Project 03: College Database

Release date: March 2nd, 2018
Due date: 11:59 pm, March 23rd, 2018

Prerequisites:

- Primitives
- Conditionals
- Loops
- Arrays
- Static Variables
- Member Fields and Functions
- Inheritance

Learning Objectives:

- Class Layout & Design
- Interfaces
- Managing and Manipulating Classes

Introduction:

In this project, you will design and implement classes that model a college. In this scenario, you are the president recently given a land grant to make your own college. You will need to start by creating software that will allow you to manage your courses, students, and faculty.

You will implement the following

1. ***Person*** interface
2. ***Student*** class (Implements ***Person***)
3. ***Teacher*** class (Implements ***Person***)
4. ***Professor*** class (Inherits from ***Teacher***)
5. ***Course*** class
6. ***College*** class
7. ***AddToCourseException*** class
8. ***DropFromCourseException*** class

Through creating new objects from the classes above, you will manage your college and perhaps even go on to build something greater after this project, like a College simulator.

Project Details

NOTE: You may be penalized if you do not observe the following

- All class fields must be **private** and **non-static**, unless otherwise provided.
- All required methods must be declared **public**.
- Do not change the **static final** variables

Overview
The majority of the classes created for this assignment are detailed within the skeleton code. Some of the points on the classes are reiterated below.

Getting to Testability
Your code will not be immediately testable from the skeleton. To make your code testable (and compilable) using our JUnit tests, you only need to implement the Professor's constructor. Read on for more details.

Person Interface
The Person interface provides the basic means for someone to be considered a Person in the context of our program.

Methods
- `void setName(String name);`
- `void setGender(String gender);`
- `void setAge(int age);`
- `int getID();`
- `String getName();`
- `String getGender();`
- `int getAge();`

Observe that it's a basic list of getters and setters. There is one method in particular that may stand out, which is getID. Every person is required to have an id (identification number). The id's should increment by one for each individual class starting at 0. Hence the id for a student will not affect the id for a Professor. For example, it is possible to have a Student, a Teacher, and a Professor all with the id 0. To help you keep track, we have allowed the use of a static variable named **nextID** for each class directly or indirectly implementing Person.

Teacher Class
The Teacher class will be one of the classes to implement the ***Person*** interface. Each Teacher object is uniquely identifiable by their **id**, meaning that if two Teacher objects have the same **id** they should be considered equal (implementing an **equals** method could be convenient).

Fields
- `protected static int nextID`
- `private String name`
- `private int age`
- `private String gender`
- `private int id`
- `private Course courses[]`
- `private int perCourseSalary`
- `private int baseSalary`

Methods

- `public void addCourse(Course course)`
- `public boolean dropCourse(Course course)`
- `public Course[] getCourses()`
- `public int teachesCourse(Course course)`
- `public int getPerCourseSalary()`
- `public int getBaseSalary()`
- `All methods required by Person`

It should be noted that when adding and dropping a Course, you will need to do error handling and double the size of the array containing the Teacher's courses should it be over capacity for the next course added. Teachers are capable of having duplicates of a course in the array returned by **getCourses**. You can take this to mean that the teacher has multiple sections of the course.

Teachers will always have two salaries: one they are always paid by the college (base salary), and one they are paid for every course they teach (per course salary).

There is no specific organization scheme you must follow for storing your courses.

More details on how your methods should behave is provided in the skeleton.

Constructors

There are 2 constructors which you need to implement.

- `public Teacher(String name, int age, String gender)`
- `public Teacher(String name, int age, String gender, int baseSalary, int perCourseSalary)`

Each of these constructors should initialize each of the private variables given. There is no specification for the size of the courses array, although something like 10 would be reasonable.

The first constructor assumes you will be giving the Teacher a default base salary of 30000 and a default per course salary of 15000.

Professor Class

The Professor class will extend ***Teacher***. Because Professor will inherit Teacher's fields and methods, the only thing you should need to do for the class is implement its constructors (which should call the Teacher's constructor with the *super* method).

The usage of Professors is different, however. While each course may have multiple Teacher objects, each course may only have one Professor object who serves as a sort of lead. Technically, via polymorphism we could add Professor objects to our courses as Teachers, but we will not be concerning ourselves with this detail until much later in the semester. Consider returning to this project once you've learned about polymorphism.

The Professor object will be responsible for remembering the courses the Professor is assigned to. Because they can only be assigned to a class once, the result of **getCourses** should not return duplicate courses.

Constructors
There are 2 constructors which you need to implement.

- `public Professor(String name, int age, String gender)`
- `public Professor(String name, int age, String gender, int baseSalary, int perCourseSalary)`

The major difference between the constructors for Teacher and Professor is the default salary. By default, Professors receive a base salary of 50000 and receive 15000 for every course they lead.

Student Class

The Student class is a basic implementation of the ***Person*** interface. Like Professors or Teachers, Student objects are uniquely identified by their ID number.

Unlike Professors and Teachers, Students are not responsible for maintaining the Courses they are enrolled in: this responsibility will be left to the Course class.

Constructors
There is 1 constructor which you need to implement.

- `public Student(String name, int age, String gender)`

This constructor just assigns the parameters to the appropriate fields and assigns the appropriate ID to the student.

Course Class

The Course class is an abstraction of a college course. For the purpose of this assignment, we will assume that courses at our college will not have the same name, meaning that Courses with the same name will be treated as the same Course. *However*, be wary of creating additional instances of a Course with the same name: if a Course has data in its fields, that data will not be carried over to the new instance of Course even if the names are the same.

Fields

- `protected final static int MAX_STUDENTS = 100;`
- `private String name;`
- `private Professor professor;`
- `private Teacher[] teachers;`
- `private Student[] students;`

Each Course has a lead professor, teachers, and students limited by **MAX_STUDENTS**. Obviously, it should also have a name as well. There are no restrictions on names.

Methods

- `public void addStudent(Student student) throws AddToCourseException`
- `public void dropStudent(Student student)throws DropFromCourseException`
- `public void addTeacher(Teacher teacher) throws AddToCourseException`
- `public void dropTeacher(Teacher teacher)throws DropFromCourseException`
- `public Professor getProfessor()`
- `public Student[] getRoster()`
- `public String getName()`
- `public Teacher[] getTeachers()`
- `public void changeProfessor(Professor professor)`

Course methods will be where you will do a majority of the administrative work for the Teachers, Students, and Professors involved.

Note that [add/drop][Student/Teacher] each throw an exception. You will finish the class for these exceptions later, but you will be expected to throw the correct exceptions under these circumstances:

- When adding a Student to a course that is full
- When adding a null Student/Teacher to a course
- When adding a Student already enrolled in the course
- When dropping a Student/Teacher who is not assigned to the course
- When dropping a null Student/Teacher

It should be noted that, because we allow Teachers to teach multiple sections, when you call **dropTeacher** it should only remove one instance of the Teacher if multiple are present. Whenever a Teacher is added or dropped from a Course, be sure to update the Teacher object as well to remove the Course from their personal list of Courses (*hint: the keyword **this** is a reference to the current object whose method you are in)*.

getRoster and **getTeachers** will vary from your other array getters. Rather than returning a reference to the array they are getting, the will return a *new* array containing references to each Student or Teacher in the original array with no *null* values. Note that duplicate teachers are included in the array returned by **getTeachers**.

Lastly, **changeProfessor** should make the passed Professor argument the new lead Professor for the course. You will want to ensure that both Professors are updated correspondingly.

Constructors

There is 1 constructor which you need to implement.

- `public Course(String name, Professor professor)`

Each Course object requires a name and a lead Professor. You must also initialize your Teacher and Student arrays. Something like 5 would work for the Teacher array, whereas the Student array should be of size **MAX_STUDENTS**.

College Class

The College class does the highest level administrative work. It will be where you keep track of Students that are enrolled at the College, Courses offered by the College, and faculty employed by the College. You will also use this class to help determine whether or not your current college set-up will be fiscally feasible (in other words, does student tuition cover the cost of faculty?).

Fields

- `protected static final int MAX_COURSES = 10;`
- `protected static final int MAX_PROFESSORS = 5;`
- `protected static final int MAX_TEACHERS = 25;`
- `protected static final int MAX_STUDENTS = 500;`
- `private String name;`
- `private Professor[] professors;`
- `private Teacher[] teachers;`
- `private Student[] students;`
- `private Course[] courses;`
- `private int tuition;`

We limit the size of our Person-type arrays to simulate the small college start-up. Each array of Person-type objects should only contain unique people (no two in a single array should have the same id). Similarly, there should be no duplicate courses (courses with the same name).

Methods

- `public void addCourse(Course course)`
- `public void hireProfessor(Professor professor)`
- `public void hireTeacher(Teacher teacher)`
- `public void addStudent(Student student)`
- `public void dropStudent(Student student)`
- `public int calculateNetBudgetChange()`
- `public Course[] getCourses()`
- `public Teacher[] getTeachers()`
- `public Professor[] getProfessors()`
- `public Student[] getStudents()`
- `public String getName()`
- `public int getTuition()`

Again, you'll of course have your basic getters and setters. We don't require that copies are made for the getters that return arrays: returning the reference to the original array will suffice.

We also have some methods for adding Person-types and Courses to the College object. For each of these methods, you should ensure that if the corresponding arrays are full, the parameter is *null*, or the parameter passed is already in the array, the array is not modified.

We also have a **dropStudent** method. In the unfortunate event that one of our students decides to drop out from our college, we need to do the proper administrative work. This means not

only removing them from the College, but also removing them from any Courses they are enrolled in.

calculateBudgetNetChange will determine how much money is made based on income from students and expenses from faculty. In short, the equation amounts to:
*numberOfStudents*tuition - (sumOfFacultyBaseSalaries) - (sumOfFacultyPerCourseSalaries)*

The last number in the equation will shake out to be a bit more complex, but should be very doable should you be doing your bookkeeping correctly.

Constructors
There is 1 constructor which you need to implement.
- `public College(String name, int tuition)`

Like the other classes, this constructor should set the corresponding field variables to the parameters and initialize the arrays for each Person-type and Course. Intuition will be the price paid be each student to attend your school. If you are having any trouble thinking of tuition values to test, just trust your *int-tuition* and pick some.

AddToCourseException and DropFromCourseException

These two exceptions will be thrown in the cases mentioned in the **Course Class** section. You will be responsible for calling the **Exception** constructor within their constructors.

Starter Content

The zip file contains the **skeleton code** and **sample test cases**. To use the test cases, place them in the same folder as your source files. If you need help beyond this step, please consult with a TA.

Because this assignment will be not be dependent on output, we will not be providing a sample demonstration (because there is simply nothing to demonstrate). Hopefully the test cases can provide an adequate guide. However, we welcome all requests for clarifying questions and any observations.

Grading Rubric

- **10% - Variable Usage**
  - 2% - Using only Static Variables as Permitted
  - 2% - Declaring no public fields
  - 6% - Each Person-type object has increasing IDs
- **30% - Teacher and Professor Classes**
- **30% - Course Class**

- **30% - College Class**
- **0% - Student Class (will be a crucial element of the above classes)**

Turning in Your Work

Submit the following files on Vocareum through Blackboard. Your solution will be auto-graded by Junit test cases.

- Person.java
- Professor.java
- Teacher.java
- Student.java
- Course.java
- College.java
- AddToCourseException.java
- DropFromCourseException.java