CS2001 Week 5: JUnit and Test Driven Development

Jon Lewis (jon.lewis@st-andrews.ac.uk)

Due date: Wednesday 16th October, 21:00
25% of Practical Mark for the Module

Objective

To gain experience with Test Driven Development and implementing and testing Abstract Data Types.

Learning Outcomes

By the end of this practical you should understand:

- how to design and implement unit tests
- how to use a Test Driven Development methodology to produce a robust implementation of a specified ADT interface.

Getting started

To start with, you should create a suitable assignment directory such as `CS2001/W05-JUnit` on the Linux lab clients. You should decompress the `zip` file at

`http://studres.cs.st-andrews.ac.uk/CS2001/Practicals/W05-JUnit/code.zip`

to your assignment directory. Please note that the `zip` file contains a number of files in the `src` directory, some of which are blank or only partially implemented. **Once you have extracted the `zip` file, you should probably delete it to avoid accidentally overwriting your `src` directory (and thereby your own implementation) with files contained in the `zip`.**

Requirements

You are to develop an implementation of the ADT interfaces for a shop, stock records and products in the `interfaces` package in the `src` directory following a TDD process. That is, your job is to write suitable tests in the `test` package (such as in the supplied `Tests` class) and implement all the classes in the `impl` package to satisfy these tests. Parts of `impl.Factory` have been implemented, for this class you only need to implement the methods containing `// TODO` comments.

Please note that you must **NOT** change the interfaces. Coding to a given interface is an important aspect of development and is seen as a valuable exercise for this assignment and many others.

Following the TDD process, try to write `JUnit5` tests for each element of functionality before writing and testing its implementation. When writing your tests, consider the following:

- Normal cases, with expected inputs.
- Edge cases, such as empty collections, duplicate bar codes for different products, no stock.
- Exceptional cases, such as dealing with `null`s.

Should you find some aspects of the interfaces ambiguous, you will need to make a decision as to how to implement the interface, which your tests should make clear.

Running the Automated Checker

You can run the automated checking system on your program by opening a terminal window connected to the Linux lab clients/servers and executing the following commands:

```
cd ~/CS2001/W05-JUnit
stacscheck /cs/studres/CS2001/Practicals/W05-JUnit/Tests
```

assuming `CS2001/W05-JUnit` is your assignment directory. This will run your own JUnit tests in the `test` package on your own ADT implementation. A single test method is included in a `Tests` class in the `test` package. It merely checks that the factory method for creating a product (which you will have to implement) calls a suitable product constructor (which you will also have to implement). The test also demonstrates how to use the factory to instantiate objects. Your first step could be to make this failing test succeed before going on to add the next failing test(s), adding the corresponding implementation, re-running all tests etc. in the TDD process. The final test `TestQ CheckStyle` runs a program called Checkstyle over your source code using the *Kirby Style* as mentioned for the exercise in week 1.

`https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/programming-style.html`

Deliverables

Hand in via MMS, by the deadline of 9pm on Wednesday of Week 5, a zip file containing:

- Your assignment directory with all your source code, including your tests and ADT implementations.
- A PDF report describing your test cases, how you chose those test cases, and how your choice of test cases influenced the implementation of your ADT. You should also explain and justify your design and implementation decisions in clarity and detail.

Marking Guidance

The submission will be marked according to the mark descriptors used for CS21001/CS2101, which can be found at:

`https://studres.cs.st-andrews.ac.uk/CS2001/Assessment/descriptors.pdf`

A very good attempt in an object-oriented fashion achieving almost all required functionality, together with a clear report showing a good level of understanding, can achieve a mark of 14 - 16. This means you should produce very good, re-usable code with very good method decomposition and provide a very good set of tests with clear explanations and justifications of design and implementation decisions in your report. To achieve a mark of 17 or above, you will need to implement all required functionality with a comprehensive set of test cases, testing all aspects of your design and covering any cases. Quality and clarity of design, implementation, testing, and your report are key at the top end.

Lateness

The standard penalty for late submission applies (Scheme B: 1 mark per 8 hour period, or part thereof): `http://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/assessment.html#lateness-penalties`

Good Academic Practice

As usual, I would remind you to ensure you are following the relevant guidelines on good academic practice as outlined at

`https://www.st-andrews.ac.uk/students/rules/academicpractice/`