TURUN AMMATTIKORKEAKOULU
ÅBO YRKESHÖGSKOLA

Bachelor’s Thesis

Development of a Plagiarism Detection Software

Meichelbeck Julien

2012

Abstract of thesis

University: Turku University of Applied Sciences (TUAS), Finland	
Degree program: Information Technology (Embedded Systems)	
Author: Meichelbeck Julien	
Title: Development of a Plagiarism Detection Software	
Instructor: Paalassalo Jari-Pekka	
Date: June 19, 2012	**Total number of pages:** 28

This thesis gives a working example on how to design and implement a plagiarism detection software in Java using various libraries. The software uses websites as datasources to determine if a text or a file is a plagiarized document or not.

The first chapter of the report is describing how the software works, which tools were used with of a focus on the different Java libraries. This part deals also with the technical requirements.

The main part is describing how the program has been designed (class diagrams, design decisions, etc.). An analysis of the implementation is also developed in this part, beginning with the Levenshtein algorithm description.

The final part is showing the user interface, especially the different windows of the software, and deals also with the possible improvements.

Keywords: Plagiarism, detection software, parsing, online data harvester.

Deposit at: Library of Turku University of Applied Sciences

Acknowledgments

I would like to express my gratitude to my supervisor, Jari-Pekka Paalassalo, whose expertise and understanding, added considerably to my graduate experience this year. I would also like to thank my family for their support and my friends, with whom I had a lot of fun during this year. And finally, I thank the Erasmus program for having given me the opportunity to study abroad in this wonderful country.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualication except as specified.

Metz, June 19, 2012 ___________________

Meichelbeck Julien

Foreword

Revision history:

18.06.2012	Final and last modifications
17.06.2012	Styles changed and two pages schema added
10.06.2012	Conclusion
03.06.2012	Project requirements, implementation and class diagrams
02.06.2012	Project testing, abstract
30.05.2012	Introduction, Technology description, Tools
20.05.2012	Project design
15.05.2012	Styles, presentation, and variables created
13.05.2012	First version of this document

Printing and copying (hard copies or pdf-files) of this document is allowed.

Contents

Introduction

Nowadays, plagiarism is a really serious issue within the professional environment, or even within the education system. Since Internet is accessible to everyone, it is easy to use Internet as a source of information. However, copying documents from Internet can be considered as plagiarism: what can be found on Internet can come from a book, a research document or an article. It can even result to some legal problems, such as copyright infringement.

Although many of the laws and concepts are not new, the intellectual property concept is relatively recent, dating from the 19th century and this notion has been reassessed the last years with the creation of new online datasources such as Wikipedia, or even through the development of advanced search engines such as Google.

Since then, a new kind of software has emerged, the plagiarism detection softwares. There are several types of them; they can use databases (of thesis, books, or articles), Internet or comparison between files. This project is focusing on the development of an application, mostly using the extraction of data from Internet to check plagiarism and file to file comparisons. Because the project is a typical software development work, the thesis has main focus on implementing this software; therefore it is mainly a technical report.

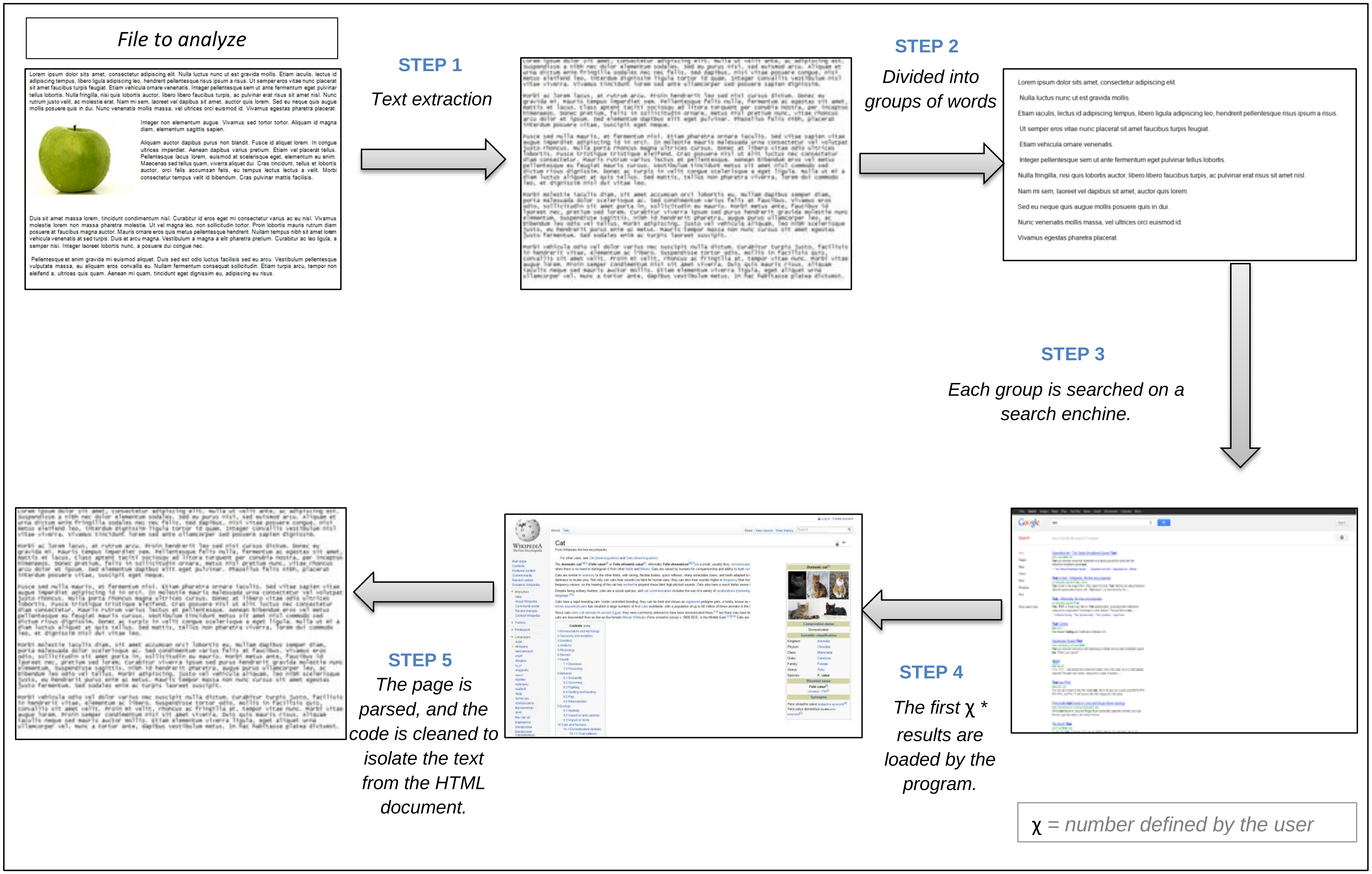
1. Technology description

This part will describe how the software works without going into details. There are two available features: Importing one single file for online plagiarism check, or importing two file for a comparative check. The online plagiarism check corresponds to the biggest part of the project, because the comparative check uses the same classes created for the online plagiarism check.

Concerning this part, the user is able to import files (doc, txt or pdf) to check for plagiarism. Then, the user can proceed to the configuration of the plagiarism detection through the configuration window. Then, the file analysis can start, and can be described in several steps.

STEP 1	The text is exported from the file ignoring the pictures, diagrams or other figures. A variable is created to save the text.
STEP 2	The text is divided into a lot of groups of words. These groups correspond to sentences by default.
STEP 3	Each group is searched by the software on a search engine.
STEP 4	The page containing the search engine results is loaded and then, the page from a result is loaded.
STEP 5	When the website has been loaded, the page is parsed, to extract the text from the HTML code. All the text contained on this page is isolated.
STEP 6	The sentence is searched inside the extracted text.
STEP 7	If a similar sentence has been found, the source is added to the source list and the next sentence starts to be analyzed (back to *STEP 3*).
STEP 8	If the sentence has not been found, another website is loaded (back to *STEP 4*) until χ results were analyzed. χ being a number defined by the user (default value: 3).

The two following pages correspond to a schema illustrating these steps.


File to analyze
STEP 1
Text extraction
STEP 2
Divided into groups of words
STEP 3
Each group is searched on a search enchine.
STEP 4
The first χ * results are loaded by the program.
STEP 5
The page is parsed, and the code is cleaned to isolate the text from the HTML document.
χ = number defined by the user

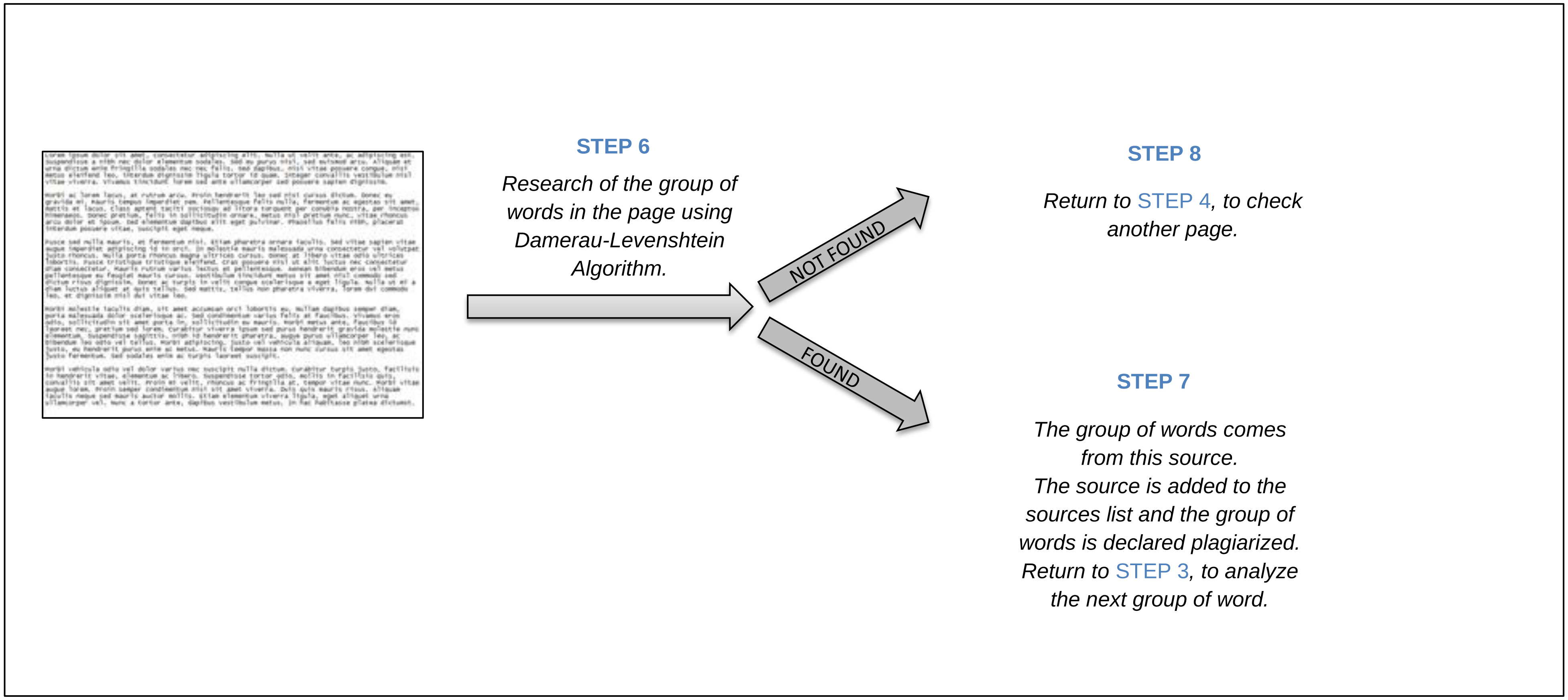

STEP 6
Research of the group of words in the page using Damerau-Levenshtein Algorithm.
NOT FOUND
FOUND
STEP 8
Return to STEP 4, to check another page.
STEP 7
The group of words comes from this source.
The source is added to the sources list and the group of words is declared plagiarized.
Return to STEP 3, to analyze the next group of word.

Figure 1 – Schema showing the steps for a plagiarism check

2. Tools

2.1 Java

This software has been programmed in Java. Java is a programming language and computing platform first released by Sun Microsystems in 1995. It is the underlying technology that powers state-of-the-art programs including utilities, games, and business applications. Java runs on more than 850 million personal computers worldwide, and on billions of devices worldwide, including mobile and TV devices (Oracle Technology Network, 2010).

The version which has been chosen for this project is *JRE System 1.7*. This version contains important enhancements to improve performance, stability and security of the Java applications.

Java is known for his large number of libraries. Indeed, Sun provides a large number of frameworks and API in order to allow a lot of diversified uses. This is why Java was probably the best choice, at least the most suitable language, for the implementation of this project.

2.2 Java libraries

Jsoup 1.6.2

Jsoup is a Java library for working with real-world HTML. It provides a very convenient API for extracting and manipulating data, using the best of DOM, CSS, and jquery-like methods.

Jsoup implements the *WHATWG HTML5* [1] specification, and parses HTML to the same DOM as modern browsers do.

- scrape and parse HTML from a URL, file, or string
- find and extract data, using DOM traversal or CSS selectors
- manipulate the HTML elements, attributes, and text
- clean user-submitted content against a safe white-list, to prevent XSS attacks
- output tidy HTML

Jsoup is designed to deal with all varieties of HTML found in the wild; from pristine and validating, to invalid tag-soup; jsoup will create a sensible parse tree.

[1] Web Hypertext Application Technology Working Group HTML5 - *http://www.whatwg.org/*

Jsoup example Code: Java

```java
//  Connection to the website to get the source code
//  USER_AGENT = Chrome/14.0.835.186 (example)
//  REFERRER   = Previous page
gDoc = Jsoup.connect(URL).referrer(REFERRER).userAgent(USER_AGENT).get();

// Selection of the DOM path, to reach what we want to extract
Elements titles = gDoc.select("h3.r > a");

// Display the result
System.out.println(titles.get(0).text());
```

Apache Poi 3.8

The Apache POI library is used to manipulate various file formats based upon the Office Open XML standards (OOXML) and Microsoft's OLE 2 Compound Document format (OLE2). In short, it allows reading and writing MS Excel, MS Word, or MS PowerPoint files using Java.

Apache Poi example Code: Java

```java
// Creation of the variable which will contain the text
String parsedText = new String();
POIFSFileSystem fs = null;
try{
	fs = new POIFSFileSystem(new FileInputStream(PATH));
	HWPFDocument doc = new HWPFDocument(fs);
	WordExtractor we = new WordExtractor(doc);
	String[] paragraphs = we.getParagraphText();
	for( int i=0; i<paragraphs .length; i++ ) {
		// Adding the extracted text to the variable
		parsedText += paragraphs[i];
	}
}
catch(Exception e){
	System.err.println("POI extraction problem");
	e.printStackTrace();
}
```

Apache PDFBox 1.6

The Apache PDFBox library is an open source Java tool for working with PDF documents. This project allows creation of new PDF documents, manipulation of existing documents and the ability to extract content from documents. Apache PDFBox also includes several command line utilities.

Apache PDFBox example Code: Java

```java
// Creation of the variable which will contain the text
String parsedText = new String();

parser.parse();
COSDocument cosDoc = parser.getDocument();
PDFTextStripper pdfStripper = new PDFTextStripper();
PDDocument pdDoc = new PDDocument(cosDoc);
// Adding the extracted text to the variable
parsedText = pdfStripper.getText(pdDoc);
```

JFreeChart

JFreeChart is a Java chart library that makes it easy for developers to display charts in their applications.

JFreeChart example Code: Java

```java
// Dataset declaration, will contain the data for the chart
DefaultPieDataset dataset = new DefaultPieDataset();

// Adding values to the pie chart
dataset.setValue(5,6)

// Creation of the chart
JFreeChart C = ChartFactory.createPieChart3D(
     "Title of the chart",  // chart title
      dataset,                 // data
      true                    // include legend
      );
PiePlot3D plot = (PiePlot3D) C.getPlot();
```

2.3 Google Search

Google Search (or Google Web Search) is a web search engine owned by *Google Inc.* Google Search, is the most-used search engine on the World Wide Web, receiving several hundred million queries each day through its various services.

There are several steps we need to go through to gather data from Internet to check if a sentence has been plagiarized. The first one is to use a search engine. Google is probably the fastest and trustworthy search engine there is nowadays. We assume that if Google cannot find a sentence at all, then it doesn't come from Internet and therefore, this sentence hasn't been plagiarized.

3. Project requirements

There are several factors we need to think about to choose the limits of the project. In fact, an internet data harvester is a wide area, and its improvement has no limit, it can always be better than it used to be. The number of data sources that can be found on Internet is almost limitless, that's why the limits must be set carefully.

Good connection speed

One of the most important factors is the speed internet connection. If the internet connection is too slow, the data gathering can be really slow, because the application will wait to load the page before it can work on it. That's why, it is not recommended to do a very detailed research, which will cause a very long data analysis, with a slow internet connection.

A Google request is made on the sentence that we are checking. Then, we can process to the analysis of the first N pages that Google has found. N is a number that the user can choose (still according to the internet speed), the default value is 3. The higher the number is, the more necessary it becomes to have a good connection.

Java Runtime Environment

JRE version 1.7 is required to launch the software, or to compile the files.

4. Project design

The following diagrams don't contain the getters and setters, since their number is too high. Almost all the private attributes have getters and setters.

4.1 Internet research class diagram

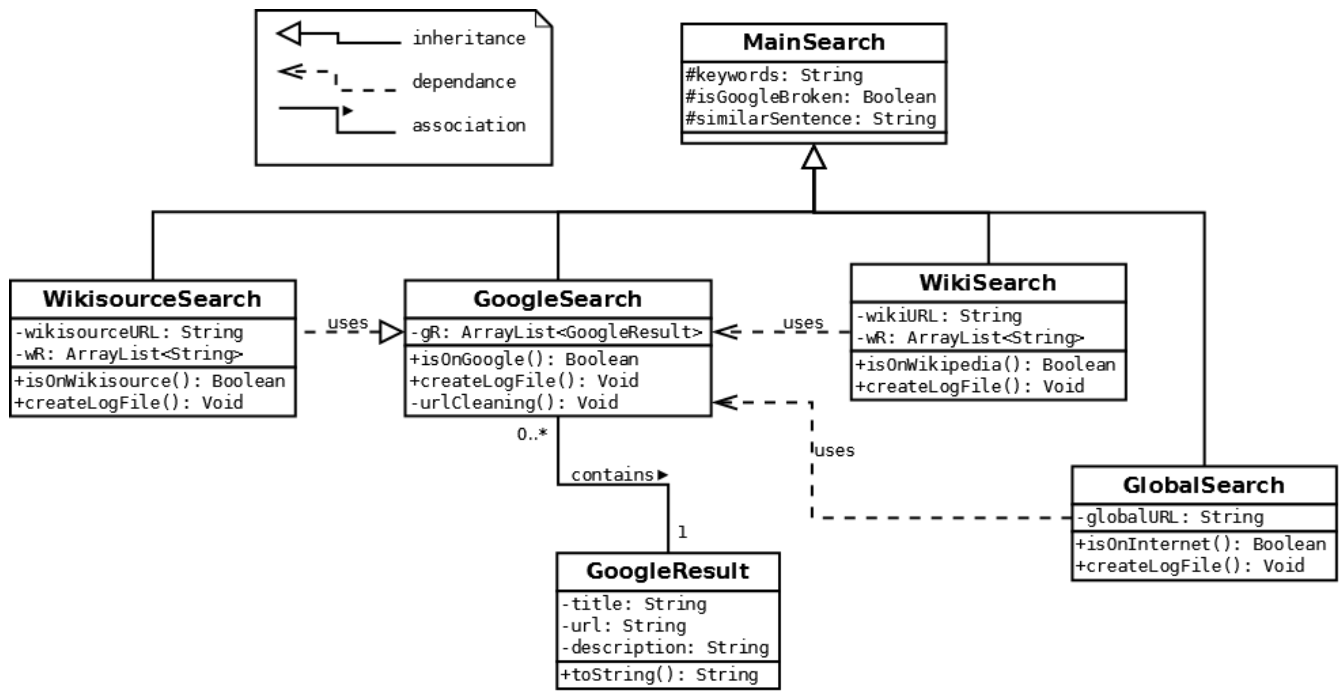


Figure 2 - Internet research class diagram

These classes are used for the internet searches. The main class *MainSearch* contains the variables shared by different classes.

During, the plagiarism analysis, the program performs a lot of Google query through the *GoogleSearch* class. For each *GoogleSearch*, there are ten *GoogleResult* variables, which are corresponding to a result given by Google.

Then, the first results are analyzed by different parsers, detailed in the others classes: *WikiSearch, GlobalSearch, WikiSourceSearch*. There are several classes to execute the research, because the website structures are different, and therefore the research is faster this way.

4.2 Plagiarism detection class diagram

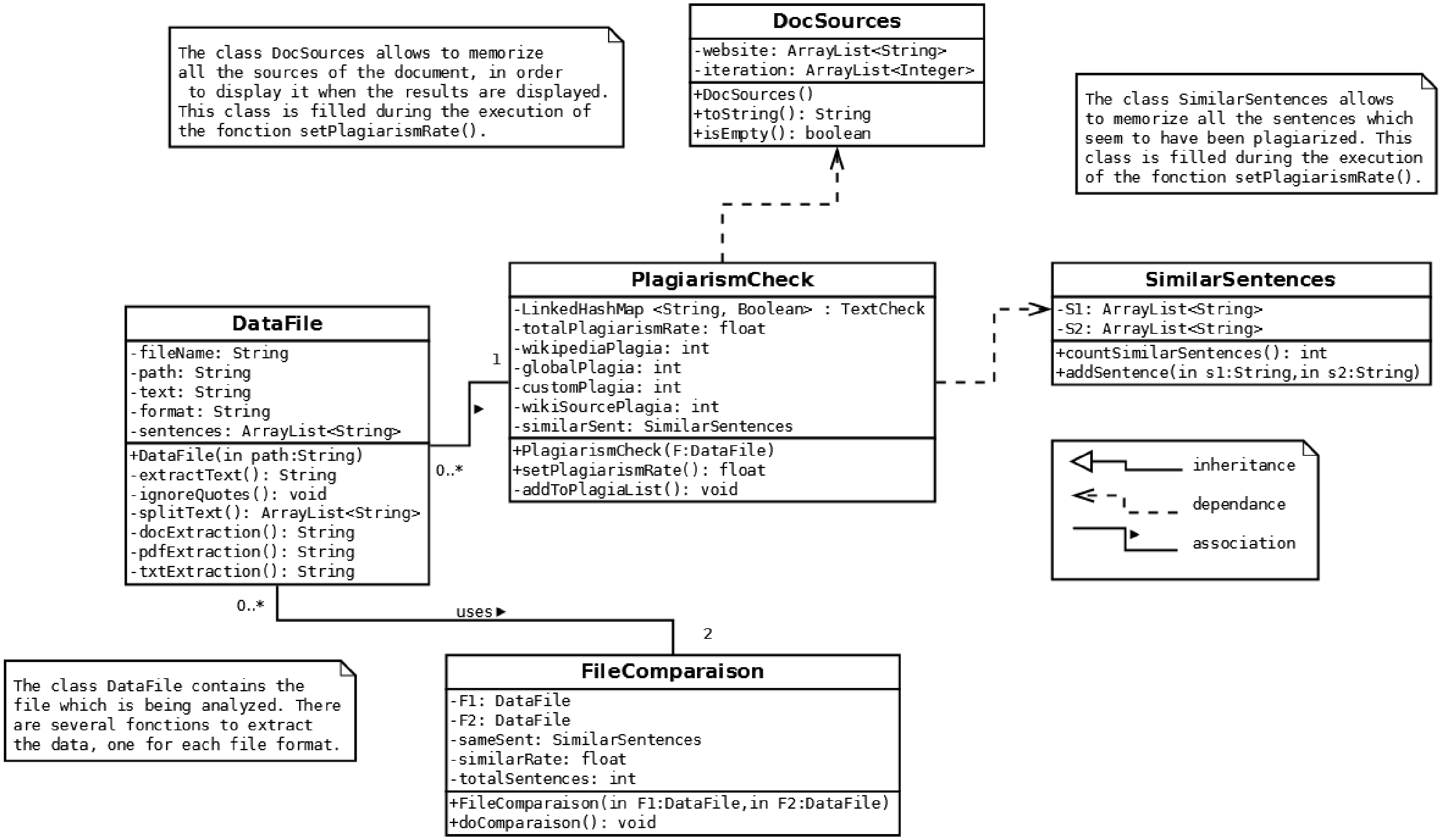

The class DocSources allows to memorize all the sources of the document, in order to display it when the results are displayed. This class is filled during the execution of the fonction setPlagiarismRate().
DocSources
-website: ArrayList<String>
-iteration: ArrayList<Integer>
+DocSources()
+toString(): String
+isEmpty(): boolean
The class SimilarSentences allows to memorize all the sentences which seem to have been plagiarized. This class is filled during the execution of the fonction setPlagiarismRate().
PlagiarismCheck
-LinkedHashMap <String, Boolean> : TextCheck
-totalPlagiarismRate: float
-wikipediaPlagia: int
-globalPlagia: int
-customPlagia: int
-wikiSourcePlagia: int
-similarSent: SimilarSentences
+PlagiarismCheck(F:DataFile)
+setPlagiarismRate(): float
-addToPlagiaList(): void
SimilarSentences
-S1: ArrayList<String>
-S2: ArrayList<String>
+countSimilarSentences(): int
+addSentence(in s1:String,in s2:String)
DataFile
-fileName: String
-path: String
-text: String
-format: String
-sentences: ArrayList<String>
+DataFile(in path:String)
-extractText(): String
-ignoreQuotes(): void
-splitText(): ArrayList<String>
-docExtraction(): String
-pdfExtraction(): String
-txtExtraction(): String
1
0..*
inheritance
dependance
association
0..*
uses ▸
2
FileComparaison
-F1: DataFile
-F2: DataFile
-sameSent: SimilarSentences
-similarRate: float
-totalSentences: int
+FileComparaison(in F1:DataFile,in F2:DataFile)
+doComparaison(): void
The class DataFile contains the file which is being analyzed. There are several fonctions to extract the data, one for each file format.

Figure 3 – Plagiarism detection class diagram

4.3 User Interface class diagram

The following diagram does not contain all the attributes. Each class has a lot of attributes due to the high number of graphic objects.

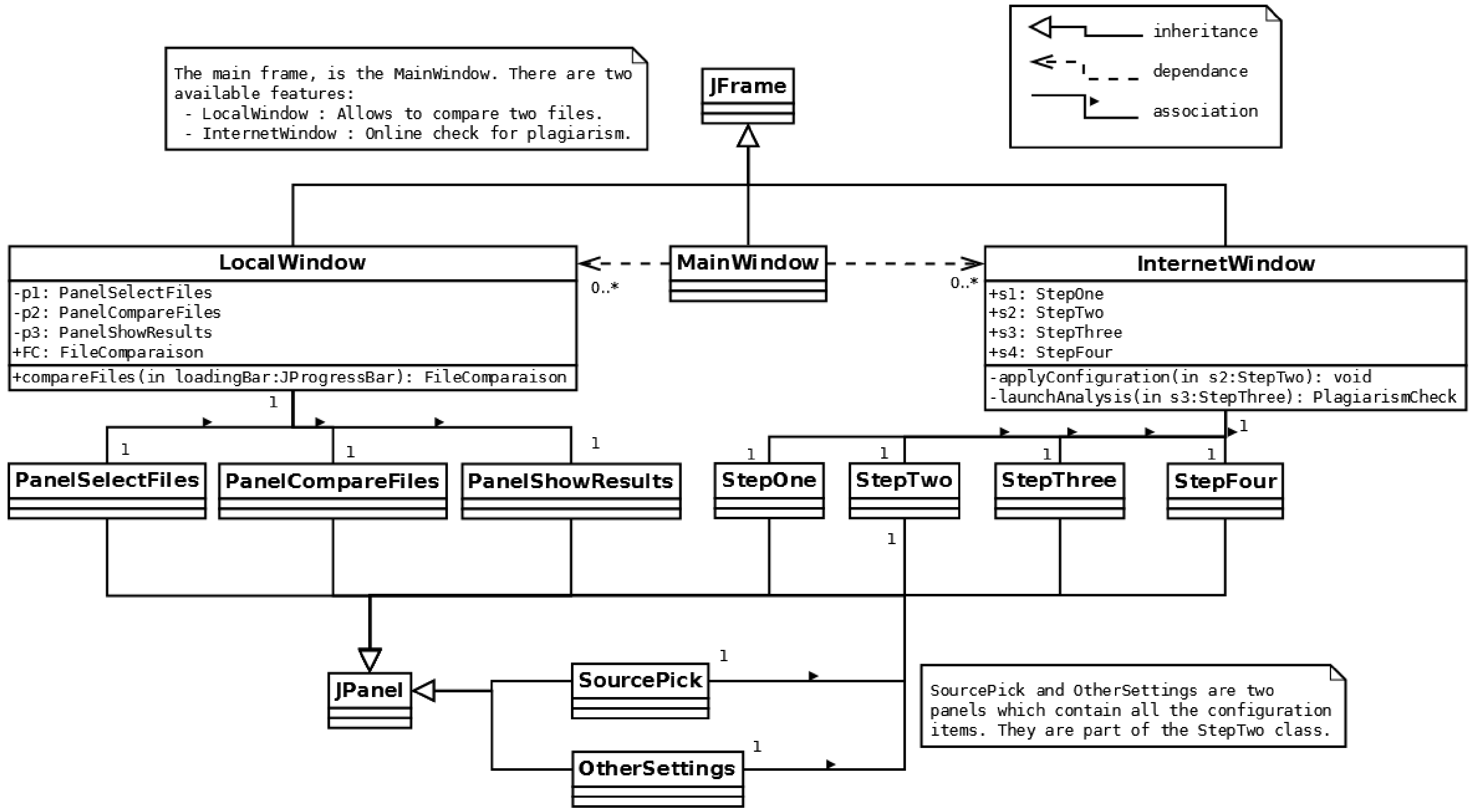

The main frame, is the MainWindow. There are two available features:
- LocalWindow : Allows to compare two files.
- InternetWindow : Online check for plagiarism.
inheritance
dependance
association
JFrame
LocalWindow
-p1: PanelSelectFiles
-p2: PanelCompareFiles
-p3: PanelShowResults
+FC: FileComparaison
+compareFiles(in loadingBar:JProgressBar): FileComparaison
MainWindow
InternetWindow
+s1: StepOne
+s2: StepTwo
+s3: StepThree
+s4: StepFour
-applyConfiguration(in s2:StepTwo): void
-launchAnalysis(in s3:StepThree): PlagiarismCheck
0..*
0..*
1
PanelSelectFiles
PanelCompareFiles
PanelShowResults
StepOne
StepTwo
StepThree
StepFour
JPanel
SourcePick
OtherSettings
SourcePick and OtherSettings are two panels which contain all the configuration items. They are part of the StepTwo class.

Figure 4 – User Interface class diagram

4.4 Configuration file and constants

All the classes of the project implement the class *Constant* which contains all the constants of the project. It can be default values for some parameters or fixed values.

There is also a configuration file: *Config* which contains all the configuration variables. These variables are static, since *Config* isn't instantiated. They can be called or modified at any moment during the execution of the program.

Example: Configuration variables.

Some Configuration variables | Code: Java

```java
//These variables are modified when the user changes the sources of the research
public static boolean CUSTOM_CHECK;
public static boolean WIKIPEDIA_CHECK;
public static boolean WIKISOURCE_CHECK;
public static boolean GOOGLE_CHECK;
public static boolean GLOBAL_CHECK;

//This variable contains the URL of the custom source, if the user uses one
public static String CUSTOM_URL;

//This variable is changed if the user wants a detailed report of the analysis

public static boolean DETAILED_REPORT;
```

5. Implementation

5.1 Damerau - Levenshtein Algorithm

The Levenshtein distance is a string metric for measuring the amount of difference between two sequences.

The Levenshtein distance between two strings is defined as the minimum number of edits needed to transform one string into the other. The allowable edit operations (they concern one single character) are:

- Insertion
- Deletion
- Substitution

Vladimir Levenshtein established this algorithm in 1965. Damerau developed a new version of the Levenshtein Distance, which takes into account a new operation: the transposition of two adjacent characters.

Example: Levenshtein Distance between "*kitten*" and "*sitting*".

1. **k**itten → **s**itten (substitution of 's' for 'k')
2. sitt**e**n → sitt**i**n (substitution of 'i' for 'e')
3. sittin → sittin**g** (insertion of 'g' at the end).

The Levenshtein Distance between these strings is **3**, because it requires three operations to go from one string to another.

Computing the Levenshtein distance is based on the observation that if we reserve a matrix to hold the Levenshtein distances between all prefixes of the first string and all prefixes of the second, then we can compute the values in the matrix in a dynamic programming fashion, and thus find the distance between the two full strings as the last value computed (Wikipedia, 2012).

Computed Damerau-Levenshtein Algorithm example:

Damerau-Levenshtein Algorithm application	Code: Java

```java
// S and T are the two strings we want to compare
int n = s.length(); // length of s
int m = t.length(); // length of t
if (n == 0) {
      return m;
}
else if (m == 0){
      return n;
}

int p[] = new int[n+1];
int d[] = new int[n+1];
int _d[];

int i;
int j;
char t_j;
int cost;

for (i = 0; i<=n; i++) {
     p[i] = i;
}

for (j = 1; j<=m; j++) {
      t_j = t.charAt(j-1);
      d[0] = j;
      for (i=1; i<=n; i++) {
            cost = s.charAt(i-1)==t_j ? 0 : 1;
            d[i] = Math.min(Math.min(d[i-1]+1, p[i]+1),  p[i-1]+cost);
      }

      _d = p;
      p = d;
      d = _d;
}

return p[n]; // Levenshtein Distance between S and T
```

5.2 Substring matching

In this particular case, the Levenshtein Distance is used to calculate the distance between different groups of words (mostly sentences). Once the distance has been calculated, it is possible to calculate a percentage of similarity between the strings by dividing the Levenshtein-Damerau distance by the size of the longest string.

Example: Percentage of similarity between "*kitten*" and "*sitting*".

Levenshtein Distance D = 3
Size of the longest string S = 7
*(D / S) * 100 = 42.85%*

This method can be applied to longer strings in order to count the percentage of similarity between these strings, using the distance and the length of the sentence (number of characters).
If this percentage is bigger than a certain value (which has been decided by the user previously, the default value is 60%), we can tell that this sentence has been plagiarized.

5.3 Parsers

The parsers are essential to get informations and data from websites. As explained earlier, Jsoup is the library used to parse the source codes.

For some websites, such as Wikipedia, Wikisource or Google, the software is checking directly inside some very specific HTML tags. For example, the HTML path to reach the texts coming from Wikipedia is: *div.mw-content-ltr > p*.

Using parsers to access directly to the HTML code that matters allows the program to run much faster. However, when the program is analyzing random websites found on Google, it doesn't know where the text is, except that it is among the HTML code. This is why the parser isn't that useful in those pages: The program will just keep the <BODY> part of the HTML code.

Algorithm running through Google results and parsing | **Code: Java**

```java
// New Google query
GoogleSearch gS = new GoogleSearch(keywords);
// Get the number of websites the user wants to analyze per Google query
int accuracy = Config.SEARCH_ACCURACY;

// Research on all the first [accuracy] pages
for (int j = 0; j < accuracy; j++) {
       // Get the URL of the GoogleResult
       globalURL = gS.get(j).getUrl();
       // Connection to the website
       Document wDoc = Jsoup.connect(globalURL).timeout(10000).
                       ignoreHttpErrors(true).
                       ignoreContentType(true).
                       referrer(REFERRER).
                       userAgent(USER_AGENT).get();

       // Get the code, parse the page to get the <BODY> part
       Elements bodyPage = wDoc.select(GLOBAL_TEXT);
       // Clean the code
       String htmlCode = StringEscapeUtils.unescapeHtml4((bodyPage.get(0).toString());

       if(!htmlCode.isEmpty()){
        // Let's split the text of the page in different sentences
        StringTokenizer sToken = new StringTokenizer(htmlCode, Config.SENTENCES_SEPARATORS);
        // For each string (they are mostly sentences), check the variation
        rate between the string we're analyzing and the string concerned
        while (sToken.hasMoreElements()) {
               String globalSentence = sToken.nextToken().trim();
               int levenPercentage = Config.LEVENSHTEIN_PERCENTAGE ;
               int dist =LevenshteinDistance.getVariationRate(this.keywords, globalSentence);
               // If they are similar, return true and add the sentence to the list of
                     plagiarized sentences
               if(dist > levenPercentage){
                      setSimilarSentence(globalSentence);
                      return true;
               }
        }
       }
}
```

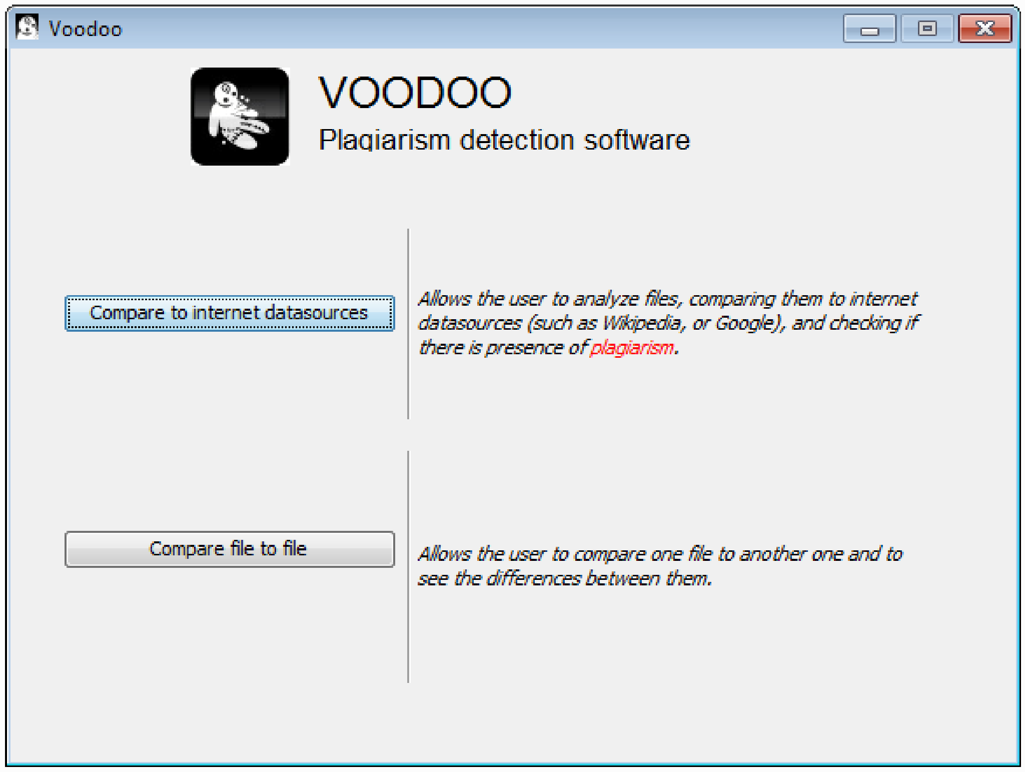
Then, the software cleans the HTML code of the website using another parser. One of the most powerful tools that can be used for that is the regular expressions. A regular expression which has been well written can get rid of most of the tags, to finally allow the program to access to the real and readable text.

6. Project testing

6.1 Main Window

This part will describe how the final version of the software actually works. The following screenshots come from the user interface developed in *Java Swing*.

When the user opens the jar file, the main window is displayed. It describes briefly the two different available features in the software, the online plagiarism check and the file to file comparison.


Voodoo
VOODOO
Plagiarism detection software
Compare to internet datasources
Compare file to file

Allows the user to analyze files, comparing them to internet datasources (such as Wikipedia, or Google), and checking if there is presence of plagiarism.

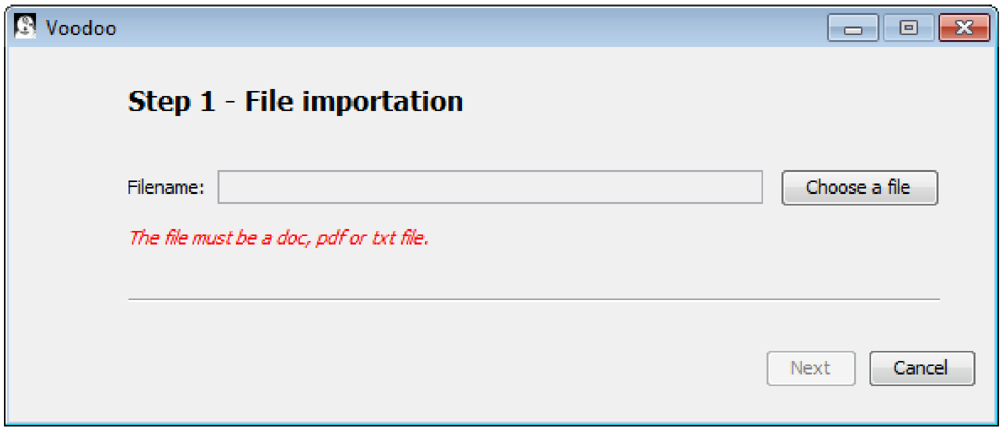
Allows the user to compare one file to another one and to see the differences between them.

Figure 5 – Main window

6.2 Online plagiarism check

Importation window

First of all, the user should import the file he wants to analyze. The file must be a MS Word document, PDF or txt file.



The file must be a doc, pdf or txt file.

Figure 6 – File importation window

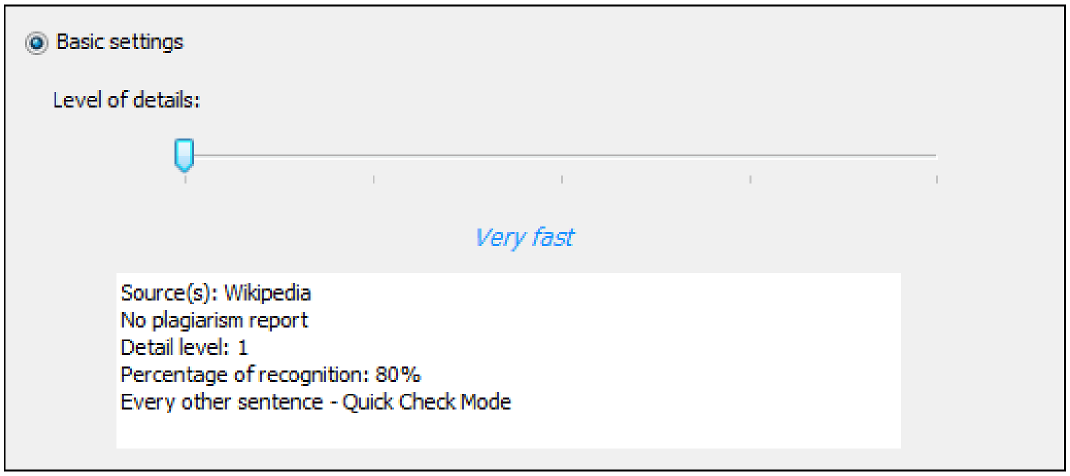
Configuration window

Once the file has been imported, the next step is the configuration of the plagiarism check. The more the research is accurate, the more it will be slow.

There are two kinds of way to configure the analysis:

1. *Basic settings*

The basic settings window contains only one item: a slider. This method is useful for a user who doesn't have the time to focus on the different parameters, or doesn't have the time to configure every parameter. The slider goes to "*Very fast analysis*" to "*Very slow analysis*". Each time the slider moves, the hidden parameters change.


◉ Basic settings

Level of details:

Very fast

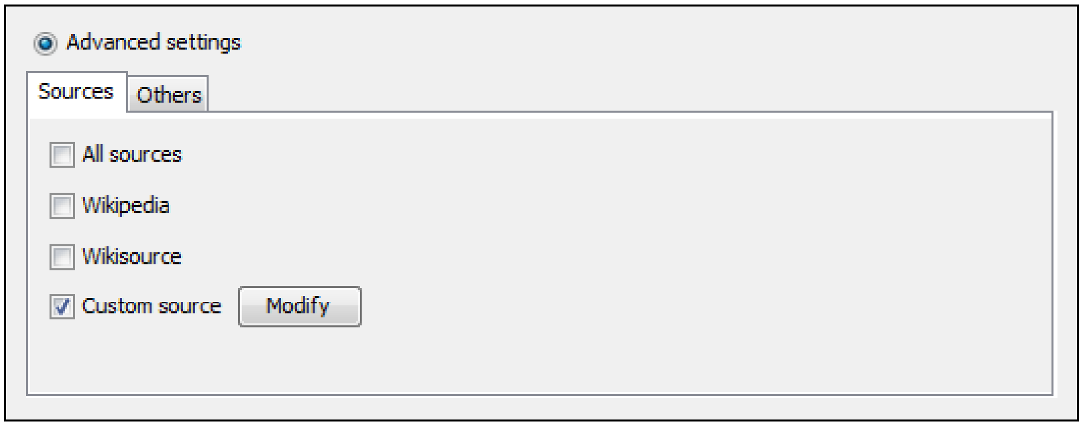
Source(s): Wikipedia
No plagiarism report
Detail level: 1
Percentage of recognition: 80%
Every other sentence - Quick Check Mode

Figure 7 – Configuration window, Basic settings panel

2. *Advanced settings*

The advanced settings window contains a lot of checkboxes, and textfields. This method is useful for a user who wants to choose his own parameters, to make the analysis accurate or not.

There are several parameters that can be changed, even if there are default values. There are two tabs: “Sources”, which allows to pick the sources and “Others” which contains the other parameters.


◉ Advanced settings

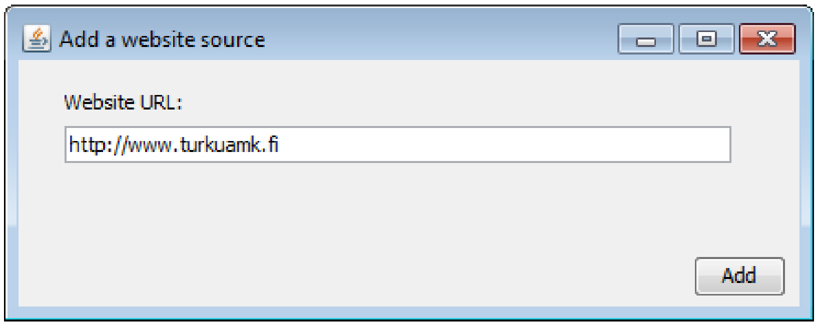
Sources | Others

☐ All sources
☐ Wikipedia
☐ Wikisource
☑ Custom source

Figure 8 – Configuration window, Advanced settings panel, Sources tab

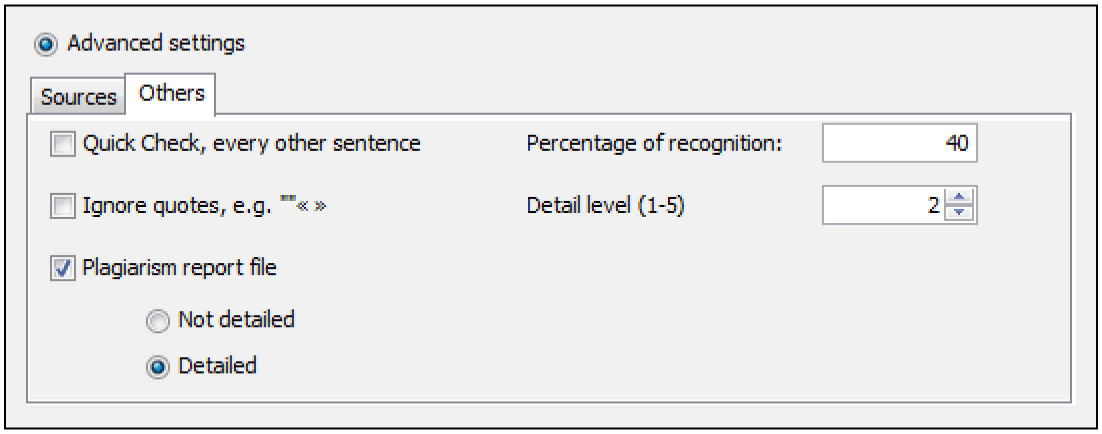
This tab allows the user to pick the source he wants to be used by the software for this research. The "All sources" checkbox will disable the other checkboxes, and the software will check for plagiarism in all the websites.

The "Custom source" checkbox allows adding any website as source. For instance, if it's the only checked checkbox, the software will check for plagiarism using only this website as a source.


Add a website source

Website URL: http://www.turkuamk.fi

Figure 9 – Adding custom source window


Advanced settings [✓]

Quick Check, every other sentence [ ]

Ignore quotes, e.g. ""« » [ ]

Plagiarism report file [✓]

Not detailed [ ]

Detailed [✓]

Percentage of recognition: 40

Detail level (1-5) 2

Figure 10 – Configuration window, Advanced settings panel, Other parameters tab

This tab contains all the parameters that can be changed to refine or stretch the search field.

Quick Check

If this option is enabled, the program will check every other sentence. The results won't be accurate, but it can be useful to have a global idea of the plagiarism rate in a very large file.

Ignore quotes

If this option is enabled, the program will ignore the quotes (which shouldn't be considered as plagiarism). Therefore, there can be a small margin of error and there should not be any mistake or typo concerning the quote characters in the file.

Plagiarism report file

This option will create a *log file* for the incoming research. This file will be a text file, describing in details all the results (plagiarism sources, percentage rate, plagiarized sentences …). The file path will be displayed in the result window. The log file can be detailed or not. A detailed report file will display all the sources, whereas a non-detailed log file will just show the major sources.

Percentage of recognition

The percentage of recognition (default value: 40%) corresponds to the percentage of similarity between two sentences (see *5.2 Substring matching*). It is recommended to keep the default values, which seems to be most reliable value.

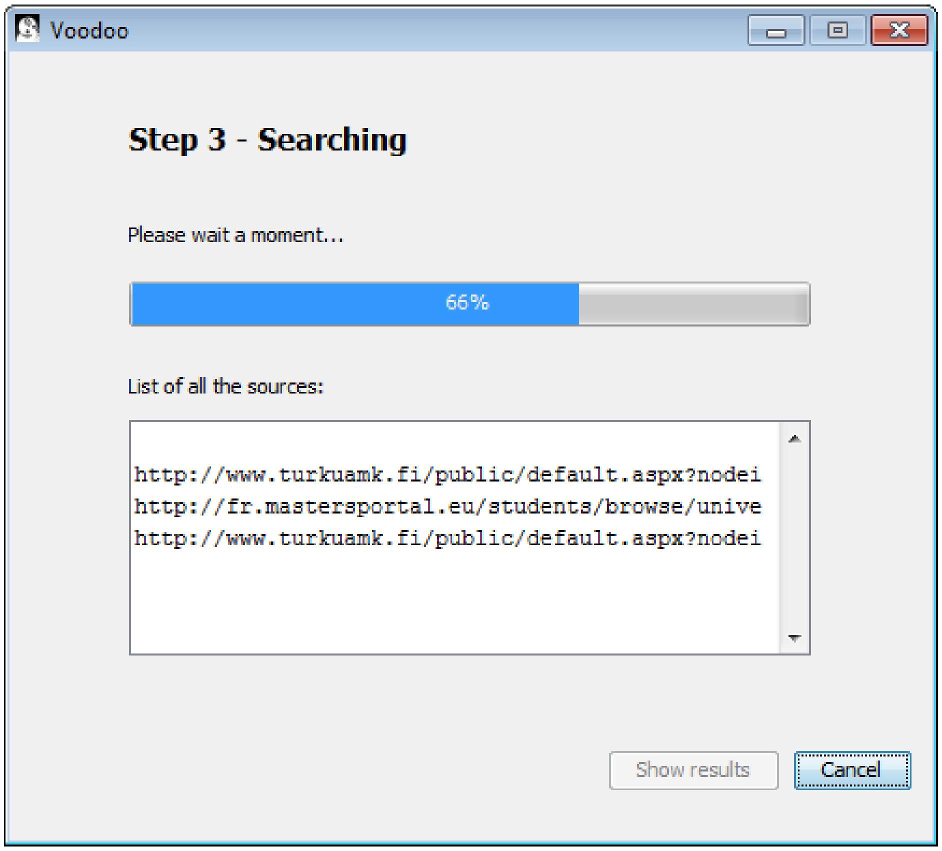
Detail level

The detail level is the maximum number of results from the search engine that will be loaded. For instance, if a sentence hasn't been found inside the first result, the program will check the next one until this number. This number has a **significant impact** on the duration of the plagiarism check.

Loading window

No action is allowed during the execution of this window. When the analysis starts, a second thread is created for its execution. Multithreading allows to display and update in real-time the loading bar and therefore the user knows approximately when the research is about to end.

The multithreading allows not only the real-time display, but also the real-time update of the sources list. In fact, this process informs the user about the sources of this document, and he is therefore able to get a global idea about the plagiarism rate during its analysis. When a group of words has been identified as plagiarism, the URL of the website is displayed on the list.



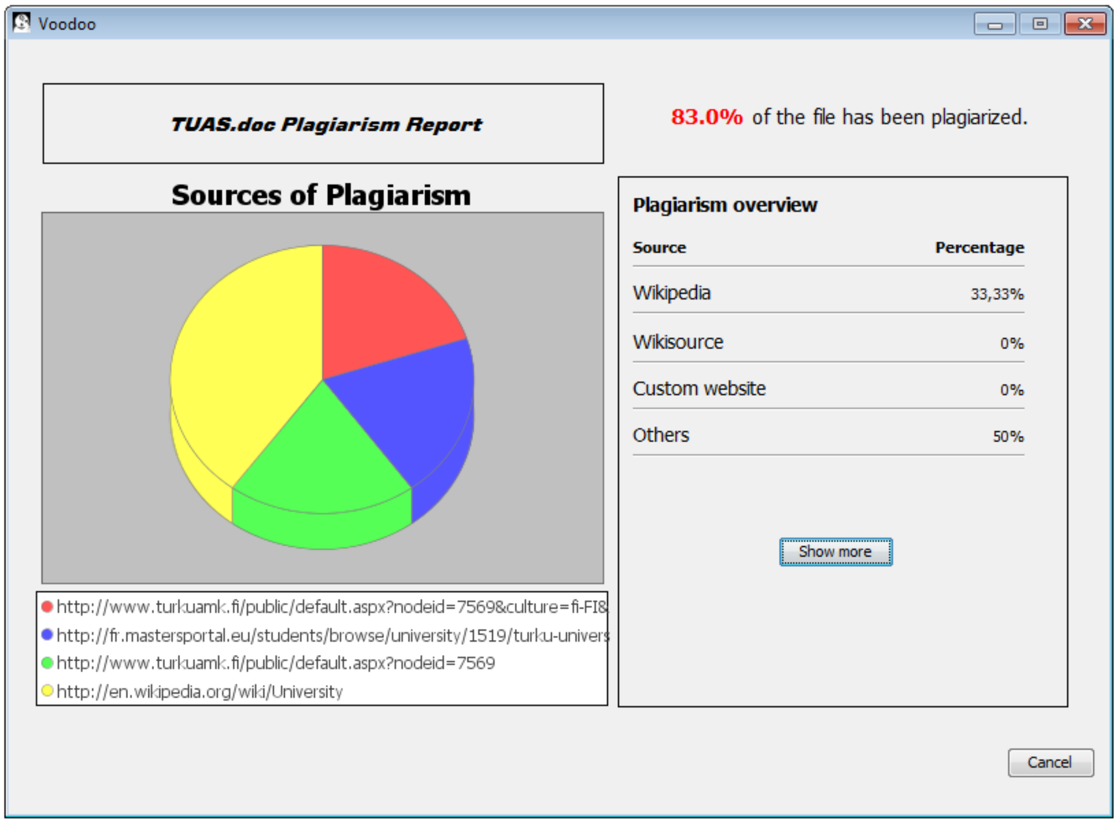
```
http://www.turkuamk.fi/public/default.aspx?nodei
http://fr.mastersportal.eu/students/browse/unive
http://www.turkuamk.fi/public/default.aspx?nodei
```

Figure 11 – Loading window

<u>Results window</u>

Once the loading is completed, the user can access to the last window, which is the results window. It displays all the informations about the plagiarism analysis that the program has calculated.

There is a pie chart showing all the distribution of the sources and the global plagiarism rate is displayed on the top-left part of the window. If a report file has been asked, its path is shown under the pie chart.



Voodoo

TUAS.doc Plagiarism Report

83.0% of the file has been plagiarized.

Sources of Plagiarism

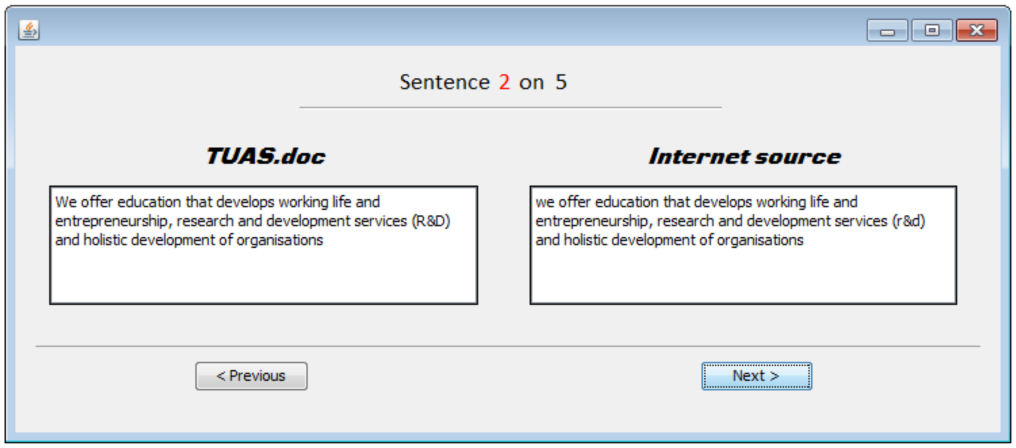
🟥 http://www.turkuamk.fi/public/default.aspx?nodeid=7569&culture=fi-FI&
🟦 http://fr.mastersportal.eu/students/browse/university/1519/turku-univers
🟩 http://www.turkuamk.fi/public/default.aspx?nodeid=7569
🟨 http://en.wikipedia.org/wiki/University

Plagiarism overview

Source	Percentage
Wikipedia	33,33%
Wikisource	0%
Custom website	0%
Others	50%

Figure 12 – Results window

The button "Show more" opens a window showing all the plagiarized sentences. The user can navigate through the plagiarized sentences using the buttons "< Previous" and "Next >". This window allows him to compare the sentences from the document to the sentences found online.



Sentence 2 on 5

TUAS.doc	Internet source
We offer education that develops working life and entrepreneurship, research and development services (R&D) and holistic development of organisations	we offer education that develops working life and entrepreneurship, research and development services (r&d) and holistic development of organisations

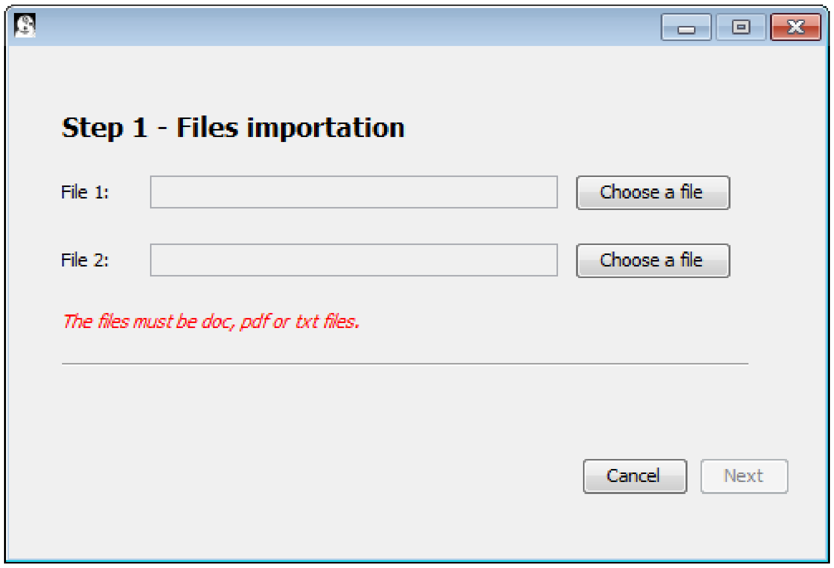
Figure 13 – Plagiarized sentences list

6.3 File to file comparison

The file to file comparison works roughly from the same way as the online plagiarism check from a user perspective.

Importation window

First of all, the user should import two files (MS word document, pdf or text).

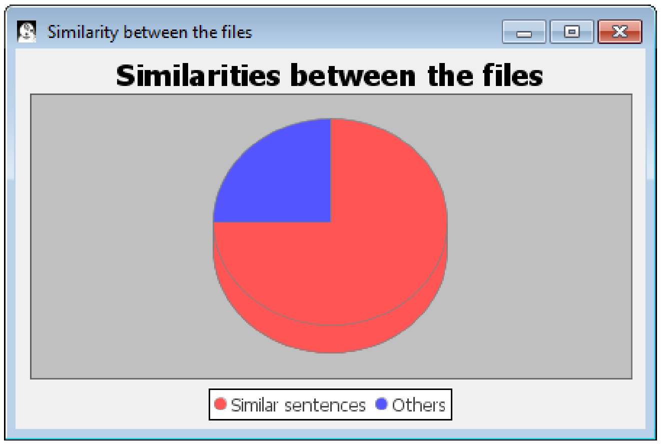


The files must be doc, pdf or txt files.

Figure 14 – Importation window (file to file comparison)

Loading window and result window

The comparison starts and a loading bar is displayed. The loading should be much faster than an online plagiarism check (less than five seconds), even for large files. The results window shows how many sentences are shared between the files and it is possible to display a pie chart, or the similar sentences exactly the same way as the result window from the online plagiarism check.



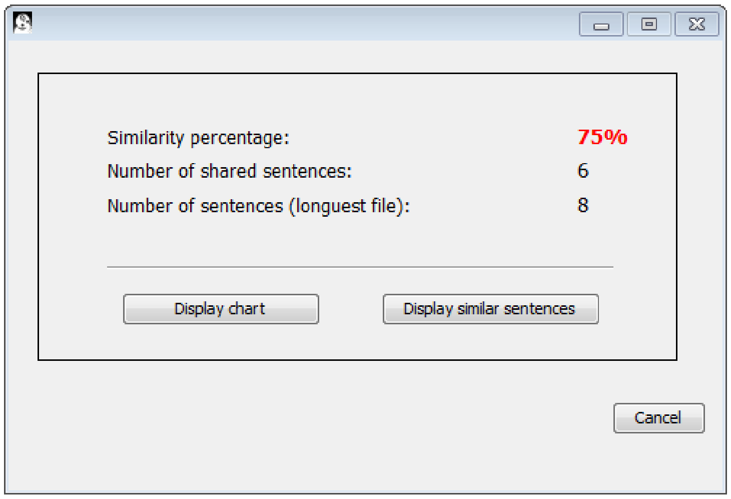

Similarity percentage: 75%
Number of shared sentences: 6
Number of sentences (longuest file): 8

Figure 15 – Results window (file to file comparison)

6.4 Possible enhancements

A plagiarism detection software can always be improved, and its development will never be over. There is thousands of possible improvement for this kind of software. However, there are ideas that could be developed concerning this specific program:

- Detection of the writing style of the author. The software could analyze the writing style and determines when it changes.
- Support for more file formats (docx, html …).
- Better user interface with more parameters and an advanced configuration.
- More accurate analysis and deletion of the false-positives.

Conclusion

As I explained in the previous chapter, the development of a plagiarism detection software is never completely done. There are so many remaining features to program to improve the detection (using databases for instance), that I don't consider this project as being totally finished. However, the program has been conceived from the perspective to continue the development thereafter: all the classes and functions can be reused and improved easily. One of the biggest parts of the project was also to deal with the internet errors and parsing errors without any crash or bug. The program has been tested successfully on a lot of files to determine its limits and to push them off as far as possible.

From a personal perspective, I think this project is a very interesting way to improve its Object-oriented programming knowledge, and more precisely its Java development skills. The use of inheritance, the management of the relations between all the classes and the use of the Model–View–Controller design were important factors regarding the achievement this project.

Programming a plagiarism detection software offers the possibility to really think deeply about the perception of plagiarism. It is not an easy task to be able to determine if a document or even a sentence has been plagiarized or not, because the notion of plagiarism itself is blurred and subjective. In fact, there is no specific rule about plagiarism, and this is why there are a lot of scandals related to plagiarism nowadays.

Table of figures

Bibliography

Oracle Technology Network. (2010). Retrieved from http://docs.oracle.com/

PDFBox API. (2010). Retrieved from http://pdfbox.apache.org

Jsoup API. (2011). Retrieved from http://jsoup.org/apidocs/

JFreeChart API. (2012). Retrieved from http://www.jfree.org/jfreechart/api/

Plagiarism.org. (2012). Retrieved from http://www.plagiarism.org

Poi API. (2012). Retrieved from http://poi.apache.org/

Wikipedia. (2012). Retrieved from http://en.wikipedia.org/wiki/