**Name**

**Advanced Programming in Java**

**Lab Exercise 11/13/2020**

**Lesson 30 – Random Numbers**

In the following problems assume that *rndm* is an object created with the *Random* class. Assume *d* is of type *double* and that *j* is of type *int*.

1. What range of random numbers will this generate?

j = 201 + rndm.nextInt(46);

2. What range of random numbers will this generate?

d = 11 + 82.9 \* rndm.nextDouble( );

3. What range of random numbers does *nextDouble( )* generate?

4. List all numbers that rnd*m.nextInt(10)* might generate.

5. Write code that will create an object called *rd* from the *Random* class.

6. Write code that will create a *Random* object and then use it to generate and print 20

floating point numbers in the continuous range 22.5 < r < 32.5

7. What import is necessary for the *Random* class?

8. Write code that will randomly generate numbers from the following set. Printout 10 such numbers.

18, 19, 20, 21, 22, 23, 24, 25

9. Write code that will randomly generate and print 12 numbers from the following set.

100, 125, 150, 175

10. Write a line of code to create a *Random* class object even though *Random* wasn’t

imported.

**Project… Generate Random Integers**

As described below, generate 33 random integers in the inclusive range from 71 to 99.

Suppose we want a range of integers from 90 to 110, inclusive for both.

First we subtract (110 – 90 = 20). Then add 1 to get 21. Now set up your code as follows

to generate the desired range of integers:

int r = 90 + rndm.nextInt(21);

Put this last line of code in a *for*-loop, and you will see a range of integers from 90 to

110. Loop through 1000 times, and likely you will see every value…most will be

repeated several times.

int r = 0, count = 0;

Random rndm = new Random( );

for(int j = 0; j < 1000; j++)

{

r = 90 + rndm.nextInt(21);

System.out.print(r + “ ”);

//For convenience in viewing on a console screen, the following loop

//produces a new line after 15 numbers are printed side-by-side.

count++;

if(count >15)

{

System.out.println(“ ”);

count = 0;

}

}

**Project… Generate Random Doubles**

As described below, generate 27 random *double*s in the inclusive range from 99.78 to 147.22.

Suppose we wish to generate a continuous range of floating point numbers from 34.7838 (inclusive) to 187.056 (exclusive). How would we do this?

First, subtract (187.056 – 34.7838 = 152.2722). Now set up your code as follows

to generate the desired range.

Random rndm = new Random( );

double r;

r = 34.7838 + 152.2722 \* rndm.nextDouble( );

// Generates continuous floating point numbers in the range

// 34.7838 < r < 187.056

**Project… Monte Carlo Technique**

Imagine a giant square painted outdoors, on the ground, with a painted circle inscribed in it. Next, image that it’s raining and that we have the ability to monitor every raindrop that hits inside the square. Some of those raindrops will also fall inside the circle, and a few will fall in the corners and be inside the square, but not inside the circle. Keep a tally of the raindrops that hit inside the square (*sqrCount*) and those that also hit inside the circle (*cirCount*).

**The ratio of these two counts should equal the ratio of the areas** as follows: (Understanding this statement is essential. It is the very premise of this problem.)

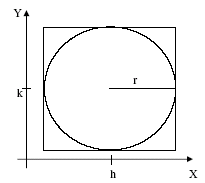
sqrCount / cirCount = (Area of square) / (Area of circle)

sqrCount / cirCount = side2/ (π \* r2)

Solving for from this equation we get

π = cirCount \* (side2) / (sqrCount \* r2)

So why did we solve for π? We already know that it’s ≅ 3.14159. We simply want to illustrate that by a simulation (raindrop positions) we can solve for various things, in this case something we already know. The fact that we already know π just makes it that much easier to check our answer and verify the technique. We are going to build a class called *MonteCarlo* in which the constructor will establish the size and position of our square and circle. Public state variables inside this class will be *h*, *k*, and *r*. These are enough to specify the position and size of our circle and square as shown in the figure to the right.



Problem 1: Write a program that generates 1000 random integers ranging from 50 to 100 inclusive. The numbers should be printed 15 per line.

Problem 2: Write a program that generates 100 random doubles ranging from 27.1352 (inclusive) to 119.3473 (exclusive). The numbers should be printed 5 per line.

Problem 3: Use Monte Carlo techniques to calculate the value of Pi.

Bunny Hill Version: Use a unit circle with center at (0, 0)

Black Diamond Version: Allow a circle with center at any (h, k) and any radius r.

When you complete these programs, submit the documented source code.