Exercises: Lambda-Related Methods in Lists and Maps

1. Make a static method that, given a size, will produce a List<Double> of that size, each value of which is a random number between 0 and 1. Use Math.random() and normal List methods.

2. Make a static method that, given a List<Double>, will output the average. Use normal List methods. (You can also use Stream methods if you can figure out how to turn a List<Double> into a DoubleStream.)

3. Make a static method that, given a List<Double> and a cutoff, will modify the List so that all numbers below the cutoff are removed. Use one of the methods from this lecture.

4. Make a large list containing random numbers between 0 and 1. Compute the average and verify that it is near to 0.5. Remove the numbers less than 0.5. Compute the average again and verify that it is near to 0.75.

5. Make a static method that, given a List<Double>, will modify the List by doubling (multiplying by 2) all the values. Use one of the methods from this lecture.

6. Make a large list containing random numbers between 0 and 1. Compute the average and verify that it is near to 0.5. Double all the values. Compute the average again and verify that it is near to 1.0.

7. Copy the Primes and RandomUtils classes from the lambas-in-collections project into your project. Give the Primes class, the following is a recursive static method that, given a number, will count how many primes are less than or equal to that number.

   ```
   public static int countPrimes1(int upperBound) {
     if (upperBound <= 2) {
       return(1);
     }
     if (Primes.isPrime(upperBound)) {
       return(1 + countPrimes1(upperBound - 1));
     } else {
       return(countPrimes1(upperBound - 1));
     }
   }
   ```

   You can either type in this method yourself, or copy it from the Utils class in the lambdas-in-collections-exercises project. (If you do this, don't peak at the other methods!)

8. Profile this method. You will notice that it is moderately slow (0.05 seconds or so) to find the number of primes up through 5,000, and you will also notice that you will get a stack overflow when counting the number of primes around 10,000.

9. Use computeIfAbsent to make a memoized version of this method. Profile it and compare performance.