






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

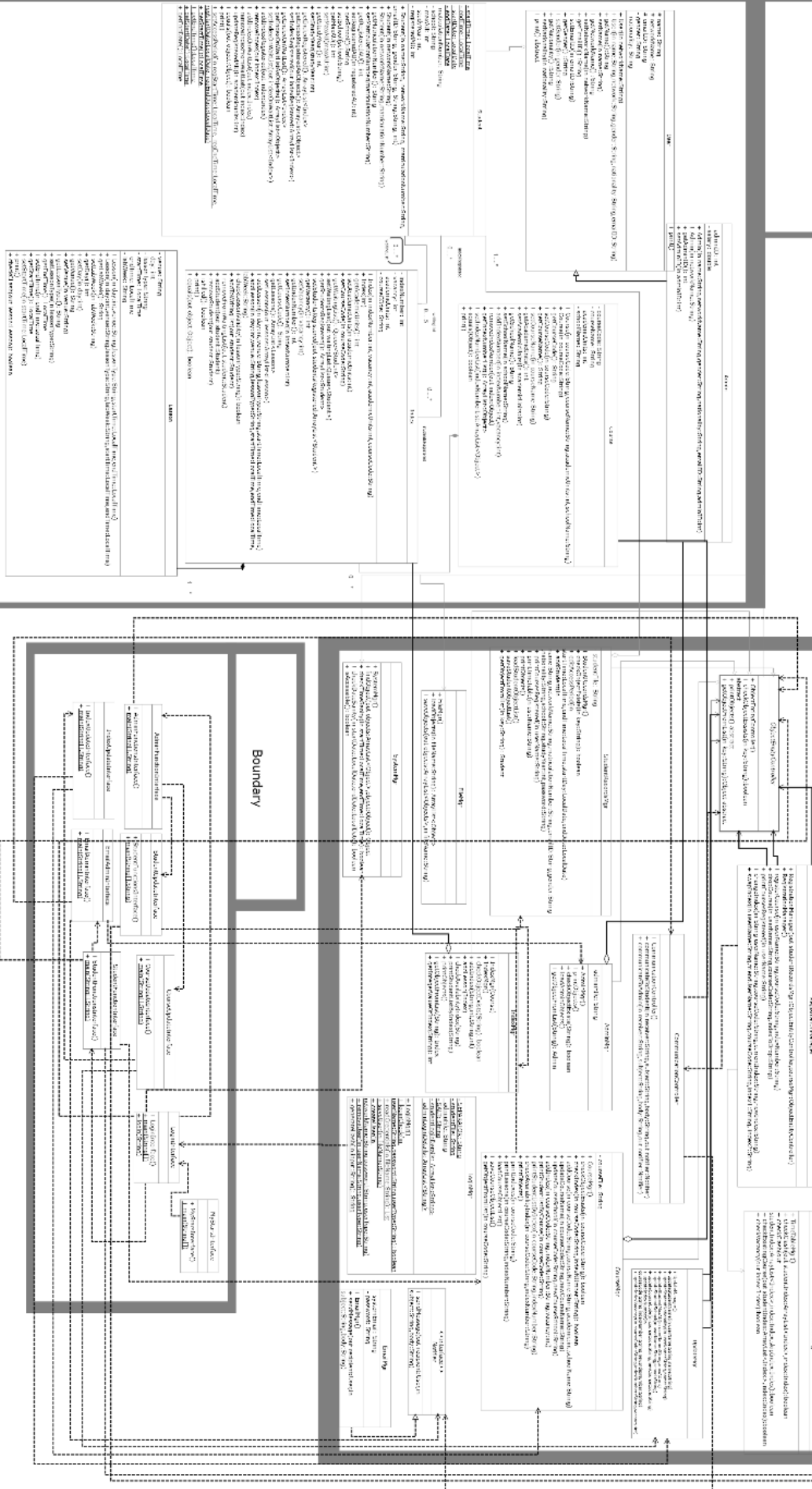
We understand that if plagiarism is found in the assignment, then lower marks or no marks will be awarded for the assessed work. In addition, disciplinary actions may be taken.

Name	Course	Lab Group	Signature/Date
Acharya Atul	CZ2002	SS10	 25/11/2020
Anil Ankitha	CZ2002	SS10	 25/11/2020
Arora Srishti	CZ2002	SS10	 25/11/2020
Parthan Muralidharan	CZ2002	SS10	 25/11/2020
Ramasubramanian Nisha	CZ2002	SS10	 25/11/2020

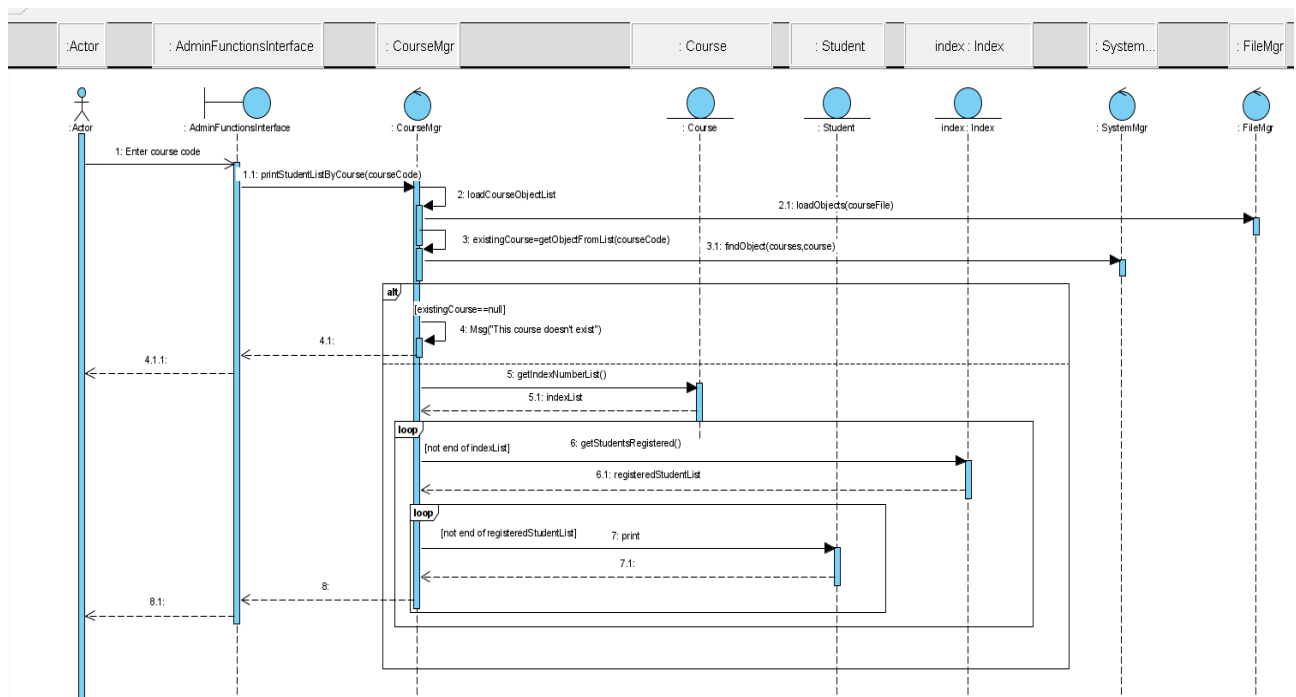
# UML Class Diagram

Entity

Controller



## UML Sequence Diagram



The actor enters the course code in the AdminFunctionsInterface which calls the printStudentListByCourse method in CourseMgr. To get the list of course objects, the recursive message loadCourseObjectList is invoked which interacts with the FileMgr class for the same. The recursive method getObjectFromList is then invoked to get the required course object. Interaction with the SystemMgr class is required for this.

If the existingCourse object is null, a course with the course code entered by the actor does not exist and it thus returns from the method printListByCourse. If the existingCourse object is not null, then the list of indices of the course is obtained using the getIndexNumberList method of the Course object.

Iterating over each index in the list, in a for loop, the list of registered students of each index is obtained by invoking the getStudentsRegistered method on each index object.

For each student in the list of registered students of each index, the details of the student are printed out. After all the details of all the students have been printed out, the printStudentListByCourse method returns to the AdminFunctionsInterface.

## **Design:**

### **a. Considerations:**

The important classes required for MyStars Planner were identified and classified as Boundary, Controller and Entity classes, respectively. It was ensured that the boundary and entity classes interact with each other only via the controller classes.

As evident from the class diagram, the design has loose coupling. Each of the classes have high cohesion. Each class has methods relevant to its purpose and is not very versatile. None of them have functionalities unrelated to its purpose.

The design has been made keeping reusability, maintainability, and extensibility in mind. The project can easily be extended to incorporate other forms of notification such as SMS without causing a ripple effect. This will be discussed in more detail in the later sections. These classes can easily be used in other projects as well.

Data abstraction and encapsulation have also been appropriately implemented

### **b. Use of OOP Design Principles:**

- **Single Responsibility Principle (SRP)**

We have ensured that each class has only a single responsibility. For example, the classes TimeTableMgr, FileMgr, LoginMgr and EmailMgr only have methods related to the timetable, file handling, login functionality and email functionality respectively. StudentRecordsMgr has methods related to displaying and adding student records i.e methods that require only student objects. RegistrationMgr has the methods related to registering for a course, dropping a course, changing as well as swapping an index

Boundary classes have been defined such that each boundary class is responsible for one major function only. For example, we have separate boundary classes i.e; user interfaces for logging in, displaying the functions for a student user and that of an admin user (LoginInterface StudentFunctionsInterface and AdminFunctionsInterface). Functions that update student and course details require more information from the user. Thus, boundary classes that get the user input for these functions have been defined in StudentUpdateInterface and CourseUpdateInterface. The boundary classes are instantiated appropriately. Thus, there is never more than one reason for a class to change.

- **Open-Closed Principle (OCP)**

In order to send a notification to a user, an object implementing the notifier interface is passed to either the `communicateToStudent` or `communicateToAdmin` function. Thus `CommunicationController` (the class that contains these functions) is closed for modification but open to addition via the notification interface. Other forms of notification (like SMS) can be added by defining classes that implement the notifier interface. The objects of these newly defined classes can be passed to the above-mentioned functions without making any changes to them.

```
/**
 * public void communicateToStudent(String receiver, String subject, String body, Notifier notifier) {
 *     objectEntityController = new StudentRecordsMgr();
 *     User student = (User) objectEntityController.getObjectFromList(receiver);
 *     if (student == null) {
 *         System.out.println(RED + "No such student exists" + RESET);
 *         return;
 *     }
 *     notifier.sendMessage(student, subject, body);
 * }
 *
 * /**
 *  * Sends a notification to an admin
 *  *
 *  * @param receiver admin who will receive the notification
 *  * @param subject subject of the notification
 *  * @param body body of the notification
 *  * @param notifier object that implements the notifier interface
 *  */
 * public void communicateToAdmin(String receiver, String subject, String body, Notifier notifier) {
 *     ObjectEntityController objectEntityController = new AdminMgr();
 *     User admin = (User) objectEntityController.getObjectFromList(receiver);
 *     if (admin == null) {
 *         System.out.println(RED + "No such admin exists" + RESET);
 *     }
 * }
```

*Code snippet 1: Methods in communication controller that depend on notifier interface*

```
public class EmailMgr implements Notifier {
    /**
     * string containing the email id the system will send emails from
     * string containing the password of the email id the system will send emails from
     */
    private static final String systemEmail = "starsplanner@gmail.com";
    private static final String password = "$tarsplanner21";
}
```

*Code snippet 2: EmailMgr class that implements notifier interface*

```
public interface Notifier {
    /**
     * Sends a notification to a user
     * @param recipient recipient of the notification
     * @param subject subject of the notification
     * @param message message in the notification
     */
    void sendMessage(User recipient, String subject, String message);
}
```

*Code snippet 3: Notifier interface*

- **Liskov Substitution Principle (LSP)**

Liskov Substitution principle has been implemented in the findObject function in SystemMgr. It takes a list of objects and another object as its parameters. Its function is to find the object in the list and return it if it exists. The equals method of the Object class is invoked and has been overridden for classes Course, Index and Student. This method will return a Boolean value indicating if the objects being compared are equal on the basis of their unique attributes.

The derived classes are substitutable for its base class method as its pre-conditions are no stronger than the base class methods and the post-conditions are no weaker than the base class method.

- **Dependency Injection Principle (DIP) / Dependency Inversion Principle**

The communication controller class depends on the interface notifier which is implemented by the EmailMgr class. Thus, communication controller is aware of notifier and EmailMgr is aware of notifier. Communication controller and EmailMgr are unaware of each other. As notifier is an interface, it has less chances of being changed as it itself is an abstraction. If any changes are made to EmailMgr, communication controller will be unaffected by it.

RegistrationMgr is associated with the abstract class ObjectEntityController which are actually instances of its derived classes courseMgr and studentRecordsMgr. RegistrationMgr is unaware of courseMgr and studentRecordsMgr and these 2 derived classes are unaware of RegistrationMgr. These classes are only aware of ObjectEntityController. Any changes made to these classes will not affect the other.

In these cases, high level modules are not dependent on low level modules. Both are dependent on abstractions. Abstractions do not depend upon details. Details depend upon abstractions.

```
public class StudentRecordsMgr extends ObjectEntityController {  
    /**  
     * List that holds all the student objects  
     * String that specifies the file name where student objects are stored  
     */  
    private static ArrayList<Object> students;  
    private static final String studentFile = "student.dat";  
}
```

*Code Snippet 4: StudentRecordsMgr class (extends ObjectEntityController)*

```

public class CourseMgr extends ObjectEntityController {

    /**
     * arraylist of existing courses
     * string containing the name of the course file
     */

    private static ArrayList<Object> courses;
    private static final String courseFile = "course.dat";

    /**

```

*Code Snippet 5: CourseMgr class (which extends ObjectEntityController)*

```

public class RegistrationManager {

    /**
     * controller for list of students
     * controller for list of all available courses
     * controller for list of indices of a particular course
     * controller for notifications
     * controller for timetable of a student
     */

    private ObjectEntityController studentRecordsMgr;
    private ObjectEntityController courseMgr;

}

```

*Code Snippet 6: Registration manager is associated with instances of ObjectEntityController*

## c. OOP Concepts

### • Inheritance

Using inheritance, is-a relationships has been defined in this project. Classes admin and student have been derived from class user. Classes CourseMgr, AdminMgr, IndexMgr and StudentRecordsMgr have been derived from ObjectEntityController.

Method overloading has been implemented in TimeTableMgr in order perform slightly different tasks. Two methods with the same parameter name but different number of parameters have been defined.

```

public boolean checkClash(ArrayList<Index> studentIndex, Index index) {
    for (Lesson lesson : index.getLessons())
    ) {
        for (Index indexRegistered : studentIndex
        ) {
            for (Lesson registeredLesson : indexRegistered.getLessons()
            ) {
                if (lesson.getDay() != registeredLesson.getDay())
                    continue;

                if (lesson.getLessonType().equals(registeredLesson.getLessonType()) && lesson.getLessonType().equals("lab")) {
                    if (lesson.getLabWeek().equals("odd") && registeredLesson.getLabWeek().equals("even"))
                        continue;
                    else if (lesson.getLabWeek().equals("even") && registeredLesson.getLabWeek().equals("odd"))
                        continue;
                }

                LocalTime time1, time2, time3, time4;
                time1 = lesson.getStartTime();
                time2 = lesson.getEndTime();
                time3 = registeredLesson.getStartTime();
                time4 = registeredLesson.getEndTime();

                if ((time1.isBefore(time3) && time2.isAfter(time3)) || time1.compareTo(time3) == 0)
                    return true;
                else if ((time1.isAfter(time3) && time1.isBefore(time4)))

```

*Code snippet 7: first signature and definition of checkClash*

```

//
public boolean checkClash(ArrayList<Index> studentIndex, Index index, Index skipIndex) {
    for (Lesson lesson : index.getLessons())
    ) {
        for (Index indexRegistered : studentIndex
        ) {
            if (indexRegistered.equals(skipIndex))
                continue;
            for (Lesson registeredLesson : indexRegistered.getLessons()
            ) {
                if (lesson.getDay() != registeredLesson.getDay())
                    continue;

                if (lesson.getLessonType().equals(registeredLesson.getLessonType()) && lesson.getLessonType().equals("lab")) {
                    if (lesson.getLabWeek().equals("odd") && registeredLesson.getLabWeek().equals("even"))
                        continue;
                    else if (lesson.getLabWeek().equals("even") && registeredLesson.getLabWeek().equals("odd"))
                        continue;
                }

                LocalDateTime time1, time2, time3, time4;
                time1 = lesson.getStartTime();
                time2 = lesson.getEndTime();
                time3 = registeredLesson.getStartTime();
                time4 = registeredLesson.getEndTime();
            }
        }
    }
}

```

*Code snippet 8: second signature and definition of checkClash*

The first signature defines a function that checks if a particular index clashes with any of the other indices in the arraylist of indices (which would be the registered/ waiting list of indices of a course for a student). This function is required when a student is adding an index of a course.

The second function performs a similar task which a slight modification. It takes in a third parameter (unlike the previous function) which indicates which index of the arraylist should not be checked with the second parameter regarding the clash. This is particularly useful in the case when a student is changing the registered index of a course or swapping an index with another student. Clash of the index to be added should not be computed with the index that is going to be dropped.

Appropriate visibility modifiers have been used for each of the classes, attributes as well as methods. Most of the data members have been declared as private or protected while the methods have been declared as public.

Abstract classes used include User and ObjectEntityController. These classes have general features that can be shared by the derived classes. No object that would be meaningful for this application can be derived from these classes.

```

//
public abstract class ObjectEntityController {
    /**
     * SystemMgr object
     * FileMgr object
     */
    protected SystemMgr systemMgr;
    protected FileMgr fileMgr;
    /**

```

*Code snippet 9: abstract class ObjectEntityController*



```

//
public abstract class User implements Serializable {
    /**
     * name is the name of the user
     * networkName is th unique identification name of the user
     * emailId is the user's email Id
     * gender is the gender of the user
     * nationality od the nationality of the user
     */
    protected String name;
    protected String networkName;
    protected String emailID;
    private String gender;
    private String nationality;

    // dummy constructor for checking if Student/Admin exist in database
}

```

*Code snippet 10: abstract class User*

The interface notifier has been implemented in EmailMgr. The function sendMessage has been overridden in order to send an email to a particular recipient with a given subject and body.

## • Polymorphism

Polymorphism has been implemented by function overriding. The findObject method of class SystemMgr uses the equals method to determine if the object required is in the list of objects. The equals method has been overridden in each of the entity classes so that comparison is made based on the unique attribute of that respective class. For example, an index object can be identified by its index number, a course object can be identified by its course number and so on.

```

public Object findObject(ArrayList<Object> objects, Object object) {
    if (objects == null)
        return null;
    for (Object objectItem : objects)
    {
        if (objectItem.equals(object)) {
            return objectItem;
        }
    }
    return null;
}

```

*Code snippet 11: findObject method with the equals method being invoked*

```

public boolean equals(Object object) { return (courseCode.equals(((Course) object).getCourseCode())); }

```

*Code snippet 12: equals method being overridden in class Course*

```

//
public boolean equals(Object object) { return (indexNumber == ((Index) object).getIndexNumber()); }

```

*Code snippet 13: equals method being overridden in class Index*

Object typecasting has also been used in several instances. In the functions of classes SystemMgr and FileMgr, the actual parameters (which will be Course, Student, Index or Admin objects in this application) are assigned to formal parameters which are of type Object (base class)

```

*/
public void saveObjects(ArrayList<Object> objects, String fileName) {
    FileOutputStream fileOutputStream;
    ObjectOutputStream objectOutputStream;
    // Write objects to file
    try {
        fileOutputStream = new FileOutputStream(new File(fileName));
        objectOutputStream = new ObjectOutputStream(fileOutputStream);
        objectOutputStream.writeObject(objects);
        objectOutputStream.close();
        fileOutputStream.close();
    } catch (EOFException e) {
        System.out.println(RED + "System error" + RESET);
    }
}

```

*Code snippet 14: Object upcasting in saveObjects method of FileMgr*

Object downcasting has also been done in class CourseMgr. In many of the functions, the existingCourse object that is of type Object has been downcasted to a Course object after checking that existingCourse contains a Course object of a valid course.

```

*/
public void updateCourseName(String courseCode, String newCourseName) {
    loadCourseObjectList();
    Object existingCourse = getObjectFromList(courseCode);
    if (existingCourse != null) {
        ((Course) existingCourse).setCourseName(newCourseName);
        saveCourseObjectList();
        System.out.println(GREEN + "Updated course" + RESET);
        ((Course) existingCourse).print();
    } else {
        System.out.println(RED + "Course doesn't exist in the database" + RESET);
        saveCourseObjectList();
    }
}

```

*Code snippet 15: Use of downcasting in updateCourseName method*

## 2. Testing

### 1. Student login

Test Case	Output
Login before allowed period	<pre> +-----+   Enter your user name :   Atul0123 +-----+   Enter your password :   +-----+ You are not allowed to register for course now Current start date and time: 2021-01-01 00:00 Current end date and time: 2021-01-20 23:59 </pre>

Login after allowed period	<pre> +-----+   Enter your user name :   Atul0123 +-----+   Enter your password :   +-----+ You are not allowed to register for course now Current start date and time: 2020-06-01 00:00 Current end date and time: 2020-06-30 23:59 </pre>
Wrong password	<pre> +-----+   Select your domain      +-----+   1: Admin                  2: Student                0: Exit                 +-----+ 2 +-----+   Enter your user name :   Atul0123 +-----+   Enter your password :   +-----+ The username or password is incorrect </pre>

## 2. Add a student

Test Case	Output
Add a student	<pre>+-----+   Enter the Student's name :   Ryan +-----+   Enter the Student's username :   Ryan0123 +-----+   Enter the Student' matriculation number :   U189263V +-----+   Enter the Student's email address :   ryan001@e.ntu.edu.sg +-----+   Enter the Student's gender :   M +-----+   Enter the Student's nationality :   Singaporean +-----+   Enter the Student's school :   EEE +-----+   Enter the Student's study year :   3 +-----+   Enter your password :  </pre>
	<pre>+-----+   Ankitha   Ankitha0123   F   Indian   +-----+   Nisha   Nisha0123   F   Indian   +-----+   Srishti   Srishti0123   F   Indian   +-----+   Anatrika   Anatrika0123   F   Indian   +-----+   Agnesh   Agnesh0123   M   Indian   +-----+   Saiteja   Saiteja0123   M   Indian   +-----+   Kartikeya   Kartikeya0123   M   Indian   +-----+   Shyam   Shyam0123   M   Indian   +-----+   Aneez   Aneez0123   M   Indian   +-----+   Kirthana   Kirthana0123   F   Indian   +-----+   Aditya   Aditya0123   M   Indian   +-----+   Ananya   Ananya0123   F   Indian   +-----+   Madhav   Madhav0123   M   Indian   +-----+   Atul   Atul0123   M   Indian   +-----+   Parthan   Parthan0123   M   Indian   +-----+   Kanishk   Kanishk0123   M   INDIAN   +-----+   Amruta   Amruta0123   F   INDIAN   +-----+   Ryan   Ryan0123   M   SINGAPOREAN   +-----+</pre>

Add an existing student	<pre> +-----+   Enter the Student's name :        Ankitha +-----+   Enter the Student's username :    Ankitha0123 +-----+   Enter the Student' matriculation number :   U1923151C +-----+   Enter the Student's email address :   ankitha001@e.ntu.edu.sg +-----+   Enter the Student's gender :       F +-----+   Enter the Student's nationality :   Indian +-----+   Enter the Student's school :       SCSE +-----+   Enter the Student's study year :   2 +-----+   Enter your password :   +-----+ System cannot add this student as student already exists </pre>
Invalid data entry	<pre> +-----+   Enter the Student's name :        Ryan +-----+   Enter the Student's username :    Ryan0123 +-----+   Enter the Student' matriculation number :   U9128357V +-----+   Enter the Student's email address :   ryan001@e.ntu.edu.sg +-----+   Enter the Student's gender :       M +-----+   Enter the Student's nationality :   Singaporean +-----+   Enter the Student's school :       EEE +-----+   Enter the Student's study year :   5 +-----+ Study year only between 1-4 </pre>

### 3. Add a course

Test Case	Output
Add a new course	<pre> Enter the course code : CZ2003 Enter the course name : Computer Graphics and Visualization Enter the academic units for this course : 3 Enter the school offering this course : SCSE +-----+   CZ2002   Object Oriented Programming   3   SCSE   +-----+   CZ2005   Human Computer Interaction   3   SCSE   +-----+   CZ2006   Software Engineering   3   SCSE   +-----+   BU0625   Marketing in 21st Century   3   NBS   +-----+   HP0806   Psychology of Stress Management   3   SSS   +-----+   CZ2003   Computer Graphics and Visualization   3   SCSE   +-----+ </pre>
Add an existing course	<pre> Enter the course code : CZ2002 Enter the course name : OODP Enter the academic units for this course : 3 Enter the school offering this course : SCSE Course already exists: System cannot add this course </pre>
Invalid data entries	<pre> Enter the course code : CZ2003 Enter the course name : CGv Enter the academic units for this course : -1 Credits cannot be less than 1 </pre>

#### 4. Register Student for a course

Test Case	Output
Add a student to a course index with available vacancies.	<pre> Enter the course code of the course to add: CZ2005 Enter the index number of the course to add: 11983 Registering Atul0123 to 11983 Successfully Registered: Please print your updated time table </pre>
Add a student to a course index with 0 vacancies in Tut / Lab.	<pre> Enter the course code of the course to add: CZ2002 Enter the index number of the course to add: 10829 No more vacancies left for this course You have been added to waiting list for: CZ2002 Index: 10829 </pre>
Register the same course again	<pre> Enter the course code of the course to add: CZ2002 Enter the index number of the course to add: 10034 Registration failed: You are already registered for this course </pre>
Invalid data entries	<pre> 1 Enter the course code of the course to add: CZ2003 Enter the index number of the course to add: 102343 Registration failed: This course doesn't exist in the system </pre>

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

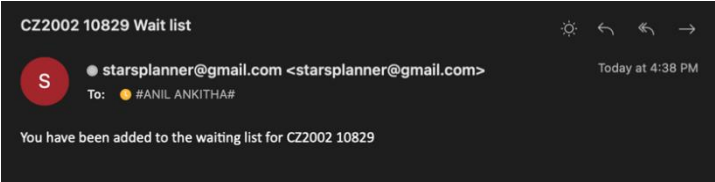
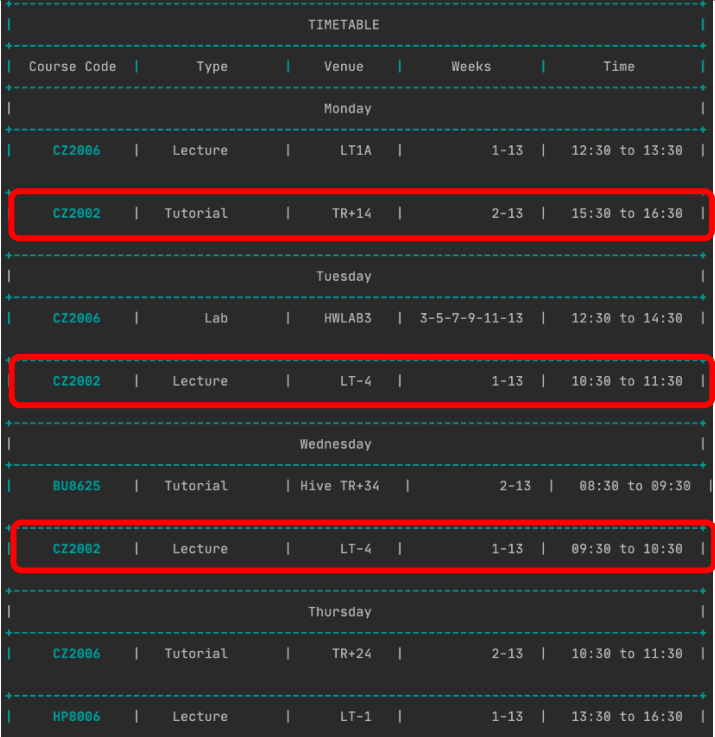
Test Case	Output
Check vacancy in course index	<pre> Enter the course code: CZ2002 Enter the index number: 10829 +-----+   Index number   Vacancy   +-----+   10829          1/10      +-----+ </pre>
Invalid data entries	<pre> Enter the course code: CZ2002 Enter the index number: 10029 Index doesn't exists in the database </pre>

#### 6. Day/time clash with another course

Test Case	Output
Add a student to a course index with available vacancies	<pre> Enter the course code of the course to add: CZ2006 Enter the index number of the course to add: 10129 Registration failed: Clashes with a registered course's index </pre>

#### 7. Waitlist notification

Test Case	Output
-----------	--------

Add student A to course index with 0 vacancies	<pre> Enter the course code of the course to add: CZ2002 Enter the index number of the course to add: 10829 No more vacancies left for this course You have been added to waiting list for: CZ2002 Index: 10829 Notification Sent to ankitha001@e.ntu.edu.sg </pre> 
Drop studentB from the same course index	<pre> CZ2002 10829 3 Registered Enter the course code to remove: CZ2002 Enter the index to remove: 10829 De-registering Parthan0123 from10829 Index Number 10829 is removed from registered list Notification Sent to parthan001@e.ntu.edu.sg </pre>
Display student A timetable	 <pre> CZ2006 10135 3 Registered HP8006 17854 3 Registered BU8625 10953 3 Registered CZ2002 10829 3 Registered </pre>

8. Print student list by index number, course

Test Case	Output
Print list by course	<pre> Enter the course code : CZ2004 +-----+-----+-----+-----+   Name   Username   Gender   Nationality   +-----+-----+-----+-----+   Atul   Atul0123   M   Indian   +-----+-----+-----+-----+ </pre>

Print list by index	<pre> Enter the course code : CZ2002 Enter the index number : 10034 -----        Name             Username             Gender             Nationality -----        Atul             Atul0123             M                  Indian ----- </pre>
Invalid data entries	<pre> Enter the course code : CZ2002 Enter the index number : 10289 Index doesn't exists in the database </pre>

**YouTube link:** <https://www.youtube.com/watch?v=FIxZXulYwXo&t=160s>

### **Notes:**

- If the class diagram is unclear or blurry, view the image “Class\_Diagram” present in the diagrams folder. All the diagrams are available in the diagrams folder.
- To run the program in Windows, execute run.bat file present in the zip folder.
- Alternatively, type the following commands from the main folder:

cd src

javac -d ../classes -cp ../jars/javax.mail.jar;../jars/activation.jar entity/\*.java  
controller/\*.java boundary/\*.java

cd ../classes

java -cp ../jars/javax.mail.jar;../jars/activation.jar; boundary.MyStarsInterface

- To run on MacOS, replace ; with : in all the commands