University of Washington
Computer Science & Engineering 142 (Computer Programming I), Summer 2005

Programming Assignment #5 (Guessing Game)
Due: Tuesday, 7/26/2005, 6:00 PM

Problem Description:

This assignment will give you practice with `while` loops and pseudorandom numbers. Write a Java class named `GuessingGame` in a file named `GuessingGame.java`. This program requires the Scanner for user input, so place `Scanner.java` in the same folder as your program.

Your program allows the user to play a guessing game in which your program thinks up an integer and allows the user to make guesses until the user gets it right. For each incorrect guess you will tell the user whether the right answer is higher or lower. Your program is required to exactly reproduce the format and behavior of the log of execution at the end of this document, so you may want to look that over first.

Program Behavior:

First, the program prints a header message describing itself; next, a game is played. The computer chooses a random number between 1 and 100 inclusive. The game repeatedly asks the user for his/her guess until the correct number is guessed. After each guess, if the guess is incorrect, the program reports to the user whether the correct number is higher or lower than that guess. When the game ends, the program reports how many guesses were needed.

After each game, the program asks the user if he/she would like to play again. You may assume that the user will give a one-word answer. You should continue playing if this answer starts with a lower- or upper-case Y. That is, answers such as "y", "Y", "YES", "yes", "Yes", or "yEs" would all indicate that the user wants to play again. If the answer is not a definitive Yes answer, assume that the user does not want to play again. For example, responses such as "no", "No", "0", and "Pass" are all assumed to mean no. If the user answers a Yes answer, another game is played.

Once the user ends a game and chooses not to play again, the program prints overall statistics about the 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 hundredth**), and best game (fewest guesses) are displayed. Your statistics should present correct information regardless of how many guesses were needed or games were played, within the limits of Java's maximum values. (In other words, the statistics should be correct even if the user has a game where they need a large number such as 1000 guesses to get the right answer, and even if the user plays a large number of games, and even if only 1 game is played, and so on.)

Implementation Guidelines:

You do not have to check for valid input. When the user is prompted for a number, assume that the user does type an integer and that the integer is between the minimum number 1 and the maximum number 100. When the user is prompted to play again, assume that the user types a one-word String answer.

Because this program uses pseudorandom numbers, you may not be able to recreate this exact log. The key requirement is that you reproduce the format and structure of this log. Also be sure that the statistics reported by your program are correct (total games, total guesses, average number of guesses, best game). This program needs to generate pseudorandom numbers. This is described in section 5.2.4 of the book.

(If you wish to attempt to recreate the exact log of output below, you can create a single Random object with a seed value of 42 and use this object to draw all random numbers for the program. However, any valid random numbers will do.)

When you ask the user whether or not to play again, you should use the `next()` method of the `Scanner` class to read a one-word answer from the user. To deal with the yes/no response from the user, you might want to use some of the String class methods described in section 3.5.1 and 4.5 of the book.

If you are having problems with integer division producing the wrong results, you may wish to read about type-casting in section 2.3.4 of the book.

Notice that all of the output below is from one run of the program; it is a single log of execution, not multiple logs as were shown in the last assignment.

Example log of execution (user input underlined)

(Your program may generate different random numbers, but your output structure should match this one)

```
This program allows you to play a guessing game.
I will think of a number between 1 and 100
and will allow you to guess until you get it right.
For each guess, I will tell you whether the
correct answer is higher or lower than your guess.

I'm thinking of a number between 1 and 100...
             Your guess? 20
It's higher. Your guess? 40
It's lower.  Your guess? 30
It's higher. Your guess? 32
It's lower.  Your guess? 31
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? 75
It's lower.  Your guess? 25
It's higher. Your guess? 50
It's higher. Your guess? 60
It's higher. Your guess? 70
It's lower.  Your guess? 65
It's lower.  Your guess? 62
It's higher. Your guess? 63
It's higher. Your guess? 64
You got it right in 9 guesses!

Do you want to play again? YES

I'm thinking of a number between 1 and 100...
             Your guess? 50
It's lower.  Your guess? 25
It's higher. Your guess? 37
It's higher. Your guess? 43
It's higher. Your guess? 47
It's higher. Your guess? 49
You got it right in 6 guesses!

Do you want to play again? no

Overall results:
total games   = 3
total guesses = 20
guesses/game  = 6.67
best game     = 5
```

Stylistic Guidelines:

For this assignment you are limited to the language features in Chapters 1 through 5; you are not allowed to use more advanced features to solve the problem. Please do not use Java features that were not covered in lecture or 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 3 methods other than `main` in your program:**

- a method to print the welcome message and instructions to the user
- a method to play one game with the user (*not* multiple games)
- a method to report the overall statistics to the user

You may define more methods than this if you find it helpful, although you will find that the limitation that methods can return only one value will tend to limit how much you can decompose this problem.

This program is more difficult to decompose into methods than past programs, so you may end up having methods that are longer. You can also include more code in your `main` method than we allowed in the last program. In particular, you may put the loop in `main` that plays multiple games and prompts the user for whether or not to play another game. You still should not have all of the program's code in `main` because you are required to have the methods described at the beginning of this write-up.

For full credit, you must define a class constant for the maximum number used in the guessing game. The sample log shows the user making guesses from 1 to 100, but the choice of 100 is arbitrary. By introducing a constant for 100, you should be able to change just the value of the constant to make the program play the game with a range of 1 to 50 or a range of 1 to 250 or any other range starting with 1. Use your constant throughout your code and do not refer to the number 100 directly. Your program should execute correctly with a different maximum value simply by changing your constant. It is a good idea to change the value of your class constant and run the program to make sure that everything works right with the new value. For example, turn it into a guessing game for numbers between 1 and 5.

You are required to properly indent your code and will lose points if you make significant indentation mistakes. See section 2.5.3 of the book for an explanation and examples of proper indentation. You should also use white space to make your program more readable, such as between operators and their operands, between parameters, and blank lines between groups of statements or methods.

Give meaningful names to methods and variables in your code. Follow Java's naming standards about the format of ClassNames, methodAndVariableNames, and CONSTANT_NAMES. Localize variables whenever possible -- that is, declare them in the smallest scope in which they are needed.

Include a comment at the beginning of your program with basic information and a description of the program and include a comment at the start of each method. **Since this program has longer methods than past programs, put brief comments inside the methods explaining the more complex sections of code.**

Submission and Grading:

Name your file `GuessingGame.java` and it them in electronically from the "Assignments" link on the course web page. You do not have to turn in `Scanner.java`. This assignment is worth 20 points total.

Part of your program's score will come from its "external correctness." External correctness measures whether your log of execution matches exactly what is expected, including identical prompting for user input.

The rest of your program's score will come from its "internal correctness." Internal correctness measures whether your source code follows the stylistic guidelines specified in this document. This includes using `while` or `do/while` loops to capture indefinite repetition, representing the structure and redundancy of the program using appropriate parameterized static methods with return values as needed, commenting, naming identifiers, and indentation of your source code.