Week 3: JUnit and Test Driven Development

CS2001

Due date: Wednesday 3rd 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 without changing the interface (for the basic implementation).

Getting started

To start with, you should create a suitable assignment directory such as `CS2001/W03-JUnit` on the Linux lab clients. You should decompress the `zip` file at

```
http://studres.cs.st-andrews.ac.uk/CS2001/Practicals/W03-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 and have started implementing your solution, avoid de-compressing the zip file to the same directory again as this will overwrite 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 simple loyalty card operator, loyalty cards and card owners in the `interfaces` package in the `src` directory following a TDD process. The main idea of the loyalty card system is for card owners to be able to earn points on their card when buying items for money. Points can be accumulated by owners on their card and subsequently used to purchase items. In the simple version outlined in the interfaces, owners earn 1 point for every 100 pence they spend. Each accumulated point is worth 1 pence when using points to make any subsequent purchase. Your job is to write suitable tests in the `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 for the basic implementation. Coding to a given interface is an important aspect of development and is seen as a valuable exercise for this assignment and many others.

For extension work (as mentioned in the "Marking and Extensions" section below), you can alter the interfaces as long as you place all your extension code in a separate folder such as *W03-JUnit-Extension* within your submission.

Following the TDD process, try to write `JUnit` 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 collections containing zero or one element, duplicate owners, insufficient points.
- 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

Similarly to the exercise in week 1, you can run the automated checking system on your program by opening a terminal window connected to the Linux lab clients/servers and execute the following commands:

```
cd ~/CS2001/W03-JUnit
stacscheck /cs/studres/CS2001/Practicals/W03-JUnit/Tests
```

assuming `CS2001/W03-JUnit` is your assignment directory. This will run the JUnit tests in your `Tests` class on your ADT implementation. A very basic test is included in the `Test` class in the `zip` file. It merely checks that the factory method for creating a loyalty card owner (which you have to implement) calls a suitable loyalty card owner constructor (which you will also have to implement). Your first step could be to make this failing test succeed before going on to add the next failing test(s) to the `Tests` class, adding the corresponding implementation, re-running all tests etc. in the TDD process. The final auto-checker 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
```

As mentioned in the exercise for week 1, you don't need to worry too much about the use of TABs versus spaces for indentation.

Deliverables

Hand in via MMS, by the deadline of 9pm on Wednesday of Week 3, 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 describe your implementation as well as any design and implementation decisions, for the basic requirements and any extension elements. You can find some useful tips on report writing in the student handbook at

```
https://info.cs.st-andrews.ac.uk/student-handbook/files/
        writing-guidance/report-writing-tips.pdf
```

Marking and Extensions

The submission will be marked on the University's common reporting scale according to the mark descriptors in the Student Handbook at

```
https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/
                    feedback.html#Mark_Descriptors
```

To achieve a mark of 14 - 16, the submission should contain clear, well-structured code achieving almost all required functionality, together with a clear report showing a good level of understanding. Please note that the basic requirements include producing a very good implementation of the ADT and a very good set of tests.

To achieve a mark of 17 or above, the submission should contain excellent code satisfying all basic requirements, together with a clear and well-written report showing real insight into the subject matter. You will need to implement a comprehensive set of test cases, testing all aspects of your ADT and covering normal, edge, and exceptional cases.

You may also consider extension activities indicated below, but please remember, it is key that you focus on getting a very good solution to the core requirements prior to touching any extensions. A weak solution with an attempt at extensions is still a weak solution.

Extension activities include:

- adding and testing suitable interfaces and functionality that permits card owners to earn additional points when buying certain *products* for which there is a special *promotion*
- providing the functionality for loyalty card owners to use their points as part of a monetary transaction
- implementing a user interface to demonstrate and potentially visualise the operation of the loyalty card system

You do not need to feel limited by these extensions; use your imagination to come up with others!
As mentioned above, for extensions, you are permitted to alter the interfaces as long as you place all your extension code in a separate folder such as *W03-JUnit-Extension* within your submission.

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/`