Breadth-First Search with Memory

Problem Overview

When I was a sophomore in college, I found myself in what would turn out to be the most influential course that I've ever taken. Everything about the course was in some way exceptional, but the professor particularly so. She made what could have been an ordinary English Composition class into an examination of thinking. One of the recurring themes in the class was "making connections"—the process that allows us to associate things and see relationships among different things. So, in honor and memory of O.A.L., this assignment is all about making connections.

The focus of the assignment is to implement a word connection game that has been played in one variation or another for over 130 years. The object of the game is to transform a start word into an end word of the same length by a sequence of steps, each of which consists of a one-letter change to the current word that results in another legal word. Charles Lutwidge Dodsgon (Lewis Carroll) invented this game and called it "Doublets." It's now more commonly known as "Word Ladders"[1].

Consider the following examples.

clash, flash, flask, flack, flock, clock, crock, crook, croon, crown, clown
cat, can, con, cog, dog
cat, bat, eat, fat, gat, hat

Each is a valid word ladder from the start word to the end word since the start and end words are the same length and each word in between is exactly one letter different from the previous word.

The game is usually played so that each player tries to find the *shortest* word ladder between two words. The shortest ladder would, of course, depend on the *lexicon*, or list of words, being used for the game. Using the SOWPODS word list (see Provided Resources), word ladders with minimum length for the start-end pairs above would be:

clash, class, claws, clows, clown
cat, cot, dot, dog
cat, hat

Requirements

The interface `WordLadderGame` defines several methods associated with the generation of word ladders. The class `Doublets` provides an implementation of this interface. You must provide a correct implementation of the `Doublets` class by completing its constructor and providing a correct implementation of each `WordLadderGame` method. You must meet all the requirements specified and implied by the Javadoc comments in these files. You may add as many private methods as you would like, but you can't add any public methods. You may add as many nested classes as you would like, but you can't add any top-level classes. Although you may import other classes that are part of the JDK, the imports already provided are the suggested ones that you will need.

[1]`https://en.wikipedia.org/wiki/Word_ladder`

Provided Resources

You are provided with the following resources as part of the assignment.

- The `WordLadderGame` interface.
- The `Doublets` class (a shell, not complete).
- A jar file, `WordLists.jar`. This jar file contains the following text files that you are allowed to use as lexicons for the Doublets class: `CSW12.txt` (the word list used in international Scrabble tournaments), `OWL.txt` (the word list used in North American Scrabble tournaments), `sowpods.txt` (SOWPODS—a commonly used combination of the CSW list and the OWL list), `words.txt` (the word list supplied in UNIX distributions), `small.txt` (a relatively small subset of the SOWPODS list), `tiny.txt` (a truly tiny word list), and `names.txt` (a list of first names used in the United States from 1880-2012, provided by the U.S. Social Security Administration[2]).
- A sample client, `ExampleClient.java`. This driver class illustrates basic calls to the methods, and it also demonstrates how to associate a text file contained in `WordList.jar` with an `InputStream` object.

[2] `http://www.ssa.gov/OACT/babynames/limits.html`