Jake Felzien
Object Oriented Programming CS5700
Clyde
9/14/2017

Homework 1 Write Up

For the first homework assignment we were asked to implement the strategy pattern and create a program that read in input from JSON and XML files, parsed the files, and conducted matches on the elements in the files based on differing strategies and report the results. As we can quickly observe the strategy class is a great candidate for this scenario as we have many instances where there will likely be many different strategies that could be implemented, and thus the strategy pattern allows us to to create a solution that is very extensible if we were to add additional strategies. At first glance we can see how differing strategies for input files would make a lot of sense, as well as different strategies for output as well.

To begin the assignment, I first implemented a generic Person class. This class is the most atomic piece and ultimately holds the pieces of data that we are most interested in. Within the Person class, a Person had several attributes. These attributes were observed to exist in the various data files that were given to us. The files given were XML and JSON files. From these files, I generated my first use of the strategy pattern. I created a generic abstract class called DataInput. This class would allow for extensibility if there was a need to add additional input file strategies in the future. So, naturally, from this abstract class I created two file input classes called JsonInput and XmlInput that inherited from the DataInput class. In the future, it would be incredibly easy to add additional input strategies. Now, in order to hold a group of people (that were read in from a given file) I generated a class titled PersonCollection that inherited from the generic C# List class and would hold a group of people. It made sense to give this PersonCollection a DataInput property, as this would allow for a collection to then read in the correct information for people, and then populate the collection.

The next major design decision was to create a Pair class that would hold two matching Person objects. Because multiple Pair[s] could exist it made sense to also generate a PairList class that was almost identical in functionality to the PersonCollection except that the PairList stored a list of Pair[s]. In my design I chose to create both an Output and a Matcher class. A PairList would have each of these classes.

The Output class was an abstract class that would provide my second use of the strategy pattern. Form this class I created two strategy classes: OutputConsole and OutputFile. These classes inherited from the Output class and contained the functionality to write the data in the Pairlist given the selected input criteria.

The Matcher class was also an abstract class that implemented the final main use of the strategy pattern. This class allowed for several algorithm strategies that would be used to match and generate the Person Pair[s]. I wrote three additional strategy matchers titled MatcherOne, MatcherTwo, and MatcherThree. These classes inherited from the Matcher class and each had a specific strategy on how to match individuals in a given PersonList and produce the given PairList.

I have included the matching UML diagram in the same directory as this report. In conclusion, we can quickly see the major advantages that come with the use of this design pattern. Procedural code would suffer greatly from change in a situation like the one written. Because of the use of the strategy pattern, our design has allowed for great extensibility in the future, if the occasion were to arise.