Comp 47480 Learning Journal – Assignment 4

1. Reflection on my Learning on the Object Orientated Principles.

The first task in the tutorial was to alter the OCP class so it adheres to the rules set out in the open closed principle. To implement the open closed principle, we need to first understand to satisfactorily adhere to this the class or module should be open to extension but closed to alteration. We can extend the functionality of a module without necessarily changing the internal code of the class itself. Instead, we can create a subclass that extends the original class in this way the original code remains the same, but we can add more functionality. This is achieved by using abstract classes or in interfaces by using these we can make or classes more cohesive and less coupled. In the example code I implemented an abstract class Shape. This means that I can now have many classes extend by shape for instance triangle square and circle etc. By doing this we can also make some of the variable private, so they cannot be changed outside of the class.

The second task in the tutorial was to alter the SRP class so it adheres to the rules set out in the single responsibility principle. The single responsibility principle means that each class or module should have one responsibility. 'If a class has more than one responsibility it is overcomplicated, and the complexity is increased. Usually, this is because the classes have dependency on each other. Also, as a result the programmer may be giving functionality to a class that the object would not have in the case of the hexapod it represents both the human and the dog however the human should not be able to bark, and the dog shouldn't be able to throw the stick. Therefore, we can see clearly that the Hexapod class doesn't met the requirements of a single responsibility. Therefore, the hexapod class was split into two classes the human and the dog class giving only the methods that belong in each class to the respective classes.

The first two tasks of the assignment were relatively easy to understand and implement. The third task which was to alter the Demeter class. I found this much more difficult to implement. I believe, I have been able to implement it as it should so that, it follows the laws of Demeter. These rules promote classes, that are loosely coupled because limits how much an object can know about its environment. In the original, Demeter code the shop keeper was able to directly access the customers wallet. However, the shopkeeper should ask for a payment and then the customer should check to see if they have enough and make the payment. Demeter in basic terms, means that it prevents the programmer accessing a third objects/classes method. When we apply this to our example Demeter class the shopkeeper should be able to interact with the customer and the customer should be able to interact with the wallet. The shop keeper should never be able to interact with the wallet. This would make your classes easier to reuse ad you code will both look cleaner and be easier to test. Due to the way Demeter enforces its laws normally the classes written would have fewer errors and because there not really intertwined

with other classes alterations in other classes are less likely to affect it. There are however some disadvantages to Demeter is that, if you need to make the third object do something then the clearest way might be to pass the third object into the method. It could also be implemented by providing another class that offers something like an actor interface that passes the request to the class required. This can make you code base larger and slower but on the other hand it will be infinitely easier to maintain and more portable.