Design Analysis

Assignment 4
Christopher Friesen cmf2536
Malvika Gupta mg42972

The purpose of this assignment was to design and implement a program that would find a word ladder between two 5 letter words. A word ladder is a chain of words each with only one letter different from the previous word that starts at the provided “start word” and ends at the provided “end word”. In this program, we are given the precondition that the provided words will be lowercase, 5 letters long, and be valid words from the provided dictionary. The way we designed this program was to first create a dictionary of valid words. We then plan on using a recursive design to find all the possibilities for the next word in the word ladder, and then check through all those possible options recursively. We made the additional consideration of optimizing the process slightly by checking previously found options for next words, so that if there was possibly a shorter chain with the next word, we could switch to using that. An obvious alternative would be to do this iteratively, however, considering the wide possibilities of lengths and types of ladders between words, it made more sense to do this recursively. Doing this recursively would also probably be faster, which would be an advantage from the user’s perspective. Our program will also be flexible to future enhancements and optimizations. For example, the program could find all the possible word ladders, but only return the shortest. We could also sort the list of valid next words for the chain, so that the program would check chains that seem more promising (the next word added to the chain is the most similar to the end word) first, possibly leading to shorter times and ladders. Our program adheres to the principles of good design by utilizing a public interface and private implementations of the methods that we used. We utilized the functionality of exceptions to prevent the program from crashing given bad input or the case in which a word ladder is not possible.

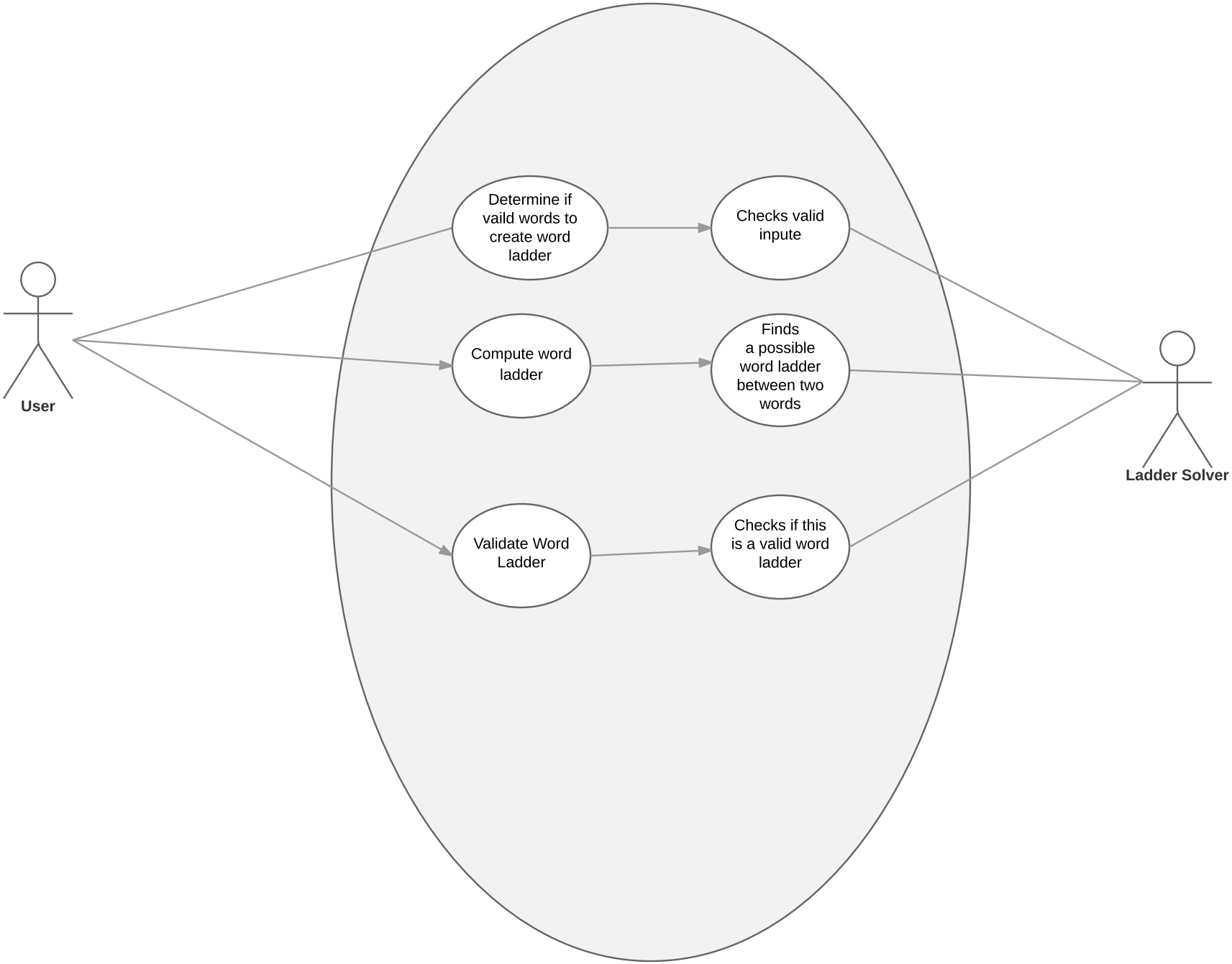
Algorithm for Driver Logic

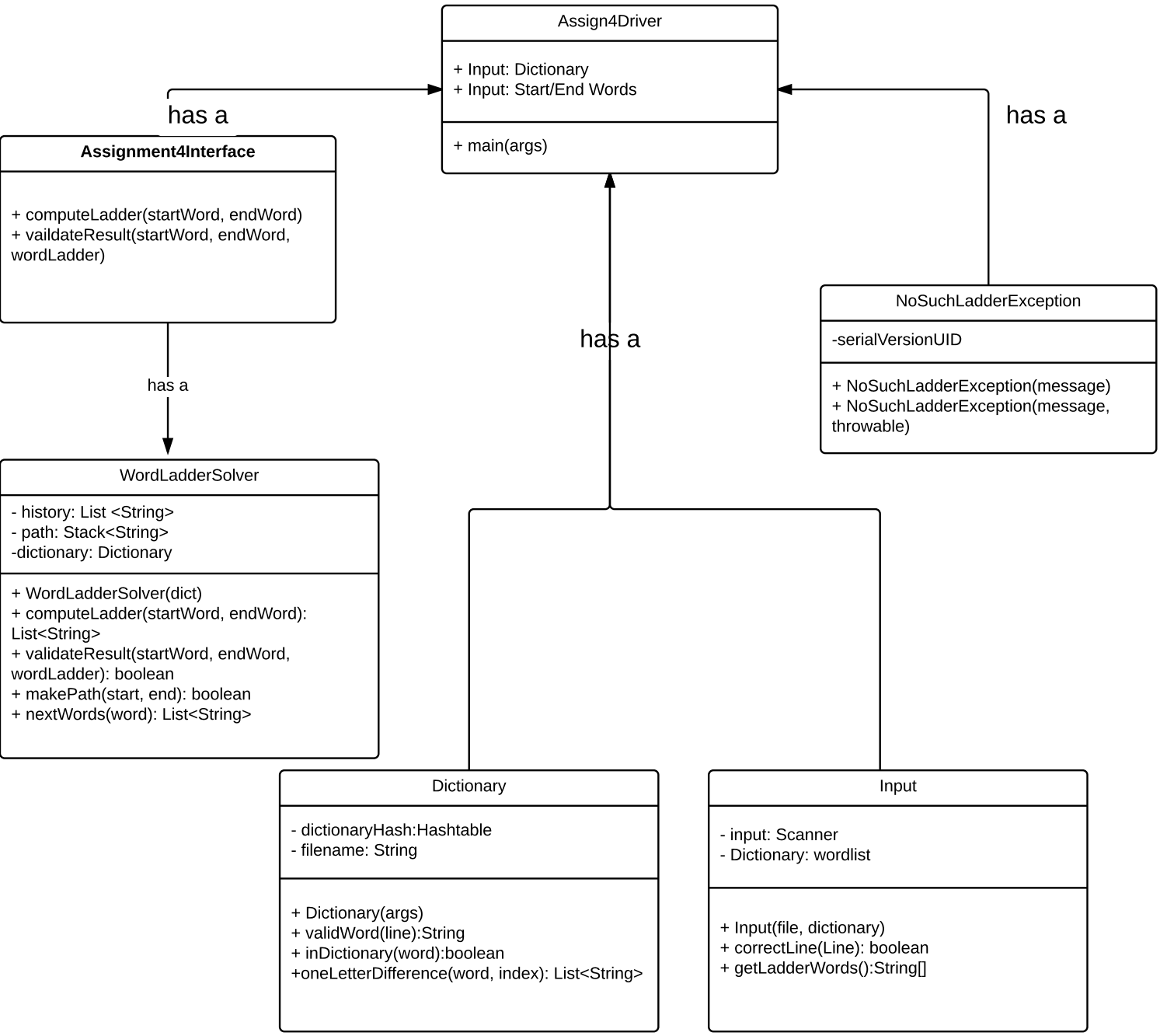
Main Method:

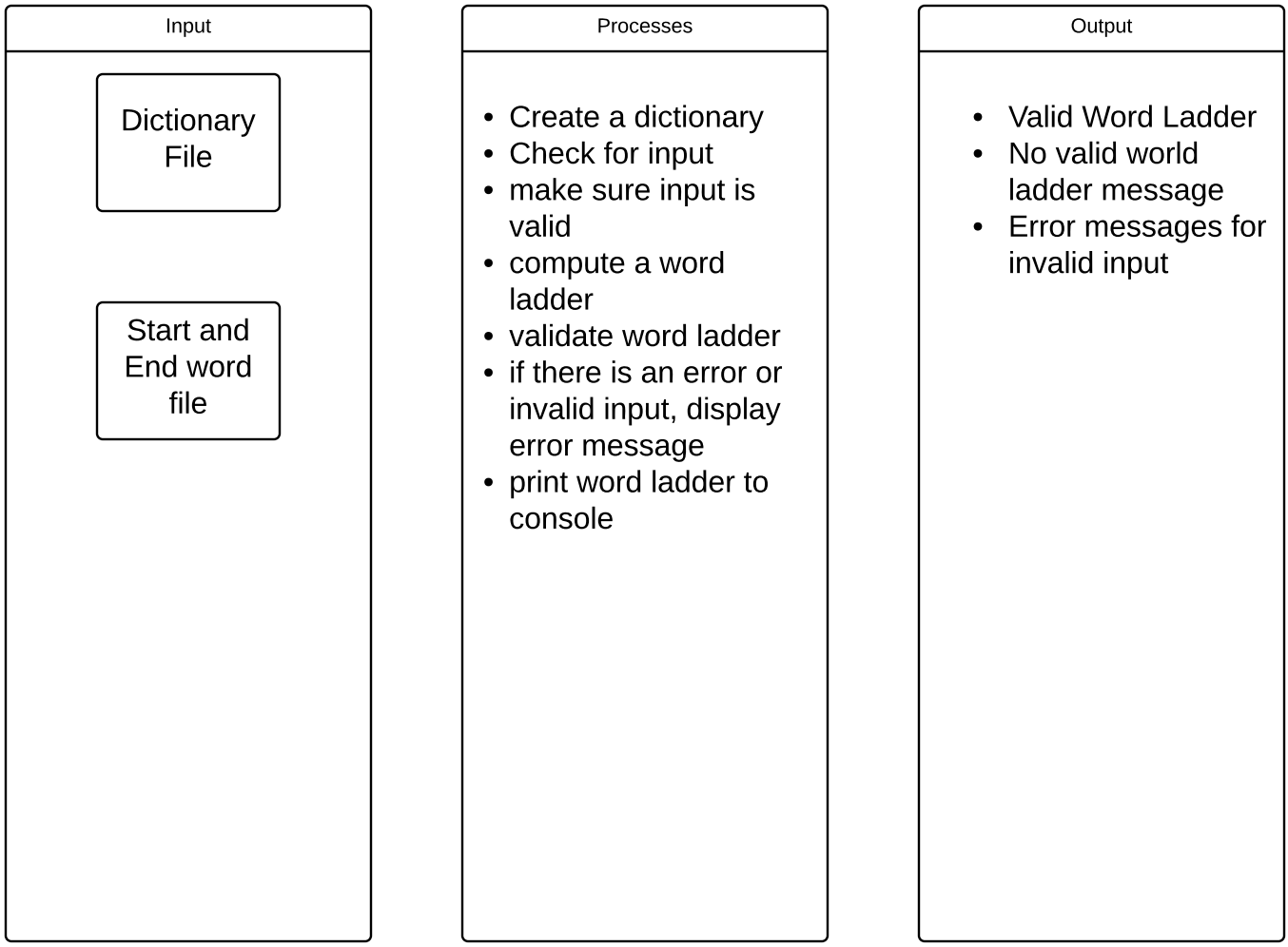
1. Take in two arguments
2. Initialize an object of the wordLadderSolver
3. While there are still more inputs in the input file
    a. Get the next words in the file
    b. Print out the start and end words to the console
    c. Compute the ladder using the compute ladder method
    d. Validate the result of the word ladder
    e. If the input words have a valid word ladder
        i. Print out the ladder
    f. Else
        i. Throw the no such ladder exception
    g. Print 10 asterisks to separate out the different inputs

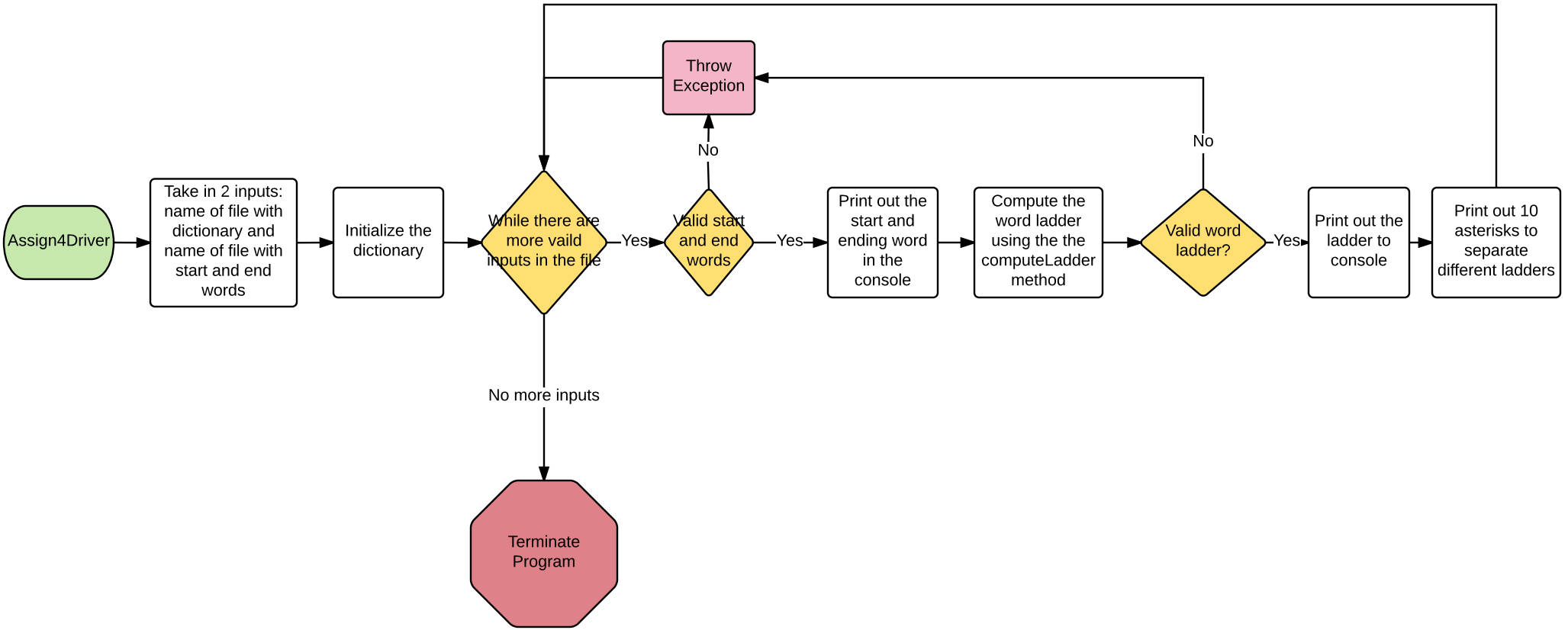
Compute Ladder Method:

1. Clear the path stack
2. Clear the history list
3. Push the start word to the path stack
4. Add the start word to the history
5. Call makePath function with the start and end words
    a. Check if the last word in the path stack is the desired end word
        i. If it is, return true
    b. Find a list of all possible next words using the nextWords function and the last word in the path
    c. Check if the words found are already in the history
        i. If they are not, add them to history and add them to a new validNext array
    d. Create a flag called check
    e. For all the words in the vaildNext array
        i. Push the word in the path
        ii. Call the makePath function (recursively) with the start and end word
        iii. If the flag is true
            1. Return true (and break out of this level of the method call)
    f. If the flag is false
        i. Pop off the last word in the path stack
        ii. Return false


User
Determine if vaild words to create word ladder
Checks valid inpute
Compute word ladder
Finds a possible word ladder between two words
Validate Word Ladder
Checks if this is a valid word ladder
Ladder Solver


Assign4Driver
+ Input: Dictionary
+ Input: Start/End Words
+ main(args)
has a
has a
Assignment4Interface
+ computeLadder(startWord, endWord)
+ vaildateResult(startWord, endWord, wordLadder)
has a
NoSuchLadderException
-serialVersionUID
+ NoSuchLadderException(message)
+ NoSuchLadderException(message, throwable)
has a
WordLadderSolver
- history: List <String>
- path: Stack<String>
-dictionary: Dictionary
+ WordLadderSolver(dict)
+ computeLadder(startWord, endWord): List<String>
+ validateResult(startWord, endWord, wordLadder): boolean
+ makePath(start, end): boolean
+ nextWords(word): List<String>
Dictionary
- dictionaryHash:Hashtable
- filename: String
+ Dictionary(args)
+ validWord(line):String
+ inDictionary(word):boolean
+oneLetterDifference(word, index): List<String>
Input
- input: Scanner
- Dictionary: wordlist
+ Input(file, dictionary)
+ correctLine(Line): boolean
+ getLadderWords():String[]


Input
Dictionary File
Start and End word file
Processes
• Create a dictionary
• Check for input
• make sure input is valid
• compute a word ladder
• validate word ladder
• if there is an error or invalid input, display error message
• print word ladder to console
Output
• Valid Word Ladder
• No valid world ladder message
• Error messages for invalid input


Assign4Driver
Take in 2 inputs: name of file with dictionary and name of file with start and end words
Initialize the dictionary
While there are more vaild inputs in the file
Yes
Valid start and end words
No
Throw Exception
Yes
Print out the start and ending word in the console
Compute the word ladder using the the computeLadder method
Valid word ladder?
No
Yes
Print out the ladder to console
Print out 10 asterisks to separate different ladders
No more inputs
Terminate Program