**Assumptions:**

1. The given files will be text. (not images, songs, etc.)
2. The given files will be of a varying size
3. Words and punctuation should be tokenized separately.

**Data Structures:**

1. I chose to store the tokenized text as an ArrayList, filled with Strings.
   1. Strings made sense because the program is meant to parse bodies of text.
   2. Had to decide between Arrays and ArrayLists. Chose ArrayLists for the following reasons:
      1. ArrayLists can grow and shrink dynamically, allowing for the program to parse files of varying sizes. Arrays’ size are predetermined when initialized, so the program would either have to store the Array as a huge block in memory, so that it could handle larger files, or would only be able to work for smaller files. The assumption here is that the programmer doesn’t have a specific size to work with.

**Weaknesses/Limitations:**

1. For any “words” with “multiple punctuation” (example: *word.”*), the program is limited in that it will only remove one of the punctuations. Whichever is first in my cascading “if” statements. Given more time I would have fixed this, but it would be easiest to fix by rewriting, and taking into consideration the problems outlined below
2. Entity matching- When the entity list is scanned into the entityArrayList object, I broke the text down by whitespace, instead of delimiting by line-separator. That was a mistake that also came about as a result of a poor design decision (# 3, below). This problem is noticed in the cases of words like “of” being recognized from the entity list (Sea *of* Tranquility).
3. I should have used a List Iterator to run through the list instead of the cascading if statement that I chose to use. This was a poor decision made early on, and then once I realized my error, I decided that I didn’t have time to adequately fix it, without risking bigger problems and time constraints.
4. After spending tons of time trying to fix the other problems, I didn’t have time to learn about the last two problems.
   1. I didn’t write step 3, since I have little to no experience with parallelism. I looked into it, and recognize that java has a parallel streams class that could have been used, but I didn’t feel I had the time to learn it.
   2. I have no experience writing XML documents. I know about XML, and I would love to learn it, since it’s a really useful skill. It seems especially useful coupled with java. Unfortunately, I didn’t have time to pull together a functional XML doc, so I scrapped it. Output is printed to the console using *System.out.println()*
5. If I could write the whole thing again, I’d make the above changes, as well as breaking the entire program into separate methods and classes, working in tandem to achieve the same (well actually, a better performing) goal.