CS102 – Algorithms and Programming II
Programming Assignment 2
Fall 2023

ATTENTION:

- Compress all of the Java program source files (.java) files into a single zip file.
- The name of the zip file should follow the below convention:
  CS102_Sec1_Asgn2_YourSurname_YourName.zip
- Replace the variables "Sec1", "YourSurname" and "YourName" with your actual section, surname, and name.
- You may ask questions on Moodle and during your section's lab.
- Upload the above zip file to Moodle by the deadline (if not, significant points will be taken off). You will get a chance to update and improve your solution by consulting with the TAs and tutors during your section's lab.

GRADING WARNING:

- Please read the grading criteria provided on Moodle. The work must be done individually. Code sharing is strictly forbidden. We are using sophisticated tools to check the code similarities. The Honor Code specifies what you can and cannot do. Breaking the rules will result in disciplinary action.

Predicting The Next Word

You will implement a Java program that generates random stories based on the word probabilities analyzed from a given text document. You will input a text document, "sentences.txt"; you may use the text file included for this assignment. It is guaranteed that this file only includes lowercase English letters, with each sentence appearing on a new line. You should read this text line by line and split the words according to the space characters. For each word, you are going to store what other words can come after it. Suppose the following sentence:

i told him i did not know his mate bill

From this sequence, we observe the following order relations:

i > told
told > him
him > i
i > did
did > not
not > know
know > his
his > mate
mate > bill

Based on this observation, we can say that the word *did* can be followed by the word *not*, but not by the word *mate*. We would like to find and store all such relationships. This requires having word objects that can keep references to other word objects. Implement a **Word** class to represent one word with its *String text*. Keep an *ArrayList<Word> canBeFollowedBy* in this class to reference the other Word objects we can encounter after this word. Include a **void addFollowedBy (Word w)** method in this class to add a new word into *canBeFollowedBy* ArrayList. Note that you will also implement any required constructors and set, get, or additional methods to make your code functional.

You can use special Word objects to indicate the beginning and ending of sentences. You will append these special words to the beginning and end of the sentences you process. For example, “hello my name is joe” will become “<START> hello my name is joe <END>”, then we have the following additional relations:

<START> > hello
joe > <END>

Implement a **WordBag** class to keep your Word objects in it using an *ArrayList<Word> allWords*. A **void processSentence (String sentence)** method in this class should process the given String by appending the starting and ending words and splitting the sentence according to space characters. For each pair of words x followed by y, you should add y to the *canBeFollowedBy* ArrayList of x. This requires you to find or create new Word objects. In your WordBag class, implement a **Word findOrCreate(String wordText)** method that first checks if the given wordText is included in your allWords ArrayList. If it already exists, return that Word; if not, then create a new Word object using the given *wordText* and add it to your *allWords* ArrayList. You should still return the newly created word. Use this **findOrCreate** method in your **processSentence** method to find the Word objects of the split sentence.

Include a **String generateSentence(int softLimit, int hardLimit)** method in your WordBag class to generate a sentence using the relations between the words. You should start with <START> word, then, among the words that can follow <START>, choose one randomly. You will choose the next word each time using a random word from the *canBeFollowedBy* ArrayList of the last word. The input *softLimit* determines when the algorithm tries to end the sentence. Suppose you are given a *softLimit* of 4, then after generating a sentence of 4 words, you try to end the sentence if possible. That is to say, you check if the last word you have can be followed by <END>; if not, you continue with a different random word from the *canBeFollowedBy* ArrayList. You continue with new words until you find a word that can be followed by <END> or you reach at the *hardLimit*. The generated sentences will have a maximum word count between softLimit and hardLimit. In case the next word that you choose randomly is <END>, you can still end the sentence; in this case, it will have word count less than softLimit. When you reach the hardLimit, the sentence ends whether the last word can be an ending word or not. You can include methods to check if a word is an ending word and to get a random word that can follow the current word in your **Word** class.

Also include a **void writeToTextFile(String outputName, int sentenceCount, int softLimit, int hardLimit)** method in your WordBag class to generate and write sentences to an output text file, each sentence occurring on a new line. Your output text file should have *sentenceCount* number of sentences in it. You will use **generateSentence** method to generate the sentences with the given hard and soft limits.

Implement a console application in your main method that creates a WordBag and processes each line of the sentences.txt file. Then, the user can choose one of the following options:

1. Generate one sentence: Get soft and hard limits from the user, and generate and print the corresponding sentence on the console.
2. Generate and write to a text file: Get a name for the output text file and the hard and soft limits for the sentences. You should also get the number of sentences the user wants to generate. You are going to generate and write the sentences on each line of the output text file.
3. Exit program.

A sample console output is given below:

```
Input file "sentences.txt" processed.

1. Generate Sentence
2. Output Sentences to Text File
3. Exit
Please choose an option: 1

Soft Limit: 4
Hard Limit: 8
Sentence: i must we must be happy again

1. Generate Sentence
2. Output Sentences to Text File
3. Exit
Please choose an option: 1

Soft Limit: 2
Hard Limit: 3
Sentence: look like to

1. Generate Sentence
2. Output Sentences to Text File
3. Exit
Please choose an option: 1

Soft Limit: 15
Hard Limit: 30
Sentence: we had only rest of our way to see the to this and tell you
decided voice that the far yet begun to come at good but i felt that

1. Generate Sentence
2. Output Sentences to Text File
3. Exit
Please choose an option: 2

File Name: test1
Sentence Count: 10
Soft Limit: 4
```

```
Hard Limit: 9

Saved to test1.txt

1. Generate Sentence
2. Output Sentences to Text File
3. Exit
Please choose an option: 3

Goodbye!
```

Content of test1.txt after running the program:

```
found it she had known her heart of only
harkers tongue as tidy little daughter
she said herr sesemanns kind to a sort of
oh you will be in the room and laid what
we all untouched save him by now making a
i could still and what is to go back
the man who is it
what i should go mad or two had to
entered originally
he can meet
```

Preliminary Submission: You will submit an early version of your solution before the final submission. This version should at least include the following:

- Word class should be finished.
- Your main method should be able to read the given text line by line, and you should be able to split the sentences word by word.

You will have time to complete your solution after you submit your preliminary solution. You can consult the TAs and tutors during the lab. Do not forget to make your final submission at the end.

Even if you finish the assignment in the preliminary submission, you should submit for the final submission on Moodle.

Not completing the preliminary submission on time results in 50% reduction of this assignment's final grade.