Take-home Assessment 5: Guessing Game *due November 4, 2020, 11:59pm*

This assignment will assess your mastery of the following objectives:

- Write a functionally correct Java program to produce specified console output.
- Write `while` loops to repeat code an indefinite number of times.
- Use `Random` to generate (pseudo)random numbers.
- Write and call methods that accept parameters and return values to manage information flow and add structure to programs.
- Follow prescribed conventions for spacing, indentation, naming methods, and header comments.

Sample Output #1

```
<< Your custom haiku goes here >>

I'm thinking of a number between 1 and 100...
Your guess? 50
It's higher.
Your guess? 75
It's lower.
Your guess? 60
It's higher.
Your guess? 65
It's lower.
Your guess? 62
You got it right in 5 guesses!
Do you want to play again? y

I'm thinking of a number between 1 and 100...
Your guess? 28
You got it right in 1 guess!
Do you want to play again? YES

I'm thinking of a number between 1 and 100...
Your guess? 50
It's higher.
Your guess? 75
It's lower.
Your guess? 60
It's lower.
Your guess? 55
It's higher.
Your guess? 58
You got it right in 5 guesses!
Do you want to play again? no

Overall results:
Total games   = 3
Total guesses = 11
Guesses/game  = 3.7
Best game     = 1
```

Program Behavior

This program allows the user to play a game in which the program thinks of a random integer and accepts guesses from the user until the user guesses the number correctly. After each incorrect guess, the program will tell the user whether the correct answer is higher or lower. As in assessment 4, this program's behavior is dependent on input from a user (user input is underlined in samples), but this program also includes random values. The format and structure of your output should exactly match the given logs, but the specific output may vary based on randomness and/or user input. You can also use random seeds (see below) to control the randomness and test your output using the Mark button in Ed.

Make sure that the format and structure of your output ***exactly*** match the given logs.

The program should begin with an introduction in the form of a haiku. A haiku is a poem consisting of three lines in a specific structure: the first line has five syllables, the second line has seven syllables, and the third line has five syllables. The content of the haiku can be any non-offensive text you like, but should somehow relate to the game.

After the introduction is printed, the game begins. In each game, the computer should choose a secret random integer between 1 and a maximum, inclusive. (The default maximum is 100.) The game should ask the user to guess a number until the chosen secret number is guessed. After each incorrect guess, the program should indicate whether the secret number is higher or lower than the guess. Once the user enters the correct number, the game ends and the program should report how many guesses were needed. This output should be grammatically correct– that is, it should print "1 guess" if the user guessed correctly on the first try, but use the word "guesses" if more than one try was required. The second game in Sample Output #1 shows what this should look like.

The guess that was correct counts in the number of guesses.

After each game ends and the number of guesses is shown, the program should ask the user if they would like to play again. If the user's response starts with a lower- or upper-case *Y* (e.g. y, Y, yes, YES, Yup, yessir, yeehaw), a new game should begin. If the user gives any other response (e.g. no, No, okay, 0, certainly, hello), no new game should begin.

A new secret number should be chosen for each game.

Once the user chooses not to play again, the program should print overall statistics about all games played. Specifically, the total number of games, total guesses made in all games, average number of guesses per game (rounded to the nearest tenth), and best game (i.e. fewest guesses needed to win any one game) should be displayed. Your statistics must be correct for any number of games or guesses ≥ 1. **You may assume that no game will require one million or more guesses.**

You should *not* assume that the user plays the game optimally, or that they use any particular strategy. For example, a user may enter the same guess more than once.

Sample Output #2

```
<< Your custom haiku goes here >>

I'm thinking of a number between 1 and 100...
Your guess? 50
It's lower.
Your guess? 25
It's higher.
Your guess? 35
It's lower.
Your guess? 30
It's higher.
Your guess? 32
It's lower.
Your guess? 31
You got it right in 6 guesses!
Do you want to play again? yAy

I'm thinking of a number between 1 and 100...
Your guess? 60
It's lower.
Your guess? 20
It's higher.
Your guess? 30
It's higher.
Your guess? 40
It's higher.
Your guess? 50
It's lower.
Your guess? 47
It's higher.
Your guess? 49
You got it right in 7 guesses!
Do you want to play again? nah

Overall results:
Total games   = 2
Total guesses = 13
Guesses/game  = 6.5
Best game     = 6
```

Development Strategy

As on previous assessments, you will likely want to approach the program one part at a time, rather than trying to write most or all of the program at once. We recommend the following approach:

(1) **Single game:** Write and *thoroughly* test a method to play a single game with numbers between 1 and 100.

(2) **Constant:** Modify your code to use a constant (see below) to be able to change the maximum possible secret number.

(3) **Multiple games:** Add code to ask the user whether or not they want to play again after a game finishes and start a new game if their response begins with a y (see above).

(4) **Statistics:** Add code to track and output the overall statistics across multiple games.

Debugging Tips

While testing your program, you may want to add code to print the secret number before the game begins. This will allow you to more easily test that the higher/lower messages and statistics are correct. You may also want to add additional output to display the state of certain values (e.g. number of guesses, current statistics) at relevant points in the program.

Remember to remove any debugging output before submitting.

If you wish to control the randomness to more reliably test your program and exactly match our sample logs, you can seed your `Random` object by passing a parameter to the constructor, similar to the following:

```
Random rand = new Random(42);
```

Each sample log on the website indicates the seed that was used to produce that log. For more information on seeding, see the Assessment 5 FAQ post on Ed.

Hints

The following suggestions and hints may help you be more successful on this assessment:

- You will likely want to use `while` loops to control the repetition in your program. Remember that `while` loops are best for indefinite loops when you do not know how many times the loop will execute.
- You *MUST NOT* define a method that calls itself, or a pair of methods that call each other, to achieve repetition. This approach (called recursion) is difficult to get right and is not allowed in CSE 142.
- You may find it useful to review fencepost loops and sentinel loops in section 5.2 of the textbook.
- If your program is generating `InputMismatchException` errors, you are likely reading the wrong type of values from your `Scanner` (for example, using `nextInt` to read text).
- As on assessment 4, you will need to round certain parts of your output. You should round only for output purposes, not during any calculations. You should use a rounding method such as the one shown in lecture and used on assessment 4. DO NOT round using `System.out.printf`.
- Output that needs to be aligned (such as the equals signs in the statistics) should be aligned using spaces, NOT tabs (`\t`). Tabs can have different widths on different computers and might result in incorrect output on our grading system.

Sample Single Game

```
I'm thinking of a number between 1 and 100...
Your guess? 50
It's lower.
Your guess? 25
It's higher.
Your guess? 48
It's lower.
Your guess? 46
You got it right in 4 guesses!
```

Implementation Guidelines

Required Methods

To recieve full credit, your program is required to include the following methods:

(1) A method to play a **single game**

(2) A method to report **all overall statistics**

Each of these methods must perform *only* the tasks indicated. Specifically, your single game method (1) must not play multiple games or ask the user to play again; and your statistics method (2) must not play any games or take any user input. (See the sample output above for what playing a single game might look like.) You may (and likely will) have additional methods beyond these two.

User Input

This program requires you to process user input, which you must do using a `Scanner`. All text input should be read using the `next` method in the `Scanner` class, not the `nextLine` method.

You may assume the user always enters valid input. Specifically, you may assume that:

- the user will always enter a value of the correct type
- the user will always enter a single word when asked to play again
- all guesses will be integers in the expected range (i.e. at least 1 and no more than the current maximum)

You should NOT make any assumptions about input not specified here.

Class Constant

The minimum possible secret number for a game will always be 1, but the maximum value should be declared as a constant. The default maximum will be 100, but this value should be able to be easily changed. Sample Output #3 below shows a sample execution with the a maximum secret number of

5. See the course website and the for additional example logs with alternate values for the constant. The Mark button in Ed will test your program with different constant values. *To ensure our testing and grading scripts work correctly, you **must** name your constant `MAX_VALUE`.* In addition, please set the value of the constant to 100 before submitting your work.

Permitted Java Features

For this assessment, you are restricted to Java concepts covered in chapters 1 through 5 of the textbook. In particular, you MUST use the `Scanner` class to accept user input and the `Random` class to generate random numbers.

Sample Output #3, constant = 5

```
<< Your custom haiku goes here >>

I'm thinking of a number between 1 and 5...
Your guess? 2
It's higher.
Your guess? 4
It's lower.
Your guess? 3
You got it right in 3 guesses!
Do you want to play again? yuppers

I'm thinking of a number between 1 and 5...
Your guess? 3
It's higher.
Your guess? 5
You got it right in 2 guesses!
Do you want to play again? Nah

Overall results:
Total games   = 2
Total guesses = 5
Guesses/game  = 2.5
Best game     = 2
```

Code Quality Guidelines

In addition to producing the desired behavior, your code should be well-written and meet all expectations described in the grading guidelines and the Code Quality Guide. For this assessment, pay particular attention to the following elements:

Capturing Structure

Your `main` method in this program may have more code than it has in previous assessments. In particular, you may include a limited amount of output and some control flow constructs (e.g. a loop to play multiple games) in `main`. However, your `main` method must remain a concise summary of your program's structure, and you must still utilize methods to both capture structure and eliminate redundancy.

Using Parameters and Returns

Your program should utilize parameters and return values effectively to produce a well-structured program as described above. Your methods should not accept unnecessary or redundant parameters. In particular, your program should include only a single `Scanner` and a single `Random` which are passed as parameters to all required methods. You should NOT declare either of these objects as a constant.

Code Aesthetics

Your code should be properly indented, make good use of blank lines and other whitespace, and include no lines longer than 100 characters. Your class, methods, variables, and constant should all have meaningful and descriptive names and follow the standard Java naming conventions. (e.g. `ClassName`, `methodOrVariableName`, `CONSTANT_NAME`) See the Code Quality Guide for more information.

Commenting

Your code should include a header comment at the start of your program, following the same format described in previous assessments. Your code should also include a comment at the beginning of each method that describes that methods behavior. Method comments should also explicitly name and describe all parameters to that method and describe the method's return value (if it has one). Comments should be written in your own words (i.e. not copied and pasted from this spec) and should not include implementation details (such as describing loops or expressions included in the code). See the Code Quality Guide for examples and more information.

Running and Submitting

You can run your Guessing Game program by clicking the "Run" button in Ed. This will compile and execute your code and show you any errors, or the output of your program if it runs correctly. If you believe your output is correct, you can submit your work by clicking the "Mark" button in the Ed assessment. You will see the results of some automated tests along with tentative grades. **These grades are not final until you have received feedback from your TA.**

You may submit your work as often as you like until the deadline; we will always grade your most recent submission. Note the due date and time carefully—**work submitted after the due time will not be accepted**.

Getting Help

If you find you are struggling with this assessment, make use of all the course resources that are available to you, such as:

- Reviewing relevant examples from lessons, section, and lab
- Reading the textbook
- Visiting office hours
- Posting a question on the message board

Collaboration Policy

Remember that, while you are encouraged to use all resources at your disposal, including your classmates, **all work you submit must be entirely your own**. In particular, you should **NEVER** look at a solution to this assessment from another source (a classmate, a former student, an online repository, etc.). Please review the full policy in the syllabus for more details and ask the course staff if you are unclear on whether or not a resource is OK to use.

Reflection

In addition to your code, you must submit answers to short reflection questions. These questions will help you think about what you learned, what you struggled with, and how you can improve next time. The questions are given in the file `GuessingGameReflection.txt` in the Ed assessment; type your responses directly into that file.