**Lab 12 – Guessing Game**

Open BlueJ, and create a new BlueJ project titled **Lab12-GuessingGame** in your CS\LABS folder.

Create a new class with this code skeleton:

//Name:

import java.util.\*;

public class PracticeProblems

{

public static void main(String[] args)

{

Scanner console = new Scanner(System.in);

}

}

A while loop is very similar to an if statement, except that the code inside the code block (between the curly brackets) **will continue to run** *while* the test condition is true. Example syntax:

int i = 0;

while (i < 50) //will execute while i is less than 50

{

System.out.print(i);

i = i + 1; //can also be written as i++

}

Much more information can be found in our notes.

**Before each problem, insert a COMMENT with the problem number.**

1. Prompt the user, asking them to enter an integer *n*. Write the code to print out all the numbers from 1 to *n*.
2. Get a new value for *n* from the keyboard. Write the code to sum up all the numbers from 1 to *n*. Example (**user input shown in red**):

Enter an integer from 1 to 1000 >>> **250**

Sum of all numbers from 1 to 250 >>> 31375 //concatenate the value of n here

1. Get a new value for *n* from the keyboard. Write the code to sum up all the even numbers from 1 to *n.* Example (**user input shown in red**):

Enter an integer from 1 to 1000 >>> **250**

Sum of all even numbers from 1 to 250 >>> 15750

1. (Riddle) What word describes a person that does not have all their fingers on one hand?
2. Get a new value for *n* from the keyboard*.* Write the code to sum up all the multiples of 7 from 1 to n*.* Example (**user input shown in red**):

Enter an integer from 1 to 1000 >>> **250**

Sum of all multiples of 7 from 1 to 250 >>> 4410

**Guessing Game app**

//Name:

import java.util.\*;

public class GuessingGame

{

public static void main(String[] args)

{

Scanner console = new Scanner(System.in);

Random randomGen = new Random();

}

}

*Note: This game doesn’t use accumulation, but it does use loops. This is the most complex program we’ve written so far. If you immediately start coding without a solid understanding of how to proceed, you could find yourself wasting significant time on a solution that isn’t remotely feasible. Before you begin, write out (by hand or in an outline in Word, etc.) your plan prior to coding. (This is called pseudo-coding.)*

Create a program that will allow the user to play a number guessing game. First, declare an integer variable called *guesses* and set it equal to 7. Next, generate a *random number*\* and save it into an integer variable called *number*. Now, prompt the user, asking them to enter a number from 1 to 1000 (your program should protect against numbers entered that are outside that range).

Your program should them tell them if they guessed correctly. If they didn’t, tell them if they were too high or too low. Allow them to guess again, repeating this process until they either guess the number correctly (and win) or run out of guesses (and lose).

Click [here](https://youtu.be/3xs0DtApO6c) to see a sample run of the program.

\* To create a random number, add the following code to your program:

Random randomGen = new Random(); //you only need to do this once at the beginning! It is sort of like the Scanner instantiation line of code.

int number = randomGen.nextInt(1000) + 1; //run this when you want to get a new random number up to 1000. The random number will be stored in the variable ‘number’.

HINT: We DON’T want to make a new random number each time our user guesses – just once per run of the game!

**Advanced (Optional)** - Are you done with Lab 12? Here’s a real challenge…

Ask the user to supply their own upper bound. Instead of 1 to 1000, ask them how high the range can be.

Then, let the user select Easy, Medium or Difficult mode. Based on the size of the guessing field, calculate how many guesses you need to give the player such that optimal strategy yields these chances for success:

Easy: about a 50% chance of success using optimal strategy.

Medium: about a 20% chance of success using optimal strategy.

Hard: about a 5% chance of success using optimal strategy.