**Task 2 – Getting Started - Fibonacci.**

***For this task, briefly describe what the error was and how you corrected it.***

The switch statement returned the wrong value of one (1) for the condition where *n* = 0. I changed the return value to zero (0) and ran the JUnit test, which resulted in no failures. Just for fun, I added a tearDown() method, separated the test cases into separate test methods, and modified the expected values to examine behavior. I noticed that only the first encountered failure is reported when multiple test cases are grouped into one test method. I then regrouped the test cases back into a single test method and ran the JUnit test, which resulted in no failures.

**Task 3 – A Little More Advanced - Rectangle.**

***For this task, briefly describe what the error was and how you corrected it. Also, provide the source code for your improved getDiagonal() and getArea() methods.***

When I ran the JUnit tests, both tests failed, but it was difficult to determine the source of the failure. After examining the code, I noticed in the constructor of the *Point* class that the class variable *x* was assigned the value of *y*. I changed the right side of the assignment to *x*, and ran the JUnit test, which resulted in no failures.

I added statements to each of the methods to print out labels to examine behavior. I noticed that the *rect1* and *rect2* objects were being created and destroyed before and after each test, which is unnecessary since the tests use the same objects. I implemented class-level fixtures to instantiate objects once before any test is executed and destroyed once after all tests are executed. I then changed expected values of the test cases to make sure the test ran as expected. I changed the expected values back to the original values and rand the JUnit test, which resulted in no failures.

To improve the code, I reduced the duplicated code by adding class variables in the *Rectangle* class to store the *x* length and *y* length of the two *Point* objects. I then added statements to calculate and assign these lengths in the constructor and updated the *getArea*() and *getDiagonal*() methods to reduce complexity. I ran the JUnit tests, which resulted in no failures.

To make the bug easier to spot, I added getter and setter methods for the *Point* class variables of the *Rectangle* class. I added assertions in the *RectangleTest* class methods to check the *x* and *y* values of each *Point* object. This will allow any assignment error in the *Point* class to be easily identified. I reinserted the assignment error and was able to directly identify that it was caused by the assignment statement in the constructor of the *Point* object. I resolved the error and ran the JUnit test again, which resulted in no failures.

**Task 4 – On Your Own – A Vending Machine.**

***For this task, briefly describe any bugs that you found. You should also upload your Vending Machine project to your GitHub account.***

The *VendingMachineItemTest* class constructor allows an empty string name to be input. I added code to throw an exception when attempting to create an item with an empty name string.

**Task 5 – Summing it All Up.**

* A description (2-3 paragraphs) of what you learned from this project (particularly Task 4)
* A description (2-3 paragraphs) of what you liked and didn’t like about JUnit’s support for unit testing

While coding the VendingMachineItemTest class I attempted to put assertions into separate test methods for each different type of input to test the constructor. I kept getting a null pointer exception in the testGetName and testGetPrice methods. I inserted label outputs in each method to examine behavior and noticed that the testGetName and testGetPrice methods were being executed before any of the methods that tested the constructor. I then consolidated the test cases for the constructor into the setUp method to ensure that these test cases executed before any of the other test methods, because the objects created were being used by the other test methods. After making the change, I ran the JUnit test, which resulted in no failures.

I then examined the Rectangle JUnit test results, where I also had function labels printed out to console to check behavior. I noticed that the test methods with similar name signatures were executed in lexical order. I then reorganized the class variables to leverage this behavior and was able to reduce complexity.

It is helpful to reuse instances of an object for different types of tests of related tests methods. It is also helpful to write out the different types of tests you want to perform. In this project, the test cases were mostly different combinations of input scenarios. To make sure you cover sufficient combinations, you need to determine the tests you want to perform and list them out (in a test plan). Writing the test cases out makes it much easier to keep track of the tests being considered and to remove unnecessary tests.

I like the fixtures and the test method execution behavior of JUnit. The fixtures at the class-level and test method level allow you to reduce the complexity of unit test code. Also the sequential execution of similarly named test methods allow you to reuse test objects and further reduce complexity.