Assignment 2: Wordle

Description

The purpose for this assignment is to learn how to write Java classes and create a program that is modular and scalable. To do this, you will have to design and implement a command line version of the popular online game Wordle. You are free to use any class and method from the Java 8 library.

The first thing you should do is play Wordle! This should give you a good idea of the mechanics of the game.

1 Implementation Requirements

At a high level, the version of the game you implement must have the following properties:

- The computer will load a dictionary of words from a file based on a configured word length.
- The computer will randomly pick a secret word and prompt the player for a guess.
- The player will try to guess the secret word and enter it in the command line.
- The computer will provide feedback on the player's guess and allow them to guess again.
- The player will have a limited number of guesses. If the player runs out of guesses, they lose the game.
- The computer will store all the user guesses, and the user can prompt the computer to show all previous guesses.
- There will be a GameConfiguration file that defines the following:
  - Length of the random word
  - Number of guesses available
  - Testing mode (meaning of this is discussed later)

Input/Output

This section defines the required inputs and outputs for the game. You must implement the game so that the string outputs are **exactly** as they're shown in this document. **All inputs are case sensitive.**

Starting the Program When the program first starts print a greeting message.

```
Hello!  Welcome to Wordle.
```

Starting a New Game Print a single line asking if the user wants to play a game.

```
Do you want to play a new game?  (y/n)
```

If the input is `y`, then start a new game. If the input is `n`, then end the program. If the input is neither, then simply reprint the prompt.

Running the Game For this example, we will assume the `GameConfiguration` is set to 5 letters and 6 guesses. User inputs are shown as bold to differentiate from console output.

The computer will randomly pick a word for the player to guess. If the testing mode is set to true, print out the word.

```
mount
```

Prompt the player for their guess.

```
Enter your guess:
```

The player will then enter their guess.

prone

The computer will first check if the word is the correct length. If it is not, then print out an error message.

```
This word has an incorrect length.  Please try again.
```

The computer will first check if this is a correct word, i.e., contained in the dictionary. If it is not, then print out an error message.

```
This word is not in the dictionary.  Please try again.
```

If the word passes the checks, it is a valid guess and the computer will then provide feedback. Print this feedback. Details of the feedback mechanism are in the next section.

```
__YG_
```

After printing the feedback, print the number of guesses the player has left. Only decrement the number of guesses when the guess is valid. Do not print the number of guesses left if the game is over.

```
You have 5 guess(es) remaining.
```

Keep prompting the player until they run out of guesses or guesses the word correctly.

If the player guesses correctly, print the appropriate feedback, along with congratulations.

```
GGGGG
Congratulations!  You have guessed the word correctly.
```

If the player runs out of guesses, print the game over message and the secret word.

```
You have run out of guesses.
The correct word was "mount".
```

At this point, the game has ended.

Feedback Mechanism The feedback mechanism should work like this:

- The computer assesses each letter of the player's guess, then provides feedback for each letter.
- For letters that are in the correct position, the feedback is green, denoted by G.
- For letters that are in the secret word but not in the correct position, the feedback is yellow, denoted by Y.
- For letters that are not in the word at all, the feedback is absent, denoted by _.
- If there are multiple instances of the same letter in the word:

  - Provide green feedback for the instances of the letter in the correct positions.

- After determining green feedback, if the word contains n remaining instances of the letter and if the guess contains $m > n$ remaining instances, then provide yellow feedback for the first n instances in the order in which they appear in the guess.
- Provide absent feedback for the remaining $m - n$ letters.
- Example (secret word: `mount`):
  * Guessing **goose** should return `_G___`
  * Guessing **attic** should return `_Y___`
  * Guessing **penne** should return `___G_`
- Example (secret word: `algae`):
  * Guessing **awake** should return `G_Y_G`
  * Guessing **every** should return `Y____`
  * Guessing **hence** should return `____G`

History Command The player may ask for all the guesses they have made previously. They will enter a special input that is the word `history` in all lowercase inside square brackets (brackets prevent it from being confused with a guessed word). This input is a special case which should not be checked for word length or presence in the dictionary.

[**history**]

Print the history of the player's guesses. See the sample runs at the very bottom to see examples.

Handling End Game Once a game has ended, print the new game prompt again.

```
Do you want to play a new game?  (y/n)
```

Repeat the new game process again. New games should pick a new random word and erase all previous guesses.

Program Structure

You must have a `Driver` class at the top level. This must contain a method `start()` which will run your program.

You must have a `Game` class that you construct inside the `Driver` class. This must take in a `GameConfiguration` and a `Scanner` as its parameters. The

`GameConfiguration` object defines the length of the word and number of guesses available to the player, as well as the testing mode. The `Scanner` object should be connected to the keyboard. This class must also contain the method `runGame()` which carries out the functions of the Wordle game itself.

You must connect **only one** `Scanner` object to the keyboard in your entire program. Do not create a new `Scanner` object connected to the keyboard when you start a new game, you must reuse the same one from the previous game. You may create other `Scanner` objects (for example, to parse a string or list of strings) if you want, but do not connect them to the keyboard.

You must create at least two classes in addition to the ones we provide.

Do not change function prototypes already given to you in the starter code.

Do not call System.exit() anywhere in your program.

Additional Requirements

Design Document

Prepare the following to show during the recitation before the project is due:

- Class document: Write down a paragraph describing which classes you will create to finish this implementation with a block diagram.
- Sequence diagram/flowchart: Give us 2 flowcharts that express the high level algorithm descriptions for a) the game runner, and b) formulating the reply to a given guess.

  - If you are familiar with UML sequence diagrams, you may use them in place of the flowcharts.

Testing

We have supplied you with a grading script and sample test cases for your final check before submission. For the extra credit part, we have supplied you only a JUnit test file.

Coding Style

Remember to have proper indentation, variable names, comments for methods, good classes and method partitioning, good choice of parameters and instance variables, etc.

General Requirements

No exceptions will be made for not following these requirements.

- Do not modify the files `GameConfiguration.java` and `Dictionary.java` as they will be ignored in our test cases. We will use our own versions of those files.
- Re-read the implementation requirements after you finished your program to ensure that you meet all of them (especially the Scanner requirement).
- Make sure that all your submitted files have the appropriate header file.
- Try out your code on our grading scripts on the ECE LRC machine
- Download your uploaded solution into a fresh directory and re-run all test cases.

Using Resources

You may not acquire, from any source (e.g., another student or an internet site), a partial or complete solution to this problem. You may not have another person (TA, current student, former student, tutor, friend, anyone) "walk you through" how to solve this assignment. Review the class policy on cheating from the syllabus.

Extra Credit

Implement Hard Mode!

The regular version of the game Wordle lets you to guess a word with a completely different set of letters from your previous guess, thus not using the feedback the game provides you. In the actual game, you can turn on Hard Mode to disable this, so that the game forces you to use the feedback from previous turns i.e., every new guess must contain the Y and G letters from all previous guesses, with the G letters being in the same position. If there are multiple matches for the same letter, new guesses should also contain multiple instances of that letter. e.g., if the secret word is "bunny" and the first guess is "nonet", the returned answer would be Y_G__. Then the next guess should have 2 n's, one of them being in the 3rd place. Invalid guesses are flagged as such:

```
This word does not use all of the feedback.  Please try again.
```

Implement this functionality in your program. The entry point for hard mode will be the `start_hardmode()` method. We will run tests for hard mode starting from this method. You do not have to implement this method if you do not wish to attempt hard mode, and you are not required to implement this method to obtain all points for the regular part of this assignment. You should ensure the regular functionality for Wordle is still run from the `start()` method.

YOUR ENTIRE CODE MUST COMPILE, with or without hard mode. If you simply do not implement hard mode, your code should still compile fine, but make sure that that if you do attempt hard mode that it does not interfere with the regular logic.

Suggestions

Recall that when designing a program in an object-oriented way, you should consider implementing a class for each type of entity you see in the problem. For example, in Wordle there is the secret word, guesses, feedback, a log of all guesses, a computer, a human player, and an over-all game runner. Some things are so simple you may choose not to implement a class for them. For example, the computer player doesn't do anything more than pick the secret code and provide feedback. Maybe that is simple enough for the `Game` class to do. Also, you may use some pre-existing type, primitive or a class, to represent things. For example, the guesses and feedback could be `String` objects.

One of the criteria of the assignment is to break the problem up into smaller classes even if you think the problem could be solved more easily with one big class. This is to help you practice breaking larger problems down to smaller pieces that are easier to solve than the overall problem.

After deciding what classes you need, implement them one at a time, simplest ones first. Test a class thoroughly before going on to the next class. You may need to alter your design as you implement and test classes.

Work on this assignment incrementally. Always have a working version of the program, then add more features to the working version. This way, you can have some functionality to the program even if you don't manage to complete all of it.

Deliverables

All deliverables must be submitted to the corresponding Canvas assignment **before the deadline**.

Design Deliverables

Canvas assignment: Project 2 Docs

- Two PDF files with your design for the program.

- Remember to show these during the recitation before the project is due.

Program

Canvas assignment: Project 2 Code

- Project2_EID.zip

  - Zip file structure:

```
Project2_EID.zip (zipped file)
    assignment2 (directory)
        Driver.java
        Game.java
        OtherClassFiles.java
```

  - Do not include your test files.

  - Do not include `GameConfiguration.java` and `Dictionary.java`, we will use the unmodified ones that we already have.

FAQ

- Only call `Dictionary`'s `getRandomWord()` method ONCE per the START of the game ("y" to the prompt of new game). Otherwise it will fail the grading script.
- Am I allowed to use an external file to store guesses?

  - No, that is bad programming style.
- Can we add new variables or methods to `GamingConfiguration` for our own purposes?

  - You can only do so for testing. Your final version has to work with our `GamingConfiguration.java`. There could be a case where you base your program off your own modified version which still works with our version, but that is very unlikely.
- Are we creating the `main()`?

  - Yes, but for testing only. It exists in `Driver.java`, but we will not call it in our test cases. All your functionality should be inside `start()`.
- What Linux command could I use to zip up my assignment2 directory? How about unzipping to check?

  - `zip -r Project2_xyz123.zip assignment2`

- Copy the zip file to a different directory, then: `unzip Project2_xyz123.zip`

- Is it alright if we use a static `Scanner` in our driver class?

  - No, it might mess up our grading script initialization. You need to check the sample grading script.

- Where exactly do we have blank lines?

  - All lines that are only whitespace will be ignored

  - All print statements should use println (in a prompt, message, etc.)

- Is it ok to print the welcome message every time you start a new game? Or, should it only print once when the program starts?

  - Only print once when the program starts.

- For the [history] command, if the player inputs [history] as their first guess, would there just be a blank list for each feedback?

  - Yes.

- Corner cases in the `GameConfiguration.java` fields like setting `wordLength = 0` (all guesses are invalid) or `numGuesses = 0` (you lose immediately) are not defined in the PDF. Should we be handling such errors or can we expect you to stick with usable values?

  - You can assume that `wordLength` will always be greater than zero, and `numGuesses` will be greater than zero.

- I cannot remove old files for new runs on kamek.

  - You might have to create a new directory as a workaround.

- When I resubmitted another zip file to canvas Project2 Code. But for my new submission, the file name changed to Project2_myEID-1.zip on canvas. Will this cause problems on the grading?

  - No, this is fine.

- From reading the PDF I wasn't sure if the history should include the total move history for the session or just one game.

  - The [history] command should output all the guesses for the current game only.

- How do I use the test_runs.zip? And is this in addition to the grading script that you have provided?

- The test_runs.zip has several input files and expected outputs. Use the `diff` command on Linux (or its equivalent) to compare your output with the provided output. And yes, it is in addition to the grading script. You MUST run the grading script. The test_runs tests are optional, but highly recommended.

- Can you suggest some test cases?

  - One test case that many of you mess up is when you correctly guess the word on the very last guess.

Here is a sample run:

```
Hello! Welcome to Wordle.
Do you want to play a new game? (y/n)
y
Enter your guess:
goose
_G___
You have 5 guess(es) remaining.
Enter your guess:
attic
_Y___
You have 4 guess(es) remaining.
Enter your guess:
penne
___G_
You have 3 guess(es) remaining.
Enter your guess:
[history]
goose->_G___
attic->_Y___
penne->___G_
--------
Enter your guess:
supercalifragilisticexpialidocious
This word has an incorrect length. Please try again.
Enter your guess:
abcde
This word is not in the dictionary. Please try again.
Enter your guess:
mount
GGGGG
Congratulations! You have guessed the word correctly.
Do you want to play a new game? (y/n)
y
Enter your guess:
[history]
--------
Enter your guess:
goose
___YY
You have 5 guess(es) remaining.
Enter your guess:
attic
Y____
```

```
You have 4 guess(es) remaining.
Enter your guess:
penne
_Y___
You have 3 guess(es) remaining.
Enter your guess:
mount
_____
You have 2 guess(es) remaining.
Enter your guess:
crate
_YY_Y
You have 1 guess(es) remaining.
shoot
Y____
You have run out of guesses.
The correct word was "laser".
Do you want to play a new game? (y/n)
n
```

Adapted from the online game Wordle.