HW1 – Strategizing about Shapes

Estimated time: 8-12 hours

Objectives

- Become familiar with using UML class diagrams to describe not trivial object structures.
- Become familiar with the Strategy pattern
- Become familiar with basic unit testing techniques

Overview

In this assignment, you will look for opportunities to apply the strategy pattern while building a program that reads person objects from an input file, computes which pairs of records represent that same real person, and writes out the results to console and to an output file.

Instructions

Your program needs to work person objects that contain personal identifying information and try to match them. A person can be either an adult or a child. Table 1 shows the data properties that may contained in person object. Tables 2 and 3 should additional properties for child and adult objects.

Table 1 – Person Properties

Property Name	Data Type	Description
ObjectId	Integer	An unique identifier for the object, not the person
StateFileNumber	String	A state file number -- highly unique, but only available to people born in the Utah and some bad value may exit
SocialSecurityNumber	String	A federal tax identifier – highly unique, but only occasionally available and prone to error
FirstName	String	The person's first name
MiddleName	String	The person's middle name
LastName	String	The person's last name
BirthYear	Integer	The year of the person's birth date
BirthMonth	Integer	The month of the person's birth date
BirthDay	Integer	The day of the person's birth date
Gender	String	The person's gender -- "F" or "M", blank, bad data

Table 2 – Additional Properties for Children

Property Name	Data Type	Description
NewbornScreeningNumber	String	A number assigned to babies born in Utah – highly unique, but only available for children born in medical facilities
IsPartOfMultipleBirth	String	The person's was born with a twin, triplet, etc. – "T" or "F"
BirthOrder	Integer	If the person was part of a multiple birth, then this is person's birth order relative to the others. For example, The first of a set of twins born would have a value of 1 and the second would have a value of 2
BirthCounty	String	The county in which the person was born
MotherFirstName	String	The person's mother's first name
MotherMiddleName	String	The person's mother's middle name
MotherLastName	String	The person's mother's last name

Table 3 – Additional Properties for Adults

Property Name	Data Type	Description
Phone1	String	A phone number for the person
Phone2	String	A phone number for the person

Any field may contain a null or bad data. A null for any property value means that the property is not known for the person. A "" or "0" are valid values and are different than a null. For example, if a person is believed not to have middle name, the *MiddleName* property will be a "", not a null. A null would mean that is no information on the middle name.

Your program must be able to read a set of person records (child or adult) in input file formatted in either XML or JSON. It is possible that other formats may be required in the future, so your design needs to anticipate that. The type of each records will be encoded in the XML or JSON tags.

After your programs creates a set of person objects from the data in the input data, it needs to try to match each person with every other person. However, how it does this for a given pair needs to depend on what data is available in the records of that pair. Here are some ideas on how the matching could be done:

- If both records include either a StateFileName, SocialSecurity, or NewbornScreenNumber and a birth year, month, day, then compare just those fields. If there are any discrepancies, then classify the pair as not matching
- If both records include either a StateFileName, SocialSecurity, or NewbornScreenNumber and two or more name fields, then compare those fields. If there are any discrepancies, then classify the pair as not matching

- If both records have complete name, gender, birth date, and some mother information, then compare those fields. If there are any discrepancies, then classify the pair as not matching

You are free to decide how the matching will be done, but you probably will want implement at least 5 different ways of doing it to accommodate a range of possible situations.

For each pair, the output of each match computation should be simply "the pair matches" or "the pair doesn't match". Keep track of the matching pairs using their ObjectId's.

Note that it is possible for a child record to match an adult record, because children become adults over time (usually).

After classifying each pair as matching or not matching, your programs needs to list the matching pairs to the console and to an output in the following format:

```
Matching pairs:
        (a1,b1)
        (a2,b2)
        (a3,b3)
        …
```

where a_1, …, a_n and b_1, …, b_n are ObjectId's and (a_i, b_i) are matched pairs.

Your program should accept as input parameters type of input, the name of input file, and the output file name.

Here is a list of steps or tasks to complete as part of the assignment:

1. Model the structure of your solution using UML class diagrams.
2. Model interesting inter-object behaviors in your solution using UML interact diagrams
3. Be sure to design your program, to gracefully handle bad data
4. Implement your program
5. Test some on the most interesting methods in your program using executable unit test cases

The instructor will provide some sample data, but you will probably want to create more on your own, particularly as you implement meaning unit test cases.

Submission Instructions

Zip up your entire solution, including test cases and sample input files, in an archive file called CS5700_hw1_<*fullname*>.zip, where fullname is your first and last names. Then, submit the zip file to the Canvas system.

Grading Criteria

Criteria	**Max Points**
A clear and concise structural model using UML Class Diagrams	20
A model of interesting and clear inter-object behaviors using UML Interaction Diagrams	20
A working implement, with good encapsulation, abstractions, inheritance, and aggregation, and appropriate use of the strategy pattern	50
Meaningful, executable unit test cases	30