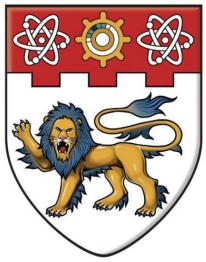


NANYANG TECHNOLOGICAL UNIVERSITY

CE/CZ2002: Object-Oriented Design & Programming

Building an OO Application

Lab Group: SS12
Assignment Group: Group 2

Team Members:

Bian Hengwei (U1923732B), He Yinan (U1922693C), Yin Jiarui (U1921776G), Lim Qian Hui (U1922364J), Tan Min, Tricia (U1922619J)

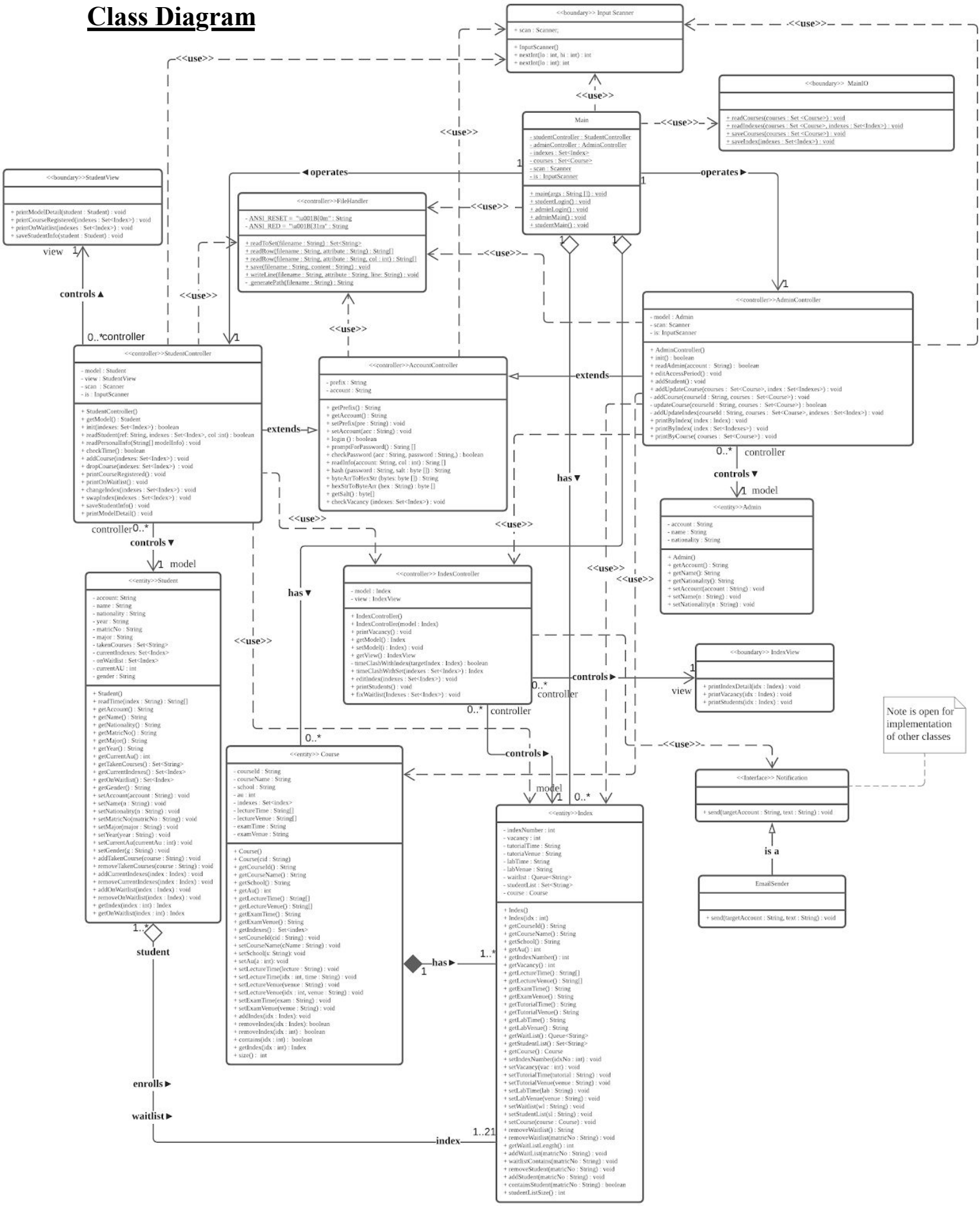
Declaration of Original Work for CE/CZ2002 Assignment

Name	Course Lab (CE2002 or CZ2002)	Lab Group	Signature/Date
Bian Hengwei	CZ2002	ss12	25/11/2020
He Yinan	CZ2002	ss12	25/11/2020
Yin Jiarui	CZ2002	ss12	25/11/2020
Lim Qian Hui	CZ2002	ss12	25/11/2020
Tan, Min Tricia	CZ2002	ss12	25/11/2020

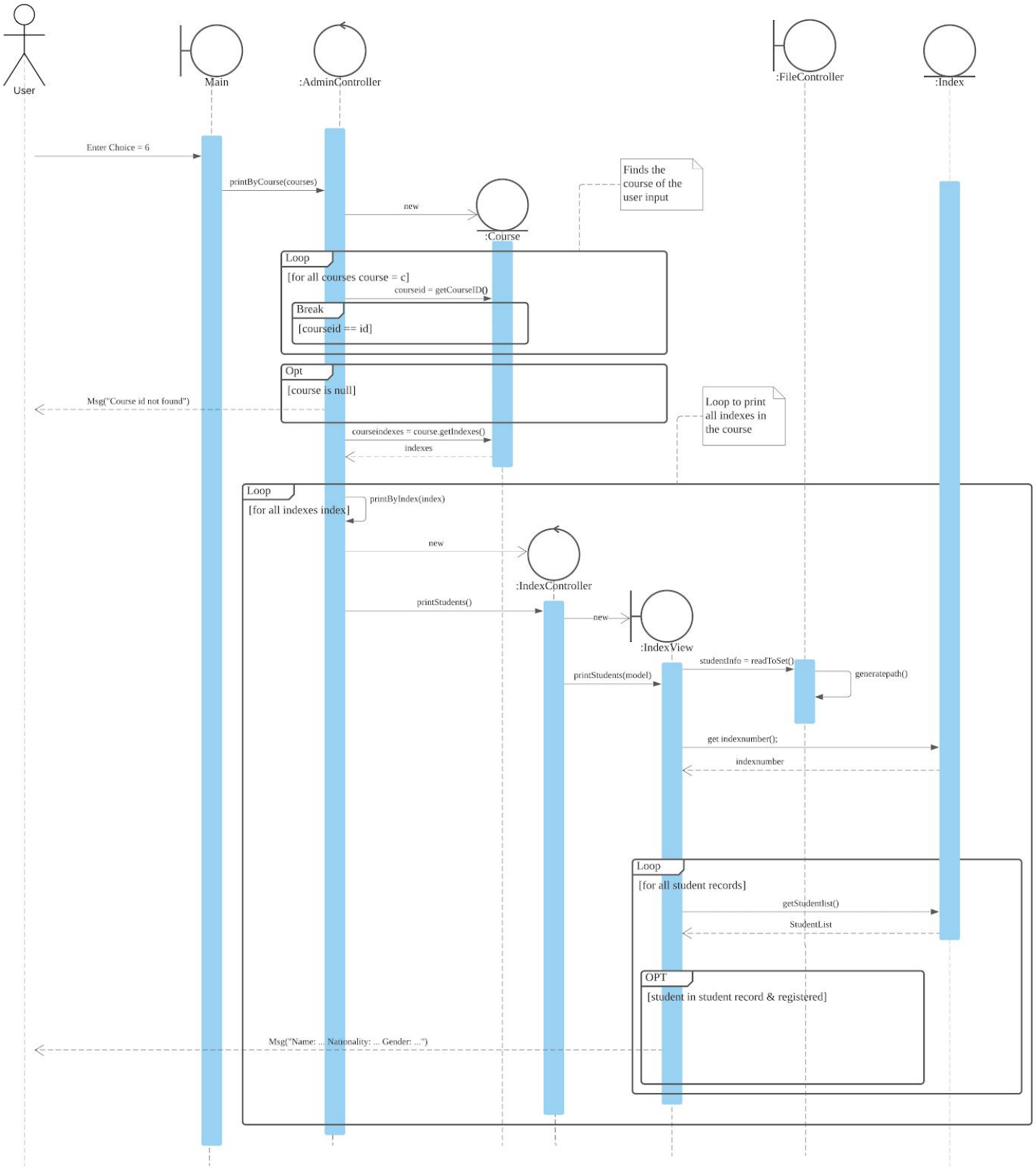
PROGRAM DEMO LINK: https://youtu.be/NH9xD7jQ4CA

Text

Class Diagram



UML Sequence Diagram



Design Considerations

Overall Design Flow

In our design, we used the Model View Controller design pattern to avoid rigidity and immobility. In our application, we divide all code into 5 subpackages: app, boundary, controller, model, and view. App contains only the Main file of the project, which is used to run the whole MYSTARS system. Boundary contains all classes that deal with interactions with the user and file system. Controller contains control classes that operate the entities and controls the view to print messages. For example, the “Student Controller” class carries the logic needed for implementing all the functions, such as "Add Course","Drop Course","Change Index Number of Course", when a user login as a student. “Student Controller” will call “Student” which is a Model, and get all the data stored in a particular student object. It will call “Student View” to display the information of this student object. Models deal with all the data stored inside an entity and only contain methods that modify the data (setter and getter). For example, the “Student” class is a Model. It contains all the necessary get and set methods to take in and return data of students such as “getAccount”, “getCurrentAu”, “setAccount”, “setCurrentAu”. Last but not least, the view takes in the model and prints all information stored inside. For example, “StudentView” class to display students’ information.

In our MYSTARS system, we have 2 types of user, administrator and student. We designed a model class for each of them to store their data read from the csv files. Since we need to perform a number of operations on the two types of users, we created Controllers for the student class and the administrator class, which handles all requests from their menu (in the main). Considering that student information and admin information both need to read data from the csv files, and they both perform a series of similar operations like login, read information, use their View classes to produce output to the console and the csv files etc. We wrote their shared methods in one superclass called AccountController which handles the shared methods of the two controllers.

The course class and index class are used to store and handle course data and index data. Course is a collection of indices, the collection of indices have some shared attributes of indices like the course name, the lecture time and venue, etc which will be stored in course class. Indices store

the individual data like vacancies and index numbers. Since we need to perform a lot of operations on index, including checking index clashes, fixing waitlists if there is vacancy, we created an index controller, which handles all index operations and generates output through Index View class.

We created some additional boundary classes to deal with different kinds of inputs and outputs. They are divided by their functionalities and they all have only one or two functions to keep their responsibility simple (Single Responsibility Principle). We have a Notification interface that provides a "send()" method to send notification to students. It is implemented by EmailSender class which sends email to the student NTU email account, this allows easy extension as adding other methods of notification only requires creating a class that implements Notification. The FileHandler class tries to read from and write to databases. The MainIO class constructs the courses and indices collection for the main program. The InputScanner class tries to deal with all simple numeric inputs and eliminate any invalid inputs.

In conclusion, since every class does its own work, making our program structure clear.

Principles

1. Single Responsibility Principle (SRP)

   We designed our classes such that each class only has one responsibility. We have split up the bigger classes into three classes. Model class only contains the setters and getters, it is responsible for the modification and retrieval of data of the object. View class contains all the reusable print functions, for example, StudentView class contains functions to print student information and functions to print the all courses a particular student is registered etc. The Controller class handles all the functions that are logic intensive, these methods are built using the model and view class, for example, StudentController class contains addCourse method and readStudent method.

2. Open Closed Principle (OCP)

   In our design, the Notification Interface makes use of the Open Closed Principle. It has a send method that takes in a target account and a piece of text. The interface is then implemented by EmailSender class, which implements the send method and sends an

email to the target account. It is open for extension in the way that we can add more classes to implement the interface, for example, a WhatsAppSender class that sends WhatsApp messages. It is closed for modification such that the new classes would not affect any of the previous classes.

3. Dependency Injection Pattern (DIP)

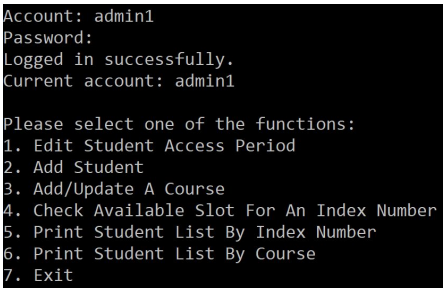
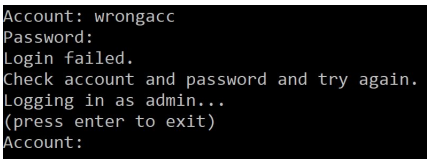
   The higher level classes in our design are independent of the lower level classes. For example, AccountController is a higher level class compared to StudentController and AdminController. AccountController is not dependent on these two classes and it can be reused quite simply should there be a new user type.

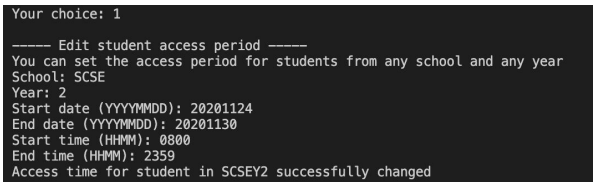
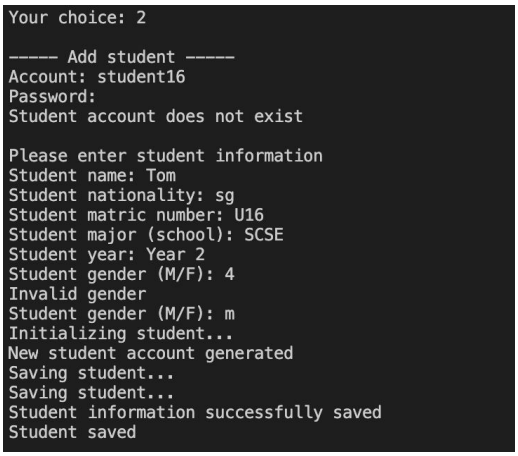
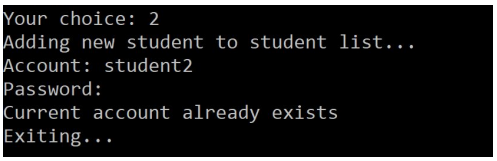
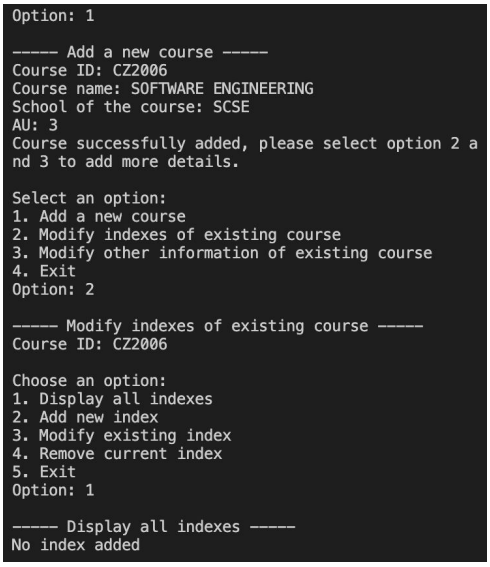
Assumptions made

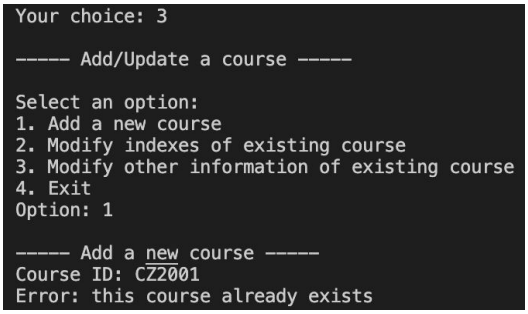
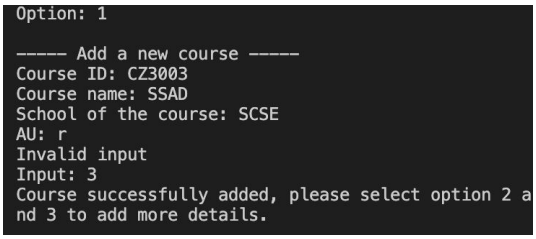
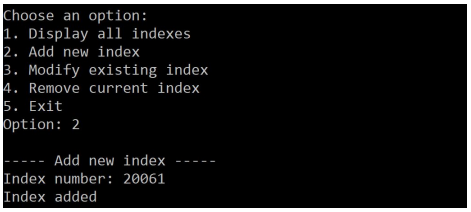
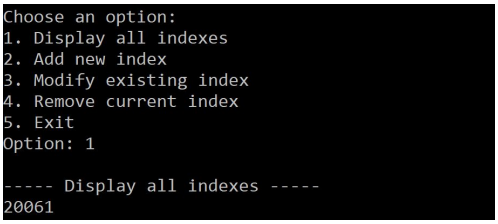
- All courses have 2 lectures. Courses with one two hours lecture could be split into two continuous one hour lectures.
- All labs are either on odd or even week.
- Email sent only when student move from waitlist to registered
- AU of all courses are between 1 and 10.
- The maximum AU students can register or put onto waitlist is 21. Sum of number of AU of registered course and number of AU of waitlisted course cannot exceed 21.

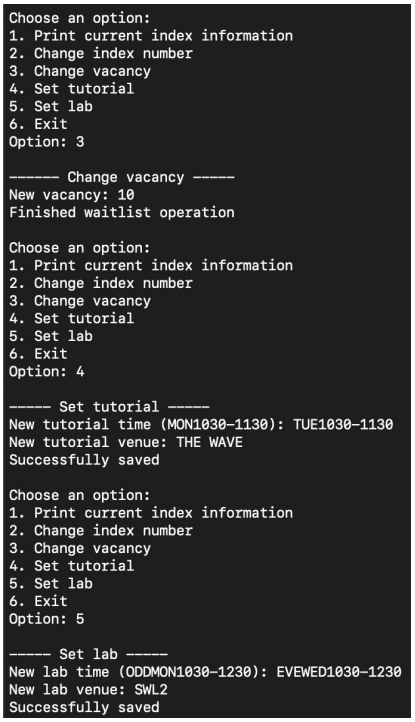
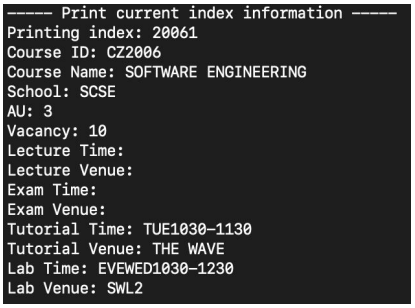
Testing

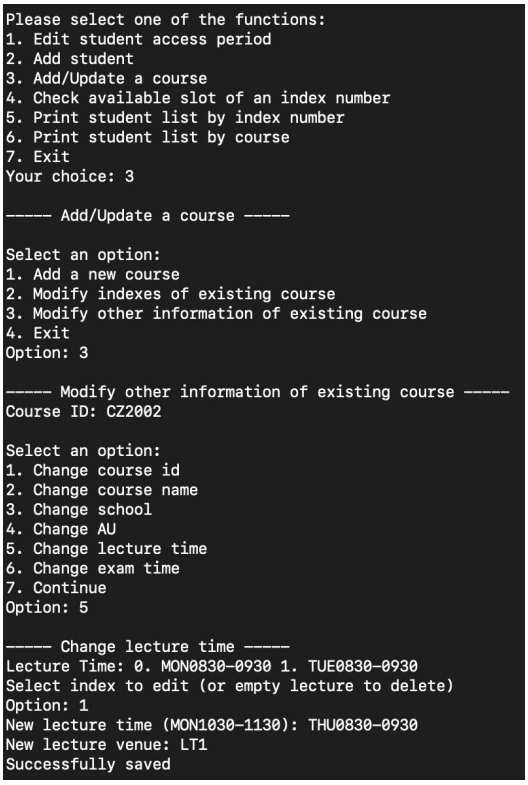
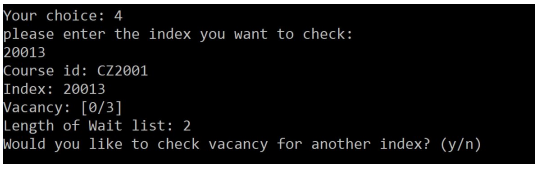
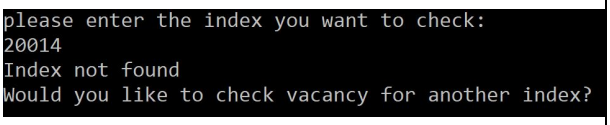
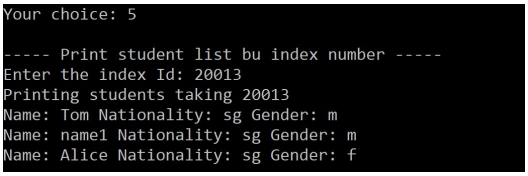
Admin Functions

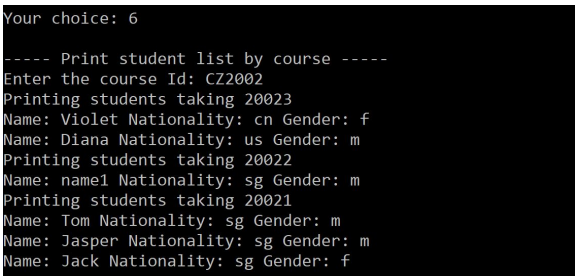
Function	Input	Expected Output
Login	Successful login Account: **admin1** Password: **pwd1**	
	Unsuccessful login Account: **wrongacc** OR Password: **wrongpwd**	

Edit Student Access Period	Your choice: **1** School: **SCSE** Year: **2** Start date: **20201124** End date: **20201130** Start time: **0800** End time: **2359**	
Add New Student	<u>Successful add student</u> You choice: **2** Account: **student17** Password: **pwd17** Student name: **John** Student nationality **:sg** Student matric number**: U17** Student major (school): **SCSE** Student year: Year **2** Student gender: **4 (Invalid gender)** Student gender: **m** **(test case with invalid entries)**	
	<u>Existing student account</u> Your choice: **2** Account: **student2** ***(already exists)*** Password: **pwd2**	
Add Course	<u>Adding a new course</u> Your choice: **3** Option: **1** Course ID: **CZ2006** Course name: **SOFTWARE ENGINEERING** School: **SCSE** AU: **3** Option: 2 **(Since the course is new, no index is found.)**	

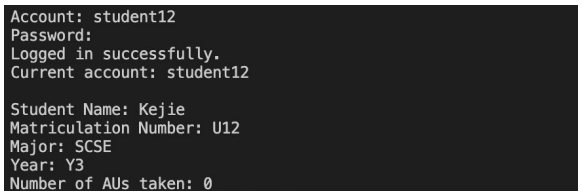
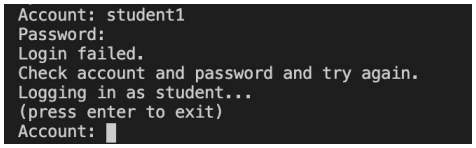
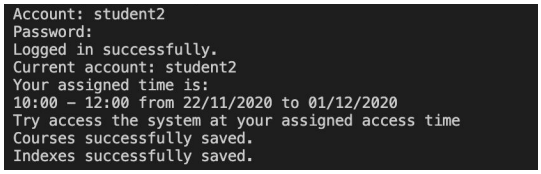
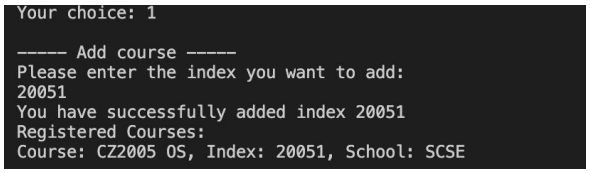
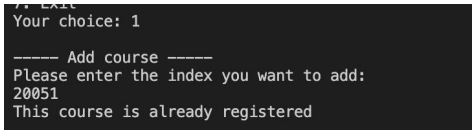
	Unsuccessful adding a new course Your choice: **3** Option: **1** Course ID: **CZ2001** **(This course already exists)**	
	Successful adding a new course Your choice: **3** Option: **1** Course ID: **CZ3003** School of the course: **SCSE** AU: **r (Invalid input)** AU: **3** **(test case with invalid entries)**	
Update Course	Adding indices to new course 'CZ2006' Your choice: **3** Option: **1** Index number: **20061** Updated information is reflected	 

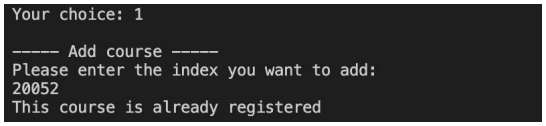
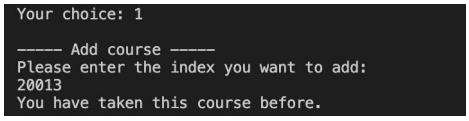
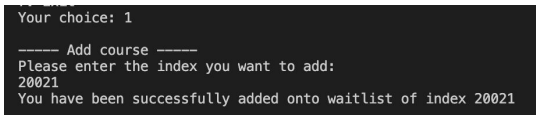
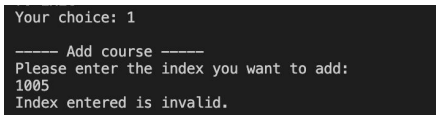
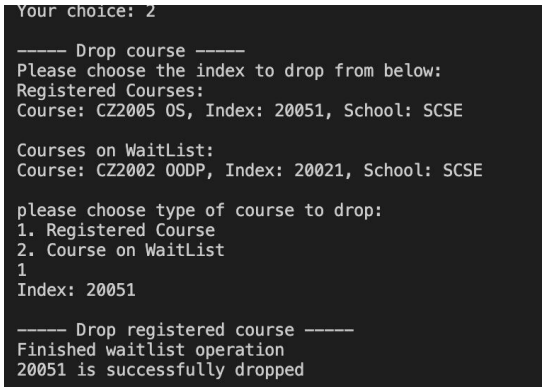
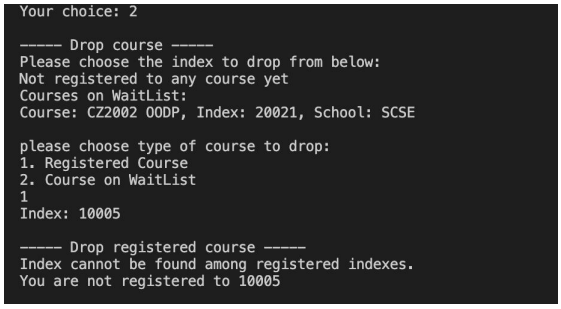
	<u>Updating index information to new index '20061' in 'CZ2006'</u> Your choice: **3** Option: **2** <u>Vacancy</u> Option: **3** New vacancy: **50** <u>Tutorial</u> Option: **4** Tutorial time: **TUE1030-1130** Tutorial venue: **THE WAVE** <u>Lab</u> Lab time: **EVEWsED1030-1230** Lab venue: **SWLAB2** <u>Check for updated information</u> Option : **1** The updated information is reflected	 

	<u>Updating lecture time of existing course ‘CZ2002’’</u> Option **3**: add/update a course Option **3**: modify other information of existing course Option **5** Option **0**: change the first lecture timing New lecture time: **THU0830-0930** New lecture venue: **LT1**	
Check Availabilities for Index	<u>Checking vacancy for index ‘20013’</u> Your choice: **4** Enter index to be checked: **20013**	
	<u>Checking vacancy for non-existent index</u> Enter index to be checked: **20014** ***(does not exist)***	
Print student list by Index	<u>Printing student list for index ‘20013’</u> Your choice: **5** Enter index Id: **20013**	
	<u>Printing student list for non-existent index</u> Enter index Id: **20014** ***(does not exist)***	

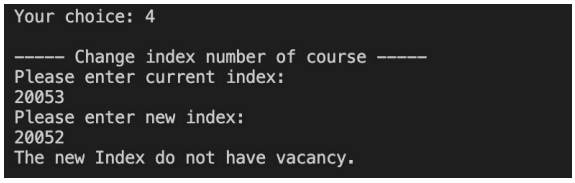
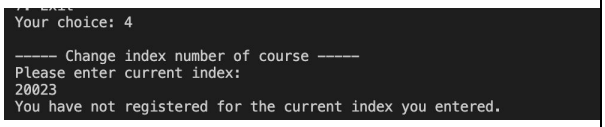
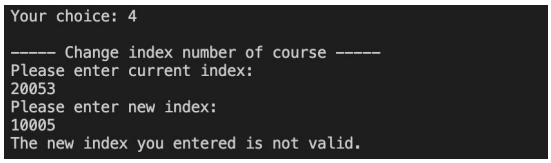
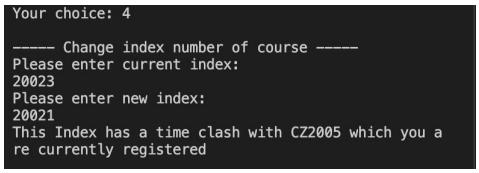
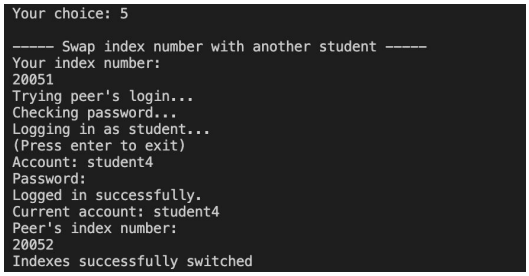
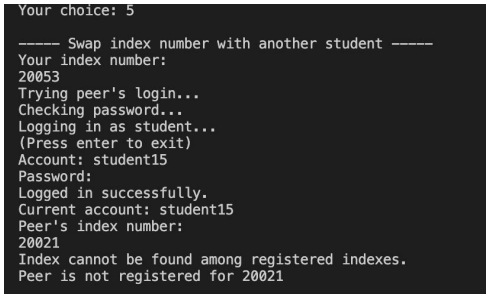
Print student list by course	Printing student list for course 'CZ2002' Your choice: **6** Enter course Id: **CZ2002**	
	Printing student list for non-existent course Enter course Id: **CZ1015** ***(does not exist)***	

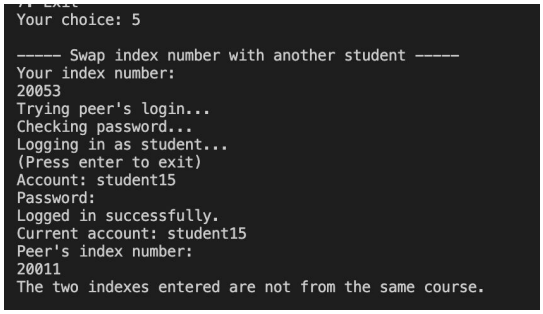
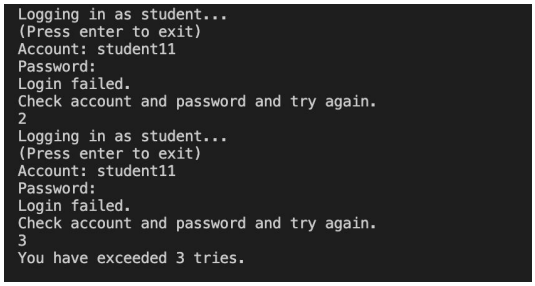
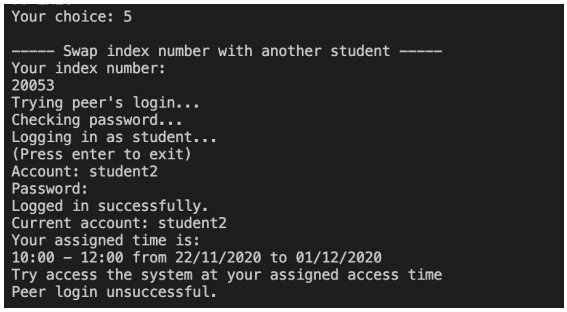
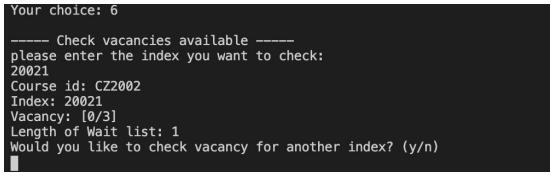
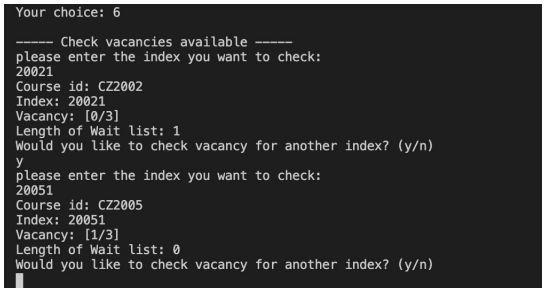
Students Functions

Function	Input	Expected Output
Student Login	Successful login Account: **student12** Password: **pwd12**	
	Unsuccessful login Account: **student1** Password: **wrongpwd** **(incorrect password or account)**	
	Unsuccessful login Account: **student2** Password: **pwd2** **(Login not between 10:00 and 12:00, not during assigned time)**	
Add Course (the test cases are generated using different student accounts)	Course registered Your choice: **1** Index: **20051**	
	Course already registered Your choice: **1** Index: **20051**	

	<u>Course already registered, student add a different index for the same module</u> Your choice: **1** Index: **20052**	
	<u>Taken course before</u> Your choice: **1** Index: **20013**	
	<u>Time clash</u> Checks exam, tutorial, lecture and lab time clash	
	<u>No Vacancy, add to waitlist</u> Your choice: **1** Index: **20021**	
	<u>Invalid index</u> Your choice: **1** Index: **1005**	
Drop Course	<u>Course registered dropped</u> Your choice: **2** Type of course: **1** Index: **20051**	
	<u>Invalid index</u> Your choice: **2** Type of course: **1** Index: **10005**	

	<u>Invalid index on waitlist</u> Your choice: **2** Type of course: **2** Index: **1003**	Your choice: 2 ----- Drop course ----- Please choose the index to drop from below: Not registered to any course yet Courses on WaitList: Course: CZ2002 OODP, Index: 20021, School: SCSE please choose type of course to drop: 1. Registered Course 2. Course on WaitList 2 Index: 1003 ----- Drop waitlisted course ----- Index cannot be found among waitlisted indexes. You are not registered to 1003
	<u>Course on waitlist dropped</u> Your choice: **2** Type of course: **2** Index: **20021**	Your choice: 2 ----- Drop course ----- Please choose the index to drop from below: Not registered to any course yet Courses on WaitList: Course: CZ2002 OODP, Index: 20021, School: SCSE please choose type of course to drop: 1. Registered Course 2. Course on WaitList 2 Index: 20021 ----- Drop waitlisted course ----- 20021 is successfully dropped
Check/Print Course Registered and on Wait list	<u>Course registered and course registered on waitlist</u> Your choice: **3**	Your choice: 3 ----- Check/Print courses registered and on waitlist ----- Registered Courses: Course: CZ2005 OS, Index: 20051, School: SCSE Courses on WaitList: Course: CZ2002 OODP, Index: 20021, School: SCSE
	<u>No course registered and no course registered on waitlist</u> Your choice: **3**	7. Exit Your choice: 3 ----- Check/Print courses registered and on waitlist ----- Not registered to any course yet Not registered on any waitlist
Change Index Number of Course	<u>Successful Change</u> Your choice: **4** Current Index: **20051** New Index**: 20053** Confirm the change: **y** **(confirm change)**	----- Change index number of course ----- Please enter current index: 20051 Please enter new index: 20053 --- Current Index Information --- Printing index: 20051 Course ID: CZ2005 Course Name: OS School: SCSE AU: 3 Vacancy: 0 Lecture Time: WED2100-2200 THU2100-2200 Lecture Venue: LT2A Exam Time: 12030830-1030 Exam Venue: LT3 Tutorial Time: FRI2100-2200 Tutorial Venue: TR15 Lab Time: EVEFRI0130-0330 Lab Venue: SWL2 --- New Index Information --- Printing index: 20053 Course ID: CZ2005 Course Name: OS School: SCSE AU: 3 Vacancy: 1 Lecture Time: WED2100-2200 THU2100-2200 Lecture Venue: LT2A Exam Time: 12030830-1030 Exam Venue: LT3 Tutorial Time: FRI0930-1130 Tutorial Venue: TR03 Lab Time: EVEFRI1430-1630 Lab Venue: HWLAB1 Please enter y to confirm the change, enter n to cancel. y You have successfully changed from index 20051 to index 20053
	<u>Unsuccessful Change</u> unconfirm change	Please enter y to confirm the change, enter n to cancel. n Changing of indexes cancelled.

	Unsuccessful Change Your choice: **4** Current Index: **20053** New Index**: 20052** **(no vacancy)**	
	Unsuccessful Change Your choice: **4** Current Index: **20023** **(invalid current index)**	
	Unsuccessful Change Your choice: **4** Current Index: **20053** New Index**: 10005** **(invalid new index)**	
	Unsuccessful Change Your choice: **4** Current Index: **20053** New Index**: 20021** **(Index has a time clash)**	
Swap Index With Another Student	Successful Swap Your choice: **5** Current Index: **20053** Peer's Account: **student4** Peer's Password: **pwd4** Peer's Index: **20051**	
	Unsuccessful Swap Your choice: **5** Current Index: **20053** Peer's Account: **student15** Peer's Password: **pwd15** Peer's Index: **20021** **(Peer is not registered that index)**	

	<u>Unsuccessful Swap</u> Your choice: **5** Current Index: **20053 (from CZ2005)** Peer's Account: **student15** Peer's Password: **pwd15** Peer's Index: **20011(from CZ2001)** **(Two indices entered are not from the same course)**	
	<u>Unsuccessful Swap</u> **Incorrect peer password and account, reach 3 attempts**	
	<u>Unsuccessful Swap</u> Your choice: **5** Current Index: **20053** Peer's Account: **student2** Peer's Password: **pwd2** **(Not peer's assigned time)**	
Check Vacancies Available	<u>Successful Check</u> Your choice: **6** Index: **20021**	
	<u>Successful Check</u> Your choice: **6** Index: **20021** Check Vacancy for another index: **y** Index: **20051**