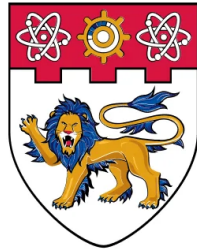


SC2002 : OBJECT ORIENTED DESIGN AND PROGRAMMING

AY22/23 SEMESTER 1 GROUP ASSIGNMENT








**NANYANG
TECHNOLOGICAL
UNIVERSITY**
SINGAPORE

Declaration of Original Work for SC/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
Gillbert Susilo Wong (U2220602C)	SC2002	SCSB	 11/25/2023
Juan Frederick (U2220075B)	SC2002	SCSB	 11/25/2023
Karl Devlin Chau (U2220525H)	SC2002	SCSB	 11/25/2023
Pascal Pandey (U2220371G)	SC2002	SCSB	 11/25/2023
Nguyen Thi Quynh Trang (U2020022C)	SC2002	SCSB	 11/25/2023

1. Design Considerations

1.1. Approach

CAMS (Camp Application and Management System) is a Java Command Line Interface (CLI) application. It is made for staff and students to manage, view and register for camps within NTU. The app implements loosely coupled classes in multiple distinct packages. This makes our system easy to maintain, improve, and extendable.

It utilizes entity, controller, and boundary (ECB) class stereotypes. Users of the app access through menus which are boundary classes. It requests data from managers, which acts as controller classes. Managers implement logic and perform error checking and exception catching to guarantee that the app works properly. The data is fetched from entity classes, which store the data. Abstractions demonstrated enable easy modification and extension.

1.2. Assumptions

1. New users cannot be created in the application, it can only be done by editing student_list.xlsx or staff_list.xlsx.
2. Once the program exits, all existing data must be saved to a .xlsx file and if the program is started again all data in the .xlsx files must be loaded into the application.
3. Once a camp has at least one registered student it can no longer be deleted.
4. Camp start date, end date, and registration deadline cannot be edited.
5. Suggestions that are approved by staff do not change the camp details automatically.

1.3. SOLID Design Principles

1.3.1. Single Responsibility Principle

The single responsibility principle ensures that a class is never overloaded with responsibilities. Each class should do at most one operation and serve one purpose. This is to ensure that each class is as small as possible, making the code not only easier to read and understand, but also more maintainable. One example in our code where this is most clear is in the various FilterStrategies. Each filter strategy is implemented by a separate class

allowing us to add more when needed and chain multiple filter strategies on top of each other for multi-filter searching. Not only this, but the atomic nature of each filter strategy also allows us to perform unit tests on each strategy easily. The automated tests ensure that our filter strategies will always work across various changes to the code base.

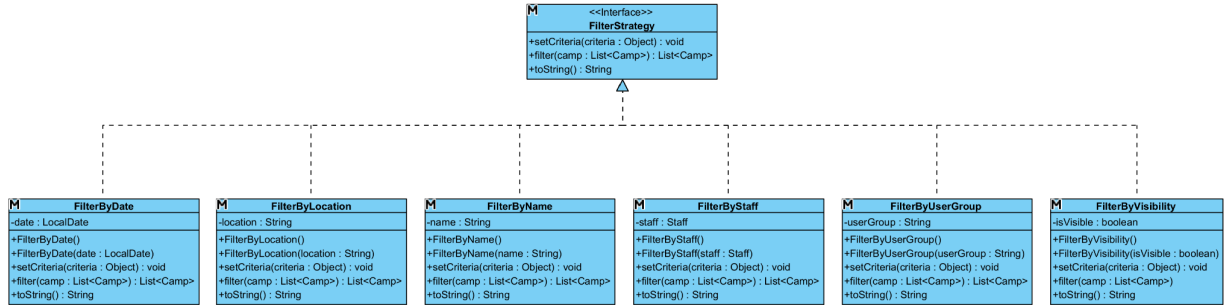


Figure 1. FilterStrategy interface and its implementations

1.3.2. Open-Close Principle

The open-close principle states that a class should be close to modification but open to extension. It means that new functionalities should be added without modifying previous classes. This can be done through polymorphism and inheritance. One example we have in our code is in Staff and Student inheriting from User. The User class provides basic attributes that all Users in the system should have like userID, passwordHash, and faculty, while the Staff and Student class implements specific attributes that are used only by them like campsCreated for Staff, which stores the camps created by a staff, and campsRegistered for Student, which stores the camps a student has registered themselves into. By creating a new class which inherits from the parent User class, we could create new user classes (like Administrator or Moderator) in the future without changing any code inside User, Student, or Staff. This ensures the extensibility of our code. New types of users can be added easily without modifying and possibly introducing bugs in the rest of the functionalities of the application.

1.3.3. Liskov Substitution Principle

The Liskov Substitution Principle states that subclasses that inherit from a superclass should be able to do all the functionalities of its superclass. That is objects of the superclass should be replaceable with its subclass without breaking the application. One example in our application is with Repliable. The Repliable superclass in our application makes sure that each Repliable object, which can be a Suggestion or Enquiry, properly stores the Student object that created it. Because Repliable is replaceable by Suggestion and Enquiry, we can generalize EnquiryEditor and SuggestionEditor into a new superclass RepliableEditor. The idea is that the RepliableEditor can interface with Repliable and SuggestionEditor and EnquiryEditor is free to downcast the Repliable parameter into each of its subclass, Suggestion or Enquiry, without the application breaking as all Repliable functionality is available in Suggestion and Enquiry. This not only promotes open close principle in RepliableEditor but also allows our code to be flexible and consistent. Any calls to Repliable are guaranteed to be able to be served by its subclasses, which means that its superclass, Repliable, can be used in other classes flexibly without the application breaking.

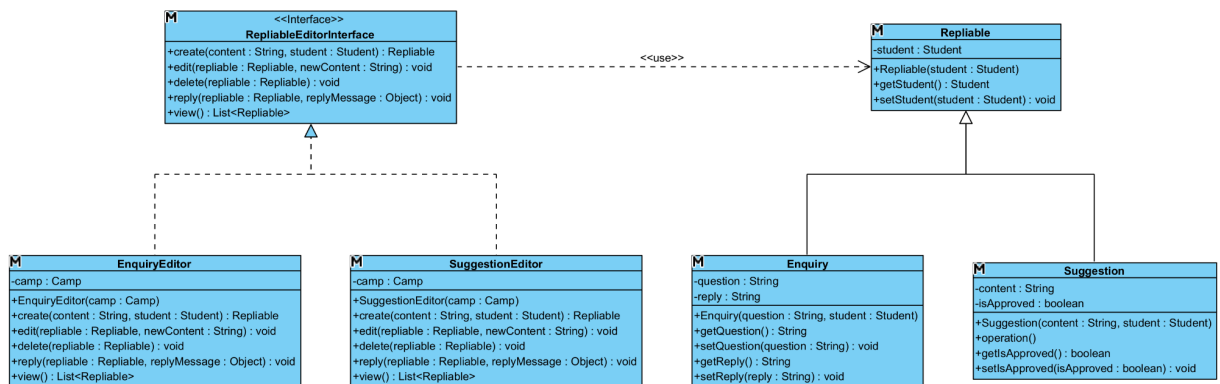


Figure 2. Repliable and its subclasses

1.3.4. Interface Segregation

Interface segregation states that when using interfaces, each interface should be as small as possible. Many specific interfaces are generally better than one general interface and a class

should implement many interfaces as opposed to one large interface. One example in our codebase are display primitives like Alert, Form, and Selection. All of these displays implement the Displayable and ItemAction interface, which enforces two functionalities on each display. It must be able to display alerts, input forms, and choices to the console and do actions based on user input. By separating these 2 interfaces, we would be able to create new display primitives which only display or only receive user input. This keeps our code extensible and flexible as it supports new functionalities that can only implement one of the interfaces and not both.

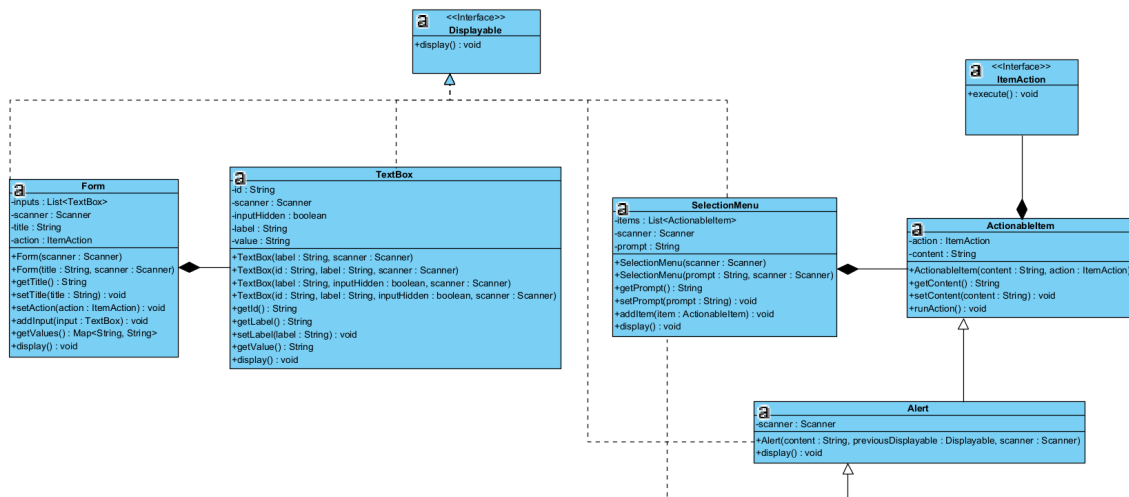


Figure 3. Use of Displayable and ItemAction interface to maintain interface segregation

1.3.5. Dependency Injection

The dependency injection principle states that high level modules should not depend on low level ones. Classes should not depend on other concrete classes, but on higher level interfaces instead. This allows for loosely coupled code as interfaces don't change often, so on extension or modification of code, minimal changes are needed to be made. We have several examples of this in our codebase, FilterStrategy and RepliableEditor interface. All the filter strategies depend on the FilterStrategy interface and both suggestion and enquiry editors depend on the RepliableEditor interface. When we call filters, for example, we don't call the concrete classes, but we call the interface. This decouples the concrete classes from each other and makes it more modular. It also allows our code to be much more flexible and

extensible. If in the future we need to implement new filters or new repliables, old filters and repliable editors don't need to be changed as they don't depend on each other. Their dependencies are injected from a higher level outside module so we will not need to edit old code, instead we can just create a new class, with its own new functionality, which depends on the same interface. Any calls to this new class would be served through the higher level interface as well, so we don't even have to modify any old classes that will use our new concrete class.

1.4. APIE Design Principles

1.4.1. Abstraction

The menus only access the required information through their respective functions. Handling only the information it needs and abstracting methods implementations keeps the system simple and secure.

1.4.2. Polymorphism

In the User abstract class, `getCamps()`, `addCamp()`, and `removeCamp()` are abstract methods. The implementations are different for the subclass of User, namely Student and Staff. For example, Students can add camp to their camp attendee list, while Staff can create a camp. Each subclass creates its own implementation of the method.

1.4.3. Inheritance

The entities Enquiry and Suggestion inherit from the superclass Repliable. Both of them store the student name that submitted the respective repliable.

1.4.4. Encapsulation and Information Hiding

Encapsulation protects an object's private data. From the outside, the data is only accessible through the respective object's public getter and setter. The user class contains sensitive attributes such as name, faculty, and, most importantly, password. To ensure that the attributes are not directly accessible and not directly mutable, it is set to private.

1.5. Other Design Patterns

1.5.1. Singleton Pattern

The singleton pattern is mostly used in our controllers. Controllers are instantiated at the start of the app and closed or destructured on program exit. The singleton pattern allows for the same controllers to be accessed globally throughout the application. This ensures that, at any one time, only one controller holds the data of the entity classes it is responsible for. This not only maintains data consistency throughout the application, but also keeps the program memory efficient as only one controller of each type is instantiated throughout the program.

1.5.2. Component Based Menus

We made base classes (SelectionMenu, Form, and Alert) for our menus. Furthermore, to chain menus together, we use a singleton DisplayController (with its setNextDisplay() method), which makes sure that only one menu at a time is instantiated, saving memory, and making it easy for us to chain multiple menus together. The idea is that these base classes can be combined with each other in order to produce any type of menu we want. Furthermore, we can easily chain menus together to easily create new user flows.

```
public class LoginForm extends Form {
    /**
     * Class constructor specifying the scanner to be used to receive user input.
     * @param scanner scanner for this menu
     */
    public LoginForm(Scanner scanner) {
        super(CommonElements.getStatusBar(status:"Login") + "Login to CAMS:\n", scanner);

        addInput(new TextBox(label:"User ID", scanner));
        addInput(new TextBox(label:"Password", inputHidden:true, scanner));

        setAction(new ItemAction() {
            public void execute() {
                AuthController authController = AuthController.getInstance();
                DisplayController displayController = DisplayController.getInstance();

                String userIDInput = getValues().get("User ID");
                String passwordInput = getValues().get("Password");

                try {
                    User currentUser = authController.login(userIDInput, passwordInput);
                    if (currentUser instanceof Staff) {
                        StaffController.getInstance().login(currentUser);
                        displayController.setNextDisplay(new StaffMenu(scanner));
                    } else {
                        StudentController.getInstance().login(currentUser);
                        displayController.setNextDisplay(new StudentMenu(scanner));
                    }
                } catch (IllegalArgumentException e) {
                    displayController.setNextDisplay(
                        new LoginErrorAlert(new LoginForm(scanner), scanner));
                }
            }
        });
    }
}
```

Figure 4. View component LoginForm, responsible for getting user credentials, redirecting to the next corresponding page and throwing exceptions.

For example, the LoginForm extends Form and uses the TextBox base class for userId and password input. It then reads the user input and sets DisplayController.setNextDisplay() based on the type of user or throws an exception Alert when the userID or password is wrong. The alert is then set to display a new LoginForm once the user receives the alert.

2. UML Class Diagram

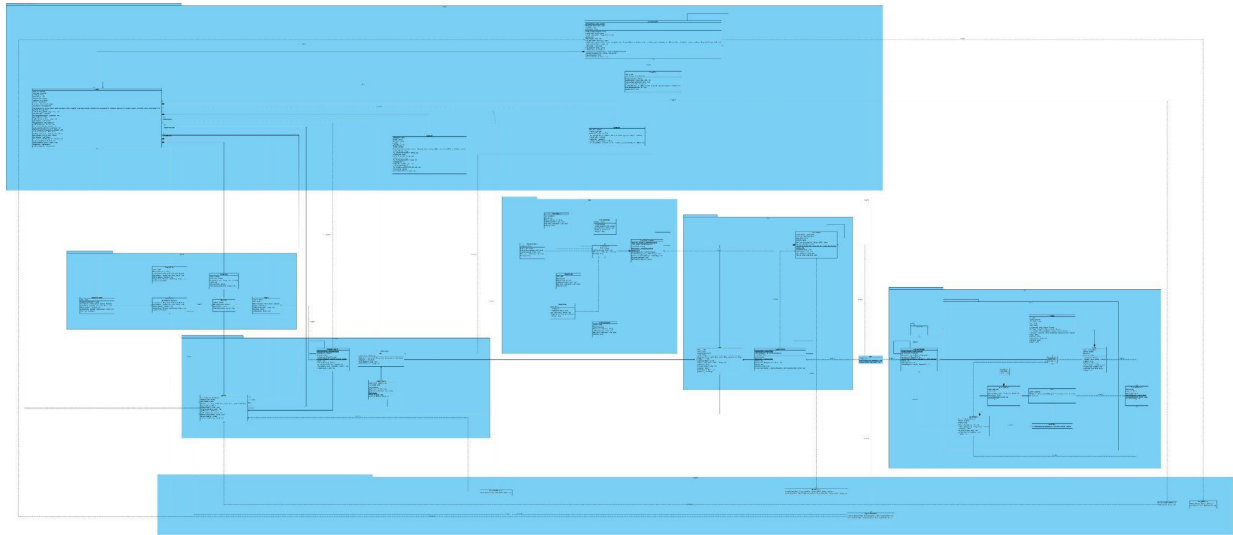


Figure 5. UML Class Diagram for CAMs

Please refer to the SC2002-CAMs.vpp and SC2002-CAMs.png files for the detailed UML diagram.

3. Testing

3.1. JUnit Test Cases

We have written JUnit test cases for our display and serializer functions. It is automated with our CI/CD pipeline and is executed on every push to GitHub. This ensures that we don't introduce new bugs to our code each time we add a new functionality or fix bugs. Furthermore, it also helps in identifying exactly which methods or functionality are causing the bugs when they do occur. All test cases can be found in the test folder of our project. The CI/CD code using GitHub actions is found in the .github folder.


```

@Test
@DisplayName("Cannot login with incorrect user ID or password")
void cannotLoginWithIncorrectUserIDorPassword() {
    provideInput(data:"1\nnonexistent\npassword\n\nUPAM\nwrongpassword\n\nUPAM\npassword\n5\n2\n");
    assertTimeoutPreemptively(Duration.ofSeconds(5), this::runApp);
    assertEndCorrectly();
}

@Test
@DisplayName("Valid user can change password")
void validUserCanChangePassword() {
    provideInput(data:"1\nUPAM\npassword\n1\npassword\n12345678\n5\n2\n");
    assertTimeoutPreemptively(Duration.ofSeconds(5), this::runApp);
    assertEndCorrectly();

    BCryptPasswordEncoder passwordEncoder = AuthController.getInstance().getPasswordEncoder();
    User user = UserController.getInstance().getUser(userID:"UPAM");
    assertTrue(passwordEncoder.matches(rawPassword:"12345678", user.getHashedPassword()));

    provideInput(data:"1\nUPAM\n\n12345678\n1\n12345678\npassword\n5\n2\n");
    assertTimeoutPreemptively(Duration.ofSeconds(5), this::runApp);
    assertEndCorrectly();
}

```

Figure 6. Example of a JUnit test case testing for login and change password.

```

Error: Failures:
Error: DisplayableIT.campCommitteeCanGenerateAttendeeList:590 expected: <true> but was: <false>
Error: DisplayableIT.campCommitteeCanGenerateCombinedStudentList:638 expected: <true> but was: <false>
Error: DisplayableIT.campCommitteeCanGenerateCommitteeList:614 expected: <true> but was: <false>
Error: DisplayableIT.staffCanGenerateAttendeeList:663 expected: <true> but was: <false>
Error: DisplayableIT.staffCanGenerateCombinedStudentList:712 expected: <true> but was: <false>
Error: DisplayableIT.staffCanGenerateCommitteeList:688 expected: <true> but was: <false>
Error: DisplayableIT.staffCanGeneratePerformanceReport:735 expected: <true> but was: <false>
[INFO]
Error: Tests run: 30, Failures: 7, Errors: 0, Skipped: 0

```

Figure 7. Example of a failed test case. It shows exactly which tests failed, allowing for easy debugging.

3.2. Manual Testing

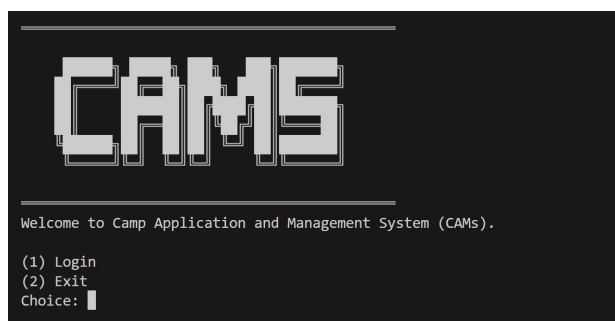


Figure 8. CAMS main menu



Figure 9. CAMS login menu

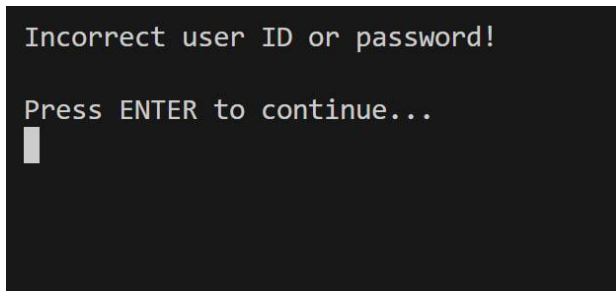


Figure 10. Failed login alert

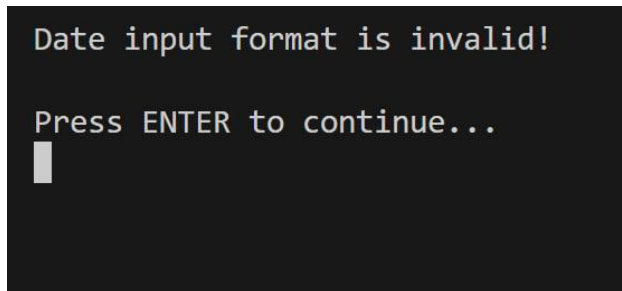


Figure 11. Invalid date alert

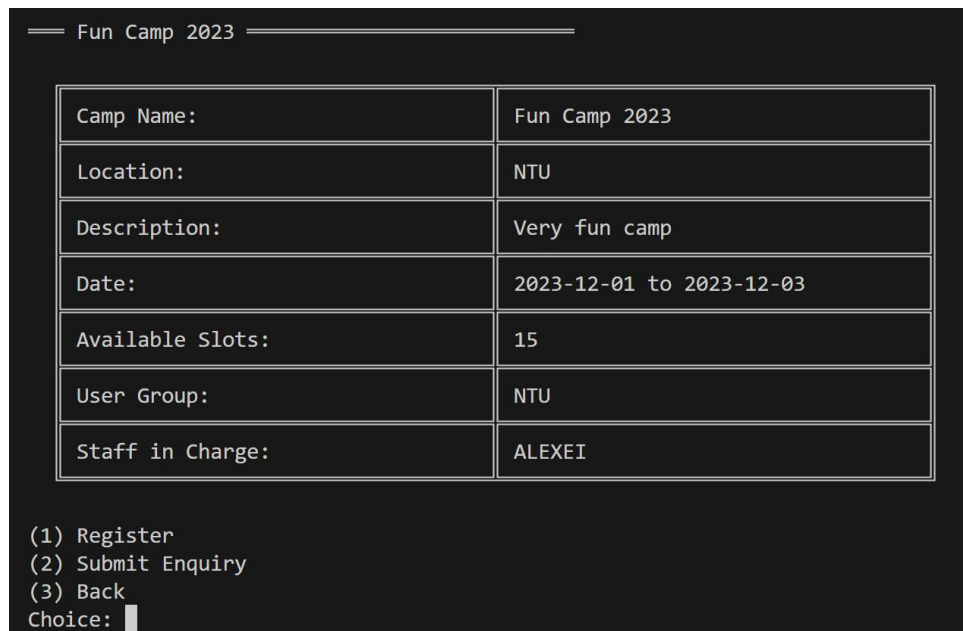


Figure 12. View camp details menu

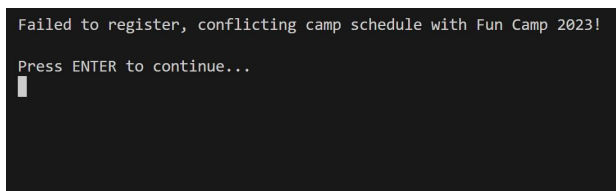


Figure 13. Clashing camp date alert

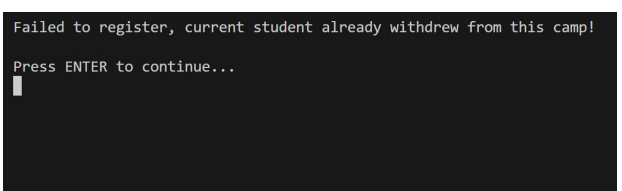


Figure 14. Unable to re-register alert

4. Reflection

4.1. Difficulties Encountered and Conquering It

We ran into a lot of problems early on in agreeing to a design for the application; what data structures to use for each attribute, and how they interact with each other. There were two considerations in each class we made: how the class interacts with other classes in the system, and which data structures to use to minimize the runtime of our application. We realized early on that a good design is important to build a maintainable, easy to understand, and extendable code base. Therefore, we took our time making careful considerations in deciding the design.

We eventually settled on the aforementioned entity, control, boundary (ECB) class stereotypes for the general design and used Maps and Sets to store data in order to minimize search time while maintaining ease of update. ECB allows for loose coupling between classes and ensures that classes only have to do as much as they need to. Furthermore, using indexed data structures like Maps and Sets not only makes for an easier development experience as searching is done through the APIs Java provides, but it also properly indexes the entity classes, which can greatly improve performance as the application scales in the future.

4.2. Knowledge Learned

We learned how to build a well-designed application that applies the SOLID and OOP principles properly. These rules are important to follow in any application design because they ensure that the code can be easily maintained and extended in the future. We also learned how to properly document our code, which allows other people who may work on our code in the future to understand the use of each function and how to add new functionalities. Lastly, we learned about how to unit test and automate those tests. It allows our code to be consistent and free of bugs. As the code base becomes more and more complex, tests are an indispensable asset as they give developers confidence that newly added features don't break old ones and that the code continues to function properly even after changes are made to it.

4.3. Further Improvements

In the future, we hope to introduce a feature where multiple users are able to use the system concurrently. This is similar to a real-world camp management system, which would allow staff

and students to update camp details, post new camps, or register for camps in real-time. Another improvement would be a graphical user interface which would be much more interactive and easy to use compared to the command line interface we currently have. Something using simple HTML, JS, and CSS would greatly improve the user experience whilst making the menu much more intuitive and easy to use.

Appendix I: Declaration on Use of GAI (Generative Artificial Intelligence) Assistance in relation to Assignment/Project (to be submitted individually even for group projects)

I Gillbert Susilo Wong (student name),
gillbert001@e.ntu.edu.sg (NTU email) honestly and sincerely make the following declaration in relation to the following course submission:

1. Name of course: Object Oriented Design and Programming
2. Course Code: SC2002
3. Instructor: Dr. Zhang Jie , Dr. Li Fang
4. Title of Assignment/Project Submission: Group Assignment CAMs

In relation to the foregoing I hereby declare that, fully and properly in accordance with the Assignment/Project Instructions I have (check where appropriate):

- i. Used GAI as permitted to assist in generating key ideas only. ☐
- ii. Used GAI as permitted to assist in generating a first text only. ☐

And/or

- iii. Used GAI to refine syntax and grammar for correct language submission only. ☐

Or

- iv. As it is not permitted: Not used GAI assistance in any way in the development or generation of this assignment or project. ☒

I also declare that I have :

- a. Fully and honestly submitted the digital paper trail required under the assignment/project instructions; and that
- b. Wherever GAI assistance has been employed in the submission in word or paraphrase or inclusion of a significant idea or fact suggested by the GAI assistant, I have acknowledged this by a footnote; and that,
- c. Apart from the foregoing notices, the submission is wholly my own work.



GILLBERT SUSILO WONG

Student Name & Signature

26 November 2023

Date

Appendix I: Declaration on Use of GAI (Generative Artificial Intelligence) Assistance in relation to Assignment/Project (to be submitted individually even for group projects)

I Juan Frederick (student name),
JUAN0012@e.ntu.edu.sg (NTU email) honestly and sincerely make the following declaration in relation to the following course submission:

1. Name of course: Object Oriented Design and Programming
2. Course Code: SC2002
3. Instructor: Dr. Zhang Jie, Dr. Li Fang
4. Title of Assignment/Project Submission: Group Assignment CAMs

In relation to the foregoing I hereby declare that, fully and properly in accordance with the Assignment/Project Instructions I have (check where appropriate):

- i. Used GAI as permitted to assist in generating key ideas only. ☐
- ii. Used GAI as permitted to assist in generating a first text only. ☐

And/or

- iii. Used GAI to refine syntax and grammar for correct language submission only. ☐

Or

- iv. As it is not permitted: Not used GAI assistance in any way in the development or generation of this assignment or project. ☒

I also declare that I have :

- a. Fully and honestly submitted the digital paper trail required under the assignment/project instructions; and that
- b. Wherever GAI assistance has been employed in the submission in word or paraphrase or inclusion of a significant idea or fact suggested by the GAI assistant, I have acknowledged this by a footnote; and that,
- c. Apart from the foregoing notices, the submission is wholly my own work.



Juan Frederick

Student Name & Signature

26 November 2023

Date

Appendix I: Declaration on Use of GAI (Generative Artificial Intelligence) Assistance in relation to Assignment/Project (to be submitted individually even for group projects)

I Karl Devlin Chau (student name),
KARL0009@e.ntu.edu.sg (NTU email) honestly and sincerely make the following declaration in relation to the following course submission:

1. Name of course: Object Oriented Design and Programming
2. Course Code: SC2002
3. Instructor: Dr. Zhang Jie, Dr. Li Fang
4. Title of Assignment/Project Submission: Group Assignment CAMs

In relation to the foregoing I hereby declare that, fully and properly in accordance with the Assignment/Project Instructions I have (check where appropriate):

- i. Used GAI as permitted to assist in generating key ideas only. ☐
- ii. Used GAI as permitted to assist in generating a first text only. ☐
- And/or
- iii. Used GAI to refine syntax and grammar for correct language submission only. ☐

Or

- iv. As it is not permitted: Not used GAI assistance in any way in the development or generation of this assignment or project. ☒

I also declare that I have :

- a. Fully and honestly submitted the digital paper trail required under the assignment/project instructions; and that
- b. Wherever GAI assistance has been employed in the submission in word or paraphrase or inclusion of a significant idea or fact suggested by the GAI assistant, I have acknowledged this by a footnote; and that,
- c. Apart from the foregoing notices, the submission is wholly my own work.



Karl Devlin Chau

Student Name & Signature

26 November 2023

Date

Appendix I: Declaration on Use of GAI (Generative Artificial Intelligence) Assistance in relation to Assignment/Project (to be submitted individually even for group projects)

I Pascalis Pandey (student name),
PASCALIS001@e.ntu.edu.sg (NTU email) honestly and sincerely make the following declaration in relation to the following course submission:

1. Name of course: Object Oriented Design and Programming
2. Course Code: SC2002
3. Instructor: Dr. Zhang Jie, Dr. Li Fang
4. Title of Assignment/Project Submission: Group Assignment CAMs

In relation to the foregoing I hereby declare that, fully and properly in accordance with the Assignment/Project Instructions I have (check where appropriate):

- i. Used GAI as permitted to assist in generating key ideas only. ☐
- ii. Used GAI as permitted to assist in generating a first text only. ☐

And/or

- iii. Used GAI to refine syntax and grammar for correct language submission only. ☐

Or

- iv. As it is not permitted: Not used GAI assistance in any way in the development or generation of this assignment or project. ☒

I also declare that I have :

- a. Fully and honestly submitted the digital paper trail required under the assignment/project instructions; and that
- b. Wherever GAI assistance has been employed in the submission in word or paraphrase or inclusion of a significant idea or fact suggested by the GAI assistant, I have acknowledged this by a footnote; and that,
- c. Apart from the foregoing notices, the submission is wholly my own work.



Pascalis Pandey

Student Name & Signature

26 November 2023

Date

Appendix I: Declaration on Use of GAI (Generative Artificial Intelligence) Assistance in relation to Assignment/Project (to be submitted individually even for group projects)

I Nguyen Thi Quynh Trang (student name),
TRANG005 @e.ntu.edu.sg (NTU email) honestly and sincerely make the following declaration in relation to the following course submission:

1. Name of course: Object Oriented Design and Programming
2. Course Code: CZ2002
3. Instructor: Dr. Zhang Jie, Dr. Li Fang
4. Title of Assignment/Project Submission: Group Assignment CAMs

In relation to the foregoing I hereby declare that, fully and properly in accordance with the Assignment/Project Instructions I have (check where appropriate):

- i. Used GAI as permitted to assist in generating key ideas only. ☐
- ii. Used GAI as permitted to assist in generating a first text only. ☐

And/or

- iii. Used GAI to refine syntax and grammar for correct language submission only. ☐

Or

- iv. As it is not permitted: Not used GAI assistance in any way in the development or generation of this assignment or project.



I also declare that I have :

- a. Fully and honestly submitted the digital paper trail required under the assignment/project instructions; and that
- b. Wherever GAI assistance has been employed in the submission in word or paraphrase or inclusion of a significant idea or fact suggested by the GAI assistant, I have acknowledged this by a footnote; and that,
- c. Apart from the foregoing notices, the submission is wholly my own work.

Nguyen Thi Quynh Trang
Student Name & Signature

26 - Nov - 2023
Date