# TCI – week 3 practical

These exercises should be executed individually. All of these are assignments are needed in the final project.

## 

## Part A: Unit Tests

### Assignment 1 – Creating parameterized tests

Look at TDD example exploring the domain <https://www.youtube.com/watch?v=Jx3Vi330o4M>

Implement the tests for canAddTwoNumbers or canAddTwoIntegers using parameterized tests.

(you might need to add something to your gradle.build file. What is it?)

### Assignment 2 – parameterized tests in existing code

* In week 1 you have worked on creating tests and later implementing the code for some basic statistic. Now we are going to expand this by using parameterized making tests for the rest of the methods in BasicStatisticInterface

|  |  |
| --- | --- |
| Task | Description |
| * Parameterized test | * Use parameterized tests to test the getHighestValue method * use separate data provider methods for ‘nice’ behaviour (not throwing exceptions), and for ‘unique’ behaviour (throwing exceptions) * do the same for getMean |
| * Use setup of your test fixture | * To be able to use the same starting point for future methods getMedian and getStandardDeviation, make sure you use a test fixture, which makes this possible. Also add parameterized tests for these methods. |

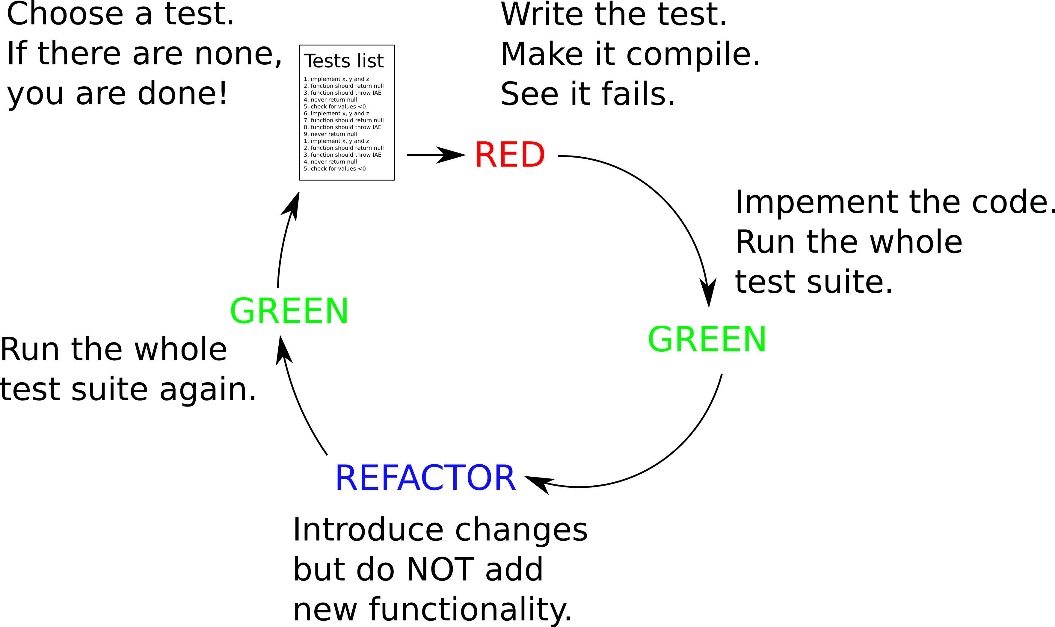
### Assignment 3 – Basic mock exercises

Tests with mock objects: Do exercise 5.7.1 from the book.

### Assignment 4 – Creating Tests based on requirements AND using Mock Objects

**Objective: Developing Unit tests using TDD with mocking**

Look at the requirements below and translate them to unit tests one by one. Make sure you use the TDD rhythm as described in chapter 4:



The case is involving modeling of a Book with Chapters

|  |  |
| --- | --- |
| Requirement | Description |
| **A chapter is described by name and number** | * Name is a not null string * number is a string like ‘2.4’, ‘3’, ‘4.3’ etc. 2 levels of chapters is maximum. * If anything is wrong with the parameters, an IllegalArgumentException is thrown. |
| **Chapters can be used in a sorted collection** | * They must implement Comparable and override equals and hashcode. |
| **A Book is described by name and author** | * Both not null, and not empty. Otherwise IllegalArgumentException. |
| **You can add a Chapter to a book** | * Parameters include the name and number of a chapter after which a Chapter Object is created and added, * Otherwise IllegalArgumentException. |
| **You can get a Table Of Contents** | * A Set is returned which contains a clone of the Chapters of the Book, sorted by their natural ordening. |

Advice:

Always start implementing tests for the constructor first, since you will need it as part of the test fixture in other tests.

*Reflection points:*

* Did you manage to stick to the TDD rhythm? If not, what happened? What do you need to be able to stay in the TDD rhythm next time? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ .
* Are the names of the test methods self-explanatory? \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ .
* Do you think that reading the body of your tests would be enough to understand what the requirement was? Are they -in other words- self describing which requirement they are meant to test? If not, what could you do to improve them?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ .

* In which tests did you need to use mock objects? Why?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ .

**When done and having used TDD as a method: Congratulations, you’ve just proved that all requirements are implemented in the code! Also all requirements are documented as tests!**

========================== End of practical. ===============================