2016/2017
SEM 2

PROJECT REPORT

CZ2002: OBJECT-ORIENTED DESIGN & PROGRAMMING

PREPARED BY:

FSP2, GROUP 5

ALVIN LEE YONG TECK	U1620768F
HTET NAING	U1620683D
LIM YAN JUN	U1622311B
LIU JIHAO BRYAN	U1621401C
TONI MIHARJA	U1521012E

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

APPENDIX B:

Attached a scanned copy with the report with the filled details and signatures.

Declaration of Original Work for CE/CZ2002 Assignment

We hereby declare that the attached group assignment has been researched, undertaken, completed and submitted as a collective effort by the group members listed below.

We have honored the principles of academic integrity and have upheld Student Code of Academic Conduct in the completion of this work.

We understand that if plagiarism is found in the assignment, then lower marks or no marks will be awarded for the assessed work. In addition, disciplinary actions may be taken.

Name	Course (CE2002 or CZ2002)	Lab Group	Signature /Date
ALVIN LEE YONG TECK	CZ2002	FSP2	[illegible] / 16-03-17
HTET NAING	CZ2002	FSP2	[illegible] / 16-03-17
Toni Miharja	CZ 2002	FSP2	[illegible] / 16-03-17
Liu Jihao Bryan	CZ 2002	FSP2	[illegible] / 16-03-17
Lim Yan Jun	CZ2002	FSP2	[illegible] / 16-03-17

Important notes:

1. Name must **EXACTLY MATCH** the one printed on your Matriculation Card.

Table of Contents

I. Introduction

My STudent Automated Registration System (MySTARS) is a console-based application designed and develop for both staff and student to manage registration of courses. The application covers the key features such as creation of new courses, registration of courses and addition of student records.

This report covers the object-oriented programming (OOP) concepts and key design considerations and used to implement the application. The design will also be represented in a UML Class Diagram and UML Sequence Diagram for one of the features, showing the interaction and relationship between the objects. Moreover, several test cases are included as well to ensure that the application meet the requirements stated beforehand.

II. Design Considerations

a. Approach Taken

OOP concepts are applied comprehensively in this project, in both the design and the implementation of the MySTARS application. As mentioned by Kernighan and Plauger, our team aims to make our application "**easy to maintain and modify**".

The architectural style that we have taken is the **n-tier architectural style** where higher layers make use of services provided by lower layers but lower layers are independent of higher layers. As can be seen in *Figure 1*, in our case, the *UI* classes depend on the *Manager* classes which then interact with the *Data* file for storage. This further reinforces separation of concerns and make sure that the design is easy to maintain and modify.

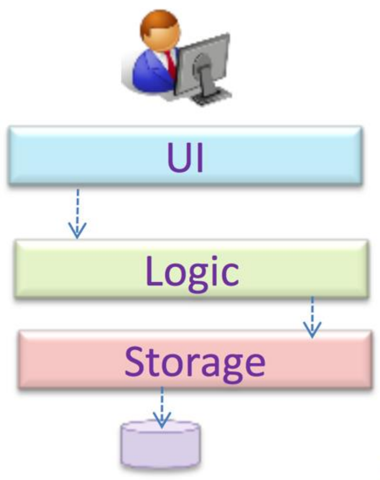


Figure 1: Architectural Design

b. Design Principles Used

b.1. Single Responsibility Principle (SRP)

The principle states that there should be no more than one reason for a class to change, making the code more cohesive. Our group makes sure that each class only has one responsibility only.

As an example, the Course class only manages the attributes of a single course – index, course code, course name, etc. Similarly, the Student class only manages the attributes of a single student. This principle is continuously applied in the design, in the UI and Manager classes as well, ensuring cohesiveness. This reduces functional overlaps and also limits the ripple effect when changes are introduced to a specific part of the system.

b.2. Open-Closed Principle (OCP)

Our group applied OCP in the implementation of some of our modules, 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 that extends to Staff class and Student class which can be seen in *Figure 2* 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 courses details can be done without changing the source code of the modules.

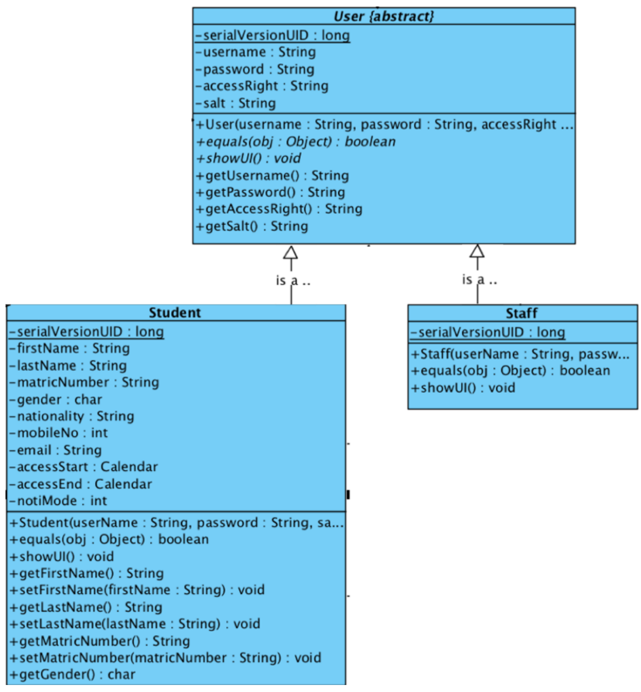


Figure 2: Abstraction of User Class

b.3. Liskov Substitution Principle (LSP)

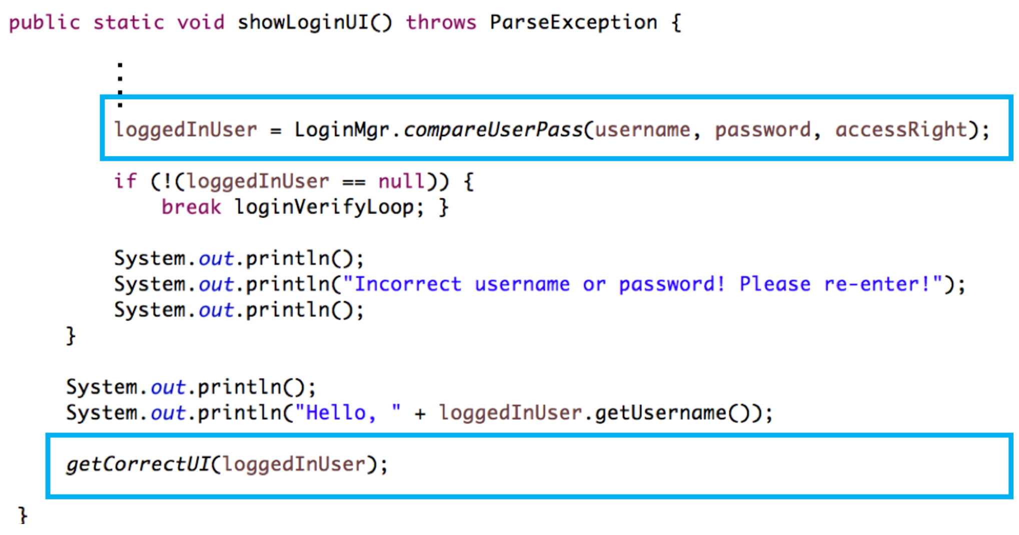

```
public static void showLoginUI() throws ParseException {

        ⋮
        loggedInUser = LoginMgr.compareUserPass(username, password, accessRight);

        if (!(loggedInUser == null)) {
            break loginVerifyLoop; }

        System.out.println();
        System.out.println("Incorrect username or password! Please re-enter!");
        System.out.println();
    }

    System.out.println();
    System.out.println("Hello, " + loggedInUser.getUsername());

    getCorrectUI(loggedInUser);

}
```

Figure 3: showLoginUI()

LSP, an extension of the OCP, is also implemented in the application. For example, the derived classes (*Student* and *Staff)* is **substitutable** for the base class, *User*, while retaining the original behaviour of the class. We made sure that the derived class's pre-conditions are no stronger than the base class method and its post-conditions are no weaker than the base class method. For example, the method *getCorrectUI* takes in user class as well as the staff and student staff to get the correct UI for the right type of user. This shows that the derived classes are substitutable to the base class.

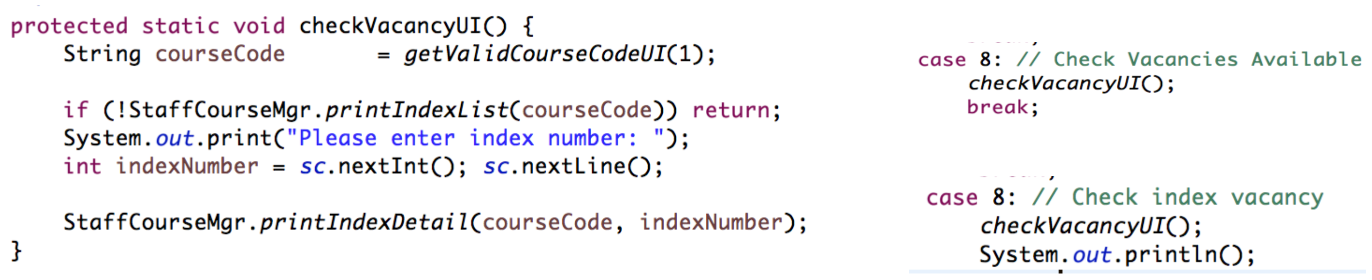
b.4. Interface Segregation Principle (ISP)

In our design, we make sure that the classes do not depend on interfaces that they do not use and we avoid fat interfaces. For example, the different *manager* classes are segregated by their functions and they do not depend on any interface that they do not need. This is mainly because of the segregation of responsibilities that we have established for each classes that allows for a good segregation of duties.

b.5. Don't Repeat Yourself (DRY)

The DRY principle states that every piece of knowledge must have a single, unambiguous, authoritative representation within a system. In our application, we make sure that there is no duplication of codes and functionality, encouraging code reuse and efficiency.

For example, we used inheritance with the parent class *CourseMgr* so that the methods which are applicable can be accessed by both the children *StudentCourseMgr* and *StaffCourseMgr*. This removes the need for duplication as methods like *printIndexInfo* and *getCourseByCode* can be used by both the child classes.



```java
protected static void checkVacancyUI() {
    String courseCode       = getValidCourseCodeUI(1);

    if (!StaffCourseMgr.printIndexList(courseCode)) return;
    System.out.print("Please enter index number: ");
    int indexNumber = sc.nextInt(); sc.nextLine();

    StaffCourseMgr.printIndexDetail(courseCode, indexNumber);
}
```

```java
case 8: // Check Vacancies Available
    checkVacancyUI();
    break;
```

```java
case 8: // Check index vacancy
    checkVacancyUI();
    System.out.println();
```

Figure 4: checkVacancyUI is reused multiple times in the application

Moreover, even in our UI, for functions that are repeated a lot, we make sure that there is only a single representation within the system. For example, the methods *checkVacancyUI* and *getValidCourseCodeUI* are available in the *CouresUI* (parent class) which makes sure that the the functions are represented once in the entire application, instead of repeating the code.

c. Object-Oriented Concepts (Explanation of UML Diagram)

c.1. Composition

We implemented composition relationship between *Course* with *Index* with *Lesson.* This is because index will not exist without course and lesson will not exist without index. In detail, *Course* has a one-to-many relationship with *Index* and *Index* has a one-to-many relationship with *Lesson*. This represent a 'whole-part' relationship. As such, in implemented *removeCourse()*, all the indexes will be removed automatically and the same principle applies to removing an index of a course.

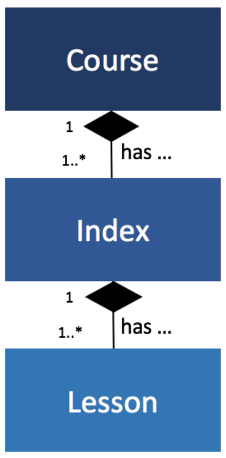


Figure 5: Composition relationship between course, index and lesson

c.2. Association Class

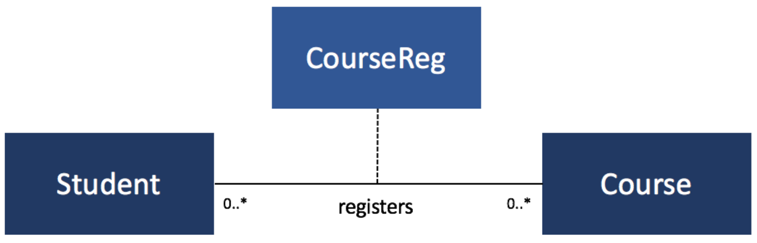


Figure 6: CourseReg Association Class

Association class *CourseReg* is used to keep track of the courses registered by each student. This is because there is a need to store additional information about an association but registration data is not specifically owned by either the *Student* or the *Course* object. As such, an additional class is introduced in this case, the *CourseReg* class.

c.3. Inheritance

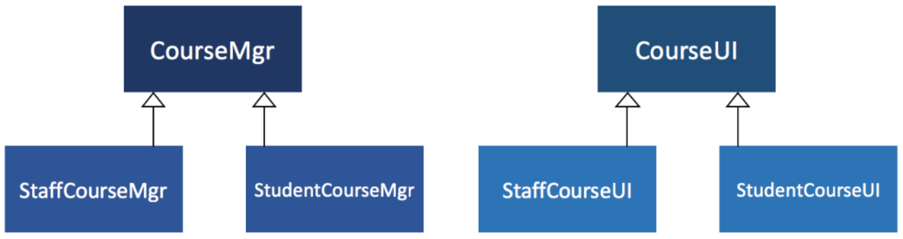


Figure 7: Inheritance

Inheritance is implemented in several classes in the design. For example, *CourseMgr* is the the parent class of *StaffCourseMgr* and *StudentCourseMgr* as both implement common methods so as to encourage code reuse. For instance, both child classes can use methods such as *printIndexDetail* and *printIndexList* which are inherited from the parent class.

The same is applied to *CourseUI* as the parent class of *StaffCourseUI* and *StudentCourseUI*. Both are able to use methods which are inherited from the parents and override any method as needed.

c.4. Encapsulation/ Information Hiding

Encapsulation builds a conceptual barrier to protect an object's private data. The entities created have private attributes which are only accessible through the get and set methods, making them a read-only class. Information hiding also hides details of the class from the users.

For example, the *Student* class has private attributes such as firstName, lastName, matricNumber, nationality, etc. As they are private, they are only accessible through their respective getter and setter methods. As such, *Student* class has full control of what is stored in its private fields.

c.5. Abstraction

Abstraction is giving information and essential characteristics that distinguish the class from other objects, relative to the point of view of the viewer, making the coding process much easier by reducing the general complexity of the code. For example, the *User* class is an abstract class and the implementation of *getUI()* method is provided by the derived classes *Student* and *Staff*. Moreover, as stated in section b, this further allows for extensibility in the future as new types of users can be added without modifying the method that calls the classes, reinforcing OCP.

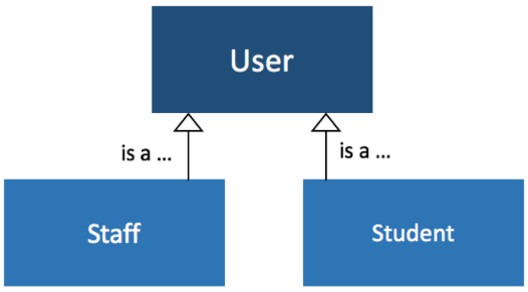


Figure 8: Abstraction

c.6. Polymorphism

Polymorphism is when different types of objects are treated as a single general type, but yet each type of object exhibits a different kind of behavior. This is implemented in the *User* class where the *getUI()* method are called depending on the the child classes (*Staff* and *Student*). The child classes will override the methods in the parent class during runtime. This enhances the functionality of the program as the behaviour of the subclasses can be extended in the future to ensure that extension can be done in the future.

d. Data Structure

Firstly for file IO, we serialize and deserialize objects to file (.dat). This is preferred over the text file as the code implemented is cleaner and only one .dat file is needed for the entire system.

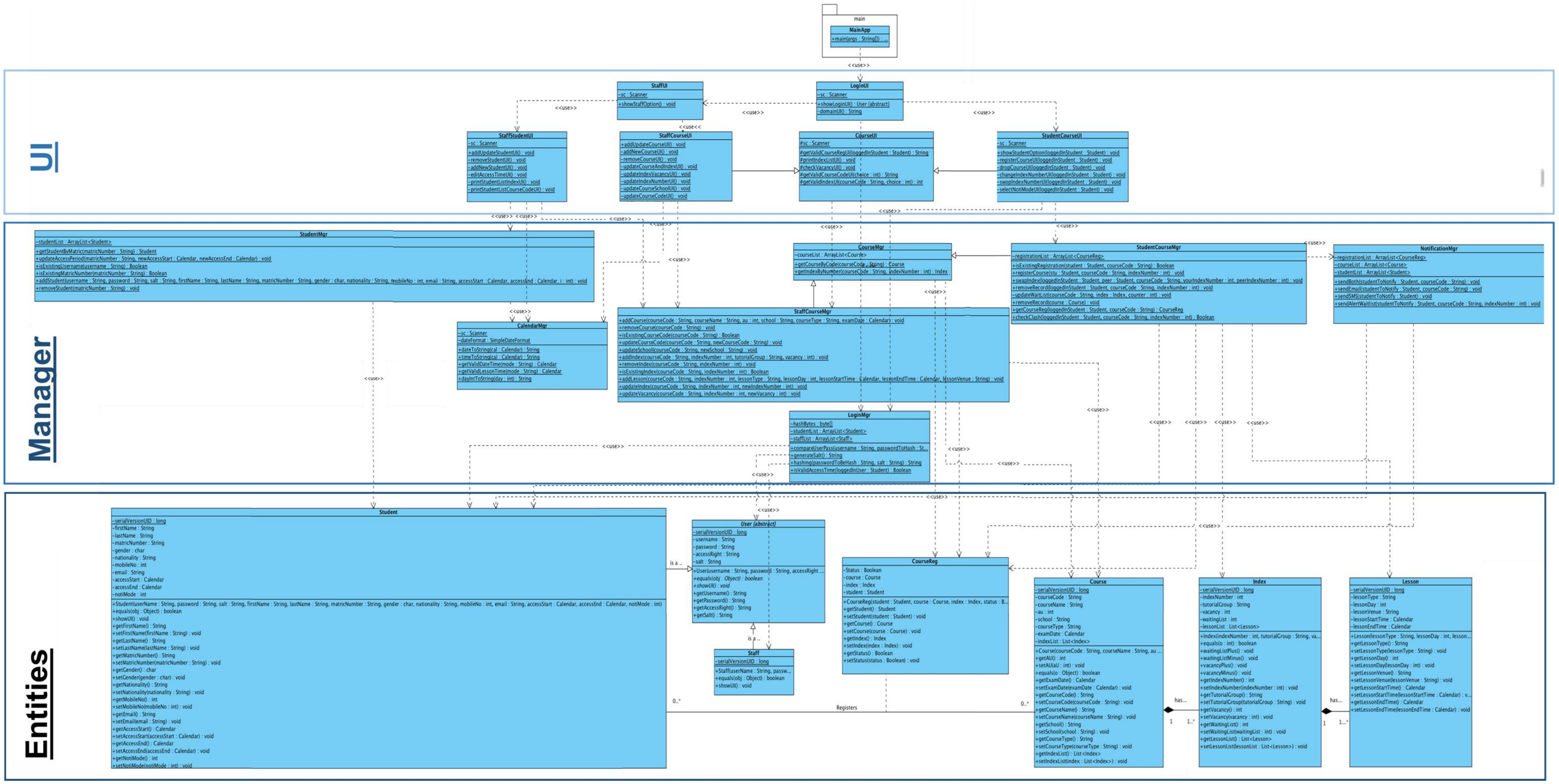
In implementing the waitlist, a **Stack** is implemented where the top of the list is popped out when a new vacancy is present. As such, the waitlist is implemented on a first-come-first-serve basis.

III. Assumption

In implementing the code, several assumptions are made:

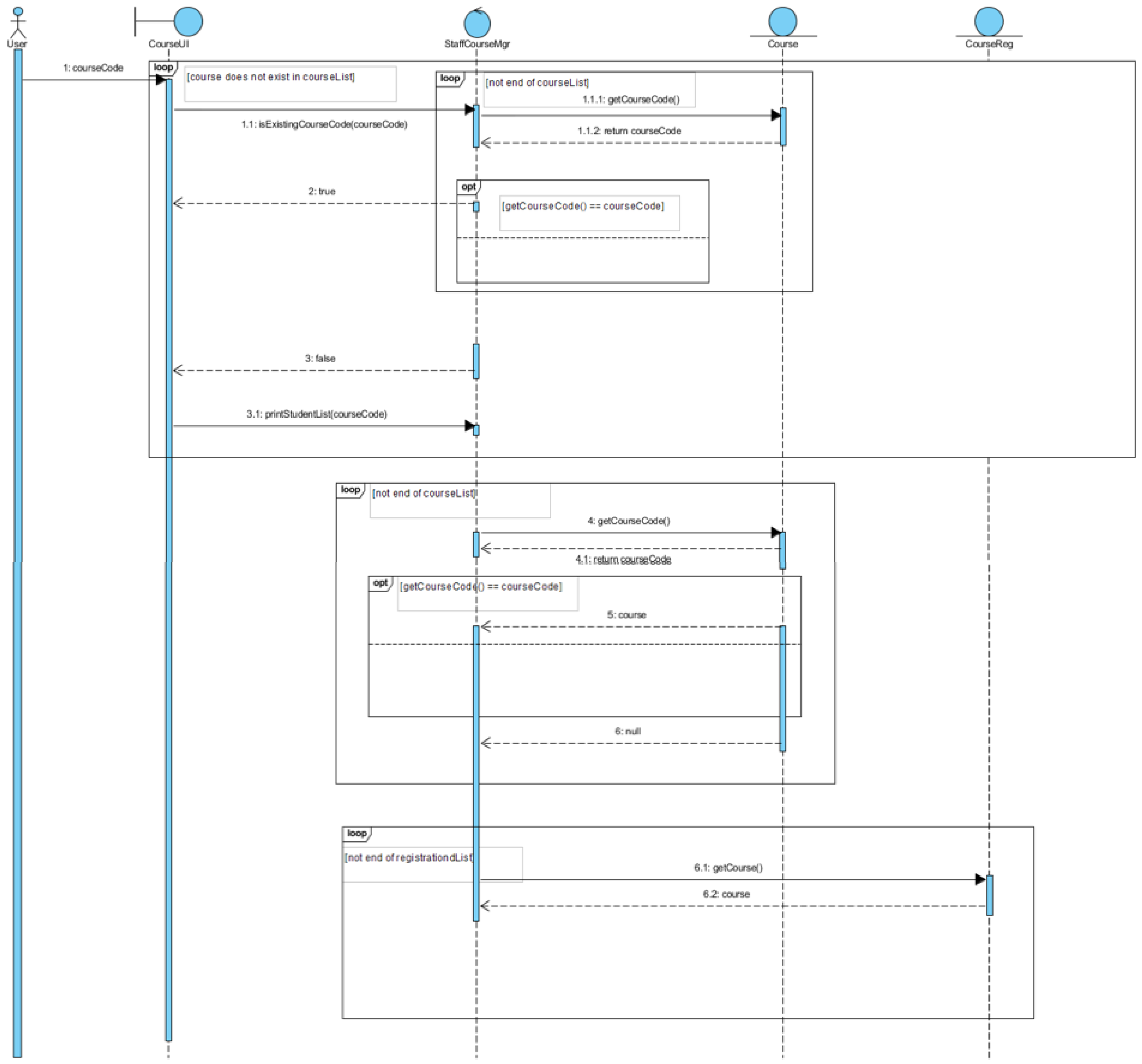
- All Staff holds the same level of authority as other staff in accessing the MySTAR application.
- Students are unable to update their particulars through the MySTAR portal as the portal is for them to register courses only. Students can, however, modify their notification modes.
- The default password for all students are the matriculation number. Students are assumed to be able to change this default password on another portal, not through MySTAR.
- Examination timings are the same everyday, meaning it is always at 9.30 am, 1.30 am and 4.30 pm from Monday to Friday.
- When swapping index number, user must input the other person's username and password to continue.

IV. UML Diagram



main
MainApp
+main(args : String[]) : ...
<<use>>
UI
StaffUI
-sc : Scanner
+showStaffOption() : void
LoginUI
~sc : Scanner
+showLoginUI() : User {abstract}
-domainUI() : String
StaffStudentUI
-sc : Scanner
+addUpdateStudentUI() : void
-removeStudentUI() : void
-addNewStudentUI() : void
-editAccessTimeUI() : void
-printStudentListIndexUI() : void
-printStudentListCourseCodeUI() : void
StaffCourseUI
+addUpdateCourseUI() : void
-addNewCourseUI() : void
-removeCourseUI() : void
-updateCourseAndIndexUI() : void
-updateIndexVacancyUI() : void
-updateIndexNumberUI() : void
-updateCourseSchoolUI() : void
-updateCourseCodeUI() : void
CourseUI
#sc : Scanner
#getValidCourseRegUI(loggedInStudent : Student) : String
#printIndexListUI() : void
#checkVacancyUI() : void
#getValidCourseCodeUI(choice : int) : String
#getValidIndexUI(courseCode : String, choice : int) : int
StudentCourseUI
-sc : Scanner
+showStudentOption(loggedInStudent : Student) : void
-registerCourseUI(loggedInStudent : Student) : void
-dropCourseUI(loggedInStudent : Student) : void
-changeIndexNumberUI(loggedInStudent : Student) : void
-swopIndexNumberUI(loggedInStudent : Student) : void
-selectNotiModeUI(loggedInStudent : Student) : void
<<use<<
Manager
StudentMgr
~studentList : ArrayList<Student>
+getStudentByMatric(matricNumber : String) : Student
+updateAccessPeriod(matricNumber : String, newAccessStart : Calendar, newAccessEnd : Calendar) : void
+isExistingUsername(username : String) : Boolean
+isExistingMatricNumber(matricNumber : String) : Boolean
+addStudent(username : String, password : String, salt : String, firstName : String, lastName : String, matricNumber : String, gender : char, nationality : String, mobileNo : int, email : String, accessStart : Calendar, accessEnd : Calendar, i : int) : void
+removeStudent(matricNumber : String) : void
CalendarMgr
~sc : Scanner
~dateFormat : SimpleDateFormat
+dateToString(cal : Calendar) : String
+timeToString(cal : Calendar) : String
+getValidDateTime(mode : String) : Calendar
+getValidLessonTime(mode : String) : Calendar
+dayIntToString(day : int) : String
CourseMgr
~courseList : ArrayList<Course>
+getCourseByCode(courseCode : String) : Course
+getIndexByNumber(courseCode : String, indexNumber : int) : Index
StaffCourseMgr
+addCourse(courseCode : String, courseName : String, au : int, school : String, courseType : String, examDate : Calendar) : void
+removeCourse(courseCode : String) : void
+isExistingCourseCode(courseCode : String) : Boolean
+updateCourseCode(courseCode : String, newCourseCode : String) : void
+updateSchool(courseCode : String, newSchool : String) : void
+addIndex(courseCode : String, indexNumber : int, tutorialGroup : String, vacancy : int) : void
+removeIndex(courseCode : String, indexNumber : int) : void
+isExistingIndex(courseCode : String, indexNumber : int) : Boolean
+addLesson(courseCode : String, indexNumber : int, lessonType : String, lessonDay : int, lessonStartTime : Calendar, lessonEndTime : Calendar, lessonVenue : String) : void
+updateIndex(courseCode : String, indexNumber : int, newIndexNumber : int) : void
+updateVacancy(courseCode : String, indexNumber : int, newVacancy : int) : void
StudentCourseMgr
~registrationList : ArrayList<CourseReg>
+isExistingRegistration(student : Student, courseCode : String) : Boolean
+registerCourse(stu : Student, courseCode : String, indexNumber : int) : void
+swapIndex(loggedInStudent : Student, peer : Student, courseCode : String, yourIndexNumber : int, peerIndexNumber : int) : void
+removeRecord(loggedInStudent : Student, courseCode : String, indexNumber : int) : void
~updateWaitList(courseCode : String, index : Index, counter : int) : void
+removeRecord(course : Course) : void
+getCourseReg(loggedInStudent : Student, courseCode : String) : CourseReg
+checkClash(loggedInStudent : Student, courseCode : String, indexNumber : int) : Boolean
NotificationMgr
~registrationList : ArrayList<CourseReg>
~courseList : ArrayList<Course>
~studentList : ArrayList<Student>
+sendBoth(studentToNotify : Student, courseCode : String) : void
+sendEmail(studentToNotify : Student, courseCode : String) : void
+sendSMS(studentToNotify : Student) : void
+sendAlertWaitlist(studentToNotify : Student, courseCode : String, indexNumber : int) : void
LoginMgr
~hashBytes : byte[]
~studentList : ArrayList<Student>
~staffList : ArrayList<Staff>
+compareUserPass(username : String, passwordToHash : St...
+generateSalt() : String
+hashing(passwordToBeHash : String, salt : String) : String
+isValidAccessTime(loggedInUser : Student) : Boolean
Entities
Student
-serialVersionUID : long
-firstName : String
-lastName : String
-matricNumber : String
-gender : char
-nationality : String
-mobileNo : int
-email : String
-accessStart : Calendar
-accessEnd : Calendar
-notiMode : int
+Student(userName : String, password : String, salt : String, firstName : String, lastName : String, matricNumber : String, gender : char, nationality : String, mobileNo : int, email : String, accessStart : Calendar, accessEnd : Calendar, notiMode : int)
+equals(obj : Object) : boolean
+showUI() : void
+getFirstName() : String
+setFirstName(firstName : String) : void
+getLastName() : String
+setLastName(lastName : String) : void
+getMatricNumber() : String
+setMatricNumber(matricNumber : String) : void
+getGender() : char
+setGender(gender : char) : void
+getNationality() : String
+setNationality(nationality : String) : void
+getMobileNo() : int
+setMobileNo(mobileNo : int) : void
+getEmail() : String
+setEmail(email : String) : void
+getAccessStart() : Calendar
+setAccessStart(accessStart : Calendar) : void
+getAccessEnd() : Calendar
+setAccessEnd(accessEnd : Calendar) : void
+getNotiMode() : int
+setNotiMode(notiMode : int) : void
is a ..
User {abstract}
-serialVersionUID : long
-username : String
-password : String
-accessRight : String
-salt : String
+User(username : String, password : String, accessRight ...
+equals(obj : Object) : boolean
+showUI() : void
+getUsername() : String
+getPassword() : String
+getAccessRight() : String
+getSalt() : String
is a ..
Staff
-serialVersionUID : long
+Staff(userName : String, passw...
+equals(obj : Object) : boolean
+showUI() : void
CourseReg
-Status : Boolean
-course : Course
-index : Index
-student : Student
+CourseReg(student : Student, course : Course, index : Index, status : B...
+getStudent() : Student
+setStudent(student : Student) : void
+getCourse() : Course
+setCourse(course : Course) : void
+getIndex() : Index
+setIndex(index : Index) : void
+getStatus() : Boolean
+setStatus(status : Boolean) : void
0..*
Registers
0..*
Course
-serialVersionUID : long
-courseCode : String
-courseName : String
-au : int
-school : String
-courseType : String
-examDate : Calendar
-indexList : List<Index>
+Course(courseCode : String, courseName : String, au ...
+getAU() : int
+setAU(aU : int) : void
+equals(o : Object) : boolean
+getExamDate() : Calendar
+setExamDate(examDate : Calendar) : void
+getCourseCode() : String
+setCourseCode(courseCode : String) : void
+getCourseName() : String
+setCourseName(courseName : String) : void
+getSchool() : String
+setSchool(school : String) : void
+getCourseType() : String
+setCourseType(courseType : String) : void
+getIndexList() : List<Index>
+setIndexList(index : List<Index>) : void
has...
1
1..*
Index
-serialVersionUID : long
-indexNumber : int
-tutorialGroup : String
-vacancy : int
-waitingList : int
-lessonList : List<Lesson>
+Index(indexNumber : int, tutorialGroup : String, va...
+equals(o : int) : boolean
+waitingListPlus() : void
+waitingListMinus() : void
+vacancyPlus() : void
+vacancyMinus() : void
+getIndexNumber() : int
+setIndexNumber(indexNumber : int) : void
+getTutorialGroup() : String
+setTutorialGroup(tutorialGroup : String) : void
+getVacancy() : int
+setVacancy(vacancy : int) : void
+getWaitingList() : int
+setWaitingList(waitingList : int) : void
+getLessonList() : List<Lesson>
+setLessonList(lessonList : List<Lesson>) : void
has...
1
1...*
Lesson
-serialVersionUID : long
-lessonType : String
-lessonDay : int
-lessonVenue : String
-lessonStartTime : Calendar
-lessonEndTime : Calendar
+Lesson(lessonType : String, lessonDay : int, lesson...
+getLessonType() : String
+setLessonType(lessonType : String) : void
+getLessonDay() : int
+setLessonDay(lessonDay : int) : void
+getLessonVenue() : String
+setLessonVenue(lessonVenue : String) : void
+getLessonStartTime() : Calendar
+setLessonStartTime(lessonStartTime : Calendar) : v...
+getLessonEndTime() : Calendar
+setLessonEndTime(lessonEndTime : Calendar) : void

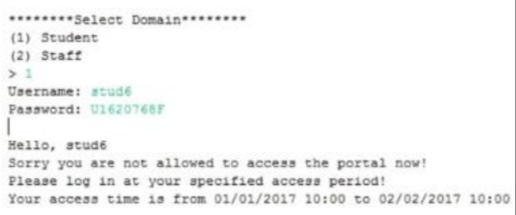
V. UML Sequence Diagram



VI. Test Cases

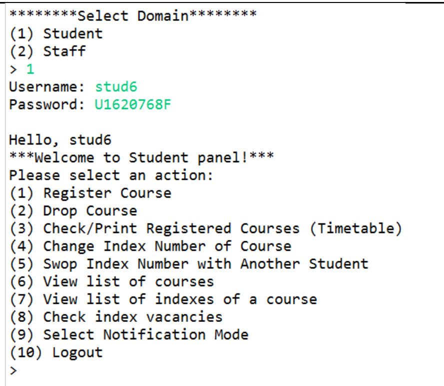
1. Student Login

a) Login before allowed period (dates)



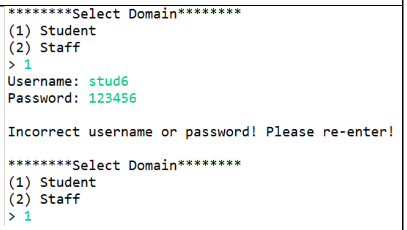
```
********Select Domain********
(1) Student
(2) Staff
> 1
Username: stud6
Password: U1620768F
|
Hello, stud6
Sorry you are not allowed to access the portal now!
Please log in at your specified access period!
Your access time is from 01/01/2017 10:00 to 02/02/2017 10:00
```

b) Login after allowed period (dates)



```
********Select Domain********
(1) Student
(2) Staff
> 1
Username: stud6
Password: U1620768F

Hello, stud6
***Welcome to Student panel!***
Please select an action:
(1) Register Course
(2) Drop Course
(3) Check/Print Registered Courses (Timetable)
(4) Change Index Number of Course
(5) Swop Index Number with Another Student
(6) View list of courses
(7) View list of indexes of a course
(8) Check index vacancies
(9) Select Notification Mode
(10) Logout
>
```

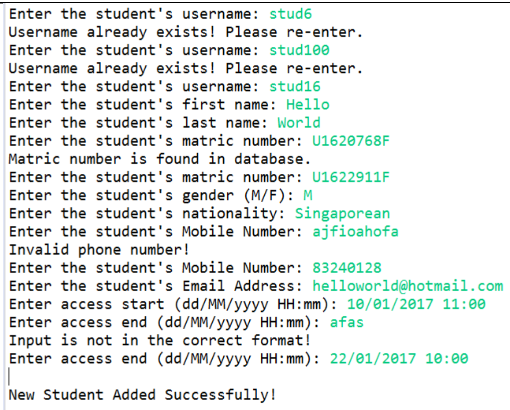
c) Wrong Password



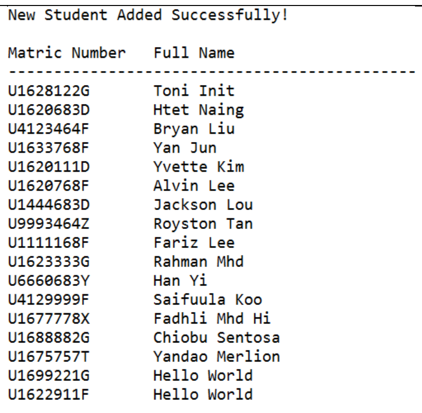
```
********Select Domain********
(1) Student
(2) Staff
> 1
Username: stud6
Password: 123456

Incorrect username or password! Please re-enter!

********Select Domain********
(1) Student
(2) Staff
> 1
```

2. Add a Student

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



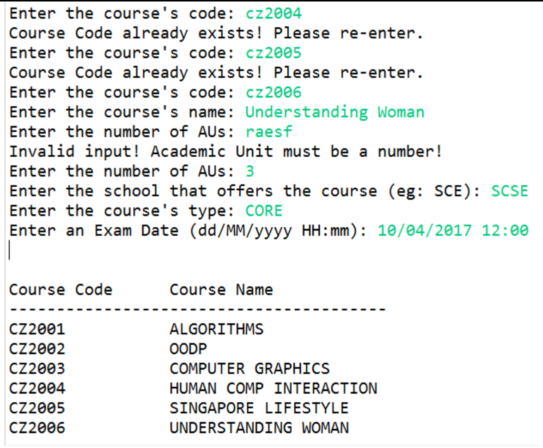
```
Enter the student's username: stud6
Username already exists! Please re-enter.
Enter the student's username: stud100
Username already exists! Please re-enter.
Enter the student's username: stud16
Enter the student's first name: Hello
Enter the student's last name: World
Enter the student's matric number: U1620768F
Matric number is found in database.
Enter the student's matric number: U1622911F
Enter the student's gender (M/F): M
Enter the student's nationality: Singaporean
Enter the student's Mobile Number: ajfioahofa
Invalid phone number!
Enter the student's Mobile Number: 83240128
Enter the student's Email Address: helloworld@hotmail.com
Enter access start (dd/MM/yyyy HH:mm): 10/01/2017 11:00
Enter access end (dd/MM/yyyy HH:mm): afas
Input is not in the correct format!
Enter access end (dd/MM/yyyy HH:mm): 22/01/2017 10:00
|
New Student Added Successfully!
```



```
New Student Added Successfully!

Matric Number   Full Name
-------------------------------------------------
U1628122G       Toni Init
U1620683D       Htet Naing
U4123464F       Bryan Liu
U1633768F       Yan Jun
U1620111D       Yvette Kim
U1620768F       Alvin Lee
U1444683D       Jackson Lou
U9993464Z       Royston Tan
U1111168F       Fariz Lee
U1623333G       Rahman Mhd
U6660683Y       Han Yi
U4129999F       Saifuula Koo
U1677778X       Fadhli Mhd Hi
U1688882G       Chiobu Sentosa
U1675757T       Yandao Merlion
U1699221G       Hello World
U1622911F       Hello World
```

3. Add a Course

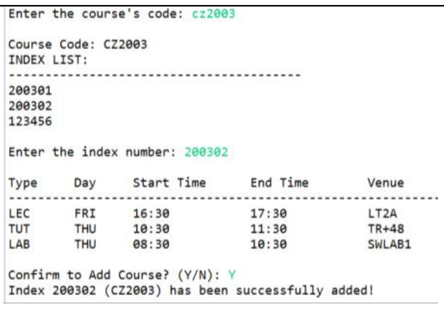
a) Add a new course
b) Add an existing course
c) Invalid data entries



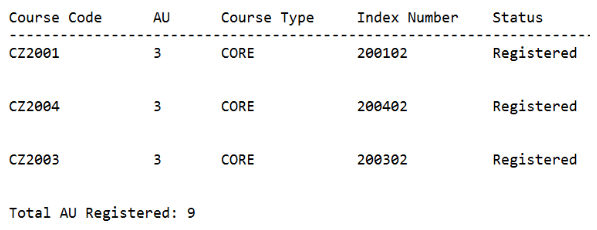
```
Enter the course's code: cz2004
Course Code already exists! Please re-enter.
Enter the course's code: cz2005
Course Code already exists! Please re-enter.
Enter the course's code: cz2006
Enter the course's name: Understanding Woman
Enter the number of AUs: raesf
Invalid input! Academic Unit must be a number!
Enter the number of AUs: 3
Enter the school that offers the course (eg: SCE): SCSE
Enter the course's type: CORE
Enter an Exam Date (dd/MM/yyyy HH:mm): 10/04/2017 12:00
|

Course Code      Course Name
----------------------------------------
CZ2001           ALGORITHMS
CZ2002           OODP
CZ2003           COMPUTER GRAPHICS
CZ2004           HUMAN COMP INTERACTION
CZ2005           SINGAPORE LIFESTYLE
CZ2006           UNDERSTANDING WOMAN
```

4. Register student for a course

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

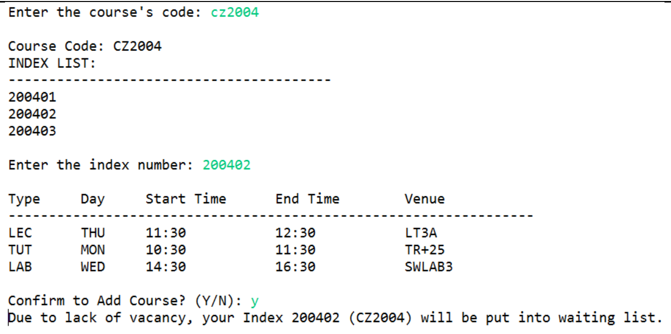


```
Enter the course's code: cz2003

Course Code: CZ2003
INDEX LIST:
----------------------------------------
200301
200302
123456

Enter the index number: 200302

Type     Day      Start Time      End Time        Venue
-----------------------------------------------------------
LEC      FRI      16:30           17:30           LT2A
TUT      THU      10:30           11:30           TR+48
LAB      THU      08:30           10:30           SWLAB1

Confirm to Add Course? (Y/N): Y
Index 200302 (CZ2003) has been successfully added!
```



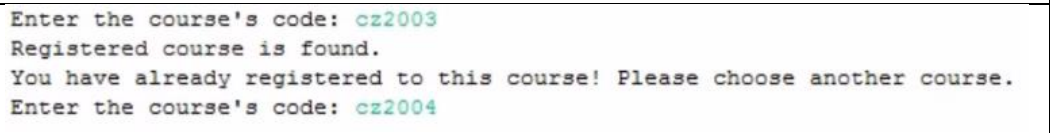
```
Course Code       AU       Course Type      Index Number      Status
------------------------------------------------------------------------
CZ2001            3        CORE             200102            Registered

CZ2004            3        CORE             200402            Registered

CZ2003            3        CORE             200302            Registered

Total AU Registered: 9
```

b) Add a student to a course index with 0 vacancies in Tut / Lab



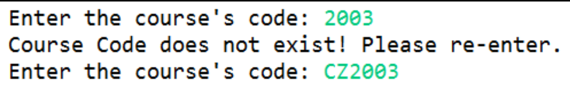
```
Enter the course's code: cz2004

Course Code: CZ2004
INDEX LIST:
----------------------------------------
200401
200402
200403

Enter the index number: 200402

Type     Day      Start Time      End Time        Venue
------------------------------------------------------------------
LEC      THU      11:30           12:30           LT3A
TUT      MON      10:30           11:30           TR+25
LAB      WED      14:30           16:30           SWLAB3

Confirm to Add Course? (Y/N): y
Due to lack of vacancy, your Index 200402 (CZ2004) will be put into waiting list.
```

c) Register the same course again



```
Enter the course's code: cz2003
Registered course is found.
You have already registered to this course! Please choose another course.
Enter the course's code: cz2004
```

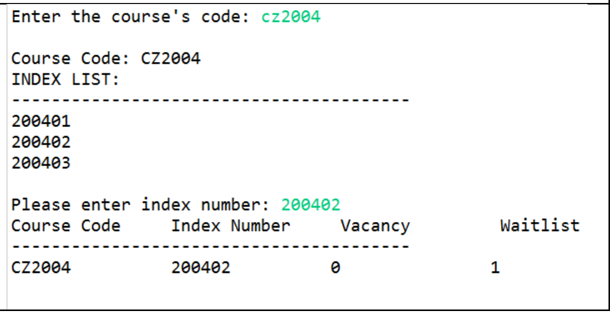
d) Invalid data entries (eg wrong student ID / course code, etc)



```
Enter the course's code: 2003
Course Code does not exist! Please re-enter.
Enter the course's code: CZ2003
```

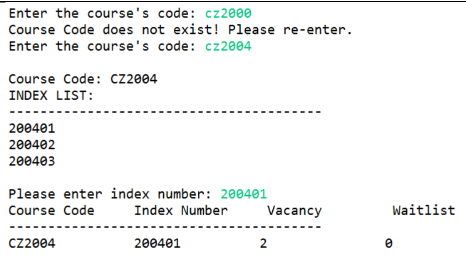
5. Check available slot in a class (vacancy in a class)

a) Check for vacancy in course index



```
Enter the course's code: cz2004

Course Code: CZ2004
INDEX LIST:
----------------------------------------
200401
200402
200403

Please enter index number: 200402
Course Code      Index Number      Vacancy           Waitlist
----------------------------------------
CZ2004           200402            0                 1
```

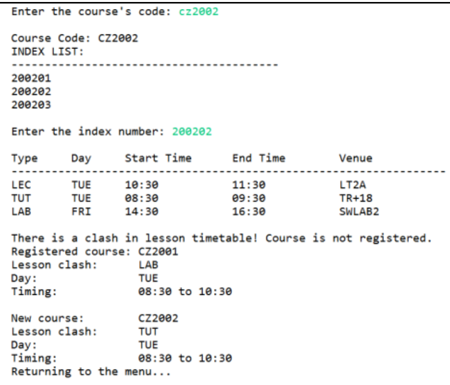
b) Invalid data entries (eg course code, class code etc)



```
Enter the course's code: cz2000
Course Code does not exist! Please re-enter.
Enter the course's code: cz2004

Course Code: CZ2004
INDEX LIST:
----------------------------------------
200401
200402
200403

Please enter index number: 200401
Course Code      Index Number      Vacancy           Waitlist
----------------------------------------
CZ2004           200401            2                 0
```

6. Day/Time clash with other course

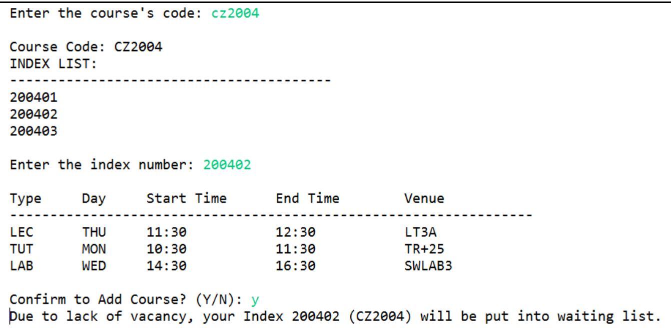
a) Add a student to a course index with available vacancies.



```
Enter the course's code: cz2002

Course Code: CZ2002
INDEX LIST:
----------------------------------------
200201
200202
200203

Enter the index number: 200202

Type     Day      Start Time       End Time         Venue
-----------------------------------------------------------------
LEC      TUE      10:30            11:30            LT2A
TUT      TUE      08:30            09:30            TR+18
LAB      FRI      14:30            16:30            SWLAB2

There is a clash in lesson timetable! Course is not registered.
Registered course: CZ2001
Lesson clash:      LAB
Day:               TUE
Timing:            08:30 to 10:30

New course:        CZ2002
Lesson clash:      TUT
Day:               TUE
Timing:            08:30 to 10:30
Returning to the menu...
```

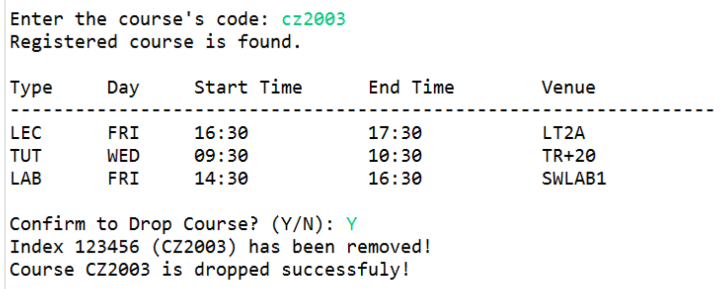
7. Waitlist notification

ai) Add studentA to a course index with 0 vacancies

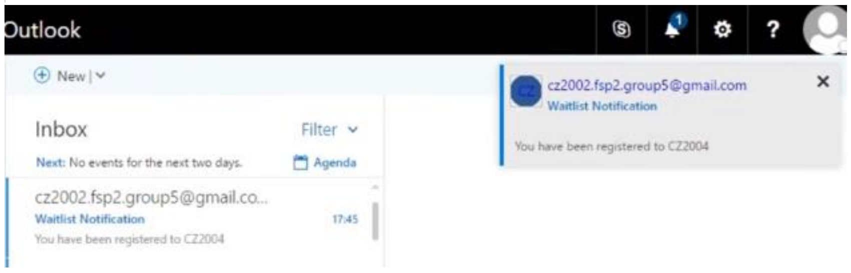


```
Enter the course's code: cz2004

Course Code: CZ2004
INDEX LIST:
----------------------------------------
200401
200402
200403

Enter the index number: 200402

Type     Day      Start Time       End Time         Venue
-----------------------------------------------------------------
LEC      THU      11:30            12:30            LT3A
TUT      MON      10:30            11:30            TR+25
LAB      WED      14:30            16:30            SWLAB3

Confirm to Add Course? (Y/N): y
Due to lack of vacancy, your Index 200402 (CZ2004) will be put into waiting list.
```

aii) Drop studentB from the same course index



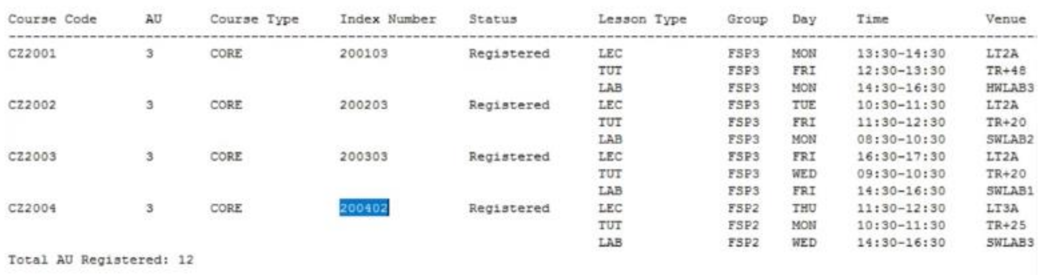
```
Enter the course's code: cz2003
Registered course is found.

Type     Day      Start Time       End Time         Venue
-----------------------------------------------------------------
LEC      FRI      16:30            17:30            LT2A
TUT      WED      09:30            10:30            TR+20
LAB      FRI      14:30            16:30            SWLAB1

Confirm to Drop Course? (Y/N): Y
Index 123456 (CZ2003) has been removed!
Course CZ2003 is dropped successfuly!
```



cz2002.fsp2.group5@gmail.com
Waitlist Notification
You have been registered to CZ2004

cz2002.fsp2.group5@gmail.co...
Waitlist Notification 17:45
You have been registered to CZ2004

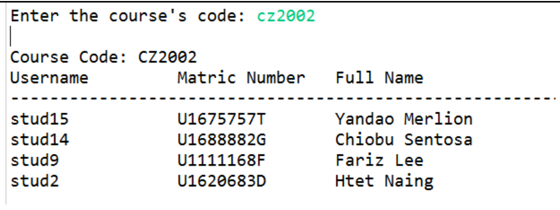
aiii) Display studentA timetable



```
Course Code   AU   Course Type   Index Number   Status       Lesson Type   Group   Day   Time          Venue
---------------------------------------------------------------------------------------------------------------
CZ2001        3    CORE          200103         Registered   LEC           FSP3    MON   13:30-14:30   LT2A
                                                             TUT           FSP3    FRI   12:30-13:30   TR+48
                                                             LAB           FSP3    MON   14:30-16:30   HWLAB3
CZ2002        3    CORE          200203         Registered   LEC           FSP3    TUE   10:30-11:30   LT2A
                                                             TUT           FSP3    FRI   11:30-12:30   TR+20
                                                             LAB           FSP3    MON   08:30-10:30   SWLAB2
CZ2003        3    CORE          200303         Registered   LEC           FSP3    FRI   16:30-17:30   LT2A
                                                             TUT           FSP3    WED   09:30-10:30   TR+20
                                                             LAB           FSP3    FRI   14:30-16:30   SWLAB1
CZ2004        3    CORE          200402         Registered   LEC           FSP2    THU   11:30-12:30   LT3A
                                                             TUT           FSP2    MON   10:30-11:30   TR+25
                                                             LAB           FSP2    WED   14:30-16:30   SWLAB3
Total AU Registered: 12
```

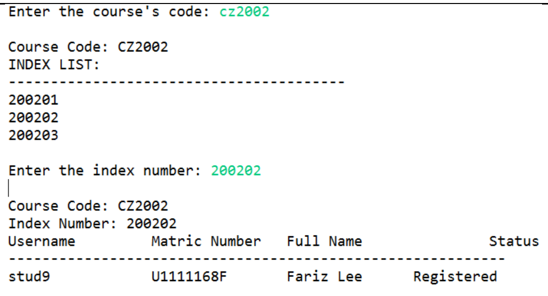
8. Print student list by index number, course

ai) Print list by Course



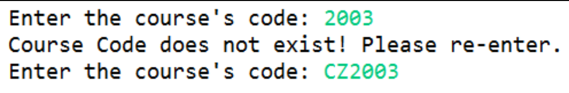
```
Enter the course's code: cz2002

Course Code: CZ2002
Username         Matric Number   Full Name
------------------------------------------------------
stud15           U1675757T       Yandao Merlion
stud14           U1688882G       Chiobu Sentosa
stud9            U1111168F       Fariz Lee
stud2            U1620683D       Htet Naing
```

aii) Print list by Index



```
Enter the course's code: cz2002

Course Code: CZ2002
INDEX LIST:
----------------------------------------
200201
200202
200203

Enter the index number: 200202

Course Code: CZ2002
Index Number: 200202
Username         Matric Number   Full Name               Status
------------------------------------------------------------
stud9            U1111168F       Fariz Lee      Registered
```

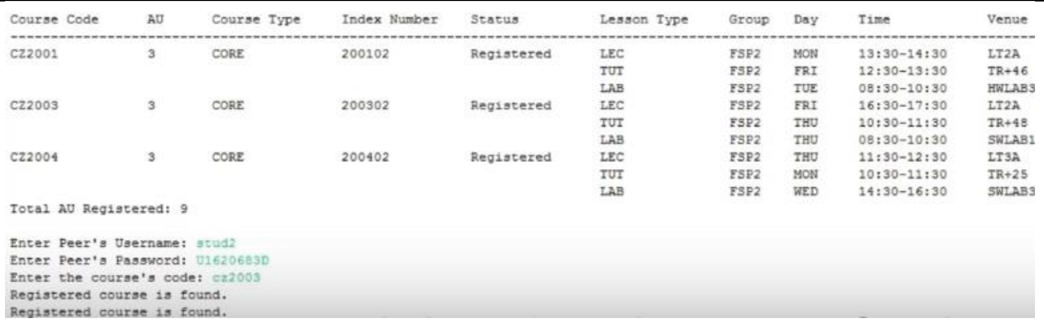
b) Invalid data entries (eg course code, index code etc)



```
Enter the course's code: 2003
Course Code does not exist! Please re-enter.
Enter the course's code: CZ2003
```

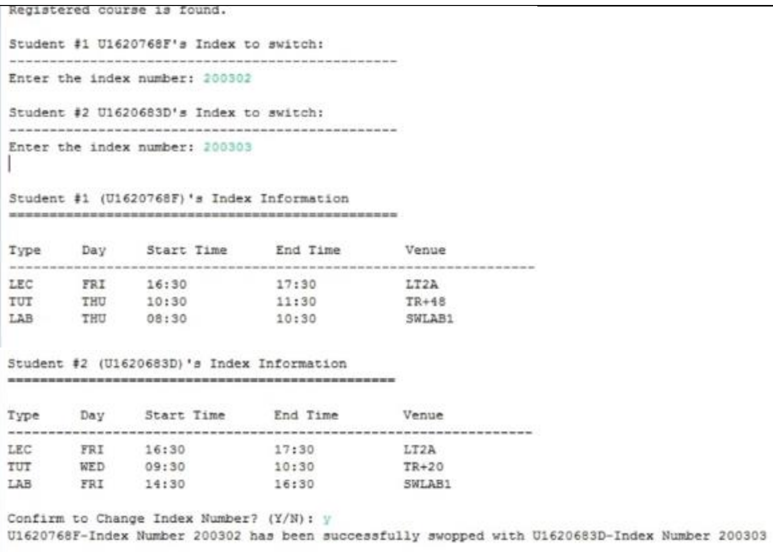
9. Swap Index Number With Another Student

a) To swap index with peer, student required to enter peer's login details



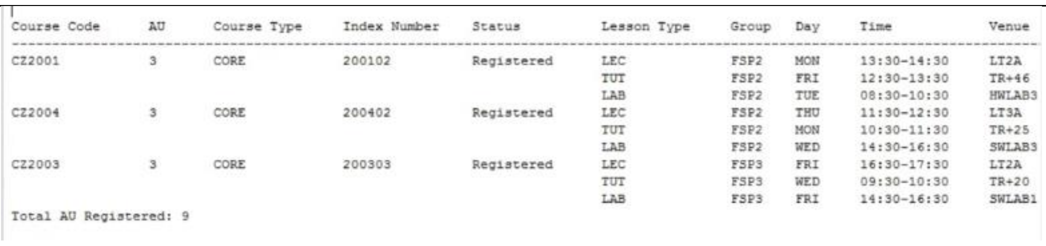
```
Course Code    AU    Course Type    Index Number   Status        Lesson Type   Group  Day   Time          Venue
-------------------------------------------------------------------------------------------------------------------
CZ2001         3     CORE           200102         Registered    LEC           FSP2   MON   13:30-14:30   LT2A
                                                                 TUT           FSP2   FRI   12:30-13:30   TR+46
                                                                 LAB           FSP2   TUE   08:30-10:30   HWLAB3
CZ2003         3     CORE           200302         Registered    LEC           FSP2   FRI   16:30-17:30   LT2A
                                                                 TUT           FSP2   THU   10:30-11:30   TR+48
                                                                 LAB           FSP2   THU   08:30-10:30   SWLAB1
CZ2004         3     CORE           200402         Registered    LEC           FSP2   THU   11:30-12:30   LT3A
                                                                 TUT           FSP2   MON   10:30-11:30   TR+25
                                                                 LAB           FSP2   WED   14:30-16:30   SWLAB3
Total AU Registered: 9

Enter Peer's Username: stud2
Enter Peer's Password: U1620683D
Enter the course's code: cz2003
Registered course is found.
Registered course is found.
```

b) Student need to enter the indexes to swap



```
Registered course is found.

Student #1 U1620768F's Index to switch:
--------------------------------------------------
Enter the index number: 200302

Student #2 U1620683D's Index to switch:
--------------------------------------------------
Enter the index number: 200303


Student #1 (U1620768F)'s Index Information
==================================================

Type     Day     Start Time      End Time        Venue
---------------------------------------------------------------------
LEC      FRI     16:30           17:30           LT2A
TUT      THU     10:30           11:30           TR+48
LAB      THU     08:30           10:30           SWLAB1

Student #2 (U1620683D)'s Index Information
==================================================

Type     Day     Start Time      End Time        Venue
---------------------------------------------------------------------
LEC      FRI     16:30           17:30           LT2A
TUT      WED     09:30           10:30           TR+20
LAB      FRI     14:30           16:30           SWLAB1

Confirm to Change Index Number? (Y/N): y
U1620768F-Index Number 200302 has been successfully swopped with U1620683D-Index Number 200303
```

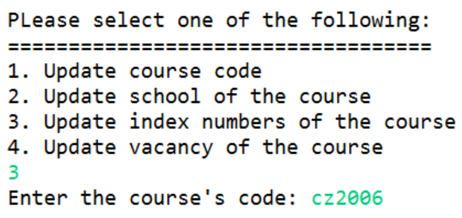
c) New index is reflected in student's timetable



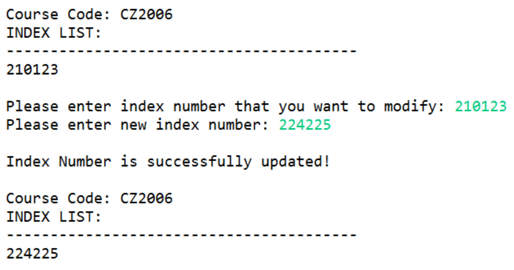
```
Course Code    AU    Course Type    Index Number    Status        Lesson Type    Group    Day    Time           Venue
-----------------------------------------------------------------------------------------------------------------------
CZ2001         3     CORE           200102          Registered    LEC            FSP2     MON    13:30-14:30    LT2A
                                                                  TUT            FSP2     FRI    12:30-13:30    TR+46
                                                                  LAB            FSP2     TUE    08:30-10:30    HWLAB3
CZ2004         3     CORE           200402          Registered    LEC            FSP2     THU    11:30-12:30    LT3A
                                                                  TUT            FSP2     MON    10:30-11:30    TR+25
                                                                  LAB            FSP2     WED    14:30-16:30    SWLAB3
CZ2003         3     CORE           200303          Registered    LEC            FSP3     FRI    16:30-17:30    LT2A
                                                                  TUT            FSP3     WED    09:30-10:30    TR+20
                                                                  LAB            FSP3     FRI    14:30-16:30    SWLAB1
Total AU Registered: 9
```

10. Change Index Number Of A Course

a) Staff select the index number to be changed and enter the new index number

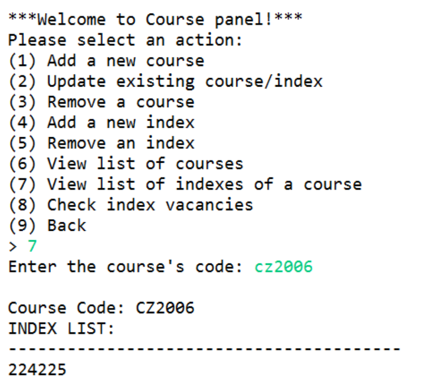


```
PLease select one of the following:
===================================
1. Update course code
2. Update school of the course
3. Update index numbers of the course
4. Update vacancy of the course
3
Enter the course's code: cz2006
```



```
Course Code: CZ2006
INDEX LIST:
----------------------------------------
210123

Please enter index number that you want to modify: 210123
Please enter new index number: 224225

Index Number is successfully updated!

Course Code: CZ2006
INDEX LIST:
----------------------------------------
224225
```

b) New index is reflected in the Course



```
***Welcome to Course panel!***
Please select an action:
(1) Add a new course
(2) Update existing course/index
(3) Remove a course
(4) Add a new index
(5) Remove an index
(6) View list of courses
(7) View list of indexes of a course
(8) Check index vacancies
(9) Back
> 7
Enter the course's code: cz2006

Course Code: CZ2006
INDEX LIST:
----------------------------------------
224225
```