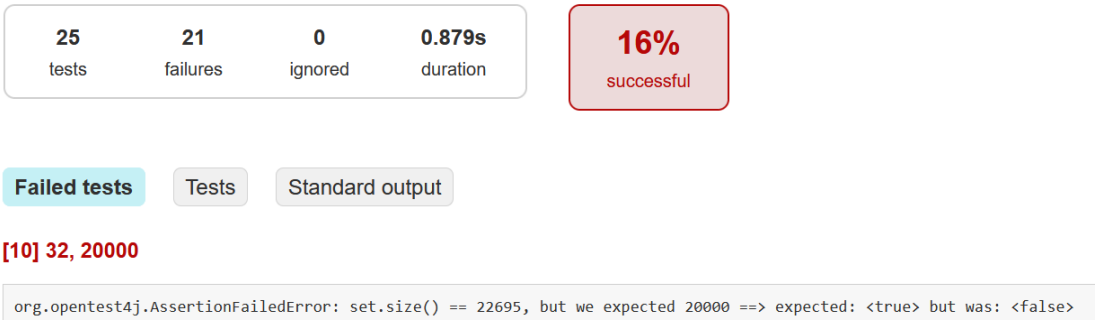
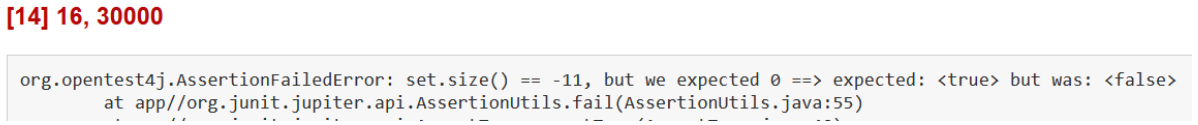
Exercise 4.1

1. See ConcurrentSetTest.java for implementation and testing.
   One of the interleavings that the given class does not work is when one thread is performing the addition of a number and another is trying to add the same one. Since add is not atomic if the second passes the check to see if the element is already present before the other has actually added it, it will be added by both.
   In the end we get an array whose size is bigger than it should due to the presence of repeated elements.



2. See ConcurrentSetTest.java for implementation and testing.
   One of the interleavings that the given class does not work is when one thread is performing the removal of a number and another is trying to do the same. Since remove is not atomic if the second passes the check to see if the element is still present before the other has actually deleted it, it will be deleted by both.
   In the end we get an array whose size is different from 0.



3. See ConcurrentIntegerTest.java for implementation.
   We add the synchronized signature to the two methods add and remove. In this way we ensure that no thread can verify the add or remove condition before another thread has successfully made the addition or removal.

4. As expected we did not find any error, that increases our confidence in saying that ConcurrentIntegerSetLibrary is thread safe

5. Yes, a failed test (if the test is well written) is an indicator of the presence of an error in the class.

6. Not passing the test does not prove correctness even if we test all the methods of the class. That is because testing can only prove the presence of errors, not its absence.

Exercise 4.2

1. One of the interleaving that the given property does not work with capacity=1
   main(release)-t1(acquire)-t2(acquire)-t1(release)-t2(release)
   here we can see that 2 threads are in the critical section at the same time, so the property is violated. We think that the problem is given by the fact that there is no happens-before relationship between acquire() and release(), you can release without acquiring nothing first.

2. Our test for sure triggers the interleaving because at first we use the main to release the semaphore and then we start the two threads that during their run will acquire the semaphore, wait for the other thread and then release it.

Exercise 4.3

1. See ReaderWriter2.java for implementation

2. See ReaderWriter2Test.java for testing