Exercise for Applied Design Patterns in Fortran

Tiziano Müller

4. July 2020

The point of the following exercise is to show you how to use object oriented programming, inheritance and design patterns in such a way that all the decision logic is done at the beginning of the program and any necessary parametrization can be collected into their respective objects such that the main part of the program (where the actual work is done) is going to be a simple single `do while` loop without any conditional statements.

To illustrate this you are going to write a simple program which given the name of a string transformation (*lower* or *upper* for example) and the name of a file, reads the file and outputs it line by line with the given string transformation (converting all uppercase characters to their respective lowercase, resp. the other way round) applied.

Based on the examples given in the lecture write first an iterator `type` which opens a file when constructed and on each iteration returns a new line from the file. The `type` should at least have a *constructor* and a type-bound procedure which does the iteration work and can be used as condition for the `do while` loop. There are some pitfalls when it comes to handling strings and reading files line-by-line, but keep it simple for now.

Secondly, to avoid the explicit condition in the loop, write an `abstract` type with a single function. The function should take a string and return the transformed string.

The loop should then simply look like this:

```
do while (.NOT. iter%end())
    print *, trafo%transform(iter%line())
end do
```

Please note: there should be no module-level (*global*) variables. By initializing the *trafo* object to the correct derived class the decision will be made implicitly which implementation for the *transform* function is going to be called.

With this in hand, you could try to add additional functionality, such that the user can specify that the line iterator filters out empty lines completely, or that the transformation applies only to every nth character. All these extensions should then only affect the

respective objects and the initialization code and illustrate why function objects are more powerful than function pointers.

Another extension one could attempt are then the encapsulation of the loop itself (for example to only print the first n transformed lines, highlight a line if it contains a specific word, make the transformation context aware, etc.), at which point you'll probably want to pass along the initialized iterator and transformation function object to this new class instance.