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:**

   This assignment will take place over two weeks and is complex enough to take that long. In recitation 1 you will have a planning session and report the results of the planning session as an ICE. In recitation 2, you will have a mini-lab assignment or a code walkthrough.

   For ICE: Mod 4 submit a UML diagram for this assignment, the names of your group members, and an overall plan for completing this assignment.

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.

   - 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.

- Include a JUnit test file that tests each of the above methods.

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.
6. Consider how the demonstration code in class was setup. Maybe you need a SentenceNode interface and a SentenceListImp file.

4. Extensions:

You can get 90% by completing all of the above requests, but to push up to 100%, you'll need to go above and beyond what's asked for. I would like you to add a few elements of your own. The number of points you get for your element depends on its complexity. Here are some suggestions:

- Add additional functions not listed like additional information, sentence splitting, or fun encoding.
- Let your user enter a sentence and turn it into a list with a string to list function.
- Add a graphical element.
- More than just simple testing.

- Stylish and complete JavaDocs

5. Report:

Each assignment must include a short report. The generation of this report should take you no more than 15 minutes. This gives you a chance to reflect back on what you learned and it makes grading easier on your grader. For this report, I want the following sections:

1. Reflection (*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.
4. Extensions (*What extensions are you requesting?)*
5. Grading Statement (*Based on the rubric, what grade do you feel you deserve? Be honest.)*

6. Submission:

Please read carefully. Failure to follow submission instructions can result in a reduced score.

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`

Your zip file should contain your .java files.

Include your report as: `"project_report_04.pdf"`

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. Rubric:

	Possible	Given
Nodes		
Word nodes logically implemented	5	0
Empty Node logically implemented	5	0
Punctuation Node logically implemented	5	0
Methods		
getNumberofWords	10	0
longestWord	10	0
toString	10	0
clone	10	0
merge	10	0
Misc		
Driver created as requested	5	0
JUnit Tests Created for all methods	10	0
Code Quality	10	0
Not included in total possible:		
Driver not included as requested	-100	0
Does not compile	-100	0
Extensions (Not calculated without report)	10	0
Late penalty	-20	0
Creative or went above and beyond	10	0
Code contains warnings	-20	0
TOTAL POINTS POSSIBLE out of 100	90	0