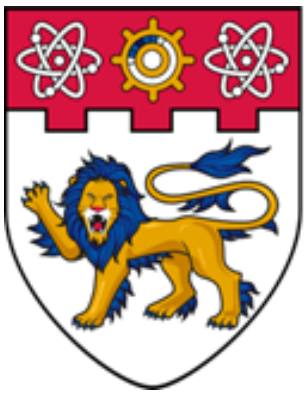


NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CZ2002 OODP Final Project

Lab Group: SS5

Group Members:

Agarwal Gopal

Garg Astha

Gunturi Kushal Sai

Tayal Aks

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
AGARWAL GOPAL	CZ 2002	SS5	21/10/20
GARG ASTHA	CZ2002	SS5	21/10/20
GUNTURI KUSHAL SAI	CZ2002	SS5	21/10/20
TAYAL AKS	CZ2002	SS5	21/10/20

Important notes:

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

Introduction

My STudent Automated Registration System (MySTARS) is a console-based application designed and developed for both staff and students 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 used to implement the application.

Approach Taken

Our main aim in designing the application was modularity, cohesiveness between the modules (with loose coupling between classes), and space for modification and extension. An application like this needs to allow for updates and changes. Therefore, we kept this goal in mind.

We divided it into three parts- UI, control and entities. Every part is hidden from others and only knows what is required for carrying out the necessary requirements.

Thus, the UI depends on the control classes which take care of different functionalities including the database handling, hence, hiding the data and preventing it from being directly accessed.

Design Considerations

Design Principles Used

Single Responsibility Principle (SRP)

The principle implies that every class should have only one responsibility. As the class would perform only one job, if there is any change in the implementation of that job, only then would the code have to be updated. Therefore, the code will be changed only for one reason. This ensures cohesiveness and makes error checking easier.

We have used this principle in assigning only one responsibility to each class. For example, one of the many manager classes - DataListManager has just one responsibility of handling all the requests from other classes when they want to retrieve data. Also, The FileHandler class has just one responsibility of interacting with the text files. The ValidationManager class also assumes responsibility of only validating certain inputs when entered by the User. Additionally, there are separate packages for handling UI, control and entities, with separate classes responsible for a set of related functions, thereby following this design principle.

Open-closed Principle (OCP)

"A model should be open for extension but closed for modification" defines this principle. In other words, we want to be able to change what a module does without changing the original source code.

For applying this principle, the User class has been made abstract, which contains the attributes common to all the users like username, password, gender, etc. and the corresponding getter and setter functions. Therefore, for introducing a new category of users, we can directly extend from this class and implement the functionality without making any changes to it, as it is done for Student and Staff.

Liskov Substitution Principle (LSP)

In simple words, the principle states that subtypes must be substituted for their base types.

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, all the classes implementing NotificationManager - SMSNotification Manager and EmailNotificationManager, can sendNotification as required, without the need for additional information from the calling method.

Interface Segregation Principle (ISP)

This principle states that classes should not depend on interfaces that they do not use. To follow this, we have avoided the usage of FAT interfaces. Instead, we have implemented many different interfaces for specific clients.

For e.g. we have separate UI for Login, Student, Staff. Only the appropriate menu is shown for specific users. The StudentUI itself has different methods for calling the appropriate function from the control classes for performing different tasks.

Dependency Injection Principle (DIP)

High level modules should not depend on low level modules rather both should depend upon abstraction. This allows the high level modules to be re-used quite simply. We have implemented this principle by making the NotificationManager an interface. Therefore, if new methods of notification have to be implemented, then a new class can be created which defines the sendNotification method according to the required mode. This allows the high level module to be reused simply.

The UML Class Diagram

The project is divided into three packages - UI, manager and entities.

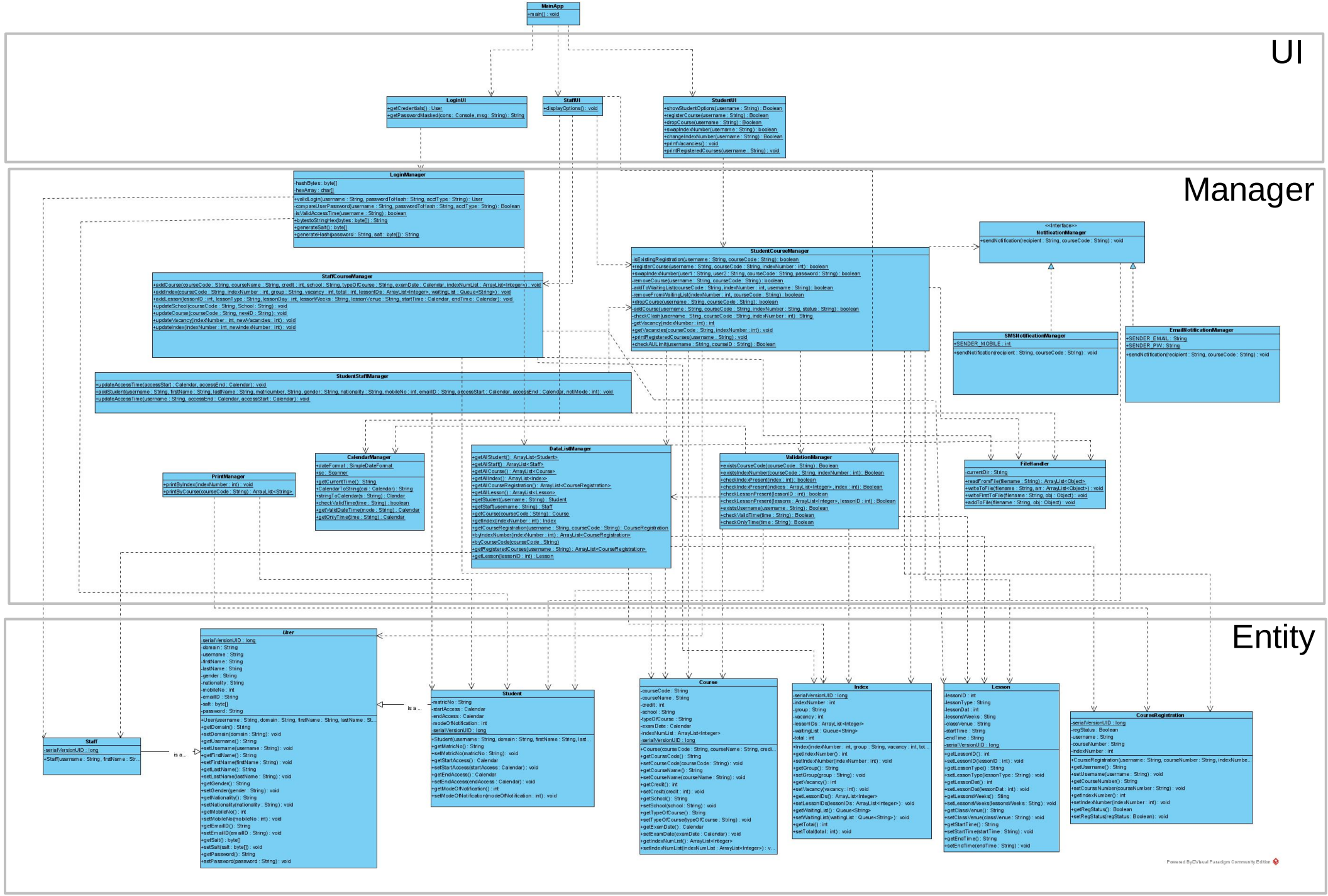
The UI package consists of classes that are concerned with interacting with the user, the different manager classes take care of the functionalities and make use of the entity classes.

There are three separate UI classes, for login, Student and Staff, which show the respective menu. These UI classes are dependent on the classes in the manager package.

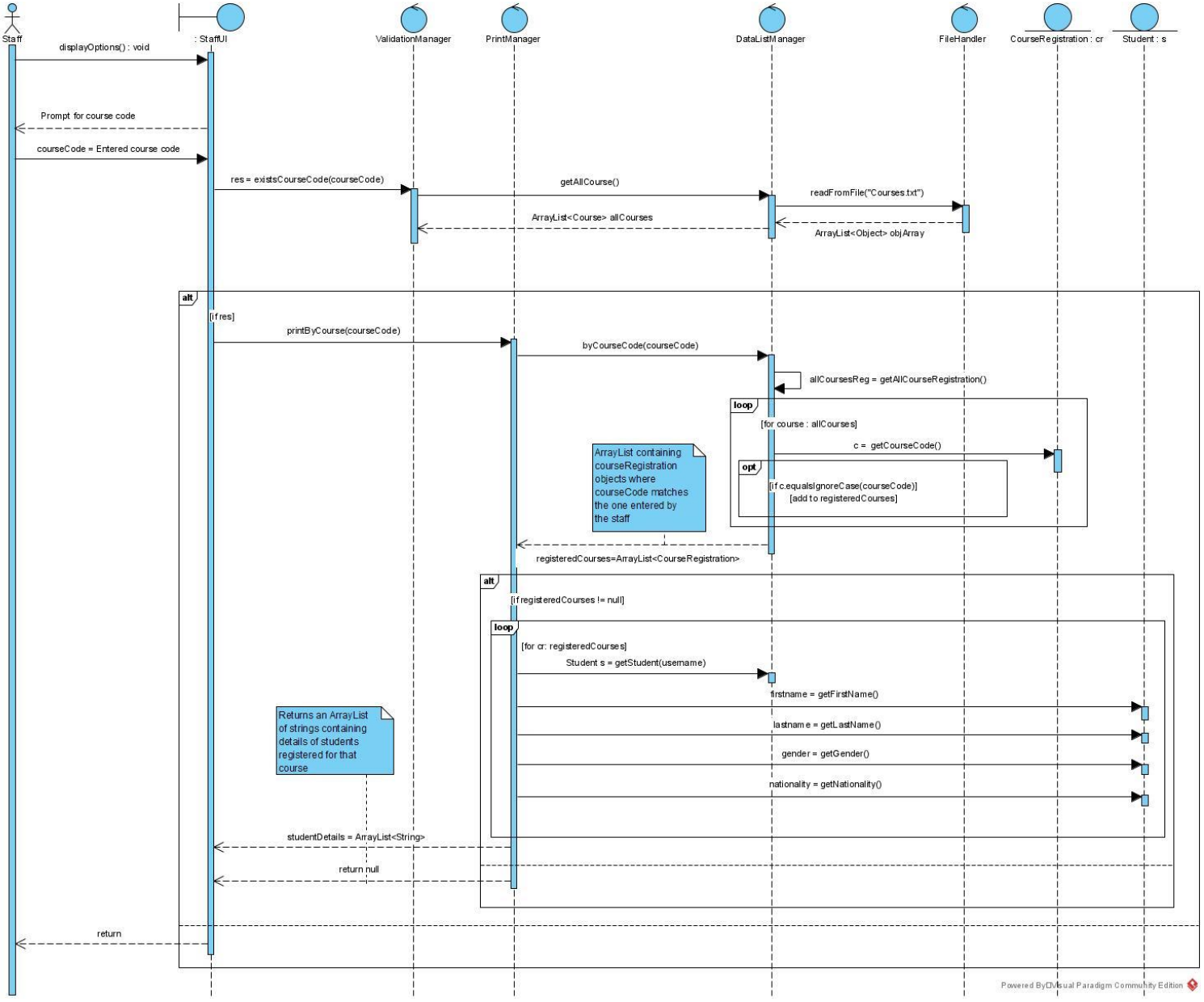
The different manager classes perform different functions for e.g.

- LoginManager handles logging in,
- the ValidationManager checks for the correctness of the input,
- FileHandler and DataListManager perform file I/O and return the appropriate information,
- CalendarManager handles all date related functions,
- StaffCourseManager is for the staff to do course related functions,
- StudentCourseManager handles registration and other course methods for students while StaffCourseManager does that for the Staff,
- StudentStaffManager performs the student functions for the staff,
- NotificationManager handles the sending of notification in the required format, after an update in the waitlist.

The User is an abstract class which is extended by Student and Staff. Other classes in this package include- Course, Index, Lesson, CourseRegistration (which is a record of all the students which registered for the course along with the indexNumber)


UI
Manager
Entity
MainApp
+main() : void
LoginUI
+getCredentials() : User
+getPasswordMasked(cons : Console, msg : String) : String
StaffUI
+displayOptions() : void
StudentUI
+showStudentOptions(username : String) : Boolean
+registerCourse(username : String) : Boolean
+dropCourse(username : String) : Boolean
+swapIndexNumber(username : String) : boolean
+changeIndexNumber(username : String) : Boolean
+printVacancies() : void
+printRegisteredCourses(username : String) : void
LoginManager
-hashBytes : byte[]
-hexArray : char[]
+validLogin(username : String, passwordToHash : String, acctType : String) : User
-compareUserPassword(username : String, passwordToHash : String, acctType : String) : Boolean
-isValidAccessTime(username : String) : boolean
+bytesToStringHex(bytes : byte[]) : String
+generateSalt() : byte[]
+generateHash(password : String, salt : byte[]) : String
<<Interface>>
NotificationManager
+sendNotification(recipient : String, courseCode : String) : void
StudentCourseManager
-isExistingRegistration(username : String, courseCode : String) : boolean
+registerCourse(username : String, courseCode : String, indexNumber : int) : boolean
+swapIndexNumber(user1 : String, user2 : String, courseCode : String, password : String) : boolean
-removeCourse(username : String, courseCode : String) : boolean
-addToWaitingList(courseCode : String, indexNumber : int, username : String) : boolean
-removeFromWaitingList(indexNumber : int, courseCode : String) : boolean
+dropCourse(username : String, courseCode : String) : boolean
-addCourse(username : String, courseCode : String, indexNumber : String, status : String) : boolean
-checkClash(username : String, courseCode : String, indexNumber : int) : String
-getVacancy(indexNumber : int) : int
+getVacancies(courseCode : String, indexNumber : int) : void
+printRegisteredCourses(username : String) : void
+checkALLimit(username : String, courseID : String) : Boolean
StaffCourseManager
+addCourse(courseCode : String, courseName : String, credit : int, school : String, typeOfCourse : String, examDate : Calendar, indexNumList : ArrayList<Integer>) : void
+addIndex(courseCode : String, indexNumber : int, group : String, vacancy : int, total : int, lessonIDs : ArrayList<Integer>, waitingList : Queue<String>) : void
+addLesson(lessonID : int, lessonType : String, lessonDay : int, lessonWeeks : String, lessonVenue : String, startTime : Calendar, endTime : Calendar) : void
+updateSchool(courseCode : String, School : String) : void
+updateCourse(courseCode : String, newID : String) : void
+updateVacancy(indexNumber : int, newVacancies : int) : void
+updateIndex(indexNumber : int, newIndexNumber : int) : void
SMSNotificationManager
+SENDER_MOBILE : int
+sendNotification(recipient : String, courseCode : String) : void
EmailNotificationManager
+SENDER_EMAIL : String
+SENDER_PW : String
+sendNotification(recipient : String, courseCode : String) : void
StudentStaffManager
+updateAccessTime(accessStart : Calendar, accessEnd : Calendar) : void
+addStudent(username : String, firstName : String, lastName : String, matricumber, String, gender : String, nationality : String, mobileNo : int, emailID : String, accessStart : Calendar, accessEnd : Calendar, notiMode : int) : void
+updateAccessTime(username : String, accessEnd : Calendar, accessStart : Calendar) : void
PrintManager
+printByIndex(indexNumber : int) : void
+printByCourse(courseCode : String) : ArrayList<String>
CalendarManager
+dateFormat : SimpleDateFormat
+sc : Scanner
+getCurrentTime() : String
+CalendarToString(cal : Calendar) : String
+stringToCalendar(s : String) : Clendar
+checkValidTime(time : String) : boolean
+getValidDateTime(mode : String) : Calendar
+getOnlyTime(time : String) : Calendar
DataListManager
+getAllStudent() : ArrayList<Student>
+getAllStaff() : ArrayList<Staff>
+getAllCourse() : ArrayList<Course>
+getAllIndex() : ArrayList<Index>
+getAllCourseRegistration() : ArrayList<CourseRegistration>
+getAllLesson() : ArrayList<Lesson>
+getStudent(username : String) : Student
+getStaff(username : String) : Staff
+getCourse(courseCode : String) : Course
+getIndex(indexNumber : int) : Index
+getCourseRegistration(username : String, courseCode : String) : CourseRegistration
+byIndexNumber(indexNumber : int) : ArrayList<CourseRegistration>
+byCourseCode(courseCode : String)
+getRegisteredCourses(username : String) : ArrayList<CourseRegistration>
+getLesson(lessonID : int) : Lesson
ValidationManager
+existsCourseCode(courseCode : String) : Boolean
+existsIndexNumber(courseCode : String, indexNumber : int) : Boolean
+checkIndexPresent(index : int) : boolean
+checkIndexPresent(indices : ArrayList<Integer>, index : int) : Boolean
+checkLessonPresent(lessonID : int) : boolean
+checkLessonPresent(lessons : ArrayList<Integer>, lessonID : int) : Boolean
+existsUsername(username : String) : Boolean
+checkValidTime(time : String) : Boolean
+checkOnlyTime(time : String) : Boolean
FileHandler
-currentDir : String
+readFromFile(filename : String) : ArrayList<Object>
+writeToFile(filename : String, arr : ArrayList<Object>) : void
+writeFirstToFile(filename : String, obj : Object) : void
+addToFile(filename : String, obj : Object) : void
User
-serialVersionUID : long
-domain : String
-username : String
-firstName : String
-lastName : String
-gender : String
-nationality : String
-mobileNo : int
-emailID : String
-salt : byte[]
-password : String
+User(username : String, domain : String, firstName : String, lastName : St...
+getDomain() : String
+setDomain(domain : String) : void
+getUsername() : String
+setUsername(username : String) : void
+getFirstName() : String
+setFirstName(firstName : String) : void
+getLastName() : String
+setLastName(lastName : String) : void
+getGender() : String
+setGender(gender : String) : void
+getNationality() : String
+setNationality(nationality : String) : void
+getMobileNo() : int
+setMobileNo(mobileNo : int) : void
+getEmailID() : String
+setEmailID(emailID : String) : void
+getSalt() : byte[]
+setSalt(salt : byte[]) : void
+getPassword() : String
+setPassword(password : String) : void
is a ...
Staff
-serialVersionUID : long
+Staff(username : String, firstName : Str...
is a...
Student
-matricNo : String
-startAccess : Calendar
-endAccess : Calendar
-modeOfNotification : int
-serialVersionUID : long
+Student(username : String, domain : String, firstName : String, last...
+getMatricNo() : String
+setMatricNo(matricNo : String) : void
+getStartAccess() : Calendar
+setStartAccess(startAccess : Calendar) : void
+getEndAccess() : Calendar
+setEndAccess(endAccess : Calendar) : void
+getModeOfNotification() : int
+setModeOfNotification(modeOfNotification : int) : void
Course
-courseCode : String
-courseName : String
-credit : int
-school : String
-typeOfCourse : String
-examDate : Calendar
-indexNumList : ArrayList<Integer>
-serialVersionUID : long
+Course(courseCode : String, courseName : String, credi...
+getCourseCode() : String
+setCourseCode(courseCode : String) : void
+getCourseName() : String
+setCourseName(courseName : String) : void
+getCredit() : int
+setCredit(credit : int) : void
+getSchool() : String
+setSchool(school : String) : void
+getTypeOfCourse() : String
+setTypeOfCourse(typeOfCourse : String) : void
+getExamDate() : Calendar
+setExamDate(examDate : Calendar) : void
+getIndexNumList() : ArrayList<Integer>
+setIndexNumList(indexNumList : ArrayList<Integer>) : v...
Index
-serialVersionUID : long
-indexNumber : int
-group : String
-vacancy : int
-lessonIDs : ArrayList<Integer>
-waitingList : Queue<String>
-total : int
+Index(indexNumber : int, group : String, vacancy : int, tot...
+getIndexNumber() : int
+setIndexNumber(indexNumber : int) : void
+getGroup() : String
+setGroup(group : String) : void
+getVacancy() : int
+setVacancy(vacancy : int) : void
+getLessonIDs() : ArrayList<Integer>
+setLessonIDs(lessonIDs : ArrayList<Integer>) : void
+getWaitingList() : Queue<String>
+setWaitingList(waitingList : Queue<String>) : void
+getTotal() : int
+setTotal(total : int) : void
Lesson
-lessonID : int
-lessonType : String
-lessonDat : int
-lessonsWeeks : Sting
-classVenue : String
-startTime : String
-endTime : String
-serialVersionUID : long
+getLessonID() : int
+setLessonID(lessonID : int) : void
+getLessonType() : String
+setLessonType(lessonType : String) : void
+getLessonDat() : int
+setLessonDat(lessonDat : int) : void
+getLessonsWeeks() : Sting
+setLessonsWeeks(lessonsWeeks : Sting) : void
+getClassVenue() : String
+setClassVenue(classVenue : String) : void
+getStartTime() : String
+setStartTime(startTime : String) : void
+getEndTime() : String
+setEndTime(endTime : String) : void
CourseRegistration
-serialVersionUID : long
-regStatus : Boolean
-username : String
-courseNumber : String
-indexNumber : int
+CourseRegistration(username : String, courseNumber : String, indexNumbe...
+getUsername() : String
+setUsername(username : String) : void
+getCourseNumber() : String
+setCourseNumber(courseNumber : String) : void
+getIndexNumber() : int
+setIndexNumber(indexNumber : int) : void
+getRegStatus() : Boolean
+setRegStatus(regStatus : Boolean) : void
Powered By:Visual Paradigm Community Edition

UML Sequence Diagram



Flow of the diagram

After logging in as a staff, the StaffUI displays the menu. Upon selecting the option for printing the student list by courseCode, the user is asked to enter the courseCode.

The courseCode is passed a parameter to the function existsCourseCode() in ValidationManager to check if this courseCode is valid. Inside this function, getAllCourse() is called from the DataListManager in order to retrieve the ArrayList or Course type objects called allCourses. The getAllCourse() function calls the readFromFile() function from the FileHandler class which returns to it an ArrayList of Object type objects. After explicitly converting them to Course type objects it is returned to the Validation Manager which loops through it to find out if the courseCode is present as the course code of any of the objects in allCourses. This function returns a boolean value which is true if the courseCode is present and false otherwise. If the received res(boolean) is true, the printByCourse function from the PrintManager is invoked and the courseCode is passed as the parameter. This function first calls byCourseCode() from the DataListManager which in turn calls getAllCourseRegistration() from its own class to retrieve all records of registered courses. We then loop over the retrieved ArrayList of courseRegistration type objects and append those objects which have the course code same as the one entered by the user. The new ArrayList to which we appended all these objects is then returned to the PrintManager. If this ArrayList called registeredCourses is not null, we loop through it and for each CourseRegistration type object CR, we get the Student S whose username matches the one in the CR. We then retrieve its first name, last name, gender and nationality from the respective getter function in the Student class. All these values are concatenated into a String and an ArrayList of String is finally returned to the StaffUI for displaying. A null ArrayList is returned in case registeredCourses is null. In the case that the courseCode does not exist, meaning that res(boolean) is false, an appropriate error message is displayed, and Staff is once again presented the menu.

Object-oriented Concepts (Explanation of the UML Diagram)

Encapsulation

This refers to binding the data and functions so as to protect them from outside. This has been used everywhere in the design. The only way two classes can interact is by the way of calling the member methods. The method performs the function and returns the result. This way, the internal functioning is hidden from the class which called the method ensuring data protection.

Abstraction

The main goal of this concept is to hide unnecessary details from the user and show only what is required. For e.g., when a student chooses to register for a course, the UI class calls the function from the StudentCourseManager without knowing how exactly it will be done. The StudentCourseManager in turn calls the DataListManager to obtain data from the files. The data control class just returns the required information. This way, the StudentCourseManager does not need to know how the actual reading of the file is performed as it is not concerned with it.

Inheritance

This allows one class to inherit properties from another class while defining its own attributes or methods. This was used in implementing Student and Staff classes that extend the User class. Since both the classes have common attributes and functions like username, password, gender etc, having a common parent class prevents duplication of these attributes and methods. Furthermore, introducing a new user is also simple as the common features can be inherited from User.

Polymorphism

This concept allows overloading of methods i.e. two methods with the same name but different parameters will behave differently. In our case, we have implemented this at several places, one of them being - updateAccessTime(String, Calendar, Calendar) and updateAccessTime(Calendar, Calendar). In the first case the access time of a particular student is changed and hence the String parameter of username is required. On the other hand, the latter function changes access time for all students present in the database and no username is needed.

Data Structure

Firstly, for implementing File I/O, we have used Serialization. This way we can easily save entire objects with all the attributes and methods. Hence, on updating something specific we can just save the updated object itself instead of reading the specific information and replacing it. There are separate text files (.txt) for every entity.

Furthermore, a queue has been used to implement the waiting list, so new students can be added from the back end and the topmost student can be removed easily to maintain the first-come-first-serve rule.

Assumptions

- The student is not given the option to edit any of his/her details as the application is for the sole purpose of registration of courses.
- The course code, index number and lesson ID are unique across all courses.

- For swapping the index number for a course with another student, the password of the other student is needed.
- When adding a student, the default password is the username which is case sensitive.
- Throughout the application, the username is not case sensitive.
- AU limit has been set to 10 for all students.
- Staff cannot be added from the application.

Demo and Test Cases

The youtube link for the video - https://www.youtube.com/watch?v=RjbVyXoEavQ&feature=youtu.be

The admin already existing has username - ADMIN1, and password ADMIN1

A README text file has been added which specifies the database information.

Test Case 1

1. **Student Login**

a. Login beyond access time

```
----------Welcome to STARS----------
Enter choice: (1)Login  (2)Quit:  1
Enter domain: (1) for Student, (2) for Staff:  1
Enter choice: (1)Enter details  (2)Back:  1
Enter username: JAS001
Enter password:
Not allowed access at this time!
Enter choice: (1)Enter details  (2)Back:
```

b. Incorrect password

```
Enter choice: (1)Enter details  (2)Back:  1
Enter username: AKS001
Enter password:
Incorrect password!
Enter choice: (1)Enter details  (2)Back:  1
```

c. Successful login

```
Enter choice: (1)Enter details  (2)Back:  1
Enter username: AKS001
Enter password:
Successfully logged in - Time valid!
```

2. Add a student

a. Wrong date format in input

```
----------Welcome to Staff panel----------
Choose from the following:
1. Edit student access period for all students
2. Edit student access period for a particular student
3. Add a student
4. Add a course
5. Update a course
6. Check availibility for index in a course
7. Print student list by index number
8. Print student list by course
9. Logout
Select your option: 3
Enter choice: (1)Enter details  (2)Back:  1
Enter student's username: SAMAY001
Enter student's firstname:  SAMAY
Enter student's lastname:  panwar
Enter student's matric number:  u19228567
Enter student's gender:  male
Enter student's nationality:  indian
Enter student's email:  s@gmail.com
Enter student's mobileNo:  91668452
Enter new access start time in "DD/MM/YYYY HH:MM" format:  22-1-2020 22:67
Invalid time, try again!
Enter choice: (1)Enter details  (2)Back:
```

c. Adding a student

```
Enter choice: (1)Enter details  (2)Back:  1
Enter student's username: SAMAY001
Enter student's firstname:  samay
Enter student's lastname:  panwar
Enter student's matric number:  u18364682f
Enter student's gender:  male
Enter student's nationality:  idnian
Enter student's email:  s@gmail.com
Enter student's mobileNo:  9374824
Enter new access start time in "DD/MM/YYYY HH:MM" format:  26/11/2020 11:00
Enter new access end time in "DD/MM/YYYY HH:MM" format:  26/11/2020 12:00
Enter Mode of Notificaiton: 1
```

```
Students in system:
AKS001- aks tayal
GOPAL001- Gopal Agarwal
ASTHA001- Astha Garg
GUNT001- CUTIE GUNTURI
JAS001- jas singh
James001- james xyz
MHD001- MD abdul
jamie001- jamie wen wong
kark001- kirk abcd
appu001- appu pappu
viper001- viper viperrr
mumma001- mumma agarwal
papa001- papa tayal
didi001- didi agarwal
bhaiya001- bhaiya abcd
tina001- tina mina
chirag001- Chirag Gupta
SAMAY001- samay panwar
Successfully updated!
```

c. Adding an existing student

```
Select your option: 3
Enter choice: (1)Enter details  (2)Back:  1
Enter student's username: samay001
Invalid Username, try again!
```

3. Add a course

a. Invalid data entries

```
Select your option: 4
Enter choice: (1)Enter details  (2)Back:  1
Enter new course code:  CZ2004
Enter course name:  HCI
Enter course credit:  3
Enter course school:  SCSE
Enter course type:  core
Enter course exam date in "DD/MM/YYYY HH:MM" format:  22-1-2020 25:67
Invalid time, try again!
```

b. Adding an existing course

```
Select your option: 4
Enter choice: (1)Enter details  (2)Back:  1
Enter new course code:  CZ2001
Invalid courseCode, try again!
Enter choice: (1)Enter details  (2)Back:  2
```

c. Adding course correctly

```
Enter choice: (1)Enter details  (2)Back:  1
Enter new course code:  CZ2004
Enter course name:  HCI
Enter course credit:  3
Enter course school:  SCSE
Enter course type:  core
Enter course exam date in "DD/MM/YYYY HH:MM" format:  22/01/2021 13:00
Enter choice: (1)Add index      (2)Back:  1
Enter a new index number for CZ2004 :  90000
Enter a group for 90000 :  SR1
Enter number of vacancies in 90000 :  13
Enter total seats:  13
Enter choice: (1)Add lesson     (2)Back:  1
Enter LessonID:  99
Enter lesson type:  lec
Enter Day of this lesson:  2
Enter lesson weeks:  all
Enter lesson Venue:  LT5
Enter lesson startTime in "HH:MM" format:  10:00
Enter lesson endTime in "HH:MM" format:  12:00
Enter choice: (1)Add lesson     (2)Back:  2
Enter choice: (1)Add index      (2)Back:  2
Courses in system:
CZ2001 - Algo
CZ2002 - OODP
HE9091 - ECONS
MH3300 - Calc-III
CZ2005 - OS
CZ2003 - CGV
CZ2004 - HCI
```

4. Register student for a course

a. Clash in timings

```
----------Welcome to Student panel-------------
1. Register Course
2. Drop Course
3. Check/Print Courses Registered
4. Check Vacancies Available
5. Change Index Number of Course
6. Swop Index Number with Another Student
7. Logout
Select your option: 1
Enter Course ID to register: CZ2001
Enter Index Number: 10001
Red! Timings clash with another course
```

b. Successful registration

```
Enter Course ID to register: CZ2005
Enter Index Number: 12001
-----Timetable-----

Course code: HE9091      REGISTERED
Index: 30001
Lessons :-
Type-Lec   Day-Monday   Weeks-all   Venue-LT2   Start time-09:00   End time-11:00
Type-Tut   Day-Wednesday   Weeks-even   Venue-TR12   Start time-12:00   End time-13:00

Course code: CZ2005      REGISTERED
Index: 12001
Lessons :-
Type-lec   Day-Thursday   Weeks-all   Venue-LT1A   Start time-12:00   End time-15:00
Successfully added
```

c. With 0 vacancy

```
Select your option: 4
Enter Course ID check vacancies: CZ2002
Enter Index Number: 11001
0/3 seats are available
```

Timetable

```
Select your option: 1
Enter Course ID to register: CZ2002
Enter Index Number: 11001
You have been put on the waiting list
-----Timetable-----

Course code: HE9091      REGISTERED
Index: 30001
Lessons :-
Type-Lec   Day-Monday   Weeks-all   Venue-LT2   Start time-09:00   End time-11:00
Type-Tut   Day-Wednesday   Weeks-even   Venue-TR12   Start time-12:00   End time-13:00

Course code: CZ2005      REGISTERED
Index: 12001
Lessons :-
Type-lec   Day-Thursday   Weeks-all   Venue-LT1A   Start time-12:00   End time-15:00

Course code: CZ2002      WAITING LIST
Index: 11001
Lessons :-
Type-Lec   Day-Friday   Weeks-all   Venue-LT2   Start time-10:00   End time-11:00
Type-Tut   Day-Wednesday   Weeks-all   Venue-TR12   Start time-13:00   End time-15:00
```

d. Registering existing course

```
Select your option: 1
Enter Course ID to register: HE9091
Enter Index Number: 30001
Error! Course is already registered
```

e. Invalid data entry

```
Select your option: 1
Enter Course ID to register: CZ1000
Enter Index Number: 1301
Wrong input!Index number does not exist
```

5. Check available slot in a class

a. Output as vacancy/total

```
Select your option: 4
Enter Course ID check vacancies: He9091
Enter Index Number: 30001
1/4 seats are available
```

b. Wrong data entry

```
Select your option: 1
Enter Course ID to register: CZ1000
Enter Index Number: 1301
Wrong input!Index number does not exist
```

6. Date/Time clash for course registration - Demonstrated in 4 a.

7. Waitlist Notification

a. Add to a course with 0 vacancy to studentA

```
Select your option: 4
Enter Course ID check vacancies: CZ2002
Enter Index Number: 11001
0/3 seats are available
```

```
Select your option: 3
Course   Index    Registration Status
---------------------------------------
HE9091   30001   Registered
CZ2005   12001   Registered
CZ2002   11001   Waiting list
```

b. Dropping course from another studentB

```
Select your option: 2
Enter Course ID to drop: CZ2002
Successfully dropped course
```

c. Timetable for studentA

```
Select your option: 3
Course   Index    Registration Status
---------------------------------------
HE9091   30001   Registered
CZ2005   12001   Registered
CZ2002   11001   Registered
```

d. E-mail notification

8. Print student list by index number, course code

a. Printing by indexNumber

```
Select your option: 7
Enter choice: (1)Enter details  (2)Back:  1
Enter courseCode:  CZ2002
Enter index:  11001
Students who have taken Index Number 11001
Name: aks tayal       Gender: male       Nationality: indian       Status: Registered
Name: Gopal Agarwal       Gender: male       Nationality: indian       Status: Registered
Name: CUTIE GUNTURI       Gender: OTHER       Nationality: Singaporean       Status: Registered
Enter choice: (1)Enter details  (2)Back:
```

b. Printing by courseCode

```
Select your option: 8
Enter choice: (1)Enter details  (2)Back:  1
Enter courseCode:  CZ2002
Students who have taken Course Number CZ2002 :
Name: aks tayal       Gender: male       Nationality: indian       Status: Registered
Name: Gopal Agarwal       Gender: male       Nationality: indian       Status: Registered
Name: CUTIE GUNTURI       Gender: OTHER       Nationality: Singaporean       Status: Registered
Enter choice: (1)Enter details  (2)Back:  2
```