## Overview

For this practical, we were asked access the OMDb API to search for movies based on a title search which returned 10 movie ID's. With this list of IDs, further calls were to be made to retrieve the plots for each of these movies. After processed the text of the plots using normalisation and stop-word removal, a combined word-frequency histogram was to be created and the top 10 most frequent words needed to be printed.

I successfully completed all parts of the specification in a relatively efficient way, despite some potential issues along the way with the choice of data-types. I also rigorously tested my code so that for each potential exception or error, a clear error message is displayed to the user.

Given more time, it would have been interesting to have created graphical representations for the word-frequencies and to look at using different stop-word lists and comparing their effectiveness in gaining useful data.

## Design

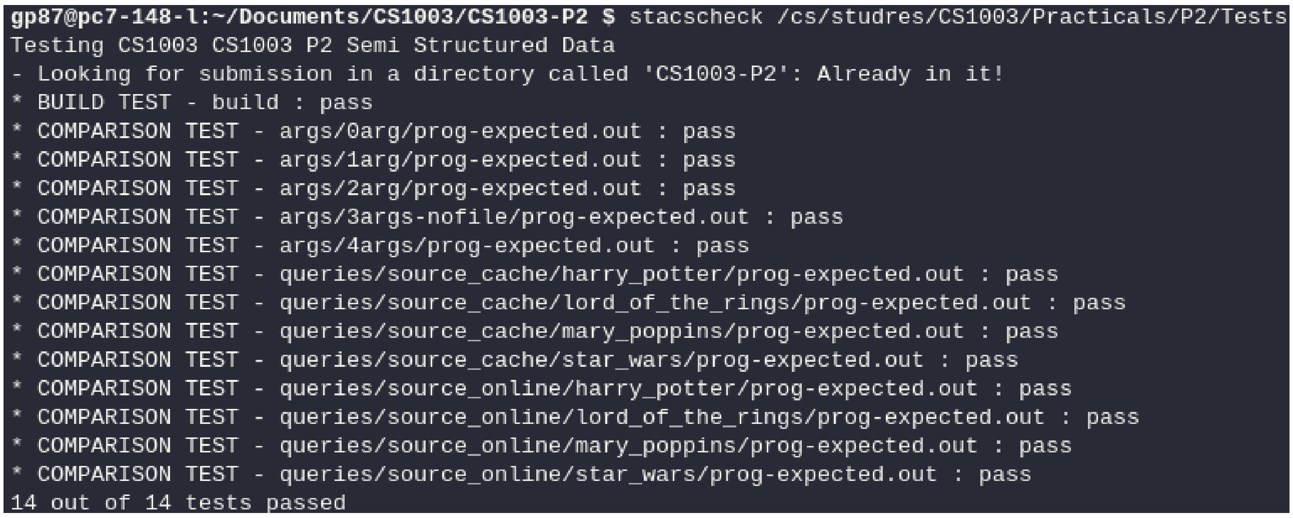
In the OMDbAPI class, I decided to create two private methods for calling the online API and cache with the generic request as the parameters as this prevents repetition for when the online/cache APIs are called for getting the list of movies and to get the details of each movie.
This also means that in the future, the application could be extended for further searching for movies using different parameters instead of / as well as the title search, which the API allows.

For exception handling, I decided that the OMDbAPI functions would not deal with any of the fatal exceptions in the program and instead throw them back to the main method. The reason for this is because returning from the OMDbAPI methods will not safely end the main method function.
With exceptions with reading the stop words, calling the API or cache, and invalid source variables, further functionality requires these functions to run successfully first and so the program needs to end if these exceptions are thrown.

For the stop-words list, I decided to use a HashSet because the stop-words are used to search for and remove the stop-words from the plot strings. Sets only allow for a word to exist once and so if the stop-words file contained the same word multiple times then it wouldn't be added again to the set, thus preventing the program for searching for the same string twice and increasing efficiency.

Also, using an ArrayList for the getMovies() method return value gives the movie IDs in the order to which they appear in the JSON response. On reflection, I would use a LinkedHashSet in future, which retains the order in which the movies would be added but doesn't allow duplicates in case the API returns repeated elements (this is unlikely but still possible).

## Testing

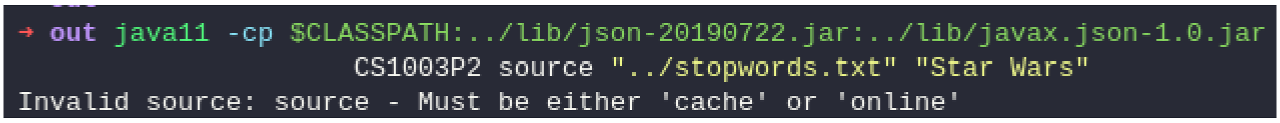


```
gp87@pc7-148-l:~/Documents/CS1003/CS1003-P2 $ stacscheck /cs/studres/CS1003/Practicals/P2/Tests
Testing CS1003 CS1003 P2 Semi Structured Data
- Looking for submission in a directory called 'CS1003-P2': Already in it!
* BUILD TEST - build : pass
* COMPARISON TEST - args/0arg/prog-expected.out : pass
* COMPARISON TEST - args/1arg/prog-expected.out : pass
* COMPARISON TEST - args/2arg/prog-expected.out : pass
* COMPARISON TEST - args/3args-nofile/prog-expected.out : pass
* COMPARISON TEST - args/4args/prog-expected.out : pass
* COMPARISON TEST - queries/source_cache/harry_potter/prog-expected.out : pass
* COMPARISON TEST - queries/source_cache/lord_of_the_rings/prog-expected.out : pass
* COMPARISON TEST - queries/source_cache/mary_poppins/prog-expected.out : pass
* COMPARISON TEST - queries/source_cache/star_wars/prog-expected.out : pass
* COMPARISON TEST - queries/source_online/harry_potter/prog-expected.out : pass
* COMPARISON TEST - queries/source_online/lord_of_the_rings/prog-expected.out : pass
* COMPARISON TEST - queries/source_online/mary_poppins/prog-expected.out : pass
* COMPARISON TEST - queries/source_online/star_wars/prog-expected.out : pass
14 out of 14 tests passed
```

Figure 1: All stacscheck tests passed

Figure [1] shows that all the stacscheck tests passed. Tests 1-5 shows the application gives relevant error messages when there is incorrect input for the number of arguments and when the stop word file doesn't exist.

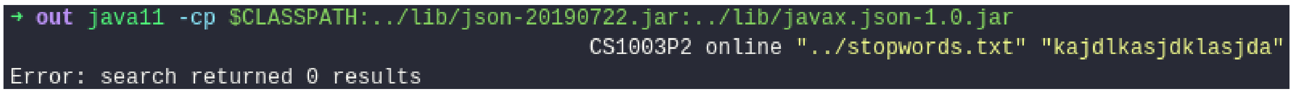
The rest of the tests show how the program works with the online and cache API sources and creates the correct word frequency histograms with the stop word filtering done.



```
➜ out java11 -cp $CLASSPATH:../lib/json-20190722.jar:../lib/javax.json-1.0.jar
                         CS1003P2 source "../stopwords.txt" "Star Wars"
Invalid source: source - Must be either 'cache' or 'online'
```

Figure 2: Output when source is invalid

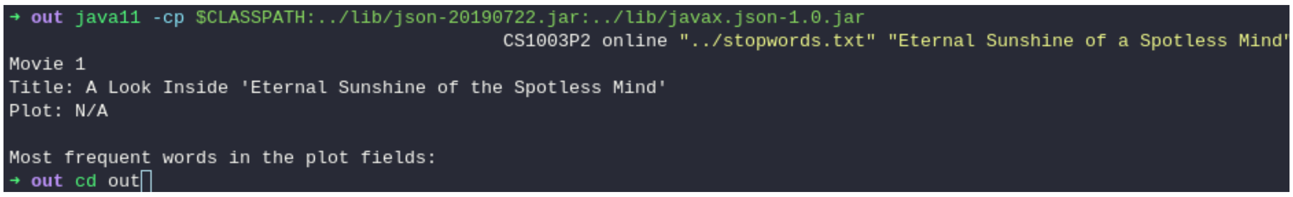
Figure [2] shows how the program handles an incorrect source argument, it tells the user that source must be either "cache" or "online" and doesn't attempt to call the API.



```
➜ out java11 -cp $CLASSPATH:../lib/json-20190722.jar:../lib/javax.json-1.0.jar
                                            CS1003P2 online "../stopwords.txt" "kajdlkasjdklasjda"
Error: search returned 0 results
```

Figure 3: Output when API title search returns 0 movies

Figure [3] shows how the program deals with an API title search which returns 0 results by printing a message telling the user and exits.



```
→ out java11 -cp $CLASSPATH:../lib/json-20190722.jar:../lib/javax.json-1.0.jar
                                                    CS1003P2 online "../stopwords.txt" "Eternal Sunshine of a Spotless Mind"
Movie 1
Title: A Look Inside 'Eternal Sunshine of the Spotless Mind'
Plot: N/A

Most frequent words in the plot fields:
→ out cd out
```

Figure 4: Output when only 1 result is returned from title search and the most frequent words list is smaller than 10 items

Figure [4] shows the program dealing with more than 0 but less than 10 returned movies from the API title search by just printing the movies returned. It also shows that when the most frequent words contains less than 10 items, it just prints out the one's it can in order and then stops.

## Evaluation

I believe my application was successful in achieving the aims from the specification. I created a robust application which returns the relevant word frequency histograms from a list of movie plots from a title search giving interesting information about how certain title strings can result in certain words occurring often in the movie plots.

I successfully added functionality for preparing an API request given a search string and encoding a URL string which then successfully calls an online API, before parsing the JSON response to get different pieces of information.

Also, I implemented file reading for the stop-word file and the cache API file and parsed their information successfully, whilst dealing with errors when the files don't exist.

Finally, I successfully added normalisation and stop-word removal for the plot strings of the movies to get a list of relevant words which could then be collated to create the relevant word frequency histogram.

## Conclusion

I am really pleased with the result. In the future, this could be extended by storing the word frequency data in a JSON file to be used further. I would have liked to add in the usage of searching for movies using different features, not just the title. The OMDbAPI allows searches using different properties of the film and so it would be interesting to see the different resulting word frequency histograms compared to title search.

With the word frequency data, it would be interesting to search for news articles relating to a specific movie because the article may possibly use the same sorts of words which appear in the word frequency histograms.