Phase 2 - Design Document
Team Variable

Updated Specification:

- Running the application initially asks the user for an existing log-in, or to create a new account using a username and password of their choosing
- Once logged in, a new user can now either:
    - choose to import a compatible calendar file,
    - manually schedule their events to a new calendar
- If the user chooses to manually schedule their events, they will be asked for:
    - the courses/recurring events they are currently participating in
    - the times and days when these events occur
- Once the user has completed the above steps, they can now:
    - View the scheduled events in their calendar
    - Add or remove events (either recurring or one-off) from their schedule
    - View which study group they are in
    - Join a new study group (via a group ID)
    - Create a new study group
- After a user has joined a study group with more than one member, they will be able to access the time in which all members within the group are free to conduct group work/study
- Data is saved automatically upon logout. And if there is any saved data present, the program retrieves the saved data automatically upon program startup.
- ***Users will have a task list, from which they can create, add, and remove tasks from.***
- ***Users can filter their task list for open tasks and closed tasks. Users will also be able to calculate the average amount of time it takes to close tasks.***

Project Design:

1. Adhering to SOLID Principles:

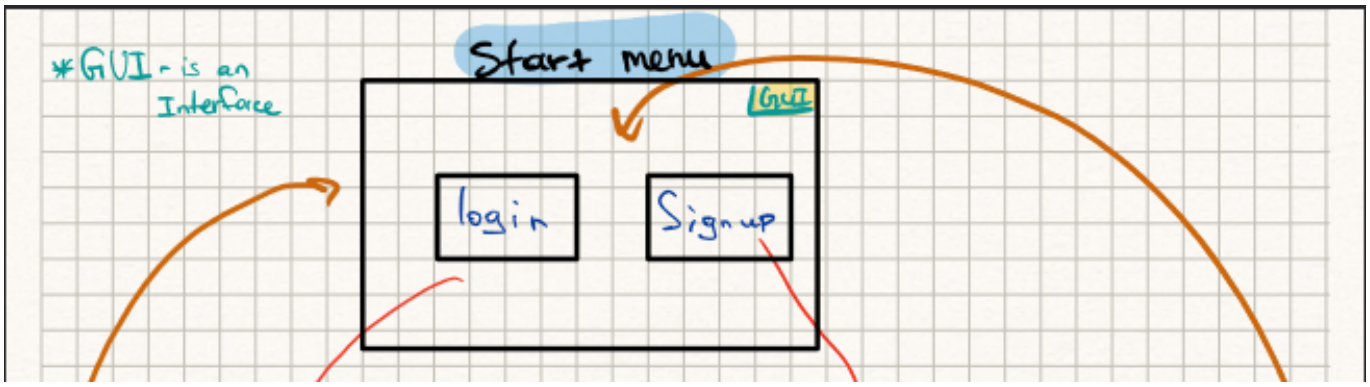
   We believe that our project in general adheres to the Single-Responsibility Principle well, as classes are well defined and each have their own job. For example, the entity classes are all separate and clear, and there is a manager class for each with a Factory class for EventInterface, a controller classes for each except EventInterface, and each GUI classes represent one single window

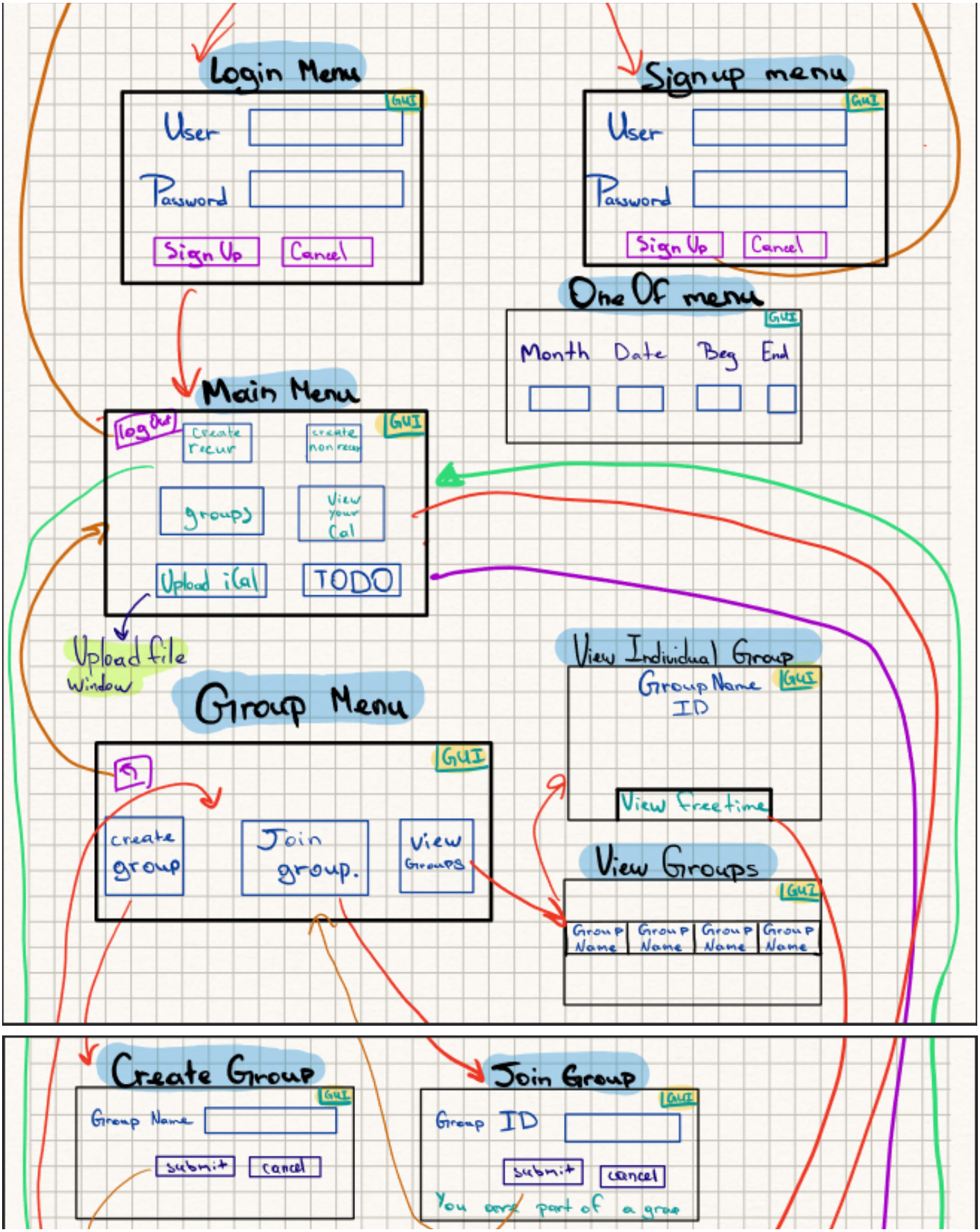
for the program. The one time the classes need to be changed is if the functionality is changed, which is ideal for the Single-Responsibility Principle.

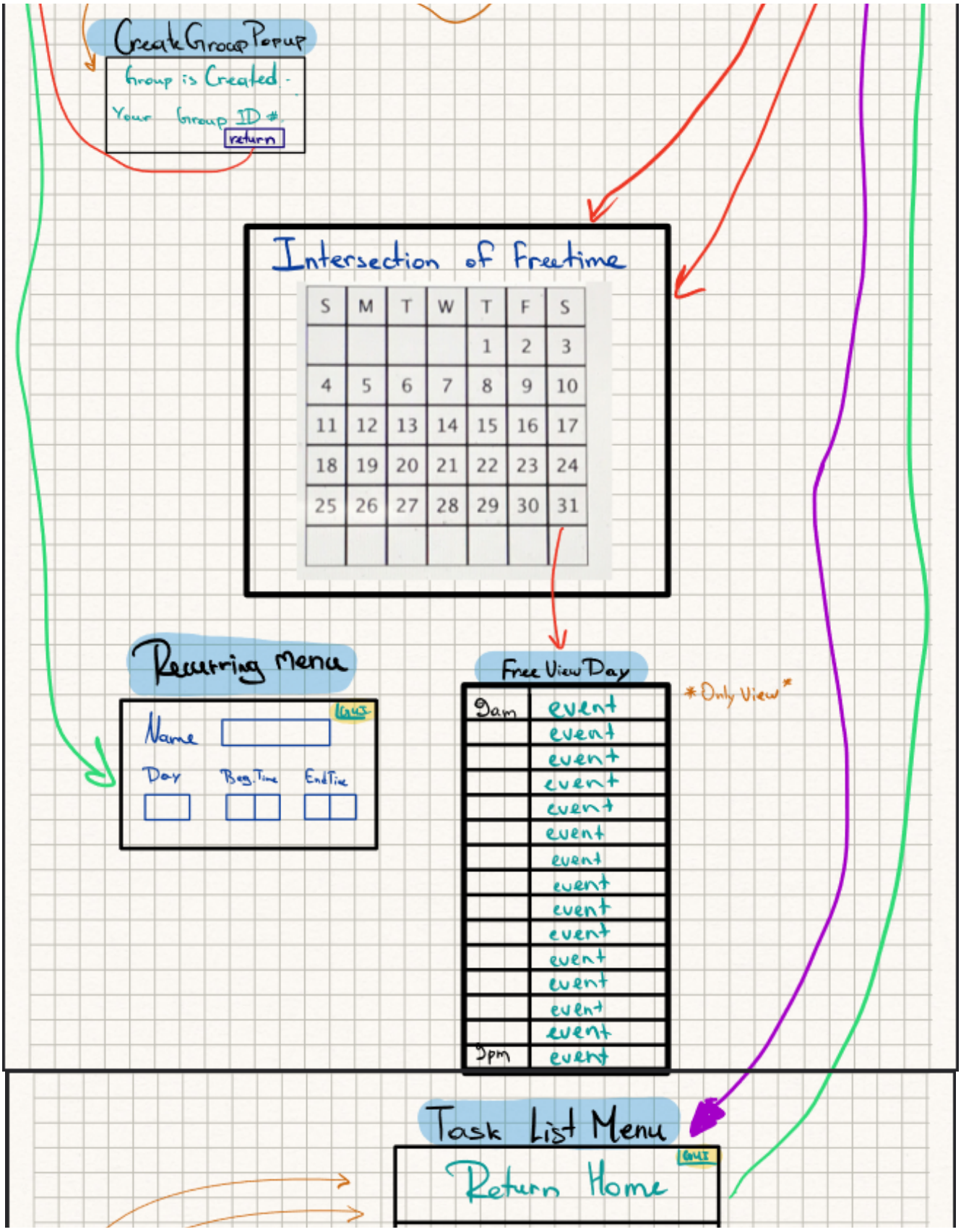
Moving on, after Phase 1 our feedback focused on how our code could adhere much closer to both the Open-Closed and Liskov Substitution principles by implementing an interface that our GUI classes would inherit. This has been implemented in Phase 2 as the GUI interface, with all common frontend classes implementing this GUI class (common meaning not user specific, i.e., calendars, etc.). Additionally in implementing this GUI interface on our frontend, we have also stuck more closely to the Interface Segregation and Dependency Inversion Principles as our GUI classes are based on an abstract concept in the form of the GUI interface and this interface allows for a shorter interface that only contains the necessary methods for each GUI class.

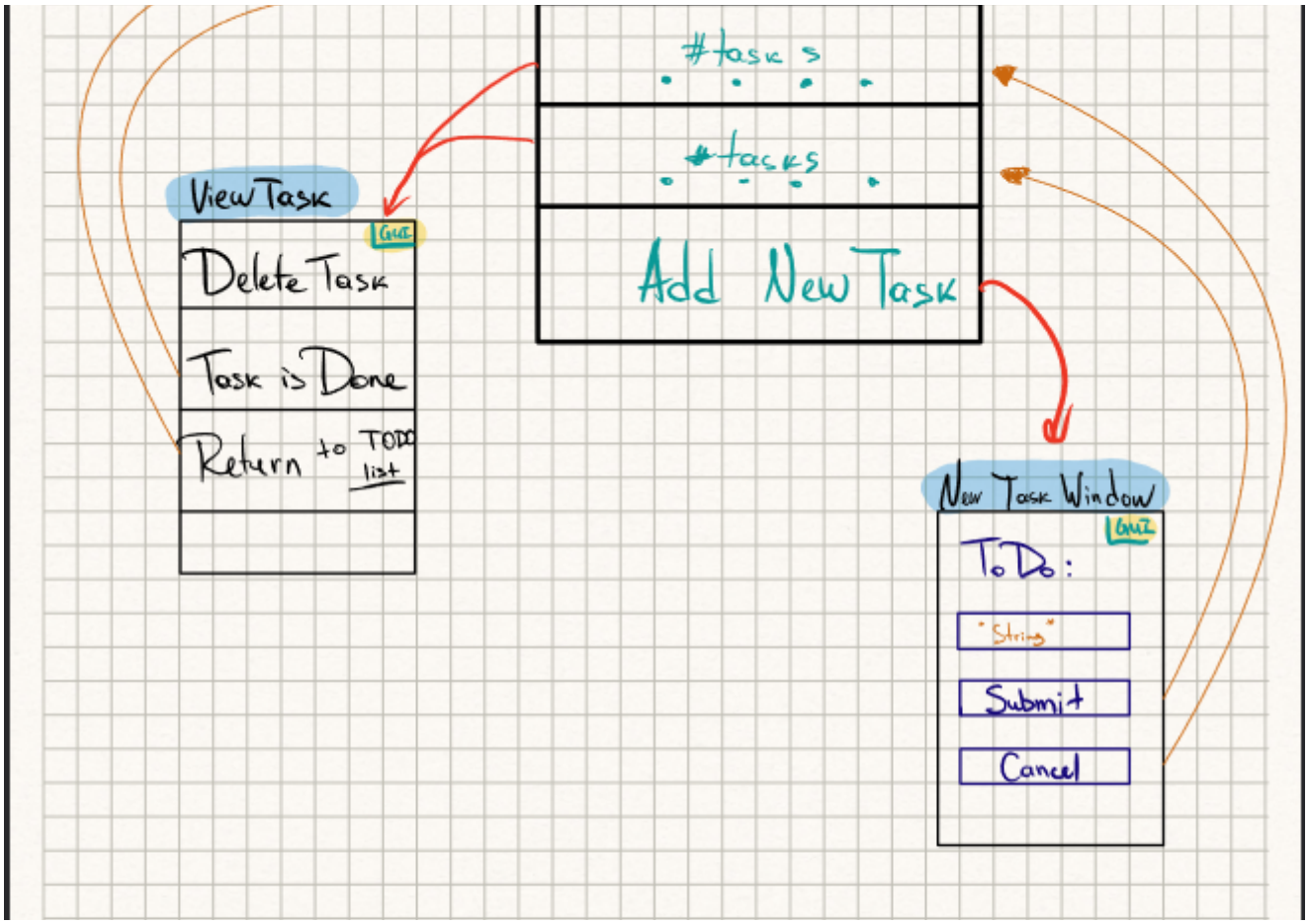
Looking deeper into our backend classes, we have also adhered closer to the Interface Segregation Principle via implementing the Person, Event, and StudentBuilderInterface interfaces, allowing for shorter interfaces that require only the necessary methods, while having classes such as Student, Group, ClaendarEvent, and OneOffEvent all depend on abstractions, allowing us to further adhere to the Dependency Inversion principle as well.

2. <u>Major Design Decisions:</u>
    - Person Interface
    - Event Interface
    - Format of OneOffEvent date (mm.dd)
    - ***GUI Interface***
    - Frontend Layout and Illustration:




Login Menu
GUI
User
Password
Sign Up
Cancel
Signup menu
GUI
User
Password
Sign Up
Cancel
One Of menu
GUI
Month
Date
Beg
End
Main Menu
log Out
GUI
Create Recur
create non recur
groups
View your Cal
Upload iCal
TODO
Upload file Window
Group Menu
GUI
create group
Join group.
View Groups
View Individual Group
Group Name
ID
GUI
View Freetime
View Groups
GUI
Group Name
Group Name
Group Name
Group Name
Create Group
GUI
Group Name
submit
cancel
Join Group
GUI
Group ID
submit
cancel
You are part of a group


Create Group Popup
Group is Created
Your Group ID #
return
Intersection of Freetime
S M T W T F S
1 2 3
4 5 6 7 8 9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30 31
Recurring Menu
GUI
Name
Day
Beg. Time
End Time
Free View Day
Only View
9am
event
3pm
Task List Menu
GUI
Return Home

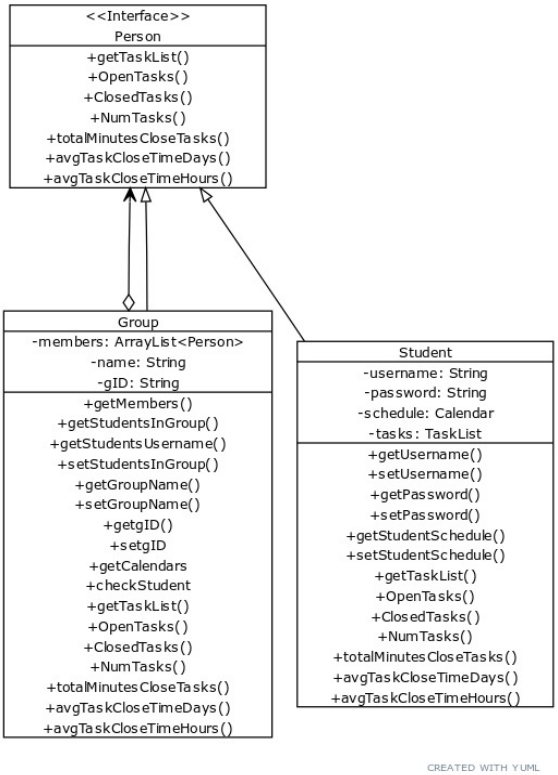


3. Design Patterns:

   a. **Composite Design Pattern (Documented as of Commit #325bde1):**
   Our project has a Student class and a Group class. The Group class can contain instances of Student as well as instances of Group. This allows Group to contain other nested Groups. This structure can be represented as a Tree-like structure where Student can be represented as a leaf and Group can be represented as a node. Since Student and Group can be represented as a Tree-like structure, we are able to make use of the **Composite Design Pattern**.

   To represent a single unit we will use the Student class as our Leaf class. And we will use the Group class as our Composite Class as Group can store both Student and Group classes. We implemented an Interface called Person, this Interface contains methods that are common to both

Student and Group and we will use the Person Interface as our Component class. These common methods are mostly related to the new Task List feature we implemented in Phase1. And finally, we implemented this Interface into both the Student class and Group class. The following is a UML class diagram to further illustrate this relationship:



b. <u>Template Method Design Pattern (Packaged and Documented as of Commit #cb6d110):</u>

The **Template Method** design pattern defines the skeleton of an algorithm as a base class that contains the standard steps across a set of classes. This abstract class contains the invariant pieces of the domain's architecture, providing a structure with either default implementation or no implementation at all. The abstract class allows its subclasses to redefine certain steps of the algorithm and specify different implementations if needed. In a way, the base class represents a "placeholder" of the algorithm, and the subclasses implement the "placeholder". Furthermore, the Template Method uses inheritance to vary part of an algorithm and modifies the logic of an entire class.

The goal of our project is to find a time slot that works for everyone across a group given their individual schedules. In our frontend, we want to have windows to show the calendar and days for viewing. For example, we have FreeViewDay to see the days people are free and ViewDay, which is a parent class of FreeViewDay. The **Template Method** design pattern is applicable here with ViewDay being the base class and FreeViewDay being its subclass. We have standard functions in ViewDay such as setTable(), getCalendarController(), getDate(). These functions are also applicable and are inherited in the FreeViewDay class. However, there needs to be alternations of steps in FreeViewDay as we have to populate it differently, which leads to an override of the function populate().

c. ***Factory Design Pattern (Packaged and Merged as of Commit #7ae86cd):***

*The **Factory Design** Pattern was implemented within our code by refactoring the EventCreator class to the EventFactory class. The reason we unconsciously implemented the properties of the **Factory Design** within our EventCreator class when finishing Phase 0, all that was remaining to turn our EventCreator into a class that followed a design pattern was to have our Event classes depend on an interface. Since we made this design decision during Phase 1 (in implementing an EventInterface) as per our TA's advice, we were able to refactor this class to match the **Factory Design** pattern.*

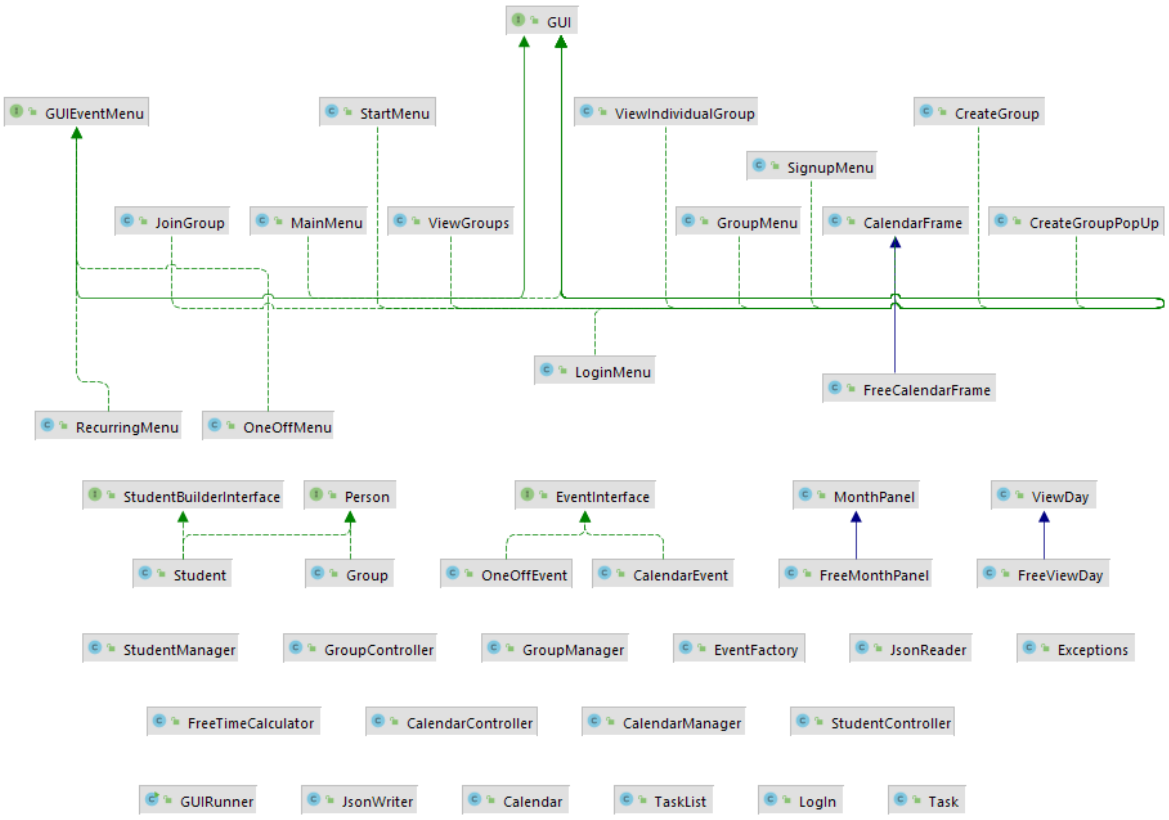
d. ***Builder Design Pattern (Packaged and Merged as of Commit #7ae86cd):***

*We implemented the **Builder Design Pattern** within our code by introducing the StudentBuilderInterface, adding the constructing methods within our Student class to said interface, refactor our code such that the constructing methods override the methods within the instructor, and make calls to these methods within the constructor of Student class. Doing so allowed us to remove the complexity from the base constructor of Student, in addition to future-proofing our code as it avoids telescoping constructors if we needed to add further attributes to the Student class, and allowing us to have multiple constructors within the Student class.*

Project Structure/Architecture:

1. ## UML Class Diagrams:

   All UML Diagrams can be found in the directory *...phase1/UML Diagrams.* The following is a screen capture of the UML Diagram generated in IntelliJ for our backend classes without dependencies:



2. ## Adhering to Clean Architecture:

   We believe our program adheres to Clean Architecture quite well. To adhere to Clean Architecture, we purposefully created Controller classes wherever and whenever we needed to access certain properties of entity classes within our outermost layer. Each necessary method that the GUI calls on will always call on it from the instances of the controller classes, furthermore no manager classes are called upon within the outermost layer. An example of the Dependency Rule being consistently followed in the GUI would be in the RecurringMenu and OneoffMenu, where we would add a new event to the current user's Calendar.

CalendarController's createRecEvent() or createOneOffEvent() method is called upon respectively, which then calls the CalendarManager's respective methods, which then operates on the entities. Another example would be a helper method called populate() in ViewDay, which helps structure and create a parameter needed to display the single day schedule. This method calls on the StudentController's getTimes() method, given a String of the current username which we have passed in as a parameter, the date, and the day of the week. This method then runs and calls upon the corresponding StudentManager method which then directly accesses the student's Calendar.
Note that in phase 2 we have implemented the Task and TaskList GUI classes which also adhere to Clean Architecture by only calling the controller classes' methods. We have added some methods in StudentManager and StudentController to manipulate the class Task and TaskList. The same goes with the import Ical files.

3. Packaging Structure

For our packaging structure in Phase 1, we mainly focused on dividing up parts of the program (both frontend and backend) into ways that would make it easier to navigate for all of us. We created packages that were appropriately named, but not over packaged such that it would be difficult to navigate when looking for a specific class. As such, we chose the packages (all in ...src/main/java) ***backend*** (responsible for dealing with solely backend related classes)*,* ***frontend*** (containing classes relating exclusively to the frontend)*,* ***calendar*** (containing classes related to any user's calendar)*,* ***events*** (containing classes related to all user events)*,* ***login*** (containing classes relating to user login and sign up)*,* ***users*** (containing classes relating to individual or groups of users). ***Given our Phase 1 feedback on how our incomplete Task feature was packaged inappropriately, now that we have fully implemented the intended functionality of this feature, we have packaged it within a new tasks package.***

Program Specifics:

1. Functionality:
   At the end of Phase 0, our team had completed the majority of the backend development needed for our program, having our Command Line Interface run most of the functionality of the original specification. Therefore, our focus for Phase 1 was ensuring the completion of our backend development, while half of our team worked on the frontend development. Additionally, to save and load information for our program we implemented the classes JsonWriter and JsonReader. These classes leverage the Gson and Jackson libraries to serialize and deserialize our program's data. The JsonWriter class is then able to take the serialized data and write them to a JSON file for storage. As well, the JsonReader class is able to read the saved JSON file and deserialize the data back into our program. ***For Phase 2, our main focus was the added functionality in the form of support for uploading iCal files as a user's calendar, while fixing and refactoring our code to adhere more closely to the SOLID principles, and further implementing applicable Design Patterns.***

2. Code Style & Documentation:
   Over the course of Phase 1, the frontend team has consistently ensured that their code was appropriately documented using JavaDoc, using appropriate method and class names to ensure our code would be easily readable. Additionally, we made sure to backtrack and also documented any source code from Phase 0 that had previously been left undocumented, and that class names and methods were consistent so that all of our code was both readable and easily understood via the documentation. ***As the majority of our code has been documented from Phase 1 and 0, for Phase 2 we ensured that we held ourselves to the standard of documentation and style that we had before, documenting any code as needed, while maintaining our style in programming conventions.***

3. Testing/Refactoring:

   a. Testing
      Moving from Phase 0 into Phase 1, we tried to fully populate our backend's test cases as much as possible, taking care of edge cases wherever appropriate and also including the test cases for exceptions. For our frontend/GUI, we realise that writing unit tests is difficult and

labour-intensive, so we tested our GUI by covering any sequences of operations we could think of, and comparing whether we obtained the desired result. ***As it was highlighted in our Phase 1 feedback that our test base was not nearly populated enough, we have tried to implement many further tests on our backend classes to cover test cases, implementing at least 5 test cases per class and aiming for at least 8.***

b. Refactoring

During Phase 1 and 0, all refactors within our code primarily occurred when we dealt with the source code for the backend. One instance of refactoring was related to our CalendarEvent class and OneOffEvent class, CalendarEvent is used for recurring calendar events and OneOffEvent represents non-recurring events. We extracted an interface by creating an EventInterface class to better align with the Open/Closed principle (Pull Request #7 on the main branch). This will allow us to more cleanly extend the functionality of calendar events by writing new interfaces. Additionally, we also refactored our code after the introduction of the Person Interface, as both Student and Group classes implement this Interface (Commit #82be76f). This refactoring also allowed us to naturally implement the Composite Design pattern. ***In Phase 2, we utilized refactoring to ensure our code adhered closely to SOLID principles and Clean Architecture; by refactoring after we implemented the StudentBuilderInterface and GUI interfaces. Furthermore, to better follow the Open/Closed principle, we refactored our Task class by creating a Task Interface. We also refactored our serialization classes (JsonReader and JsonWriter) by creating a SaveData and LoadData interface.***
Additionally, we also used refactoring to implement Design Patterns, and repackage our classes according to our existing packaging structure (more information regarding these refactors can be found in the mentioned sections).

c. Code Smells

When looking at potential Code Smells within our code, we can analyse a case of potentially *Long Methods* within our FreeTimeCalculator and GUI classes as they contain methods that are more than 20 lines long. Furthermore, as with our given hierarchy structure, even though we believe our code adheres to both the SOLID principles and Clean architecture well, we could see higher

level classes (Controller level classes in particular) being seen as *Middle Men*. So we paid careful attention to some of our higher level classes, particularly, our Controller classes. Since these Controller classes often delegate work to Use case classes, we needed to pay special attention so that these Controller classes don't become middle men.

4. Use of GitHub Features:

With regards to GitHub features, our team has made use of Pull Requests wherever appropriate, always letting the others know before making any significant changes to the main branch. Furthermore, we explored the Issues feature within GitHub to address major issues when merging or running code, such as whether we should have a Build Configuration for the project given the various dependencies within our code. Furthermore, when handling any debugging within the team, we made efforts to specifically point out Actions that resulted in said error to give context when addressing the issue within our code. ***Additionally, within Phase 2, we found the rollback feature of the GitHub suite to be useful as we rolled back our repository to a previous commit when we realised that a commit had been made that was not properly documented, allowing us to recommit our code with an appropriate commit title and proper commenting. Furthermore, we also realised how being able to locate the difference between previous commits was significantly important while debugging when one of our members had accidentally deleted a block of code that was required for the data to persist across different instances of our application.***

Progress Report:

1. Open Questions:
    - What would be the best resources to learn how to host this program/application on the web?
    - Would we gain this knowledge from courses, if so, which courses would they be and would we be able to take them in third year?

2. Advantages of Current Design:

The advantages of our current design include (but are not limited to):

- A straightforward means of adding new types of Event due to our EventInterface
- A straightforward means of adding new types of Persons due to our PersonInterface
- HashMaps used in appropriate places to improve efficiency over ArrayLists (considering we are primarily focused with accessing specific indexes)
- A GUI that is very flexible due to our use of Swing as Swing components follow the Model-View-Controller paradigm, in addition to the ability to include 'extras' (icons, etc.) for different components
- ***A GUI that is also open to extension in adhering to the Open Closed Principle via our GUI Interface***
- ***A new means of allowing our users to upload an existing calendar file (iCal) as opposed to manually entering weekly recurring events***

3. Individual Contributions Since Phase 1:

Name of Individual	Contributions Made Since Phase 1
Cathlyn	● Created **TaskInterface** to allow comparisons between tasks based on start times ● Used **TaskInterface** to sort **TaskList** based on start times ● Implemented new filtering feature for **TaskList** ● Further populated test cases for backend classes ○ CalendarController ○ CalendarManager ○ EventCreator ○ FreeTimeCalculator ○ OneOffEvent ● Aided in debugging serialization issues for **Student, Group,** and **Task** classes
Gilbert	● Refactored GUI classes ○ Added **GUI** interface and made most GUI classes implement this interface to adhere to SOLID ○ Added **GUIEventMenu** interface and made RecurringMenu and OneOffMenu implement this interface to adhere to SOLID ● Worked on ical File upload ○ Helped structuring the files to use the Ical4j library

	● Worked on new GUI classes ○ Helped getting started on the Task and TaskList GUI classes ○ Helped work on the TaskListMenu class ○ Changed ViewTask so that when a task is closed, it conveyed to the user
Raaghav	● Worked on ical File upload Feature ○ Used Ical4j library to read in ical files ○ Parsed through ical files, and identified if an event was recurring or non-recurring ○ If event was recurring, used CalendarEvent class to add the event ○ If event was non-recurring, used OneOffEvent class to add event ● Cleaned up some of the GUI classes/displays ○ Modified main menu so that it has a file upload button ○ Fixed aligning of GUI buttons ● Implemented Pop-Up Display ○ Created a pop up display window in the GUI ○ The window indicates once a calendar is successfully imported ● Helped handle exceptions ○ Handles exceptions in GUI
Rashid	**Implemented GUI (TODO list)** - connected everything with backend (**Task and TaskList**) - Implemented **NewTaskWindow class** - Implemented **ViewTask class** - Implemented **TaskListMenu class** - Added methods to **StudentController** class - Added methods to **StudentManager** class - Drew the GUI diagram
Tajwaar	● Implemented **StudentBuilderInterface** so **Student class** would better adhere to SOLID ● Refactored **EventCreator** and **Student** classes to implement Factory and Builder Design Patterns within the respective classes ● Repackaged classes to better fit existing packaging method ● **Lead contributor for all documentation across all phases** (Design Document, Accessibility Report, etc.) ● Aided in further populating test base for backend classes ● Aided in debugging serialization issues for **Student,**

	Group, and **Task classes** ● **Lead contributor to Exception handling**
Vergil	**Fixed serialization issues:** ● Fixed issues related to the Group class in regards to the saving and loading of the Group class. **Fixed bug related to the GUI frontend of ViewGroups:** ● Fix a bug related to the GUI View Group function in a multi-user environment. **Refactored code:** ● Refactored the JsonReader and JsonWriter classes by implementing a SaveData interface and a LoadData interface from which both JsonReader and JsonWriter will implement. **Added functionality in the GUI frontend:** ● Any Tasks that are marked as closed will appear as red when viewing TODO list (TaskList). **Testing:** ● Created a test suite for classes Task and TaskList ● Added additional test for Calendar and CalendarEvent classes

4. Links and Descriptions of Individual Contributions:

Name of Individual	Link(s) to Important Contribution(s)	Description/Reasoning of Contribution
Cathlyn	https://github.com/CSC207-UofT/course-project-variable/pull/23 https://github.com/CSC207-UofT/course-project-variable/pull/39	● TaskList features ● Unit tests
Gilbert	Commit #c2b01a8 https://github.com/CSC207-UofT/course-project-variable/commit/c2b01a8f37dffce73ad9ab12863d8e47cd23147a	These are all refactoring of GUI classes to implement the interface **GUI** and the event GUI classes to implement **GUIEventMenu**, this demonstrates a significant contribution as it completely refactored the

		GUI classes to adhere to SOLID principles better, an issue from phase 1.
Raaghav	1. Commit #1a0418e (https://github.com/CSC207-UofT/course-project-variable/commit/1a0418e9a113a9c606414f400ce7104e98e69169) 2. Commit #cfc7208 (https://github.com/CSC207-UofT/course-project-variable/commit/cfc7208fb0a7ed1c65363cac7108aa637c9614fa) (This commit was me testing and playing around with the Ical4j library)	This demonstrates a significant contribution as this was one of the main features we wanted to implement in our original rough specification despite not having sufficient time for it. The feature allows for users to seamlessly import their own personal calendars into our program, hence making the program more user-friendly. **Note:** Although I mostly worked on the ICal file feature, it was Gilbert who created the final pull request for the feature as I was having issues with git.
Rashid	https://github.com/CSC207-UofT/course-project-variable/pull/24	New feature of a simple TODO list was added and I implemented the GUI representation of it and also made changes in backend classes. Users will be allowed to add, delete, and close the task in their Todo list.
Tajwaar	1. Commit #7ae86cd (https://github.com/CSC207-UofT/course-project-variable/tree/7ae86cd2159eaa8825415f7ff3bb4f628ea55932)	1. Contributes significantly to how our project adheres to SOLID principles and Clean Architecture by implementing

	2. Commit #04c3e81 (https://github.com/CSC207-UofT/course-project-variable/tree/04c3e81d893f51fa471d4620cacc829d0f9470ae)	**StudentBuilderInterface** and using Design Patterns 2. Contributes significantly so our project can adhere to the Single Responsibility and Open Closed SOLID principles
Vergil	1. https://github.com/CSC207-UofT/course-project-variable/commit/6e1cf4ab5fcac71924580602a21144037708fb7f 2. https://github.com/CSC207-UofT/course-project-variable/commit/fcce0fa86308eb49f08f29bf129b7635eaa40c58	This was significant since it allowed our program to save and load program data. This was significant since it implemented a new core feature of our program.

Accessibility Report:

1. Adhering to Principles of Universal Design;

    a. Equitable Use:
    - Our application provides the same user interface for all users, regardless of when the user was made, or any other external factor, adhering to this universal principle
    - Similarly, the application does not segregate any users, providing an equivalent experience for all, while providing the same amount of privacy and security to all

    b. Flexibility in Use:

- As our application does not specifically allow for the flexibility in its use, to facilitate this, we could:
    - Allow an option to set the size of the buttons within our user interface
    - Allow an option to choose the colour scheme/colourway of the user interface and/or a simple dark mode for those who prefer it

c. Simple and Intuitive Use:
- Our application provides a base UI, with a maximum of 4-5 options on each individual page of the application, eliminating unnecessary complexity with a highly complicated design
- Additionally, our application also simple and concise language to label buttons, and gives precise feedback for incorrect inputs

d. Perceptible Information:
- We believe that our application provides an adequate contrast between necessary information and the background, however to make the application further adhere to this principle, we could:
    - Provide a greater contrast in buttons and their labels as opposed to backgrounds using a ‘Dark Mode’
    - Provide tactile feedback (preferably through a Mobile Version) for when an incorrect input is entered and feedback is provided

e. Tolerance for Error:
- Within our application, we constantly provide warning and feedback for incorrect inputs entered at all steps in the running of the program, thus we believe that in this way the project adheres to the above universal design principle
- Furthermore, we chose to have ‘logout’ or ‘signout’ options that would be inconvenient if pressed at the furthest corners wherever possible, keeping most other buttons within the centre and away from the corners

f. Low Physical Effort:
- We believe our application allows for a low sustained physical effort, never requiring any options to be pressed and held down,

and once events are updated, the user only needs very few clicks to keep their calendar open and visible with zero physical effort
- In addition to this, once a user calendar is opened and updated, the user can view their calendar in a neutral position and keep it open without any physical effort

g. Size + Space for Approach and Use:
   - As our application does not closely adhere to this universal design principle, in order to improve this we could:
     - Implement the scaling of buttons and the background window of our user interface to improve visibility from large distances
     - While technically the buttons could be activated from a large difference using a wireless mouse, we could integrate support for assistive devices for the selection and activation of buttons

2. Target Demographic:

   If we were to sell or license this application, we would target it towards individuals that have many recurring events throughout their week; for example, students with many different classes that recur on certain days of the week, or office employees who frequently have weekly meetings and calls. Using this as a general demographic, we would then specify a target audience as individuals who are not only within this group of individuals, but also require a means of project/team management by advertising our 'Group' feature, allowing for multiple groups to be formed, in addition to 'Group Tasks' and 'Individual Task Lists' so both teams and individuals can assign tasks to both organise and structure their workflow at different levels of collaboration.

3. Accessing Audiences the Application is Likely to be Less Used by:

   Similarly, following from the paragraph above, our application is less likely to be used by individuals who do not have many recurring events in their schedule. The reasoning is because updating a user's calendar with only one-off-events is both time-consuming and laborious given our current implementation. Furthermore, accessibility wise, our application is less likely to be used by those with mobility related disabilities (as our program does not scale in its size with regards to window-size and buttons, and does not support assistive devices) and visual impairments that make

seeing smaller buttons, or reading relatively smaller text on a screen more difficult.