SC2002: Object Oriented Design Project - CAMs

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 the 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
Lim Zhi Yong (U2222162A)	SC2002	SCSA	[illegible] **25/11/2023**
Trisha Bankata Mishra (U2222358J)	SC2002	SCSA	[illegible] **25/11/2023**
Sam Ye Zhi (U2220290D)	SC2002	SCSA	[illegible] **25/11/2023**
Trakantannarong Chindanai (U2223461D)	SC2002	SCSA	[illegible] **25/11/2023**

1. Introduction:

The main goal of this project is to demonstrate the Object-Oriented concepts that we have learnt in this module by implementing a Camp Application and Management System (CAMs). The concepts include both code wise, as well as good coding practices and SOLID principles. In our project, we have followed the requirements strictly, as well as included additional features to make the user interface and user experience more pleasant.

2. Design Considerations:

We kept in mind various SOLID principles while coding to allow the code to be more readable, understandable, as well as flexible.

2.1 Single Responsibility Principle:

The Single Responsibility Principle states that there should never be more than one reason for a class to change, so each class should only take on one responsibility. Consequently, we ensured that the attributes and methods defined in each class were for the purpose of fulfilling their sole responsibilities, achieving high cohesion in each class. For example, the CampDeletor class only has one responsibility, which is to successfully remove the camp information.

2.2 Open Closed Principle:

The Open-Closed Principle states that a module should be open for extension but closed for modification. This is to prevent us from modifying existing code, potentially leading to a need to change more codes that depend on this one change. To achieve this, our design incorporates the use of interfaces, the concept of polymorphism and method overriding. For instance, as seen in CampDataManager, whereby it provides 2 abstract methods, namely load and save, that are to be overridden by the subclasses subsequently. As we have many different sets of data such as enquiries, suggestions, and camp, if new data details are to be included into the system such as reporting of bugs, we can create a new class that extends this interface, and have it implement the abstract methods, making it easy to extend this code onto the new classes without much changes to the existing code. This reduces the coupling between the different Java files that we have, ensuring extensibility as well as maintainability.

2.3 Liskov Substitution Principle:

The Liskov Substitution Principle states that a user of a base class should continue to function properly if a derivative of that base class is passed to it. This means that an object of a superclass has to be replaceable by objects of its subclasses without changing the behavior of the program. The DataViewer interface is implemented by EnquiryViewerStudent and EnquiryViewerStaff which displays the principle. Instances of EnquiryViewerStudent and EnquiryViewerStaff can be used anywhere DataViewer is expected. Another demonstration of the principle is the superclass User and the subclasses Staff and Student. Methods where User is expected to be passed can be substituted by Student or Staff instead, displaying the Liskov Substitution Principle. This is due to the subclasses being able to provide the appropriate function to serve its purpose, with higher capability than the base class. The subclasses are also coded in a way that they will not result in any exceptions or error messages that the superclass does not create. Additionally, we ensured that the subclasses do not expect anything more than the superclass, abiding by the principle of "expecting no more and providing no less".

2.4 Interface-Segregation Principle:

In brief, this principle states that one must not depend on interfaces that are not being used. This essentially means that we had to break down larger and complex interfaces into smaller, precise and efficient interfaces to align with this principle. For example, we used various interfaces such as CampDataManager, EnquiryDataManager, DataViewer etc. We made sure all these interfaces served an integral purpose to the code and were being used. CampDataManager for example, saves and loads the CampExcelData. Thus, it is clearly an integral part of the code.

2.5 Dependency Inversion Principle:

The Dependency Inversion Principle states that high level modules should not depend upon low level modules, and that both should depend upon abstractions only. This means the high level model can simply be reused because the high and low level are independent. In our project, each Student and Staff hold abstract classes and interfaces for system management. For example, Student has CampDataManager, DataViewer, EnrollmentManager, MessageUploader and ReportGenerator which are all interfaces and abstract classes. Whenever we change the functionality of subclasses extending from these interfaces and abstract classes, we don't need to modify Student and Staff. Thus, this will reduce the coupling between classes and allow our codebase to adapt to changes.

3. Object Oriented Approach:

3.1 Abstraction:

Abstraction is the process of simplifying complex systems by modeling classes based on the essential properties and behaviors relevant to the application, while hiding unnecessary details. For example, CampManager is an abstract class. In abstract classes, the implementation details of various methods are left to the details. This helps us create a unified way to interact with different views without worrying about their specific implementations.

3.2 Encapsulation / Information Hiding:

The encapsulation principle is evident, with classes encapsulating data (attributes) and behavior (methods). For instance, in the User class, attributes like userID, name, and faculty are private, encapsulating the internal state of a user. Access to these attributes is provided through public methods, ensuring controlled access and modification. The encapsulation principle is evident throughout the code, especially in classes like User, Camp, and CampInfo. Private attributes are accessed and modified through public methods, maintaining control over the internal state. Encapsulation helps with security concerns throughout the code.

3.3 Inheritance:

Inheritance is a mechanism that allows a new class (subclass/derived class) to inherit the properties and behaviors of an existing class (superclass/base class). It promotes code reuse and establishes a hierarchy of classes. This can be seen in a lot of the code. For example, Camp inherits CampInfo. CampCreator extends CampManager as well. There are many other such instances in the code.

3.4 Polymorphism:

Polymorphism in Java and object oriented programming is the ability for one method to have different independent implementations. In other words, the classes that inherit a method from a superclass or interface will all have the same method, but the method will have different implementations. This can be seen in our EnquiryViewerStaff and SuggestionViewerStaff classes. While both of them implement the interface DataViewer which contains the method ViewData, there is no implementation of the ViewData in DataViewer. Instead, there are individual, different implements of ViewData in EnquiryViewerStaff and SuggestionViewerStaff. When the program is run and the method ViewData is called, each class responds differently to the call for ViewData and only then is the decision for which method implementation to use made.

4. Assumptions Made:

This program is coded with the assumption that this program will not be used in an online context whereby multiple users will be accessing the data at the same time. The issue with having multiple users at the same time is the issue of mutual exclusivity which may lead to data inconsistency.

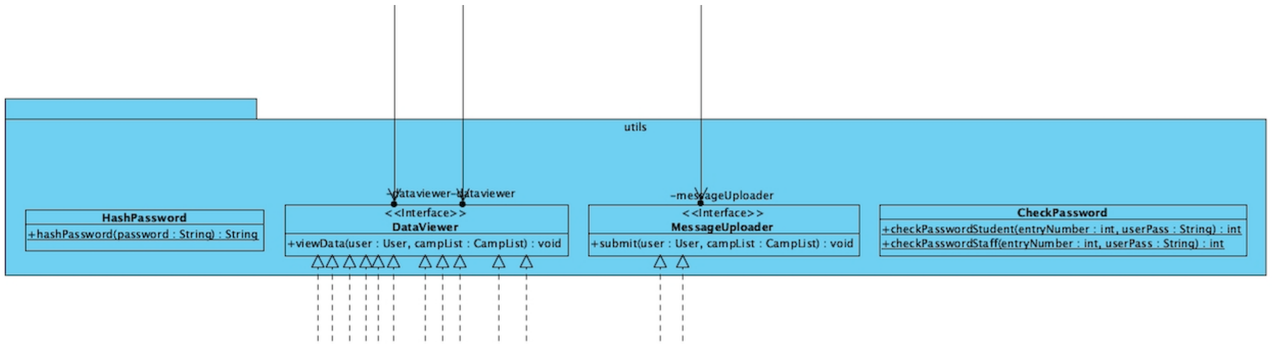
The program is also made with the assumption that there will not be an extremely large amount of data that must be saved and loaded. Hence, binary search was not implemented since the search time for students inside the lists are negligible even with a linear search.

Another assumption made is that the excel sheet containing the information will not be accessible to normal users and thus will not be open at the same time as the data is being edited through our program, which can lead to an error.

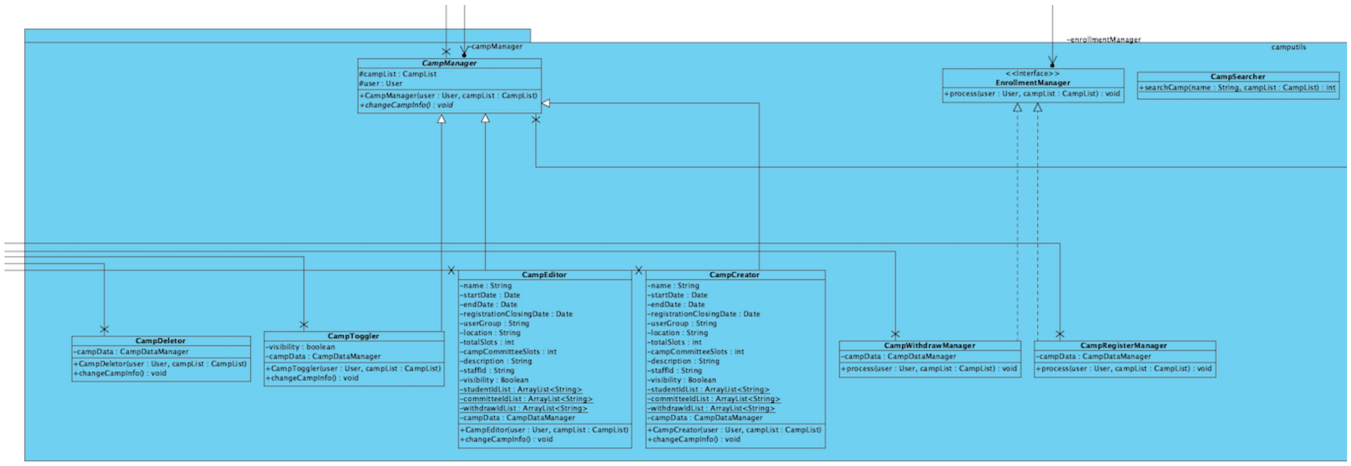
5. UML Diagram:

Please refer to UMLDiagram.jpg for the full UML Diagram.

Below, I have selected portions of the UML Diagram that showcases the main bulk of our functions. As seen in the DataViewer interface, there are many realizations made. This represents our attempt in fulfilling the SOLID principles.



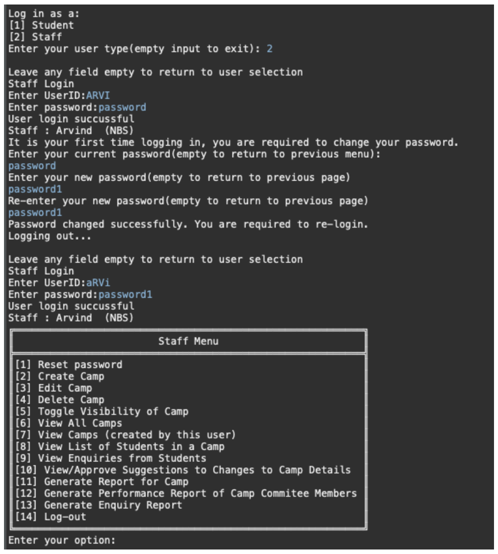
utils
-dataviewer
-dataviewer
-messageUploader
HashPassword
+hashPassword(password : String) : String
<<Interface>>
DataViewer
+viewData(user : User, campList : CampList) : void
<<Interface>>
MessageUploader
+submit(user : User, campList : CampList) : void
CheckPassword
+checkPasswordStudent(entryNumber : int, userPass : String) : int
+checkPasswordStaff(entryNumber : int, userPass : String) : int



-campManager
-enrollmentManager
camputils
CampManager
#campList : CampList
#user : User
+CampManager(user : User, campList : CampList)
+changeCampInfo() : void
<<Interface>>
EnrollmentManager
+process(user : User, campList : CampList) : void
CampSearcher
+searchCamp(name : String, campList : CampList) : int
CampDeletor
-campData : CampDataManager
+CampDeletor(user : User, campList : CampList)
+changeCampInfo() : void
CampToggler
-visibility : boolean
-campData : CampDataManager
+CampToggler(user : User, campList : CampList)
+changeCampInfo() : void
CampEditor
-name : String
-startDate : Date
-endDate : Date
-registrationClosingDate : Date
-userGroup : String
-location : String
-totalSlots : int
-campCommitteeSlots : int
-description : String
-staffId : String
-visibility : Boolean
-studentIdList : ArrayList<String>
-committeeIdList : ArrayList<String>
-withdrawIdList : ArrayList<String>
-campData : CampDataManager
+CampEditor(user : User, campList : CampList)
+changeCampInfo() : void
CampCreator
-name : String
-startDate : Date
-endDate : Date
-registrationClosingDate : Date
-userGroup : String
-location : String
-totalSlots : int
-campCommitteeSlots : int
-description : String
-staffId : String
-visibility : Boolean
-studentIdList : ArrayList<String>
-committeeIdList : ArrayList<String>
-withdrawIdList : ArrayList<String>
-campData : CampDataManager
+CampCreator(user : User, campList : CampList)
+changeCampInfo() : void
CampWithdrawManager
-campData : CampDataManager
+process(user : User, campList : CampList) : void
CampRegisterManager
-campData : CampDataManager
+process(user : User, campList : CampList) : void

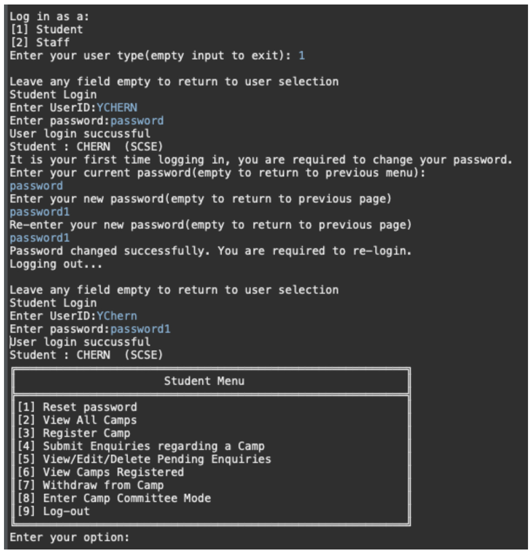
6. Testing and Result:

Log in as Student



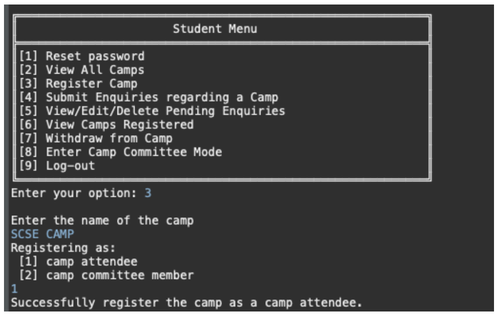
```
Log in as a:
[1] Student
[2] Staff
Enter your user type(empty input to exit): 2

Leave any field empty to return to user selection
Staff Login
Enter UserID:ARVI
Enter password:password
User login succussful
Staff : Arvind  (NBS)
It is your first time logging in, you are required to change your password.
Enter your current password(empty to return to previous menu):
password
Enter your new password(empty to return to previous page)
password1
Re-enter your new password(empty to return to previous page)
password1
Password changed successfully. You are required to re-login.
Logging out...

Leave any field empty to return to user selection
Staff Login
Enter UserID:aRVi
Enter password:password1
User login succussful
Staff : Arvind  (NBS)
                    Staff Menu
[1] Reset password
[2] Create Camp
[3] Edit Camp
[4] Delete Camp
[5] Toggle Visibility of Camp
[6] View All Camps
[7] View Camps (created by this user)
[8] View List of Students in a Camp
[9] View Enquiries from Students
[10] View/Approve Suggestions to Changes to Camp Details
[11] Generate Report for Camp
[12] Generate Performance Report of Camp Commitee Members
[13] Generate Enquiry Report
[14] Log-out
Enter your option:
```

Log in as Staff



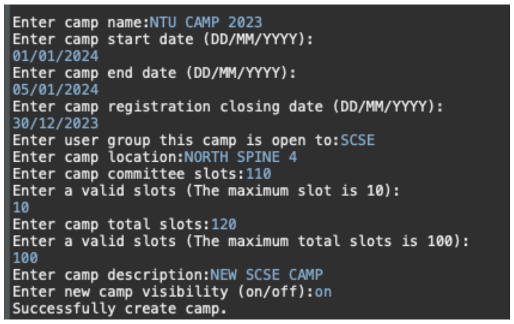
```
Log in as a:
[1] Student
[2] Staff
Enter your user type(empty input to exit): 1

Leave any field empty to return to user selection
Student Login
Enter UserID:YCHERN
Enter password:password
User login succussful
Student : CHERN  (SCSE)
It is your first time logging in, you are required to change your password.
Enter your current password(empty to return to previous menu):
password
Enter your new password(empty to return to previous page)
password1
Re-enter your new password(empty to return to previous page)
password1
Password changed successfully. You are required to re-login.
Logging out...

Leave any field empty to return to user selection
Student Login
Enter UserID:YChern
Enter password:password1
User login succussful
Student : CHERN  (SCSE)
                    Student Menu
[1] Reset password
[2] View All Camps
[3] Register Camp
[4] Submit Enquiries regarding a Camp
[5] View/Edit/Delete Pending Enquiries
[6] View Camps Registered
[7] Withdraw from Camp
[8] Enter Camp Committee Mode
[9] Log-out
Enter your option:
```

Joining a Camp



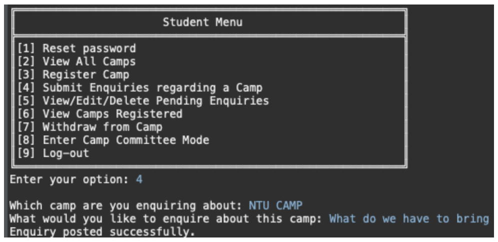
```
                    Student Menu
[1] Reset password
[2] View All Camps
[3] Register Camp
[4] Submit Enquiries regarding a Camp
[5] View/Edit/Delete Pending Enquiries
[6] View Camps Registered
[7] Withdraw from Camp
[8] Enter Camp Committee Mode
[9] Log-out
Enter your option: 3

Enter the name of the camp
SCSE CAMP
Registering as:
 [1] camp attendee
 [2] camp committee member
1
Successfully register the camp as a camp attendee.
```

Creating a Camp



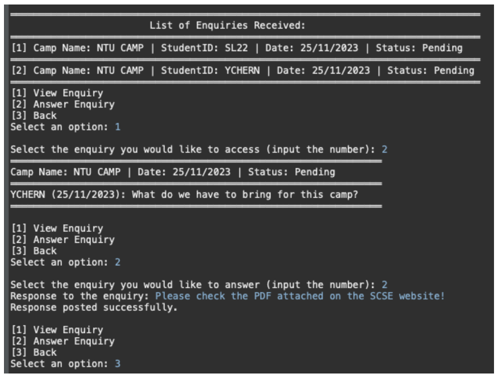
```
Enter camp name:NTU CAMP 2023
Enter camp start date (DD/MM/YYYY):
01/01/2024
Enter camp end date (DD/MM/YYYY):
05/01/2024
Enter camp registration closing date (DD/MM/YYYY):
30/12/2023
Enter user group this camp is open to:SCSE
Enter camp location:NORTH SPINE 4
Enter camp committee slots:110
Enter a valid slots (The maximum slot is 10):
10
Enter camp total slots:120
Enter a valid slots (The maximum total slots is 100):
100
Enter camp description:NEW SCSE CAMP
Enter new camp visibility (on/off):on
Successfully create camp.
```

Student Posting an Enquiry



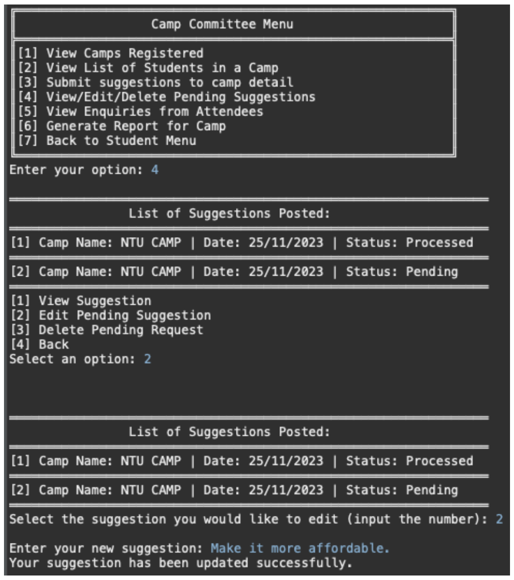
```
                    Student Menu
[1] Reset password
[2] View All Camps
[3] Register Camp
[4] Submit Enquiries regarding a Camp
[5] View/Edit/Delete Pending Enquiries
[6] View Camps Registered
[7] Withdraw from Camp
[8] Enter Camp Committee Mode
[9] Log-out
Enter your option: 4

Which camp are you enquiring about: NTU CAMP
What would you like to enquire about this camp: What do we have to bring
Enquiry posted successfully.
```

Staff Answering Enquiry



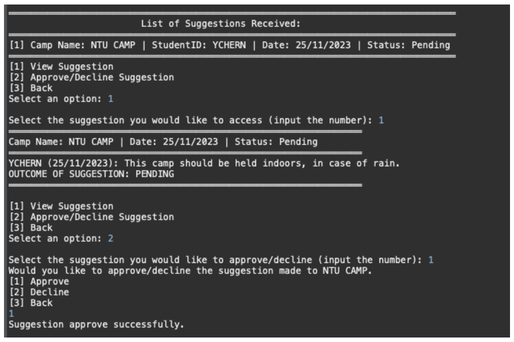
```
                    List of Enquiries Received:
[1] Camp Name: NTU CAMP | StudentID: SL22 | Date: 25/11/2023 | Status: Pending
[2] Camp Name: NTU CAMP | StudentID: YCHERN | Date: 25/11/2023 | Status: Pending
[1] View Enquiry
[2] Answer Enquiry
[3] Back
Select an option: 1

Select the enquiry you would like to access (input the number): 2
Camp Name: NTU CAMP | Date: 25/11/2023 | Status: Pending
YCHERN (25/11/2023): What do we have to bring for this camp?

[1] View Enquiry
[2] Answer Enquiry
[3] Back
Select an option: 2

Select the enquiry you would like to answer (input the number): 2
Response to the enquiry: Please check the PDF attached on the SCSE website!
Response posted successfully.

[1] View Enquiry
[2] Answer Enquiry
[3] Back
Select an option: 3
```

Camp Committee Editing Suggestion



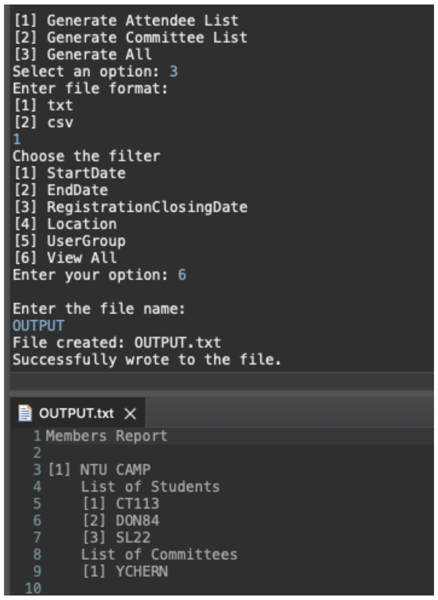
```
                    Camp Committee Menu
[1] View Camps Registered
[2] View List of Students in a Camp
[3] Submit suggestions to camp detail
[4] View/Edit/Delete Pending Suggestions
[5] View Enquiries from Attendees
[6] Generate Report for Camp
[7] Back to Student Menu
Enter your option: 4

              List of Suggestions Posted:
[1] Camp Name: NTU CAMP | Date: 25/11/2023 | Status: Processed
[2] Camp Name: NTU CAMP | Date: 25/11/2023 | Status: Pending
[1] View Suggestion
[2] Edit Pending Suggestion
[3] Delete Pending Request
[4] Back
Select an option: 2


              List of Suggestions Posted:
[1] Camp Name: NTU CAMP | Date: 25/11/2023 | Status: Processed
[2] Camp Name: NTU CAMP | Date: 25/11/2023 | Status: Pending
Select the suggestion you would like to edit (input the number): 2

Enter your new suggestion: Make it more affordable.
Your suggestion has been updated successfully.
```

Staff Approving Suggestions



```
                    List of Suggestions Received:
[1] Camp Name: NTU CAMP | StudentID: YCHERN | Date: 25/11/2023 | Status: Pending
[1] View Suggestion
[2] Approve/Decline Suggestion
[3] Back
Select an option: 1

Select the suggestion you would like to access (input the number): 1
Camp Name: NTU CAMP | Date: 25/11/2023 | Status: Pending
YCHERN (25/11/2023): This camp should be held indoors, in case of rain.
OUTCOME OF SUGGESTION: PENDING

[1] View Suggestion
[2] Approve/Decline Suggestion
[3] Back
Select an option: 2

Select the suggestion you would like to approve/decline (input the number): 1
Would you like to approve/decline the suggestion made to NTU CAMP.
[1] Approve
[2] Decline
[3] Back
1
Suggestion approve successfully.
```

Staff Generating Report for Camps



```
[1] Generate Attendee List
[2] Generate Committee List
[3] Generate All
Select an option: 3
Enter file format:
[1] txt
[2] csv
1
Choose the filter
[1] StartDate
[2] EndDate
[3] RegistrationClosingDate
[4] Location
[5] UserGroup
[6] View All
Enter your option: 6

Enter the file name:
OUTPUT
File created: OUTPUT.txt
Successfully wrote to the file.
```

```
Members Report

[1] NTU CAMP
    List of Students
     [1] CT113
     [2] DON84
     [3] SL22
    List of Committees
     [1] YCHERN

```

7. Reflection:

This project has given us a deeper look into OOP principles that are involved in any piece of code. There were also a few hurdles we faced, which we pondered over together as a group so as to approach the solution effectively. There were various takebacks and things we all learnt.

- Firstly, we learnt the importance of having a UML diagram that accurately depicts the classes and their respective relationships. Without this, coding a large program often became difficult, so we made it our first priority to make a well designed UML diagram keeping in mind different considerations. We often had to revisit our diagram anytime we found our code did not adhere to it. The UML diagram also helped divide the work easily, with each member tackling different parts of the diagram on their own.
- We also realized how effective comments can really change the readability of our program. Earlier, our program was devoid of comments and it did not make much sense to the reader as it was just code. So, we modified it, including various comments to improve the readability and understandability of it.
- To ensure that our program could run smoothly without any exceptions or errors, we need to be able to handle edge cases properly. It was imperative that we take exception handling into account, by repeated testing. This made sure the code worked properly in every scenario.
- When refining the code, it was necessary to find ways to avoid tight-coupling between classes as to reduce dependencies between classes, thus improving the reusability of the code and adhering to the SOLID principles.
- We learnt the use of external libraries such as Apache POI in our projects, to enable us to edit excel sheets.

8. Difficulties Faced:

1. The concepts of object-oriented programming and SOLID design are new concepts for us. It takes a lot of time and revision to adapt the concepts to our code as we did the sequential coding in previous courses.
2. Some of us were unfamiliar with collaborating on Github as it was a somewhat new platform for us.
3. We had to figure out how other members implemented classes and methods in order to build on them.
4. We had to figure out how to download and use libraries such as Apache Poi in our project and add the JAR files to the classpath of the project.
5. Learning how to use the additional libraries needed for this project was difficult and we needed to understand the things the libraries provided us with such as workbook and cells

9. Solutions:

1. We have reviewed lecture examples and online learning resources on youtube to gain familiarity with coding in object-oriented design. This helps us to apply new knowledge to our code, especially in reasonable and effective class splittings.
2. We referred to the same external teaching materials which allowed us to employ the use of GitHub, simplifying the process of merging codes.
3. Proper documentation and commenting of functions helped us to understand what each of the classes and methods do.
4. We shared the know-how on how to download and set up those libraries within our group to make sure we did not spend too much time trying to set up those libraries individually.
5. When faced with questions about the available tools from the libraries, we checked online through the homepages of the libraries to verify the usage and syntax available. Those libraries typically provide instructions on how to use them on their homepage.

10. New Features:

In this project, we went above and beyond to try and create a system that closely mimics existing CAMs and other platforms.

1. We understand the importance of security especially when privacy concerns are on the rise. Thus, it is crucial that we encrypt the sensitive information on the database, such that if any intruders were to gain access to the database, the consequences would be mitigated.
2. We made use of external libraries (UUID) to generate random identity numbers for each inquiry so that we can access this particular entry to respond to it. This is the most logical way to refer to an inquiry as it would be difficult to assign a name to it.
3. Additionally, we understand that as the list of camps continues to grow, it will be more user friendly to be able to filter the camps that students are interested in. Thus, we also implemented the filters in the student's view camps function, to allow them to find the list of camps he/she would be interested in.

11. Further Suggestions and Features:

1. Rather than having to loop through the list of camps and locate which one contains the particular user ID, we could also store this information in the user's entry within the excel sheet, this will allow us to get the related camp information with regards to a user in a quicker manner. This is a trade off between time and space complexity. However, on a bigger scale, this trade off would be worth it as a lot of time is saved by simply using up a few more bytes of data per user.

2. As time complexities were not the focus of this project, our code is not designed to achieve efficiency especially when it comes to searching for a particular camp. This can be further improved by using what we have learnt in the Algorithm Design and Analysis module. By using heap sort to arrange the camps according to camp ID, we can locate the camps more effectively. In order to achieve that, we first use a simple algorithm to convert the camp names inputted by any user into numeric values which will then be used to search for their respective camp IDs in the hashtable implemented. This camp ID will in turn be used in the binary search algorithm to locate the actual camp data within the sorted array. On a large scale, this will certainly be more efficient as linear search takes

O(n) time but binary search takes only O(log n). This is a great representation of how we can leverage on the knowledge points from different modules to achieve greater results and better efficiency.

3. We can alter parts of the code such that it runs on a terminal instead of an IDE. This helps improve on certain aspects such as blocking out the password as the user keys into the console. This makes the project much more secure. Taking the project one step further, There is also the probability of implementing more aesthetic UIs through libraries or frameworks from Java.

4. We also considered implementing a notification system. For instance, when a new enquiry or suggestion is posted, a notification can alert the respective in-charge. This will improve the overall user experience. However due to time constraints, we were not able to implement it.

5. The Singleton Pattern is a design pattern that ensures a class has only one instance and provides a global point of access to that instance. It is used when exactly one object is needed to coordinate actions across the system. This is achieved by declaring a private static variable (instance) within each class and providing a public static method (getInstance()) for accessing this instance. The private constructor of each class prevents external instantiation, and the static method checks whether an instance already exists. If no instance is present, a new one is created. This design choice is particularly useful for centralizing and managing user and camp-related operations. It ensures that all parts of the application interact with a consistent and singular instance of these controllers, promoting better organization, modularity, and avoiding unnecessary resource overhead. This could have also been implemented in the project, as it is an important and useful feature.

6. We can improve the extensibility of the code by introducing abstract classes of Camp, Enquiry, and Suggestion objects. These classes will be passed as lists through the higher modules instead of CampList, EnquiryList, and SuggestionList objects. For instance, if

new types of camp are introduced, the camp can extend from the camp abstract class which will be used through the modules without changing the original code.