# —Assignment #2—

Make sure to review the submission requirements on pg. 4 of the Lab #1 document.

Follow the Style Guide as given in the Resources section of our conneX course page.

## Learning Outcomes

When you have completed this assignment, you should understand:

- How to use procedural decomposition to break down a problem into methods.
- How to use nested `for` loops to create triangular ascii art.
- How to use a `Scanner` to process `int` data type user input.
- The flow of control of `if-else` statements. (We cover this on May 25)
- How to develop a strategy to test code for correctness.

A simple game to play between two people:

(a) first player picks a random number from 0 to 100.

(b) the second player guesses the number.

(c) if the second player guessed correctly, they win! The game stops.

(d) otherwise, the guess is either too low or too high and the first player says so.

(e) repeat steps (b)–(d) some number of times, perhaps 10 times.

(f) Once the number of guesses is up without the second player winning, instead they lose.

(g) the first player tells the second player the number.

You will write a program to play this number guessing game. Call your program `GuessingGame.java` and compile and run it frequently as you work through each piece to build it. When fully programmed and it is compiled and executed, the computer will play as the first player and the user will play as the second player. Please make your output look like the following two sample runs given on the next page (winner in left column, loser in the right column). The differences between the two sample runs emphasize which parts of the program are user input.

The best strategy to win is to simply choose the number halfway in the range that you know the secret number must be between. As in the winning sample run, the first best guess is 50, because the number must be from 0 to 100, and 50 is halfway between. If the secret number is higher than fifty, the next best guess is 75. If the secret number is less than 50, the next best guess is 25. This way, each guess reduces the range you know where the secret number is by half.

```
Welcome to Guessing Game!
-------------------------
Enter a number to play:
(1 to 2147483647) -> 1
Generating random number.
----RANDOM NUMBER SET----
Guess my number (0--100):
You have 10 guesses left.
What is your next guess? 50
Your guess was too low.
You have 9 guesses left.
What is your next guess? 75
Your guess was too high.
You have 8 guesses left.
What is your next guess? 62
Your guess was too high.
You have 7 guesses left.
What is your next guess? 57
Your guess was too low.
You have 6 guesses left.
What is your next guess? 60
You guessed it!
-------------------------
            .
           ..*
          *....
         ....*..
        .*..**.**
       **.**.*.**.
      ..**..**.**..
     ...**....**..*.
    *.***.**.*..*.**.
   .**.***.**.***.**.*
  ***.*...*.**..*.*.*.*
 *..*.****..*...**.*....
*****.....*..*.....*.*.*.
CONGRATULATIONS, YOU WIN!
```

```
Welcome to Guessing Game!
-------------------------
Enter a number to play:
(1 to 2147483647) -> 78
Generating random number.
----RANDOM NUMBER SET----
Guess my number (0--100):
You have 10 guesses left.
What is your next guess? 1
Your guess was too low.
You have 9 guesses left.
What is your next guess? 2
Your guess was too low.
You have 8 guesses left.
What is your next guess? 3
Your guess was too low.
You have 7 guesses left.
What is your next guess? 4
Your guess was too low.
You have 6 guesses left.
What is your next guess? 5
Your guess was too low.
You have 5 guesses left.
What is your next guess? 6
Your guess was too low.
You have 4 guesses left.
What is your next guess? 7
Your guess was too low.
You have 3 guesses left.
What is your next guess? 8
Your guess was too low.
You have 2 guesses left.
What is your next guess? 9
Your guess was too low.
You have 1 guesses left.
What is your next guess? 10
Your guess was too low.
The number was 48
-------------------------
Sorry, you lost. Try again!
```

The game will need to get input from the user, so make sure you have an import declaration at the top of your `GuessingGame.java` file:

```
import java.util.Scanner;
```

There are two class-scope variables you need to declare at the top of your `GuessingGame` class:

```
static final Scanner CONSOLE = new Scanner(System.in);
static int version = 1;
```

Recall that constants are declared with the `final` keyword when we only need one value to stay the same throughout the execution of our program. We only need one `Scanner` instance to get input from the user, and the methods of `GuessingGame` will share this instance. We will set the value of `version` later, and it will be used to control which random integer is created for the game.

Let's go through the `public static` methods you need for your `GuessingGame` class:

- `int rand(int max, int x)`
- `void welcome()`
- `int getNum()`
- `boolean play(int num)`
- `void confetti()`
- `void goodbye(boolean win)`

The `main` method of `GuessingGame` will be very simple once you are done:

```
public static void main(String[] args) {
    welcome();
    int num = getNum();
    boolean win = play(num);
    goodbye(win);
}
```

Only add the method calls you have working as you build up your program to test it.

We have already seen the `rand` method before, that lets you control which random number is generated. We could use `Math.random()`, but this would make it more difficult to test your program. `Math.round()` returns data type `long`, but here is the method with the return value cast as an `int` data type:

```
public static int rand(int max, int i) {
    double frac = Math.abs(9437 * Math.sin(1009*i)) % 1.0;
    return (int)Math.round(max * frac);
}
```

The first parameter `max` controls the range of integer values that can be generated, from zero to `max`. The second parameter `i` controls which random number is returned from the sequence of possible random numbers the formula generates.

The `welcome` method is simple:

```
public static void welcome() {
    System.out.println("Welcome to Guessing Game!");
    System.out.println("-------------------------");
    System.out.println("Enter a number to play:");
    System.out.print("(1 to 2147483647) -> ");
}
```

Notice the last `print` statement is set so that you can later prompt the user to enter an integer on the same line.

The `goodbye` method is also very simple:

```
public static void goodbye(boolean win) {
    System.out.println("-------------------------");
    if (win) {
        confetti();
        System.out.println("CONGRATULATIONS, YOU WIN!");
    } else {
        System.out.println("Sorry, you lost.  Try again!");
    }
}
```

Notice that this method calls the `confetti` method, which you will be asked to write later. For now, you could comment this call statement out until you start working on it, but remember to uncomment it later if you do!

Write the method `getNum` with `int` return data type. It should:

(a) use `CONSOLE` to prompt the user for an integer value that gets stored in the variable `version` (our class-scope variable).

(b) print the "generating random number" statement.

(c) make a call to `rand(100, version)` and store the return value in a local variable.

(d) print the "random number set" statement.

(e) return the value of your local variable, which is the random number for the game.

Compile your program so far and make sure there are no errors. You can begin to test your program's correctness by calling `getNum` in `main` and printing the returned integer. It should print the same random number depending on the integer you input as the user of your program. For example, when I run and test it with the input value 1, I get the return value 60.

For your `play` method, it needs a return data type `boolean` and one `int` parameter. Break up your

work on this method by starting small and building it up. First, simply print "`Guess my number (1--100):`" on one line, with the next line telling the user they have ten guesses left. Ask them for their next guess, and then store it in a variable. Then print the guess back to test your use of `CONSOLE`. The last line of your method needs a return statement, so simply return `true` for now. Compile and test your work.

Then improve your `play` method to ask the user for input ten times, and print their input for each time. Model your output to look like the sample runs given. Compile and test your work.

Now use `if` and `if-else` statements to complete your `play` method to test the user's guess against the parameter variable. There are three cases to check: equal to, less than, or greater than. Make sure to add a return `true` statement in the case the guess is equal to the parameter, so that the user can win. Also, change the end of your method, it should return `false`, because after you guess and test ten times, if the user never guessed the parameter correctly, then they should lose the game. Compile and test your method by calling it in `main` with an argument like `7`, so when you input your guesses, you can check each case easily (equal, too low, too high).

Once you get both `getNum` and `play` working to your satisfaction, you can then actually play the game by using them as shown in the above `main` method, without the call to `goodbye`, of course.

Remember to uncomment the call to `confetti` in the `goodbye` method before you start working on it. The `confetti` method takes no parameters and returns nothing. Once complete, it should print a pyramid of random star and period characters 13 lines high and 25 characters wide. Again, start small and build up to the goal. You know how to print a rectangle of stars, so begin by using nested `for` loops to print a rectangle of 25 stars wide by 13 lines high. Compile and test (you could simply have `confetti` as your only call in `main` while you test it).

Then modify your rectangle to print a pyramid of stars, the exact same shape as in the sample runs given. You will need a second inner `for` loop to print the space characters needed to form the left side of your pyramid properly. You don't need to print the spaces on the right of the pyramid. It helps to think of the number of stars you need at the top of the pyramid and how many more stars are needed for the next line. Once you figure out the expressions you need in the headers of your loops, compile and see if the shape matches with the pyramid for the sample run output in the winner's case.

Now before your nested loops, add an integer variable `count` and set it to zero. Replace the statement you used to print stars with code to generate a random number 0, or 1. So, instead of printing star characters, add the statement:

```
int binaryNum = rand(1, (count++) + 239*version);
```

This sets `binaryNum` randomly to either zero or one. After this statement, add `if-else` statements to test the value of binaryNum whether it is zero or one. If it is one, print a star. Otherwise, print a period. Compile and test. If you are only calling `confetti` in `main`, then your pyramid should match the output of the sample run's winning pyramid. This is because the value of `version` is default set to 1.

The variable `count` is incrementing by one each time we generate a random number, so that a

different random number is generated each time we call `rand`. The formula uses `239*version` because there are exactly 239 characters in the pyramid, so we get a different pattern of stars and periods depending on the first input the user entered to play the game.

Now you can put all your methods together and play the full game with statements exactly as given in `main` above!

## Grading

Marks will be allocated out of 30 points:

- (0 points) You do not submit any program file.
- (5 points) Your program does not compile. The marker cannot fix the complier error easily, or if they can, the output is severely mismatched with the assignment requirements.
- (10 points) Your program does not compile, but the marker can fix the error easily.
- (15 points) Your program compiles, but only outputs a few of the assignment requirements.
- (20 points) Your program compiles, outputs most of the assignment requirements, and is missing documentation comments or has formatting issues.
- (25 points) Your program compiles, outputs everything required, but is missing documentation comments or has formatting issues.
- (30 points) Your program compiles, outputs everything required, and has proper documentation and formatting.

---

This is not part of your submission of the above assignment. If you want to make your program generate a different pseudorandom number each time you play without the user controlling which number is generated, then you can skip ahead to pages 320 and 321 of our textbook, and modify your program using a `Random` object. If you do, you will need an import declaration at the top of your file to use it, as follows:

```
import java.util.Random;
```

Then you could modify your game so that it does not need to ask the user for the first input number, and instead let the user start playing the game immediately. This eliminates the need for the variable `version`.