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
BHARGAV SINGAPURI	CZ2002	DSAI 2	24/11/2020
GAN KAH EE	CZ2002	DSAI 2	24/11/2020
HAY SHU YING NICOLETTE	CZ2002	DSAI 2	24/11/2020
TIMOTHY TAN CHOONG YEE	CZ2002	DSAI 2	24/11/2020
WOON JIA HUI	CZ2002	DSAI 2	29/11/2020

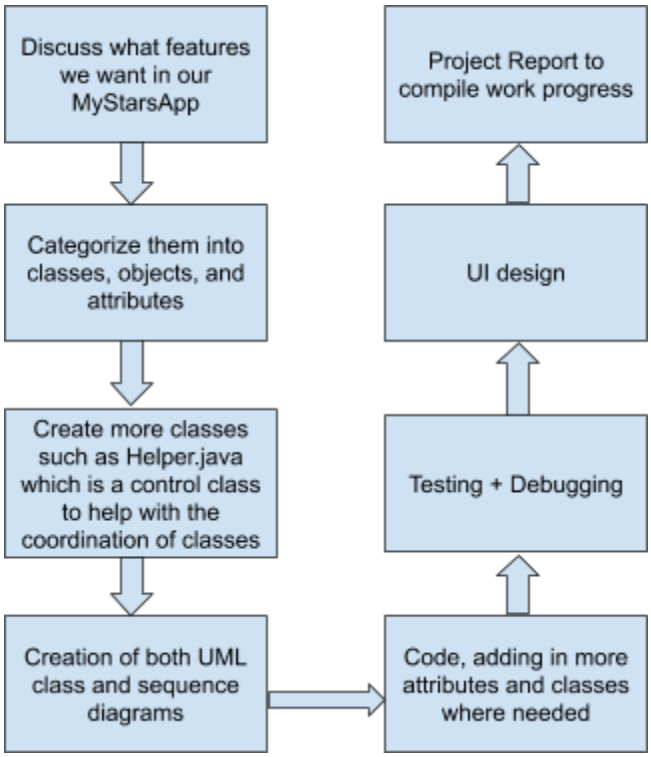
Important notes:

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

Contents Page

A. APPROACH & DESIGN CONSIDERATIONS

Approach



Design Considerations

The application will contain 3 types of classes. There is the User class that will be used by users of the application, the Views, which the User interacts with and the Database, which handles the reading and writing to file. Users will be separated into Students and Administrators based on their respective roles. They will also see their own respective views based on the functions that they are allowed to perform on the application, which will be managed by the Views. The Database will ensure that all the changes made by the users is persistent and the application will return to its previous state on restart. This demonstrates the **Single Responsibility Principle** where each type of class will have its own role. We used various other OO Concepts as well as SOLID design principles in the development of the application, which will be explained in the pages to come.

For **encapsulation**, we hide information with the use of private modifiers. For example, the implementation of password encryption in the User class is hidden. Even its subclasses, `Admin` and `Student` do not need to know how it is implemented. This simplifies the program and increases password protection. Additionally, information hiding is controlled using getter and setter methods such as `getUsername()` and `changePassword()`.

```
protected User(String username, String password, AccessLevel accessLevel) {
       this.username = username;
       this.password = encryptPassword(password);
       this.accessLevel = accessLevel;
}
public boolean changePassword() {...}
public String getUsername() {...}
private byte[] encryptPassword(String rawString) {...}
```

We used **inheritance** and **abstraction**, which can be seen from the various subclasses such as `StudentMainView` gaining properties from the abstract `View` parent class. The abstract method `print()` is a necessary method for all `View` classes since the purpose of `View` classes are to print the user interface. By making this method abstract, we ensure it is implemented in all subclasses. The protected method `clearScreen()` is inherited and reused. This method should only be used by `View` classes in the context of the `MySTARS` package, hence it is protected.

```
       public abstract void print();

       protected void clearScreen(String directory) {...}
```

The overriding of the `print()` method could be used to demonstrate **polymorphism**, but it was unnecessary in our implementation since there is no instance when a generic `View` type object is created. As shown below, we always know exactly which view we are navigating to.

```
       switch (choice) {
                      case 1:
                          VacanciesView vacanciesView = new VacanciesView();
                          vacanciesView.print();
                          break;
                      case 2:
                          AddCourseView addCourseView = new AddCourseView();
                          addCourseView.print();
                          Break;
       ...
       }
```

Some **exception handling** was done to catch errors, including those for course codes, usernames and date and time. All non-super classes are also marked final to improve compile efficiency.

Open closed principle: Under the `User` class, there is **abstraction** for subclasses `Admin` and `Student`. We can modify the methods in the subclass by extending or overriding methods in the `User` class. `User` is open for extension but closed for modification.

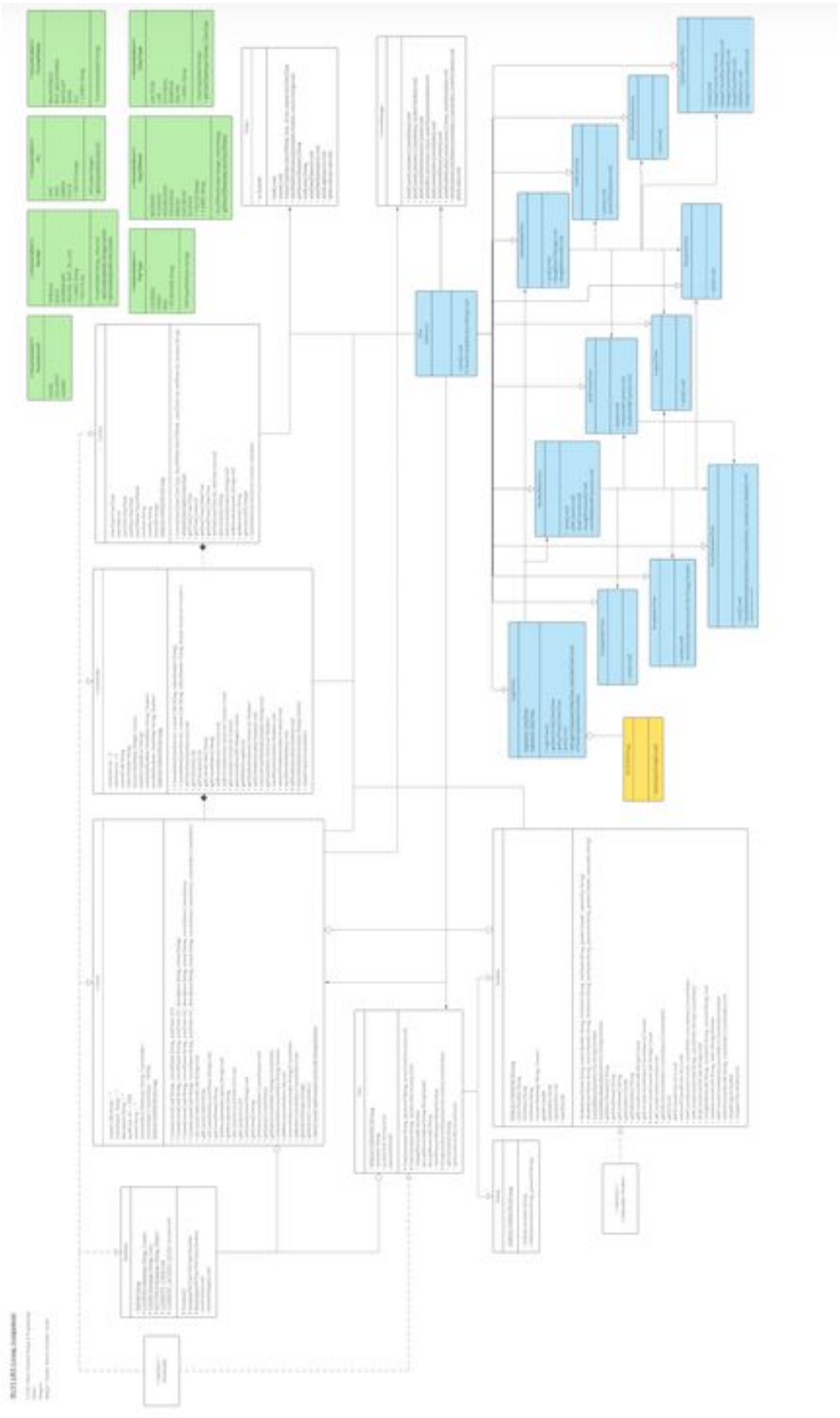
Liskov substitution principle: Again, a form of **inheritance**, for the `View` subclasses such as `LoginView` and `LogoutView`, when we override the `print()` method, we do not expect more nor provide less than the base class.

Additionally, for a **user-friendly interface**, we have incorporated features such as a coloured console header and also the student name list that is printed in alphabetical order. The quit (Q) option is available to allow for internal locus of control. Moreover, there is a path to the current page, for example, '`Student Main > Check Vacancies`', to indicate to the User which page they are on.

Assumptions: Only 1 student can only access MySTARS at any one time; When course is created by Admin, overlaps in timing are only considered for the same course; No waivers are allowed i.e. students are not allowed to take 2 or more courses with the same timing for any lesson; No student can access the MySTARS portal past the cut off time.

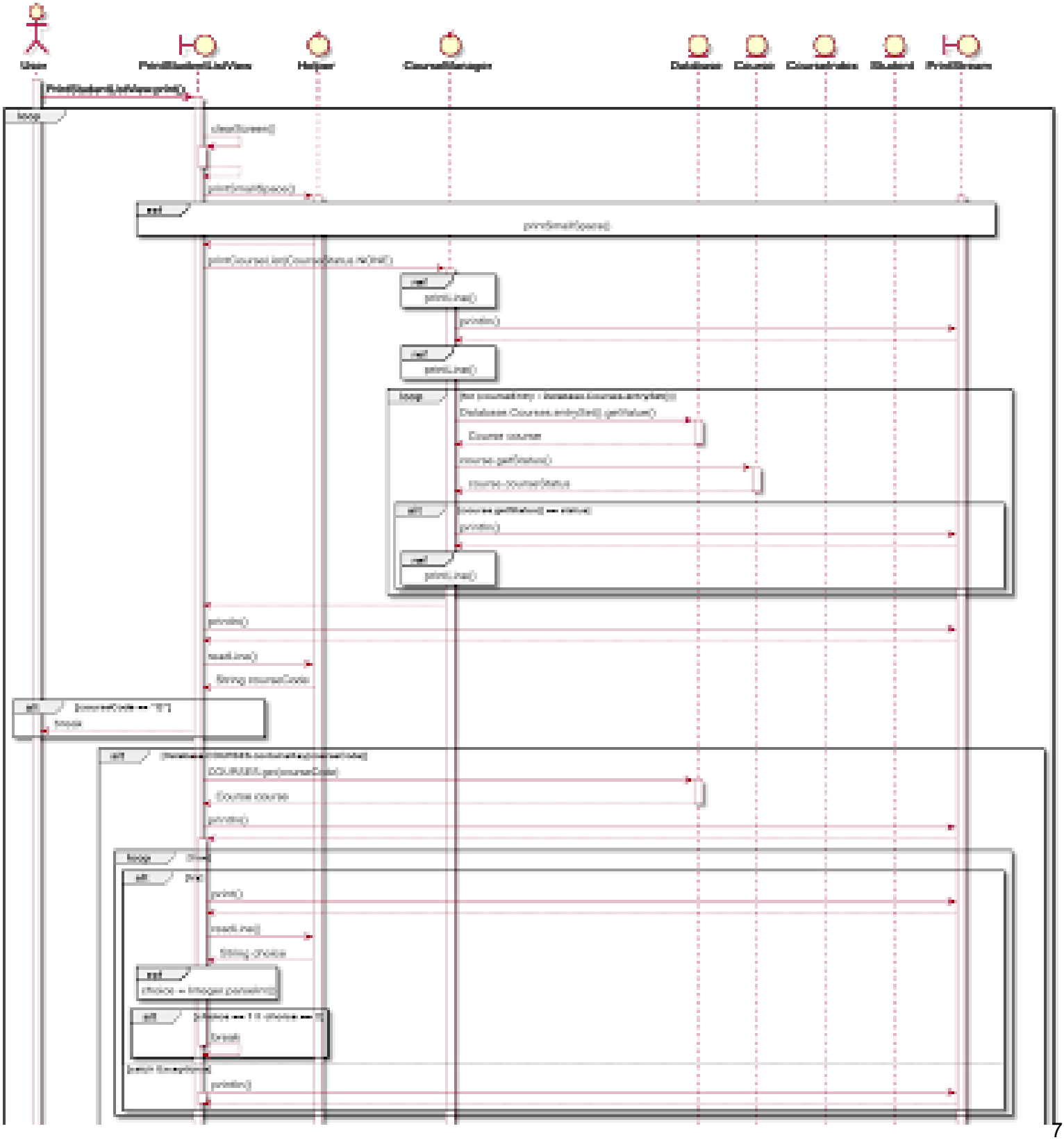
Link to the video demonstration: https://youtu.be/GoBr9TRGkOs

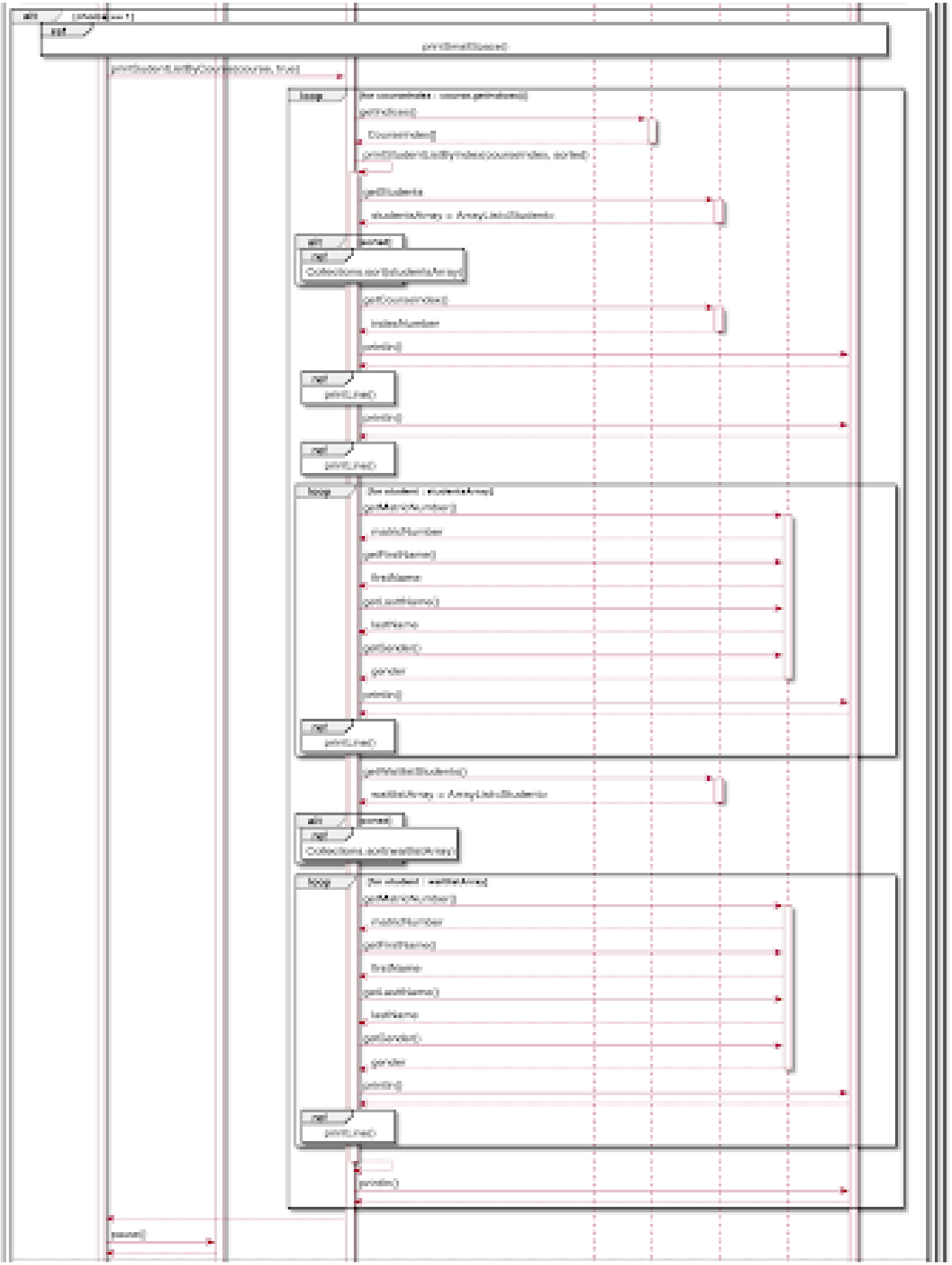
B. UML Class Diagram *(for higher resolution please refer to: https://github.com/majulahsingapuri/MySTARS/blob/master/UML/UML_Class_Diagram_withTitle.png)*

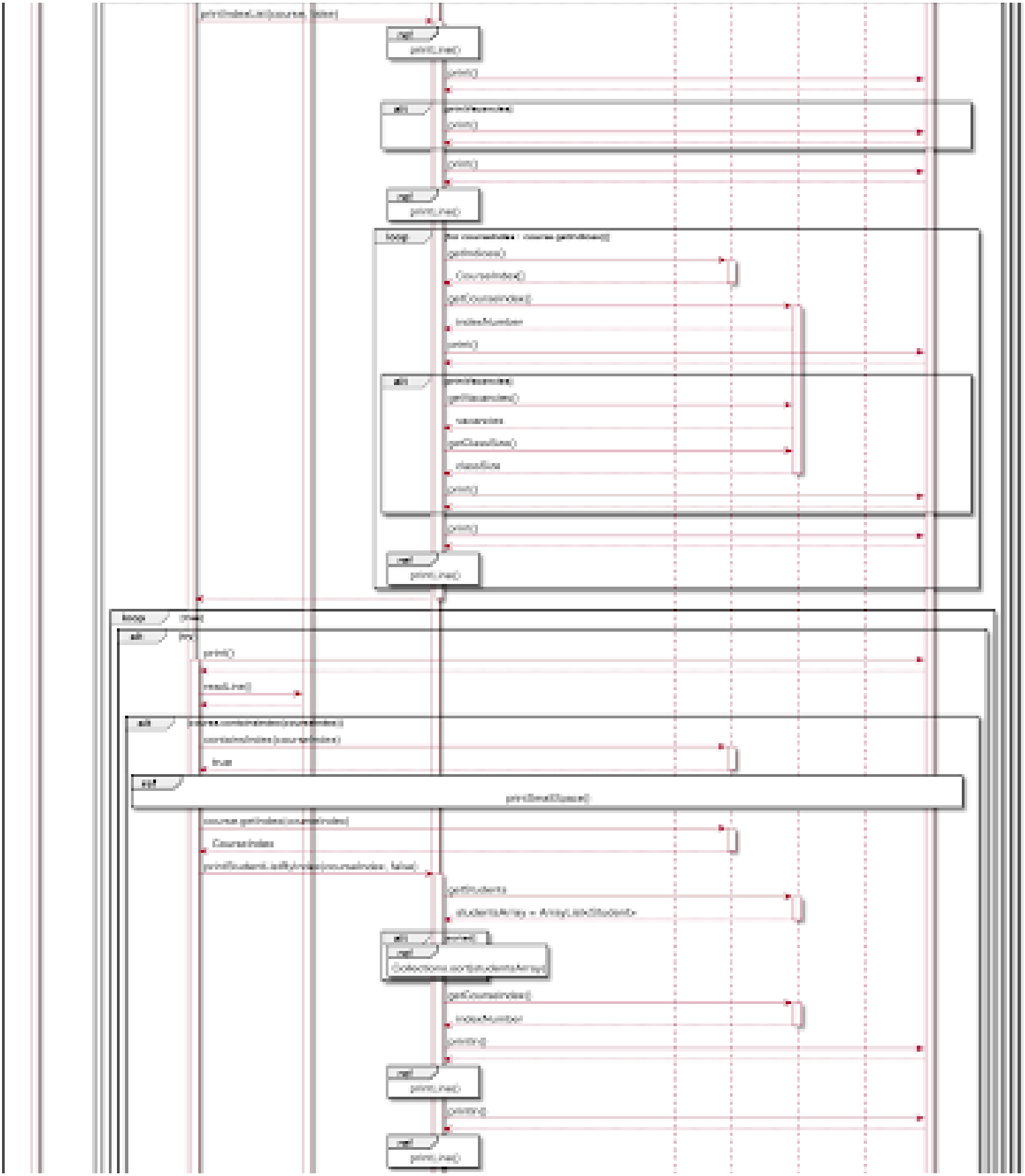


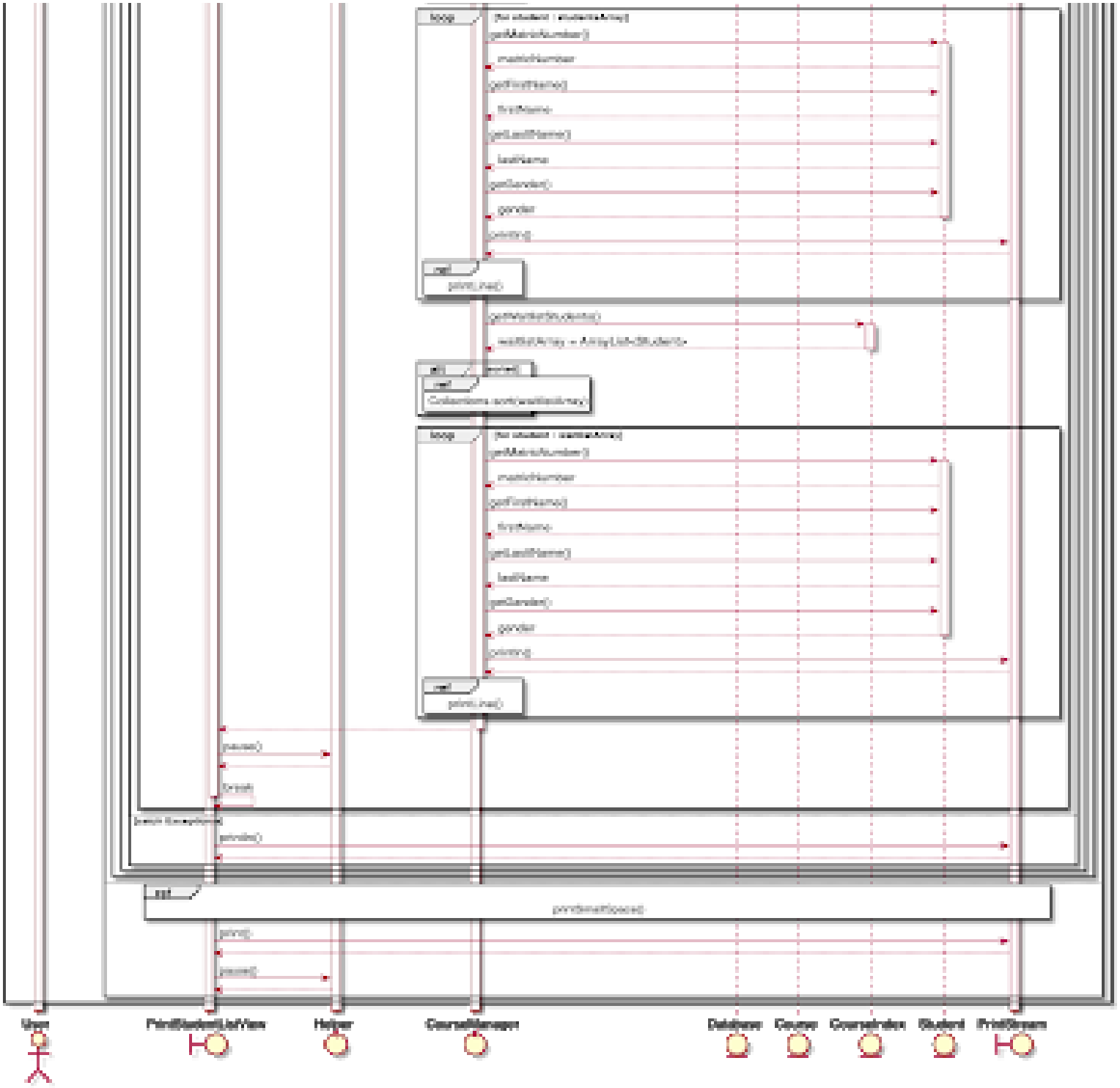
C. UML Sequence Diagram *(for higher resolution please refer to: https://github.com/majulahsingapuri/MySTARS/blob/master/UML/MySTARS.png)*

The use case below is when the Admin wants to print a student list by course.





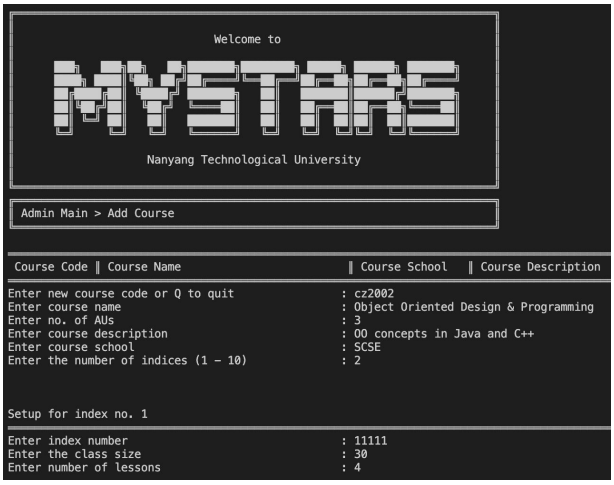


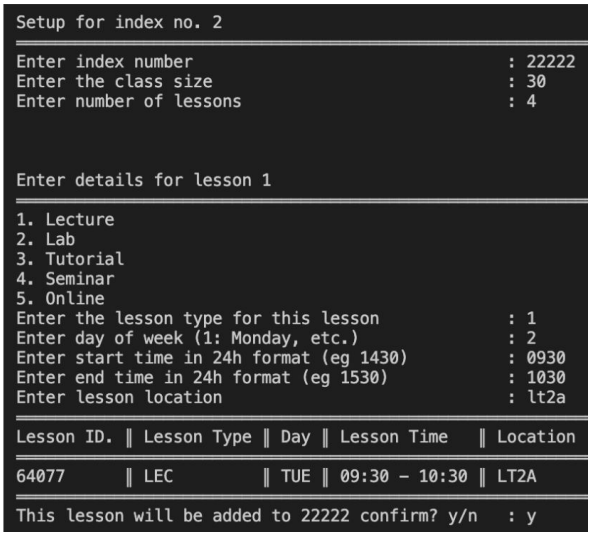

loop
getMatricNumber()
matricNumber
getFirstName()
firstName
getLastName()
lastName
getGender()
gender
println()
opt
printList()
getMatricStudents()
matricArray = ArrayList<Student>
alt
[else]
opt
Collections.sort(matricArray)
loop
getMatricNumber()
matricNumber
getFirstName()
firstName
getLastName()
lastName
getGender()
gender
println()
opt
printList()
pause()
finish
[catch Exception]
println()
opt
printInstitution()
print()
pause()
User
PrintStudentListView
Helper
CourseManager
Database
Course
CourseIndex
Student
PrintStream

D. TEST CASES & RESULTS

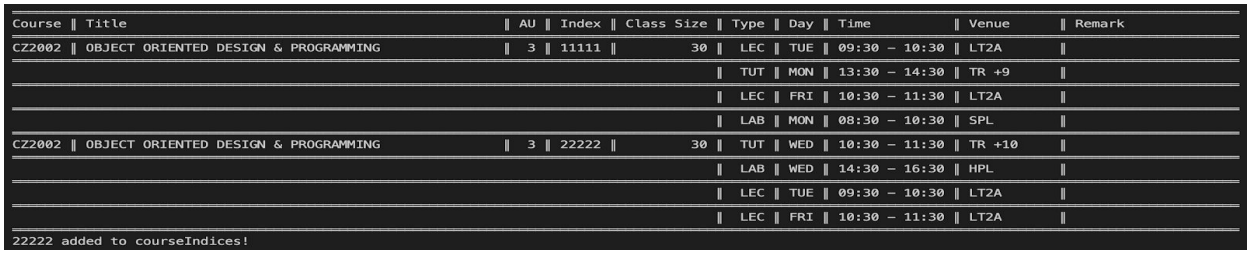
Admin username: 'Admin' or 'admin'; Password: AdminPassword

Within Admin mode, as seen in the above picture, we are able to create courses. For example, we created a course CZ2002 OODP with 2 indices 11111 and 22222, as seen in the pictures below. The 2 indices have different tutorial and lab timings.





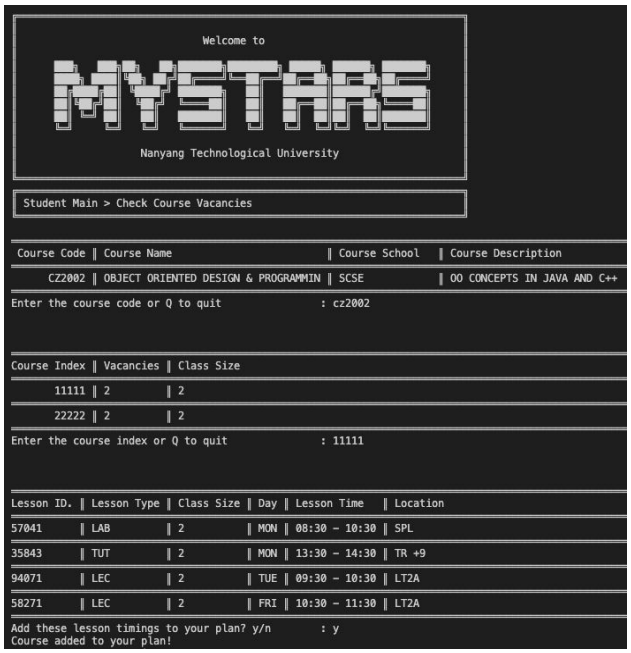
After finalising the details for both indices for CZ2002, all courses created so far will be displayed.

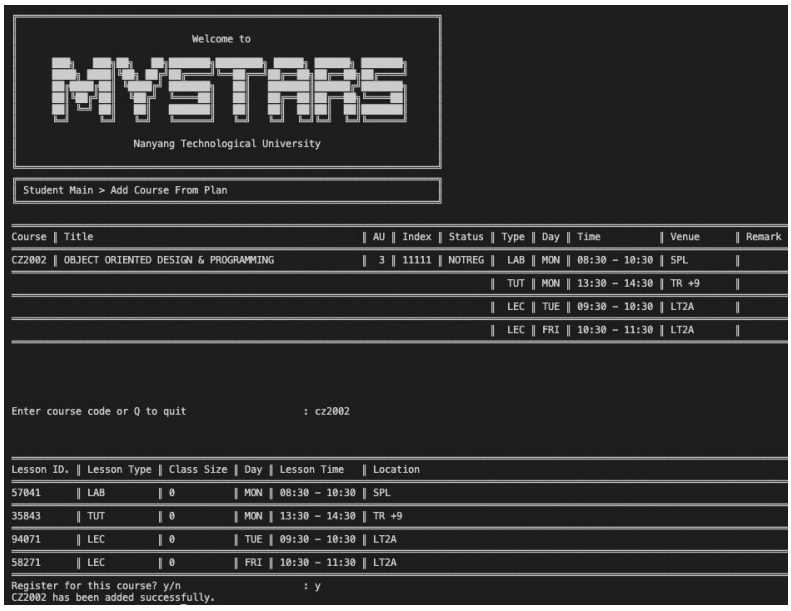


Student to enter personal username; Default password: OODP1s7heB3st

Now in Student mode, we are able to check vacancies of courses and add to plan before registering for them.

For illustration purposes, we are adding CZ2002 index 11111 to the plan, before registering for it, as seen in the pictures below.check

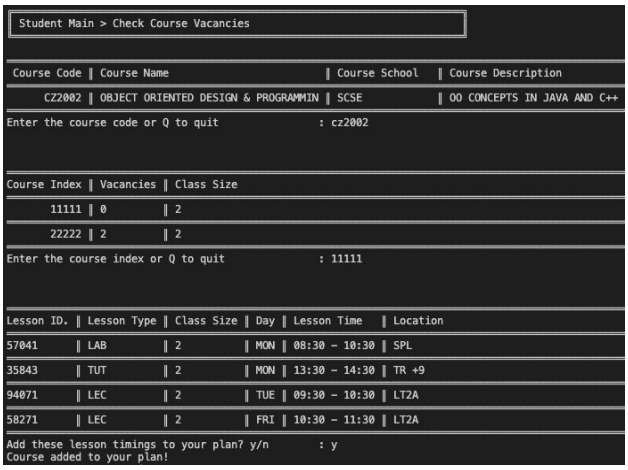


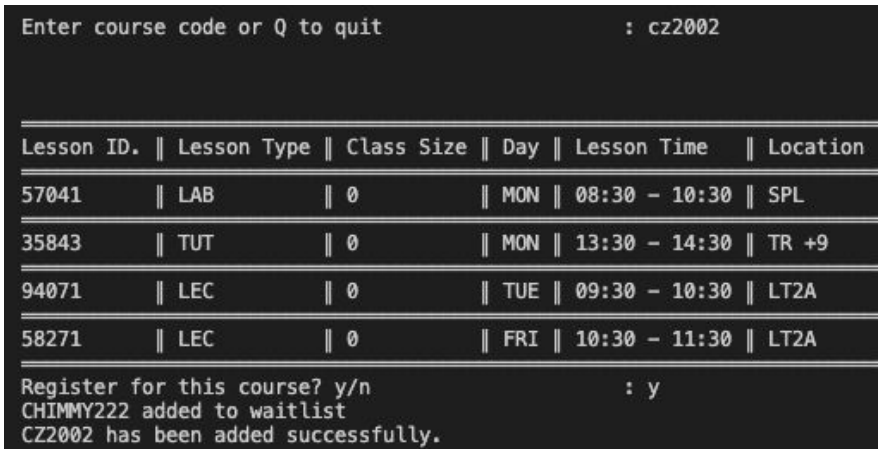


After registering for their desired courses, the Student can check and print their timetable. For example, in the picture below, a student by the name of Gary has registered for CZ2002 OODP, and he needs to

attend the following classes on the specified days.

In the event when a Student tries to register for an index with 0 vacancies, he or she will be placed on the waitlist. As shown in the pictures below, when a Student checks vacancies for CZ2002 OODP, there are 0 vacancies in index 11111 but the course can still be added to the plan. However, when the Student





wants to register for CZ2002, he is added to the waitlist instead.

When another student is removed from the register for CZ2002 and the waitlisted student is removed from the waitlist, he or she will be automatically registered for the course. An email notification will also be sent, as shown below.