My STudent Automated Registration System (MySTARS)

**SCHOOL OF COMPUTER SCIENCE AND ENGINEERING**

**NANYANG TECHNOLOGICAL UNIVERSITY**

CZ 2002 Object-Oriented Design & Programming

Assignment Report

Date: 25nd November 2020

Declaration of Original Work for CE/CZ2002 Assignment

We hereby declare that the attached group assignment has been researched, undertaken, completed and submitted as a collective effort by the group members listed below.

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

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

|  |  |  |  |
| --- | --- | --- | --- |
| Name | Course | Lab Group | Signature/Date |
| BUI THUY TIEN | CZ2002 |  |  |
| CHIAM BON YEAK | CZ2002 |  |  |
| FRANCIS NAI JIAN XIONG | CZ2002 |  |  |
| LEE YIK SIANG | CZ2002 |  |  |
| MO JUN YI | CZ2002 |  |  |

TABLE OF CONTENT

1. Introduction

1.1 Objective

1.2 Scope

1.3 Functional Requirements

2. Design Considerations

3. Details Design

3.1 UML Class Diagram

3.2 UML Sequence Diagram

4. Testing

5. Conclusion

1. Introduction

The assignment serves as a final wrap up to implement what we have learned in the entire semester using Java programming language. At the beginning of the course, we have learned the java coding methodologies and later, the UML class and sequence diagram in the manner of object orientated programming (OOP). The assignment allows us to use these skills to implement a student automatic registration system in OOP ways.

* 1. Objective

The main objectives of this assignment are

• to apply the Object-Oriented (OO) concepts learnt in the course,

• to model, design and develop an OO application.

• to gain familiarity with using Java as an object oriented programming language.

• to work collaboratively as a group to achieve a common goal.

1.2 Scope

The scope of this assignment is to implement a student automatic registration system (MySTARS) to be used by the school’s academic staff and undergraduate students. The application allows the creation of courses and adding of student records as well as registration of courses and students. There will be an administrator mode for academic staff and user mode for students. At the start of each semester registration period, students will be required to register for their courses. Each course (subject) will have its course code, its corresponding index number information, the class schedules and venue, and available vacancy. Courses may have lectures only, lectures and tutorial only or lectures, tutorial and laboratory sessions.

Students on waitlist will be placed in a queue whereby when there is available slot, the first in queue will be allocated the slot. A notification will be sent to the student.

1.3 Functional requirements

There will be 3 main parts for the student automatic registration system (MySTARS):

* + 1. Login/ Logout page
    2. Student page (only accessible during the registration period)
    3. Admin page (for administration purposes)

On Login/Logout page, the student will be brought to a Student/Admin Menu upon successfully login.

Student is the user of the Student page and the student will be able to add course, change course…etc. In general, a student shall be able to perform the following tasks in the Student page:

1. Add 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

The administrator is the controller of the admin page, he will register the student, edit in the access period, update course, etc. In general, the administrator shall be able to perform the following tasks in the admin page:

1. Edit student access period
2. Add a student (name, matric number, gender, nationality, etc.)
3. Add/Update a course (course code, school, its index numbers and vacancy)
4. Check available slot for an index number (vacancy in a class)
5. Print student list by index number
6. Print student list by course (all students registered for the selected course)
7. DESIGN CONSIDERATION

This part presents the design considerations and the assumptions that we have made to implements the MySTARS.

We have considered the followings in our software design:

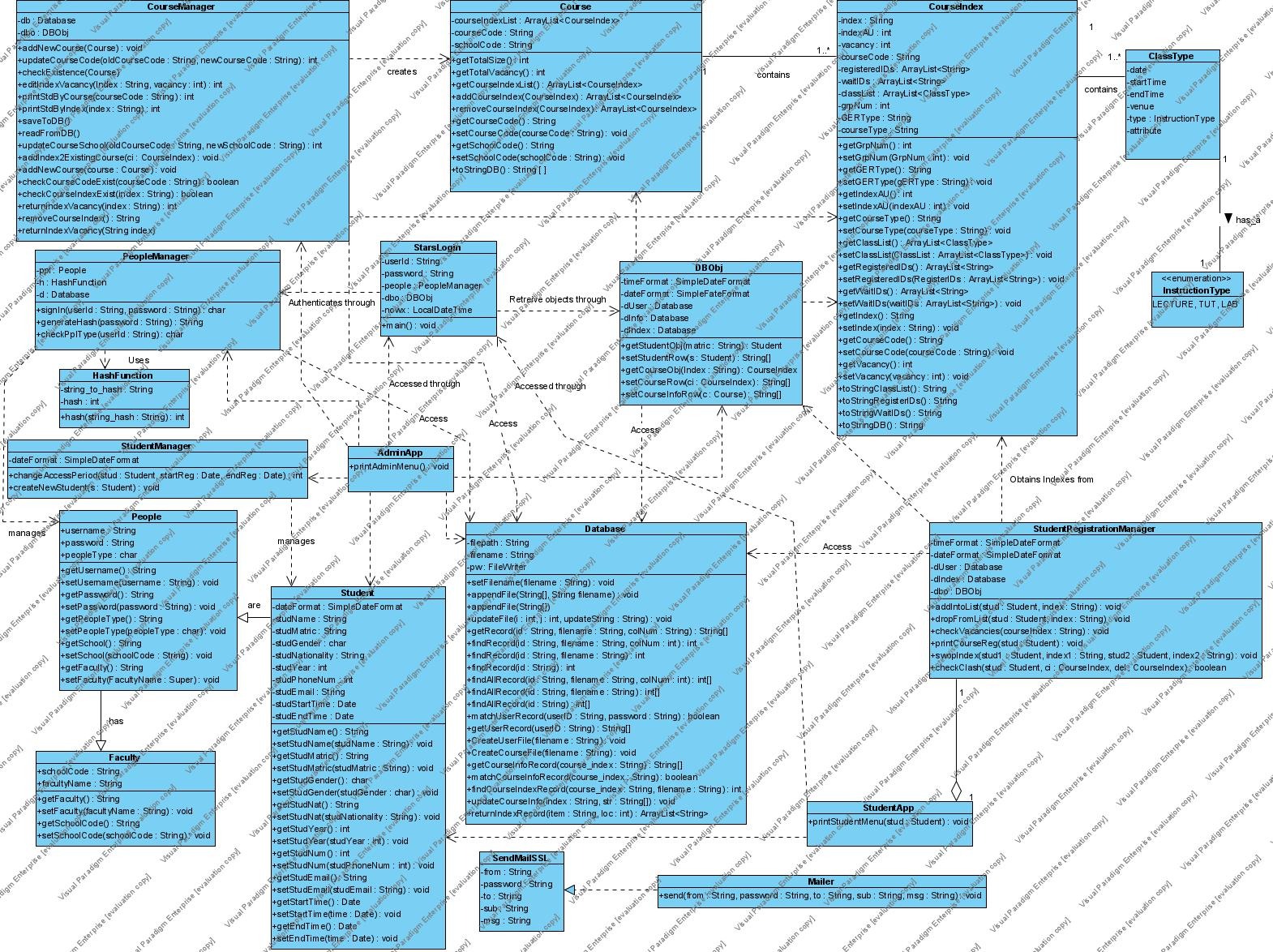
1. A Login/ Logout page for Student or Administrator access
2. The password is implemented in hashed format and stored in the Users.csv. When a password is entered, it will be hashed with the user id and will be compared with the hashed password stored in the database. If the password tallied, the user is then allowed to proceed.
3. The student class is implemented using inheritance method, which inherits from people class. People class will inherits from faculty class.
4. The student data is stored in file Users.csv. Each student record has a method for storage and retrieves the student information.
5. The course information is stored in file Course.csv, same as student record, each course information will have a method for storing and retrieving information.
6. Notification will be sent via email to student when the student is added to his/her pending index automatically.

The following assumptions are made in design consideration.

**Assumptions:**

1. Need not consider multi-users concurrent login.
2. Need not consider pre-requisite conditions when registering course.
3. Notification will be via external API which will be available later. System will just show a message that a notification is sent.
4. External source implies pre-existing records and can be loaded from the file/s.
5. Courses and Students records are to be stored in files.
6. DETAIL DESIGN

The design concept is used to develop the class diagram and is implemented below.



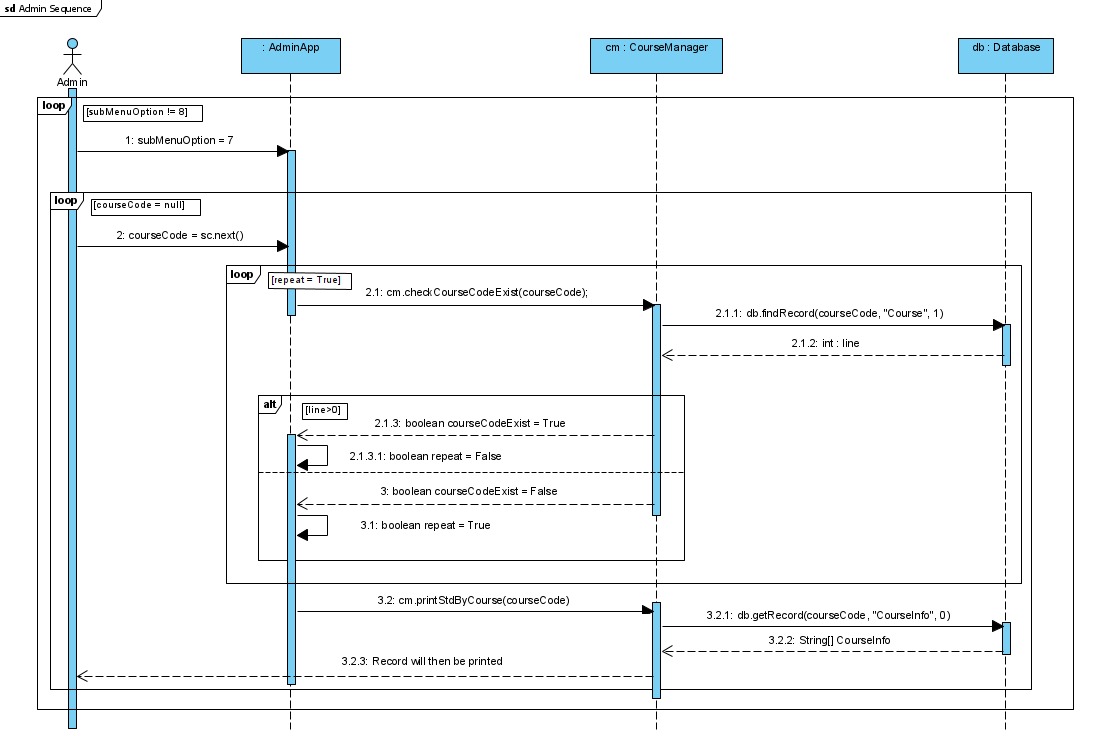
The respective class function is described below:

|  |  |
| --- | --- |
| **Class** | **Elaboration** |
| StarsLogin | holds the main class and is use for login purposes. In MySTARS login page, the user needs to key in the username and password, upon successfully log in, the system will be able to retrieve the user type and enter to the admin or student page. |
| AdminApp | Displaying the menu of the admin page for admin to add and amend information |
| StudentApp | Displaying student menu for course selection and amendment. |
| CourseManager | Manage, update, return and print course’s vacancy, course index |
| PeopleManager | For people sign in, check people type such as student or admin |
| StudentManager | For create student object and manage student registration period |
| StudentRegistractionManager | Manage student registration such as check vacancy, swap index, etc |
| CourseType | get and set class type such as days, time and venue |
| Course | get and set courses |
| CourseIndex | get and set course index, interacts with ClassType class |
| DBObj | set student and course |
| Faculty | set and get faulty object |
| People | get and set password, user name and user type, etc |
| Student | get and set student information |
| Mailer | For sending email, the main server is implemented using goggle mail service. |

Besides the above classes use for information exchange, we also have the following utilities class:

|  |  |
| --- | --- |
| **Class** | **Elaboration** |
| Database | Help to management database |
| DateTimeHelper | Handling the registration period |
| ErrorException | help to handle errors encountered |
| HashFunction | For converting password to hash code |

The following UML sequence diagram shows the flow of the “Print student list by Course” function of the student registration system.



The “Print student list by Course” method is resided in the CourseManager class.

The admin will enter in to a Menu list after log in. Once the admin select the print student list by course code (item number 7), the program will get keyboard entry of course code and check for existence. If exist, findRecord method will call get the data of student of the course and list the student number in course index format.

1. TESTING

We have created a test case using the following data and scenarios:

1. Data created

* The student database contains 15 students
* There are total 3 courses for selection
* Each courses shall have 10 vacancies
* Only 12 students can register a course where 10 will be confirmed and 2 in waiting list

2. The test scenarios is described below:

a. Student log in

|  |  |  |  |
| --- | --- | --- | --- |
|  | Test Case | Expected Outcome | Actual Outcome |
| a | Login before allowed period (dates) | Appropriate message display |  |
| b | Login after allowed period (dates) | Appropriate message display |  |
| c | Wrong password | Appropriate error message display |  |

b. Add a Student

|  |  |  |  |
| --- | --- | --- | --- |
|  | Test Case | Expected Outcome | Actual Outcome |
| a | Add a new student | the listing of **all** students should be displayed after the addition |  |
| b | Add an existing student | Appropriate error message display |  |
| c | Invalid data entries | Appropriate error message display |  |

c. Add a Course

|  |  |  |  |
| --- | --- | --- | --- |
|  | Test Case | Expected Outcome | Actual Outcome |
| a | Add a new course | listing of **all** courses should be displayed after the addition. |  |
| b | Add an existing course | Appropriate error message display |  |

d. Register student for a course

|  |  |  |  |
| --- | --- | --- | --- |
|  | Test Case | Expected Outcome | Actual Outcome |
| a | Add a student to a course index with available vacancies | Appropriate error message display |  |
| b | Add a student to a course index with 0 vacancies in Tut / Lab. | Appropriate error message display |  |
| c | Register the same course again | Appropriate error message display |  |
| d | Invalid data entries  (eg wrong student ID / course code, etc) | Appropriate error message display |  |

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

|  |  |  |  |
| --- | --- | --- | --- |
|  | Test Case | Expected Outcome | Actual Outcome |
| a | Check for vacancy in course index | Appropriate message display, eg 3/10 [vacancy/total size] |  |

f. Day/Time clash with other course

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

g. Waitlist notification

|  |  |  |  |
| --- | --- | --- | --- |
|  | Test Case | Expected Outcome | Actual Outcome |
| a | Add student A to a course index with 0 vacancies | Appropriate message display to inform student on waitlist |  |
| b | Drop student B from the same course index | Student B successfully dropped and student A successfully added. Simulate a notification sent. |  |
| c | Display student A timetable | Show the course on waitlist added. |  |
| d | Display confirmation message of waitlist successfully allocated a vacancy | Show the email message on waitlist allocated. |  |

h. Print student list by index number, course

|  |  |  |  |
| --- | --- | --- | --- |
|  | Test Case | Expected Outcome | Actual Outcome |
| a | Print list by  (i) Course | Appropriate display. |  |
| a | Print list by  (ii) index | Appropriate display. |  |
| b | Invalid data entries  (eg course code, index code etc) | Appropriate error message display |  |

1. CONCLUSION

The automated student registration system (MySTARS) designed and implemented by us is able to perform the task to be used by university for new student registration in the new semester. The basic requirements of the implementation and objective of allowing course student to apply the skill sets leaned in the semester have also achieved.

The existing MySTARS is designed with some assumption and limitations. Further improvements can be made by removing those assumptions and adding some of the following features:

1. Add in the features of prerequisite for the course
2. Implement graphical GUI
3. Add a data backup system using modern technology such as data backup storage at cloud
4. On top of email notification, add in a SMS notification system
5. Implement a web base platform