Obed Dominguez

AIT 642 / COSC 603

Software Testing and Maintenance

Spring 2016

Project 3 Tasks Write Ups

**Task 2 – Getting Started - Fibonacci**

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

The error that was created in this task was a number “0” was expected to be returned, but it was returning a number “1.” To correct the error, I looked at the Fibonacci class and the method that it had. In the int fibonacci method there was a “case 0” that returned “1.” To fix the error that I was getting in the JUnit test, I changed in the “case 0” the return value “1” to a value of “0.” That change fixed the error in the JUnit test and turned the red bar into a green bar.

**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.***

The error in this task was in the point class inside the Double x, y method. The “this.x = y;” was where the code was erroneous. Instead, it should have been “this.x = x;”, so that is how I corrected the error. For the source code for getDiagonal and getArea methods, I did the following: First, I refactored the “x and y” coordinates to instead represent “length and width.” This was to make the code less confusing when looking at it. Second, I changed the code like this:

For the getArea I changed it to:

**double** length = p2.length - p1.length;

**double** width = p2.width - p1.width;

**double** area = length \* width;

**return** Math.*abs*(area);

For the getDiagonal I changed it to:

**double** length = Math.*pow*((p2.length - p1.length), 2);

**double** width = Math.*pow*((p2.width - p1.width), 2);

**return** Math.*sqrt*(length + width);

This is essentially doing the same function, but it's broken down to a less complex math logic.

**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 GitHub account.***

For this task, the bugs that I found in the code were in the VendingMachine class. Assuming that my JUnit test was correct, I had to make changes to the VendingMachine class by changing two methods to return a value. The methods that I added a return value were the addItem and removeItem methods.

**Task 5 – Summing it All Up**

* **A description (2-3 paragraphs) of what you learned from this project (particularly Task 4)**

What I learned from this project was how to setup and run simple JUnit testing using the JUnit framework. This simple JUnit testing was mainly done at the Fibonacci and Rectangle projects. The two projects were simple enough to follow the logic and see how a JUnit test could fail if the test was done incorrectly. These two first projects were designed to show how JUnit testing is supposed to work, but I don’t think it was enough to learn how to write more complex JUnit testing cases.

The task 4 or the VendingMachine project was difficult for me while trying to write my JUnit tests. Following the idea from the previous project was not enough to start writing tests for this task. I searched for additional resources to try and get a grasp on what I need to learn to write the tests, but I only found tutorials that showed you how to write simple JUnit tests. I felt that for task 4 the tests were more complex, and that is why I had a hard time writing them. I was able to complete task 4, but I’m still unsure if my tests are even 100% correct.

* **A description (2-3 paragraphs) of what you liked and didn’t like about JUnit’s support for unit testing**

Based on my experience mainly from task 4, there were things that I did like about the JUnit’s support and things that I did not like. For example, the one thing that I liked was that when writing the tests the framework gave suggestions while writing the tests. If something was not making sense, the framework would underline a word, and you could hover and click on the word to get suggestions for a fix. That feature was nice to have in order to make the test work properly. Another nice feature of the framework is the JUnit test suite and how it puts together the JUnit test cases in a one suite test case.

The not so good support from the framework for me was trying to follow the trace route when a test failed. Some of the information was not descriptive enough, or I did not know what certain errors meant. Not knowing some of those errors made for a frustrating time of trying to find out where the error in the code might be. Even after consulting resources like Stack Overflow and the like, it was hard to figure out some of those errors.