**Assignment 3 Sandeep Saripalli | COSC603**

Task 4

For Monopoly java code initially chance card was declared to chance by changing to CC junit throws an error after running 44 test cases as shown in the screenshot it got one failure and the failure happened with the testCardType and junit shows why it failed junit.framework.AssertionFailedError: expected:<2> but was:<1>.

Task 5

This omission is not oversight as class files and java files are not changed. Changes are done to only developer defined argument names, Eclipse changed getter and setter methods as they are defined by the developer and it is not a java file or class file code.

/\*\*

\* Sets the theOwner.

\*

\* **@param** theOwner the new theOwner

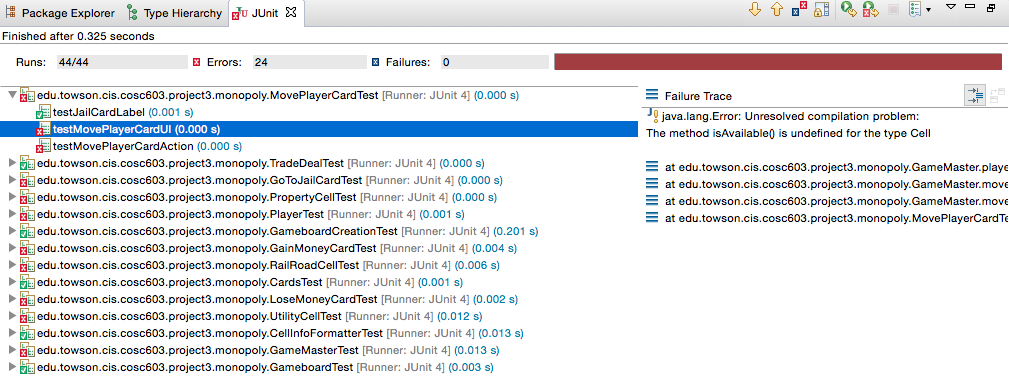
\*/

we can see javadoc files are changed to meaningful description of method or variable.

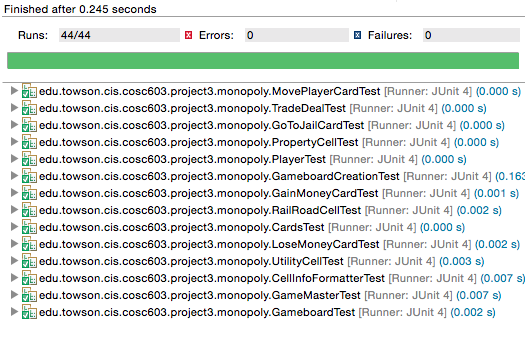
It is completely different than search and replace as Refactoring not only changes the variable names but methods and arguments to with out disturbing the code which cannot possible with search and replace as it needs exact words.

Task 6

Pulling down broke the code lot many errors unit test showed many more errors here the screen shot we can see.

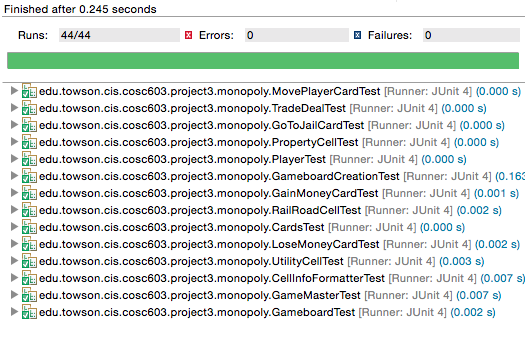


Pulling up fixed the errors, which has been caused because of the Refactor pulling down. Pushing down and pulling up can be used in many scenarios for methods or variables to the classes where they have strong presence for cohesion.



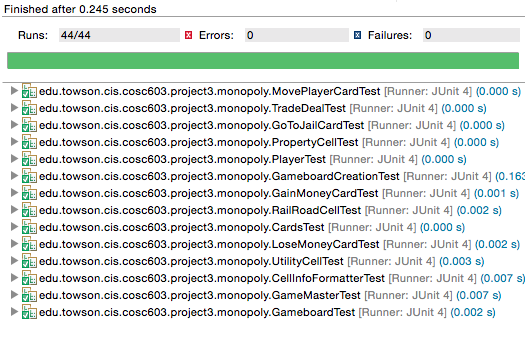
Taks7

Interface enforces implementing classes to use the methods we selected during refactor interface. This enforces certain kind of structure while coding. It gives flexibility and writing the code in the more generic terms and gives the freedom to extension and subtyping the classes. Junit test cases were successful. This changed many subclasses where these subclasses were overriding the methods in the superclass, with the introduction of interface they were accessing methods of the interface which were being implemented by the superclass. Thus, creating a sense of uniform behavior throughout the application.



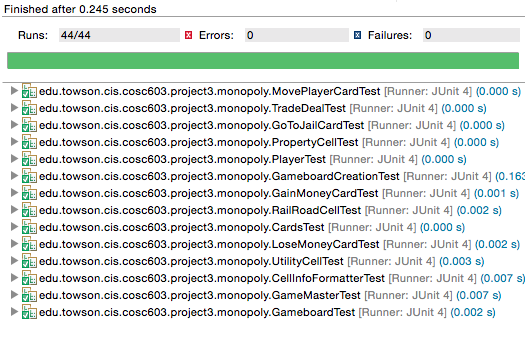
Task 8

I chose second method as arguments are minimum in this scenario and it did more processing with less memory usage and more encapsulation. Thus hiding most of the algorithm hence it made code more readable and understandable. Junit test was successful.



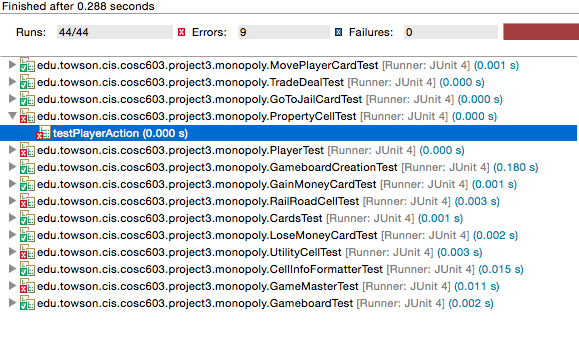
Task 9

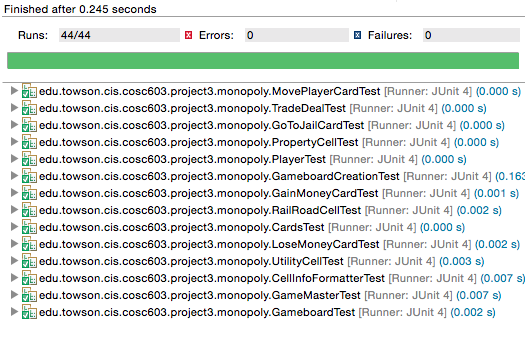
We can create a local variable for repeated method usage, which helps in maintainability and readability of the code. But this won’t apply in every scenario as sometimes same methods change the static variables or volatile variables which may cause errors for example ehcache implementation where every method call updates the cache. Junit test was successful.



Task10

We modified a signature of a method, which caused many errors which can be seen in the screenshot to fix this I went manually to all the files which has issues and added a return value. With this change we are modifying the behavior of the method and by using this kind of refactoring we can modify the method as per our requirements.





Task11

* I never used refactoring tool or refactoring before so I have learnt lot of new things without breaking the code we can see the changes with the preview feature. Refactoring and design smells in class has practical meaning now. I could gain some practical knowledge .
* I never used refactoring in eclipse too. I can for now only see the advantages in finding the design smells.
* In order to maintain the integrity of the code or to avoid unintentional code breakage or to avoid accidental change of the behavior of the code we tend to use JUnit test or unit testing to prevent such changes.