Technical Report

We will be using the following ADTs for the following reasons:

- Hashtable<Integer, WordAndFreqStorage> to store our big data set containing the words children are able to say at a given age and the frequency that they're able to say each of those words at
    - the keys will be the ages, the valued will be the sample 22 words we can ask the parents to tell us if their kids can say
- LinkedList<String> to store a variable amount of words at a given frequency and age
- BinarySearchTree<String>

We will have five classes. Our first two classes are pre-computational classes.

The class WordAndFreqStorage would contain information related to a specific age. This information is a linked list of words that children of a certain age can say at a certain frequency and an integer variable that stores what that particular frequency is. WordAndFreqStorage objects will be stored in a hash table in the Children class. The Children class represents all the data of the children captured in our raw dataset. This hash table contains ages as keys and WordAndFreqStorage objects as values. The hash table will be initialized in the Children constructor. The get15Words method in the Children class will be able to find 15 words of a predetermined frequency (we are still debating what that should be) by indexing the hash table at a certain age, then looking through the WordAndFreqStorage objects to find the one with the correct predetermined frequency. We will try to find 15 words with the predetermined frequency, but if we cannot find enough words with that frequency, we will use a while loop to find the next closest frequency.

The next class we have is the parent input class Child. This class has a method called getAge that gets the child's age from the parents. Using this age, another method called retrieve15Words stores the information returned from the get15Words method in the Children class. We will turn this collection of words into a queue. We are using a queue so it can be easily iterated through to ask the parents if their child can say the word or not. We will have another method called wordQueueToFreqArray that asks the parents whether their child can say the 15 words or not. For each word the parent will put in yes or no. We will store a frequency of 1 in the array if the parents answers yes and 0 if no. This will be useful in the next class.

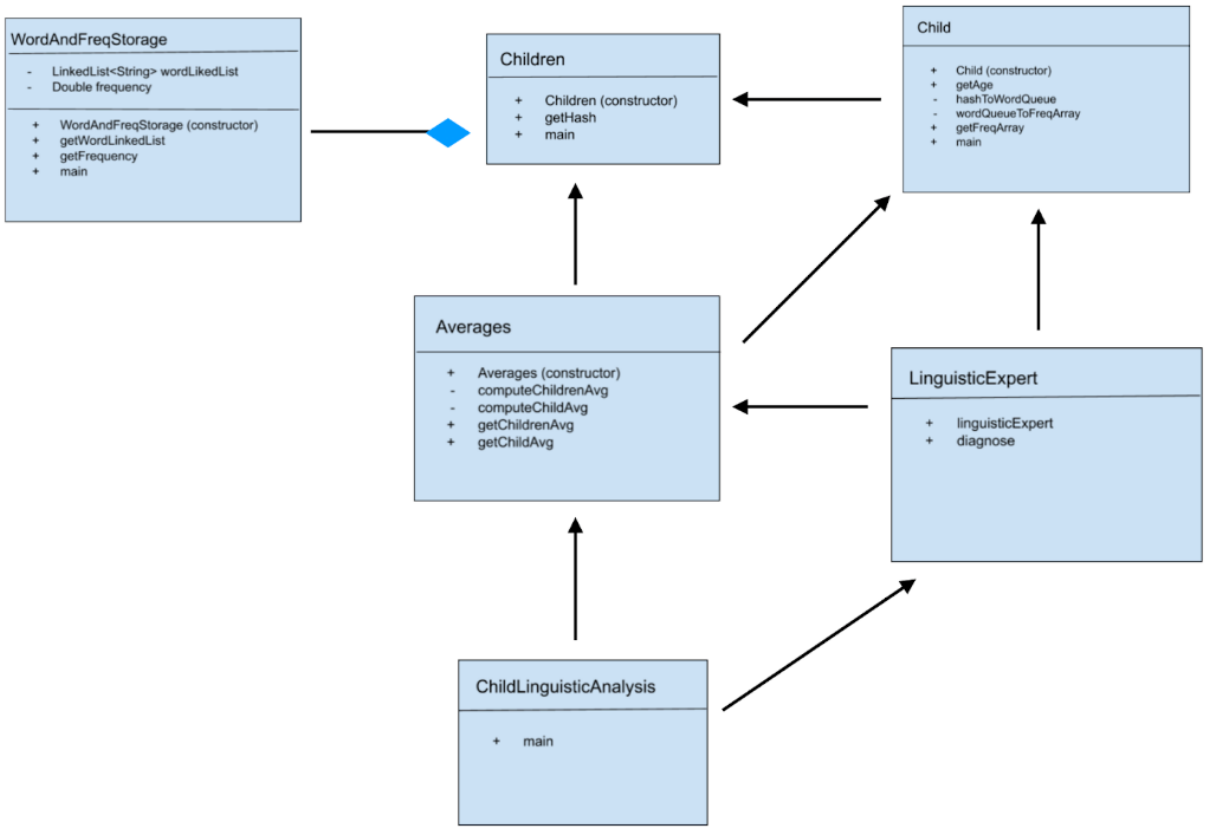
The next class is a computing class Averages. It will take the information we gathered in the pre-computational "Children" and parent input "Child" classes and calculate two average frequencies. The first main action method will take the the arrays from parent input and compute an average frequency for the 15 sample words.

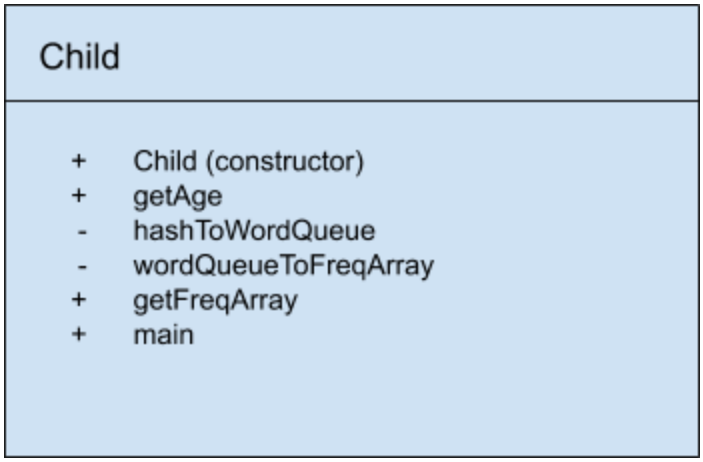
The next method called compute child is a private class that takes the specific child's frequencies (gathered in getFreqArray in parent input class) and calculate the average.

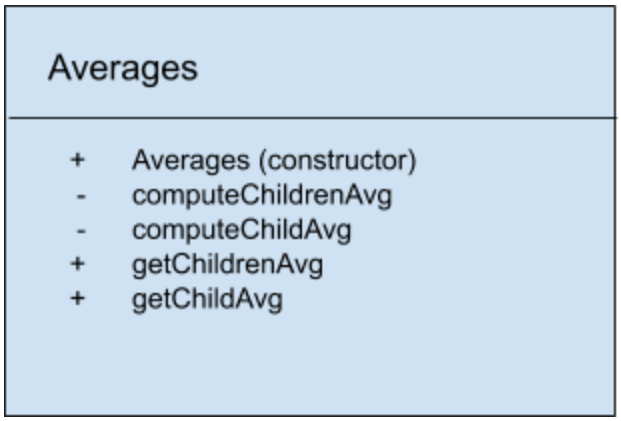
We will compare the specific child's average to the general average of the sample words in the last class, called "ChildLinguisticAnalysis." These two averages will be reported back to the parent to give them an idea of where their child stands in their linguistic development. This driver class will only have a main method that runs and tests the program.

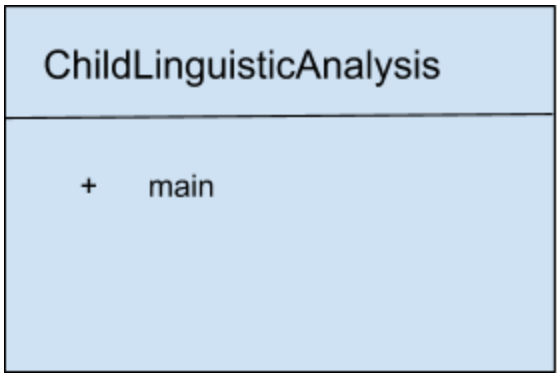
HashTable

- Decision tree and categories of words, 22 paragraphs, our program decides which paragraph to give to the user based on what their child can say, on an expert decision tree, categorize children's vocabulary




Child
+ Child (constructor)
+ getAge
- hashToWordQueue
- wordQueueToFreqArray
+ getFreqArray
+ main


Averages
+ Averages (constructor)
- computeChildrenAvg
- computeChildAvg
+ getChildrenAvg
+ getChildAvg


ChildLinguisticAnalysis
+ main