IT7320 Assignment 3 part one

# Discussions of part 4e:

In this part of the assignment, I have written all the necessary classes for the testing purpose. The PayRollCompute object is being instantiate and all necessary variables are also initialized in the setup() method of the PayRollComputeTest class.

For this first part I have written the test for the computeTax() method, with no mocking:

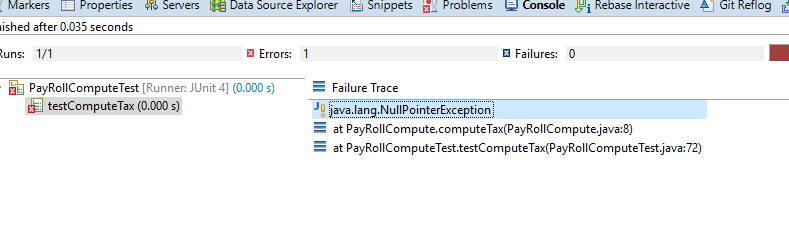
*@Test*

***public******void*** *testComputeTax() {*

*assertEquals((taxRate \* mySalary), myPayRoll.computeTax(mySalary), 0.01);*

*}*

The result of the test case running is shown in the screen capture below:

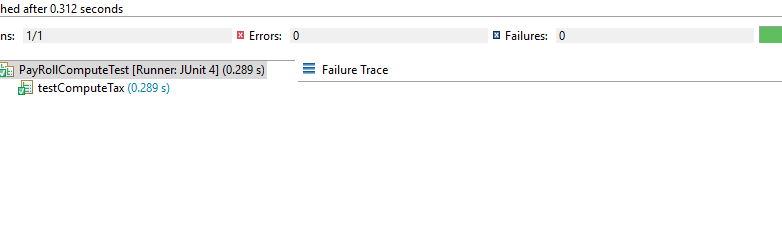


As shown in the above screen capture, the test case results in an error instead of pass or failure. This can be explain by the fact that the computeTax method makes a call to the computeTax method in the ComputeTax interface object inside PayRollCompute object, however, this ComputeTax object is not instantiated yet, and therefore is null, when we run the test, the myPayRoll.computeTax() calls a null object’s method, which results in the error.

This shows us the use of mocking, as in this case, we might be unable to instantiate the ComputeTax object due to circumstances like it does not exist yet or codes for it is not finished, and the fact that we want unit test to be independent of other methods, so that failures of external methods does not affect the accuracy of our unit test. Therefore, we want to create a replacement object – a mock which offer controlled behaviors e.g. we can always sure of what it returns. The mock object is therefore essential in keeping unit testing independent of external failures and also eliminated dependency on other people’s work and coded to do our unit testing.

# Discussion of part 4i:

I have carried out the work as instructed on part f to part h, after rerunning the test case, the following result in shown in the screen capture below:



As shown above, the test case now passes when run, which is what is expected, since now we have created a mock of the ComputeTax interface and added behaviors into its computeTax() method, now when we call it when testing PayRollCompute’s computeTax() method, we can always get the controlled return, instead of getting the error of nullPointer since there is a mock object in place of the previously null ComputeTax interface object.

# Part m – Screenshot of junit test Runner

