Lab Exercise 4: More testing –using NUnit and structuring things nicely

In this exercise, you will write some more tests using NUnit. A skeleton-solution (i.e. incomplete solution) for a stack based calculator can be found on CampusNet (SWT_Ex4.zip) – download, unzip and open.

The important part of this exercise is to comply, to the fullest extent possible, with Roy Osherove's guidelines for structuring and naming your test code.

In the solution you will find 2 projects: Calculator and DemoMain.

- **Calculator** – This is a skeleton class library that you must complete and test. More precisely: You must complete the *StackCalculato*r class so that all its methods are complete and works as described in the interface (i.e. the methods must return the right values and throw the right exceptions at the right times). Some of the methods have already been written partly or in full – these should also be tested (and completed, if necessary).

- **Calculator.Test.Unit** – This is a skeleton class library for unit testing the code in the Calculator project. You should add all your test cases here. You should follow the naming convention Osherove describes in section 6.4 and 7.3 in the book.

- **DemoMain** – This is just a demonstration console application that shows how the stack calculator may be used later on. You can run it and use it for "happy testing". You can also use it to see how the "index" feature of the StackCalculator class works.

The recommended approach is to write and test one method to completeness at a time.

Exercise 4-A:

You must first test, using the guidelines for structuring and naming your test code, the methods (including the "index method") that have already been implemented. You should write some test cases that uses "classic" Assert testing, and some test cases that use constraints-based asserts. The NUnit documentation provides good help here.

You should write test cases that tests for exceptions using all 3 methods presented in the lecture:

- Using attributes [ExpectedException ….]
- Using Assert.Throws(<expression>)
- Using Expect(<expression> , <constraint>)

Exercise 4-B:

Complete all the missing methods and test them, using a mixture of classic Assert's and constraints. Continue using the guidelines for structuring and naming your test code.

Exercise 4-C:

Use the Test Review Guidelines found at http://artofunittesting.com/unit-testing-review-guidelines at your code. Some of the rules are not relevant (such as the stuff that talks about using mocks and stubs in the right way), but try to check as many rules as possible. If you find that your tests violates one or more rules (by using logic, for instance) refactor your test code as needed.