***Task 2 - For this task, briefly describe what the error was and how you corrected it.***

This was a simple fix, in the Fibonacci.java there was a statement that said case 0 return 1. The test case only showed that there was an error in the Fibonacci function. Looking at the code, the return should had been a 0 and not a 1. Simple fix, but it would had been faster if each testFibonacci had its own test case.

|  |  |
| --- | --- |
| error | correct |
| Case 0: return 1; | Case 0: return 0; |

***Task 3 - 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.***

Finding this error was a little more difficult than task 2 because it was in a class that was not being tested. Again each test case was testing two things, which is bad practice and made finding the error a little more difficult. Looking through the code in getDiagonal() and getArea() the math looked right so I went to Point.java. I found the error here. It was saying the this.x = y, which is not logically correct. If there was a test case or a getx/gety to test the actual value for x and y this would had been easier to find.

My modified code went beyond just getDiagonal() and getArea(). I actually created two new variables, change the constructor and creating two new functions.

|  |  |
| --- | --- |
| **private** Double height, length; | These variables were added to the class |
| calculateHeight();  calculateLength(); | These calls were added to the constructor |
| **public** Double getArea() {  **return** ( height \* length );  } | This is my getArea function |
| **public** Double getDiagonal() {  /\*  \* Pythagorean Theorem  \* a^2 + b^2 = c^2  \* c = sqrt(a^2 + b^2)  \* a is the height  \* b is the length  \*  \*/  **return** (Math.*sqrt*(Math.*pow*(height, 2) + Math.*pow*(length, 2)));  } | This is my getDiagonal() function. It is still a little messy but it is cleaner. I could have added  a2 = height \* height or math.pow(height,2);  b2 = length \* length or math.pow(length,2);  return math.sqrt(a2 + b2);  but I think that return is clean. |
| **public** **void** calculateHeight(){  height = Math.*abs*(p2.y - p1.y);  } | Simple calculate the height |
| **public** **void** calculateLength(){  length = Math.*abs*(p2.x - p1.x);  } | Calculate the length |

I felt like the length and height of a rectangle is very common in many formulas dealing with them. So having to compute the height and length each time, regardless how easy it is, was pointless. So once time call in the constructor and we are done.

***Task 4 -*** ***For this task, briefly describe any bugs that you found. You should also upload your Vending Machine project to your GitHub account.***

I couldn’t find any bug. This is disappointing to me because I feel like I’m a pretty good programmer. There are things that could be done for maintainability but there are no logic bugs in this program as far as I can tell.

***Task 5 – Summing it All Up.***

**It All Adds Up Now!!**

I learned a few things form this project the first thing I learned was how to create JUnits. The second thing I learned having 100% coverage does not mean I will find all the bugs. I’m still not sure on a few things as to why I test them because I’ll just be testing them somewhere else. For example insertMoney and getBalance, I have to test getBalance in my insertMoney. After testing the exceptions, I have to use getBalance after inserting money to see if it was right. So for a good case in both insertMoney() and getBalance() my code was exactly the same. This could be a clue to where a bug could be but I was not able to find it. Testing exceptions were interesting, I had to throw and catch in the test case and then compare the exception. I also learnt that there were many different ways to do the same thing assertNull(object x) and assertEqual(null, object x),for example, or so I hope. Like any other part of coding there is always more than one way of doing something.

The second thing was evening though having 100% coverage, finding bugs can still be difficult. I tried to mix and match things (adding, removing, in different orders) but I still could not find the stupid bugs in VendingMachine.java. I also learnt that even though I could be testing in one class, it could point to a bug in another class. Like task 3.

***O JUnit, JUnit, wherefore art thou JUnit?***

There are a few things I like about JUnit, a few things that I did not like and a few that were a double edge sword. I liked that JUnit does tests for me now so I do not necessary have to add print out statements everywhere. I like that I can make an exception on a function and throw it and then catch it to verify that the try/catch block are working in the code. If I ever have a situation where I need to catch a divide by 0, I think it would be able to test that.

Something I did not like about JUnit is I have to make a million and one tests. For example, in the function MakePurchase, I had to test balance in one case and then I have to test the return (true) in another. I wish I could just test once and it would tell me if return true was wrong or the balance was incorrect.

The setup() function was a double edge sword. It is very useful to a point. I did not have to create a candy bars over and over again which was good but I also wish I could have different setups functions. There were a few test where I called the same thing in multiple test but I was not ready for in other test. Like insertMoney. I needed that function in later test but I wanted to make sure it worked when I was actually doing the insertMoney test.