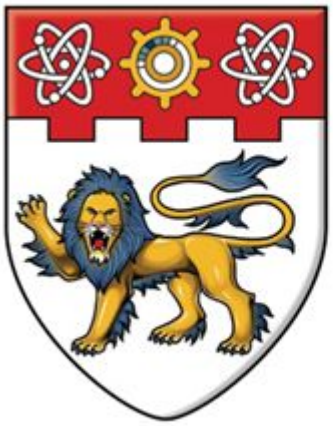


NANYANG TECHNOLOGICAL UNIVERSITY

CZ2002: OBJECT-ORIENTED DESIGN & PROGRAMMING

Submission Date:

25th November 2020

Names of group members

Ong Eng Hao

Kelvin Wong Wai Leong

Tran Hien Van

Hoo Jia Kai

Project video - Group 03: https://youtu.be/VnDHqP6Y_vo

Table of Contents

I. Introduction

My STudent Automated Registration System (MySTARS) is an university application meant for both undergraduate students and the admin staff. There are two modes: administrator mode for academic staff and user mode for students. MySTARS application can be used for key features such as creating courses and adding student records as well as course registration.

This report will show the OOP concepts and design principles applied to model the application. The UML Class Diagram and UML Sequence Diagram for one feature will be used to illustrate the design. In addition, several test cases will be considered to show that the application will meet all requirements.

II. Design Considerations

a. Approach Taken

N-Tier architecture was used to implement this project. The 3 main tiers are presentation tier, logic tier and data tier. The user interface is where the presentation tier lies, the controllers of the project lie in the logic tier, and the data of the project (Students, Users, etc..) are in the data tier. The N-tier approach was chosen as it is easy to manage, add new features and high reusability.

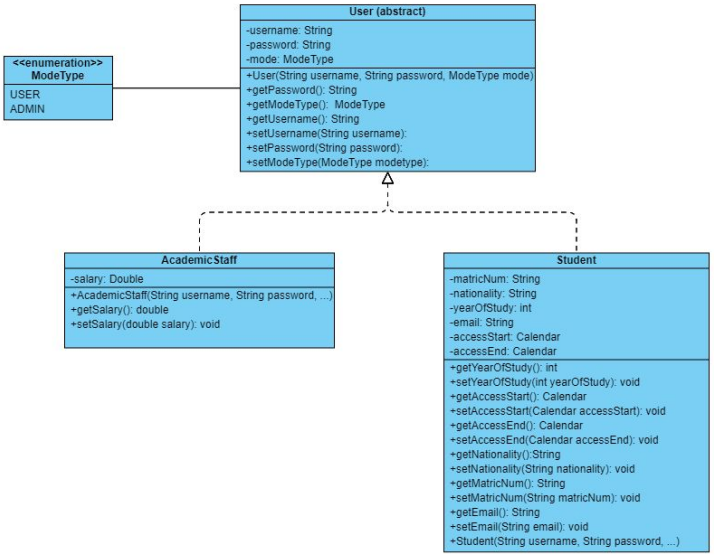
b. SOLID design principles used

b.1. Single Responsibility Principle (SRP)

To ensure cohesion, the principle states that there should be no more than one reason for a class to change. This is because each responsibility a class possess is a potential reason for change. As such we aimed to divide roles among different classes. For example, to avoid a "God" class for Course, we separated the classes into different types of classes each handling one responsibility such as the – UI classes, File handling classes, Controller classes, etc. The SRP principle helps us reduce functional overlaps and reduce the rigidity in our design allowing changes to be introduced to a specific part of the system without much hiccups.

b.2. Open-Closed Principle (OCP)

Our group applied OCP in the implementation of our login module, making sure they are open for extension but close for modification. This is to allow modification of the functionality of the modules without changing the source code. For example, the User class is an abstract class and it is extended by the AcademicStaff and Student class which can be seen in *Figure 1* on the right.



This allows for **extension** to more types of users of the application. For example the addition of a class to represent Student Teacher Assistant (TA) who may need to do both course registration and modification of course details can be done without changing the source code of the modules.

b.3. Interface Segregation Principle (ISP)

The ISP design principle states that many client specific interfaces are better than one general purpose interface and as such, we created many different interface classes to ensure that controller classes are segregated by their functions. This is mainly because of the segregation of responsibilities that we have established for each class that allows for a good segregation of duties.

b.4. Dependency Injection Principle (DIP)

In the design, the classes do not depend on interfaces which they did not use.

c. Other principles

c.1. Delegation Principle

The delegation principle states that a class should not deal with everything all by itself, delegate to the respective classes. An example of this can be seen in our implementation. An example can be seen from the figure. In the following implementation, we delegated the checking of string to String class (course code is an object of String class)

```
while(!validInput){
    courseCode = getStringInput( prompt: "Enter Course Code: ");

    if(courseCode.equals("-1")) return;

    if(adminCrsCtrl.isExistingCourse(courseCode)==true){
        System.out.println("Course code already exists!");
        System.out.println("Please try again!");
    } else validInput=true;
}
```

d. A Object-Oriented Concepts (Explanation of UML Diagram)

d.1. Composition

Compositions are parts that make up the whole class. Composition is chosen for the following example as a course “has a” index lists and indexes are part of a course as seen in figure 1 below.

d.2. Association Class

Association is the relation between two separate classes which is established through their objects. Association Classes are implemented by the StudentUI which has the StudentCtrl, PrintInfoCtrl and CourseCtrl as its attributes as seen in figure 2 below.

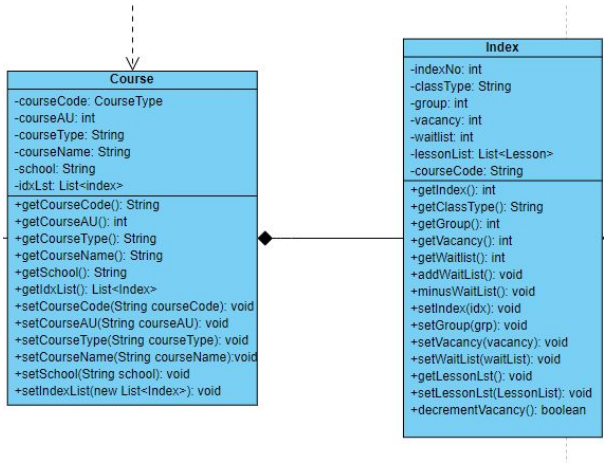


Figure 1

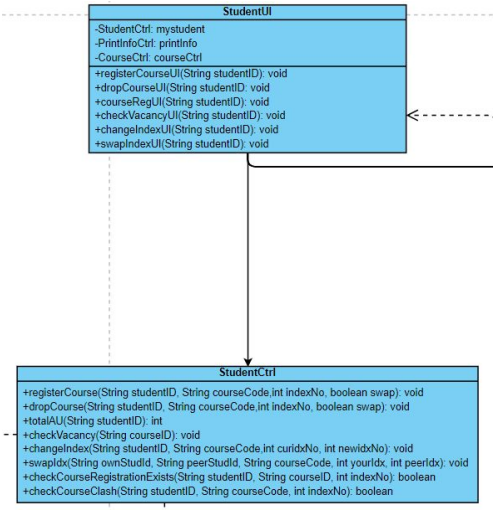
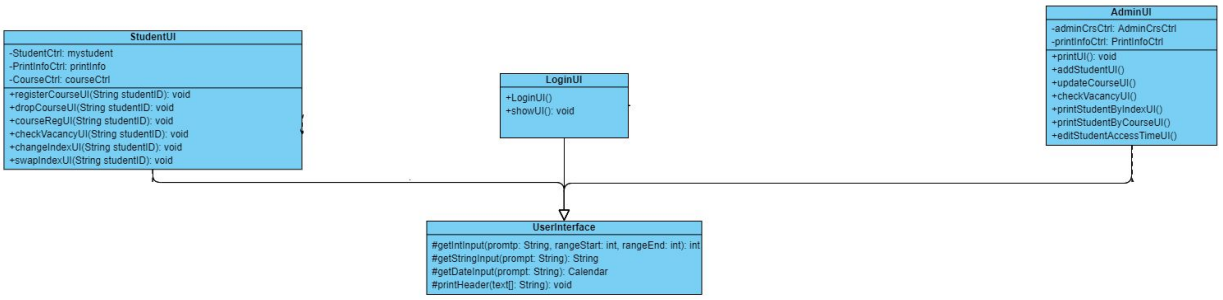


figure 2

d.3. Inheritance

In the following example, the three user interfaces (StudentUI, LoginUI and AdminUI) inherit UserInterface class to acquire the same set of methods such as printHeader and getStringInput. This way, it improves the usability of the code for the three child classes.



d.4. Encapsulation/ Information Hiding

Encapsulation and Information Hiding are used for the entity classes. They have private attributes which can only be accessed by get/set methods. Using the methods exposed by the object to manipulate the data, users can ignore the internal complexity and implementation details. In this case, information hiding hides the internal details of the class from users.

For example, the User class has private attributes such as username, password and mode. Such attributes are only accessible through their respective get and set methods.

d.5. Abstraction

Abstraction was used to have an abstract user class. Student and Academic staff extends the abstract class as seen in figure 1 below. This strategy was implemented as we can define a template for both academic staff and students and the methods can be used later.

d.6. Polymorphism

In the UserInterface class, there is a base method of PrintUI() which will be overridden or overloaded in the AdminUI and StudentUI respectively as seen in figure 2 below. The method overriding in AdminUI is a form of dynamic polymorphism whereas the method overloading in StudentUI represents static polymorphism.

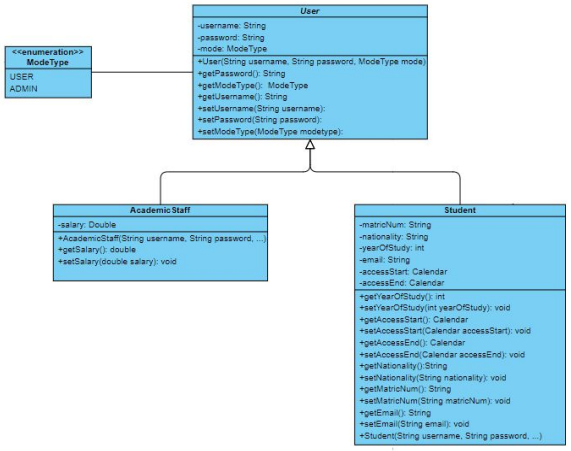


figure 1

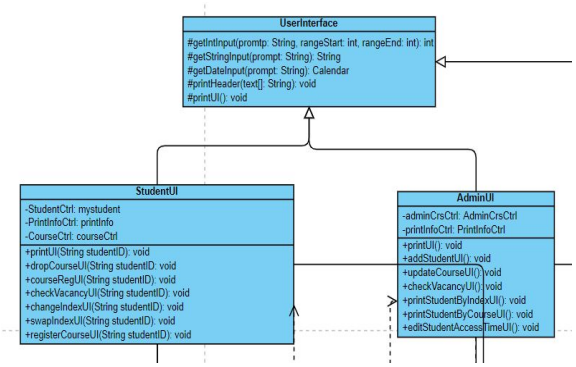


figure 2

e. Notifications

e.1. Email-Guide

There is a problem sending emails from Gmail because of Gmail account protection. To allow notification to be sent from Gmail, users have to fix the issue by going to this link https://www.google.com/settings/security/lesssecureapps and click 'allows less secure applications: OFF. Next, retrieving of the emails can be done through searching in a Student text file under the *'data'* folder. To configure the emails, users can directly alter in the text file. The dummy emails used in this implementation is mapleseaok@gmail.com which acts as the sender.

e.2. Extensibility to other modes

By adopting a strategy design + factory design pattern on our NotificationCtrl class, we can allow for easy extensibility in our notification modes simply by adding new subclasses with the respective APIs of the delivery mode.

III. Assumption

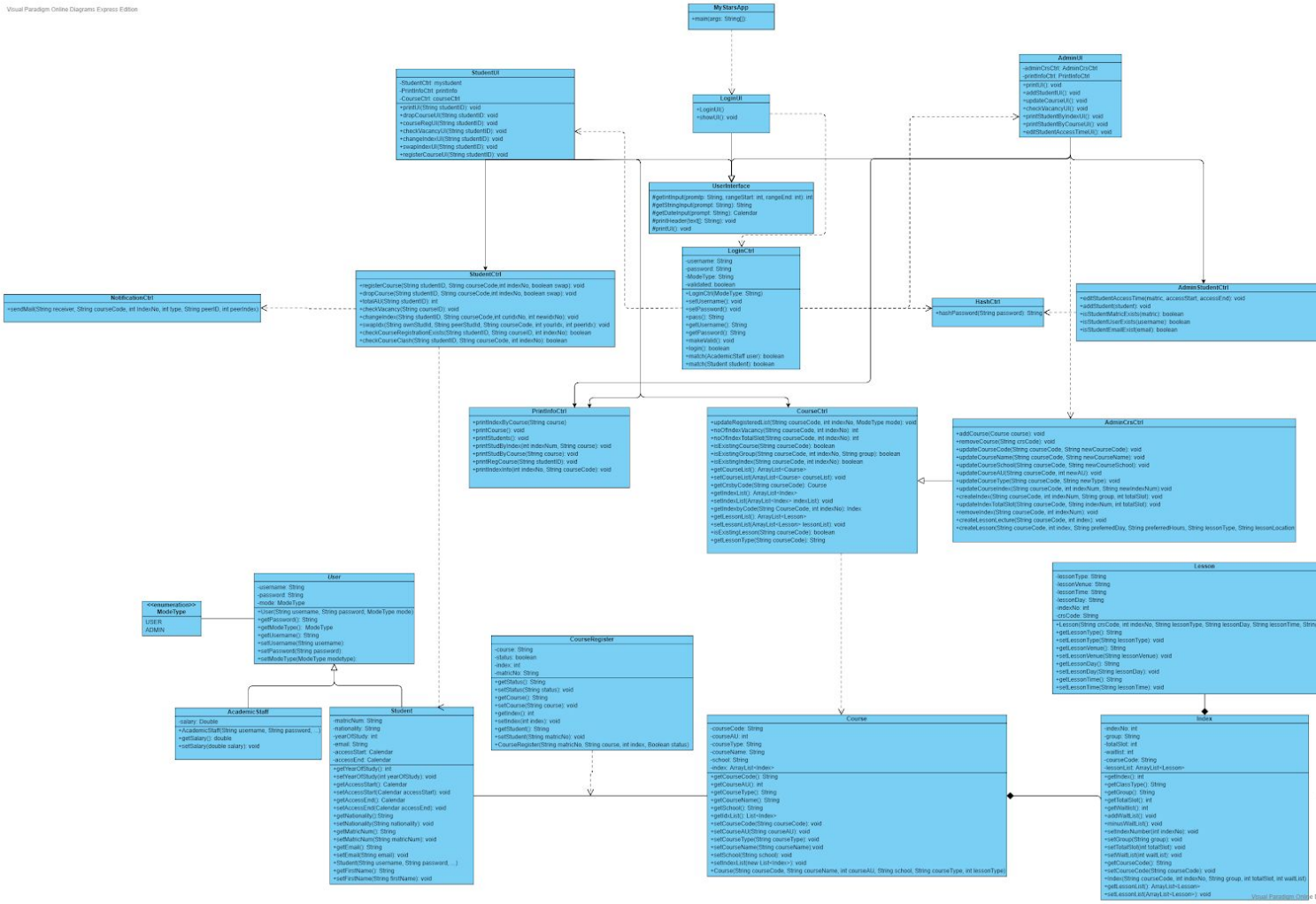
In implementing the code, several assumptions are made:

- Students are unable to update their particulars through the MySTAR portal as the portal is for them to register courses only.
- The default password for all students is the matriculation number.
- The priority of a vacancy goes to the first person on the waiting list.
- Students can only register a course if it is not clashed with existing courses including

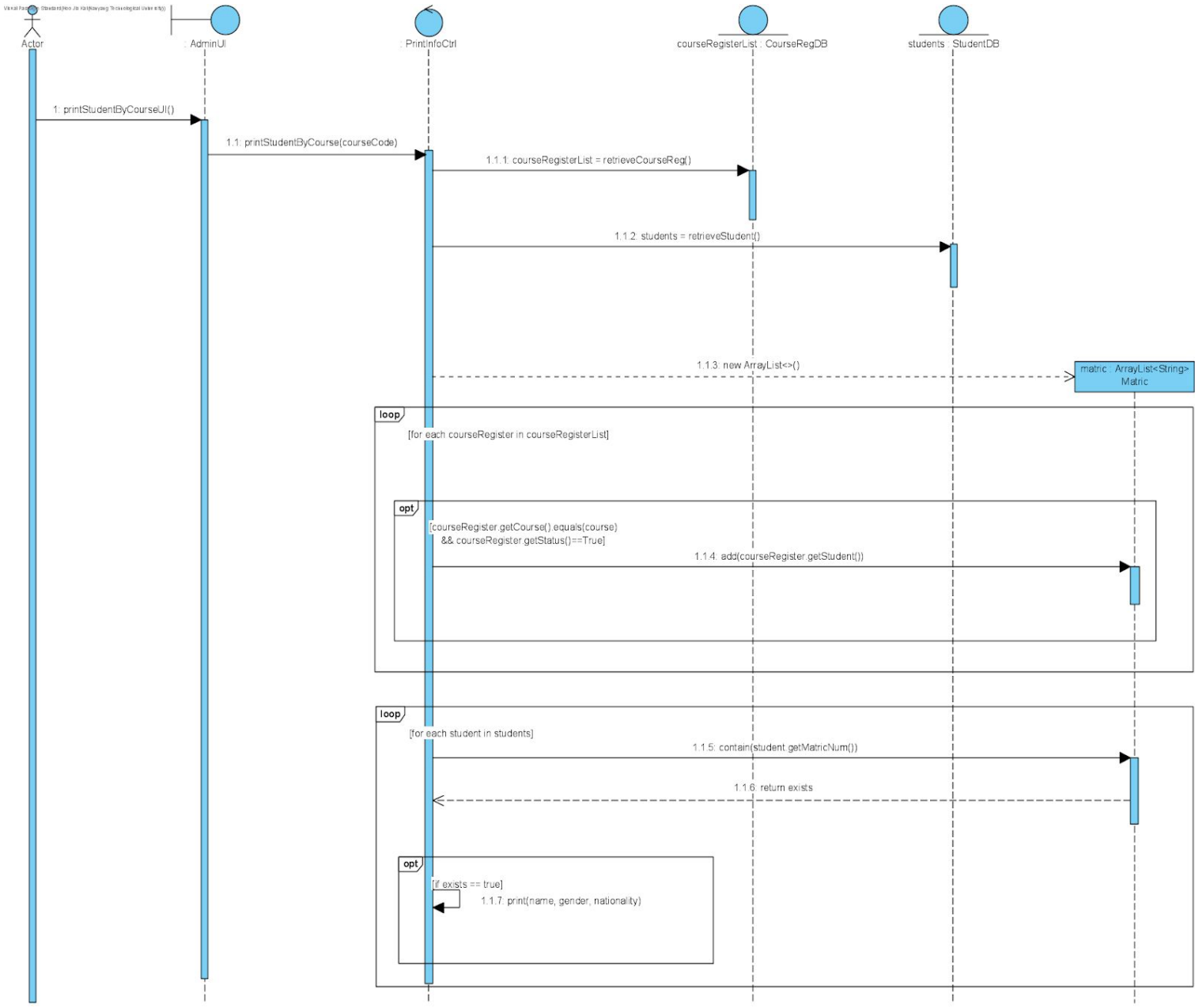
both registered and on waitlist courses.

- Students are only allowed to have up to 21 registered AU.

IV. UML Diagram



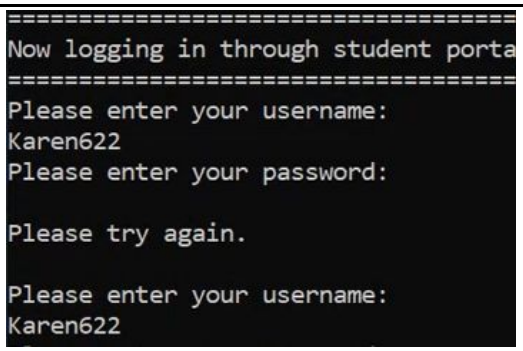
V. UML Sequence Diagram



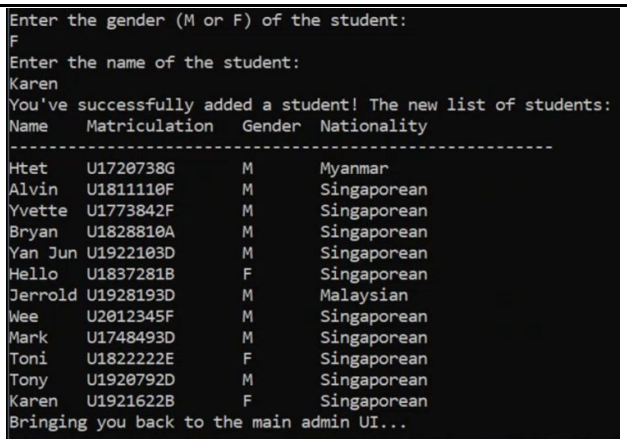
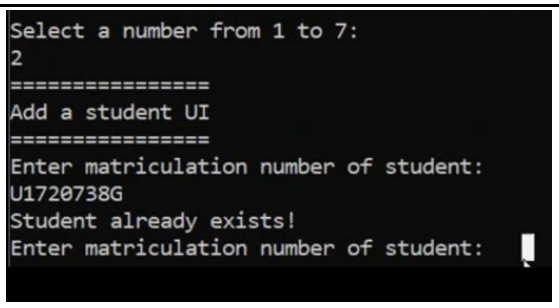
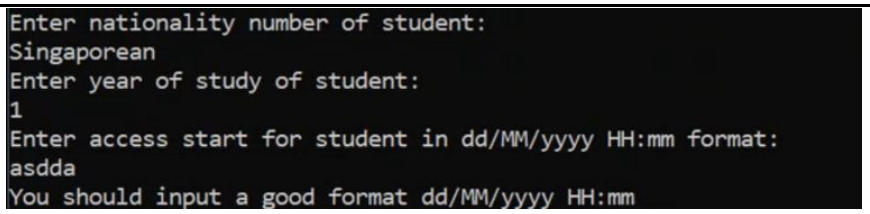
VI. Test Cases

1. Student Login

a) Login outside allowed **period (dates)**	Checking access time... Current time is 25/11/2020 19:19 valid access time! Successfully logged in! ==========
b) Login during allowed period (dates)	Checking access time... Current time is 25/11/2020 19:18 Invalid access time! Your access period is from 26/11/2020 11:30 to 26/11/2020 12:30 You have been logged out. Please try again.

c) Wrong Password	

2. Add a Student

a) Add a new student	
b) Add an existing student	
c) Invalid data entries	

3. Add a Course

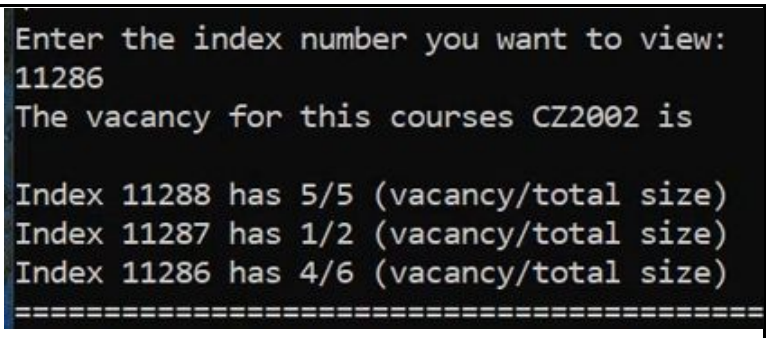
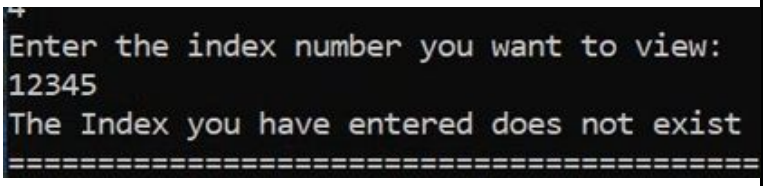
a) Add a new course	

b) Add an existing course	``` =============== Add a course UI =============== Enter Course Code: CZ3005 The course: CZ3005,Artifical Intelligence Course code already exists! ```
c) Invalid data entries	``` 1. Lectures only 2. Lectures and tutorials only 3. Lectures, tutorial and labs 4 Invalid range. Please enter from.... 1 to 3: 2 ```

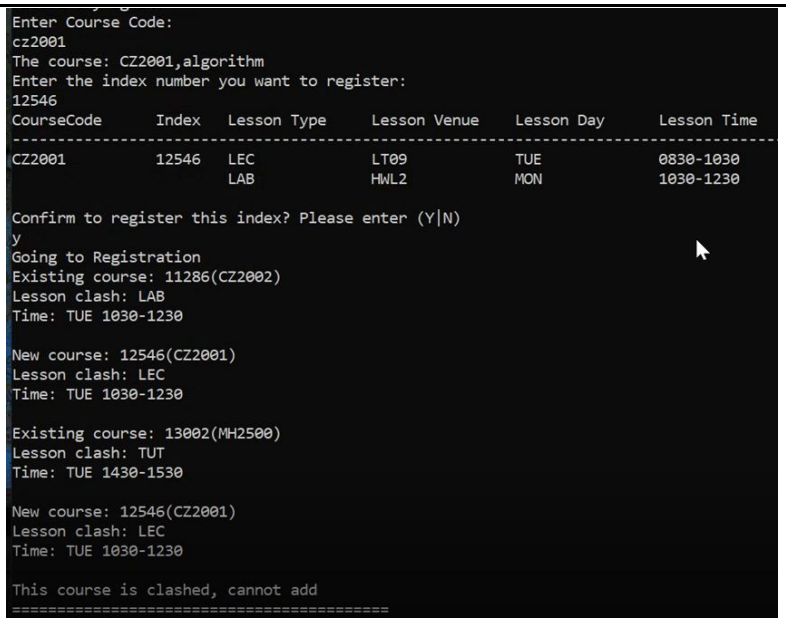
4. Register student for a course

a) Add a student to a course index with available vacancies	``` Enter Course Code: cz2002 The course: CZ2002,OODP Enter the index number you want to register: 11286 CourseCode Index Lesson Type Lesson Venue Lesson Day Lesson Time --- CZ2002 11286 LEC LT10 THUR 0830-1030 LAB HWL2 TUE 1030-1230 Confirm to register this index? Please enter (Y|N) y Going to Registration Index 11286 (CZ2002) has been successfully added! ```
b) Add a student to a course index with 0 vacancies in Tut / Lab	``` Enter Course Code: mh2500 The course: MH2500,Prob Enter the index number you want to register: 13002 CourseCode Index Lesson Type Lesson Venue Lesson Day Lesson Time --- MH2500 13002 LEC LT27 THUR 1730-1930 TUT TR+25 TUE 1430-1530 Confirm to register this index? Please enter (Y|N) y Going to Registration Sorry, the course has no vacancies any more. Student kelvinwahh wants to register MH2500 Due to lack of vacancy, your Index 13002 (MH2500) will be put into waiting list. ========== ```
c) Register the same course again	``` Enter Course Code: cz2002 The course: CZ2002,OODP Enter the index number you want to register: 11286 CourseCode Index Lesson Type Lesson Venue Lesson Day Lesson Time --- CZ2002 11286 LEC LT10 THUR 0830-1030 LAB HWL2 TUE 1030-1230 Confirm to register this index? Please enter (Y|N) y Going to Registration This student is already registered for this course. ```
d) Invalid data entries (eg wrong student ID / course code, etc)	``` Enter Course Code: cz200 Input courseCode does not exists! Please try again! ```

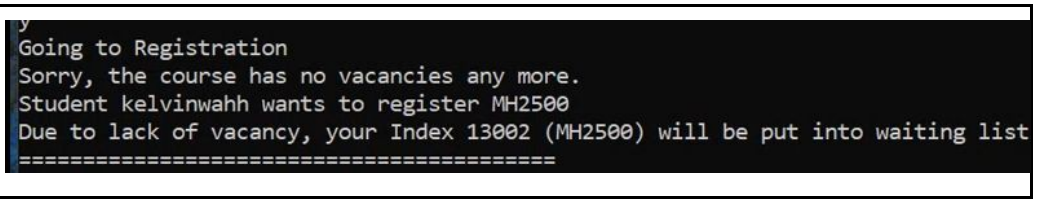
5. Check available slot in a class (vacancy in a class)

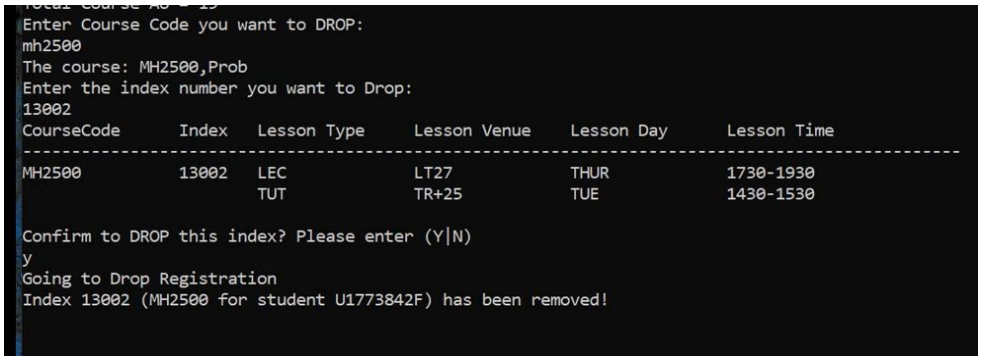
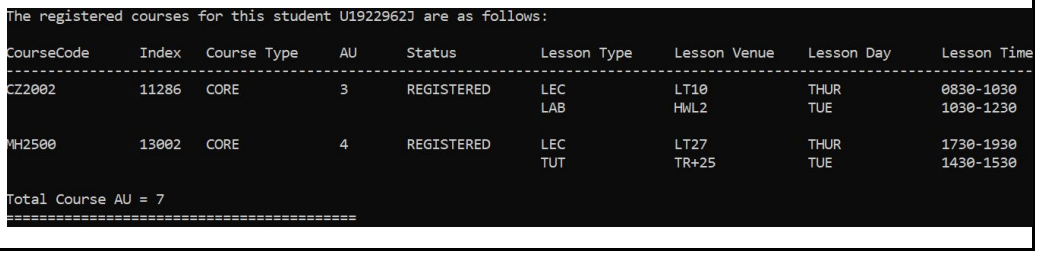
a) Check for vacancy in course index	
b) Invalid data entries (eg course code, class code etc)	

6. Day/Time clash with other course

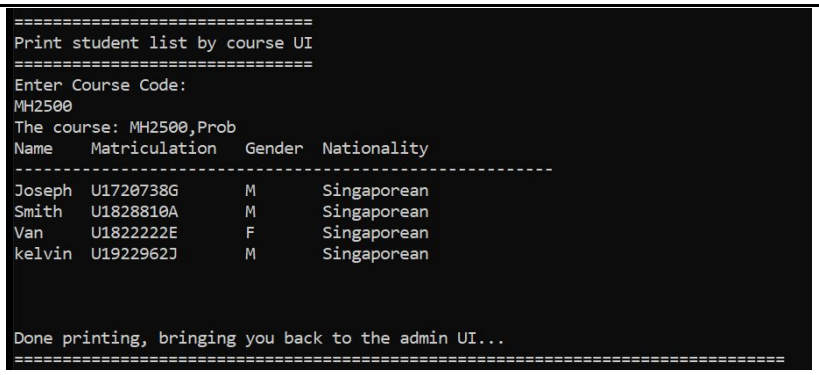
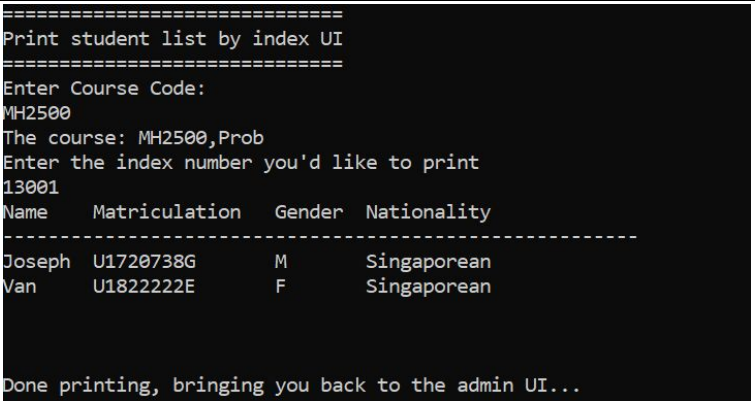
a Add a student to a course index with available vacancies.	

7. Waitlist notification

ai) Add studentA to a course index with 0 vacancies	

aii) Drop studentB from the same course index	
aiii) Display studentA timetable	

8. Print student list by index number, course

ai) Print list by Course	
aii) Print list by Index	
b) Invalid data entries (eg course code, index code etc)