**Kendrick Kwok – CSC 413 – 8/8/2016 – 912351666 – Final Project**

**OVERVIEW** :: The goal of this assignment was to change our last project into another for us to run unit tests and learn more about how to iterate through our collection of shapes. Since this is the first time I had to do unit testing, the hardest part was reading the documentation of unit tests and how it actually worked. Once I realized how to do it, everything became easy.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Index:: i – **Changes of Projects**

ii- **Local Testing VS. Instrumental testing**

iii- **References**

**iv. UML Diagram**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **Changes of Projects**

The changes of this project are provided as followed.

-- I found that AbstractFactory had to be more organized. I had to put a more organized switch statement that allowed the user to switch between the styles and views. Depending on the style and color, these specific looks were placed in the created category. Unit testing can now be performed.

-- I had to put ShapeFactory in a more organized way due to the change of the switch statement in AbstractFactory. I can now grab a style from AbstractFactory.

--A getShapeType() method was created in Shape to test if the shapes transferred to the class has been transferred. I used assertEquals() to see if the getShapeType() method returned the expected string. For example, if I tried to retrieve a circle, the getShapeType() should return me the string “CIRCLE”.

-- Circle() and Rectangle() was given the getShapeType() to accommodate for the change in Shape().

--I created a class called StyleIterator and called the class to iterate on the style button. This is needed to test if the iterator function of style is working correctly.

--New ShapeIterator() class is created to traverse the list of shapes in the vector

--Organized the buttons more carefully. I put all the functions in their right category. The MainActivity() is now more organized and more unit tests can now be performed.

--Unit tests added to test the functionality of the program

1. **Local Testing vs. Instrumental testing**

According to developer.android.com, Instrumented test are tests that run on physical devices and emulators, and they can take advantage of the Android framework APIs and supporting APIs, such as the Android Testing Support Library. Local tests are tests that are performed on local folders. Knowing these definitions, each of my classes had to be split to have a specific test done. Most of my local testing was done with the classes that evoked some sort of design pattern. This included ShapeFactory, AbstractFactory, and Shapes. These design patterns are what tells my program on how to function which is most used in the local folders of my program. Most of my instrumented testing was done in MainActivity. Because my MainActivity is basically the intertwining class that knit all my classes together, most of the android frameworks and APIs were done on my MainActivity. This includes functionalities as setting up buttons and changing text views in my android device.

**Local Testing:**

MainActivity -- Test if the UpdateShape() method sees the vectors and is able to update shape count. Tests if Alpha is setting Alpha up correctly.

AbstractFactory – Test if the retrieval of styles has successfully transferred to ShapeFactory.

ShapeFactory -- Test if the retrieval of shapes has successfully transferred to Shapes.

Shapes – Test if the shape Alpha function works on a circle or rectangle.

Circle – Test if Circle can be set up correctly.

Rectangle – Test if Rectangle can be set up correctly.

**Instrumented Testing:**

MainActivity—Test the functionality of buttons and text views of Android.

1. **References**

<http://stackoverflow.com/questions/4913875/android-how-to-test-a-custom-view>

--I had a lot of trouble on trying to set up the custom view. Using this I was able to see what my options are in trying to do unit testing on a view.

<https://docs.oracle.com/javase/7/docs/api/java/util/Iterator.html>

--This helped me understand iterators and what I had to do to iterate through my collection of shapes

<http://junit.sourceforge.net/javadoc/org/junit/Assert.html>

--A collection of methods that allowed me to understand more on how to perform unit tests

<http://stackoverflow.com/questions/1201927/java-is-assertequalsstring-string-reliable\>

--I used this to see if using this assert method is a good way in how to perform unit tests. I tried looking for more ways on how to test my programs.

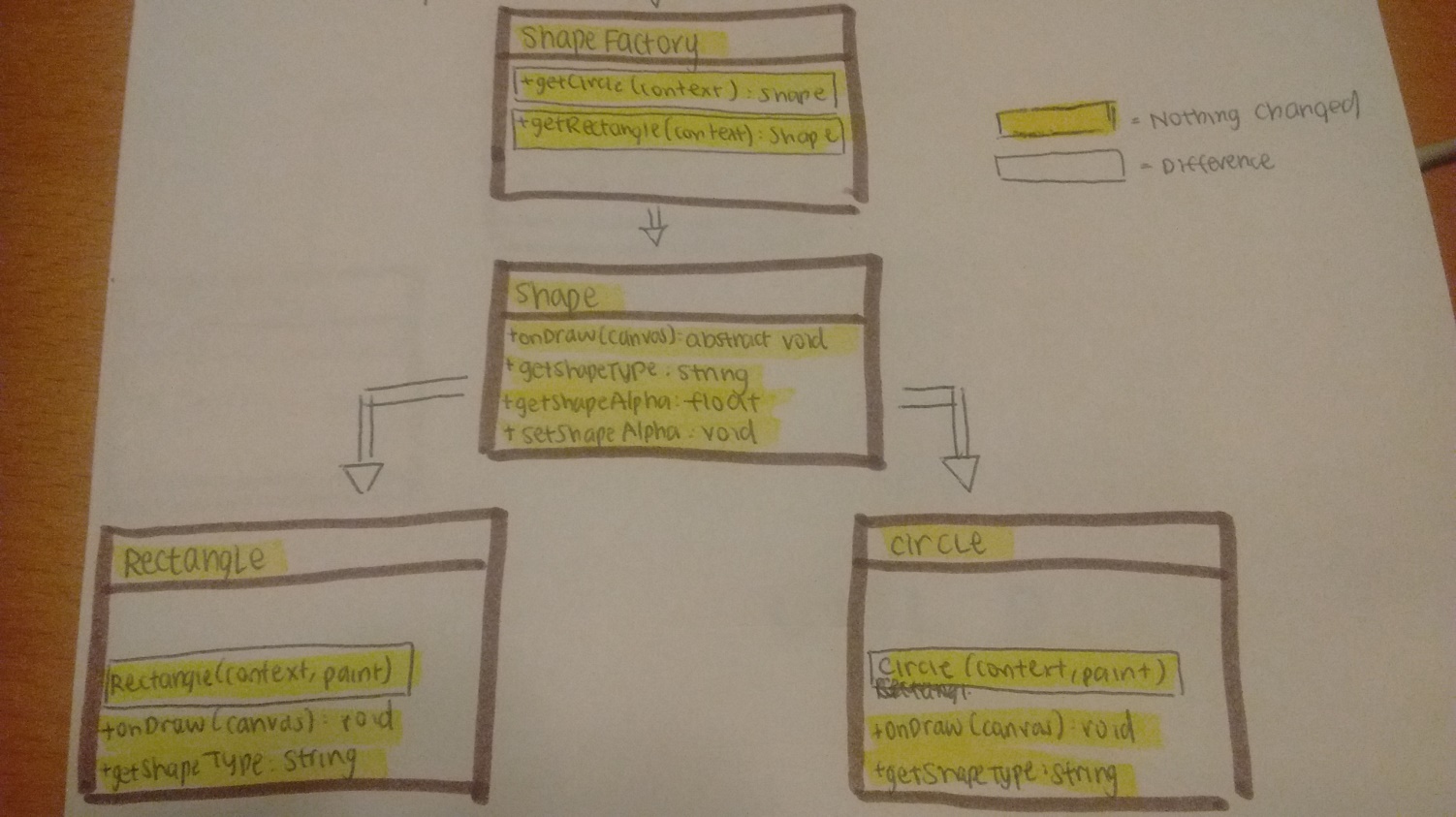
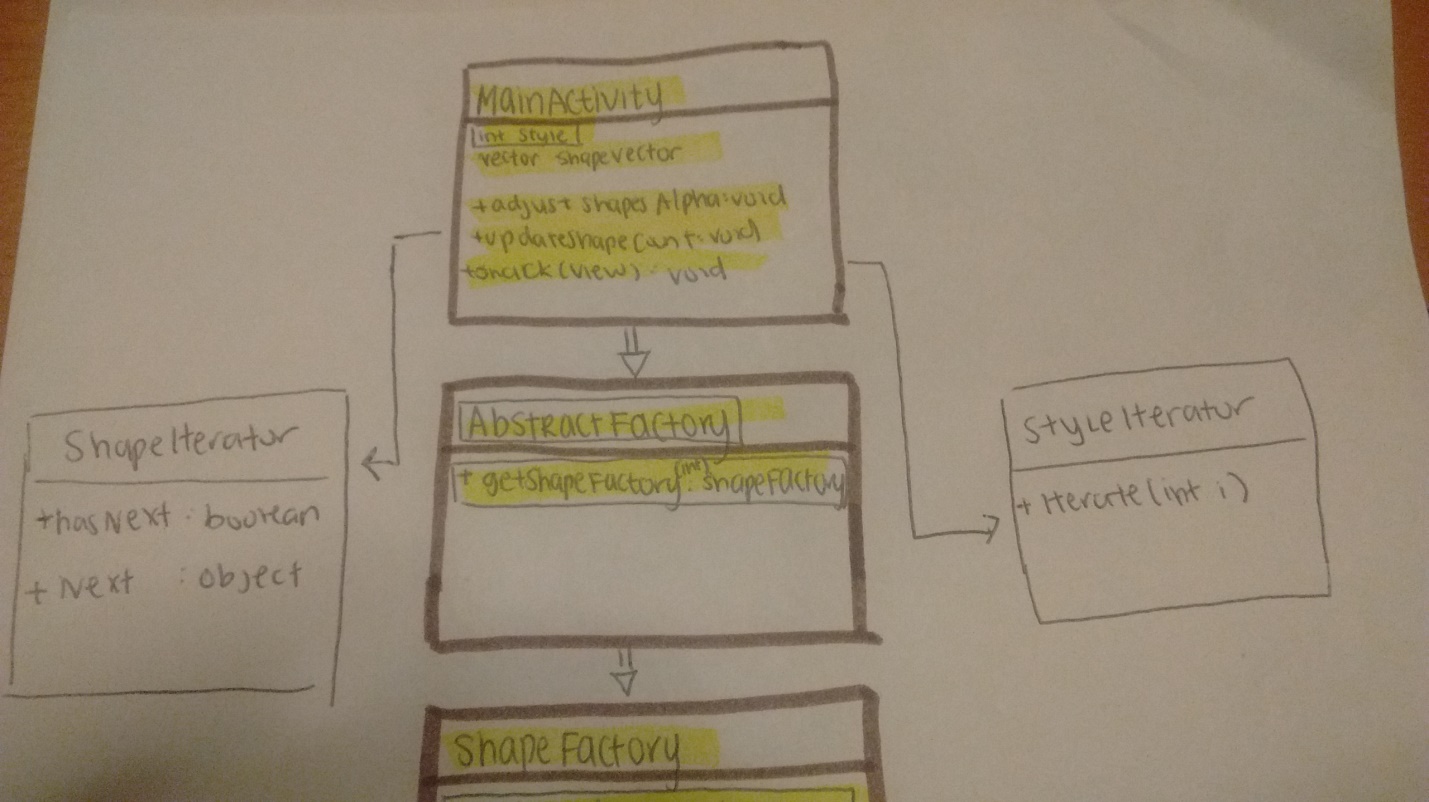
<http://searchsoftwarequality.techtarget.com/definition/unit-testing>

--This provides documentation on how the unit testing actually works. This gave me more hind sight on what I need to in order to perform these tests.

<https://sourcemaking.com/design_patterns/iterator/java/1>

--More about what iterators and how I am able to build this iterator class

1. **UML DIAGRAM**

****