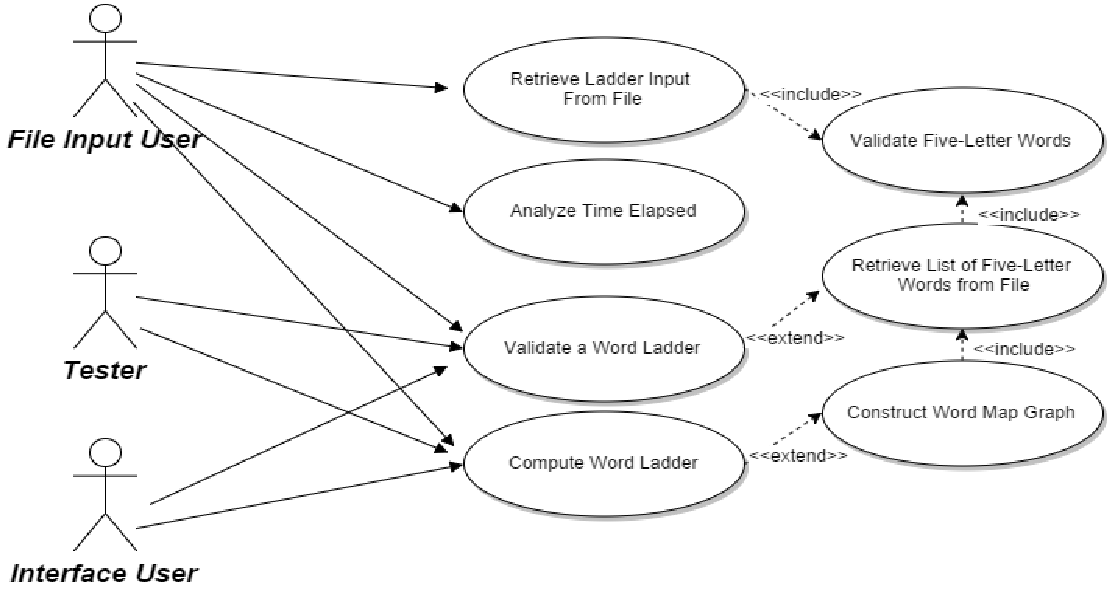
Cooper Raterink, Brandon Arindell
Cdr2678, bja733
Lab Section: Friday 2-3:30pm with Mehtaab

Main Driver Algorithm:

```
Start a stopwatch
Initialize ArrayList of five letter words by getting them from the file given to us on Canvas (file needs to be in src folder)
Initialize a WordMap graph object from this list of words
Initialize a WordLadderSolver object from the word map graph
Record the elapsed time using the stopwatch, and then reset and restart it
Try to open the input file specified in args for reading
If there's an error reading the file, print a message explaining the error and exit.
Otherwise, read through the input file line-by-line and for each line:
    Try to:
        Parse the input line for start and end five-letter words
        Compute a word ladder from the start and end words using the WordLadderSolver object's computeLadder(startWord, endWord) method
        Check if the result is a correct word ladder using the WordLadderSolver object's validateResult(startWord, endWord, wordLadder) method
        Print a message showing the found word ladder and whether that word ladder is correct
    If there's an InvalidInputException: (when the input words aren't valid)
        Print a message explaining the invalid input
    If there's a NoSuchLadderException: (when a ladder cannot be computed between two words)
        Print a message explaining that a ladder cannot be computed
Stop the stopwatch and record the elapsed time again
Print a message showing the set-up and computation times for the program
```

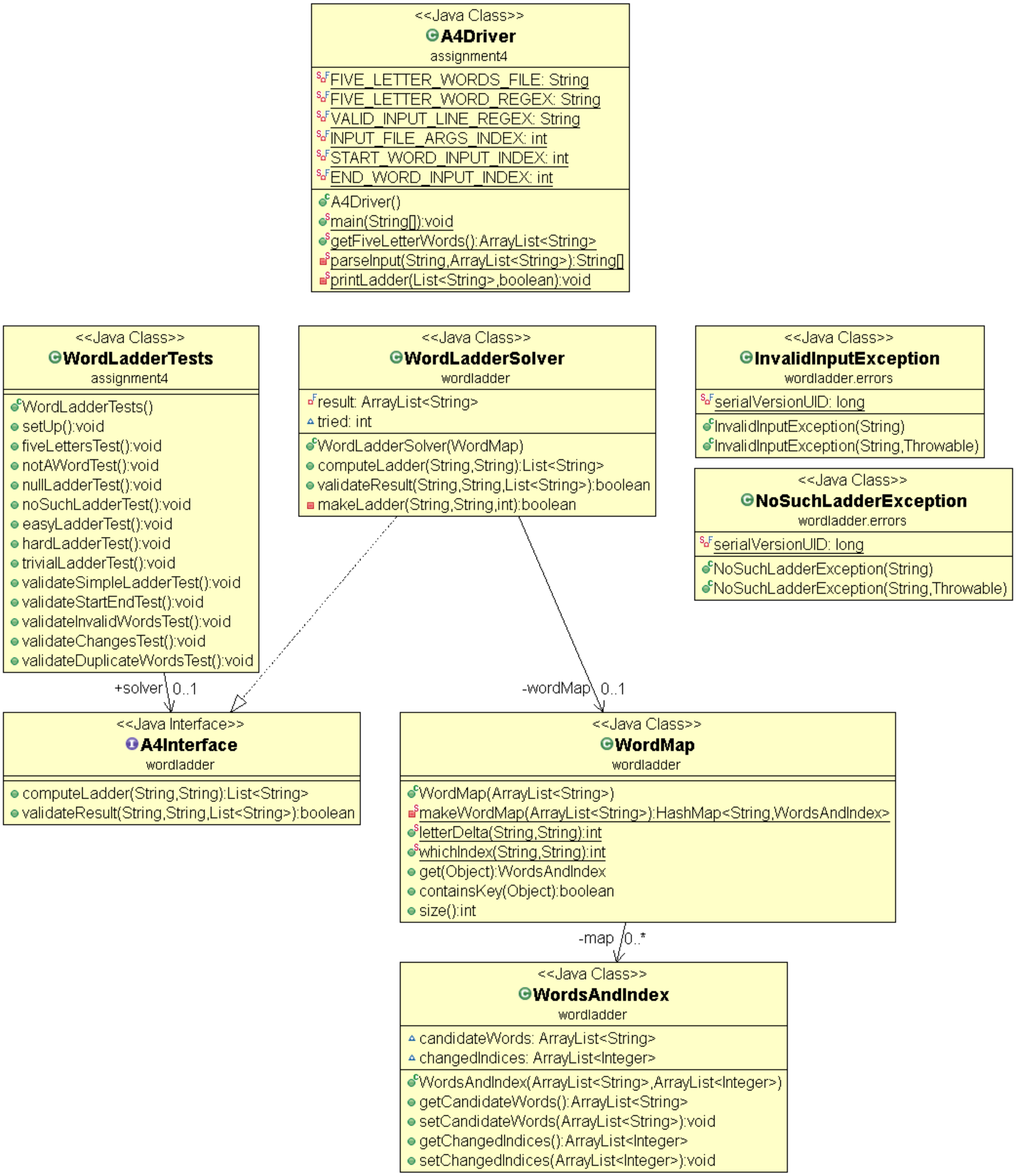
Use Case Diagram:



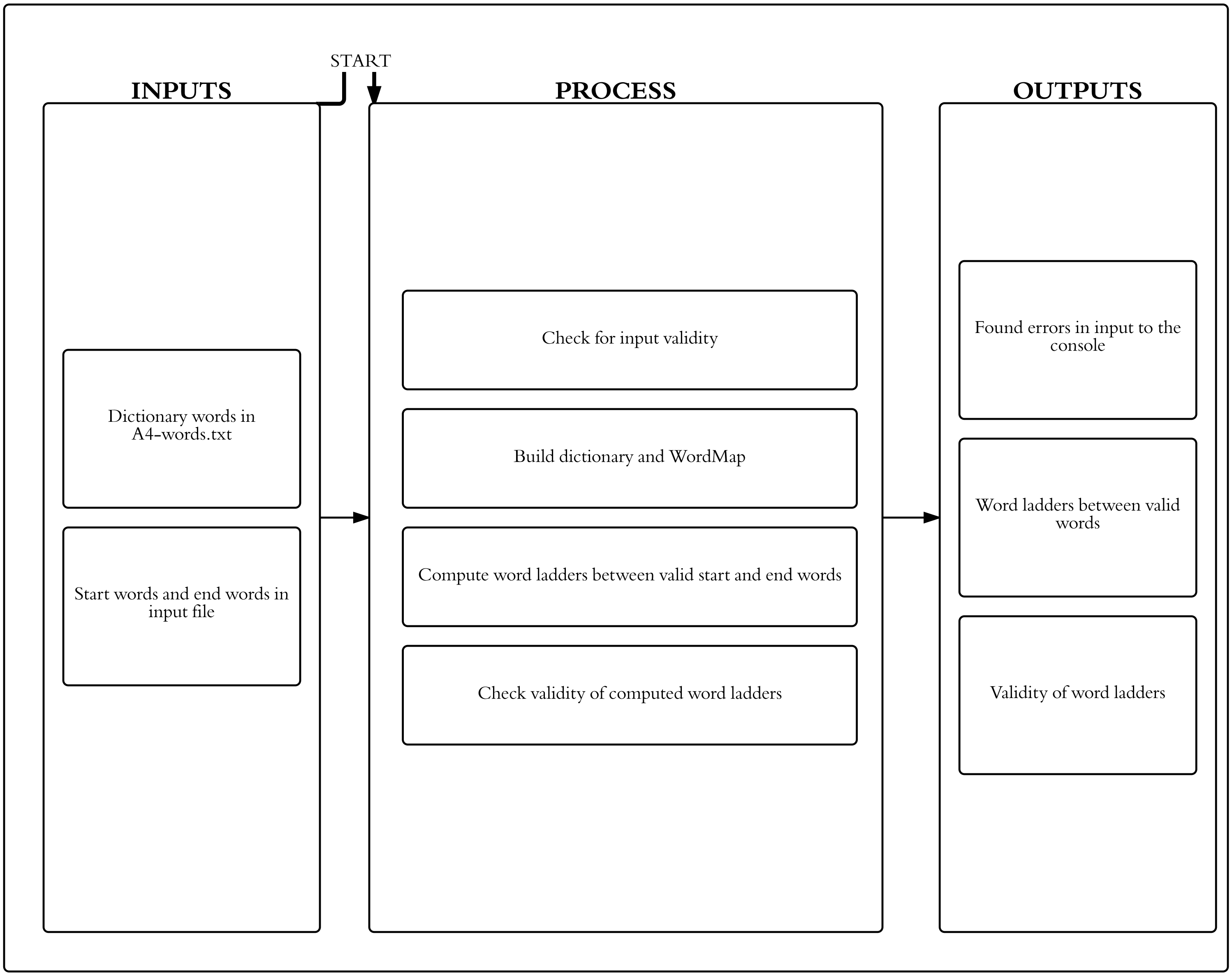
UML Diagram:

UML Diagram

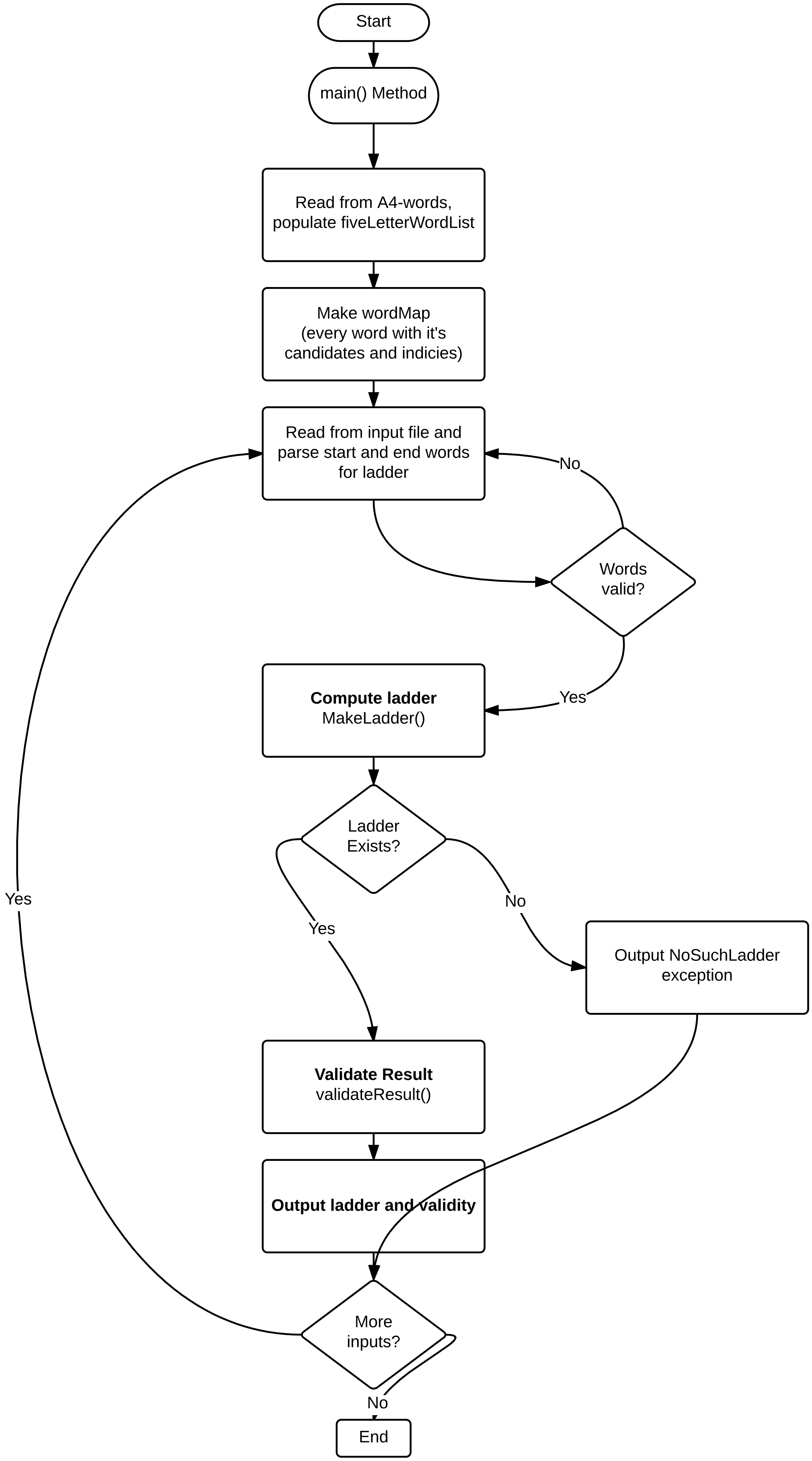
Cooper Raterink, Brandon Arindell
Cdr2678, bja733
Lab Section: Friday 2-3:30pm with Mehtaab


<<Java Class>>
A4Driver
assignment4
FIVE_LETTER_WORDS_FILE: String
FIVE_LETTER_WORD_REGEX: String
VALID_INPUT_LINE_REGEX: String
INPUT_FILE_ARGS_INDEX: int
START_WORD_INPUT_INDEX: int
END_WORD_INPUT_INDEX: int
A4Driver()
main(String[]):void
getFiveLetterWords():ArrayList<String>
parseInput(String,ArrayList<String>):String[]
printLadder(List<String>,boolean):void
<<Java Class>>
WordLadderTests
assignment4
WordLadderTests()
setUp():void
fiveLettersTest():void
notAWordTest():void
nullLadderTest():void
noSuchLadderTest():void
easyLadderTest():void
hardLadderTest():void
trivialLadderTest():void
validateSimpleLadderTest():void
validateStartEndTest():void
validateInvalidWordsTest():void
validateChangesTest():void
validateDuplicateWordsTest():void
<<Java Class>>
WordLadderSolver
wordladder
result: ArrayList<String>
tried: int
WordLadderSolver(WordMap)
computeLadder(String,String):List<String>
validateResult(String,String,List<String>):boolean
makeLadder(String,String,int):boolean
<<Java Class>>
InvalidInputException
wordladder.errors
serialVersionUID: long
InvalidInputException(String)
InvalidInputException(String,Throwable)
<<Java Class>>
NoSuchLadderException
wordladder.errors
serialVersionUID: long
NoSuchLadderException(String)
NoSuchLadderException(String,Throwable)
+solver
0..1
-wordMap
0..1
<<Java Interface>>
A4Interface
wordladder
computeLadder(String,String):List<String>
validateResult(String,String,List<String>):boolean
<<Java Class>>
WordMap
wordladder
WordMap(ArrayList<String>)
makeWordMap(ArrayList<String>):HashMap<String,WordsAndIndex>
letterDelta(String,String):int
whichIndex(String,String):int
get(Object):WordsAndIndex
containsKey(Object):boolean
size():int
-map
0..*
<<Java Class>>
WordsAndIndex
wordladder
candidateWords: ArrayList<String>
changedIndices: ArrayList<Integer>
WordsAndIndex(ArrayList<String>,ArrayList<Integer>)
getCandidateWords():ArrayList<String>
setCandidateWords(ArrayList<String>):void
getChangedIndices():ArrayList<Integer>
setChangedIndices(ArrayList<Integer>):void

System IPO Diagram


START
INPUTS
PROCESS
OUTPUTS
Dictionary words in A4-words.txt
Start words and end words in input file
Check for input validity
Build dictionary and WordMap
Compute word ladders between valid start and end words
Check validity of computed word ladders
Found errors in input to the console
Word ladders between valid words
Validity of word ladders

Functional Block Diagram


Start
main() Method
Read from A4-words, populate fiveLetterWordList
Make wordMap (every word with it's candidates and indicies)
Read from input file and parse start and end words for ladder
Words valid?
No
Yes
Compute ladder
MakeLadder()
Ladder Exists?
Yes
No
Output NoSuchLadder exception
Validate Result
validateResult()
Output ladder and validity
More inputs?
Yes
No
End

EE 422C March 6, 2016
Brandon Arrindell, Cooper Raterink

Design Rationale

To compute word ladder recursively, we focused on doing a major part of the computation up front. Namely, we created a HashMap called WordMap, which takes in the dictionary and finds words that differ by one letter, called candidates, which are stored in an ArrayList. WordMap is a HashMap of words, their candidates, and the indices of the letter changed. We achieved this by created a class called wordsAndIndicies that contains an ArrayList of candidates and an ArrayList of which index was changed. This way, we do the majority of the work before we start to compute ladder and thus we don't have to run through the dictionary looking for candidates every time we find a new word, we just look-up the word in the WordMap and instantly get it's candidates. Our MakeLadder algorithm runs by checking a given word's candidates, sorting them by similarity to the endWord, and then going through them one-by-one until a match is found. A match is when the word is one letter different to the previous word and that letter is not in the same place as the previous changed letter (the first time we set changed index to -1). Once a match is found, we add the word to ladder and repeat the process. We try candidates most similar to the final word first so that we have a higher chance of finding the final word, and thus a shorter ladder. If the process is repeated and there is no match for the final word, we remove the word from the ladder and check other candidates until a match is found. If we have exhausted all candidates, we return false and ComputeLadder throws a NoSuchLadderException. We considered alternatives like doing computation at run-time and looking for a new candidate every time, but this approach would be far more time and process demanding. Form the users perspective, we did not want it to take long for the program to compute a word ladder.

EE 422C March 6, 2016
Brandon Arrindell, Cooper Raterink

Testing

Word Ladder Testing	
fiveLettersTest()	validateSimpleLadderTest()
notAWordTest()	validateStartEndTest()
nullLadderTest()	validateInvalidWordsTest()
noSuchLadderTest()	validateChangesTest()
easyLadderTest()	validateDuplicateWordsTest()
hardLadderTest()	
trivialLadderTest()	
stressTest()	

Black Box Testing: The program takes in a dictionary file and an input file and outputs word ladders between inputs. Black Box testing assumes we have no knowledge of how the program operates, but do have knowledge of the inputs and expected outputs. To test the program as a "black box" we created a series of start words and end words, some with ladders and some without, and tested that our program operated as expected. This file is called *BasicTestCases.txt* in our project. These tests give us an idea of whether the program operates efficiently and correctly and that input error handling is working as expected (edge cases, bad inputs, etc.)

White Box Testing: White Box testing assumes we have full knowledge of how the program operates (its structure, what exceptions exists and when should they be thrown, etc). To test the full coverage of our program and all of it's parts, we created JUnit test cases to test some of the key functionality. In the table above you can see all of the test cases that exist. The table on the left tests the expected behavior of the computeLadder() and MakeLadder() methods. With the tests above, we are testing the main recursive algorithm for finding ladders, the HashMap creation of words and their candidates, and the validation method.

fiveLettersTest() - Tests that a NoSuchLadderExcpetion is thrown when input words are more or less than five letters.
notAWordTest() - Tests that a NoSuchLadderExcpetion is thrown when the one of the input word does not exist in the dictionary.
nullLadderTest() - Tests that NoSuchLadderExcpetion is thrown when the inputs are null.
easyLadderTest() - Tests that NoSuchLadderExcpetion is *not* thrown (AKA there is a ladder) between two similar words.
hardLadderTest() - Tests that NoSuchLadderExcpetion is *not* thrown (AKA there is a ladder) between two very different words.
trivialLadderTest() - Tests that NoSuchLadderExcpetion is *not* thrown when the start and end word are the same.
sterssTest - Ignoring NoSuchLadderExcpetions, this test creates ladders between the first 200 words in the dictionary, two at a time. Useful for testing reasonable time and ensuring that the program does not crash under heavy load.
validateSimpleLadderTest() - A correct ladder is inputted and we check that the validate method works.
validateStartEndTest() - Should not validate a ladder with incorrect start/end words.
validateInvalidWordsTest() - Should not validate a ladder with any invalid words.
validateChangesTest() - Should not validate a ladder with sequential words not differing by exactly one letter.
validateDuplicateWordsTest() - Should not validate a ladder with duplicate words.