Computing and Algorithms I

Programming Project 4 (75 points)

Due at 11:59 pm, Jun 16 (Tue), 2020

CS-101 **Spring 2020**

This project mainly focuses on sorting arrays of objects and practice inheritance and polymorphism of classes. In this project, you will write a program to prepare reports for a small database on people.

Class Hierarchy

You will define a class named *Person*. *Person* will have two subclasses, *Student* and *Employee*. *Student* will have two subclasses, *UndergraduateStudent* and *GraduateStudent*. Likewise, *Employee* will have two subclasses, *Faculty* and *Staff*.

The Database

The database consists of records of four types of person: undergraduate students, graduate students, faculty, and staff. Each of the four subclasses *UndergraduateStudent*, *GraduateStudent*, *Faculty*, and *Staff* will have six data fields (with common data fields declared in a superclass). Each data field is declared as a specific data type with a particular restriction on its value (for which you should perform validation in its mutator method and the constructor) as follows:

- *UndergraduateStudent* has the following six data fields:
  1. first name: a String
  2. last name: a String
  3. e-mail address: a String
  4. student ID: an `int` (a number with 9 digits, between 100,000,000 and 999,999,999)
  5. GPA: a `double` (a number between 0 and 4, inclusive)
  6. status: an `int` (either 1 for freshman, 2 for sophomore, 3 for junior, or 4 for senior)
- *GraduateStudent* has the following six data fields:
  1. first name: a String
  2. last name: a String
  3. e-mail address: a String
  4. student ID: an `int` (a number with 9 digits, between 100,000,000 and 999,999,999)
  5. GPA: a `double` (a number between 0 and 4, inclusive)
  6. level: a `char` (either 'm' for master or 'd' for doctoral)
- *Faculty* has the following six data fields:
  1. first name: a String
  2. last name: a String
  3. e-mail address: a String
  4. office: a String
  5. hiring year: an `int` (a number between 1980 and 2020, inclusive)

6. rank: a `char` (either 't' for assistant professor, 'c' for associate professor, 'p' for full professor, 'l' for lecturer, or 'r' for researcher)

- *Staff* has the following six data fields:
  1. first name: a String
  2. last name: a String
  3. e-mail address: a String
  4. office: a String
  5. hiring year: an `int` (a number between 1980 and 2020, inclusive)
  6. title: a String

Input

The entire database will come from an input `.txt` file. The data is stored in the file one person per line, with a single '#' character separating each data field. Note that no data field has a '#' in it.

- The very first character of each line will be either 'U' (for *UndergraduateStudent*), 'G' (for *GraduateStudent*), 'F' (for *Faculty*), or 'S' (for *Staff*), followed by a '#', followed by the data fields, in the order given above for each type of person.
- There is no '#' after the last data field in each line.
- The beginning character of each input line will not be stored in any instance variable of any of the objects defined in the hierarchy.

Here is a list of example data stored in the input file (say, `people.txt`):

```
U#John#Doe#jdoe@kettering.edu#123456789#3.25#2
S#Maria#Allen#mallen@kettering.edu#123 Main Building#2016#Registrar
G#Ella#Pettit#epettit@kettering.edu#987654321#3.30#m
F#James#Neal#jneal@kettering.edu#321 Main Building#2005#c
```

Output

You will sort the database or part of the database using different sort keys as follows:

1. Sort the entire database by last name in ascending order.
2. Sort all the students (undergraduate and graduate students) by GPA in descending order.
3. Sort all the employees (faculty and staff) by hiring year in descending order.
4. Sort all the undergraduate students by status in ascending order.

Each of these four sorted lists of data should be written to an output `.txt` file, so there will be four output `.txt` files in total, and each file will consist of:

1. Your project title and your name.
2. A header describing the data in this file (for example: "All the students sorted by GPA in descending order").
3. The sorted list of data.

There will be NO print statements in any of the classes. An easy way to do the printing is to have the `toString` method of each subclass return the *String* to be printed. For each object, all its data fields need to be printed in an easy to read format by following these requirements:

- Print the type of object (type of person, such as "Graduate Student") in the very first line.
- Each field will be printed with a label at the beginning of the line, with the line tabbed in from the type of object on the first line.
- For each undergraduate student, the status should be output as "Freshman" (for input 1), or "Sophomore" (for input 2), or "Junior" (for input 3), or "Senior" (for input 4).
- For each graduate student, the level should be output as "Master" (for input 'm') or "Doctoral" (for input 'd').
- For each faculty, the rank should be output as "Assistant Professor" (for input 't'), or "Associate Professor" (for input 'c'), or "Full Professor" (for input 'p'), or "Lecturer" (for input 'l'), or "Researcher" (for input 'r').
- Print a blank line between each object output.

Here is a list of example data (sorted by first name) printed to the output file, with `people.txt` shown on page 2 being the input file:

```
Graduate Student
   First name: Ella
   Last name: Pettit
   e-mail address: epettit@kettering.edu
   Student ID: 987654321
   GPA: 3.30
   Level: master

Faculty
   First name: James
   Last name: Neal
   e-mail address: jneal@kettering.edu
   Office: 321 Main Building
   Hiring year: 2005
   Rank: Associate Professor

Undergraduate Student
   First name: John
   Last name: Doe
   e-mail address: jdoe@kettering.edu
   Student ID: 123456789
   GPA: 3.25
   Status: Sophomore

Staff
   First name: Maria
   Last name: Allen
   e-mail address: mallen@kettering.edu
   Office: 123 Main Building
   Hiring year: 2016
   Title: Registrar
```

Program Requirements

In total, you will write eight classes (seven service classes: *Person*, *Student*, *Employee*, *UndergraduateStudent*, *GraduateStudent*, *Faculty*, *Staff*, and one client class), each saved in a `.java` file.

1. Add data fields to each of the seven classes properly. Note that you should not add an array as a data field for any class.
2. Add an accessor method and a mutator method for each instance variable. In a mutator method for each non-String instance variable, perform validation according to its value restriction described in the "The Database" part on page 1.
3. For each of the four subclasses *UndergraduateStudent*, *GraduateStudent*, *Faculty*, and *Staff*, add exactly one constructor with six parameters which will be the values constructed from the corresponding input data fields. Perform validation for each non-String instance variable according to its value restriction.
4. There will be four sorting methods in total.
   - Add each one to the corresponding class (for example, the sorting method used to sort all the students by GPA should be added to the *Student* class).
   - Each sorting method takes an array of *Person* objects as the only parameter and returns a sorted array of *Person* objects.
   - Use any of the three sorting algorithms, Selection Sort, Insertion Sort, and Bubble Sort, to code your sorting methods.
5. Add an overriding `toString()` method in each class, which will return a *String* for the corresponding object following the format requirements described on page 3.
6. In your client program, you should create ONLY array(s) of *Person* objects. Do NOT declare any array of *Student*, *Employee*, *UndergraduateStudent*, *GraduateStudent*, *Faculty*, or *Staff* objects.
7. Create an input `.txt` file with at least 16 correct lines of input data (a minimum of 4 for each type of person). For each data field which will be a sort key, make sure the values for different objects are also different.
8. To break an input line into its fields, you can use a *Scanner* object on the *String* of the input line and then set the delimiter appropriately.
9. Your program should run for any valid input of 1 to 100 people (without editing any part of your code).

UML Class and Object Diagram Requirements

For this project, you are also required to create a document containing the UML class diagram for all the seven classes and the object diagram all the objects instantiated in your client class. If you are using a software application other than Microsoft Word or Excel, please convert it to a `.pdf` file for submission.

Follow the UML notation introduced at the end of Chapter 7 slides. Include arrows to indicate the relationships between subclasses and superclasses. For an object created in your client program, if you have ever changed the data values, please create one table for each different set of data values.

Style and Documentation Requirements

- **Names of classes, variables, and constants:**
  - A meaningful class name is an important part of the style. It should describe the purpose of the class.
  - Likewise, give meaningful names to variables and constants as well.
  - All class, variable, and constant names should follow Java's naming conventions.
- **Comments:**
  - For each class, include a block comment at the beginning to specify the purpose of the class and identify the programmer.
  - Add proper line/block comments to each part of your code as well.
  - Add a well-formatted data table for each class (except for your client class) and each method defined in every class with declared variables or constants, including the `main` method in the client class. Each data table should be added immediately before the class header or the method header. It will contain all the declared variables and constants (including parameters declared in the method header), and for each variable or constant, please include the name, data type, and purpose of it.
- **Blocks:** Please use either the next-line style or the end-of-line style in your program.
- **Use white space** (indentation, blank lines) properly to clearly show the program structure. For indentation, please indent each subcomponent or statement at least two spaces more than the construct within which it is nested.

Submission

Please compile and run your program several times, with different text files, to check the correctness before submitting your work.

For this project, you will need to submit one single `.zip` file named `<YourLastNameProject4>.zip` to Blackboard. It will contain

- Eight `.java` files (the seven classes and the client program);
- One input `.txt` file with at least 16 correct lines of input data;
- Four output `.txt` files of the sorted database or part of the database in four different ways as required;
- One document containing the UML class and object diagram.

Grading

The total maximum score for this project is 75 points. If your code does not compile, your grade for this project will be 0.

1. **Correctness – 20 points:** You should correctly create the input file and your program should correctly generate four output files (for any valid input file) by following all the formatting requirements.

2. **The Code – 30 points:** All your eight classes should be coded by following all the program requirements.
3. **Style and Documentation – 10 points:** Each of your classes should follow all the style and documentation requirements. You should also create a well-formatted data table for each class and each method following all the requirements.
4. **UML Class and Object Diagram – 8 points:** You should create the UML class and object diagram following the requirements.
5. **Submission – 7 points**: You must submit all the files with the required content.

You will still be able to submit your project assignment three times to Blackboard in case you want to make changes after the first submission, but I will ONLY go over and grade your last submission. You are only allowed to submit it late by at most 24 hours, but with a 15% penalty.