CS 1331 Homework 1 - Word Guess

Introduction

In this programming project you will practice:

- simple console input and output,
- using primitive values and variables,
- using objects of two classes from the Java standard library, and
- control structures.

Problem Description

You're a bored Georgia Tech student without enough work to fill your days and you like word games, so you decide to make a word guessing game to pass the time.

Solution Description

Download WordGuess.java (/spring2018/hw1/WordGuess.java) and complete the definition of the `WordGuess` class so that it implements a simple word guessing game. WordGuess.java (/spring2018/hw1/WordGuess.java) includes a skeleton `main` method so that you can run the class as a console program. Fill in the rest of the main method – under the `// Your code here:` line – with code that implements a console-based word guessing game.

Do not modify the code we give you.

Rules of Word Guess

- If the player makes five incorrect guesses ("misses"), the player loses.
- Player can only guess one letter at time.
- If a player guesses a letter that occurs in the secret word multiple times, each occurrence of the letter is revealed.

Game Play

- As long as the player has made fewer than five misses and has not guessed all the letters in the word:
  - Print a line reporting the current missed letters and the number of remaining misses before the player loses.
  - Print a line reporting the current guess with underscores, `'_'` , for letters which haven't yet been guessed.
  - Print a prompt for the user to enter a letter on the same line as the prompt.
  - Read a letter from the console.
  - Update the current guess and the number of misses accordingly.
- Print a line reporting all the letters the player missed in the game.
- Print a line reporting the final guess, including `'_'` characters for letters that weren't guessed.
- If the player lost, print a line saying "Sorry, too many missess." and what the secret word was.
- If the player won, print a line saying "Congratulations! You got it!" (the full secret word will have already been printed).

Sample Output

Successful run with a couple of misses and a repeated letter:

```
$ java WordGuess
Missed letters (5 left):
Current guess: ___
Guess a letter: c

Missed letters (4 left): c
Current guess: ___
Guess a letter: a

Missed letters (4 left): c
Current guess: _a_
Guess a letter: t

Missed letters (3 left): ct
Current guess: _a_
Guess a letter: d

Missed letters: ct
Final guess: dad
Congratulations! You got it!
```

Unsuccessful run with a couple of hits:

```
$ java WordGuess
Missed letters (5 left):
Current guess: ___
Guess a letter: q

Missed letters (4 left): q
Current guess: ___
Guess a letter: w

Missed letters (3 left): qw
Current guess: ___
Guess a letter: e

Missed letters (2 left): qwe
Current guess: ___
Guess a letter: r

Missed letters (2 left): qwe
Current guess: r__
Guess a letter: t

Missed letters (2 left): qwe
Current guess: r_t
Guess a letter: y

Missed letters (1 left): qwey
Current guess: r_t
Guess a letter: u

Missed letters: qweyu
Final guess: r_t
Sorry, too many misses. The secret word was rat
```

Solution Constraints

- **IMPORTANT**: Your output must match the examples above **exactly**.
- You must use a Scanner (https://docs.oracle.com/javase/9/docs/api/java/util/Scanner.html) reading from `System.in` to get the player's input.
- You may not import any classes other than the ones already imported in the provided code. Yes, certain classes from, e.g., `java.util` make sense for this task, but we want you to practice using primitive features of Java.

Tips and Considerations

- You get the length of a String (https://docs.oracle.com/javase/9/docs/api/java/lang/String.html) with the `length` method, e.g., `"cat".length()` returns `5`.
- You may want to use an instance of the StringBuilder (https://docs.oracle.com/javase/9/docs/api/java/lang/StringBuilder.html) class to hold the player's guess.
  - After you create a new `StringBuilder`, e.g., `StringBuilder sb = new StringBuilder()`, it's empty.
  - Use the `append` method to add a character to the String, e.g. `sb.append('_')`.
    - How would you append a particular number of `'-'` characters?
  - Use the `toString` method to get a printable `String` with the current contents of the `StringBuilder`.
  - Use the `setCharAt` method to set the `char` at a particular index of a `StringBuilder` object, e.g., if you have a `StringBuilder sb` whose content is `"___"`, then after `sb.setCharAt(1, 'a')` its content is `"_a_"`.
  - You can play around with `StringBuilder` in JShell (https://docs.oracle.com/javase/9/jshell/introduction-jshell.htm) to get a feel for it.
- There is no `Scanner` method that returns a `char`. You can use the `next` method, which returns a `String` and use `String`s `charAt` method to get a `char` value. For example `"a".charAt(0)` returns `'a'`.
- The provided code includes a means to pre-select a secret word instead of randomly selecting a secret word. For example, if you run `java WordGuess 0` the secret word will be `"cat"`, if you run `java WordGuess 1` the secret word will be `"dad"`. You can use this feature to test your code. We will use this feature to auto-grade your code.

Grading

- [5] Correctly prints missed letters at beginning of game.
- [5] Correctly prints current guess at beginning of game.
- [5] Correct prompt for player input.
- [5] Reads player input on same line as prompt.
- [5] Uses first character of player's input and ignores the rest, if supplied.
- [5] Correctly prints missed letters after a miss.
- [5] Correctly prints number of misses remaining after a miss.
- [5] Correctly prints current guess after a miss.
- [5] Correctly prints missed letters after a hit.
- [5] Correctly prints number of misses remaining after a hit.
- [5] Correctly prints current guess after a hit that occurs once in the secret word.
- [5] Correctly prints current guess after a hit that occurs more than once in the secret word.
- [5] Terminates after five misses.
- [5] Terminates after correctly guessing secret word.
- [5] Correctly prints total misses at end of game.
- [5] Correctly prints final guess at end of game.

- [5] Prints correct message on loss.
- [5] Prints correct message on win.
- [10] No extraneous blanks, lines, etc.

Javadocs

Starting from this homework, you will need to write Javadoc comments and watch for checkstyle errors with your submission.

- Every class should have a class level Javadoc that includes `@author <GT Username>`.
- Every public method should have a Javadoc explaining what the method does and includes any of the following tags if applicable:
    - `@param <parameter name> <brief description of parameter>`
    - `@returns <brief description of what is returned>`
    - `@throws <Exception> <brief explanation of when the given exception is thrown>`

See the CS 1331 Style Guide (http://cs1331.gatech.edu/cs1331-style-guide.html) for details.

Checkstyle

For each of your homwork assignments we will run checkstyle and deduct one point for every checkstyule error.

For this homework the **checkstyle cap is 10**, meaning you can lose up to 10 points on this assignment due to style errors. This limit will increase with each homework.

- If you encounter trouble running checkstyle, check Piazza for a solution and/or ask a TA as soon as you can!
- You can run checkstyle on your code by using the jar file found on the course website that includes xml configuration file specifying our checks. To check the style of your coed run `java -jar checkstyle-6.2.2.jar *.java`.
- To check your Javadocs run `java -jar checkstyle-6.2.2.jar -j *.java`.
- Note that the command for checking code and the command for checking Javadocs are different. You will have to run both commands to fully test for style errors.
- Javadoc errors are the same as checkstyle errors, as in each one is worth a single point and they are counted towards the checkstyle cap.
- **You will be responsible for running checkstyle on *ALL* of your code.**
- Depending on your editor, you might be able to change some settings to make it easier to write style-compliant code. See the customization tips (http://cs1331.gatech.edu/customization-tips.html) page for more information.

Collaboration

When completing homeworks for CS1331 you may talk with other students about:

_ What general strategies or algorithms you used to solve problems in the homeworks _ Parts of the homework specification you are unsure of and need more explanation _ Online resources that helped you find a solution _ Key course concepts and Java language features used in your solution _ **You may not discuss, show, or share by other means the specifics of your code, including screenshots, file sharing, or showing someone else the code on your computer, or use code shared by others.**

Examples of approved/disapproved collaboration:

OKAY: “Hey, I’m really confused on how we are supposed to implement this part of the homework. What strategies/resources did you use to solve it?”

BY NO MEANS OKAY: “Hey… the homework is due in like 20 minutes… Can I see your code? I *promise* won’t copy it directly!”

In addition to the above rules, note that it is not allowed to upload your code to any sort of public repository. This could be considered an Honor Code violation, even if it is after the homework is due.

Submission

- Submit your `WordGuess.java` file as an attachment to the `hw1` assignment on Canvas. You can submit as many times as you want, so feel free to submit as you make substantial progress on the homework. We only grade your **last** submission, meaning we will ignore any previous submissions.
- As always, late submissions will not be accepted and non-compiling code will be given a score of 0. For this reason, we recommend submitting early and then confirming that you submitted ALL of the necessary files by re-downloading your file(s) and compiling/running them.

Good Luck! \(°□°)/