Exercises: Streams Part 3

1. Make a *very* large array of random doubles, each of which ranges from 0 to 1. A quick and easy way to do this is with “new Random().doubles(size).toArray()”.

2. Compute the sum of the square roots of the numbers in the array. Find a shorter and simpler way than making a loop to tally the sum. Hint: review the notes on number-specialized streams, especially the fact that you make a DoubleStream from a double[] with DoubleStream.of, not Stream.of.

3. Repeat the process in parallel. Once you have #2 working, this should be *very* simple.

4. Verify that you get the “same” answer with the parallel approach as with the sequential approach. Why do I have “same” in quotes in the previous sentence?

5. Test whether the parallel approach is faster than the sequential approach. Doing the timing is a little bit tedious, but if you think it simplifies things, you can steal the Op interface from streams-3-exercises project, then do something like this:

```
Op.timeOp(() -> {
  double sum = MathUtils.sqrtSumParallel(nums);
  System.out.printf("  Sum is %,.8f.%n", sum);
});
```

6. Make an “infinite” stream that generates random doubles between 0 and 10. Use it to

- Print 5 random doubles
- Make a List of 10 random doubles
- Make an array of 20 random doubles

Note: in general, if you are dealing with numbers, DoubleStream is preferred over Stream<Double> because DoubleStream uses primitives and has more convenient methods (e.g., min, max, sum, average). In this case, however, use Stream<Double> because it is hard to turn a DoubleStream into a List and because it is hard to print a double[] but easy to print a Double[] (e.g., pass the array to Arrays.asList and print the resultant List). So, for this part of the exercises, use Stream.generate, not DoubleStream.generate.