COSC1337 Programming Fundamentals II /ITSE2357 Advanced Object-Oriented Programming

The submission opens two days before the due day, and closes right at the due time on due day. One submission is allowed per project. It is students' responsibilities to submit timely and correctly in order to get credits. Students MUST start each project when it is first discussed in class in order to complete it on time. All projects are individual projects. Project due days may be changed to better align with lectures.

If one of the following is true, not credits will be given:

- The project is late.
- The algorithm or class design is missing.
- The project has syntax errors.
- There are no comments (Javadoc format required) at all in your code.
- Wrong files are submitted.
- **A project is a copy and modification of another student's project. (Both will receive 0.)**

Files to be submitted through Canvas:

- The UML design (UML class diagram)
- All source codes and driver programs
- All supporting files if any

Software Development Project Rubric: (Design is worth 20%; the rest is worth 80 %.)
Analysis: There will be no credit if the software doesn't meet customer specification at all.

- Does the software meet the exact customer specification?
- Does the software read the exact input data and display the exact output data as they are shown in sample runs?
- Does each class include all corresponding data and functionalities?

Design: There will be no credit if the design is missing.

- Is the design (a UML class diagram) an efficient solution?
- Is the design created correctly?

Code: There will be no credit if there are syntactic errors.

- Are there errors in the software?
- Are code conventions and name conventions followed?
- Does the software use the minimum computer resource (computer memory and processing time)?
- Is the software reusable?

Debug:

- Are there bugs in the software?

Documentation: There will be no credit if comments are not included.

- Are there enough comments included in the software?
- Class comments must be included before a class header.
- Method comments must be included before a method header.
- More comments must be included inside each method.
- All comments must be written in Javadoc format.

Project 2 The *ArrayList* Class, static members, and *Comparable* interface **Due: See calendar**

Design and implement a class called *Course* that represents a course taken at a school. A course object should have a name, and a list of students enrolled in the class. Revise the *Student* class implemented in last project to represent the students in the course. The revisions are included in part I. The course design is included in part II.

Part I:

Add a unique student id, and test scores along with associated methods into the *Student* design:

Instance variables and static variables:

- `name`: The name of this student. The data type is type *Name*.
- `homeAddress`: The home address of this student. The data type is type *Address*.
- `schoolAddress`: The school address of this student. The data type is type *Address*.
- `id`: The unique id of this student.
- `nextID`: the next available id. This is a static variable. The initial first student id can be 1000.
- `scores`: The test scores of this student. The data type is *int[]*.

Instance methods and static methods:

- Add a *getter* of the student id.
- Adds a *getter* of the student's scores.
- Add a method (*calculateAverage()*)to calculate the average grade of this student.
- Add a static method to return the next id.
- Revise *toString* and *equals*
- Add a class natural ordering by completing the class's *compateTo* method. The natural order of students can be the order of student ids, or student names.

Part II:

Instance variables and static variables:

- `name`: The name of this course. The data type is type *String*.
- `id`: The id of this course.
- `nextID`: the next available id. This is a static variable. The initial first student id can be 10000.
- `students`: The list of students enrolled in this course. The data type is type *ArrayList<Student>*.
- `MAX_ENROLLMENT`: The maximum enrollment of this course. The data type is type *int*. Its value is *20*.

Instance methods and static methods:

- `public Course(String)`
  The constructor should accept only the name of the course.
- Add getters for all, and setters for non-read-only variables
- `public void addStudent(Student)`
  Add a student when the class is not full.
- `public void average()`
  Calculate the average scores of the entire class.
- `public String toString()`
  Represents the entire course as a string
- `public boolean equals(Object)`
  Indicates if this course is "equal to" to some other object

- `public int compareTo(Course c)`

  Defines a class natural ordering by completing the class's *compateTo* method. The natural order of course can be the order of course ids, or course names.

Part III: Write a program *TestCourse.java* to test your program. (More discussions in class)