CSE 142, Autumn 2009
Programming Assignment #5: Guessing Game (20 points)
due Tuesday, November 3, 2009, 11:30 PM

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

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.

```
<< 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 a 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. Your haiku can be posted to Facebook.

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? 71
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 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
```

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 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.

As in Homework 2, we suggest that you add the constant last, after all of the other code works properly.

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; this is 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 4.3 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 5 guesses!
```

(suggested initial simple version of program)

We suggest that you begin by 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.

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`. As a reference, our solution has 4 methods other than `main` and occupies roughly 90 lines.

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.

Extra Credit:

On this assignment you can earn a limited amount of extra credit by completing some optional features in your program. The features relate to testing for valid user input. See the course web site for sample logs that show extra credit output.

Extra Feature #1: Making sure the user's input is an integer and is in the proper range

```
I'm thinking of a number between 1 and 100...
Your guess? 50
It's lower.
Your guess? 25
It's higher.
Your guess? -1
Your guess? 37
It's higher.
Your guess? 43
It's higher.
Your guess? 105
Your guess? -10
Your guess? 123456
Your guess? 47
It's lower.
Your guess? 1000
Your guess? what?
Your guess? dunno
Your guess? 0
Your guess? 999
Your guess? 101
Your guess? 3.14159
Your guess? yes
Your guess? 46
You got it right in 6 guesses!
...
```

For your first extra feature, repeatedly re-prompt the user as needed until he/she types a valid guess. A valid guess is an integer in the range from 1 to the maximum in the game (as per the class constant), such as 1-100.

If the user types an invalid value, such as an integer outside the legal range or a token that cannot be interpreted as an integer, reject this input and prompt the user again. You may still assume the user types a 1-word answer.

It is helpful to think of this as two separate tests: One to see whether the user typed an integer (as opposed to some other token such as a real number or string), and a second test to see whether that integer is in a valid range. To detect whether the user has typed an integer token, use the `Scanner`'s `hasNextInt` method described in textbook section 5.3. If the user types a non-integer token, tell the `Scanner` to consume it (remove from the input) by calling the `next` method on the `Scanner` and ignoring the token that is returned.

Invalid guesses do not count toward the total needed to find the answer. Notice the total of 6 in the log shown.

Extra Feature #2: Making sure the user types a "y" or "n" answer (case-insensitive)

```
...

Do you want to play again? what?
Do you want to play again? ok
Do you want to play again? please
Do you want to play again? 42
Do you want to play again? :-)
Do you want to play again? yup

...

Do you want to play again? maybe
Do you want to play again? NO
```

For your second extra feature, modify your program so that when each game ends and the user is prompted to play again, it will repeatedly re-prompt the user until he/she types a word that begins with Y/y or N/n.

Any word or token that starts with any other character should be ignored, and the user should be re-prompted. This is a modification to the original specification, in which a word that began with a letter besides Y/y was assumed to mean "no."

The reward for completing these features is the following:

- If you complete one of the above extra features correctly, or more than one but with minor mistakes:
  - **+1 extra late day**
- If you complete all of the above extra features, and all of them work successfully:
  - **+1 point added to your HW5 score**, if your HW5 score is less than a perfect 20 / 20; or,
  - **+1 extra late day**, if your HW5 score is already a perfect 20 / 20.

In other words, this is a limited kind of extra credit that can be used to improve your HW5 score, but cannot be used to improve your score on other assignments outside of HW5. If your HW5 score is already full credit or if your attempt to solve the extra credit features has minor mistakes, you will instead be awarded an extra late day.

We suggest that you add methods to handle the above features. You may want to return a `boolean` value as appropriate. If you choose to add these features, your program is still subject to the grading and style guidelines specified previously.

After adding the above features, our solution is roughly 110 lines long and has 6 methods besides `main`.