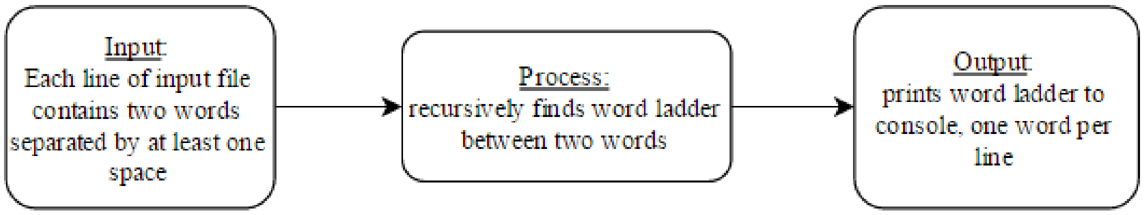
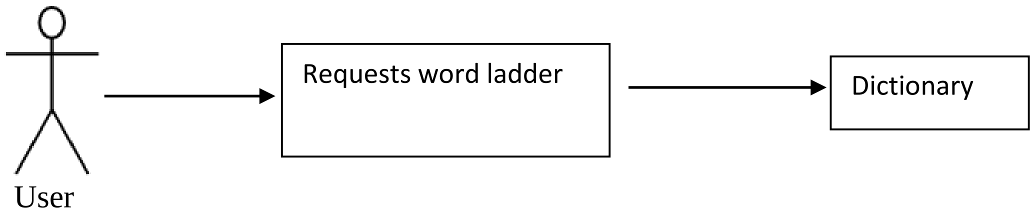
Design

Katelyn Ge (kbg488) and Zain Rasheed Rajput (zr2352)

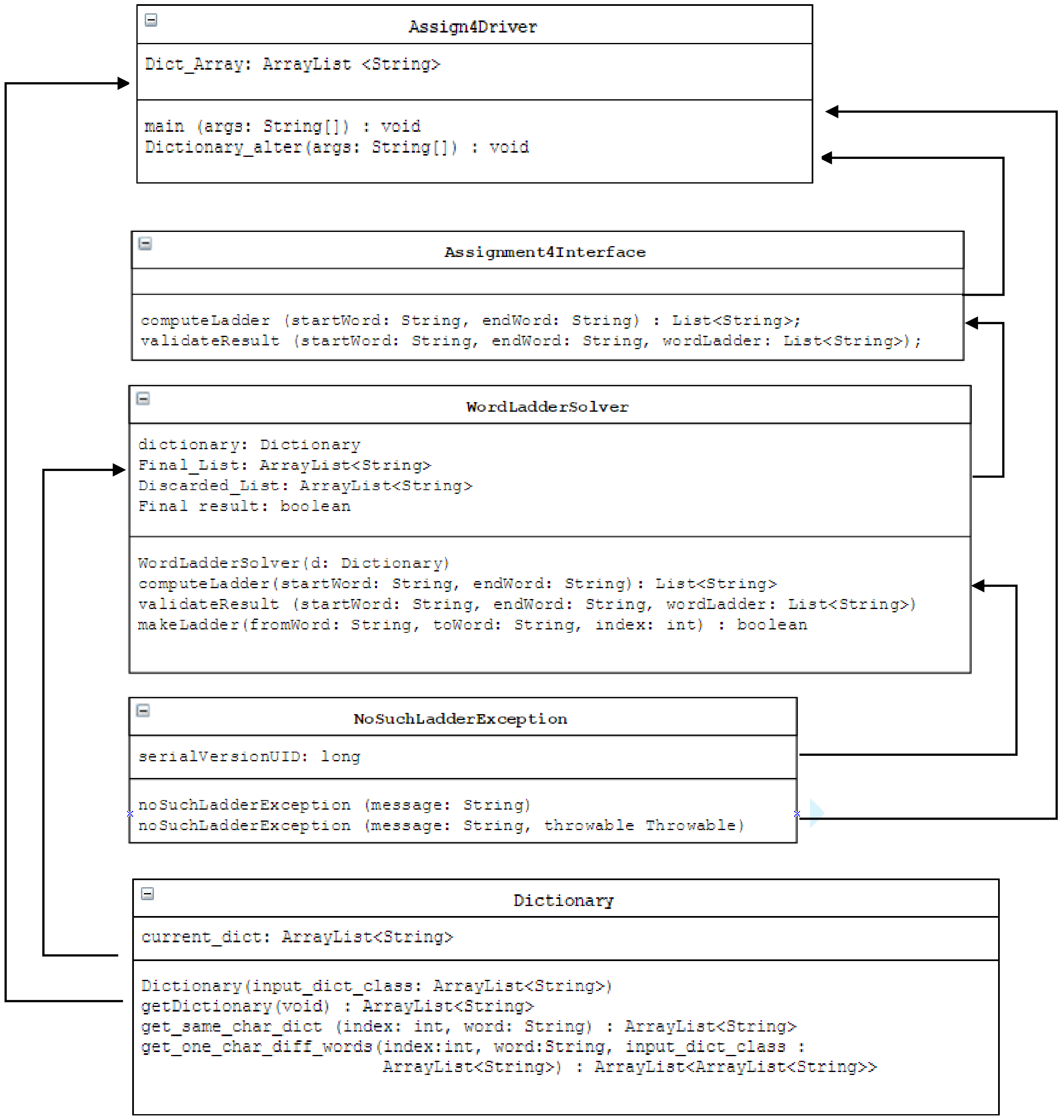
1. System IPO Diagram:


Input:
Each line of input file contains two words separated by at least one space
Process:
recursively finds word ladder between two words
Output:
prints word ladder to console, one word per line

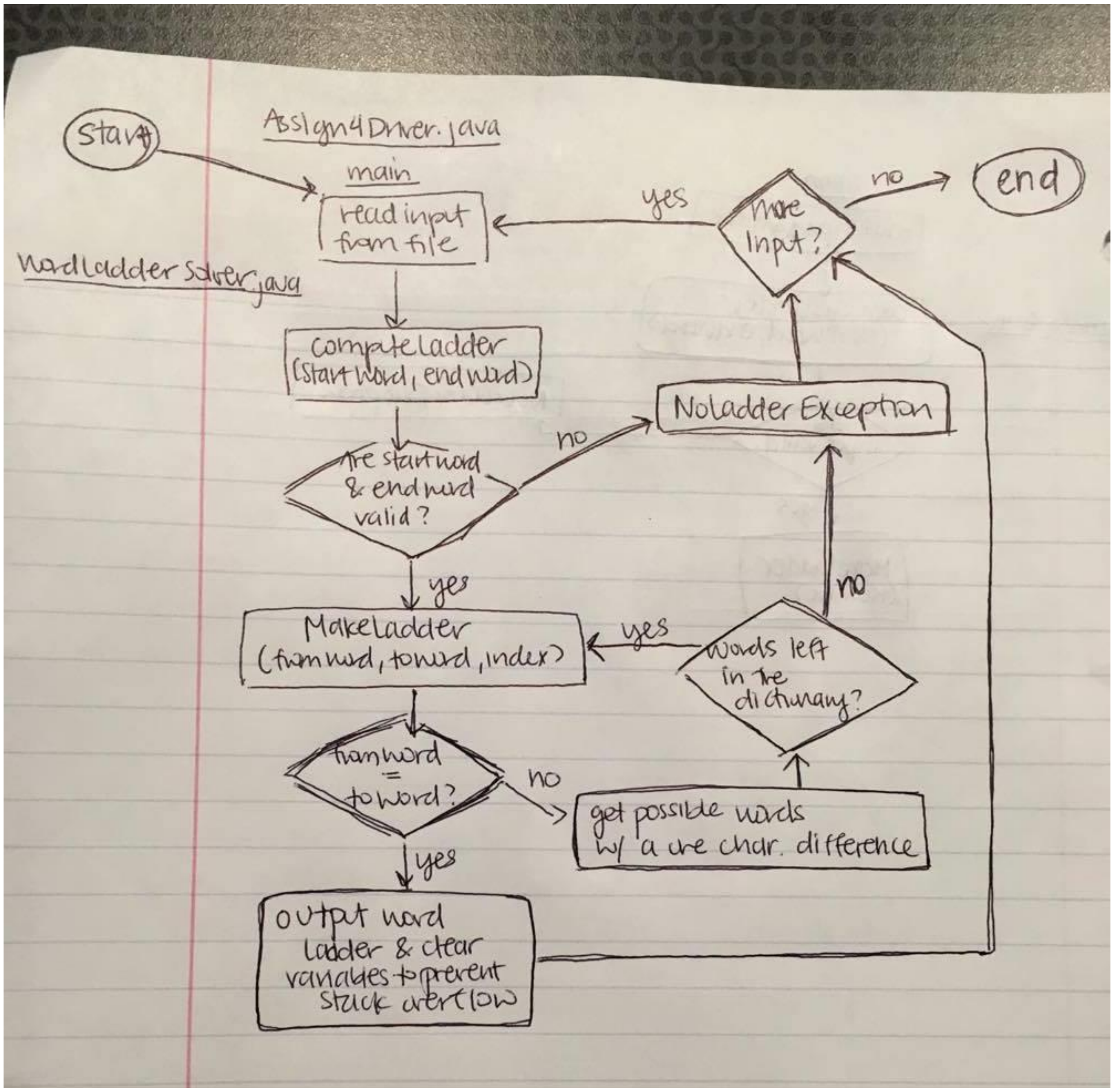
2. Use Case Diagram



3. UML model


Assign4Driver
Dict_Array: ArrayList <String>
main (args: String[]) : void
Dictionary_alter(args: String[]) : void
Assignment4Interface
computeLadder (startWord: String, endWord: String) : List<String>;
validateResult (startWord: String, endWord: String, wordLadder: List<String>);
WordLadderSolver
dictionary: Dictionary
Final_List: ArrayList<String>
Discarded_List: ArrayList<String>
Final result: boolean
WordLadderSolver(d: Dictionary)
computeLadder(startWord: String, endWord: String): List<String>
validateResult (startWord: String, endWord: String, wordLadder: List<String>)
makeLadder(fromWord: String, toWord: String, index: int) : boolean
NoSuchLadderException
serialVersionUID: long
noSuchLadderException (message: String)
noSuchLadderException (message: String, throwable Throwable)
Dictionary
current_dict: ArrayList<String>
Dictionary(input_dict_class: ArrayList<String>)
getDictionary(void) : ArrayList<String>
get_same_char_dict (index: int, word: String) : ArrayList<String>
get_one_char_diff_words(index:int, word:String, input_dict_class : ArrayList<String>) : ArrayList<ArrayList<String>>

4. Functional Block Diagram


Start
Assign4Driver.java
main
read input from file
WordLadderSolver.java
computeLadder (startWord, endWord)
Are startword & endword valid?
no
yes
NoLadder Exception
more Input?
yes
no
end
MakeLadder (fromWord, toWord, index)
yes
words left in the dictionary?
no
fromWord = toWord?
no
get possible words w/ a one char. difference
yes
output word ladder & clear variables to prevent stack overflow

5. Algorithm

```
main:
try
     read file by line
     split string into two words
     try
          List result = wordLadderSolver.computeLadder(first
          word, second word)
          print result
     catch NoSuchLadderException
catch FileNotFoundException

Dictinoary_alter:
     opens A4-words.dat
     places content into an arrayList of strings

WordLadderSolver:
     check if words are 5 chars long
     check if words are the same
     else, call makeLadder(first word, second word, index)
     if (makeLadder = true)
          return the word ladder
     else
          throw NoSuchLadderException

boolean makeLadder (fromWord, toWord, index)
     if fromWord == toWord
          return true
     else
          new index = toWord.length – 1
          compare fromWord to every word in dictionary which is
               one character away at (new index)
          puts options into solutions/deletions list
```

6. Rationale

   a. How does your OOD reflect the interaction and behavior of the real-world objects that it models?
      - We created an object class called dictionary that represents a real-world dictionary. A real dictionary determines what sequences of characters makes a real word, as does our object class
   b. What alternatives did you consider? What were the advantages/disadvantages of each alternative both from a programming perspective and a user perspective?
      - We considered coding this iteratively instead of recursively, but since we were encouraged to program this part of the code recursively, that is what we did. The advantages of this would be that it would be easier to understand from a user perspective. The disadvantages of this would be from a programming perspective, as the code would be a lot longer in length.
   c. What are some expansions or possible flexibilities that your design offers for future enhancement?
      - We can expand the program to support more than 5 letter words. Since our design separates functions into their appropriate classes, it would be easy to expand our dictionary to include all words, and throw Exceptions in our WordLadderSolver if words are different lengths
   d. How does your design adhere to principles of good design: OOD, cohesion, coupling, info hiding, etc?
      - Our design adheres to the principles of good design. The Assign4Driver takes care of reading input, calling WordLadderSolver, and displaying output. WordLadderSolver is in charge of finding a wordLadder using a recursive method. And our Dictionary class keeps track of our list of words. Changes in the WordLadderSolver will not strongly affect the Dictionary class and vice versa.