Comp165 Major Programming Assignment 2 Spring 2017

Implement the following classes that will be used to track personnel at a university. Use data encapsulation to hide implementation details from client classes. Also, make sure you review the coding standards posted at the beginning of the semester. Programs that does not adhere to the standard will be severely penalized.

Common Behaviors:

Each class will have mutator and accessor methods for each property except ArrayLists. ArrayList properties will have num, get, set, add and delete behaviors as described in MP1. Date properties will be set with a String and return Strings with getter methods but will be represented internally by Date objects.

Class Descriptions:

Person:

This class includes the following **protected** properties: firstname:String, lastname:String, studentId:String, birthDate:Date, phoneNumber:String and represents a generic person. In addition to the mutator and assessors, include a no-arg constructor that sets the properties to reasonable default values and a constructor with parameters for each property – Person(fname:String, lname:String, sId:String, birthDate:String, phoneNumber:String). Note: your constructor code should use the String format of the date "mm/dd/yyyy" to initialize the instance variable that is of type Date. Include a toString() method returns each field in the class separated by a comma:

Kelvin, Bryant, 8574945, 9/26/1964, 336-444-4444

Student:

This class inherits from Person and also includes these protected properties: major:String, creditHours:int (total credit hours, not semester credit hours), gpa:float, classSchedule:ArrayList<String> (an ArrayList of course numbers – e.g. {"COMP365", "MATH201", "GEEN345"}). In addition to a no-arg constructor, include a constructor that includes one parameter for each field of Person and the fields of Student (except classSchedule). The toString() method returns all the properties of Person and Student in the following format:

```
Kelvin, Bryant, 8574945, 9/26/1964, 336-444-4444
Computer Science, 65, 3.52
COMP365, MATH201, GEEN345
```

GraduateStudent:

This class inherits from Student and also includes these protected properties: thesis:String, concentration:String, assistanceType:String (either RA or TA). The class just has a no-arg constructor. The toString() method should return everything from the toString() of Student plus the additional GraduateStudent properties separated by commas:

```
Kelvin, Bryant, 8574945, 9/26/1964, 336-444-4444
Computer Science, 65, 3.52
COMP365, MATH201, GEEN345
Generic Register Allocation on CISC Architectures, Software Engineeering, RA
```

Employee:

This class inherits from Person and includes these additional protected properties: hireDate:Date, status:String (FT for full-time, PT for part-time), department:String. Include a no-arg constructor only. The toString() returns the Person properties followed by the additional Employee properties:

Kelvin, Bryant, 8574945, 9/26/1964, 336-444-4444
08/15/2007, FT, Computer Science

Faculty:

Extends Employee and includes these additional protected properties: rank:String (Assistant, Associate, Full), researchArea:String, currentCourses:ArrayList<String> (the equivalent of classSchedule in the Student class). Include a no-arg constructor only. Here is the toString() format:

Kelvin, Bryant, 8574945, 9/26/1964, 336-444-4444
08/15/2007, FT, Computer Science
Associate, High Performance Computing
COMP365, GEEN165

Staff:

Inherits from Employee and includes these additional protected properties: jobTitle:String, supervisor:String, careerBand:String (Contributing, Journey, Advanced).
toString() format:

Kelvin, Bryant, 8574945, 9/26/1964, 336-444-4444
08/15/2007, FT, Computer Science
Director of Gaming and Fun Activities, Patrice Bryant, Advanced

Department:

Department	
-name:String -location:String -employees:ArrayList<Employee> -students:ArrayList<Student>	Department name (e.g. Computer Science) Building name and office number All the staff and faculty All the undergraduate and graduate students
+Department() +Department(name:String, location:String) +readDepartment(filename : String) : void +writeDepartment(filename : String) : void +toString():String +//Behaviors for the employees and students	Use default values Instantiate with name and location Append file contents to the Department. Write out the department in input file format. Return a String representation of the department in the input file format.

Input File Format

```
Department Name
Location
F/S/U/G   //Faculty Staff Student GradStudent
//Refer to the toString() format of each class for the other properties
F/S/U/G   //Faculty Staff Student GradStudent
//Refer to the toString() format of each class for the other properties
...

F/S/U/G   //Faculty Staff Student GradStudent
//Refer to the toString() format of each class for the other properties
```

Grading

Level 1: (15%)

Create UML diagrams for the Person, Student, GraduateStudent, Employee, Faculty and Staff classes. Also, create a UML Domain model showing the relationships between all the classes (including the Department class).

Level 2: (65%)

Implement each of the described classes except for the Department class (it is part of Level 3). In a separate file, create a main method that will instantiate each of the classes and call their respective toString() methods. Send your results to a JOptionPane.

Level 3: (100%)

Create the Department class in a separate file. Modify the main method so that it obtains an input file from the command-line arguments. If there is no command-line argument then start with an empty Department. Use the input file to populate the Department via the readDepartment method. Create extra Student and Employee objects and add them to the department. Prompt the user for an output file name and call the writeDepartment method to save the updated department contents.

Extra Credit (10%)

Implement a GUI using the Netbeans GUI builder. Details will be given in an Addendum document.