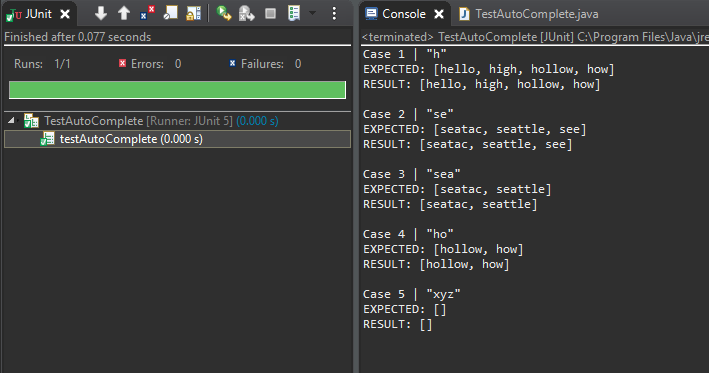
Auto-Complete Java Design Document

I designed my code for homework 7 problem 3, an autocomplete question, around two things that I learned about in class. Firstly, I based my solution off of the trie abstract data type that stores nodes that have n amount of children based on the characters of the local language’s alphabet. I did this because dividing words into this structure allowed me to reduce the number of possible results dramatically with each letter that my algorithm checks through. The trie structure was built with an array container, specifically for its fast random indexing that sped up the algorithm as a whole. Subsequently, the other strategy that I used to search for the words that fit the user’s input for autocomplete was recursive backtracking; when data is stored into the trie structure the letter is inherently stored in the index of the array of children of the node, so some sort of recursion was necessary after using each character in the input phrase was used to, drastically, reduce the number of possibilities of words that the autocomplete could output. Overall the time complexity of this algorithm should be O(n) with the spacial complexity being O(n \* m) with m being the number of words inside of the trie structure.

RESULTS:

This is definitely the fastest way to do this autocomplete function that is within my knowledge. The other abstract data types that I know of don’t seem to fulfill this task as quickly. They would all end up being slower since I wouldn’t end up being able to eliminate as many possibilities of subsequent word combinations with anything but trie. The alternative would be to go through every word in the collection which would take much more time considering the comparison would compare each individual character within the strings.