CSE 142, Summer 2008
Programming Assignment #5: Guessing Game (20 points)

Part A (single game) due Wednesday, July 30, 2008, 11:00 PM
Part B (complete program) due Friday, August 1, 2008, 11:00 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.

This assignment will be due in two parts: an initial version A that plays only a single game, and the second complete (multi-game) version B a few days later. The initial Part A will not be worth as many points as the second Part B.

Program Behavior (Part B):

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

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 lower.
Your guess? 45
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? 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? YES

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? no

Overall results:
total games   = 3
total guesses = 20
guesses/game  = 6.7
best game     = 5
```

First, the program prints a header message describing itself. 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 reports to the user whether the correct number is higher or lower. 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. A new game should begin if this answer starts with a lower- or upper-case Y. That is, 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", and "hello" are all assumed to mean no. Assume the user will always give a one-word answer.

Once the user chooses not to play again, the program prints overall statistics. 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) are displayed. Your statistics should present correct information for any number of games ≥ 1, and any number of guesses ≥ 1 in each game. You may assume that no game will require one billion or more guesses.

You should not write any special code to handle the case where the user guesses the correct number on the first try. Print the same message as usual:

```
You got it right in 1 guesses!
```

Your program will have different random numbers, but your output's structure should match the output shown.

Program Behavior (Part A):

```
I'm thinking of a number between 1 and 100...
(The answer is 48)
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? 48
You got it right in 6 guesses!
```

In Part A, a single guessing game is played. It does not print a welcome message or prompt to play more games.

Notice that in Part A, the program should print the game's correct answer first ("`(The answer is 48)`", at left). This is for your own debugging purposes, so that you can test your program logic and make sure you are giving the correct lower/higher hints. Before you turn in Part B this correct answer hint message should be removed, but you may want to leave it in while you're developing Part B to help you test your code.

Implementation Guidelines:

In Part B, 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 you should be able to change just the value of the constant to cause the program to play the game with other ranges, such as a range of 1 to 50, a range of 1 to 250, or any range starting with 1. Use your constant throughout your code and do not refer to the number 100 directly. Test your program by changing the value of your constant and running the program again to make sure that everything works right with the new value. For example, run a guessing game for numbers between 1 and 5. The web site shows expected output for such a case.

Assume valid user input. When prompted for numbers, the user will type valid integers in proper ranges. When the user is prompted to play again, the user will type a one-word answer. Read the answer using the `Scanner`'s `next()` method. To check for a yes/no user response, you may want to use `String` methods described in Chapters 3-4 of the book. If you get an `InputMismatchException` error, it means you are trying to read the wrong type of value from a Scanner. For example, you are trying to read an integer when the user has typed a word.

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 particularly relevant example for this assignment.

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

Stylistic Guidelines:

For this assignment you are limited to the language features in Chapters 1-5 shown 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 **3 non-trivial methods other than main** in your program. Two of these must be the following:

1. a method to play one game with the user (not multiple games)
2. a method to **report the overall statistics** to the user (and nothing more)

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.

You may define other methods if they are useful for structure or to eliminate redundancy. Unlike in past programs, it is okay to have some `println` statements in `main`, as long as your program has good structure and `main` is still a concise summary of the program. For example, you can place the loop that performs multiple games and the prompt to play again in `main`. As a reference, our solution has 4 methods other than `main` and occupies between 80-100 lines total.

For this assignment you are limited to the language features in Chapters 1-5 of the textbook. Use whitespace and indentation properly. Limit lines to 100 characters. Give meaningful names to methods and variables, and follow Java's naming standards. Localize variables whenever possible. Include a comment at the beginning of your program with basic description information and a comment at the start of each method. Since this program has longer methods than past programs, also put brief comments inside the methods explaining relevant sections of your code.