CSE 142, Autumn 2018
Programming Assignment #5: Guessing Game (20 points)
Due: Tuesday, October 30, 2018, 9:00 PM

This assignment focuses on `while` loops and random numbers. Turn in a file named `GuessingGame.java`.

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, you will tell the user whether the correct answer is higher or lower. Your program must exactly reproduce the format and behavior of the logs in this document and on the course web site.

The log below shows one sample execution of your program. Your output will differ depending on the random numbers chosen and user input typed, but the overall output structure should match that shown below.

```
<< your haiku intro message 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? y

I'm thinking of a number between 1 and 100...
Your guess? 50
It's higher.
Your guess? 75
It's lower.
Your guess? 65
It's lower.
Your guess? 64
You got it right in 4 guesses!
Do you want to play again? YES

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? no

Overall results:
Total games   = 3
Total guesses = 17
Guesses/game  = 5.7
Best game     = 4
```

First, the program prints an introduction in the form of a haiku. A haiku has 3 lines: one with 5 syllables, the second with 7 syllables, and the third with 5 syllables.

Next, a series of guessing games is played. In each game, the computer chooses a random number between 1 and 100, inclusive. The game asks the user for guesses until the correct number is guessed. After each incorrect guess, the program gives a clue about whether the correct number is higher or lower than the guess. Once the user types the correct number, the game ends and the program reports how many guesses were needed.

After each game ends and the number of guesses is shown, the program asks the user if they would like to play again. You may assume that the user will type a one-word string as the response to this question.

A new game should begin if the user's response starts with a lower- or upper-case Y. For example, answers such as "y", "Y", "yes", "YES", "Yes", or "yeehaw" all indicate that the user wants to play again. Any other response means that the user does not want to play again. For example, responses of "no", "No", "okay", "0", "certainly", and "hello" are all assumed to mean no.

Once the user chooses not to play again, the program prints overall statistics about all games played. The total number of games, total guesses made in all games, average number of guesses per game (as a real number rounded to the nearest tenth), and best game (fewest guesses needed to solve any one game) are displayed.

Your statistics should be correct for any number of games or guesses ≥ 1. You may assume that no game will require one million or more guesses.

```
I'm thinking of a number between 1 and 100...
Your guess? 71
You got it right in 1 guess!
```

You should correctly handle the case where the user guesses the correct number on the first try, as shown in the box to the left.

You may assume all user input is valid. When prompted for numbers, the user will type integers only, and they will be in the expected ranges.

Implementation Guidelines:

```
<< your haiku intro message 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? yes

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
```

You should define a **class constant** for the maximum number used in the games. The log on the previous page shows games from 1 to 100, but you should be able to change the constant value to use other maximum values.

Use your constant throughout your code and do not refer to the number 100 directly. Test your program by changing your constant and running it again to make sure that everything uses the new value. A guessing game for numbers from 1 to 5 would produce output such as that shown at left. The web site shows other expected output cases.

Produce randomness using a single `Random` object, as seen in Chapter 5. Remember to `import java.util.*;`

Display rounded numbers using the `System.out.printf` command or a rounding method of your own.

Read user yes/no answers using the `Scanner`'s `next` method (not `nextLine`, which can cause strange bugs when mixed with `nextInt`). To test whether the user's response represents yes or no, use `String` methods seen in Chapters 3-4 of the book. If you get an `InputMismatchException`, you are trying to read the wrong type of value from a `Scanner`.

Produce repetition using `while` or `do/while` loops. You may also want to review fencepost loops from Chapter 4 and sentinel loops from Chapter 5. Chapter 5's case study is a relevant example. Some students try to avoid properly using `while` loops by writing a method that calls itself, or a pair of methods A and B where A calls B and B calls A, creating a cycle of calls. Such solutions are not appropriate for this assignment (or others in this course) and may result in a deduction. For help solving the "best game" part of the program, you may want to read textbook section 4.2 on min/max loops.

```
I'm thinking of a number between 1 and 100...
*** HINT: The answer is 46
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!
```

(suggested initial simple version of program)

Consider first writing a simpler version that plays a single guessing game. Ignore other features such as multiple games and displaying overall statistics.

While debugging it is useful to print a temporary "hint" message like that shown at left. This way you will know the correct answer and can test whether the program gives proper clues for each guess. This is also helpful for testing the "1 guess" case.

Style Guidelines:

For this assignment you are limited to the language features in Chapters 1-5 shown in lecture and the textbook.

Structure your solution using static methods that accept parameters and return values where appropriate. For full credit, you must have at least the following two methods other than `main` in your program:

1. a method to **play one game** with the user
   This method should *not* contain code to ask the user to play again. Nor should it play multiple games in one call.
2. a method to **report the overall statistics** to the user
   This method should print the statistics *only*, not do anything else such as `while` loops or playing games.

You may define more methods if you like, although the fact that methods can return only one value will limit how you can decompose the problem. **For this assignment, it is acceptable for some `println` statements or other coding constructs to be in `main`, as long as you use good structure and `main` is a concise summary of the program.** For example, you can place the loop for multiple games and the prompt to play again in `main`.

Give meaningful names to methods, variables, and parameters. Follow Java's naming standards. Use proper indentation and whitespace. Limit lines to 100 characters. Localize variables. Include header comments at the start of your program and of each method. Since this program has longer methods, you should also put comments inside methods near complex sections of code.