

## Overview:

The first project of the module required us to develop a program that could take a string prefix of a word and return a specified number of results (sorted by their weight) that started with that prefix from a file entered by the user. This is in essence auto completion. To begin with we were given an API skeleton with guidance on how to develop the program listed in the project specification. More specifically this project required us to implement code that could efficiently and correctly: read data into the program, store the data suitably, sort the data, quickly find the first and last occurrence of a prefix, learn and create a GUI. This project was definitely challenging and thus we have declined to implement any extensions as the initial problem was time-consuming enough

## Build Instructions:

When the folder is downloaded from MMS it should the code and relevant files should be left in their respective folders. To run the program simply compile all the files in the source folder and then run the program using the following command:

```
java AutocompleteGUI <Relative File Reference> <Terms to be Displayed>
```

For Example:

```
java AutocompleteGUI ../wiktionary.txt 10
```

## Design:

The design section will be composed of an initial overview of the program's structure and then refined into more detailed analysis and explanation of individual classes and modules that may need to be explained or clarified.

Overview:

Much of the programs overarching structure is defined and dictated by the project specification and the provided API skeleton. Essentially the flow of the program is that in the main method the program catches any preliminary errors, reads the data in from the file and then invokes the event queue, so that the rest of the program's flow is essentially dictated by the user. As the user enters a string into the box the auto complete flow is called. This means that the program will find the first and last occurrence of the entered string as a prefix, return a list of matching terms using those values, sort them by their weights and then produce a list of terms that the user could be looking for. In order to terminate the program the user must click the 'x' button in the top right corner of the window. Although the API's require exceptions to be thrown this does not have to be caught by our program as the way they're utilised in our programs their exception cases can never be met

- Term

This class is used to store the data read in from the text files. Each term has two attributes, a string for the text and a weight associated with the text's frequency. Furthermore the class implements comparable so that we can define the natural ordering of the Term class. The class also has a toString method that will return a string containing the weight and text.

- By prefix order

This static method returns a comparator (an interface) that can be used to compare two objects. This method works by taking in the length of the prefix ( $n$ ) and comparing the two strings using only the first  $n$  terms of each terms query. However if either term is shorter than the prefix then the method compares the strings normally.

- By weight ordering

This method also returns a comparator however this one is much simpler, it simply compares the integer weight of two terms and returns a number corresponding to which one is bigger/smaller or if they're the same.

- Compare To

This method simply defines the natural of the Terms which will be a string comparison of the queries

- **Binary Search Deluxe**  
This method is for used to find the first and last occurrence of a generic term in a sorted list. The methods receive a list of generics, a query generic as well as a comparator. These are used to take the query and search through the list for it using the comparator to determine whether or not the two generics are equal. Specifically, for the first index, the binary search is implemented in a way so that the start of the array sub-list that is being search can never go pass the first instance of the searched generic. Once the sub-lists beginning position is equal to (or near the first instance) the end of the sub-list will close in on the position of the first instance until either the first/end of the sublist are one apart and one of them will contain the searched generic should it exist in the list. The last index simply works in the opposite way
- **Autocomplete**  
This class is the simplest. It utilises the Term and Binary search classes to return the number of matches as well as return a list containing the matches.
  - **Number of Matches**  
This Method simply calls the methods to find the first and last occurrence of a query prefix and then returns a number corresponding to the number of terms with that prefix
  - **All Matches**  
This method loops through from the first occurrence of the term initialising a list of terms that have the prefix and then returns that list.
- **AutocompleteGUI**  
This method is by far the largest and has many components that can be rather confusing. Most of this method is strictly to do with the GUI however some small parts of the program deal with the programs start up and preliminaries. This class defines all that is necessary for the GUI, it also reads in the data from the file and produces output for any methods that throw an error detailing why the error was thrown and gracefully. It also deals with calling all methods necessary for the actual auto complete. We decided to go for a minimalist clean look, all white and black and no clutter, purely for aesthetic look, we considered having buttons for extra functions but ultimately decided they were superfluous.
  - **Main**  
The main method checks that the arguments have been entered and of the correct format. The method then reads in all the files and sorts them. The event queue is then called allowing the GUI to be implemented and waits for events for it to resolve.
  - **Constructor**  
The constructor has many parts to it. It takes the arguments and terms and sets them equal to the relevant class variables. The constructor then initialises all the Java swing elements, setting them to visible, sorting the layout of the panel and adding them to the panel. The constructor also defines a document listener for the query box, this is so that an event occurs every time the query box is edited. This calls the results output method that allows the relevant matches to be displayed
  - **Results Output**  
This method calls the auto complete run method and then uses the results to output all the terms up until the number of terms asked for is found
  - **Auto complete Run**  
This method is used to call the auto complete methods that use the other classes to return all the matching terms, this method then sorts them by weight and returns that list, if there is an array index out of bounds error then there are matches so the method returns a n array that contains no matches.

## Testing:

To test the program we will firstly run a search and check the output is: correctly sorted, the right number are output, check that all the matches were printed by adding some temporary code to output all the matches found.

After this we will test forgetting to put in arguments and putting in illegal arguments for the number of matches to be returned.

### Test 1:

We will search for the prefix “tha” with argument 10 and manually work out the results expected before seeing what is returned.

#### Expected output

The GUI should display the top 10 results in weight order and the terminal should display all the terms from the file that begin with “tha”.

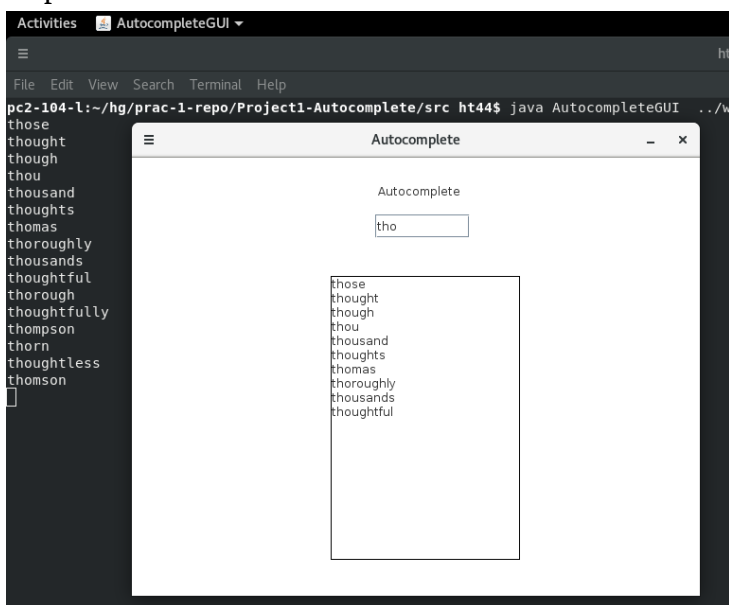
#### Top 10:

Those, Thought, Though, Thou, Thousand, Thoughts, Thomas, Thoroughly, Thousands, Thoughtful, Thorough

#### All:

Those, Thought, Though, Thou, Thousand, Thoughts, Thomas, Thoroughly, Thousands, Thoughtful, Thorough, Thoughtfully, Thompson, Thorn, Thoughtless, Thomson

#### Output:



### Test 2:

We will do two tests to ensure that an illegal file name or number format or running the program without any argument outputs a message and then terminates

#### Test part 1

A message to terminal about how to run the program and the GUI shouldn't build

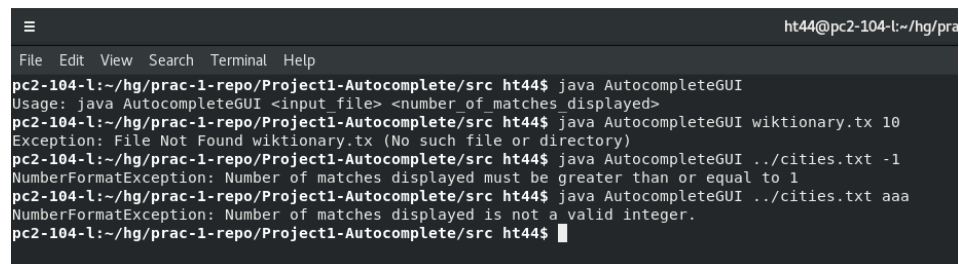
#### Test part 2:

Expected output:

A message about file not being found

#### Test part 3:

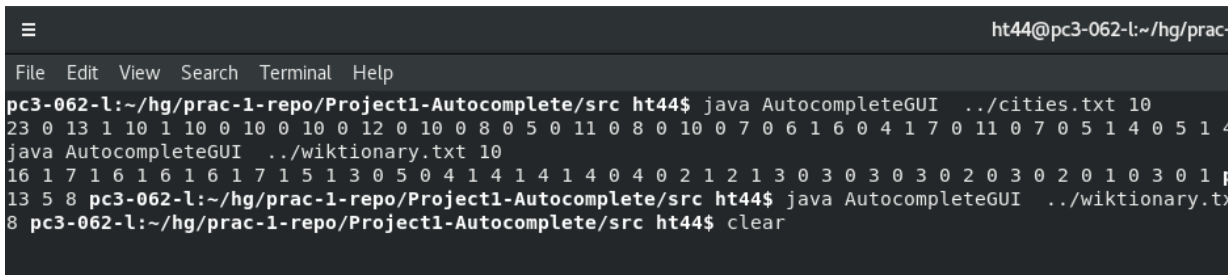
A message about number being ill formatted or below 1



## Speed:

The specification suggests that an auto complete must be quicker than 50ms to be used, we shall take that as our benchmark. The specification also states that it should make at most  $1 + \log_2 n$  comparisons in the worst case, the binary search will make  $\log_2 n$  comparisons, and the if statement will make an extra comparison however big o notation ignores constants.

To test the speed of the program we will take the time from entering a string to output and then see what times we are getting for each letter in the alphabet.



```
ht44@pc3-062-l:~/hg/prac-1
File Edit View Search Terminal Help
pc3-062-l:~/hg/prac-1-repo/Project1-Autocomplete/src ht44$ java AutocompleteGUI ../cities.txt 10
23 0 13 1 10 1 10 0 10 0 10 0 12 0 10 0 8 0 5 0 11 0 8 0 10 0 7 0 6 1 6 0 4 1 7 0 11 0 7 0 5 1 4 0 5 1 4
java AutocompleteGUI ../wiktionary.txt 10
16 1 7 1 6 1 6 1 6 1 7 1 5 1 3 0 5 0 4 1 4 1 4 1 4 0 4 0 2 1 2 1 3 0 3 0 3 0 3 0 2 0 3 0 2 0 1 0 3 0 1 0
13 5 8 pc3-062-l:~/hg/prac-1-repo/Project1-Autocomplete/src ht44$ java AutocompleteGUI ../wiktionary.txt
8 pc3-062-l:~/hg/prac-1-repo/Project1-Autocomplete/src ht44$ clear
```

The average for the cities file was 8.2ms and the average for wiktionary file was 4.2.

## Evaluation:

Overall our program can be reviewed as a success and a good solution. When comparing the solution to the specification it is evident that all the criteria is met, and our programs speed is near instantaneous proving we have an efficient solution. The API's are also commendable, their can be an attraction to designing the API's so that they only suit your program however that is bad practice as the binary search for example should be implemented in a way that they could be used in many contexts. We took extra care to make sure that in their design they could be as flexible as possible and not narrowed to the project. The project is also, we feel, rather robust there are many errors that we have eliminated and we have designed the program so that later errors are never actually thrown, we however would not say it is indestructible. We also should note that the GUI, although simplistic, could be criticised for being rather basic, although that was are aim for an intuitive interface it also could have been

## Conclusive remarks:

To conclude this task was very tough, the programming logic was hard and problem detection to a certain degree was also tough. Getting off the ground was and always is the hardest part of the program as it takes a long time to try and get your head around the problem as well as finding a starting point. As the program got larger and larger error detection became harder and harder. For example we had to stay in the labs for 5 hours trying to solve one mistake, only to find another mistake. However as partners we both enjoyed this experience and genuinely would have enjoyed extending our solution so that clicking a city would return the weather in that city, or clicking a word gave you a definition. This would have been an enjoyable and hard extension that we were both keen to implement.

## Personal Contributions:

AJ87 – For this project I wrote the Binary Search, Created the GUI and did some minor work on the Term class, I also wrote the report. All of my changes/work had to go through ht44 as I have had severe issues with the repository and could not figure out how to fix it (we would of started a new repository however we may only link one repository I believe).

HT44- For this project, I worked on the main method, the Term class and the Auto-complete class, as well as the linking all the classes into a single running program. I performed all of the empirical work and provided aj87 the results to use in the report. I also helped layout the work in a more easily readable way throughout the project, with many of the comments and javadoc comments being created by myself.

Repo Link: <https://ht44.hg.cs.st-andrews.ac.uk/prac-1-repo/>