Words in a Sentence
CS 5004 Object Oriented Design

1. **Goals:**

   To practice representing data using protocols (set of interfaces) and lists, define operations on it and also to work with interfaces and abstract classes to refactor code. Since part of the goal of this assignment is to give you practice with understanding and implementing a linked list like the example given in class, limit yourself by not using any Collection classes provided by Java, and by using a recursive data structure (as in lecture). Do not use looping and iteration via for/while constructs (which haven't been discussed in class).

2. **In Recitation:**

   The module associated with this assignment will take place over two weeks. Instead of having two lab assignments, you will have this assignment over the course of two weeks.

   In each recitation, you will receive support from your TAs in developing a deeper understanding of recursion and linked lists in relation to this assignment. Feel free to use this extra time to try things out and experiment.

3. **Instructions:**

   A sentence in English is made of several words in a particular sequence. You must represent such a sentence as a list of words. Much like the list example we saw in class, this one is made of nodes. There is one word per node, called `WordNode`. The sentence may also contain zero or more punctuation marks, which are represented by a `PunctuationNode`. The end of the sentence is denoted by a special empty node, called `EmptyNode`. All this needs to be put together using an implementation file similar to the module and the demonstrations.

   The following operations need to be available for your linked list of words.

   -Interface for Node, emptyNode, wordNode and PunctuationNode will implement Node

   - A method `getNumberOfWords` that computes and returns the number of words in a sentence. The punctuation does not count as a word.

   - A method `String longestWord` that determines and returns the longest word in a sentence. The word returned is just the word, and should not begin or end with punctuation. If the sentence contains no words, `longestWord` should return an

empty string.

- A method `toString` that will convert the sentence into one string. There must be a space between every two words. A punctuation mark should have no space between it and whatever precedes it. There is no space between the last word and the end of this sentence. If there is no punctuation mark at the end of the sentence, this string should end with a period (it shouldn't add the period to the original sentence)

- A method `Sentence clone()` that returns a duplicate of a given sentence. A duplicate is a list that has the same words and punctuation in the same sequence, but is independent of the original list. (Not an alias.)

- A method `Sentence merge(Sentence other)` that will merge two sentences into a single sentence. The merged list should preserve all the punctuation. The merged list should be returned by this method, and the original lists should be unchanged.

Tips

1. Design the interfaces and classes that you need for this purpose.
2. In a JUnit test class, create examples of sentences and write tests for each operation.
3. Design the fields and methods for your classes, and verify that your tests pass on them.
4. Check if you can abstract any common parts of your design/code. If so, take a "bottom up" approach and push these common features into an appropriate superclass.
5. Create a simple driver testing each method at least once.

5. Reflection:

Remember to include a reflection of roughly 200 words that speaks to these questions:

1. What did you learn?
2. What do you think the advantages are of using a linked list in this application?
3. Do you think this could have been implemented without a linked list? Explain your answer.

6. Submission:

Submit all files on Canvas under the appropriate assignment. Make sure to include the following named as follows:

Submit your files as a single zip file named: `"Your Name"_"Assignment".zip`

Submission checklist:

- Did you include adequate comments?
- Did you include comment blocks at the top of each file?
- Did you name your files as requested?
- Does your code compile?
- Did you remove any package lines generated by your IDE?
- Did you take care of any warnings presented by your IDE?

7. Checklist:

	Included?
Nodes	
Word nodes logically implemented	
Empty Node logically implemented	
Punctuation Node logically implemented	
Methods	
getNumberofWords	
longestWord	
toString	
clone	
merge	
Misc	
Driver created as requested	
JUnit Tests Created	