NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

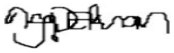
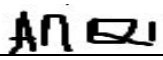
CZ2002 Assignment:
My STudent Automated Registration System
(My STARS)

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	Lab Group	Signature / Date
Chua Ming Hui	CZ2002	DSAI2	
Liu Wing Lam	CZ2002	DSAI2	
Ng Pek Han	CZ2002	DSAI2	
Ng Xiang Qing, Erica	CZ2002	DSAI2	
Yee An Qi	CZ2002	DSAI2	

Design Considerations

Login Page

`LoginPage` class uses several dependency relations such as `PasswordReader` and `LoginInfoManager` as seen in the class diagram below. This also promotes loose coupling.

Editing Student Access Period

This function is designed for the admin to edit the access period of current students and new students. There is a *`boolean`* `edit` parameter passed into `editStudentAccessPeriod()` method in the `StudentAccessPeriodEditor` class. This *`boolean`* parameter customises this edit operation -- (1) for a current student, program edits access period via prompting for student's username and; (2) for a new student, program edits access period via using `username` parameter passed into the `editStudentAccessPeriod()` method and adds student into the system. This design displays reusability as *`boolean`* `edit` can be incorporated with minimal disruption to the operation body code to reuse the main purpose of the edit operation.

```
public void editStudentAccessPeriod(boolean edit, String _username){
    Scanner sc = new Scanner (System.in);
    LoginInfoFileEditor accessPeriod = new LoginInfoFileEditor();
    String username = null;
    int accessPeriodLineNo;

    if (edit == true && _username == null) {
        System.out.println("You are editing the Student Access Period.");
    }

    do {
        if (edit == true && _username == null) {
            System.out.print("Enter the username of the student: ");
            username = sc.next();
        }

        else if (edit == false && _username != null){
            username = _username;
        }

        accessPeriodLineNo = accessPeriod.lineNo(username);
    } while (accessPeriodLineNo == -1);
```

Adding New Student

Applying Single Responsibility, we tried to create classes that are catered to one responsibility / functionality such as the `PasswordModifier` class which is used to change the student's password. We adapted encapsulation to store the details of the new student in the `NewStudent` class. To ensure information hiding and prevent the administrator from knowing the password of the new student, we randomly generate a password of length 8 consisting of all digits, lowercases and uppercase for the new student, and email them their personalised login information. Beyond that, we have an added function for students to change their password.

File Format

We applied abstraction to categorise the information used in storing data in text files while catering to limitations (e.g.: password can contain any character), we could not use a delimiter to store usernames and passwords. Thus, arranging the login information in separate lines as shown below is a better alternative.

```
m200033
fOWWQ
10/12/2020|09:00|12/12/2020|13:00
```

Notification Mode

We took into account various ways to notify the student such as email, whatsapp etc. using polymorphism and abstraction. With reference to the class diagram, we have an *abstract* `NotificationMode` class which includes an *abstract* `sendNotification` method. This class generalises `EmailNotification`. As shown in the screenshot below, we can create `WhatsappNotification` and `SMSNotification` classes by inheriting `NotificationMode` and implementing `sendNotification` in both subclasses.

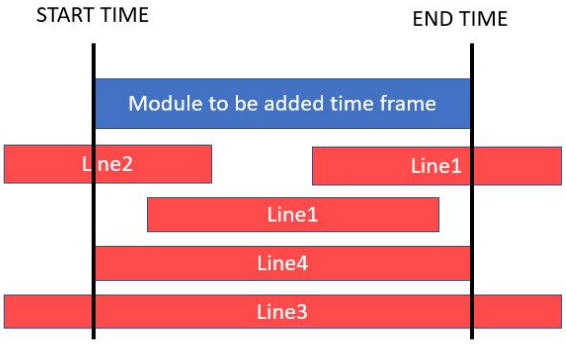
```
public class StudentNotification {
    public StudentNotification() {}

    public void notifyStudent(NotificationMode notiMode) {
        notiMode.sendNotification();
    }
}
```

Error Checking Class

Noticing that several inputs of similar types (dates, times) are required from the user to store into the text files, we used abstraction to simplify our codes by constructing different methods in the error checking class for different types of inputs.

Checking for Clash in Timetable




```
public boolean checkTimeClash(int i, int j, int[][] _time) {
    if (time[i][0] < _time[j][0] && _time[j][0] < time[i][1]) // cl
        return true;
    else if (time[i][0] < _time[j][1] && _time[j][1] < time[i][1])
        return true;
    else if (time[i][0] > _time[j][0] && time[i][1] < _time[j][1])
        return true;
    else if (time[i][0]==_time[j][0] && time[i][1]==_time[j][1]) /
        return true;
    else
        return false;
```

Line 1
Line 2
Line 3
Line 4

: Modules currently registered for student

Comparison performed after checking that the day coincides

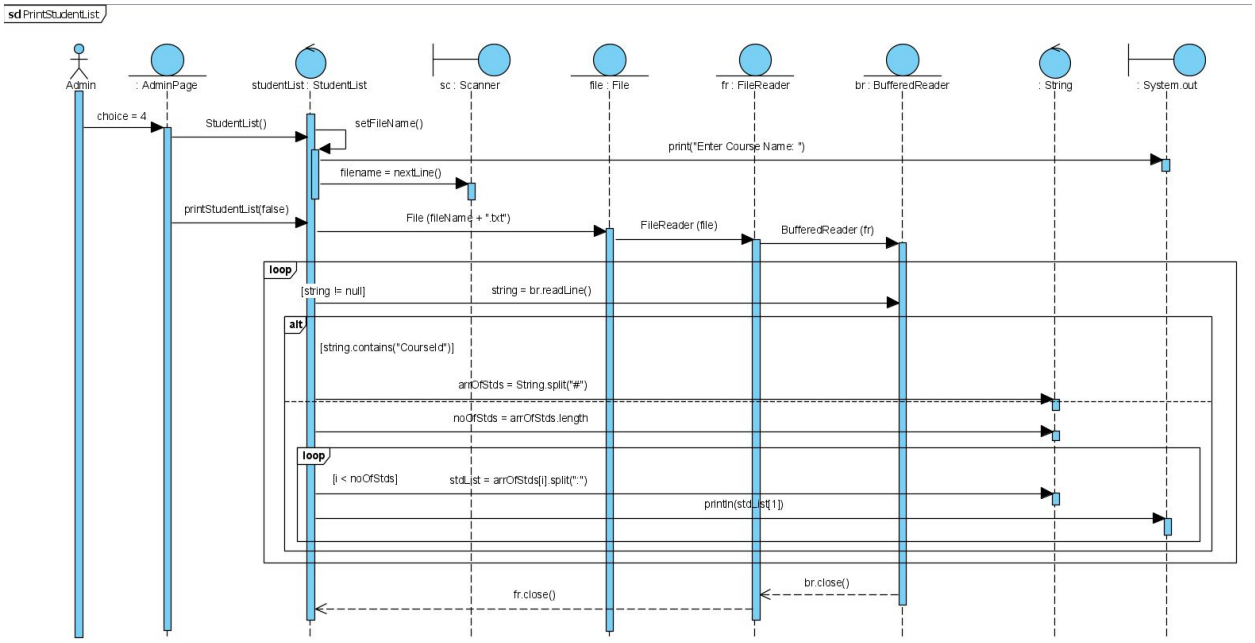
Student: Adding and Dropping Courses

As seen from the diagrams above, both classes are the great-grandchildren of the `Course` class. To fulfill Single Responsibility Principle (SRP), inheritance is used to pass general attributes and methods from parent to child while more specific attributes and methods are created in the appropriate children classes. This design is extendable in a way that the `StudentNewCourse` and `StudentDropCourse` classes are used to derive the functionality of swapping indexes. The `StudentNewCourse` and `StudentDropCourse` classes can also be reused in different ways. For example, if the school decided to implement a bidding system, the classes can be reused with additional conditions added.

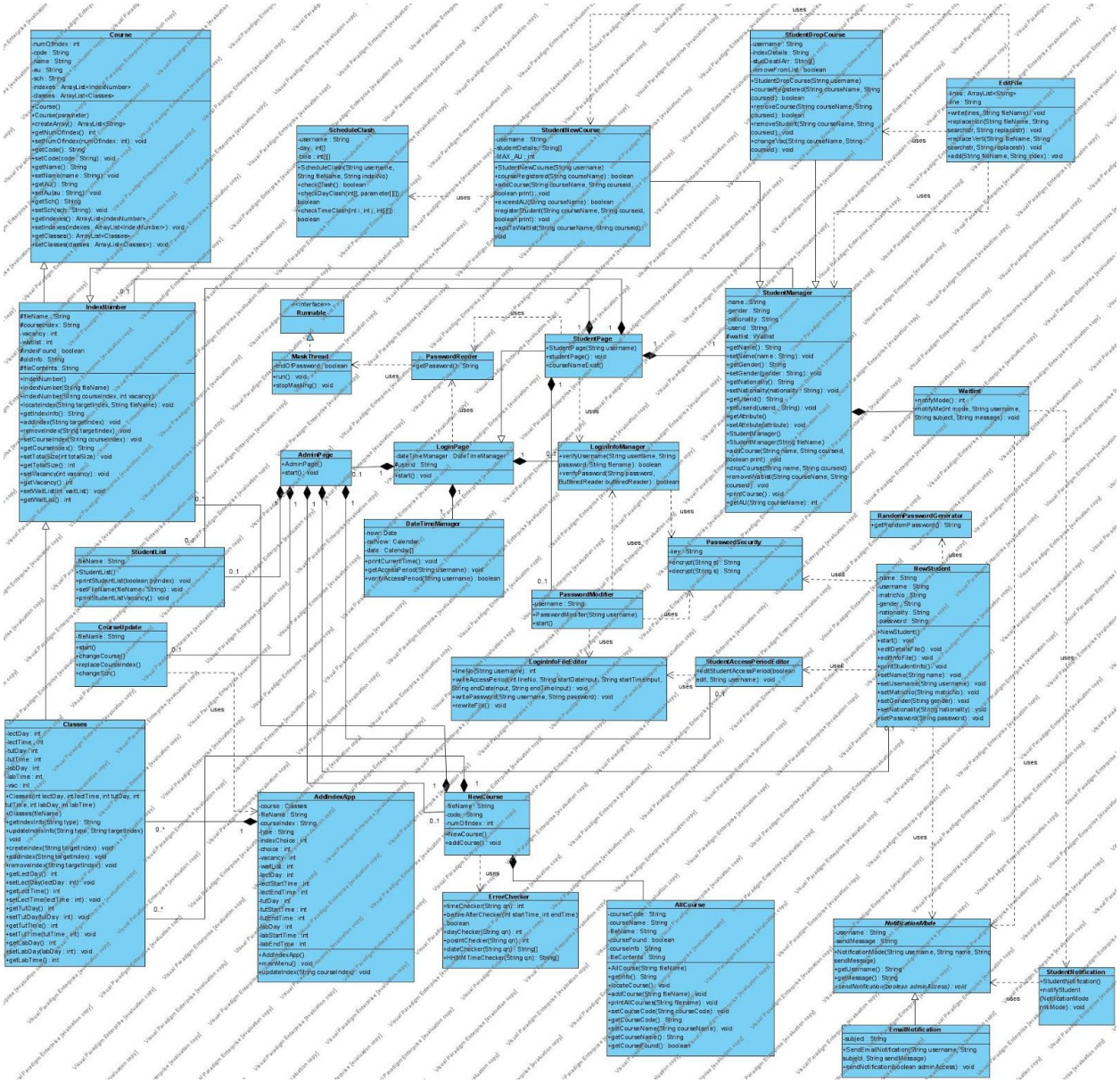
Handling Courses

`ArrayList` is used to store all the information of each course, where information of each index number and information of each type of classes (e.g.: lecture, tutorial, lab) are stored in an `ArrayList`. The aim of using `ArrayList` is to store information dynamically as it is resizeable. `ArrayList` also allows for random access of the information, allowing easier retrieval of information.

Detailed UML Sequence Diagram (“Print Student List By Course”)

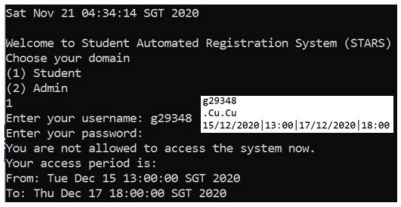
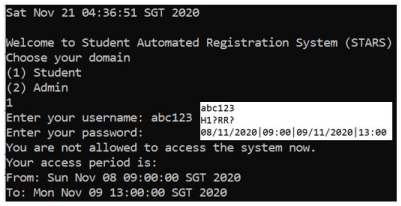
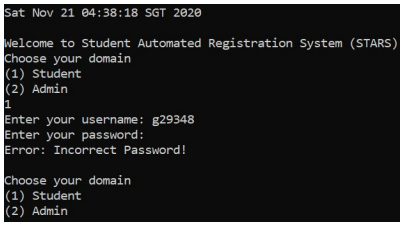


Class Diagram

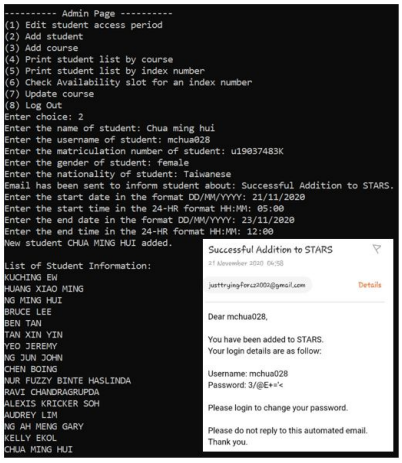


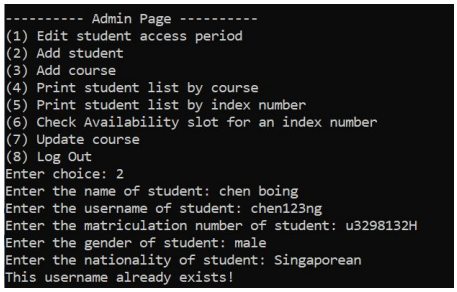
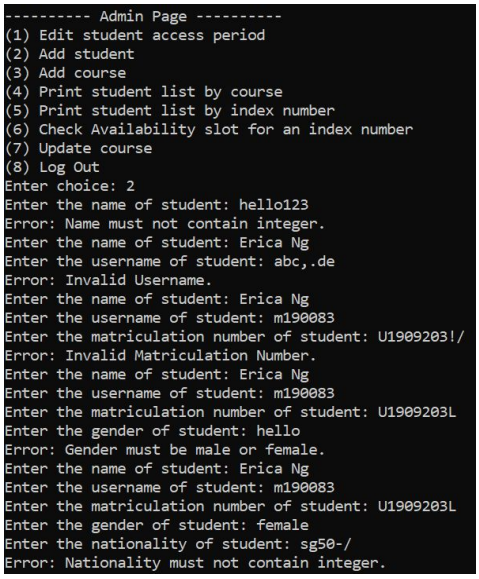
Test Cases for MySTARS Application

1. Student Login

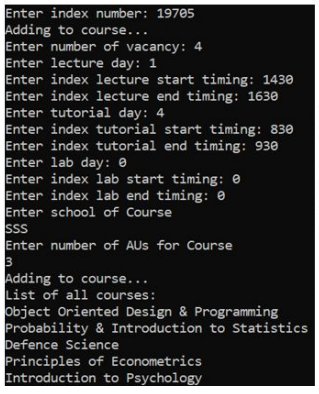
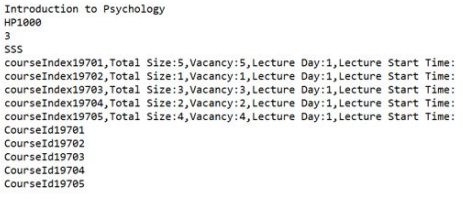
	Test Case	Expected Outcome	Actual Outcome
1	Login before allowed period (dates)	Appropriate error message display	
2	Login after allowed period (dates)	Appropriate error message display	
3	Wrong password	Appropriate error message display. Prompted to re-enter login credentials.	

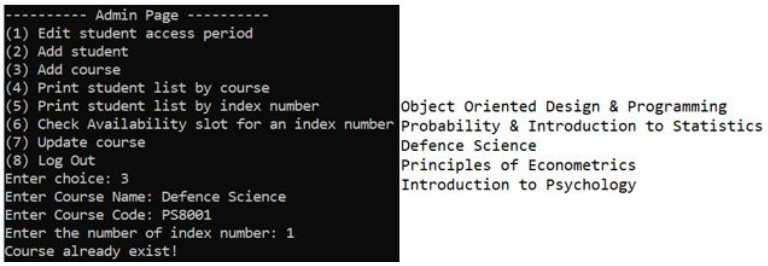
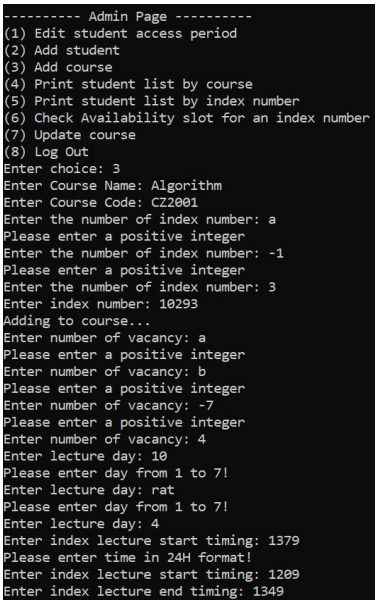
2. Add a student

	Test Case	Expected Outcome	Actual Outcome
1	Add a new student	A list of all students will be shown upon successful addition of student. An email will be sent to the student to inform them of their username and random password generated.	

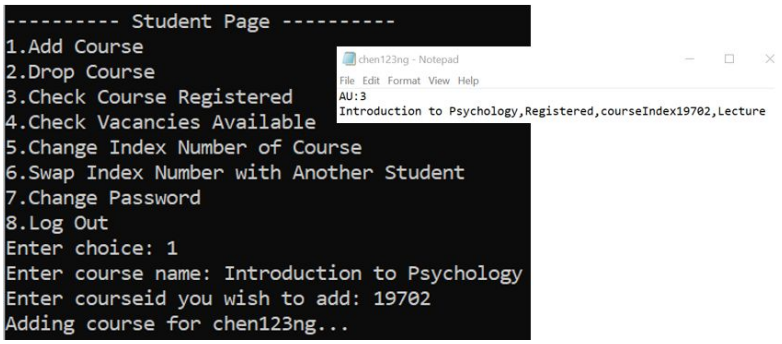
2	Add an existing student	Appropriate error message display	
3	Invalid data entries	Appropriate error message display	

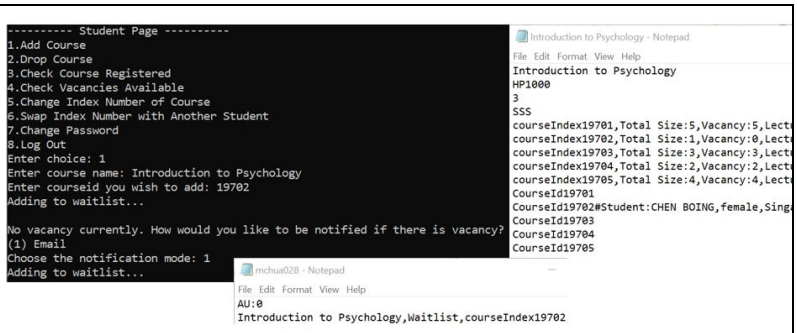
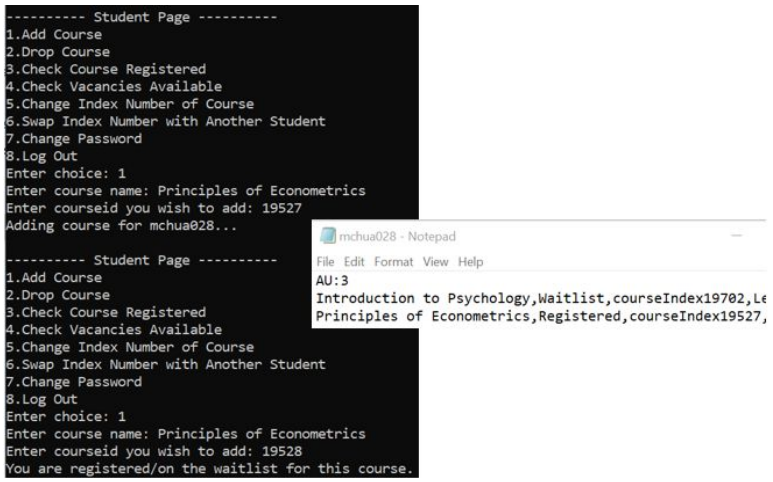
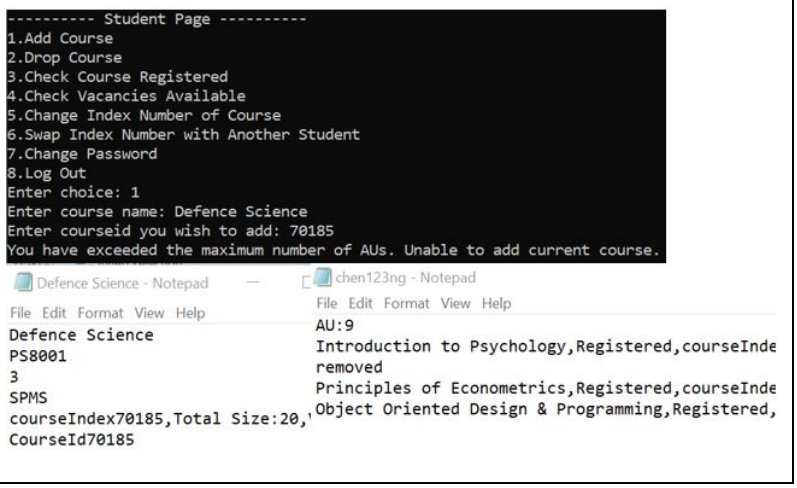
3. Add a course

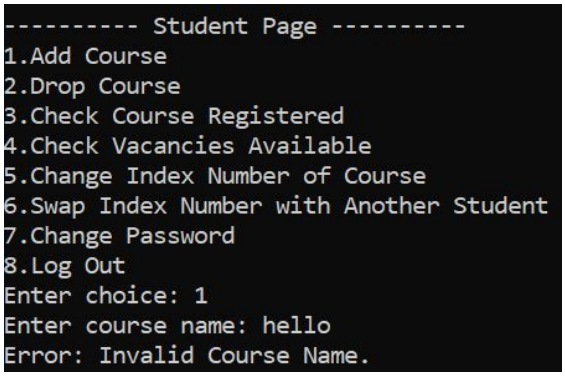
	Test Case	Expected Outcome	Actual Outcome
1	Add a new course	A list of all courses will be shown upon successful addition of the course	 

2	Add an existing course	Appropriate error message display	
3	Invalid data entries	Appropriate error message display	

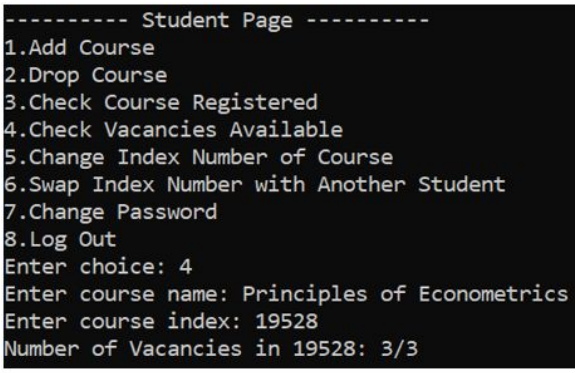
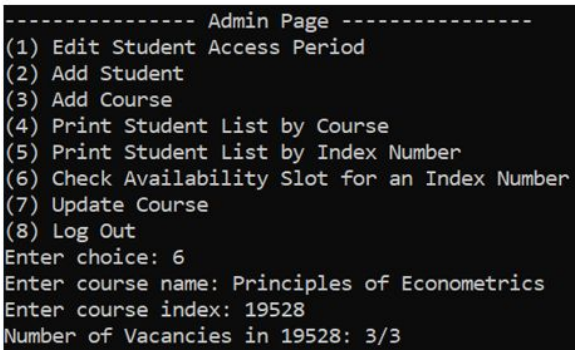
4. Register student for a course

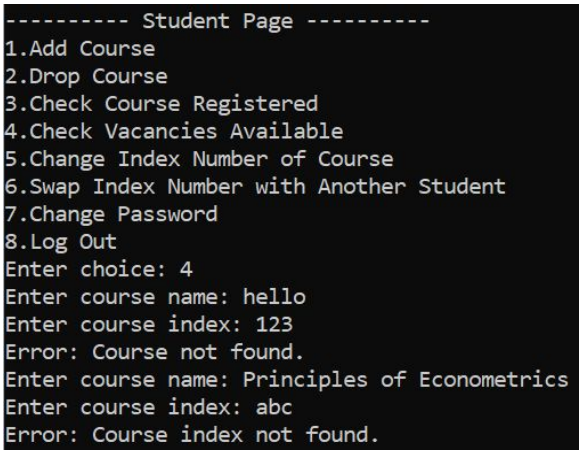
	Test Case	Expected Outcome	Actual Outcome
1	Add a student to a course index with available vacancies	Successfully added, AU increases.	

2	Add a student to a course index with 0 vacancies in tutorial or lab.	Appropriate error message display. Inform student they are placed on waitlist.	
3	Register same course again	Appropriate error message display	
4	Register for new course which exceeds the maximum number of AU	Appropriate message display, unable to add course	

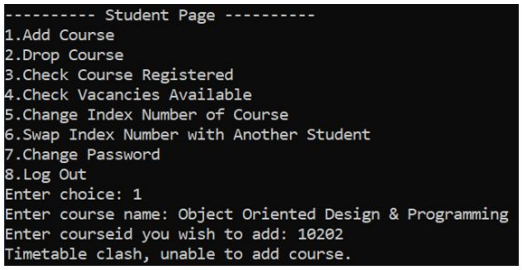
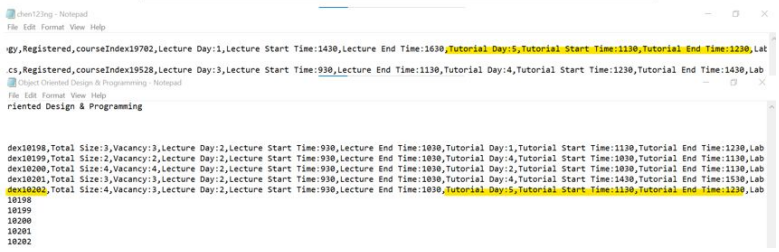
4	Invalid data entries	Appropriate error message display	

5. Check vacancy in a class

	Test Case	**Expected Outcome**	**Actual Outcome**
1	Check for vacancy in course index	Appropriate error message display	 

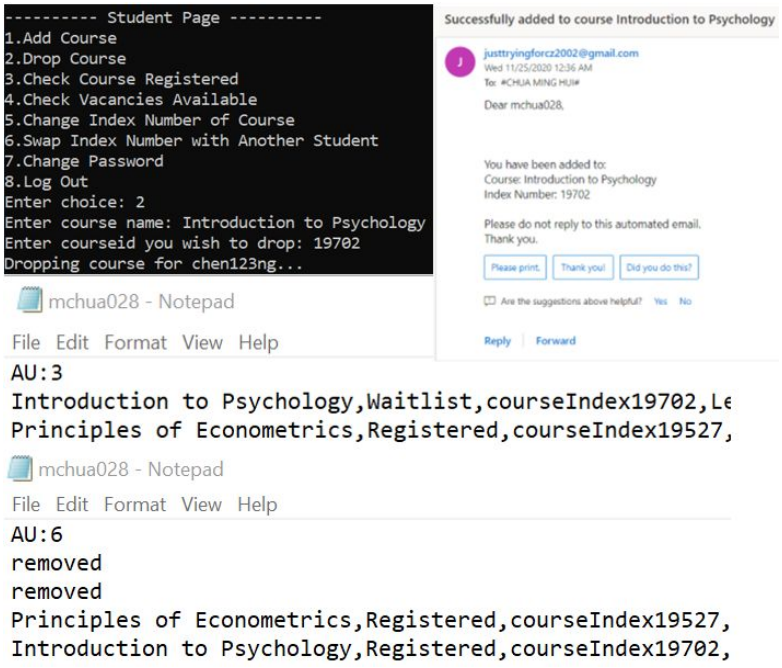
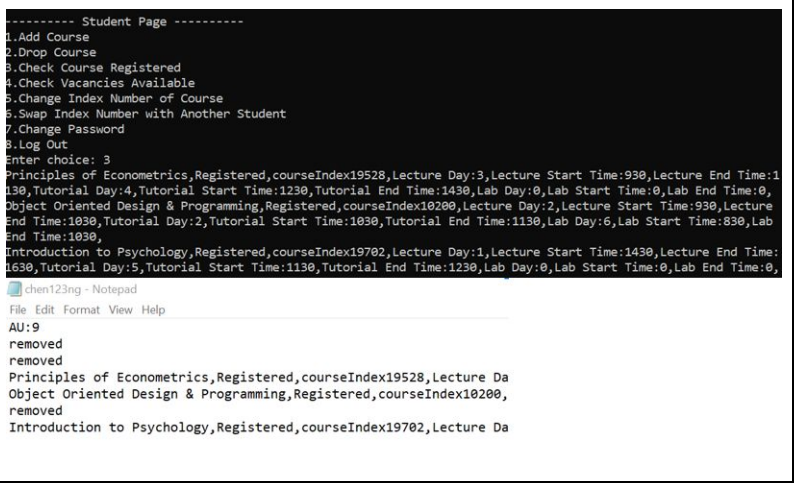
2	Invalid data entries	Appropriate error message display	

6. Day/Time clash with other course

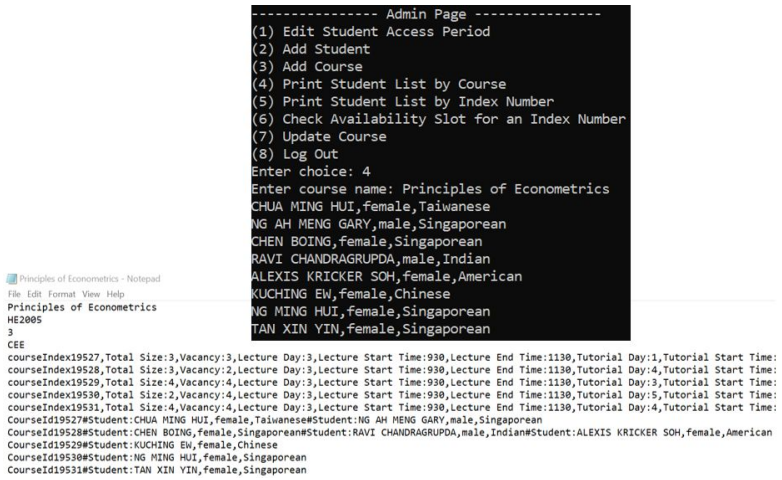
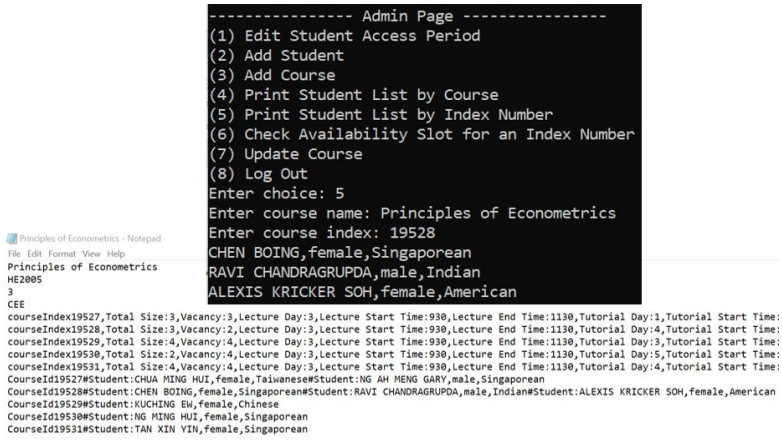
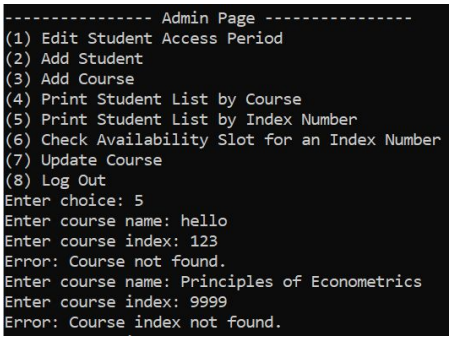
	Test Case	Expected Outcome	Actual Outcome
1	Add a student to a course index with available vacancies	Appropriate message display, eg day/time clash	 

7. Waitlist notification

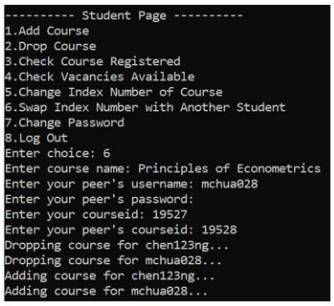
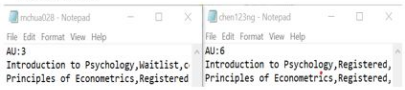
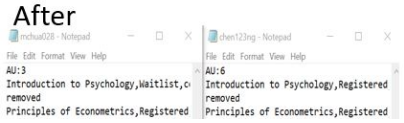
	Test Case	Expected Outcome	Actual Outcome
1	Add a studentA to a course index with 0 vacancy	Appropriate error message display. Inform studentA that he/she is placed on	Refer to Test Case 4.2.

		waitlist.	
2	Drop a studentB from the same course index	StudentB successfully dropped and studentA successfully added. Notification will be sent to StudentA to inform them that they are added.	
3	Display StudentA timetable	Show their timetable and also the new course added	

8. Print student list by index number, course

	Test Case	Expected Outcome	Actual Outcome
1	Print list by course	Student list by course is displayed	
2	Print list by Index Number	Student list by index number is displayed	
3	Invalid data entries	Appropriate error message display	

9. Swap course index with another student

	Test Case	Expected Outcome	Actual Outcome
1	Swap index with another student from another index	Appropriate message display, swapped.	 Before  After 

Youtube link to our video:

https://youtu.be/BJYpQbWpIbk