NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

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
SEAH KAH YEN	SC2002	SMACS	[illegible] 18/11/23
GORDON TAN AU AUN	SC2002	SMACS	[illegible] 18/11/23
PHUA GUAN YUAN	SC2002	SMACS	[illegible] 18/11/23
CHAN ZI SHEN	SC2002	SMACS	[illegible] 18/11/23
CHOO YI KEN	SC2002	SMACS	[illegible] 18/11/23

Part A: Design Consideration

Our application design aims to ensure that it is easy to maintain, allows for extensibility and allows for ease of edits by adding functionality. This can be done by properly using classes' dependencies and minimizing the impact of changes in our system through SOLID design principles.

SOLID Design Principles

Single Responsibility Principle:

The Single Responsibility Principle (SRP) encourages high class cohesion. In this case, we meet SRP by having each class perform only one highly specific task. Here are some examples of how we used SRP in our design.

The `CampInformation` class only display all the information of the camps available, which is entered by the staff when staff creates a camp using `CreateCamp()` method.
Furthermore, by making the method in `CampInformation` class public, other classes who wish to display the information can use this method without repeating the code.

The `Suggestion` class is responsible for displaying information such as the list of suggestions made by which camp committee members for changes to their camp details, and whether their suggestions is answered by staff.

Similarly, the `Enquiry` class is responsible for displaying information such as the list of enquiries made by which students regarding a specific camp, and replies to their enquiry by either the camp committee member or a staff.

Open-Closed Principle:

This principle highlights the importance of open for extension but closed for modification. With this principle, we want to be able to change what modules do without changing what source modules do. This is visible in some of the flexibility in the design of classes.

For example, `User` superclass is designed in such a way that it can be extended for more subclasses. Currently in our design, `Student` class and `Staff` class are subclasses that are extending our `User` class. In future, when more types of users are considered in our designed program such as when we need another type of user: Guest to represent users with limited access or temporary access to our application, we can just extend `User` class with another new `Guest` class, thus `User` class is 'open for extension'.

Another example in our design is using `Student` class as superclass and opening it for further extensions. In our current design, we used `CampCommittee` class and `CampAttendee` class to extend our base class. When we want to alter what specific roles that `CampCommittee` and `CampAttendee` do, we can just directly change the code from subclasses, without altering code in our superclass as we want base class to be 'closed for modification'. Therefore, Open-Closed Principle is not violated.

Liskov Substitution Principle:
The Liskov Substitution Principle (LSP) states that subtypes must be substitutable for their base types. In other words, pre-conditions of sub-class methods should not be stronger than the base class methods and post-conditions of sub-class methods should not be weaker than the base class method.

Here are some examples provided to show how we used LSP in our design. For example, we used `UserAccount` interface as base class and `User` class as subclass. The method `changePassword()` in `UserAccount` interface is implemented in the subclass `User` class.

Not only that, another example is using `CreateCamp` interface as base class and `Staff` class as subclass. The methods `createCamp()`, `editCamp(Camp camp)` and `deleteCamp(Camp camp)` are implemented in the subclass, reiterating the idea that subclasses are indeed substitutable for their base classes.

Observe that both the subclasses does everything their respective base classes do and even more, thus fulfilling the ‘expect no more, provide no less’ criteria of LSP. Therefore, Liskov Substitution Principle is not violated, contributing to a more flexible and robust design.

Interface Segregation Principle:

The Interface Substitution Principle (ISP) states that classes should not be exposed to methods that they do not need. In other words, classes should not depend on interfaces that they do not use.

In our design, we implemented ISP through `GenerateStaffReport` and `GenerateStudentReport` interfaces, instead of combining the both interfaces into one `'GenerateReport'` interface. In general, the two interfaces do the same thing, which is generating reports (the CampList report and Enquiry report). `GenerateStaffReport` interface is used by `Staff` class to generate reports specifically catered to staff usage while `GenerateStudentReport` is used by `CampCommittee` class to generate reports specifically catered to camp committe usage. Using separated interfaces instead of a combined `'GenerateReport'` interface helps in avoiding unnecessary dependencies. `Staff` class and `CampCommittee` class do not need to implement methods that they do not need, resulting in a clean and modular design. Thus, Interface Segregation Principle is not violated.

<u>Assumptions made</u>

- Students cannot join as committee for more than 1 camp
- Number of attendees included number of committees
- Camp visibility is set to not visible when created
- Staff can only approve of reject a suggestion, and cannot input string as reply message for suggestions
- Points are awarded even the suggestion made is deleted after
- Password is allowed to leave blank
- Students can see the list of committee but not attendees
- Staff in charge cannot be changed
- Minimum slots for committee and attendee is 1
- Students can only submit enquiry after joining as attendee
- Students can still see the camps from their faculty including those are full or having date clash with their registered camps
- Withdrawal request from a student is approved by default

Object-Oriented Features

Abstraction:

Abstraction is a fundamental OOP concept that involves distinguishing the essential characteristics of an object from all other kinds of objects. For example, real-world objects such as Camps, Students and Staffs are abstracted alongside their real life behavior/states and are used as java classes alongside their respective attributes and methods. One such example would be the CampInformation class, which contains all corresponding camps, locations, name and attendees.

Encapsulation and Information Hiding:

Encapsulation refers to protecting an object's private data through a barrier while information hiding hides the details or implementation of the class from users. In our case, the attributes of our classes are set to private which are only accessible through our getter and setter methods.For example, to get the camp's name or location, they can only be accessed via `getCampName()` or `getLocation()` getter methods respectively. One such example is our enquiry class `getReply()` method. It allows staff to generate replies to enquiries to generate the enquiry report without knowing the implementation of the method.

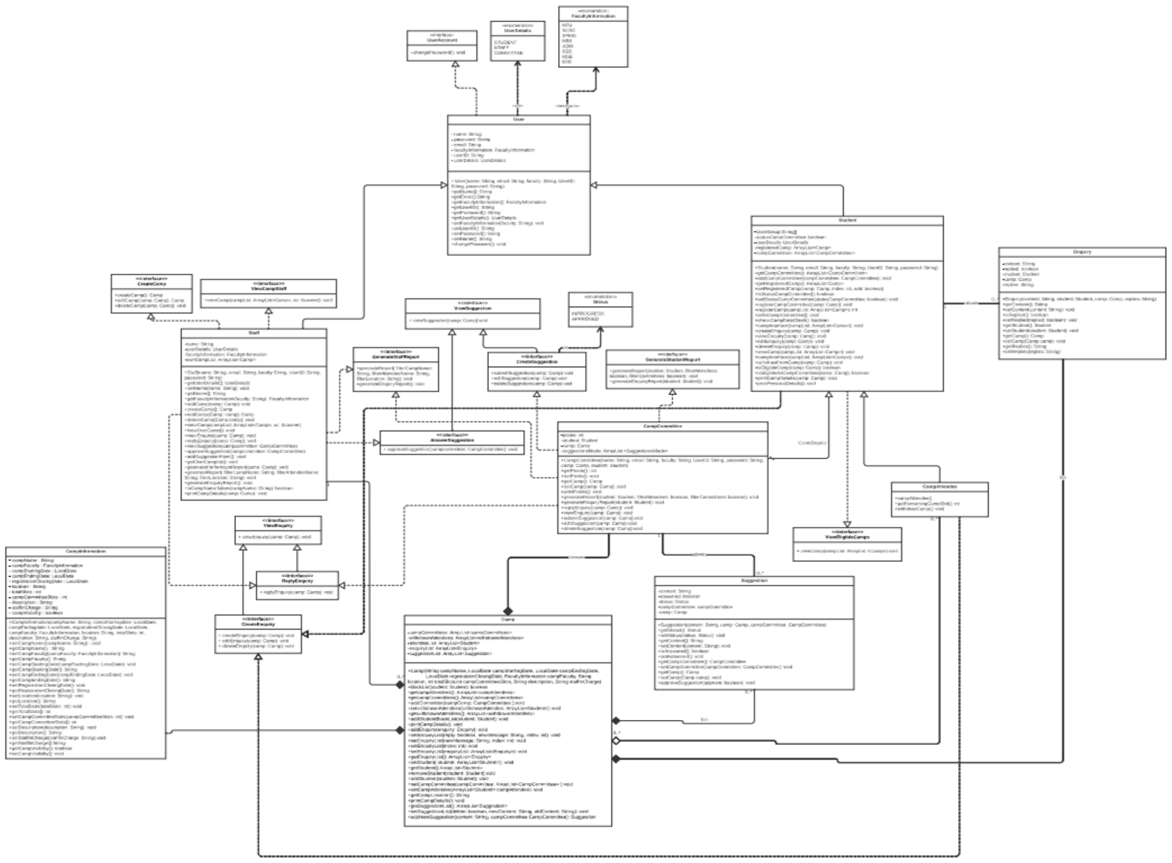
Inheritance:

Inheritance is powerful OOP tool that enables subclasses to inherit behaviours of superclasses to support code reusability and reduce programming effort in implementing new classes. For example in our design, `CreateCamp` interface is superclass of `Staff` class. Since `Staff` class, a concrete class has no subclasses that inherits from it, it has to implement all methods inside `CreateCamp` interface which are `createCamp()`, `editCamp(Camp camp)` and `deleteCamp(Camp camp)` due to inheritance.

Polymorphism:

Polymorphism is an OOP tool that enables objects of different subclasses to be treated as objects of a common base class by performing a single action in different ways using the same base class. For instance, both `CampCommittee` and `CampAttendee` is a subclass of `Student`, and thus objects from both classes is able to be added into an array list of `Student`, enabling `Staff` to call `studentList()`, for the list of students participating in the camp.

Part B: Detailed UML Class Diagram



(we will attach a PDF so that you can zoom in clearly to see the functions)

Part C: Testing

No.	**Description**	**Expected Result**
1	**Login Page**	
1.1	Users are required to input their registered Username and default password 1. Inputing an Invalid UserName 2. Password changing during first time login 3. Invalid password login 4. Invalid password login after 3 tries	1. "No such available UserID in database!" 2. "Please change your password. Right NOW!" 3. "Incorrect password! Try again! Tries left: 2" 4. "Incorrect password! Try again! Tries left: 0" Password Incorrect, directing to login page…"

2	**Staff Menu**	
2.1	Change user’s password 1. Input choice (1)	Users will be prompted to change password if logging in for the first time.
2.2	Viewing all camps 1. Input choice (2)	All camps created by any staff in system will be displayed
2.3	Viewing camps created by staff 1. Input choice (3) 2. Input the camp number to register	All camps created by this specific staff will be displayed
2.4	Creating a new camp 1. Input choice (4) 2. Input invalid format for date 3. Input camp ending date that is before camp starting date 4. Input camp registration deadline that is after camp starting date 5. Input more than 10 for camp committee slot	Staff will need to input details of the camp correctly to create. Errors will be as follow: 1. “Invalid input. Please enter a date in the format yyyy-mm-dd.” error message and prompt 2. “Camp ending date must after the actual camp.” 3. “Deadline must before the actual camp” 4. “Invalid number of slots, it must between 1 and 10 inclusively”
2.5	Logging out 1. Input	Redirected to login page
3	**Student Menu (also contains function 2.5)**	
3.1	Change user’s password 1. Input choice (1)	Users will be prompted to change password if logging in for the first time.
3.2	Viewing available camps according to student’s faculty 1. Input choice (2)	If no such camp exist, “No available camps” will be printed Else, camps will be printed out in sequence according creation sequence
3.3	Registering Camp 1. Input choice (3) 2. Input the camp number to register 3. Input choice of camp attendee or camp committee	Student can choose to register for any camp they can view as camp attendee or camp committee. 1. Students will only see the camps with no clash on date with their registered ones 2. A student can only register as camp committee given enough slots 3. A student can only register as camp

		committee for only 1 camp
3.4	Viewing registered camps	Student can generate the list of camps they have registered for, either as an attendee or committee.
4	**Staff Camp Menu**	
4.1	Printing camp details 1. Input choice (1)	All the information on a camp will be printed
4.2	Editing camp details 1. Input choice (2)	Staff can edit any of their camp information
4.3	Deleting camp 1. Input choice (3)	Staff can delete any of their camps
4.4	Get all attendees 1. Input choice (4)	Staff can get all the names of their attendees
4.5	Get all committees 1. Input choice (5)	Staff can get all the names of their committees
4.6	Toggle visibility of camp 1. Input choice (6)	Staff enable or disable the visibility of their camps to any students
4.7	View all suggestion 1. Input choice (7)	Staff is able to see the suggestions made by committees on their camp
4.8	Approve suggestion 1. Input choice (8)	Staff is able to approved suggestions made by committees on their camp, thus adding 1 point
4.9	Viewing enquiries 1. Input choice (9)	Content of that student's enquiry will be printed
4.10	Replying enquiries 1. Input choice (10)	Staff can reply to any of the student's enquiry
4.11	Generate enquiry report 1. Input choice (12)	Staff can see all the enquiries by the students and their replies to it
4.12	Generate report of camp details 1. Input choice (11) 2. Choose the type of filter 3. Enter filter input	1. Committees can generate report of the camp details, no filter 2. Committees can generate report of the camp details, with filter on to search for specific **camp name, committee member or location**

4.13	Exiting main camp menu 1. Input choice (7)	Exit from camp menu back into the user menu
5	**Student Camp Menu (Also contains function 4.9, 4.13)**	
5.1	Printing camp details 1. Input choice (1)	All the information on a camp except its visibility and attendees will be printed
5.2	Creating enquiries 1. Input choice (3)	Student can enter and submit their enquiry
5.3	Editing enquiries 1. Input choice (4) 2. Input the enquiry number to edit	1. Student can edit their enquiries 2. Otherwise, if not created, they cannot edit.
5.4	Deleting enquiries 1. Input choice (5) 2. Input the enquiry number to delete	1. Student can delete their enquiries if they have created 2. Otherwise, if not created, they cannot delete
5.5	Withdraw from camp 1. Input choice (6)	1. Camp attendee will withdraw from camp 2. Camp committees are not allowed to withdraw from camp
6	**Committee Camp Menu (Also contains function 4.1, 4.9-4.11, 4.13)**	
6.1	Printing of current points 1. Input choice (2)	Print that committee member's current number of points
6.2	Submitting of suggestion 1. Input choice (5)	Committees can enter and sumit their suggestion
6.3	Viewing of suggestion 1. Input choice (6)	Content of that committee's suggestion and status (whether its replied) will be printed
6.4	Editing of suggestion 1. Input choice (7) 2. Input the suggestion number to edit	1. Committees can edit their suggestion if created 2. Otherwise, if not created, they cannot edit.
6.5	Deleting of suggestion 1. Input choice (8) 2. Input the suggestion number to delete	1. Committees can delete their suggestion if created 2. Otherwise, if not created, they cannot perform deletion.
6.6	Generating of report of details	1. Committees can generate report of the

	1. Input choice (9) 2. Choose the type of filter 3. Enter filter input	camp details, no filter 2. Committees can generate report of the camp details, with filter on **to search for specific attendees or committees.**

Part D: Reflection

Difficulties encountered and way to conquer in our OODP project

1. Data type checking when executing functions - implementing SOLID design principle
2. File organization - splitting to different packages to improve readability
3. Frequent input checking - create utility functions in a folder, making them reusable

Knowledge learnt from this course and further improvement suggestions

SC2002 Object Oriented Design & Programming

1. Create a driver program to test out the updated codes will vastly increase efficiency when debugging, instead of manually input all the test cases
2. Drafting UML diagram before starting implement the functions helps in project organization