CSC 250 – Project #2
Fall 2014

Goals: Gain experience working with sorting arrays of polymorphic data.

Part 1: Project Summary (Due: 5pm Oct. 23rd; Drop dead date: 5pm Oct. 24th)

Implement a program to read in a large amount of personal data for customers of a factitious gym. Then, sort the data according to the criteria defined below. Finally, write the resulting sorted data out to an output file.

Part 2: Processing

There are two types of customers of **squale.com** (a social-network where people exercise (a gym)): those that have registered to use our facilities and those that have actually become "gold members". We need to organize our large amount of data for our reports, so we hired you.

Read in the `personalData.txt` file (located in the data sources directory shown below). Your program should use the `java.net.URL` class and `java.util.Scanner` class to read directly from the web server. (That is, do NOT download the file and then have your program open and read from the file directly.) Data contained in the input file is provided as a series of rows (one row per member, with a single header row as the first line of the input file). Within each row, there are a number of data fields, each separated by a tab character (`\t`). Each row of data includes the following columns of data:

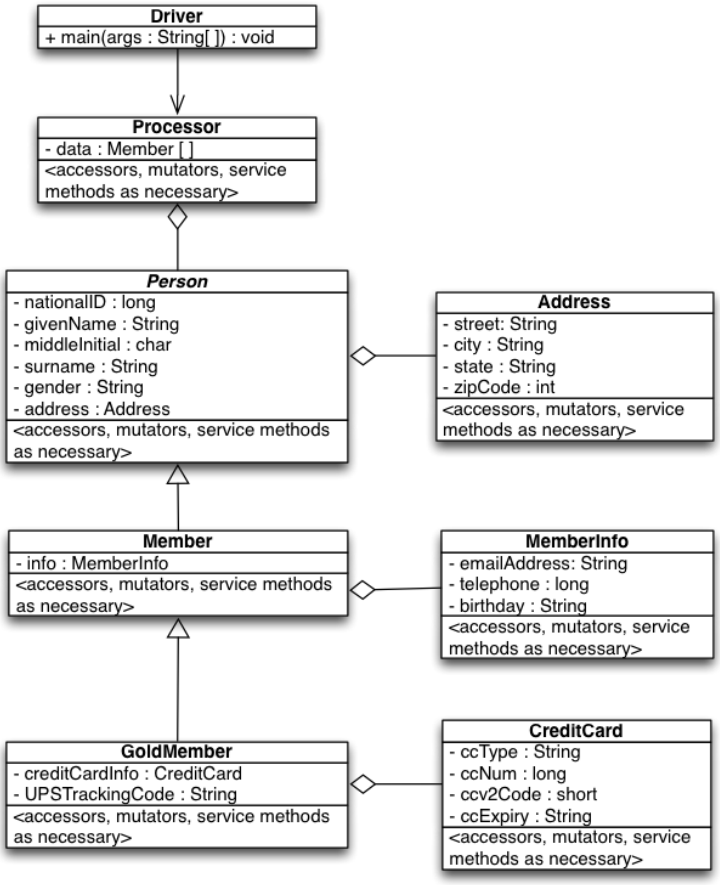
- Number
- Gender
- GivenName
- MiddleInitial
- Surname
- StreetAddress
- City
- State
- ZipCode
- EmailAddress
- TelephoneNumber
- NationalID
- Birthday
- CCType
- CCNumber
- CVV2 (card verification value)
- CCExpires
- UPS Tracking Code

Most of this data can be treated as String data in your objects. However, you should treat the record number, and zip code number as integers. The credit card number should be represented as a long value, and the cvv2 code should be a short. Your program should include an array of polymorphic `Member` and `GoldMember` objects, which are the contents of the data file. Those members without credit card information are represented as `Member` objects; those with credit card and shipping data are both `Member` and `GoldMember`. (Hint: `GoldMember` is a subclass of `Member`). Include an abstract `Person` class in the hierarchy that represents a person's national ID, given name, middle initial, surname, gender, and address data (only!).

Note that people who do not make a purchase (`Member`) do not have data in the final five fields (cctype, ccnumber, cvv2, ccexpires, ups code) of the input file. The storage array should not be of type `Comparable` (see below), though the array objects will be. There are 40,000 people listed in the input file, so you can hard-code your array to that size. Do not count the number of lines in the input file.

The following data for each member should be encapsulated together into their own object and aggregated accordingly from the `Person` object: street address, city, state, and zip code. Make this an `Address` object. Similarly, the email address, telephone number, and birthday data fields should be encapsulated as a `MemberInfo` object and aggregated in a `Member` object. Credit card information (cc type, cc number, ccv2, cc expires) should be encapsulated as a `CreditCard` object and aggregated within the `GoldMember` class. The

image below shows some of desired structure (note that methods are not included in the diagram and are left up to you to design/create). Note that the diagram does not show the inclusion of the Comparable class.



Once read in, sort the data using the **`java.util.Arrays`** class. (There's a method to sort **`Objects`** that implement the **`Comparable`** interface present in the class.) Each data item should be sorted based on the following criteria:

- sort on the last name (ascending)
- then on the state abbreviation (descending), and finally
- then on the street address (descending)

After the sorting is completed, open a new output file named **`sortedData.txt`**. Write your sorted data to this file, one record per line, in a similar manner as the input file (I strongly suggest a tab-delimited output file). All data should be appropriately formatted based on conventional rules. That is, some zip codes may need to be prefixed with a 0 (see Rhode Island). Use the **`DecimalFormat`** class to correctly format your output. Check all of the fields for this type of error; we see others that have "issues".

Part 3: Resources

- The Java API documentation is located at: http://download.oracle.com/javase/6/docs/api/index.html
- The input file used for this project is located on a server in the Amazon.com cloud. You can find the file at the following address: https://s3.amazonaws.com/depasquale/datasets/personalData.txt
- Since the full "input file" is over 6.4 MB in size, you can use a smaller "test" input file for your development purposes. Your final submission should read the "full" input file listed above. The test file is located at: https://s3.amazonaws.com/depasquale/datasets/personalData2.txt

Part 4: Specifics for the delivery of this project:

1. Create and fully test a small ANT build file (named build.xml) as shown in class. The build file should contain a target that compiles the project (name the target *compile*) and a target that cleans the .class and output file from the build area (name that target *clean*).
2. To submit your project, jar up the source code file(s) and the build.xml file for the project into a jar file named userid.jar, where userid is your TCNJ userid (used for your Unix account and your email address). The task of organizing and creating the jar file for each deliverable will be up to you. You should always check your work, as this file is what I will be grading. You want to ensure that only the files I requested are contained in the jar file. This means nothing else (no data files, no .class files, no .java~ (backup) files, etc.)

   You may submit the project any number of times until the due date. Only the last submission will be downloaded, examined, and graded by me. Late points will be deducted for projects turned in starting 1 minute after the due date (that's why they are called due dates!) Following the drop-dead date for a project, no solutions will be graded for the project. At such a point in time, you will receive a zero grade for the work if you have not uploaded a solution to Canvas.
3. Late points will be deducted for projects turned in starting 1 minute after the due date (that's why they are called due dates!) Following the drop-dead date for a project, no solutions will be graded for the project. At such a point in time, you will receive a zero grade for the work if you have not uploaded a solution to Canvas.

Part 5: Advice and Disclaimer

Plan your schoolwork and life accordingly. Many students don't adequately plan their work schedules and attempt to finish programs at the last minute. Doing so usually introduces bugs/problems into the solution. Consider how much time you will require to adequately test your solution for boundary cases and bugs. Failure to plan on your part does not constitute an emergency on my part!

I do not give individual extensions for projects, and rarely give class extensions for work. You have nearly a week and a half to complete this project. I suggest you start your design as soon as possible. Consider what you need to build to solve the problem at hand. Programming involves designing a solution, implementing your solution, and testing your solution.

Part 6: Do You Need Help?

Are you stuck? Ask yourself if you have planned the design of your project. Have you asked questions in class? Did you attend the lectures, and labs? Have you come to see me? Like a textbook or web site, I am a resource that I expect you to use throughout the course of the semester. If you are stuck with your project, I strongly suggest you make an appointment to come see me in person. Generally, I can offer some guidance or help to get you back on the path.

When you come to see me, plan on bringing your source code to me on a memory stick or laptop. Do not email me your source code, I do not provide email-based help for coursework. I prefer that you make time to visit me in my office where we can sit down and discuss your source code together. I can ask you questions and you can tell me why you coded something the way you did. We can use my whiteboard to draw large diagrams and talk about good solutions and designs. This is not something that can be done easily through email.

Part 7: Code Commenting

Be absolutely sure that you refer to and follow the code commenting guideline posted for all of my classes. Detailed comments should appear at the start of each source code file, each class, and method. For blocks of contiguous code in methods, comments should be placed above the code blocks. Also, each instance variable should be named appropriately (no one letter variable names) and commented. While this is most of what I'm looking for, it's not all of it. Refer to the code commenting guidelines for the correct listing of comments that are required for all projects.