CS 1632 - DELIVERABLE 4

Members: Christian Boni

Project: Property-Based Testing

Summary

Out of the two options of property-based testing and combinatorial testing I chose the former for this deliverable. This was actually a tough decision as the lecture on combinatorial testing was quite interesting. It was astonishing to learn that combinations of one or two interactions tend to result in 50% to 90% of the defects in most software systems. On the other hand, combinatorial testing continues to test the expected behavior against the observed behavior as with the previous deliverables in this course. I liked the new ideas behind property-based testing in that it transitions to a higher level of abstraction. Now the test cases only test the expected properties of the behavior and the observed properties of the behavior. By testing the properties of the behavior we can now let the computer do the tedious work of generating all of the specific tests. Additionally, if I were to chose combinatorial testing I would have to create a manual test plan. While it was nice to gain experience writing manual test plans in the first deliverable, I found them to be rather dull. In contrast, I really enjoy writing automated test cases using jUnit. I find it to be quite rewarding to be able to execute test cases at the touch of a button and to have them complete in seconds.

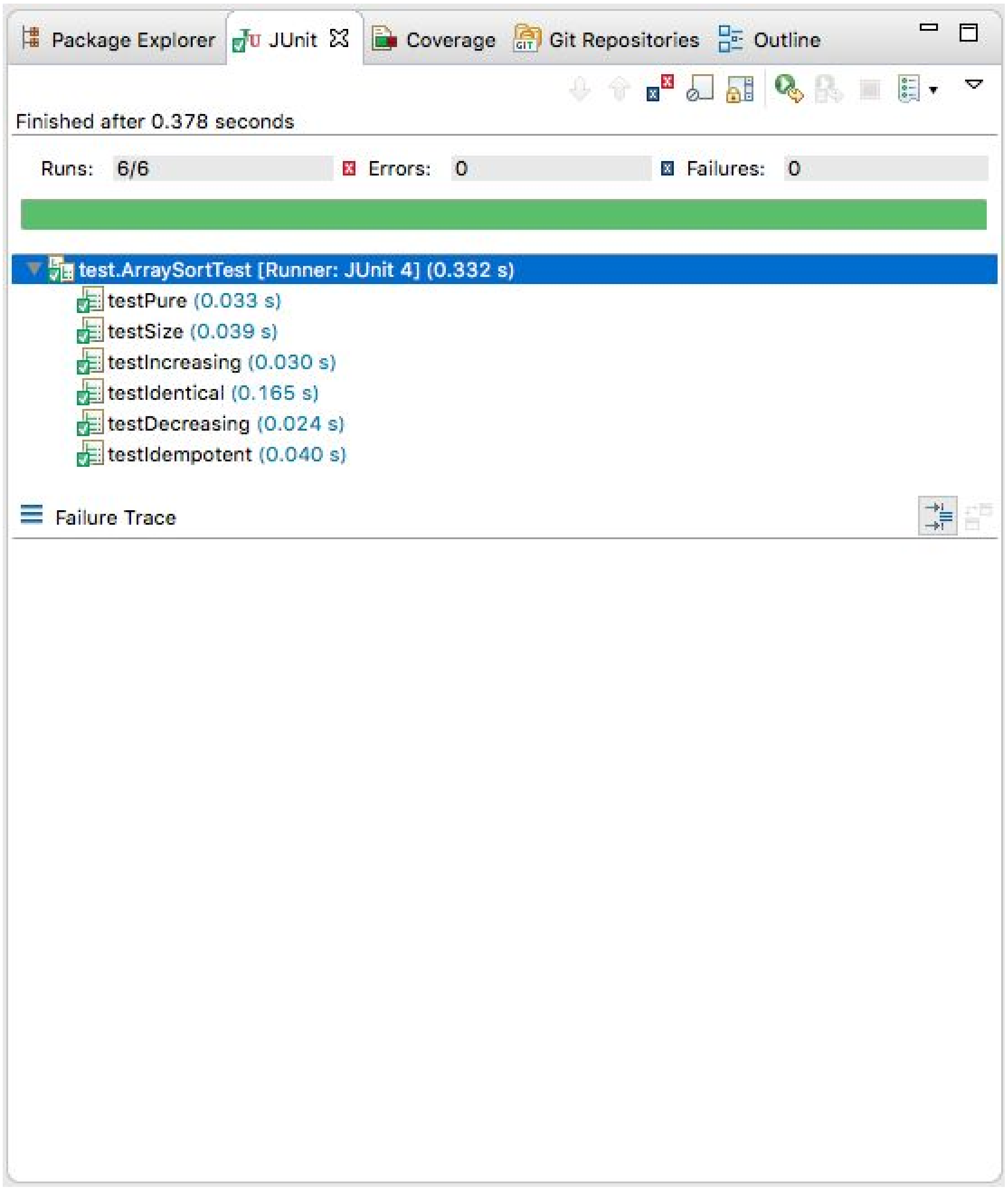
This deliverable states that Java's built-in Arrays.sort() method shall be tested using a minimum of three property-based tests. After reviewing the notes for property-based testing I came up with six different properties for testing the method. The first property proclaims the size of the array to be immutable, meaning the sorted output array should have the same exact size as the original input array. The second and third properties require the array to be sorted in ascending ordering. As you move forward in the array the elements shall always be in increasing or staying the same. Similarly, as you move backward in the array the elements shall always be in decreasing or staying the same. The fourth property states that every element in the original input array shall be present in the sorted output array and that no additional elements have been added to the sorted output array. The fifth and sixth properties require the method to be idempotent and pure, respectively. Where idempotent means that running the method a second time on the sorted output array yields the same sorted output array. Lastly, pure means that running the method twice on the original input array yields the exact same sorted output array both times.

Throughout the course of this deliverable I did not run into many issues. Most of the property-based test cases were rather straight-forward. The most complex of all the test cases was testing if two arrays were identical. From past experience, I learned the most efficient way to implement this for numerical arrays is to sort the arrays first and then to compare the two arrays. Unfortunately, this conflicts with the entire test plan since its sole purpose is testing the sort method in the first place. As a result, I had to come up with a less efficient way to test if the two arrays were identical. After writing and executing the test cases for this project I learned that property-based testing is a rather efficient and effective way to test. I only wrote a mere six property-based tests but I was able to test a massive amount of random arrays each of their own random size. By having the computer do all the heavy-lifting through generating specific test cases, property-based testing allows you to focus more on testing the overall functionality.

Source Code

https://github.com/thisbechristian/quality-assurance/tree/master/deliverables/d4

Unit Tests



```
Finished after 0.378 seconds

Runs: 6/6    Errors: 0    Failures: 0

test.ArraySortTest [Runner: JUnit 4] (0.332 s)
    testPure (0.033 s)
    testSize (0.039 s)
    testIncreasing (0.030 s)
    testIdentical (0.165 s)
    testDecreasing (0.024 s)
    testIdempotent (0.040 s)
```