Tuan Ding Wei - Project Portfolio

PROJECT: Modulo 

For the passionate.

I am excited to introduce you to Modulo in this document. I have enjoyed building it despite the many challenges and sleepless nights. I hope you will enjoy Modulo as much as I do!

Overview (What exactly is Modulo?)

Modulo is a NUS student life application specially designed for busy and motivated NUS students.

As busy student developers in NUS, we understand that it can be daunting to keep track of various tasks and concurrently do well for exams. Modulo features a one-stop app to keep track of one's timetable, finances and academic progress. It also includes a quiz feature that facilitates students with actively recalling their study concepts to improve their study outcomes. We are strong believers of passion and we believe that students with passion in their fields of studies can leverage on Modulo to be more productive and successful.

The user interacts with it using a CLI, and it features an outstanding, modern, user-friendly GUI created with JavaFX. It is written in Java, and has about 40 kLoC.

Summary of contributions

The Dashboard in Modulo Grades Tracker provides an overview of the list of modules to be taken with their accompany grades attained. It is also coupled with a statistical pie chart, degree classification, Cumulative Average Point (CAP) and Modular Credit (MC) displayed for student users.

Major Enhancement: `Add` Command and the primary features of Grades Tracker

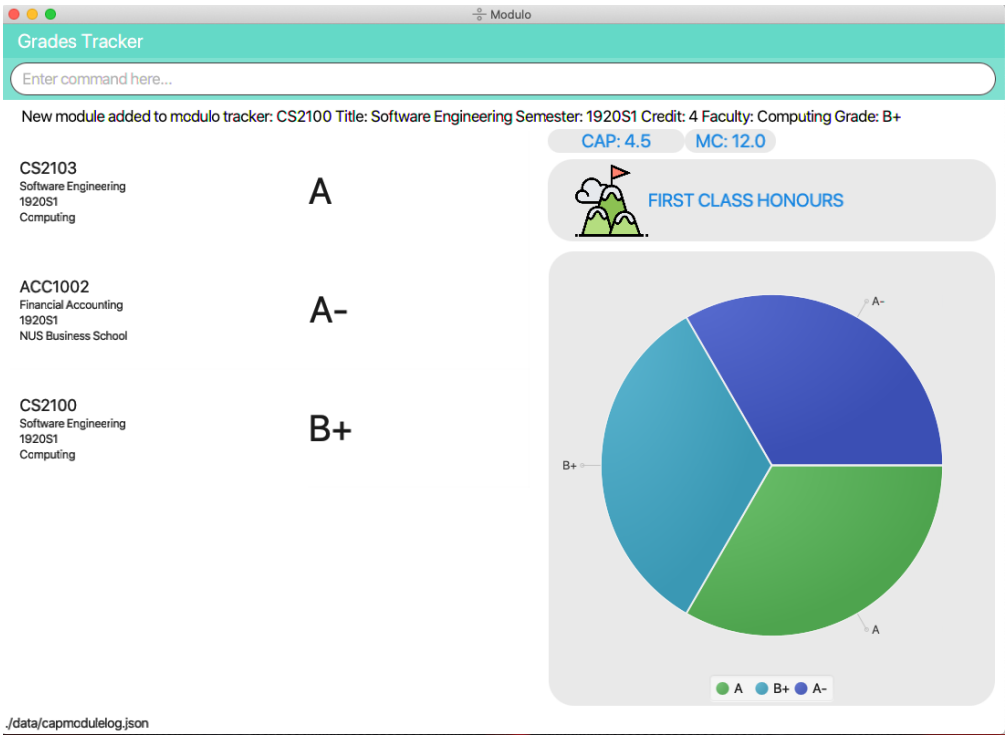
The `Add` was greatly enhanced to allow the student user to add new modules to Grades Tracker. The single command executes the codes to generate the pie chart, information display and degree classification. The person class had to be developed into module class. Whenever, a new add command is executed the 3 aforementioned components will refresh to show the updated information.

- What it does: The CAP tracker allows our user to keep track of his/her grades, overall Cumulative Average Point (CAP), MC taken and the degree classification (also known as "class") the user is in.
- Justification: This feature complements the function of Modulo and is a highly cohesive fit for the product as an all-in-one application for NUS students to manage their life in NUS. Being able to track grades is crucial as motivated NUS students have to know their progress and the

modules that have taken so far. For students in NUS who are well-concerned and motivated. The CAP Tracker allows students to keep track of their CAP and grades with ease. Moreover, there are no existing applications that facilitate such a need.

- Highlights: The implementation of the CAP feature was challenging but satisfying. It required in-depth design considerations around the students and a good knowledge of the grading system in NUS. I had to sought for advice from the academic department on various guidelines such as the range of approved modular credit. Defensive programming means that constraint requirements have to be well-thought of. Moreover, the implementation also required an overhaul of existing commands and codes. The structure remains intact but the code required heavy refractoring along with new implementations for the new feature. The single 'add' command initiates a series of major operations that generates the CAP, displays the right degree classifications and generates the pie chart. The implementation requires knowledge in object-oriented programming, data structure and algorithms. Many classes are interlinked hence increased the complexity of the modification and implementation.

Major Enhancement: Ravamped User-Interface (UI)



The UI is revamped to a new design that features modern curved edges and shadow-like background panes. Borders are taken away to promote a more cohesive flow of content. The aim is to deliver a "less is more" simplicity concept. Much attention is given to the effects of having a grey-like background matching with the black fonts and white background. It is designed in a way to display the services that Modulo offers without ambiguity, in order to draw our users' attention and keep them on our site.

Minor Enhancement: `Find` Command

The find command was not adequately powerful. It has a number of limitations such as not being able to work if the student user does not use very specific keywords that match the fields exactly.

The find command is enhanced to search for keywords in module code and module title. The enhanced search returns any module that contains the keyword in the module code and/or module title.

Special attention has to be given to the implementation as find returns only matching modules to the user but it would have caused the various pie chart, CAP, MC and degree classification information to change. Thus, the implementation requires the developer to isolate the updates from this command.

Minor Enhancement: `List` Command

The `List` command became necessary as the find command returns only matching modules to the student user. Thus, the `List` command will allow the user to return to the full list of modules again.

Minor Enhancement: `Sort` Command

The `Sort` command was created to complement the feature of Modulo Grades Tracker. The Modulo Grades tracker first order the modules by the sequence that they are added.

The `Sort` command sorts the modules in chronological order by the semesters that the modules were taken. The implementation requires knowledge of comparators and streams.

Minor Enhancement: `Delete` Command

As the Grades Tracker feature does not have indexes the original delete command became obsolete. The `Delete` command was modified to allow the user to delete any desired modules based simply by specifying the module code. This tremendously reduces the time needed to search for a module and identify the index if an index system is applied and when the module list becomes long.

Code contributed: Visit this link to view the codes that I have contributed.

Project management:

- Managed releases `v1.3` - `v1.4` (3 releases) on GitHub, with a v1.3.1 pre-release after a major revamp of the UI.
- Wrote additional tests for existing features to increase coverage from 46% to 49% (Pull requests #166) Major components of the tests have to be refracted as the major classes of the addressbook have been replaced.
- Documentation: Classes and methods are properly documented.
- PRs reviewed (with non-trivial review comments): #159
- Contributed to forum discussions (example: 1, 2, 3)
- Reported bugs and suggestions for other teams in the class (examples: Mortago, T.A.rence)
- Integrated a third party library (Tooltip) for the pie chart animation (#42)

Contributions to the User Guide

You may visit this link to view the entire User Guide. Following are the links to the PR:

UG Contribution 1 UG Contribution 2 UG Contribution 3

Introduction

Welcome to the User Guide for Modulo!

Modulo is an all-in-one student life application which is mainly targeted to NUS students. The app features four different functions which consists of weekly organizer, financial record system, quiz revision, and grades tracker.

Those who prefer to work with a Command Line Interface (CLI) might find using Modulo to be more straightforward than the usual Graphical User Interface (GUI) applications. The application has the visual benefits of a GUI but stands strongly rooted in command line usage. Modulo does not require an internet connection to run so there's no worry when the school wifi goes MIA yet again. The only time you need to be connected is at the start, when downloading the application. Look to [Quick Start] to find out how to get started on Modulo!

Grades Tracker

To enter the Module section please enter the command: `switch cap`

Table 1. Quick Reference sheet

Available Commands	Prefixes / Input required	Use
add	<m> MODULE_CODE, <t>TITLE, <s> SEMESTER, <c> CREDIT <g> GRADE	Adds a new module.
delete	MODULE_CODE	Deletes a specific module.
list		List all the modules, often required after the find command.
find	KEYWORD	Find a specific keyword within module code and/or module description.
sort		Display the modules in order of their semesters and academic years taken.
clear		Clear all existing modules.

Manually add module to record : `add`

The user can add new modules into the grade tracker.

The fields of a module consist of:
- `MODULE_CODE` The module code of the module. e.g. CS2103 Prefix: `<m>`
- `TITLE` The title of the module. E.g. Software Engineering Prefix: `<t>`.
- `SEMESTER` Semester includes 2 components that are seperated by a 'S' character. i.e. the academic year and semester period. E.g. 1920S1 Prefix: `<s>`.
- `CREDIT` Modular credit for the module. Module credit only accepts 2 to 23 credits. E.g. 4 Prefix: `<c>`.
- `GRADE` Grades for the attained for the module. Only permitted grades are accepted. E.g. A Prefix: `<g>`.

Table 2. Input constraints for Add command

Field	Validity
ModuleCode	Module code should contain a set of 4 integers and no excessive characters.
Semester	Semester should contain valid academic years and semester period. The academic year stated can only be +/-5 years than the current year. While the semester period allows input from 1 to 4.
Title	The title of the module is left for you to define. It is valid as long as it does not have special characters.
Credit	As stated by NUS, the range of modular credit ranges from 2 to 23.
Grade	Only NUS approved grades are allowed i.e. A+, A, A-, B+, B, B-, C+, C, D+, D and F

Table 3. Semester Period Classifications

Input	Semester Period
1	Semester 1
2	Semester 1
3	Special Term 1
4	Special Term 2

Format: `add MODULE_CODE MODULE_TITLE MODULE_YEAR_AND_SEMESTER MODULAR_CREDIT GRADE` E.g. `add <m>CS2103 <t>Software Engineering <s>1920S1 <c>4 <g>A`, `add <m>ACC1002 <t>Financial Accounting <s>1920S1 <c>4 <g>A`

Table 4. List of sample commands

Module	Command
CS2103	add <m>CS2103 <t>Software Engineering <s>1920S1 <c>4 <g>A
EC3343	add <m>CS3343 <t>International Finance I <s>2021S1 <c>4 <g>A
CS3202	add <m>CS3202 <t>Software Engineering Project II <s>2021S2 <c>4 <g>B+

Module	Command
CS3211	add <m>CS3211 <t>Parallel and Concurrent Programming <s>2021S2 <c>4 <g>A-
CS3218	add <m>CS3218 <t>Multimodal Processing in Mobile Platforms <s>2122S1 <c>4 <g>B

Delete module from record : `delete`

Delete a specific module from the list of modules by specifying the module code. The command is very strict on matching the module code given to the existing modules in modulo.

Format: `delete MODULE CODE` e.g. `delete CS2103`

Listing all tasks : `list`

Shows the list of all modules in the Grades Tracker.
Format: `list`

Sort all tasks in chronological order : `sort`

Sorts the list of modules in chronological order, the modules taken earlier to the modules taken later. The determination is based on the academic year then the semesters.
Format: `sort`

Locating module by module code or title : `find`

Finds the module with the matching module code or module description.

The enhanced search is case insensitive and matches any keyword in the module code or description e.g. `cs1010s` matches `CS1010S`

The search method adopts the inclusive OR search method that returns any modules with the keyword. e.g. searching for `prog metho` will return `Programming Methodology I` and `Programming Methodology II`.

Format: `find KEYWORD` e.g. `find CS1010S`, `find Programming`

Clear all modules the record : `clear`

Deletes all modules in the record. The user can avoid clearing modules one-by-one when large number of modules have to be cleared. When the single keyword `clear` is entered, Modulo would recognise it as a command to clear all the modules. The action is irreversible!

Format: `clear` e.g. `clear`

Categorisation [Coming in V2.0]

Categorisation of the modules in their respective semesters and predictive text.

Contributions to the Developer Guide

You may visit this link to the entire developer guide. Following are the links to the PR:

DG Contribution 1 DG Contribution 2

Architecture

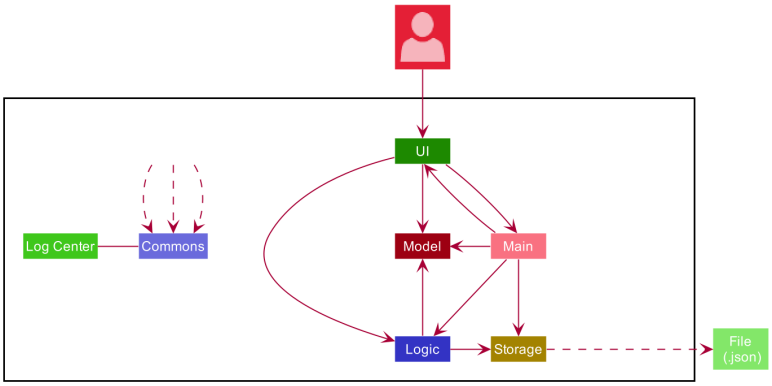


Figure 1. Architecture Diagram for a feature

`Main` has two classes called `Main` and `MainApp`. It is responsible for,

- At app launch: Initializes the components in the correct sequence, and connects them up with each other.
- At shut down: Shuts down the components and invokes cleanup method where necessary.

`Commons` represents a collection of classes used by multiple other components. The following class plays an important role at the architecture level:

- `LogsCenter` : Used by many classes to write log messages to the App's log file.

The rest of the App covers the four features in *Modulo* (Calendar, Cap, Quiz, Finance). All the features have the very similar structure, each consisting of four components.

- **`UI`**: The UI of the App.
- **`Logic`**: The command executor.
- **`Model`**: Holds the data of the App in-memory.
- **`Storage`**: Reads data from, and writes data to, the hard disk.

Each of the four components

- Defines its *API* in an `interface` with the same name as the Component.
- Exposes its functionality using a `{Component Name}Manager` class.

For example, the `Logic` component (see the class diagram given below) defines it's API in the `Logic.java` interface and exposes its functionality using the `LogicManager.java` class.

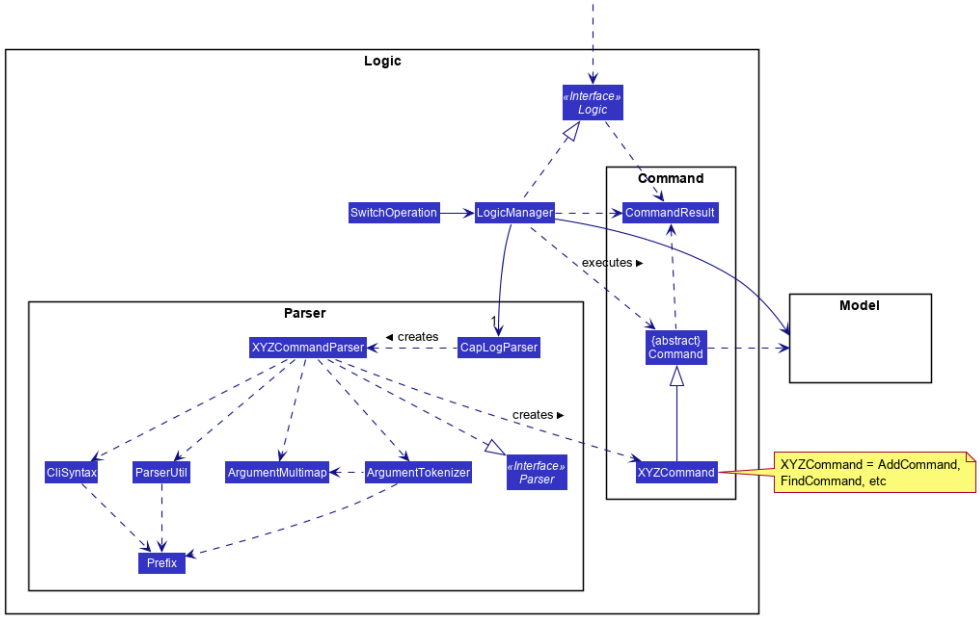


Figure 2. Class Diagram of the Logic Component for the feature Cap

Rationale for having multiple `UI`, `Logic`, `Model`, `Storage` classes in *Modulo*

The features in *Modulo* are quite distinct and having minimal overlaps, each feature has its own data file to read and modify. The logic to execute commands are also handled differently in each feature. As such, our decision to have 4 components for each feature was to reduce coupling and allow flexibility in our implementations. This means that the same `add` command is now able to trigger a different sequence of actions depending on the feature the user is currently in. This architecture style allows each feature to be developed independently, free from the constraints set by other features. Each time a user switches to another feature, the `Ui`, `Logic`, `Model` and `Storage` classes will be replaced by that of the new feature.

How the architecture components interact with each other

The *Sequence Diagram* below shows how the components interact with each other for the scenario where the user issues the command `delete CS2103` when in the feature *Module*.

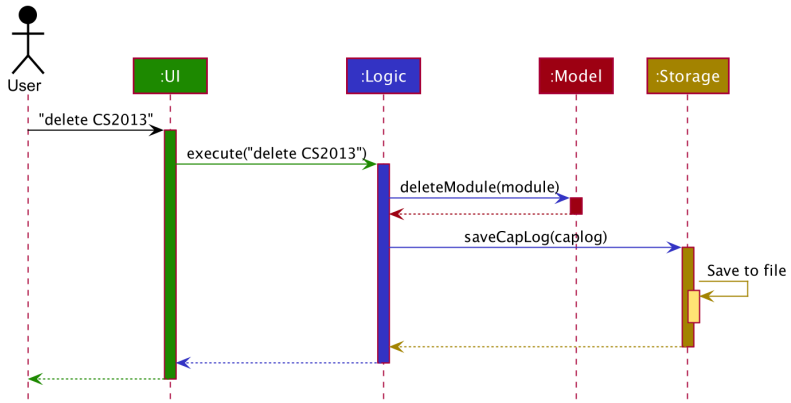


Figure 3. Component interactions for `delete CS2103` command in Cap feature

The sections below give more details of each component.

Grades Tracker

Implementation

The following activity diagram displays the sequence of events after the user inputs to add a new module.

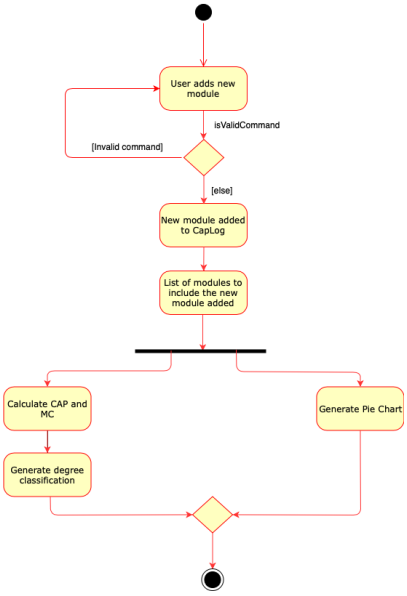


Figure 4. Class Diagram of Module

For the user to add new modules to the Modulo Grades Tracker, the `Module` has to be designed to satisfy the requirements of a module and the needs of the user.

- Each `Module` consists of `ModuleCode`, `Title`, `Semester`, `Credit`, `Grade`.
- All fields in `Module` are unique.
- Each class has their respective getter methods and validity-check method.

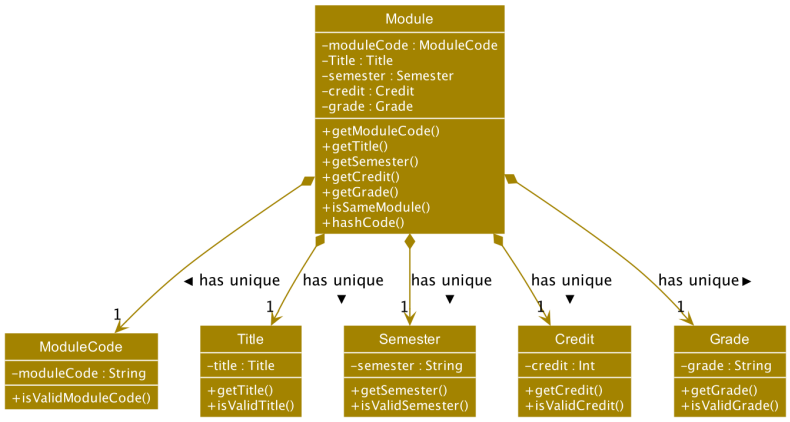


Figure 5. Class Diagram of Module

Implementation of Module commands

Module class supports multiple commands. They include:

- `AddCommand` - Adds a module to the Grades Tracker.

- `DeleteCommand` - Removes a module from the Grades Tracker by specifying the module code.
- `FindCommand` - Find and returns any modules with `ModuleCode` and/or `Title` that matches the keyword specified by the user.
- `SortCommand` - Sorts the modules in chronological order based on the `Semester` the module was taken.
- `ListCommand` - List all the modules. Often required after the find command.

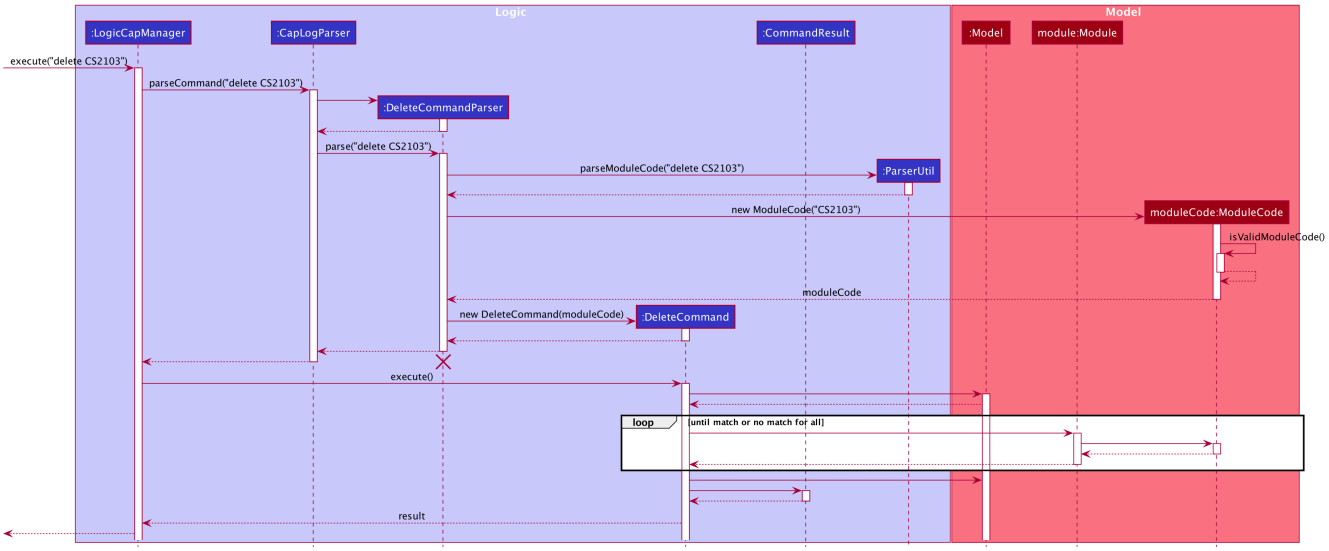


Figure 6. Sequence Diagram of DeleteCommand

NOTE	The lifelines for `DeleteCommandParser` should end at the destroy marker (X) but due to a limitation of PlantUML, the lifelines reaches the end of diagram.

After a successful execution, the module with the specified module code will be deleted from Modulo Grades Tracker.

Design Consideration

All fields should not be have special characters or left blank as the details are important to the implementation and the user experience.

The class diagram below gives an overview of the Module class.

The search for the matching module code is implemented with a linear search. With 70 modules in Modulo, the linear search is still responsive and there were not observable lags.

Alternative

An alternative would be to use a search algorithm that comprises a Hash set and linear search.