## 

## 

## 

## 

## 

## 

## 

## Richard Kotermanski

## CS 1632 - DELIVERABLE 4

## PROPERTY-BASED TESTING

## 

## 

## 

## 

## 

## 

## 

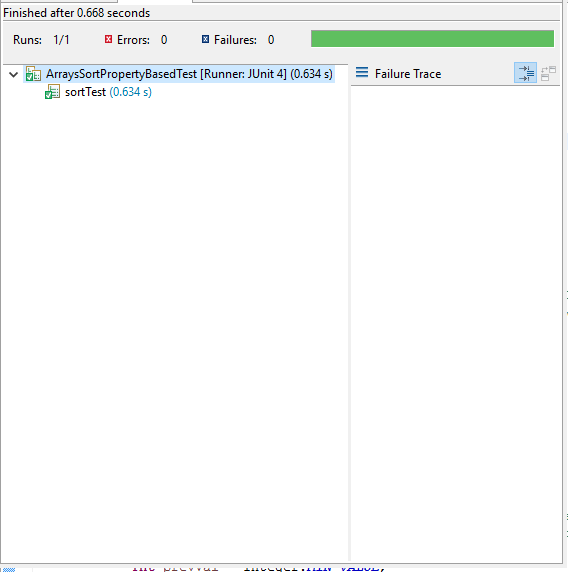
## Summary

I chose the JUnit-based property-based tests for this deliverable. Since I understood the combinatorial testing better than property-based testing, choosing the property-based tests route seemed like a better option. It not only allowed for me to spend more time looking through the notes and reading the book, which I would have done regardless, but I also had a chance to practice the concept through this project if chosen. Furthermore, since we already received and followed along with a detailed example in class for combinatorial testing, I felt that I needed a way to match that with property-based testing.

Property-based testing involves checking the expected properties of a behavior against the observed properties of the behavior. In this case, it was necessary at each iteration to generate an array of random length that each were comprised of also randomly generated integer values. These arrays each served as input into the Arrays.sort() function, and the random generation is part of the abstraction necessary to make property-based testing effective. Each of these arrays are considered input for the Arrays.sort() function, and since the function does not clone the array, the array at the same reference is considered the output. The properties then tested for each input include the array length size, proper sort order of the output array, and pureness of the function. By recording the original, randomly generated length of the input array, it was then possible to test that with the output array’s length was the same. Testing this ensures that no values were added or removed due to the sort function. Next, it was possible to test if the output had the proper sort order by comparing each neighbors’ values. This comparison allowed for the test to ensure that each increasing index’s value of the output array was greater than or equal to the immediately previous index’s value. Finally, to test if the function was pure, it was necessary to generate a second array along with the original test input array that is identical in values and order. To ensure that the function upheld its pure properties, the output array from the original input to the second output array that used the second input array to check if the values and their order are still identical after sorting.   
 I did experience some minor issues that were mostly due to the limitations and properties of Java, the sort function, and my knowledge. Since the Arrays.sort() function operates on the input array without copying it, I had to create a second, identical array each time, as discussed above, to test whether or not the Arrays.sort() function was pure for that input. Another issue faced during this project was due to Java’s limitation on maximum array size. Due to memory limitations and variations between machines, I had to find a reasonable range for my random array generation. Since the maximum value of an integer was much too large for both memory as well time sensitivity due to both the costs of generating the arrays as well as sorting them both. This meant that I had to find a good balance between testing arrays of larger length and test feasibility. I ended up choosing a ranging in size of 0 to 100,000.

From this deliverable, I learned that property-based testing is an effective method that leverages abstraction for variable input scenarios to detect input sets that cause defects.

## Screenshot



### Github Location:

<https://github.com/rickykoter/CS1632_Deliverable4>

* The source for the project is contained within the 'src' directory and is named 'ArraysSortPropertyBasedTest.java'.