Lab 8 Classes and Objects - Part 2:

You have already written a Java class to represent a class section for a course like CIS 131, but you must wonder, what about the students who have enrolled in it? We'll take care of that now by creating another class called Student and adding an attribute of an ArrayList of this type to the ClassSection you created in Lab 6. Now the complete set of attributes for ClassSection will be:

1. CRN, like 20008
2. Department code, like CIS
3. Course number, like 131
4. Instructional mode, like online, classroom or hybrid
5. Campus, like East or West (classroom and hybrid only, otherwise "N/A")
6. Meeting days (classroom and hybrid only, otherwise "N/A")
7. Meeting times (classroom and hybrid only, otherwise "N/A")
8. Capacity (maximum number of students who can enroll in it)
9. Enrollment (number of students actually enrolled in it)
10. Instructor's ID number (in schedules you see the instructor's name due to a database operation called a join. Take CIS162 and 182 for more info on this.)
11. Enrollee list (an ArrayList of Student enrollees)

Now we need to create a **Student** class which has these attributes:

studentID

grade (For simplicity let's use numbers like 4, 3, 2, 1 and 0.)

We don't include student attributes like lastName, firstName, majorCode, etc. because we assume that the database designers have correctly designed these tables to eliminate redundancy. Therefore, they represent student in an enrollee list by its ID which points to a record in a separate student table that contains the complete set of student attributes. Take CIS 162 and 182 for more information on this!

Because these attributes, like most data attributes in object-oriented applications, are private, they need setter and getter methods. They must also have a no parameter constructor, a constructor that accepts parameters for both attributes, and a toString method.

Once you've written the Java file for the Student class, you can add an ArrayList of student enrollees to the class section class. We use an ArrayList because it has the ability to expand to accommodate new elements, as opposed to an array which has a fixed size.

We declare ArrayLists this way:

List <Class> nameOfList = new ArrayList<Class >(optional number of elements);

For instance, to create an empty ArrayList of objects of a class named Book:

List <Book> bookList = new ArrayList<Book>();

To create an ArrayList of objects of a Book class which can initially hold 10 Book objects:

```
List <Book> bookList = new ArrayList<Book>(10);
```

Now the classSection class needs several new methods related to the student enrollee list:

- A method, let's call it addStudent, to add an individual student to the enrollee list
- A method to locate the ArrayList element which holds (actually points to) the enrollee object that has a particular student ID
- A method to withdraw a student
- A method to assign a grade to a student
- A method to display a list of enrollees

Unless you really want to do extra work, **don't** add a parameter for the ArrayList of students to the ClassSection constructor. We can just add them one at time using an addStudent method described later. This more closely resembles the actual enrollment process anyway.

The next several paragraphs describe some activities which you will probably find complex (and they are!). The sample code in this module demonstrates the below techniques. Study them carefully.

Adding a Student: There is a slight complication in working with our ArrayList. We cannot interact with it directly because it is a private member of ClassSection. Therefore, we cannot simply perform an action like adding adding a student by invoking the add method of the ArrayList class.

Instead, we must create a public addStudent method of the ClassSection class which will pass the Student object to the add method of the ArrayList class.

Withdrawing a Student: Likewise, we cannot withdraw a student by simply invoking the remove method of ArrayList, We must create a withdrawStudent method that invokes the remove method of ArrayList.

In order to either withdraw or assign grades to a student, we must first locate her element in the ArrayList. We must write a method that does this. It will accept a student ID as a parameter then loop through the ArrayList until it finds the object which has it. Then it can return the index of this element. If no element has a particular ID, this method should return a negative one (-1) to indicate that the search failed.

Withdrawing a student involves first locating the index of the object with the ID you want to delete then using the remove method of the ArrayList class to delete it.

Assign a Grade: To assign a grade to an enrollee, invoke the setGrade method of the enrollee class via the object with the index of the student whose grade you want to change. Again, you must first find the index belonging to that student.

Calculate the Enrollment: Change the getEnrollment method of the class section class to return the number of enrollees in the ArrayList by invoking its size method.

Now for the driver program: create an instance of a ClassSection class that accepts hard coded values for all parameters. Then do the following:

1. Display the results of the ClassSection object's toString method, just as you did in the previous project.
2. Next add several students to the enrollee ArrayList using the method that accepts one student. Give them sequential IDs (such as A001 to A004), and grades of 0 because the semester just started and they haven't earned any grades yet.
3. After adding the students, display the enrollee list.
4. Change the grades of the students to 1, 2, 3 and 4.
5. Display the enrollee list again.
6. Now delete a student then display the enrollee's again.

Provide your Java source code for the class section class, the student class and the driver program. Since we have multiple files, compress them together into a file such as CIS131-DFreitag-Lab-8.zip and then upload it to D2L.