Clarifications of CW3 expectations

A. <u>From 4th of April</u>

A very important piece of advice: **<u>you should consider the shielding individual classes (i.e. interfaces + implementing class), supermarket classes and catering company classes all as being part of 3 different clients</u>**:

- the shielding individual classes (i.e. interfaces + implementing class) are part of a shielding individual client
- the supermarket classes (i.e. interfaces + implementing class) are part of a separate supermarket client
- the catering company classes (i.e. interfaces + implementing class) are part of a separate supermarket client

You have them all part of the same skeleton code just for simplicity, but they are in fact to be seen as 3 separate clients, keeping things modular. Therefore **(IMPORTANT!) you should not associate classees from these different clients together, the clients need to be kept separate. When we said you need to re-use part of the hierarchy of classes, we mostly referred to classes referring to orders and their contents that would be kept by the shielding individual client**. Think of the shielding individual keeping a cache of orders so that e.g. an elderly person who doesn't have a stable internet connection can still see what orders they had placed according to when the system was last online. *Also think about other reasons of why this is useful, and what makes ir problematic*. As a result, you still don't have a way to retrieve orders from the server.

You are welcome to also 'cache' other information that is also stored on the server, but again make sure you don't connect the different clients which must be kept separate!!!

B. <u>From 11th of April</u>

1. **DON'T change the server code**

2. **DON'T change method signatures provided in the skeleton code, not even for adding 'throws ...' for exceptions.** Therefore, these methods should not have 'throws....', but **you can add other private methods in the ClientImp classes (in general and) which do if you want**.

3. **When I said 'cache' orders I did not mean using libraries for caching, but just having the classes and subclasses needed for them and saving these in the shielding individual client**. This is something that you should do there for sure as there is no way to get the orders from the server. You can also 'cache' other things, but only if you have a good justification for doing this vs. re-calling the server endpoints, which (the justification) you should make in your report.

4. The shielding individual's method **getFoodBoxNumber should return the complete number of food boxes, and not or any dietary preference**

5. The shielding individual's **methods getting information (number of items, quantities etc) for a food box can be used with any food box id, and not only those for a certain dietary preference**.

6. updateOrderStatus from the supermarket and catering company classes use the same name, but they are different methods.

7. **The example tests in the skeleton for ClientImp classes are more sysem tests than unit tests, please don't use them as examples of unit tests.**

8. **For unit tests, it's best if in the @Before methods you use http requests (and not method calls) to set up the environment before testing**. Then, each test method should be about one single method, with several test methods for this method considering the different possible classes of inputs (also see self-assessment in the instructions).

9. For system tests, realise a use case through the call of all methods representing its steps inside one test method. Use an assert for each call (and not only for the last method call) within that test method so that you can catch where the procss fails. You should have a different test method for each possible scenario (i.e. way of going through the use case steps: main success scenario, with each and a combination of different alternative scenarios).

10. Here is an explanation of how to distinguish editing an order in place order vs later in system tests: https://piazza.com/class/kjta3xoiy3879?cid=678

11. **You should avoid clearing the .txt files on the server before re-running tests as our paths to these files may be different to the markers', and different OSs use different paths. We will clear these files manually if we are to run your tests several times, so that you don't need to do this.**

12. **The methods should return false when the server called as part of them returns false, with the exception of registration where the returns of the server ('registered new', 'already registered') should both result in the register methods returning true**. See https://piazza.com/class/kjta3xoiy3879?cid=458

13. **The methods should additionally return false in case preconditions for them don't hold**, e.g. a non-registered individual attempts to place an order, or had already placed an order within the last 7 days.

14. **Only use inline assertions or exceptions for input validation, i.e. checking whether inputs are provided and not null (where null is not accepted), and whether their format is correct (e.g. for CHI)**. Assertions will stop the processing, while some exceptions can allow the continuation of the processing and this can be exploited. See here: https://piazza.com/class/kjta3xoiy3879?cid=666 . **We only EXPECT you to use inline assertions, not exceptions**, but taking the extra effort for anything including exceptions can lead to an excellent mark, provided the rest of the solutions are very good,

15. **Here is advice as to how you can validate inputs representing CHI numbers**: https://piazza.com/class/kjta3xoiy3879?cid=638

16. We only expect JavaDoc in the classes mentioned in the instructions, but doing more including for tests can also help you get towards an excellent mark, provided everything else is good.