Jonathan Friesen and Vincent Ip
**Design Rationale**
EE422C Assignment 4: Fun with Word Ladders

The Driver functions like the front end of a business establishment. The WordLadderSolver class is akin to an employee at the back end of the business establishment. The program (or business) takes in inputs (or customer requests) and calls the necessary classes (or employees) to handle the request. In this case, the requests are represented by text files relaying the String input, and the employee handling the request is the WordLadderSolver class. Maintaining this kind of client-worker relationship effectively organizes the code into manageable and easily-distinguished modules, and makes for easier coding. The way a word ladder is created lends itself naturally to a recursive structure, wherein the computer arrives at the each word in the ladder and, much like a person who works this out by hand, repeats its process of determining the next word.

An alternative design could be to better optimize the length of the word ladder. Have multiple solution lists building at the same time in the recursive calls; each solution list corresponds with a unique way to traverse through the temporary list of candidate words (the default way is to start at the top of the list and work down; other ways could include starting at the bottom of the list and working up to the top). Once the execution completes, the method returns the shortest solution list. An obvious advantage would be that the user would see a more optimized solution (i.e. shorter ladder) each time. The disadvantage, though, would be that more memory is used to keep track of multiple lists. Execution time would also be drawn out, since multiple lists would be building simultaneously.

A possible expansion for this project would be to make the method dynamic with respect to word length. What if, for example, you wanted to compute word ladders for 4-letter words? 7?

Our design adheres to good design principles. Our design is objected-oriented: the driver instantiates an object of a class (WordLadderSolver), and calls methods from that class (computeLadder). No one function is too long or incoherent; computeLadder itself was broken into several private helper functions called only within the class to aid in computing the ladder. The driver and WordLadderSolver class are tightly coupled, as the class takes multiple input parameters and types (starting and ending words, starting position, and the dictionary file) from the driver in order to work.