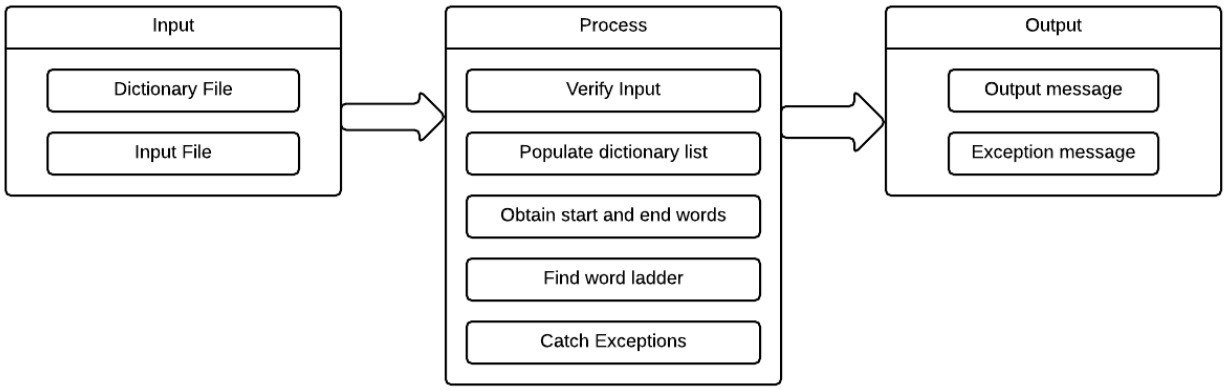
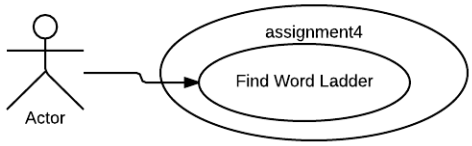
Project : Assignment 4
Written by: Bharat Kulkarni - bsk524
Dung Le - dkl524
Completed : 03/06/16

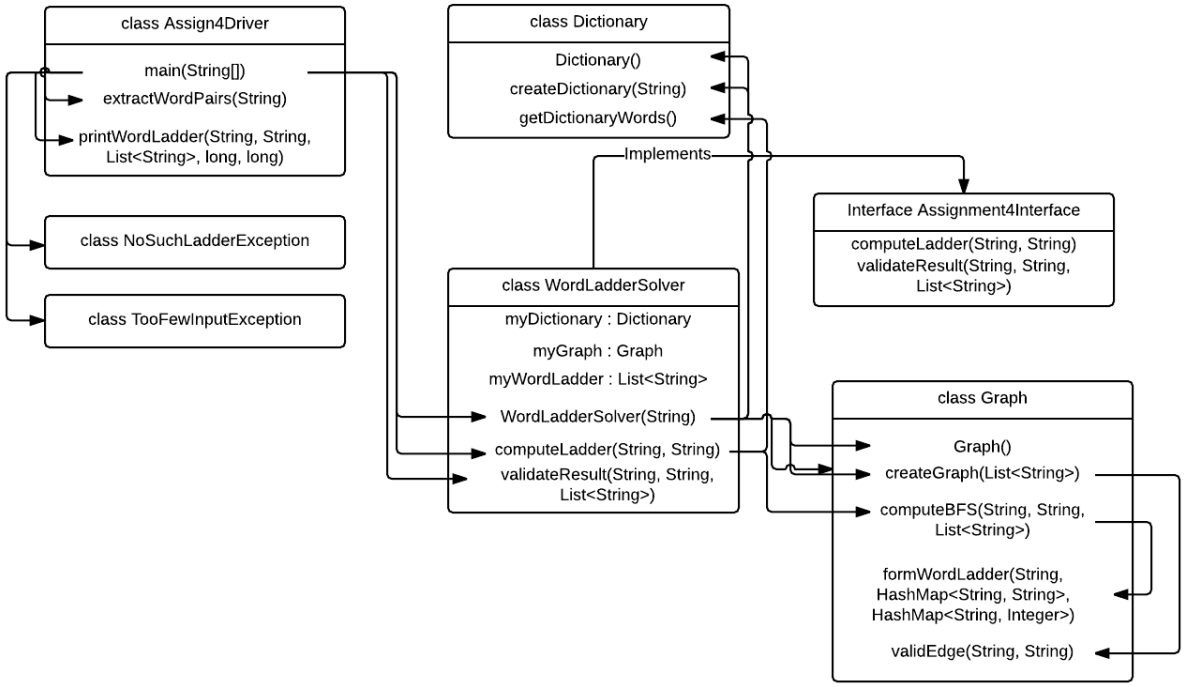
System IPO Diagram:



Use-case diagram:



UML and functional block diagram:



Main Method Design:

The main method implements a simple algorithm of iterating through a processed list of inputs. During each iteration, we use the try-catch block in order to catch and process different Exception cases while calling our computeLadder method in order to search for word ladder between the given start and end words. If a ladder is found, we call the printWordLadder method in order to correctly print the output.

Algorithm Rationale:

The algorithm we implemented was the Breadth First Search (BFS) algorithm using the criteria that the children nodes strings must differ from their parent node string only by 1 letter. This way, we could easily search for the potential word ladder from any given node.

In this design, we implemented a dictionary class that allows for storing the given dictionary where only the words that matter are recorded, a graph class where we use a BFS to store the words in the dictionary in the manner that allows us to easily use to look up word ladder. In the graph class, we also have a method that computes the ladder between 2 words using the BFS graph we created.

We considered the recursion method mentioned however, recursion posed much more difficult to test and difficult to picture while programming as well as testing.

This design allows us to potentially use a different graph method or different criteria to process the dictionary for either a more specific functionality or a different functionality. Also because the design separates different objects and methods that it's easier to test and search for optimization or enhancement options.

While the classes and methods could be separated more, all related functionalities are grouped together in a class. The data flow is easy to follow. User interface (main method) is easy to understand and variables are properly protected.

Test Plans:

Black Box: In order to test the over all functionality of our code, we used the given dictionary and coming up with our own test cases. The most basic test case is to give a word that exists in the dictionary both as the start and the end word to make sure that code can handle the basic case. Then we set the start word and end word to be 2 words that require few ladder words in between like busts and buses. We increase the difficulty level by picking words that require more and more ladder words in between. Then we test if the code throws correct exceptions and messages when the words have no ladder in between, when the words don't exist in the dictionary, when the input format differs from the expected format (more than 1 space in between words, space before start word, if there are more than 2 words, just 1 word, no word, etc). The program needs to be able to throw the correct exception with the correct message while not halting the program until it has gone through the entire input file. Since it stated that the dictionary file given is the dictionary file used for this assignment, we bypassed testing for valid dictionary input (if file is empty or contains only comments, if no words in dictionary differ from another by 1 character, etc.) However, the code can easily be edited to catch exceptions regarding the dictionary as well.

White Box: Since we kept the main method relatively simple, the white box testing would focus more on the Graph, Dictionary, and WordLadderSolver. The basic test case

would be to test if the dictionary file is correctly populated in the program. To test this, we write a test case where we feed a dictionary test file and create another dictionary object that holds the expected populated dictionary values and compare the dictionary object populated using the test file and the expected dictionary object to see if they match. Next, is the graph, from the populated dictionary object, we again feed it to the create graph method and compare it to the expected graph. Once we have verified that the dictionary and the graph work correctly, we can test the computeLadder method. Since this program is meant to find just 1 ladder between the start and the end word out of potentially many options, we can test the output ladder to ensure that each word only differs from the previous word by 1 character. For optimization purposes, we can purposely try to find the shortest ladder path between 2 words, which is plausible since we used the BFS graph to map the dictionary.