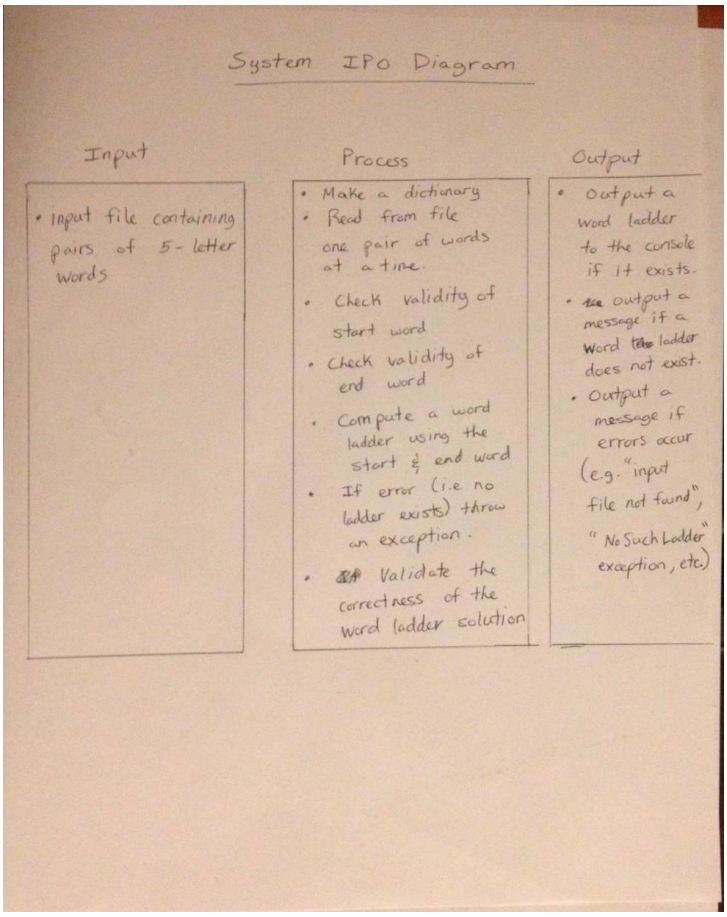
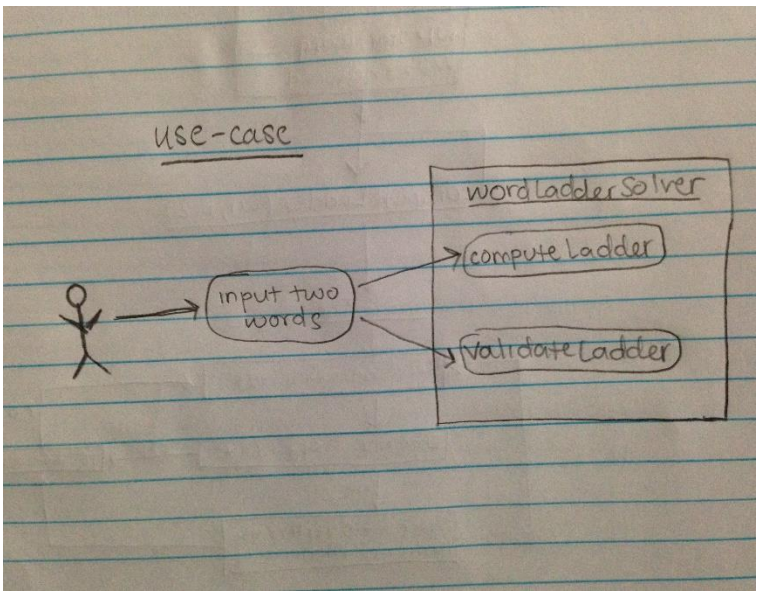
Sharmistha Maity and Cesar Gonzales Assignment 4 Design

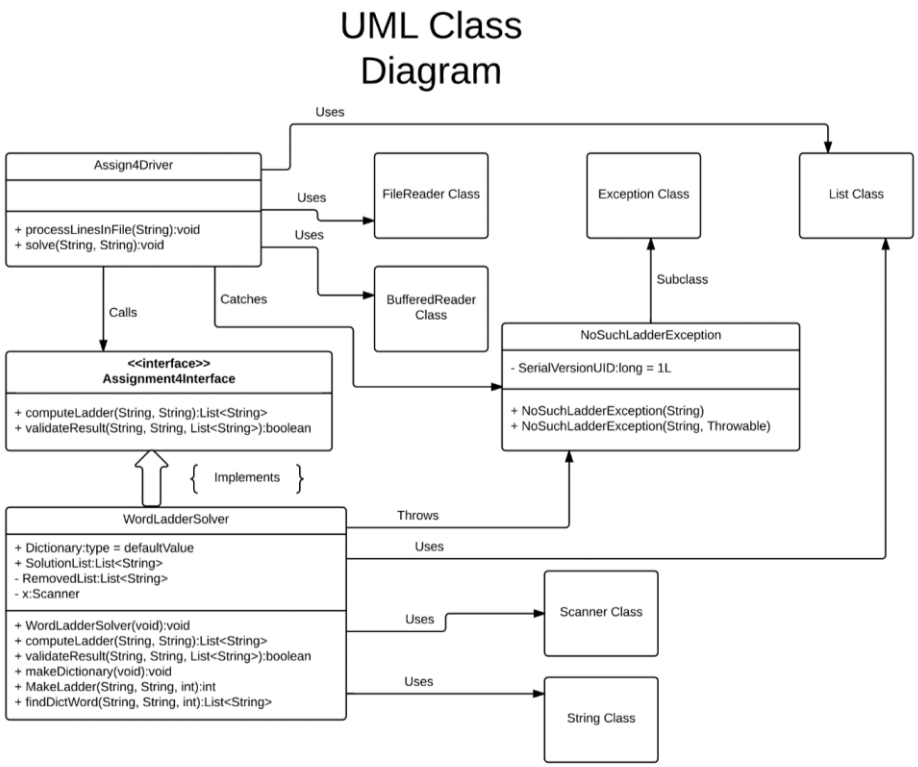
1. IPO



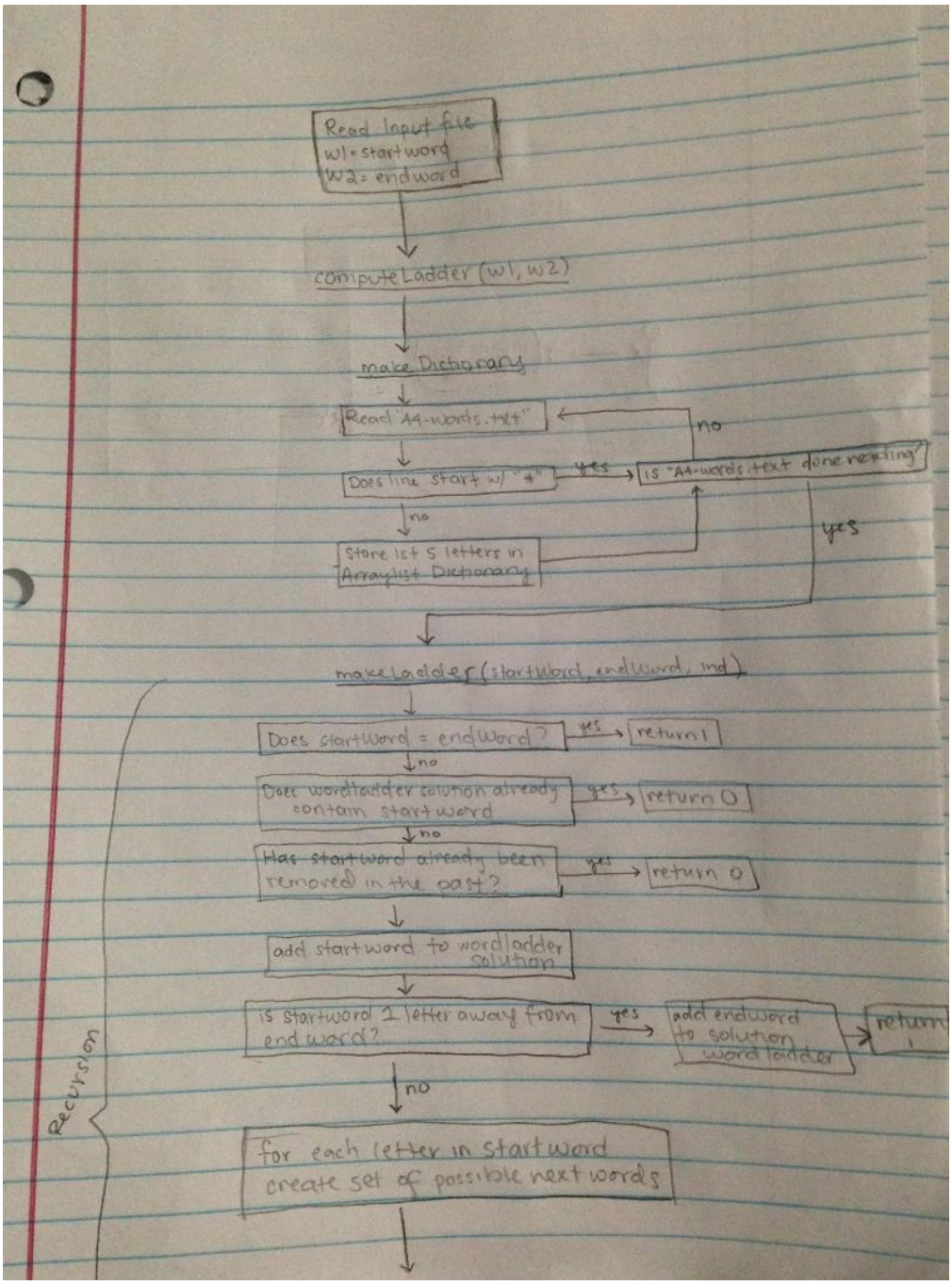
2. Use-Case Diagram

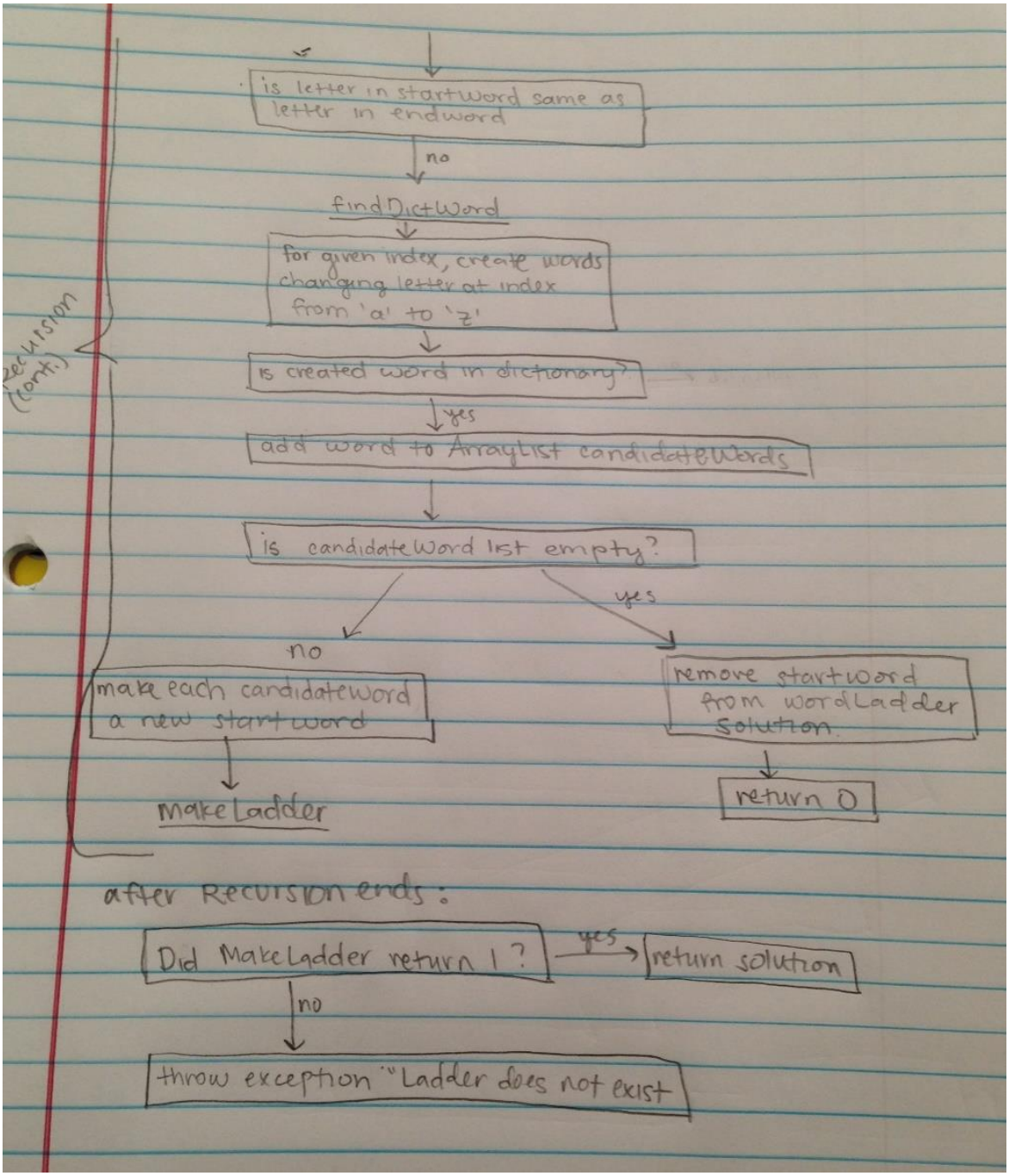


3. UML

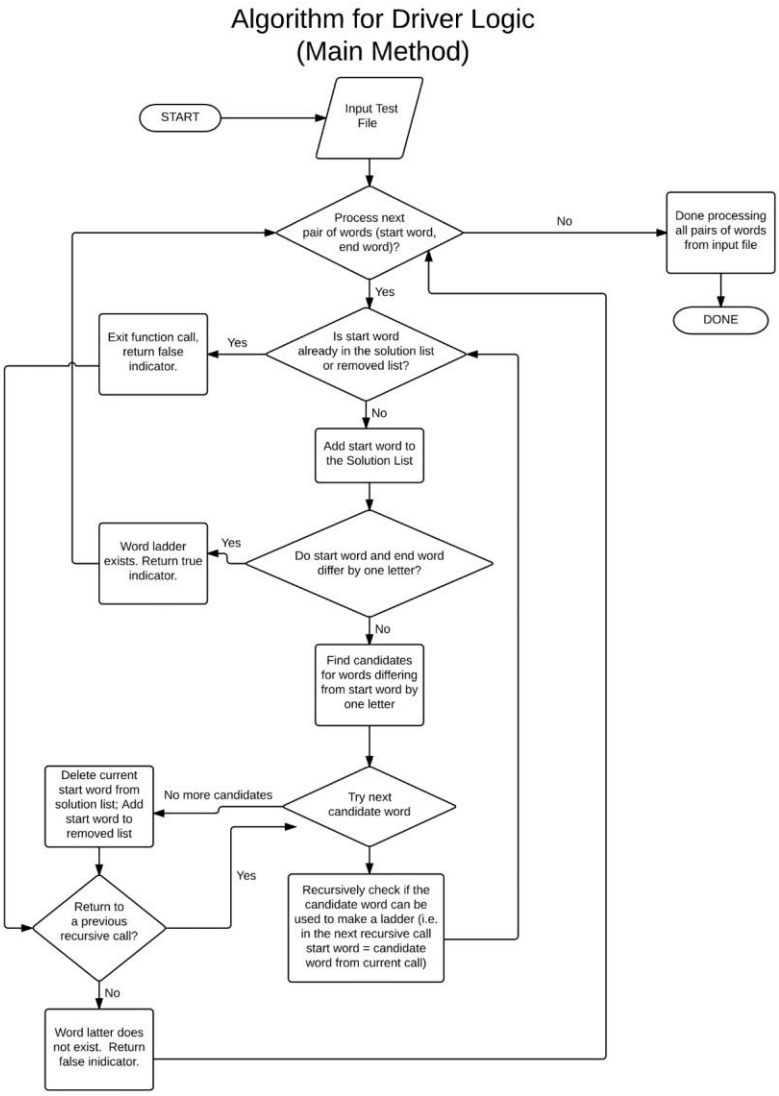

UML Class Diagram
Uses
Assign4Driver
+ processLinesInFile(String):void
+ solve(String, String):void
Uses
FileReader Class
Uses
BufferedReader Class
Exception Class
List Class
Subclass
Calls
Catches
NoSuchLadderException
- SerialVersionUID:long = 1L
+ NoSuchLadderException(String)
+ NoSuchLadderException(String, Throwable)
<<interface>>
Assignment4Interface
+ computeLadder(String, String):List<String>
+ validateResult(String, String, List<String>):boolean
Implements
WordLadderSolver
+ Dictionary:type = defaultValue
+ SolutionList:List<String>
- RemovedList:List<String>
- x:Scanner
+ WordLadderSolver(void):void
+ computeLadder(String, String):List<String>
+ validateResult(String, String, List<String>):boolean
+ makeDictionary(void):void
+ MakeLadder(String, String, int):int
+ findDictWord(String, String, int):List<String>
Throws
Uses
Uses
Scanner Class
Uses
String Class

4. Functional Diagram




is letter in startWord same as letter in endWord
no
findDictWord
for given index, create words changing letter at index from 'a' to 'z'
is created word in dictionary?
yes
add word to ArrayList candidateWords
is candidateWord list empty?
yes
no
remove startWord from wordLadder solution
return 0
make each candidateWord a new startWord
makeLadder
recursion (cont.)
after Recursion ends:
Did MakeLadder return 1?
yes
return solution
no
throw exception "Ladder does not exist

5. Algorithm Logic



6. Paragraph describing the rationale behind your design:

For our design, we wanted to incorporate interfacing and inheritance. To compute a word ladder for a given input for start word and end word, we wanted to design a function that would recursively create a set of words that differ by one letter compared to the start word until a word that is one letter off of the end word is found. By using recursion to test each set of words created, we were able to find the first word ladder that could be created. We considered designing the program in a way that would create four different sets of words and recursively find the ladder through each set. This way the set that returned the shortest ladder could be used as the solution. However from a programmer's

perspective it would be more challenging to determine which four indexes would be used during each recursive iteration, and additional methods would be required to compare the size of each word ladder returned. Even though this is more challenging, if organized properly, this can be a future enhancement that would allow the user to not only determine if a word ladder exists but also view shorter word ladders. Our design was also able to properly use inheritance when throwing errors. Instead of calling if statements or causing the program to crash, our design incorporated throw statements that would state how the user input was incorrect, and then gracefully continue to the next input statement.