HW5 A Machine Learning Project Using Linked Lists

Motivation:
In October each year, the Nobel Prize Committee in Sweden announces the winners of the prize in various fields in Science and literature. In celebration the event this year, we will do some silly work based on a song written by Bob Dylan, who won the prize in literature in 2016. The song is entitled "Blowing in the Wind", arguably his most famous one.
In case you are not familiar with the song, here is the link on youtube: https://www.youtube.com/watch?v=3l4nVByCL44
I have its lyrics cleaned up a little so you have something like the following:

how many roads must a man walk down before you call him a man how many seas must a white dove sail before she sleeps in the sand yes and how many times must the cannon balls fly before they're forever banned the answer my friend is blowing in the wind the answer is blowing in the wind yes and how many years can a mountain exist before it is washed to the sea yes and how many years can some people exist before they're allowed to be free yes and how many times can a man turn his head and pretend that he just doesn't see the answer my friend is blowing in the wind the answer is blowing in the wind yes and how many times must a man look up before he can see the sky yes and how many ears must one man have before he can hear people cry yes and how many deaths will it take 'till he knows that too many people have died the answer my friend is blowing in the wind the answer is blowing in the wind

1. First, you need to store the above version of the lyrics into a text file called "blowingInTheWind.txt". You will store it in a folder called "data" under your project folder.

2. You are going to use Linked Lists (**use Java API**!) to store all the words in the text file and keep track of which word following which word and how frequently. Here is a sketch of what needs to be stored:

Keyword	Followed by Words
"how"	"many", "many", "many", "many", "many", "many", "many", "many", "many"
"many"	"roads", "seas", "times", "years", years", "times", times", "ears", "deaths", "people"
"roads"	"must"
"must"	"a", "a", "the", "a", "one"
etc.	

Keywords are unique and each of which is stored in a **parent** Link in a parent linked list. Each parent link also contains a baby linked list, which stores baby links holding the next word following that unique keyword in the parent link, one for each occurrence throughout the entire text. Once a parent link is created for a keyword, no new parent link shall be created for that keyword again. For example, the first time when "how" is encountered, a parent link will be created for it and the word following it ("many" in this case) will be created in a baby link inserted into the newly created baby linked list for that key word. After that, when the word "how" is encountered again, no new master link for it shall be created again. Instead, the word that follows it ("many", again in this case) will be simply inserted into its existing baby linked list as another baby link. So by now, the parent link contains the keyword "how" and a baby linked list that contains two baby links (both contain the word "many" so far).

3. Upon completion, you will have the following:
   a. A parent linked list. The list is composed of parent links.
   b. A bunch of parent links, each of which contains a unique word from the text AND a baby linked list.
   c. Each baby linked list is composed of all the word occurrences found following the unique word in the said parent link.

4. Show time! Your program will present a JavaFX GUI that allows the user to enter a keyword from the text and a number. In response, your program will produce a paragraph in a text area. The paragraph will start with the word and contains the number of words as entered by the user. Starting from the second word, every word is a randomly selected word from the baby linked list associated with the parent link that contains the word immediately in front of it.

5. In addition to displaying the paragraph in a **Text Area**, you need also save it into a text file called "**output.txt**", which in turn is automatically placed into a folder called **output** under the project folder. The GUI also needs to have a menu bar with a file chooser that allows the user to choose any text file on your computer to use in the future.

6. Finally, write a document called README.txt in which you will discuss the Big O time complexity level of the various stages of the operation.