EECS 2030: Lab 5

(1.5 % of the final grade, may be done in groups of up to three students)

Motivation

The purpose of this lab is to design and implement a simple hierarchy of classes representing people.

Part 1: Getting Started

Download a zip file containing the Lab 5 Eclipse project.

Import the project into Eclipse by doing the following:

1. Under the **File** menu choose **Import...**
2. Under **General** choose **Existing Projects into Workspace** and press **Next**
3. Click the **Select archive file** radio button, and click the **Browse...** button.
4. In the file browser that appears, navigate to your download directory (exactly where this is depends on what computer you working on; on the lab computers the file will probably appear in your home directory)
5. Select the file **lab5.zip** and click **OK**
6. Click **Finish**.

Explore the existing methods and the test cases, try to understand the purpose of each line. For example, what parameters the constructors are expected to take, what the output should be, and why some operation(s) should be disallowed.

Part 2: Design and Implementation

The following classes are included:

Person

Student

GraduateStudent

Employee

HourlyEmployee

SalariedEmployee

and

PeopleClassesTester

You are to design the class hierarchy and the API in such a way that the main method in the last class works correctly and produces a correct output, and the classes satisfy the following:

- any person must have a name and a date of birth; once set, they cannot be changed.
- any student has a student number; it can never be changed and may include digits and hyphens

- a graduate student has a supervisor. The supervisor can be changed and her or his name can be queried;
- any employee has an employee number; it can never be changed and may include digits and hyphens. Also, an employee must be either hourly or on a salary (but not both).
- hourly employees have an hourly rate, e.g., in Canadian dollars per 3600 seconds, which may be changed and may be queried.
- salaried employees have a salary, e.g., in Canadian dollars per 12-month period, which may be changed and may be queried.
- the classes (other than the tester) should override the **Object**'s **toString** method in order to produce the desired output from the main method provided.

In your implementation, use calls to superclasses' constructors and methods as much as possible to avoid code duplication. Most constructors and methods will require a couple of lines of code (1–3). It is possible that you will need to make some class(-es) or method(-s) abstract. The main constraint is the public constructors and the methods are available for the tester class to function.

The tester class is provided mainly to illustrate the intended behaviour. Feel free to modify it any way you desire during the development; only the first 6 classes will be considered during the grading process. The output of the original tester:

```
(Testing getDob()) Thu Jul 16 16:56:10 EDT 2020
Name: Person One
DOB: Thu Jul 16 16:56:10 EDT 2020

Student
Name: Student One
DOB: Thu Jul 16 16:56:10 EDT 2020
Student Number: 123456

(Testing getSupervisor()) Supervisor One
Graduate Student
Name: Grad One
DOB: Thu Jul 16 16:56:10 EDT 2020
Student Number: 123457
Thesis Supervisor: Supervisor One

Employee
Name: Employee Salaried
DOB: Thu Jul 16 16:56:10 EDT 2020
Employee Number: 987
Salary: 100000.0

Employee
Name: Employee Hourly
DOB: Thu Jul 16 16:56:10 EDT 2020
Employee Number: 988
Hourly rate: 0.0

List printing: polymorphism
[Name: Person One
DOB: Thu Jul 16 16:56:10 EDT 2020
```

```
, Student
Name: Student One
DOB: Thu Jul 16 16:56:10 EDT 2020
Student Number: 123456
, Graduate Student
Name: Grad One
DOB: Thu Jul 16 16:56:10 EDT 2020
Student Number: 123457
Thesis Supervisor: Supervisor One
, Employee
Name: Employee Salaried
DOB: Thu Jul 16 16:56:10 EDT 2020
Employee Number: 987
Salary: 100000.0
, Employee
Name: Employee Hourly
DOB: Thu Jul 16 16:56:10 EDT 2020
Employee Number: 988
Hourly rate: 0.0
]
```

If you have questions, don't hesitate to post your questions on the course forum on Moodle, or contact the instructor directly (andriyp@cse.yorku.ca).

Grading

The assignment will be graded using *the Common Grading Scheme for Undergraduate Faculties*[1]. We look at whether the code passes the unit tests, satisfies the requirements of this documents, and whether it conforms to the code style rules.

Submission

Find all the `java` files in your project and submit them electronically via Moodle (no zipping is required). There should be 6 files (or 7 if tester class is submitted) in total.

If working in a group, make only one submission and include a `group.txt` file containing the names and the student numbers of the group members. The deadline is firm. Contact the instructor *in advance* if you cannot meet the deadline explaining your circumstances.

Academic Honesty

Direct collaboration (e.g., sharing your work results across groups) is not allowed (plagiarism detection software may be employed). However, you're allowed to discuss the assignment requirements, approaches you take, etc. Also, make sure to state any sources you use (online sources – including web sites, old solutions, books, etc.). Although using outside sources is allowed – with proper citing, if the amount of non-original work is excessive, your grade may be reduced.

[1] https://secretariat-policies.info.yorku.ca/policies/common-grading-scheme-for-undergraduate-faculties/