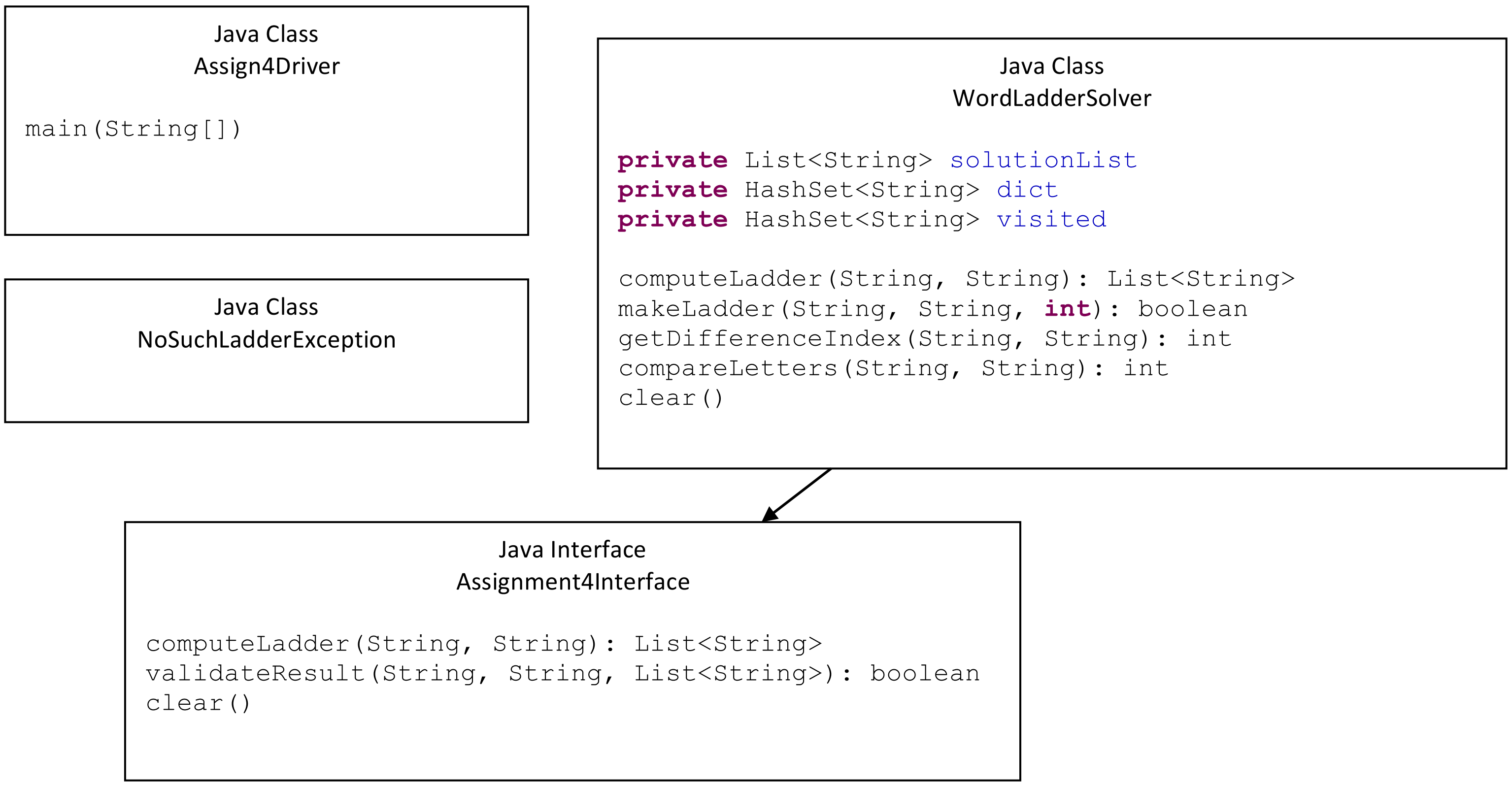
Assignment 4 IPO Diagram

Input	Process	Output
Word pairs	-check for proper format -parse each word -check for word ladder solution -prepare word ladder solution for output -clear previous solution to prepare for next word pair if any	-display error message for improper input format or improper word -display word ladder solution or no solution
Dictionary	-process file based off assignment 4 guidelines -store dictionary into hashset stored as member variable in WordLadderSolver	None

Assignment 4 UML



```
main(String[])
```

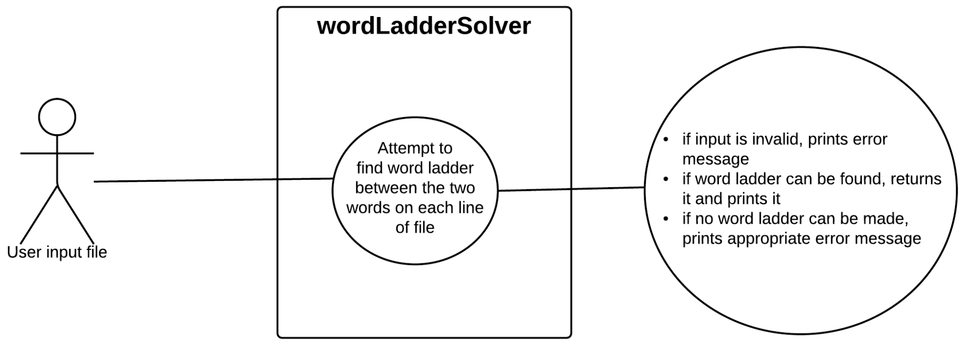
```
private List<String> solutionList
private HashSet<String> dict
private HashSet<String> visited

computeLadder(String, String): List<String>
makeLadder(String, String, int): boolean
getDifferenceIndex(String, String): int
compareLetters(String, String): int
clear()
```

```
computeLadder(String, String): List<String>
validateResult(String, String, List<String>): boolean
clear()
```

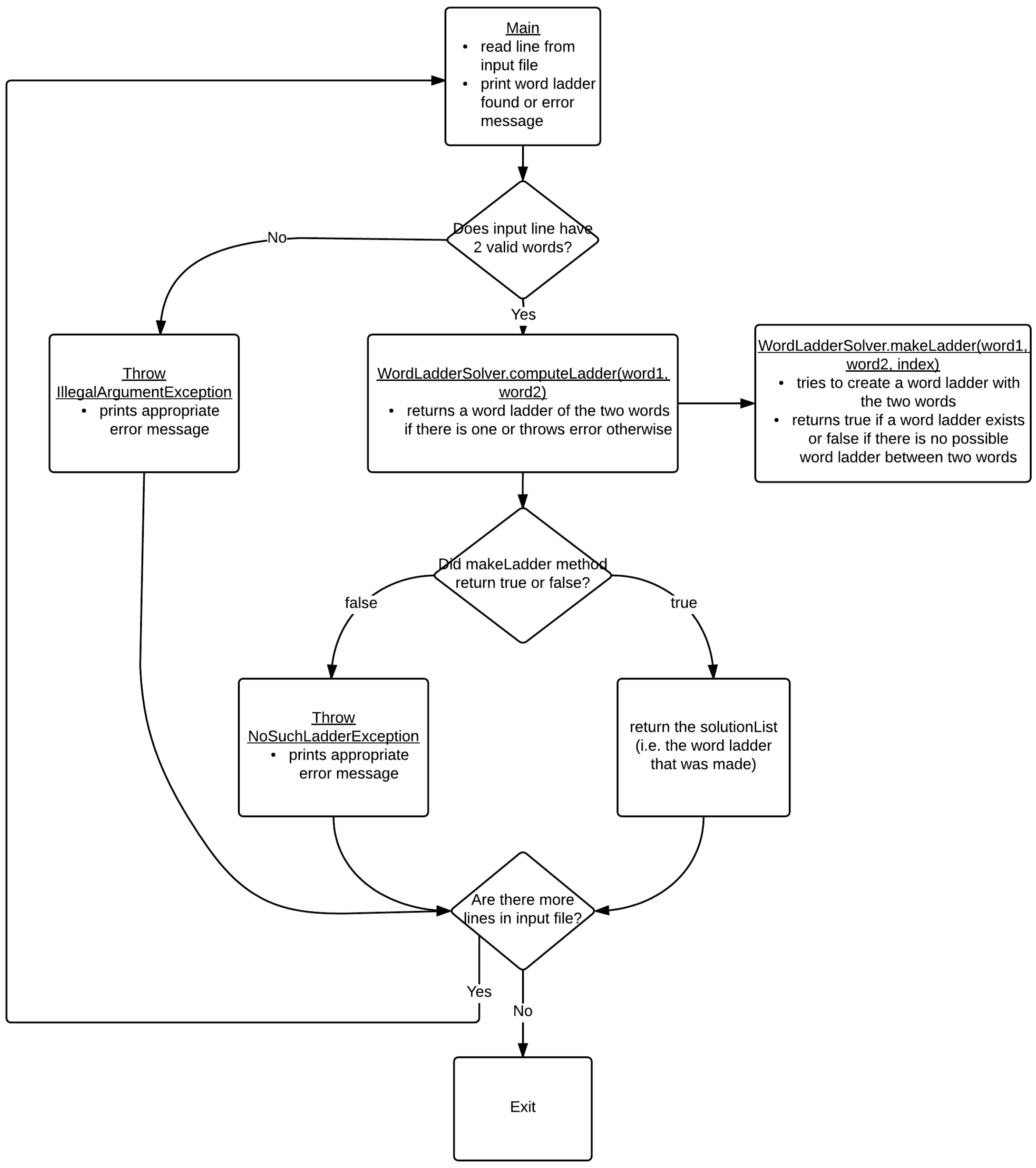
The algorithm needed for the driver logic (main method)

The main method first checks for the appropriate number of command line arguments. If this fails, the program exits after outputting an error message. Then it initializes a WordLadderSolver object with the appropriate dictionary input. Then it begins to read the word pair inputs. The main method checks for proper syntactical formatting when parsing the word pairs. The WordLadderSolver then takes the pair to find its word ladder if one exists. The main takes these results and outputs the ladder if there is one or an error message as appropriate. This process is repeated until there are no more lines in input file.

Use Case Diagram


wordLadderSolver
User input file
Attempt to find word ladder between the two words on each line of file
• if input is invalid, prints error message
• if word ladder can be found, returns it and prints it
• if no word ladder can be made, prints appropriate error message

Block Diagram


Main
• read line from input file
• print word ladder found or error message
Does input line have 2 valid words?
No
Yes
Throw IllegalArgumentException
• prints appropriate error message
WordLadderSolver.computeLadder(word1, word2)
• returns a word ladder of the two words if there is one or throws error otherwise
WordLadderSolver.makeLadder(word1, word2, index)
• tries to create a word ladder with the two words
• returns true if a word ladder exists or false if there is no possible word ladder between two words
Did makeLadder method return true or false?
false
true
Throw NoSuchLadderException
• prints appropriate error message
return the solutionList (i.e. the word ladder that was made)
Are there more lines in input file?
Yes
No
Exit

Design Rationale

Our program is a word ladder solver, and there aren't really any real world objects that this models, other than a computational solver. We solved the problem using a depth-first search and recursion, but an alternative is to use a breadth-first search in which there is no recursion. One advantage is that if recursion is hard to understand for you, then breadth-first is more straightforward and easier to debug. However, recursion usually simplifies the amount of code needed to solve the problem. From a user perspective, the speed of the two searches may differ, so one might be faster than the other for the user, and efficiency is always an advantage. The program could be optimized to have a method that finds the shortest word ladder out of all possible word ladders between two words, but this would likely require longer computing times. Our design makes sure that the scope of methods is appropriate, encapsulates the word ladder solver class and allows for flexibility in the dictionary used to find word ladders as well as the user input file used.