HW 1 – Person Matcher
Steven Proctor
CS 5700

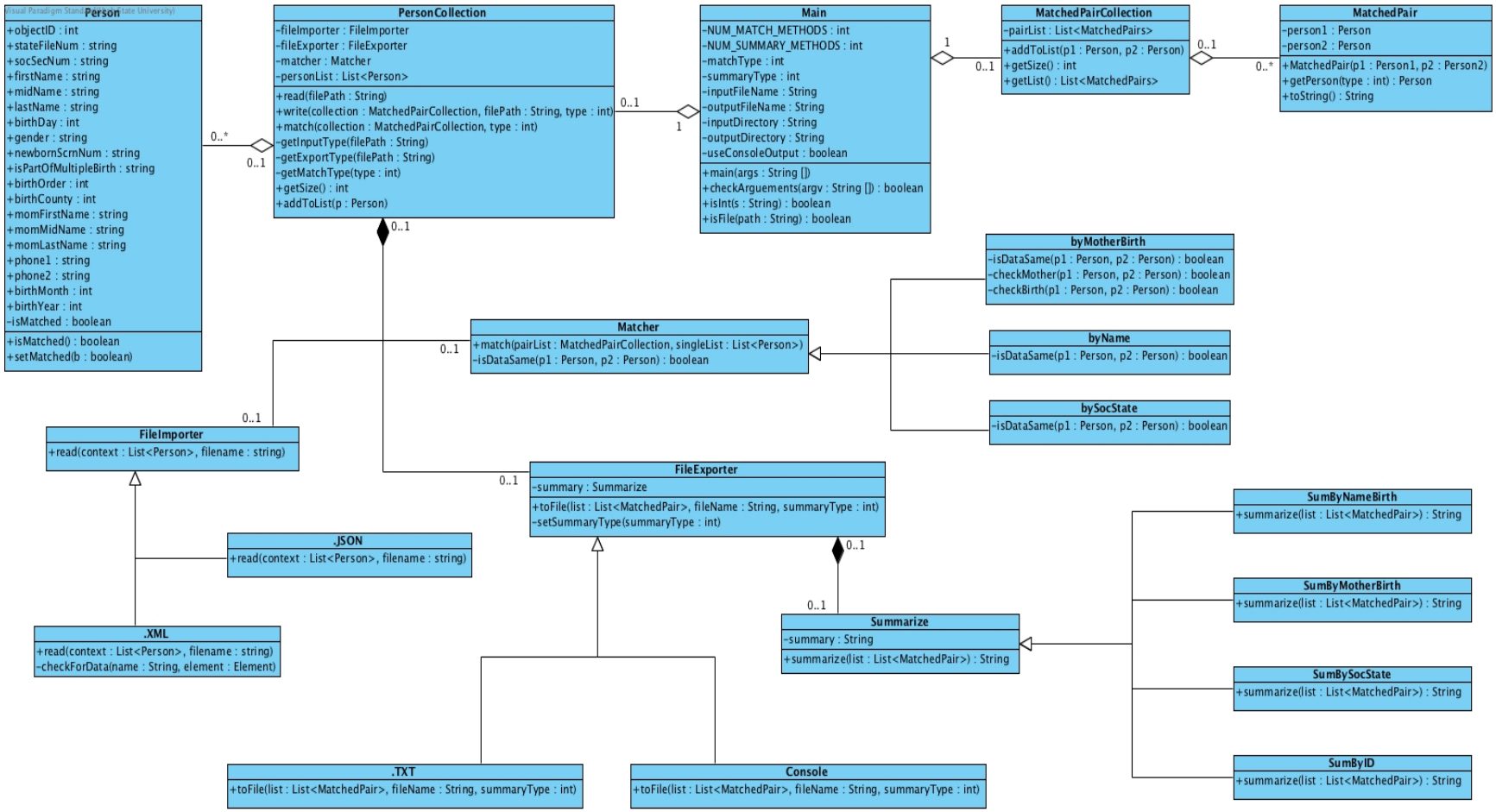
As stated in the problem description, the main focus of this assignment was to become more familiar with UML diagrams, unit testing, and implementation of the strategy pattern. I began the assignment with an attempt to create a UML diagram, but was quickly discouraged by a lack of understanding and unfamiliarity with the program. Instead I began to program the classes I felt most confident about: Person, PersonCollection, MatchedPair, and MatchedPairCollection. For the most part I feel these classes turned out well, despite the lack of preparation.

However, it became apparent as I began implementing the strategy pattern that a UML diagram would have helped me place and connect classes more logically. I used the strategy pattern for a FileImporter, FileExporter, and Summarize class. On their own I feel like they were designed well, just not implemented well. When I went back to update my UML diagram, I noticed that all of these strategies branched off of the PersonCollection class. On the diagram, it looks a little awkward, and if I were to start over I would have branched the strategies off of the Main class.

I would say that my design is not as loosely coupled as it should be. PersonCollection is a very "heavy" class, and every strategy implement and called through it. Instead I should have passed in a PersonCollection and a MatchedPairCollection into the strategy classes themselves, rather than attaching them to the PersonCollection class. This was reinforced when I went to create unit tests for some of the strategy methods. Obtaining all the data I needed for the tests proved to be more difficult than expected, and required that I add a few extra getter methods to existing classes. In the end, it worked without being too awkward, but could have been cleaner.

One other issue I encountered was finding the appropriate time and place to decide which strategy method to initialize. Because most of the classes were related to the PersonCollection class, the majority of the decision making occurred inside of said class. Again, this probably should have been done in Main, which became obvious when the UML was fully updated.

Despite the difficulties, this project provided an excellent introduction to the strategy pattern, unit testing, and UML diagrams. Now that I feel more comfortable reading and writing UML diagrams, I don't want to start coding a project of this magnitude unless I've written a full UML diagram. I also suspect that I will find myself writing better classes and methods if I think about simple, easy ways to test their results. It certainly would've helped me implement the strategies in this projects in more convenient places, had I thought about testing sooner. With the exception of the decision making, I feel very confident in using Strategy patterns and was thrilled with how easy it was to add functionality to my project when I used it. While these skills and strategies would have made this project much easier had I been familiar with them beforehand, the struggle I experienced trying to find ways to implement them (even inefficiently) helped me understand the need and importance of quality preparation befor you ever write a line of code.


Person
+objectID : int
+stateFileNum : string
+socSecNum : string
+firstName : string
+midName : string
+lastName : string
+birthDay : int
+gender : string
+newbornScrnNum : string
+isPartOfMultipleBirth : string
+birthOrder : int
+birthCounty : int
+momFirstName : string
+momMidName : string
+momLastName : string
+phone1 : string
+phone2 : string
+birthMonth : int
+birthYear : int
-isMatched : boolean
+isMatched() : boolean
+setMatched(b : boolean)
PersonCollection
-fileImporter : FileImporter
-fileExporter : FileExporter
-matcher : Matcher
-personList : List<Person>
+read(filePath : String)
+write(collection : MatchedPairCollection, filePath : String, type : int)
+match(collection : MatchedPairCollection, type : int)
-getInputType(filePath : String)
-getExportType(filePath : String)
-getMatchType(type : int)
+getSize() : int
+addToList(p : Person)
Main
-NUM_MATCH_METHODS : int
-NUM_SUMMARY_METHODS : int
-matchType : int
-summaryType : int
-inputFileName : String
-outputFileName : String
-inputDirectory : String
-outputDirectory : String
-useConsoleOutput : boolean
+main(args : String [])
+checkArguements(argv : String []) : boolean
+isInt(s : String) : boolean
+isFile(path : String) : boolean
MatchedPairCollection
-pairList : List<MatchedPairs>
+addToList(p1 : Person, p2 : Person)
+getSize() : int
+getList() : List<MatchedPairs>
MatchedPair
-person1 : Person
-person2 : Person
+MatchedPair(p1 : Person1, p2 : Person2)
+getPerson(type : int) : Person
+toString() : String
byMotherBirth
-isDataSame(p1 : Person, p2 : Person) : boolean
-checkMother(p1 : Person, p2 : Person) : boolean
-checkBirth(p1 : Person, p2 : Person) : boolean
Matcher
+match(pairList : MatchedPairCollection, singleList : List<Person>)
-isDataSame(p1 : Person, p2 : Person) : boolean
byName
-isDataSame(p1 : Person, p2 : Person) : boolean
bySocState
-isDataSame(p1 : Person, p2 : Person) : boolean
FileImporter
+read(context : List<Person>, filename : string)
FileExporter
-summary : Summarize
+toFile(list : List<MatchedPair>, fileName : String, summaryType : int)
-setSummaryType(summaryType : int)
.JSON
+read(context : List<Person>, filename : string)
.XML
+read(context : List<Person>, filename : string)
-checkForData(name : String, element : Element)
Summarize
-summary : String
+summarize(list : List<MatchedPair>) : String
SumByNameBirth
+summarize(list : List<MatchedPair>) : String
SumByMotherBirth
+summarize(list : List<MatchedPair>) : String
SumBySocState
+summarize(list : List<MatchedPair>) : String
SumByID
+summarize(list : List<MatchedPair>) : String
.TXT
+toFile(list : List<MatchedPair>, fileName : String, summaryType : int)
Console
+toFile(list : List<MatchedPair>, fileName : String, summaryType : int)