|  |  |  |  |
| --- | --- | --- | --- |
|  |  | C:\Documents and Settings\ParryA\Desktop\New School Logo.JPGC:\Documents and Settings\ParryA\Desktop\New School Logo.JPG | |
|  |  |  | |
|  |  |  | |
|  |  |  | |
| **Academic Year 2016/17** | | | |
|  |  |  | |
|  |  |  | |
| **ENG554 Pervasive Network Applications** | | | |
|  |  |  | |
| **Project-Based Learning Exercise** | | | |
|  |  |  | |
| **Final Coursework Report** |  | |
|  |  |  | |
|  |  |  | |
| **PBL team**: | Loopa | | |
| **Student numbers:** | 782716 | | |



Table of Contents

[1. Intro 3](#_Toc481604052)

[2. Design and implementation of the search engine 4](#_Toc481604053)

[2.1 Overall design 4](#_Toc481604054)

[2.2 Graphic user interface 5](#_Toc481604055)

[2.2.1 Graphic design 5](#_Toc481604056)

[1.1.1 Arrangement 7](#_Toc481604057)

[1.1.2 Programming 7](#_Toc481604058)

[2.3 Web crawler 8](#_Toc481604059)

[2.3.1 Initial design 8](#_Toc481604060)

[2.3.2 Jsoup 8](#_Toc481604061)

[2.3.3 Multi-threading 11](#_Toc481604062)

[2.4 Inverted File Builder 13](#_Toc481604063)

[2.5 Searcher 15](#_Toc481604064)

[3. Testing 16](#_Toc481604065)

[4. Evaluation 23](#_Toc481604066)

[4.1 Flexibility 23](#_Toc481604067)

[4.2 Possible extensions 23](#_Toc481604068)

[5. Appendix 24](#_Toc481604069)

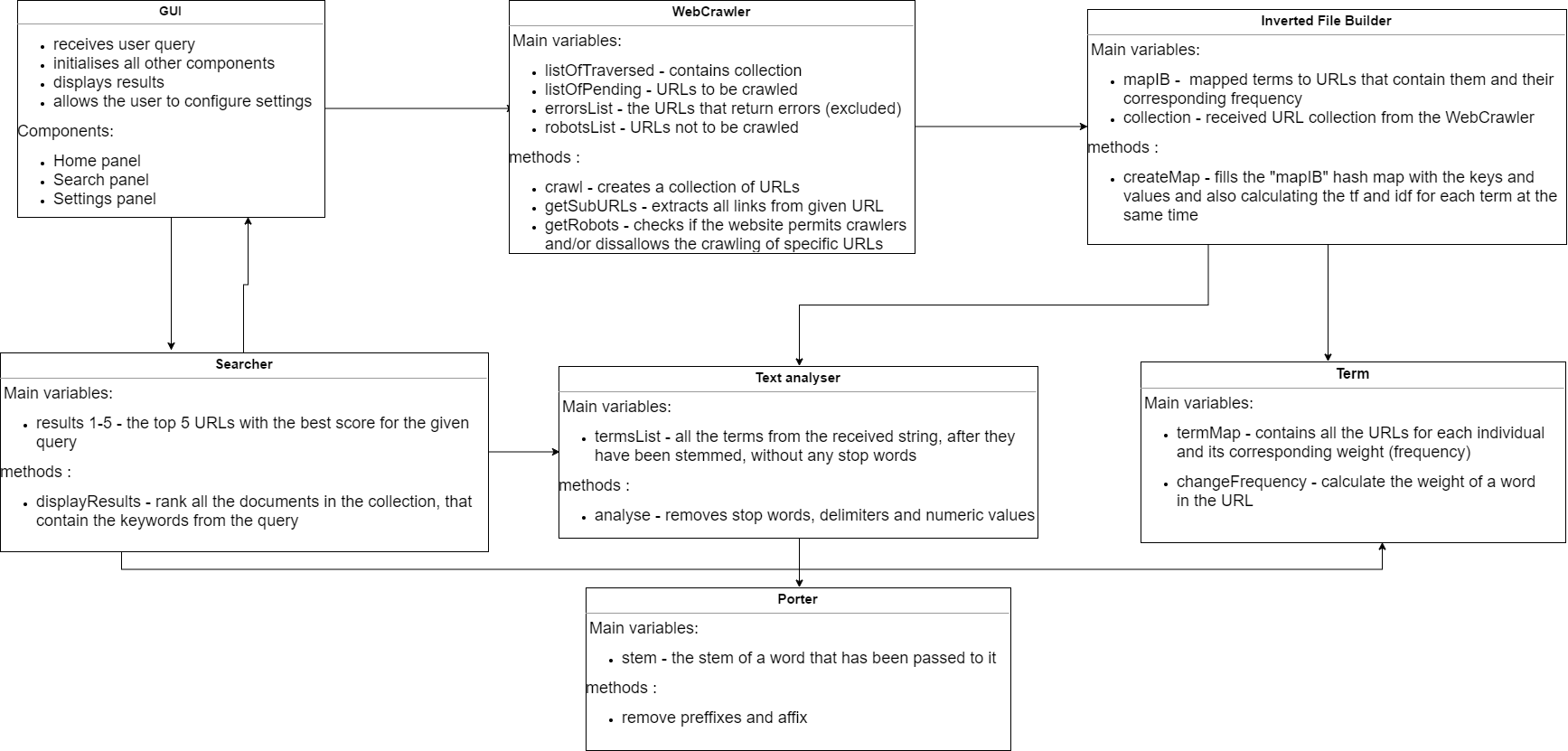
# Intro

Loopa is an easy to use search engine with a clean and friendly interface, which allows you to input a search query and is going to return the most relevant to it results from the already built collection. It uses a very simple ranking algorithm, based on normalized frequency and “tf.idf” weighting scheme to return only the most relevant documents.

Although Loopa is limited to its collection size, it strives to provide you with the most relevant results by giving you the choice to set its settings manually. However, if the current collection is not relevant to your query, Loopa would also let you know.

# Design and implementation of the search engine

## Overall design



## Text Analyser

### Structure of the TA class

The role of the text analyser class is to process a long string of text and from it create a list of terms. It does that in three main processes:

* **Tokenizing –** when creating a new Text Analyser object, the constructor takes a String and fills the list of terms. In order to do that all the delimiters in the string are being replaced with white spaces. Loopa uses the standard delimiters that were suggested plus a few others to make sure that no delimiters are encountered when creating the File containing the Inverted Builder information (which will later be explained in more details). An important step is to make all the characters in the string lowercase, otherwise the SE won’t recognize that for example “stop” and “Stop” are the same word.
* **Removing stop words -** The string is then split into an array of separate terms, but not all of them do not carry any meaning, for example, most of the stop words such as “a” , “and”, “or”. Loopa gives the user a choice – if the option is selected, all stop words are removed, which reduces the size of the inverted file and increases performance. Otherwise, Loopa removes only the most common stop words (“a , and , are , be , if , in , is , it , of , on , or , so , the , they , there , this , which , why”) that carry no meaning and do not reduce the precision of the query. To create our own list of stop words, the following website was used, which contains a few lists that include a different number of stop words: <http://www.ranks.nl/stopwords> .
* **Stemming**: Another step towards improving the performance of the inverted file builder is to stem words – usually, words that have the same stem are considered to have the same meaning. Loopa uses the Porter stemming algorithm to remove prefixes and affixes from words. The downside to stemming is that in some cases the precision of the SE may be lowered as some words have the same stem, but different meaning. However, that doesn’t happen often, so stemming is included in the Loopa text analyser.

### Evolution of the TA

Like most components in Loopa, the text analyser went through a couple of changes and was improved a few times. Originally, the TA was planned to only be a method ready to be invoked anywhere in the program. However, this was changed to the TA being a separate class in Loopa, creating a TA object, which contains a list of terms, ready to be used by either the Inverted File Builder or the Searcher. It is important to give the user the option to choose whether stop words are included in the inverted file or not, so in the GUI a check box is provided in the Advanced Settings Panel for the user to click on.

## Graphic user interface

Designing the user interface and connecting it to the functions of Loopa, was the most time-consuming task of all. The Loopa user interface is very clean, neutral and easy to use. The three main tasks of making the GUI are described below.

### Graphic design

The task of designing the graphical design of Loopa would have been impossible without the NetBeans IDE, which is an excellent tool, that provides and easy-to-use interface of drag-and-drop design.

However, it is limited to some extent, since the fonts, buttons and such are very basic. This led us to designing all the icons, buttons and label using another tools. “Logomakr” is a wonderful free website, which allows the user to design graphics, by providing a large collection of images and fonts. Originally, only the Loopa logo was designed using the website, but then to keep the design consistent all the graphics were also designed through the websites. Attached below are previous versions of Loopa’s graphic interface.

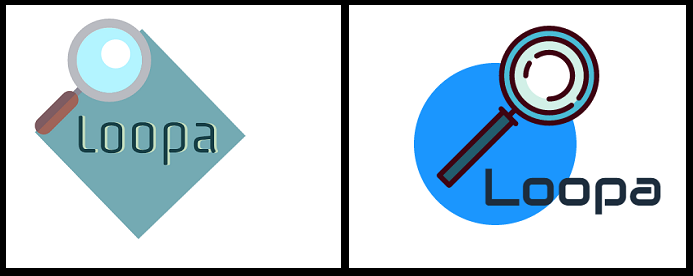


Fig. 1 Loopa’s first logo ideas

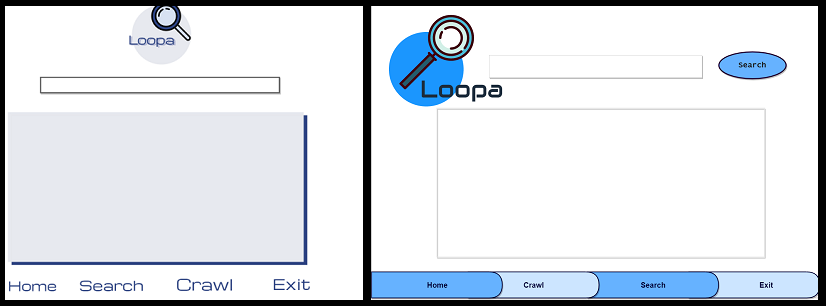


Fig. 2 Loopa’s initial design ideas

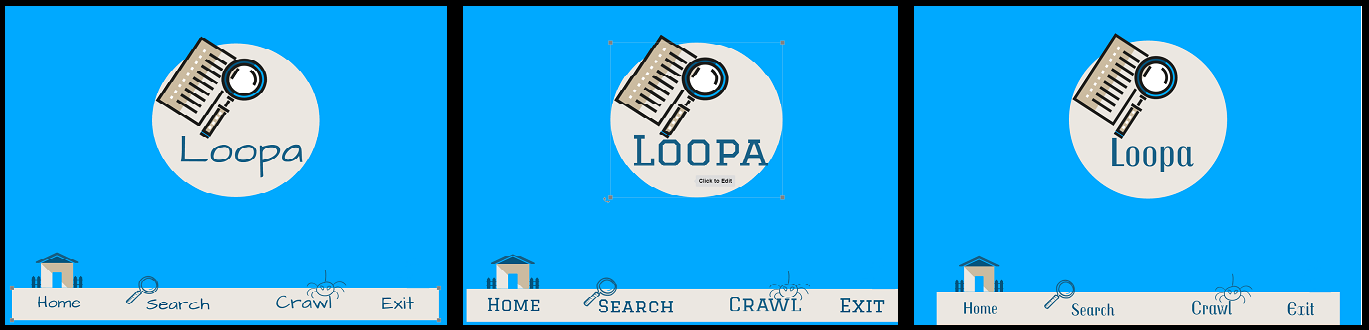


Fig. 3 More design ideas

Fig. 4 Final design

### Arrangement

After designing the icons for all the buttons and headings, components had to be added and arranged inside the JFrame. This is where NetBeans showed a little bit of a .., since moving one little component would sometimes shift and resize the whole Frame and make a chaos out of everything. However, with a little bit of patience, all the components were finally put in their place resulting in the design showed below.

### Programming

A YouTube tutorial was used as a starting point to achieve Loopa’s dynamic design. This code allows us to switch between panels at the press of a button and was used in both creating the main panels as well as changing the look inside the panels.



The rest of the code was automatically generated by NetBeans, all that was left to do it connect the methods to the ActionListeners.

## Web crawler

Loopa has originally got two types of web crawlers – single-threaded and multi-threaded. The one that was put in the final design was singe-threaded and serves for creating a small collection of URLs, which the InvertedFileBuilder class indexes and creates the inverted file.

### Initial design

As a starting point, originally the provided web crawler was used, with some additions.

Similarly to the text analyser, the web crawler used to be a method in Loopa, but was later created as a separate class. A “robots” method had to be added to make sure that our web crawler does not visit URLs that forbid indexing. An additional error’s list was also added, so that the final collection of websites does not include web pages that do not work.

In general, all versions of Loopa’s web crawler use the following algorithm:

**Pseudo code:**

*Start*

*Get starting URL*

*Visit “robots.txt”*

*Fill robots list*

*If URL allows web crawlers*

*Connect to starting URL*

*Add to collection*

*Extract sub URLs*

*If sub URL isn’t in the robots list  
 add to list of pending URLs*

*While the collection is smaller than 100*

*Extract sub URLs*

*End*

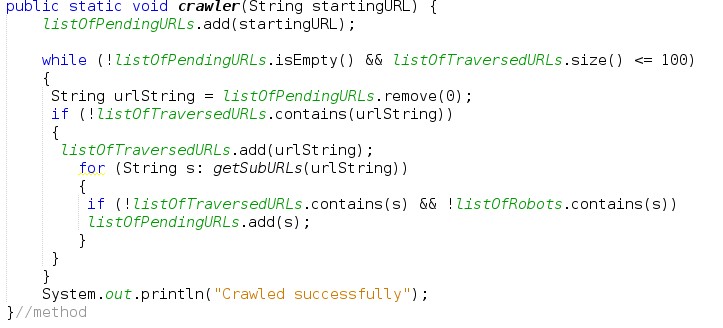
### Jsoup

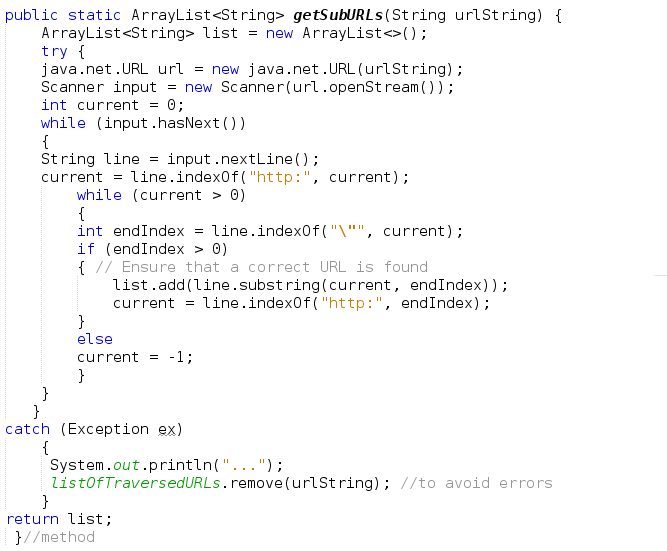
Jsoup is a Java library for working with real-world HTML. It provides a very convenient API for extracting and manipulating data, using the best of DOM, CSS, and jquery-like methods.

Jsoup was used to scrape and parse HTML from a URL, in order to implement the web crawler and the “robots” method. There are many advantages to using this specific library – it reduces the code significantly, give us better performance and you have the option to only choose the absolute URLs which excludes links that lead to images, as we haven’t used any in our collection. It was also useful for our text analyser, since it allows you to select the text from the page you connect to.

The difference in the code is evident from the pictures below:

*First version:*





*New version:*

### Multi-threading

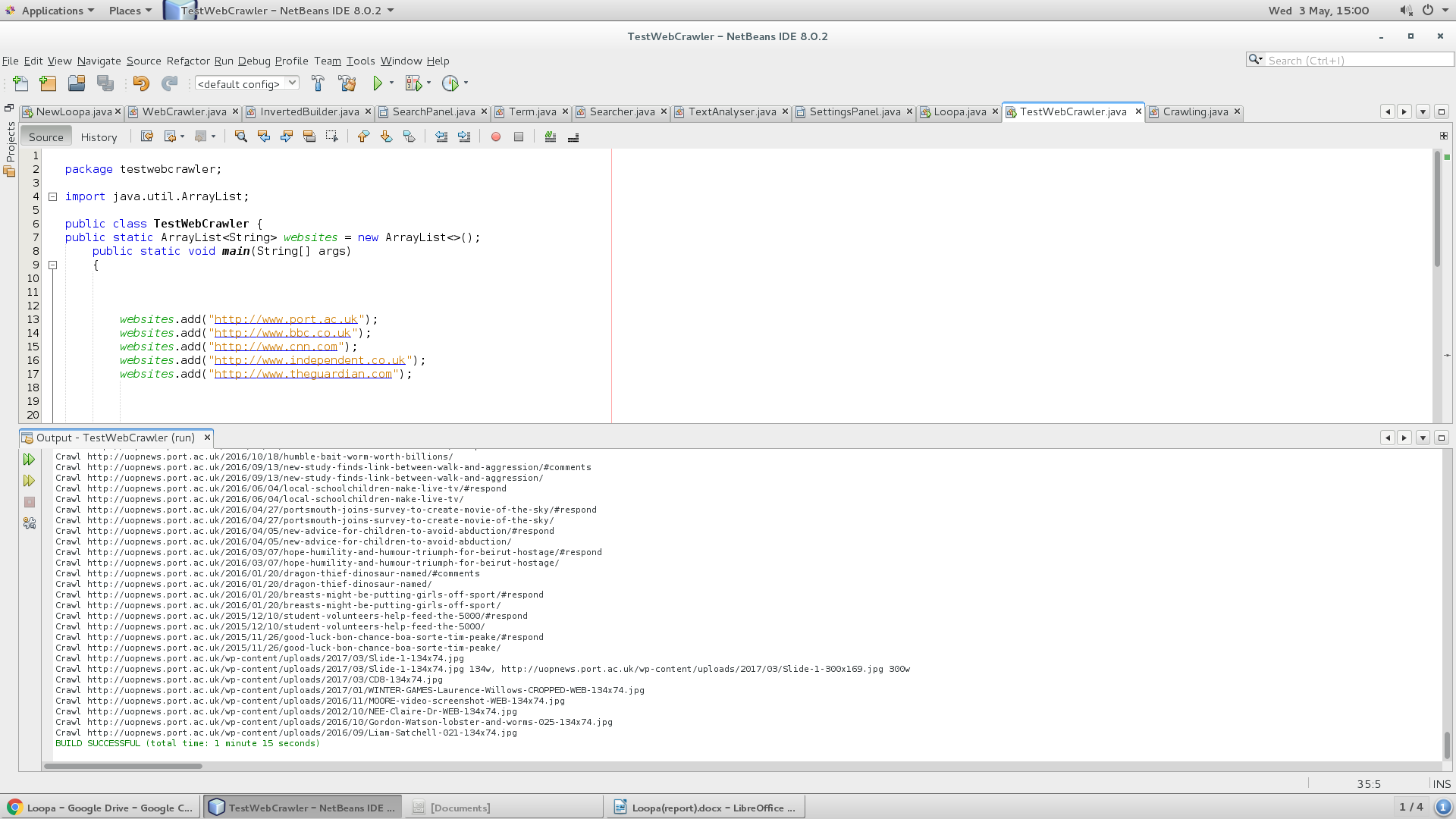
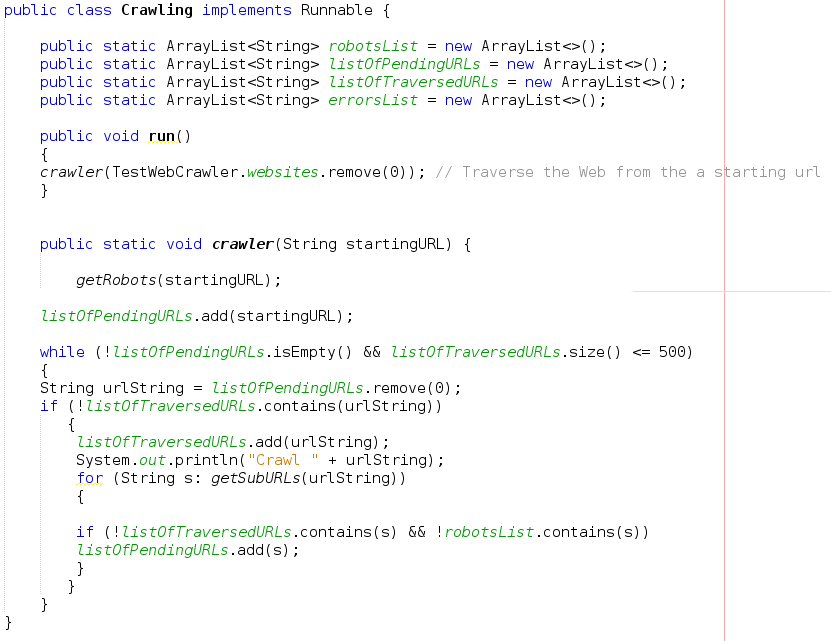
To further improve the performance of Loopa, a multi-threaded web crawler was developed. Unfortunately, when testing the web crawler and GUI together, the threads from the web crawlers prevented further access to the GUI.

When creating the multi-threaded web crawler , we used very similar methods, except for starting from different URL for each thread. However, only one queue exists that provides further URLs for the web crawler to crawl.

There was another problem with the algorithm – if only one starting URL was provided, instead of 4, the speed of the crawler slows down considerably, because it would sometimes take a long time to fill the queue (listOfPending) with more URLs. Another downside was, that in order to control how many URLs to crawl from each starting URL, the queues had to be separated, which also slows down the performance.

The following code was used, which gave us good results (500 web pages crawled in just above a minute).



*Testing the web crawler:*

## Inverted File Builder

The aim of the Inverted File Builder is to create the inverted index from the collection passed from the web crawler, that would contain word/posting pairs. Originally, the simplest inverted file was created just as a method in Loopa and it would only contain the frequency of the word in the given document. It follows the already provided algorithm:

*Create an empty index term list I;*

*For each document D in the document set*

*{*

*For each term T in D*

*{*

*If (T is not already in I)*

*{*

*Find the location for T in I;*

*insert T into I;*

*}*

*If ((T, D) is in the posting list for T)*

*increase its term frequency for T;*

*Else*

*{*

*create (T, D);*

*add it to the posting list for T;*

*}*

*}*

*}*

*Original code:*

However, the array lists prove to be an unreliable structure in this case, so they have been replaced with a second hash map in the new version of the Inverted File Builder. Not only that, but also the “tf.idf” weighing scheme was added, to make the job of the searcher easier.

When inserting the term into the map, instead of saving it’s frequency, a normalized frequency is calculated in the following way:

***Normalized frequency = number of occurances of the term / total words in the URL***

***IDF = log (Total number of URLs in collection / number of documents in which the term occurs)***



**Term** is an object that is being created every time a new term is found in the collection. It contains a single map, that has URL/frequency pairs mapped in it. It allows for easy referral to each term’s posting list.

## Searcher

The Searcher component uses the information from the Inverted File to answer queries provided by the user interface. Once the user enters a query, it is being processed by the text analyser and then the best matches are found

Loopa follows the already provided algorithm for this component as well:

“*For each term in the query, the posting list is requested from the Inverted File Builder. For each document term that matches the query term, the document id is retrieved and stored for later use. Based on the ranking algorithm, you can compute the score of all the relevant documents and rank them.* “

The searcher was the component that for me was the most difficult one, so it is a very simple one. Since the GUI took so long to make, very little time was left for the searcher. However, the results are still very accurate, since the frequency of the terms is normalized and the idf scheme was also included.

The weight of each term in the document is already calculated by the “tf.idf” scheme. What the searcher does is, add the weights of all the terms that are both in the query and the document. Then it find the biggest 5 values and displays them in the GUI.

If the searcher does not find at least 5 websites in the collection that have the terms in them, it assumes that the collection does not contain relevant results to the user's query.

The ranking was made in the following way :

*get all the URLs mapped in the inverted file that contain the keywords*

*for each URL*

*if it contains single word from search query*

*add to results map with the weight of the word*

*if it contains more than one*

*add weight of first word to the weight of the second word*

*// do the same for more*

*get all values from the results map  
sort them to find the highest 5*

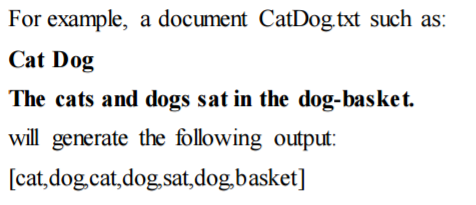
*find the keys for the corresponding values*

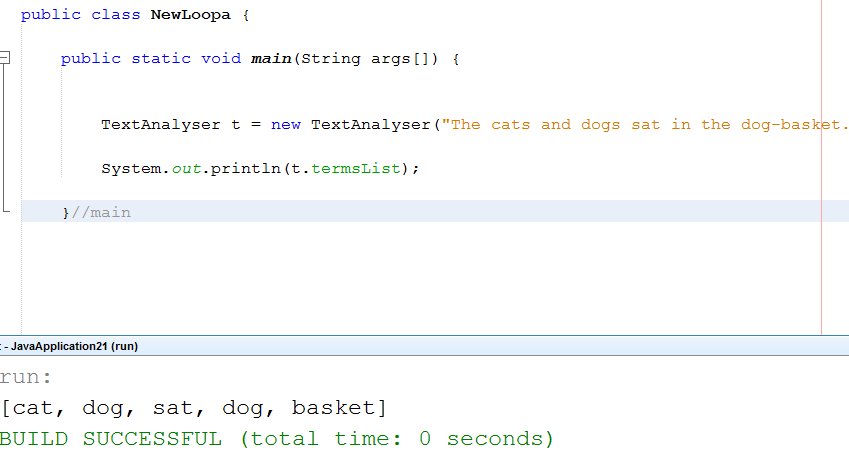
*display results in GUI*

# Testing

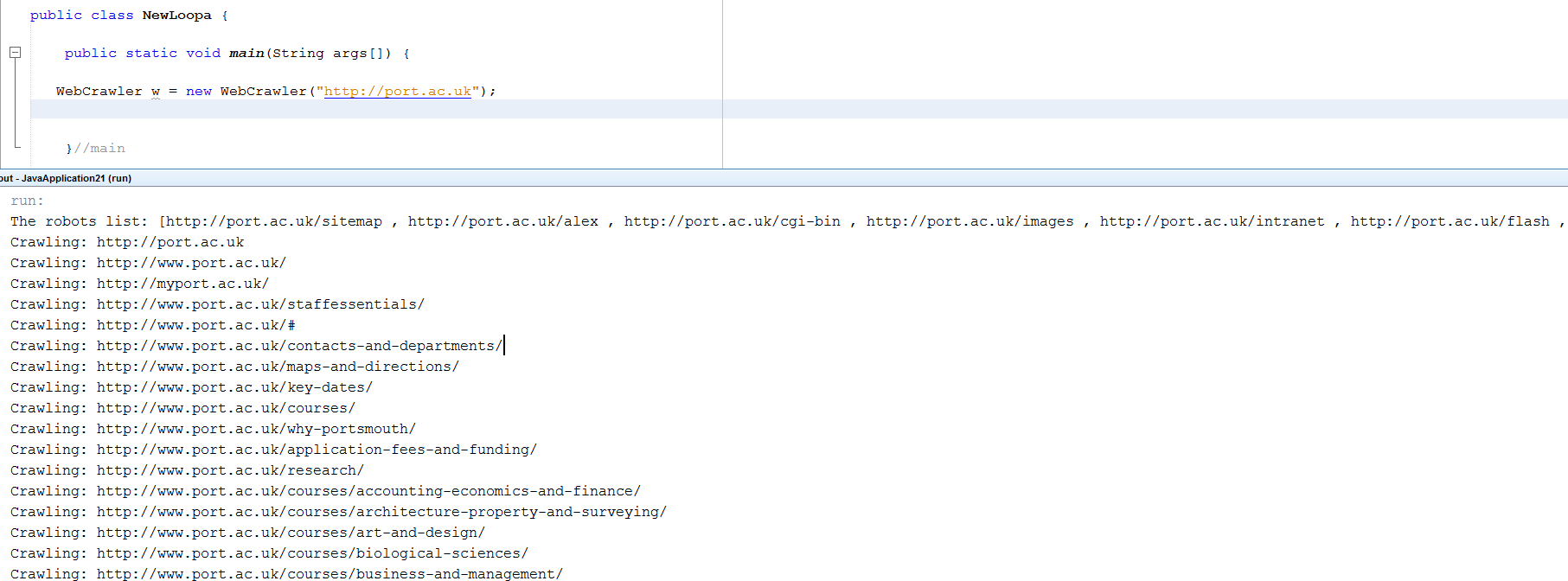
Loopa was thoroughly tested in two ways – individual components both one by one and as a part of one single program.

***Text Analyser:***

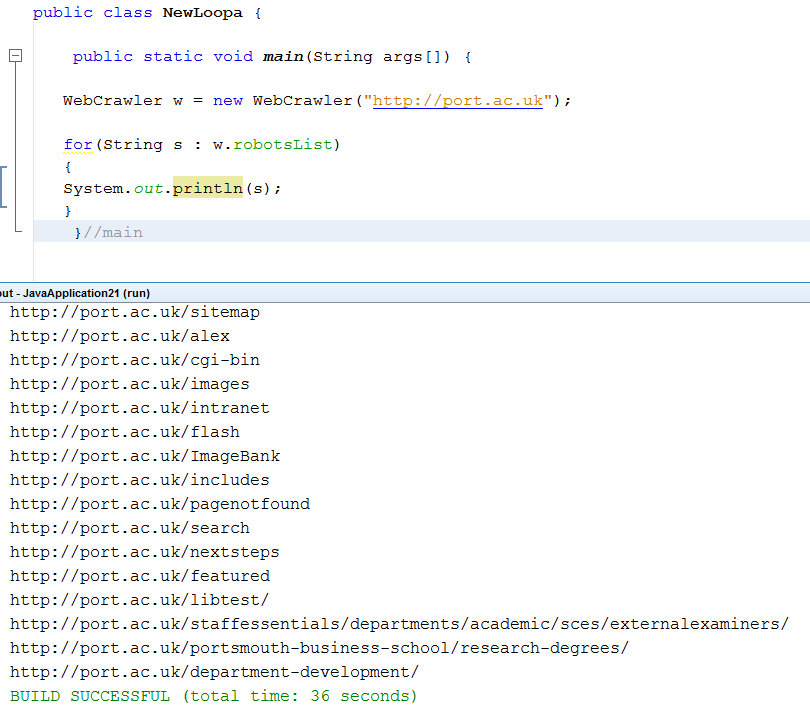


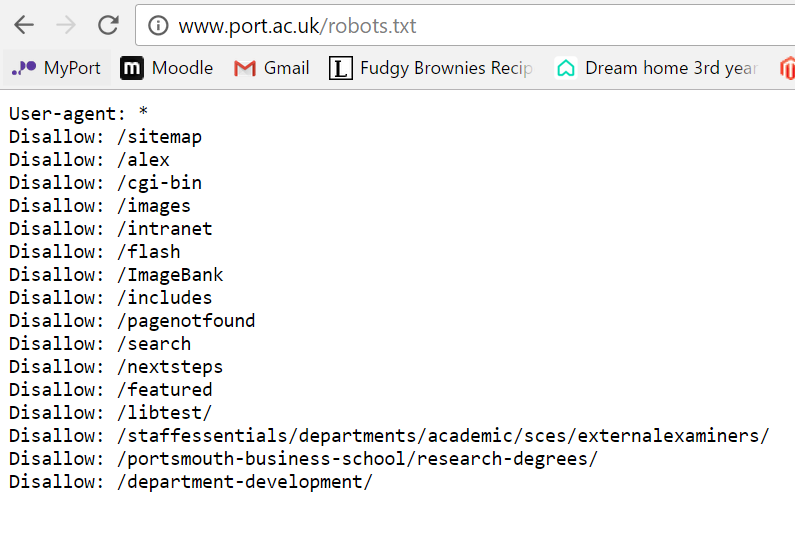


***WebCrawler:***

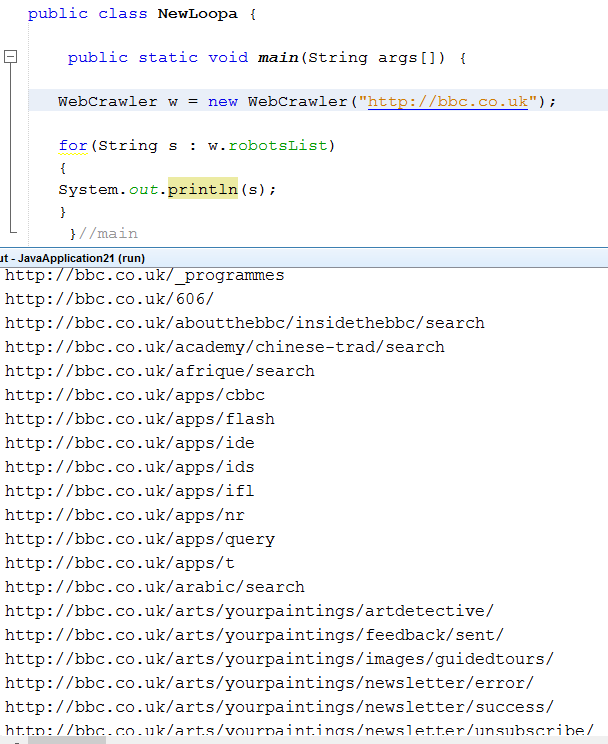


***getRobots method:*** e.g. Portsmouth university website





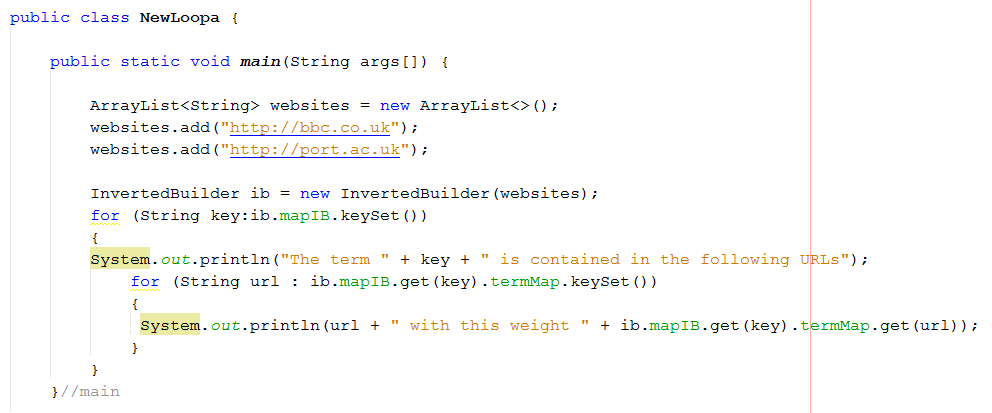
BBC robots:

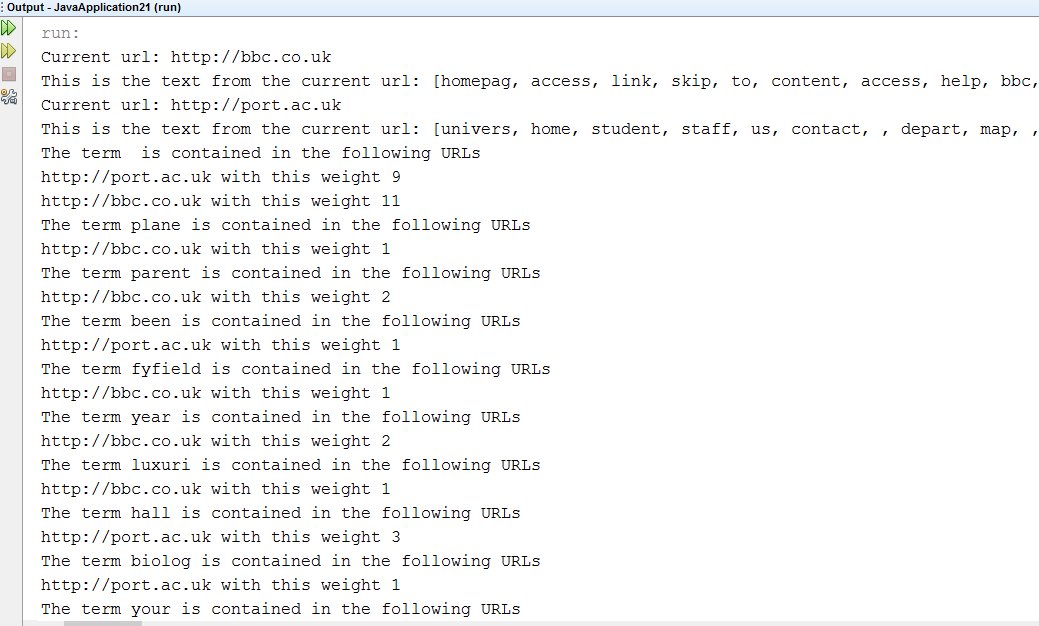
 

***Note:*** *Even thought BBC’s robots protocol file has multiple crawlers that are forbiden, Loopa’s algorithm find which one applies to it.*

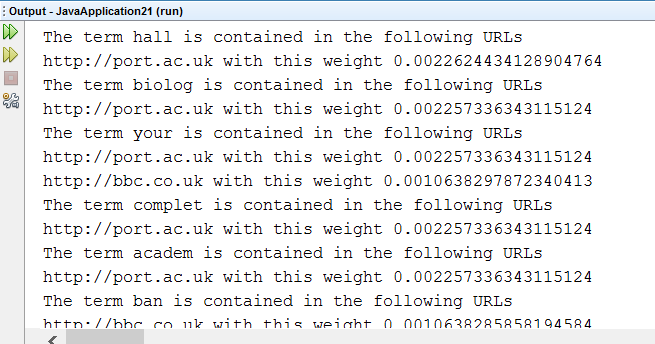
***Inverted File Builder:***

*First version without TF IDF scheme:*



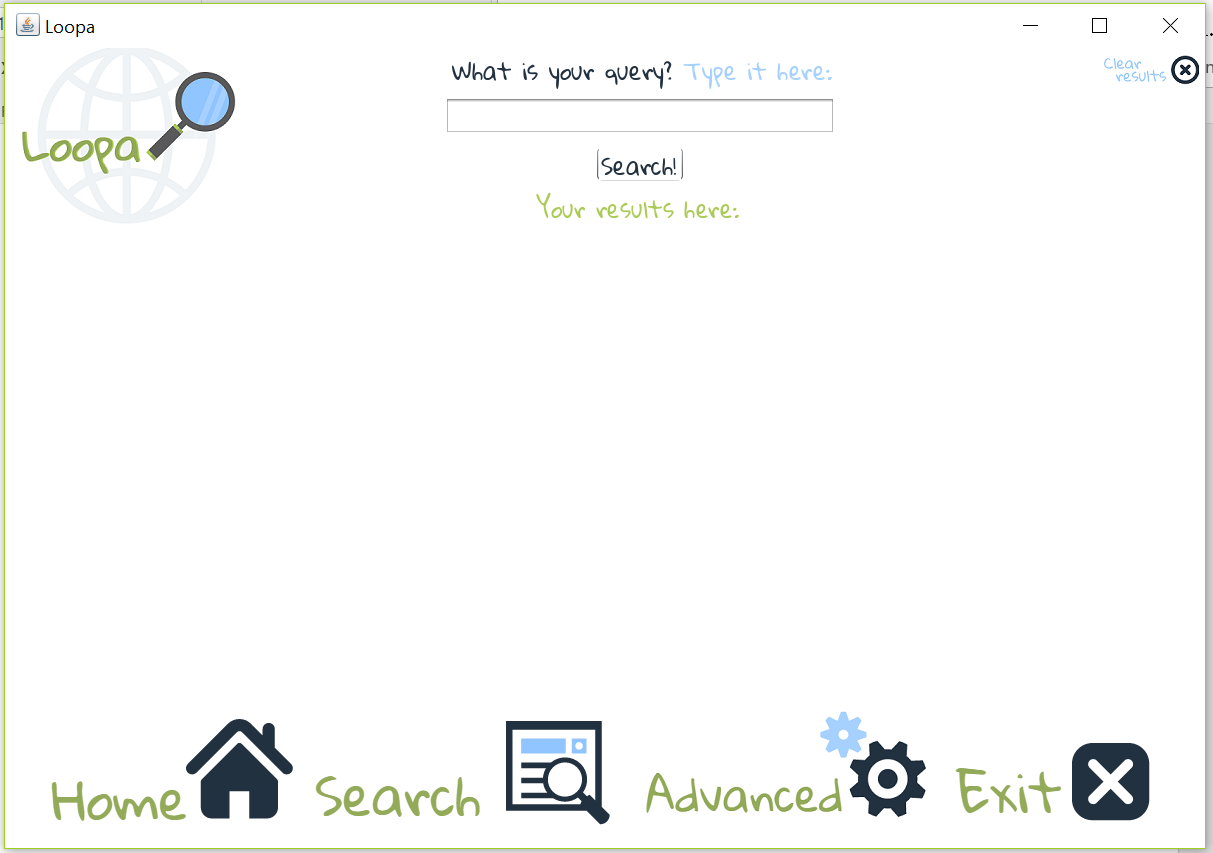


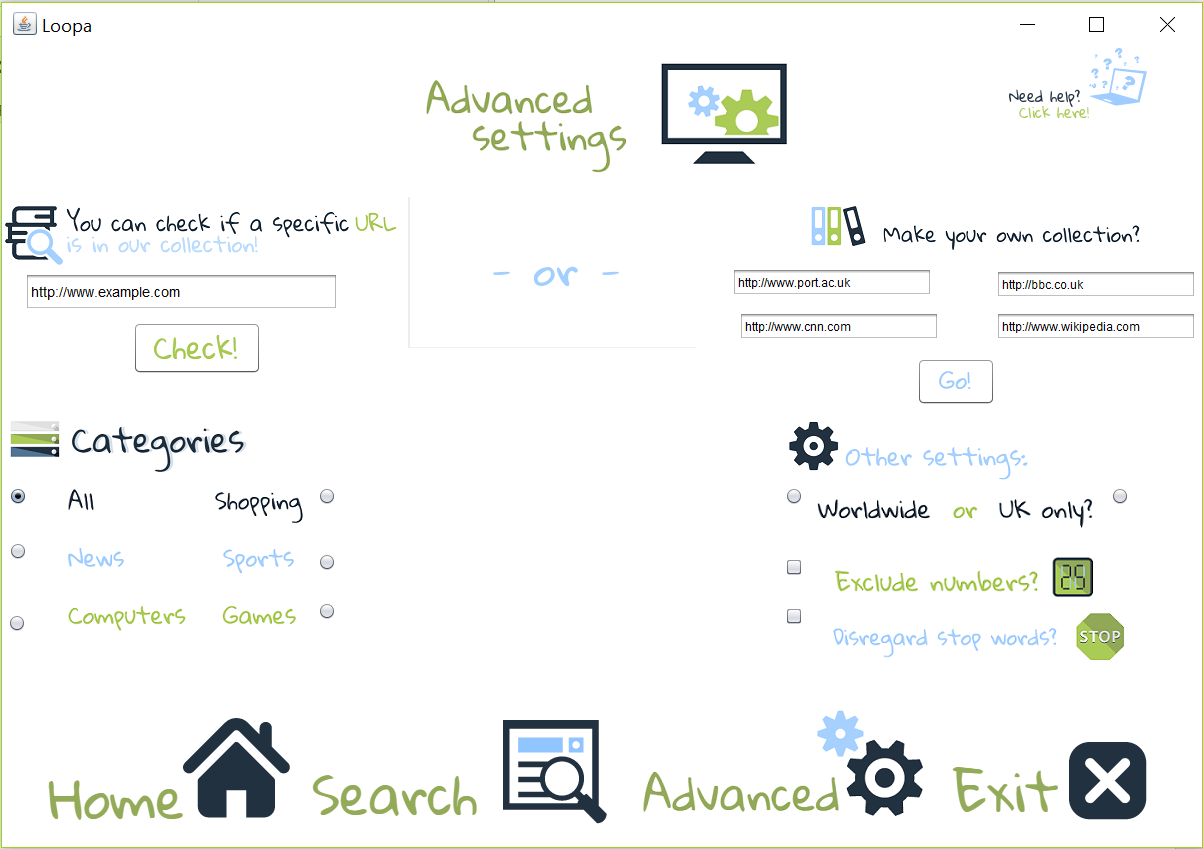
*After TF IDF:*

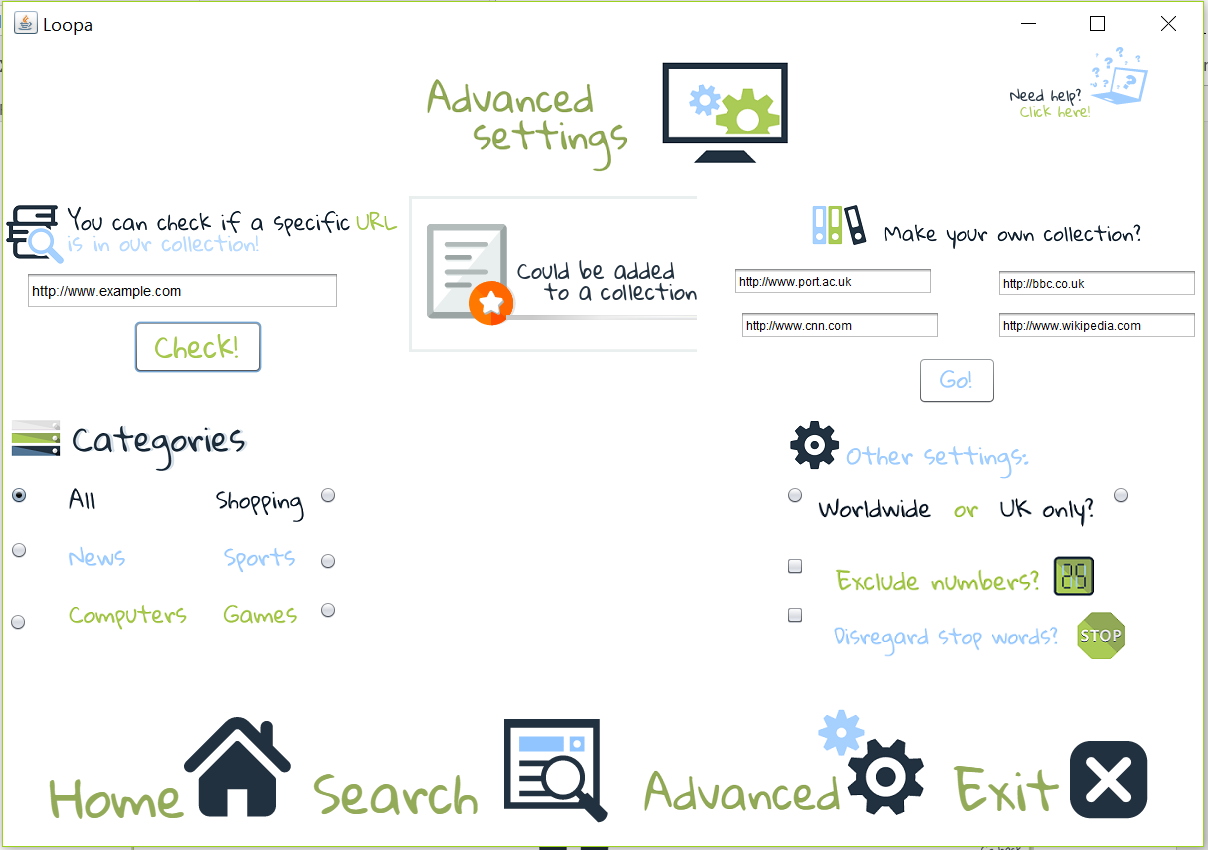


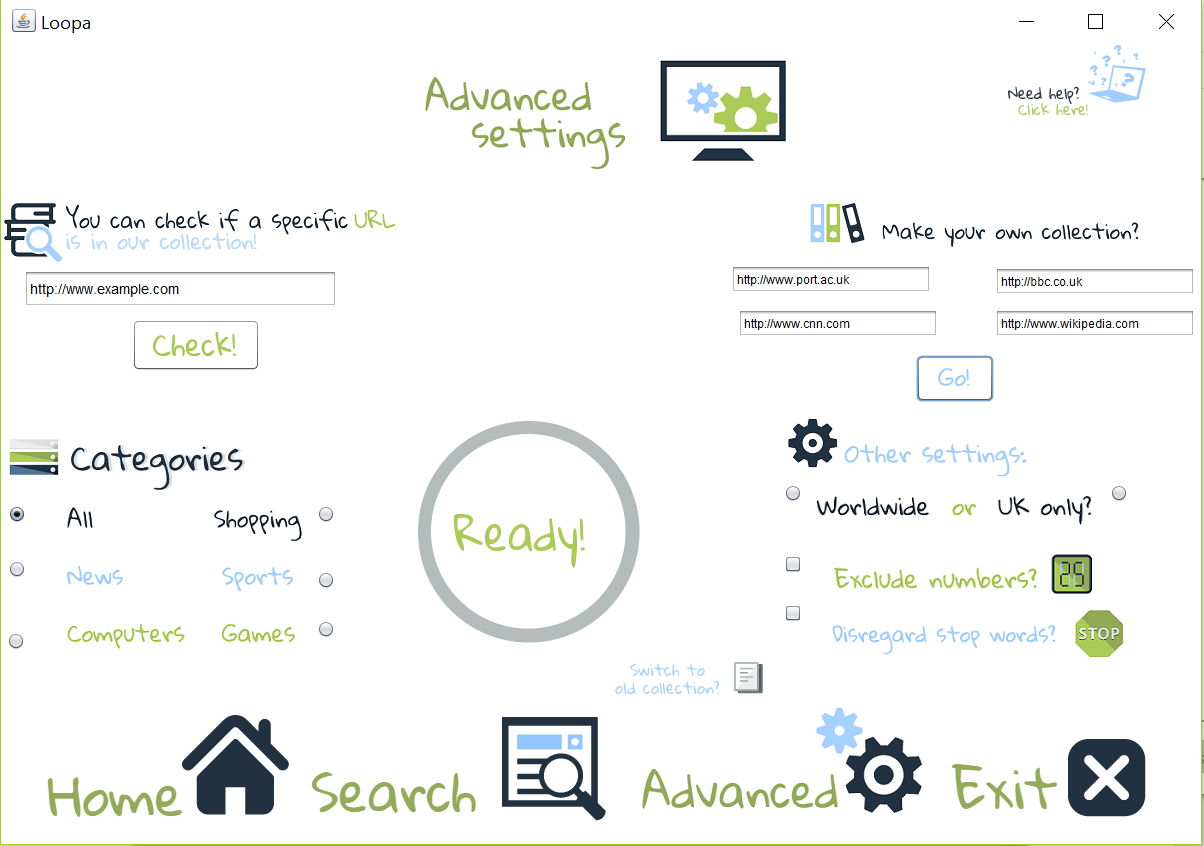
***GUI***



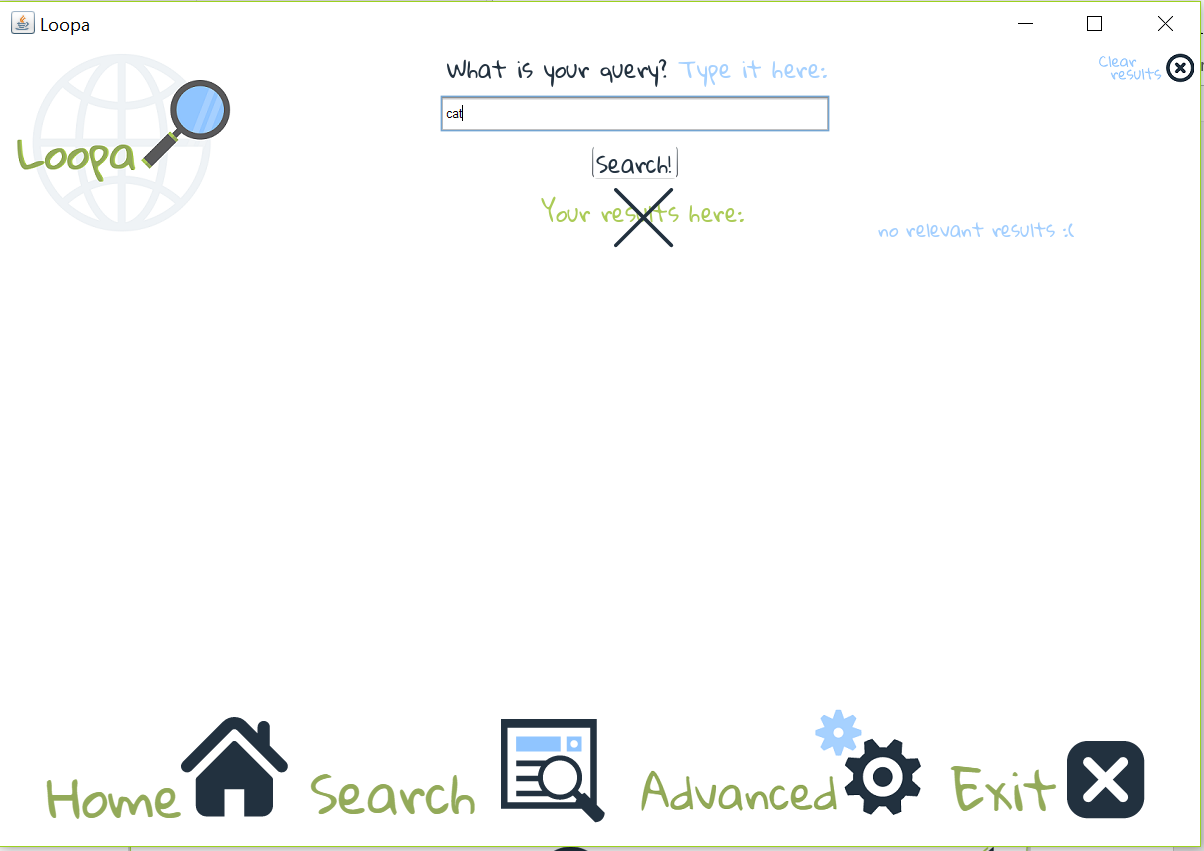












# Evaluation

## Flexibility Loopa is quite flexible when it comes to giving choice to the user, as he can decide whether he wants to include stop words or numbers. However, some of the features that didn’t make it to the final version, would have given the user even greater flexibility. For its current state Loopa chooses not process any different types of formats.

## Possible extensions

Possible extensions and improvements have already got a place in Loopa and have prototypes. Here are some of them:

* Loopa was originally intended to store the inverted file in a text file (as you can see from the included “FileWrite” class ), which would save the user time – instead of crawling the web every time, the searcher would use the inverted text file as its source.
* Different collections – creating one file wouldn’t be enough – with a little but more work, Loopa could easily create (as intended) different files from different collections, so that the user can choose which one his query fits the most.
* Multi-threaded crawler – although it was implemented, further tests were needed in order to combine the multithreaded web crawler with the GUI.
* Last thing included in the interface, but not in the program was being able to see more than 5 results. Due to the lack of remaining time, Loopa was simplified (and also limited) to showing only 5. However, arrows are added below the results buttons to navigate between pages, which would work with a few minor changes.

# Appendix

If you are trying to run Loopa from the source code, simply run the “NewLoopa” class. If you do that you'd be able to also see how the components work, since it displays a message after a method has been run successfully. Please, ensure that the Jsoup library is added – sometimes the compiler leaves it out of the libraries for the project.

There are instructions on how to work with Loopa inside the SE itself, unfortunately, not all the functionality of Loopa, was connected to the interface. However, you can see the implementation of those classes and methods inside the report.

When you search or create new collection, allow about a minute (depending on the computer), in order for Loopa to initialize its components. Results are displayed as soon as they are available or an appropriate message, depending on whether the search was successful or not.

Recommended resolution is 1200x800 or more.