GAZEEEBO - Developer Guide

Content page

1. Introduction

1.1 Purpose

This document explains the design, implementation, and testing for the planner, Gazeeebo.

1.2 Target User Profile

The target user profile is undergraduate Computer Engineering students from the National University of Singapore (NUS).

1.3 Overview of Product

The product, Gazeeebo, is an integrated planner for NUS CEG students. It aims to aid them in drafting their 4-year study plan in NUS. The main features of Gazeeebo are: a four-year module planner, a specialization and prerequisites help feature and a CAP calculator.

1.4 Markups used

- Words in `Consolas` are buttons to be clicked or files.
- Words in `Courier New` font are commands to be typed into a terminal, for example in Command Prompt on Windows.
- Words highlighted in grey are classes or components.

2. Setting up

2.1. Prerequisite

- JDK 11
- IntelliJ IDE

2.2. Setting up the project on your computer

1. Fork this repo, and clone the fork to your computer.
2. Open IntelliJ (if you are not in the welcome screen, click `File` > `Close Project` to close the existing project dialog first).
3. Set up the correct JDK version for Gradle.

   a. Click `Configure` > `Project Defaults` > `Project Structure.`
   b. Click `New...` and find the directory of the JDK.
4. Click `Import Project.`
5. Locate the `build.gradle` file and select it. Click `OK.`
6. Click `Open as Project.`
7. Click `OK` to accept the default settings.
8. Open a console and run the command `gradlew processResources` (Mac/Linux: `./gradlew processResources`). It should finish with the 'BUILD SUCCESSFUL' message.
   This will generate all the resources required by the application and tests.
9. Open `Gazeeebo.java` and check for any code errors.

2.3 Verifying the setup

1. Run the Gazeeebo class and try a few commands.
2. Run the tests to ensure they all pass.

2.4 Configuration to do before writing code

This project follows oss-generic coding standards. IntelliJ's default style is mostly compliant with ours but it uses a different import order from ours. To rectify,

1. Go to `File` > `Settings...` (Windows/Linux), or `IntelliJ IDEA` > `Preferences...` (macOS)
2. Select `Editor` > `Code Style` > `Java`
3. Click on the `Imports` tab to set the order
   - For `Class count to use import with '*'` and `Names count to use static import with '*'`: Set to `999` to prevent IntelliJ from contracting the import statements
   - For `Import Layout`: The order is `import static all other imports`, `import java.*`, `import javax.*`, `import org.*`, `import com.*`, `import all other imports`. Add a `<blank line>` between each `import`

Optionally, you can follow the UsingCheckstyle.adoc document to configure Intellij to check style-compliance as you write code.

3. Design

3.1. Architecture

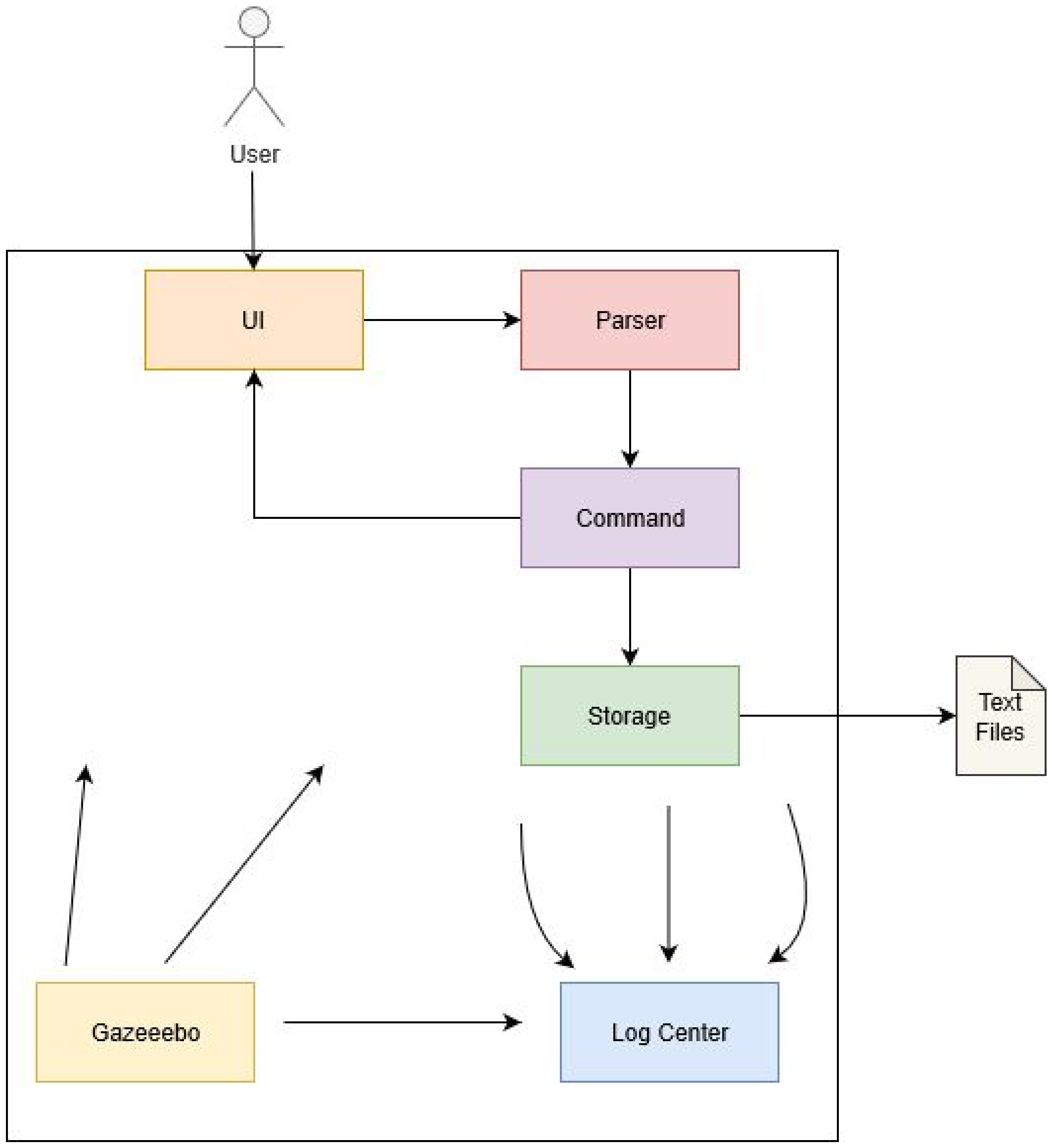


Figure 3.1.1 Architecture Diagram

The *Architecture Diagram* given above in Figure 3.1.1 explains the high-level design of the App. Given below is a quick overview of each component.

Gazeeebo contains the main class for the App. It is responsible for,

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

The rest of the App consists of four components.

- UI: The UI of the App.

- Command: The command executor.
- Parser: The decoder that decodes the user's input.
- Storage: Reads data from, and writes data to the hard disk.

3.2. UI Component

The UI Component mainly deals with interactions with the user. It also plays a part in the initialisation of the program by printing the welcome message and main menu page. This component only has one class Ui.

3.3. Parser Component

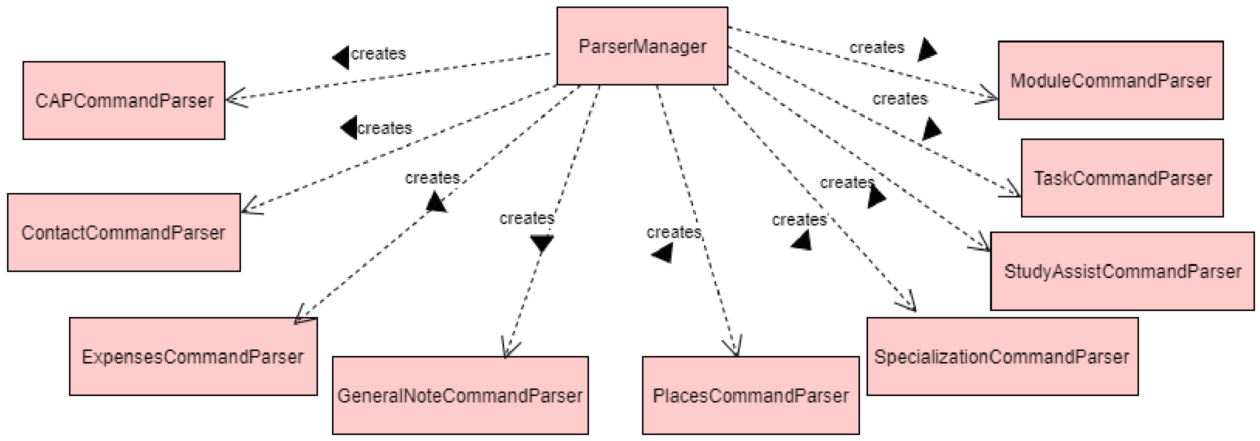


Figure 3.3.1. Class Diagram of Parser Component

- The Parser Component uses ParserManager to parse the user's input into different command parsers.
- The respective command parsers will then parse the user's input again into different commands.

3.4. Command Component

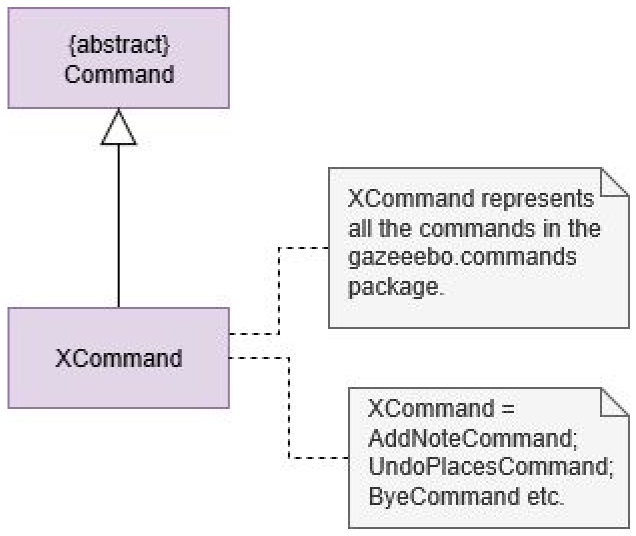

{abstract}
Command
XCommand
XCommand represents all the commands in the gazeeebo.commands package.
XCommand = AddNoteCommand; UndoPlacesCommand; ByeCommand etc.

Figure 3.4.1 Class diagram of Command Component

- All the classes in the gazeeebo.commands package are inherited from the Command class, which is an abstract class.
- The main method of Command is the execute() method which is used to implement the main code for each command.
- The Command Component is the main component that implements all the features in Gazeeebo.

3.5. Storage Component

The Storage Component contains the read and write methods of the various features. It helps to initialise the program by loading stored data into the program and it writes data created by the user to text files. Below is a list of classes in the storage component:

- CapPageStorage
- ContactPageStorage
- NotePageStorage
- NoteStorage
- PasswordStorage
- PlacesPageStorage
- TasksPageStorage
- Storage
- CompletedElectivesStorage
- ExpensePageStorage
- SpecializationPageStorage

- StudyAssistPageStorage
- TriviaStorage

The first time the jar file is run, the Storage class creates text files outside the jar file and copies the preloaded data to those text files. This enables the program to be able to save data to the text files.

4. Implementation

This section describes some noteworthy details on how certain features are implemented.

4.1. Password Feature

4.1.1 Implementation

The Password feature is facilitated by Ui. When Gazeeebo is initially run, the system will be prompted for the password. The default password is jjjry. The user is able to change the password by calling out ChangePasswordCommandParser. The password is saved by calling out ChangePasswordStorage and writes the new password in the reverse direction. Eg. password = jason, txt file change to nosaj.

Here is a class diagram to show the structures that implement the Password and Change Password features:

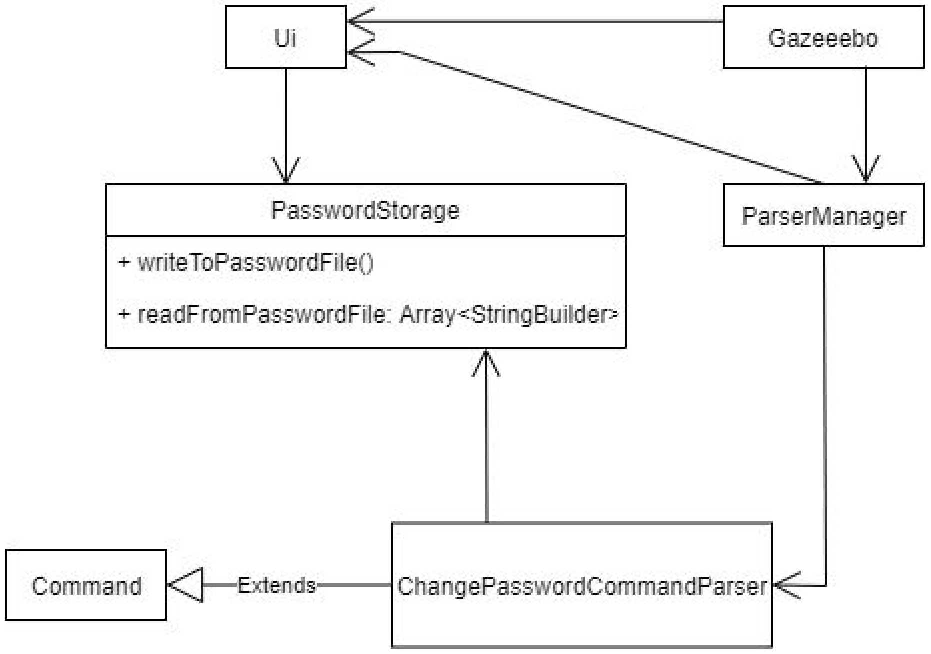


Figure 4.1.1.1. Class diagram for Password page feature

The following steps show a usage scenario:

When Gazeeebo is started up:

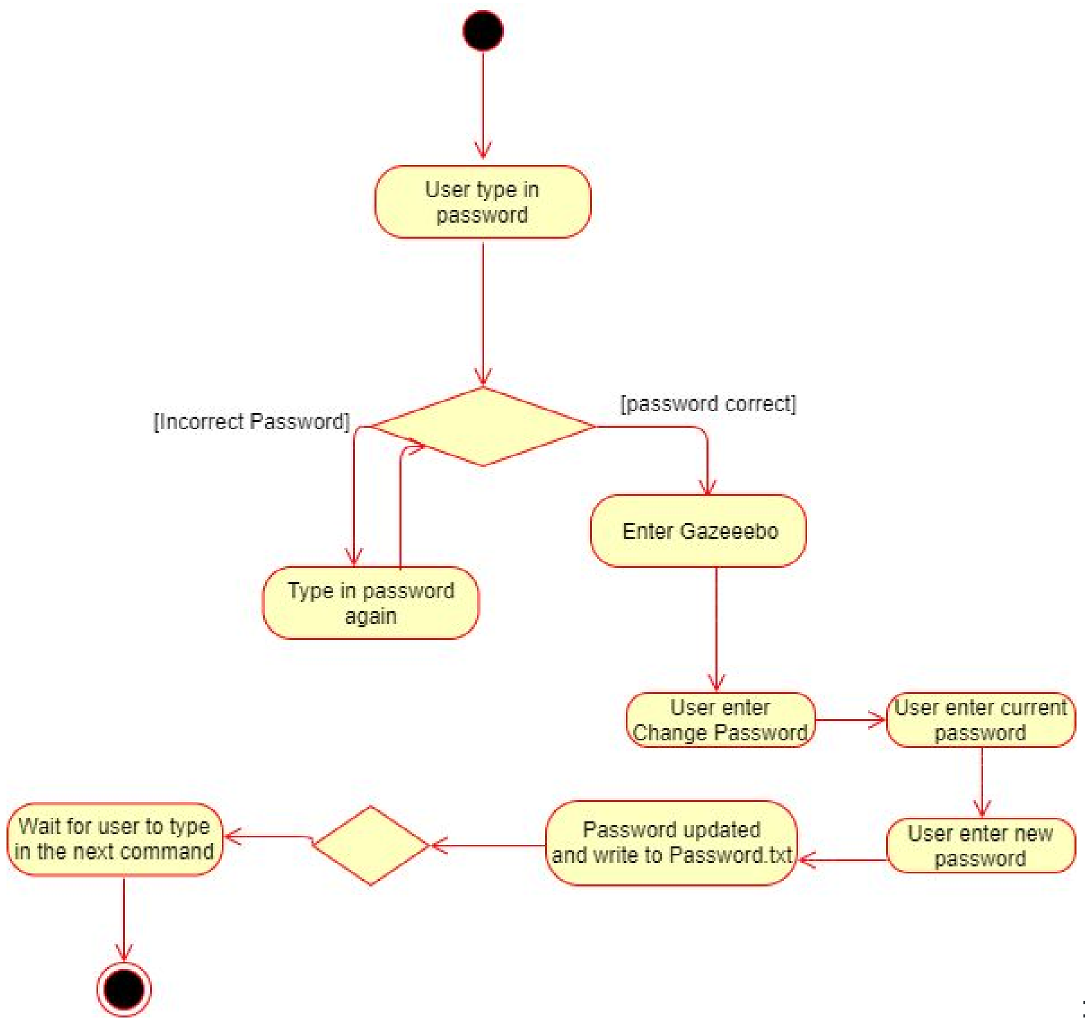
Step 1. User will have to input the password to enter the system. User can input `jjjry` to enter the system. `jjjry` is the default password.

When an ChangePasswordCommandParser is called:

Step 1. User will have to input the current password and when it matches, he is able to input a new password.

Step 2. The PasswordStorage is called, updating the new password in the Password.txt.

The following activity diagram summarizes what happens when Gazeeebo is started up and get prompted by the password and when the user wants to change the password.


User type in password
[Incorrect Password]
[password correct]
Type in password again
Enter Gazeeebo
User enter Change Password
User enter current password
User enter new password
Password updated and write to Password.txt
Wait for user to type in the next command

:

Figure 4.1.1.2. Activity diagram for Password and ChangePasswordParse

4.1.2 Design Considerations

Aspect: Data structure to support Password

- Alternative 1(current choice): Using TreeMap to store all the module's information (semester number, code, credit, grade).
  - Pros: TreeMap is has a natural ordering and the keys are sorted, thus reducing the need to sort the map when it is listed out.
  - Cons: TreeMap's time complexity of insertion and delete is O(log(n)), which is slower than HashMap.
- Alternative 2: Using a HashMap to store names and numbers
  - Pros: HashMap has a time complexity of O(1) for insertion and deletion, which is faster than TreeMap.
  - Cons: Null values/keys are allowed in a HashMap. Requires an additional validation to ensure users do not input null values/keys.

4.2. Cap Calculator Feature

4.2.1 Implementation

The CAP feature is facilitated by CapCommandParser. The user will go into the CAP page by typing `cap` on the main page The CAP feature is mainly implemented by AddCapCommand, DeleteCapCommand, ListCapCommand, FindCapCommand, CalculateCapCommand, and ConvertGradeToScoreCommand take care of the logic for the CAP page, which is implemented by CapCommandParser class. In turn, CapCommandParser implements the following operations:

- CapCommandParser#add() - Adds a module into the caplist.
- CapCommandParser#delete() - Delete a module from caplist.
- CapCommandParser#find() - Find a module from caplist based on the name.
- CapCommandParser#list() - list modules from caplist and it shows the cap.
- CapCommandParser#calculatecap() - calculate the CAP in the semester or for all modules
- CapCommandParser#convertergradetoscore() - converts alphabetical grade to numerical score

Here is a class diagram to show the structure of the classes that implement the CAP page feature:

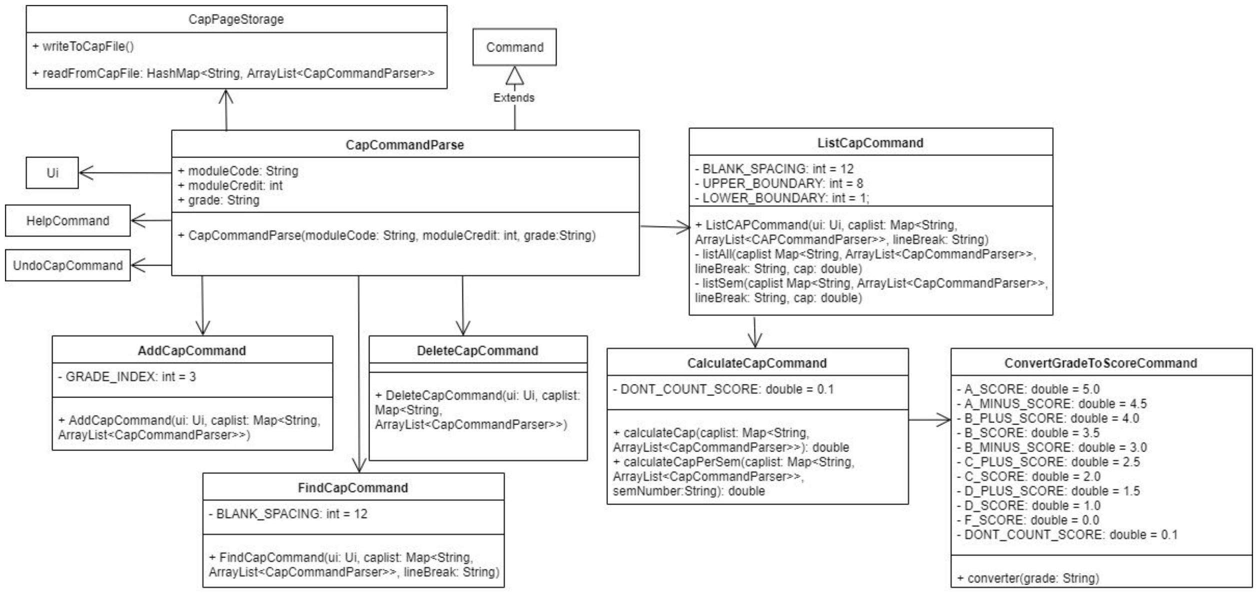

CapPageStorage
+ writeToCapFile()
+ readFromCapFile: HashMap<String, ArrayList<CapCommandParser>>
Command
Extends
CapCommandParse
+ moduleCode: String
+ moduleCredit: int
+ grade: String
+ CapCommandParse(moduleCode: String, moduleCredit: int, grade:String)
Ui
HelpCommand
UndoCapCommand
ListCapCommand
- BLANK_SPACING: int = 12
- UPPER_BOUNDARY: int = 8
- LOWER_BOUNDARY: int = 1;
+ ListCAPCommand(ui: Ui, caplist: Map<String, ArrayList<CAPCommandParser>>, lineBreak: String)
- listAll(caplist Map<String, ArrayList<CapCommandParser>>, lineBreak: String, cap: double)
- listSem(caplist Map<String, ArrayList<CapCommandParser>>, lineBreak: String, cap: double)
AddCapCommand
- GRADE_INDEX: int = 3
+ AddCapCommand(ui: Ui, caplist: Map<String, ArrayList<CapCommandParser>>)
DeleteCapCommand
+ DeleteCapCommand(ui: Ui, caplist: Map<String, ArrayList<CapCommandParser>>)
CalculateCapCommand
- DONT_COUNT_SCORE: double = 0.1
+ calculateCap(caplist: Map<String, ArrayList<CapCommandParser>>): double
+ calculateCapPerSem(caplist: Map<String, ArrayList<CapCommandParser>>, semNumber:String): double
ConvertGradeToScoreCommand
- A_SCORE: double = 5.0
- A_MINUS_SCORE: double = 4.5
- B_PLUS_SCORE: double = 4.0
- B_SCORE: double = 3.5
- B_MINUS_SCORE: double = 3.0
- C_PLUS_SCORE: double = 2.5
- C_SCORE: double = 2.0
- D_PLUS_SCORE: double = 1.5
- D_SCORE: double = 1.0
- F_SCORE: double = 0.0
- DONT_COUNT_SCORE: double = 0.1
+ converter(grade: String)
FindCapCommand
- BLANK_SPACING: int = 12
+ FindCapCommand(ui: Ui, caplist: Map<String, ArrayList<CapCommandParser>>, lineBreak: String)

Figure 4.2.1.1. Class diagram for CAP page feature

The following diagram shows the interactions for the method call execute() on a CapCommandParser object:

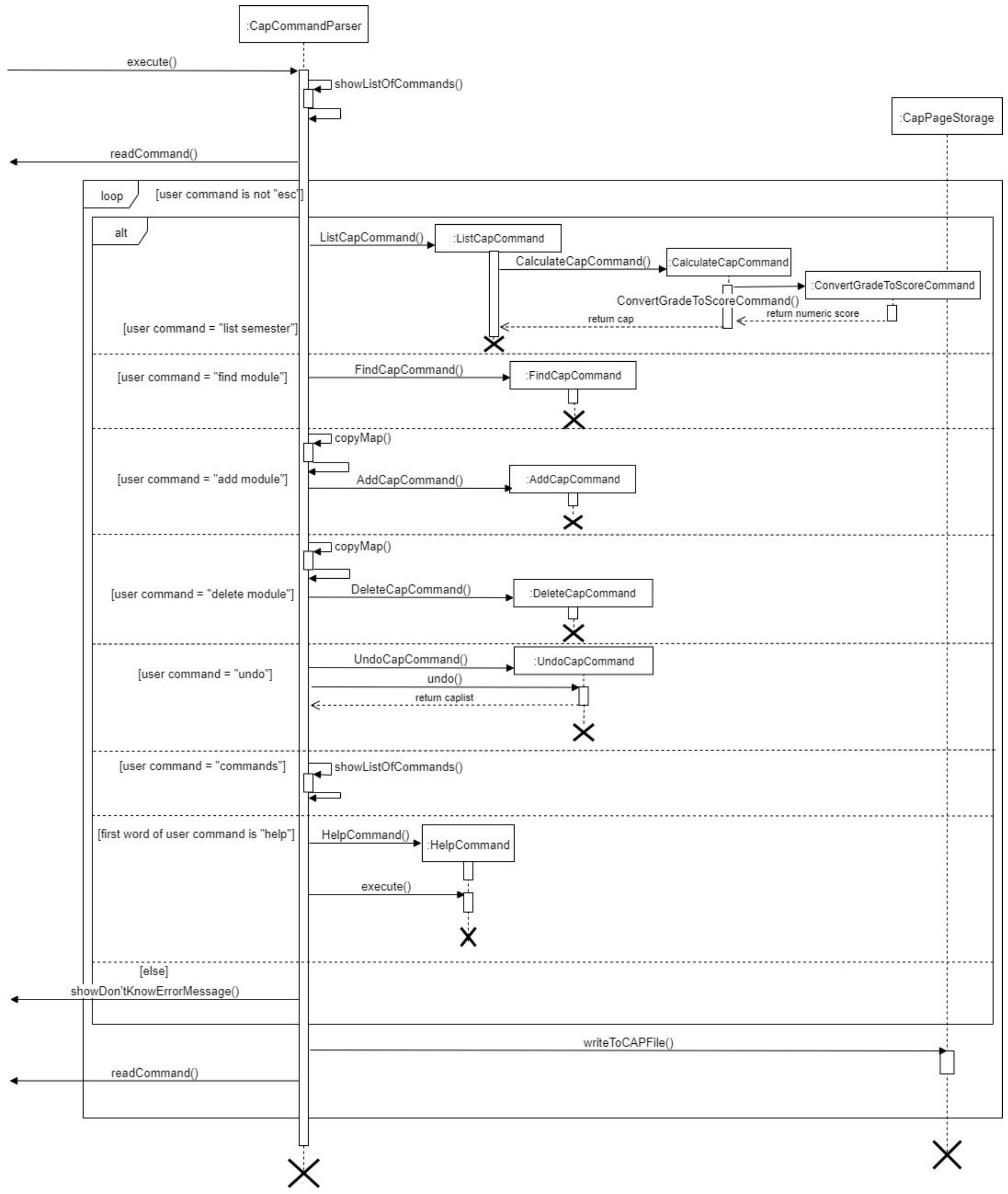


Figure 4.2.1.2. Sequence diagram for method execute() of CapCommandParse.

The following steps show a usage scenario:
When an AddCapCommand is called:

Step 1. User will have to input the name and the user wants to add. User can choose a two step process by entering `add` and the `sem number,module’s code,module’s credit,module’s grade` or one step process by entering `add sem number,module’s code,module’s credit,module’s grade.`

Step 2. The new module will be added to `Map<String, ArrayList<CAPCommandParser>> caplist.`

When an DeleteCapCommand is called:

Step 1. User will have to input the name the user wants to delete. User can choose a two step process by entering `delete` and the `module’s code` or one step process by entering `delete module’s code.`

Step 2. The module specific will be removed from `Map<String, ArrayList<CAPCommandParser>> caplist.`

Step 3. If there is no such name in the Map<String,String> `contactList`, a “not found” message will be shown.

When a ListCapCommand is called:

Step 1. User will have to input which semester the user wants to list. User can choose a two step process by entering `list` and the `semester number` or one step process by entering `list sembester number.`

Step 2. CalculateCapCommand is called to calculate the cap of the semester. In CalculateCapCommand, ConvertGradeToScoreCommand is called to convert the user’s alphabetical grade to numerical grade.

Step 3. The `Map<String, ArrayList<CAPCommandParser>> caplist` and the CAP will be printed out.

When a FindCapCommand is called:
Step 1.User inputs the module code the user wants to find. User can choose a two step process by entering `find` and the `module code` one step process by entering `find module’s code.`
Step 2. The module’s code information will be shown.
Step 3. If there is no such module code in the The `Map<String, ArrayList<CAPCommandParser>> caplist`, a “not found” message will be shown.

The following activity diagram summarizes what happens when a user executes a new command:

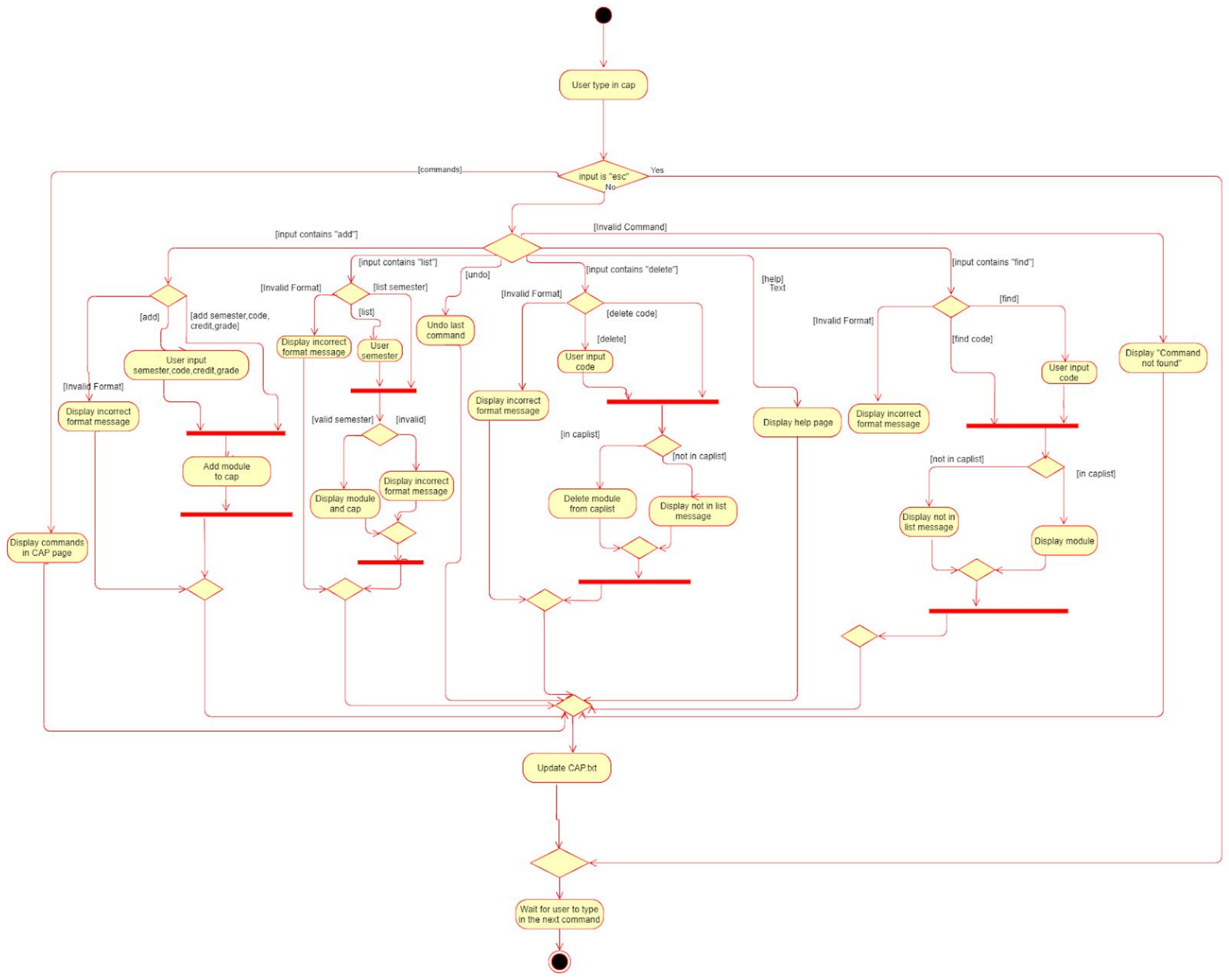


Figure 4.2.1.3. Activity diagram for CAP page.

4.2.2 Design Considerations

Aspect: Data structure to support CAP

- Alternative 1(current choice): Using TreeMap to store all the module's information (semester number, code, credit, grade).
    - Pros: TreeMap is has a natural ordering and the keys are sorted, thus reducing the need to sort the map when it is listed out.
    - Cons: TreeMap's time complexity of insertion and delete is O(log(n)), which is slower than HashMap.
- Alternative 2: Using a HashMap to store names and numbers

- Pros: HashMap has a time complexity of O(1) for insertion and deletion, which is faster than TreeMap.
- Cons: Null values/keys are allowed in a HashMap. Requires an additional validation to ensure users do not input null values/keys.

4.3. Calendar View Feature

4.3.1 Implementation

The Calendar view feature is facilitated by CalendarView. It contains the following methods:

- CalendarView#isLeapYear(year) - Check if the year passed in as a parameter is a leap year.
- CalendarView#StartDay(month,day,year) - Check the day of the 1st of each month.
- CalendarView#MonthlyView(list) - Print out the monthly calendar onto the command line interface and if there is an event on a particular day, a '*' will be placed beside its' date.
- CalendarView#AnnualView(list) - Print out the annual calendar onto the command line interface and if there is an event on a particular day, a '*' will be placed beside the date.

Here is a class diagram to show the structure of the classes that implement the Calendar View feature:

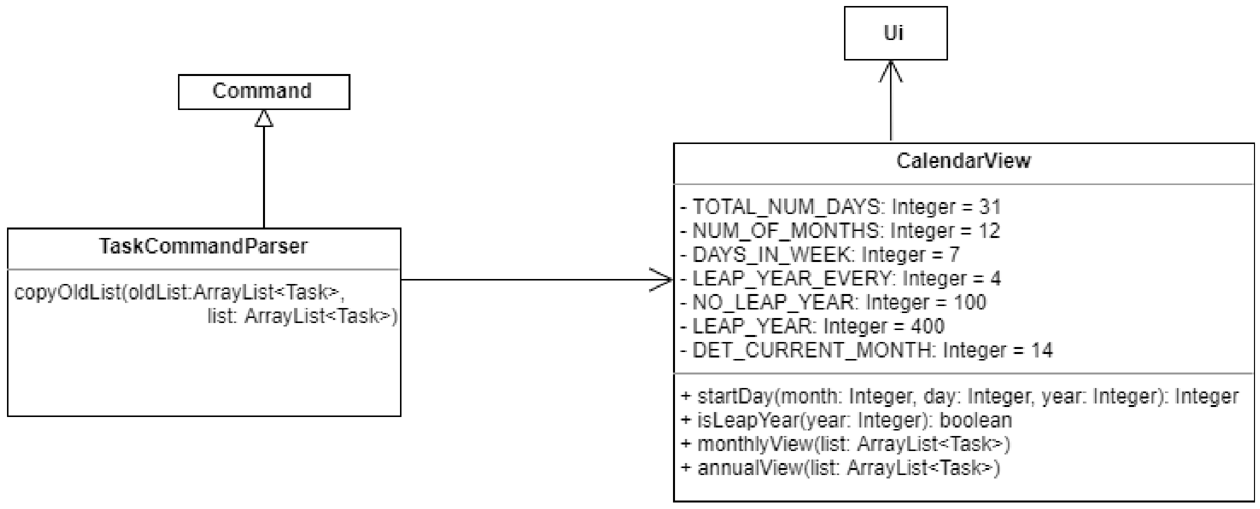

Ui
Command
TaskCommandParser
copyOldList(oldList:ArrayList<Task>, list: ArrayList<Task>)
CalendarView
- TOTAL_NUM_DAYS: Integer = 31
- NUM_OF_MONTHS: Integer = 12
- DAYS_IN_WEEK: Integer = 7
- LEAP_YEAR_EVERY: Integer = 4
- NO_LEAP_YEAR: Integer = 100
- LEAP_YEAR: Integer = 400
- DET_CURRENT_MONTH: Integer = 14
+ startDay(month: Integer, day: Integer, year: Integer): Integer
+ isLeapYear(year: Integer): boolean
+ monthlyView(list: ArrayList<Task>)
+ annualView(list: ArrayList<Task>)

Figure 4.3.1.1 Class diagram for Calendar View feature

The following steps show how the Monthly View Calendar is built :

1. The constructor of CalendarView is called, thereby creating a new instance of CalendarView
2. MonthlyView method is called which calls get(calendar.MONTH), get(calendar.YEAR) and get(calendar.DATE), from an instance of Calendar.
3. It then calls a method isLeapYear(year) to determine if the current year is a leap year or not.
4. Next, it calls a method StartDay(month,day,year) to determine the day of the first day of the month (e.g monday)
5. Finally, the calendar is printed onto the command line interface

The sequence diagram below outlines MonthlyView process:

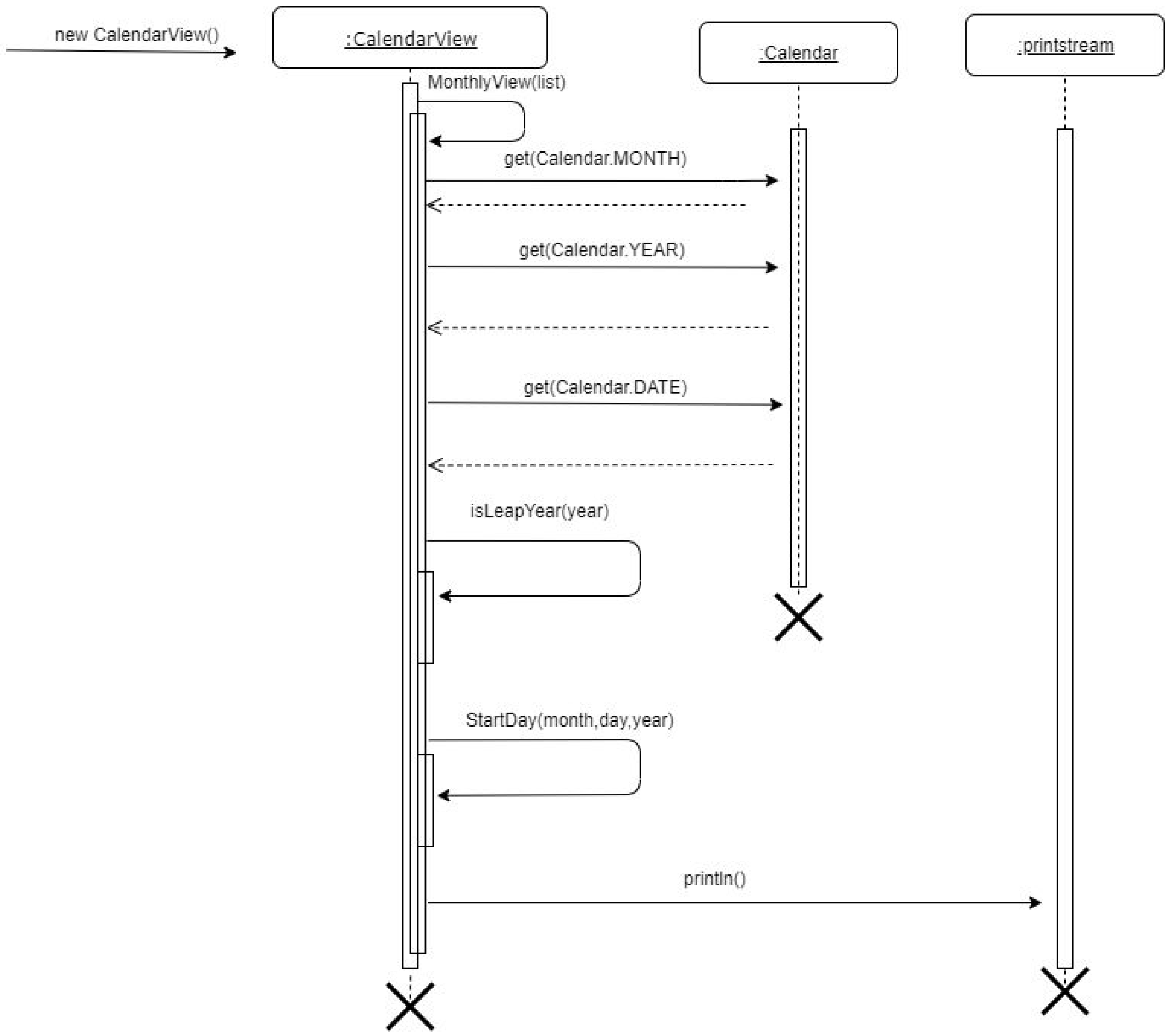


Figure 4.3.1 UML Sequence Diagram of CalendarView

The following activity diagram summarizes what happens when a user executes a new command:

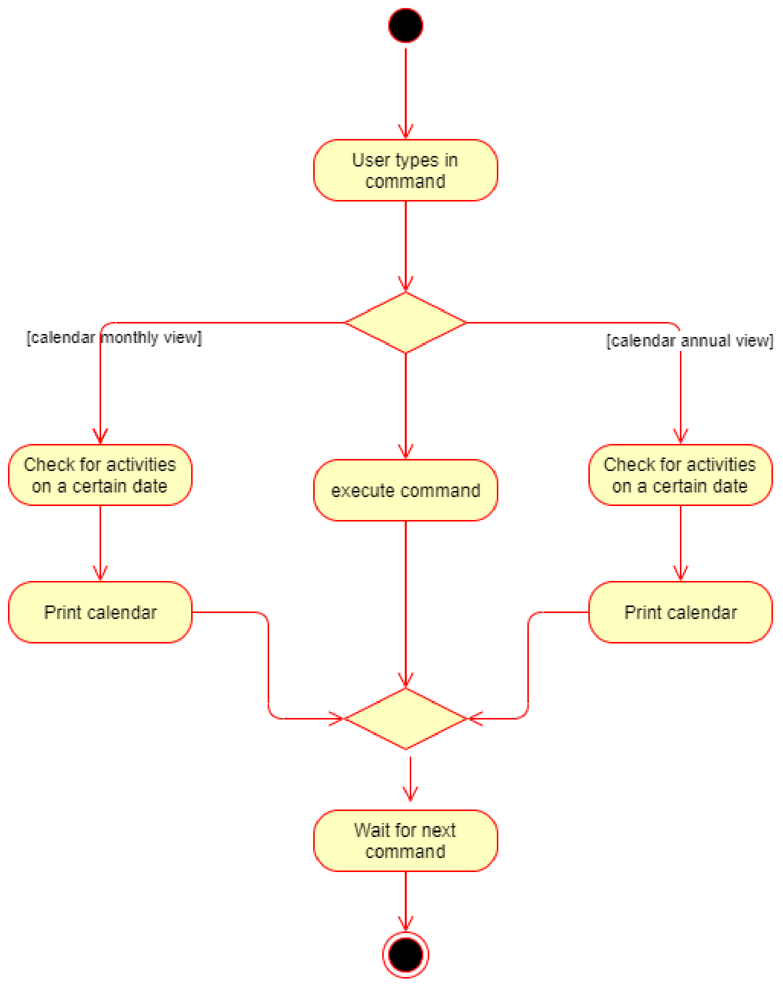


Figure 4.3.2 UML Activity Diagram of Calendar view feature

4.3.2 Design Considerations

Aspect: Displaying task on each day

- Alternative 1(current choice): Use a marker to demarcated days with tasks on the calendar.
    - Pros: The design of the calendar would look more aesthetic and compact. This reduces the need for scrolling to view the entire month.
    - Cons: Not as user friendly, additional commands needed to view tasks on the particular day.
- Alternative 2: Show every task on that day on the calendar
    - Pros: Users will be able to view every single task on each day.
    - Cons: Printing the calendar would be an issue as the calendar would require more space on the command line interface.

Aspect: Showing the current date

- Alternative 1(current choice): Print the current date between “|”.
  - Pros: Works for all types of command line interfaces.
  - Cons: Might disrupt the columns of the calendar causing the dates to be printed slightly out of position.
- Alternative 2: Change the colour of the date being printed.
  - Pros: Aesthetically more pleasing using colours instead of symbols to demarcate the current date.
  - Cons: The use of ANSI escape codes to print coloured numbers on the command line does not work on Windows Command Prompt.

4.4. Contacts Feature

4.4.1 Implementation

The Contacts feature is facilitated by ContactCommand. It extends Command and stores the contacts’ information internally as an `Map<String, String>` contactList and also externally as Contact.txt. Additionally, it implements the following operations:

- ContactCommand#add() - Adds a contact into the contactList.
- ContactCommand#delete() - Delete a contact from contactList.
- ContactCommand#find() - Find a contact from contactList based on the name.
- ContactCommand#list() - list contacts from contactList.

Here is a class diagram in Figure 4.4.1.1 to show the structure of the classes that implement the Contact page feature:

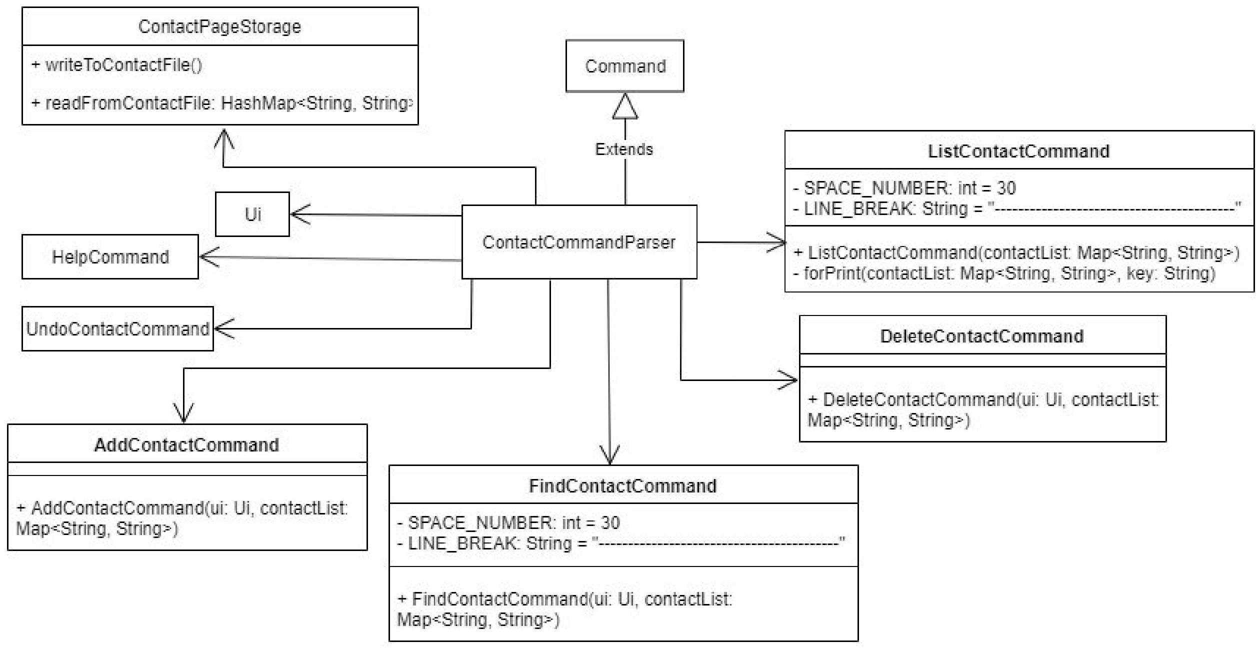

ContactPageStorage
+ writeToContactFile()
+ readFromContactFile: HashMap<String, String>
Command
Extends
ListContactCommand
- SPACE_NUMBER: int = 30
- LINE_BREAK: String = "--"
+ ListContactCommand(contactList: Map<String, String>)
- forPrint(contactList: Map<String, String>, key: String)
Ui
HelpCommand
ContactCommandParser
UndoContactCommand
DeleteContactCommand
+ DeleteContactCommand(ui: Ui, contactList: Map<String, String>)
AddContactCommand
+ AddContactCommand(ui: Ui, contactList: Map<String, String>)
FindContactCommand
- SPACE_NUMBER: int = 30
- LINE_BREAK: String = "--"
+ FindContactCommand(ui: Ui, contactList: Map<String, String>)

Figure 4.4.1.1. Class diagram for Contact page feature

The following diagram in Figure 4.4.1.2 shows the interactions for the method call execute() on a ContactCommandParser object:

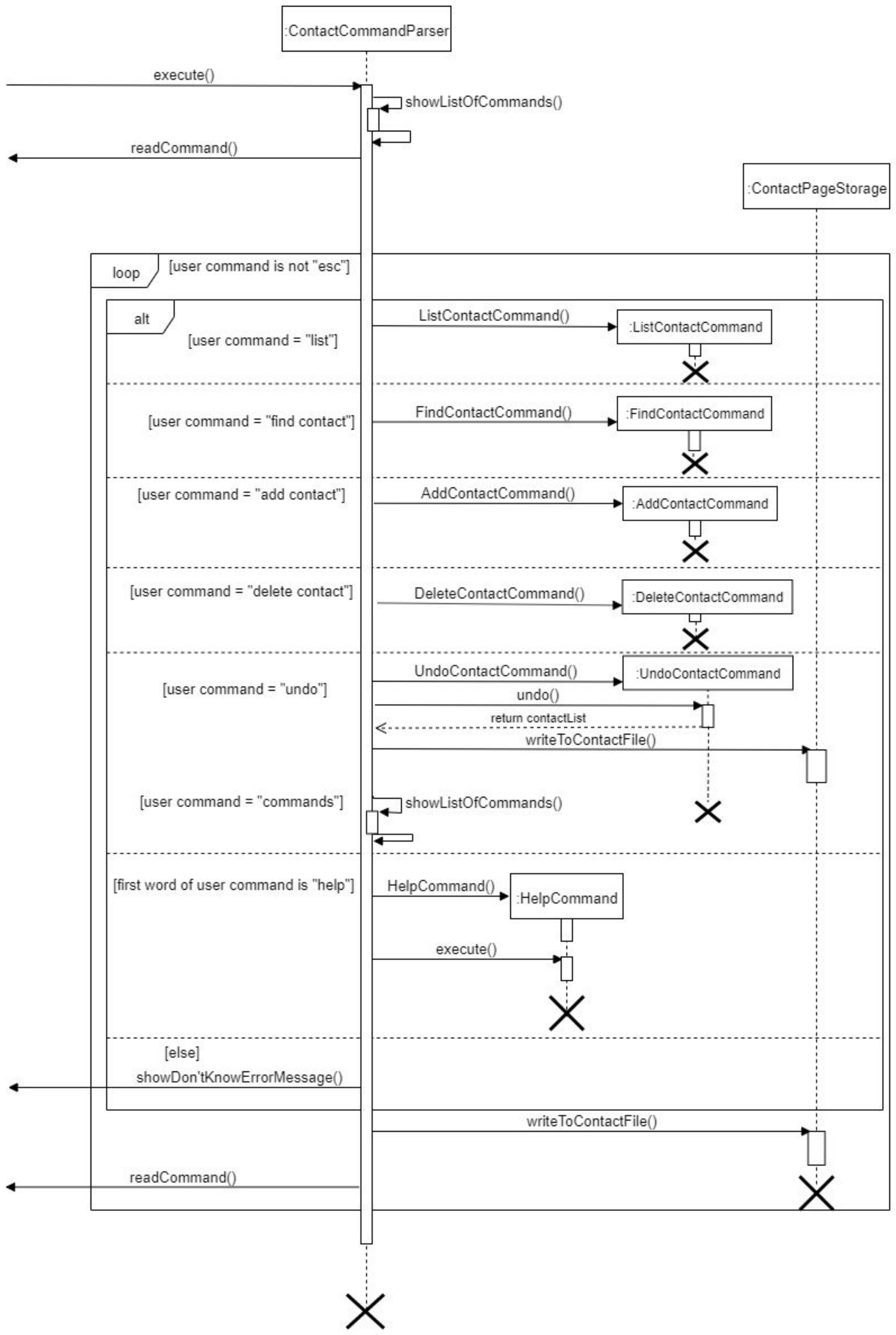


Figure 4.4.1.2. Sequence diagram for Contact page.

When an AddContactCommand is called:

Step 1. User will have to input the name and number. User can choose a two step process by entering `add` and the `name,number` or one step process by entering `add name,number.`

Step 2. The new contact will be added to `Map<String, String> contactList`.

When an DeleteContactCommand is called:

Step 1. User will have to input the name they want to delete. User can choose a two step process by entering `delete` and the `name` or one step process by entering `delete name.`

Step 2. The name specific will be removed from `Map<String, String> contactList`.

When a ListContactCommand is called:

Step 1. User inputs `list` into the command line.

Step 2. The `Map<String, String> contactList` will be printed out.

When a FindContactCommand is called:

Step 1.User inputs the name they want to find. User can choose a two step process by entering `find` and the `name` or one step process by entering `find name.`

Step 2. The number that have similar names will be printed out.

Step 3. If there is no such name in the Map<String,String> `contactList`, a "not found" message will be shown.

The following activity diagram in Figure 4.4.1.3 summarizes what happens when a user executes a new command:

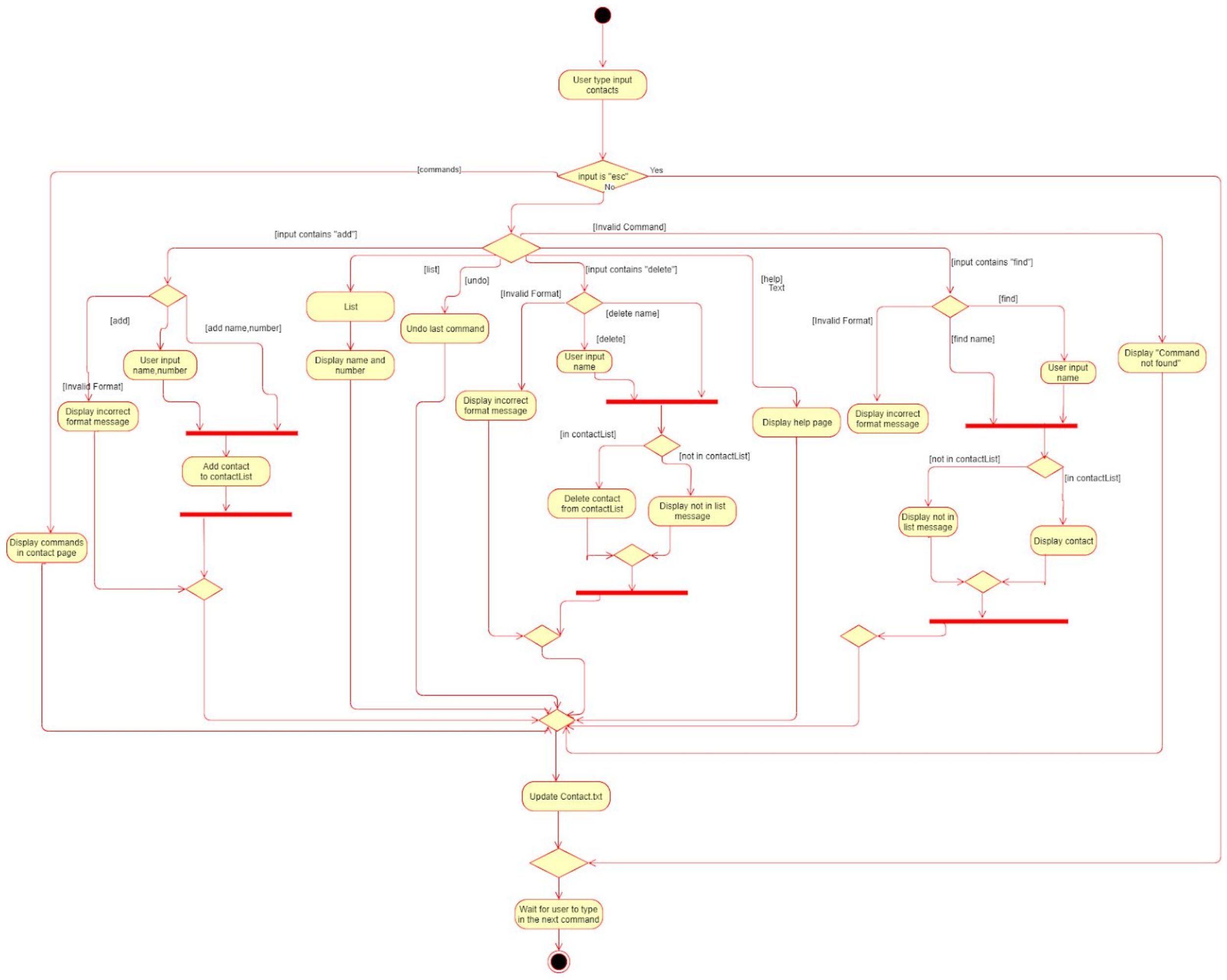


Figure 4.4.1.3. Activity diagram for Contact page.

4.4.2 Design Considerations

Aspect: Data structure to support Contacts

- Alternative 1(current choice): Using TreeMap to store name and number.
  - Pros: TreeMap is has a natural ordering and the keys are sorted, thus reducing the need to sort the map when it is listed out.
  - Cons: TreeMap's time complexity of insertion and delete is O(log(n)), which is slower than HashMap.
- Alternative 2: Using a HashMap to store names and numbers
  - Pros: HashMap has a time complexity of O(1) for insertion and deletion, which is faster than TreeMap.

- Cons: Null values/keys are allowed in a HashMap. Requires an additional validation to ensure users do not input null values/keys.

4.5. Expense Feature

4.5.1 Implementation

The Expense feature is facilitated by ExpenseCommandParser. It extends the Command class and stores the financial spending of the user internally as a `Map<LocalDate, ArrayList<String>> expenses` and also externally as `Expenses.txt.` Additionally, it implements the following operations:

- ExpenseCommandParser#add() — Add a new expense to the current list of expenses
- ExpenseCommandParser#delete() — Delete an expense from the current list of expenses.
- ExpenseCommandParser#find() — Find the expenses made on a certain date
- ExpenseCommandParser#list() — Shows the current list of expenses
- ExpenseCommandParser#undo() — Undo the previous command

The following diagram in Figure 4.5.1.1 shows the structure of the classes that implement the Expenses page feature:

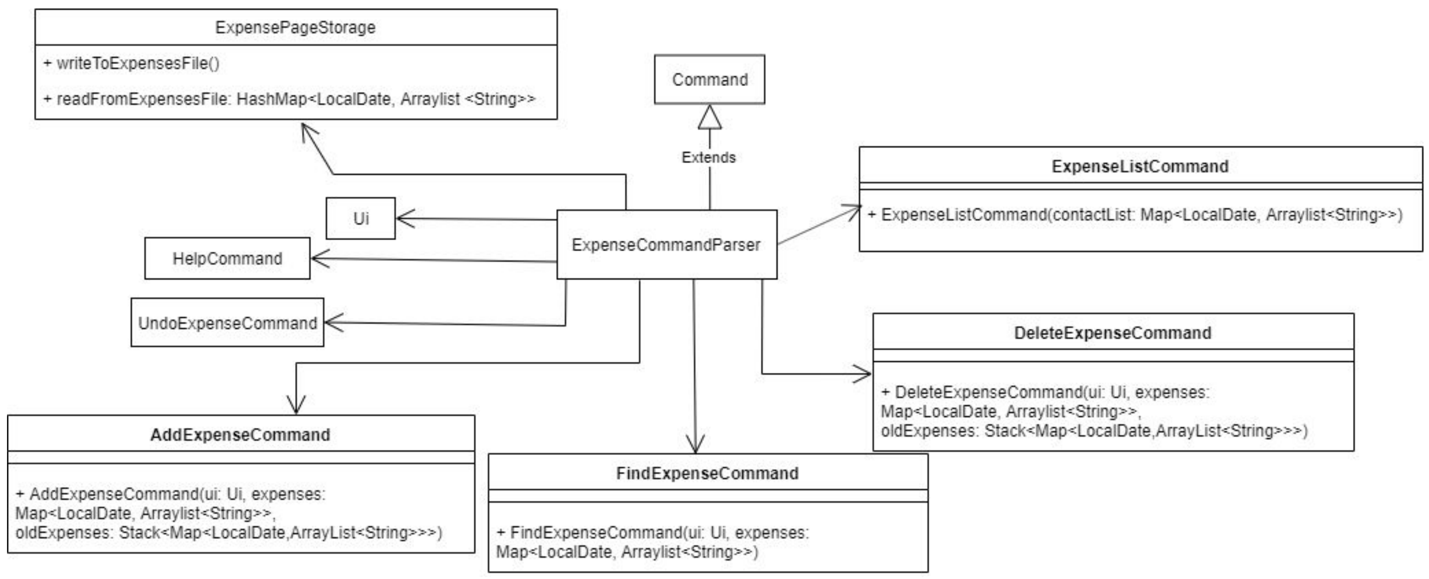

ExpensePageStorage
+ writeToExpensesFile()
+ readFromExpensesFile: HashMap<LocalDate, Arraylist <String>>
Command
Extends
ExpenseListCommand
+ ExpenseListCommand(contactList: Map<LocalDate, Arraylist<String>>)
Ui
HelpCommand
ExpenseCommandParser
UndoExpenseCommand
DeleteExpenseCommand
+ DeleteExpenseCommand(ui: Ui, expenses: Map<LocalDate, Arraylist<String>>, oldExpenses: Stack<Map<LocalDate,ArrayList<String>>>)
AddExpenseCommand
+ AddExpenseCommand(ui: Ui, expenses: Map<LocalDate, Arraylist<String>>, oldExpenses: Stack<Map<LocalDate,ArrayList<String>>>)
FindExpenseCommand
+ FindExpenseCommand(ui: Ui, expenses: Map<LocalDate, Arraylist<String>>)

Figure 4.5.1.1 Class diagram for Expense page feature

The following sequence diagram, Figure 4.5.1.2, illustrates the initialization steps:

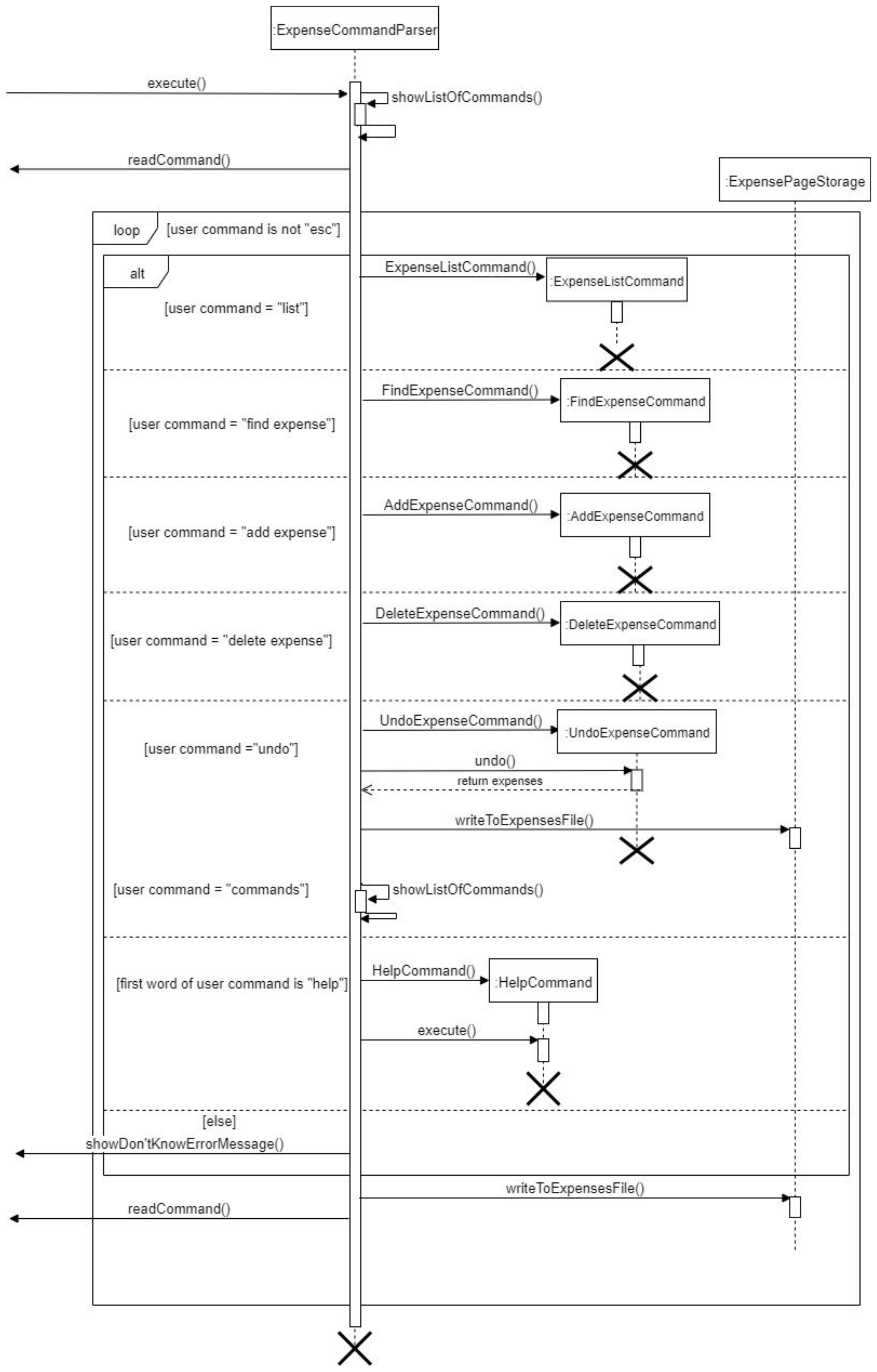


Figure 4.5.1.2 UML Sequence Diagram of ExpenseCommandParser

The following steps show a usage scenario and how the expenses mechanism behaves at each step:

Step 1. The user enters the application for the first time. The ExpenseCommandParser is initialised.

Step 2.The user executes `add` to add a new expense by calling AddExpenseCommand. User must input in the sequence of the item, price and date of purchase (e.g. `add bread, $4, 2019-09-09`). The added expense is added in to the internal `Map<LocalDate, Arraylist<String>> expenses` and it is saved in the external storage `Expenses.txt.`

Step 3. The user deletes an existing expense by calling the DeleteExpenseCommand through the command `delete`. User can either enter the name of the item, or the expense bullet number to be deleted from the list (e.g. `delete bread, delete 1`). This removes the deleted expense from the internal `Map<LocalDate, Arraylist<String>> expenses` and the deleted expense is deleted from the external storage `Expenses.txt.`

Step 4. The user types `find` which calls FindExpenseCommand to find the list of expenses spent on a particular date in the internal `Map<LocalDate, Arraylist<String>> expenses.` User can key in, for example, `find 2019-09-09`, to find the items purchased on that day.

Step 5. The user can call the ExpenseListCommand through the command `list`. This will show the list of expenses saved in `Map<LocalDate, Arraylist<String>> expenses.`

Step 6. The user types `undo` which calls UndoExpenseCommand to undo the previous command.

The following activity diagram, Figure 4.5.1.3, summarizes what happens when a user executes a new command:

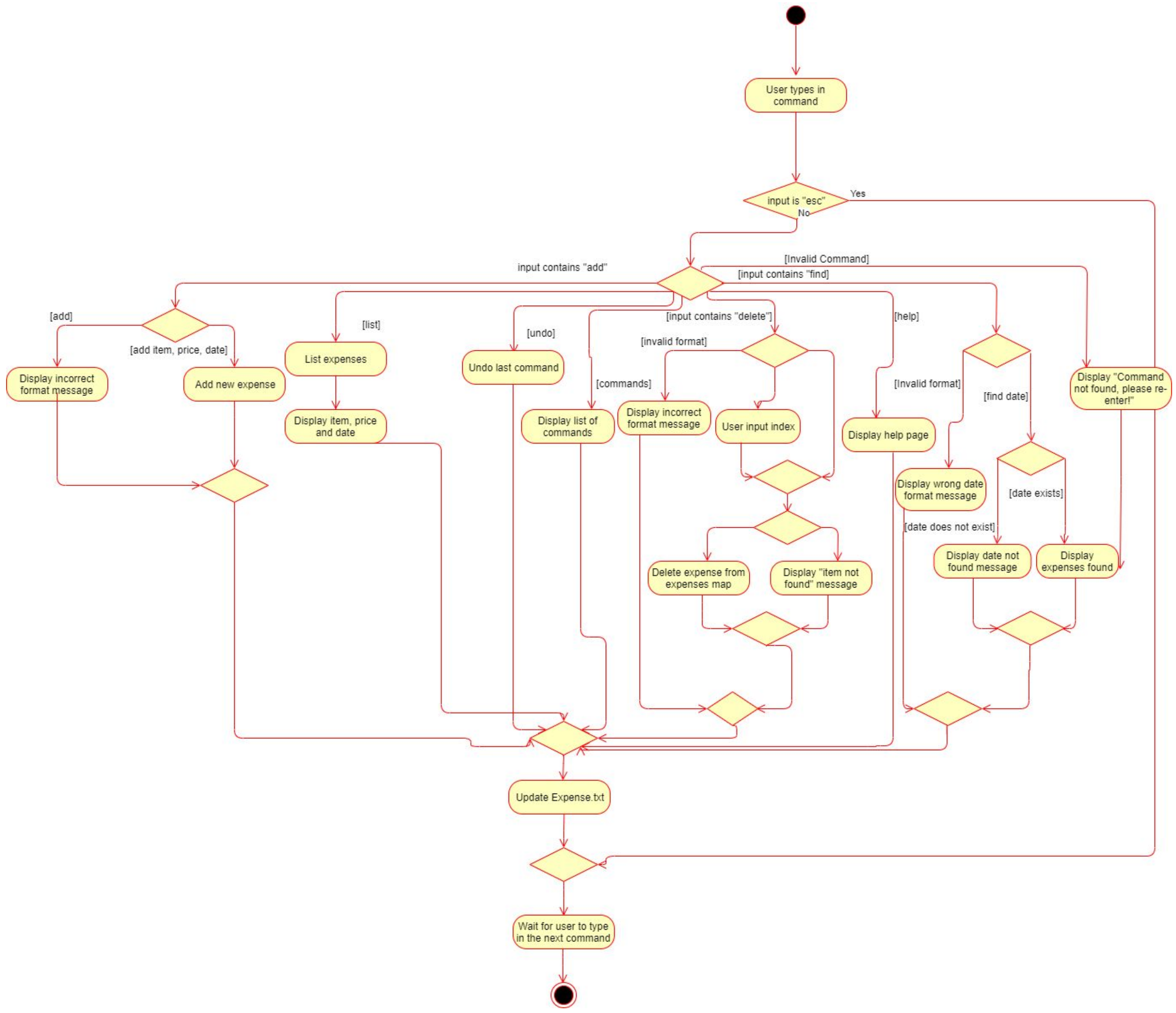


Figure 4.5.1.3 UML Activity Diagram of Expense feature

4.5.2 Design Considerations

Aspect: Data structure of Expense feature

Alternative 1 (current choice): Using TreeMap to store expenses with key as the LocalDate and values as the list of expenses recorded on that date (item name and price).

    - **Pros:** TreeMap has a natural ordering of keys, hence there is no need to manually sort the map in order.

    - **Cons:** TreeMap is slower than HashMap as it has a time complexity of O(log(n)) for deletion and insertion.

- Alternative 2: Using ArrayList to store LocalDate and string of expenses
    - **Pros:** HashMap operates faster than Map as its time complexity for deletion and insertion is O(1).

    - **Cons:** HashMap allows the storing of null keys and values, allowing less optimal amount of allocation of memory to store items compared to TreeMap.

4.6. Places Feature

4.6.1 Implementation

The Places feature is facilitated by PlacesCommand. It extends Command and stores the location information internally as an `Map<String, String> places` and also externally as `Places.txt`. Additionally, it implements the following operations:

- PlacesCommand#add() - Adds a location into the list of places.
- PlacesCommand#delete() - Delete a location from list of places.
- PlacesCommand#find() - Find a location from places based on the name.
- PlacesCommand#list() - List out the locations from places.
- PlacesCommand#undo() - Undo the previous command.

Here is a class diagram to show the structure of the classes that implement the Places page feature:

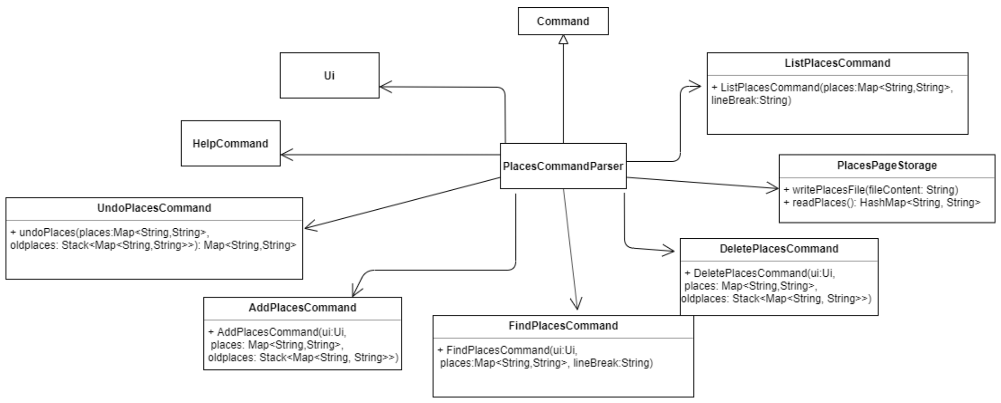

Command
Ui
HelpCommand
PlacesCommandParser
ListPlacesCommand
+ ListPlacesCommand(places:Map<String,String>, lineBreak:String)
PlacesPageStorage
+ writePlacesFile(fileContent: String)
+ readPlaces(): HashMap<String, String>
UndoPlacesCommand
+ undoPlaces(places:Map<String,String>, oldplaces: Stack<Map<String,String>>): Map<String,String>
DeletePlacesCommand
+ DeletePlacesCommand(ui:Ui, places: Map<String,String>, oldplaces: Stack<Map<String, String>>)
AddPlacesCommand
+ AddPlacesCommand(ui:Ui, places: Map<String,String>, oldplaces: Stack<Map<String, String>>)
FindPlacesCommand
+ FindPlacesCommand(ui:Ui, places:Map<String,String>, lineBreak:String)

Figure 4.6.1.1 Class Diagram of PlacesCommand

The following steps show a usage scenario:

Step1. The user enters the application for the first time. PlacesCommand is initialised

Step 2. The user inputs a command, `add, delete, list or find.`

Step 3. The respective command will be initialised.

Step 4. The user exits by inputting `esc`.

The following sequence diagram illustrates the initialization steps :

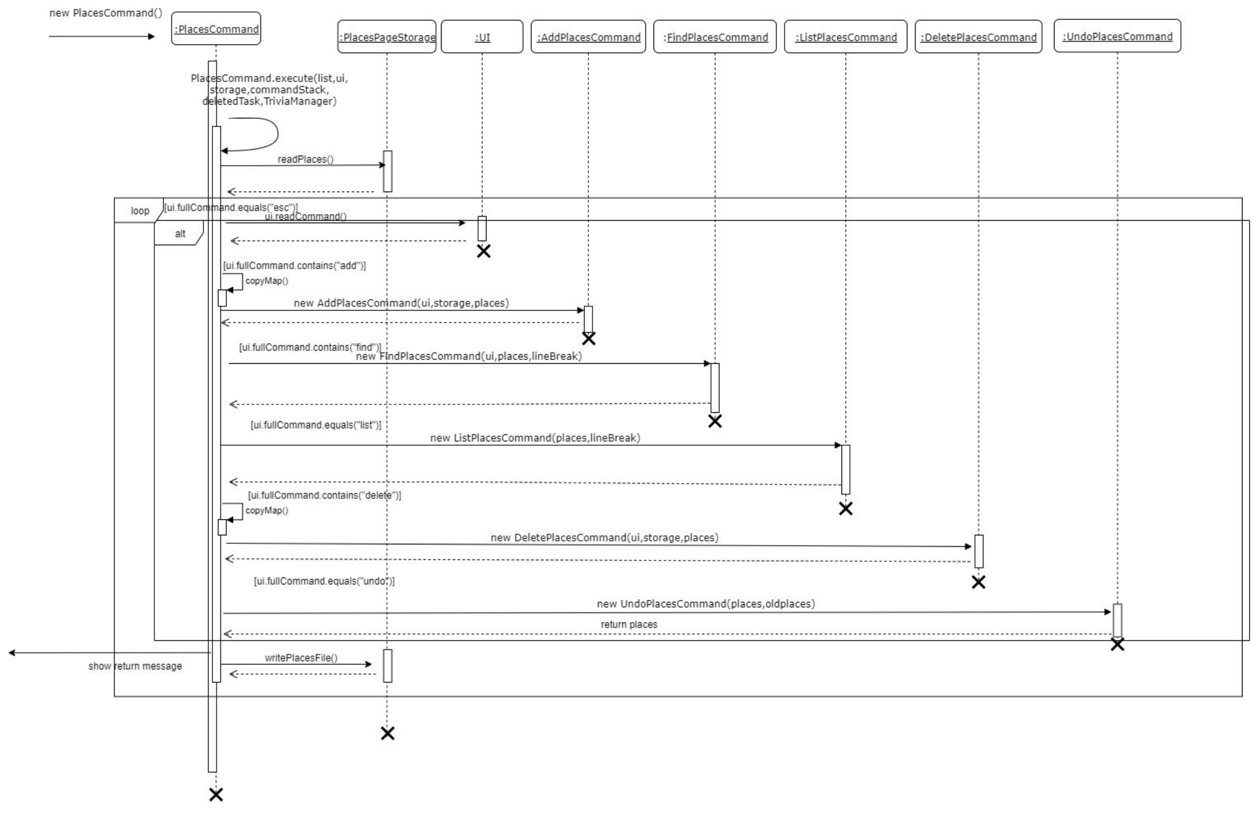

new PlacesCommand()
:PlacesCommand
:PlacesPageStorage
:UI
:AddPlacesCommand
:FindPlacesCommand
:ListPlacesCommand
:DeletePlacesCommand
:UndoPlacesCommand
PlacesCommand.execute(list,ui, storage,commandStack, deletedTask,TriviaManager)
readPlaces()
loop
[ui.fullCommand.equals("esc")]
ui.readCommand()
alt
[ui.fullCommand.contains("add")]
copyMap()
new AddPlacesCommand(ui,storage,places)
[ui.fullCommand.contains("find")]
new FindPlacesCommand(ui,places,lineBreak)
[ui.fullCommand.equals("list")]
new ListPlacesCommand(places,lineBreak)
[ui.fullCommand.contains("delete")]
copyMap()
new DeletePlacesCommand(ui,storage,places)
[ui.fullCommand.equals("undo")]
new UndoPlacesCommand(places,oldplaces)
return places
show return message
writePlacesFile()

Figure 4.6.1.2 UML Sequence Diagram of PlacesCommand

When an AddPlacesCommand is called:

Step 1. User will have to input the place and location they want to add.

Step 2. The new location will be added to `Map<String, String> places`.

Step 3. storagesPlaces() is called to save the new `Map<String, String> places`.

When an DeletePlacesCommand is called:

Step 1. User will have to input the place and location they want to delete.

Step 2. The location specific will be removed from `Map<String, String> places`.

Step 3. storagesPlaces() is called to save the new `Map<String, String> places`.

When a ListPlacesCommand is called:

Step 1. User inputs `list` into the command line.

Step 2. The `Map<String, String> places` will be printed out.

When a FindPlacesCommand is called:

Step 1.User inputs the name of the place they want to find.

Step 2. The places that have similar names will be printed out.

Step 3. If there is no such place in the `Map<String,String> places`, a message will be printed which indicates the place searched is not in `Map<String,String> places.`

The following activity diagram summarizes what happens when a user executes a new places command:

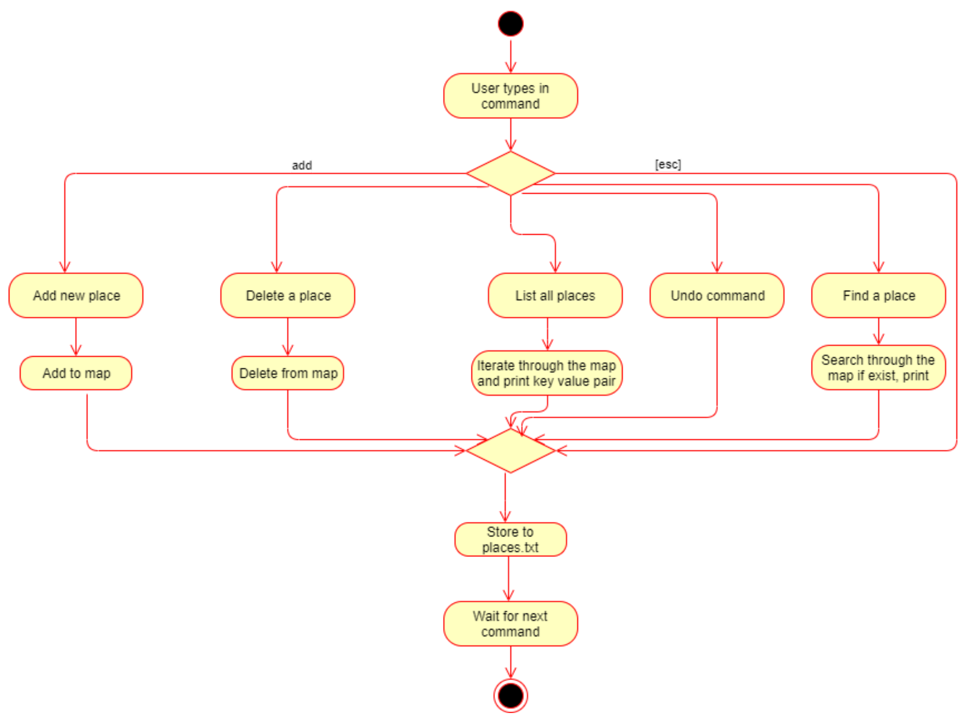


Figure 4.6.1.3 UML activity Diagram of PlacesCommand

4.6.2 Design Considerations

Aspect: Data structure to support Places

- Alternative 1(current choice): Using TreeMap to store locations and places.
  - Pros: TreeMap is has a natural ordering and the keys are sorted, thus reducing the need to sort the map when it is listed out.
  - Cons: TreeMap's time complexity of insertion and delete is O(log(n)), which is slower than HashMap.
- Alternative 2: Using a HashMap to store locations and places
  - Pros: HashMap has a time complexity of O(1) for insertion and deletion, which is faster than TreeMap.
  - Cons: Null values/keys are allowed in a HashMap. Requires an additional validation to ensure users do not input null values/keys.

4.7 Module Planner Feature

4.7.1 Implementation

The Module Planner is mainly facilitated by StudyAssistCommandParser, it extend Command and stores information internally in data structure of an ArrayList<ArrayList<String>> and externally in Study_Plan.txt file. StudyAssistCommandParser takes care of the logic for the module plan page and StudyAssistCommanParser implements the following operations:

- StudyAssistCommandParser#add() —Add module to specified semester in the module plan.
- StudyAssistCommandParser#delete() — Delete module from specified semester in the module plan.
- StudyAssistCommandParser#shift() — Shift module from existing semester to another semester in the module plan.
- StudyAssistCommandParser#plan() — Print out existing study plan and display reminder/suggestions in the command line interface.
- StudyAssistCommandParser#prerequisite() — Check prerequisite of certain module.
- StudyAssistCommandParser#undo() —- Undo previous commands.

Here is a class diagram which shows the structure of the classes that implement the module planner feature:

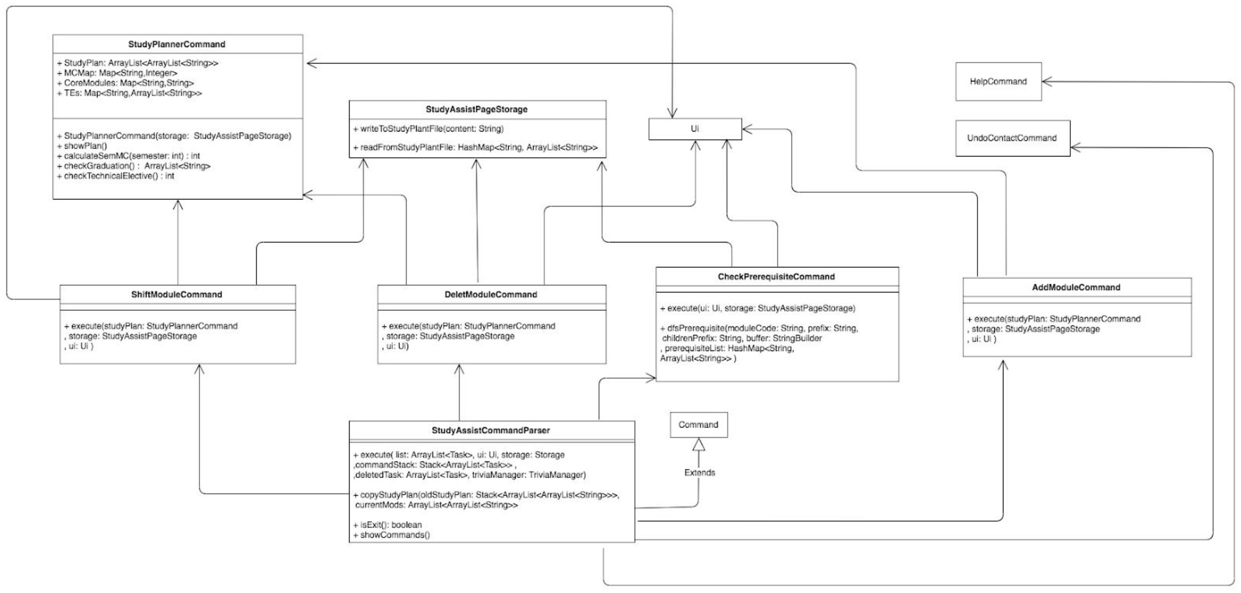

StudyPlannerCommand
+ StudyPlan: ArrayList<ArrayList<String>>
+ MCMap: Map<String,Integer>
+ CoreModules: Map<String,String>
+ TEs: Map<String,ArrayList<String>>
+ StudyPlannerCommand(storage: StudyAssistPageStorage)
+ showPlan()
+ calculateSemMC(semester: int) : int
+ checkGraduation() : ArrayList<String>
+ checkTechnicalElective() : int
StudyAssistPageStorage
+ writeToStudyPlantFile(content: String)
+ readFromStudyPlantFile: HashMap<String, ArrayList<String>>
Ui
HelpCommand
UndoContactCommand
ShiftModuleCommand
+ execute(studyPlan: StudyPlannerCommand
, storage: StudyAssistPageStorage
, ui: Ui)
DeletModuleCommand
+ execute(studyPlan: StudyPlannerCommand
, storage: StudyAssistPageStorage
, ui: Ui)
CheckPrerequisiteCommand
+ execute(ui: Ui, storage: StudyAssistPageStorage)
+ dfsPrerequisite(moduleCode: String, prefix: String,
childrenPrefix: String, buffer: StringBuilder
, prerequisiteList: HashMap<String,
ArrayList<String>>)
AddModuleCommand
+ execute(studyPlan: StudyPlannerCommand
, storage: StudyAssistPageStorage
, ui: Ui)
StudyAssistCommandParser
+ execute(list: ArrayList<Task>, ui: Ui, storage: Storage
,commandStack: Stack<ArrayList<Task>> ,
,deletedTask: ArrayList<Task>, triviaManager: TriviaManager)
+ copyStudyPlan(oldStudyPlan: Stack<ArrayList<ArrayList<String>>>,
currentMods: ArrayList<ArrayList<String>>
+ isExit(): boolean
+ showCommands()
Command
Extends

Figure 4.7.1.1. Class diagram indicates the structure of the module planner feature.

The following steps show a usage scenario:
Step1. The user enters the application for the first time. StudyAssistCommandParser is initialised by calling execute() method

Step 2. The user inputs a command, `add`, `delete`, `shift`, `plan undo or prerequisite.`

Step 3. The respective command will be initialised.

Step 4. The user exits by inputting `esc`.

The following sequence diagram shows the initialization step, when execute() method of StudyAssistCommandParser is called.

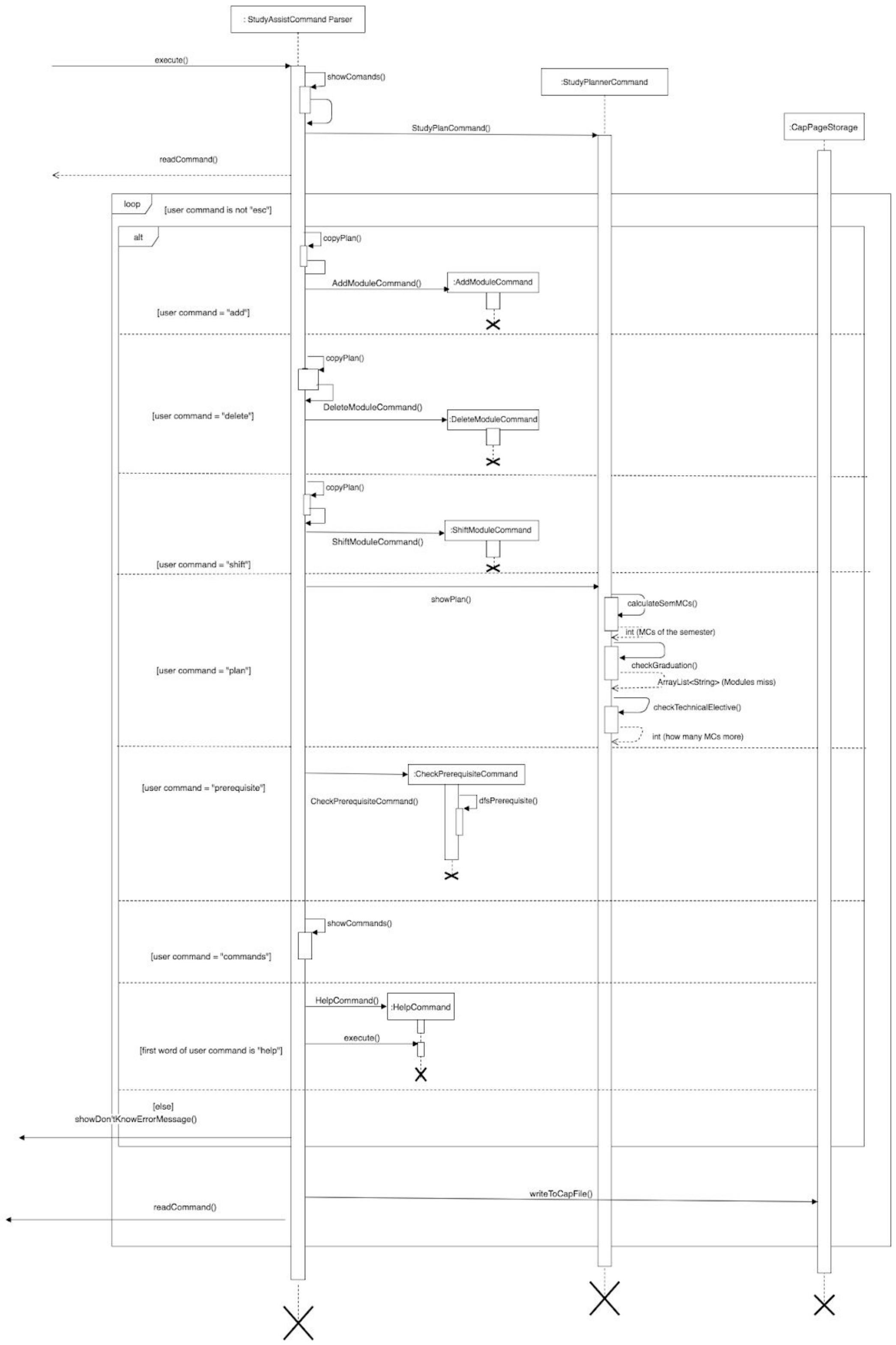


Figure 4.7.1.2. UML Sequence diagram of initialization step.

When an AddModuleCommand is called:

- Step 1. The user types in add command to add a module to a specified semester.
- Step 2. The add command calls addModuleCommand, insert a new module into the StudyPlan, modifying in internal Arraylist StudyPlan.
- Step 3. writeToStudyPlan() method is called to store updates in external Study_Plan.txt.

When a DeleteModuleCommand is called:

- Step 1. The user types in delete command to delete certain module from specific semester of the existing Study Plan.
- Step 2. The delete command calls deleteModuleCommand to delete module from the StudyPlan, modifying in internal Arraylist StudyPlan.
- Step 3. writeToStudyPlan() method is called to store updates in external Study_Plan.txt.

When a ShiftModuleCommand is called:

- Step 1. The user types in shift command to shift certain module from its belonging semester to another semester of the study plan.
- Step 2. The shift command calls shiftModuleCommand to delete module from the StudyPlan, modifying in internal Arraylist StudyPlan.
- Step 3. writeToStudyPlan() method is called to store updates in external Study_Plan.txt.

When a StudyPlannerCommandis called:

Step 1. User inputs `plan` into the command line.

Step 2. The "plan" command calls showPlan() method of StudyPlannerCommand, displays study plan in table format in the command line interface.

Step 3. Then, calculateSemMCs(), checkGraduation() and checkTechnicalElectives() methods of StudyPlannerCommand are called to show some comments and suggestions of the plan.

When a CheckPrerequisiteCommand is called:

Step 1.User inputs the module code that he wants to find prerequisite for.

Step 2. execute() method will be called, inside execute() method, it runs dfsPrerequisite() method to go through a depth-first search in the data structure.

Step 3. Tree of prerequisites is printed out

The following activity diagram summarizes what happens when execute() method of StudyAssistCommandParser is called.

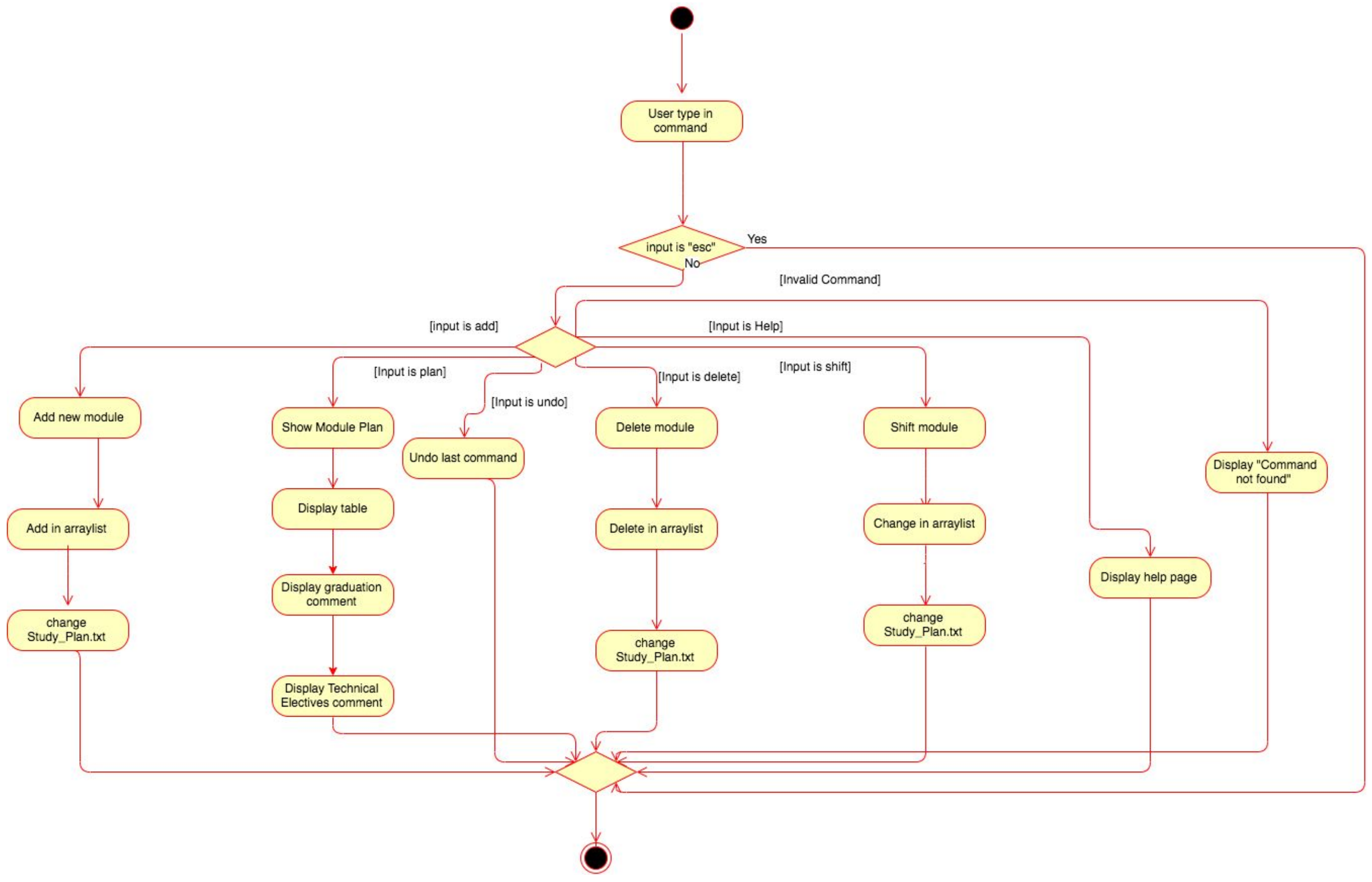


Figure 4.7.1.3. UML Activity diagram StudyAssistCommandParser

4.7.2 Design Considerations

- ❏ Aspect: How to store the Study Plan internally and externally
  - Alternative 1 (Current choice): Using an ArrayList of ArrayList of String to store modules of the Study plan, According to semesters.
    Also stores the ArrayList line by line in a txt file.

        - Pros: Easy to implement. This data structure matches other data structure type in the project and make it very easy to implement and also using other features' methods.
        - Cons: ArrayList has a time complexity of O(n) when insert or delete an element of it, it may become time consuming when ArrayList size increases. Also,when there is huge data, txt file may become large in size and causing the programme size to increase.
    - Alternative 2: Using HashMap to store modules and use json file to store study plan externally.
        - Pros: HashMap 's time complexity of delete and insert is O(1), when size increases it performs faster. Json file is more compact in size also.
        - Cons: HashMap's time complexity for access certain element in the HashMap is O(n), it has to check previous elements before accessing the exact element. Json file may be different to implement.

- ❏ Aspect: How to check total MCs of the modules
    - Alternative 1 (Current Choice): Using static HashMap to record MCs of all modules, module code is the key, MC number is the value.
        - Pros: The MC HashMap is static and can be accessed globally; HashMap find' s time complexity is O(1) make it fast to search and find MC of certain module.
        - Cons: It takes up a lot of memory usage.

    - Alternative 2: Create a module class, store all information (MCs, Description, Code and so on) in the class. When checking total MCs, just traverse the whole ArrayList and access each element's MC and count them up.
        - Pros: It convenience not only MC calculation but also benefit other operations of modules;
        - Cons: It needs to change other methods and Command to cooperate with the new module class, thus may be troublesome to implement.

4.8 Note Page Feature

4.8.1 Implementation

The note page feature has 2 subparts to it. There is a note page which the user will first come to when using the note page feature and there is a module page which is where the user will go to when the user types in `module /n MODULE_NAME` in the notes page.

The note page feature is mainly implemented by GeneralNoteCommandParser, ModuleCommandParser, GeneralNotePage, Module, Assessment and NotePageStorage.

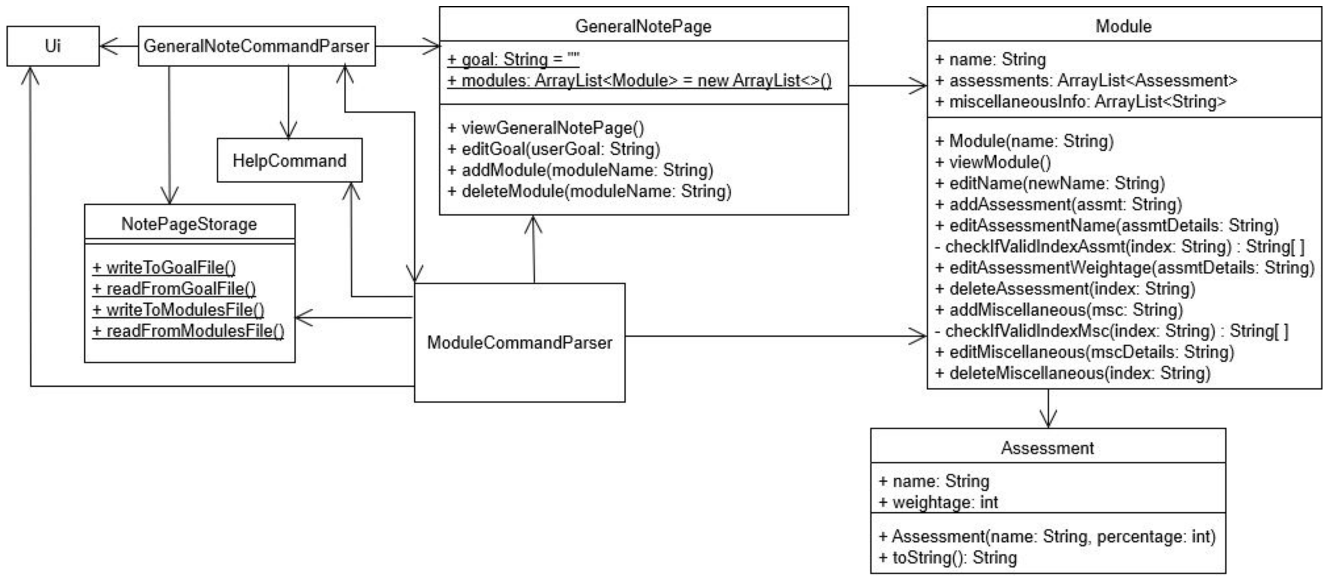
GeneralNoteCommandParser takes care of the logic for the notes page, which is implemented by the GeneralNotePage class. In turn, GeneralNotePage implements the following operations:

- GeneralNotePage#viewGeneralNotePage() – shows the user the contents of his/her note page
- GeneralNotePage#editGoal() – edits the goal of the user
- GeneralNotePage#addModule() – adds a module to the user's note page
- GeneralNotePage#deleteModule() – deletes a module from the user's note page

ModuleCommandParser takes care of the logic for the module page, which is implemented by the Module class. In turn, Module implements the following operations:

- Module#viewModule() – shows the user his/her notes for the module being edited/viewed
- Module#editName() – edits the name of the module being edited/viewed
- Module#addAssessment() – adds an assessment to the module being edited/viewed
- Module#editAssessmentName() – edits the name of an existing assessment
- Module#checkIfValidIndexAssmt() – makes the user input a valid index in the user's assessments list
- Module#editAssessmentWeightage() – edits the weightage of the assessment corresponding to the index specified by the user
- Module#deleteAssessment() – deletes an assessment corresponding to the index specified by the user
- Module#addMiscellaneous() – adds a miscellaneous information to the module being edited/viewed
- Module#checkIfValidIndexMsc() – makes the user input a valid index in the miscellaneous information list
- Module#editMiscellaneous() – edits a miscellaneous information corresponding to the index specified by the user
- Module#deleteMiscellaneous() – deletes a miscellaneous information corresponding to the index specified by the user

Here is a class diagram to show the structure of the classes that implement the note page feature:



Ui
GeneralNoteCommandParser
HelpCommand
NotePageStorage
+ writeToGoalFile()
+ readFromGoalFile()
+ writeToModulesFile()
+ readFromModulesFile()
ModuleCommandParser
GeneralNotePage
+ goal: String = ""
+ modules: ArrayList<Module> = new ArrayList<>()
+ viewGeneralNotePage()
+ editGoal(userGoal: String)
+ addModule(moduleName: String)
+ deleteModule(moduleName: String)
Module
+ name: String
+ assessments: ArrayList<Assessment>
+ miscellaneousInfo: ArrayList<String>
+ Module(name: String)
+ viewModule()
+ editName(newName: String)
+ addAssessment(assmt: String)
+ editAssessmentName(assmtDetails: String)
- checkIfValidIndexAssmt(index: String) : String[ ]
+ editAssessmentWeightage(assmtDetails: String)
+ deleteAssessment(index: String)
+ addMiscellaneous(msc: String)
- checkIfValidIndexMsc(index: String) : String[ ]
+ editMiscellaneous(mscDetails: String)
+ deleteMiscellaneous(index: String)
Assessment
+ name: String
+ weightage: int
+ Assessment(name: String, percentage: int)
+ toString(): String

Figure 4.8.1.1 Class diagram for note page feature

The following diagram shows the interactions for the method call execute() on a GeneralNoteCommandParser object:

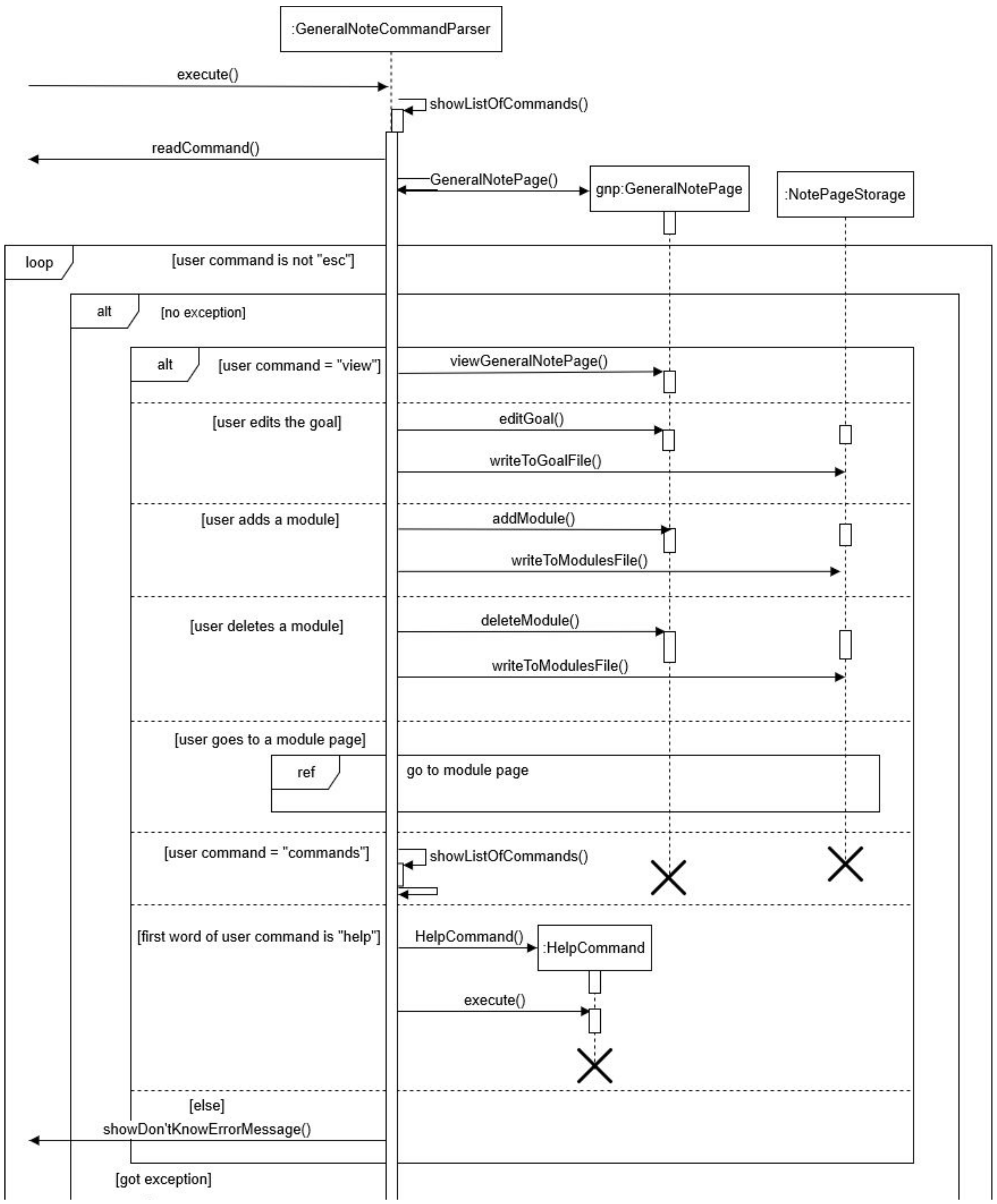


Figure 4.8.1.2 Sequence diagram for execute() of GeneralNoteCommandParser

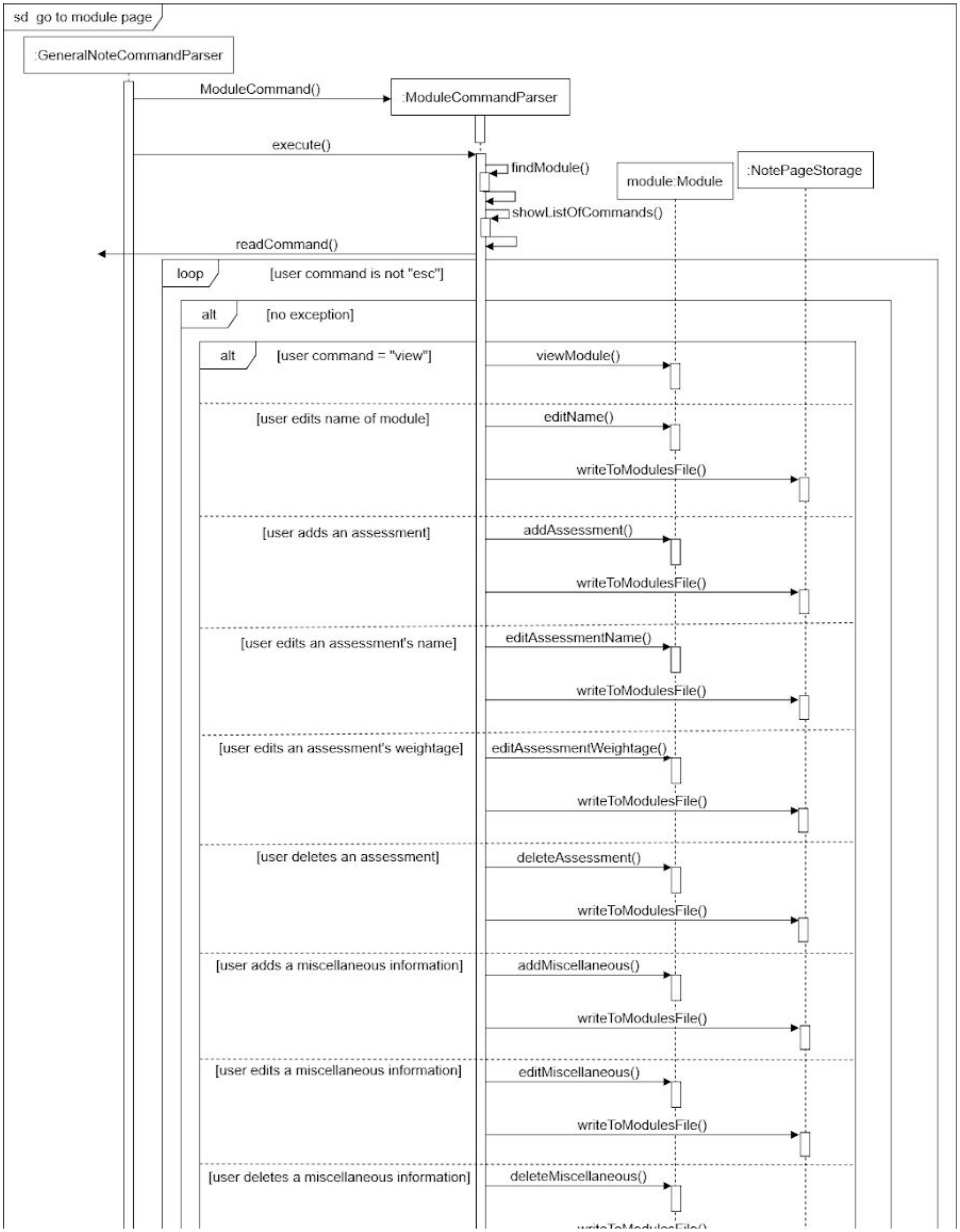


Figure 4.8.1.3 Sequence diagram for execute() of ModuleCommandParser

4.8.2 Design Considerations

Aspect: How the storage for this feature should be implemented.

- Alternative 1 (current implementation): Have a separate class to group all the read and write methods for this feature instead of putting all the storage methods for this feature into a general storage class which stores the storage methods for the whole program.
    - Pros: The storage methods would be more organised and this alternative applies single responsibility principle and separation of concerns principle. Also, it is easier to update the storage methods for this feature if we decide to update the way the notes are stored as it is easier to locate the methods.
    - Cons: When the general way the storage of the whole program works is changed, for example, if the location of all the text files is changed, it is more difficult to ensure that all relevant storage methods are updated according to the change.
- Alternative 2: Put the storage methods for this feature into the general storage class.
    - Pros: All the storage methods are gathered in one class so when an aspect of the way the storage works, only one class needs to be changed. This makes it easier to ensure that all storage methods are updated.
    - Cons: Many storage methods for different features are in one class which is messier and does not satisfy single responsibility principle and separation of concerns principle.

4.9 Help feature

4.9.1 Implementation

The help feature is implemented by HelpCommand and HelpText. HelpText stores the data required for the help feature. These data include descriptions of the various commands as well as information on how to interpret the syntax in the help page. HelpCommand deduces what help information has been requested by the user and shows the correct information to the user.

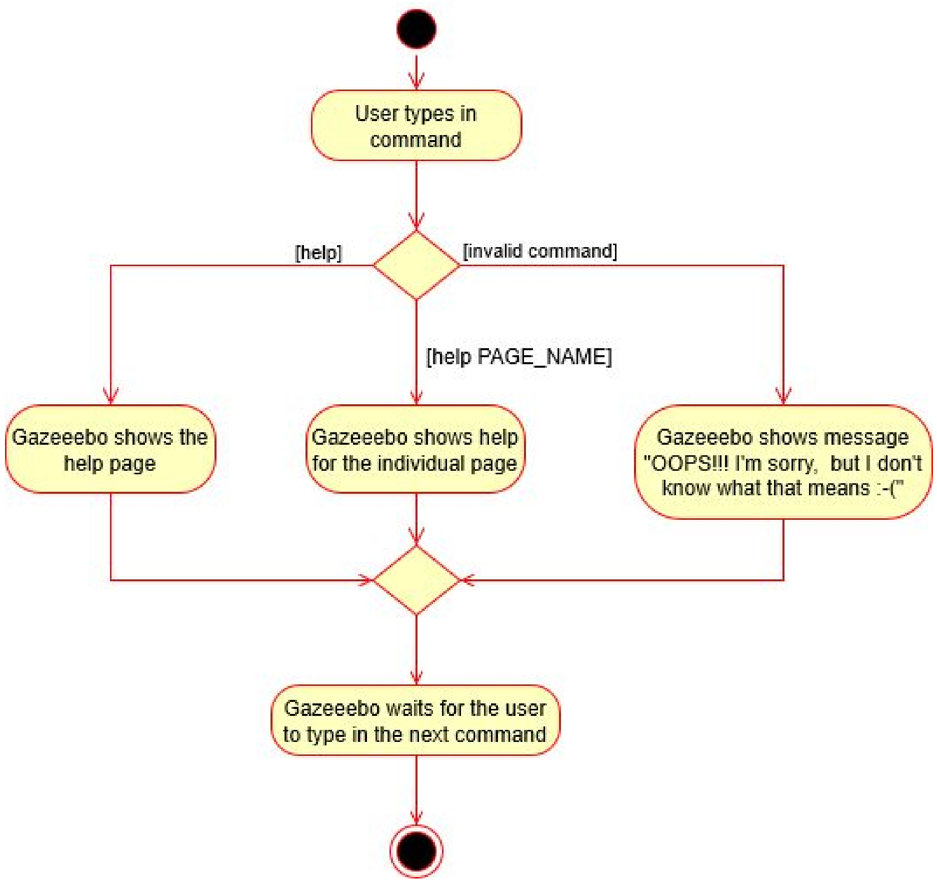

User types in command
[help]
[invalid command]
[help PAGE_NAME]
Gazeeebo shows the help page
Gazeeebo shows help for the individual page
Gazeeebo shows message "OOPS!!! I'm sorry, but I don't know what that means :-("
Gazeeebo waits for the user to type in the next command

Figure 4.9.1.1 Activity diagram for Help Feature

4.9.2 Design Considerations

Aspect: Data structure to store the descriptions of the commands.

- Alternative 1 (current implementation): Store the description of each individual command as a String without any container.
  - Pros: Fast retrieval of the descriptions of each command as there is no need to search for the required description.
  - Cons: The code becomes long as the code to enumerate all the command descriptions is long.
- Alternative 2: Use a HashMap to store the description of the commands as objects with the command name as the key.
  - Pros: Searching for the description of the command is fast in O(1) and code to enumerate the container is short.

    - Cons: Order of the commands in the help page might not be the same over time as the order of the map is not guaranteed to be the same over time.
- Alternative 3: Use a TreeMap to store the description of the commands as objects with the command name as the key.
    - Pros: The order of the commands in the help page will not change over time and code to enumerate the container is short.
    - Cons: Retrieval of a command description is the slowest out of all the 3 alternatives in O(log n), where n is the number of commands in the TreeMap.

Aspect: The storage location of the descriptions of the commands.

- Alternative 1: Store the descriptions in a text file.
    - Pros: Good organisation of program components as all data would be stored in the same place.
    - Cons: Program slows down as reading data from the text file takes up time.
- Alternative 2 (current implementation): Store the descriptions directly in a class in the Java file.
    - Pros: Program is faster as time required to read from a text file is eliminated.
    - Cons: Poor organisation of program components as data is mixed into the source code.

4.10. Specialization Feature

4.10.1 Implementation

The Specialization feature is facilitated by SpecializationCommandParser. It extends the Command class and stores list of specializations and technical electives completed by the user internally as a `Map<String, ArrayList<ModuleCategory>> specMap` and `Map<String, ArrayList<String>> completedEMap`, and also externally as `Specialization.txt` and `CompletedElectives.txt`. Additionally, it implements the following operations:

- SpecializationCommandParser#list() — Shows list of specializations and completed technical electives, and the number of modular credits completed
- SpecializationCommandParser#complete() — Mark a technical elective as done
- SpecializationCommandParser#command() — Shows the list of commands in specialization page
- SpecializationCommandParser#help() — Shows the general help page and all the commands in Gazeeebo

The following diagram in Figure 4.10.1.1 shows the structure of the classes that implement the Specialization page feature:

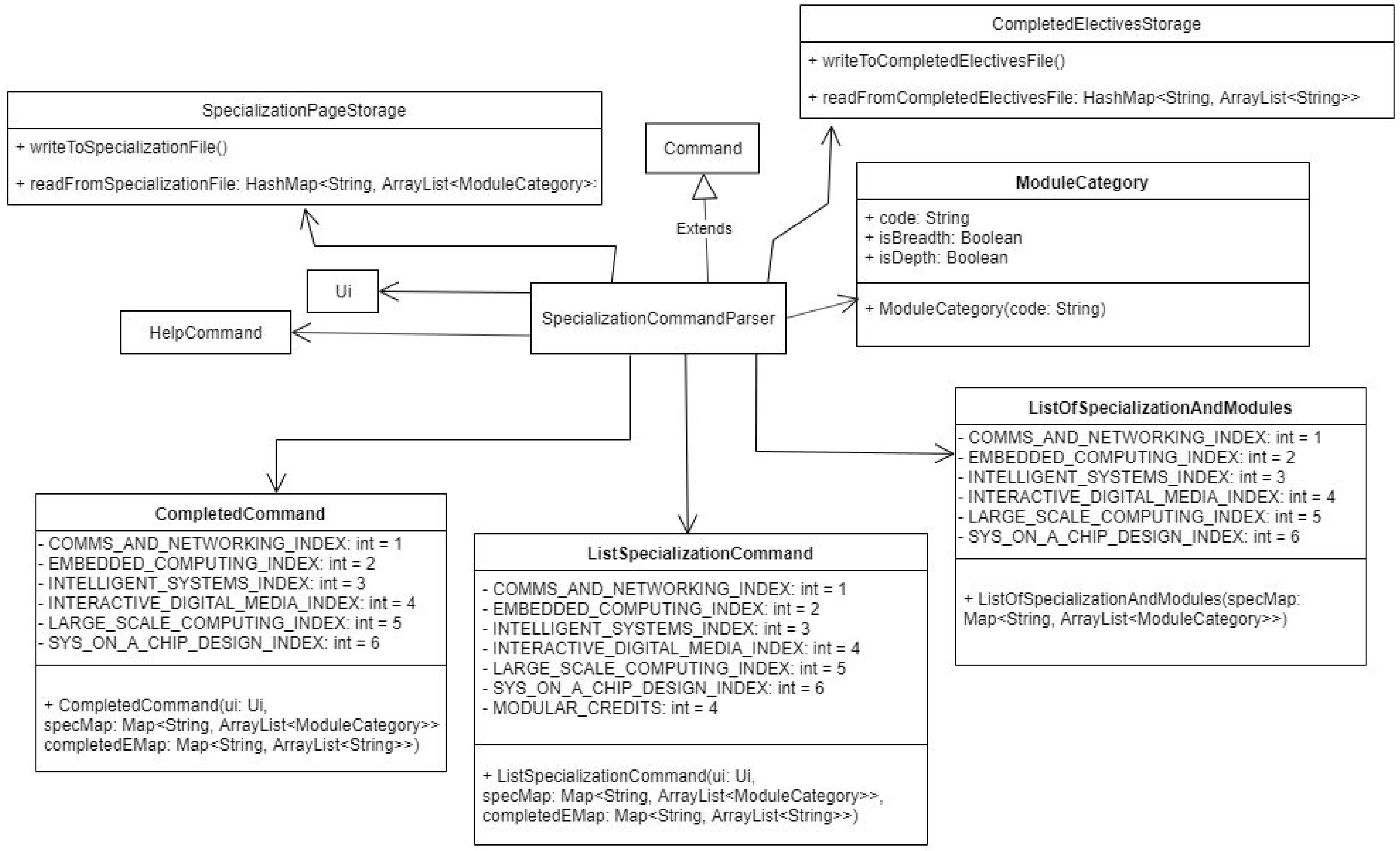

CompletedElectivesStorage
+ writeToCompletedElectivesFile()
+ readFromCompletedElectivesFile: HashMap<String, ArrayList<String>>
SpecializationPageStorage
+ writeToSpecializationFile()
+ readFromSpecializationFile: HashMap<String, ArrayList<ModuleCategory>:
Command
Extends
ModuleCategory
+ code: String
+ isBreadth: Boolean
+ isDepth: Boolean
+ ModuleCategory(code: String)
Ui
HelpCommand
SpecializationCommandParser
ListOfSpecializationAndModules
- COMMS_AND_NETWORKING_INDEX: int = 1
- EMBEDDED_COMPUTING_INDEX: int = 2
- INTELLIGENT_SYSTEMS_INDEX: int = 3
- INTERACTIVE_DIGITAL_MEDIA_INDEX: int = 4
- LARGE_SCALE_COMPUTING_INDEX: int = 5
- SYS_ON_A_CHIP_DESIGN_INDEX: int = 6
+ ListOfSpecializationAndModules(specMap: Map<String, ArrayList<ModuleCategory>>)
CompletedCommand
- COMMS_AND_NETWORKING_INDEX: int = 1
- EMBEDDED_COMPUTING_INDEX: int = 2
- INTELLIGENT_SYSTEMS_INDEX: int = 3
- INTERACTIVE_DIGITAL_MEDIA_INDEX: int = 4
- LARGE_SCALE_COMPUTING_INDEX: int = 5
- SYS_ON_A_CHIP_DESIGN_INDEX: int = 6
+ CompletedCommand(ui: Ui, specMap: Map<String, ArrayList<ModuleCategory>> completedEMap: Map<String, ArrayList<String>>)
ListSpecializationCommand
- COMMS_AND_NETWORKING_INDEX: int = 1
- EMBEDDED_COMPUTING_INDEX: int = 2
- INTELLIGENT_SYSTEMS_INDEX: int = 3
- INTERACTIVE_DIGITAL_MEDIA_INDEX: int = 4
- LARGE_SCALE_COMPUTING_INDEX: int = 5
- SYS_ON_A_CHIP_DESIGN_INDEX: int = 6
- MODULAR_CREDITS: int = 4
+ ListSpecializationCommand(ui: Ui, specMap: Map<String, ArrayList<ModuleCategory>>, completedEMap: Map<String, ArrayList<String>>)

Figure 4.10.1.1 Class diagram for Specialization page feature

The following sequence diagram, Figure 4.10.1.2, illustrates the initialization steps:

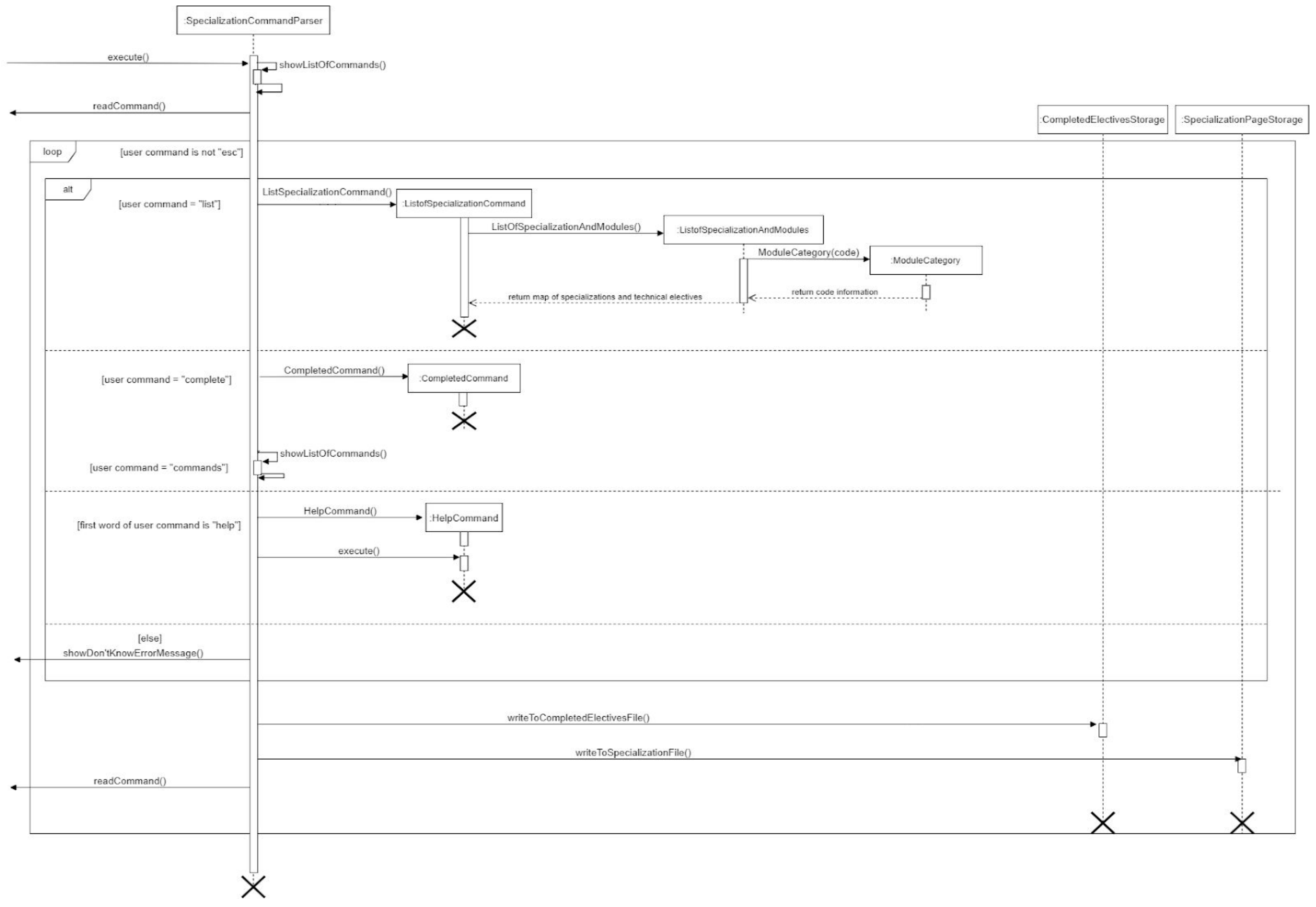

:SpecializationCommandParser
execute()
showListOfCommands()
readCommand()
:CompletedElectivesStorage
:SpecializationPageStorage
loop
[user command is not "esc"]
alt
[user command = "list"]
ListSpecializationCommand()
:ListofSpecializationCommand
ListOfSpecializationAndModules()
:ListofSpecializationAndModules
ModuleCategory(code)
:ModuleCategory
return map of specializations and technical electives
return code information
[user command = "complete"]
CompletedCommand()
:CompletedCommand
showListOfCommands()
[user command = "commands"]
[first word of user command is "help"]
HelpCommand()
:HelpCommand
execute()
[else]
showDon'tKnowErrorMessage()
writeToCompletedElectivesFile()
writeToSpecializationFile()
readCommand()

Figure 4.10.1.2 UML Sequence Diagram of SpecializationCommandParser

The following steps show a usage scenario and how the specialization mechanism behaves at each step:

Step 1. The user enters the application for the first time, SpecializationCommandParser is initialised.

Step 2. The user marks a technical elective complete by calling the CompletedCommand through the command `complete`. User then enters `specialization index` to choose a specialization. After choosing a specialization, the user then enters the `technical elective index` that he has completed. The completed technical elective is then recorded in the internal `Map<String, Arraylist<String>> completedEMap` and it is saved in the external storage `CompletedElectives.txt`.

Step 3. The user can call the ListSpecializationCommand through the command list. This will show the list of specializations and all the technical electives, the user's completed technical electives and the completed modular credits that are saved in `Map<String, Arraylist<ModuleCategory>> specMap` and `Map<String, Arraylist<String>> completedEMap`.

The following activity diagram, Figure 4.10.1.3, summarizes what happens when a user executes a new command:

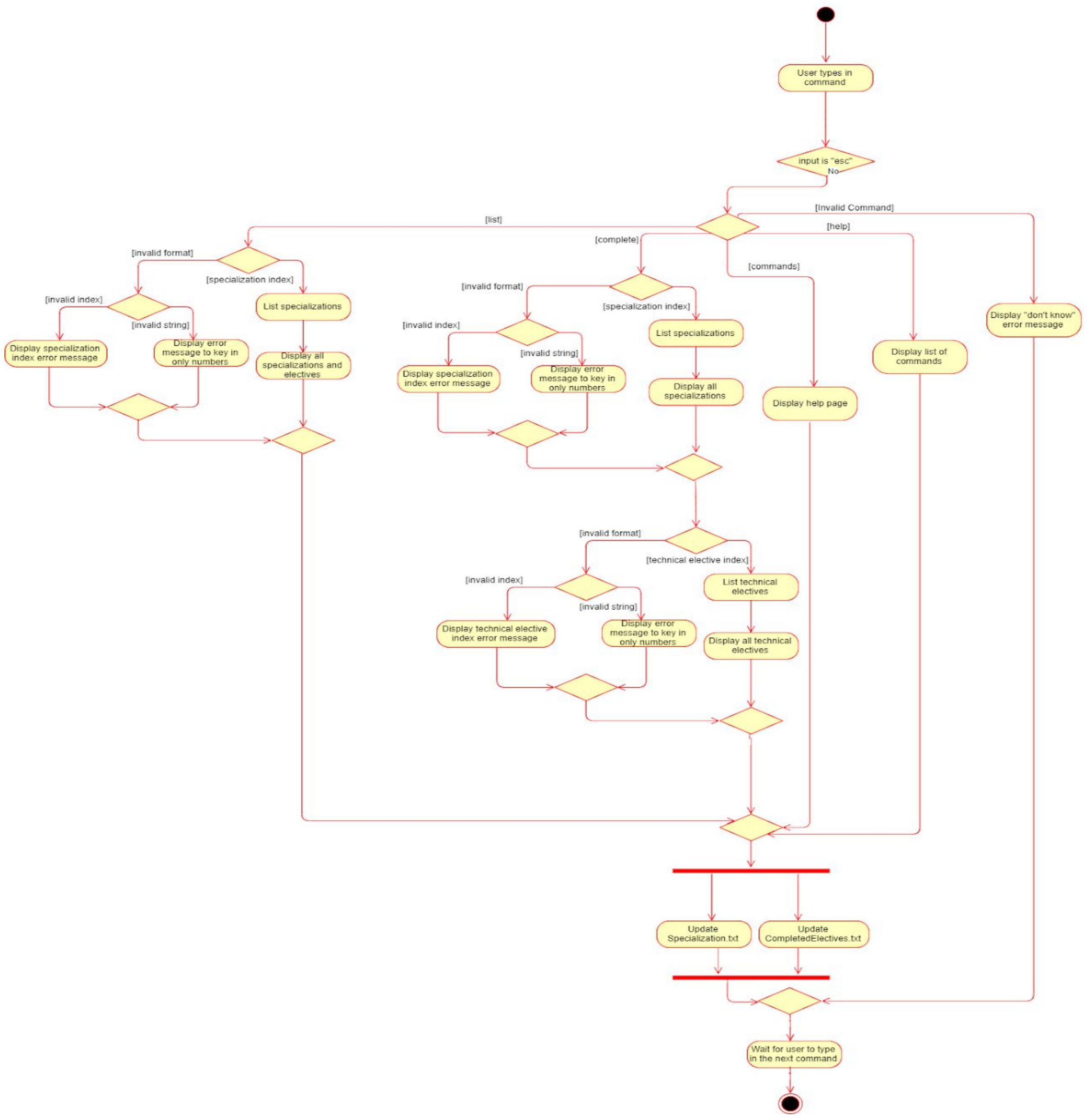

User types in command
input is "esc"
No
[Invalid Command]
[list]
[help]
[complete]
[commands]
[invalid format]
[specialization index]
[invalid index]
[invalid string]
List specializations
Display specialization index error message
Display error message to key in only numbers
Display all specializations and electives
[invalid format]
[specialization index]
[invalid index]
[invalid string]
List specializations
Display specialization index error message
Display error message to key in only numbers
Display all specializations
Display help page
Display list of commands
Display "don't know" error message
[invalid format]
[technical elective index]
[invalid index]
[invalid string]
List technical electives
Display technical elective index error message
Display error message to key in only numbers
Display all technical electives
Update Specialization.txt
Update CompletedElectives.txt
Wait for user to type in the next command

Figure 4.10.1.3 UML Activity Diagram of Specialization feature

4.5.2 Design Considerations

Aspect: Saving of data externally

- Alternative 1 (current choice): Using text files to read and write data
    - **Pros:** It is straight-foward and simple to read and write to the file system, and you can alter the data easily through the external text files.
    - **Cons:** For large data, the text files may become very large in size, increasing the programme size.
- Alternative 2: Using JSON file to read and write data
    - **Pros:** Handling JSON format requires less processing power and it is light-weight when handling complete and large-scale data.
    - **Cons:** It is difficult to implement and a parser has to be programmed to access the data in the JSON file which might require technical labor.

4.11 Undo Feature

4.11.1 Implementation

The undo function is facilitated by UndoTaskCommand, UndoPlacesCommand, UndoContactsCommand, UndoExpensesCommand, UndoStudyPlannerCommand. It alters the current list by changing it to the previous list before the previous command was called. The modified list will then be saved by calling the storage.store() command. Internally, the previous lists will be stored in a stack of array list. When an undo command is executed, the top of the stack will be accessed and copied to the current list. The top of the stack will then be popped.

Given below is an example usage scenario and how the undo mechanism behaves at each step:

Step 1. The user enters `tasks` page and executes `todo Coding exercises` to add a new todo task. Before this command is executed and a method copyOldList() will be called to deep copy the current list. This copied list will then be pushed into the stack of array list.

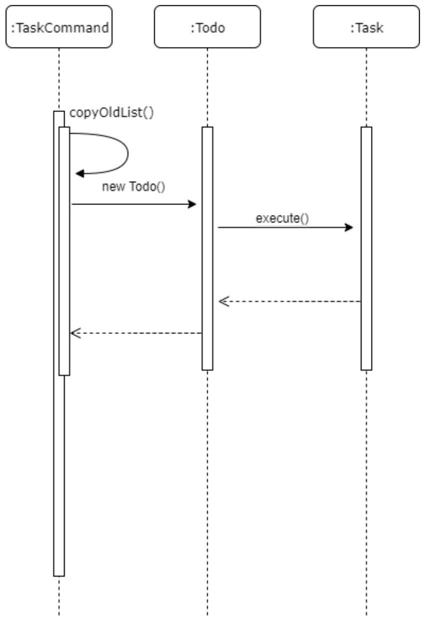


Figure 4.11.1.1 UML Sequence Diagram of adding a Todo task

Step 2. The user realised he made a mistake and decides to `undo`. This will call UndoTaskCommand, which will replace the current list with the old list at the top of the stack. The new list will then be saved using the storage.writeToSaveFile() method.

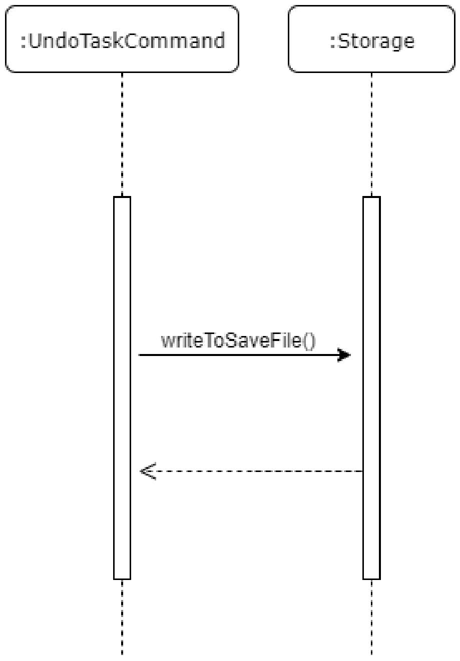


Figure 4.11.1.2 UML Sequence Diagram of undo feature

Step 3. The top of the stack will be popped. If the stack is empty and the user tries to `undo` a command. A message will be printed out to notify the user the previous command cannot be undone.
The following activity diagram summarizes what happens when a user executes an undo command:

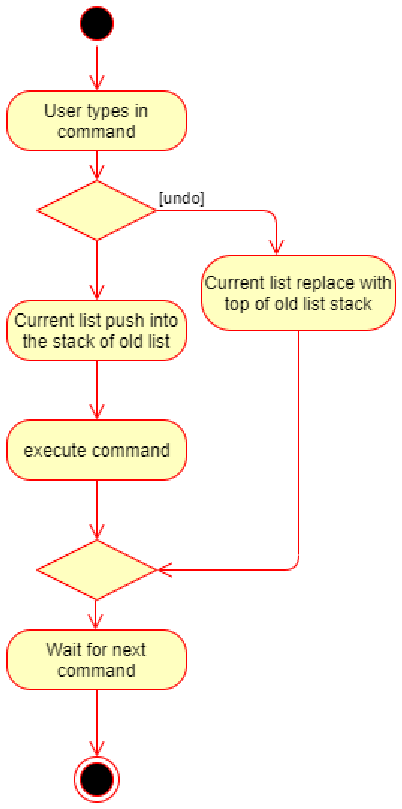

User types in command
[undo]
Current list replace with top of old list stack
Current list push into the stack of old list
execute command
Wait for next command

Figure 4.11.1.3 Class diagram for Undo feature

4.11.2 Design Considerations

Aspect: How undo executes

- Alternative 1(current choice): Saving the entire array list of tasks.
  - Pros: Easy to implement. Easy maintenance of code since a change in the command's logic will not affect the undo function.
  - Cons: Higher memory usage as more commands are executed.
- Alternative 2: Individual command has an undo function that contains the opposite logic.
  - Pros: Use less memory and more efficient as individual task will be changed instead.
  - Cons: Undo function for each command needs to be changed when the execute function change.

Aspect: Data structure used to store the old list of tasks.

- Alternative 1(current choice): Using a stack to store the array list of tasks
    - Pros: Peeking at the top of the stack, pushing into the stack and popping from the stack is O(1) time complexity. Easier to implement and maintain.
    - Cons: Using a stack will not be able to implement a redo function.
- Alternative 2: Using an Array list to store the array list of tasks.
    - Pros: Able to use pointers to implement the redo function.
    - Cons: Checking if the Arraylist is empty will take O(n) time complexity which is slower than stack.

4.11.2 Proposed Implementation in v2.0

Extending undo function to the other features:
On top of the current implementation, the undo feature could be extended to both the Notes feature and the Specialization feature as well. The CompletedCommand for Specialization can be undone through the undo command. Similarly for the Notes feature AddNoteCommand, DeleteNoteCommand and EditNoteCommand can be undone through the undo command.

Implementing a Redo function:
A Redo function can be integrated by creating another stack of old list and pushing the state of the list just before the undo command is executed. When the Redo command is called, the current list will be replaced by the list at the top of the stack.

Implementing an Uncheck function:
An Uncheck function is to mark a technical elective as incomplete if it was once marked as completed. It can be integrated by using the boolean function. If a technical elective is completed, it will be marked as “true”. Hence, to uncheck a completed technical elective, if a technical elective is “true”, change the boolean to “false”.

5. Documentation

5.1 Introduction

We used JavaDocs for documentation of the programs methods and classes as it easily allows the documentation to be generated from the in code comments into HTML format.

5.2 Editing documentation

To edit the documentation, you can edit the in code comments and generate a new HTML document with the updated documentation.

Generating JavaDocs documentation from Intellij

1. Open IntelliJ.

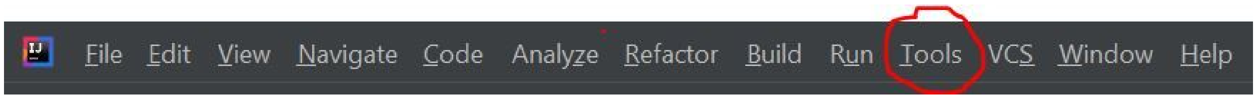
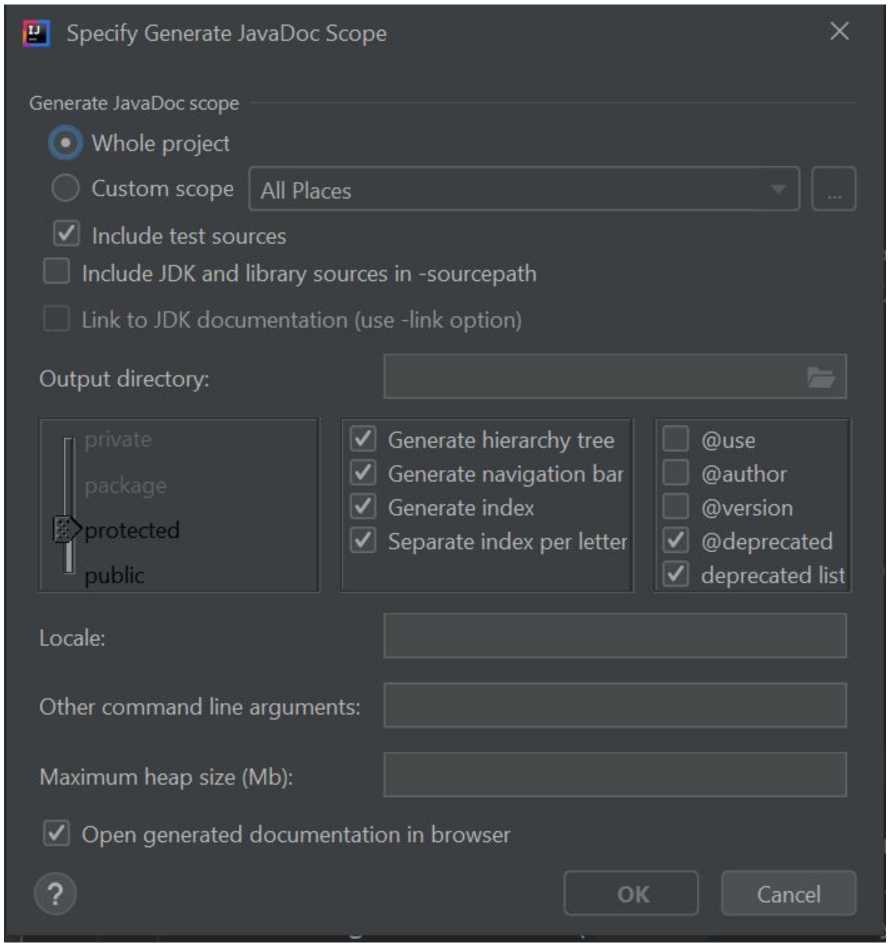


Figure 5.1 Click on the 'Tools' button

2. Click on `Tools` in the main menu, as shown in Figure 5.1 above, which is at the top of the IntelliJ editor window.
3. From the drop down menu, click on `Generate JavaDoc`.



Specify Generate JavaDoc Scope

Generate JavaDoc scope

(●) Whole project

( ) Custom scope: All Places

[✓] Include test sources

[ ] Include JDK and library sources in -sourcepath

[ ] Link to JDK documentation (use -link option)

Output directory:

private / package / protected / public

[✓] Generate hierarchy tree

[✓] Generate navigation bar

[✓] Generate index

[✓] Separate index per letter

[ ] @use

[ ] @author

[ ] @version

[✓] @deprecated

[✓] deprecated list

Locale:

Other command line arguments:

Maximum heap size (Mb):

[✓] Open generated documentation in browser

Figure 5.2 Pop up window

4. In the window that pops up, as shown in Figure 5.2 above, you can:
    a. Select the scope, which is the set of files, which you want to generate the documentation for.

    b. Set which directory where you want the documentation to be generated to.
    c. Select what members to include into the documentation based on their visibility, such as private, package, protected or public,
    d. Specify the locale, other command line arguments and the maximum heap size.
5. For more detailed instructions on how to configure the documentation file from IntelliJ, you can refer to JetBrains' website: https://www.jetbrains.com/help/idea/working-with-code-documentation.html.

5.3 Editing diagrams

We used the website https://www.draw.io/ to draw the UML and architecture diagrams in this developer guide. To edit the diagrams, you can download the diagrams from this link and open the diagrams in the draw.io as stated above. After you have finished editing the diagram, you can download them as a picture and insert them into this developer guide.

6. Testing

Method 1: Using Intellij JUnit test runner

- To run all tests, right-click on the `src/test/java` folder and choose `Run 'Tests in 'duke.test''`
- To run a subset of tests, you can right-click on a test package, test class, or a test and choose `Run 'file_name'`.

Method 2: Using Gradle

- To run all tests, type in the command `gradlew test` in the terminal in IntelliJ and press ENTER.

Appendix A

User Stories

Priorities: High (must have) - * * *, Medium (nice to have) - * *, Low (unlikely to have) - *

No.	**Priority**	**As a NUS student, I want to be able to ...**	**So that ...**
1.	* * *	Add a new task Eg. deadline/todo/task/birthday	I am able to record my agenda
2.	* * *	Delete a task	When I can delete a wrongly added task/completed

			task
3.	* * *	Delete multiple tasks	I can quickly delete more than one tasks at once
4.	* * *	List out my task	I can see all my added agendas
5.	* * *	Categorize my task	I can make my list of agendas neater
6.	* * *	Find tasks by inputting keywords	I can efficiently locate a certain specific task
8.	* * *	Rearrange my tasks based on the task importance which I have set	I can prioritize my tasks and locate the most important to the least important tasks
9.	* * *	See the dates/months in a calendar view	I can have an overview of the dates
10.	* * *	Toggle the done function	I can change the task to undone if I accidentally marked it as done
11.	* * *	Save the calendar to a text file	I can print it out if I want to
12.	* * *	See all the tasks that are marked as done	I can see more easily what I have done
13.	* * *	Schedule tasks daily, weekly or monthly based on my preference	I can plan my recurring schedules efficiently
14.	* * *	Undo previous function	I can undo a function in case I added or deleted something by mistake
15.	* * *	Note section to jot down my miscellaneous notes	I can jot down anything I want and save it in my planner
16.	* * *	Save contact numbers of people	I can record my important contact numbers into the planner
17.	* * *	Save locations	I can record different locations to remind me where certain places are, especially in school where the compound is very big.
18.	* * *	Record my expenses	I can manage my finances and record my expenses

19.	* * *	Plan my school modules	I can organize my school course plan throughout the semesters
20.	**	Calculate my CAP grades	I can record my grades from previous semesters and track my grades
21.	**	Plan my course specialization	I can see my progress of completing my specialization
22.	**	Have a calendar view	I can see the months and days, and see which days I have things scheduled on that day
23.	**	Tag my tasks to organize them better	I can easily search the hashtags for tasks with the same tags
24.	**	See today’s agenda when I run my project	I can easily see what I need to to today
25.	**	Have a note section for each month	I can have general notes for a particular month
26.	**	Enable notification for my tasks	I can be reminded about what I need to do
27.	**	Create a recurring task	I can have a long running task
29.	**	Set a goal for a particular date/month	I can have a focus for that date/month
30.	**	Have motivational quotes appearing randomly on dates that I am very busy	I can be cheered on by the quotes
31.	**	Have previous tasks automatically deleted	I do not have to clear it manually(convenience purposes)
32.	**	Lock my list with a password	Others cannot see the tasks in my secret list.
33.	**	See today’s date in the app	It is more convenient and easier for me to tell which task is more urgent.
34.	**	Find the particular date	I can see task only on that date
35.	**	Have a progress indicator for my task	I can know the status of my tasks

36.	**	Hide a lengthy description	I can better describe my task with more details but preserve the compactness of the list
37.	*	Have a countdown timer	I can know how many days left to the task

Appendix B: Use cases

For all use cases below, the System refers to Gazeeebo and the Actor refers to CEG students, unless otherwise specified.

1. Set reminder

1. The actor selects a task and chooses the option to set a reminder
2. System prompts for date
3. Actor selects date
4. System prompts for time
5. Actor selects time and presses enter
6. System sets the reminder for the task specified.

2. Setting recurring tasks (e.g. every week on a specific day, every month, etc.)

1. The actor adds a task (e.g. Lecture)
2. System prompts for a date
3. Actor selects the frequency of recurrence (e.g. every Monday)
4. System prompts for time/duration
5. Actor selects time/duration (e.g. 2PM - 4PM) and presses enter
6. System sets the recurring reminder for the task (Lecture, every Monday, 2 PM - 4 PM)

3. Rearranging the task list based on the priority of importance

1. The actor selects a task from the task list
2. System provides the option of prioritizing it in terms of rank
3. The actor selects a priority rank number (Rank 1 being the most important)
4. System rearranges the task in descending priority order

4. Creating an event task

1. The actor opens system.
2. System prompts actor to input password.
3. User inputs password.
4. The system displays a welcome message and today’s task list.
5. The actor types in event command in the console, indicating the description and the date-time of the event he/she wants to add.
6. System shows successfully added messages in the console.

5. Showing all the tasks that have been done

1. The actor opens system
2. System prompts actor to input password.
3. User inputs password.
4. System displays a welcome message and today's task list.
5. The actor types in show command in the console.
6. The system outputs all tasks that have been marked done today in the console.

6. Toggling the done function

1. The actor accidentally marked an undone task as done
2. System displays the task as done and displays task list
3. The actor types in the undo command for the particular task in the console
4. The system outputs that task as undone (toggled the completion of task)

7. Categorising my task

1. The actor opens the system
2. System prompts actor to input password.
3. User inputs password.
4. System displays a welcome message and today's task list.
5. The actor types in the categorise command in the console.
6. System asks if the actor wants to create a new category or to add to an existing category
7. If actor chooses to create a new category,
    a. System ask which tasks the actor wants to add to the new category
    b. Actor uses the add command to add tasks to the new category
    c. System displays a message to confirm the success of creating a new category.
8. If the actor chooses to edit existing category,
    a. The system asks the actor which category he/she wants to edit.
    b. The actor keys in the name of the category in the console.
    c. The system then asks the actor whether he/she wants to add or delete.
    d. The actor key in add command / delete command.
    e. If the actor keys in add command, the system asks the index of task that he wants to add.
    f. The actor types in the index in the console.
    g. The system shows successfully added message.
    h. If the actor keys in delete command, the system asks the index of task that he wants to delete.
    i. The actor

8. Viewing the monthly calendar

1. Actor opens the system.
2. System prompts actor to input password.
3. User inputs password.
4. System displays a welcome message and today's task list.
5. Actor types in the task page.

6. System tells the user that the commands there are in the task page.
7. Actor calls the monthly calendar view command.
8. Calendar for the current month is displayed.

<u>9. Viewing the annual calendar</u>

1. Actor opens the system.
2. System displays a welcome message and today’s task list.
3. Actor types in the task page.
4. System tells the user that the commands there are in the task page.
5. Actor calls the annual calendar view command.
6. Calendar for the current year is displayed.

<u>10. Searching for a location of LT19</u>

1. Actor opens the system.
2. System prompts actor to input password.
3. User inputs password.
4. System displays welcome message and today’s task list.
5. Actor inputs “places”.
6. System displays the welcome message for places page.
7. Actor types in “find-LT19”.
8. System displays the location of LT19.
9. If the location of LT19 does not exist in the System.
10. System will tell the actor that the location search is not found.

<u>11. Adding a location to places page</u>

1. Actor opens the system.
2. System prompts actor to input password.
3. User inputs password.
4. System displays welcome message and today’s task list.
5. Actor inputs “places”.
6. System displays the welcome message for places page.
7. Actor types in “add-LT19”.
8. System will display a confirmation message.

<u>12. Deleting a place and Undoing a places command</u>

1. Actor opens the system.
2. System prompts actor to input password.
3. User inputs password.
4. System displays welcome message and today’s task list.
5. Actor inputs “places”.
6. System displays the welcome message for places page.
7. Actor types in “delete-LT19”.
8. System displays a confirmation message.

9. Actor realised he input the wrong command and wants to undo his previous command
10. Actor inputs “undo”.
11. System will display the confirmation message.

<u>13. Listing all the places in the System</u>

1. Actor opens the system.
2. System prompts actor to input password.
3. User inputs password.
4. System displays welcome message and today’s task list.
5. Actor inputs “places”.
6. System displays the welcome message for places page.
7. Actor inputs “list”.
8. System will display all the locations stored.

<u>14. Deleting an assessment from a module in the module page</u>

1. Actor opens the system.
2. System prompts actor to input password.
3. Actor inputs password.
4. System displays a welcome message and today’s task list.
5. Actor types in “notes”.
6. System displays the welcome message for the notes page.
7. Actor types in “module /n “cg2028”.
8. If the module code exists:
    a. System displays the module page welcome message.
    b. Actor types in “delete assmt /n 2”.
    c. If the index is valid:
        i. System displays a confirmation message.
    d. If the index is invalid:
        i. System asks actor to enter a valid index.
9. If the module does not exist:
    a. System tells actor that the module does not exist.

<u>15. Adding a module to the note page</u>

1. Actor opens the system.
2. System prompts actor to input password.
3. Actor inputs password.
4. System displays a welcome message and today’s task list.
5. Actor types in “notes”.
6. System displays the welcome message for the notes page.
7. Actor types in “add /n cs2113t”.
8. If the module does not already exist in the note page:
    a. System displays a success message saying that the module has been added and adds that module to the note page.

9. If the module already exist in the note page:
    a. Systems informs actor that there is already an existing module in the note page with the same name and does not add that module to the note page.

<u>16. Editing the goal in the note page</u>

1. Actor opens the system.
2. System prompts actor to input password.
3. Actor inputs password.
4. System displays a welcome message and today's task list.
5. Actor types in "notes".
6. System displays the welcome message for the notes page.
7. Actor types in "edit /n to become a CEO".
8. System displays a success message saying that the goal has been updated.

<u>17. Adding an expense to expense page</u>

1. Actor opens the system.
2. System prompts actor to input password.
3. User inputs password.
4. System displays welcome message and today's task list.
5. Actor inputs "expenses".
6. System displays welcome message for expenses page
7. Actor types in "add bread, $4, 2019-09-09"
8. System displays a confirmation message.

<u>18. Deleting an expense and Undoing an expense command</u>

1. Actor opens the system.
2. System prompts actor to input password.
3. User inputs password.
4. System displays welcome message and today's task list.
5. Actor inputs "expenses".
6. System displays welcome message for expenses page.
7. Actor types in "delete bread".
8. System displays a confirmation message.
9. Actor realizes he has input the wrong command and wishes to undo his previous command.
10. Actor inputs "undo".
11. System displays a confirmation message.

<u>19. Finding the list of expenses in on a specific date</u>

1. Actor opens the system.
2. System prompts actor to input password.
3. User inputs password.
4. System displays welcome message and today's task list.
5. Actor inputs "expenses".

6. System displays the welcome message for expenses page.
7. Actor types in “find 2019-09-09”.
8. System displays the list of expenses spent on that date.
9. If there are no expenses on that date, the system will tell the actor that the date is not found in the list.

<u>20. Listing all the expenses in the system</u>

1. Actor opens the system.
2. System prompts actor to input password.
3. User inputs password.
4. System displays welcome message and today’s task list.
5. Actor inputs “expenses”.
6. System displays the welcome message for expenses page.
7. Actor inputs “list”.
8. System displays all the expenses.

<u>21. Mark a technical elective as completed under a specialization</u>

1. Actor opens the system.
2. System prompts actor to input password.
3. User inputs password.
4. System displays welcome message and today’s task list.
5. Actor inputs “spec”.
6. System displays the welcome message for specialization page.
7. Actor inputs “complete”.
8. System displays specializations that actor can choose from and prompts for the specialization index number that actor wants to see.
9. Actor inputs “1”.
10. System will display all the technical electives under that specialization and prompts for the technical elective that the actor has completed.
11. Actor inputs “1”.
12. System displays a confirmation message.

<u>22. Listing all the specialization and technical electives in that specialization</u>

1. Actor opens the system.
2. System prompts actor to input password.
3. User inputs password.
4. System displays welcome message and today’s task list.
5. Actor inputs “spec”.
6. System displays the welcome message for specialization page.
7. Actor inputs “list”.
8. System displays specializations that actor can choose from and prompts for the specialization index number that actor wants to see.
9. Actor inputs “1”.

10. System will display all the technical electives under that specialization, the technical electives that are completed, and the total modular credits completed.

23. Adding a contact to contacts page

1. Actor opens the system.
2. System prompts actor to input password.
3. Actor inputs password.
4. System displays welcome message and today's task list.
5. Actor inputs "contacts".
6. System displays the welcome message for contacts page.
7. Actor types in "add jason,96251322".
8. System will display a confirmation message.

24. Deleting a contact and Undoing a contact command

1. Actor opens the system.
2. System prompts actor to input password.
3. Actor inputs password.
4. System displays welcome message and today's task list.
5. Actor inputs "contacts".
6. System displays welcome message for contacts page.
7. Actor types in "delete jason".
8. System displays a confirmation message.
9. Actor realizes he has input the wrong command and wishes to undo his previous command.
10. Actor inputs "undo".
11. System displays a confirmation message

25. Listing all the contacts in the contacts page

1. Actor opens the system.
2. System prompts actor to input password.
3. Actor inputs password.
4. System displays welcome message and today's task list.
5. Actor inputs "contacts".
6. System displays the welcome message for contacts page.
7. Actor inputs "list".
8. System displays all the contacts.

26. Finding a contact in the contacts page

1. Actor opens the system.
2. System prompts actor to input password.
3. Actor inputs password
4. System displays welcome message and today's task list.
5. Actor inputs "contacts".
6. System displays the welcome message for contacts page.

7. Actor types in “find jason”.
8. System displays the name “jason” and the phone number.
9. If there are no contacts that contain the keyword “jason”, the system will tell the actor that the name is not found in the list.

27. Adding a module to CAP page

1. Actor opens the system.
2. System prompts actor to input password
3. Actor inputs password
4. System displays welcome message and today’s task list.
5. Actor inputs “cap”.
6. System displays the welcome message for CAP page.
7. Actor types in “add 1,CS1231,4,B+”.
8. System will display a confirmation message.

28. Deleting a module from CAP page

1. Actor opens the system.
2. System prompts actor to input password.
3. Actor inputs password.
4. System displays welcome message and today’s task list.
5. Actor inputs “cap”.
6. System displays welcome message for cap page.
7. Actor types in “delete CS1231”.
8. System displays a confirmation message.
9. Actor realizes he has input the wrong command and wishes to undo his previous command.
10. Actor inputs “undo”.
11. System displays a confirmation message.

29. Listing all the modules in the CAP page

1. Actor opens the system.
2. System prompts actor to input password
3. Actor inputs password
4. System displays welcome message and today’s task list.
5. Actor inputs “cap”.
6. System displays the welcome message for cap page.
7. Actor inputs “list 1”.
8. System displays all the modules in semester 1 and the CAP in semester 1.

30. Finding a module in the CAP page

1. Actor opens the system.
2. System prompts actor to input password.
3. Actor inputs password.
4. System displays welcome message and today’s task list.

5. Actor inputs “cap”.
6. System displays the welcome message for cap page.
7. Actor types in “find CS1231”.
8. System displays the module’s information(semester number, code, credit, grade)..
9. If there are no modules that contain the keyword “jason”, the system will tell the actor that the module is not found in the list.

31. Change password

1. Actor opens the system.
2. System prompts user to input password.
3. Actor inputs password.
4. System displays welcome message and today’s task list.
10. Actor input change password.
11. System prompts actor to input current password.
12. Actor inputs current password.
13. System prompts actor to input new password.
14. Actor inputs new password.
15. System displays success message in changing password.

Appendix C: Non-functional Requirements:

- Documentation: All features, functions, classes and other files should be well explained with JavaDocs inside the project. A ReadMe/ UserGuide should be provided so that users could get familiar with features.
- Performance requirements: the system should respond within two seconds.

Appendix D: Glossary

- Task: an item in the user’s list
- Theme: The general color scheme of the calendar

Appendix E: Instructions for Manual Testing

Given below are some test cases that can be used to test the app manually.

E.1. Launch

To launch the jar file, run it from the command line by typing in `java -jar` and then drag the jar file into the command line window. Next, press ENTER. For a more detailed explanation, please refer to the user guide Quick Start section.

E.2. Note Feature

Most of the preloaded data for the module page feature is in the module ‘cg2028’.

1. Go to note page
    a. Prerequisites: you must be in the main page.
    b. Test case 1: `notes`
        i. Expected: welcome message for note page appears.
    c. Test case 2: `9`
        i. Expected: welcome message for note page appears.
2. View you goal and list of modules
    a. Prerequisites: you must be in the note page.
    b. Test case 1: `view`
        i. Expected: your goal and your list of modules appear.
3. Edit your goal
    a. Prerequisites: you must be in the note page.
    b. Test case 1: `edit /n to become a CEO`
        i. Expected: success message for editing goal appears and when you type in view, your goal has changed to “to become a CEO”.
    c. Test case 2: `edit /ni want to be superman`
        i. Expected: success message for editing goal appears and when you type in view, your goal has changed to “i want to be superman”.
    d. Test case 3: `edit /n`
        i. Expected: error message shows and tells you that the description of the command cannot be empty.
4. Adding a module
    a. Prerequisites: you must be in the note page; there must be a module called ‘easy module’ in the module list when you type in `view`.
    b. Test case 1: `add /n annoying module`
        i. Expected: success message is shown and when you type in view, the module ‘annoying module’ has been added to the list of modules.
    c. Test case 2: `add`
        i. Expected: error message shown shows the correct input format for the command.
    d. Test case 3: `add /n easy module`
        i. Expected: error message shown tells you that you already have a module called ‘easy module’.
5. Deleting a module
    a. Prerequisites: you must be in the note page; you must have added a module called ‘annoying module’; there cannot be a module called ‘hello’ in the list of modules.
    b. Test case 1: `delete /n annoying module`

            i. Expected: success message is shown and when you type in view, the module "annoying module" is gone from the list of modules.
    c. Test case 2: `delete /n hello`
        i. Expected: error message shown says that there is no such module.
6. Go to module page
    a. Prerequisite: you must be in the note page; there must be a module called 'cg2028' and there cannot be a module called 'hello' in the list of modules when you type in `view`.
    b. Test case 1: `module /ncg2028`
        i. Expected: welcome message for the module page appears.
    c. Test case 2: module /n hello
        i. Expected: error message shown says that there is no such module.
7. View the modules notes
    a. Prerequisite: you must have typed in `module /n cg2028` and pressed ENTER before this.
    b. Test case 1: view
        i. Expected: the module's assessments and miscellaneous information appears.
8. Editing a module's name
    a. Prerequisite: you must be in the module page for 'cg2028'
    b. Test case 1: `edit mod /n CG2028`
        i. Expected: when you type in `view`, the module's name has changed.
9. Adding an assessment
    a. Prerequisite: you must be in the module page for 'cg2028'.
    b. Test case 1: `add assmt /n hello /a`
        i. Expected: error message shown asks you to input a weightage.
    c. Test case 2: `add assmt /n hello /a -1`
        i. Expected: error message shown asks you to input a positive number for the weightage.
    d. Test case 3: `add assmt /n hello /a hello`
        i. Expected: error message shown asks you to input a number for the weightage.
    e. Test case 4: `add assmt /n presentation /a 45`
        i. Expected: success message is shown and when you type in `view`, the assessment 'presentation' is added to the list of assessments with its weightage as 45%.
10. Editing an assessment's name
    a. Prerequisite: you must be in the module page for 'cg2028'.
    b. Test case 1: `edit assmt /n-1 /a`
        i. Expected: error message shown tells you to input a new assessment name.
    c. Test case 2: `edit assmt /n -1 /a hello`

        i. Expected: error message shown tells you that there is no such index.

11. Editing an assessment's weightage
    a. Prerequisite: you must be in the module page for 'cg2028'.
    b. Test case 1: `edit weight /n 1`
        i. Expected: error message shown shows you the correct format for the command.
12. Deleting an assessment
    a. Prerequisite: you must be in the module page for 'cg2028'; there must be at least 1 assessment in the list when you type in `view`.
    b. Test case 1: `delete assmt /n hello`
        i. Expected: error message shown tells you to input a number for the index.
    c. Test case 2: `delete assmt /n 1`
        i. Expected: success message is shown and the assessment that was at index 1 is gone from the list when you type in `view`.
13. Adding a miscellaneous information
    a. Prerequisite: you must be in the module page for 'cg2028'.
    b. Test case 1: `add msc`
        i. Expected: error message shown shows you the correct format for the command.
14. Editing a miscellaneous information
    a. Prerequisite: you must be in the module page for 'cg2028'; there must be at least 1 miscellaneous information in the list when you type in `view`.
    b. Test case 1: `edit msc /n 1 /a no webcast for this mod`
        i. Expected: success message is shown and the miscellaneous information at index one is changed to 'no webcast for this mod' when you type in `view`.
15. Deleting a miscellaneous information
    a. Prerequisite: you must be in the module page for 'cg2028'; there must be at least 1 miscellaneous information in the list when you type in `view`.
    b. Test case 1: `delete msc /n 1`
        i. Expected: success message is shown and the miscellaneous information that was originally at index 1 in the list when you type in `view` has been deleted from the list.
16. Viewing the list of possible commands for the module page
    a. Prerequisite: you must be in a module page.
    b. Test case 1: `commands`
        i. Expected: the list of possible commands for the module page is shown.
17. Viewing the list of possible commands for the note page
    a. Prerequisite: you must be in a note page.
    b. Test case 1: `commands`

            i. Expected: the list of possible commands for the note page is shown.

E.3. Note Section for a Particular Period Feature

This feature can be found in the user guide Section 3.2.28. Most of the preloaded data is for the day '2019-11-09'.

1. Adding a note to a particular period
    a. Test case 1:
        1. First line of input: `addNote day 2019-11-09`
        2. Second line of input: `got exams on this day :(`

        ii. Expected: success message and if you type in `listNote day 2019-11-09`, the note is in the list.
    b. Test case 2: `addNote day`
        i. Expected: error message shown telling you that you need to input a date.
    c. Test case 3:
        1. First line of input: `addNote week 2019-10-14`
        2. Second line of input: `got exams on this day :(`

        ii. Expected: success message and if you type in `listNote week 2019-10-14`, the note is in the list.
    d. Test case 4: `addNote week 2019-11-09`
        i. Expected: error message shown telling you that the date specified in the command has to be a Monday.
    e. Test case 5: `addNote month 2019-11-09`
        i. Expected: error message shown tells you the correct format for the command.
    f. Test case 6: `addNote`
        i. Expected: error message shown tells you that the description of the command cannot be empty.
2. Editing a note for a particular period
    a. Prerequisite: there must be at least one note for the period and date you specify.
    b. Test case 1:
        1. First line of input: `editNote day 2019-11-09 1`
        2. Second line of input: `no exams on this day :)`

        ii. Expected: success message shown and if you type in `listNote day 2019-11-09`, the note is updated.
    c. Test case 2:
        1. First line of input: `editNote month 2019-11 1`
        2. Second line of input: `no exams on this day :)`

        ii. Expected: success message shown and if you type in `listNote month 2019-11-09`, the note is updated.
3. Deleting a note for a particular period

    a. Prerequisite: there must be at least one note for the period and date you specify.
    b. Test case 1: `deleteNote week 2019-10-14 1`
        i. Expected: success message shown and if you type in `listNote week 2019-10-14`, the note is removed.

E.4. Expenses Feature

1. Go to the expenses page
    a. Prerequisites: you must be in the main page.
    b. Test case 1: `expenses`
        i. Expected: Welcome message for expenses page appears.

2. Add an expense
    a. Prerequisite: you must be in the expenses page.
    b. Test case 1: `add`
        i. Expected: Error message that prompts user to input the correct format.
    c. Test case 2: `add bread, $4, 2019-04-03`
        i. Expected: Success message is shown
    d. Test case 3: `add bread, 4, 2019-04-03`
        i. Expected: Error message shown shows the correct price input format for the command.
    e. Test case 4: `add bread, $4, 2019-4-3`
        i. Expected: Error message shown shows the correct date input format for the command.

3. Delete an expense
    a. Prerequisites: you must be in the expenses page; you have to add an expense “bread, $4, 2019-04-03”; there cannot be an expense “curry, $5.50, 2019-09-09”.
    b. Test case 1: `delete bread`
        i. Expected: Success message is shown.
    c. Test case 2: `delete curry`
        i. Expected: Error message shown shows that item is not found.

4. Find an expense
    a. Prerequisites: you must be in the expenses page; you have to add an expense “bread, $4, 2019-04-03”; there cannot be an expense “curry, $5.50, 2019-09-09”.
    b. Test case 1: `find 2019-04-03`
        i. Expected: System displays the list of expenses found on that date.
    c. Test case 2: `find 2019-09-09`
        i. Expected: Error message shown shows that date is not found.

   d. Test case 3: `find 2019-9-9`
      i. Expected: Error message shown shows that date format is wrong.

5. Show list of expenses
   a. Prerequisites: you must be in the expenses page.
   b. Test case 1: `list`
      i. Expected: List of all the expenses will be displayed.

6. Undo previous expense command
   a. Prerequisites: you must be in the expenses page; you have to add an expense “bread, $4, 2019-04-03” for the test case 1, and you have to add an expense “curry, $5.50, 2019-09-09” and delete it for test case 2
   b. Test case 1: undo
      i. Expected: Success message is shown. The add command is undone and “bread, $4, 2019-04-03” will not appear in the list when the list command is called.
   c. Test case 2: undo
      i. Expected: Success message is shown. The delete command is undone and “curry, $5.50, 2019-09-09” will appear in the list when the list command is called.

E.5. Contacts Feature

1. Go to contact page
   a. Prerequisites:
      i. You must be in main page.
   b. Test case 1:
      i. Input: `contacts`
      ii. Expected: Welcome message for contacts page appears.
   c. Test case 2: 2
      i. Expected: Welcome message for contacts page appears.
2. Add a contact
   a. Prerequisites: You must be in contacts page.
   b. Test case 1:
      i. Input: `1`
      ii. Expected: System prompts you to input name,number.
      iii. Input: `jason,96251322`
      iv. Expected: Success message is shown.
   c. Test case 2:
      i. Input: `add`

        ii. Expected: System prompts you to input name,number.
        iii. Input: `janel,92251322`.
        iv. Expected: Success message is shown
    d. Test case 3:
        i. Input: `add jess,92351922`
        ii. Expected: Success message is shown.
3. Delete a contact
    a. Prerequisites:
        i. You must be in contacts page.
        ii. Name must be in the contact list. Input `list` to check. If not in the list, refer to Test case 4.
    b. Test case 1:
        i. Input: `3`
        ii. Expected: System prompts you to input name.
        iii. User inputs `jason`
        iv. Expected: Success message shown.
    c. Test case 2:
        i. User inputs `delete`
        ii. Expected: System prompts you to input name.
        iii. Input: `janel`
        iv. Expected: Success message shown.
    d. Test case 3:
        i. Input: `delete jess`
        ii. Expected: Success message shown.
    e. Test case 4:
        i. Input a name not in the list.
        ii. Expected: Not found in the list message shown.
4. Find a contact
    a. Prerequisites:
        i. You must be in contacts page.
        ii. Name must be in the contact list. Input `list` to check. If not in the list, refer to Test case 4.
    b. Test case 1:
        i. Input: `2`
        ii. Expected: System prompts user to input name.
        iii. User inputs `jason`
        iv. Expected: Displays jason and number.
    c. Test case 2:
        i. Input `find`

ii. Expected: System prompts you to input name
iii. Input: `jason`
iv. Expected: Displays jason and number

d. Test case 3:
   i. Input: `find jason`
   ii. Expected: Displays jason and number
e. Test case 4:
   i. Input: `find jess` (name is not in the list)
   ii. Expected: Not found in the list message shown.

5. List contacts
   a. Prerequisites:
      i. You must be in contact page
   b. Test case 1:
      i. Input: `list`
      ii. Expected: Displays all contacts in the contact list
6. Undo
   a. Prerequisites:
      i. You must be in the contact page.
      ii. You must add or delete a contact.
   b. Test case 1:
      i. Input: `undo`
      ii. Expected: Success message is shown. Add/Delete contact is now undone.

E.6. Places Feature

1. Go to the places page
   a. Prerequisites: you must be in the main page.
   b. Test case 1:
      i. User inputs: `places`
      ii. Expected: Welcome message for places page appears.
   c. Test case 2:
      i. User inputs: `4`
      ii. Expected: Welcome message for places page appears.

2. Add a place
   a. Prerequisite: You must be in the places page.
   b. Test case 1:
      i. User inputs: `add`
      ii. Expected: Message that prompts you to input the place you want to add.
      iii. User inputs: `LT45,COM5`

        iv. Expected: Success message is shown.
    c. Test case 2:
        i. User inputs: `1`
        ii. Expected: Message that prompts you to input the place you want to add.
        iii. User inputs: `LT45,COM5`
        iv. Expected: Success message is shown.
    d. Test case 3:
        i. User inputs: `add-LT45,COM5`
        ii. Expected: Success message is shown.

3. Delete a place
    a. Prerequisites: You must be in the places page; There is a place “LT45,COM5” in the list of places; “LT99,COM99” is not in the list of places.
    b. Test case 1:
        i. User inputs: `delete-LT45`
        ii. Expected: Success message is shown.
    c. Test case 2:
        i. User inputs: `delete`
        ii. Expected: Message that prompts you to input the place you want to delete.
        iii. User inputs: `LT45`
        iv. Expected: Success message is shown.
    d. Test case 3:
        i. User inputs: `3`
        ii. Expected: Message that prompts you to input the place you want to delete.
        iii. User inputs: `LT45`
        iv. Expected: Success message is shown.
    e. Test case 4:
        i. User inputs: `delete-LT99`
        ii. Error message which indicates `LT99` is not in the list of places.

4. Find a place
    a. Prerequisites: You must be in the places page; There is a place “LT45,COM5” in the list of places; “LT99,COM99” is not in the list of places.
    b. Test case 1:
        i. User inputs: `find-LT45`
        ii. Expected: System displays the list of places that are related to LT45.
    c. Test case 2:
        i. User inputs: `find-LT99`
        ii. Expected: Error message shown shows that date is not found.

   d. Test case 3:
      i. User inputs: `find`
      ii. Expected: Message that prompts you to input the place you want to find.
      iii. User inputs: `LT45`
      iv. Expected: Success message is shown.
   e. Test case 3:
      i. User inputs: `2`
      ii. Expected: Message that prompts you to input the place you want to find.
      iii. User inputs: `LT45`
      iv. Expected: Success message is shown.

5. Show list of places
   a. Prerequisites: You must be in the places page.
   b. Test case 1:
      i. User input: `list`
      ii. Expected: List of all the places will be displayed.
   c. Test case 2:
      i. User input: `4`
      ii. Expected: List of all the places will be displayed.

6. Undo previous places command
   a. Prerequisites: You must be in the places page; For test case 1, you have to add a place “LT45,COM5”. For test case 2 and 3, you have to add an expense “LT53,COM6” and delete it.
   b. Test case 1:
      i. User input: `undo`
      ii. Expected: Success message is shown. The add command is undone and “LT45,COM5” will not appear in the list when the list command is called.
   c. Test case 2:
      i. User input: `undo`
      ii. Expected: Success message is shown. The delete command is undone and “LT53,COM6” will appear in the list when the list command is called.
   d. Test case 3:
      i. User input: `5`
      ii. Expected: Success message is shown. The delete command is undone and “LT53,COM6” will appear in the list when the list command is called.

E.7. CAP Feature

1) Go to CAP page

    a) Prerequisites:
        i) You must be in main page.
    b) Test case 1:
        i) Input: `cap`
        ii) Expected: Welcome message for CAP page appears.
    c) Test case 2: 6
        i) Expected: Welcome message for CAP page appears.

2) Add a module
    a) Prerequisites: You must be in CAP page.
    b) Test case 1:
        i) Input: `1`
        ii) Expected: System prompts you to input semester number,code,credit,grade.
        iii) Input: `1,CS1231,4,B`
        iv) Expected: Success message is shown.
    c) Test case 2:
        i) Input: `add`
        ii) Expected: System prompts you to input semester number,code,credit,grade.
        iii) Input: `1,CS1010,4,B`
        iv) Expected: Success message is shown
    d) Test case 3:
        i) Input: `add 1,CG1111,6,A`
        ii) Expected: Success message is shown.

3) Delete a module
    a) Prerequisites:
        i) You must be in contacts page.
        ii) Name must be in the contact list. Input `list all` to check. If not in the list, refer to Test case 4.
    b) Test case 1:
        i) Input: `3`
        ii) Expected: System prompts you to input code.
        iii) User inputs `CS1231`
        iv) Expected: Success message shown.
    c) Test case 2:
        i) User inputs `delete`
        ii) Expected: System prompts you to input name.
        iii) Input: `CS1010`
        iv) Expected: Success message shown.
    d) Test case 3:
        i) Input: `delete CG1111`

        ii) Expected: Success message shown.
    e) Test case 4:
        i) Input a code not in the list. `delete CS4444`
        ii) Expected: Not found in the list message shown.

4) Find modules
    a) Prerequisites:
        i) You must be in CAP page.
        ii) Name must be in the CAP list. Input `list all` to check. If not in the list, refer to Test case 4.
    b) Test case 1:
        i) Input: `2`
        ii) Expected: System prompts user to input code.
        iii) User inputs `CS1231`
        iv) Expected: Displays module's information.
    c) Test case 2:
        i) Input `find`
        ii) Expected: System prompts you to input code
        iii) Input: `CS1231`
        iv) Expected: Displays module's information.
    d) Test case 3:
        i) Input: `find CS1231`
        ii) Expected: Displays module's information.
    e) Test case 4:
        i) Input: `find CS4444`(code is not in the list)
        ii) Expected: Not found in the list message shown.

5) List modules
    a) Prerequisites:
        i) You must be in CAP page
    b) Test case 1:
        i) Input: `list all`
        ii) Expected: Displays all modules in the CAP list
    c) Test case 2:
        i) List a semester. (1-8). Input: `list 1`
        ii) Expected: Displays all modules in semester 1 in the CAP list
    d) Test case 3:
        i) List a semester not recognise by the system. Input: `list 9`
        ii) Expected: Display incorrect format message.

E.8. Calendar View Feature

1. View monthly calendar
    a. Prerequisites: you must be in the tasks page.
    b. Test case 1:
        i. User inputs: `calendar monthly view`
        ii. Expected: Calendar for that particular month appears.
2. View annual calendar
    a. Prerequisites: you must be in the tasks page.
    b. Test case 1:
        i. User inputs: `calendar annual view`
        ii. Expected: Calendar for that particular year appears.

E.9. Password/Change Password Feature

1. To input password
    a. Test case 1:
        i. Upon start up the program, the system will prompt you to input your password. (default password: `jjjry`)
        ii. Input: jjjry
        iii. Expected: Welcome message for tasks page appears.
2. To change password
    a. Prerequisites:
        i. You must be in main page.
    b. Test case 1:
        i. Input: `change password`
        ii. Expected: system will prompt you to input your current password.
        iii. Input: `jjjry`
        iv. Expected: System will prompt you to input your new password.
        v. Input: `jason`
        vi. Expected: Display successful message.
    c. Test case 2:
        i. Input: 10
        ii. Expected: system will prompt you to input your current password.
        iii. Input: `jason`
        iv. Expected: System will prompt you to input your new password.
        v. Input: `jjjry`
        vi. Expected: Display successful message.

E.10. Module Planner Feature

1. Go to Module Planner page
    1) Prerequisites:
        a) You must be in main page.
    2) Test case 1:
        a) Input: `moduleplanner`
        b) Expected: Welcome message for Module Planner page appears.
    3) Test case 2: 8
        a) Expected: Welcome message for Module Planner page appears.

2. Add a module to the module plan
    1) Prerequisites: You must be in module plan page.
    2) Test case 1:
        a) Input: `add CS1456 to 5`
        b) Expected: Exception shows, the module code is not a recognized module
    3) Test case 2:
        a) Input: `add CS1231 to 5`
        b) Expected: Exception shows, the module is already inside Semester 1 in by default recommended study plan.
    4) Test case 3:
        a) Input: `add CG3207 to 5`
        b) Expected: Success message is shown.

3. Delete a module from the module plan.
    1) Prerequisites: You must be in module plan page.
    2) Test case 1:
        a) Input: `delete CS1456 from 5`
        b) Expected: Exception shows, the module code is not a recognized module
    3) Test case 2:
        a) Input: `delete`
        b) Expected: Exception shows, the input format is not correct.
    4) Test case 3:
        a) Input: `delete CS1231 from 1`
        b) Expected: Success message is shown.

4. Shift module to specific column.
    1) Prerequisites:
        a) You must be in module planner page.
    2) Test case 1:
        a) Input: `shift CS1245 to 5`

        b) Expected: Exception shows, the module code is not a recognized module
    3) Test case 2:
        a) Input :`shift`
        b) Expected: Exception shows, the input format is not correct.
    4) Test case 3:
        a) Input: `shift CS1231 to 5`
        b) Expected: Successful message shows.

5. Display the module plan.
    1) Prerequisites:
        a) You must be in module planner page
    2) Test case 1:
        a) Input: `plan`
        b) Expected: Displays the study module plan in the console, suggestions of graduation requirement and technical elective requirements are also displayed.

6. Check prerequisite of module.
    1) Prerequisites:
        a) You must be in module planner page.
    2) Test case 1:
        a) Input: `prerequisite CS2040C`
        b) Expected: Displays a tree of prerequisite modules of CS2040C in the console.
    3) Test case 2:
        a) Input: `prerequisite CS3230`
        b) Expected: Displays a tree of prerequisite modules of CS2040C in the console.