**Leo Le**

**Programming Assignment 2**

**Table of Contents**

**How to start** --------------------------------------------------- **(2 - 3)**

**Main program testing (how to use)** ------------------ **(4 - 16)**

**How it works** -------------------------------------------------- **(17 - 26)**

**Problems encountered & lessons learned** --------- **(27)**

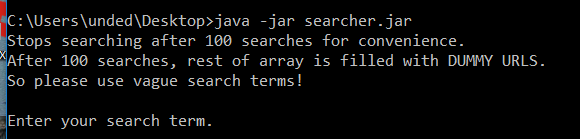
**Miscellaneous testing** ------------------------------------- **(28 - end)**

**Original Google doc:**

https://docs.google.com/document/d/1KlA6n0lU3\_Z2lkkOE9mfNjvAIZKUqj6AcdFcKs0MeH8/edit?usp=sharing

**HOW TO START:**

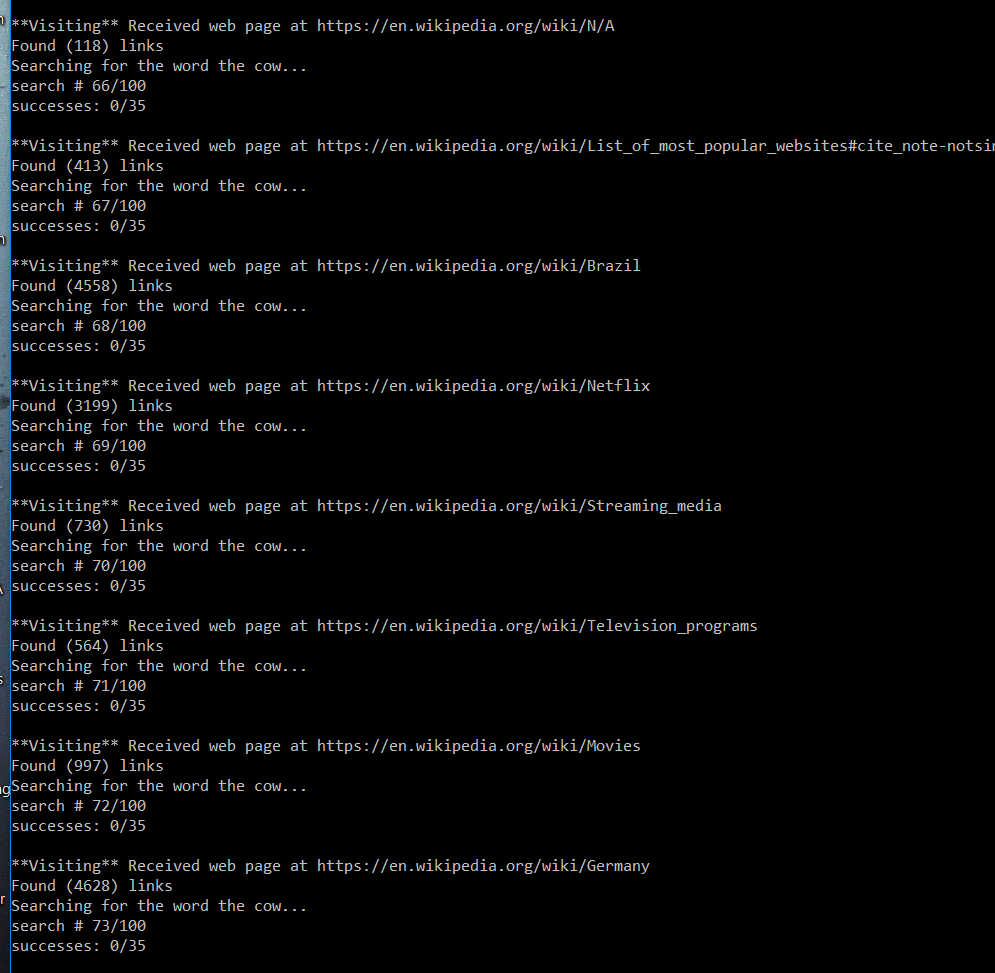
I use winrar to extract and zip my files.



1. Extract the files (With winrar, right click the zip and click “extract here”)
2. Open command prompt
3. Navigate to directory where searcher.jar is. I held it in my desktop
4. Type “**java -jar searcher.jar**”
5. The program will start in your command prompt

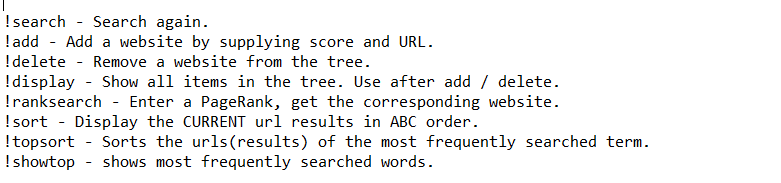
…

Enter your search term, as prompted.



^ If you did it right, you should see that.

**Type “!commands” for a list of commands**

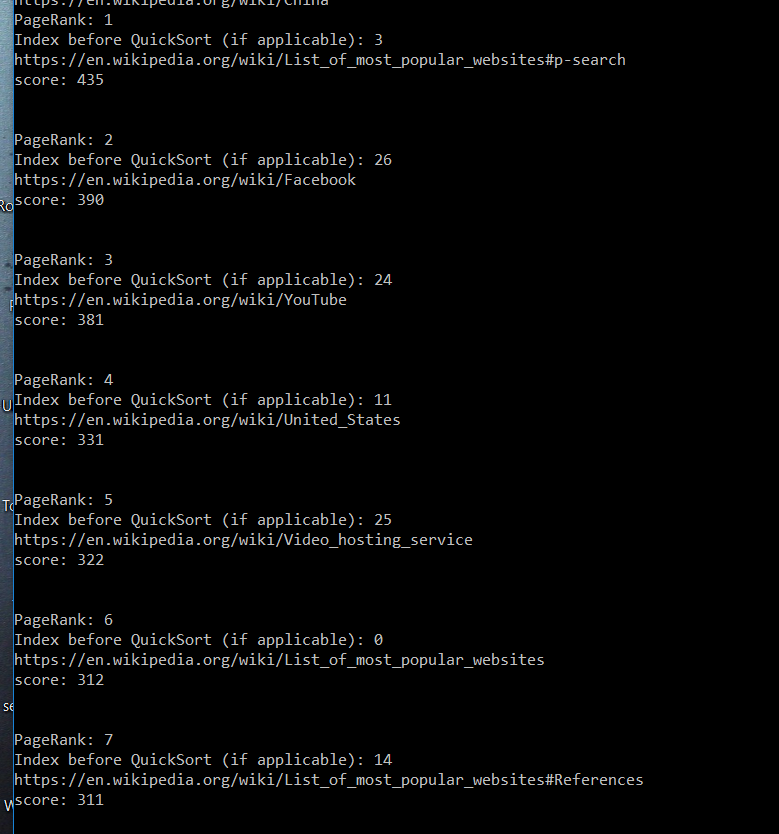


**TESTING (aka how to use)**

We need to display 30 Urls, my screenshot window would only fit 7 results. Scroll down/up to see more.

Launch program

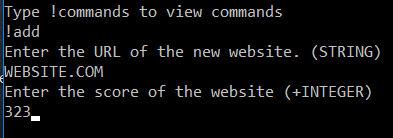
Search for word “the” by typing “the”. You may search any word.



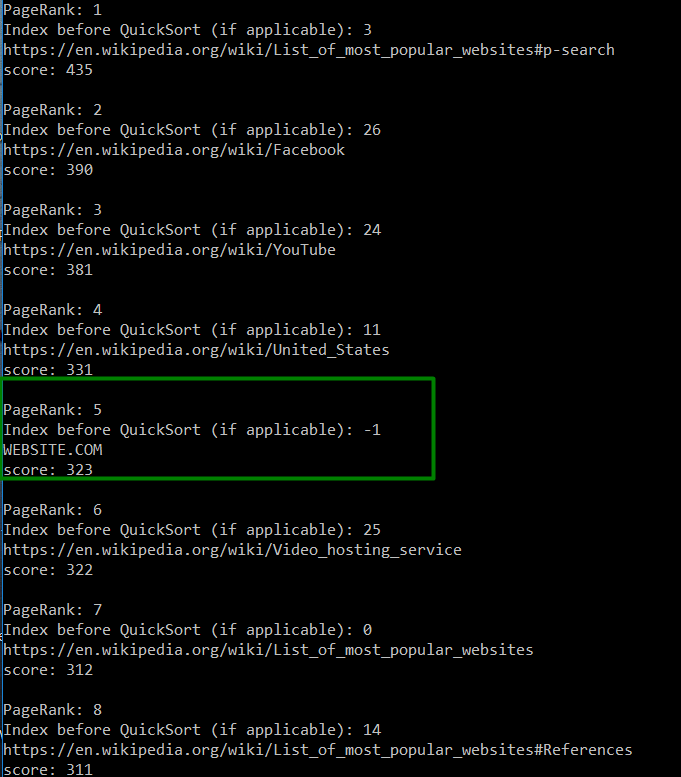
Type **“!add”** to add a url

Lets put the website between 4 and 5, with a score of 323

Name the website “WEBSITE.COM”



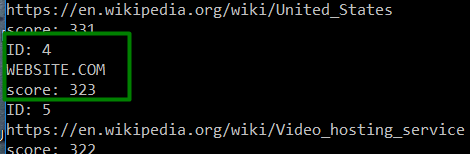
Type **“!display”** to view the list



Type **“!delete”** to delete a node from the tree

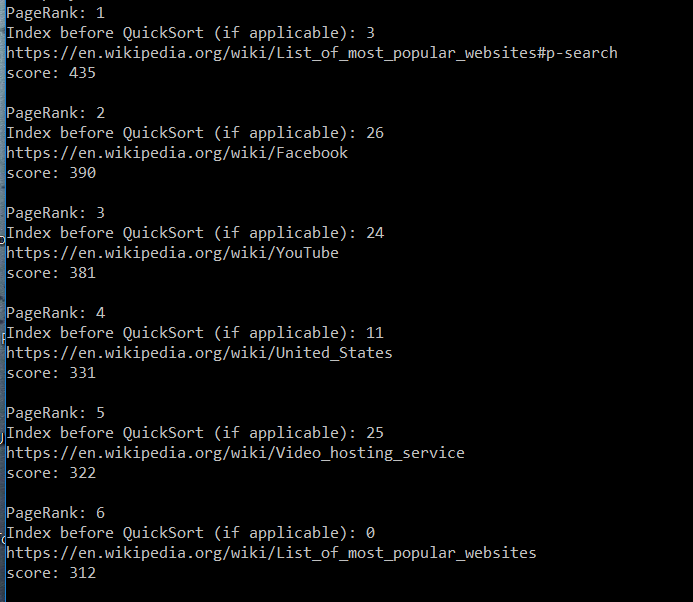
After you type “!delete”, you will be shown all the websites with their IDs

Select the ID of the website you want to delete



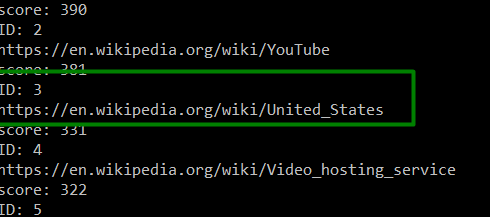
Type “4” to delete the website we added

Type **“!display”** again. Notice how the deleted website is gone.



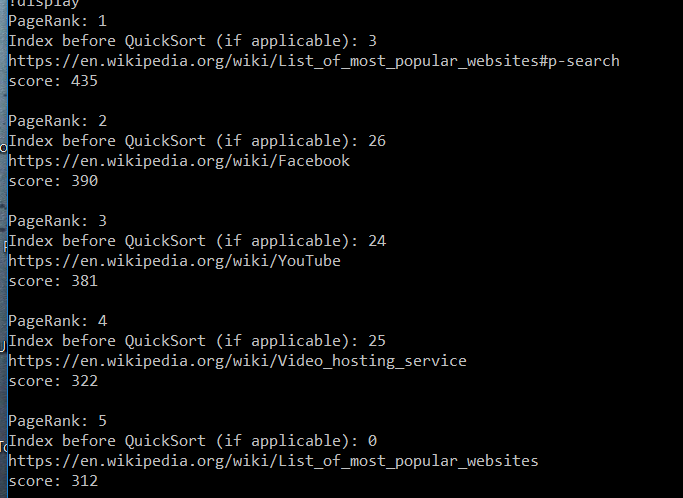
Type **“!delete”** again

Type “3” to delete the united states



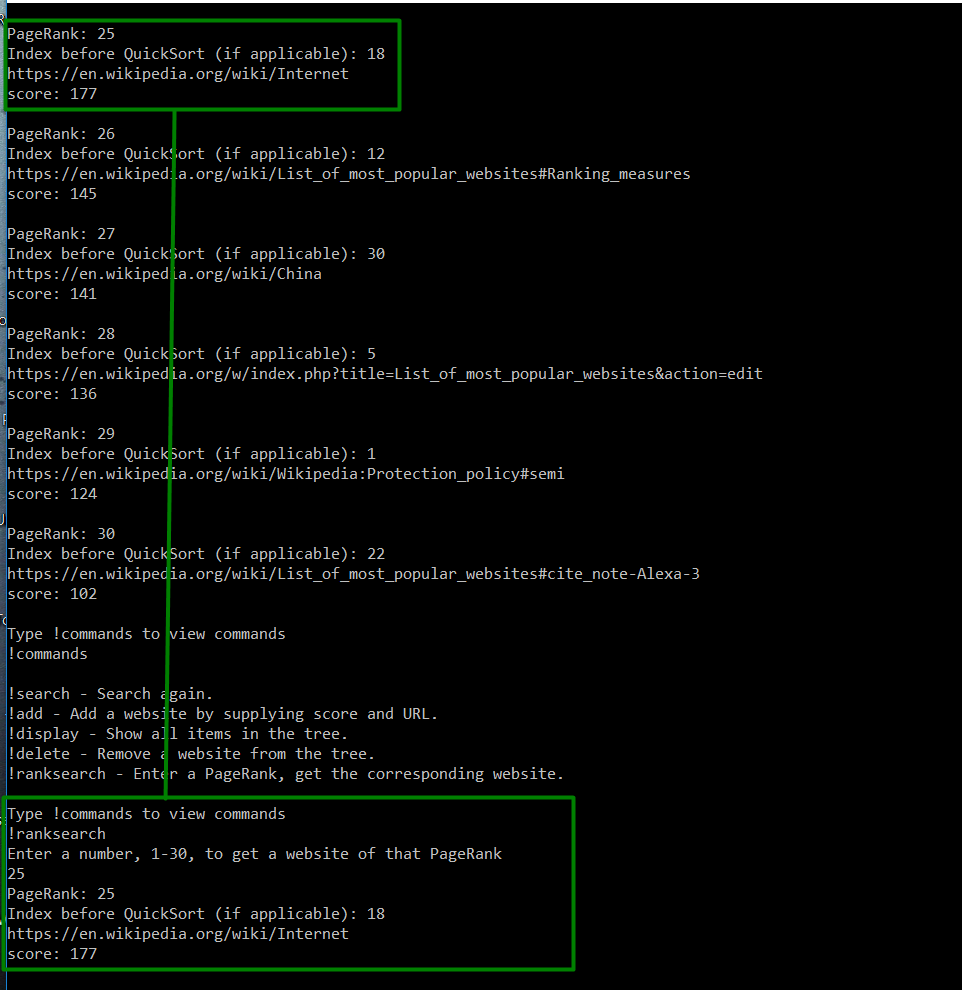
Type **“!display”** to view results

United states is deleted



Type **“!ranksearch”** to search by rank.

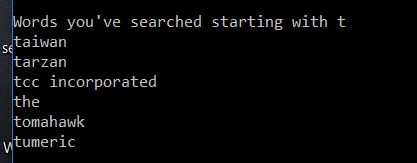
In this case, we want the website with pagerank 25



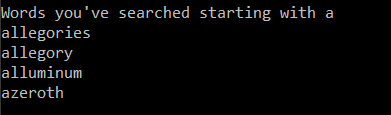
Now lets test the buckets

Do **“!search”** and search for random things starting with the letter t

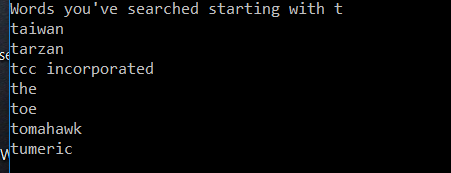
After you search for a word starting with t, it will tell you all the words you’ve searched for in the past starting with t



Now do **“!search”** and search for random things starting with letter a



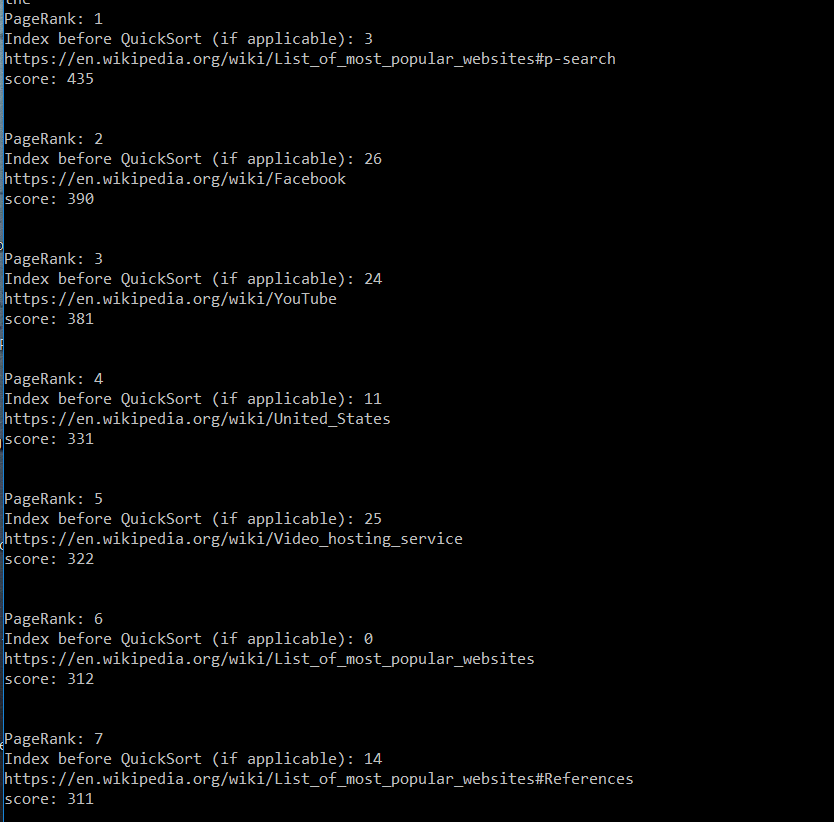
Now do **“!search”** and search for a word starting with t again



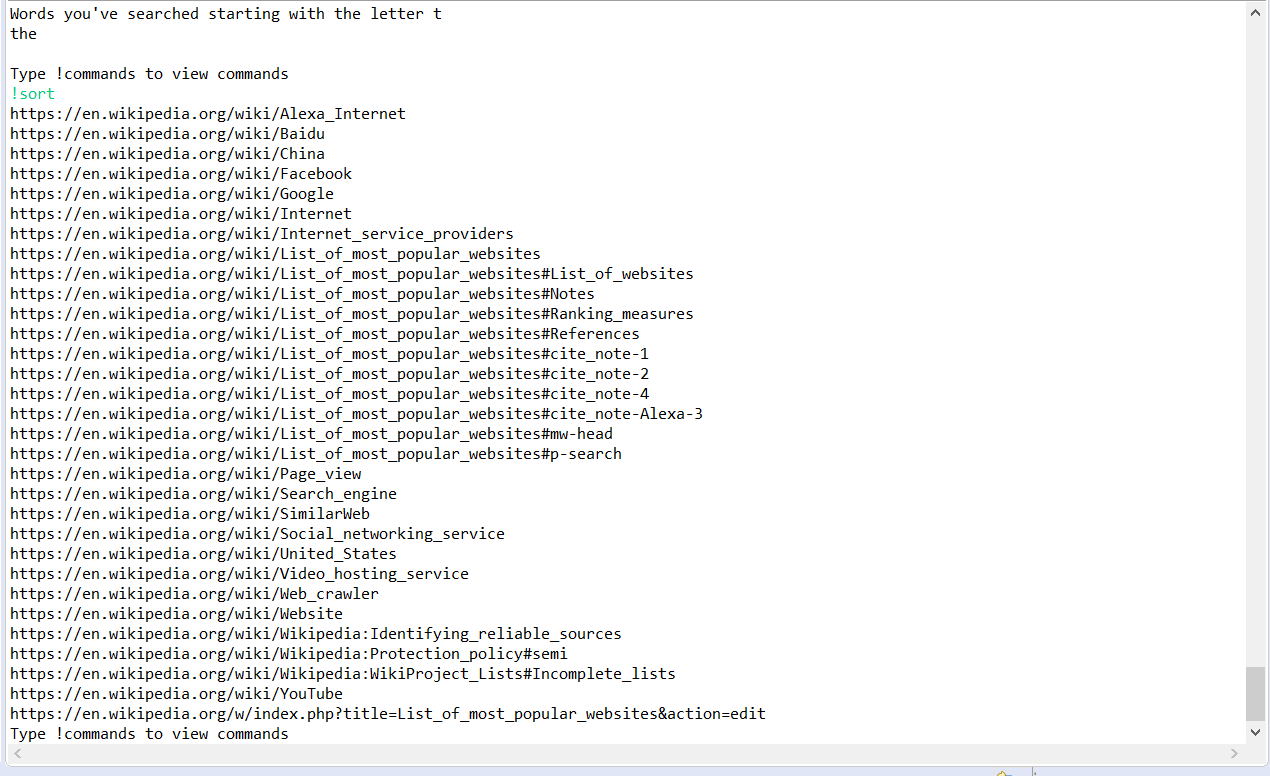
Do **“!search”** and search for “the”. After giving you the results, it will tell you all the words you’ve searched for in the past starting with the letter t, since “the” starts with “t”.

Since you already searched for “the” in the past, it will not invoke the web crawler and immediately give you the results.

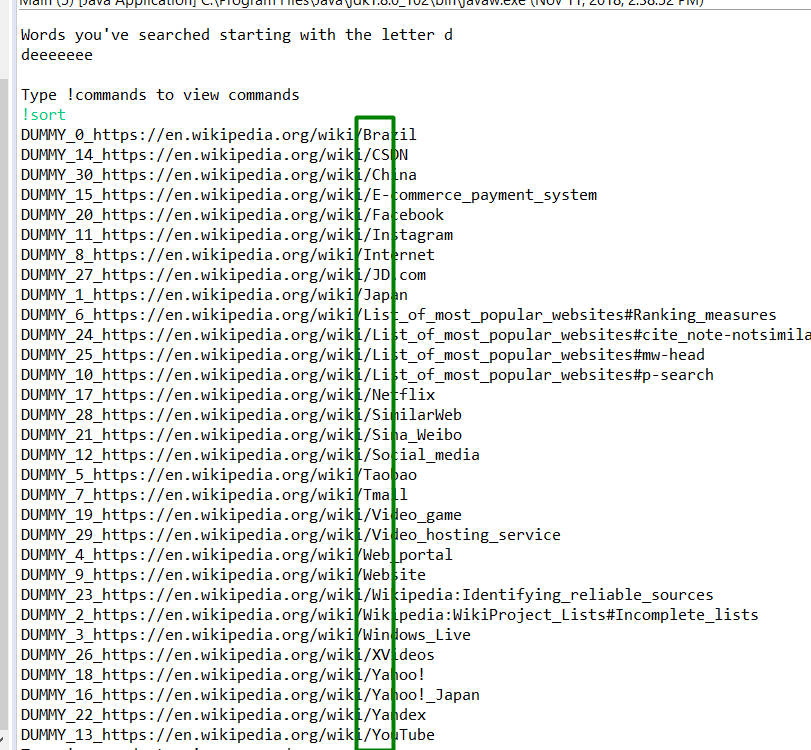
If you search for any of the words you’ve searched for in the past again, it will not invoke the web crawler and give you the results tied to that word.



I’m not sure if we are supposed to bucket sort urls or keywords, so to be safe, I added a method called **“!sort”.** It takes the url results of your current search and prints them out in ABC order using bucket sort.

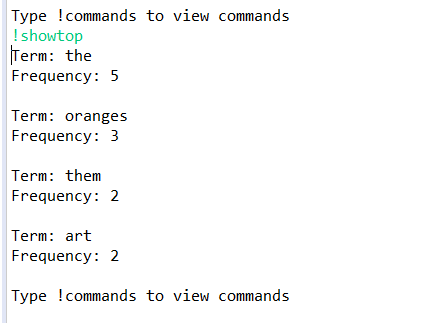


The sort only works with wikipedia Urls. It sorts based off of the first letter after the fourth dash. If there is no fourth dash, it will just sort the regular url. I did it like this so all the items don’t get placed in the “H” bucket since all urls start with https://

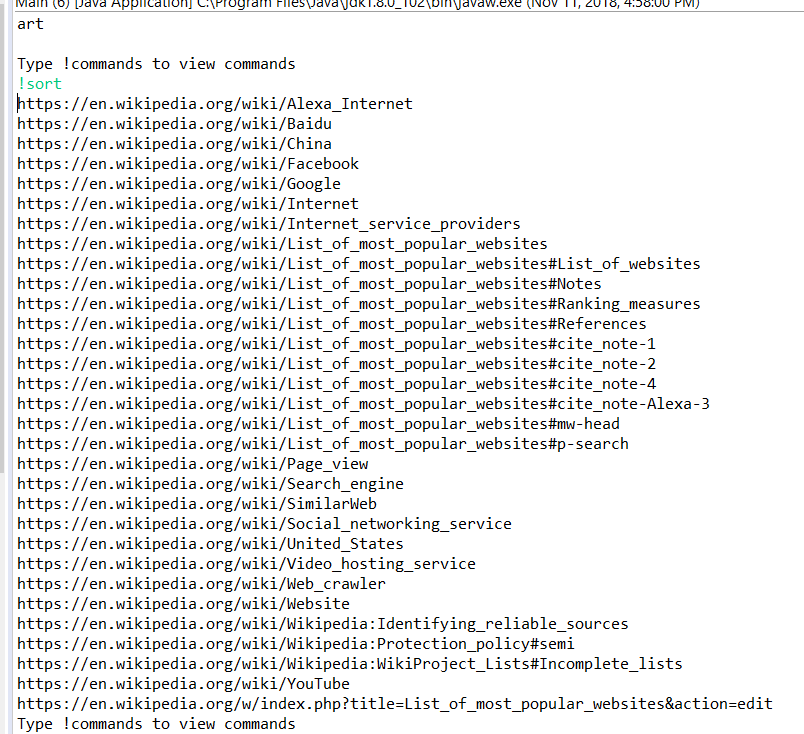
****

After some recent announcements, I also recycled code from my last program to add two commands, “!showtop” and “!topsort”

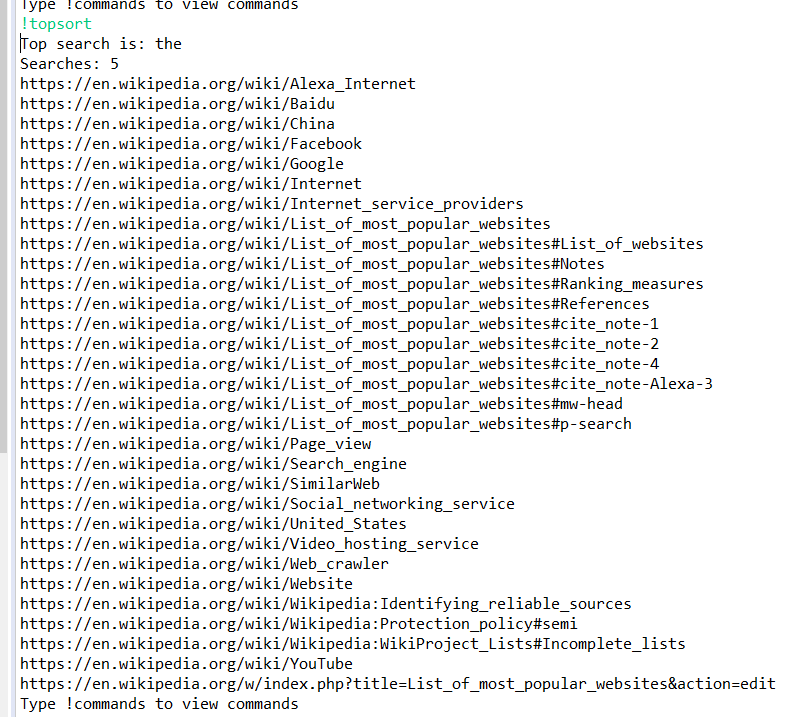
In the example below, if you do “!showtop”, after searching the word “the” 5 times, “oranges” 3 times, “them” 2 times, and “art” 2 times, you will get…



If you type “!search” then “art”, you will get the results for art. If you do “!sort”, you will get the sorted results for art

****

**But,** if you follow it up by doing “!topsort”, since “the” is the most popular search term, you will get the ABC ordered results for “the”



So keep in mind there is a difference between “!sort” and “!topsort”.

**HOW IT WORKS**

The webcrawler searches the wikipedia page for the terms and adds websites when the term exists to the list of successful URLs. It makes a maximum of 100 attempts and if the number of successes is less than 31, it fills the list of successful URLs with dummy URLs pulled from unsuccessful websites.

If you search for a term that was not previously searched for, the term will be put into a SearchBucket object. The SeachBucket object has 127 indexes, each holding a linked list. Each index represents an ascii code for a respective letter. For example, the ascii code for “j” is 106. So index 106 will have a linked list for search terms starting with j.

If you type !search and search for a term, the program will go to the SearchBucket index based off of the first letter of your search term and search the linked list inside the index for your term. If not found, it will add the term to the bucket. If found, meaning you searched for that term before, it will give you the array of Urls tied to that search and give you that as the result. This means that the web crawler will not be invoked if you search for the same term twice.

Either way, you will have a list of Url objects with their url and score. These Urls are converted into UrlNodes, with the Url object being the key of the nodes. The UrlNodes are put into the tree for you to manipulate with the commands.

**Url class**

I needed a way to store urls and their scores together. So I put it into one object. The 4 variables that are added up to calculate the score are random.

**UrlNode class**

A normal node with left, right, and parent, with a url object as the key. I made this class because trees can only manipulate nodes that have left, right, and parent pointers.

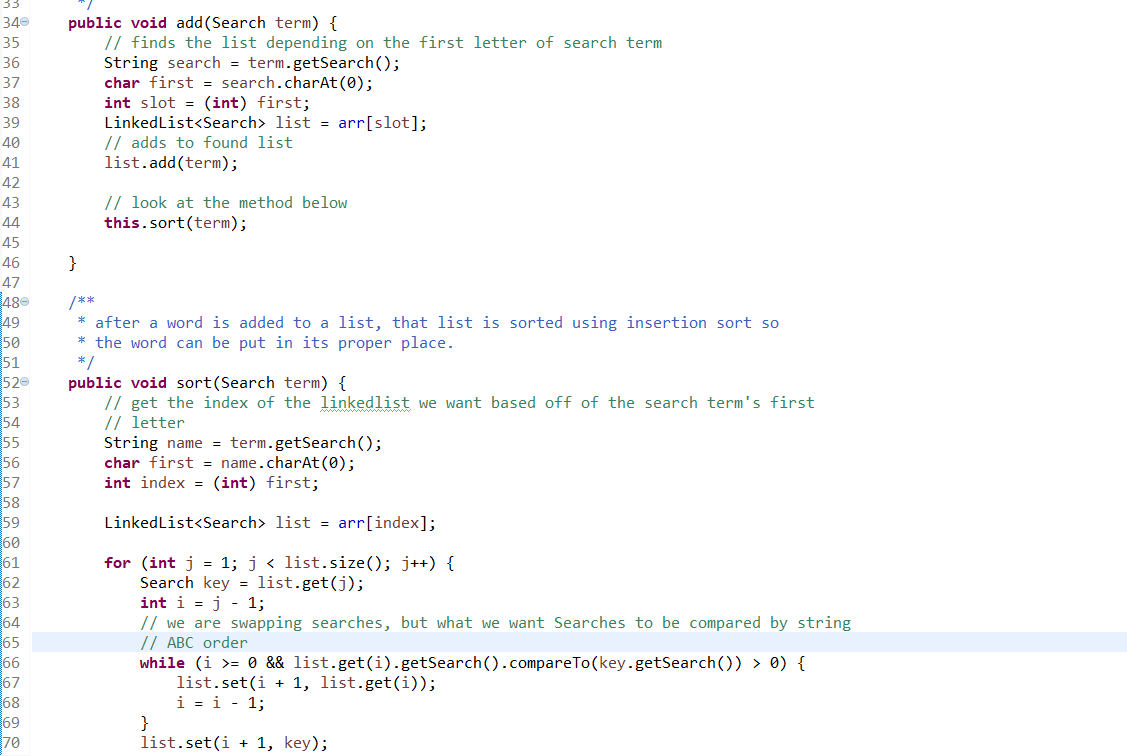
**UrlTree class**

It Manipulates UrlNode objects, which hold Urls, which hold the website url its score. Therefore, this is my way of manipulating the results and changing their scores / deleting / adding them. The only notable change I made was by adding searchByNode and deleteByName. These two methods exist because the tree is sorted based off of score, but if two websites had the same score, I need a way to delete them based off of name.

**SearchBucket class**

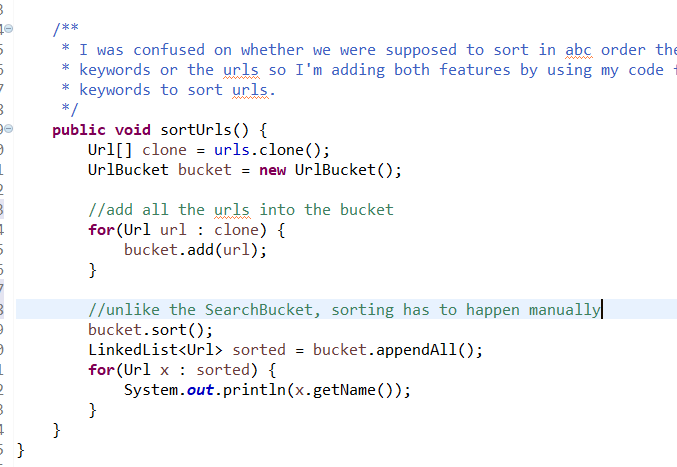
I described this in my explanation of how my program worked, except for the fact that whenever a new search term is added to a linked list within the bucket, the linked list is sorted with insertion sort so that the entire linked list will be in ABC order.

Notice how sort() is at the end of the add() method



**UrlBucket class**

I added this because I was confused on whether we should bucket sort the urls or the search keywords. So I recycled SearchBucket code to do both. This class exists for the Program class method below.



The code for UrlBucket is below. Unlike SearchBucket, instead of sorting the linked list every time you add the item, I add all the items to the bucket and then insertion sort on every list within the bucket when I need to.

Within the insertion sort, the items urls are compared based off of whatever is after the fourth dash

For example, from the urls

https://en.wikipedia.org/wiki/Internet\_service\_providers

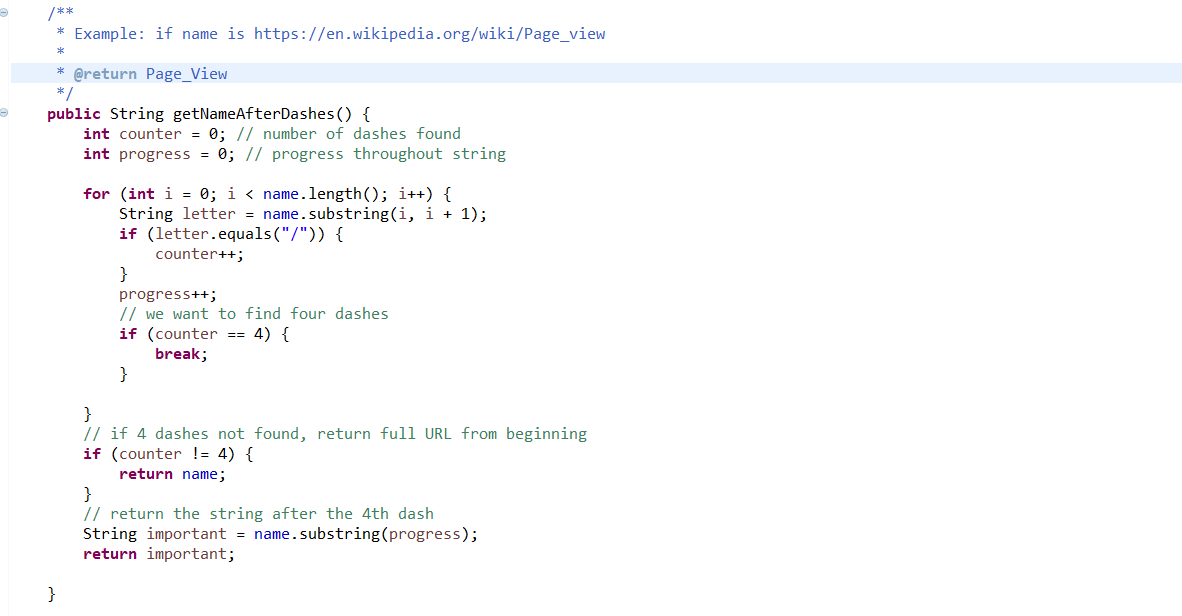
<https://en.wikipedia.org/wiki/Page_view>

We would be comparing

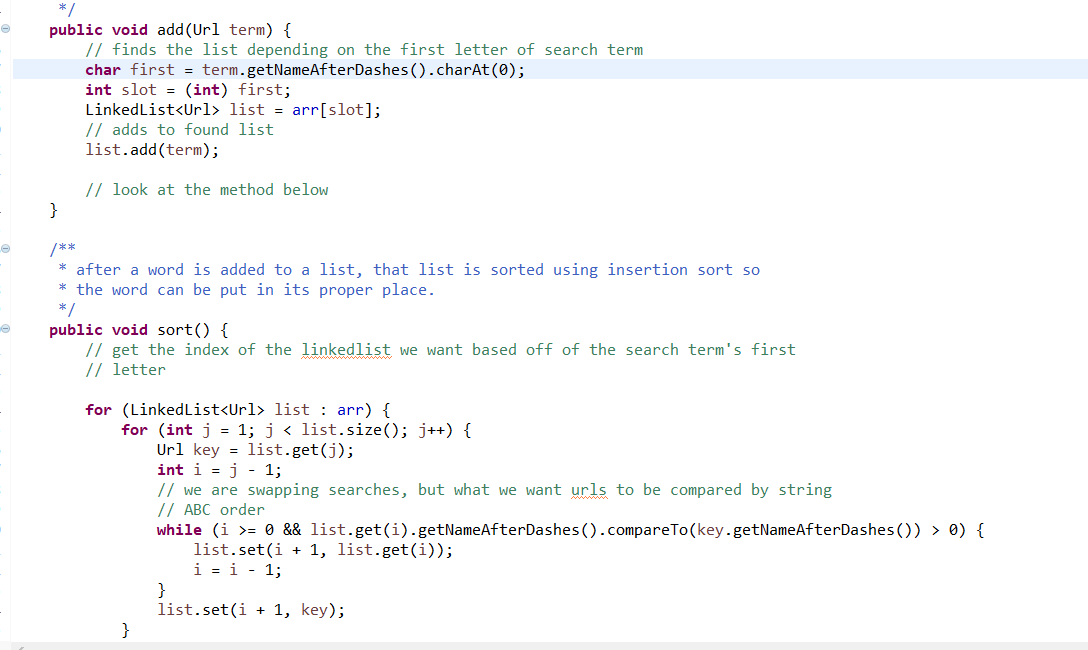
“Internet\_service\_providers” and

“Page\_View”

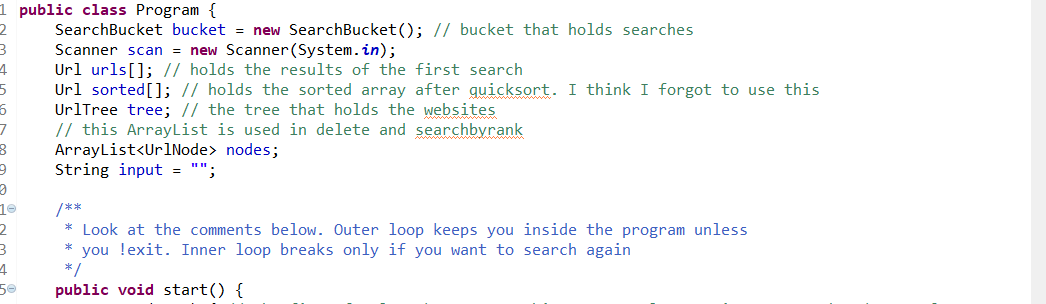
Also, a URL like “<https://en.wikipedia.org/wiki/Page_view>” would be placed in the “P” bucket. I do this by using the Url method below



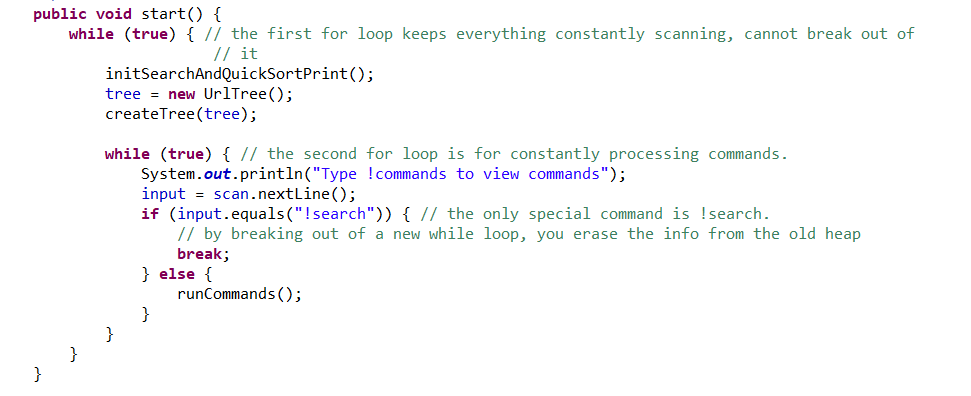
Below is code for the bucket, explained above



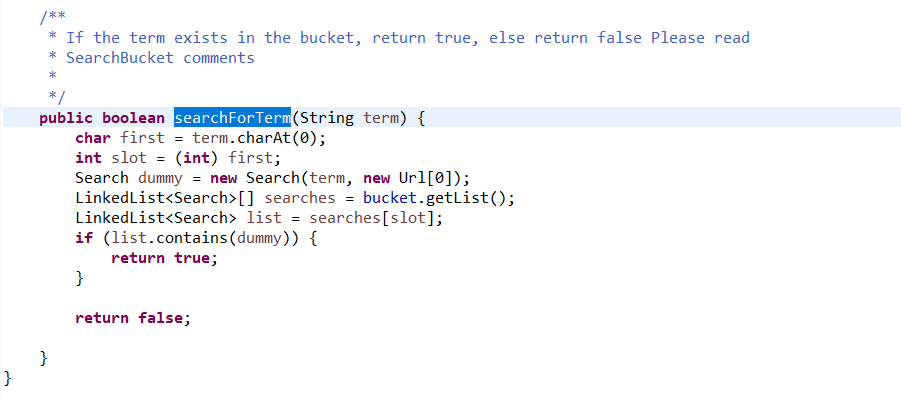
**Program class**

****

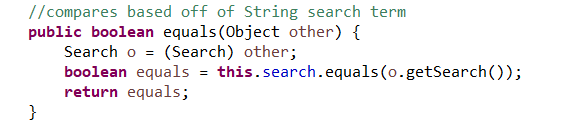
^The instance of the bucket is outside the scope of all the methods because the bucket holds the searches. Urls and the tree which manipulate the urls change with every search, but we need to keep track of the searches after every search so that we don’t have to invoke the web cralwer if the same search happens twice.



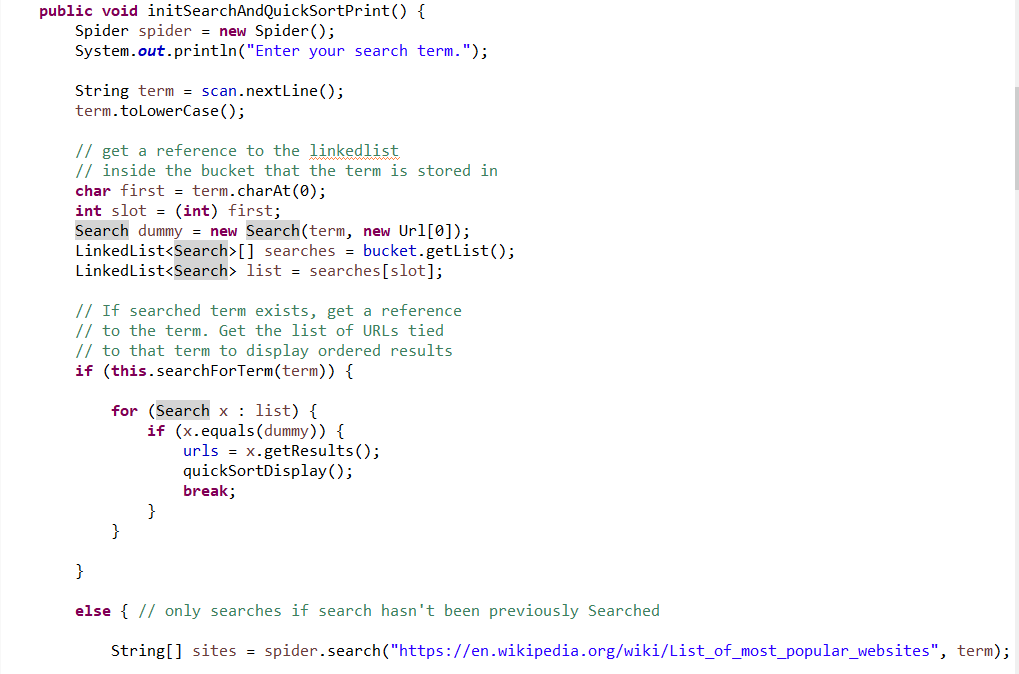
^The first while loop keeps you inside the program, we only break out of the second while loop to search for another term.



^You need to understand the bucket in my overall description to understand this. This method just checks if the term exists in the bucket. It will exist if the term has been searched before, and will not exist otherwise. I use a dummy Search object because Search terms are compared based off of their field variable String which holds the searched term.

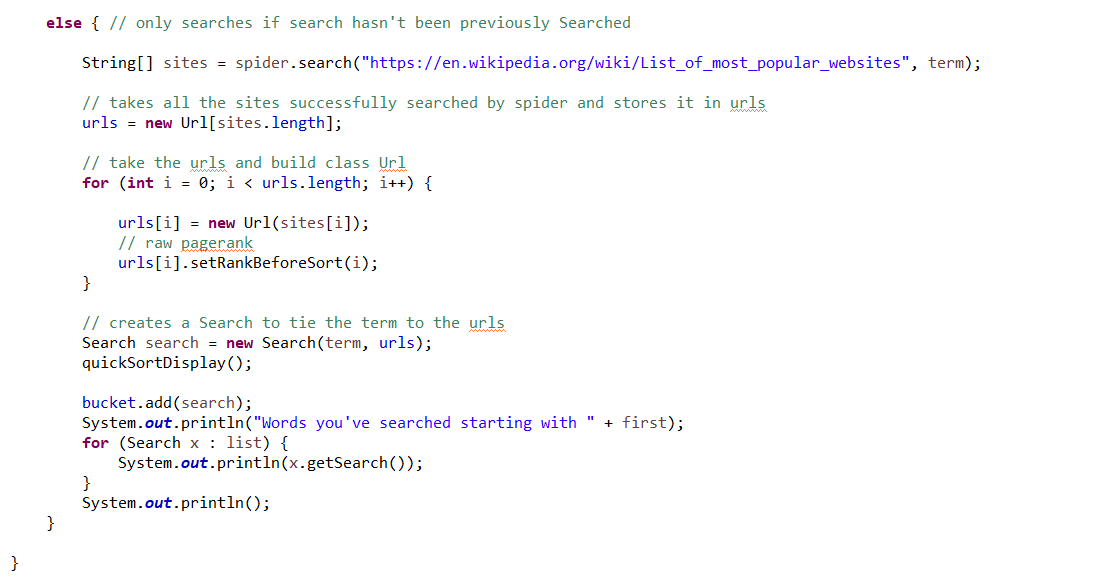


So I create a dummy Search with the term I’m looking for and check the list for that term.

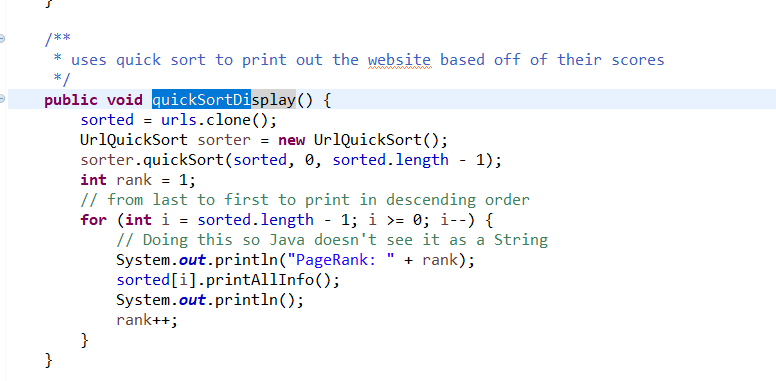


^initSearchAndQuickSort uses searchForTerm method to check if the term has been searched before. If it returns true (meaning it has been searched before), then we will get the list of Urls tied to that term and use quicksort to print the sorted Urls without invoking the web crawler,

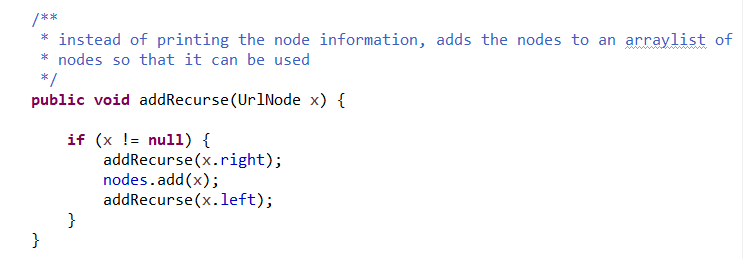
Else…



^Else the term hasn’t been searched before, so it will invoke the web crawler and add the search term to the list of searched terms. Every time a term is added to a linked list inside a bucket, the linked list is sorted so that it will be in ABC order. Either way urls will be sorted and printed with quicksortdisplay(), which is shown below.



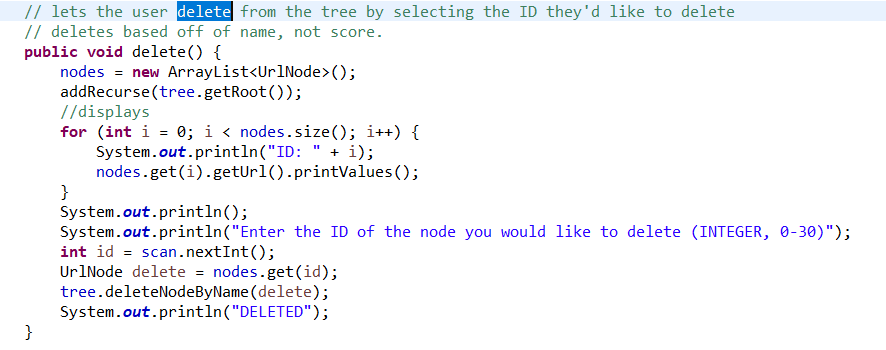
^I sort a clone of the array instead of the actual array because I do not want to manipulate the original array. I was also worried that if I sorted the array then added all the items to the tree, it would make it a height = n tree.



^This method is used in other methods like delete. It adds all the nodes in the tree into an arraylist of UrlNodes outside all the scopes



^this is what is being added to.



^uses addRecurse to fill the ArrayList of nodes with everything that is in the tree. I do it this way so that I can list all the nodes and the user can select which node they want to delete based off of the index in the array the node is in. The user selects that node, and I use the tree’s method to delete that node from the tree.

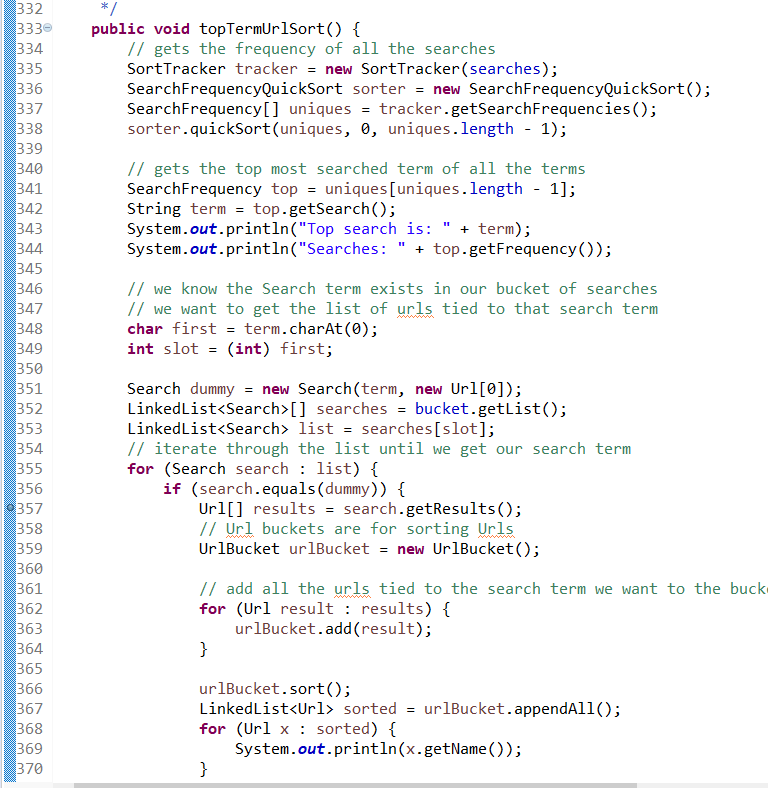
**SearchFrequency, SearchFrequencyQuickSort, and SortTracker** are mostly copy pasted from PA1. They are used to calculate the frequency of search terms. They are mainly used for the method below.

From lines 335 - 342...

SortTracker takes all the searches and returns SearchFrequency objects. SearchFrequency objects have a search term and the number of times that term was searched. The array of searchFrequency objects is sorted using quickSort and ordered by the number of times each search was searched. The last item, AKA the most frequently searched term, is taken.

From lines 348+...

The top search term has a string tied to that term. We want to find the urls tied to the top search term. I have a bucket called SearchBucket that holds all the terms searched in the past and the results tied to that search. I get a reference to the Search object that has the search term and search result I am looking for and sort the results using my UrlBucket.



**PROBLEMS ENCOUNTERED**

* I was worried that adding to a BST immediately after quicksorting would just create a tree with height n, so I quicksorted a clone of the array instead.
* I did not know how to create buckets for letters, so I used <https://www.cs.cmu.edu/~pattis/15-1XX/common/handouts/ascii.html>

To see that ascii codes go to 127, so I made the buckets based off of the first letter of a search’s ascii code and gave the bucket 127 indexes.

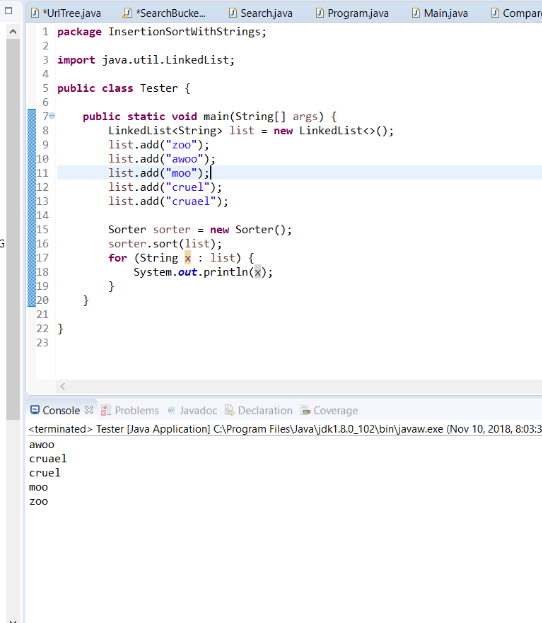
* Trees only deal with nodes, so I created a UrlNode class that has Url as a key.
* It’s hard to use the tree’s inorder method to manipulate the items from the tree in mass, so I created an arraylist of nodes that exist outside all the scopes. Then, I changed inOrderPrint to add all the items to the arraylist of nodes instead of printing. This is helpful for methods like delete(), where I need to let the user select which node they want to delete. The arraylist is emptied every time it is needed by creating a brand new instance of the object.
* I added the deleteByName and searchByName methods to the tree because it’s possible for two websites to have the same score. Then I need to delete by name, not score.
* At first I thought I could not insertion sort linked lists so I sorted it using Java and Comparable interface. Then I found out Java’s LinkedLists support indexes, so I used insertion sort.
* To bucket sort the urls, I did not know how to avoid putting all the urls into the “h” bucket, since all urls start with https://. So I made a method to return the important part of a url, explained in my UrlBucket class.

**LESSONS LEARNED**

* How to use ascii codes to quickly store and access words based off of their first letter. It will bring you directly to the index you need instead of searching for the bucket, and should work with numbers and symbols (~,!,@,#,$).
* The fact that when you call certain methods on the tree, like search and inOrderPrint, you need to pass the root as the argument.
* Java’s linked lists support indexes. I thought they wouldn’t because I was taught linked lists only had a pointer to the next node. I thought it would be impossible to use insertion sort on linked lists because insertion sort depended on indexes.
* You need to turn your data into a node class with a key to use it in a tree
* Changing the psuedo code of the tree to fit my needs. For example, making the delete method delete by a String name instead of an integer and making the inOrderPrint add to a list of nodes instead of just printing. Or making the insertion sort sort compare based off of String instead of integer.
* Insertion sort can be used on Strings because you can compare Strings.

**MISCELLANEOUS TESTING**

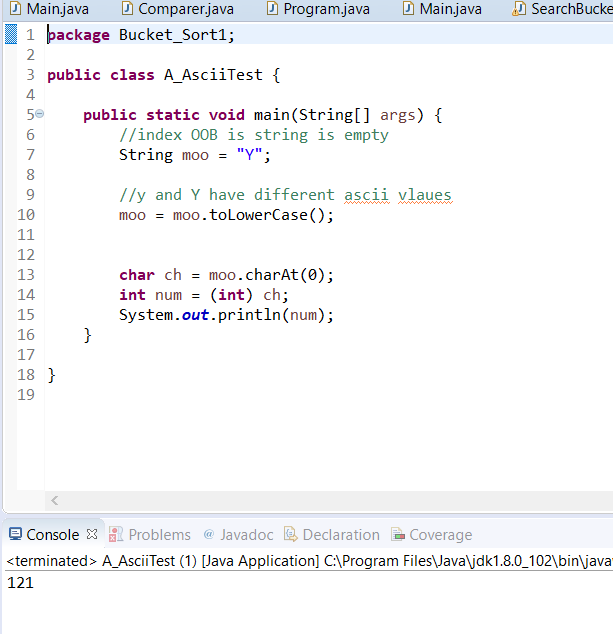
Testing my insertion sort with linked lists and strings.

****

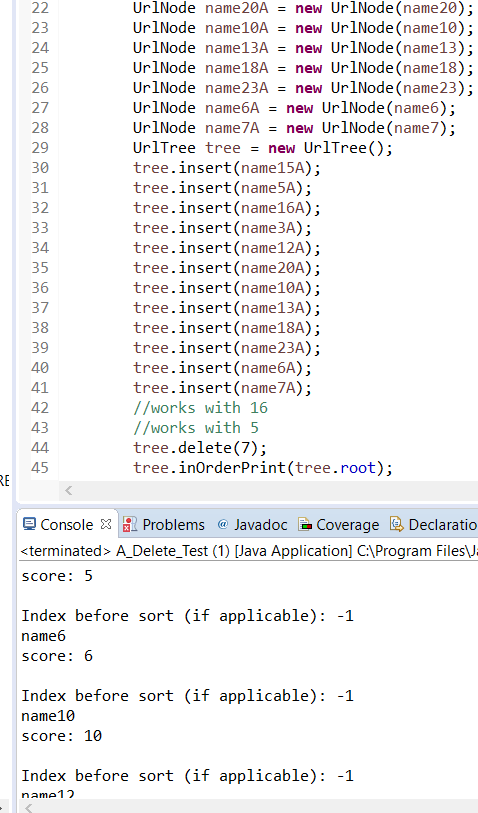
Testing ascii



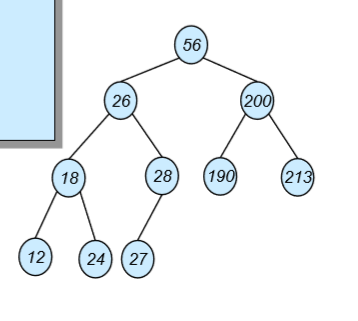
Y’s ascii code should be 121

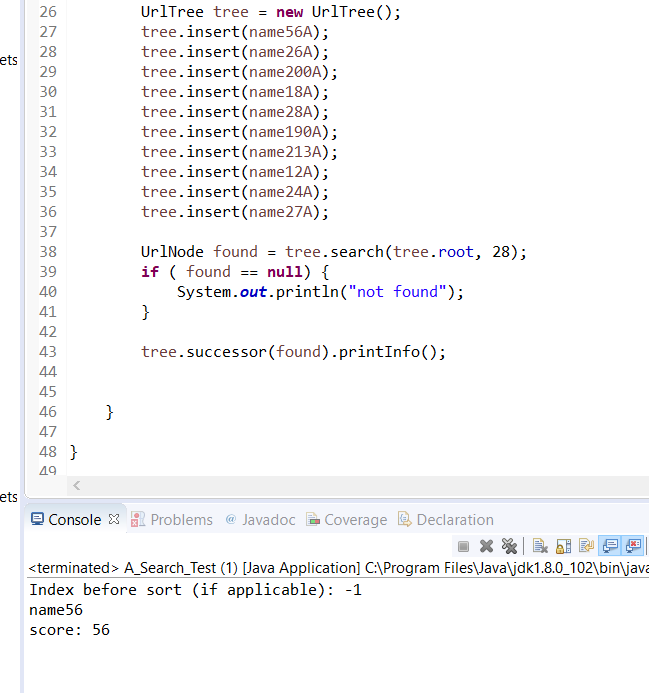
****

Testing node delete with integer. Node name7A would have value 7. It’s not printed in console means it’s deleted.

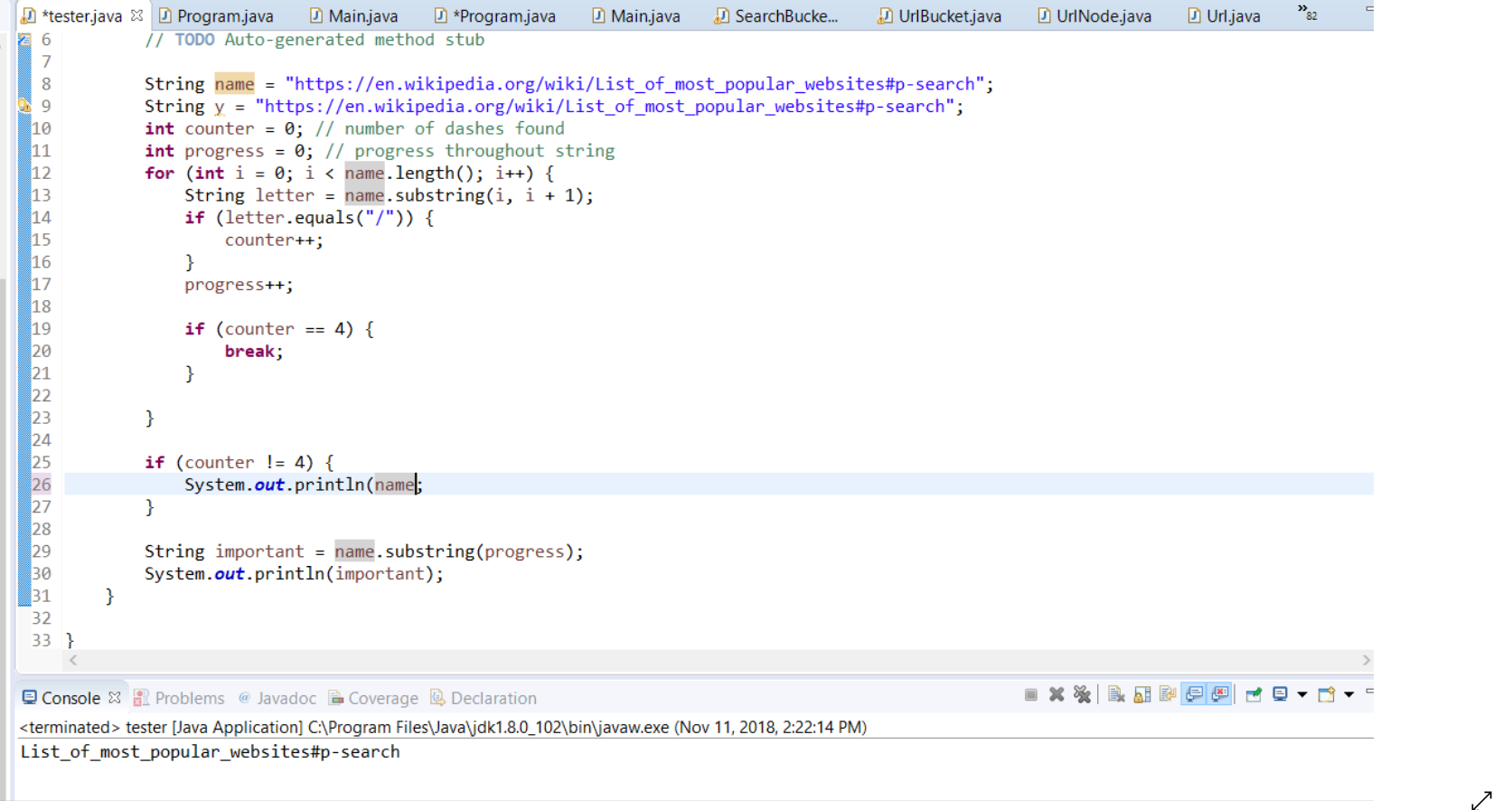
****

Testing successor with this tree

****

****

Testing a part of my getNameAfterDashes() method



Since my old version of sorting the bucket used Comparable and not insertion, they’re kind of useless screenshots. If you want it, here’s the link.

https://docs.google.com/document/d/1VqePO\_M0ioK6XFlz75DVogIoz9ay7qy5fruCvjcBEpc/edit?usp=sharing