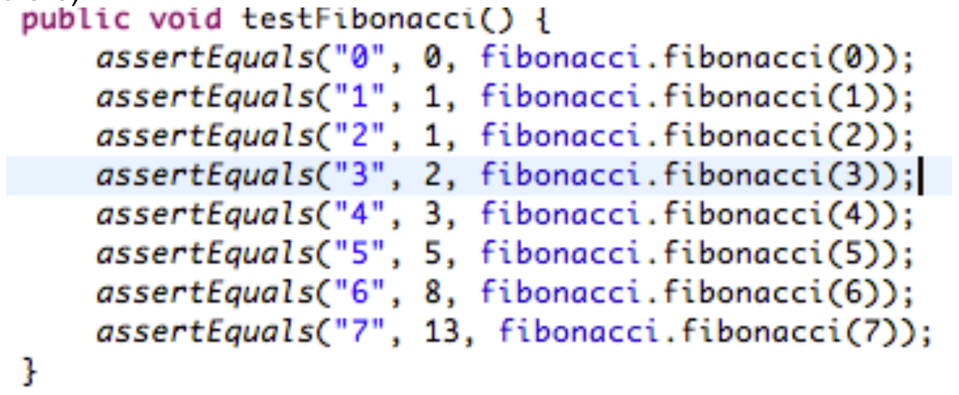
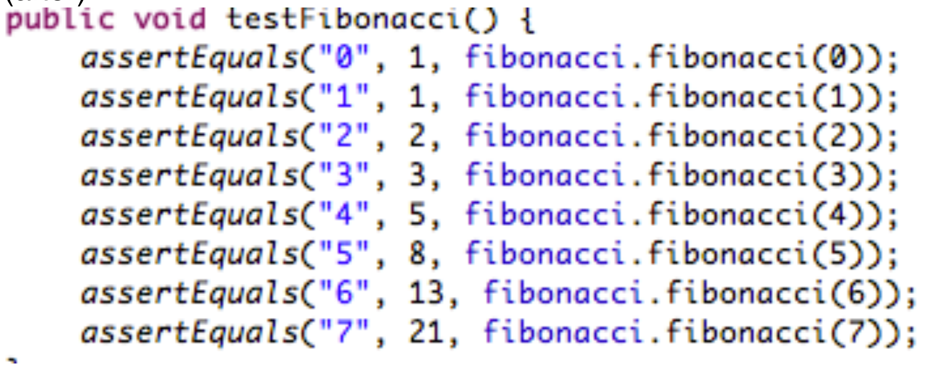
Task 2

(before)



```
public void testFibonacci() {
    assertEquals("0", 0, fibonacci.fibonacci(0));
    assertEquals("1", 1, fibonacci.fibonacci(1));
    assertEquals("2", 1, fibonacci.fibonacci(2));
    assertEquals("3", 2, fibonacci.fibonacci(3));
    assertEquals("4", 3, fibonacci.fibonacci(4));
    assertEquals("5", 5, fibonacci.fibonacci(5));
    assertEquals("6", 8, fibonacci.fibonacci(6));
    assertEquals("7", 13, fibonacci.fibonacci(7));
}
```

(after)



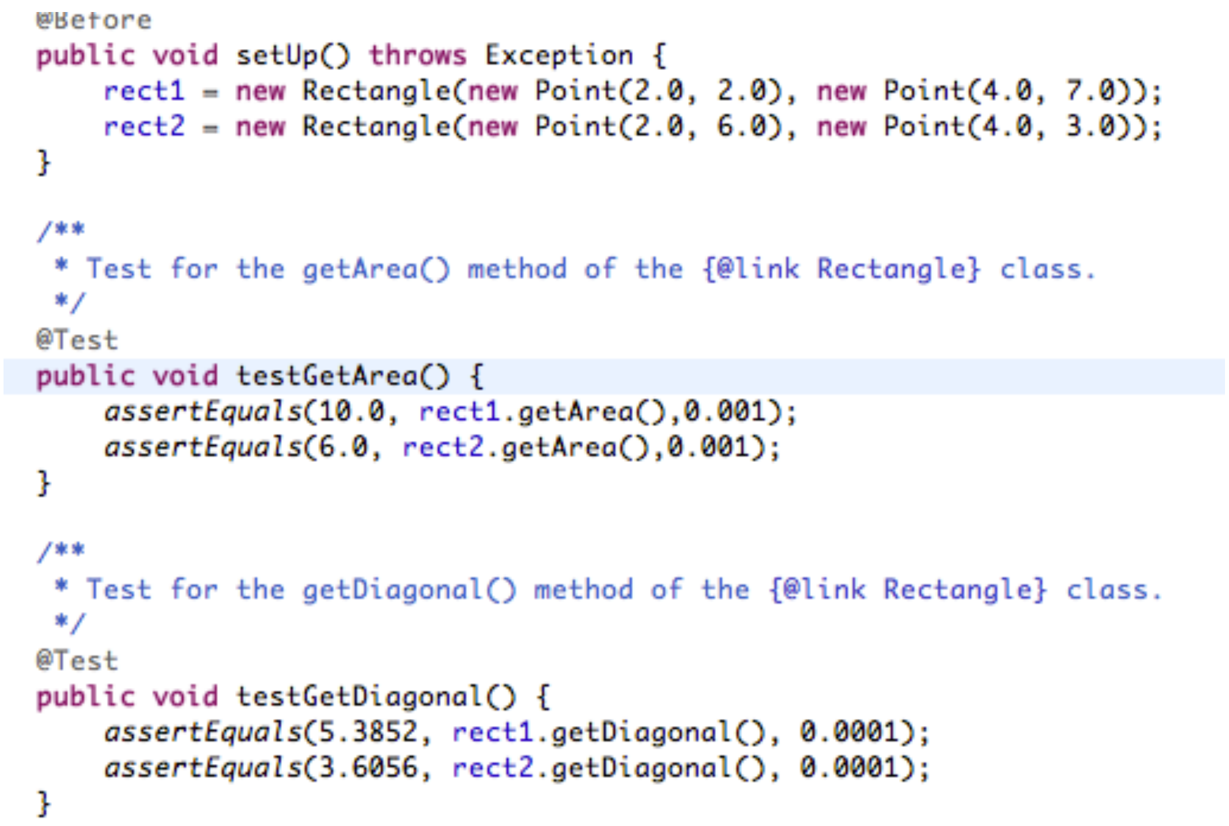
```
public void testFibonacci() {
    assertEquals("0", 1, fibonacci.fibonacci(0));
    assertEquals("1", 1, fibonacci.fibonacci(1));
    assertEquals("2", 2, fibonacci.fibonacci(2));
    assertEquals("3", 3, fibonacci.fibonacci(3));
    assertEquals("4", 5, fibonacci.fibonacci(4));
    assertEquals("5", 8, fibonacci.fibonacci(5));
    assertEquals("6", 13, fibonacci.fibonacci(6));
    assertEquals("7", 21, fibonacci.fibonacci(7));
```

Initially when you run the test for Fibonacci method, it showed failed test case. It was expecting zero value when it parsing zero instead of returning one . I shifted expected outcome for the rest of values one more up and calculated the fibonacci for 7, which was 21.

Task 3

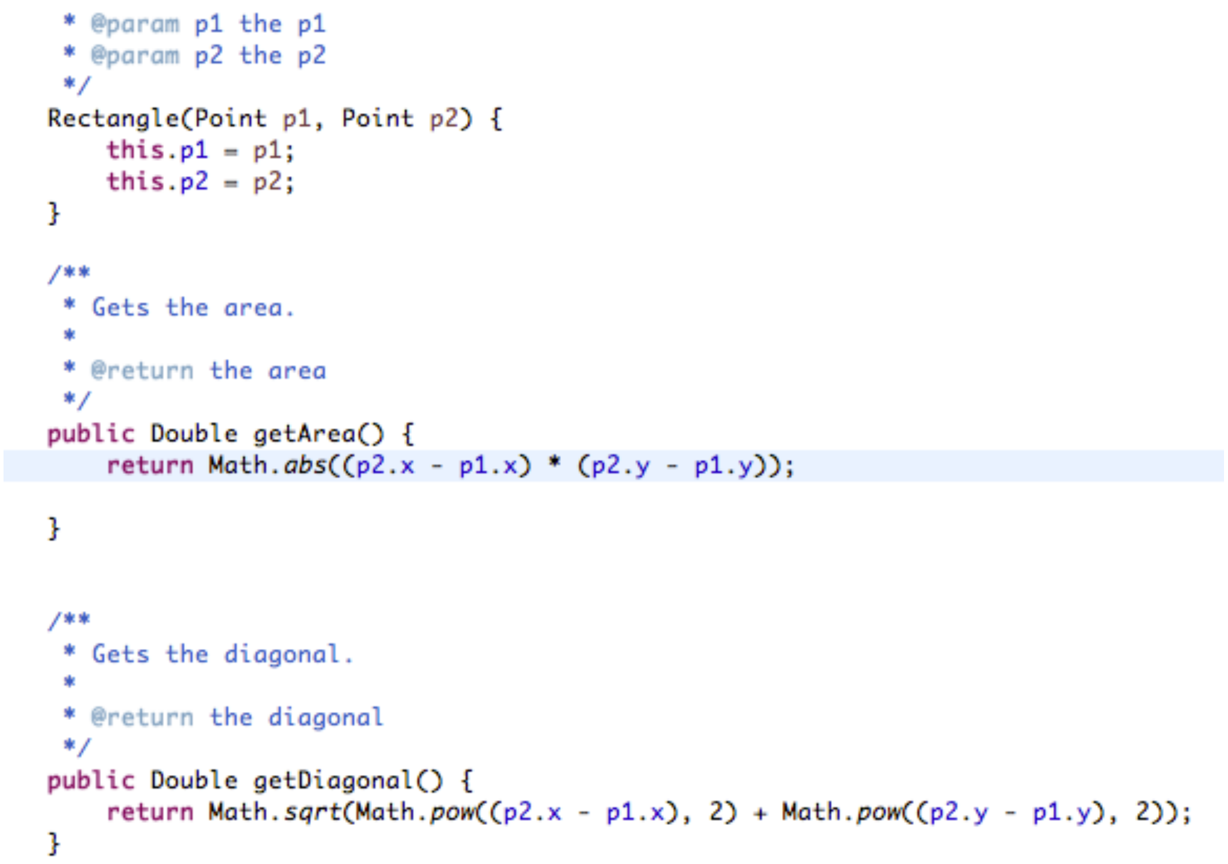
(These were the classes that were initially given)

RectangleTest.java



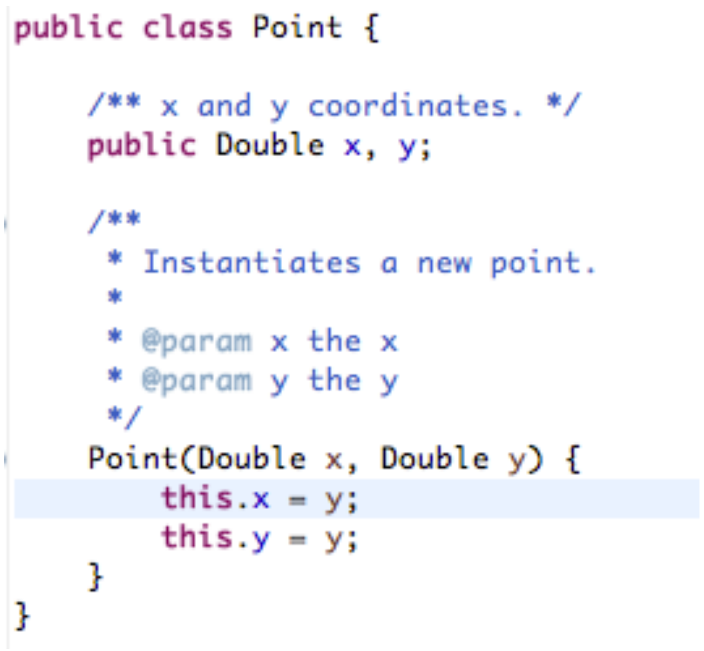
```
@Before
public void setUp() throws Exception {
    rect1 = new Rectangle(new Point(2.0, 2.0), new Point(4.0, 7.0));
    rect2 = new Rectangle(new Point(2.0, 6.0), new Point(4.0, 3.0));
}

/**
 * Test for the getArea() method of the {@link Rectangle} class.
 */
@Test
public void testGetArea() {
    assertEquals(10.0, rect1.getArea(),0.001);
    assertEquals(6.0, rect2.getArea(),0.001);
}

/**
 * Test for the getDiagonal() method of the {@link Rectangle} class.
 */
@Test
public void testGetDiagonal() {
    assertEquals(5.3852, rect1.getDiagonal(), 0.0001);
    assertEquals(3.6056, rect2.getDiagonal(), 0.0001);
}
```

Rectangle.java



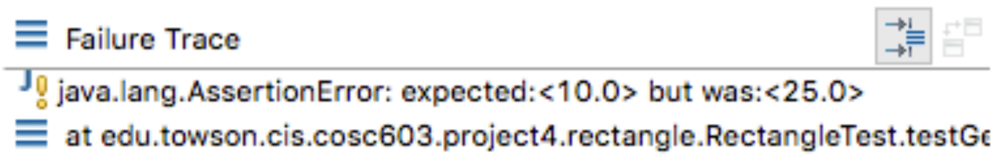
```java
     * @param p1 the p1
     * @param p2 the p2
     */
    Rectangle(Point p1, Point p2) {
        this.p1 = p1;
        this.p2 = p2;
    }

    /**
     * Gets the area.
     *
     * @return the area
     */
    public Double getArea() {
        return Math.abs((p2.x - p1.x) * (p2.y - p1.y));

    }


    /**
     * Gets the diagonal.
     *
     * @return the diagonal
     */
    public Double getDiagonal() {
        return Math.sqrt(Math.pow((p2.x - p1.x), 2) + Math.pow((p2.y - p1.y), 2));
    }
```

Point.java



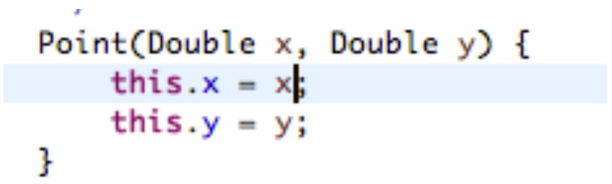
```java
public class Point {

    /** x and y coordinates. */
    public Double x, y;

    /**
     * Instantiates a new point.
     *
     * @param x the x
     * @param y the y
     */
    Point(Double x, Double y) {
        this.x = y;
        this.y = y;
    }
}
```

When running the JUnit test it gave me a red bar. This was the failure trace description i received :



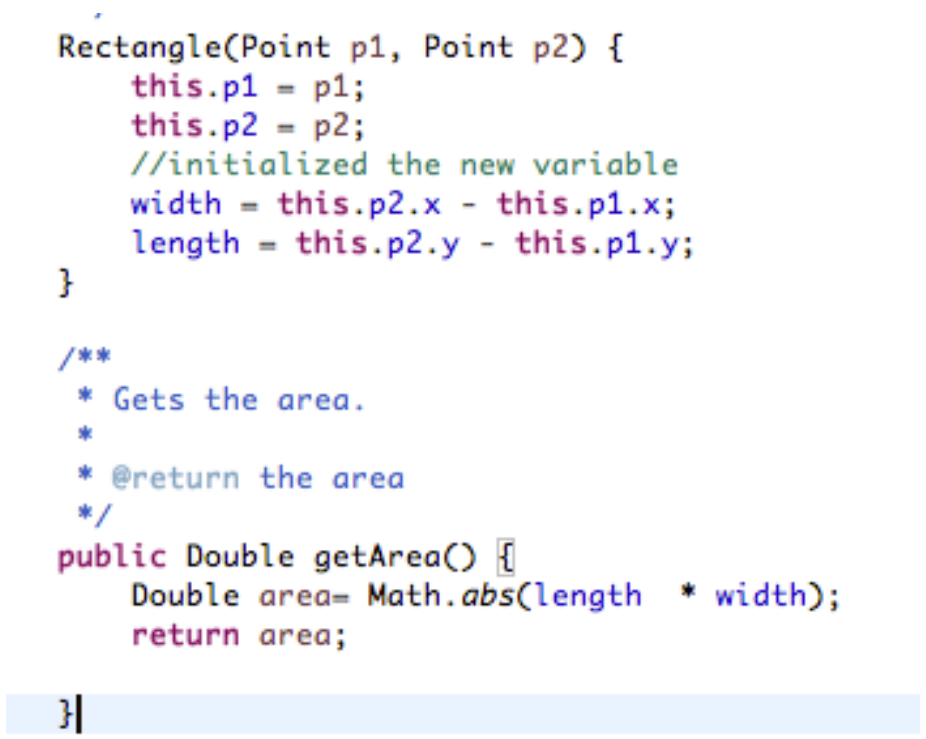
```
java.lang.AssertionError: expected:<10.0> but was:<25.0>
at edu.towson.cis.cosc603.project4.rectangle.RectangleTest.testGe
```

To correct this issue, I discovered a line in Point class that was initialized with the wrong parameter. Originally, it was this.x = y, this was the issue that was failing my getArea method. To fix this, I initialized this.x = x.



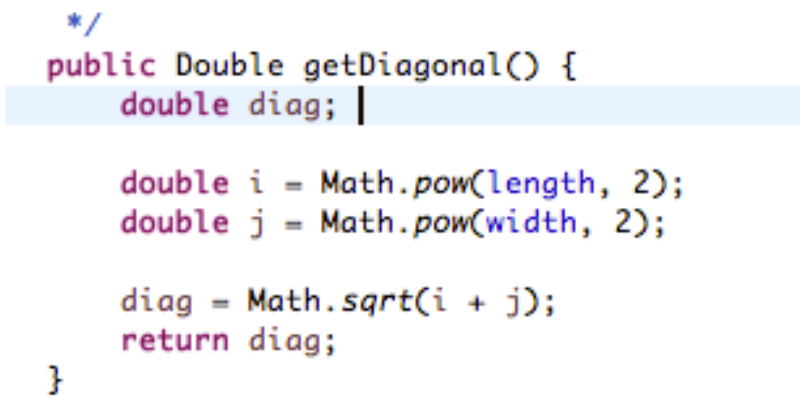
```
Point(Double x, Double y) {
    this.x = x;
    this.y = y;
}
```

To improve getArea method quality, I created two new private variables in rectangle class (width and length). Then I initialized them in the constructor with formula for width and length respectively. In the getArea(), since the formula is initialized in the constructor with private variable, i simplified it by using the private variable in the method.



```
Rectangle(Point p1, Point p2) {
    this.p1 = p1;
    this.p2 = p2;
    //initialized the new variable
    width = this.p2.x - this.p1.x;
    length = this.p2.y - this.p1.y;
}

/**
 * Gets the area.
 *
 * @return the area
 */
public Double getArea() {
    Double area= Math.abs(length  * width);
    return area;

}
```

Correcting the point class also corrected testGetDiagnol () failure notice. However, since I have modified the getArea (), I will also apply that for getDiagnol(). I added two new variables in getDiagnoal () and calculated the squares width and length separately represented by j and i respectively. Also created a variable diag to return the calculated value.



```
 */
public Double getDiagonal() {
    double diag;

    double i = Math.pow(length, 2);
    double j = Math.pow(width, 2);

    diag = Math.sqrt(i + j);
    return diag;
}
```

TASK 4

Found 1 bug;

`When purchasing an item at invalid code slot, the makePurshase method should throw an exception.`

TASK 5

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

Testing is a good resource to invest on. It helps identify bugs and errors that were created during development stage. It nurtures customers satisfaction and confidence in stakeholders specification for building application. This ensures the quality of the product thats being developed: helps software companies with their credibility, which can invite more business. By applying testing, it lower maintenance costs, which means the product that was developed will produce more quality, reliable and consistent result.

From this project, and mainly task 4, I learned concept of writing test cases. Since there weren't main class available to execute these codes, I had to rely on the documentation to visualize the output. Using test cases following the documentation made me realize the importance of having updated documentation. With tools like Junit and testing strategies (test cases) can save the quality of the product in long run, and help costs.

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

I enjoyed using JUnit on eclipse. The simplicity of the framework for writing automated and self-verifying tests in java, test-suite development, and its capability immediate test reporting are what attracted me about using JUnit support for unit test. I believe it's an important feature to add when building a product. It will assure the developer the method product being produced consistent and reliable results.

The only drawback I can speak on JUnit is creating test cases. It may requires deeper understanding of the products and mapping what the codes are doing. For instance, when fixing area and diagonal method for Rectangle. java, I assumed the codes in method were correct. I didn't notice the tiny bug hidden in the Point class until I started mapping it out.