# TCI – week 2 practical

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

## Unit Tests

### Assignment – Basic mock exercises

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

### Assignment – Creating Tests based on requirements using TDD

### 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. |

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

Do exercise 5.7.2 from the book.

*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?

For the most part I was able to stick in the rhythm TDD even if it was only for one cycle. It became sometimes hard to refractor code when it was already simplified as far as possible. Also it could be frustrating to change habits of writing code making it difficult to stay in the rhythm.

* Are the names of the test methods self-explanatory?

Yes

* 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?

Mostly apart from some of the exception tests that were made that were only calling one method

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

I needed to use mocks in the tests where I did not have the implementation of the objects yet so I could only mock the behavior. This was also not a problem as they were unit tests so it is not to worry about what the other component had to do just that it was called or returned the expected value.

**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. ===============================