DELIVERABLE #4
PROPERTY-BASED TESTING

Joseph David Bender

jdb117@pitt.edu

INFSCI 2545 SOFTWARE QUALITY ASSURANCE

Why I Chose This Project:

I chose this project, and the properties, because they are fairly universal properties that could be applied to multiple situations. Mapping functions are fairly common in object oriented programming. There are many methods that take in an argument, modify it, and return it. My first test of the lengths of the arrays is simple confirmation that the sum of the array was appended to the new returned array. It is a basic way to test that the routine did one of its functions. More confirmation that the sum was appended successfully is that it is the largest element of the new array, which is what my second property test checks. Finally, purity in programming is vital. Having reliable methods and functions that will perform the same action regardless of the data is critical to consistency. My third test ensures the that the billify() method is pure. You could pass it the same array 1000 times and it would return the same squared array, with the sum appended, every single time. Each of my tests were premeditated to be functional and relevant to the program.

How I Went About Completing it:

I first considered the data in the program before the tests. Most importantly was creating 100 arrays, of random length, with random data. I figured a multidimensional array would be the most efficient way of storing the data. The assignment called for a minimum of 100 arrays being passed to billify(), so I initialized a 2D array of 100 elements. Each of these elements was an array itself. I then initialized the hundred arrays, using a random number generator, to be of random size from 1-100 inclusively. Since the assignment states that billify() should only take in a one dimensional array, I had to pass them one by one into the method to be modified and returned. Once I had a reliable generation and modification of data, I translated the routine into the three test cases. From there, it was a simple tweaking to reach the assertions. The first test just compared the length of the original array to the returned array. The second test compared the final element in the returned array (the sum) to the rest of the array. The final test executed billify() twice on the same array and compared the results to ensure they were identical. All three tests pass perfectly, and prove the billify() method is consistent and reliable!

Three Properties Tested:

1. Property #1: Array Length
    a. The first test ensures that the array returned from billify() is one element longer than the array that is passed in as an argument. This is due to the fact that the billify() method squares all elements in the array, but then adds one more index in order to append the sum of all the squares to the end of the array. Therefore, if the array passed to billify() has length n, every array returned should have length n+1. This test checks this property for all one hundred random arrays and will fail if the returned array is not exactly one element longer than the original.
2. Property #2: Sum Check
    a. The second test ensures that the last element in the array returned from billify() is greater than or equal to all other elements. This is due to the fact that the billify() method squares all elements in the array, but then adds one more index that is the sum of all previous elements in the array. Since it is the sum of all elements in the array, it should always be the greatest element in the array. The one occasion it would not be is

when it is a one element array. In this instance, the sum would be equal to the single element in the array. This test is prepared for that and will still return true, due to the fact that there are still no elements greater than the sum.

3. Property #3: Purity Test
    a. The term purity in property testing means that a certain routine, when passed the identical data, will always produce the same result. In this occurrence, it would mean that if the billify() method is passed the same array twice, the returning array would be the same. The billify() method squares all the elements, and appends the sum of all the new squares. It is a simple mathematical procedure, which should not fluctuate if the same array is passed as an argument multiple times. This test passes each of the random arrays twice to make sure that the same array is returned both times.

Issues Faced:

One of the issues was appending the sum of the squares to the array in billify(). At first I was using Arraylists so I could easily change the length of the array. However, I found those harder to work with and a static length was a property I wanted to work with for assertions. Therefore, when I needed to append the sum I just created a new array one size greater than the original.

Another issue was that billify() only accepted a one dimensional array and I was using a 2D array to store all the data. I had to store each array in the 2D array in its own one dimensional array, pass it to billify(), then re-store the returned array in a new 2D array.

It is also complicated sometimes to keep track of the lengths of the arrays, and spans of the random number generator. It could be easy to have an “off by one” error if you are not diligent about the generation of arrays or random numbers. I used a lot of print statements in development to provide feedback and ensure I was getting the data I wanted.

Testing Failures:

None of the tests failed, all were completed successfully.

Github Repository Link:

https://github.com/jdbender66/arrayTest

Screenshot of Completed Tests: