CS 251 Intermediate Programming
Lab 7: Evil Hangman*

Brooke Chenoweth

Fall 2018

1 Cheating at Hangman

It's hard to write computer programs to play games. When we as humans sit down to play a game, we can draw on past experience, adapt to our opponents' strategies, and learn from our mistakes. Computers, on the other hand, blindly follow a preset algorithm that (hopefully) causes it to act somewhat intelligently. Though computers have bested their human masters in some games, most notably checkers and chess, the programs that do so often draw on hundreds of years of human game experience and use extraordinarily complex algorithms and optimizations to outcalculate their opponents.

While there are many viable strategies for building competitive computer game players, there is one approach that has been fairly neglected in modern research – cheating. Why spend all the effort trying to teach a computer the nuances of strategy when you can simply write a program to play dirty and win handily all the time? In this assignment, you will build a mischievous program that bends the rules of Hangman to trounce its human opponent time and time again. In doing so, you'll cement your skills with abstract data types and iterators, and will hone your general programming savvy. Plus, you'll end up with a piece of software which will be highly entertaining. At least, from your perspective. :)

In case you aren't familiar with the game Hangman, the rules are as follows:

1. One player chooses a secret word, then writes out a number of dashes equal to the word length.
2. The other player begins guessing letters. Whenever she guesses a letter contained in the hidden word, the first player reveals each instance of that letter in the word. Otherwise, the guess is wrong.
3. The game ends either when all the letters in the word have been revealed or when the guesser has run out of guesses.

Fundamental to the game is the fact the first player accurately represents the word she has chosen. That way, when the other players guess letters, she can reveal whether that

*Based on the project at `http://nifty.stanford.edu/2011/schwarz-evil-hangman/`

letter is in the word. But what happens if the player doesn't do this? This gives the player who chooses the hidden word an enormous advantage. For example, suppose that you're the player trying to guess the word, and at some point you end up revealing letters until you arrive at this point with only one guess remaining:

DO-BLE

There are only two words in the English language that match this pattern: "doable" and "double". If the player who chose the hidden word is playing fairly, then you have a fifty-fifty chance of winning this game if you guess 'A' or 'U' as the missing letter. However, if your opponent is cheating and hasn't actually committed to either word, then there is no possible way you can win this game. No matter what letter you guess, your opponent can claim that she had picked the other word, and you will lose the game. That is, if you guess that the word is "doable", she can pretend that she committed to "double" the whole time, and vice-versa.

Let's illustrate this technique with an example. Suppose that you are playing Hangman and it's your turn to choose a word, which we'll assume is of length four. Rather than committing to a secret word, you instead compile a list of every four-letter word in the English language. For simplicity, let's assume that English only has a few four-letter words, all of which are reprinted here:

ALLY BETA COOL DEAL ELSE FLEW GOOD HOPE IBEX

Now, suppose that your opponent guesses the letter 'E'. You now need to tell your opponent which letters in the word you've "picked" are E's. Of course, you haven't picked a word, and so you have multiple options about where you reveal the E's.

If you'll notice, every word in your word list falls into one of five "word families":

- `----`, which contains the word ALLY, COOL, and GOOD.
- `-E--`, containing BETA and DEAL.
- `--E-`, containing FLEW and IBEX.
- `E--E`, containing ELSE.
- `---E`, containing HOPE.

Since the letters you reveal have to correspond to some word in your word list, you can choose to reveal any one of the above five families. There are many ways to pick which family to reveal – perhaps you want to steer your opponent toward a smaller family with more obscure words, or toward a larger family in the hopes of keeping your options open. In this assignment, in the interests of simplicity, we'll adopt the latter approach and always choose the largest of the remaining word families. In this case, it means that you should pick the family `----`. This reduces your word list down to

ALLY COOL GOOD

and since you didn't reveal any letters, you would tell your opponent that his guess was wrong.

Let's see a few more examples of this strategy. Given this three-word word list, if your opponent guesses the letter O, then you would break your word list down into two families:

- `-OO-`, containing COOL and GOOD.
- `----`, containing ALLY.

The first of these families is larger than the second, and so you choose it, revealing two O's in the word and reducing your list down to

COOL GOOD

But what happens if your opponent guesses a letter that doesn't appear anywhere in your word list? For example, what happens if your opponent now guesses 'T'? This isn't a problem. If you try splitting these words apart into word families, you'll find that there's only one family – the family `----` in which T appears nowhere and which contains both COOL and GOOD. Since there is only one word family here, it's trivially the largest family, and by picking it you'd maintain the word list you already had.

There are two possible outcomes of this game. First, your opponent might be smart enough to pare the word list down to one word and then guess what that word is. In this case, you should congratulate him – that's an impressive feat considering the scheming you were up to! Second, and by far the most common case, your opponent will be completely stumped and will run out of guesses. When this happens, you can pick any word you'd like from your list and say it's the word that you had chosen all along. The beauty of this setup is that your opponent will have no way of knowing that you were dodging guesses the whole time – it looks like you simply picked an unusual word and stuck with it the whole way.

2 Program Description

For this project, you will create two implementations of hangman game internals, one that will play the standard game and an "evil" one that cheats as described earlier.

2.1 Provided Files

I am providing you with the following files, which you should not change.

- `HangmanInterface.java` contains the interface which you will need to implement.
- `Hangman.java` constructs the appropriate game implementation and uses it to play a game of hangman with the user.

2.2 Hangman Implementations

You will submit at least two files:

- `FairHangman.java`, to play the standard "fair" hangman game
- `EvilHangman.java`, to play an "evil" game

Each of these classes should have a constructor that takes a `String` of the name of a file containing a list of words to use as the dictionary for the game. You may assume each word appears on a separate line. (You may have other constructors as well, but the filename one is what I'll be testing.)

You may realize that both game implementations will have a lot of common functionality (reading the word list from the dictionary file, tracking how many guesses are left, which letters have been guessed, etc.) which you should not repeat. I suggest that you either make a common abstract parent class to contain these common items or you make the evil implementation extend or use the fair one.

3 Turning in your assignment

Submit your `FairHangman.java`, `EvilHangman.java`, and any other source files you wrote for the project to UNM Learn. Do not submit `HangmanInterface.java` or `Hangman.java`, since you should not have changed them.