

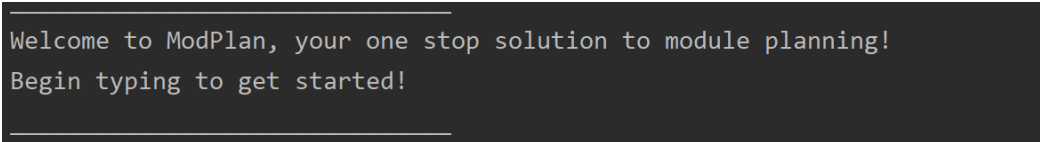
# Kyawt Kyawt San – Project Portfolio for ModPlan

## About the project

---

My team of 4 software engineering students and I were tasked to come together to enhance a basic command line interface desktop application for our CS2113T software engineering project. We chose to morph our Duke project into NUS student modules online management system called ModPlan. This enhanced application enables NUS students to record their modules data, their CCA schedule and calculate their CAP result effortlessly just by inputting the grade.

This is what our project looks like:



```
Welcome to ModPlan, your one stop solution to module planning!
Begin typing to get started!
```

**Figure 1.** The command line for **ModPlan**.

My role was to design and write the codes for the `show` and `reminder` features. The sections below depict these enhancements in more detail, as well as the relevant documentation I have added to the user and development guides regarding these enhancements.

Note the following symbols and formatting used in this document:

`show` A light grey highlight with a black word indicates that this is a command that can be inputted into a command line and executed by the application.

`ModuleCommand` A bright blue highlight with a black word indicates that this is a component, class or object in the architecture of the application.

## Summary of contributions

---

This section shows a summary of my coding, documentation, and other helpful contributions to the team project.

### Enhancement added:

I added the ability to `show reports and lists of the cca` and module via the `show` command.

- What it does: The `show` command allows the user to see a list of specified modules or ccas added in the module or cca list respectively.
- Justification: The user can know how much different types of modules they need to take in order to graduate in time.
- Highlights: This enhancement works with existing as well as future commands. This enhancement currently works well in showing the existing modules and CCAs added in the first semester. In the future, the existing modules can be added to the second semester and further extended to other years. The implementation was challenging as it was difficult to print out the list of only the core modules added. Since the list of the core module taken by the CEG students does not follow a certain algorithm, I had to hard code the list of core modules taken by the CEG students.
- Credits: As the `ShowCommand` is an extension of the parent class, `ModuleCommand`, the credit would go to my teammate, Iman, who have created the `ModuleCommand` parent class.

Besides this, I have also added the ability to **start and stop reminder messages**.

- What it does: The `reminder` command allows user to set or stop reminder messages regarding the update of the module information.
- Justification: Since the module information may change according to the semesters, it is important to update the module information. Reminder messages are set so that users will remember to update the module information. This feature allows users to customise the time interval of which the reminder messages pops up, from a time interval of 10 seconds to 2 minutes.
- Highlights: This enhancement works with existing as well as future commands. This enhancement currently works well in setting frequent reminder for the update of the module information. In the future, the reminder messages can be customised for other types of reminders. The time interval of the reminder message can be more customised as well. The reminder messages can also be expanded to mobile phone notifications. The implementation was challenging because of a few reasons. It was difficult to set up a reminder thread without interfering other commands. Moreover, the application could not stop even when the `bye` command is being input, as the thread continues running. To overcome these challenges, I used a reminder thread which extends from the `Timer` class. The `Timer` class also allows me to set specific intervals of the reminder thread by using “`time.schedule()`”. In order to stop the reminder message, a `killAllTimers()` method was introduced in both the `bye` command and the `reminder stop` command.
- Credits: The credits go to the in-built `Timer` class in java.

**Code contributed:** Please click the following links to view a sample of my code.

- <https://nuscs2113-ay1920s1.github.io/dashboard/#=undefined&search=kyawtsan99>

#### Other contributions:

- Introduced new features:
  - Show command (Pull request [#96](#))
  - Reminder command (Pull request [#127](#))
- Enhancements to existing features:
  - Updated the report command, which is now integrated in the show command (Pull request [#80](#), [#96](#), [#117](#))
  - Updated the reminder command (Pull request [#127](#), [#129](#), [#196](#), [#212](#))
  - Updated the clear command (Pull request [#201](#))
  - Added test cases for existing features (Pull request [#216](#))
- Fixed bugs:
  - Fixed bugs for show command (Pull request [#193](#))
  - Fixed bugs for report command (Pull request [#199](#))
- Community:
  - Reviewed and merged pull requests (with non-trivial review comments) : , [#49](#), [#50](#) , [#57](#), [#85](#), [#86](#), [#88](#), [#95](#), [#101](#), [#102](#), [#122](#), [#126](#), [#128](#), [#132](#), [#191](#), [#192](#), [#197](#), [#198](#), [#200](#), [#205](#), [#214](#), [#215](#), [#217](#)
  - Has issues: [#7](#), [#19](#), [#22](#), [#23](#), [#29](#), [#39](#), [#107](#)

## Contributions to the User Guide

---

We had to update the original addressbook User Guide with instructions for the enhancements that we had added. I have written for the `Features`, `Errors` and `Command Summary`. The following is an

excerpt from our [ModPlan User Guide](#), showing additions that I have made for the **show** and **reminder** features.

### Show Command:

Shows a list of specified modules or ccas added in the module or cca list respectively.

1. To show a list of all modules added in the module list:

- a. Type: **show module** into the command line, and press **Enter** to execute it.

```
show module
```

- b. It will then display a list of all the modules. For each module, the information on the module code, the number of MCs, if the module can be S/U'ed, if the module is taken and the grade of the module if the module is taken, are all shown.

```
All modules in the list!
```

```
1. [taken] CS1010 | ModuleCode:CS1010, MC:4.0, SU: can S/U, grade:A
2. [not taken] GES1012 | ModuleCode:GES1012, MC:4.0, SU: can S/U, grade:
3. [not taken] LAJ1201 | ModuleCode:LAJ1201, MC:4.0, SU: can S/U, grade:
4. [not taken] GES1002 | ModuleCode:GES1002, MC:4.0, SU: can S/U, grade:
```

2. To show a list of all the core modules added in the module list:

- a. Type: **show core** into the command line, and press **Enter** to execute it.

```
show core
```

- b. It prints out a report on all the core modules taken in the semester, together with the number of core modules left to take for graduation.

```
Here is your list of core modules being added:
```

```
1. [taken] CS1010 | ModuleCode:CS1010, MC:4.0, SU: can S/U, grade:A
```

```
Number of core modules required to take for graduation:
```

```
21
```

3. To show a list of all the general educational modules added in the module list:

- a. Type: **show ge** into the command line, and press **Enter** to execute it.

```
show ge
```

- b. It prints out a report on all the General Education(GE) modules taken in the semester, together with the number of GE modules left to take for graduation. If more than one kind of GE module is input, the application will inform the user.

```
Here is your list of general education modules being added:
```

```
1. [not taken] GES1012 | ModuleCode:GES1012, MC:4.0, SU: can S/U, grade:
```

```
2. [not taken] GES1002 | ModuleCode:GES1002, MC:4.0, SU: can S/U, grade:
```

```
There are more than one type of GE modules added.
```

```
Please add only one type of GE module each.
```

```
Number of general education modules required to take for graduation:
```

```
4
```

4. To show a list of all the unrestricted electives modules added in the module list:

- a. Type: **show ue** into the command line, and press **Enter** to execute it.

```
show ue
```

- b. It prints out a report on all the Unrestricted Electives(UE) modules taken in the semester, together with the number of UE modules left to take for graduation.

```
Here is your list of unrestricted elective modules being added:  
1. [not taken] LAJ1201 | ModuleCode:LAJ1201, MC:4.0, SU: can S/U, grade:  
  
Number of unrestricted elective modules required to take for graduation:  
7  
_____
```

5. To show a list of all ccas added in the module list:

- a. Type: `show cca` into the command line, and press `Enter` to execute it.

```
show cca  
_____
```

- b. It prints out the list of all ccas.

```
All ccas in the list!  
1. [C] SOCCER | 16:00 - 18:00 on MONDAY  
_____
```

### Reminder command:

Allows the user to the start and stop the reminder to update the module data for a specified period.

1. To show a list of all the different specified time interval of reminder:

- a. Type: `reminder list` into the command line, and press `Enter` to execute it.

```
reminder list  
_____
```

- b. It gives four options to determine how often the user wants to set the reminder.

```
Would you like to set your reminder to every:  
1) for 10 seconds  
2) for 30 seconds  
3) for 1 minute  
4) for 2 minutes  
*helpline* : for 1), enter 'reminder one'  
_____
```

2. To choose the desired time interval of reminder:

- a. Type: `reminder NUMBER` into the command line, and press `Enter` to execute it.

```
reminder one  
_____
```

- b. Allows the user to choose the desired period for the reminder to appear, which ranges from 10 seconds to 2 minutes. In this case, the reminder message is chosen to appear every 10 seconds.

```
Please remember to update your module information!  
To do so, you can input the update command in the following format:  
update module  
_____
```

3. To stop the reminder message:

- a. Type: `reminder stop` into the command line, and press `Enter` to execute it.

```
reminder stop  
_____
```

- b. Allows the user to stop the reminder and the reminder message will disappear.

```
Your reminder for the update is being stopped.  
To activate the reminder again, type reminder list.
```

### Show reports taken per semester [coming in v2.0]

Modules for each semester can be stored into different lists. Reports for specific year and semester can be generated.

### More customisable reminder message [coming in v2.0]

In the future, the reminder messages can be customised for other types of reminders and the time interval of the reminder message can be more customised as well. The reminder messages can also be expanded to other notifications such as the mobile phone notifications.

## Contributions to the Development Guide

---

The following is an excerpt from our *ModPlan Developer Guide*, showing additions that I have made for the `show` and `reminder` features.

### Show command:

#### Current implementation

The `show` feature is operated by the `ShowCommand` class, which is called by the Parser class. Upon user input of `show TYPE`, the Parser will return a new `ShowCommand`.

Since `ShowCommand` inherits the `ModuleCommand` class, it must override the `execute` method to specially execute the `show` command.

The parameter `TYPE` can take five forms according to the user input:

- `show module`
- `show core`
- `show ge`
- `show ue`
- `show cca`

These `TYPE` parameters will be parsed by the Parser class and pass the corresponding argument of `toShow` into the `ShowCommand` class. A switch case statement will handle the `toShow` argument. Upon construction of the `ShowCommand` class, one variable involved in the generation of the report is initialised, which refers to the `coreModList`. This variable is being used to store the strings of core modules taken by CEG students throughout the four years. The list of core modules is being taken from the NUS module requirements website.

As stated above, there are five methods that can be executed depending on the `TYPE` that the user inputs.

- Case 1: `show module`  
If the argument read for `toShow` is "module", an iterator loops through the array list of `ModuleTask`. It then prints out the entire list of modules being added.
- Case 2: `show core`  
If the argument read for `toShow` is "core", an iterator loops through the array list of `ModuleTask` and checks it against the set of `coreModList`. If the `moduleCode` in `ModuleTask` matches the `moduleCode` from the `coreModList`, that `moduleCode` is being printed out.

- The second part subtracts the number of core modules taken from the total number of core modules required to be taken. This information is taken from the NUS module requirements website. Hence, it shows the user the number of core modules left to take for the rest of the years.
- Case 3: show ge
 

If the argument read for `toShow` is "ge", an iterator loops through the array list of `ModuleTask` and checks the starting two letters of the `moduleCode`. If the `moduleCode` starts with a "GE", it is classified as a General Elective (GE) module and will be printed out.

  - The second part subtracts the number of GE modules taken from the total number of GE modules required to be taken. This information is taken from the NUS module requirements website. Hence, it shows the user the number of GE modules left to take for the rest of the years.
- Case 4: show ue
 

If the argument read for `toShow` is "ue", an iterator loops through the array list of `toShow`. It checks the starting two letters of the `moduleCode` and checks the `moduleCode` against the set of `coreModList`. If the `moduleCode` does not match the `moduleCode` from the `coreModList` and it does not start with a GE, it is classified as a Unrestricted Elective (UE) module and will be printed out.

  - The second part subtracts the number of UE modules taken from the total number of UE modules required to be taken, which the information is taken from the NUS module requirements website. Hence, it shows the users the number of UE modules left to take for the rest of the years.
- Case 5: show cca
 

If the argument read for `toShow` is "cca", an iterator iterates through the array list of ccas, which includes the CCAs added, and prints it out.

The sequence Diagram (Figure 2.) at the last page shows how `ShowCommand` works. `ShowCommand` inherits the attributes and methods from the `ModuleCommand`, which is shown by the 1.1 arg arrow connecting `ShowCommand` and `ModuleCommand`. The `ModuleCommand` calls itself as it uses its own attributes method as shown by 1.1.1 arg. The `ShowCommand` also calls certain methods of `PlannerUi` as shown by the arrows from 1.2 to 1.11 args. This is because the `ShowCommand` uses the methods in `PlannerUi` such as the `listCcaMsg` and `coreModReport`.

## Reminder command:

### Current implementation

The reminder feature is operated by the `ReminderCommand` class, which is called by the Parser class. Upon user input of reminder TYPE, the Parser will return a new `ReminderCommand`. Since `ReminderCommand` inherits the `ModuleCommand` class, it must override the execute method to specially execute the reminder command.

The parameter NUMBER can take six forms according to the user input:

- reminder list
- reminder one
- reminder two
- reminder three

- reminder four
- reminder stop

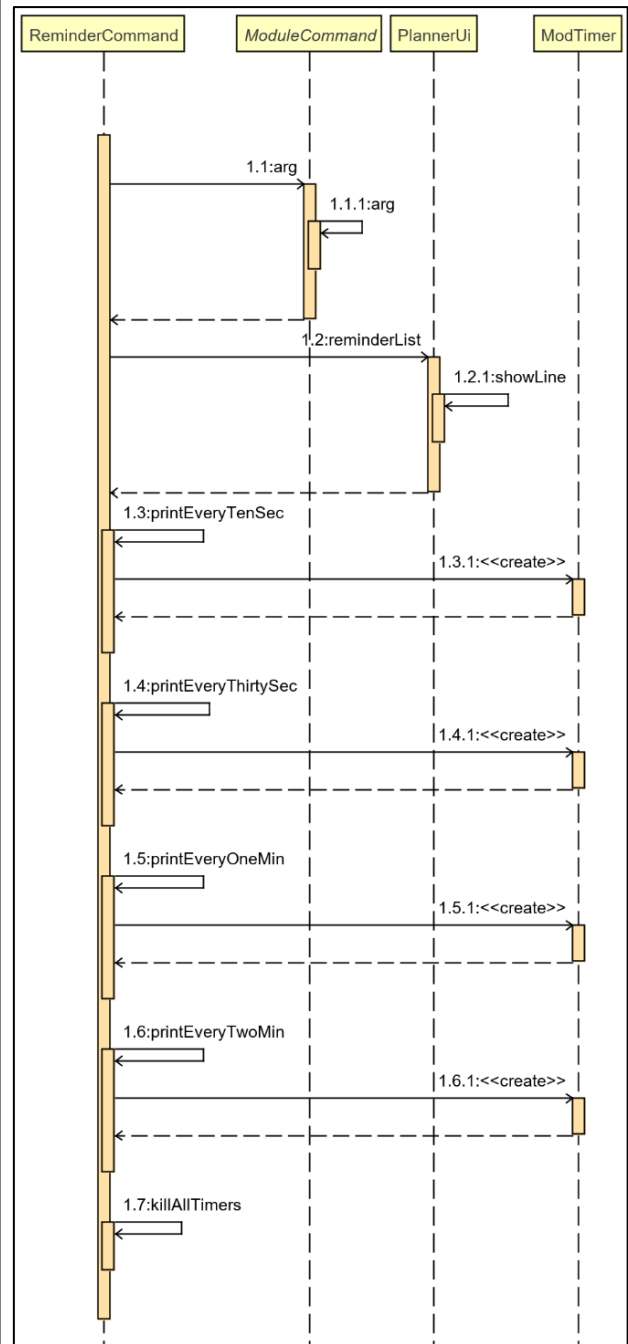
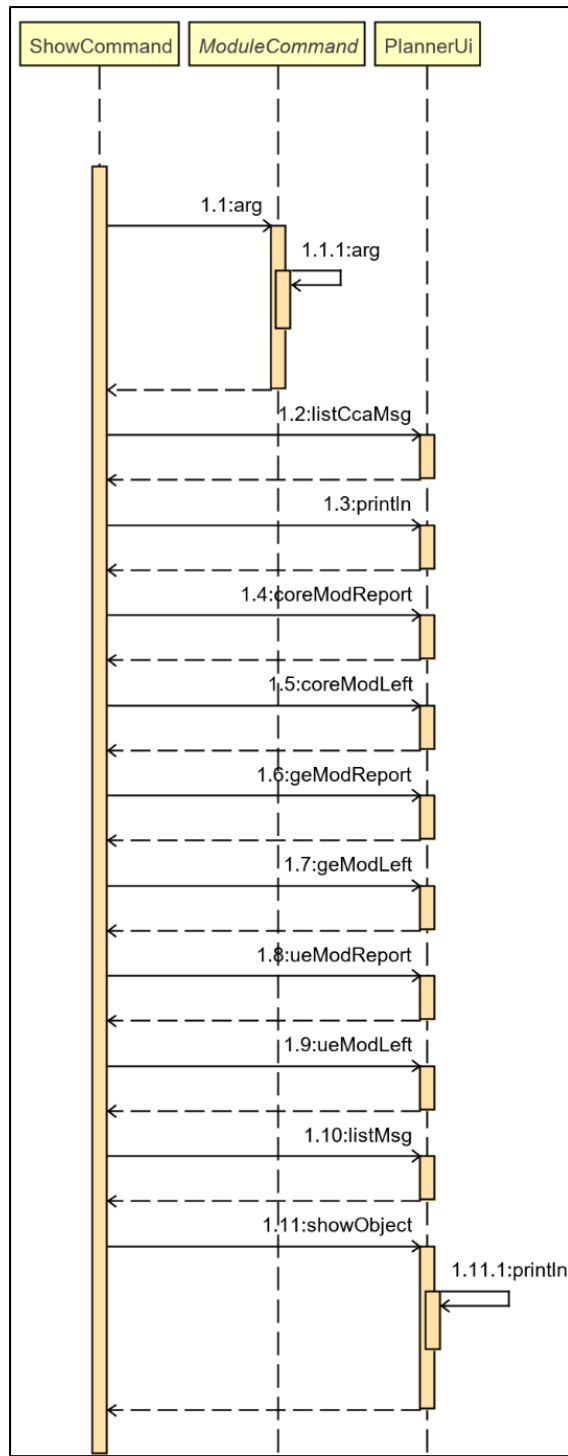
These TYPE parameters will be parsed by the Parser class and pass the corresponding argument of `toReminder` into the `ReminderCommand` class. A switch case statement will handle the `toReminder` argument.

Upon construction of the `ReminderCommand` class, variables using `Timer` class and `ScheduledTask` class are initialised.

As stated above, there are six methods that can be executed depending on the TYPE that the user inputs.

- Case 1: reminder list  
If the argument read for `toReminder` is "list", the `reminderList()` method in `plannerUI` class is called. A list of the different reminder interval options is being printed. The four different options allow user to determine how often the user want to set the reminder message. The user can set the reminder message to pop up every 10 seconds, 30 seconds, 1 minute or 2 minutes.
- Case 2: reminder one  
If the argument read for `toReminder` is "one", the `printEveryTenSec()` method is called. The time variable initialised by the Timer class is used to schedule the reminder message in `ScheduledTask` class every 10000ms, which equates to 10 seconds.
- Case 3: reminder two  
If the argument read for `toReminder` is "two", the `printEveryThirtySec()` method is called. The time variable initialised by the Timer class is used to schedule the reminder message in `ScheduledTask` class every 30000ms, which equates to 30 seconds.
- Case 4: reminder three  
If the argument read for `toReminder` is "three", the `printEveryOneMin()` method is called. The time variable initialised by the Timer class is used to schedule the reminder message in `ScheduledTask` class every 60000ms, which equates to 1 minute.
- Case 5: reminder four  
If the argument read for `toReminder` is "four", the `printEveryTwoMin()` method is called. The time variable initialised by the Timer class is used to schedule the reminder message in `ScheduledTask` class every 120000ms, which equates to 2 minutes.
- Case 6: reminder stop  
If the argument read for `toReminder` is "stop", the `killAllTimer()` method is called. A Timer iterator loops through the list of all the Timer threads and cancels all the Timer tasks. A message is then printed to notify the user that the reminder message for the update is being stopped.

The sequence Diagram (Figure 3.) at the last page shows how `ReminderCommand` works. `ReminderCommand` inherits the attributes and methods from the `ModuleCommand`, which is shown by the 1.1 arg arrow connecting `ReminderCommand` and `ModuleCommand`. The `ModuleCommand` calls itself as it uses its own attributes method as shown by 1.1.1 arg. The `ReminderCommand` also calls the `reminderList()` method in `PlannerUI` as shown by the 1.2 arrow. The `reminderList()` method, in turn calls itself by using the `showLine` method as shown by 1.2.1 arrow. The `ReminderCommand` calls its own method, such as `printEveryTenSec()` as shown by the 1.3 arrow. It then uses an object in `ModTimer` class, which is removed from the memory, as shown by the 1.3.1 arrow. This repeats three more times as shown by the 1.4 to 1.6 arrows. Lastly, the `ReminderCommand` calls itself by using the `killAllTimer()` method as shown by the 1.7 arrow.



**Figure 2.** Show Command Sequence Diagram.

**Figure 3.** Reminder Command Sequence Diagram.