ASSIGNMENT 2
Part I
CSc 221
Spring 2016

It is advisable you carefully read the *entire* assignment and think about it before raising questions. Also read http://whatis.techtarget.com/definition/normal-distribution for a quick intro to the normal distribution.

I. The program

Write a program comprising *three classes*: **MonteCarlo** (with the **main()** method), **Simulation**, and **Metrics**. The program stores many (say 100,000) normally distributed (or Gaussian) random numbers in an ArrayList, organizes them in a visually useful way, and performs simple operations on them to test their validity.

Simulation contains the following methods.

generateNormalRandomNumbers(ArrayList<Double>, int)

The ArrayList<Double> parameter is an (empty) ArrayList, which the method populates with random numbers; the int parameter is the number of random numbers to be generated (in other words, it determines the size of the ArrayList).

makeBins(ArrayList<Double>, int)

The ArrayList<Double> parameter is an ArrayList with random numbers; the int parameter is the number of "bins" (or "buckets") that will be returned by the method as an array. makeBins() will create a number (passed as the int parameter) of bins in one of which each random number in ArrayList will fall. Each bin will keep count of the number of random numbers in ArrayList that fall within a certain range represented by the bin. So, the value of each bin after all random numbers have been considered will represent the *count* of random numbers in the range represented by the bin (and this is the array that will be returned by the method).

For example, if we have 10 random numbers—say -1.33, 6.49, 5.49, -2.50, 7.50, -0.67, 3.99, 7.01, 5.24, 4.51—and 10 bins, then we can have each bin represent a range of size 1.00, with bin[0] keeping a count of random numbers in the range [-2.50, -1.50), bin[1] keeping a count of random numbers in the range [-1.50, -0.50), bin[2] keeping a count of random numbers in the range [-0.50, 0.50), and so on, all the way through bin[9] keeping a count of random numbers in the range [6.50, 7.50].

makeBins() will call the method

getBin(double[, double, double, double])

which will return the number or subscript of the bin in which a certain random number (the first double parameter) will fall. The other three double parameters might be the bin size (computed in makeBins()); the minimum random number in the ArrayList of random numbers, and the maximum random number in the ArrayList of random numbers, all of which are likely to be useful in the computation of the number or subscript. In the example above, the bin size, the same thing as the size of its range, is 1.00; the minimum random number is -2.50; and the maximum random number is 7.50.

makeBins() will also invoke the following methods:

getMin(ArrayList<Double>)
getMax(ArrayList<Double>)
getRange(ArrayList<Double>)

Metrics contains the following **static** method.

verifyDistribution(ArrayList<Double>, double, double, double)

The ArrayList<Double> parameter is, again, your ArrayList with random numbers; the first double parameter is the mean (0.0 in this case); the second double parameter is the standard deviation (1.0 in this case); and the third double parameter represents *the number of standard deviations from the mean within we which are considering data*. We want ultimately to know what percentage of numbers in our list have values between

"mean" – ("standard deviation" * "number of standard deviations away from the mean")

and

"mean" + ("standard deviation" * "number of standard deviations away from the mean").

For example, if we call verifyDistribution() with our ArrayList of random numbers and the three parameters 0.0, 1.0, and 1.0, respectively, we want to count how many numbers in our list are within 1.0 (as indicated by the third double argument) standard deviations from the mean; in other words, we want to know how many numbers in our list are between 0.0 – (1.0 * 1.0) and 0.0 + (1.0 * 1.0). If we call verifyDistribution() with the three double parameters being 0.0, 1.0, and 1.96, respectively, we want to know how many numbers in our list are between 0.0 – (1.0 * 1.96) and 0.0 + (1.0 * 1.96).

verifyDistribution() returns the *percentage* of random numbers within the specified range.

II. Output

Run the program for 11 bins and 100,000 normally distributed random numbers.

Output the count of each bin both on the console and in a file (called **Gauss.txt**). The output might look like

```
21
331
2586
10963
25513
30902
20566
7488
1505
120
5
```

Of course, you cannot expect the output to look exactly the same on any two runs (since we are after all dealing with random numbers), but if the numbers are from a normal distribution (as they are in this case), the bins near the center should have higher values (i.e., higher counts) and the bins further away from the center should have increasingly smaller values (i.e., smaller counts).

Also call the **verifyDistribution()** method from **main()** with the following three sets of double arguments and print the return values on the console:

0.0, 1.0, 1.0
0.0, 1.0, 2.0
0.0, 1.0, 3.0

III. Some guidelines

Use the **nextGaussian()** method of the **Random** class to get a normally distributed random number (from a distribution with mean 0 and standard deviation 1).

Limit the use of instance variables: pass parameters where it makes sense.

Use **final** variables where it makes sense.

Do not declare fields and methods as **static** unless it can be justified why they are static.

Do not use **double** for **Double** (even if you might be getting away with it), or vice versa.

Use the *enhanced* **for** loop wherever possible.

Modularize code.

Comment code.

Submit project exported into a zip file.