The design of this program that we chose involves interaction between a Dictionary object and a WordLadderSolver object, which is basically what would happen in real life to find a word ladder. A dictionary would be used (or Google or some store of information on words) to look up potential words that are valid, and then the word ladder solver, or the user, would then test words that differ in one letter to try and find a ladder between two words. We considered having a Graph object that the Dictionary would then hold as a wrapper, and then would interact with the WordLadderSolver object, but the way did it cuts out the Graph and just implements the Graph as a Dictionary, instead of having the Dictionary hold a Graph object. From a programming perspective, it makes sense to have the dictionary be a graph instead of holding a graph, and from a user perspective, they just have a dictionary object as a graph to manipulate if they were to manipulate it. Future enhancements could come in the form of improved computation capabilities through BFS iteration by modifying the makeLadder method that is private to the WordLadderSolver class. I think that our design separates the Dictionary from being implemented in the WordLadderSolver class, so we can have a has-a relationship that is actually modeled, and then we can hide the graph's implementation through the private lists of nodes and adjacency lists in the Dictionary class. Overall, it's a pretty good OOD for this problem.