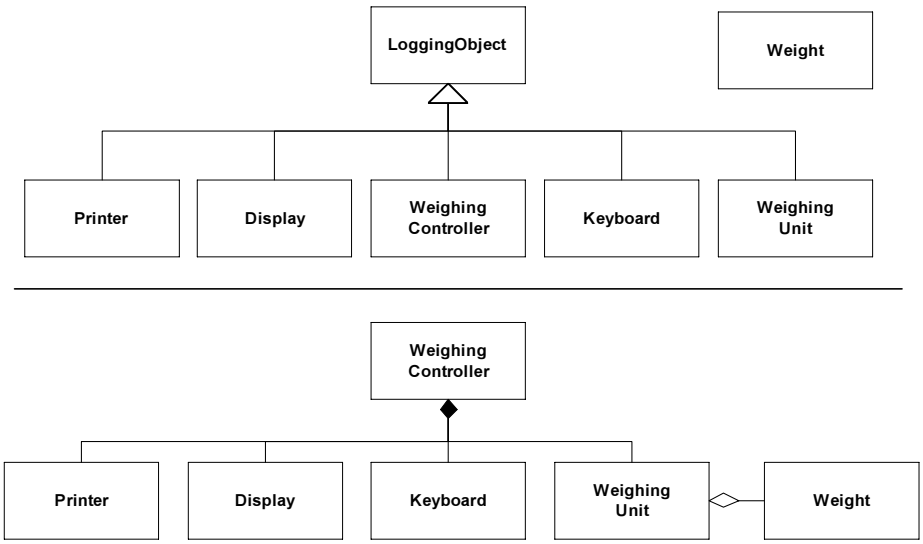
Lab Exercise 3.2:
C# Interfaces and Unit Testing

In this exercise, you will a bit more with C# interfaces and combine that with the creation of some tests with NUnit.

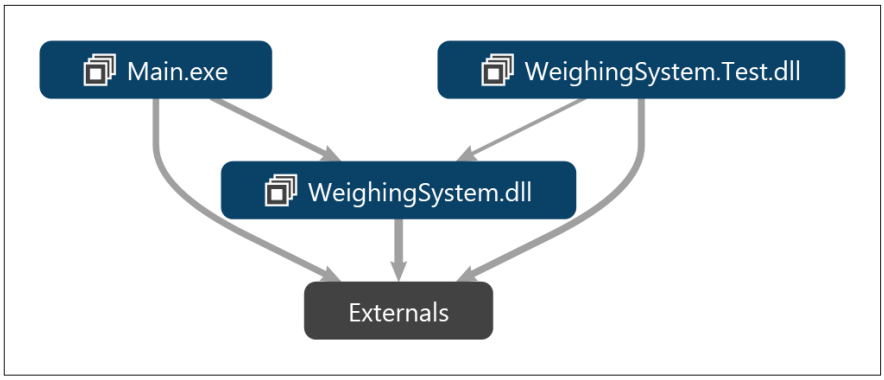
Exercise 3.2-A:

Fetch the "pre-made" solution called SWT_Ex3.2.zip from CampusNet. Unzip it where you want to work with it, and open it in Visual Studio. Have a look around the code … it's the same system (grocery weight) we have looked at earlier (although with some additional features to make it easy for each object to log to a file):



You can look at the sample code in the Main project to see a few examples of uses of some these classes. You can also run the Main-project (it's a console application) and take a look at the resulting logfile. It's called "testlog.txt" and it gets created in the folder …\SWT_Ex3.2\WeighingSystem\Main\bin\Debug.

The structure (dependency graph) of the Visual Studio Solution looks like this:



Exercise 3.2-B:

Define interfaces *`IWeight, IWeighingUnit, IKeyboard, IDisplay, IPrinter`* for each of the classes

- `Weight`
- `WeighingUnit`
- `Keyboard`
- `Display`
- `Printer`

so that they each implement an interface. The methods that should be part of the interface for each class have been marked like this:

```
// (*) IClassName
```

The purpose of this (sub-) exercise is to learn to spot interfaces and getting used to refactor code to use explicit interfaces.

Exercise 3.2-C:

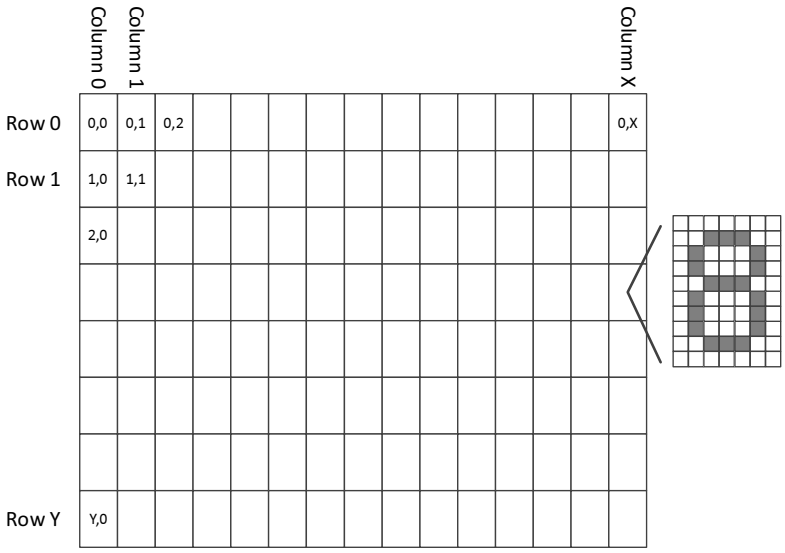
Add some unit tests of the WeighingUnit class, by adding test cases to the class WeighingUnitTest in the file WeighingUnitTest.cs. You can copy/paste code from previous exercises into both the Weight class, WeighingUnit class and WeighingUnitTest class (i.e. you can re-use your previous code and tests if you want to). But add a couple more tests using NUnit, just to get the hang of it.

(Exercise continues on next page)

Exercise 3.2-D (optional):

(Note: The exercises above are the important ones. But if you have made your way through those and feel ready for a slightly more complex test assignment, try your new skills on this sub-exercise.)

Now take a look at the class *Display*. The intention is that it will control an LCD dot-matrix display, where characters can be written in a matrix:



So the upper-leftmost cell has address (row=0,column=0). To write something in the display, one can place the "cursor" (the address where text is written) at the next spot to write, and simply write whatever text one wishes. The rules for cursor movement are these:

1. For each single character written at position (r,c), the cursor moves 1 cell to the right (to position(r,c+1)).
2. If the cursor is positioned at the right edge at position (r,c) when you write a character, the cursor wraps around to the next line at position (r+1,0).
3. If the cursor is positioned at the right edge, at the bottom line at position (r,c), when you write a character, the cursor wraps around to the top line at position (0,0).

You should now (the easy part):

1. Add code to the Display class so that it keeps track of the cursor when stuff are written in the display or the MoveCursor() method is used.
2. Write unit tests that verify that your cursor code works.

After that comes the hard part:

3. Add code to the Display class so that it keeps an "image" of what is currently shown at the display, by maintaining a two-dimensional char array, i.e:
   ```
   char[][] displaymemory = new char[_rows][_columns];
   ```
4. Add code to the Display class so that the EraseAll()-method works.
5. Write unit tests that verify that *displaymemory* always contains what you expect when you write text in the display. It's ok to refactor the Display class (for instance by adding methods) as long as your Display class conforms to the IDisplay interface.