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.

```
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
```

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:

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 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 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.