|  |
| --- |
| PROPERTY-BASED TESTING |
| CS 1632 — DELIVERABLE 4 |

|  |
| --- |
| Joe Meszar (jwm54@pitt.edu)  3/24/2016 |

# INTRODUCTION

I chose the JUnit property testing project over the NIST ACTS combinatorial testing project simply because I was more familiar with it after the last deliverable. It’s a fairly straight-forward process to perform JUnit testing; decide what you wish to test, and then create assertions about what you expect.

The project revolved around testing the Java Arrays.sort() method. To do this, I used the special JUnit method setUpClass to create a 2D array of randomly-sized and randomly-valued arrays. Each array size was randomly chosen between zero and 666 (with the very first array forced to be zero-sized just for testing purposes). Furthermore, each array index was populated with an integer value between -32768 and 32767.

Once all of the arrays were created and populated, I then was able to start testing the Arrays.sort() method. The first thing to do was copy the 2D array so that it could be compared against itself after being sorted. This was done using the built-in clone() method. Before using the cloned array, I first made assertions to make sure the clone equaled the original and also contained the same hash value. Afterwards, using the cloned 2D array, I performed an ascending sort on each and every array held within it. During each iteration, I tested to make sure the size of the newly-sorted array was the same as the old size—and I also tested to ensure the hash values were the same. Once every array was sorted in ascending manner, a test was performed to make sure each array value was **correctly** sorted in ascending manner by testing each consecutive value against the previous value to make sure the current value was greater than or equal to the previous value. This entire process was done again while sorting in a descending manner (and instead testing that the current value was less than or equal to the previous value).

The only issues I faced dealt with the built-in methods for int and Integer data types. I originally tried to test using the int data type, but found it did not have the sorting methods for the descending sort—a function of the Collections class which does not handle primitive data types. After converting to the Integer data type, everything else was fairly simple to code.

If there is anything to take away from this exercise, it would be the trust that all programmers have in the built-in Java classes and methods. We use these on a daily basis when coding our projects, but never give thought to the optimization and reliability of these methods. It’s actually pretty cool to test these “given” methods just to get an idea of how quickly it allows us to move beyond custom algorithms and data types and instead create a common model for all programmers to build their projects upon.

# JUNIT EXECUTION RESULTS



