

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

*Building an OO Application*

**SS10 Group 1**

Noah Seah Kaixian: U1922944F

Richard Yang Chen Xiao: U1921319D

Selvira Junita Melia: U1921926L

Shen Li Qin: U1920168F

Yeow Ying Sheng: U1722395B

Contents

[1. Design Considerations](#_b51jpw1e8qm8) **2**

[1.1 Approach](#_xpf5g71scb7x) 2

[1.2 Design Patterns](#_2baznud664bk) 2

[1.3 Principles Used](#_59r5xgwtamnp) 3

[1.4 New Functionalities](#_w7aja9g2lm5e) 4

[1.5 Assumptions](#_hsz80kfuw61x) 4

[2. UML Class Diagram](#_teh6s6lc485b) **5**

[3. UML Sequence Diagram](#_f3dmbds6aks0) **6**

[4. Testing](#_g2s7bbg990u) **7**

[5. Appendices](#_4857kgd7i4oo) **14**

# 1. Design Considerations

## 1.1 Approach

To align our design with object oriented design goals, the design aims to minimise impact of change in managing dependencies. We started off by sitting down and thinking about how the classes should be drawn out. We analysed the assignment requirements to deduce what are our test cases, and entities. With our test cases, we then thought about how to dedicate the tasks so that minimal code is repeated for different functionalities. As seen in the class diagram (see Section 2), the classes are designed such that they are only linked to a few others to achieve loose coupling. Furthermore, the classes only perform their sole functions for high cohesion.

During implementation, exception handling was taken into special consideration. The classes’ responsibilities include checking that whatever information they are getting is correct. Lower level classes (entity) throw exceptions when bad data is received, while the higher level classes (control) catch these errors and conduct error handling and display relevant error messages. Furthermore, we designed the system with the intention for it to be easily extendable in the future. We made use of abstract classes and interfaces so that the system can be easily extended to GUI, webpage, etc.

## 1.2 Design Patterns

In creating objects, a couple of design patterns are used. First, by using the Factory Pattern, our design has a Factory class that creates various entity objects. This pattern ensures the details involved in creating objects are not shown to client classes, so high level classes do not have to worry about low level classes. Second, we used the Singleton Pattern, which ensured that only one instance of the class is created. In our case, this prevents classes from changes to different versions of the database objects.

The Data Access Object (DAO) pattern separates low level data from high level functions. Interface classes outline the functions to be performed on the objects, which are then implemented by concrete classes such as TextCourseDataAccessObject and TextUserDataAccessObject. Furthermore, authorisation is used to ensure security. For example, students can only read the data, while admins can read and write the data via the appropriate interfaces. The Transfer Object pattern is used to pass data with multiple attributes such as Course/AbstractUser. This helps to make updating of data easy and safe.

## 1.3 Principles Used

Based on the **Single Responsibility Principle (SRP)**, the classes have been defined such that their methods only support the main functionality of the class. This strengthens the cohesion of each class, while also preventing a class from having multiple responsibilities that can potentially be interdependent. To illustrate, the LoginControl class only contains a login method to get the user object. Ultimately, SRP discourages classes from having more than one reason to change.

Using the **Open-Closed Principle (OCP)**, the classes have been defined such that their internal functionalities can be modified, but can be open for extension by having the methods be overridden, when appropriate. For instance, the ILoginControl class is implemented by ConsoleLoginControl. The main function for instance, will only need to know about ILoginControl class. This means that in the future, it will be easy to implement a GUILoginControl class since they will still be based on the same interface.

The **Liskov Substitution Principle (LSP)** ensures that subclasses are substitutable for their base classes by having appropriate pre-conditions and post-conditions. To illustrate, the ConsoleLoginControl class implements the login method in ILoginControl class. With the principle, the subclasses are to expect no more and provide no less than what is predefined by the interface. Thus, when ConsoleLoginControl returns an AbstractUser after running login(), it is expected that the AbstractUser returned is a valid user in the database.

There are specific interface classes implemented to align our design with the **Interface Segregation Principle (ISP)**. This prevents a module from depending on general interfaces containing irrelevant methods. For instance, the data access object functionality is divided into ICourseDataAccessObject, IRegistrationDataAccessObject, IUserDataAccessObject classes. These classes contain only methods that are relevant to the data they contain.

The **Dependency Injection Principle (DIP)** states that high-level classes should not depend on concrete classes, but rather on interfaces or abstract classes. Additionally, abstractions should not depend on concrete classes. This encourages high-level classes to be independent of lower-level implementations. For example, the TextCourseDataAccess class implements the ICourseDataAccessObject class to implement its methods, while the interface class is independent of concrete details. Furthermore, the Factory class forces all classes to depend solely on the Factory to generate new objects, which accomplishes DIP.

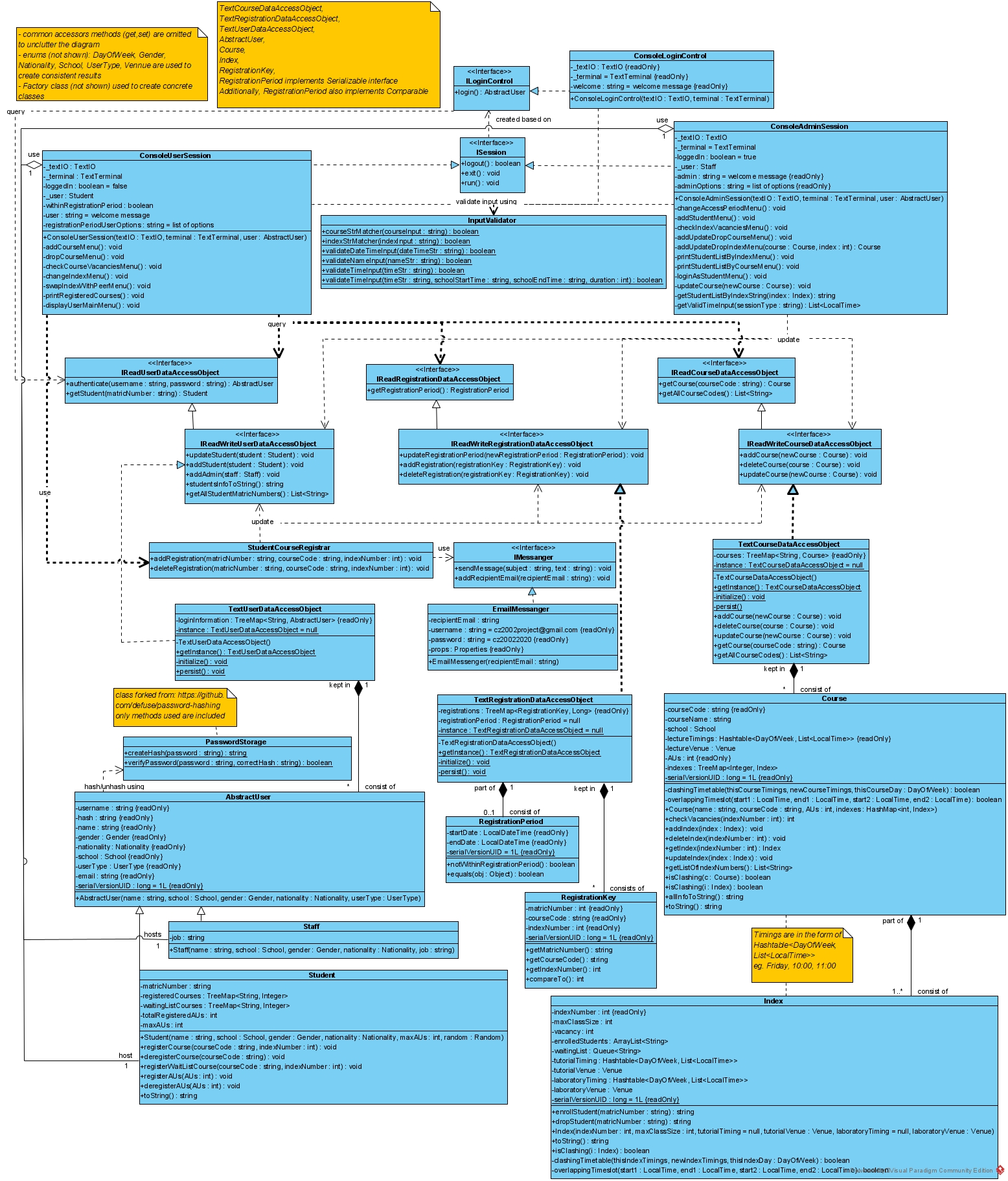
## 1.4 New Functionalities

1. Delete index/course function - auto disenroll (waitlist) students in the index/course. Students’ list of registered courses will be updated too.
2. Option to email all students when there is a change in access period.
3. Ability to press alt Z to cancel current action and go back to the main page.
4. Ability to log in on as students and bypass registration period restriction (ie register for courses after registration period is over).

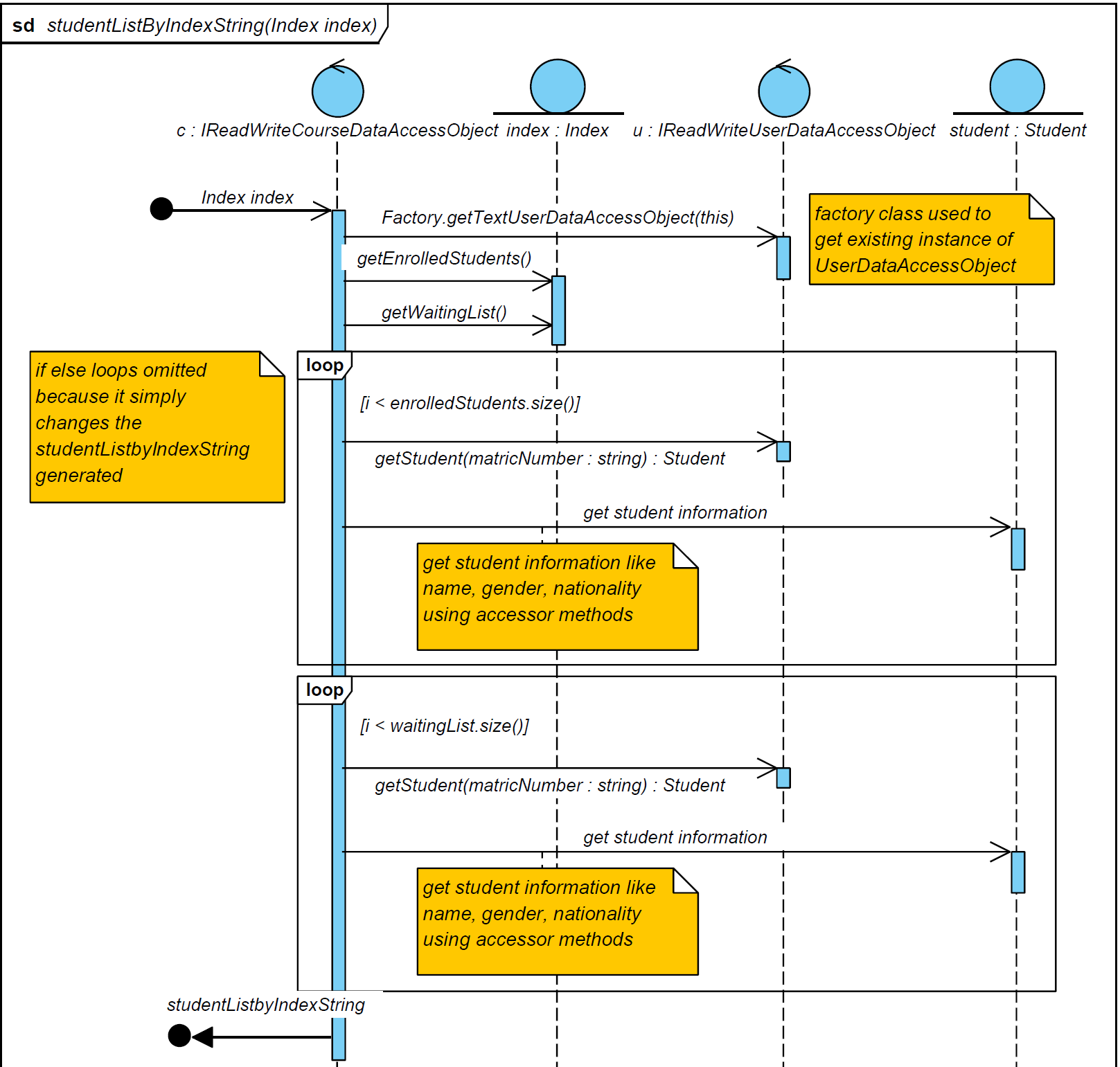
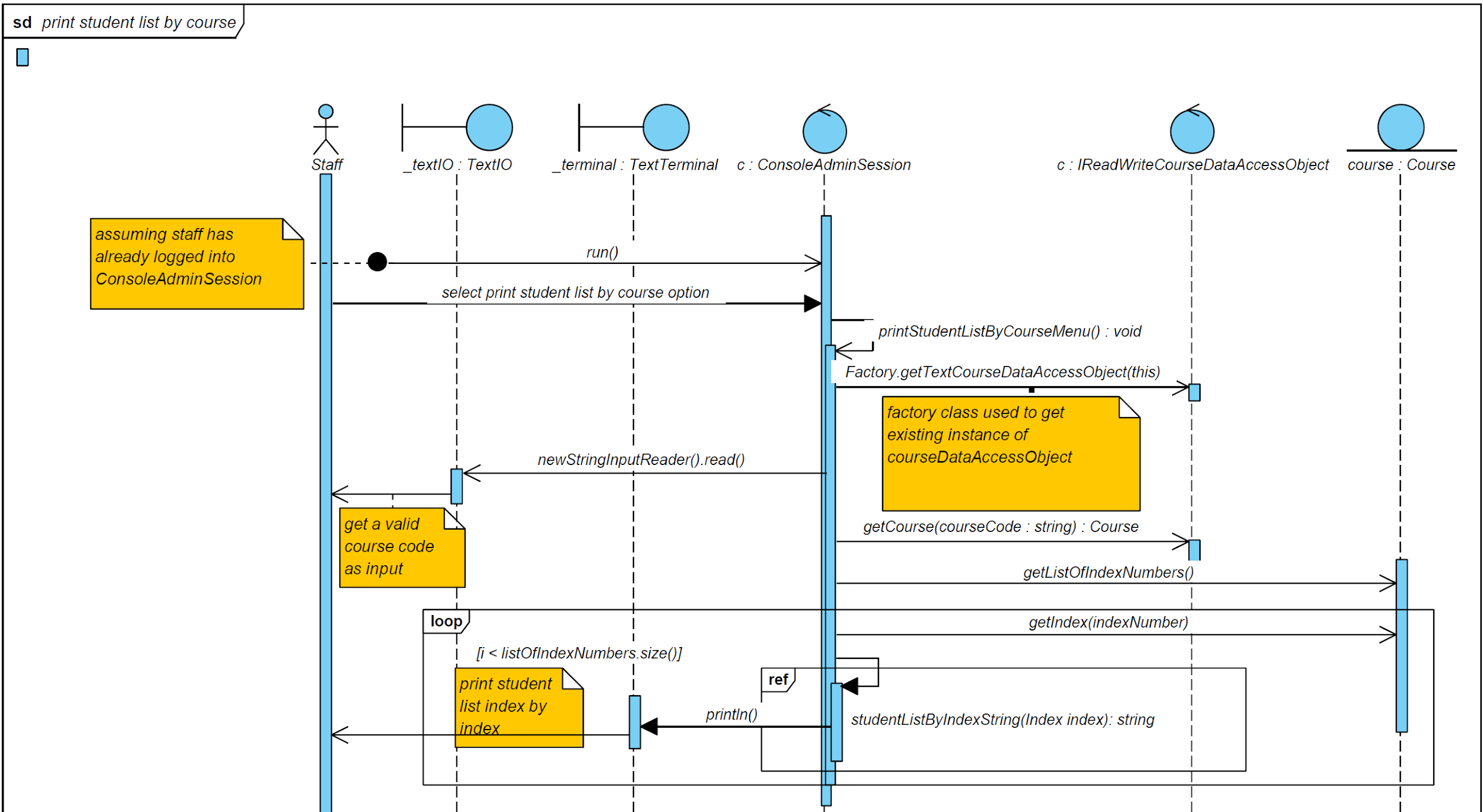
## 1.5 Assumptions

1. Access period only affects ‘Add Course’, ‘Drop Course’, ‘Change Index Number of Course’, ‘Swap Index Number Peer’ functions. Students are still allowed to login outside of access period to ‘Check/Print Courses Registered’ and ‘Check Vacancies Available’.
2. There exists only one access period that applies to all students.
3. Classes (indexes) can occur at the same venue simultaneously.
4. Courses can have the same name. Courses are unique only in terms of their course codes.
5. Usernames are unique.
6. When changing index, Student wanting to change into an index class with no vacancies will be put on waiting list, and have their current index dropped.
7. Class size cannot be decreased.

# 2. UML Class Diagram



# 3. UML Sequence Diagram



# 4. Testing

Tests 1-14 on the different functionalities of the program. Note that there is a general test case A in Appendix A that is referred to several times.

1. Student Login / Access period *(refer to Assumption 1 regarding access period)*

|  |  |  |
| --- | --- | --- |
|  | **Test Case** | **Expected Outcome** |
| a | Login before allowed period (dates) | **User Dashboard Display:**  “It is not registration period, only checking registered courses/course vacancies is functional”    **Student tries to:**   * Register/drop a course * Change index of a course * Swap index number with a peer   Outcome: “registration period has not started/over” |
| b | Login after allowed period (dates) | Same as in 1a. |
| c | Wrong password | “Wrong username/password” |

1. Add a student

|  |  |  |
| --- | --- | --- |
|  | **Test Case** | **Expected Outcome** |
| a | Add a new student (Bobby) | List of all students, including Bobby, and their matric numbers are printed out. |
| b | Add an existing student  (student with the same name) | The programme randomly generates a unique matric number for every student added and will not be able to add an existing student per se. |
| c | Invalid data entries | **Invalid Name Input**   * Contains numbers   + Output: “name cannot contain number” * Less than 3 characters   + Output: “Invalid value. Expected a string with at least 3 characters.”   **Invalid Gender Input** - Same as in A, max 2  **Invalid Nationality Input** - Same as in A, max 3 (Singaporean, Malysian, Others)  **Invalid MaxAUs Input**   * Number more than 25 * Number less than 0 * Letters/special characters: abc!@   Output: “Expected an Integer value between 0 and 25.” |

1. Edit student access period

|  |  |  |
| --- | --- | --- |
|  | **Test Case** | **Expected Outcome** |
| a | Update registration period | “Successfully changed access period.” |
| b | Invalid data entries | **Invalid start date or end date Input**:   * Contains alphabets: abc * Contains symbols: !@# * Not in yyyy-MM-dd HH:mm format   Output: “invalid date format”  **Start date is later than end date**:   * Start date: 2019-01-01 09:00 * End date: 2018-01-01 09:00   Output: “start period should occur after end period”  **Start date and end date is the same**:   * Start date: 2019-01-01 09:00 * End date: 2019-01-01 09:00   Output: “Start and end period cannot be equal” |

1. Add a course

|  |  |  |
| --- | --- | --- |
|  | **Test Case** | **Expected Outcome** |
| a | Add a new course |  |
| b | Add an existing course | Output:  “Course already exist. Updating course instead of adding…”  Redirects admin user to update course instead (refer to 6) |
| c | Invalid data entries for adding course details | **Invalid Course Input**   * Consists of symbols: -c2002 * Consists of only integers: 123455 * Exceeds 6 alphanumeric characters: czs2002   Output: “invalid course code format”  **Invalid School Input** - Same as in A, max is 16  **Invalid AUs Input**   * More than 4 * Less than or equal to 0 * Consists of alphabets/symbols   Output: “Expected an integer value between 1 and 4”  **Invalid Duration Input** - Same as above (Invalid AUs)  **Invalid Start Time Input**   * Starts before school hours: 07:00 * Ends beyond school hours: Starts at 21:00 with a duration of 1hr * Starts after school hours: 22:00 * Not at a 30min interval: 08:39 * Invalid format: 08:63 * Consists of symbols: -9:30   Output: “timing is invalid. school should start earliest at 07:30 AND classes should start at a 30min interval.”  **Invalid Selection for Lecture Day** - Same as in A, max 6 (Monday - Saturday)  **Invalid Selection for Lecture Venue** - Same as in A, max 13  **Invalid Y/N Input to Continue Adding Lecture**   * String value not Y nor N: p * Numbers or special characters: 8!   Output: “Invalid value. Expected: Y / N”  **Sessions/Index Groups to Course**   * Consists of symbols: @y * Inputs other than ‘y’, ‘n’, ‘Y’ or ‘N’, for example, ‘yes’   Output: “Invalid value. Expected Y / N”  **Invalid Max Class Size Input for Index Group**   * Less than or equal to 0 * Non-integer values: 4.5 * Consists of alphabets/symbols: @!1   Output: “Expected an integer value greater than or equal to 1” |
| d | Add a lecture day (Wednesday) when another lecture session falls on the same day (Wednesday) | “there is already a lecture session on WEDNESDAY. please select another day.” |
| e | Add course details (course code, course name, school, lecture sessions) | Program prompts the admin user to add a new index group with an automatically-generated index group number. |
| f | New index group created | “Successfully added new index”    Displays current index information and redirects user to update index information (refer to 6g) |

1. Delete a course/index group

|  |  |  |
| --- | --- | --- |
|  | **Test Case** | **Expected Outcome** |
| a | Delete an Existing Course | “Course deleted”  All students enrolled in the course (if any) are de-registered. |
| b | Delete an Existing Index Group | “Index deleted”, “Successfully updated course”  All students enrolled in the index group (if any) are de-registered.  **[If there are no longer any index groups for the course]**  “Course has no more index, deleting course…”  “Course deleted” |

1. Update a course

|  |  |  |
| --- | --- | --- |
|  | **Test Case** | **Expected Outcome** |
| a | Update course name | “Successfully updated course name” |
| b | Update school | “Successfully updated school” |
| c | Update Lecture Venue | “Successfully updated lecture venue” |
| d | Add/Update index group   1. Add/Update Tutorial   Timing   1. Add/Update Laboratory Timing 2. Add/Update Tutorial   Venue   1. Add/Update Laboratory Venue 2. Update Maximum   Class Size | 1. **Add/Update Tutorial Timing**   Output: “Successfully added/updated tutorial timing”   1. **Add/Update Laboratory Timing**   Output: “Successfully added/updated laboratory timing”   1. **Add/Update Tutorial Venue**   Output: “Successfully updated tutorial venue”   1. **Add/Update Laboratory Venue**   Output: “Successfully updated laboratory venue”   1. **Update Maximum Class Size**   Output: “Successfully updated maximum class size”  Note that if Tutorial/Laboratory Venues are null, adding/updating Tutorial/Laboratory Timings will prompt 3 / 4   * Output: “Tutorial timing cannot exist without venue.”   Similarly, if Tutorial/Laboratory Timings are null, adding/updating Tutorial/Laboratory Venues will prompt 1 / 2.   * Output: “Tutorial venue cannot exist without timing.” |
| e | Invalid data entries   * Selecting course * Selecting course info to add/update * Selecting index info to add/update | **Invalid Course Selection** - Same as in A, max is (number of existing courses + 1)  **Invalid Course Info Selection** - Same as in A, max 6  **Invalid Index Info Selection** - Same as in A, max 7 |
| f | Invalid data entries - options under “**course info** to add/update”: | **Invalid School Selection** - Same as in A, max 16  **Invalid Lecture Venue Selection** - Same as in A, max 13  **Invalid Index Group Selection** - Same as in A, max is (number of existing index groups for course + 1) |
| g | Invalid data entries - options under “**index info** to add/update”:  Index info to add/update:   1. Add/Update Tutorial Timing 2. Add/Update Laboratory Timing 3. Add/Update Tutorial Venue 4. Add/Update Laboratory Venue 5. Update Maximum Class Size 6. Return to previous menu | 1. Add/Update Tutorial Timing   **Invalid Tutorial Day Selection** - Same as in 4c  **Invalid Duration Selection -** Same as in 4c.  **Invalid Start Time Selection -** Same as in 4c.   1. Add/Update Laboratory Timing   **Invalid Laboratory Day Selection** - Same as in 4c  **Invalid Duration Selection -** Same as in 4c.  **Invalid Start Time Selection -** Same as in 4c.   1. Add/Update Tutorial Venue   **Invalid Tutorial Venue Selection -** Same as in 4c (Invalid Selection for Lecture Venue).   1. Add/Update Laboratory Venue   **Invalid Laboratory Venue Selection** - Same as above (Invalid Tutorial Venue Input).   1. Update Maximum Class Size   **Invalid Class Size Input** - Same as in 4c. |

1. Register student for a course

|  |  |  |
| --- | --- | --- |
|  | **Test Case** | **Expected Outcome** |
| a | Add a student to a course index with available vacancies. | Output: “successfully added course”. Prints out registered courses, and waiting list courses. |
| b | Add a student to a course index with 0 vacancies in Tut / Lab. | Output: “maximum class size reached, adding you to waiting list instead.” |
| c | Register the same course again | Output: “course already registered” |
| d | Invalid data entries  (eg wrong student ID / course code, etc) | **Invalid Course Selection** - Same as in A, max is same as number of existing courses  **Invalid Index Selection** - Same as in A, max is same as number of existing index groups available in the course |
| e | Add course that clashes with registered/waiting list courses | “Unable to add course, time table clashes.” |
| f | Add course that when it exceeds student’s max AU | “You exceed max AU limit.” |

1. De-register student for a course

|  |  |  |
| --- | --- | --- |
|  | **Test Case** | **Expected Outcome** |
| a | Drop a student from a course index. | Output: “Successfully dropped course”. Prints out registered courses and waiting list courses. |
| b | Invalid data entries | **Invalid Course Selection** - Same as in A, max is same as number of registered courses student has |

1. Check available slots in a class (Admin/Student)

|  |  |  |
| --- | --- | --- |
|  | **Test Case** | **Expected Outcome** |
| a | Check for vacancy in course index   * Admin/Student checks vacancy for index 200100 of course cz2001 | “Successfully retrieved vacancies. There is 0/10 vacancy for 200100 for cz2001. There is 0 student in the waiting list.” |
| b | Invalid data entries   * Course Selection * Index Selection | **Invalid Course Selection** - Same as in A, max is same as number of existing courses  **Invalid Index Selection** - Same as in A, max is same as number of existing index groups available in the course |

1. Day/Time clash with registered course

|  |  |  |
| --- | --- | --- |
|  | **Test Case** | **Expected Outcome** |
| a | Add a student to a course index with available vacancies.   * Student tries to register for a course index that clashes with their current timetable. | Output: “unable to add course, time table clashes” |

1. Waitlist notification

|  |  |  |
| --- | --- | --- |
|  | **Test Case** | **Expected Outcome** |
| a(i) | Add studentA to a course index with 0 vacancies | “maximum class size reached, adding you to waiting list instead.”  Prints out student waiting list |
| a(ii) | Drop studentB from the same course index | First student who was added to the waiting list (studentA) will be automatically registered to the course index. Email notification sent to studentA. |
| a(iii) | Display studentA timetable | Course index now listed under registered courses. |

1. Print student list by index number, course

|  |  |  |
| --- | --- | --- |
|  | **Test Case** | **Expected Outcome** |
| a | Print list by   1. Course 2. Index | i) Printing by **course**.  Prints out enrolled students with name, gender and nationality details, and students on waiting list.  ii) Printing by **index**.  Prints out enrolled students with name, gender and nationality details, and students on waiting list. |
| b | Invalid data entries  (eg course code, index code etc) | **Invalid Course Selection** - Same as in A, max is number of existing courses  **Invalid Index Selection** - Same as in A, max is number of existing index groups in the course |

1. Change course index

|  |  |  |
| --- | --- | --- |
|  | **Test Case** | **Expected Outcome** |
| a | Change course index | Output: “Successfully swapped course index.” |
| b | Invalid data entries  (eg selection etc) | **Invalid Course Selection** - Same as in A, max is number of registered courses that student has  **Invalid Index Selection** - Same as in A, max is number of existing index groups in the course |
| c | User has not registered for any course | Output: “No course registered.” |
| d | Index group that the student wishes to change to is full | Student wishes to change index group from 200101 to 200100.  Output: “maximum enrolled students, added to waiting list instead.”  Student is no longer registered for 200101. |
| e | User already in the index | Output: “You are already in the index” |
| f | New index schedule clash with user schedule | Output: “Unable to add course, timetable clashes deleted course with clashing timetable…” |

1. Swap index with peer

|  |  |  |
| --- | --- | --- |
|  | **Test Case** | **Expected Outcome** |
| a | Swap index with peer   * Current index is 200101 | “Successfully swapped indexes. Your new index is: 200100”  New index is reflected under the list of registered courses for both students. |
| b | Invalid data entries | **Invalid Course Selection** - Same as in A, max is number of registered courses student has  **Password input is less than 6 characters**  Output: “Invalid value. Expected a value of at least 6 characters”  **No existing user with the given username and password**  Output: “Wrong username and password. Try again.” |
| d | User has not registered for any courses | Output: “no course registered” |
| e | Peer did not register for the course | Output: “Peer is not registered for this course” |
| f | New index schedule clash   * Clashes for user(/peer) | Output: “Unable to swap. There is a clashing timeslot in your(/your peers’) timetable” |

# 5. Appendices

*Appendix A*

1. Invalid data entry for Selection

|  |  |  |
| --- | --- | --- |
| a | Invalid data entry for Selection | **Invalid data entry for Selection.**   * Number out of range: input greater than max value * Non-integer number: 2.5 * Letters/special characters: abc!@#   Output: “Invalid value. Enter a value between 1 and <max value>.” |

*Appendix B*

Link to video demonstration: https://youtu.be/X6qpepYBXTc