

Input

Dictionary Words

Word Ladder Pairs

Process

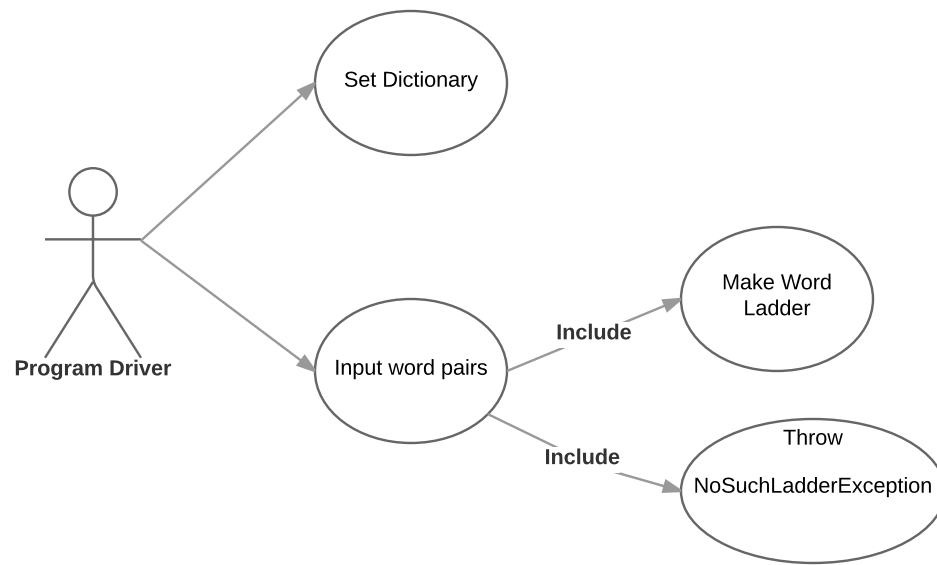
- Create dictionary HashMap
- Create list of word ladder pairs
- Check for word pairs in dictionary
- Recursively search for word ladders by changing single characters of words.
- Return error message if no ladder found or if one or both words for ladder aren't in Dictionary.

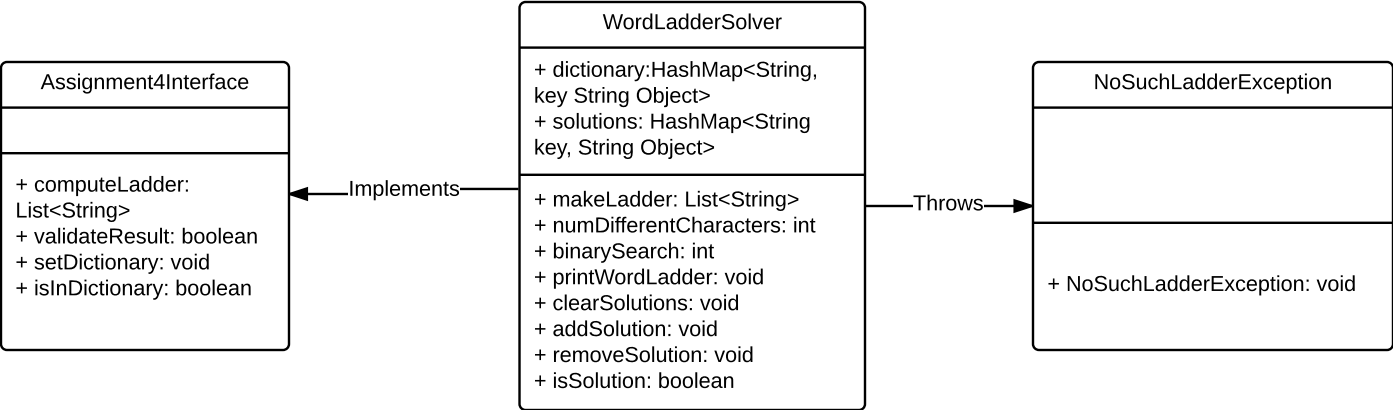
Output

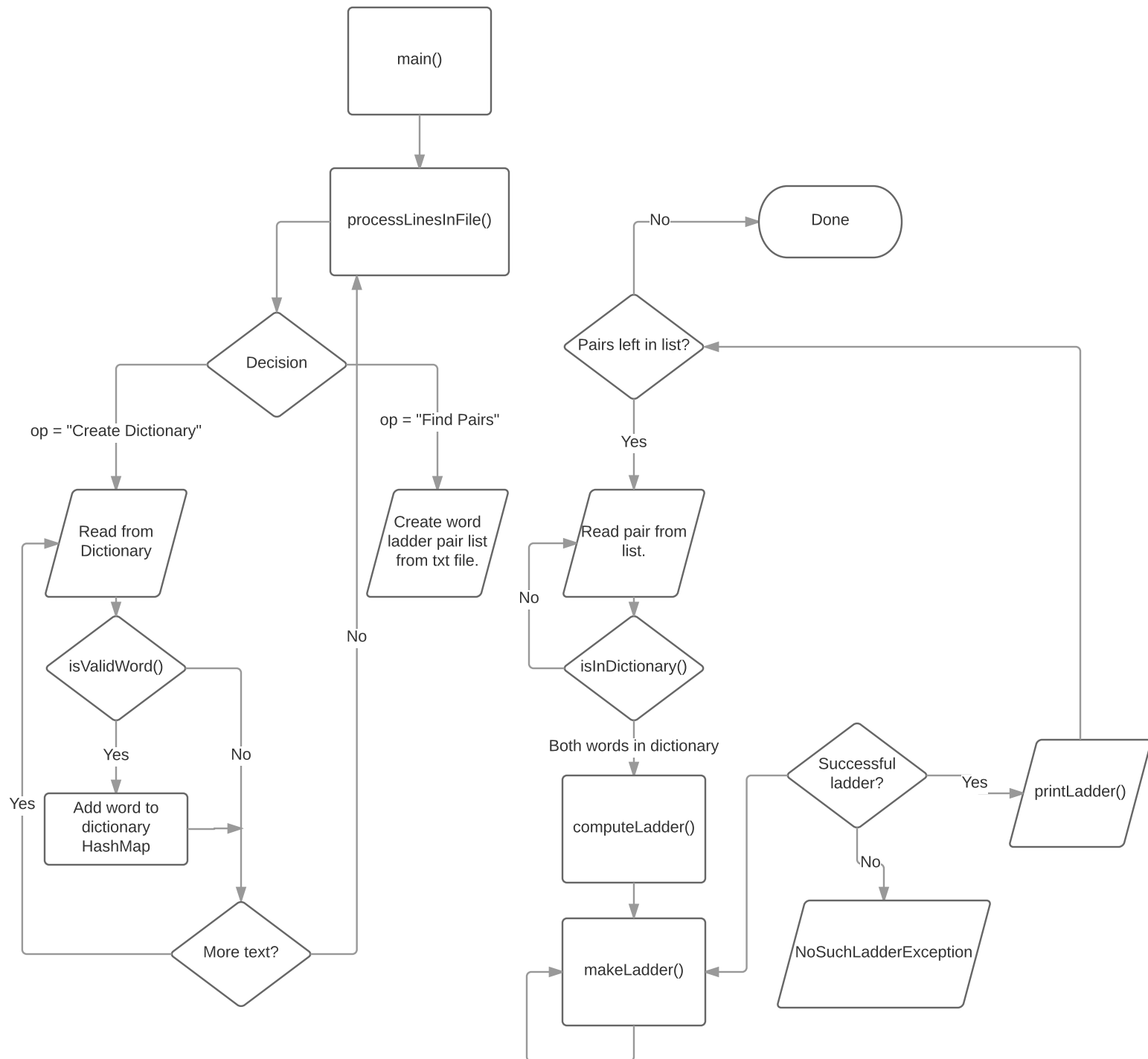
Word Ladder

Invalid Start/End Word

No Word Ladder Found Exception







Main() Algorithm

Create new WordLadderSolver object.

Access dictionary .dat file

processLinesInFile(<path to dictionary file>, "Create Dictionary"): returns list of strings

While text left in dictionary

Read line from dictionary

If(line from dictionary = valid word)

Add word to list of string for words

Add list from processLinesInFile call to WordLadderSolver dictionary.

processLinesinFile(<path to input pairs file>, "Create Pairs"): returns list of strings

While text left in file

Read line from file

Add to list

Iterate through pairs list, parsing pairs and checking if they are in the dictionary

If pair is in dictionary, wordLadder = wordLadderSolver.computeLadder(word 1, word 2)

Check if wordLadder is valid via validateLadder()

If valid, print out ladder

Else, NoSuchLadderException

Design Rationale

In order to compute word ladders, we used two HashMaps: one to store both a dictionary file containing all valid five letter words according to input and a second to store solutions as we processed them in order to cut down on the time to check potential new rungs of the WordLadder against words already used in the ladder. The use of a HashMap simulates a dictionary only in that it allows for the storage of words. Rather than storing the words in sequential order, however, the data structure stores words by hash so we can search through the dictionary in constant time rather than in logarithmic time using a binary search (which is typically the fastest way to search through a dictionary). We did consider using Lists of Strings for both the dictionary and the solutions list, but doing so would give us unnecessarily high time complexity in searching our dictionary ($O(n)$ time for dictionary search) and a quadratic time complexity for the total amounts of checks of new word ladder rungs against solutions ($O(n^2)$ for new word solution checking). The use of HashMaps cut down on time drastically because searches in our dictionary and solutions database can be performed in constant time. In terms of adaptability, the dictionary could feasibly be modified to contain definitions of words. Our design follows the rules of good OOD in that it modularizes functions for populating our dictionary apart from the word ladder computation process such that the logic of our code is fairly easy to design and debug.