Title : Wordle
Modules : 03 Simple Control Flow & 04 Subroutines
Point Value : 50 (assignment)
Deadline : Due before 11:00 pm on Thursday October 6

Topics Covered

Conditional statements and repetition statements with simple and complex logic, writing and calling subroutines, Java's Random class, and more practice with Strings. *You must not use any type of Java arrays, ArrayLists, Vectors, Lists, etc. to solve this problem, because we haven't covered these yet in this course. You must not add any import statements beyond what is already there.*

You might want to review Zybook section 7.4 on using the Java Random class to generate pseudorandom integer data. Also visit the Java API for the Random class. In particular, read the detailed documentation for using the method `nextInt()`.

You might also want to review Zybook section 4.4 on using indexOf to search a string for the first occurrence of a specific character.

Project Description

This project has you implement a variant of the word game Wordle. You can find the original game that our game is based on here, but keep in mind our version has a few different rules. The complete rule list for our version is as follows:

1. At the beginning of a game a new word is randomly selected. The word's length is randomly chosen to be either 4, 5, or 6 letters.
2. The player has 6 chances to guess the selected word. We do **not** require that the guess is a valid word in our version.
3. After each guess the game indicates which guessed letters are in the word and in the correct position, which guessed letters are in the word but not in the correct position, and which guessed letters aren't in the word at all.
    a. A guessed letter that is not found anywhere in the word is indicated with a question mark: ?
    b. A guessed letter that is in the word but not in the correct position is indicated with the guessed letter in lower-case.
    c. A guessed letter that is in the word and is in the correct position is indicated with the guessed letter in upper-case.

4. A player can request a hint which will reveal one randomly selected letter in the word. A player gets no more than 2 hints per game.
5. **Win condition:** the player wins by correctly guessing the word in 6 or less guesses. Otherwise they lose.

Guess Feedback Examples:

```
PARTED – Correct word
POTATO – Player’s guess
P?tat? - Output

PARTED – Correct word
PAIRED – Player’s guess
PA?rED - Output

POTATO – Correct word
AOEUIY – Player’s guess
aO???? - Output
```

This project is an exercise in working with strings, arithmetic operations, random number generation and writing subroutines (i.e., methods). We have provided starter code that you need to download before starting to work on this project. The starter code includes several files that you need to keep in the same directory. You will only be working in the `Wordle.java` file. This will be a menu-driven game and your job is to complete the parts shown by “`TODO`” comments as follows:

newWord function:

This function will generate and return a new word using the `WordProvider.getWord` method we’ve provided for you. You will need to generate a random number between 4 and 6 (inclusive) in this this function for the length of the word. This function should also print the message “`New word length: _`” where _ is the length of the word.

giveHint function:

This function displays a hint to the player of the format “`HINT! The word contains the letter: _.`” where the _ is a randomly selected letter from the correct word.

Keep in mind:

- The letter must be selected at random, uniformly distributed over all characters in the word. This means in a word like “freeze”, there is a 50% chance that an “e” will be revealed because 3 of the 6 letters are “e”.
- Because of the random nature of the hints, it is easily possible that the player will get the same hint twice in a row. For this exercise this is acceptable (and required for consistent grading).

validateGuess function:

This function checks that the player’s guess is of the correct length and only contains valid characters: 'A' through 'Z' and 'a' through 'z'. It returns true if both conditions are met, otherwise false. It also displays a message indicating the problem with an invalid guess. The output should be exactly as follows (according to the error):

1. `“You must enter a guess of length _”,` where _ is the length of the word that the player is trying to guess.
2. `“Your guess must only contain upper case letters and lower case letters”`

checkGuess function:

This function takes two inputs: a string representing the correct word and a string representing the player’s guess. It returns true if the words match **ignoring capitalization,** otherwise it returns false.

It also displays feedback in the form of a string that indicates correctly placed letters, incorrectly placed letters that are contained somewhere in the word, and letters that are not contained anywhere in the word. **See the sections above for details on this and some examples.**

A few hints:

- It may be helpful, but is not required, to use the `indexOf` string method.
- Carefully consider how your implementation will handle cases where the correct word or the guessed word (or both!) contain multiple occurrences of the same letter.

main function:

Our main function will serve two purposes: (1) manage the menu and (2) handle most of the game logic (our implementation of the rules of the game).

To start with you'll want to define and initialize the game state. Generate and store a new random word by calling your `newWord` method. Think about and declare other variables as needed to manage the problem.

Managing the menu requires outputting text to the screen (HINT: use the `printMenu` method already provided to you!) Then, we display the message `"Please enter a choice: "` and capture the user's input. The program must continue to execute until the user chooses to exit by entering `"e"` or `"E"`.

Below is the description of each menu choice:

`"n"` or `"N"`: Generate a new word by using your `newWord` method.

`"h"` or `"H"`: Display a hint to the player if they still have hints remaining. Additionally,

1. If the player has hints remaining, provide one and then display `"You have _ hint(s) remaining."` where _ is the number of hints remaining. **IMPORTANT**: use correct plurality ("1 hint remaining" vs. "0 hints remaining")
2. Otherwise, display `"Sorry, you're out of hints!"`

`"g"` or `"G"`: This is how the player will be prompted to enter a guess.

1. If the player is out of guesses, print `"Sorry, you're out of guesses! Use the "n"/"N" command to play again."`
2. Otherwise,
   a. Prompt the user to enter a guess with the message `"Enter your guess:"` including a newline at the end of the output.
   b. If the guess is valid:
      i. If the guess is correct, display `"Congrats! You won!"`
      ii. Otherwise, decrement the number of guesses remaining. Then, if the number of guesses remaining is > 0 display `"You have _ guess(es) remaining."` where _ is the number of guesses remaining. **IMPORTANT:** use correct plurality ("2 guesses remaining" vs. "1 guess remaining"). Otherwise display `"Sorry, you're out of guesses! The word was ___. Use the "n"/"N" command to play again."` where ___ is the correct word.

`"e"` or `"E"`: If this one is selected, the program simply terminates without doing or displaying any further messages!

Note 1: At any point during the execution if a choice other than the ones described above is entered, the program must show the message `"Invalid option! Try again!",` print the menu options and let the user make a choice again.

Note 2: See a sample run here. Once you finish the implementation, make sure you can reproduce the sample run.

Development Plan

It's time to think about how you will implement this task in Java. We have provided starter code for you which describes all of the required methods in detail. You must begin the project by downloading the zip file, extracting the files, and modifying **only** the Wordle.java file to complete the assignment. You can write additional helper methods in the Wordle.java file if you feel it would be beneficial.

You may NOT modify the given method signatures (name, parameters, and return type) and you must implement the functionality exactly as described in this writeup and the given Javadoc comment for each method. When grading your assignment, we'll test these methods directly in addition to running your main method. You should test them individually yourself to ensure they behave as expected. Also be sure to remove the "// TODO: " comments in each method (and elsewhere) as you work.

You'll also notice in the starter code that we have created a (global) static Random variable `gen` outside of any methods at the start of the class definition, and initialized it with the seed value 0. You must use this one object to generate random values (not the Math.random() method), and must not change its declaration in any way. This will give you consistent results from one run to the next (as long as your input is the same) and will allow us to automate some of the grading of your program.

General Project (Hard) Requirements:

- **You will earn a 0 for any code that does not compile!**
- Your solution must be named **`Wordle.java`** (capitalization matters!)
- Make sure your output matches our specifications above exactly! This is important to pass the autograder tests.
- Your file must have a javadoc style class comment that explains briefly the purpose of the program, the author's name, JHED, and the date.
- Each subroutine must have a complete javadoc style comment that explains briefly the purpose of the method, the purpose of each parameter, and what is returned (if anything).

- The programs must be fully checkstyle-compliant using our course required `check112.xml` configuration file. This means that for each file, the checkstyle audit completes without reporting any errors or warnings.
- You must also use good style, including with respect to class/method/variable names, etc.
- You must not use any type of Java arrays, vectors, etc. to solve this problem.
- Submit **only the `Wordle.java` file** to the Project 2 assignment on Gradescope before the deadline date.
- You must submit the code before the deadline. You can resubmit as many times as you want before the deadline. We will only grade your final (most recent) submission.

Grading Breakdown:

The following points are awarded only if your submitted files compile since **assignments that don't compile earn a 0 grade!**

- [16] Main method functionality
- [24] Required helper method implementations
- [4] Appropriate use of required methods
- [4] General style
- [2] Submission & runtime errors