CS102 **HW01** - The Hangman Game Spring 2021

This homework asks you to implement a simple Hangman game based on the design we came up with in class. This will require you to recall and practice many of the basic programming skills you learnt in CS101, as well as introduce you to some of the joys and frustrations of teamwork!

You will already have worked in groups during the Requirements and UI design stages. Following the Detailed-Design stage, you will have divided the Implementation (the actual coding) into 5 separate coding tasks, with each group being assigned to implement just one of them. The last part of the exercise involves integrating the 5 separate pieces of code to produce a working program. You should do this individually, recording the difficulties you face and reflecting on how you might avoid similar problems in the future. These reflections must be uploaded together with your code.

Requirements ~ *(description)*

In the game of Hangman the player must find a word chosen secretly by the program. The player can try one letter at a time and the program says how many times and where the letter appears in the secret word. If the player tries more than a certain number of incorrect letters (i.e. letters which do not appear in the secret word), they lose the game and the program tells them the secret word, otherwise they continue until they uncover the complete word.

Usually, players are able to see the partially formed word made by the letters so far correctly guessed, any unknown letters being left blank. They may also be able to see the set of all letters that might be in the word (with those already used possibly being removed), as well as the number of incorrect tries made so far (this is often shown graphically, by the number of visible body parts of a cartoon character which is hung when complete -hence the name of the game!)

UI-Design

For this exercise you will produce a simple text-only console application (as in CS101), leaving the fancy graphical interfaces for later! The controller and the view will be the main program.

Detailed Design

The (UML-like class diagram of the) design we produced in class is below. This is what you should implement Note that + means public and - means private.

class Hangman

- properties
  - secretWord : StringBuffer
  - allLetters : StringBuffer
  - usedletters : StringBuffer
  - numberOfIncorrectTries : int
  - maxAllowedIncorrectTries : int
  - knownSoFar : StringBuffer // secretWord but with chars not yet found blanked out
- constructors
  - + Hangman() // default max 6 incorrect tries, English alphabet, // chooses secretWord, etc.
- methods
  - + getAllLetters() : String
  - + getUsedLetters() : String
  - + getNumOfIncorrectTries() : int
  - + getMaxAllowedIncorrectTries() : int
  - + getKnownSoFar() : String // returns partial word formed with known letters only
  - + tryThis(letter) : int // returns number of occurrences of letter in secretWord
  - + isGameOver() : boolean
  - + hasLost() : boolean
  - - chooseSecretWord() // initially use fixed list, called from constructor

---

Implementation

Tasks

The following division of work has been decided upon and allocated to specific groups during class:

1. main method - plays the game by creating instance of Hangman class & calling its methods.
2. constructor for Hangman class
3. tryThis method
4. chooseSecretWord method
5. Hangman class (but with the bodies of the constructor, chooseSecretWord, & tryThis methods being empty.)

Additional Notes:

main method:

play the game, by creating an instance of Hangman class and allowing the user to interact with it (in text mode using the keyboard only.) The program must repeatedly get letters from the user and try them...

constructor:

set all letters to English alphabet, set max allowed incorrect tries to 6, no of incorrect tries to 0, used letters to empty set, secret word to result of calling choose secret word method, knownsofar to StringBuffer of same length as secret word, but all characters are stars ('*').

tryThis(letter):

returns number of times the letter occurs in the secret word. Adds letter to used letters. Updates known so far to show the letter at each position it exists in secret word. If the letter is not in secret word then increment number of incorrect tries. Method should return error values to indicate if the letter is not valid (-1), if the letter was already used (-2), and indicate if game over (-3).

chooseSecretWord:

returns a word chosen at random from a fixed array of words (defined in program code.) Also write another version of the method that reads the set of words from a text file and selects one from that.

Hangman class

The actual Hangman class, but without the constructor, the chooseSecretWord method, and the tryThis method, being implemented.

Individual Homework (graded)

Integration & Testing of Hangman code

As soon as it is available, you should download the complete set of code pieces for your section and attempt to get the whole program working. Please **keep a record of what you do** --in particular, record what went wrong and how you went about fixing it-- but also think about and record why the problems occurred in the first place and what might be done to avoid similar difficulties in the future. Try to **keep track of the amount of time** you spend on this exercise.

All this information should be written in a text file named **reflections.txt**
which must begin with the following information:

- the course code & section number,
- your full name and id number,
- the date,
- your estimate of the time spent on this exercise,
- and the status of your code: working/not-working.

and be included in the root of the *zip* file you upload to Moodle.

Please upload your work *(the Java code and your reflections)*
to the Moodle activity,
(i.e. HW01)
naming your file: **HW01_*yourname*.zip**
(where *yourname* is hopefully obvious!)

Good luck!
