CIS351

Homework 3

Academic Integrity

- Remember the Academic Integrity Policies, of this course.

Objectives:

- Writing moderately complicated algorithms.
- Using loops of various kinds.
- Reinforcing what you have already learned about integer arithmetic and nesting of if statements.
- Finally, it is intended to demonstrate the value of reusable methods and classes

Download Materials

For this assignment you will implement six methods to support the development of "Word Guess" games like Hangman. For this homework, you need to download the following starter code from Blackboard.

- WordGuess.java - **create from scratch**
- Dictionary.java - Code for reading words from Words.txt (DO NOT MODIFY)
- Hangman.java- Driver/User-interface (DO NOT MODIFY)
- Sample.txt - Sample output from one the game we are developing
- Words.txt- A list of guessable words
- PA4.jar - Execute this jar file to see how a correct implementation of the solution will look like
- Words.csv - This is the same as Words.txt
- WordGuessTest.java - This is the JUnit test class

Background

When Hangman.java is executed, a player is presented with a series of blanks representing a hidden word. The player guesses letters; if the letter is in the word, the word is updated to reveal the letter of that guess filled into the correct spots in the word. If the player does not guess a letter in the word, he or she receives a strike. If the player guesses the word (reveals the last letter) in fewer than 6 guesses, they win. If it takes more than 6 guesses, they lose. In this assignment, if a player guesses a letter that they have already guessed, it is not counted as a strike. See the sample output from one such game in Sample.txt.

The last three files are provided *only as an example*, and they will not be used in grading your assignment. You should build and test WordGuess.java incrementally -- as you are writing each method -- rather than trying to test with Hangman.java all at once.

Required Methods for WordGuess.java

The following terms will be used consistently throughout the method descriptions below.

`theWord`
The current word to guess. It will be preserved throughout play.

`userWord`
A pattern that represents the progress the user is making on the word.

`guess`
The letter that the user is guessing.

`strikes`
The number of bad guesses the user has made.

`guesses`
A list of characters that the user has guessed.

Method Specifications

`public static String makeUserWord(String theWord)`
Takes the word the player is trying to guess and generates a pattern of '*' values to represent letters that have not yet been guessed. For example, if the word were "dog" this method would return "***".

`public static boolean isInWord(char guess, String theWord)`
Returns true if the guessed character is in the word, false otherwise. For example, if the guess were 'x' and the word were "xylophone" this method would return true.

`public static String updateUserWord(char guess, String userWord, String theWord)`
Returns a userWord with all occurrences of '*' corresponding to the current guess replaced with that guess. For example, if the word was "fetch" and the user word was "****h" and the user guessed 'e', the return string would be "*e**h".

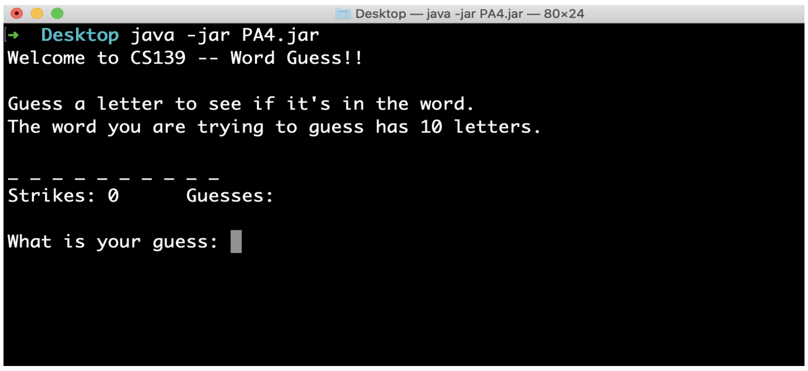
`public static String updateGuesses(String guesses, char guess)`
Updates the list of guesses with the current guess. The update should only add the guess if it does not already exist in the list. The new guess must be placed at the end of the existing list of guesses. For example, if guesses were "tabg" and the current guess were 'f', this method would return "tabgf".

`public static String displayUserWord(String userWord)`
Returns a String that is the userWord with spaces between each letter and each '*' replaced with an '_'. For example, if the userWord is "fe***" this method would return "f e _ _ _". Note that there is no space before the 'f' and after the last '_'.

`public static String displayGuesses(int strikes, String guesses)`
Returns a String in the form "Strikes: %d\tGuesses: %s", with the list of guesses separated by spaces. For example, if there were 3 strikes and guesses were "bcd", this method would return"Strikes: 3\tGuesses: b c d".

How to use the jar file?

You are provided with a jar file, PA4.jar, which you can run as many time as you want to play the game and get an idea about how a correct implementation will behave. NOTE: The output **Guesses** produced by this jar file is sorted, (like if you type the Guesses with the sequence of "dcb", but the output change it to the "bcd" automatically). However, you DO NOT need to sort the guesses in your solution.
You can run the PA4.jar file from your terminal/command prompt. Make sure to have the Words.csv and the jar file in the same folder. The command to to execute jar file: java - jar PA4.jar. Here is how you run the jar file from the terminal/command prompt:



```
→  Desktop java -jar PA4.jar
Welcome to CS139 -- Word Guess!!

Guess a letter to see if it's in the word.
The word you are trying to guess has 10 letters.

_ _ _ _ _ _ _ _ _ _
Strikes: 0      Guesses:

What is your guess: 
```

How to test your solution?

To test the correctness of your implemented methods, you need to have all the given files in the same folder, including `Words.txt`. Now compile them all and execute the Hangman.java. If all your methods are correct, you should be able to see output similar to `Sample.txt`.
Since this assignment uses random number, it will be tricky to reproduce output. However, the provided `Sample.txt` can provide you a Sample output of the game you are developing.

How to use JUnit test class?

Once you submit your solution, we use JUnit test classes to test if your program output is correct. If you are interested you can use `WordGuessTest.java` test files to test your code. These will be the same tests we will be using to test your solution. You can see this video on how to use JUnit to test your program, which is posted in BB -> Content -> Resources.

Grading Criteria

Total points: 100 points

Correctly compiles and executes without any failure - 10 pt
Correctness testcases - 70 pt
Documentation and styling - 10 pt
Filled out Reflection Form - 10 pt

Submission Instructions

First zip the following and then submit the zipped folder in Blackboard

- final version of **WordGuess.java**
- filled out Reflection Form (https://farahman.github.io/reflection-form.pdf).
- filled out cover sheet (https://farahman.github.io/CoverSheet.pdf).

Farzana Rahman / frahman@syr.edu