Jeffrey Callender

Text Analyzer

For the text analyzer project one of the most difficult part for me was figuring out how to parse the html tag and only getting the p tag text that we were supposed to get. Of several options I could have done included using java regex API. The regex would allow me to search for specific string and filter through text that only had the poem which in this case would be the p tag within the body. I however, decided that this would not be enough and wanted to see if there was anything else within the java libraries that would allow me to easily navigate through html tags without having to worry about parsing through unnecessary tag.

jsoup was one of the essential extended libraries that I use for this project. The jsoup is a java library for working with real HTML. It provides for very convenient API for fetching URL and extracting and manipulating data using HTML5 DOM method and CSS selectors. Since I was using eclipse I had to make sure I downloaded this extended libraries and added it to my the project folder I was working on. Using the jsoup allowed me to selected which tag I wanted so if I wanted the p tag all I had to do was select the p tag my calling doc.select(“p”) and it would return the string within the selected tag. To connect to the URL I use jsoup as well by using their connect() to pass in the URL string and get() to connect to the URL.

After passing the URL by using a scanner object and prompt the user for a URL I start to strategize how to store the information that I will receive. For the next part I decided to use an array list of type string to store the entire of the p tag text (the entirety of the raven poem). By using an enhance for loop I was able to add all the text from the poem and only the poem into one big item in my array list. Once I had my poem I decided to call the toString method to store the array list a string variable called str in order to parse the string of any non-alphabetical characters. I then proceeded to call the replaceAll method on str(variable that is storing the poem) and got rid of any character that was not a alphabetical one. I also made sure that all the character was lower case by using toLowerCase() method to ensure that there was no problem words that are the same but are characterize as different because one letter is upper case.

Now that I have the fully sparse string with nothing put words and spaces I made a array called words to put the string back into. I this is in order to be able to traverser the string again. I declare a HashMap of string to store all the words of the poem. The next step was making a enhance for loop that goes through all the words in the array words (currently holds raven poem) and adds them to the HashMap. In this for loop is where I do the text analyzing part. The basic logic of the loop is that it goes through the array words and put then in the HashMap. If the words are already their then it update the count variable and adds the words in the HashMap. If the word does not appear then it simply adds the word and assigned a value of one to that key.

Once all that is done the next step is to print out the result in an descending order. They are many ways in which I could have done this, but I choose the method of using an Link HashMap to help print out the words. I chose this because the link hash map keeps the ordered in which words were in sorted so I don’t have to worry about losing that, I then simply use the comparingByValue method and using the Comparator.rerverseOrder to get the entries of words in descending order. After that I printed out the content of the words.