The Strategy Pattern Through Person Matching

Dave Browning | A01256705

September 14, 2017

1 Introduction

The purpose of this report is to relay insights uncovered through the design, implementation, and testing of the person matching libraries. This will be done both verbally and through class diagrams.

2 Diagrams

2.1 Visual Studio Code Map

Figure 1 is not a standard UML diagram, but I have found it useful nonetheless. Green arrows represent inheritance, Pink arrows indicate that the block "calls" the other block, and Blue arrows indicate a file being written or read.

2.2 Class Diagram

Figure 2 is my class diagram. Though it is still not quite standard UML, it is much closer to what a person familiar to UML would be comfortable dealing with. The class diagram clearly displays use of the strategy pattern in Serializer, Outputter, and the SimpleMatcher classes.

3 Insights

3.1 Design

Initially as I was designing the person class, it quickly became clear that I hadn't understood the strategy pattern. I was too focused on the person object, and hadn't wrapped my mind around what a "behavior" was or how to abstract behaviors into classes which could be implemented.

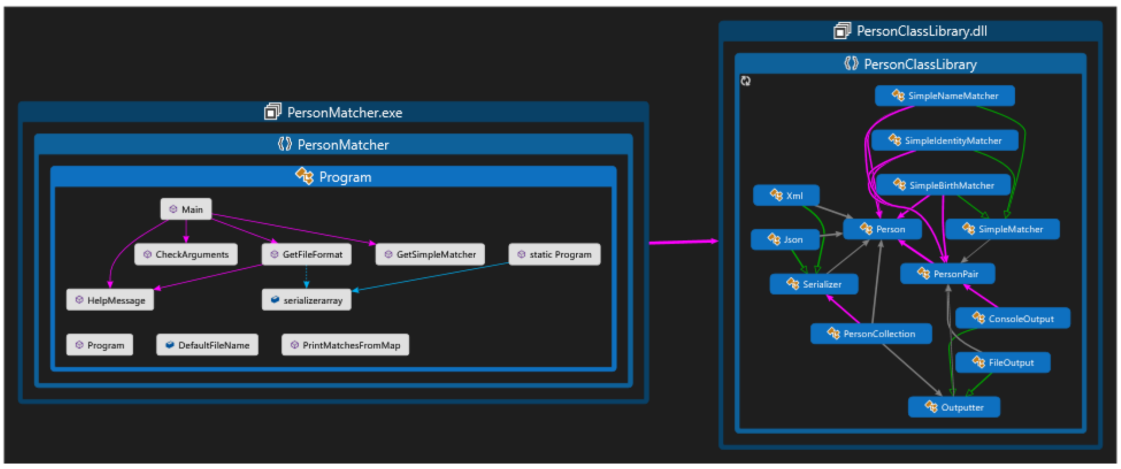

Figure 1: Code Map generated by Visual Studio

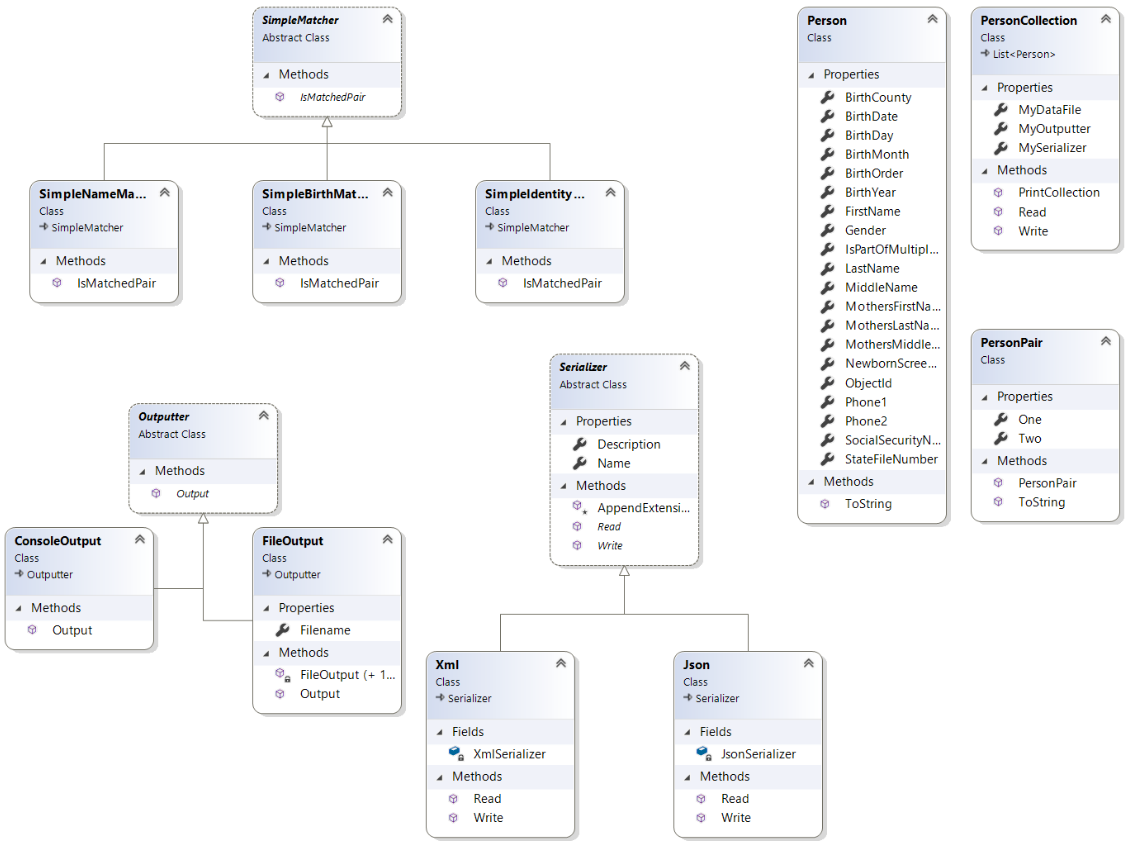

Figure 2: Class diagram generated by Visual Studio

I had a difficult time thinking through practical application of the strategy pattern when I first took a stab at this assignment. Through the implementation phase and through classes held last week, it became much more obvious what the strategy pattern is and how nicely it applies to this assignment.

I have also found tools for writing code diagrams to be very clunky and extremely difficult to maintain (mostly because of programmer laziness). I much prefer either paper/pencil or a white board for my early design diagrams.

3.2 Implementation

Each class period, I would take the sample code given in class and try to apply it to my assignment. Some pieces I've borrowed heavily from were the ImporterExporter class as well as the Json and XML ImporterExporters. This was the most important code sample for me firstly because I don't have much experience with Json, and secondly because of how well it lays out the strategy pattern (specifically from example 2).

Having finished the project, I find it amazing how effective the strategy pattern is when applied to the right type of problem. Regarding input and output specifically, it makes a lot of sense to be able to quickly and easily make a new instance of the abstract input and output classes. For example, if someone wanted to output to a web page, they could easily do so. Or even build an SQL query!

I believe it would make sense especially for applications that have the concept or notion of a Plug-In to use the strategy pattern for their Plug-In library.

3.3 Testing

It was during the initial testing of the person matcher program that I saw the need to condense the input checking into a single function as I have seen done in libraries like boost program_options.

The Visual Studio Test framework is fabulous, and it was enjoyable to write the few tests that

I have. I decided the most important piece to test was the Matcher classes, so I wrote unit tests for SimpleBirthMatcher, SimpleIdentityMatcher, and SimpleNameMatcher. I did find myself ending up with test failures because I had forgotten how I wrote the matching algorithm in the first place.

This is often one of the biggest pitfalls in testing, unless you write the tests and the test procedures extremely carefully, you may stumble upon tests failing not because the code doesn't work as designed, but because the test was not designed properly.