CE2002: Object-Oriented Design & Programming

SE2 Assignment Group 3

My Student Automated Registration System (MySTARS)

Done by:

Hoy Di Sheng Max (U1922110F)

Goel Tejas (U1923301G)

Vedula Kartikeya (U1923891G)

Agrawal Rachita (U1922919L)

Gavin Neo Jun Hui (U1921265L)

Table of Contents:

I. Introduction

My Student Automated Registration System(MySTARS) is a console-based non-graphical user interface application. Key features have been covered in this application such as students not being able to login outside the access period, admin updating course code and students getting time clashes between modules.

This report will cover the concepts learnt in this course and key design considerations used to implement the application. Appropriate class and sequence diagrams have been shown to understand the code better. Moreover test cases have been included which proves that the application works well and passes all the test cases mentioned in the assessment requirement.

II. Design Considerations

A. Approach Taken

OOP concepts have been applied comprehensively in this project which helped us to provide a more efficient and cleaner style of design, implementation and error detection. This allowed us to better visualize and understand the requirements. Hence, the team was able to communicate well with each other and come to a common ground.

The architectural style that we have applied is the Layered (n-tier) architectural style. The code is arranged such that input/data enters the top layer (boundary class) and works its way down each layer (control classes) until it reaches the last layer which is the database (entity classes). Along the way, each layer has a specific task, i.e. the top layer is responsible for taking the user inputs, middle layers are responsible for maintaining data consistency and control behavior specific to one or more use cases and the last layer is responsible for storing all the information required i.e. database. Each layer can be updated and enhanced separately which makes sure that the code and design is easy to maintain and modify.

B. Designs Principles Used

B.1 Single Responsibility Principle

The principle states that no more than one justification for a class to change should be given, making the code more coherent. Our team ensures the principle that each class has only one responsibility.

An example would be the Student class, it only manages the attributes of a single student. Similarly for the Course class, it only manages the attributes of a single course - course index, course code, course name, etc. This principle is applied throughout the rest of the code and design structure, including the Control and Boundary classes. This is done to ensure cohesiveness and it reduces functional overlaps.

B.2 Open Closed Principle

The principle uses interfaces instead of superclasses to allow different implementations which we could easily substitute without changing the code that uses them. The interfaces are closed for modifications, and you can provide new implementations to extend the functionality of the software. With this approach it adds another level of abstraction which enables loose coupling. The implementation of classes are independent of each other and don't need to share any code amongst each other. Therefore, you can consider the benefits of both and use either inheritance or composition.

B.3 Liskov Substitution Principle

The principle defines that objects of a superclass shall be replaceable with objects of its subclasses without breaking the application. The Student and Admin classes are able to replace the Person class, and outputs the necessary outputs, i.e. getName() method. The preconditions for the 2 subclasses are not stronger than the base class, also its post conditions are not weaker than the base case.

III. Object Oriented Concepts (UML diagram)

Composition

We use composition to show a whole-part relationship between 2 classes. An example of composition that we have implemented is between classes IndexNumber and Lesson.

This is because an index of a course needs to have some kind of lessons, such as tutorials and lectures, to exist. The composition shows the IndexNumber 'has a' relationship with the Lesson class, and the Lesson class cannot exist without the index group. Classes that exist as a composition relationship are Course and IndexNumber classes, also Person and Password classes.

Dependency

Dependency is used when there are no relationships between the 2 classes, but a class uses another class' method to function and outputs required information. Our Admin and Student boundary classes use the methods in AdminControl and Student Control to function and work, such as adding a course, based on the user inputs.

Data Structure

For file I/O, we serialized and deserialized objects to the file. We used a ".dat" file(binary file) over a text file because a ".dat" file is a generic data file that stores specific information relating to the code that created the file and it stores data which is not necessarily plain text. It is easier to perform read/write operations on a ".dat" file than a ".txt" file.

For students on waitlist, we implemented a queue so that whenever a vacancy is present in a specific index number, the student at the head of the queue will be allocated the course first.

Abstraction

Abstraction is used when we have classes that have similar attributes and methods. A superclass is created and contains general features, such that it can be shared among the subclasses. For our assignment, we created a Person abstract class, and subclasses Admin and Student. The two subclasses similar attributes, such as name and email address. The sub classes also have their unique attributes and methods.

Encapsulation/Information Hiding

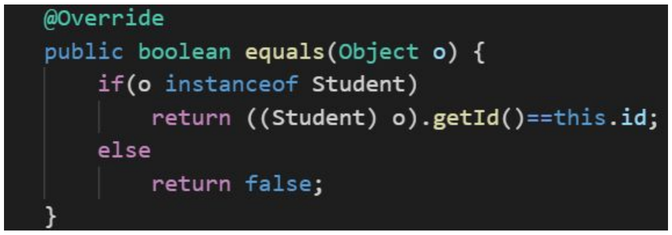
With encapsulation it allows us to reuse functionality without sacrificing on security. It is a very useful concept as it can help us save a lot of time. For example, we set the attributes in every class to private as we only want to allow that particular class to access it. However, in the "Person" class, we set the attributes to protected as the attributes have to be accessed by the "Admin" and "Student" class which are in the same package as the "Person" class i.e. Entity class and classes/methods of other packages should not be able to modify the details of student and admin.

Inheritance

Inheritance is defined as the process where one class acquires the properties (methods/behaviour) of another. The class which inherits the properties of others is known as subclass (derived class/child class) and the class whose properties are inherited is known as superclass (base class, parent class). To inherit a parent class, the child class has to use the keyword "extends". In our code, the "Person" class is the parent and the "Admin" and "Student" class are the child classes inheriting the "Person" class.

Polymorphism

Polymorphism works by using a reference to a parent class to affect an object in the child class. In method overriding, the child class can use the OOP polymorphism concept to override a method of its parent class. That allowed us to use one method in different ways depending on whether it's invoked by an object of the parent class or an object of the child class.



```
@Override
public boolean equals(Object o) {
    if(o instanceof Student)
        return ((Student) o).getId()==this.id;
    else
        return false;
}
```

IV. Extensibility

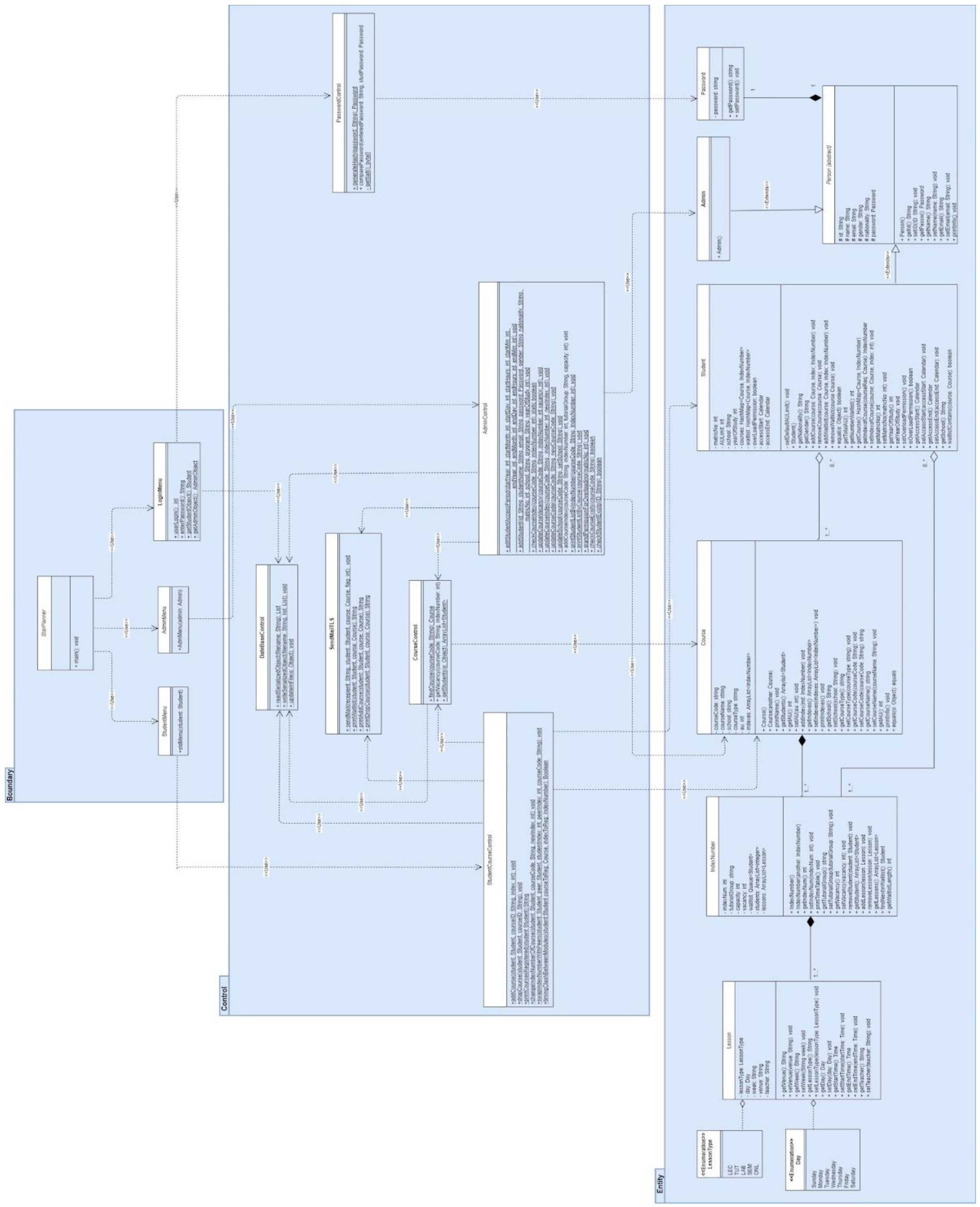
In order to cater the additional requirement of sending notifications other than email, classes can be to be implemented to the Notification interface, which its methods can be used by both Student Control and Admin Control classes, to send a mode of notification, based on the calling of the method.

Notify via SMS and WhatsApp: In this case, instead of a class named "SendMailTLS" we can have a class named "NotificationViaChoice". In this new class the notification will be sent according to the choice of the admin by adding the relevant functions for SMS and WhatsApp.

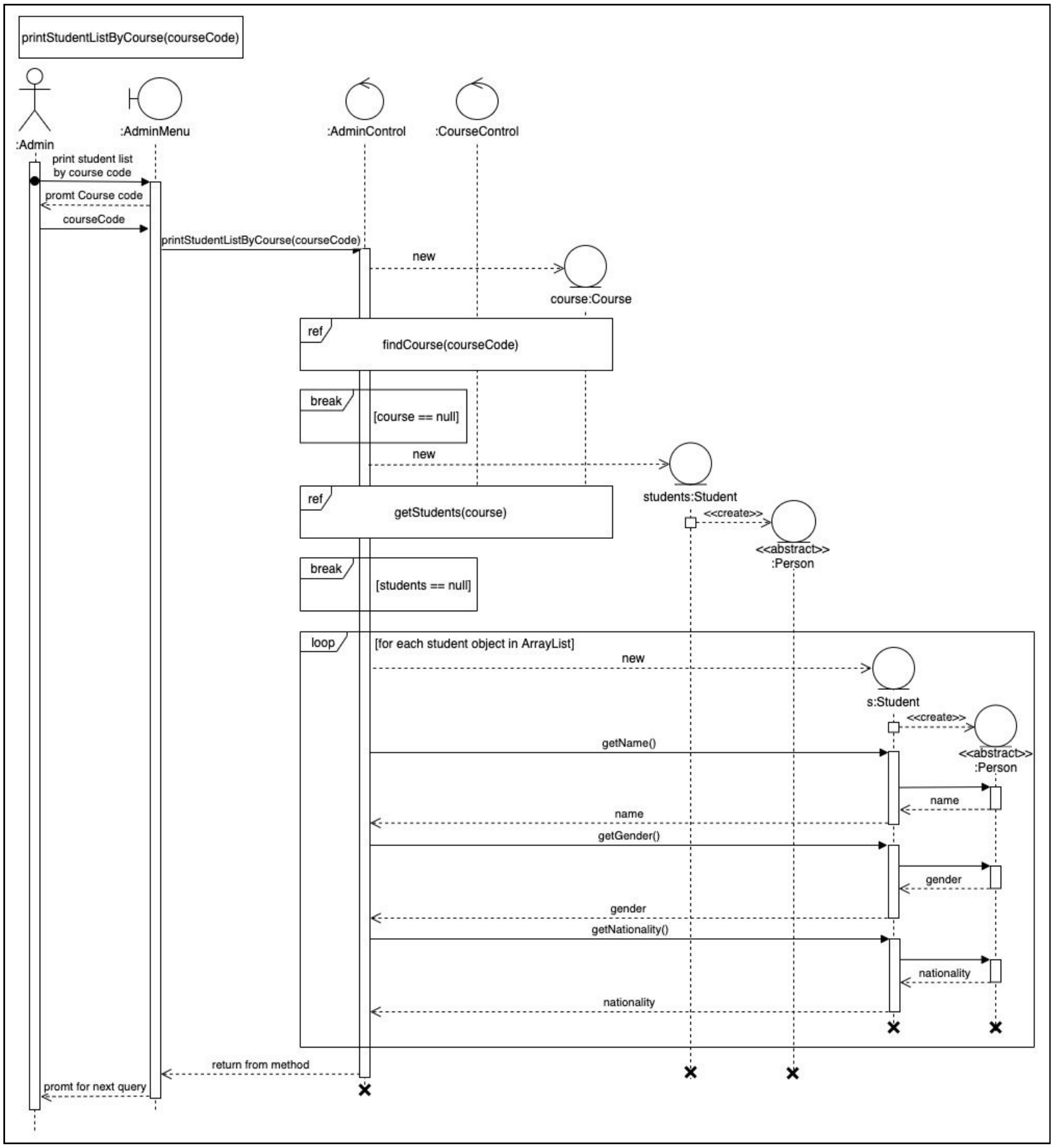
References:

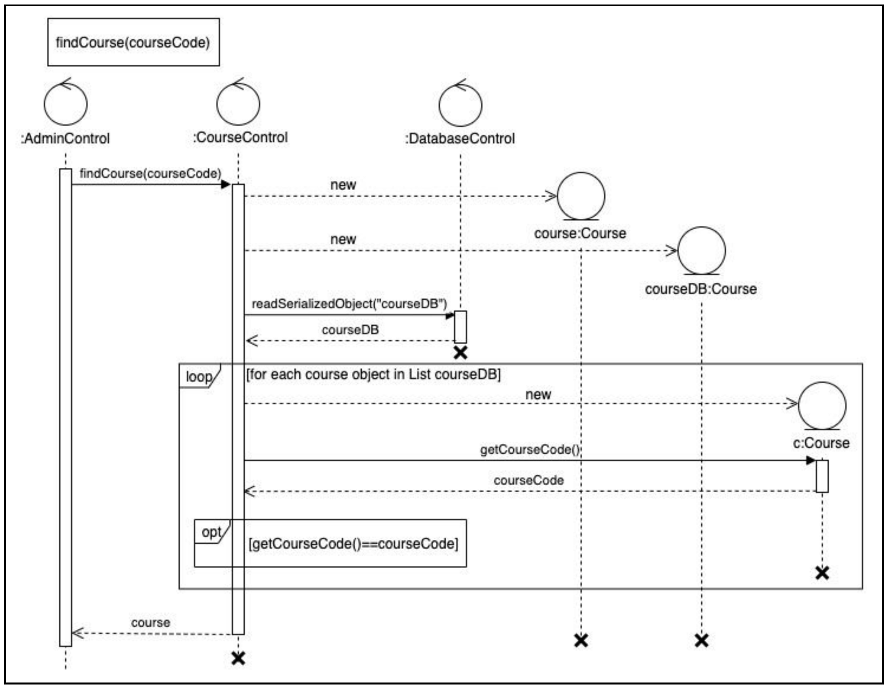
https://howtodoinjava.com/java/java-security/how-to-generate-secure-password-hash-md5-sha-pbkdf2-bcrypt-examples/

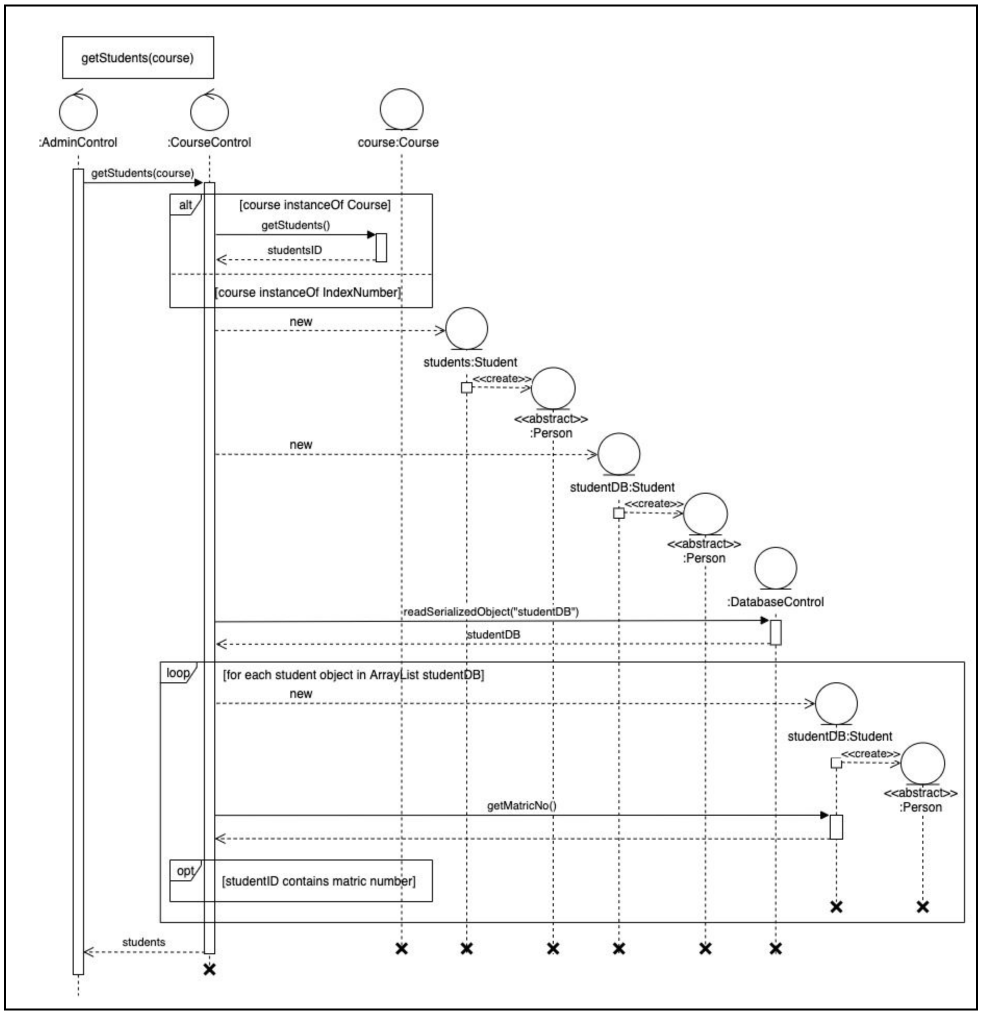
V. UML Class Diagram



VI. UML Sequence Diagram (main) & two references

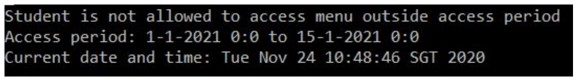
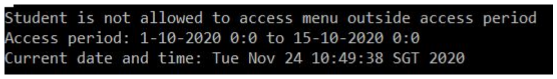
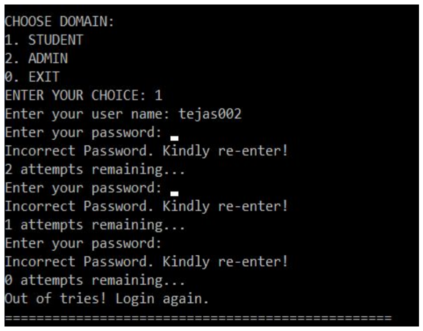



findCourse(courseCode)
:AdminControl
:CourseControl
:DatabaseControl
findCourse(courseCode)
new
course:Course
new
courseDB:Course
readSerializedObject("courseDB")
courseDB
loop
[for each course object in List courseDB]
new
c:Course
getCourseCode()
courseCode
opt
[getCourseCode()==courseCode]
course


getStudents(course)
:AdminControl
:CourseControl
course:Course
getStudents(course)
alt
[course instanceOf Course]
getStudents()
studentsID
[course instanceOf IndexNumber]
new
students:Student
<<create>>
<<abstract>>
:Person
new
studentDB:Student
<<create>>
<<abstract>>
:Person
:DatabaseControl
readSerializedObject("studentDB")
studentDB
loop
[for each student object in ArrayList studentDB]
new
studentDB:Student
<<create>>
<<abstract>>
:Person
getMatricNo()
opt
[studentID contains matric number]
students

VII. Test Cases

1. Student Login

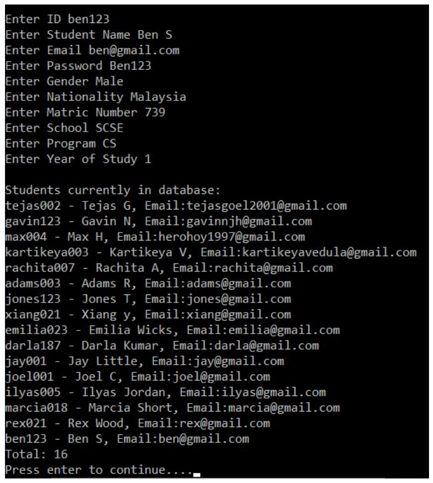
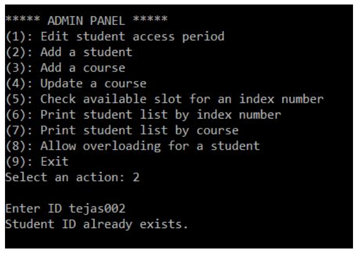
	Test Case	Outcome
a	Login before allowed period	
b	Login after allowed period (dates)	
c	Wrong password	

```
Student is not allowed to access menu outside access period
Access period: 1-1-2021 0:0 to 15-1-2021 0:0
Current date and time: Tue Nov 24 10:48:46 SGT 2020
```

```
Student is not allowed to access menu outside access period
Access period: 1-10-2020 0:0 to 15-10-2020 0:0
Current date and time: Tue Nov 24 10:49:38 SGT 2020
```

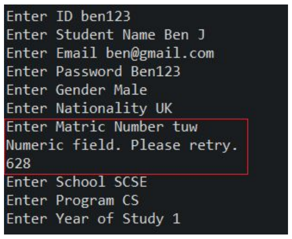
```
CHOOSE DOMAIN:
1. STUDENT
2. ADMIN
0. EXIT
ENTER YOUR CHOICE: 1
Enter your user name: tejas002
Enter your password:
Incorrect Password. Kindly re-enter!
2 attempts remaining...
Enter your password:
Incorrect Password. Kindly re-enter!
1 attempts remaining...
Enter your password:
Incorrect Password. Kindly re-enter!
0 attempts remaining...
Out of tries! Login again.
==================================================
```

2. Add a student

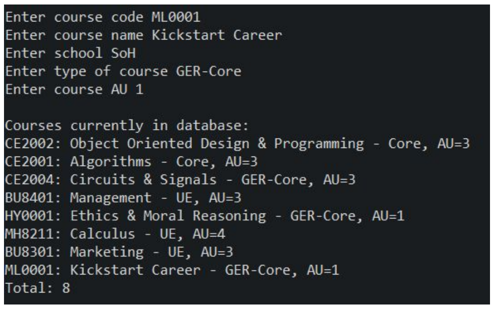
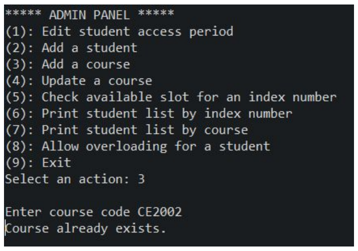
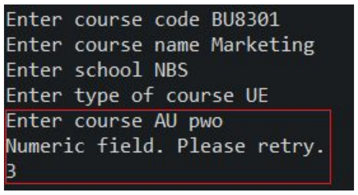
	Test Case	Outcome
a	Add a new student	
b	Add an existing student	

```
Enter ID ben123
Enter Student Name Ben S
Enter Email ben@gmail.com
Enter Password Ben123
Enter Gender Male
Enter Nationality Malaysia
Enter Matric Number 739
Enter School SCSE
Enter Program CS
Enter Year of Study 1

Students currently in database:
tejas002 - Tejas G, Email:tejasgoel2001@gmail.com
gavin123 - Gavin N, Email:gavinnjh@gmail.com
max004 - Max H, Email:herohoy1997@gmail.com
kartikeya003 - Kartikeya V, Email:kartikeyavedula@gmail.com
rachita007 - Rachita A, Email:rachita@gmail.com
adams003 - Adams R, Email:adams@gmail.com
jones123 - Jones T, Email:jones@gmail.com
xiang021 - Xiang y, Email:xiang@gmail.com
emilia023 - Emilia Wicks, Email:emilia@gmail.com
darla187 - Darla Kumar, Email:darla@gmail.com
jay001 - Jay Little, Email:jay@gmail.com
joel001 - Joel C, Email:joel@gmail.com
ilyas005 - Ilyas Jordan, Email:ilyas@gmail.com
marcia018 - Marcia Short, Email:marcia@gmail.com
rex021 - Rex Wood, Email:rex@gmail.com
ben123 - Ben S, Email:ben@gmail.com
Total: 16
Press enter to continue....
```

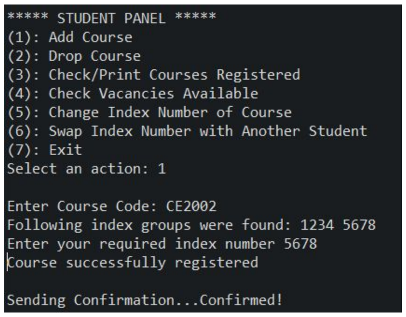
```
***** ADMIN PANEL *****
(1): Edit student access period
(2): Add a student
(3): Add a course
(4): Update a course
(5): Check available slot for an index number
(6): Print student list by index number
(7): Print student list by course
(8): Allow overloading for a student
(9): Exit
Select an action: 2

Enter ID tejas002
Student ID already exists.
```

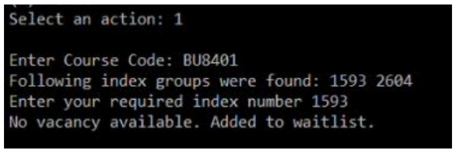
c	Invalid data entries	 Enter ID ben123 Enter Student Name Ben J Enter Email ben@gmail.com Enter Password Ben123 Enter Gender Male Enter Nationality UK Enter Matric Number tuw Numeric field. Please retry. 628 Enter School SCSE Enter Program CS Enter Year of Study 1

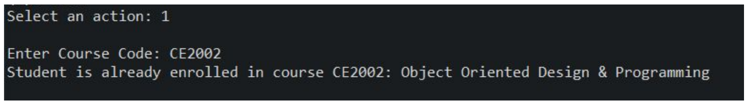
3. Add a course

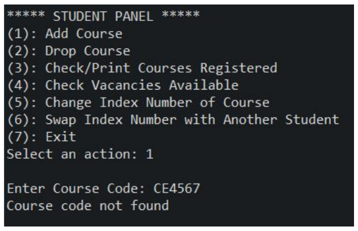
	Test Case	Outcome
a	Add a new course	 Enter course code ML0001 Enter course name Kickstart Career Enter school SoH Enter type of course GER-Core Enter course AU 1 Courses currently in database: CE2002: Object Oriented Design & Programming - Core, AU=3 CE2001: Algorithms - Core, AU=3 CE2004: Circuits & Signals - GER-Core, AU=3 BU8401: Management - UE, AU=3 HY0001: Ethics & Moral Reasoning - GER-Core, AU=1 MH8211: Calculus - UE, AU=4 BU8301: Marketing - UE, AU=3 ML0001: Kickstart Career - GER-Core, AU=1 Total: 8
b	Add an existing course	 ***** ADMIN PANEL ***** (1): Edit student access period (2): Add a student (3): Add a course (4): Update a course (5): Check available slot for an index number (6): Print student list by index number (7): Print student list by course (8): Allow overloading for a student (9): Exit Select an action: 3 Enter course code CE2002 Course already exists.
c	Invalid data entries	 Enter course code BU8301 Enter course name Marketing Enter school NBS Enter type of course UE Enter course AU pwo Numeric field. Please retry. 3

4. Register student for a course

	Test Case	Outcome
a	Add a student to a course index with available vacancies.	 ***** STUDENT PANEL ***** (1): Add Course (2): Drop Course (3): Check/Print Courses Registered (4): Check Vacancies Available (5): Change Index Number of Course (6): Swap Index Number with Another Student (7): Exit Select an action: 1 Enter Course Code: CE2002 Following index groups were found: 1234 5678 Enter your required index number 5678 Course successfully registered Sending Confirmation...Confirmed!

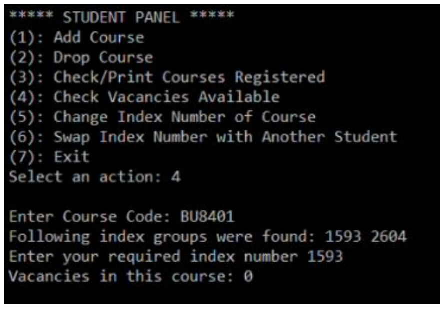
b	Add a student to a course index with 0 vacancies in Tut / Lab.	
		`Select an action: 1`
		
		`Enter Course Code: BU8401`
		`Following index groups were found: 1593 2604`
		`Enter your required index number 1593`
		`No vacancy available. Added to waitlist.`

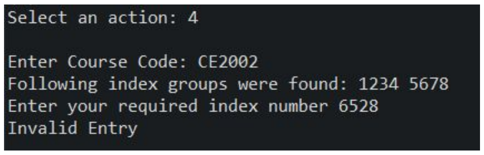
c	Register the same course again	
		`Select an action: 1`
		
		`Enter Course Code: CE2002`
		`Student is already enrolled in course CE2002: Object Oriented Design & Programming`

d	Invalid data entries (eg wrong student ID / course code, etc)	
		`***** STUDENT PANEL *****`
		`(1): Add Course`
		`(2): Drop Course`
		`(3): Check/Print Courses Registered`
		`(4): Check Vacancies Available`
		`(5): Change Index Number of Course`
		`(6): Swap Index Number with Another Student`
		`(7): Exit`
		`Select an action: 1`
		
		`Enter Course Code: CE4567`
		`Course code not found`

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

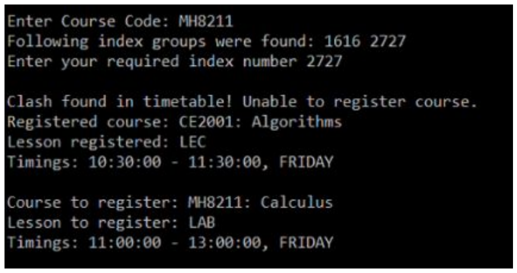
	Test Case	Outcome

a	Check for vacancy in course index	
		`***** STUDENT PANEL *****`
		`(1): Add Course`
		`(2): Drop Course`
		`(3): Check/Print Courses Registered`
		`(4): Check Vacancies Available`
		`(5): Change Index Number of Course`
		`(6): Swap Index Number with Another Student`
		`(7): Exit`
		`Select an action: 4`
		
		`Enter Course Code: BU8401`
		`Following index groups were found: 1593 2604`
		`Enter your required index number 1593`
		`Vacancies in this course: 0`

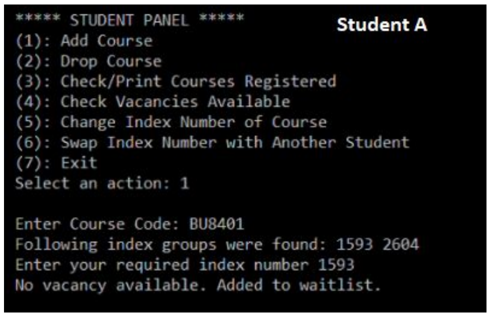
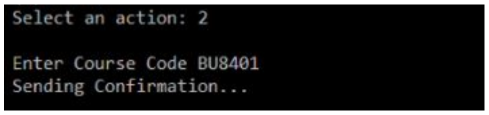
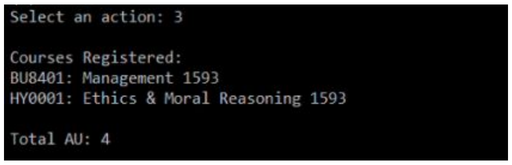
b	Invalid data entries (eg course code, class code etc)	
		`Select an action: 4`
		
		`Enter Course Code: CE2002`
		`Following index groups were found: 1234 5678`
		`Enter your required index number 6528`
		`Invalid Entry`

6. Day/Time clash with other course

	Test Case	Outcome

a	Add a student to a course index with available vacancies.	
		`Enter Course Code: MH8211`
		`Following index groups were found: 1616 2727`
		`Enter your required index number 2727`
		
		`Clash found in timetable! Unable to register course.`
		`Registered course: CE2001: Algorithms`
		`Lesson registered: LEC`
		`Timings: 10:30:00 - 11:30:00, FRIDAY`
		
		`Course to register: MH8211: Calculus`
		`Lesson to register: LAB`
		`Timings: 11:00:00 - 13:00:00, FRIDAY`

7. Waitlist notification

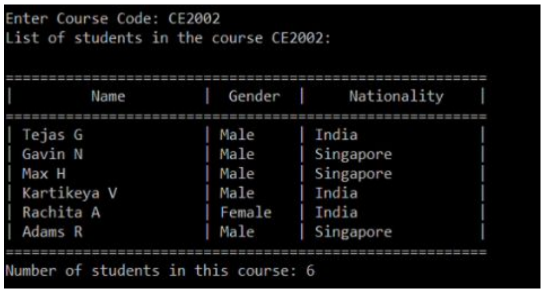
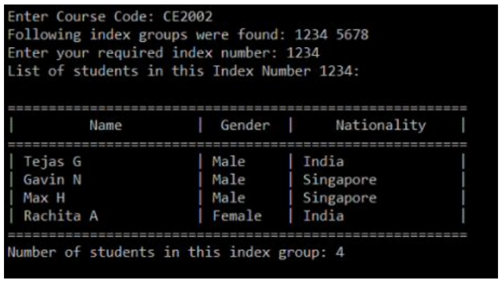
	Test Case	Outcome
a(i)	Add studentA to a course index with 0 vacancies	
a(ii)	Drop studentB from the same course index	
a(iii)	Display studentA timetable	

```
***** STUDENT PANEL *****                Student A
(1): Add Course
(2): Drop Course
(3): Check/Print Courses Registered
(4): Check Vacancies Available
(5): Change Index Number of Course
(6): Swap Index Number with Another Student
(7): Exit
Select an action: 1

Enter Course Code: BU8401
Following index groups were found: 1593 2604
Enter your required index number 1593
No vacancy available. Added to waitlist.
```

```
Select an action: 2

Enter Course Code BU8401
Sending Confirmation...
```

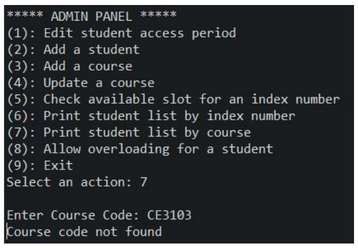
```
Select an action: 3

Courses Registered:
BU8401: Management 1593
HY0001: Ethics & Moral Reasoning 1593

Total AU: 4
```

8. Print student list by index number, course

	Test Case	Outcome
a	Print list by: (i)Course	
a	Print list by: (ii) Index	

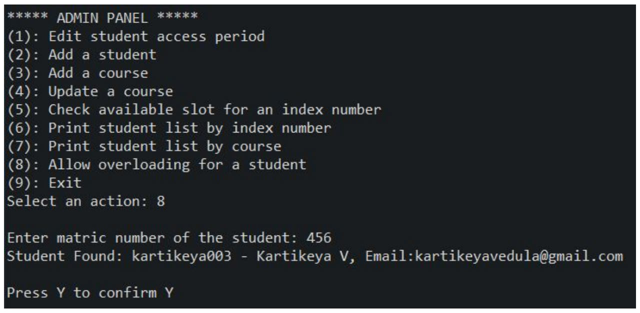
```
Enter Course Code: CE2002
List of students in the course CE2002:

=========================================================
|          Name          |  Gender  |    Nationality    |
=========================================================
| Tejas G                | Male     | India             |
| Gavin N                | Male     | Singapore         |
| Max H                  | Male     | Singapore         |
| Kartikeya V            | Male     | India             |
| Rachita A              | Female   | India             |
| Adams R                | Male     | Singapore         |
=========================================================
Number of students in this course: 6
```

```
Enter Course Code: CE2002
Following index groups were found: 1234 5678
Enter your required index number: 1234
List of students in this Index Number 1234:

=========================================================
|          Name          |  Gender  |    Nationality    |
=========================================================
| Tejas G                | Male     | India             |
| Gavin N                | Male     | Singapore         |
| Max H                  | Male     | Singapore         |
| Rachita A              | Female   | India             |
=========================================================
Number of students in this index group: 4
```

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

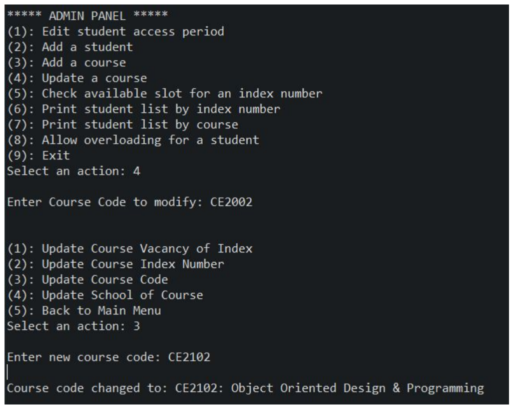
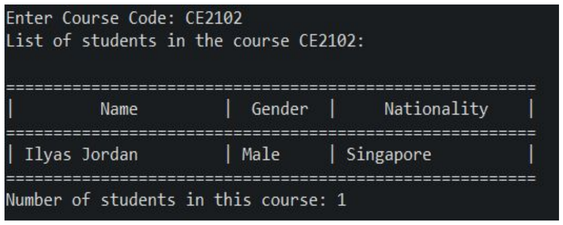
```
***** ADMIN PANEL *****
(1): Edit student access period
(2): Add a student
(3): Add a course
(4): Update a course
(5): Check available slot for an index number
(6): Print student list by index number
(7): Print student list by course
(8): Allow overloading for a student
(9): Exit
Select an action: 7

Enter Course Code: CE3103
Course code not found
```

9. Course Overload **(EXTRA)**

	Test Case	Expected Outcome	Outcome
a	Allow overloading of course	Course should be allocated to the student	

```
***** ADMIN PANEL *****
(1): Edit student access period
(2): Add a student
(3): Add a course
(4): Update a course
(5): Check available slot for an index number
(6): Print student list by index number
(7): Print student list by course
(8): Allow overloading for a student
(9): Exit
Select an action: 8

Enter matric number of the student: 456
Student Found: kartikeya003 - Kartikeya V, Email:kartikeyavedula@gmail.com

Press Y to confirm Y
```

10. Update Course **(EXTRA)**

	Test Case	Expected Outcome	Outcome
a	Update the course code of a course	The course code should be updated	
b	Print the student list in this course with the new course code	The same student students be printed in the updated course	

```
***** ADMIN PANEL *****
(1): Edit student access period
(2): Add a student
(3): Add a course
(4): Update a course
(5): Check available slot for an index number
(6): Print student list by index number
(7): Print student list by course
(8): Allow overloading for a student
(9): Exit
Select an action: 4

Enter Course Code to modify: CE2002


(1): Update Course Vacancy of Index
(2): Update Course Index Number
(3): Update Course Code
(4): Update School of Course
(5): Back to Main Menu
Select an action: 3

Enter new course code: CE2102

Course code changed to: CE2102: Object Oriented Design & Programming
```

```
Enter Course Code: CE2102
List of students in the course CE2102:

========================================================
|         Name         |  Gender  |    Nationality     |
========================================================
| Ilyas Jordan         | Male     | Singapore          |
========================================================
Number of students in this course: 1
```