Project One: Poetry Writer

In this project you will practice implementing linked list data structures by writing poetry. More specifically, given a sample text, your PoetryDriver class will use your PoetryWriter class to generate a "random" text from it. The generated "random" text must follow some rules described below which are based on patterns in the text that is read and which will cause the output to be somewhat poetry-like.

If you did PoetryWriter as a CS 151 Honors Contract in Java, then you must redo it in Python. If you did not do PoetryWriter in CS 151, then you can do it in Java or Python. If you complete the project in Python, you may use either the PyPy platform (pypy.org) or the python compiler on agora. Please include a readme file indicating which language you used and (if implemented in Python) which compiler. You must use linked lists, not arrays, as your data structure for this project. Do not use the ArrayList and LinkedList classes in the Java API. Instead you are writing your own MainList, NextList, and Node classes that implement kinds of linked lists that are unique to this project. This is true if you are using Python also. For example, each node in the MainList has four fields. Your implementation must look like Figure 1 below.

User-Visible Requirements

Your implementation must satisfy the following requirements that are visible to the user.

1. Your program should be called from the command line as illustrated by the following example.

   ```
   java PoetryDriver input.txt 1 2 3
   ```

   will read in the text in the file input.txt and display on the console a poem that has one stanza, two lines per stanza, and three words per line.

2. The output must be organized into stanzas with the number of stanzas, the number of lines per stanza, and the number of words per line being with those values specified as command-line arguments when your PoetryDriver class is called.

3. Every word in the output text must occur in the input text. The first word in the output text is probabilistically picked from all the words in the input text based on the frequency of that word appearing in the input text. For example, if ten percent of all the word occurrences in the input text are occurrences of the word "red", then there is a ten percent probability that the first word in the output text is the word "red".

4. Every pair of words in the output text must occur in the input text. The second word in each pair of words in the output text is probabilistically picked from all the

words in the input text that occur after the first word in the pair based on the frequency of that second word appearing immediately after that first word in the input text. For example, if ten percent of the word occurrences that immediately follow occurrences of the word “red” in the input text are occurrences of the word “blue”, then there is a ten percent probability that in generating your output text after you output the word “red”, the next word you output will be “blue”.

5. Your program should treat punctuation as words. For example, “red.” is the word “red” followed by the word “.” and “, the” is the word “,” followed by the word “the”.

6. The last word (usually a period) of the text by definition has no following word. How should this special case be handled? View the first word of the text as following the last word of the text for the purposes of computing the frequency of pairs of words.

Implementation Requirements

Your implementation must satisfy the following requirements.

1. Your data structure consists of one linked list named MainList and for each node in MainList, a linked list named NextList. You will also have a MainNode class and a NextNode class. Since a field in the NextNode class points to a MainNode object, the MainNode class cannot be an inner class of the MainList class. Having both the MainNode class and the NextNode class be top level classes is fine.

2. There is one node in the MainList for each different word that appears in the text. So if a word occurs multiple times in the text, there is still only one node for that word in the MainList. In the example poem, the word are has only one node for it in the MainList.

3. For each node in the MainList (that is, for each different word X in the text) there is a NextList which has one node for each different word Y that immediately follows word X at least once in the text. A node in a NextList has several fields including one that counts the number of times that Y occurs immediately after X. This information will be used in the following way: when the program writes the word X, the next word that it prints is a “randomly” selected word from the words in X’s NextList where it is more likely to print a word that occured in the input text more frequently after X.

4. A node in the MainList has the following fields:

   (a) a field for the String of the word

   (b) a field that is a pointer to the next node in the MainList

(c) a field that is a pointer to the NextList that is associated with this word. The NextList for a word has one node for each word that occurs immediately after this word.

(d) a field which is how many times this word appears in the text. In the example, the node for the word are has a value of 3 in this field since the word are occurs three times in the text.

5. A node in a NextList has the following fields:

   (a) a field for the String of the word

   (b) a field for a pointer to the node for the current word in the MainList

   (c) a field for the number of times this word appears immediately after the word in the MainList that this NextList is associated with

   (d) a field that is a pointer to the next node in this NextList

The data structure that you must create is introduced in the context of the following example.

An Example

If your program reads the following poem:

Roses are red,
violets are blue,
sugar is sweet,
and so are you.

then it should construct a data structure a detail of which is shown in Figure 1. In the Figure 1 diagram, there are four instances of one linked class (say MainList) and eight instances of another linked class (say NextList). Each word in the poem will appear in exactly one instance of the MainList class. This instance will have a reference to a linked list (a kind of association list) called NextList). The MainList object knows how many times its word appears in the poem. Each NextList object keeps track of how many times the word it represents appears immediately after the MainList word that points to it.

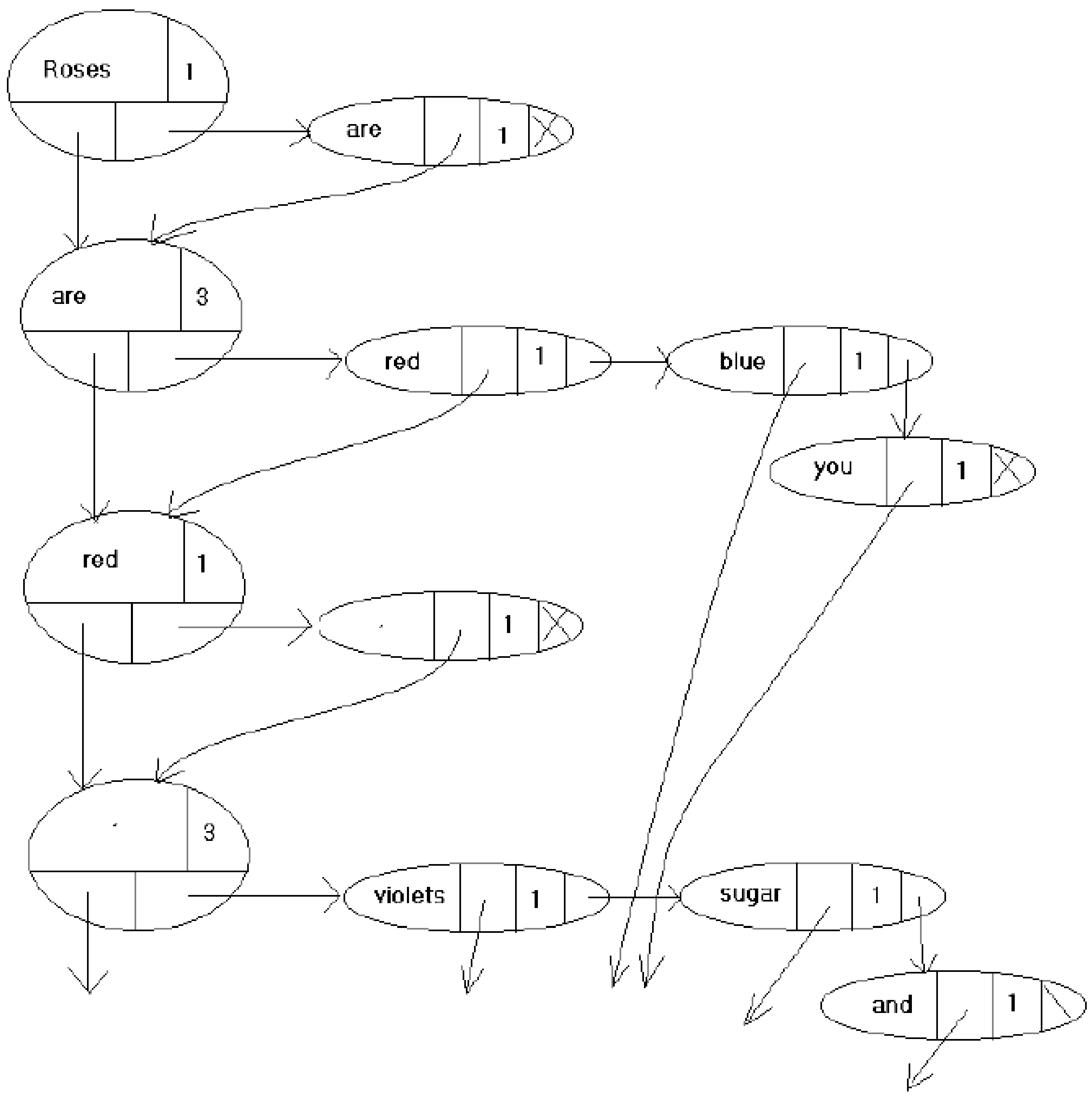

Figure 1: An example of the linked list structure.

For instance, the words “red”, “blue”, and “you” each appear one time immediately after the word “are”. The word “,” occurs one time immediately after the word “red” but it occurs three times in the poem. Notice that each NextList instance has a reference to the MainList instance containing the same word. This helps when writing the poem.

Missing from this diagram are the arrows that would be pointing up from nodes lower down to the nodes pictured. See Blackboard for a pdf of the example we developed in class.

Other Comments

1. Reading words from the text file containing the original poem is straightforward using the Scanner and file input/output. Check with me if you have any questions about how to do this.

2. On the course website for this project I have the following files

   (a) An example input poem and the data structure created from it.

   (b) Some classic texts for your program to read.

   (c) Some output generated by the program I wrote based on the poems provided.

Administrative

1. The project is due at 3 pm on Friday, the 27th of September.

2. The score is 0 if the program does not compile.

3. The project can be turned in late. Three points will be taken off for every date late (not counting weekends and holidays) up to a maximum of 20 late points. For example, if the project is turned in between 5 pm Monday and 5 pm Tuesday, it is one day late. Past the 20 point deadline, you may turn it in until Nov 1 for a maximum of half credit. If you turn it in late, you must notify me through Blackboard.

4. You may work in teams of one or two students from the class. You may talk with other students in the class about the concepts involved in doing the project, but anything involving actual code needs to just involve your team. In other words, you cannot show your team's code to students outside of your team and you cannot look at the code of another team.

5. Your solution must follow the course coding standards including

   (a) no lines longer than 80 characters

   (b) no magic numbers; use named constants

   (c) javadoc class headers, fields, constructors, and methods

(d) use // comments to document all the code in method bodies.

6. Submit your files jar file (or tar file) via handin on agora. If your jar file is called project1.jar, then the command will be:

```
handin.351.1 1 project1.jar
```

7. The original version of this project was developed by Dr. Benjamin Shults.