Kata04: Data Munging - Questions

Jared F Bennatt

July 12, 2020

- Was the way you wrote the second program influenced by writing the first?

  Very much so. Since both were basically the same task just on different data and formatted slightly differently, the process of parsing each line was the same. Likewise finding the "optimal" record (day for weather and team for football), was essentially the same with a minor difference that one was a max and one was a min.

- To what extent did the design decisions you made when writing the original programs make it easier or harder to factor out common code?

  Going back to the first question, it made it fairly straightforward to combine the functionality into a single archetype. I wouldn't say it was easy to do, but I knew that since I wrote both programs almost identically, it should be possible to be done.

- Is factoring out as much common code as possible always a good thing? Did the readability of the programs suffer because of this requirement? How about the maintainability?

  I think the answer depends on what purpose the codebase is going to serve. The *readability* most definitely suffered. I think if you told someone the task and then showed them either of the first two programs, anyone with some basic knowledge of Java I/O would be able to understand what is going on. The problem with the third program, in terms of *readability*, is that there are more moving parts and it's less linear. So for instance, you have to understand *how do you read a table*? OK, the `DataParser` can read input and give you a table. Except the `DataParser` is `abstract` so you have to understand how to extend it to use it (I actually think looking at the two implementations, `WeatherDataParser` and `FootballDataParser` are most helpful for understanding how to use `DataParser`).

  On the other hand, the *maintainability* is greatly improved in the third program. I spent quite a bit of time on the weather program because I had to figure out what to do, how I was going to do it, write the code, go back and decide I should do it differently, run the code, debug the code, etc. The football program didn't take nearly as much time, although it still took some time because the data was slightly differently formatted, so I had to adapt what I had done for the weather data, and

then duplicate a lot of the code I wrote for the weather data. The DRY Fusion on the other hand, while took considerably longer to write than either of the first two, was *considerably* faster to adapt. So let's say I spent 3 hours writing the third program and getting the weather data to work. After the weather data worked, I then wrote the part for the football data, which probably took 10 minutes to do and get working because basically all I needed to do was tell the `FootballDataParser` what the labels were (and type), what delimiter to use and then I had to tell the `FootballAnalyzer` what measure to use and how to compare it.

So is factoring out common code a good thing? First, I think the answer is always yes, to some extent. You *certainly* do not want duplicated code (as in duplicated verbatim) because that's going to cause problems if that duplicated code ever needs to be refactored (because you'll have to remember the other place it's duplicated and make changes there as well). Is what I did with DRY Fusion always the best idea? I think no. I probably went a little overboard with abstraction and I think that makes this overkill for such a simple project and I think readability suffers more than it needs to. That is–*unless* we are expecting many new data sets and we want something capable of adapting to many different data sets that are formatted differently. Then I think what I did is very practical; yes there is an upfront cost in terms of learning the codebase, however once you are familiar, you can quickly adapt to many different data sets quickly.

In conclusion, if you know this is a "one time" (or maybe even two or three) time program that's going to be run for a very specific task and adapting to new data isn't foreseen to be an issue, I would say it's better to maintain readability. If however you know that the function of the program will be needed often and for many different types of input, then I think trying to abstract away (or factor out) as much functionality as possible is appropriate (and should be strived for).