Programming Assignment 1 Report

(SERP Simulator)

Yang (Liam) Li

CS146-S7

1)Installation(p2)

2) Software Design(p3)

3) Implementation (p4 - p7)

4)Features/Functions/Classes

(p8 - p11)

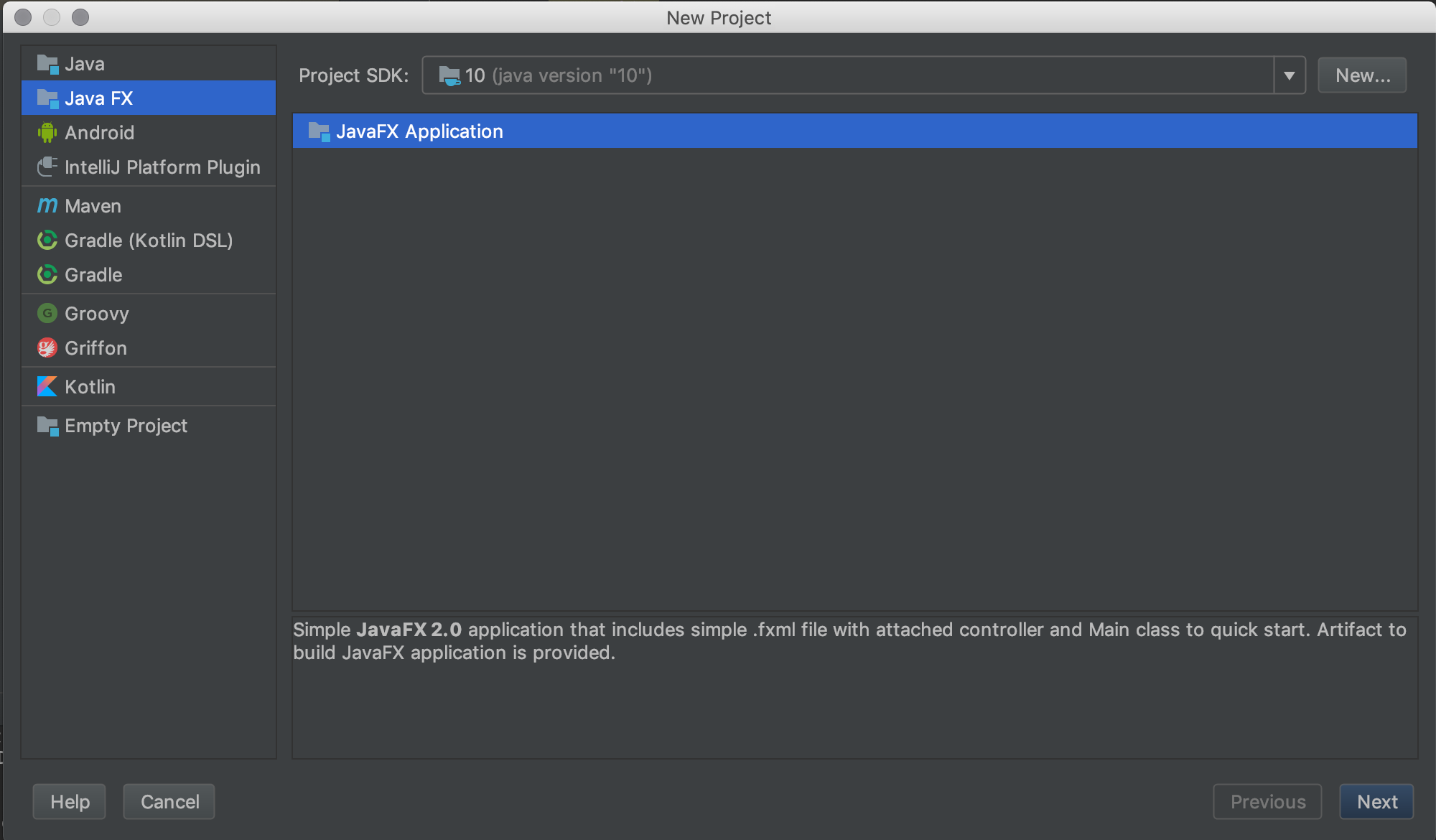
5) Testing (p12 - p14)

6) Problems (p15)

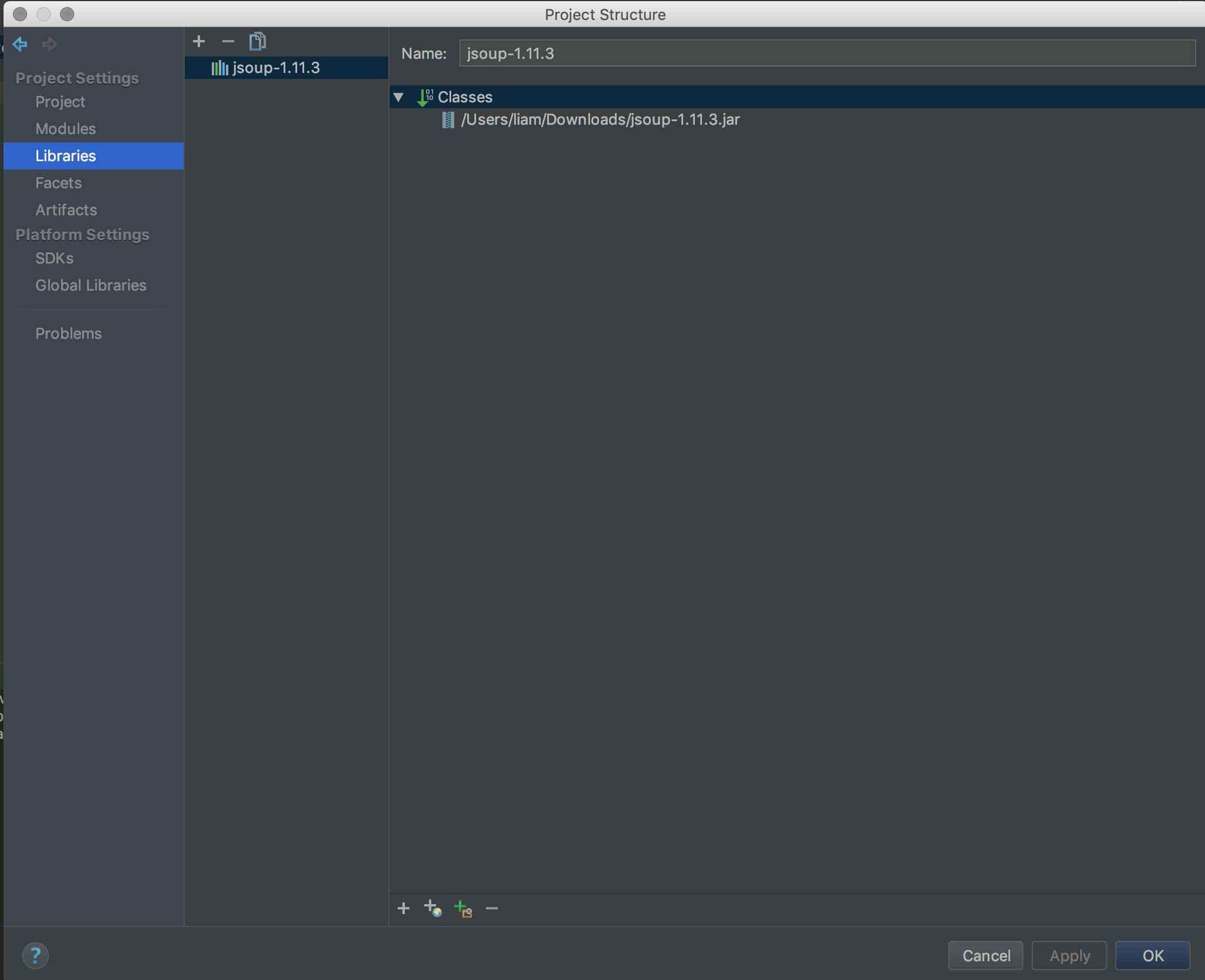
7) Learnings (p15)

**1) Installation**

* Unzip the zip folder, there are three part, an SRC folder that contains all of the .java files, a Program Report which is this Microsoft Word Document you are reading it now and a runnable .jar file called “Google Search”.
* Open the jar file and start testing! Warning: if you have the alert the “can’t be opened because it is from an unidentified developer”, go to the System Preference -> Security and Privacy -> General -> Allowed apps downloaded from and let this app open!
* You can modify the code, or compile/run by yourself, you can simply create a **Java FX projects** in any IDE (Eclipse and IntelliJ are tested and highly recommended) and copy the files from SRC folder to your project.

****

* This program is using JSOUP library to web crawl the results from google search, you need to download JSOUP whether in Marven or I have downloaded the library for you, simply **click the file -> Project Structure ->Project Settings ->Libraries ->click the plus symbol and select the JSOUP Zip file in the same folder as the SRC files(Only if the JSOUP library is not imported yet) -> click OK -> All the needed external things are installed.**

****

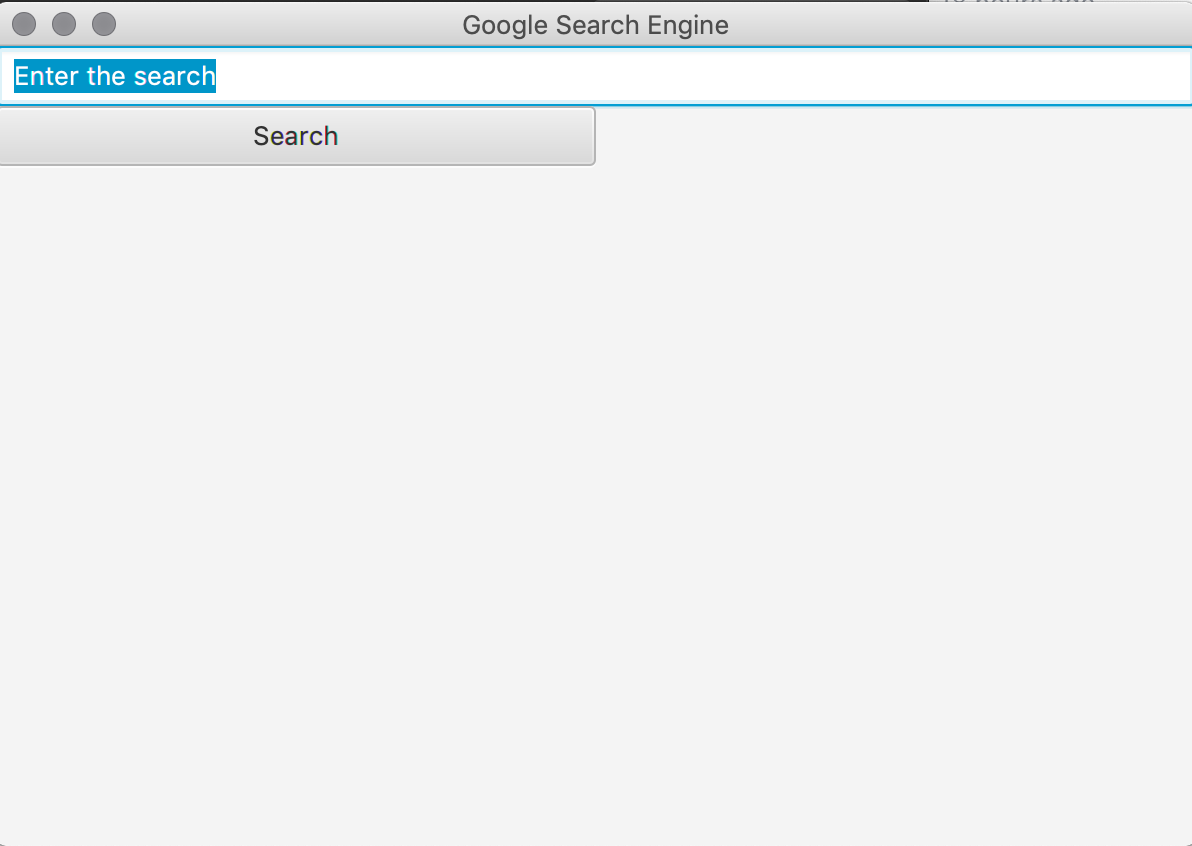
**2) Software Design**

1. Using JavaFX to create the UI part of this program
2. Using JSOUP library to web crawl the searches from google search
3. Create a search Class called Node.java that store the variables including title, URL, position, 4 factors of PageRank, total PageRank and able to set them and get them.
4. Create a Heap Tree file that called Heap.java can mainly compare the PageRank for each search result and build a heap tree for it
5. Create another Class called Unique.java that store the searches only by the keyword and the count (how many times the user searches the same keyword).
6. Create a Class that able to sort the count for the Unique.java using insertion sort, because we only display the top 10 unique search and sort it in place would be better, less memory and less time.

7. By using the JavaFX, create three main scenes, that can show the 30 results, the ten highest PageRank searches and the ten unique searches.

**3) Implementation**

1. **Main scene (Scene 1)**

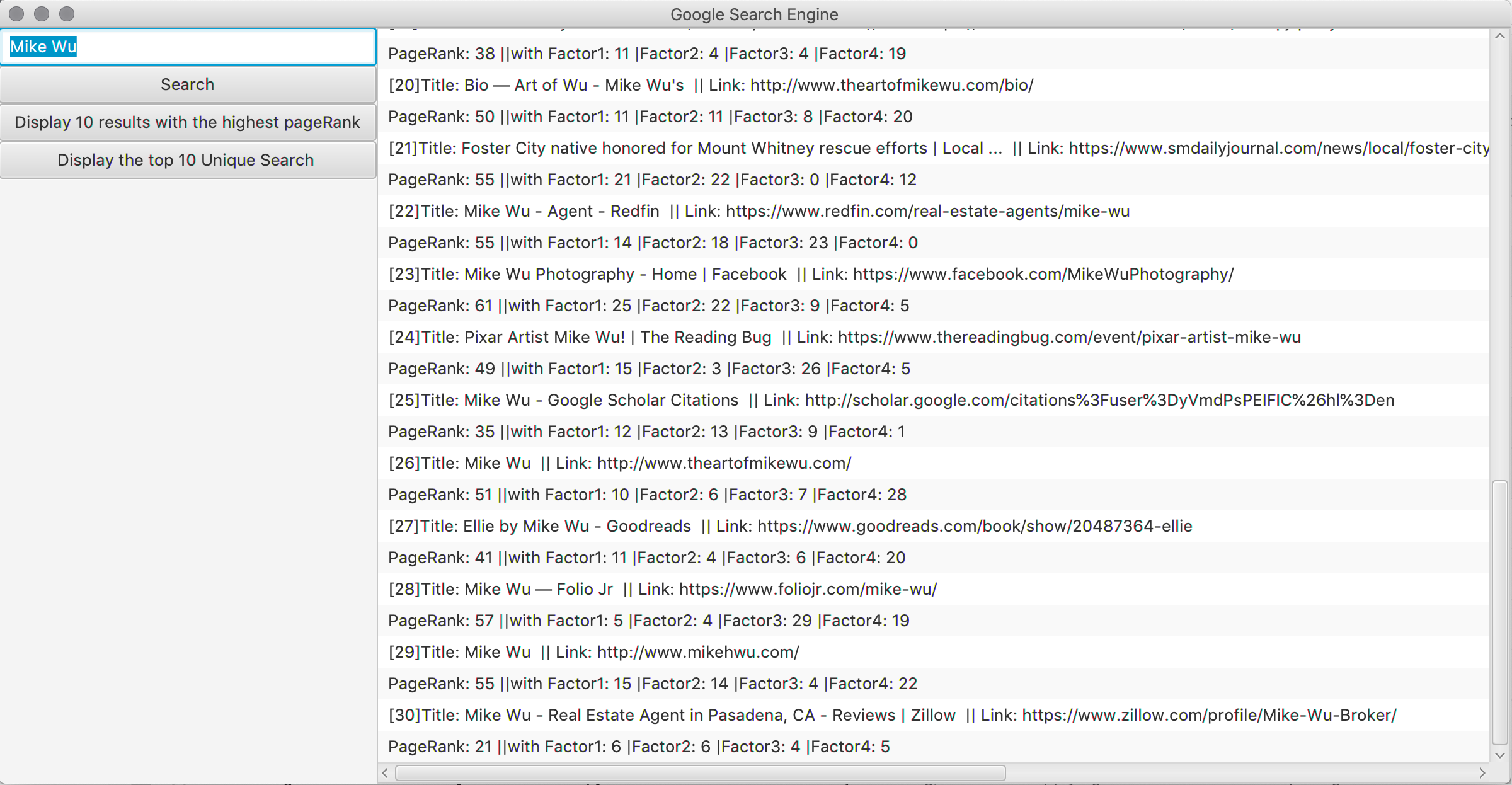
****

Text Field: Enter the keyword that you want to search

Search Button: have the **setOnAction** event, when it clicked, getting the Keyword (String) from the Text Field Above, and calling the web crawl function by using **JSOUP library,** getting 30 results from google search, and store each’s properties into an object called **Node as ArrayList called nodeList,** and changing this **ArrayList into a** **heap tree** and doing the **Heap-build-max** and able to display them in second scene.

1. **Display all 30 results in second scene (scene 2)**

****

****

The Second Scene have the left view and right view, left View contain the Text Field and Search Button that you research whatever you want, and additional two buttons:

1)Display 10 results with the **highest PageRank from highest to slowest** in order, it also have the **setOnAction** event, when it clicked, it creates a another ArrayList of the object “Node” called **displayList**, extract the Node with the maximum PageRank score 10 times from **nodeList** using **heap-extract-max method** from the and put them into the new ArrayList, and display them from highest to slowest as it shown below as Scene 3.

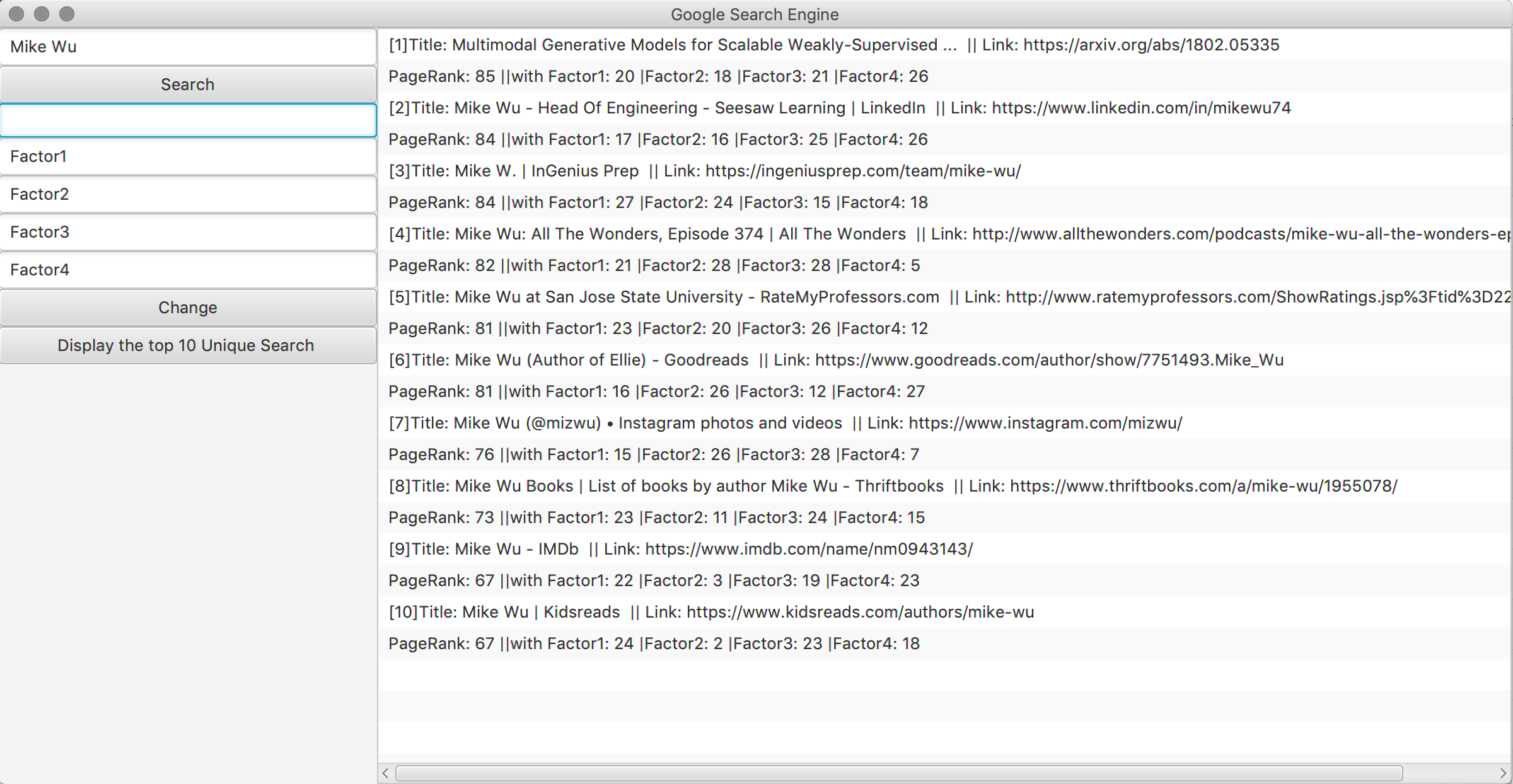
2)Display the top Unique Search, I would talk about this in Scene 6.

1. **Display the 10 top results with the highest PageRank**

****

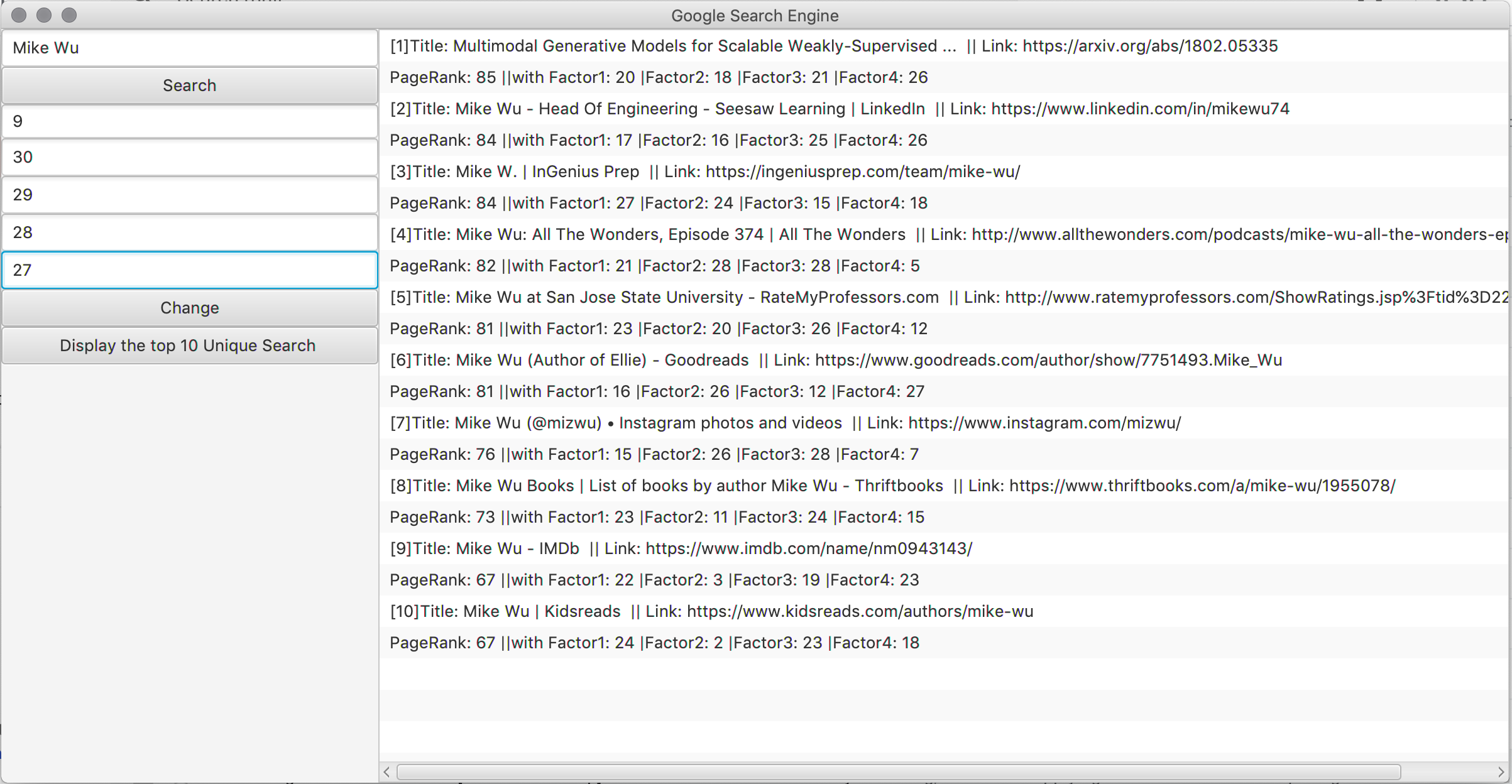
As you noticed, we have a new button called “Setting the PageRank”, it comes with the **setOnAction** event that when we clicked, it shows us the Scene 4

1. **Setting the PageRank Manually Scene (Scene 4)**

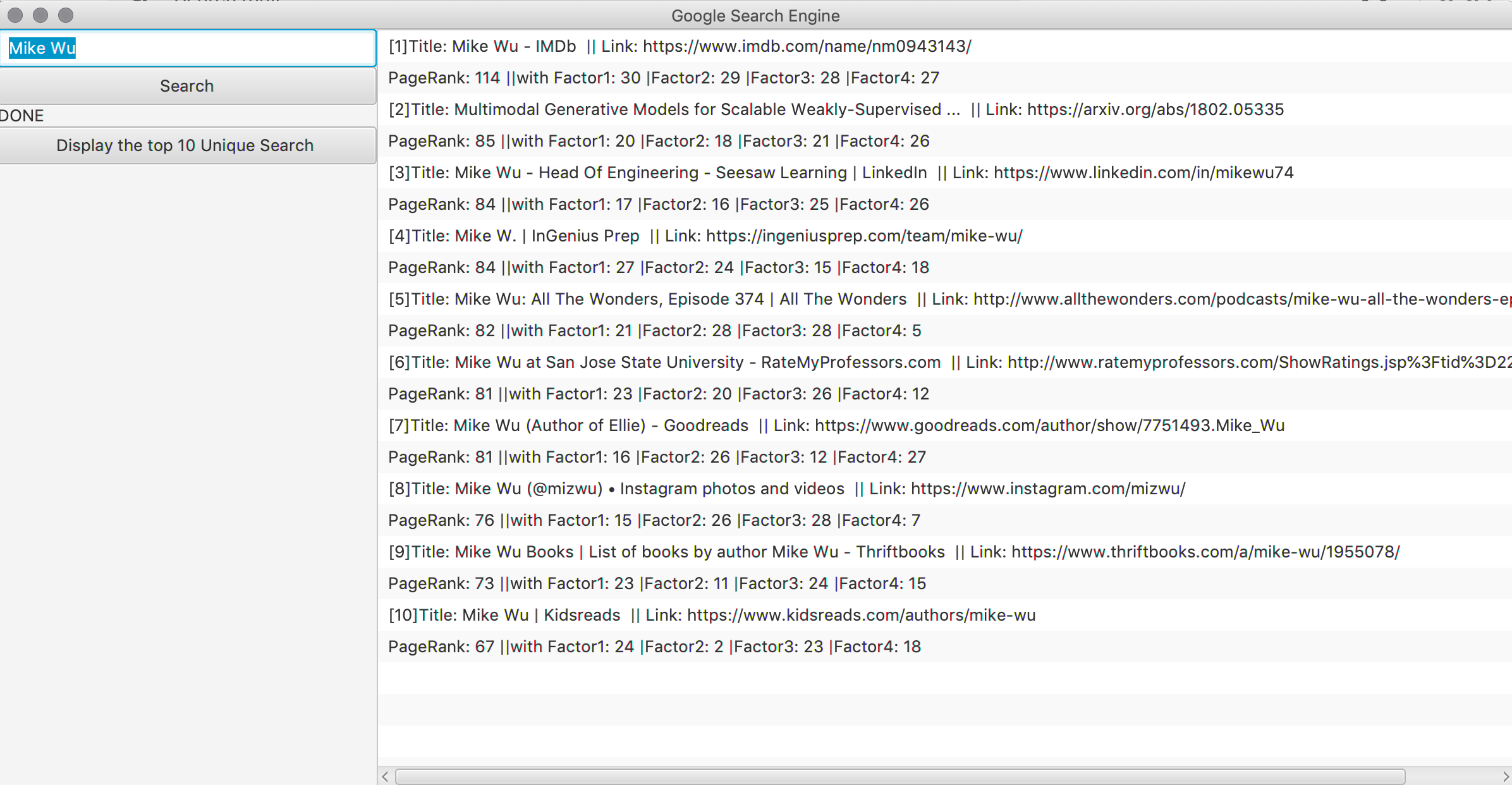


It comes with 5 Text Fields, the first one is choosing the position (as you can see the number in front of the result on the right side), and the four other Text Fields are the 4 factors that we can manually input.

For example, Let’s choose position 9, and setting the 4 factors as 30, 29, 28, 27.

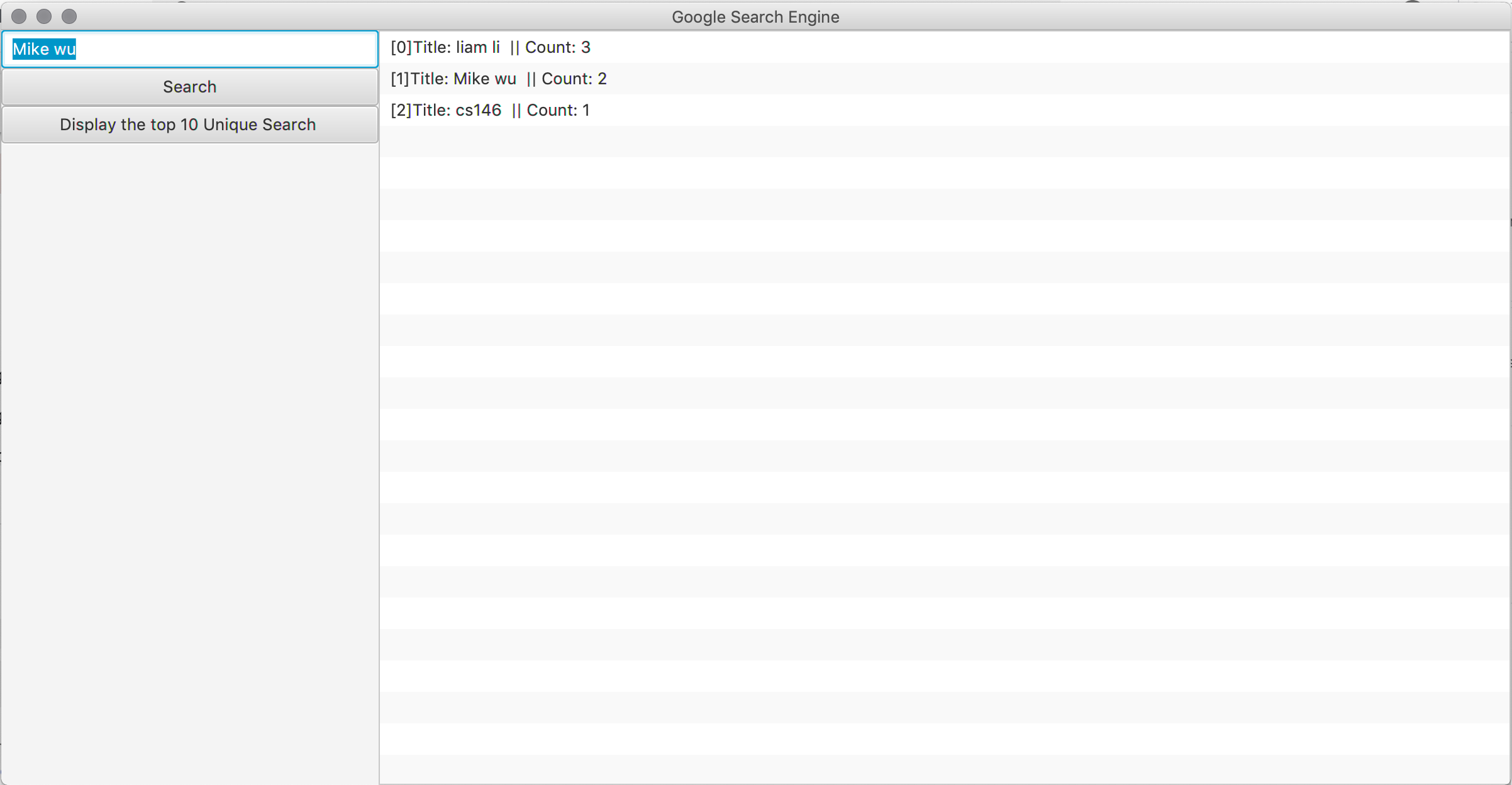


After this, we have a “Change” button, that come with the **setOnAction** event, when it clicked, the change button accesses the 5 Text Fields and changed the 4 factors of the selected position of result, as we can see below. Using the **Heap-Increase-Key and Heap-Sort method** to resort the ArrayList after setting the 4 factors of the PageRank.



Let’s look at the right side of the view, the first result was previously at position 9, after increase the 4 factors, it moves to the first position which have more chances to explore it to the Internet. Now, let’s talk about what happens when the “Display the top 10 Unique Search” Button clicked.

1. **Display the top 10 unique searches (Scene 5)**

****

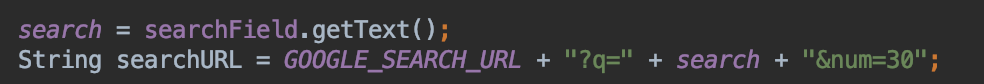
I search my name “liam li” for 3 times, “Mike wu” for 2 times and “cs146” only one time, these keywords and count are stored automatically every time I clicked the “search” button into objects called “Unique”. When I clicked the “Display the top 10 Unique Search”, it added the object into a new ArrayList called “uniqueArrayList”, and sort them from highest count to lowest count using insertion sort. Because insertion sort is less memory, sorted in place.

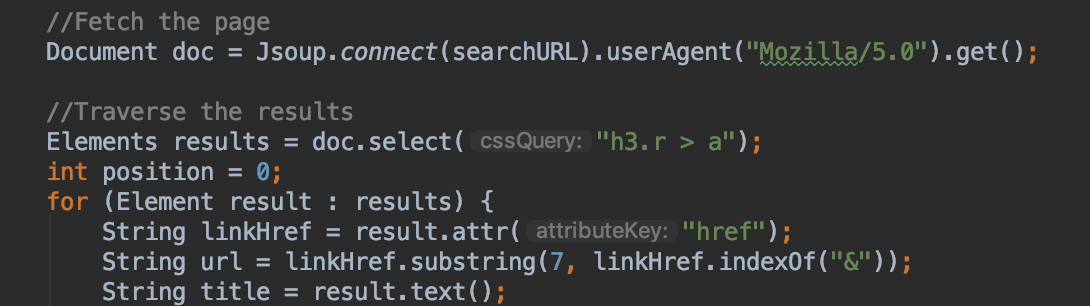
These 5 main Scenes showed my implementations of this assignment and **using priority queue and heap tree.**

**4) Features/Functions/Classes**

**Main.java**

* Web crawler: Using JSOUP to connect the URL and setting the userAgent, from the div of the google search website, we know the search result is contained in the div of h3, so we can just only get what we want from the div of h3.
* UI components: ListView, Button, TestField

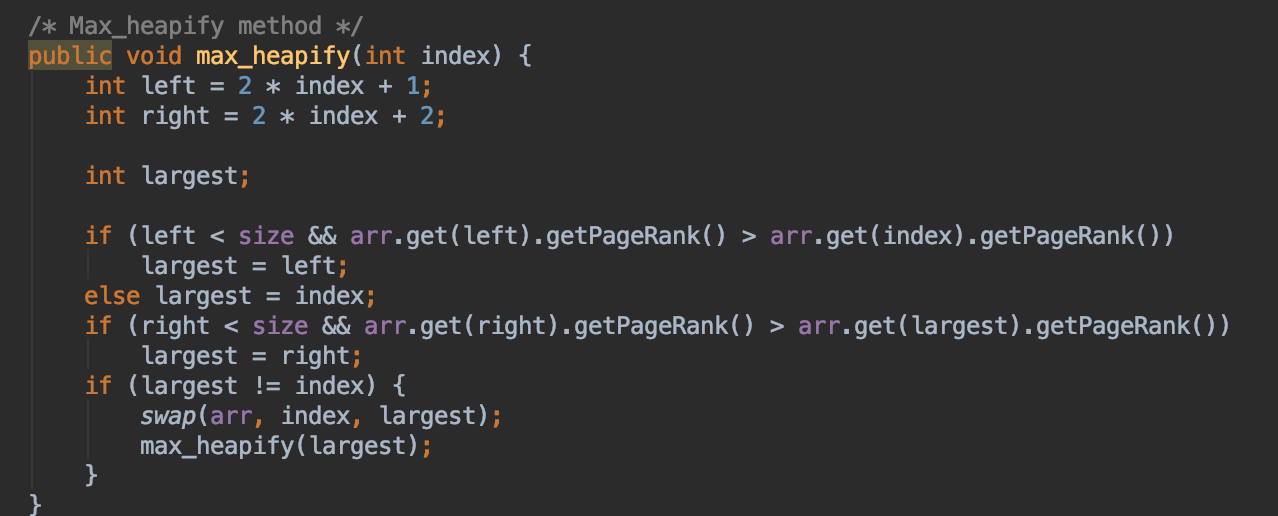




**Heap.java**

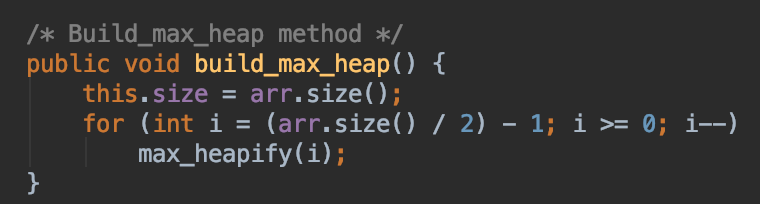
* **Heap-Heapify**

This method is the main method for this project, having a max-heapify is important for heap structure, it compared the left and the parent, then it compared the right and the parent, when it has the largest, then swap. It’s called one heapify.

****

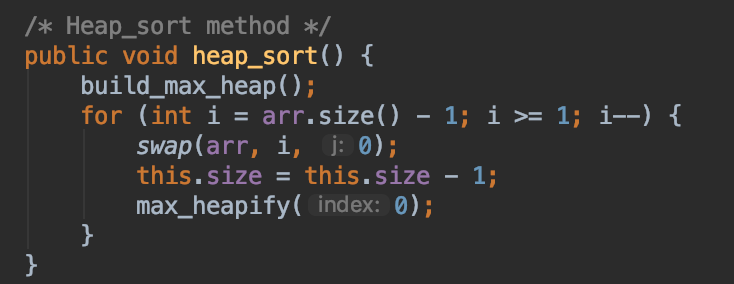
* **Build-max-heap**

This method is to build max-heap which the tree has the property that **parents** are always greater than **children** by using multiple max-heapify.

****

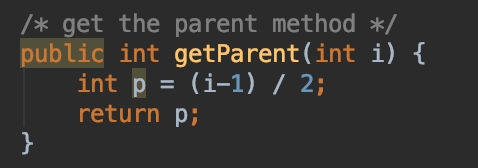
* **Heap-sort**

This method is sorting algorithm that sort the data from lowest to highest, by using build-max-heap first to get a max tree, then use max-heapify multiple times in order to sort the data by swapping the highest number/data to every end. After n times, the data sorted.

****

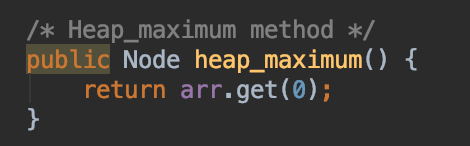
* **getParent**

This method is simply getting the parent Node of index.

****

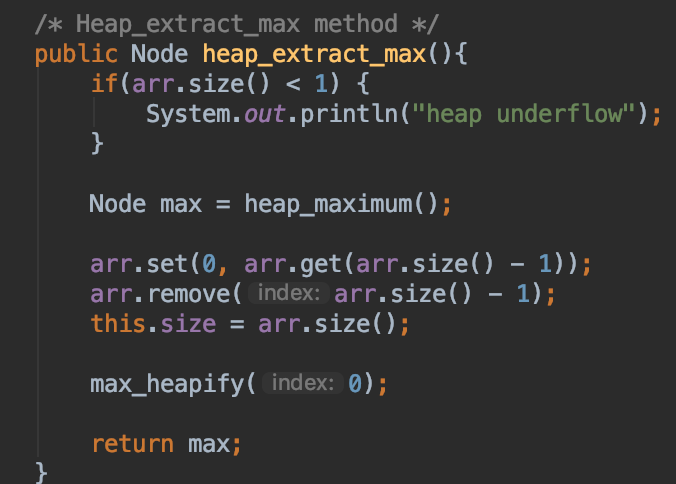
* **Heap-maximum**

This method is get the root of the tree, which often called after building the max-heap first, because the root would be the highest.

****

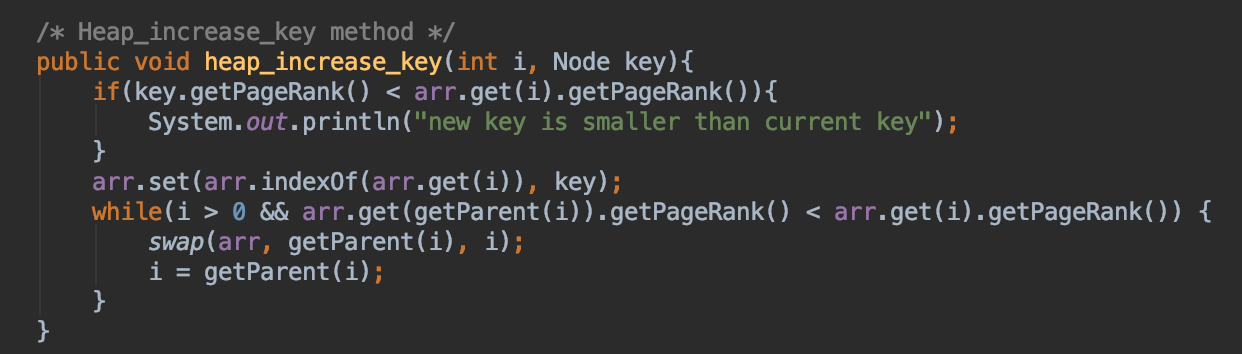
* **Heap-extract-max**

This method is taking out/return the max node but also reset the tree as the size decrease one as the new tree. Usually uses when the current max Node is used for other purpose.

****

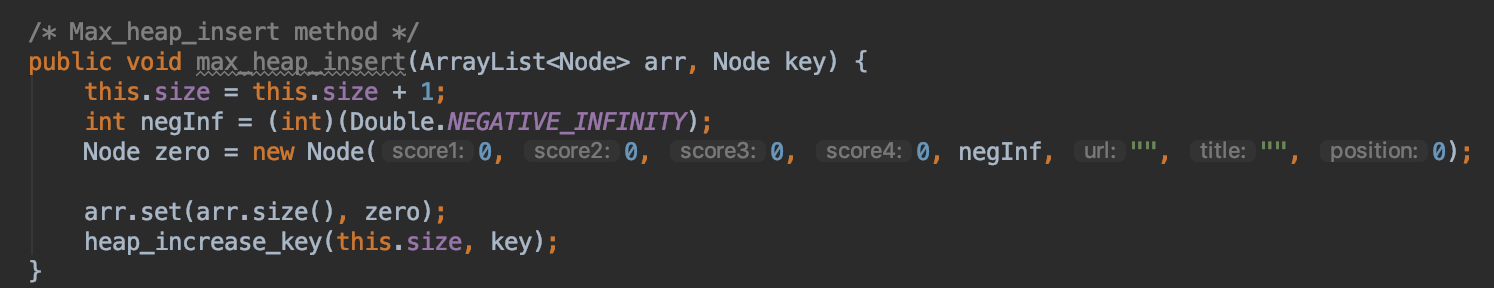
* **Heap-increase-key**

This method is called when you want to increase the number/value of chosen Node, the first if statement determent if the new number/value of the chosen Node is whether smaller or larger, if it larger and larger than the parent, it would swap with the parent, the parent of parent in order to have the max-heap tree still.

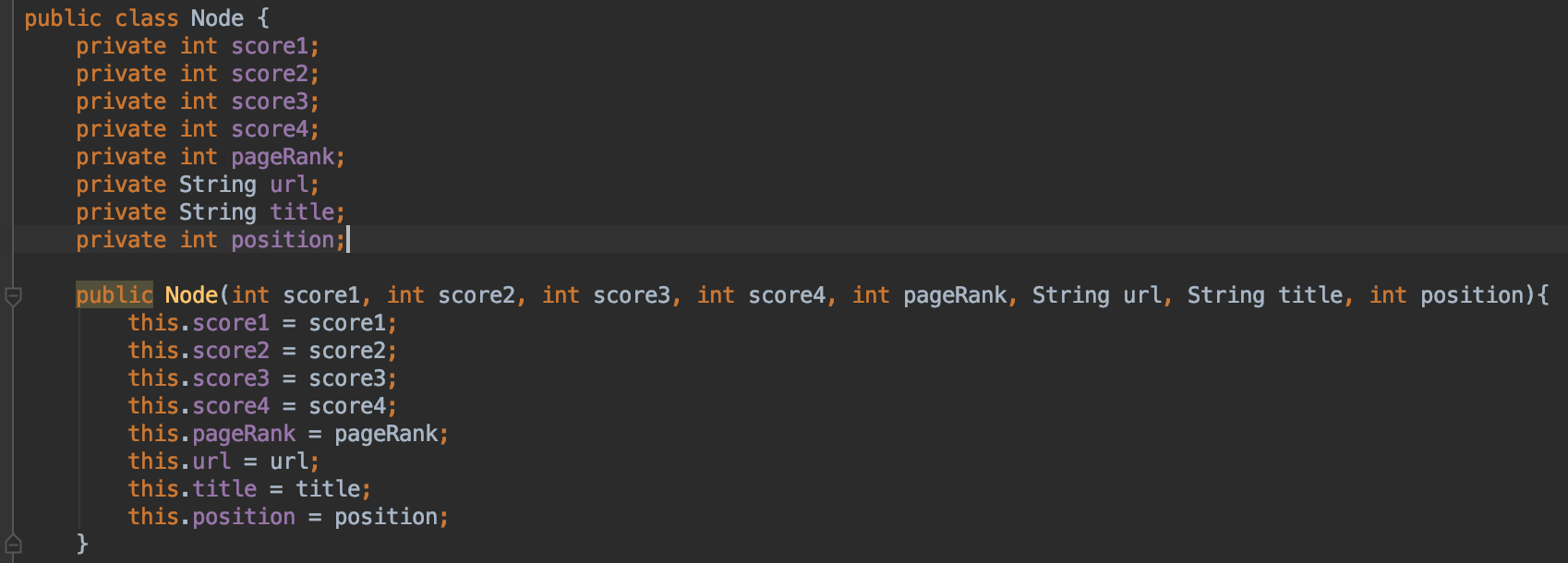
****

* **Max-heap-insert**

This method is called when you want to insert a new Node, it increase the size and also set the Node as very end first, and called the heap-increase-key method to put the inserted Node to the right position.

****

**Node.java**

****

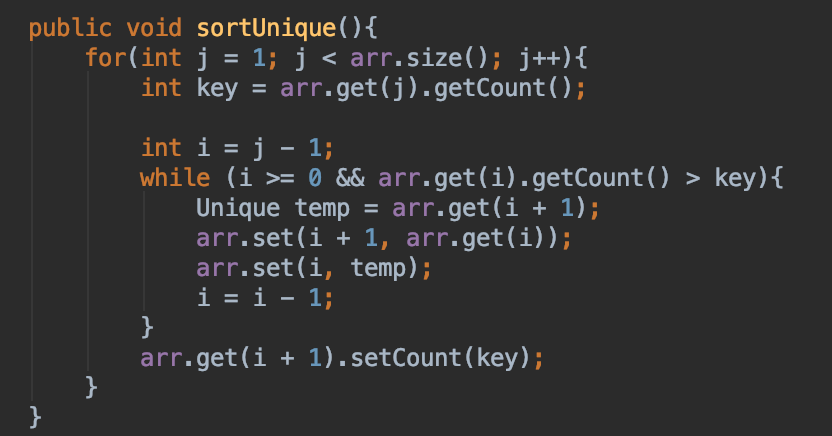
Contains all variables by one search result and their position, also have all the getters and setters.

**Unique.java**

****

Contains one keyword String and count Integer, used for storing the Unique Search, if same keyword, count plus one, if new keyword, create a Unique object and count as one.

**UniqueSort.java**

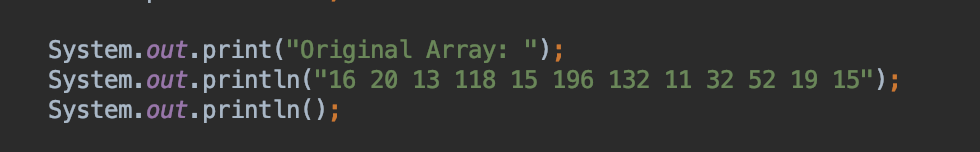
****

Contains the sort function to sort the count times for Unique Keyword using the insertion sort as the Code from Homework 1/ Textbook.

**5) Testing**

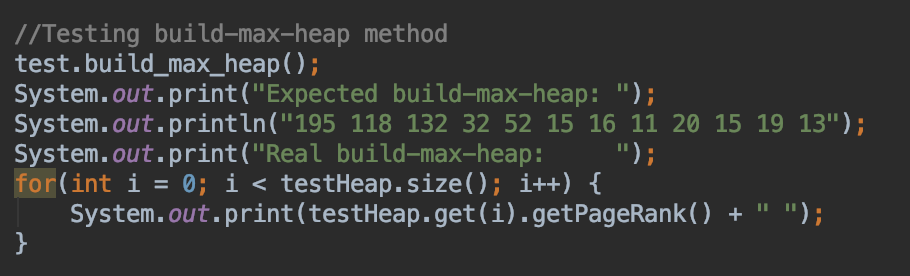
**Because the UI have delay time of let the user input, I tested each function and calculated the Time complexity of using custom case**

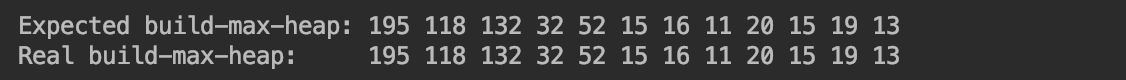
**For Custom Case, I manully Insert 12 Nodes with setted pageRank Score:**



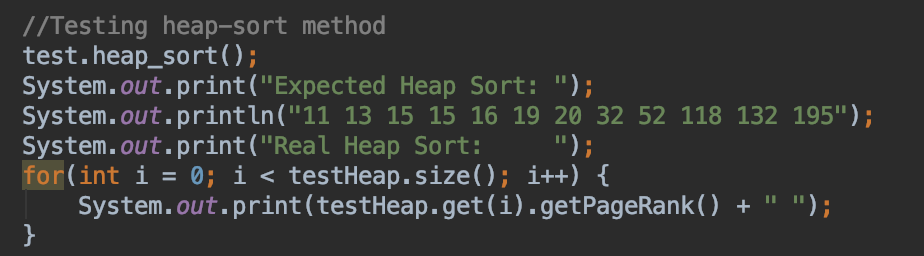


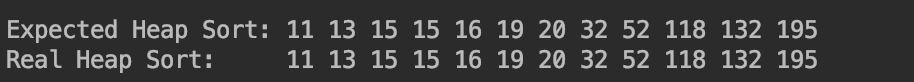
* **Testing Build-Max-Heap**

****

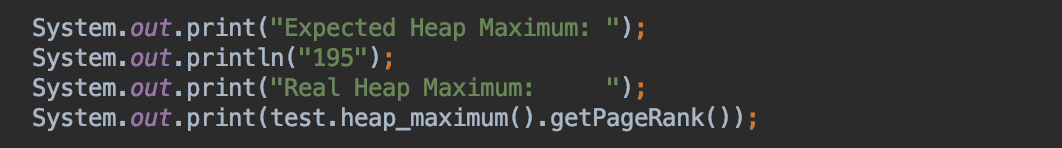


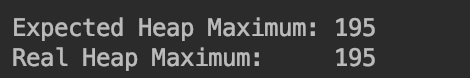
* **Testing Heap-Sort**



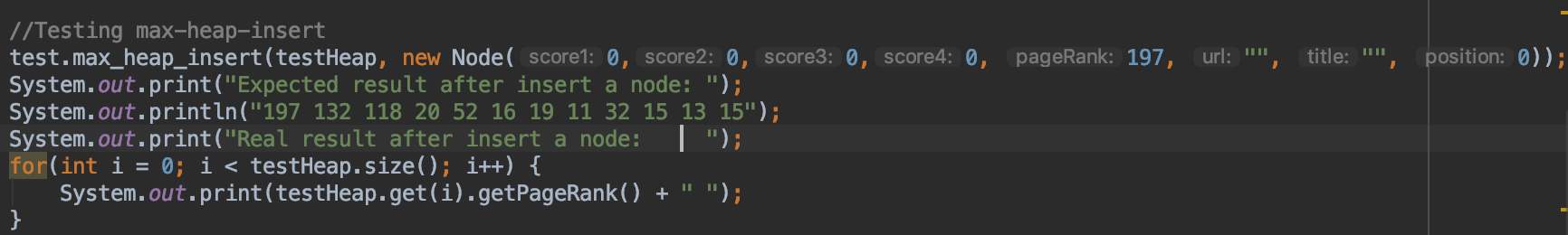


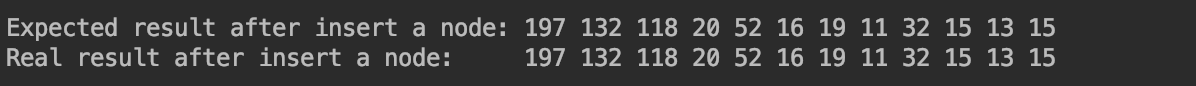
* **Testing Heap-Maximum**

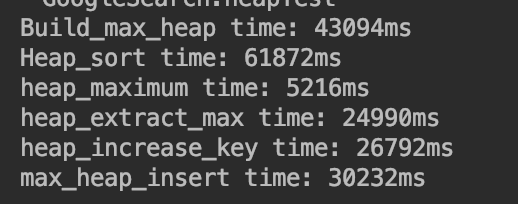




* **Testing Heap-Insert**



  
**Time Complexity of each functions:**

****

**6) Problems**

* First time using Java FX, encountered many difficulties including passing values, changing values, storing values by listening event and clicking event.
* Unable to display the time complexity on the UI, would try to implement it in the future
* Implement ArrayList of objects needed to slightly change the code from the textbook, but eventually pretty much match what the pseudo code.

**7) Learnings**

* Be able to figure out the bugs in heap implementations.
* Understand the advantages of priority queue – Dynamic data insert and extract.
* Understand more the JavaFX and able to set up basic UI.
* First time writing the programming report, so it helps me understand more what I did, how I did and what should I do to make a better version of the assignment.