

**CE/CZ2002: Object-Oriented Design & Programming**

*Building an OO Application*

**Submitted by:**

**Lab group: FEP2**

**Matriculation Number:**

### School of Computer Science and Engineering

**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 | Signature /Date |
|  | (CE2002 or CZ2002) | Group |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Important notes:

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

**1 Program Design**

**1.1 Design Consideration**

During the process of designing the structure of the program there are a few considerations that our team had identified as our guiding principle namely:

1. Reduce inter-class dependency
2. Reusability
3. Open for extension & ease of change
4. Segregation of responsibility

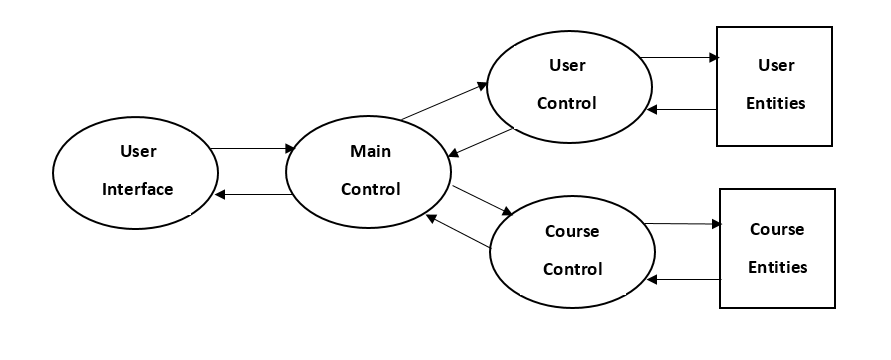
The program is design as such to reduce inter-class dependency. There are two main areas that our team have identified in the application where the control logic should be segregated: User management and Course Management. The segregation is achieved by having two separate control classes that works on its corresponding set of entities. The interaction between this two control class is handled by a main control class which also acts as an abstraction layer between the user interface and the two lower level data control classes.

This way, we are able to hide the implementation of the control class from the user interface class where the user interface does not need to know the flow of logic to operate. It is only responsible for the input and output of information. The two entities specific control classes will also be independent on each other. Therefore, when we need to make changes to either side of the segregation we only need to update the main control class. This minimise the impact of change on other parts of the software.

With this separation of control, it also greatly increases the reusability of the classes. The independent set of control class and entities can be ported over to application that utilises similar structure. For example, the user management can be re-used in system such as NTULEARN which is also a portal based system that required personnel management.

**1.1 Basic Flow of Design**

Upon receiving the signal from the user interface. The request for manipulation of data is initiated by the main control class to the course or/and user control class. The real data manipulating process operation is done within the respective entity specific control classes. Results are then passed back to the main control class. The returned data is then processed and packaged before passing on to the user interface for displaying.



**1.2 Design Structure**

|  |  |  |
| --- | --- | --- |
| **Boundary** | **Control** | **Entity** |
| UI | STARS (Main) | User (User) |
| ConsoleUI | UserManager (User) | Admin (User) |
|  | CourseManager (Course) | Student (User) |
|  | DataManager | AccessPeriod (User) |
|  | Notification | Course (Course) |
|  | AESEncrypter | Index (Course) |
|  |  | Timeslot (Course) |

**1.2.1 Boundary**

UI is an interface class which contain methods that defines the various user interface menu displayed in the application. For this project setting, the required user interface is via the console. Therefore, a ConsoleUI class is created to implement the UI interface. This give possibility of adding new display methods in the future such as a Graphical User interface or Web User interface without changing any functionality of the ConsoleUI implementation.

This extension of functionality is made possible by clearly segregating the functionality of interaction to the user(input/output) away from the Control and Entity classes. The boundary classes are strictly independent of the various entities. Therefore, it allows extension while reducing the needs for modification to the existing codes.

**1.2.2 ­Control**

**STARS**

The main control class of the program which is an abstraction layer between the UI and the lower level control class. This layer will hide the implementation details contained in UserManager and CourseManager from the high-level user interface modules.

It also serves as the middle man for both UserManager and CourseManager where it allows both classes to be independent of each other. All interaction between the two

**UserManager**

User object creation is done in this class where it also contains the data structure of all User objects. The AccessPeriod entities is also created and managed here. This class only contains User specific methods. It is not dependent on Course specific entities such as Course, Index, and Timeslots.

**CourseManager**

Course object creation is done in this class where it also contains the data structure of all Course objects. It only contains Course specific methods. It is not dependent on User specific entities as well.

**DataManager**

The class that contains all implementation of file read/write operation. The read/write operation are the same for all data structures that are to be serialize. Therefore, a common implementation is written here and shared with the respective managers to read/write data from files.

**1.2.3 Entity**

**User**

Contains user general information such as the login credential and email. It is the base class for Student and Admin entities.

**Student**

Extends on the User base class to include student specific information such as matric

number and course index list.

**Admin**

Extends on the User base class to include admin specific information.

**AccessPeriod**

Contain all information required for the creation of access period. An entity that is created and managed by UserManager control class to facilitate the control of user access to STARS application.

**Course**

Contains all information related to Courses in STARS application. It also composed of the index entity.

**Index**

Contains index specific information of a course such as the time slot information of tutorials and laboratory session. It composed of the TimeSlot entity and belongs in its corresponding Course entity.

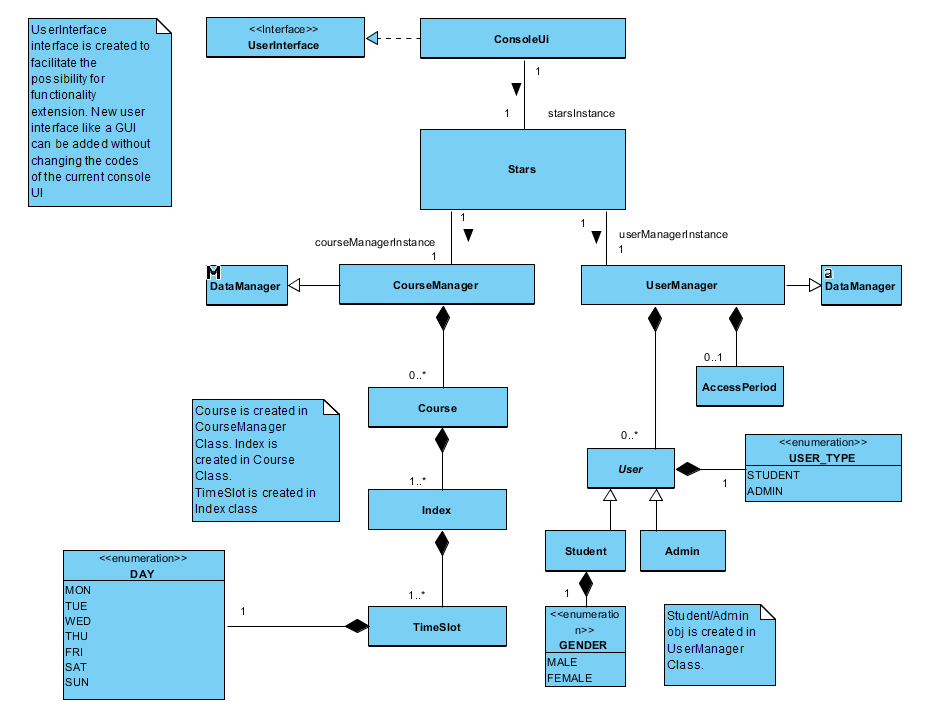
**TimeSlot**

Contain time specification of classes where it is composed in Index entity to specify timing information of classes in Index.

**2 UML Class Diagram & Classes relationship**

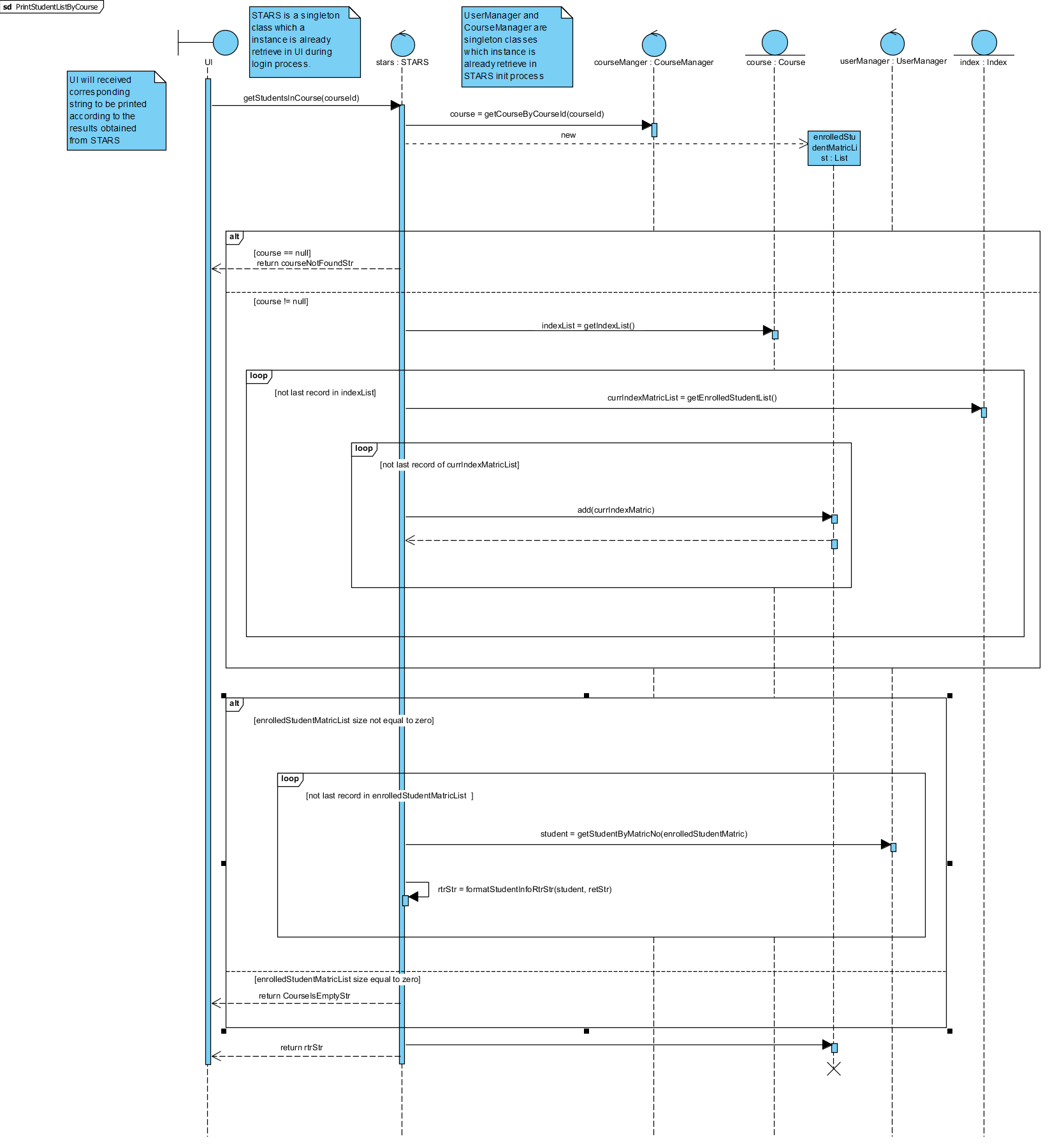
**2.1 UML Class Diagram**

A reduced UML class diagram is attached here to show the classes relationship. A full UML diagram complete with class attributes and methods can be found in the appendix.

****

**2.2 Classes relationship breakdown**

**3 UML Sequence Diagram**



**4 Test Case and Expected Output**

**4.1 Admin**

**4.1.1 Access Period and Password Check**

|  |  |  |
| --- | --- | --- |
|  | **Test Cases** | **Expected Outcome** |
| a | Login with account that doesn’t exist | Appropriate error message shown |
| b | Login with wrong password | Appropriate error message shown |
| c | Invalid data entries for set access period   * Non-date format * Overflow date * Start date before today * End date before start date | Appropriate error message shown |
| d | Student login outside of access period | Student is reminded of access period and will be log-out of system |
| e | Student login during access period | Shown the Student menu |

**4.1.2 Admin add a new student to STARS**

|  |  |  |
| --- | --- | --- |
|  | **Test Cases** | **Expected Outcome** |
| a | Invalid data entries for adding student   * Illegal email format * Adding an existing student * Invalid gender input * Adding of password that already exist | Appropriate error message shown |

**4.1.3 Admin add a new course to STARS**

|  |  |  |
| --- | --- | --- |
|  | **Test Cases** | **Expected Outcome** |
| a | Invalid data entries for adding course   * Adding an existing course * Invalid course ID format * End-time before start-time * Tut/Lab timing clash with each other or clash with lecture timing | Appropriate error message shown |

**4.1.4 Admin update a course to STARS**

|  |  |  |
| --- | --- | --- |
|  | **Test Cases** | **Expected Outcome** |
| a | Add index to course | * Steps to create time slot for Lab and Tut will be shown. * Index can have no Lab and Tut. * Within same index, Tut/Lab/Lec time slot are not supposed to clash with each other. |
| b | Delete index from course | * A list of students is de-enrolled due to index deletion will be shown. |
| c | Add lecture to course   * Add lecture that start time clashes with other classes in index | * Steps to create time slot for new Lec will be shown. * Lecture will not be added if timeslot for new lecture clash with classes(Lec/Tut/Lab) of the same course. |
| d | Remove lecture from course | * List of lectures to be delete will be shown |
| e | Add Tut/Lab to index   * Add classes that start time clashes with other classes in index | * Steps to create time slot for Lab or Tut will be shown. * Tut/Lab will not be added if timeslot for new Tut/Lab clash with classes(Lec/Tut/Lab) of the same course. |
| f | Remove Tut/Lab from index | * List of Tut/Lab to be delete will be shown |
| g | Delete Course | * A list of students is de-enrolled due to index deletion will be shown. |
| h | Invalid data entries for adding course   * Wrong format of input typed in * Invalid course ID format * End-time before start-time * Input time slots is already taken | * Appropriate error message shown |

**4.1.5 Check Vacancy**

|  |  |  |
| --- | --- | --- |
|  | **Test Cases** | **Expected Outcome** |
| a | Invalid data entries for check vacancy   * Invalid index entered | Appropriate error message shown |

**4.1.5 Print student list by index no./course**

|  |  |  |
| --- | --- | --- |
|  | **Test Cases** | **Expected Outcome** |
| a | Invalid data entries for print student list   * Invalid index/course entered | Appropriate error message shown |

**4.2 Student**

**4.2.1 Student register index**

|  |  |  |
| --- | --- | --- |
|  | **Test Cases** | **Expected Outcome** |
| a | Student add index | * A list of course with various index will be shown * Once user choose a course, the details (time slot of class) of index will be shown for reference. |
| b | Student choose an index of a course that he/she is already enrolled in. | * He/she will not be allowed to enrol into another index of the same course unless he/she choose explicitly in the switch index option. |
| c | Student choose an index with class timing that clashed with classes that he/she already enrolled in. | * He/she will not be allowed to enrol into index |
| d | Student enrolled into index with zero vacancy. | * A message will be shown that he/she is added into the waitlist of the course. |

**4.2.2 Student drop index/Waitlist Notification**

|  |  |  |
| --- | --- | --- |
|  | **Test Cases** | **Expected Outcome** |
| a | Student select to de-enrol from index | * A list of index that he/she is enrolled in will be shown for reference. |
| b | Student input index that he/she are not enrolled in | He/she will be prompt to input again |
|  | Student A de-enrol from an index with zero vacancy | * Student B which is first in line in waitlist will be enrolled into course with email notification sent out. |

**4.2.3 Student check courses registered**

|  |  |  |
| --- | --- | --- |
|  | **Test Cases** | **Expected Outcome** |
| a | Student select to check all courses that he/she registered | * A timetable of his/her classes will be shown |

**4.2.4 Switch Indexes in the Same Course**

|  |  |  |
| --- | --- | --- |
|  | **Test Cases** | **Expected Outcome** |
| a | Student choose to switch his/her currently enrolled index to another index in the same course.   * Student chooses new index that has time slot conflicts with classes that he/she is already in. * Student choose to switch to an index that has zero vacancy. | * A timetable of his/her classes will be shown * He/she will be shown an error message when there are time slot conflicts. * If new index has zero vacancy, student will be placed in waitlist. In an event of a new vacancy, Student will be checked again by system for new time slot clashes (student might enrol to another index while in waitlist). If there is a clash, student will be informed via email. |

**4.2.3 Student swap index with peer**

|  |  |  |
| --- | --- | --- |
|  | **Test Cases** | **Expected Outcome** |
| a | Student select an enrolled index | * Student will be told to input peer username, password and index to swap to authorise the change. |