**DESIGN CONSIDERATION**

Students, professors and courses are specific and essential classes for the course and marks registration system. Since students and professors are people, they extend or inherit from the class Person which contains all personal details of the student or professor. In this case, person alone is meaningless to a course registration system, thus it is abstract. However, due to the need to distinguish each student and professor, the Person class defines an abstract method, getID() and setID(int i) to be implemented by the subclasses.

In differentiating students and professors, they need to have a unique ID. Therefore, they implements an interface called PrimaryKeyManager which requires the implementation of method autoIncrement(int i). This applies to Course and Group classes as well where ID is used to distinguish them. Besides the method, these classes need to define a static integer value, named pk, which starts from 1. The object ID of the above classes will be determined by this “pk” value, and this value will be incremented upon construction of the object by the above method. This ensures that the objects created are distinct with different IDs. In addition, to check whether or not the objects are equal, the ‘equals’ method is overridden by comparing the classes (‘getClass’) and ID values.

Student, Professor and Course are entity classes, and they are aggregated under School class. This is to ensure that when Student, Professor and Course objects are saved by Java Serializable, the objects will not be duplicated when loaded and the associations are still intact with the right reference. This ensures that when you change a course by using the professor’s ‘getCourse’ method, the same course aggregated under School will also receive the change.

In determining course types, a CourseType enumeration class is used since the types will hardly change and thus act as a constant. These constants are used to determine whether or not a course contains lecture, tutorials or labs. A CourseGroup object acts as a control to manage the groups. Depending on the CourseType, CourseGroup will determine instantiate the right Group object for the course. Hence, Group objects are entities containing the students and capacity information.

In determining course assessment components, a key-value data structure, HashMap, is used to store the name of the assessment as the key and the weights as the value.  
Calculation formula for total marks:

Registration of a student to a course occurs only in a new academic semester. Therefore, there is a need to create a new academic semester before registration. The Semester object will contain all courses that were added in by the administrator and are open for registration. Students are registered in by inputting their appropriate ID and tutorial and lab group IDs. In this application, every course can only have a maximum of 1 lecture group. During the academic semester, students’ grades or marks for the final exam as well as other assessment components can also be input by specifying the student ID and course ID. Finally by ending the semester, the overall marks will be calculated and students’ CGPA will be updated.

Therefore, there are 4 major parts of our application which involves students, professors, courses and semesters. Each of them will have a static main method defined to interact with the user, thus acting as a boundary method. However, these main methods cannot be invoked by running their class files, they can only be invoked by the application class SCRAME depending on the user inputs. Although the main methods are classified as boundaries, there exists some control logic to interact with each object’s interface.

The use of polymorphism may be vague in our application classes due to the need of using and returning precise and accurate types in the program. For example, we cannot pass a Person casted object in registration of courses because professors are not students. However, some of the data structures, like HashMap, can be referred using the superclass Map, but was not shown in implementation for the sake of simplicity. One particular example of polymorphism is shown in Person’s ‘equals’ method, which check for the class and cast it to Person type regardless of whether or not the object is of Student or Professor class.

@Override

**public** **boolean** equals(Object obj) {

**if** (!(obj **instanceof** Person)) **return** **false**;

Person person = (Person) obj;

**if** (person.getClass().equals(**this**.getClass())) {

**if** (person.getID() == **this**.getID()) **return** **true**;

**if** (person.getName().equals(**this**.getName()) &&

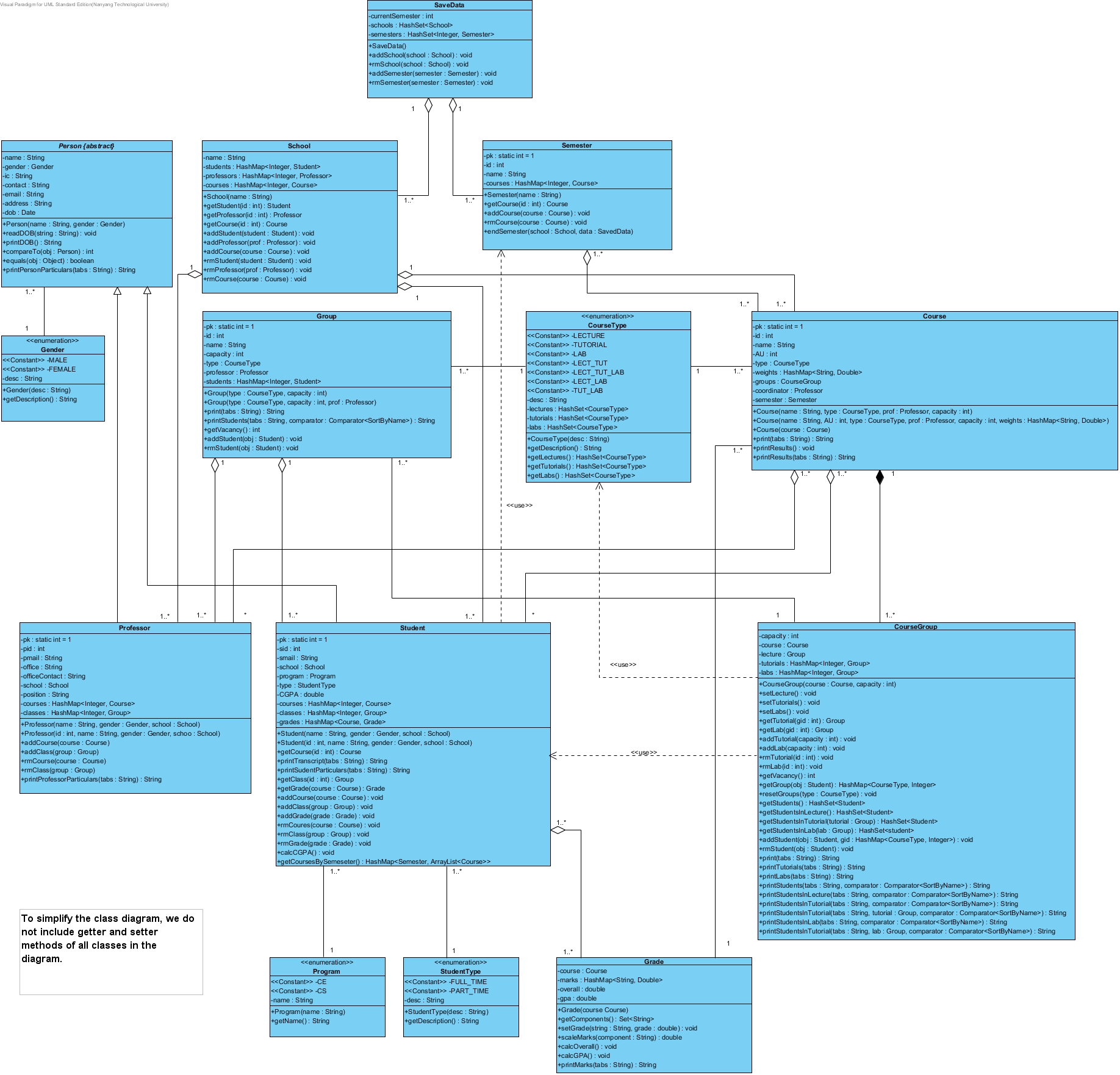
person.getGender() == **this**.getGender() ) **return** **true**;

}

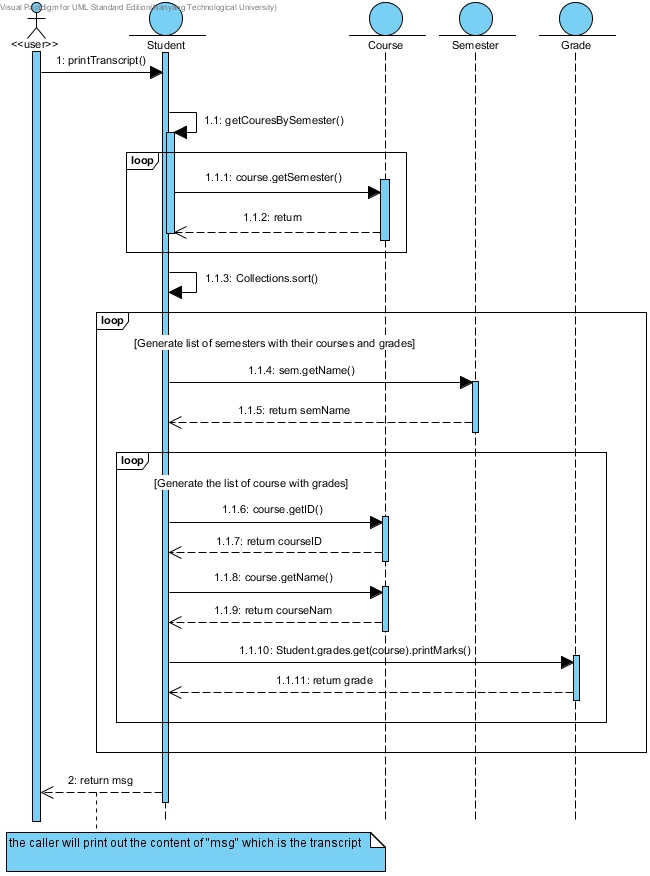
**return** **false**;

}

**CLASS DIAGRAM**



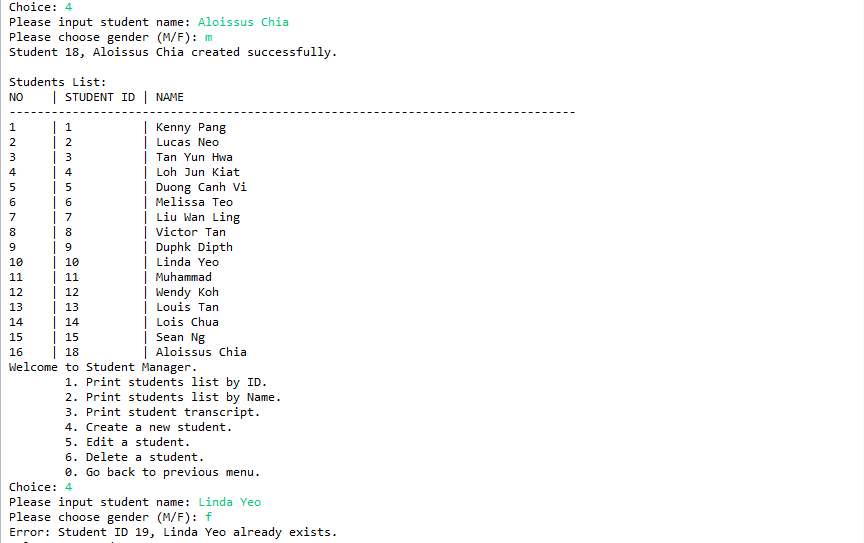
**SEQUENCE DIAGRAM**



**TEST CASE**

Test case 1: Student

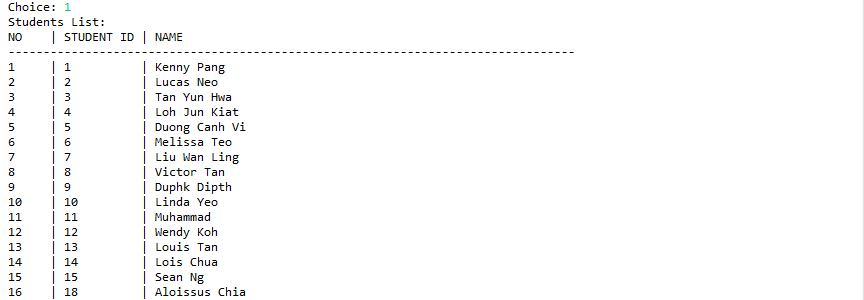
* Create a new student
* Add an existing student



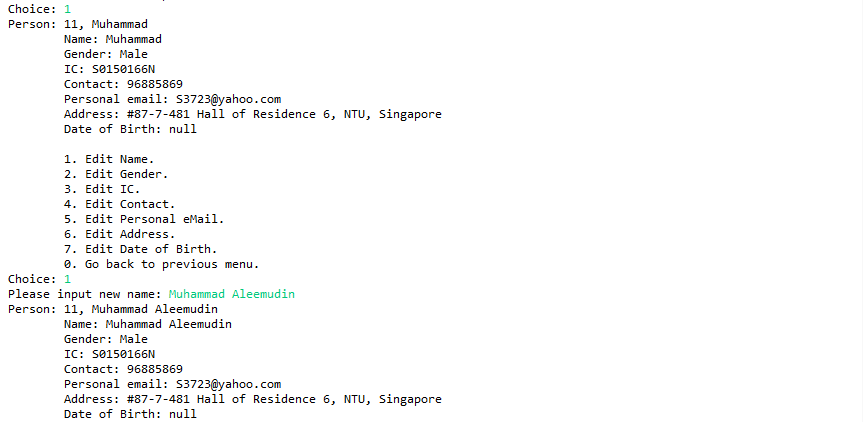
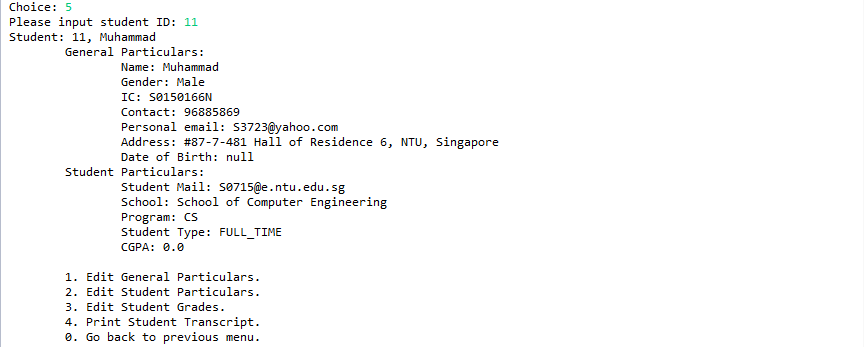
* Invalid input



* Print student list by ID



* Edit student’s particulars



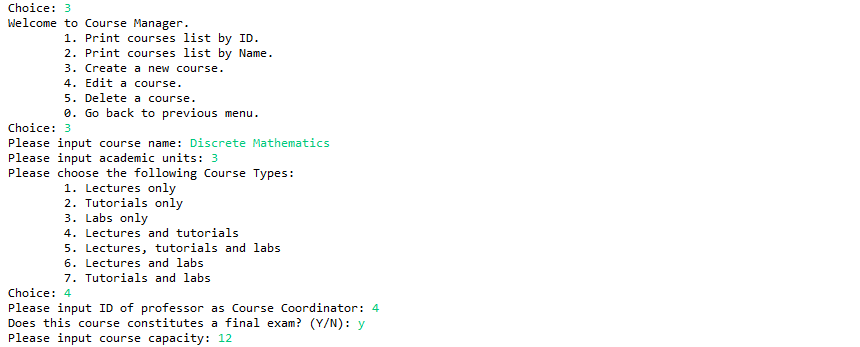
* Delete a student



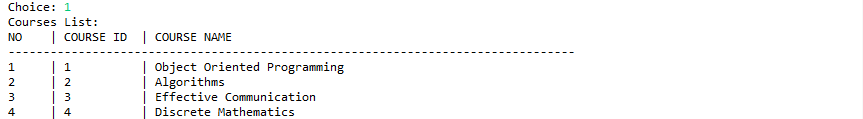
* Similar for creating and managing Professors (with different particulars)

Test case 2: Course

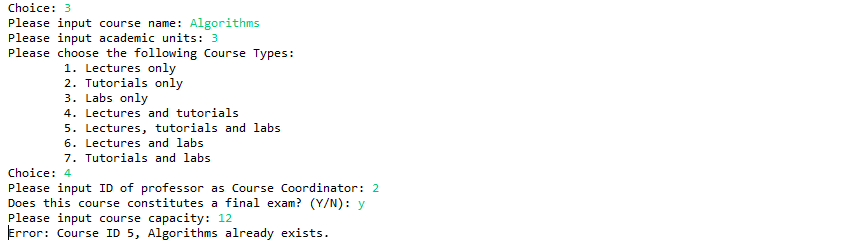
* Create a course (including its type, AU weightage, professor in charge, inclusion of final exam and capacity)



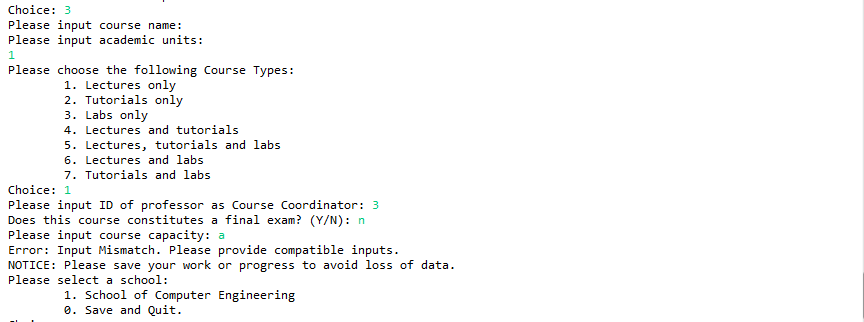
* Print course list

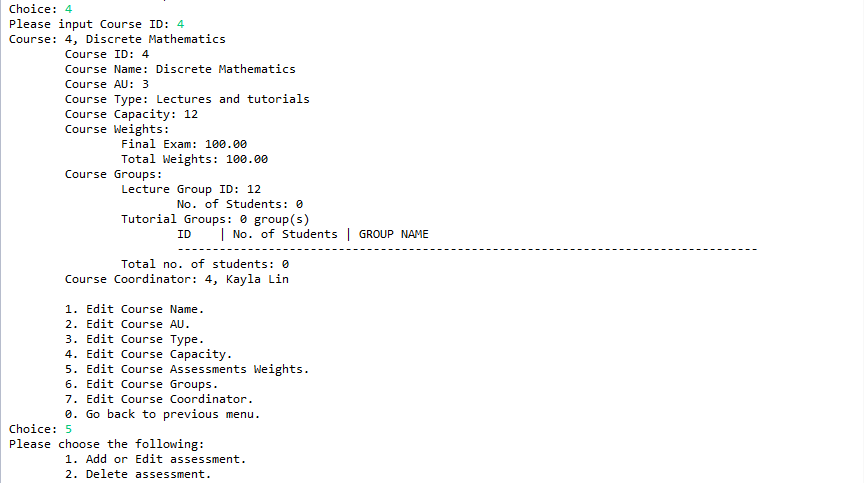
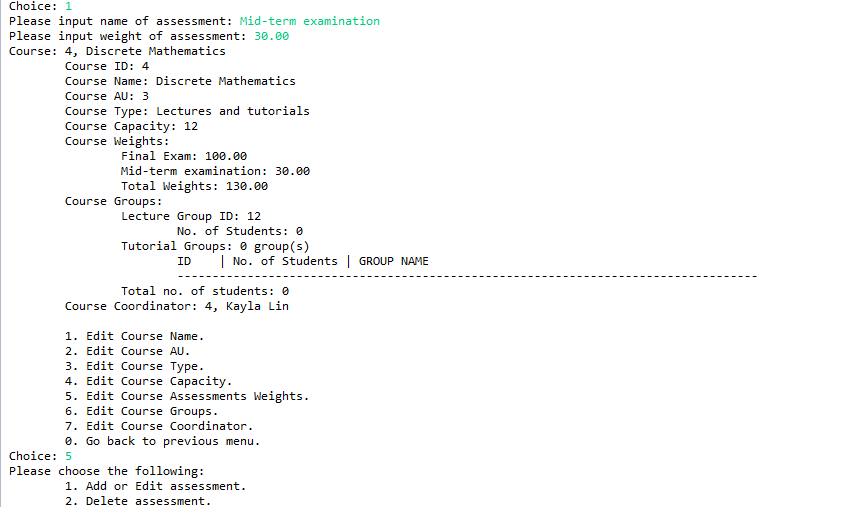


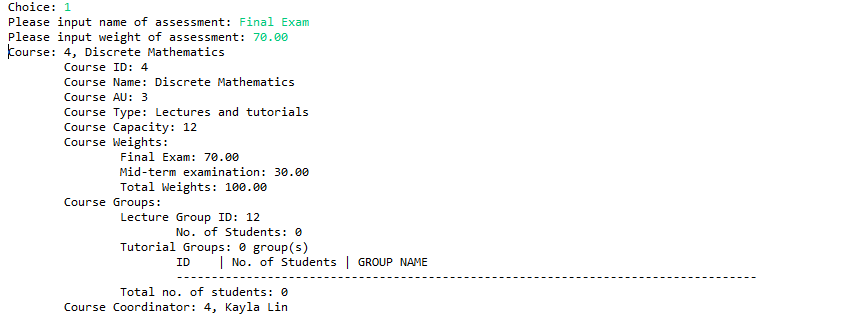
* Add an existing course



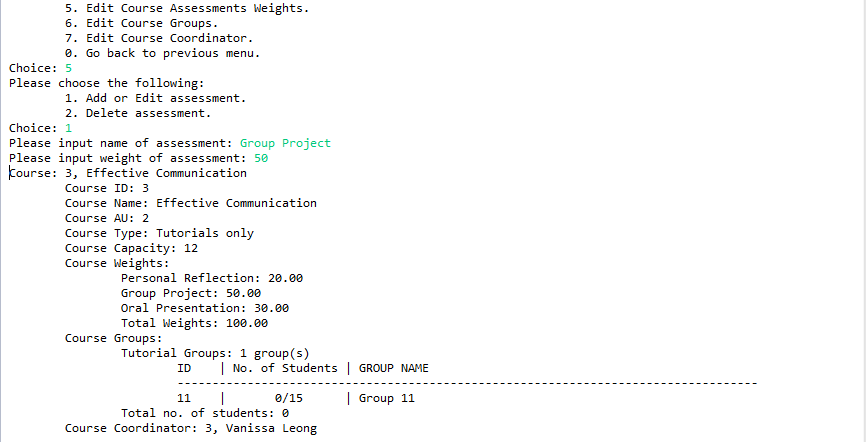
* Invalid entries.



* Edit a course details, including entering course assessment weightage

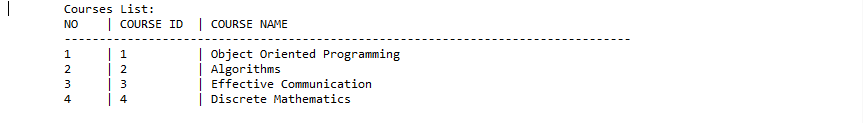
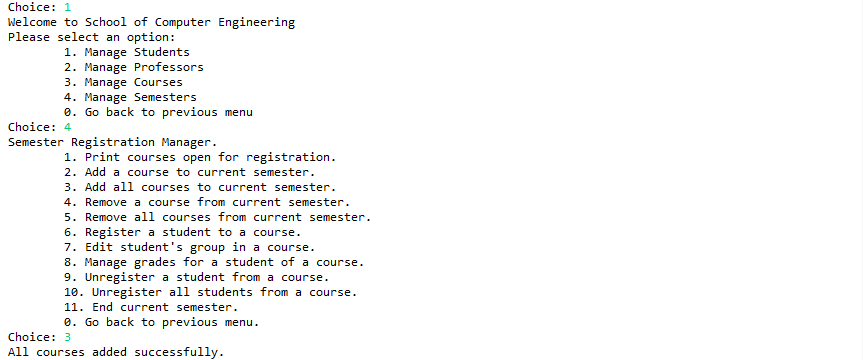


* 3 components

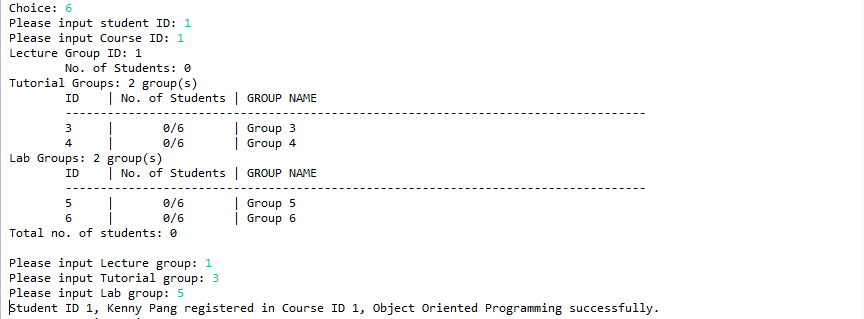


Test case 3: Semester management

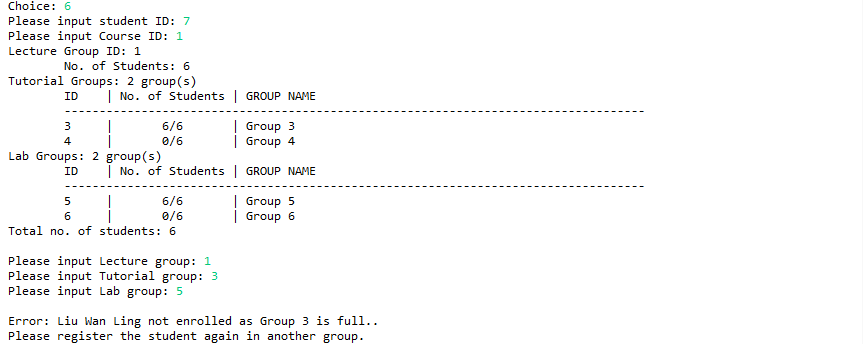
* Add courses to a semester



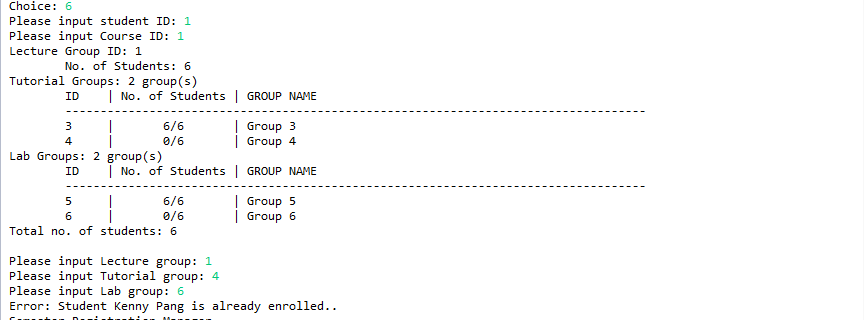
* Register a student to a course, showing course vacancies



* Register a student to a tutorial or lab group with no vacancies

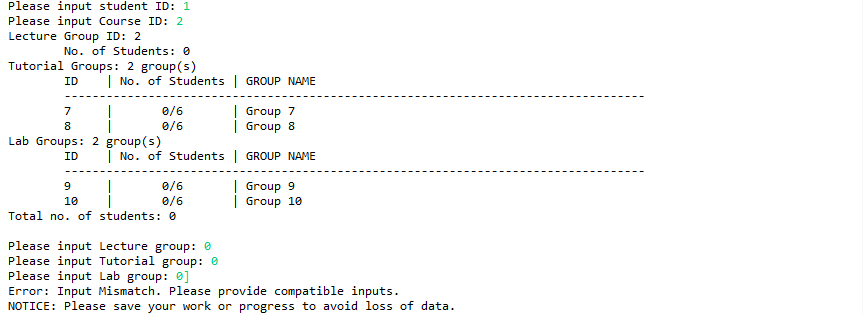


* Register for the same course again

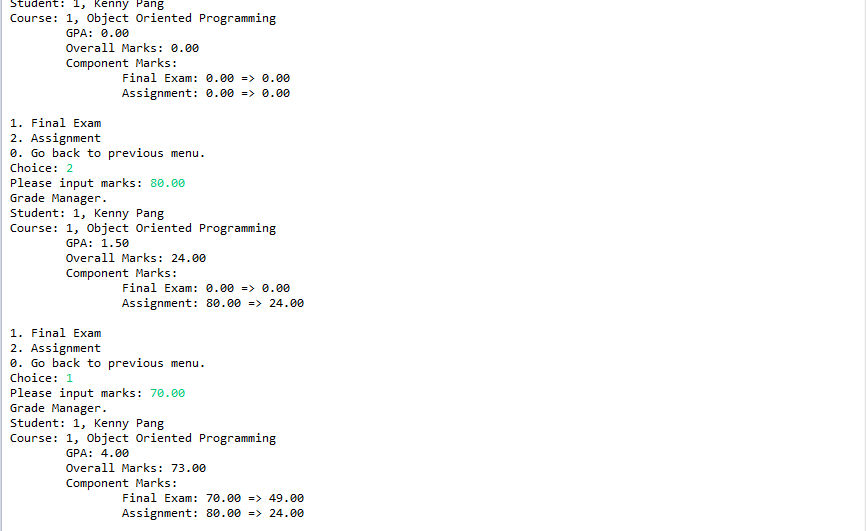


* Invalid data entries





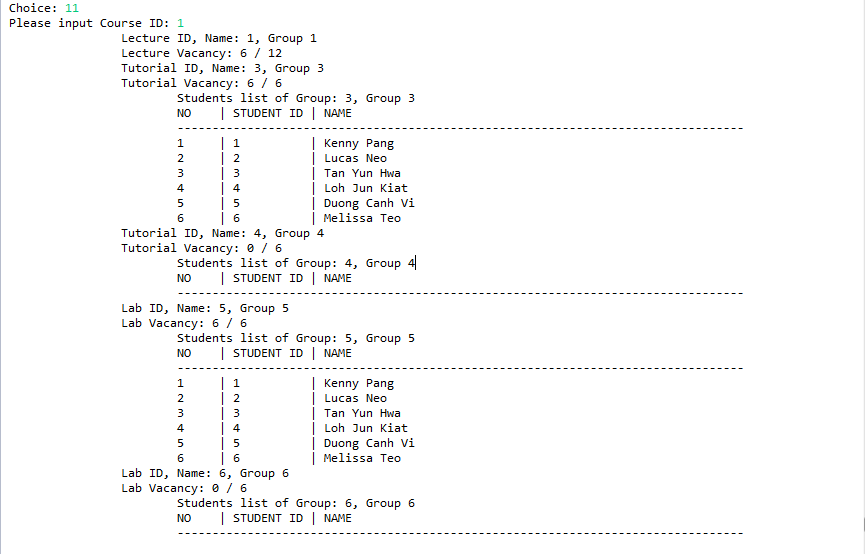
* Add exam and coursework grade



* Invalid entries



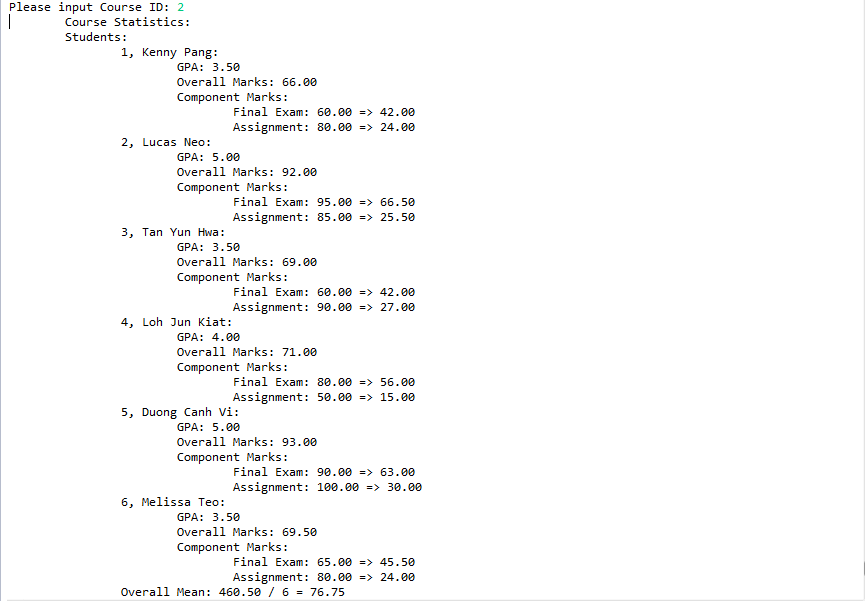
* Print students enlisted in a lecture/tutorial/lab group



* Invalid entries



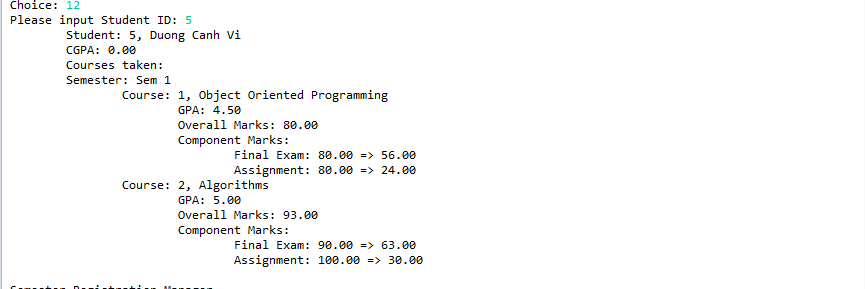
* Print course statistic



* Invalid entries



* Print student transcript



* Invalid entries



---END---