AP CS A – 2019-20, BHS

Programming Assignment #4: Guessing Game

This assignment focuses on `while` loops, if-statements, and random numbers. This is material covered in Chapters 4 and section 5.1.

In Grade-it, turn in a file named `GuessingGame.java`.
Use this link to turn in your work: https://gradeit-hs.cs.washington.edu/northcreek

Your 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.

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.

First, the program prints an introduction in the form of a haiku poem. Recall that 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 he/she would like to play again. 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. 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.

<< your haiku intro message here >>

```
I'm thinking of a number between 1 and 100...
      Your guess? 50
      It's higher.
      Your guess? 70
      It's lower.
      Your guess? 60
      It's lower.
      Your guess? 55
      You got it right in 4 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? 80
It's lower.
Your guess? 70
It's lower.
Your guess? 57
It's lower.
Your guess? 53
It's higher.
Your guess? 55
It's higher.
Your guess? 56
You got it right in 7 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? 30
It's lower.
Your guess? 15
It's lower.
Your guess? 7
It's higher.
Your guess? 10
It's higher.
Your guess? 13
It's lower.
Your guess? 12
It's lower.
Your guess? 11
You got it right in 8 guesses!
Do you want to play again? No

Overall results:
Total games   = 3
Total guesses = 19
Guesses/game  = 6.3
Best game     = 4
```

First, the program prints an introduction in the form of a haiku poem. Recall that 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 he/she would like to play again. 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. 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.

You should handle the special case where the user guesses the correct number on the first try. Print a message as follows:

```
I'm thinking of a number between 1 and 100...
Your guess? 53
You got it right in 1 guess!
```

Assume valid user input. When prompted for numbers, the user will type integers only, and they will be in proper ranges.

Implementation Details

Define a **class constant** for the maximum number used in the games. The previous page's log shows games from 1 to 100, but you should be able to change the constant value to use other ranges such as from 1 to 50 or any maximum.

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 every-thing 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 on this assignment and will result in a deduction. To help you solve the "best game" part of the program, you may want to read textbook section 4.2 on min/max loops.

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 may be useful to print a temporary "hint" message like that shown at below. This way you will know the correct answer and can test whether the pro-gram gives proper clues for each guess. This is also helpful for testing the "1 guess" case.

```
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!
```

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 limitation that methods can return only one value will limit how much you can decompose the problem. It is okay for some `println` statements to be in `main`, as long as you use good structure and `main` is a concise summary. For example, you can place the loop for multiple games and the prompt to play again in `main`.

Use whitespace and indentation properly. Limit lines to 100 characters. Give meaningful names to methods/variables, and follow Java's naming standards. Localize variables. Put descriptive comments at the start of your program and each method. Since this program has longer methods, also put brief comments inside methods on complex sections of code.

Guessing Game: FAQ

Q: Where is there an output comparison tool?

A: https://courses.cs.washington.edu/courses/cse142/19au/diff.html?assignment=a5
Yes, this is on the UW site.

Q: Where do I start?

A: Read the write-up in its entirety. Try to develop the program in stages. Review example programs from section and lecture, such as LoginSystem, Roulette, and AdditionGame.

Q: What is an `InputMismatchException`? Why does my program give me that error when reading from the `Scanner`?

A: This error occurs when you try to read the wrong kind of value from the `Scanner`. You are probably calling the wrong method on it. If you want to read a word, use `.next()`. If you want to read an `int`, use `.nextInt()`.

Q: How can I return two values from one method?

A: You can't. Maybe you have chosen the wrong method structure; for example, maybe the method should pass those two values as parameters to another method, instead of returning them. Or maybe your method is too large and you should break it into smaller pieces.

Q: How do I compute the best game (fewest guesses)?

A: You mostly have to figure this out for yourself, but here are some hints:

- In each game you should keep track of how many guesses were needed.
- You should probably also keep track of the best game you have seen so far, and update that value accordingly after every game, if the game is better than the best one seen previously.
- You will have to carefully manage your returns and parameters to make sure that the right information reaches the right parts of your code, but it can be done.

Q: Is it okay to have a method that calls itself? Or to have a Method A that calls Method B, which calls A, which calls B, ...?

A: No, you should not do this on Homework 5. Having a method call itself is actually an advanced computing technique called *recursion*. Recursion is not an appropriate algorithmic technique for solving this problem. You can solve the problem correctly using `while` loops for repetition, instead of recursion.

Q: Since the program is so random, how can I possibly match the expected output? How can I use the Output Comparison Tool?

A: You don't have to match the randomly generated numbers shown in our output logs. But you should have the same format as our logs.

If you *do* want to exactly match our output so that you can see the coveted "No differences found" in the Output Comparison Tool, it can be done. To do it, you can force your `Random` object to return the same sequence of random numbers on every run of the program. This is called *seeding* the `Random` object. Random numbers are generated from a mathematical function, and a seed is an integer you pass to the `Random` object as it's being constructed. For example, to seed your `Random` object with the value 42, you'd write:

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

The Output Comparison Tool page shows the seeds we used to generate our logs of expected output.

Please also note that in order for the above technique to work, you must create only a single `Random` object throughout your program and pass it to each of your various methods. If you create several new `Random` objects, such as one for every round of the guessing game played, your output will not match the output shown.

Q: Do I have to create a Random object, or should I use Math.random()?

A: I highly suggest that you use Math.random() and not the Random object. The book teaches the Random class, but the AP Exam does not use Random, and neither will Mr. Stride's quizzes or exams. You absolutely need to know how Math.random().

Furthermore, if you use Math.random(), you are much more likely to pass the unit tests because you won't create any Random objects and the requirement to create at most one object is obviated.

Rubric

30 : Total Score

- **15 : External Correctness**
  - **6 : Single game**
    - **2 :** Prompts until a correct guess is made
    - **2 :** Gives correct higher/lower clues
    - **2 :** Correct count of guesses for each game
  - **4 : Multiple games**
    - **2 :** Can play arbitrarily many games
    - **1 :** Chooses a new random value from 1-MAX each game
    - **1 :** Handles Y/N prompt perfectly
  - **3 : Statistics**
    - **2 :** Total games and total guesses
    - **1 :** Guesses per game (including rounding to one digit after decimal)
  - **2 :** All other output matches exactly (intro, formatting, etc.)
- **15 : Internal Correctness**
  - **2 :** Constant is declared properly (public static final) and used throughout program
  - **2 :** Play Game method: has a method that plays at least 1 game
  - **2 :** Has a method to print overall statistics
  - **3 : Procedural design heuristics**
    - **2 :** Each method has a clear task; no redundancy between methods; parameters used properly
    - **1 :** Main is a concise summary; no chaining or coupling (returns used properly)
  - **2 :** Comments on header and at top of methods
  - **4 :** Otherwise good style