nombre_archivo.substring(0, nombre_archivo.lastI
    writer.write("Id;Text;Size\n");

    int c = 0;

    for(Map.Entry<String, Integer> i : words){
      c++;
      writer.write(c+";"+i.getKey()+";"+i.getValue()+"\n");
    }

    writer.close();

    System.out.println("Finalizado procesamiento del analizador " + nombre_analizador + "analyzer");
}
```

Creamos una función Analizador , que al pasarle un objeto tipo Analyzer nos da la salida ordenada, y nos la escribe en un archivo .csv. La implementación es muy parecida a la de la práctica 1, al ordenar las frecuencias y crear el .csv. La última actualización que he hecho añade el total de tokens del archivo.

```
System.out.println("Número de tokens: "+c+"\n");
```

```java
public static void main(String[] args) throws IOException{
    Tika tika = new Tika();
    Metadata metadata = new Metadata();
    File archivo = new File("../test/hamlet.txt");

    String text = new String();

    try{
        text = tika.parseToString(archivo);
    }catch (Exception e){
        System.out.println("No se puede parsear...\n\n");
    }

    tika.parse(archivo,metadata); //Parseamos el fichero de texto plano
    String nombre_archivo = archivo.getName();
```

En el main leemos el archivo hamlet.txt de la carpeta /test, lo parseamos a String usando Tike y lo pasamos a la función Analizador, junto a 4 analizadores distintos.

- WhitespaceAnalyzer: Divide el texto por espacios solamente.
- SimpleAnalyzer: Pone las palabras en minúsculas y luego divide el texto por caracteres especiales (los que no son letras).
- StopAnalyzer: Pone las palabras en minúsculas, quita los “stop words” (como las palabras vacías), y divide el texto por caracteres especiales. Le hemos especificado un set de palabras vacías contenido en la biblioteca StandardAnalyzer.
- StandardAnalyzer: Pone las palabras en minúsculas, quita los “stop words”, y usa las reglas de rotura de palabras del algoritmo de segmentación de texto Unicode.

```java
Analizador(new WhitespaceAnalyzer(), "whitespace", text, nombre_archivo);
Analizador(new SimpleAnalyzer(), "simple", text, nombre_archivo);
Analizador(new StopAnalyzer(StandardAnalyzer.ENGLISH_STOP_WORDS_SET), "stop", text, nombre_archivo);
Analizador(new StandardAnalyzer(), "standard", text, nombre_archivo);
```

La salida es la siguiente:

```
jcpicco@DumbleDogg:/mnt/c/Users/jmcon/Documents/GitHub/ri/p2/ej1$ j
../lucene-7.1.0/analysis/common/lucene-analyzers-common-7.1.0.jar:.
Finalizado procesamiento del analizador whitespaceanalyzer
Número de tokens: 4982

Finalizado procesamiento del analizador simpleanalyzer
Número de tokens: 3089

Finalizado procesamiento del analizador stopanalyzer
Número de tokens: 3056

Finalizado procesamiento del analizador standardanalyzer
Número de tokens: 3164
```

El número de tokens de WhitespaceAnalyzer hace que no haga falta explicar por qué no deberíamos ni de tenerlo en cuenta. Es demasiado alto, y está lleno de palabras vacías. Este es case sensitive y por eso hay muchos tokens: porque se repiten de varias maneras.

SimpleAnalyzer vemos que ha disminuído en casi 2000 tokens el conteo total. Esto se debe a que se han pasado todos los tokens a minúsculas. El problema de este es que se observa que las URLs no las controla correctamente, por lo que hay mala información, además de que hay palabras vacías.

StopAnalyzer hace lo mismo que SimpleAnalyzer, pero quitando las palabras vacías especificadas. Vemos que ha disminuído poco, pero lo hace correctamente. Se pueden especificar palabras vacías propias. Vemos una comparación de los dos .csv:

Id	Text	Size	Id	Text	Size
1	the	541	1	i	292
2	and	510	2	you	289
3	to	418	3	my	269
4	of	410	4	ham	143
5	i	292	5	d	134
6	you	289	6	his	134
7	my	269	7	lord	125
8	a	257	8	your	112
9	in	224	9	so	112
10	it	215	10	he	103
11	that	196	11	me	102
12	is	170	12	what	96
13	this	158	13	s	93
14	not	152	14	have	93
15	for	144	15	we	87
16	ham	143	16	him	74
17	with	139	17	pol	71
18	d	134	18	do	71

A la derecha tenemos Simple, y a la izquierda Stop. Vemos cómo se han suprimido palabras como the, and, to, a, in, etc, que no aportan nada en absoluto. No parece un gran cambio pero esto facilita las cosas bastante.

StandardAnalyzer es simplemente la rotura en tokens con un algoritmo distinto. Encuentra más que SimpleAnalyzer, pero no es posible estimar si esto es bueno o malo.

Como conclusión final yo escogería, de los por defecto, el StopAnalyzer, ya que limpia mucho la salida de palabras que no nos sirven, pero realmente lo mejor que podemos hacer es crear un Analyzer o filtros que se ajusten a nuestras necesidades, como veremos en el ejercicio 3 y 4.

Ejercicio 2 (Óscar)

En este ejercicio se pide comprobar el resultado que se obtiene al usar diferentes tokenFilters. Para ello, hemos hecho el siguiente programa:

```
public class practica2_2{
    public static void Filtro(TokenStream stream, String nombre_filtro) throws IOException{
        stream.reset(); //Se le llama antes de usar incrementToken()

        System.out.println("Filtro: " + nombre_filtro);
        while(stream.incrementToken()){ //Itera de token en token
            System.out.print(" || "+stream.getAttribute(CharTermAttribute.class).toString()); //Pasamos el apartado de texto del token a String
        }
        System.out.print("\n\n");

        stream.end(); //Se le llama cuando se termina de iterar
        stream.close(); //Liberas los recursos asociados al stream
    }

    public static void main(String[] args) throws IOException{
        SynonymMap.Builder builder = new SynonymMap.Builder(true);
        builder.add(new CharsRef("have"), new CharsRef("deprived"), true);
        builder.add(new CharsRef("for"), new CharsRef("his"), true);
        SynonymMap sinonimos = builder.build();

        Tika tika = new Tika();
        Metadata metadata = new Metadata();
        File archivo = new File("./test.txt");

        String text = new String();

        try{
            text = tika.parseToString(archivo);
        }catch (Exception e){
            System.out.println("No se puede parsear...\n\n");
        }

        CharArraySet palabras = new CharArraySet(0,false);
        palabras.add("very");
        palabras.add("poor");
        palabras.add("away");

        tika.parse(archivo,metadata); //Parseamos el fichero de texto plano
        String nombre_archivo = archivo.getName();
```

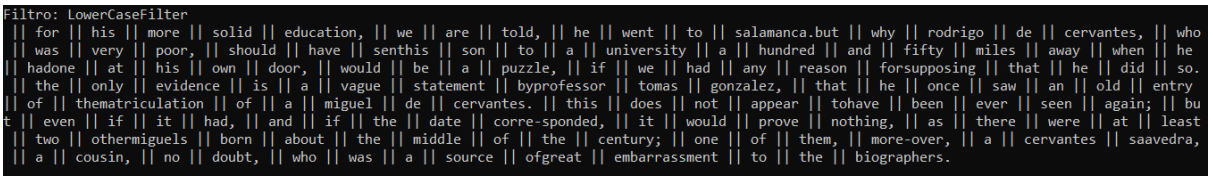
```
        Filtro(new StandardFilter(new WhitespaceAnalyzer().tokenStream(null, text)), "StandardFilter");
        Filtro(new LowerCaseFilter(new WhitespaceAnalyzer().tokenStream(null, text)), "LowerCaseFilter");
        Filtro(new StopFilter(new WhitespaceAnalyzer().tokenStream(null, text), StandardAnalyzer.ENGLISH_STOP_WORDS_SET), "StopFilter");
        Filtro(new SnowballFilter(new WhitespaceAnalyzer().tokenStream(null, text), "English"), "SnowballFilter");
        Filtro(new ShingleFilter(new WhitespaceAnalyzer().tokenStream(null, text)), "ShingleFilter");
        Filtro(new EdgeNGramTokenFilter(new WhitespaceAnalyzer().tokenStream(null, text),2,4), "EdgeNGramTokenFilter");
        Filtro(new NGramTokenFilter(new WhitespaceAnalyzer().tokenStream(null, text),2,4), "NGramTokenFilter");
        Filtro(new CommonGramsFilter(new WhitespaceAnalyzer().tokenStream(null, text),palabras), "CommonGramsFilter");
        Filtro(new SynonymFilter(new WhitespaceAnalyzer().tokenStream(null, text), sinonimos, true), "SynonymFilter");
    }
}
```

Basándonos en la práctica anterior, hemos usado Tika para parsear los documentos y poder acceder a la información. Para poder modularizar la ejecución de los diferentes filtros, hemos creado un método Filtro que permite pasarle un TokenStream del filtro correspondiente, el texto parseado y el nombre del filtro. Este método ejecuta un analizador del tipo WhitespaceAnalyzer con el filtro que se le haya pasado por párametro. El resultado de la ejecución de cada filtro se muestra a continuación:

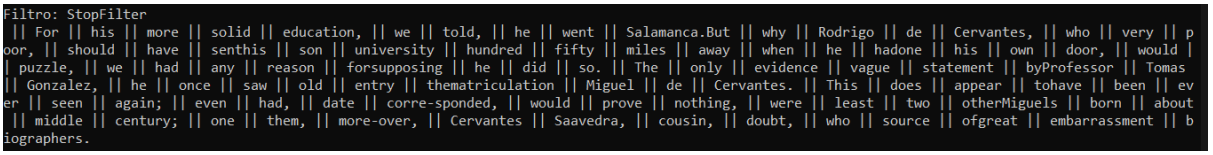
- StandardFilter: devuelve los tokens tal y como lo ha devuelto el analizador.

```
Filtro: StandardFilter
 || For || his || more || solid || education, || we || are || told, || he || went || to || Salamanca.But || why || Rodrigo || de || Cervantes, || who
 || was || very || poor, || should || have || senthis || son || to || a || university || a || hundred || and || fifty || miles || away || when || he
|| hadone || at || his || own || door, || would || be || a || puzzle, || if || we || had || any || reason || forsupposing || that || he || did || so.
 || The || only || evidence || is || a || vague || statement || byProfessor || Tomas || Gonzalez, || that || he || once || saw || an || old || entry
|| of || thematriculation || of || a || Miguel || de || Cervantes. || This || does || not || appear || tohave || been || ever || seen || again; || bu
t || even || if || it || had, || and || if || the || date || corre-sponded, || it || would || prove || nothing, || as || there || were || at || least
 || two || otherMiguels || born || about || the || middle || of || the || century; || one || of || them, || more-over, || a || Cervantes || Saavedra,
 || a || cousin, || no || doubt, || who || was || a || source || ofgreat || embarrassment || to || the || biographers.
```

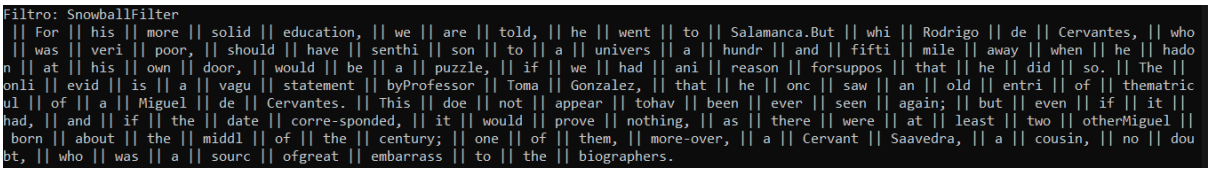
- LowerCaseFilter: devuelve los tokens que devuelve el analizador sin mayúsculas.



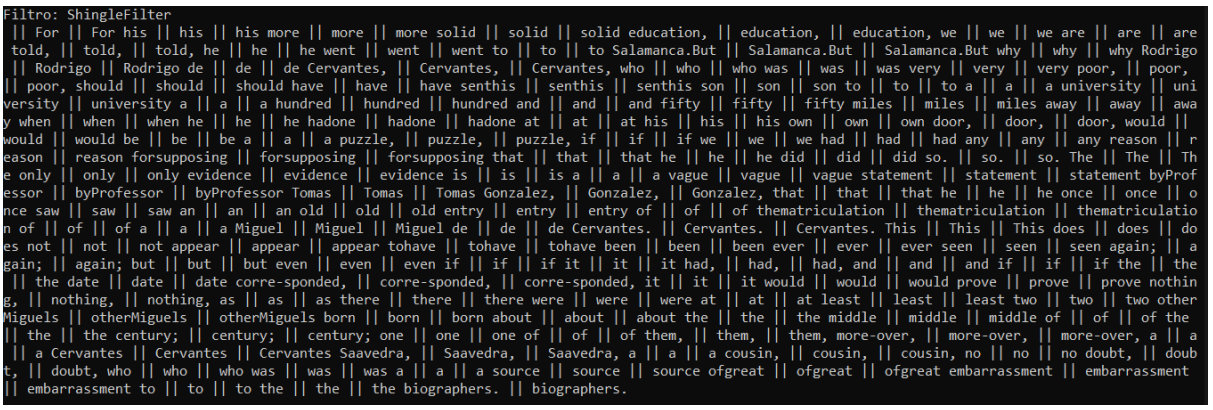
- StopFilter (se ha usado el conjunto de stopwords inglés): devuelve los tokens que devuelve el analizador separando y eliminando las stopwords que se le haya especificado al filtro.



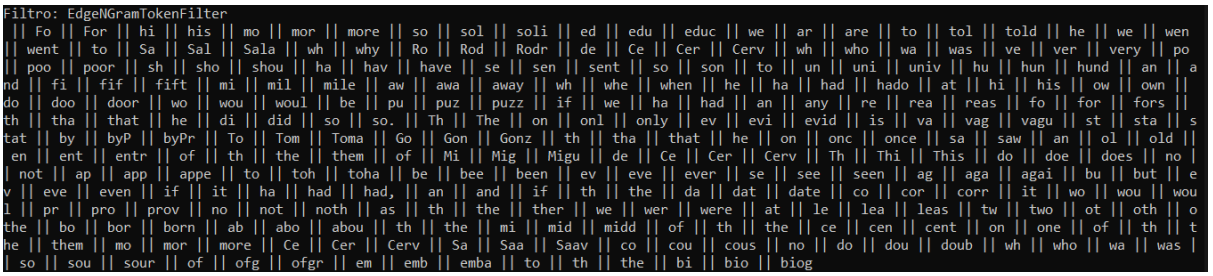
- SnowballFilter (se ha usado el idioma inglés): devuelve las raíces de los tokens acorde con el idioma que se le haya especificado al filtro.



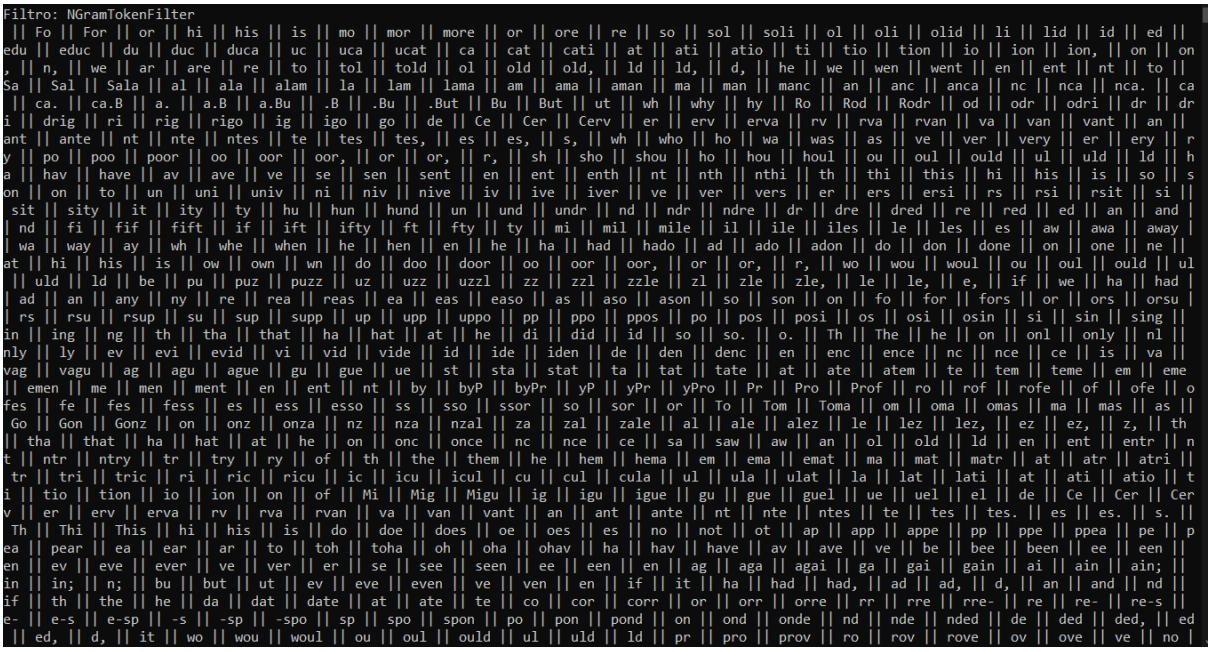
- ShingleFilter: crea combinaciones de 2 tokens.

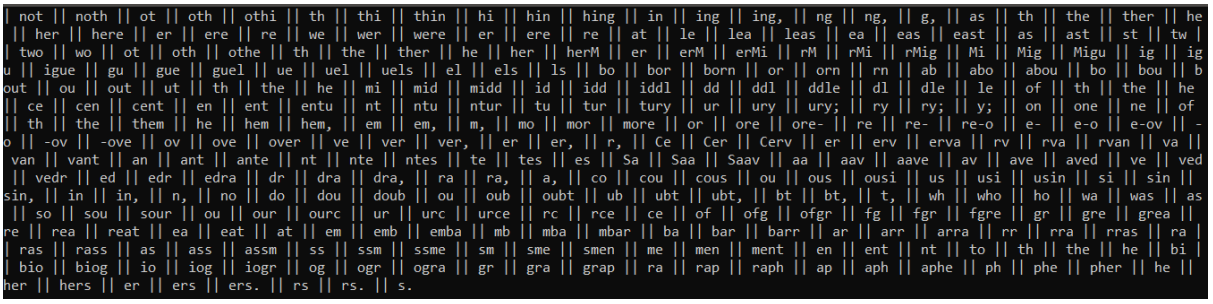


- EdgeNGramTokenFilter (tamaño mínimo 2, tamaño máximo 4): tokeniza cada token en n-gramas de 2 a 4 de longitud desde el principio del token original.

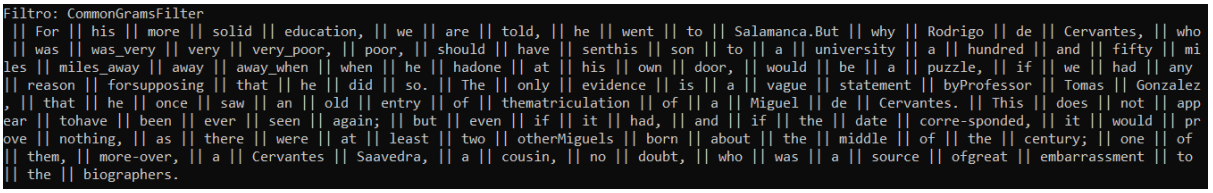


- NGramTokenFilter (tamaño mínimo 2, tamaño máximo 4): tokeniza cada token en n-gramas de 2 a 4 de longitud.

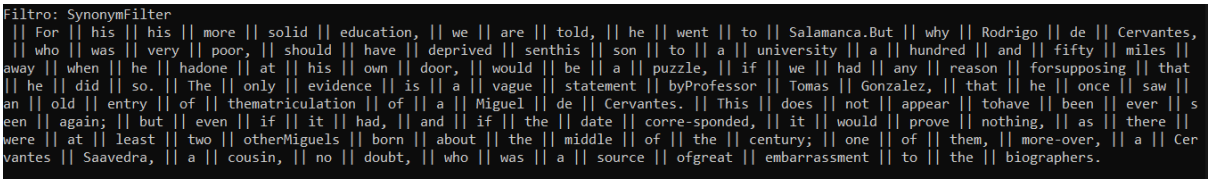




- CommonGramsFilter (se han determinado como comunes las palabras “very”, “poor” y “away”): elimina del TokenStream los tokens que hayan sido determinados como muy comunes, y por tanto irrelevantes.



- SynonymFilter (se han determinado como sinónimos “have”/“deprived” y “for”/”his” -no son sinónimos reales): empareja los tokens que hayan sido determinados como sinónimos.



Ejercicio 3 (Juan Manuel)

En este ejercicio se nos pedía hacer un analyzer propio, usando los tokenizer y los filtros que quisiesemos. Nosotros hemos utilizado el CustomAnalyzer para ello:

```
Analyzer analyzer = CustomAnalyzer.builder()
    .withTokenizer("standard")
    .addTokenFilter("lowercase")
    .addTokenFilter("apostrophe")
    .addTokenFilter("nGram","minGramSize","2","maxGramSize","2")
    .build();
```

Para referirnos a los distintos tokenizer y filtros hemos usado los SPI names, que nos ahorran tener que escribir el .class entero para cada cosa. Los utilizados son los siguientes:

- StandardTokenizer: Usa las reglas de rotura de palabras del algoritmo de segmentación de texto Unicode para separar los distintos tokens.
- LowerCaseFilter: Convierte los tokens a minúsculas.
- ApostropheFilter: Separa tokens de sus apostrofes.
- NGramTokenizer: Crea ngramas con cada token. El tamaño máximo y mínimo, especificado en la llamada al filtro, es de 4.

En este ejercicio vamos a utilizar hamlet.txt otra vez para generar la salida de tokens. La estructura inicial del main es igual a la del ejercicio 1, así que no hace falta explicarla.

```
Tika tika = new Tika();
Metadata metadata = new Metadata();
File archivo = new File("../test/hamlet.txt");

String text = new String();

try{
    text = tika.parseToString(archivo);
}catch (Exception e){
    System.out.println("No se puede parsear...\n\n");
}
```

Para mostrar el resultado simplemente hemos generado el TokenStream del Analyzer que acabamos de crear. Ahora recorremos el TokenStream con stream.incrementToken() y vamos imprimiendo cada uno de los tokens, habiendolos pasado a String con toString().

```
TokenStream stream = analyzer.tokenStream(null, text);

stream.reset(); //Se le llama antes de usar incrementToken()
while(stream.incrementToken())
    System.out.print(" || "+stream.getAttribute(CharTermAttribute.class).toString());

System.out.println();

stream.end(); //Se le llama cuando se termina de iterar
stream.close(); //Liberas los recursos asociados al stream
```

La salida es la siguiente:

```
jcpicco@DumbleDogg:/mnt/c/Users/jmcon/Documents/GitHub/ri/p2/ej3$ java -cp .:../lucene-7.1.0/core/lucene-core-7.1.0.jar:
../lucene-7.1.0/analysis/common/lucene-analyzers-common-7.1.0.jar:../tika-app-1.24.jar practica2_3 2>/dev/null
 || th || hi || is || et || te || ex || xt || is || ty || yp || po || co || or || rr || re || ec || ct || te || ed || ve
 || er || rs || si || io || on || of || sh || ha || ak || ke || es || sp || pe || ea || ar || re || ha || am || ml || le
 || et || pr || ro || oj || je || ec || ct || gu || ut || te || en || nb || be || er || rg || fi || il || le || 1w || ws
 || s2 || 26 || 61 || 10 || tx || xt || th || hi || is || eb || bo || oo || ok || wa || as || on || ne || of || pr || ro
 || oj || je || ec || ct || gu || ut || te || en || nb || be || er || rg || ea || ar || rl || ly || fi || il || le || es
 || pr || ro || od || du || uc || ce || ed || at || ti || im || me || wh || he || en || pr || ro || oo || of || fi || in
 || ng || me || et || th || ho || od || ds || an || nd || to || oo || ol || ls || we || er || re || no || ot || we || el
 || ll || de || ev || ve || el || lo || op || pe || ed || th || he || er || re || is || an || im || mp || pr || ro || ov
 || ve || ed || ed || di || it || ti || io || on || of || th || hi || is || ti || it || tl || le || wh || hi || ic || ch
 || ma || ay || be || vi || ie || ew || we || ed || as || eb || bo || oo || ok || 10 || 00 || at || ht || tt || tp || ps
 || ww || ww || w. || .g || gu || ut || te || en || nb || be || er || rg || g. || .o || or || rg || eb || bo || oo || ok
 || ks || 10 || 00 || th || hi || is || et || te || ex || xt || fi || il || le || is || pr || re || es || se || en || nt
 || te || ed || by || pr || ro || oj || je || ec || ct || gu || ut || te || en || nb || be || er || rg || in || co || oo
 || op || pe || er || ra || at || ti || io || on || wi || it || th || wo || or || rl || ld || li || ib || br || ra || ar
 || ry || in || nc || fr || ro || om || th || he || ei || ir || li || ib || br || ra || ar || ry || of || th || he || fu
 || ut || tu || ur || re || an || nd || sh || ha || ak || ke || es || sp || pe || ea || ar || re || cd || dr || ro || om
 || ms || pr || ro || oj || je || ec || ct || gu || ut || te || en || nb || be || er || rg || of || ft || te || en || re
 || el || le || ea || as || se || es || et || te || ex || xt || ts || th || ha || at || ar || re || no || ot || pl || la
 || ac || ce || ed || in || th || he || pu || ub || bl || li || ic || do || om || ma || ai || in || th || hi || is || et
 || te || ex || xt || ha || as || ce || er || rt || ta || ai || in || co || op || py || yr || ri || ig || gh || ht || im
 || mp || pl || li || ic || ca || at || ti || io || on || ns || yo || ou || sh || ho || ou || ul || ld || re || ea || ad
 || th || hi || is || el || le || ec || ct || tr || ro || on || ni || ic || ve || er || rs || si || io || on || of || th
 || he || co || om || mp || pl || le || et || te || wo || or || rk || ks || of || wi || il || ll || li || ia || am || sh
 || ha || ak || ke || es || sp || pe || ea || ar || re || is || co || op || py || yr || ri || ig || gh || ht || 19 || 99
 || 90 || 19 || 99 || 93 || by || wo || or || rl || ld || li || ib || br || ra || ar || ry || in || nc || an || nd || is
 || pr || ro || ov || vi || id || de || ed || by || pr || ro || oj || je || ec || ct || gu || ut || te || en || nb || be
 || er || rg || wi || it || th || pe || er || rm || mi || is || ss || si || io || on || el || le || ec || ct || tr || ro
 || on || ni || ic || an || nd || ma || ac || ch || hi || in || ne || re || ea || ad || da || ab || bl || le || co || op
```

Ejercicio 4 (Óscar)

En este ejercicio se pide crear un analizador que, dado un token, se quede con los últimos 4 caracteres de éste; en el caso de que tenga menos de 4 caracteres debe ignorarse. Para ello, hemos creado un filtro denominado FourLettersFilter.

```
public final class FourLettersFilter extends TokenFilter {
    private CharTermAttribute charTerm;
    private PositionIncrementAttribute posicion;


    public FourLettersFilter(TokenStream input){
        super(input);
        this.charTerm = addAttribute(CharTermAttribute.class);
        this.posicion = addAttribute(PositionIncrementAttribute.class);
    }

    @Override
    public boolean incrementToken() throws IOException{
        int saltos = 0;

        while(input.incrementToken()){
            if (charTerm.length() >= 4) {
                if (saltos != 0) {
                    posicion.setPositionIncrement(posicion.getPositionIncrement() + saltos);
                }

                char buffer[] = charTerm.buffer();
                char newBuffer[] = new char[4];

                for (int i = 0; i < 4 ; i++) {
                    newBuffer[i] = buffer[charTerm.length() - 4 + i];
                }

                charTerm.setEmpty();
                charTerm.copyBuffer(newBuffer, 0, newBuffer.length);
                return true;
            }

            saltos += posicion.getPositionIncrement();
        }

        return false;
    }
}
```

La clase del filtro tiene como variables privadas un objeto CharTermAttribute y un objeto PositionIncrementAttribute. El primero sirve para almacenar el texto de un token, el segundo para determinar la posición relativa del token actual con uno anterior. Hemos usado el objeto PositionIncrementAttribute porque, de no hacerlo, el analizador funciona correctamente, pero no ignora los tokens de menos de 4 caracteres y los incluye en el resultado.

Para que funcione el filtro debemos sobreescribir el método incrementToken de la clase abstracta superior FilteringTokenFilter. El funcionamiento es el siguiente: mientras que haya un token en el flujo, se comprueba si el token contiene 4 o más caracteres; en caso contrario se termina la ejecución del método devolviendo falso. Si es verdadero, se comprueba cuántos saltos se han realizado previamente. Si han habido saltos, se recalculan los saltos que han tenido lugar. Una vez hecho esto, se procede a procesar el token almacenándolo en un buffer. Se realiza entonces un bucle de 4 iteraciones en el que se obtienen y se almacenan los últimos 4 caracteres del token, y se devuelve verdadero como resultado del método.

Para la ejecución del analizador, hemos decidido reutilizar el código del ejercicio 2, puesto que implementa una modularización de la ejecución de filtros. Con ciertas variaciones, este es el programa resultante:

```java
public class practica2_4{
    public static void Filtro(TokenStream stream, String nombre_filtro) throws IOException{
        stream.reset(); //Se le llama antes de usar incrementToken()

        System.out.println("Filtro: " + nombre_filtro);
        while(stream.incrementToken()){ //Itera de token en token
            System.out.print(" || "+stream.getAttribute(CharTermAttribute.class).toString()); //Pasamos el apartado de texto del token a String
        }
        System.out.print("\n\n");

        stream.end(); //Se le llama cuando se termina de iterar
        stream.close(); //Liberas los recursos asociados al stream
    }

    public static void main(String[] args) throws IOException{
        Tika tika = new Tika();
        Metadata metadata = new Metadata();
        File archivo = new File("./test.txt");

        String text = new String();

        try{
            text = tika.parseToString(archivo);
        }catch (Exception e){
            System.out.println("No se puede parsear...\n\n");
        }

        Filtro(new FourLettersFilter(new StopAnalyzer().tokenStream(null, text)), "FourLettersFilter");
    }
}
```

El analizador escogido es el StopAnalyzer, pues al usar otros analizadores incluían también las stopwords como caracteres dentro de cada token. El resultado de la ejecución es el siguiente:

```
Filtro: FourLettersFilter
 || more || olid || tion || told || went || anca || rigo || ntes || very || poor || ould || have || this || sity || dred || ifty || iles || away || w
hen || done || door || ould || zzle || ason || sing || only || ence || ague || ment || ssor || omas || alez || once || ntry || tion || guel || ntes |
| does || pear || have || been || ever || seen || gain || even || date || orre || nded || ould || rove || hing || were || east || uels || born || bou
t || ddle || tury || them || more || over || ntes || edra || usin || oubt || urce || reat || ment || hers
```