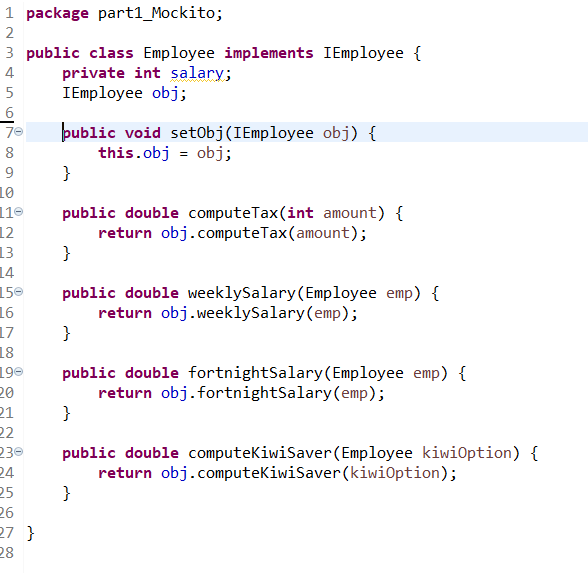
|  |
| --- |
| Wellington Institute of Technology |
| IT7320 – Development and Testing of Software |
| Assignment 3 Part 1 – Mockito |

|  |
| --- |
| Dan Mota ID: 2150708  Due Date: 26/09/18 |

# JUnit – Mockito

The exercise starts with the creation of the class Employee and interface IEmployee, as well as the appropriate implementation of the interface methods using the keyword ‘implements’ on the Class Employee definition.

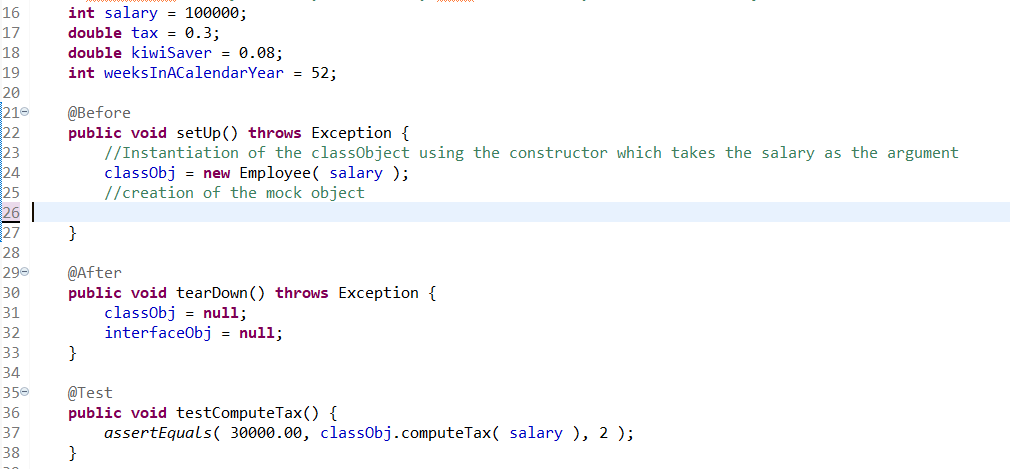
Unimplemented methods were added, an object of IEmployee interface was created which is used to call its own methods as thus:



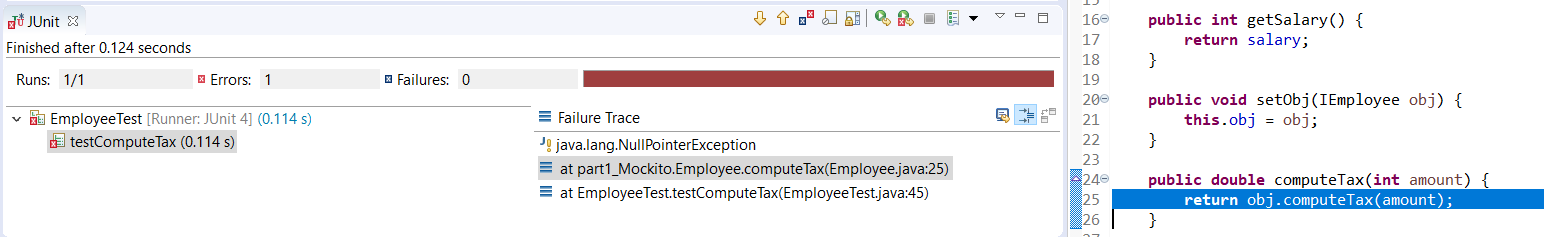
A new JUnit test case for class Employee was then created. Objects of class Employee and interface IEmployee were created, and assumed test variables were initialised: salary of type int as 100,000, tax of type double as 0.3, kiwiSaver of type double as 0.8, weeksInACalendarYear of type int as 52.

In the setUp method the class object is instantiated using variable salary as the parameter for the constructor.

A test is created where the expected value of compute tax is 30,000 based on our assumptions: salary = 100,000 and tax = 30%.



Running this test results in a NullPointerException, given that the method called executes the method inside the interface IEmployee which is a skeleton code.



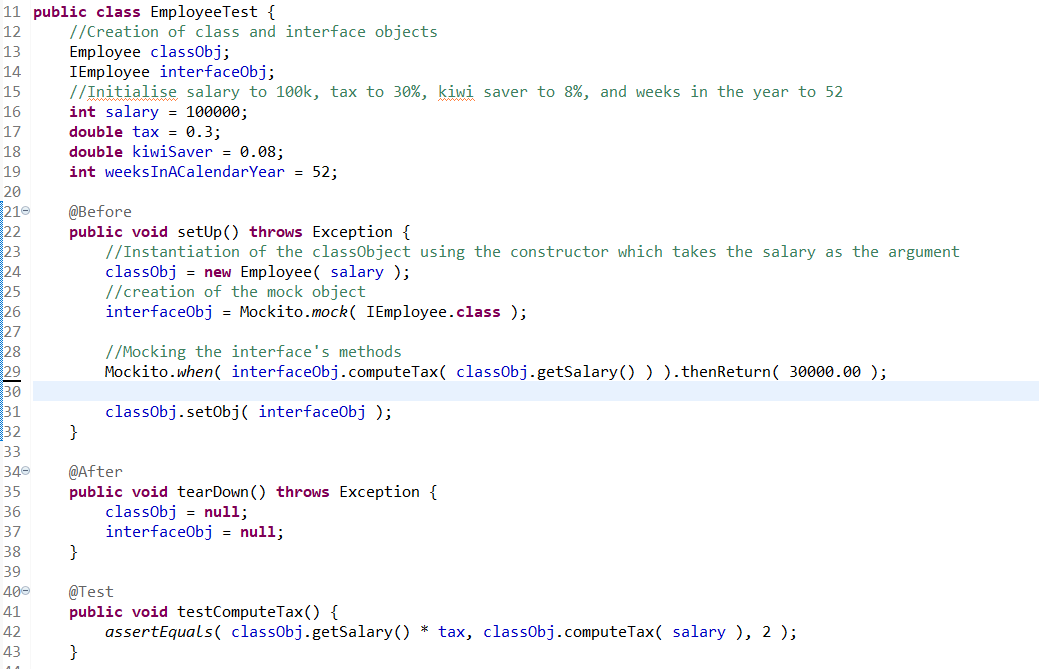
## Why Mock?

“The purpose of mocking types is to sever dependencies in order to isolate the test to a specific unit.” [1]

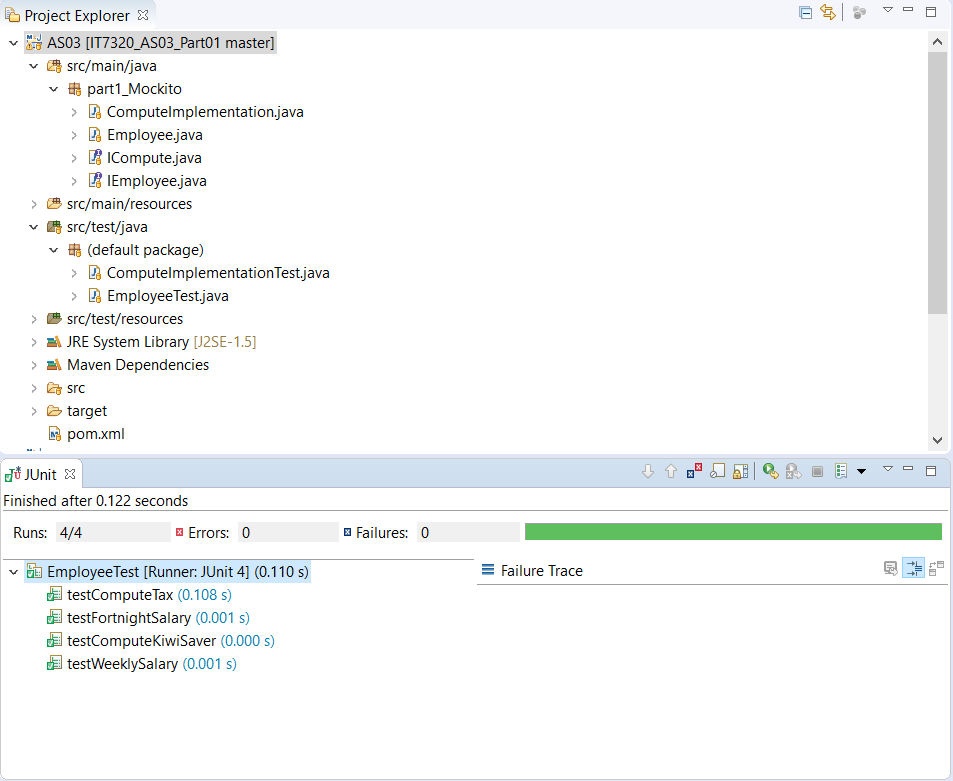
An interface was selected because it gives enough a degree of abstraction that is necessary when mocking. Classes are concrete and can be tested. What an interface does and how it works is often considered beyond the knowledge of a developer.

Likewise, parts of the system cannot be disconnected for the purposes of error fixing/patch creation in real life. Code that performs changes through a network – i.e. database and server connections – must be mocked to avoid any possibility of corrupting data or interfering with business processes.

Lastly, these connections are unit dependencies – and as stated in the first line of this section – dependencies must be severed in order to properly conduct a unit test.



As can be seen in the code above, a mock object is created in line 26. In line 29, the processed data return for the method computeTax is mocked. Given the salary is 100k and the tax rate 30%, the expected result is 30k as a double.



The same process of mocking method calls is carried out for the remaining methods of IEmployee interface: fortnightSalary, computeKiwiSaver, weeklySalary. Test cases were also created for each of the methods by making use of Mockito’s when().thenReturn() static method returning the appropriate expected return for each interface method.

3,846.15 is returned for fortnightSalary ( 100,000 / 26 )

1,538.46 is returned for computeKiwiSaver ( 100,000 / 52 \* 0.8 )

1,923.08 is returned for weeklySalary ( 100,000 / 52 )

The image above illustrates the test result for all our test cases. Methods successfully mocked, and assertions successfully performed.

# Appendix A – GitHub

## Profile - Dan

The author’s profile and GitHub project for this exercise can be found [here](https://github.com/Seagold8930/IT7320_AS03_Part01).

## Source Code

The source code for this exercise can be retrieved from GitHub [here](https://github.com/Seagold8930/IT7320_AS03_Part01/tree/master/AS03/src).

# Appendix B – Resources

[1] <https://stackoverflow.com/questions/2665812/what-is-mocking> retrieved on 19/09/18 at 3:51PM