Topics to Review

- Arrays

1. Updating $FlipStats$ class.

We want to analyze a series of random coin flips represented by an array of integers, 0 for heads, 1 for tails. Specifically, we want to identify a "**run**", which is a consecutive sequence of the same result for the coin flip. Someone wrote a $FlipStats$ class with the following methods:

- A non-default constructor that performs a deep copy from its single argument, an array of integers (more specifically, 0's and 1's). You can assume the argument is correct.
- A $firstRun(\text{int side},\ \text{int length})$ method that takes 2 arguments, an integer denoting what side we are looking for, and an integer denoting what exact length run of that side we are looking for. For example, $firstRun(0,\ 3)$ is looking for the first run of 0's of length exactly 3 in the corresponding $FlipStats$ object.
  The method should return the index position that the first run of that exact length starts or return -1 if not found.

```
public class FlipStats {
   private int [] data;
   public FlipStats(int [] d) {
      data = new int[d.length];
      for (int i=0;i<d.length;i++) {  // deep copy of array
         data[i]=d[i];
      }
   }
   public int firstRun(int side, int length) {
      int currentRun=0, i=0;
      boolean found=false;
      while (!found && i<data.length) {
         if (data[i] == side) {
            currentRun++;
         }
         else {
            currentRun=0;
         }
         if ( ((i+1)==data.length || data[i+1]!=side) && currentRun==length) {
            found=true;
         }
         i++;
      }
      if (found) {
         return i-length;
      }
      else {
         return -1;
      }
   }
}
```

Sample Testing:

```
//                  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
int [] flipData = {0,1,1,1,1,0,0,1,1,1,0,1,0,1,0,0};
FlipStats data = new FlipStats(flipData);
System.out.println(data.firstRun(0, 2));  // prints 5
System.out.println(data.firstRun(1, 3));  // prints 7
System.out.println(data.firstRun(0, 3));  // prints -1
System.out.println(data.firstRun(1, 1));  // prints 11
```

What if we want to expand the functionality of our $FlipStats$ class to answer the questions like the following?

- What is the longest run?
- What is the longest run of heads? Or tails?
- What is the average run length?
- What is the average run length of heads? Or tails?

We could write methods for each of these questions individually by scanning the $data$ array. Or, once the original data (the array of 0's and 1's) is stored, we could start "pre-process" the data and identify all the runs; and we could store these runs in another array for run tracking. In this way, we can use a shorter time to answer each of the above questions (since we only need to read and process the input array once, and then we only need use the "run track" array, which is shorter and has more information about runs).

For example, if our input array is this:

$$0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0$$

We could process it once and create this $runTrack$ array which denotes head runs ($0'$s) with a negative number for the length of the run. And denotes tail runs ($1'$s) with a positive number for the length of the run.

$$-1 ,1 , -3 ,2, -1 ,1 , -1 ,2 , -3$$

Then process that $runTrack$ array to answer these questions:

- Index location of first run of heads of length 2?

- Index location of first run of tails of length 2?

- What is the longest run?

- What is the longest run of tails?

- What is the average run length?

- What is the average run length of heads?

Please check $FlipStats.java$. Here, I updated the attributes and constructor and wrote the private method "$createRunRunTrack$" which walks the "data" array, keeping track of runs, and loads the length of each run into the "$runTrack$" private array attribute as discussed above.

Let's create the following methods in the $FlipStats$ class:

- Rewrite the "$\text{public int } firstRun(\text{int side}, \text{int length})$" method that processes the "$runTrack$" array instead of the "$data$" array.
- Write a new method "$\text{public int } longestRun(\text{int side})$" that return the length of the longest run of that side.
- Write a new method "$\text{public int } longestRun()$ " that return the length of the longest run of either side.
- Write a new method "$\text{public double } averageRun(\text{int side})$" that return the average length of the runs of that side. If no runs of that side, return 0.