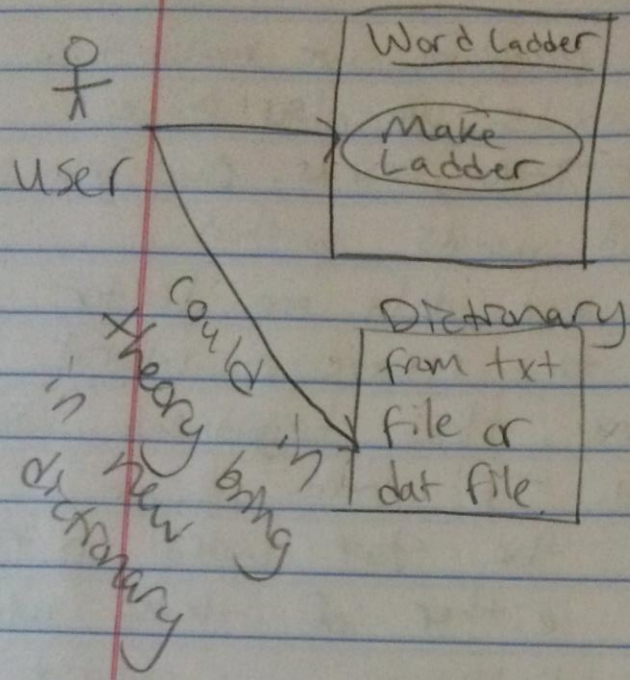
Analysis and Design

Analysis - In this program, we receive two inputs that are words. We also have a dictionary full of 5 letter words. Our job is to locate valid words within the dictionary that are only one letter apart in order to create a word path / ladder from the beginning word to the end word. If there is no possible path from the start word to the end word or if either of those words are not in the dictionary, we must print an error and move on to the next test case.

IPO diagram

inputs	processes	outputs
- list of word ladders to try - dictionary file with valid words	- starting word put in recursive BFS method, checks one letter changes and sees if the word is valid. If so adds to queue for BFS - If invalid, returns an error for invalid.	- if a word ladder is found, output the entire ladder - If not, an error messa is output.

Use case diagram

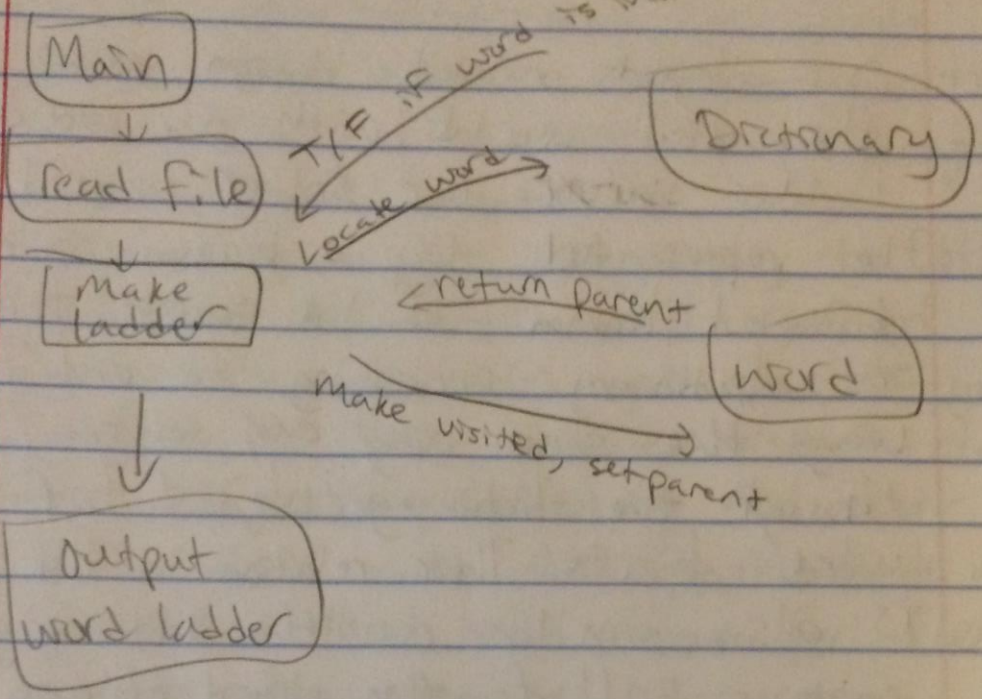


UML Diagram

A4Driver has a exception

Dictionary has words (stored in hashmap

Block Diagram



Algorithm for driver logic (main method)

start, check for 2 arguments

try (open file, while file has lines {
- take line and split 2 words
- check for invalid input, > 2 words
else call output word ladder)

catch file not found

catch IO exception

Rationale of our design

Our object oriented design mimics the real world with our word ladder solver. Our solver would be represented by a human in real life. A human would search through a dictionary searching for valid words the same way our solver looks through the dictionary object to find word objects. Their relation in the program is also similar as the dictionary contains all of the word objects, like a real dictionary.
We considered using either a BFS or a DFS. After some research in both searches, we concluded in order to make a quality program that output quality answers, we should use a BFS in order to ensure finding the fastest possible ladder.
The user would like this output however may be upset with the amount of time to find the answer. If runtime is a large issue, the user may feel a DFS (which can be faster) might be a more appropriate solution.

our program has been built for flexibility with other dictionarys or words. Those are simply exchanged with new text files and our program can handle that.
Our program is very easy to follow and has been kept well organized in order to maintain principles of good design. We have seperate distinct classes for the words, dictionary and solver with their own distinct methods and variables that help allow for flexibility within the program. Our program is also cohesive with inputs, errors, and other parts of the module being grouped appropriately and neatly. The dictionary and word classes are coupled closely but can be seperated from the driver and solve word ladder and used for other programs as well.